Web programming

Tópicos de Informática para Automação

Mário Antunes

November 17, 2025

Exercícios

Este guião acompanha os slides teóricos sobre Páginas Web Dinâmicas. Irão construir uma aplicação web completa de raiz, começando com um perfil estático e evoluindo para um sistema dinâmico com autenticação de utilizador, mapas em tempo real e suporte via chat.

Tecnologias utilizadas: * **Frontend:** HTML5, CSS3 (Tema Nord Light), Vanilla JavaScript. * **Backend 1:** Node.js (Express) para Autenticação e Chat. * **Backend 2:** Python (FastAPI) para Geolocalização e processamento de Dados. * **Infraestrutura:** Docker & Docker Compose.

Fase 1: Configuração do Projeto e Estrutura Estática

Passo 0: Instalação e Verificação

Antes de escrever código, garantam que o vosso ambiente está pronto.

1. **Abram o vosso terminal.**
2. **Verifiquem o Docker:** `docker --version` e `docker compose version`
3. **Verifiquem o Node.js (Opcional):** `node -v`

Passo 1: Estrutura de Pastas

1. Criem uma pasta principal chamada `my-web-project`.
2. Dentro dela, criem três subpastas: `frontend`, `auth-service`, e `geo-service`.
3. Criem um ficheiro `docker-compose.yml` na raiz.

Passo 2: O Docker Compose Base

Abram o `docker-compose.yml` e colem este código:

```yaml
services:
  # 1. Frontend Server (Nginx)
  web:
    image: nginx:alpine
    container_name: frontend_server
    ports:
      - "8080:80"
    volumes:
      - ./frontend:/usr/share/nginx/html
```

Passo 3: O Perfil Estático (HTML)

Abram o `frontend/index.html`. Reparem na div `chatWidget` no fundo; esta será usada no Passo 11.

```html
<!DOCTYPE html>
<html lang="pt">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>O Meu Perfil Dinâmico</title>
    <link rel="stylesheet" href="[https://unpkg.com/leaflet@1.9.4/dist/leaflet.css](https
    <link rel="stylesheet" href="style.css">
    <script src="app.js" defer></script>
</head>
<body>
    <header>
        <h1>Perfil de Utilizador</h1>
        <nav>
            <button id="loginBtn">Entrar</button>
            <button id="logoutBtn" style="display:none;">Sair</button>
        </nav>
    </header>

    <main id="app">
        <dialog id="loginDialog">
            <form id="loginForm">
                <h2>Bem-vindo de Volta</h2>
                <input type="text" id="username" placeholder="Nome de Utilizador" require
                <input type="password" id="password" placeholder="Palavra-passe" required
                <button type="submit">Entrar</button>
                <button type="button" id="cancelLogin">Cancelar</button>
            </form>
        </dialog>

        <div id="contentArea" class="hidden">
            <section class="card profile-card">
                <img src="[https://ui-avatars.com/api/?name=User&background=88C0D0&color=
                <h2 id="welcomeMsg">Olá, Utilizador</h2>
                <p>Estudante Full Stack</p>
            </section>

            <section class="card gallery-card">
                <h3>Galeria de Fotos</h3>
                <div id="galleryGrid" class="grid"></div>
            </section>

            <section class="card map-card">
                <h3>Rastreador de Localização em Tempo Real (WebSocket)</h3>
                <div id="map"></div>
            </section>

            <div id="chatWidget">
                <div id="chatHeader">Chat de Suporte</div>
                <div id="chatMessages"></div>
                <input type="text" id="chatInput" placeholder="Escreva uma mensagem ... ">
            </div>
        </div>

        <div id="guestMessage">
            <p>Por favor inicie sessão para ver o painel.</p>
        </div>
    </main>

    <footer>
        <p>&copy; 2025 Engenharia Web</p>
    </footer>

    <script src="[https://unpkg.com/leaflet@1.9.4/dist/leaflet.js](https://unpkg.com/leaf
</body>
```

```
</html>
```

Passo 4: Estilo Nórdico (CSS)

Abram o `frontend/style.css`. Isto define o layout e a posição da Janela de Chat.

```
:root {
    --polar-night: #2E3440;
    --snow-storm: #ECEFF4;
    --frost-1: #8FBCBB;
    --frost-2: #88C0D0;
    --frost-3: #81A1C1;
    --frost-4: #5E81AC;
    --aurora-red: #BF616A;
}
body {
    font-family: 'Noto Sans', sans-serif;
    background-color: var(--snow-storm);
    color: var(--polar-night);
    margin: 0;
    display: flex;
    flex-direction: column;
    min-height: 100vh;
}
header {
    background-color: var(--frost-4);
    color: white;
    padding: 1rem 2rem;
    display: flex;
    justify-content: space-between;
    align-items: center;
}
button {
    background-color: var(--frost-3);
    color: white;
    border: none;
    padding: 0.5rem 1rem;
    border-radius: 4px;
    cursor: pointer;
    font-weight: bold;
}
button:hover { background-color: var(--frost-2); }
main {
    flex: 1;
    padding: 2rem;
    max-width: 1200px;
    margin: 0 auto;
    width: 100%;
}
.card {
    background: white;
    padding: 1.5rem;
    border-radius: 8px;
    margin-bottom: 2rem;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
}
.hidden { display: none; }
dialog {
    border: 1px solid var(--frost-2);
    border-radius: 8px;
    padding: 2rem;
```

```
}
dialog::backdrop { background: rgba(46, 52, 64, 0.5); }

/* Estilo do Widget de Chat */
#chatWidget {
    position: fixed;
    bottom: 20px;
    right: 20px;
    width: 300px;
    background: white;
    border: 1px solid var(--frost-3);
    border-radius: 8px;
    overflow: hidden;
    display: flex;
    flex-direction: column;
    box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}
#chatHeader {
    background: var(--frost-4);
    color: white;
    padding: 10px;
    font-weight: bold;
}
#chatMessages {
    height: 200px;
    padding: 10px;
    overflow-y: auto;
    font-size: 0.9rem;
    background-color: #fff;
}
#chatInput {
    border: none;
    border-top: 1px solid #eee;
    padding: 10px;
    outline: none;
}

/* Mapa & Galeria */
#map { height: 300px; width: 100%; border-radius: 4px; }
.grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(150px, 1fr));
    gap: 1rem;
}
.grid img { width: 100%; border-radius: 4px; }
```

Passo 5: JavaScript Básico

Criem o `frontend/app.js`.

```
console.log("Aplicação Carregada");

const loginBtn = document.getElementById('loginBtn');
const loginDialog = document.getElementById('loginDialog');
const cancelLogin = document.getElementById('cancelLogin');

loginBtn.addEventListener('click', () ⇒ loginDialog.showModal());
cancelLogin.addEventListener('click', () ⇒ loginDialog.close());
```

Teste da Fase 1: Executem `docker compose up -d` e visitem `http://localhost:8080`.

Fase 2: Backend Node.js (Auth & Chat)

Passo 6: Configurar o Serviço Node

1. Vão para `auth-service/`.
2. Criem o `package.json`:

```
{
  "name": "auth-service",
  "main": "server.js",
  "scripts": { "start": "node server.js" },
  "dependencies": { "express": "^4.18.2", "cors": "^2.8.5", "ws": "^8.13.0" }
}
```

3. Criem o `Dockerfile`:

```
FROM node:18-alpine
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

Passo 7: Implementar a Lógica do Servidor

Criem o `auth-service/server.js`.

```
const express = require('express');
const cors = require('cors');
const http = require('http');
const WebSocket = require('ws');

const app = express();
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });

app.use(cors());
app.use(express.json());

// Endpoint de Login
const VALID_USER = { username: "admin", password: "123" };
app.post('/login', (req, res) => {
    const { username, password } = req.body;
    if (username === VALID_USER.username && password === VALID_USER.password) {
        res.json({ success: true, token: "jwt-123" });
    } else {
        res.status(401).json({ success: false, message: "Credenciais inválidas" });
    }
});

// Lógica de WebSocket do Chat
wss.on('connection', (ws) => {
    ws.send('Suporte: Olá! Como posso ajudar?');
    ws.on('message', (message) => {
        setTimeout(() => {
            ws.send(`Suporte: Recebi "${message}"`);
        }, 1000);
    });
});

server.listen(3000, () => console.log('Auth/Chat a correr na porta 3000'));
```

Passo 8: Atualizar o JS do Frontend

Modifiquem o `frontend/app.js` para lidar com o login.

```
// Adicionar estas referências
const loginForm = document.getElementById('loginForm');
const contentArea = document.getElementById('contentArea');
const guestMessage = document.getElementById('guestMessage');
const logoutBtn = document.getElementById('logoutBtn');

loginForm.addEventListener('submit', async (e) ⇒ {
    e.preventDefault();
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;

    try {
        const response = await fetch('http://localhost:3000/login', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ username, password })
        });
        const data = await response.json();

        if (data.success) {
            loginDialog.close();
            handleLoginState(true);
        } else {
            alert(data.message);
        }
    } catch (err) { console.error(err); }
});

function handleLoginState(isLoggedIn) {
    if (isLoggedIn) {
        contentArea.classList.remove('hidden');
        guestMessage.classList.add('hidden');
        loginBtn.style.display = 'none';
        logoutBtn.style.display = 'inline-block';

        // Carregar funcionalidades dinâmicas
        loadGallery();
        initChat();
        initMap();
    } else {
        location.reload();
    }
}
logoutBtn.addEventListener('click', () ⇒ handleLoginState(false));
```

Passo 9: Atualizar o Docker Compose

Atualizem o `docker-compose.yml` para incluir o serviço `auth`.

```
services:
  web:
    # ... (configuração existente)
  auth:
    build: ./auth-service
    container_name: auth_server
    ports:
      - "3000:3000"
```

Fase 3: Funcionalidades (Galeria & Chat)

Passo 10: A Galeria de Fotos (JS)

Acrescentem isto ao `frontend/app.js`:

```
function loadGallery() {
    const galleryGrid = document.getElementById('galleryGrid');
    const images = [
        '[https://picsum.photos/id/101/300/200](https://picsum.photos/id/101/300/200)',
        '[https://picsum.photos/id/102/300/200](https://picsum.photos/id/102/300/200)',
        '[https://picsum.photos/id/103/300/200](https://picsum.photos/id/103/300/200)',
        '[https://picsum.photos/id/104/300/200](https://picsum.photos/id/104/300/200)'
    ];
    galleryGrid.innerHTML = '';
    images.forEach(url => {
        const img = document.createElement('img');
        img.src = url;
        galleryGrid.appendChild(img);
    });
}
```

Passo 11: O Cliente de Chat (WebSocket)

Acrescentem isto ao `frontend/app.js`. Este código torna funcional a Janela de Chat definida no HTML/CSS.

```
function initChat() {
    const chatInput = document.getElementById('chatInput');
    const chatMessages = document.getElementById('chatMessages');

    // Ligar ao WebSocket Node.js
    const socket = new WebSocket('ws://localhost:3000');

    socket.addEventListener('message', (event) => {
        addMessage(event.data, 'server');
    });

    chatInput.addEventListener('keypress', (e) => {
        if (e.key === 'Enter') {
            const text = chatInput.value;
            socket.send(text);
            addMessage("Eu: " + text, 'user');
            chatInput.value = '';
        }
    });

    function addMessage(text, sender) {
        const div = document.createElement('div');
        div.innerText = text;
        div.style.textAlign = sender === 'user' ? 'right' : 'left';
        div.style.color = sender === 'user' ? '#5E81AC' : '#BF616A';
        div.style.marginBottom = '5px';
        chatMessages.appendChild(div);
        chatMessages.scrollTop = chatMessages.scrollHeight;
    }
}
```

Fase 4: Backend FastAPI (Geolocalização)

Passo 12: Configurar o Serviço Python

1. Vão para `geo-service/`.
2. Criem o `requirements.txt`:

   ```
   fastapi
   uvicorn
   websockets
   ```

3. Criem o `Dockerfile`:

   ```
   FROM python:3.9-slim
   WORKDIR /app
   COPY requirements.txt .
   RUN pip install --no-cache-dir -r requirements.txt
   COPY . .
   CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
   ```

Passo 13: Implementar a Lógica Python

Criem o `geo-service/main.py`.

```
from fastapi import FastAPI, WebSocket
from fastapi.middleware.cors import CORSMiddleware
import asyncio, random

app = FastAPI()
app.add_middleware(CORSMiddleware, allow_origins=["*"], allow_methods=["*"])

@app.websocket("/ws/location")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    lat, lon = 40.64427, -8.64554
    try:
        while True:
            lat += random.uniform(-0.001, 0.001)
            lon += random.uniform(-0.001, 0.001)
            await websocket.send_json({"lat": lat, "lng": lon})
            await asyncio.sleep(2)
    except: print("Disconnected")
```

Passo 14: Atualizar o Docker Compose

Adicionem o serviço `geo` ao `docker-compose.yml`.

```
services:
  # ... web e auth existentes ...
  geo:
    build: ./geo-service
    container_name: geo_server
    ports:
      - "8000:8000"
```

Passo 15: O Mapa (Leaflet + WebSocket)

Acrescentem isto ao `frontend/app.js`:

```
function initMap() {
    const map = L.map('map').setView([40.64427, -8.64554], 13);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '© OpenStreetMap contributors'
```

```
    }).addTo(map);

    const marker = L.marker([40.64427, -8.64554]).addTo(map);

    // Ligar ao WebSocket Python
    const geoSocket = new WebSocket('ws://localhost:8000/ws/location');

    geoSocket.addEventListener('message', (event) => {
        const data = JSON.parse(event.data);
        const newLatLng = [data.lat, data.lng];
        marker.setLatLng(newLatLng);
        map.panTo(newLatLng);
    });
}
```

Passo Final: Executem `docker compose up -d --build` e desfrutem da vossa app!

Fase 5 (Opcional)

Implementem duas grandes alterações no código:

1. Para uma abordagem mais realista, o servidor deve comparar hashes de palavras-passe em vez das palavras-passe diretamente. Implementem esta modificação na página web e no servidor de autenticação.
2. Criem múltiplas janelas de chat, atualizando a página web e o servidor para suportar isto.