

WebPage & deployment

Introdução Engenharia Informática

Mário Antunes

November 17, 2025

Exercises

This guide accompanies the theoretical slides on Dynamic Web Pages. You will build a complete web application from scratch, starting with a static profile and evolving it into a dynamic system with user authentication, real-time maps, and chat support.

Technologies used: * **Frontend:** HTML5, CSS3 (Nord Light Theme), Vanilla JavaScript. * **Backend 1:** Node.js (Express) for Authentication and Chat. * **Backend 2:** Python (FastAPI) for Geolocation and Data processing. * **Infrastructure:** Docker & Docker Compose.

Phase 1: Project Setup & Static Structure

Step 0: Installation & Verification

Before writing code, ensure your environment is ready.

1. **Open your terminal.**
2. **Verify Docker:** docker --version and docker compose version
3. **Verify Node.js (Optional):** node -v

Step 1: Folder Structure

1. Create a main folder named my-web-project.
2. Inside it, create three subfolders: frontend, auth-service, and geo-service.
3. Create a docker-compose.yml file in the root.

Step 2: The Base Docker Compose

Open docker-compose.yml and paste this code:

```
version: '3.8'

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

Step 3: The Static Profile (HTML)

Open frontend/index.html. Note the chatWidget div at the bottom; this will be used in Step 11.

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Dynamic Profile</title>
  <!-- Leaflet CSS for the Map (Step 15) -->
  <link rel="stylesheet" href="[https://unpkg.com/leaflet@1.9.4/dist/leaflet.css](https://unpkg.com/leaflet@1.9.4/dist/leaflet.css)" type="text/css">
  <link rel="stylesheet" href="style.css" type="text/css">
  <script src="app.js" defer></script>
</head>
<body>
  <header>
    <h1>User Profile</h1>
    <nav>
      <button id="loginBtn">Login</button>
      <button id="logoutBtn" style="display:none;">Logout</button>
    </nav>
  </header>

  <main id="app">
    <!-- Login Modal -->
    <dialog id="loginDialog">
      <form id="loginForm">
        <h2>Welcome Back</h2>
        <input type="text" id="username" placeholder="Username" required>
        <input type="password" id="password" placeholder="Password" required>
        <button type="submit">Sign In</button>
        <button type="button" id="cancelLogin">Cancel</button>
      </form>
    </dialog>

    <!-- Content Section (Hidden until logged in) -->
    <div id="contentArea" class="hidden">
      <section class="card profile-card">
        
        <h2 id="welcomeMsg">Hello, User</h2>
        <p>Full Stack Student</p>
      </section>

      <section class="card gallery-card">
        <h3>Photo Gallery</h3>
        <div id="galleryGrid" class="grid"></div>
      </section>

      <section class="card map-card">
        <h3>Live Location Tracker (WebSocket)</h3>
        <div id="map"></div>
      </section>

      <!-- Chat Widget (For Step 11) -->
      <div id="chatWidget">
        <div id="chatHeader">Support Chat</div>
        <div id="chatMessages"></div>
        <input type="text" id="chatInput" placeholder="Type a message ... ">
      </div>
    </div>

    <div id="guestMessage">
      <p>Please log in to view the dashboard.</p>
    </div>
  </main>

```

```

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

<!-- Leaflet JS -->
<script src="[https://unpkg.com/leaflet@1.9.4/dist/leaflet.js](https://unpkg.com/leaf
</body>
</html>

```

Step 4: Nordic Styling (CSS)

Open frontend/style.css. This defines the layout and the Chat Window position.

```

:root {
    --polar-night: #2E3440;
    --snow-storm: #ECEEFF4;
    --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); }

/* Chat Widget Styling */
#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;
}

/* Map & Gallery */
#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; }

```

Step 5: Basic JavaScript

Create frontend/app.js.

```

console.log("Application Loaded");

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());

```

Test Phase 1: Run docker compose up -d and visit <http://localhost:8080>.

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

Step 6: Setup Node Service

1. Go to auth-service/.

2. Create 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. Create Dockerfile:

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

Step 7: Implement Server Logic

Create 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());

// Login Endpoint
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: "Invalid credentials" });
  }
});

// Chat WebSocket Logic
wss.on('connection', (ws) => {
  ws.send('Support: Hello! How can I help you?');
  ws.on('message', (message) => {
    setTimeout(() => {

```

```

        ws.send(`Support: I received "${message}"`);
    }, 1000);
});

server.listen(3000, () => console.log('Auth/Chat running on 3000'));

```

Step 8: Update Frontend JS

Modify frontend/app.js to handle login.

```

// Add these references
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';

        // Load dynamic features
        loadGallery();
        initChat();
        initMap();
    } else {
        location.reload();
    }
}
logoutBtn.addEventListener('click', () => handleLoginState(false));

```

Step 9: Update Docker Compose

Update docker-compose.yml to include auth.

```

services:
  web:

```

```

# ... (existing config)
auth:
  build: ./auth-service
  container_name: auth_server
  ports:
    - "3000:3000"

```

Phase 3: Features (Gallery & Chat)

Step 10: The Photo Gallery (JS)

Append this to 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);
  });
}

```

Step 11: The Chat Client (WebSocket)

Append this to frontend/app.js. This code makes the Chat Window defined in HTML/CSS functional.

```

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

  // Connect to Node.js WebSocket
  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("You: " + 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;
}

```

```
    }
}
```

Phase 4: FastAPI Backend (Geolocation)

Step 12: Setup Python Service

1. Go to geo-service/.
2. Create requirements.txt:

```
fastapi
uvicorn
websockets
```

3. Create 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"]
```

Step 13: Implement Python Logic

Create 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, "lon": lon})
            await asyncio.sleep(2)
    except: print("Disconnected")
```

Step 14: Update Docker Compose

Add the geo service to docker-compose.yml.

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

Step 15: The Map (Leaflet + WebSocket)

Append this to 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);

  // Connect to Python WebSocket
  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);
  });
}

```

Final Step: Run `docker compose up -d --build` and enjoy your app!