Guia de Testes de Segurança para a Aplicação

Este documento serve como uma checklist para verificar e melhorar a segurança da sua aplicação Flask.

1. Segurança no Backend (Flask)

A maior parte da lógica de segurança reside no servidor.

✅ 1.1. Validação de Entradas (Input Validation)

- **Risco:** Entradas maliciosas podem causar vários tipos de ataques, como Injeção de SQL (não aplicável aqui, pois não há BD SQL), Cross-Site Scripting (XSS) ou Negação de Serviço (DoS) ao enviar dados excessivamente grandes.
- **Mitigação no Código:**
  - Limitamos o tamanho máximo das strings de entrada nas rotas /clima e /mare. Se um utilizador enviar um nome de cidade com 1000 caracteres, o servidor irá rejeitá-lo com um erro 400 Bad Request em vez de tentar processá-lo.
- **Como Testar:**
  - Use ferramentas como o Postman ou curl para enviar pedidos às suas APIs com campos muito longos (ex: cidade com 500 caracteres). A aplicação deve retornar um erro 400 e não um erro 500 (erro interno).
  - Tente enviar caracteres especiais ou scripts nos campos, por exemplo, no campo cidade: Recife<script>alert('XSS')</script>. Embora o nosso frontend não vá executar isto, é uma boa prática o backend ser robusto.

✅ 1.2. Prevenção de Cross-Site Scripting (XSS)

- **Risco:** Se a sua API refletisse a entrada do utilizador de volta para o HTML sem "limpeza", um atacante poderia injetar scripts maliciosos que seriam executados no navegador de outros utilizadores.
- **Mitigação no Código:**
  - **Content Security Policy (CSP):** A biblioteca Flask-Talisman foi configurada com uma CSP estrita. Ela diz ao navegador para apenas carregar scripts e estilos de fontes confiáveis que nós definimos (unpkg.com para o mapa, por exemplo). Isso bloqueia a execução de scripts injetados de fontes desconhecidas.
  - **Frontend Seguro:** O seu JavaScript atual insere os dados da API usando .textContent ou .innerHTML com dados controlados, o que é mais seguro. No entanto, a CSP é a defesa mais forte.
- **Como Testar:**
  - Abra as ferramentas de programador do seu navegador (F12), vá para a aba

"Console" e verifique se não há erros de CSP. Tente injetar um script simples num campo de entrada e veja se a CSP o bloqueia (aparecerá um erro no console).

✅ 1.3. Cabeçalhos HTTP Seguros

- **Risco:** Cabeçalhos mal configurados podem expor a sua aplicação a ataques como *clickjacking* (onde um site malicioso carrega o seu site num iframe invisível para enganar os utilizadores) ou *MIME-sniffing*.
- **Mitigação no Código:**
  - Flask-Talisman faz todo o trabalho pesado. Ele adiciona automaticamente cabeçalhos como X-Frame-Options: SAMEORIGIN (previne clickjacking), X-Content-Type-Options: nosniff, e Strict-Transport-Security.
- **Como Testar:**
  - Use as ferramentas de programador do navegador (F12), vá para a aba "Network", recarregue a página, clique no pedido principal (o primeiro da lista) e inspecione os "Response Headers". Verifique a presença destes cabeçalhos de segurança.

✅ 1.4. Limitação de Taxa (Rate Limiting)

- **Risco:** Um atacante (ou um script) pode sobrecarregar o seu servidor fazendo milhares de pedidos em poucos segundos, causando uma Negação de Serviço (DoS) e tornando a sua aplicação indisponível para utilizadores legítimos.
- **Mitigação no Código:**
  - Flask-Limiter foi configurado para permitir apenas um número razoável de pedidos por minuto/hora de um mesmo endereço IP. Se o limite for excedido, o servidor responde automaticamente com um erro 429 Too Many Requests.
- **Como Testar:**
  - Num terminal, use um loop para fazer vários pedidos seguidos à sua API. Por exemplo, com curl:
    for i in {1..10}; do curl https://sua-app.onrender.com/clima?cidade=recife; echo; done
  - Após o 6º pedido num minuto, você deverá começar a receber o erro 429.

✅ 1.5. Gestão de Segredos

- **Risco:** Colocar chaves de API, senhas ou outros segredos diretamente no código é um risco enorme. Se o seu código for partilhado ou vazar, os seus segredos serão expostos.
- **Mitigação no Código:**
  - Movemos a OPENWEATHER_API_KEY para uma variável de ambiente

(os.environ.get(...)). Isto é exatamente o que fizemos para a hospedagem no Render.

- **Como Testar:**
  - Verifique se não há nenhuma chave ou senha "hardcoded" (escrita diretamente) em nenhum dos seus ficheiros (.py, .html, .js).

2. Segurança no Frontend (JavaScript)

✅ 2.1. Manipulação Segura de Dados da API

- **Risco:** Inserir dados de uma API diretamente no HTML com element.innerHTML pode ser perigoso se a API estiver comprometida ou se os dados contiverem scripts maliciosos.
- **Mitigação no Código:**
  - O seu JavaScript atual já é bom, mas a melhor defesa é a **Content Security Policy (CSP)** que configurámos no backend. Mesmo que um script malicioso seja inserido no HTML, a CSP impedirá que ele seja executado.

3. Segurança na Hospedagem (Render)

✅ 3.1. HTTPS

- **Risco:** Sem HTTPS, toda a comunicação entre o navegador do utilizador e o seu servidor é em texto plano, podendo ser interceptada.
- **Mitigação no Código:**
  - O Render configura automaticamente certificados SSL/TLS para a sua aplicação, garantindo que todo o tráfego seja encriptado (HTTPS).
- **Como Testar:**
  - Acesse o seu site e verifique se o navegador mostra um cadeado na barra de endereços.

✅ 3.2. Manter as Dependências Atualizadas

- **Risco:** Bibliotecas desatualizadas (como Flask, Requests, etc.) podem ter vulnerabilidades de segurança conhecidas.
- **Mitigação no Código:**
  - Use ferramentas de scan de segurança. O GitHub tem uma ferramenta integrada (Dependabot) que pode ser ativada no seu repositório. Ela irá notificá-lo automaticamente quando uma das suas dependências no requirements.txt tiver uma vulnerabilidade conhecida e até sugerir a atualização.
- **Como Ativar:**
  - No seu repositório do GitHub, vá para **Settings > Code security and**

analysis e ative o **Dependabot alerts**.

Seguir esta checklist e usar o código reforçado colocará a sua aplicação num patamar de segurança muito bom para um projeto deste tipo.