RELATÓRIO DE SEGURANÇA DA INFORMAÇÃO

Disciplina:Disciplina: Segurança da Informação
Professor:Professor: Gilvan Justino

Projeto de Software Seguro

v2.0

Sumário

Introdução

Este relatório apresenta a implementação de comunicação segura via HTTPS utilizando TLS na aplicação desenvolvida em Node.js. O objetivo é demonstrar a criação e instalação de um certificado digital, configuração do servidor para uso obrigatório de criptografia e validação da conexão segura no navegador.

A. Configuração do TLS

Esta seção descreve detalhadamente o processo de ativação do HTTPS na aplicação Node.js, incluindo a criação do certificado digital, configuração do servidor e o procedimento para instalação da Autoridade Certificadora (CA) no sistema operacional, garantindo que o navegador reconheça a conexão como segura.

---

1. Passos realizados para ativar HTTPS na aplicação

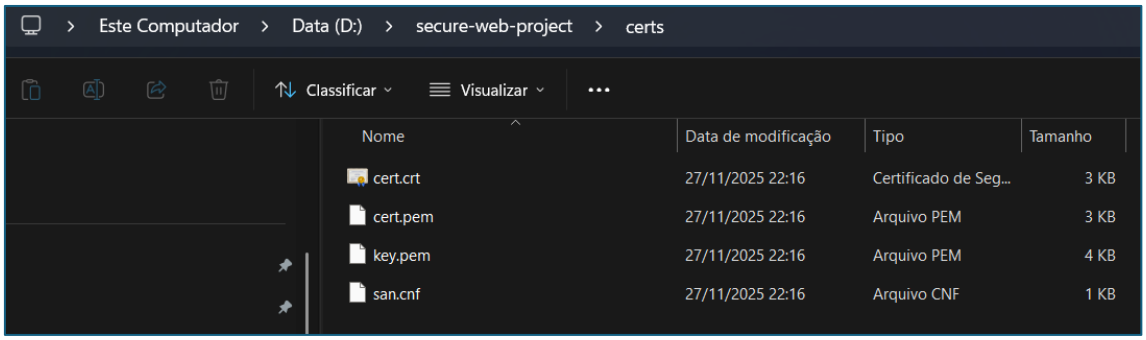
A aplicação originalmente operava sobre HTTP e foi adaptada para suportar o protocolo HTTPS utilizando TLS 1.2+.
O processo envolveu quatro etapas principais:

1. **Criação de um certificado digital autoassinado com suporte a CA raiz**
2. **Configuração do Node.js para carregar a chave privada e o certificado**
3. **Alteração de variáveis de ambiente (.env)**
4. **Implementação do servidor HTTPS nativo e redirecionamento automático do HTTP**

1.1 Criação do diretório de certificados

Antes da criação dos arquivos de chave e certificado, foi criado o diretório:

/secure-web-project/certs



B. Criação do Certificado

O certificado utilizado na aplicação foi gerado por meio do OpenSSL, permitindo que ele fosse usado como **Autoridade Certificadora (CA)** e simultaneamente como certificado de servidor.

A escolha por esse modelo simplifica o ambiente de desenvolvimento local e facilita a aprovação do navegador, desde que a CA seja instalada manualmente no sistema operacional.

2.1 Criação do certificado digital (com arquivo san.cnf)

Durante o desenvolvimento, foi identificado que o Git Bash no Windows converte automaticamente caminhos iniciados com "/C=" em caminhos do sistema (ex.: C:/Program Files/Git/...). Isso impossibilita o uso do parâmetro -subj no formato tradicional para gerar certificados, resultando em erro no OpenSSL.

Para contornar esse comportamento e garantir conformidade com os requisitos atuais do TLS (incluindo SAN – *Subject Alternative Name*), optou-se pelo método recomendado de utilizar um arquivo de configuração dedicado.

Navegadores modernos (Chrome, Edge, Firefox) rejeitam certificados sem SAN (Subject Alternative Name), mesmo que o CN esteja correto. Por esse motivo, o arquivo san.cnf inclui explicitamente o DNS:localhost

Criação do arquivo san.cnf

Foi criado o arquivo:

/secure-web-project/certs/san.cnf

Com o conteúdo:

[req]

default_bits = 4096

prompt = no

default_md = sha256

distinguished_name = req_distinguished_name

x509_extensions = v3_ca

[req_distinguished_name]

C = BR

ST = SC

L = Blumenau

O = MeuAmbiente

OU = Dev

CN = localhost

[v3_ca]

subjectAltName = @alt_names

basicConstraints = critical, CA:true, pathlen:0

keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[alt_names]

DNS.1 = localhost

Esse arquivo contém todas as informações necessárias do certificado, incluindo:

- Dados da organização
- Identidade do servidor (**Common Name = localhost**)

- Extensão SAN obrigatória para navegadores modernos
- Parâmetros de CA raiz (CA:TRUE)

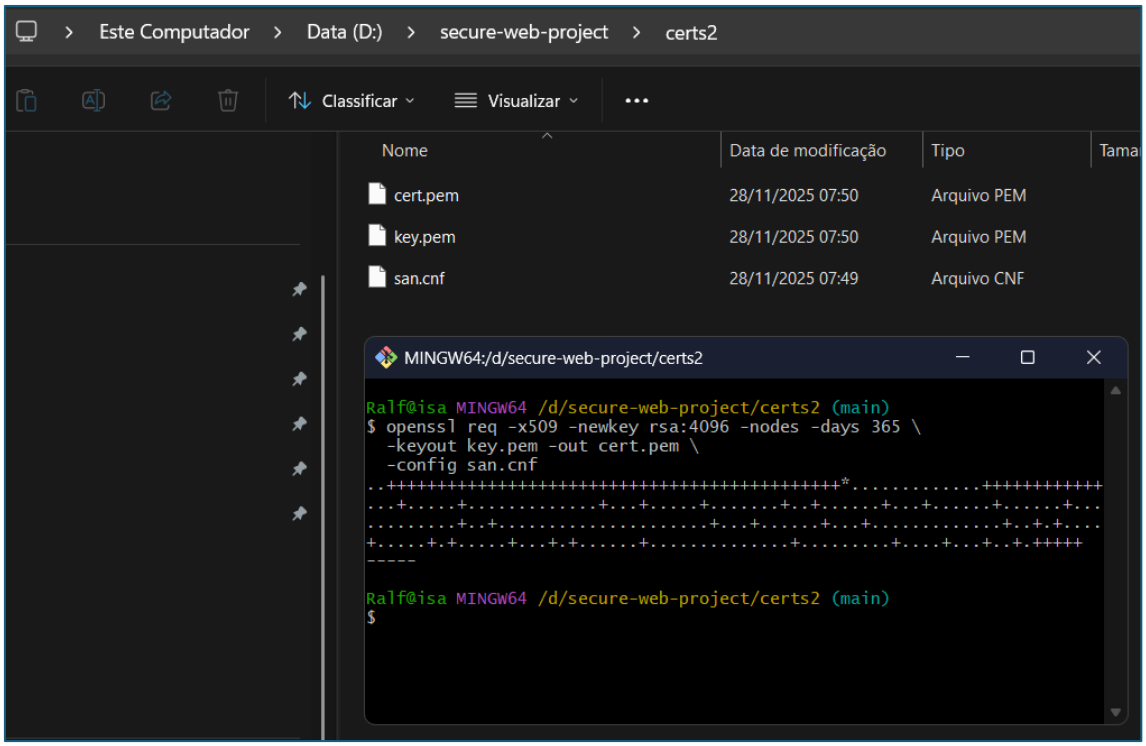
Comando utilizado para gerar o certificado

Com o arquivo criado, o certificado e a chave privada foram gerados com o comando:

```
openssl req -x509 -newkey rsa:4096 -nodes -days 365 \
 -keyout key.pem -out cert.pem \
 -config san.cnf
```

Esse processo produz dois arquivos:

- **key.pem** → chave privada RSA de 4096 bits
- **cert.pem** → certificado digital no padrão X.509, contendo:
    - campo *Issuer* adequado
    - SAN configurado
    - CA:TRUE
    - validade de 1 ano



2.2 Verificação do certificado

Após a geração, foi validado com:

openssl x509 -in certs/cert.pem -text -noout

Confirmando que:

- O certificado possui **Basic Constraints: CA:TRUE**
- Pode ser importado como **Autoridade Certificadora Raiz**
- É válido por 1 ano
- Pertence ao host **localhost**

Ralf@isa MINGW64 /d/secure-web-project/certs2 (main)

$ openssl x509 -in cert.pem -text -noout

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

06:cf:b5:8c:b7:27:bf:eb:4a:61:6a:44:ac:78:29:ae:01:df:04:16

Signature Algorithm: sha256WithRSAEncryption

Issuer: C=BR, ST=SC, L=Blumenau, O=MeuAmbiente, OU=Dev, CN=localhost

Validity

Not Before: Nov 28 10:50:22 2025 GMT

Not After : Nov 28 10:50:22 2026 GMT

Subject: C=BR, ST=SC, L=Blumenau, O=MeuAmbiente, OU=Dev, CN=localhost

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (4096 bit)

Modulus:

00:bb:47:49:94:22:ce:81:db:c5:14:ec:a6:bb:fc:

76:0b:cc:06:f9:c3:55:f7:50:f2:ae:d4:75:aa:15:

77:c3:40:df:f5:2a:8a:17:45:91:60:b0:0c:63:3e:

fb:16:e2:3c:38:c0:20:4f:4d:b5:42:73:2d:4a:55:

65:85:4c:16:b4:1d:4b:fe:7b:bb:9f:09:b1:9d:d1:

f1:66:b3:1a:d2:70:20:b3:be:4d:c9:46:55:94:a2:

05:7a:93:09:08:b2:d5:d8:31:53:8d:eb:48:f8:36:

54:b4:a8:86:c2:e5:c3:ee:c2:90:59:6f:78:ec:4a:

3e:f4:cf:fe:ae:5f:36:f6:ab:50:38:4e:25:84:af:

fe:20:65:86:e2:82:43:eb:4d:3e:8f:bb:3b:6e:8a:

1d:d4:22:df:74:a9:54:9f:ee:53:9d:0d:c9:fa:8c:

2e:fa:5d:dc:8c:33:73:b7:56:e9:7a:1b:d6:18:9a:

38:08:f6:ad:81:02:48:53:49:45:a3:4f:30:12:00:

54:7d:7f:e3:be:01:66:f0:90:c5:56:f3:01:7d:53:

28:9f:7e:d6:4d:9a:50:4e:1f:f7:50:6d:99:58:d0:

66:f3:18:93:90:16:8a:cc:da:a9:9c:6c:e5:45:ae:

58:79:55:53:33:a1:cd:a8:a5:09:36:9d:75:3a:f4:
6b:10:b4:c4:23:21:77:f7:5d:83:44:28:d8:fe:3a:
91:d6:1d:b2:f9:70:8a:89:a5:bf:30:25:3f:87:82:
a6:7e:26:6e:88:bc:2a:9a:f9:a0:c6:ec:87:49:45:
82:56:7a:2f:d5:51:4b:63:83:56:26:15:c9:80:b9:
e8:6c:d5:55:92:50:d2:db:27:eb:e3:42:c2:b5:f9:
ea:99:07:25:d5:8f:ad:8a:4c:ea:9c:2b:26:ae:d3:
2e:64:12:c5:52:b3:c0:9e:0b:f5:55:5b:5e:51:8c:
2a:1d:41:b3:49:cf:6e:37:2a:a4:d8:b0:48:ae:cb:
0d:5e:cf:19:ee:8a:73:3f:26:d1:2a:a5:bf:75:15:
53:b1:d6:00:6e:16:7c:e6:46:67:d7:4e:6c:1b:88:
9b:4e:cb:d0:18:09:53:56:6c:08:af:cc:1c:2c:8f:
71:0e:76:83:f2:d8:1b:4e:cf:d0:4b:d0:0f:1e:a2:
6a:8d:4e:d1:30:9d:81:58:a5:08:c4:88:ac:ae:92:
92:97:d7:a6:62:7f:6a:fc:a4:a3:3f:2a:51:26:b0:
7a:9d:d7:a7:73:58:d2:85:c0:c0:f6:ee:f4:2c:e8:
8a:a6:23:72:06:43:22:43:4e:2c:16:4e:dd:12:e1:
bd:50:2f:5e:30:56:b5:f3:6e:2a:77:31:d6:be:1c:
55:bb:29

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Subject Alternative Name:

DNS:localhost

X509v3 Basic Constraints: critical

CA:TRUE, pathlen:0

X509v3 Key Usage: critical

Digital Signature, Certificate Sign, CRL Sign

X509v3 Subject Key Identifier:

24:C0:D6:56:74:4C:FC:50:84:5A:13:EB:7B:C5:8D:CA:8E:C4:41:33

Signature Algorithm: sha256WithRSAEncryption

Signature Value:

72:6e:9a:b7:ec:6c:15:21:39:9b:8f:fd:fa:a8:ac:26:af:9a:
d1:4a:3b:a0:46:8c:39:d1:d6:5c:b0:34:5f:91:0e:4b:26:51:
05:dc:90:eb:26:8d:c8:99:82:2d:18:e9:62:dd:41:9e:aa:23:
36:71:79:fe:5d:19:e3:f8:65:22:90:83:5a:71:26:6b:c0:bf:
77:eb:1c:1e:97:f7:9b:a4:cb:75:32:6b:eb:d7:3d:3f:c7:47:
e8:a6:58:ca:f1:a7:73:ec:f5:90:6d:2d:4b:2f:7a:7e:3a:a2:
0f:20:a1:82:c7:4e:d0:37:fd:31:44:40:98:97:bd:59:ba:eb:
bb:4d:f9:ee:fe:f7:35:f2:a6:ca:9f:57:07:fe:34:c0:15:21:
28:60:87:71:9e:34:71:87:6d:3f:f3:f1:5f:71:05:25:fa:4e:
4a:58:72:8e:10:ec:2c:e8:88:28:35:9d:45:71:b5:3e:d0:b2:
56:4c:8c:98:f8:b1:48:c3:d2:da:c5:84:a8:72:19:b4:53:6c:
8d:15:09:21:74:4c:c4:b1:cc:29:41:2a:2b:cb:6a:63:e8:cc:
8d:37:ac:ba:c9:99:8e:a6:d3:53:09:c7:81:b8:c8:8d:14:9c:
b3:54:65:87:cc:0c:a0:19:f5:6e:16:8a:78:11:ba:fe:44:e4:
cd:90:5d:c6:7b:78:7d:64:89:e9:e8:e6:88:67:0d:a5:b3:55:
2b:eb:87:eb:6f:60:96:58:b8:e8:9b:2e:b8:cc:7c:4b:31:0d:
35:c7:26:5c:b6:86:cc:ba:a3:99:82:24:09:7a:42:f3:1e:f2:
b2:cf:95:35:36:83:66:03:ef:1f:74:91:b4:99:69:1a:03:5e:
1a:9f:d2:a3:d1:b7:db:76:67:cc:d6:3f:bb:0d:3c:7f:6e:42:
f3:ab:65:5d:39:3b:41:0d:a5:fb:01:a3:e0:a8:22:d8:d6:de:
43:b5:e2:55:4c:3b:54:ab:b5:2e:72:1a:2b:d7:a7:64:50:77:
f1:33:87:43:ea:fb:5e:8d:dc:31:6f:97:8e:da:34:81:36:65:
8f:39:8b:90:01:5a:a7:ad:a2:48:ea:97:1f:91:7d:20:80:42:
59:18:74:59:52:49:2e:fe:24:d0:27:a3:b8:e3:02:28:5c:49:
0a:65:d3:5e:c2:03:13:c1:49:f5:97:e4:29:cf:14:30:ce:5b:

fb:71:96:68:6d:34:03:b8:01:a6:ae:b2:f6:8b:1f:71:62:0f:

89:93:d8:4e:6f:d6:fb:1f:1a:11:f8:fc:f7:78:b0:37:af:a6:

65:40:be:af:f3:57:94:d8:99:6c:ef:7e:59:86:7a:8c:fc:49:

41:0f:cc:1d:18:60:ca:e2

C. Configuração do Servidor

Após gerar o certificado, a aplicação foi ajustada para operar exclusivamente em HTTPS.

3.1 Ajuste das variáveis de ambiente

No arquivo .env, foram adicionados os caminhos do certificado e da chave:

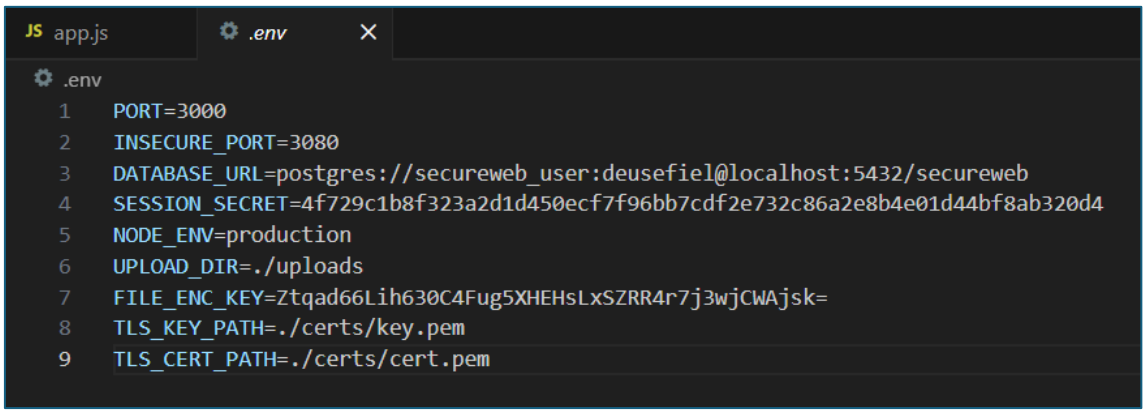
TLS_KEY_PATH=./certs/key.pem

TLS_CERT_PATH=./certs/cert.pem

Também foi mantida a porta HTTPS (3000) e a porta HTTP auxiliar (3080) para redirecionamento:

PORT=3000

INSECURE_PORT=3080



3.2 Implementação do servidor HTTPS no app.js

O backend foi configurado para iniciar dois servidores:

Servidor HTTPS (principal)

Utiliza os arquivos key.pem e cert.pem carregados via fs.readFileSync():

```
const httpsOptions = {
  key: fs.readFileSync(process.env.TLS_KEY_PATH),
  cert: fs.readFileSync(process.env.TLS_CERT_PATH),
};

https.createServer(httpsOptions, app)
  .listen(PORT, () => {
    console.log(`HTTPS server running at https://localhost:${PORT}`);
  });
```

Servidor HTTP (somente redirecionamento)

```
http.createServer((req, res) => {
  res.writeHead(301, { Location: `https://localhost:${PORT}${req.url}` });
  res.end();
}).listen(INSECURE_PORT);
```

Esse servidor secundário garante que **todas as requisições HTTP sejam automaticamente redirecionadas para HTTPS**, atendendo o requisito de "operar obrigatoriamente sobre HTTPS".

D. Instalação da CA Raiz no Windows / Navegador

Para que o navegador reconhecesse o certificado como confiável, foi necessário importar o certificado gerado (cert.pem, convertido para .crt) como **Autoridade Certificadora Raiz Confiável**.

Como o certificado é auto assinado, ele não pertence a nenhuma CA pública confiável. Por isso, para evitar avisos de "Conexão não segura", a CA gerada localmente deve ser instalada manualmente no Windows como Autoridade de Certificação Raiz Confiável.

4.1 Conversão para .crt

O Windows reconhece melhor certificados no formato CRT, então foi feito:

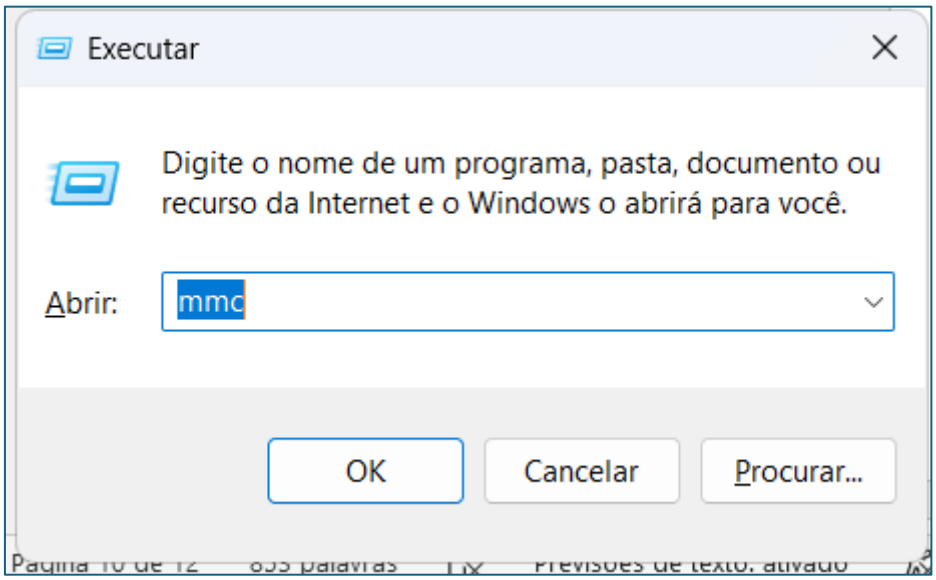
openssl x509 -in certs/cert.pem -out certs/cert.crt

4.2 Instalação da CA Raiz no Windows (via MMC)

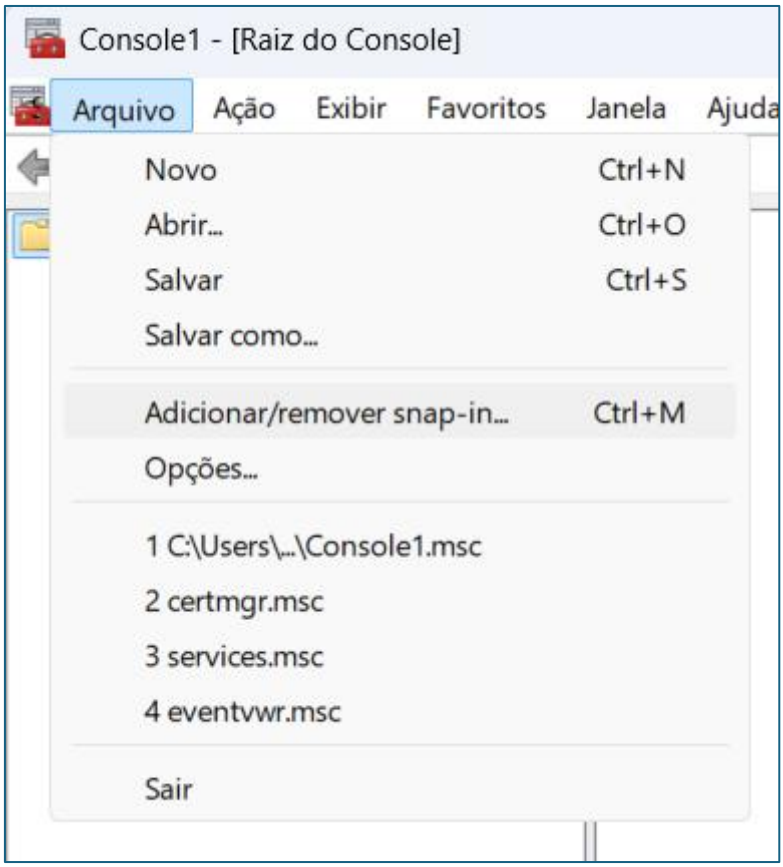
Passos realizados:

Pressionar **Windows + R**

Digitar: mmc

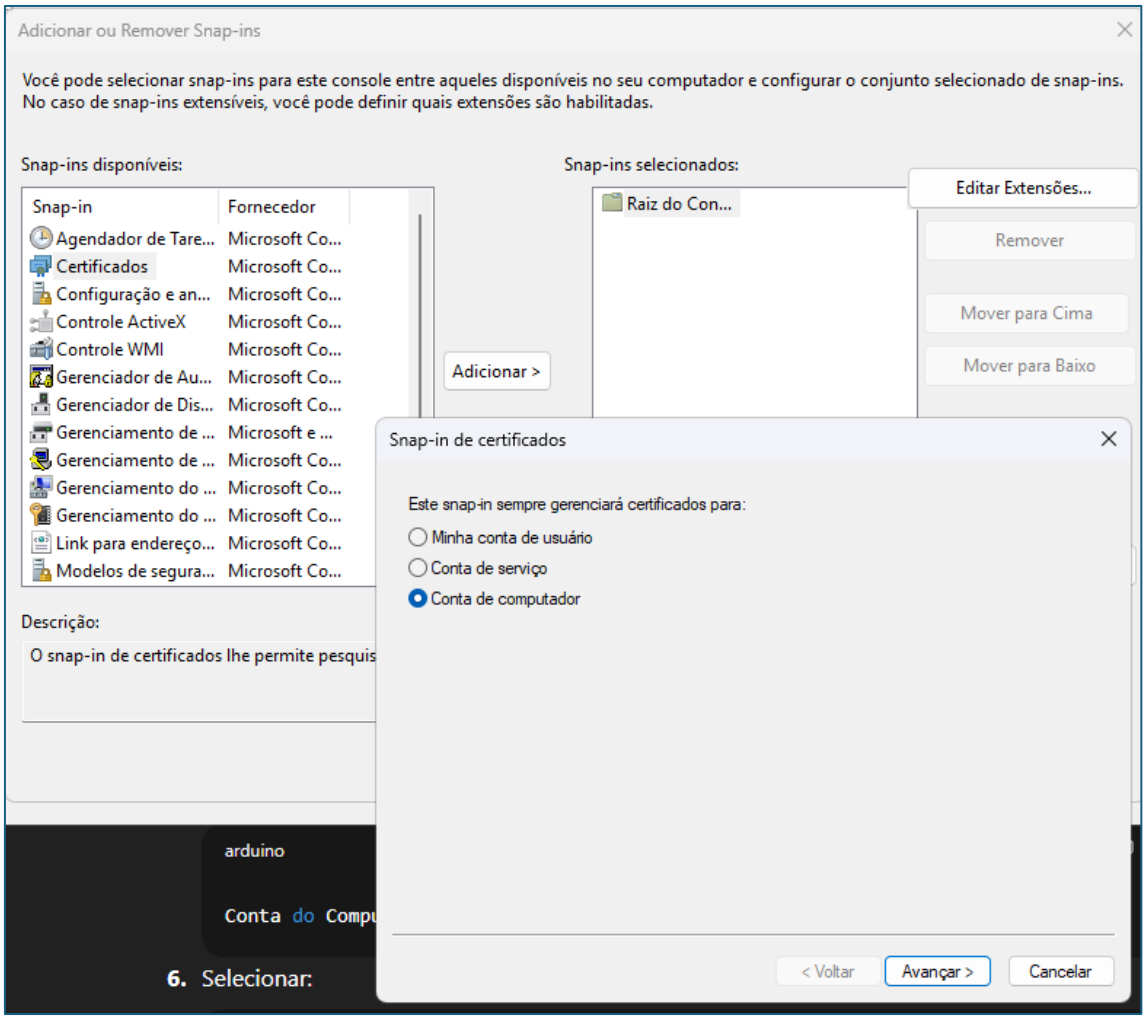


Em **Arquivo → Adicionar/Remover Snap-in...**

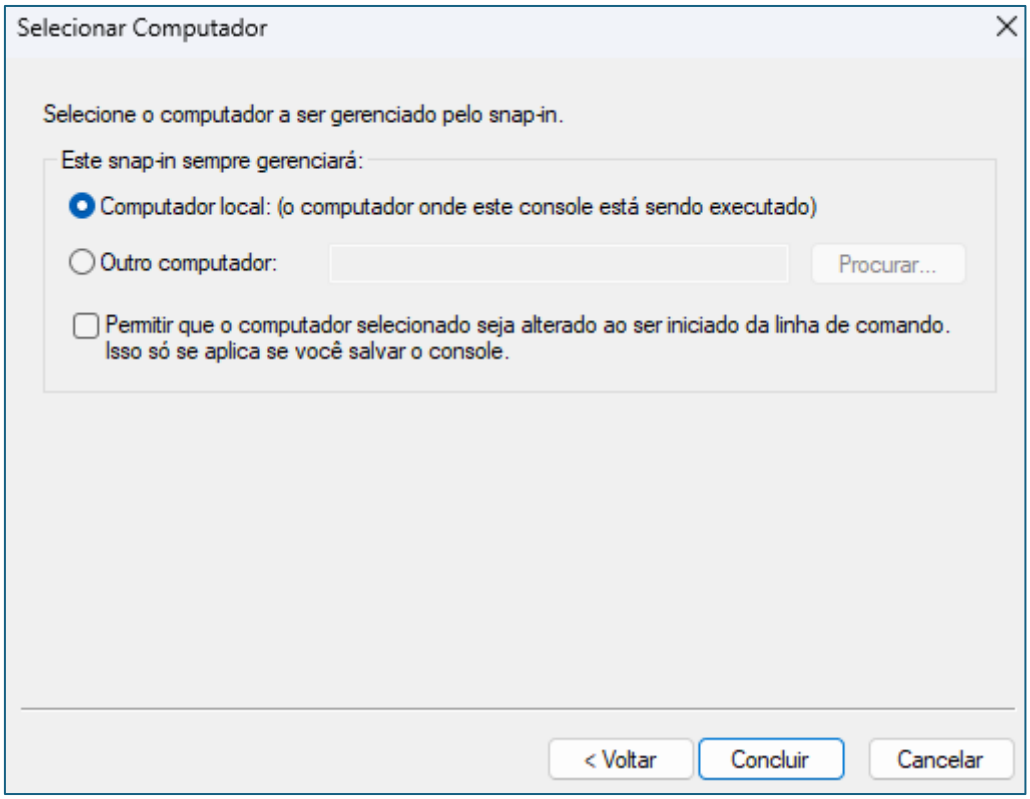


Selecionar **Certificados**

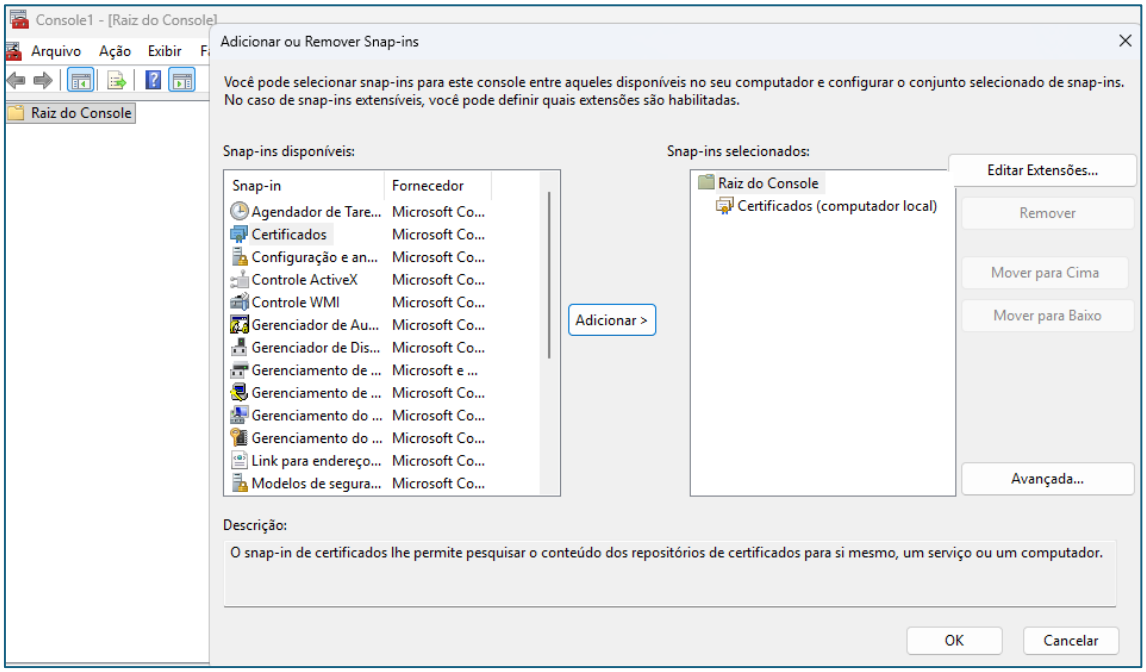
Escolher: Conta do Computador



Selecionar: Computador Local



Dê um concluir e dê um ok



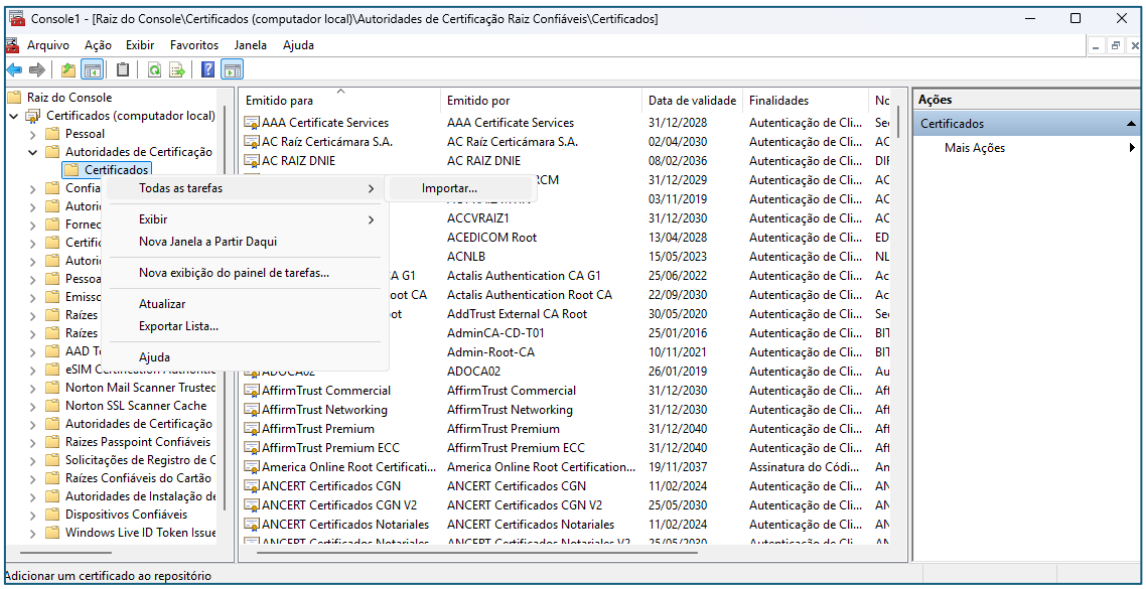
Na nova janela que abrir

Navegar até:

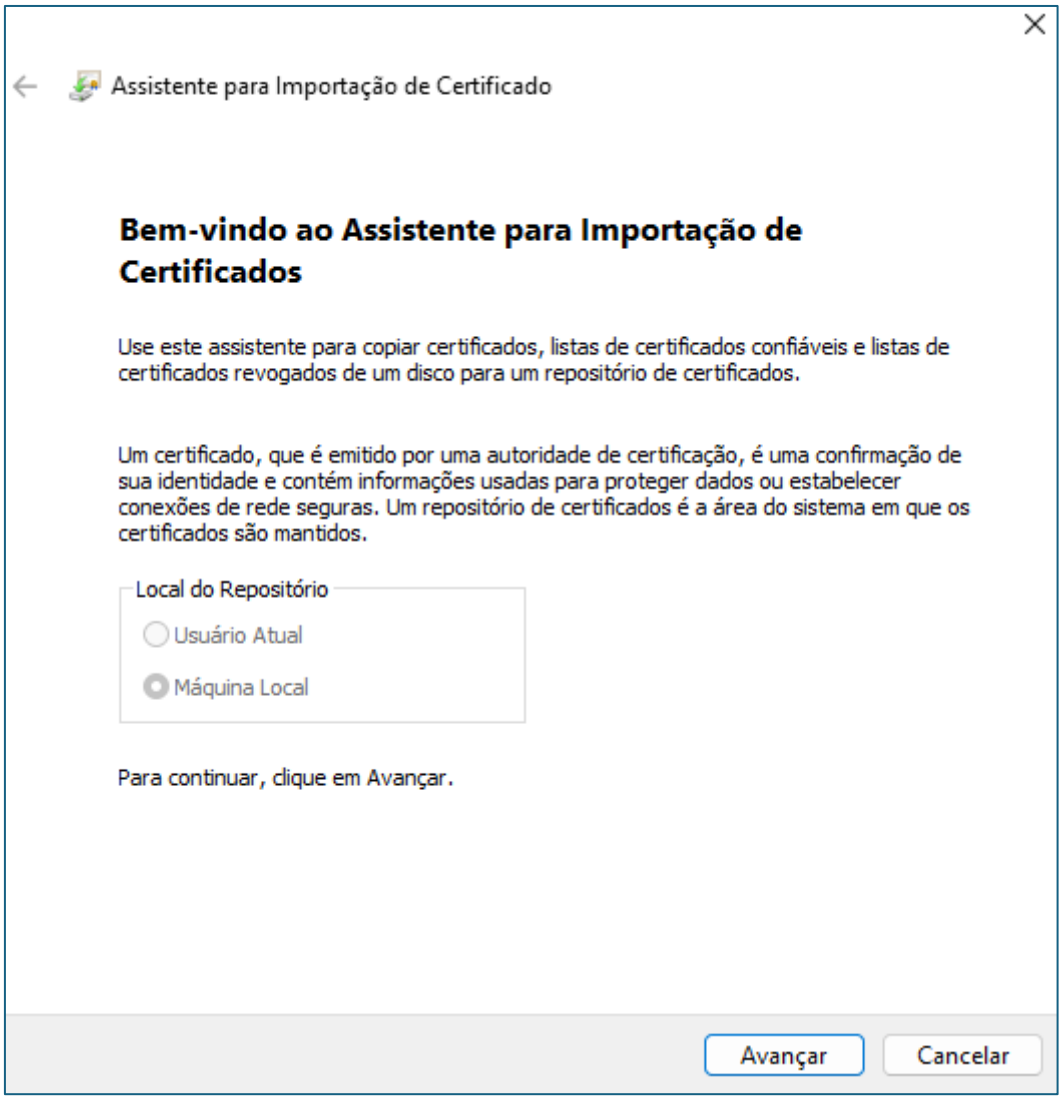
Autoridades de Certificação Raiz Confiáveis → Certificados

Botão direito → **Todas as Tarefas → Importar**

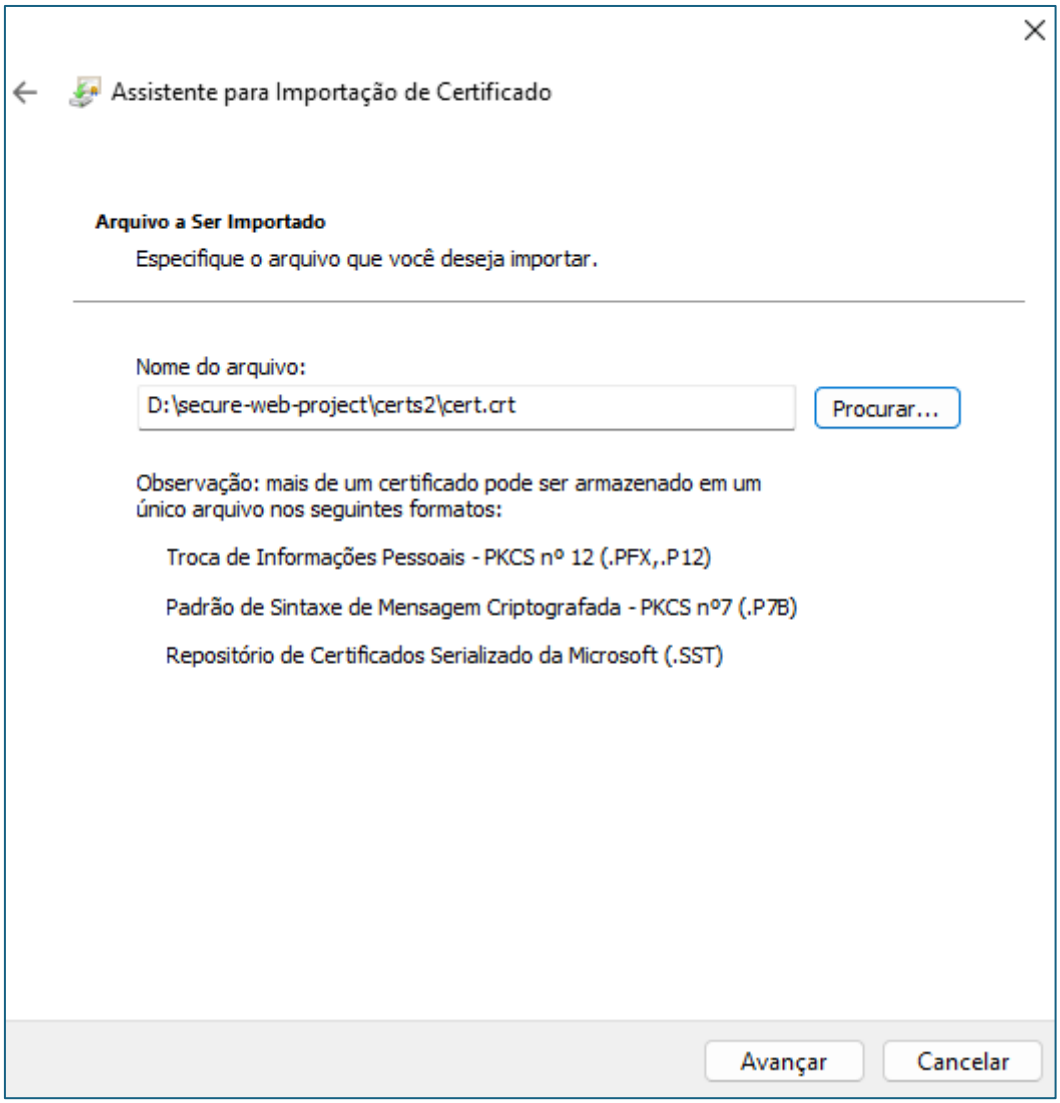
Selecionar o arquivo:



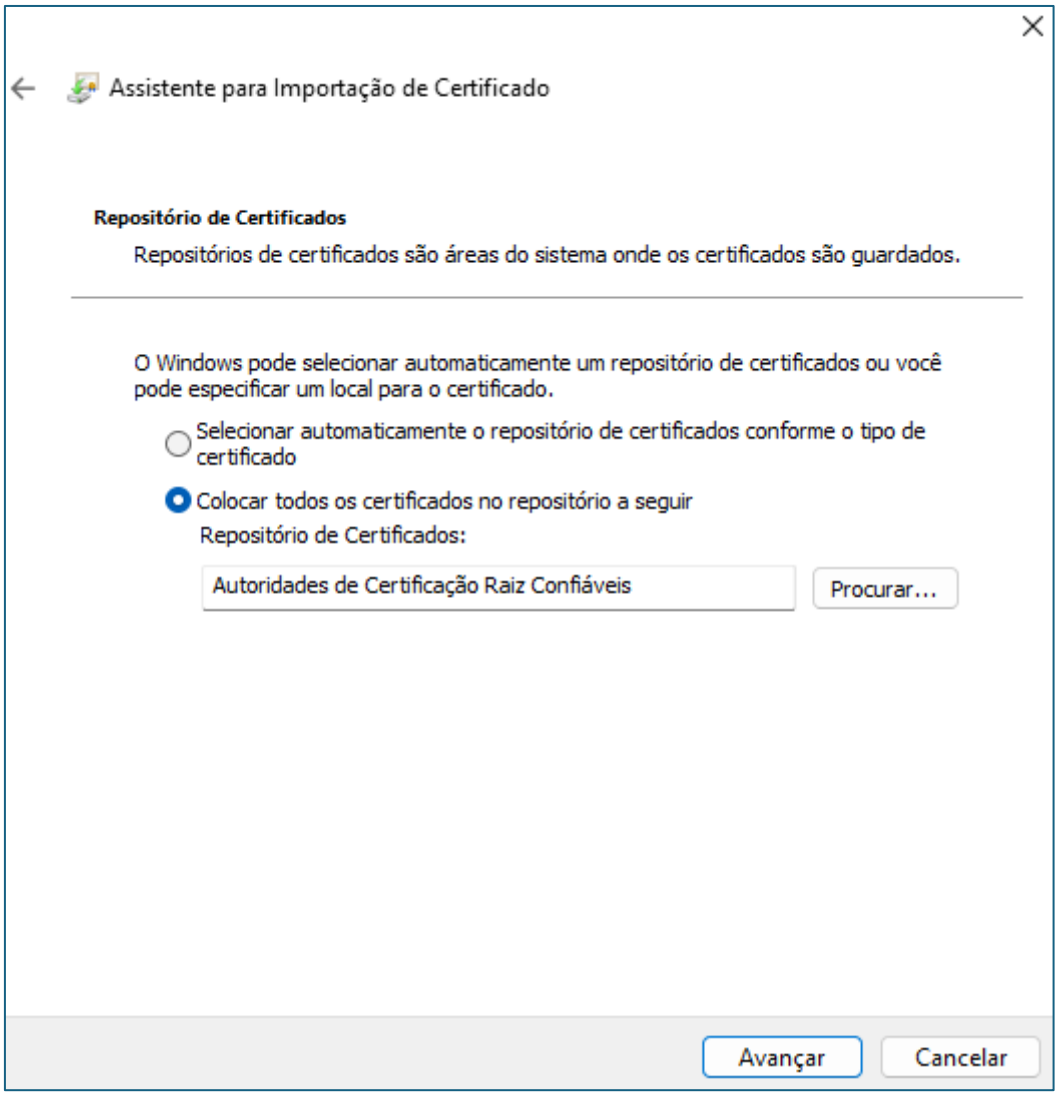
Dê um avançar



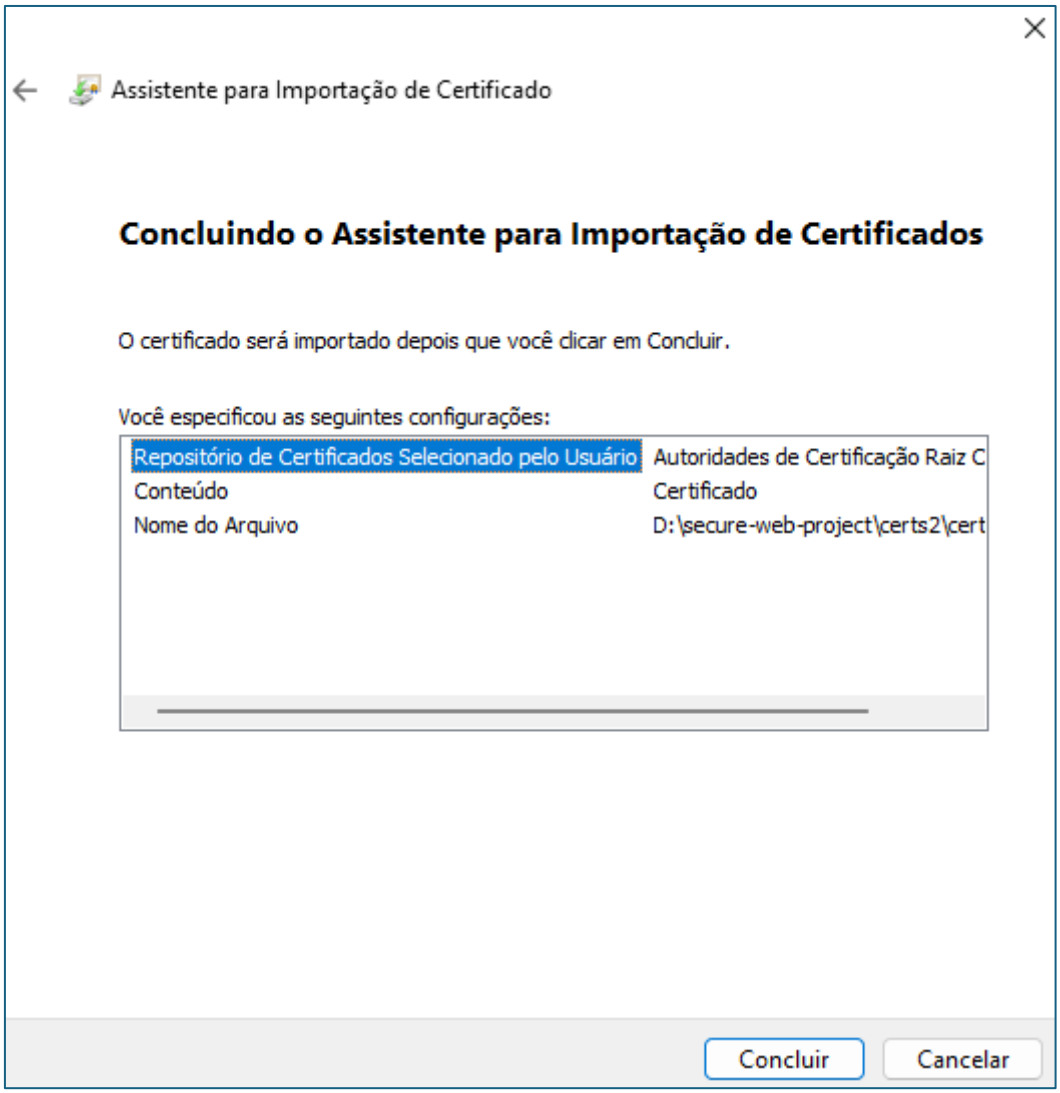
Importe o arquivo crt gerado



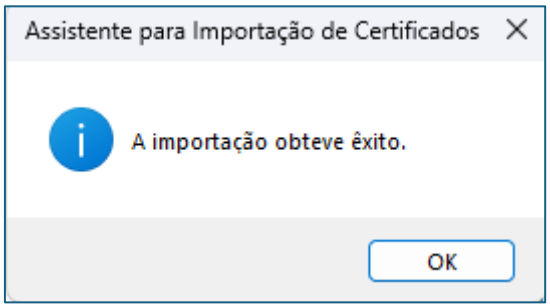
Configure para guardar na raiz de certificados



Dê um concluir



Vai aparecer uma mensagem de êxito, basta dar um ok



4.3 Validação pelo navegador

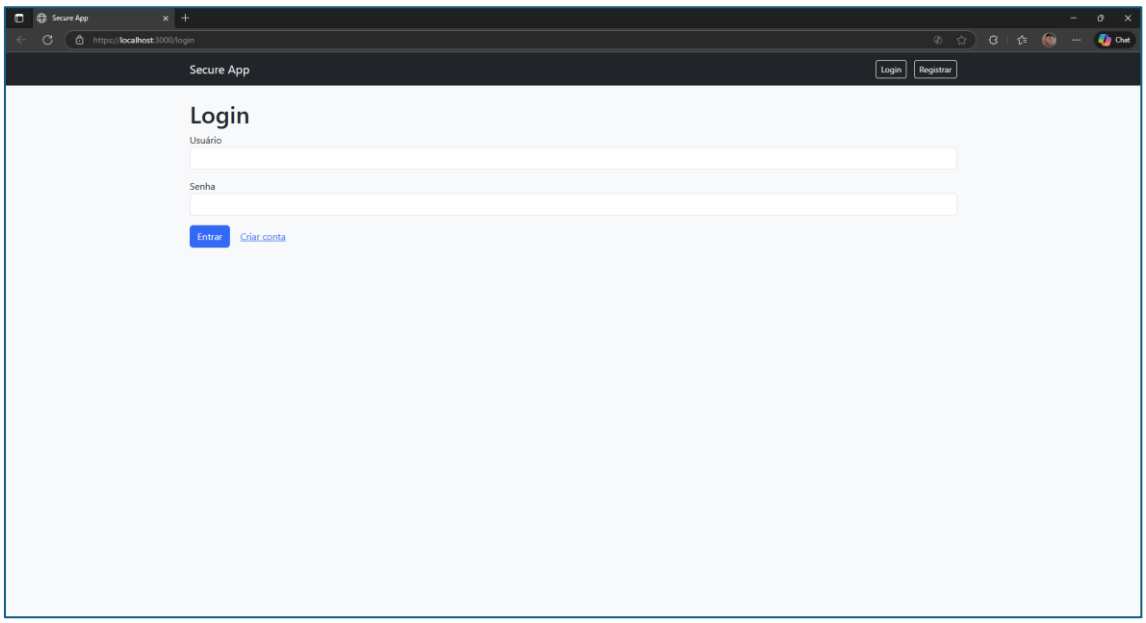
Após importar a CA, foi necessário:

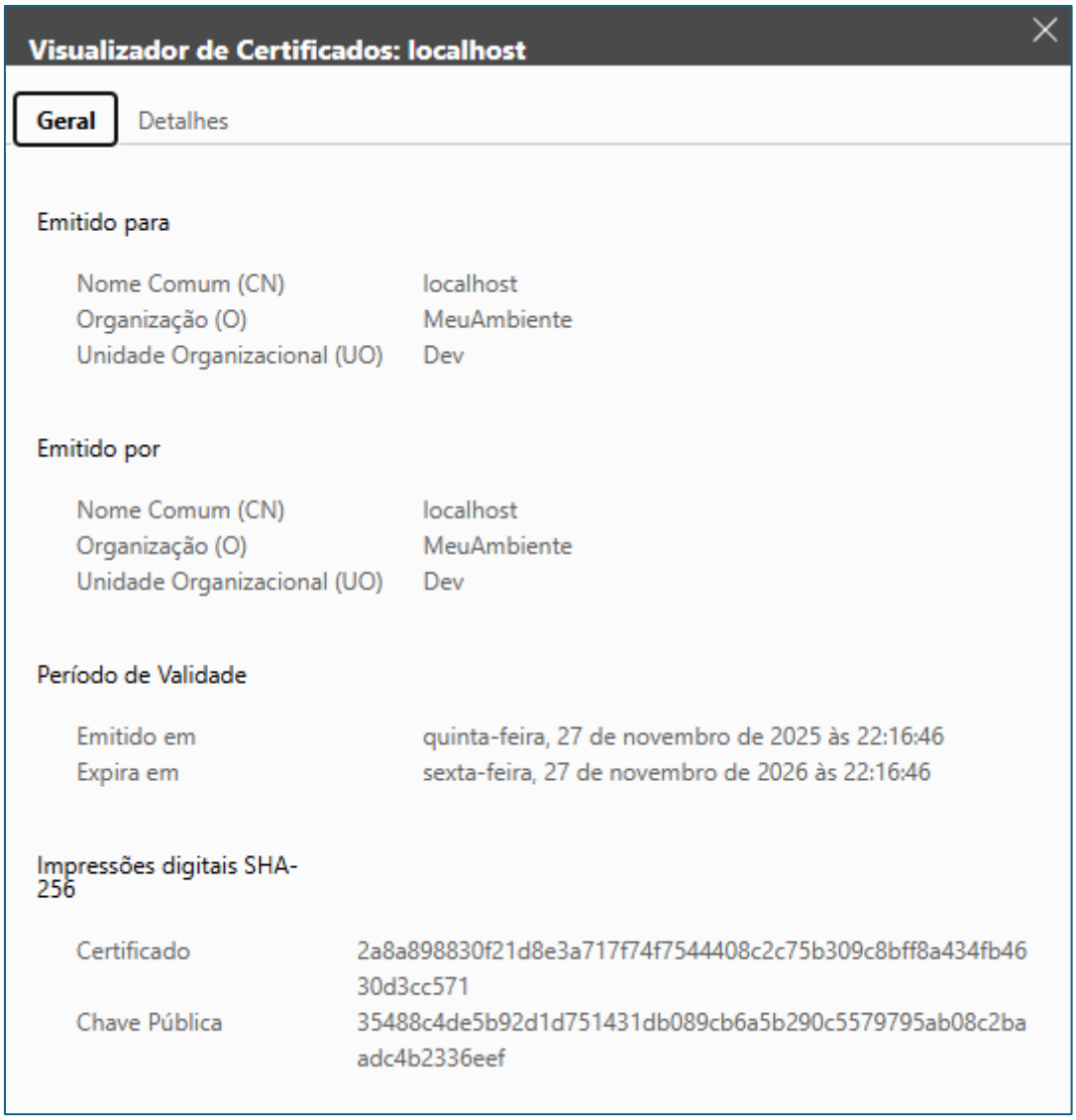
- fechar e reabrir o navegador
- acessar:

https://localhost:3000

O cadeado apareceu corretamente, confirmando que:

O certificado foi aceito
A CA está confiável
O TLS está funcionando sem alertas
A criptografia ponta-a-ponta está ativa





Conclusão

Com todos os passos concluídos — criação da CA, geração do certificado, configuração do servidor HTTPS e instalação no Windows — a aplicação passou a operar com criptografia TLS válida, exibindo cadeado verde no navegador e garantindo comunicação segura ponta-a-ponta.

E. Criptografia dos Arquivos Recebidos

Além da implementação do protocolo TLS, a aplicação foi aprimorada para garantir a proteção dos arquivos enviados ao servidor. A partir desta versão, todos

os arquivos de upload passam obrigatoriamente por criptografia antes de serem armazenados, e são descriptografados dinamicamente no momento do download, atendendo integralmente ao requisito de proteção de dados em repouso.

1. Geração e configuração da chave criptográfica

Para garantir segurança adequada, a aplicação utiliza o algoritmo **AES-256-GCM**, que oferece:

- criptografia autenticada (integridade + confidencialidade)
- chave de 256 bits (32 bytes)
- geração de IV único por arquivo
- tag de autenticação para detectar corrupção ou adulteração

A chave é gerada pelo comando:

node -e "console.log(require('crypto').randomBytes(32).toString('base64'))"

Essa chave é então armazenada na variável de ambiente:

FILE_ENC_KEY=<valor-base64>

Essa chave **não fica no código-fonte**, atendendo boas práticas de segurança.

2. Processo de Criptografia no Upload

Quando um arquivo é enviado pelo usuário:

1. Ele é carregado na memória (sem ser salvo em disco ainda).
2. O servidor gera um **IV aleatório de 12 bytes**.
3. O conteúdo é criptografado com **AES-256-GCM**.
4. O arquivo é gravado no disco no formato:
5. [IV (12 bytes) | ciphertext | authTag (16 bytes)]

   A extensão é alterada para .enc, indicando que o conteúdo está cifrado.

Armazenamento cifrado

O arquivo armazenado no diretório:

/uploads/<userId>/<arquivo>.enc

fica completamente ilegível, apresentando apenas bytes aleatórios.
Esse comportamento é demonstrado no vídeo solicitado na entrega.

---

3. Processo de Descriptografia no Download

Quando o usuário solicita o download:

1. O servidor lê o arquivo .enc.
2. Extrai o IV e a tag de autenticação.
3. Carrega a chave da variável de ambiente.
4. Reconstrói o texto original utilizando **AES-256-GCM**.
5. Envia o arquivo já legível ao usuário.

Nenhum arquivo descriptografado é armazenado em disco, apenas transmitido.

Este fluxo garante:

- **confidencialidade dos dados em repouso**
- **proteção contra adulteração**
- **privacidade entre diferentes usuários**

F. Códigos-Fonte Implementados

Nesta seção são apresentados apenas os trechos essenciais do projeto que implementam as funcionalidades solicitadas.

1. Carregamento da chave e funções de criptografia

```
// Carregamento da chave AES-256-GCM (32 bytes base64)
const FILE_ENC_KEY = Buffer.from(process.env.FILE_ENC_KEY, 'base64');

// Funções de criptografia e descriptografia
function encryptBufferToFile(buf, destPath) {
    const iv = crypto.randomBytes(12);
    const cipher = crypto.createCipheriv('aes-256-gcm', FILE_ENC_KEY, iv);
    const encrypted = Buffer.concat([cipher.update(buf), cipher.final()]);
    const authTag = cipher.getAuthTag();
    fs.writeFileSync(destPath, Buffer.concat([iv, encrypted, authTag]));
}

function decryptFileToBuffer(srcPath) {
    const data = fs.readFileSync(srcPath);
    const iv = data.slice(0, 12);
    const authTag = data.slice(data.length - 16);
    const ciphertext = data.slice(12, data.length - 16);
    const decipher = crypto.createDecipheriv('aes-256-gcm', FILE_ENC_KEY, iv);
    decipher.setAuthTag(authTag);
    return Buffer.concat([decipher.update(ciphertext), decipher.final()]);
}
```

2. Criptografia durante o processo de upload

```
app.post('/upload', upload.single('file'), async (req, res) => {
    const dest = path.join(userDir, uuidv4() + '.enc');
    encryptBufferToFile(req.file.buffer, dest);
    // grava no banco e redireciona
});
```

3. Descriptografia durante o download

```
app.get('/files/:id', async (req, res) => {
    const decrypted = decryptFileToBuffer(filePath);
    res.setHeader('Content-Disposition', `attachment; filename="${originalName}"`);
    res.send(decrypted);
});
```

Conclusão – Criptografia de Arquivos

A implementação da criptografia de arquivos adicionou uma camada essencial de segurança ao sistema, garantindo que todos os dados armazenados no servidor permaneçam protegidos, mesmo em cenários de acesso indevido ao ambiente físico ou lógico.
O uso de **AES-256-GCM**, aliado ao gerenciamento seguro de chaves via variáveis de ambiente, assegura confidencialidade, integridade e autenticidade dos arquivos.

O processo de criptografar no upload e descriptografar apenas no momento do download garante que:

- não existam versões em texto puro dos arquivos no disco,
- cada arquivo esteja protegido individualmente com IV exclusivo,
- a aplicação respeite boas práticas modernas de segurança da informação,
- usuários diferentes jamais acessem dados de outros por falhas de configuração ou vazamento.

Essa implementação cumpre plenamente o requisito de "proteção de dados em repouso" e complementa o uso de TLS, que protege os mesmos dados **em trânsito**. Juntas, essas duas camadas elevam significativamente o nível de segurança do sistema.

Conclusão Geral do Trabalho

A evolução do projeto para suportar comunicação segura e proteção de dados em repouso demonstrou, na prática, a aplicação dos principais pilares da Segurança da Informação: **confidencialidade, integridade, autenticidade e proteção contra acessos não autorizados**.

A ativação do protocolo **HTTPS com TLS 1.2+**, a criação de uma **Autoridade Certificadora local**, a instalação do certificado raiz no sistema operacional e o redirecionamento obrigatório de HTTP para HTTPS garantiram uma comunicação criptografada e confiável entre cliente e servidor, eliminando riscos de interceptação e adulteração de dados durante o transporte.

Complementando essa camada de proteção, a implementação da criptografia de arquivos com **AES-256-GCM** assegurou que todos os arquivos armazenados no servidor permanecessem protegidos mesmo em repouso, preservando sua integridade e confidencialidade. A divisão clara dos processos de upload

criptografado e download com descriptografia garantiu privacidade e isolamento entre os usuários.

Como resultado, o sistema passou a contar com:

- proteção em trânsito (TLS),
- proteção em repouso (criptografia de arquivos),
- validação explícita no navegador (cadeado HTTPS),
- chave de criptografia gerenciada de forma segura,
- arquitetura mais robusta e alinhada com boas práticas da indústria.

O projeto, em sua versão atualizada, cumpre todos os requisitos propostos e demonstra a importância de integrar técnicas de segurança ao ciclo de desenvolvimento, refletindo diretamente nos princípios de um software seguro, confiável e preparado para cenários reais.