Segurança da informação

Conceitos e aplicações na API

Prof. João Pedro Parella

Introdução

Segurança se mostra cada vez mais importante no nosso dia a dia

- Brasil ocupa o 12º lugar no ranking de vazamentos de dados
- Cerca de 286mil brasileiros tiveram dados expostos no inicio de 2022[1]

[1]https://proximonivel.embratel.com.br/brasil-ocupa-12o-lugar-no-ranking-de-vazamento-de-dados/

Como vazaram?

Em sua maioria, os dados são expostos por vazamentos ou por brechas de segurança

Entre os principais pontos estão:

Empresas estatais e órgãos públicos

Invasões

Vulnerabilidades

Importância da segurança

LGPD – Lei geral de proteção de dados

- Lei nº 13.709/2018
- Essa lei regulamenta o tratamento e manipulação de dados de usuários
- Sendo necessário sempre ter cuidado ao tratar esses dados
- Em caso de qualquer tipo de vazamento, pode ser gerada multas de grande valor, podendo comprometer um projeto

Confiança

Uma vez perdida... Difícil recuperar

- Uma empresa de tecnologia tem como objetivo principal, tratar dados
- Caso essa empresa tenha seus dados vazados, ela vai perder a confiança do mercado
- Como confiar numa empresa que teve dados expostos? Como vou colocar meus dados pessoais e senhas num lugar que sei que pode vazar tudo ?

E muito mais...

- Poderia ficar horas dando motivos e mais motivos para ter atenção a segurança dos dados
- Pois são muitos motivos, muitos pontos que devem ser abordados e que devem ter atenção
- Com tudo isso, é de extrema necessidade que sejam sempre analisados os pontos críticos de uma aplicação
- Após essa análise é preciso pensar em como podemos fechar as brechas e furos de segurança

- Primeiro vamos pensar em alguns pontos críticos da nossa aplicação

Usuario Controller

Vamos analisar os métodos que temos

- RetornoUsuario
- criaUsuario
- removeUsuario
- atualizaUsuario
- adicionaAssinatura
- buscaAssinatura

- Esses métodos contam com recebimento de informações e retorno dessas informações
- Em geral devemos tratar para que essas informações só sejam acessadas por quem realmente importa

Criptografia

História

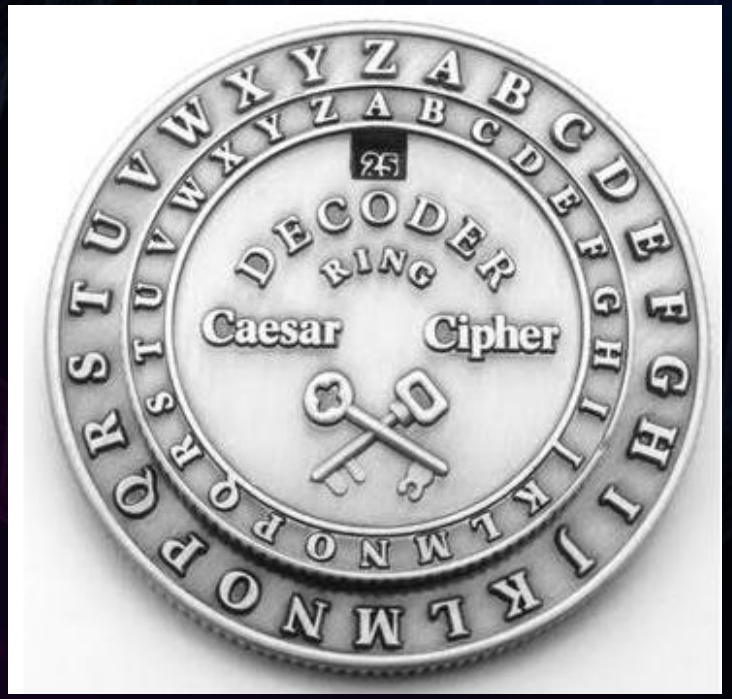
- Podemos dividir em duas épocas a criptografia:
  - Clássica
  - Moderna

Criptografia - História

Criptografia clássica

- Indo da época dos povos antigos, passando pela idade média e até a segunda guerra mundial
- Hebreus usavam muito, o livro de Jeremias por exemplo, foi escrito usando a técnica da criptografia. Os hebreus utilizavam a cifra de substituição simples, monoalfabética e monogâmica
- Também é possível citar uma das mais famosas, a cifra de César, criada para se comunicar com os generais em momentos de guerra
  - César trocava determinada letra do alfabeto por aquela que vinha 3 vezes à sua frente. A letra A era trocada pela letra D, B era trocado por E e assim adiante.

Cifra de César



https://blog.mastermaq.com.br/como-surgiu-a-criptografia/

https://pt.quora.com/Quantas-configura%C3%A7%C3%B5es-poss%C3%ADveis-existem-em-uma-m%C3%A1quina-Enigma-Como-as-configura%C3%A7%C3%B5es-do-Enigma-foram-comunicadas

Criptografia - História

Criptografia moderna

- Esse tipo de criptografia surge na segunda guerra mundial, pois era um momento que era necessário manter informações confidenciais seguras, pois invasões e dados estratégicos eram essenciais para uma boa ofensiva ou defesa.
- Para isso, foi desenvolvida a máquina ENIGMA pelos alemães
- Essa máquina era responsável por criptografar e descriptografar as mensagens, por meio de rotores eram feitas as mudanças que tornavam praticamente impossíveis de serem quebradas sem o código original
- O numero total de combinações possíveis estava por volta de :

31.291.969.749.695.380.357.632.000

- Caso queiram saber mais, podem procurar o filme

O Jogo da Imitação

Enigma

Criptografia - Atualmente

- Atualmente as criptografias consistem em sistemas de chaves criptográficas, que consiste em um conjunto de bits baseado num algoritmo, que pode codificar ou decodificar a informação.
- Podemos dividir as chaves em dois tipos:
  - Simétrica: Mais simples, qual o remetente e destinatário tem a mesma chave para codificar e decodificar a informação
  - Assimétrica: é utilizado um par de chaves, pública e privada, onde a primeira é responsável por criptografar a informação, enquanto a privada pode decriptografar ou criptografar, nesse tipo de criptografia normalmente somente o receptor da mensagem tem a chave privada, e todos que enviam tem a pública e somente o receptor vai poder ler as informações

Usabilidade

Os diferentes tipos de criptografia são utilizadas em diversas aplicações, entre elas:

- WhatsApp
- Senhas em banco de dados
- Arquivos na nuvem
- Sites financeiros
- Diversos outros locais

Nest.js

Nos próximos slides vamos ver como aplicar criptografia no Nest.Js

- Primeiro vamos ver criptografia do tipo Hashing

Hashing

Segurança na hora de armazenar

- Hash consiste em pegar dados de tamanho variável e transformar num dado de tamanho fixo
- Com essa transformação, uma informação gerada não pode ser retornada ao seu estado original
- Mas, se não pode retornar, como vamos usar na senha? Não tem como validar?

Hashing - Senha

Aplicação de hash em senhas

- Nas senhas é necessário para armazenar de forma segura a senha, não possibilitando um possível atacante de descobrir a senha original
- Mas como validar?
- No caso de uma validação, é necessário criar o hash antes mesmo de fazer a comparação da senha, assim não é necessário transitar informações sensíveis pela rede
- Mas nesse primeiro momento vamos ver como fazer enviando a senha como texto mesmo

Aplicação nest

Bora aplicar um exemplo de hash

- Primeiro é necessário instalar a biblioteca bcrypt e seus tipos

```
npm i bcrypt
npm i -D @types/bcrypt
```

Inserindo usuario

Vamos ver as alterações na hora de inserir o usuario

```
You, 25 minutes ago | 1 author (You)
import Datas from "../utils/datas";
import * as bcrypt from 'bcrypt';
```

```
constructor(id: string,nome: string,idade: BigInteger,cidade:
    const saltOrRounds = 10;

    this.#datas = new Datas();
    this.id = id;
    this.nome = nome;
    this.idade = idade;
    this.cidade = cidade;
    this.email = email;
    this.telefone = telefone;
    this.senha = bcrypt.hashSync(senha, saltOrRounds);
}
    You, 2 weeks ago • alterações p/ add sistema de assinatura
```

```
trocaSenha(senha){
    const saltOrRounds = 10;        You, 10 minutes ago • Un
    this.senha = bcrypt.hashSync(senha, saltOrRounds);
}
```

Fazendo login

Vamos ver o passo a passo para implementar o login

Usuario.dm.js

```
validarLogin(email:string,senha:string){
    const usuario = this.buscaPorEmail(email);
    return usuario.login(senha);

}
```

Usuario.Entity.js

```
login(senha){
    return bcrypt.compareSync(senha,this.senha);
}
```

Usuario.dm.js

```
buscaPorEmail(email:string){
    const possivelUsuario = this.#usuarios.find(
        usuario => usuario.email === email
    );
    return possivelUsuario;
}       You, 54 seconds ago • Uncommitted changes
```

Usuario.dm.js

```
atualizaUSuario(id: string, dadosAtualizacao: Partial<UsuarioEntity>){
    const usuario = this.buscaPorID(id);

    Object.entries(dadosAtualizacao).forEach(
        ([chave,valor]) => {
            if(chave === 'id'){
                return
            }else if(chave === 'senha'){
                usuario.trocaSenha(valor);
                return
            }

            usuario[chave] = valor;
        }
    )
  You, 3 weeks ago • alterações em aula …
    return usuario;
}
```

Fazendo login

Vamos ver o passo a passo para implementar o login

Usuario.controller.js

```
@Get('/login')
async Login(@Body() dadosUsuario: LoginUsuarioDTO){
    var login = this.clsUsuariosArmazenados.validarLogin(dadosUsuario.email,dadosUsuario.senha);
    return {
        status: login,
        message: login ? "Login efetuado" : "Usuario ou senha inválidos"
    }
}
```

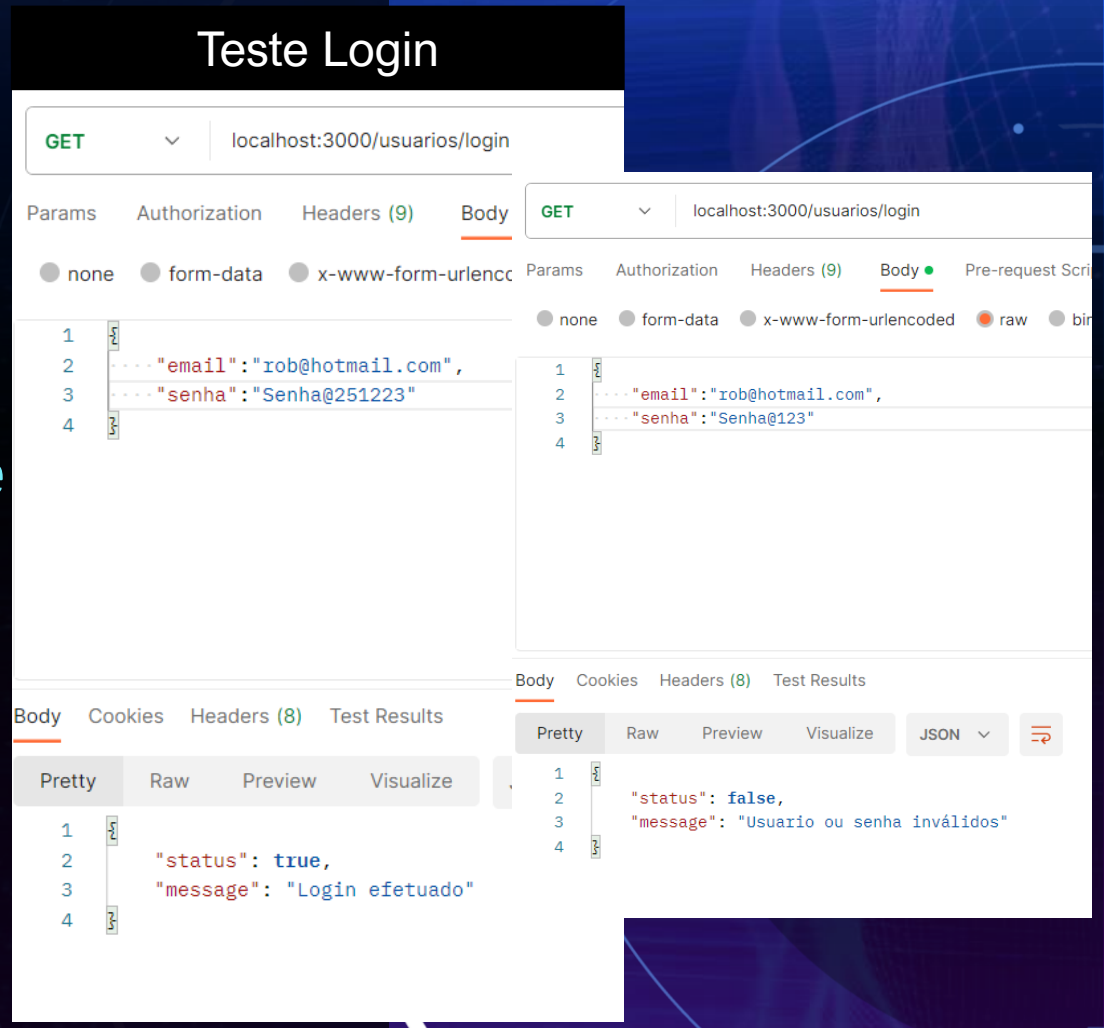
Testando

Vamos testar agora se deu certo, vamos executar e fazer os testes

Get com a senha criptografada

```
[
    {
        "id": "4b73e9c9-9430-4460-adc1-68d62237c4a0",
        "nome": "robersval",
        "cidade": "bauru",
        "email": "rob@hotmail.com",
        "assinatura": "2023-12-19",
        "senha": "$2b$10$KU/THa7TTFcQhxVZhHVxO.acRWQZ8rka8SN9XtTRj1yzpVrp5w18W"
    }
]
```

Teste Login



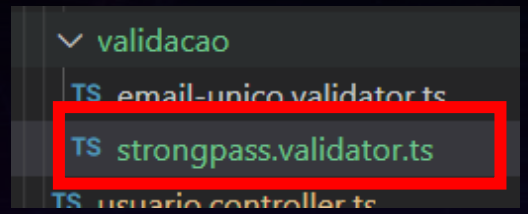
Senha forte

Reimplementando a senha forte

- Quando falamos de segurança, temos que levar em conta a outra ponta do sistema, o usuário
- Implementar políticas de segurança e senhas fortes pode ajudar a reduzir a incidência de invasões
- Não é só por criptografias que mantemos a segurança, para isso vamos resolver implementando novamente a validação de senha forte

Criando decorator

Primeiro passo, precisamos criar um decorator



- Precisamos criar o arquivo do decorator
- Também precisamos instalar uma biblioteca de validação

```
npm install zxcvbn
```

```typescript
import { Injectable } from "@nestjs/common";
import { registerDecorator, ValidationArguments, ValidationOptions, ValidatorConstraint, ValidatorCo
import * as zxcvbn from 'zxcvbn';

@Injectable()
@ValidatorConstraint({async:true})
export class strongPassValidator implements ValidatorConstraintInterface{

    async validate(value: string, validationArguments?: ValidationArguments): Promise<boolean> {
        const result = zxcvbn(value);
        var validarSenha = (result.score <= 2)
        return !validarSenha;
    }
}

export const SenhaForte = (opcaoValidacao: ValidationOptions)=>{
    return (objeto: Object, propriedade: string) => {
        registerDecorator({
            target: objeto.constructor,
            propertyName: propriedade,
            options: opcaoValidacao,
            constraints: [],
            validator: strongPassValidator,
        })
    }
}
```

Essa parte do código define o grau de segurança da senha, quanto maior o score, mais complexa e segura

Alterando DTOS

Depois de criar o decorator, podemos implementar ele no DTO

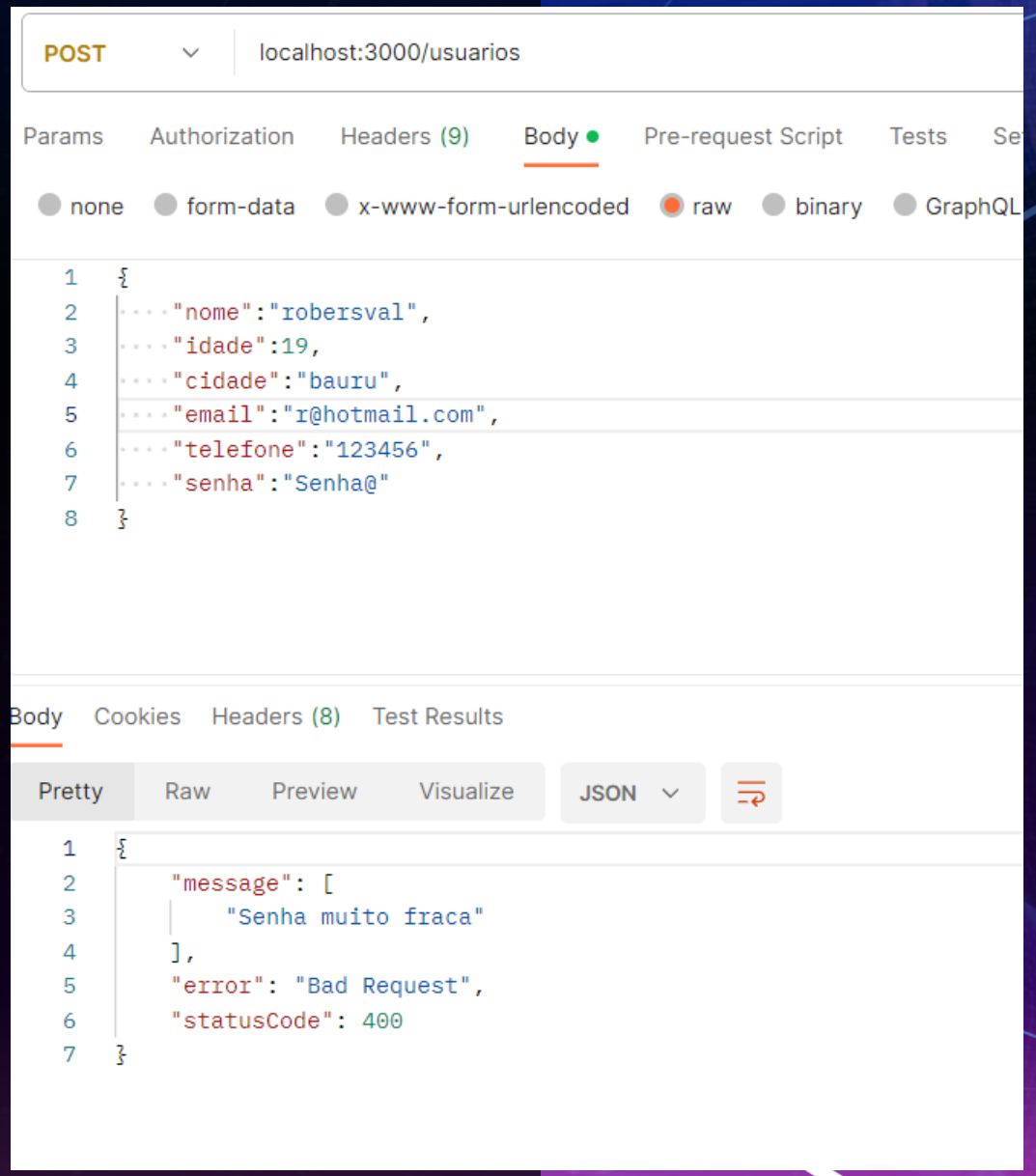
- Vamos implementar no “usuario.dto.ts” e no “atualizaUsuario.dto.ts”

```
@MinLength(6,{message: "Senha precisa de pelo menos 6 digitos"})
@SenhaForte({message: "Senha muito fraca"})
senha: string;
    You, 3 weeks ago • alterações …
```

```
@MinLength(6,{message: "Senha precisa de pelo menos 6 digitos"})
@IsOptional()
@SenhaForte({message: "Senha muito fraca"})      You, 44 minutes a
senha: string;
```

Testando

Agora vamos ver como fica na prática



HTTPS

HTTPS – Segurança ponta a ponta

- Quando usamos o protocolo HTTP, estamos enviando textos pela rede de um ponto a outro, porém sem nenhum tipo de segurança
- Ai entra o HTTPS, que nada mais é, uma melhoria do protocolo HTTP, onde é implementada a segurança ponta a ponta para que as comunicações sejam feitas de forma segura

HTTPS

HTTPS – Segurança ponta a ponta

- Essa segurança consiste em embaralhar as informações de quem envia, quando chega no destino essa informação é desembaralhada
- Mas para que seja possível utilizar essa camada de segurança, é necessário ter um certificado, ele serve para definir que você é o dono daquela informação, com base nele aquela informação vai ser decriptografada

HTTPS

HTTPS – Segurança ponta a ponta

- Primeiro é necessários criar os certificados, mais pra frente vamos ver como criar para testes
- Mas no ambiente corporativo não são emitidos por você, mas sim por uma empresa especializada, chamada autoridade de certificação válida, ela emite e valida os certificados

HTTPS

HTTPS – Segurança ponta a ponta

- A validação é de extrema importância, pois é responsável por falar que o seu site é seguro e que realmente é real
- O seu certificado é único, te identifica e é o único capaz de decifrar as mensagens criptografadas enviadas pelos usuários

1 – Passo

- O usuário se comunica com seu servidor (api)

HTTPS

HTTPS – Segurança ponta a ponta

2 – Passo

- O usuário se comunica com a entidade certificadora para confirmar a autenticidade do certificado

HTTPS

HTTPS – Segurança ponta a ponta

3 – Passo

- O usuário criptografa os dados usando a chave pública, envia e ao chegar no servidor, é usada a chave privada para voltar os dados aos seu estado original

HTTPS

HTTPS – Segurança ponta a ponta

HTTPS

HTTPS – Segurança ponta a ponta

- Desta forma conseguimos 2 pontos positivos em utilizar o HTTPS:
  - Confirmar autenticidade do site acessado
  - Proteger informações no caminho
- Por se tratar de um certificado único, ele deve ser muito bem armazenado para que não tenha risco de vazamento, pois quem tem esse certificado tem o poder de quebrar a criptografia do HTTPS

Implementando na API

Agora vamos ver o passo a passo de como implementar

1. Criar certificado
2. Habilitar certificados no projeto
3. Testar se está funcionando a comunicação HTTPS

Criando Certificado

Para iniciar é necessário criar o certificado

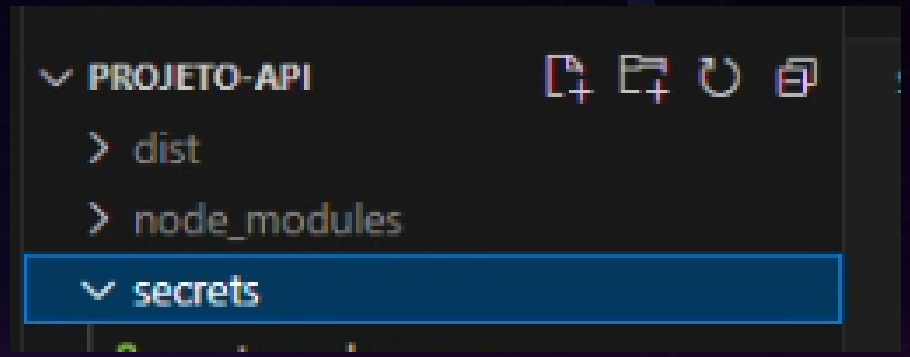
- Primeiro vamos instalar a biblioteca responsável pelo certificado

```
npm install mkcert -g
```

Criando Certificado

Para iniciar é necessário criar o certificado

- Agora vamos criar a pasta que vão ficar os arquivos de certificado
- Depois vamos entrar na pasta com o comando **CD secrets**



Criando Certificado

Para iniciar é necessário criar o certificado

- Agora vamos criar os arquivos da autoridade certificadora

```
mkcert create-ca
```

- Agora vamos criar os certificados com o comando

```
mkcert create-cert
```

IMPORTANTE

Apesar de ensinado aqui a criar o certificado, por problemas de permissão de usuário talvez não seja possível criar nos computadores do SENAC, mas nada impede de criar no computador de vocês

IMPORTANTE

Algo que pode acontecer também é erro de politica do Windows, isso pode ser corrigido no PC de vocês também com o tutorial abaixo no powershell

https://pureinfotech.com/change-execution-policy-run-scripts-powershell/

Habilitando na API

Agora precisamos habilitar o certificado na API

- Para isso no arquivo main.ts

```
async function bootstrap() {
  const fs = require('fs');
  const httpsOptions = {
    key: fs.readFileSync('./secrets/create-cert-key.pem'),
    cert: fs.readFileSync('./secrets/create-cert.pem'),
  };


  //const app = await NestFactory.create(AppModule,{httpsOptions});

  const app = await NestFactory.create(AppModule);

  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      whitelist: true,
      forbidNonWhitelisted: true,
    })
  )


  useContainer(app.select(AppModule),{fallbackOnErrors:true})
  app.enableCors();
  await app.listen(3000);
}
bootstrap();
```

IMPORTANTE

Apesar de funcionar corretamente local, o certificado que criamos e estamos utilizando não tem capacidade de prover o HTTPS na rede, pois esses certificados são para testes locais.

Para que seja possível criar certificados reais, é necessário contratar uma empresa certificadora.

Obrigado

Prof. João Pedro Parella

Joao.pparella@sp.senac.br