

**FECAP**

**FUNDAÇÃO ESCOLA DE COMÉRCIO ÁLVARES PENTEADO**

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**São Paulo**

**2023**

## **Integrantes do grupo:**

Aline Germano Costa RA 21022950

Bruna Dias Albuquerque RA 22074311

Bruna Rodrigues da Silva Pavechi RA 21022951

Mary Alice Aparecida Guilherme RA 22023051

Mateus Quintino Vieira Santos RA 22023854

Raquel Anício RA 21022954

Yasmim Renata da Silva RA 19020393



# **Pass Manager**

# **CRIPTOGRAFIA – PROJETO PASS MANAGER**

Trabalho documental referente aos métodos criptográficos utilizados no Aplicativo Pass Manager - Projeto Interdisciplinar do 4º Semestre para o curso Análise e Desenvolvimento de Sistema para a Fundação Escola de Comércio Álvares Penteado (FECAP)

**Orientador: Prof. Ronaldo Araujo Pinto**

**São Paulo**

**2023**

## SUMÁRIO

1	INTRODUÇÃO.....	5
2	CRIPTOGRAFIA .....	5
3	RELEVÂNCIA.....	5
4	DESENVOLVIMENTO DO APLICATIVO .....	6
5	FUNCIONALIDADES DO PROJETO PASS MANAGER .....	7
6	VANTAGENS DA CRIPTOGRAFIA NO APP PASS MANAGER .....	7
7	APLICABILIDADE DE CRIPTOGRAFIA NO APLICATIVO.....	8
8	CRIPTOGRAFIA PARA SENHA DE USUÁRIO: .....	8
9	CRIPTOGRAFIA PARA SALVAR AS SENHAS DAS APLICAÇÕES DOS USUÁRIOS:.....	10
10	TELAS DO APLICATIVO QUE UTILIZAM OS MÉTODOS DESCRITOS.....	12
11	CONCLUSÃO.....	12
	REFERÊNCIAS.....	13

## 1 INTRODUÇÃO

**Pass Manager** é uma solução de tecnologia que visa trazer segurança e mobilidade para o gerenciamento de senhas em num único lugar, proporcionando ao usuário facilidade e flexibilidade para manipulação das suas senhas pessoais, nosso foco é trazer para o usuário a melhor experiência possível durante o uso do aplicativo.

*“A segurança das suas senhas num único lugar com praticidade e mobilidade”*

## 2 CRIPTOGRAFIA

A criptografia é uma técnica utilizada para proteger informações sensíveis, tornando-as ilegíveis para qualquer pessoa que não tenha a chave de descryptografia adequada. Ela desempenha um papel fundamental na segurança da comunicação e no armazenamento de dados em sistemas de computador e redes. A principal finalidade da criptografia é garantir a confidencialidade, a integridade e, em alguns casos, a autenticidade das informações.

A criptografia dificulta a exploração de vulnerabilidades em aplicativos, como ataques de injeção de SQL, interceptação de dados, ou acesso não autorizado a informações armazenadas.

## 3 RELEVÂNCIA

A utilização da criptografia em aplicativos oferece vários benefícios e é de extrema importância para garantir a segurança e a privacidade dos dados dos usuários, assim como, manter a integridade das informações e garantir a confiabilidade de aplicativos. Ela desempenha um papel fundamental e relevante na construção de aplicativos seguros e na promoção da confiança dos usuários, além de estar em conformidade com regulamentações de privacidade e segurança de dados. Em resumo, a criptografia desempenha um papel crítico em um aplicativo do tipo gerenciador de

senhas, garantindo que as informações sensíveis dos usuários, como senhas e credenciais de login, estejam protegidas contra ameaças internas e externas. Ela constrói a confiança dos usuários, ajuda a manter a conformidade legal e é uma parte fundamental da segurança cibernética em aplicativos desse tipo.

## 4 DESENVOLVIMENTO DO APLICATIVO

O desenvolvimento do aplicativo foi realizado utilizando as ferramentas e tecnologias Angular e Javascript que permitem as conexões Back-End e Front- End do projeto utilizando Angular JS e Node JS, Docker e PostgreSQL.

The image shows a development environment with two main panels. The left panel displays the 'Pass Manager Frontend' README, which includes instructions for running the development server, code scaffolding, building, and running tests. The right panel shows a web browser displaying the application's login page. The browser's address bar shows the URL 'https://localhost:4200/auth/sign-in'. The login page features a 'Pass Manager' logo, a welcome message 'Bem vindo de volta!', a prompt to enter credentials, and input fields for 'Email' and 'Senha'. A blue 'Entre' button is at the bottom, along with a link to 'Cadastre-se' for new users.

**Pass Manager Frontend**

This project was generated with **Angular CLI** version 15.2.4.

**Development server**

Run `ng serve` for a dev server. Navigate to `http://localhost:4200/`. The application will automatically reload if you change any of the source files.

**Code scaffolding**

Run `ng generate component component-name` to generate a new component. You can also use `ng generate directive|pipe|service|class|guard|interface|enum|module`.

**Build**

Run `ng build` to build the project. The build artifacts will be stored in the `dist/` directory.

**Running unit tests**

Run `ng test` to execute the unit tests via **Karma**.

**Running end-to-end tests**

Run `ng e2e` to execute the end-to-end tests via a platform of your choice. To use this command, you need to first add a package that implements end-to-end testing capabilities.

**Further help**

To get more help on the Angular CLI use `ng help` or go check out the [Angular CLI Overview and Command Reference page](#).

**Pass Manager**

**Bem vindo de volta!**

Por favor insira seus dados:

Email:  
insira seu email

Senha:  
insira sua senha

**Entre**

Ainda não tem uma conta? [Cadastre-se](#)

## 5 FUNCIONALIDADES DO PROJETO PASS MANAGER

Cadastro, login, autenticação, trazer os dados dos usuários, deleção e atualização dos dados dos usuários, sistema de criptografia, sistema de TOKEN, gerenciamento de senha: cadastrar e trazer a senha e deleção e atualização de senhas.

## 6 VANTAGENS DA CRIPTOGRAFIA NO APP PASS MANAGER

**Proteção das Senhas do Usuário:** A criptografia é usada para proteger as senhas e informações de login armazenadas no aplicativo. Quando um usuário insere uma senha no aplicativo, ela é criptografada antes de ser armazenada em um banco de dados. Isso impede que terceiros não autorizados acessem as senhas diretamente, mesmo que obtenham acesso ao banco de dados do aplicativo. Apenas o usuário legítimo, com a chave de descriptografia correta (sua senha principal ou outra forma de autenticação), pode acessar suas senhas.

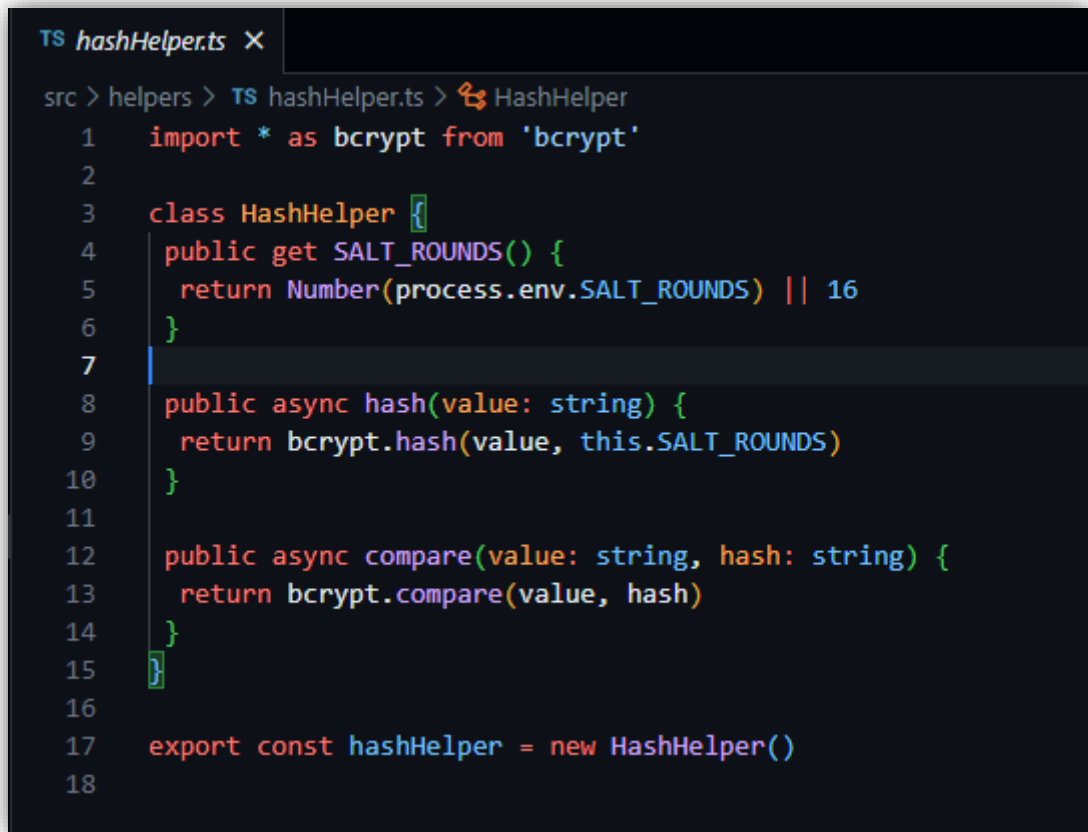
**Segurança contra Ataques Internos e Externos:** A criptografia protege as senhas e outras informações de login contra ataques de hackers, como vazamentos de dados, roubo de informações do aplicativo ou exploração de vulnerabilidades. Mesmo se um atacante conseguir acessar o banco de dados do aplicativo, as senhas e informações estarão cifradas e, portanto, inutilizáveis sem a chave de descriptografia, o que aumenta significativamente a segurança.

**Confiança dos Usuários:** O uso da criptografia demonstra o comprometimento do aplicativo com a segurança e a privacidade dos dados dos usuários, construindo a confiança dos clientes que vierem a utilizar o aplicativo Pass Manager.

**Proteção contra Ransomware:** A criptografia também protege as informações armazenadas contra ataques de ransomware, que podem criptografar dados de forma maliciosa. As informações criptografadas já são inacessíveis para ransomware.

## 7 APLICABILIDADE DE CRIPTOGRAFIA NO APLICATIVO

### 8 Criptografia para senha de usuário:



```
TS hashHelper.ts X
src > helpers > TS hashHelper.ts > HashHelper
1  import * as bcrypt from 'bcrypt'
2
3  class HashHelper {
4    public get SALT_ROUNDS() {
5      return Number(process.env.SALT_ROUNDS) || 16
6    }
7
8    public async hash(value: string) {
9      return bcrypt.hash(value, this.SALT_ROUNDS)
10     }
11
12    public async compare(value: string, hash: string) {
13      return bcrypt.compare(value, hash)
14    }
15  }
16
17  export const hashHelper = new HashHelper()
18
```

O código utilizado acima utiliza a biblioteca bcrypt para realizar operações de criptografia. O bcrypt é uma biblioteca amplamente usada para o hash seguro de senhas. Ele utiliza uma técnica chamada "bcrypt" para proteger senhas e outros dados sensíveis por meio de um processo chamado "hashing" com "salting" (saldos).

#### **Aqui está uma explicação de como a criptografia funciona no código:**

Definição do número de rounds de salt (saldos): A constante SALT\_ROUNDS define o número de iterações (rounds) que o algoritmo de hash usará para proteger a senha.



Mais rounds tornam o processo de hash mais seguro, mas também mais lento. O valor padrão é 16 se não for especificado nas variáveis de ambiente.

### **Método hash:**

O método hash é usado para criar um hash seguro a partir de uma string de entrada (provavelmente uma senha), ele chama a função `bcrypt.hash`, que utiliza o algoritmo `bcrypt` para realizar o hash da string de entrada.

O número de rounds (iterações) é definido pela constante `SALT_ROUNDS`.

O resultado do hash é uma sequência de caracteres criptograficamente segura.

### **Método compare:**

O método `compare` é usado para verificar se uma string de entrada corresponde a um hash armazenado previamente. Ele chama a função `bcrypt.compare`, que compara a string de entrada com o hash armazenado.

Se a string de entrada corresponder ao hash, a função retornará `true`; caso contrário, retornará `false`.

Em resumo, o código utiliza o `bcrypt` para criar hashes seguros das senhas e para verificar se uma senha inserida corresponde ao hash armazenado. O uso de um número configurável de rounds de salting torna o processo mais seguro, uma vez que torna mais difícil para um atacante realizar ataques de força bruta ou de dicionário.

## 9 Criptografia para salvar as senhas das aplicações dos usuários:

```
import { randomBytes, createCipheriv, createDecipheriv } from 'crypto'

import { env } from '../env'

export class CryptoHelper {
  private static readonly algorithm = 'aes-256-gcm'
  private static readonly ivLength = 12
  private static readonly secretKey = env.ENCRIPTION_KEY

  public static encrypt(text: string) {
    const iv = randomBytes(this.ivLength)
    const cipher = createCipheriv(this.algorithm, this.secretKey, iv)

    const encrypted = Buffer.concat([cipher.update(text, 'utf8'), cipher.final()])

    const tag = cipher.getAuthTag()

    return `${iv.toString('hex')}:${encrypted.toString('hex')}:${tag.toString('hex')}`
  }

  public static decrypt(encryptedText: string) {
    const [ivHex, encryptedHex, tagHex] = encryptedText.split(':')

    const iv = Buffer.from(ivHex, 'hex')
    const encrypted = Buffer.from(encryptedHex, 'hex')
    const tag = Buffer.from(tagHex, 'hex')

    const decipher = createDecipheriv(this.algorithm, this.secretKey, iv)
    decipher.setAuthTag(tag)

    const decrypted = Buffer.concat([decipher.update(encrypted), decipher.final()])

    return decrypted.toString('utf8')
  }
}
```

O código acima utiliza a biblioteca crypto para realizar operações de criptografia, especificamente criptografia simétrica. Aqui está uma explicação de como a criptografia funciona no código:

### Algoritmo de Criptografia:

O código utiliza o algoritmo de criptografia simétrica AES (Advanced Encryption Standard) com uma chave de 256 bits no modo GCM (Galois/Counter Mode). Isso é

especificado pela constante `algorithm`, que é definida como `'aes-256-gcm'`. AES é amplamente considerado seguro e eficaz para criptografia simétrica.

### **Chave de Criptografia:**

A chave de criptografia é lida a partir da variável de ambiente `ENCRYPTION_KEY` definida em algum lugar do código. É importante que essa chave seja mantida em segurança, pois é usada para criptografar e descriptografar os dados.

### **Método `encrypt`:**

O método **`encrypt`** é usado para criptografar um texto de entrada.

Ele gera um vetor de inicialização (IV) aleatório com um comprimento de 12 bytes usando `randomBytes`. Em seguida, ele cria um objeto de cifra usando `createCipheriv` com o algoritmo AES-256-GCM, a chave secreta e o IV gerado.

O texto de entrada é então criptografado usando `cipher.update` e `cipher.final()`.

A autenticação (tag de autenticação) é obtida usando `cipher.getAuthTag()`.

O resultado é uma string no formato `${iv}:${encrypted}:${tag}`.

### **Método `decrypt`:**

O método `decrypt` é usado para descriptografar o texto criptografado gerado pelo método `encrypt`.

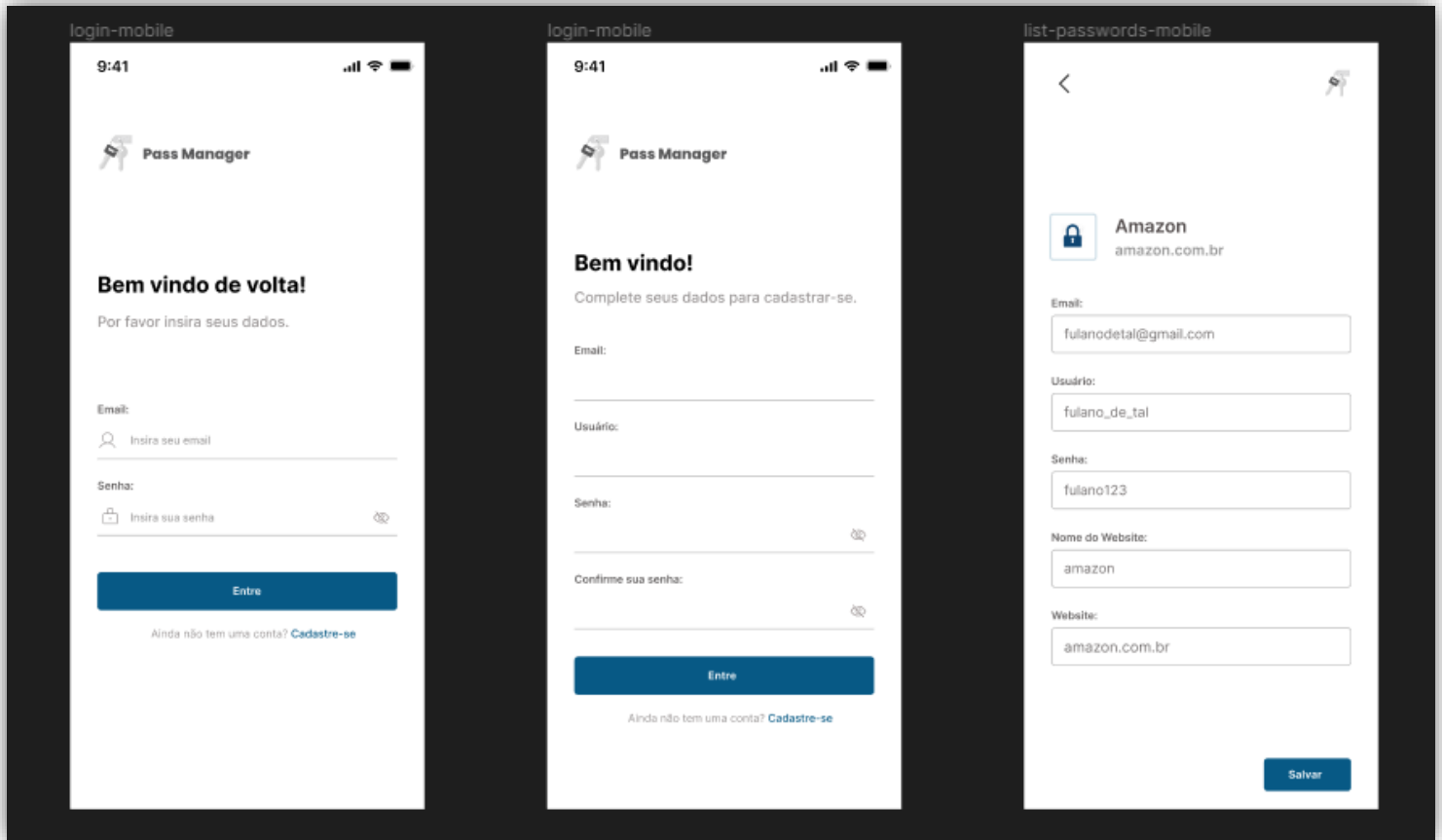
A string criptografada é dividida em três partes usando o caractere `:`, representando o IV, o texto criptografado e a tag de autenticação.

O IV, o texto criptografado e a tag são convertidos de hexadecimal de volta para objetos de buffer. Um objeto de decifração é criado usando `createDecipheriv` com o mesmo algoritmo, chave secreta e IV.

A tag de autenticação é configurada usando `decipher.setAuthTag`. O texto criptografado é descriptografado usando `decipher.update` e `decipher.final()`, o resultado é uma string descriptografada.

Este código implementa criptografia simétrica usando o algoritmo AES-256-GCM, que é uma técnica segura para proteger dados.

## 10 TELAS DO APLICATIVO QUE UTILIZAM OS MÉTODOS DESCRITOS



## 11 CONCLUSÃO

Por fim, a escolha de cada algoritmo foi realizada de acordo com as ferramentas para elaboração do código, **neste caso Angular e Node JS que foram as tecnologias disponibilizadas e recomendadas pelo professor de Programação Mobile do 3º Semestre (Professor Victor), pois, era pré-requisito da disciplina.** Deste modo, nosso aplicativo apresenta todas as funcionalidades necessárias para utilização do aplicativo, garantindo segurança nas transações realizadas pelos usuários durante o uso do aplicativo.

## REFERÊNCIAS

Repositório do Projeto Disponível em: <<https://github.com/2023-1-NADS3/E9-PassManager>>. Acesso em 01 de outubro de 2023.