



*Inovação e Excelência **desde 1902***

Sistemas Operacionais e Arquiteturas Cloud Native



Requisitos:

1º O servidor deverá operar dentro de uma das ferramentas de cloud computing.

Utilizamos Render para a plataforma Cloud e Docker no banco de dados **PostgreSQL**.

Segue abaixo imagem do arquivo **docker-compose** do nosso repositório que gera a imagem do banco localmente e logo após o **Dockerfile** do projeto com ele rodando:

```
docker-compose.yml x
docker-compose.yml
To-Mat, 2 months ago | 1 author (To-Mat)
1  services:
2    postgres:
3      image: postgres:14-alpine3.14
4      environment:
5        POSTGRES_USER: ${DATABASE_USER}
6        POSTGRES_PASSWORD: ${DATABASE_PASS}
7        POSTGRES_DB: ${DATABASE_NAME}
8        PG_DATA: /var/lib/postgresql/data
9      volumes:
10       - /var/lib/postgresql/data
11      ports:
12       - '6222:5432' To-Mat, 2 months ago • feat: initial
```



```
1 # Estágio 1: Compilar a aplicação
2 FROM node:18-alpine AS build
3
4 # COPY package.json and package-lock.json files
5 COPY package*.json ./
6
7 # generated prisma files
8 COPY prisma ./prisma/
9
10 # COPY ENV variable
11 COPY .env ./
12
13 # COPY tsconfig.json file
14 COPY tsconfig.json ./
15
16 # COPY
17 COPY . .
18
19 RUN npm install
20 RUN npx prisma generate
21
22 # Run and expose the server on port 3000
23 EXPOSE 3000
24
25 # A command to start the server
26 CMD npm start
27
```



```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE GITLens
• → pass-manager-api git:(main) ✕ sudo docker ps
[sudo] password for mateus:
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES
3eebea584c1d   my-app    "docker-entrypoint.s..." 4 minutes ago Up 4 minutes   3000/tcp     trusting_clarke

```

Segue abaixo tabelas de usuários e tabela para cadastro das senhas de usuários respectivamente:

User X +

↺

Filters

None

Fields

All

Showing 1 of 1

Add record

	id #	email A	username A	password A	createdAt 📅	updatedAt 📅	Credentials []
	1	mateusquent2003@gmail...	mateusKent123	\$2b\$04\$berMHSUyVJhHrd...	2023-05-06T18:40:11.9...	2023-05-06T18:40:11.9...	0 Credentials



Inovação e Excelência desde 1902

Credentials X +				
⌂	Filters	None	Fields	All
	Showing	0 of 0	Add record	
<input type="checkbox"/>	id #	emailCredential A?	usernameCredential A?	credentialPassword A
websiteName A				

	websiteUrl A?	user {}	userId #	createdAt 📅	updatedAt

3º Deve conter um código de servidor mínimo para responder o aplicativo mobile:

Segue abaixo trechos de códigos que demonstram a conexão com o servidor:

```
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3 import { environment } from 'src/environments/environments';
4 import { map } from 'rxjs';
5 import { signUpUserDTO } from '../DTOs/signUpUserDTO';
6 import { signInUserDTO } from '../DTOs/signInUserDTO';
7 import { User } from '../../user/models/User';
8 import jwt_decode from 'jwt-decode';
9
10 interface AuthResponse {
11   username: string;
12   email: string;
13   password: string;
14   token: string;
15 }
16
17 @Injectable({
18   providedIn: 'root',
19 })
20 export class AuthService {
21   url: string;
22
23   constructor(private http: HttpClient) {
24     this.url = `${environment.url}/users`;
25   }
26
27   signUp(input: signUpUserDTO) {
28     return this.http.post<AuthResponse>(`${this.url}/create`, input).pipe(
29       map((data) => {
30         sessionStorage.setItem('token', data.token);
31         return data;
32       })
33     );
34   }
35
36   signIn(input: signInUserDTO) {
37     return this.http.post<AuthResponse>(`${this.url}/auth`, input).pipe(
38       map((data) => {
39         sessionStorage.setItem('token', data.token);
40         return data;
41       })
42     );
43   }
44
45   logout() {
46     sessionStorage.removeItem('token');
47   }
48
49   isAuthenticated() {
50     return !!sessionStorage.getItem('token');
51   }
52
53   getToken() {
54     return sessionStorage.getItem('token');
55   }
56
57   getUserIdFromToken() {
58     const decodedToken: any = jwt_decode(sessionStorage.getItem('token')!);
59     return decodedToken.sub.userId;
60   }
61 }
62
```



```
1 import { HttpClient, HttpHeaders } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { map } from 'rxjs';
4 import { environment } from 'src/environments/environments';
5 import { registerAndUpdateCredentialDTO } from '../DTOs/registerAndUpdateCredentialDTO';
6 import { Credential } from '../models/Credential';
7
8 import { AuthService } from '../../auth/services/auth.service';
9
10 @Injectable({
11   providedIn: 'root',
12 })
13 export class CredentialService {
14   url: string;
15
16   constructor(private http: HttpClient, private authService: AuthService) {
17     this.url = `${environment.url}/credentials`;
18   }
19
20   register(input: registerAndUpdateCredentialDTO) {
21     return this.http.post(`${this.url}/create`, input).pipe(
22       map((data) => {
23         return data;
24       })
25     );
26   }
27
28   obtain(id: number) {
29     return this.http.get<Credential>(`${this.url}/getCredential/${id}`).pipe(
30       map((data) => {
31         return data;
32       })
33     );
34   }
35
36   list() {
37     return this.http.get<Credential[]>(`${this.url}/getCredentials`).pipe(
38       map((data) => {
39         return data;
40       })
41     );
42   }
43
44   update(id: number, input: registerAndUpdateCredentialDTO) {
45     return this.http.put(`${this.url}/updateCredential/${id}`, input).pipe(
46       map((data) => {
47         return data;
48       })
49     );
50   }
51
52   delete(id: number) {
53     return this.http.delete(`${this.url}/deleteCredential/${id}`).pipe(
54       map((data) => {
55         return data;
56       })
57     );
58   }
59 }
60
```

```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { map } from 'rxjs';
4 import { environment } from 'src/environments/environments';
5 import { UpdateUserDTO } from '../DTOs/updateUserDTO';
6 import { User } from '../models/User';
7
8 @Injectable({
9   providedIn: 'root',
10 })
11 export class UserService {
12   url: string;
13
14   constructor(private http: HttpClient) {
15     this.url = `${environment.url}/users`;
16   }
17
18   obtain(id: number) {
19     return this.http.get<User>(`${this.url}/getUser/${id}`).pipe(
20       map((data) => {
21         return data;
22       })
23     );
24   }
25
26   update(id: number, input: UpdateUserDTO) {
27     return this.http.put(`${this.url}/update/${id}`, input).pipe(
28       map((data) => {
29         return data;
30       })
31     );
32   }
33
34   delete(id: number) {
35     return this.http.delete(`${this.url}/delete/${id}`).pipe(
36       map((data) => {
37         return data;
38       })
39     );
40   }
41 }
42
```




*Inovação e Excelência **desde 1902***

Programação Mobile



Inovação e Excelência desde 1902

Requisitos:

1º Desenvolvimento de um aplicativo usando Angular e Javascript:

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

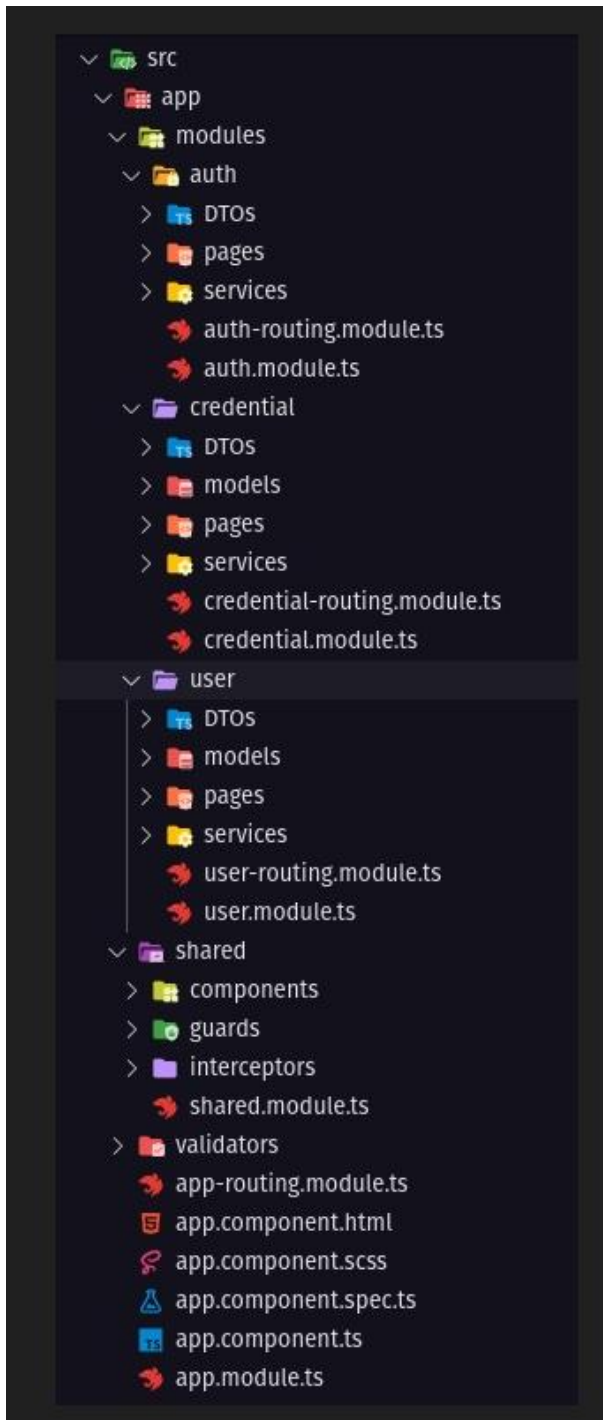
Entre

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

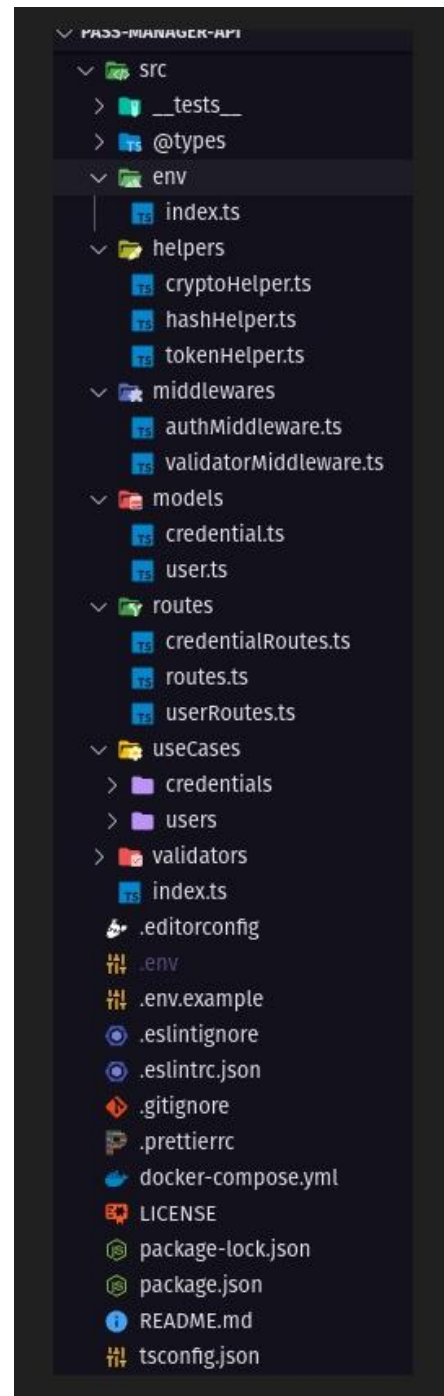
2º O aplicativo deverá realizar conexão com servidores online:

Segue abaixo as estruturas de pastas dos códigos que permitem as conexões Back-End e Front- End do projeto utilizando Angular JS e Node JS, Docker e PostgreSQL.

Estrutura de pasta Angular **Front-End**



Estrutura de pasta Angular **Back-End**





3º Deverá apresentar, ao menos, 6 funcionalidades distintas. Estas deverão ter aprovação do professor da disciplina Programação Mobile.

FUNCIONALIDADES DO PROJETO PASS MANAGER - GRUPO [E9]

- ✓ 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;
- ✓ Deleção e atualização de senhas;



Inovação e Excelência desde 1902

Testes e Qualidade de Software (DevOps)



Requisitos:

1º Apresentação de três testes unitários e dois testes de componentes:

Para realização dos testes unitários no backend utilizamos a ferramenta:

No backend para executamos **os testes unitários usamos a biblioteca Vitest** (<https://vitest.dev/>) que é uma plataforma de teste automatizada que permite a criação, execução e gerenciamento de testes de software de forma eficiente.

É uma ferramenta desenvolvida para facilitar e otimizar os processos de teste, ajudando equipes de desenvolvimento a garantir a qualidade e a confiabilidade de seus produtos. A definição para o Vitest na documentação de testes pode ser a seguinte: Vitest é uma plataforma de teste automatizada projetada para simplificar e aprimorar o processo de teste de software.

Ele permite que os desenvolvedores criem, executem e gerenciem testes de forma eficiente, ajudando a identificar problemas e garantir a qualidade dos aplicativos. Com o Vitest, as equipes de desenvolvimento podem automatizar testes repetitivos, acelerar o ciclo de testes e obter resultados confiáveis.

A plataforma fornece recursos abrangentes para criar cenários de teste, realizar testes de regressão, integrar-se a pipelines de CI/CD e gerar relatórios detalhados. Com o Vitest, as equipes podem melhorar a eficiência do teste, reduzir erros e entregar software de alta qualidade aos usuários finais.



Essa definição destaca as principais funcionalidades e benefícios do Vitest, incluindo automação de testes, gerenciamento de testes, integração com pipelines de CI/CD e relatórios detalhados.

Ela ressalta como o Vitest pode ajudar a equipe de desenvolvimento a melhorar a eficiência do teste e a entregar software de qualidade. Certifique-se de personalizar a definição para atender às necessidades e ao contexto específico de sua documentação de testes.

KARMA para o FrontEnd, ferramenta que vem com o Angular:

No Angular para **testes de componentes usamos o Karma** que é uma ferramenta de teste automatizado desenvolvida para o ecossistema do Angular. Ela fornece um ambiente robusto e flexível para executar testes de unidade em aplicações Angular de forma eficiente. Com o Karma, desenvolvedores podem automatizar a execução de testes em vários navegadores, garantindo a consistência e a qualidade do software em diferentes ambientes. A definição para o Karma na documentação pode ser a seguinte: Karma é uma ferramenta de teste automatizado projetada para facilitar e agilizar o processo de teste de unidade em aplicações Angular.

Com o Karma, você pode executar testes em diferentes navegadores, verificar a correta funcionalidade de componentes, serviços, diretivas e outros elementos do Angular, e obter relatórios detalhados sobre os resultados dos testes. Além disso, o Karma permite a automação de testes contínuos, recarregando automaticamente os testes quando ocorrem alterações nos arquivos de origem.

Ele é amplamente utilizado em ambientes de integração contínua (CI) e implantação contínua (CD) para garantir a qualidade do software em cada etapa do processo de desenvolvimento. Com o Karma, você pode ter confiança na estabilidade e no comportamento esperado de suas



aplicações Angular, melhorando a eficiência e a confiabilidade do processo de teste. Essa definição destaca as principais funcionalidades e benefícios do Karma, como a execução de testes em vários navegadores, a verificação de componentes e a integração com CI/CD.

Certifique-se de personalizar a definição para atender às necessidades e ao contexto específico de sua documentação de testes.

As 03 primeiras imagens da próxima página correspondem **aos testes unitários**, que estão testando as seguintes funcionalidades: **criar, atualizar e deletar usuário do sistema**.

As outras imagens seguintes estão mostrando **os testes de componentes** que realizam os testes das seguintes funcionalidades do projeto:

A página de **Sign Up** e o **O authService**, componentes que correspondem aos serviços de **página de criação de usuário** e **o serviço de autenticação**, que são funcionalidades que realizam a comunicação com serviços externos para realização da validação e autenticação dos dados.

```
1 import { faker } from '@faker-js/faker'
2 import { PrismaClient } from '@prisma/client'
3 import { test, expect, describe, afterAll, beforeAll } from 'vitest'
4
5 import { createUser } from '../models/user'
6
7 const prisma = new PrismaClient()
8
9 describe('createUser', () => {
10   beforeAll(async () => {
11     await prisma.user.delete({ where: {} })
12   })
13
14   afterAll(async () => {
15     await prisma.user.delete({ where: {} })
16     await prisma.$disconnect()
17   })
18
19   test('Should create an user', async () => {
20     const user = {
21       username: faker.internet.userName(),
22       email: faker.internet.email(),
23       password: faker.internet.password(),
24       createdAt: new Date(),
25       updatedAt: new Date()
26     }
27
28     const createdUser = await createUser(user)
29
30     expect(createdUser.username).toBe(user.username)
31     expect(createdUser.email).toBe(user.email)
32     expect(createdUser.password).toBe(user.password)
33   })
34 })
35
```

```
1 import { faker } from '@faker-js/faker'
2 import { PrismaClient } from '@prisma/client'
3 import { afterAll, beforeAll } from 'vitest'
4 import { test, expect, describe } from 'vitest'
5
6 import { deleteUser } from '../models/user'
7
8 const prisma = new PrismaClient()
9
10 const email = faker.internet.email()
11
12 describe('deleteUser', () => {
13   beforeAll(async () => {
14     await prisma.user.delete({ where: {} })
15
16     await prisma.user.upsert({
17       create: {
18         username: 'john.doe@example.com',
19         email: email,
20         password: faker.internet.password(),
21         createdAt: new Date(),
22         updatedAt: new Date()
23       },
24       update: {
25         username: 'aaaaaa',
26         updatedAt: new Date()
27       },
28       where: {
29         email: 'john.doe@example.com'
30       }
31     })
32   })
33
34   afterAll(async () => {
35     await prisma.$disconnect()
36   })
37
38   test('Excluir um usuário existente', async () => {
39     const user = await prisma.user.findUnique({ where: { email: email } })
40
41     if (!user) {
42       throw new Error('Usuário não encontrado')
43     }
44
45     const deletedUser = await deleteUser(user.id)
46     expect(deletedUser.id).toBe(user.id)
47
48     const checkUser = await prisma.user.findUnique({ where: { id: user.id } })
49     expect(checkUser).toBeNull()
50   })
51 })
52
```



```
1 import { faker } from '@faker-js/faker'
2 import { PrismaClient } from '@prisma/client'
3 import { afterAll, beforeAll } from 'vitest'
4 import { test, expect, describe } from 'vitest'
5
6 import { updateUser } from '../models/user'
7
8 const prisma = new PrismaClient()
9
10 describe('updateUser', () => {
11   beforeAll(async () => {
12     await prisma.user.delete({ where: {} })
13
14     await prisma.user.upsert({
15       create: {
16         username: 'john.doe@example.com',
17         email: 'john.doe@example.com',
18         password: faker.internet.password(),
19         createdAt: new Date(),
20         updatedAt: new Date()
21       },
22       update: {
23         username: 'aaaaaa',
24         updatedAt: new Date()
25       },
26       where: {
27         email: 'john.doe@example.com'
28       }
29     })
30   })
31
32   afterAll(async () => {
33     await prisma.$disconnect()
34   })
35
36   test('Should update an User', async () => {
37     const user = await prisma.user.findUnique({ where: { email: 'john.doe@example.com' } })
38
39     if (!user) {
40       throw new Error('Usuário não encontrado')
41     }
42
43     const updatedUser = await updateUser(user.id, { username: 'Jane Doe' })
44
45     expect(updatedUser.id).toBe(user.id)
46     expect(updatedUser.username).toBe('Jane Doe')
47   })
48 })
49
```

```
1 import { ComponentFixture, TestBed } from '@angular/core/testing';
2
3 import { SignUpComponent } from './sign-up.component';
4
5 describe('SignUpComponent', () => {
6   let component: SignUpComponent;
7   let fixture: ComponentFixture<SignUpComponent>;
8
9   beforeEach(async () => {
10     await TestBed.configureTestingModule({
11       declarations: [ SignUpComponent ]
12     })
13     .compileComponents();
14
15     fixture = TestBed.createComponent(SignUpComponent);
16     component = fixture.componentInstance;
17     fixture.detectChanges();
18   });
19
20   it('should create', () => {
21     expect(component).toBeTruthy();
22   });
23 });
24
```

```
1 import { TestBed } from '@angular/core/testing';
2
3 import { AuthService } from '../auth.service';
4
5 describe('AuthService', () => {
6   let service: AuthService;
7
8   beforeEach(() => {
9     TestBed.configureTestingModule({});
10    service = TestBed.inject(AuthService);
11  });
12
13  it('should be created', () => {
14    expect(service).toBeTruthy();
15  });
16 });
17
```

2º Apresentar um teste de sistema:

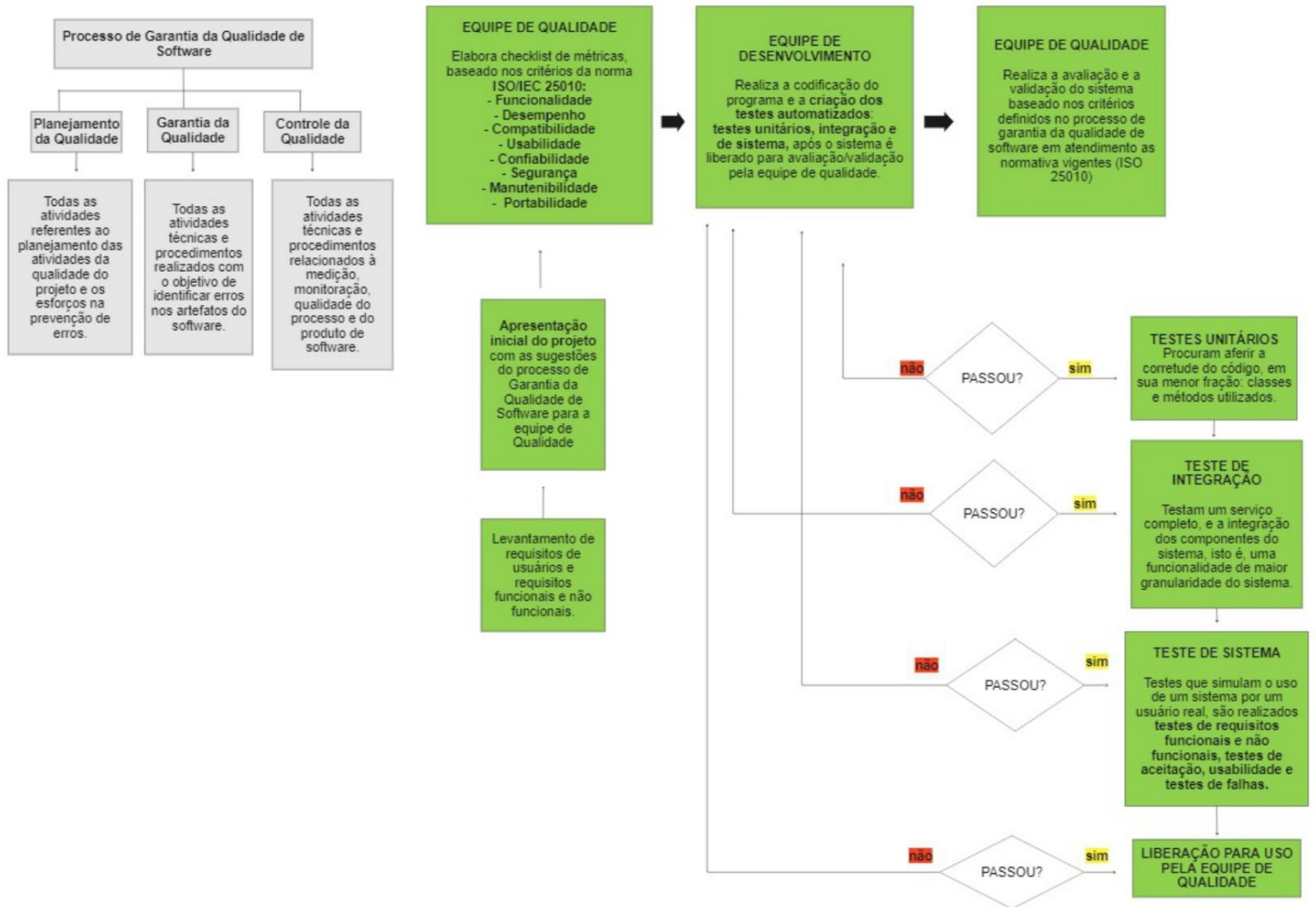
Através do link abaixo, realizamos um teste de sistema a nível de usuário, testando todas as funcionalidades do nosso projeto que pode ser conferido através do link abaixo que apresenta a interface do sistema e suas demais conexões:

https://drive.google.com/file/d/1ir1Lks2FBAbpvGVYqWwLG37Uuz-I0V_/view?usp=sharing

3º Um diagrama do processo de qualidade de software:

PROJETO:  Pass Manager

PLANO DE GERENCIAMENTO DE QUALIDADE DE SOFTWARE





4º Quatro atributos de qualidade de software e informar como foi aplicado no projeto.

Confiabilidade:

A confiabilidade refere-se à capacidade do software de executar suas funções de forma consistente e correta ao longo do tempo. Isso implica em evitar falhas e erros, bem como em lidar adequadamente com situações imprevistas.

Realizamos as devidas adequações e testes para garantir o cumprimento deste requisito não funcional que está implantado no nosso projeto de maneira a proporcionar para os usuários a confiabilidade necessária para a execução do aplicativo.

Segurança:

Proteção do software contra acessos não autorizados e uso indevido.

Nosso aplicativo possui o recurso de criptografia que permite uso de algoritmos codificados, hashes e assinaturas trazendo segurança para o armazenamento das senhas dos usuários durante todo o processo de uso do aplicativo, nosso app criptografa as senhas dos usuários e as senhas que ele pode cadastrar.

Desempenho:

O desempenho se refere à capacidade do software de executar suas tarefas de maneira eficiente, respondendo rapidamente às solicitações do usuário. Isso inclui aspectos como tempo de resposta, tempo de processamento, utilização de recursos do sistema e escalabilidade.

Por utilizarmos um framework (angular) que possui muitas ferramentas otimizadas que proporcionam a simplicidade e reutilização de código isso de certa forma garante mais



leveza na execução do app tanto local quanto remotamente, trazendo mais eficiência e desempenho para a execução do aplicativo.

Manutenibilidade

A manutenibilidade diz respeito à facilidade com que um software pode ser modificado, atualizado ou corrigido. Um software com alta manutenibilidade é aquele que possui uma estrutura clara e bem documentada, facilitando a compreensão e a realização de alterações. Além disso, um código fonte bem organizado e modularizado contribui para a manutenibilidade do software.

Nosso aplicativo possui toda estruturação do código bem organizada, e otimizada para futuras alterações e manutenções que se fizerem necessárias, o fato de termos os arquivos principais no Gith Hub nos permite ter um controle maior de versão do código para a manutenibilidade do mesmo.

Usabilidade

A usabilidade refere-se à facilidade de uso e à experiência do usuário ao interagir com o software. Um software com alta usabilidade é aquele que é intuitivo, de fácil aprendizado e eficiente na realização das tarefas desejadas pelo usuário. Isso envolve ter uma interface amigável, com boa organização das informações, feedback adequado, além de ser adaptado às necessidades e expectativas dos usuários.

Nosso aplicativo foi estruturado de acordo com as normativas pertinentes a usabilidade, e seguindo o que preceitua o Design Thinking, nos preocupamos em trazer um layout clean para o projeto, para que o usuário se preocupe apenas com o propósito que o app se destina, afinal de contas “menos é mais”, cabe informar que nosso aplicativo teve a validação dos requisitos de usabilidade do nosso professor de UX, Francisco Escobar.



Inovação e Excelência desde 1902

Segue o endereço eletrônico do nosso aplicativo:

<https://pass-manager-frontend.vercel.app/auth/sign-in>

Endereço do nosso repositório no Gith Hub:

<https://github.com/2023-1-NADS3/E9-PassManager>

MEMBROS DO PROJETO

Guilherme Poleti RA: 22023673

Heitor Dias RA: 22023799

Mary Alice Aparecida Guilherme RA: 22023051

Maurício Cardoso dos Santos Junior RA: 22023081

Mateus Quintino Vieira Santos RA: 22023854

Thais Landim de Carvalho RA: 22023274