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.

```ts
import { faker } from '@faker-js/faker'
import { PrismaClient } from '@prisma/client'
import { test, expect, describe, afterAll, beforeAll } from 'vitest'

import { createUser } from '../../models/user'

const prisma = new PrismaClient()

describe('createUser', () => {
 beforeAll(async () => {
  await prisma.user.delete({ where: {} })
 })

 afterAll(async () => {
  await prisma.user.delete({ where: {} })
  await prisma.$disconnect()
 })

 test('Should create an user', async () => {
  const user = {
   username: faker.internet.userName(),
   email: faker.internet.email(),
   password: faker.internet.password(),
   createdAt: new Date(),
   updatedAt: new Date()
  }

  const createdUser = await createUser(user)

  expect(createdUser.username).toBe(user.username)
  expect(createdUser.email).toBe(user.email)
  expect(createdUser.password).toBe(user.password)
 })
})

```

```
import { faker } from '@faker-js/faker'
import { PrismaClient } from '@prisma/client'
import { afterAll, beforeAll } from 'vitest'
import { test, expect, describe } from 'vitest'

import { deleteUser } from '../../models/user'

const prisma = new PrismaClient()

const email = faker.internet.email()

describe('deleteUser', () => {
 beforeAll(async () => {
  await prisma.user.delete({ where: {} })

  await prisma.user.upsert({
   create: {
    username: 'john.doe@example.com',
    email: email,
    password: faker.internet.password(),
    createdAt: new Date(),
    updatedAt: new Date()
   },
   update: {
    username: 'aaaaaa',
    updatedAt: new Date()
   },
   where: {
    email: 'john.doe@example.com'
   }
  })
 })

 afterAll(async () => {
  await prisma.$disconnect()
 })

 test('Excluir um usuário existente', async () => {
  const user = await prisma.user.findUnique({ where: { email: email } })

  if (!user) {
   throw new Error('Usuário não encontrado')
  }

  const deletedUser = await deleteUser(user.id)
  expect(deletedUser.id).toBe(user.id)

  const checkUser = await prisma.user.findUnique({ where: { id: user.id } })
  expect(checkUser).toBeNull()
 })
})

```

```ts
import { faker } from '@faker-js/faker'
import { PrismaClient } from '@prisma/client'
import { afterAll, beforeAll } from 'vitest'
import { test, expect, describe } from 'vitest'

import { updateUser } from '../../models/user'

const prisma = new PrismaClient()

describe('updateUser', () => {
 beforeAll(async () => {
  await prisma.user.delete({ where: {} })

  await prisma.user.upsert({
   create: {
    username: 'john.doe@example.com',
    email: 'john.doe@example.com',
    password: faker.internet.password(),
    createdAt: new Date(),
    updatedAt: new Date()
   },
   update: {
    username: 'aaaaaa',
    updatedAt: new Date()
   },
   where: {
    email: 'john.doe@example.com'
   }
  })
 })

 afterAll(async () => {
  await prisma.$disconnect()
 })

 test('Should update an User', async () => {
  const user = await prisma.user.findUnique({ where: { email: 'john.doe@example.com' } })

  if (!user) {
   throw new Error('Usuário não encontrado')
  }

  const updatedUser = await updateUser(user.id, { username: 'Jane Doe' })

  expect(updatedUser.id).toBe(user.id)
  expect(updatedUser.username).toBe('Jane Doe')
 })
})

```

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';

import { SignUpComponent } from './sign-up.component';

describe('SignUpComponent', () => {
  let component: SignUpComponent;
  let fixture: ComponentFixture<SignUpComponent>;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [ SignUpComponent ]
    })
    .compileComponents();

    fixture = TestBed.createComponent(SignUpComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});

```

```typescript
import { TestBed } from '@angular/core/testing';

import { AuthService } from './auth.service';

describe('AuthService', () => {
  let service: AuthService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(AuthService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

2º Apresentar um teste de sistema:

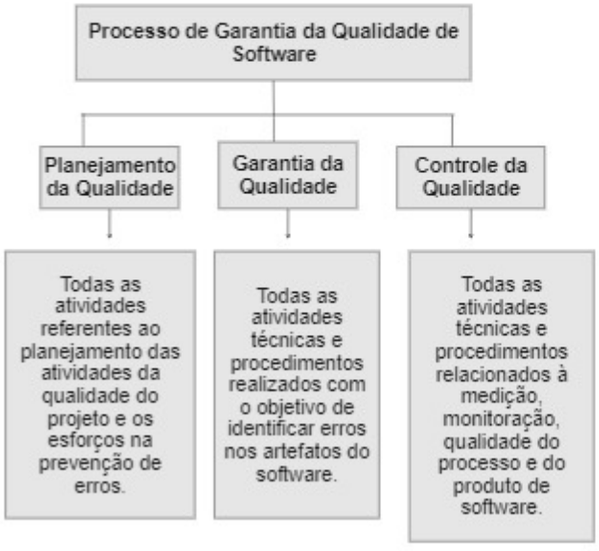
Através do link abaixo, realizamos um teste de sistema a nível de usuário, testando todas as funcionalidades do nosso projeto que pode ser conferido através do link abaixo que apresenta a interface do sistema e suas demais conexões:

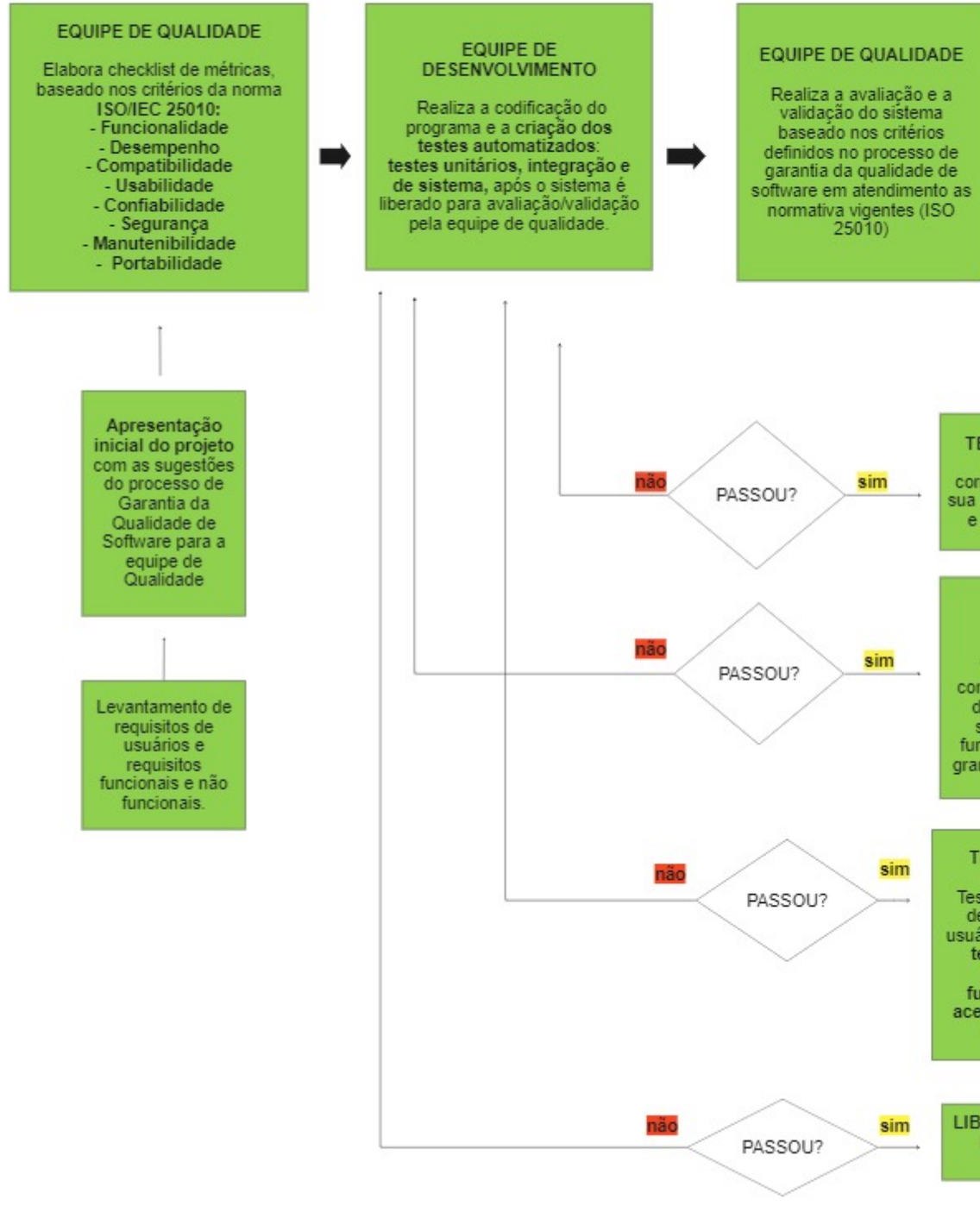
https://drive.google.com/file/d/1ir1Lks2FBAbspvGVYqWwLG37Uuz-l0V_/view?usp=sharing

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 preocupa-se 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.

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: 2

Mateus Quintino Vieira Santos RA:2

Thais Landim de Carvalho RA: 22023274