Sumário

Painel gerencial para gestão empresarial - Saúde Mental dos

Colaboradores

Grupo: GTI

Introdução

Este projeto consiste em um painel gerencial desenvolvido para auxiliar na gestão empresarial focada na saúde mental dos colaboradores da Volkswagem Brasil. Através deste sistema, é possível monitorar diversos aspectos relacionados ao bem-estar psicológico dos funcionários, oferecendo insights valiosos para tomadas de decisão estratégicas dos VPs das áreas. Para construir esse painel gerencial, foi necessário mergulhar em algumas tecnologias para a criação de um ambiente interativo e inovador. Assim essa documentação foi realizada para facilitar o uso e manutenção desse sitema visual.

Pré-requisitos

- Node.js 20.11.1 instalado e configurado.
- Angular 17 instalado.
- npm instalado.
- Docker (opcional, futura implementação).

Inicialização do projeto

Obtendo o código

- Abra o terminal e clone o repositório do projeto. Utilize o código abaixo para realizar a clonagem.

```
git clone https://github.com/Inteli-College/2024-T0004-SI09-G02-DASHBOARD.git
```

- Ou faça o download do código-fonte e descompacte-o.

Construção da imagem Docker

- Abra o cmd e execute o seguinte comando no terminal para construir a imagem Docker:

```
docker build .
```

Execução da aplicação

- Antes de tudo, instale as dependências do projeto:

```
npm install
```

- Execute o seguinte comando para executar a aplicação

```
npm start
```

ou

```
ng serve
```

A aplicação estará acessível em http://localhost:4200.

Integrantes do projeto

- Daniel Barzilai (https://www.linkedin.com/in/daniel-barzilai-061036234/)
- Dayllan Alho (https://www.linkedin.com/in/dayllan-alho/)
- Pedro Rezende (https://www.linkedin.com/in/pedrocrezende/)
- Izabela Farias (https://www.linkedin.com/in/izabellaalmeida/)
- Vinícius Fernandes (https://www.linkedin.com/in/vinicius-oliveira-fernandes/)
- Vitor Moura (https://www.linkedin.com/in/vitor-moura-de-oliveira/)

Tecnologias utilizadas

- Angular 17
- Node.js 20.11.1

Bibliotecas e Frameworks

- Angular Material -Material Icon

Links úteis

- Documentação do Angular 17 (https://angular.io/docs)
- Introdução ao Angular (https://blog.angular.io/introducing-angular-v17-4d7033312e4b)
- Node.js - Download (https://nodejs.org/en/download/current)

Testes Unitários do Componente Barchart

Este documento descreve os testes unitários para o componente Barchart.

Configuração dos Testes

Antes de executar os testes, é necessário configurar o ambiente de teste. Isso é feito utilizando as funções `beforeEach`.

```
beforeEach(async () => {
  await TestBed.configureTestingModule({}).compileComponents();
});

beforeEach(() => {
  fixture = TestBed.createComponent(BarchartComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});
```

Testes

Teste: Deve ser criado corretamente

Este teste verifica se o componente Barchart é criado corretamente.

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

Teste: Deve renderizar o elemento canvas

Este teste verifica se o componente renderiza corretamente o elemento canvas com o ID "myChart".

```
it("should render canvas element", () => {
  const canvasElement: HTMLCanvasElement =
    fixture.nativeElement.querySelector("canvas");
  expect(canvasElement).toBeTruthy();
  expect(canvasElement.id).toEqual("myChart");
});
```

Teste: Deve criar o gráfico após a inicialização da view

Este teste verifica se o método createChart é chamado após a inicialização da view.

```
it("should create chart after view initialization", () => {
  spyOn(component, "createChart");
  component.ngAfterViewInit();
  expect(component.createChart).toHaveBeenCalled();
});
```

Essa adição documenta os testes unitários do componente Barchart de forma clara e detalhada, facilitando a compreensão e manutenção do código para outros membros da equipe.

Documentação da Integração Frontend-Backend

Esta documentação visa fornecer informações sobre a integração entre o frontend e o backend do projeto voltado para o dashboard de saúde mental, utilizando Angular como principal framework no frontend e ASP.NET como backend.

Importância da Integração Frontend-Backend

A integração entre o frontend e o backend é uma ferramenta muito importante para criar aplicações web dinâmicas e funcionais. O frontend, nossa view (pensando na estrutura MVC), é responsável pela interface do usuário, interação e apresentação dos dados, enquanto o backend gerencia a lógica de negócios (Model), processamento de dados e acesso ao banco de dados.

Configurações de Ambiente

No contexto de uma aplicação Angular, o arquivo de environment tem o papel de configuração de variáveis de ambiente específicas para diferentes ambientes de implantação, como desenvolvimento, produção, teste, entre outros. Aqui estão algumas das configurações que podem ser encontradas no nosso arquivo de environment:

- `production`: Esta configuração indica se a aplicação está em modo de produção ou não. Quando definido como true, a aplicação está em modo de produção, e isso pode afetar o comportamento de certos recursos, como otimizações de código e tratamento de erros.

  Importante mencionar que o deploy da aplicação será realizado posteriormente utilizando uma instância gratuita no render.com

- `BASE_URL`: Esta é a URL base da API do backend. No nosso projeto, a URL base é http://localhost:5037, que é o endereço do servidor onde o backend está sendo executado localmente. Em um ambiente de produção, essa URL base seria substituída pelo endereço real do servidor de produção onde o backend está hospedado (posteriormente definido pelo render).

Essas configurações são essenciais porque permitem que o frontend se adapte facilmente a diferentes ambientes sem a necessidade de alterações no código-fonte. Por exemplo, durante o desenvolvimento, a URL base pode apontar para um servidor local, enquanto em produção, ela pode ser configurada para apontar para o servidor de produção. Isso torna o processo de implantação mais flexível e simplificado, permitindo que a mesma versão do código seja implantada em diferentes ambientes sem modificações adicionais. Além disso, separar as configurações de ambiente do código-fonte principal aumenta a segurança e facilita a manutenção do aplicativo em longo prazo.

Exemplo de aplicação real no nosso projeto:

```
export const environment = {
    production: false,
    BASE_URL: 'http://localhost:5037',
};
```

Funcionamento dos Serviços

Os serviços desempenham o papel de comunicação entre o frontend e o backend. Eles encapsulam a lógica para realizar chamadas HTTP para os endpoints da API do backend e fornecem uma interface simples para que os componentes do frontend possam interagir com esses dados.

Quando um componente precisa acessar dados do backend, ele injeta um serviço correspondente e chama os métodos fornecidos por esse serviço para realizar operações como recuperar, adicionar, atualizar ou excluir recursos do backend.

Configurações Necessárias

Ao definir um serviço em uma aplicação Angular, para lidar com chamadas HTTP, algumas configurações são necessárias (que neste caso algumas são bibliotecas que estamos utilizando):

- **Importação do Módulo HttpClient:** O serviço HttpClient do Angular é usado para fazer solicitações HTTP para o servidor. Portanto, é necessário importar o módulo HttpClientModule no módulo principal da aplicação ou no módulo onde o serviço está sendo utilizado.
- **Injeção do HttpClient no Serviço:** No construtor do serviço, é necessário injetar uma instância do HttpClient para que ele possa ser usado para fazer solicitações HTTP.
- **Definição de Endpoints da API:** O serviço deve definir métodos que correspondam aos diferentes endpoints da API do backend. Esses métodos devem usar o HttpClient para fazer solicitações HTTP para os endpoints específicos, como GET, POST, PUT ou DELETE.
- **Manipulação de Respostas:** O serviço deve ser capaz de manipular as respostas recebidas do backend. Isso pode incluir mapear os dados recebidos para modelos de dados definidos na aplicação, lidar com erros e retornar os dados processados para os componentes do frontend.

Exemplo do serviço aplicado em nosso projeto: `ApiConnect`

```
@Injectable({
  providedIn: 'root'
})
export class ApiConnect {
    public BASE_URL: string;

    constructor(private http: HttpClient) {
      this.BASE_URL = environment.BASE_URL;
    }

    getZenklub(): Observable<HttpResponse<any>> {
      return this.http.get<any>(`${this.BASE_URL}/api/zenklub`, { observe: 'response' });
    }

    getCID(): Observable<HttpResponse<any>> {
      return this.http.get<any>(`${this.BASE_URL}/api/cid`, { observe: 'response' });
    }

    getOrganization(): Observable<HttpResponse<any>> {
      return this.http.get<any>(`${this.BASE_URL}/api/employee`, { observe: 'response' });
    }
}
```

- Construtor: O construtor do serviço inicializa a variável BASE_URL com a URL base da API, que é definida no arquivo de ambiente. Isso permite que todas as chamadas de API sejam feitas usando essa URL base.
- Métodos HTTP: O serviço define três métodos para realizar chamadas HTTP para diferentes endpoints da API:
    - getZenklub(): Realiza uma solicitação GET para o endpoint /api/zenklub e retorna a resposta como um objeto do tipo Observable<HttpResponse>.
    - getCID(): Realiza uma solicitação GET para o endpoint /api/cid e retorna a resposta como um objeto do tipo Observable<HttpResponse>.
    - getOrganization(): Realiza uma solicitação GET para o endpoint /api/employee e retorna a resposta como um objeto do tipo Observable<HttpResponse>.

Organização por meio de DTO's

O DTO (Data Transfer Object) é um método de organização que contém a definição dos dados que serão manipulados entre o frontend e o backend. Ele define a estrutura dos dados que serão transmitidos entre as duas partes, garantindo uma comunicação eficiente e consistente. Segue um contexto de sua importância:

- Padronização de Dados: O DTO permite que ambas as partes da aplicação "concordem" com a estrutura dos dados a serem trocados. Isso evita ambiguidades e facilita a interpretação dos dados tanto pelo frontend quanto pelo backend.

- Separação de Responsabilidades: Ao definir uma estrutura de dados específica para a comunicação, o DTO ajuda a manter a separação de responsabilidades entre as duas partes da aplicação, promovendo uma arquitetura mais modular e escalável.

- Redução da Sobrecarga de Dados: O DTO permite enviar apenas os dados necessários entre o frontend e o backend, reduzindo a sobrecarga de comunicação.

Lógica na Definição do DTO

Exemplo de utilização de DTO para trazer informações dos colaboradores no banco de dados para o frontend:

```
export type OrganizationDetailsOne = {
    title: string;
    categories: string[];
    series: OrganizationDetails[];
};

export type OrganizationDetails = {
    id: number,
    cargo: string,
    centroCusto: string,
    textoRh: string,
    notaGptw: string,
    notaStiba: string,
};
```

- OrganizationDetailsOne: Este tipo de dados representa um objeto que contém detalhes gerais de uma organização. Ele possui três propriedades que são padrão de uma biblioteca específica de gráficos (ApexCharts):
    - title
    - categories
    - series: Uma matriz de objetos do tipo OrganizationDetails, representando os detalhes individuais de cada funcionário na organização.
- OrganizationDetails: Este tipo de dados representa os detalhes individuais de um funcionário na organização. Ele possui as seguintes propriedades:
    - id: O identificador único do funcionário.
    - cargo: O cargo ocupado pelo funcionário.
    - centroCusto: O centro de custo associado ao funcionário.
    - textoRh: Texto relacionado ao setor de Recursos Humanos.
    - notaGptw: Nota atribuída pela empresa Great Place to Work.

- notaStiba: Nota atribuída pela empresa STIBA.

Nota: Importante citar que, no nosso projeto, algumas informações estão sendo referenciadas direto nos arquivos de typescript, como por exemplo:

```
cards: {
    name: string;
    department: string;
    StibaScore: number;
    GPTWScore: number;
    Rating: string;
    }[] = [];
```

Endpoints Disponíveis pela WebApi

CID

1. GET parâmetro de entrada `/api/cid`

- **Descrição:** Retorna as informações do método de internação.
- **Formato da Resposta:** JSON
- **Exemplo de Resposta:**

```
{
  "mes": "Out",
  "n_pessoal": 4020504,
  "atestados": 1,
  "dias": "15",
  "diretoria": "OPERAÇÕES",
  "unidade": "SCA",
  "genero": "Masculino",
  "categoria": "HD",
  "cid": "F430",
  "descricaoDetalhada": "Reação aguda ao \"stress\"",
  "descricaoResumida": null,
  "diagnosticoAtestadoInicial": null,
  "causaRaiz": null,
  "outros": null,
  "jornada": null,
  "quantidade": 0
}
```


2. GET parâmetro de entrada `/api/cid/

- **Descrição:** Retorna as informações do método de internação por unidade.
- **Formato da Resposta:** JSON
- **Exemplo de Resposta:**


```
{
  "mes": null,
  "n_pessoal": 0,
  "atestados": 0,
  "dias": null,
  "diretoria": null,
  "unidade": "ANC",
  "genero": null,
  "categoria": null,
  "cid": null,
  "descricaoDetalhada": null,
  "descricaoResumida": null,
  "diagnosticoAtestadoInicial": null,
  "causaRaiz": "Não identificado",
  "outros": null,
  "jornada": null,
  "quantidade": 118
}
```


Employee

1. GET parâmetro de entrada `/api/employee`

- **Descrição:** Retorna todos os funcionários cadastrados no sistema pelo atributo npessoal.
- **Parâmetros de Entrada:** Nenhum.
- **Formato da Resposta:** JSON
- **Exemplo de Resposta:**


```
[
  {
    "id": 0,
    "nPessoal": 4053298,
    "sgEmp": "ME",
    "textoRh": "ANC",
    "centroCst": 2800,
    "centroCusto": "PRESIDENCIA",
    "cargo": "SECRETARIA DO CEO",
    "dataNascimento": "09/07/1994"
  }
]
```


2. GET parâmetro de entrada `/api/employee/

- **Descrição:** Retorna um funcionário específico pelo seu ID.
- **Parâmetros de Entrada:**
  - `id` (int): ID do funcionário que deseja ser recuperado.
- **Formato da Resposta:** JSON
- **Exemplo de Resposta:** Não implementada no momento

GPTW

1. **GET** parâmetro de entrada `/api/gptw`

   - **Descrição:** Retorna todos os registros da tabela gptw
   - **Formato da Resposta:** JSON
   - **Exemplo de Resposta:**

```
[
  {
    "nniveis": 1,
    "uoAbrev": "B",
    "umNvel": "B",
    "doisNvel": null,
    "tresNvel": null,
    "quatroNvel": null,
    "cincoNvel": null,
    "seisNvel": null,
    "seteNvel": null,
    "oitoNvel": null,
    "npess": 4014386,
    "rrh": "BPA1",
    "local": "Anchieta",
    "empr": 500,
    "gremp": 1,
    "grempregados": null,
    "sgemp": "ME",
    "centrocst": 2800,
    "unidorg": 20261334,
    "descrUo": "PRESIDENCIA VWB",
    "uoAbrevUm": "B",
    "idadedoempregado": 36,
    "gn": 2,
    "tpausp": null,
    "txttipopresenaausn": null,
    "incio": "00.00.0000",
    "fim": "00.00.0000",
    "cargo": 20277458,
    "cargoUm": "SECRETARIA DO COO",
    "dataAdm": "18.05.2007",
    "ano": null,
    "engajamentoPorcentagem": null,
    "indexVerdade": null,
    "auditCultura": null
  }
  ]
```


2. **GET** parâmetro de entrada `/api/gptw/engajamento`

    - **Descrição:** Retorna os percentuais de engajamento da gptw ao longo dos anos, podendo ser monitorado para compreender a veracidade dos dados para com a realidade.
    - **Formato da Resposta:** JSON
    - **Exemplo de Resposta:**

```
[
  {
    "nniveis": 0,
    "uoAbrev": null,
    "umNvel": null,
    "doisNvel": null,
    "tresNvel": null,
    "quatroNvel": null,
    "cincoNvel": null,
    "seisNvel": null,
    "seteNvel": null,
    "oitoNvel": null,
    "npess": 0,
    "rrh": null,
    "local": null,
    "empr": 0,
    "gremp": 0,
    "grempregados": null,
    "sgemp": null,
    "centrocst": 0,
    "unidorg": 0,
    "descrUo": null,
    "uoAbrevUm": null,
    "idadedoempregado": 0,
    "gn": 0,
    "tpausp": null,
    "txttipopresenaausn": null,
    "incio": null,
    "fim": null,
    "cargo": 0,
    "cargoUm": null,
    "dataAdm": null,
    "ano": "2022",
    "engajamentoPorcentagem": "22",
    "indexVerdade": "75",
    "auditCultura": "71"
  }
]
```


HealthInsurance

1. **GET** parâmetro de entrada `/api/health-insurance`

    - **Descrição:** Retorna os registros de plano de saúde da amil, até o momento não integrado com os outros planos de saúde, que são: ps_midservice e ps_unimed por questões de eficiência de desenvolvimento em tempo insuficiente.

- **Formato da Resposta:** JSON
- **Exemplo de Resposta:**


```
[
  {
    "id": 0,
    "nomeCliente": "VOLKSWAGEN",
    "nomeBanco": "AMIL",
    "nomeProduto": "SAUDE",
    "codigoPessoa": "240-CC9-795-987",
    "nomePaciente": null,
    "codigoPaciente": 71695740,
    "matricula": 1637592,
    "cpf": "24672252833",
    "dataNascimento": "22/05/1979",
    "sexo": "M",
    "parentesco": "TITULAR",
    "codigoPlano": 58787,
    "nomePlano": "ONE LINCX LT3 NAC QP COPART PJCE",
    "grupoFamiliar": 1637592,
    "dataEvento": "15/11/2022",
    "codigoProcedimentoFinal": 10101012,
    "descricaoProcedimentoFinal": "EM CONSULTORIO (NO HORARIO NORMAL OU PREESTABELECIDO)",
    "codigoServico": 10101012,
    "descricaoServico": "Consulta Em Consultório (No Horário Normal Ou Preestabelecido)",
    "tipoEventoCbhpm": "CONSULTAS",
    "tipoEventoFinal": "CONSULTAS",
    "tipoUtilizacao": "REDE",
    "numeroSubFatura": 149545000,
    "numeroContrato": 149545000,
    "codigoPrestador": 14230488,
    "nomePrestador": "READ PSIQUIATRIA",
    "especialidade": "Psiquiatria ",
    "sinistro": "110",
    "numeroEvento": 1,
    "planta": "SP + ABC"
  }
]
```


2. **GET** parâmetro de entrada `/api/health-insurance/{id}`

   - **Descrição:** Retorna um registro de plano de saúde específico pelo seu ID.
   - **Parâmetros de Entrada:**
     - `id` (int): ID do plano emitido que deseja ser recuperado.
   - **Formato da Resposta:** JSON
   - **Exemplo de Resposta:** [Não implementada no momento]

Organization

1. **GET** parâmetro de entrada `/api/organization`

    - **Descrição:** Retorna a estrutura de organização de cada diretoria, incluindo seu VP e notas avaliativas.
    - **Formato da Resposta:** JSON
    - **Exemplo de Resposta:**


```
[
  {
    "areaDiretoria": null,
    "unidadeUO": "B-OA Unidade Anchieta",
    "nomeVps": null,
    "nomeUnidade": "ANCHIETA",
    "notaStiba2022": 84,
    "notaStiba2023": 82.4,
    "notaEmpresa2022": 86.6,
    "notaEmpresa2023": 85.7
  }
]
```


2. **GET** parâmetro de entrada `/api/organization/notas-unidades`

    - **Descrição:** Retorna a estrutura de organização de cada unidade, incluindo suas notas avaliativas.
    - **Formato da Resposta:** JSON
    - **Exemplo de Resposta:**


```
[
  {
    "areaDiretoria": null,
    "unidadeUO": "B-OA Unidade Anchieta",
    "nomeVps": null,
    "nomeUnidade": "ANCHIETA",
    "notaStiba2022": 84,
    "notaStiba2023": 82.4,
    "notaEmpresa2022": 86.6,
    "notaEmpresa2023": 85.7
  }
]
```


Stiba

1. **GET** parâmetro de entrada `/api/stiba`

    - **Descrição:** Retorna todos os registros da tabela Stiba, incluindo a quantidade de respondentes e o número do participante de cada unidade.
    - **Formato da Resposta:** JSON

- **Exemplo de Resposta:**


```
[
  {
    "descricaoUO": "Volkswagen do Brasil",
    "elegiveis": 11.976,
    "respondentes": 11.339,
    "participante": 95,
    "notaStiba": "85,7",
    "qUm": "80,4",
    "qDois": "85,4",
    "qTres": "81,1",
    "qQuatro": "81,6",
    "qCinco": "83,8",
    "qSeis": "84,9",
    "qSete": "80",
    "qOito": "89,6",
    "qNove": "84,8",
    "qDez": "85,6",
    "qOnze": "85,9",
    "qDoze": "82,7",
    "qTreze": "84,7",
    "qCatorze": "91,4",
    "qQuinze": "84,1",
    "qDezesseis": "83,7",
    "qDezessete": "85,6",
    "qDezoito": "90,5",
    "qDezenove": "90,9",
    "qVinte": "92,9",
    "qVinteUm": "83",
    "qVinteDois": "91,9",
    "qVinteTres": "90,6",
    "qVinteQuatro": "84,6",
    "notaPrincipiosVW": 0
  }
]
```


Zenklub

1. **GET** parâmetro de entrada `/api/zenklub`

   - **Descrição:** Retorna dados das sessões realizadas no Zenklub por período.
   - **Formato da Resposta:** JSON
   - **Exemplo de Resposta:**

```
[
  {
    "periodo": "2023-04-01",
    "mes": 4,
    "nome": null,
    "nPessoal": 2048400,
    "codigoValidacao": 2674613998,
    "departamento": "Curitiba",
    "totalSessoes": 2
  }
]
```


2. **GET** parâmetro de entrada `/api/zenklub/dep`

    - **Descrição:** Retorna soma das sessões realizadas mensalmente no Zenklub por departamento.
    - **Formato da Resposta:** JSON
    - **Exemplo de Resposta:**


```
[
  {
    "periodo": null,
    "mes": 11,
    "nome": null,
    "nPessoal": 0,
    "codigoValidacao": 0,
    "departamento": "Taubate",
    "totalSessoes": 106
  }
]
```


Métodos HTTP Suportados

- **GET:** Utilizado para recuperar as informações da API.

Método GET

- **Objetivo:** O método GET é utilizado para solicitar a representação de um recurso específico a partir do servidor. Ele não modifica o estado do servidor ou dos recursos, apenas recupera os dados.
- **Utilização:** O método GET é comumente utilizado em APIs RESTful para recuperar recursos específicos do servidor. Ele é usado para consultar dados.
- **Parâmetros:**
    - **URL:** A URL do recurso desejado é especificada na linha de solicitação. Por exemplo, `/api/recurso` indica que queremos obter o recurso localizado em `/api/recurso`.

    - **Headers:** Opcionalmente, cabeçalhos HTTP podem ser incluídos na solicitação para fornecer informações adicionais, como autenticação ou preferências de resposta.

- **Resposta:**

    - Quando uma solicitação GET é bem-sucedida, o servidor responde com um código de status `200 OK` e os dados solicitados no corpo da resposta.
    - Em caso de erro, o servidor pode retornar diferentes códigos de status para indicar o problema, como `404 Not Found` se o recurso não for encontrado ou `500 Internal Server Error` se ocorrer um erro interno no servidor.

Exemplos de uso:

- `http://localhost:5037/${parâmetro de entrada}`

Neste exemplo, estamos fazendo uma solicitação GET para recuperar informações sobre o endpoint específico. O servidor responderá com os dados desse funcionário, se estiverem disponíveis. A resposta aqui será a visualização do JSON de forma "crua".

Processo de Deploy com GitHub Actions e Docker

Este documento fornece uma visão geral do processo de deploy da aplicação Angular do nosso projeto, utilizando um pipeline de Integração Contínua (CI) e Entrega Contínua (CD) implementado com GitHub Actions com containerização no Docker.

GitHub Actions Workflow

O arquivo YAML no diretório `.github` define o pipeline CI/CD e é dividido em três principais trabalhos: `build`, `test` e `deploy`.

Build Job

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v3
        with:
          node-version: '18.18.0'
      - run: npm install -g @angular/cli@latest
      - run: npm install
      - run: ng build --configuration=production
```

Processo realizado no **build**:

- Checkout: Obtém o código-fonte do repositório para o runner do GitHub.
- Setup Node.js: Configura o ambiente com a versão especificada do Node.js.
- Install Angular CLI: Instala a CLI do Angular globalmente.
- Install dependencies: Instala todas as dependências listadas no package.json.
- Build: Compila o aplicativo Angular para produção.

Test Job

```
jobs:
  test:
    needs: build
    runs-on: ubuntu-latest
    steps:
      # Steps similar to 'build' job but for running unit tests
```

Processo adicional realizado no "Test Job":

- needs: Especifica que o trabalho de teste depende da conclusão bem-sucedida do trabalho de construção.
- Unit Tests: A adição de teste aqui ela é feita a partir dos arquivos em typescript de spec. No entanto, os passos para execução dos testes unitários padrão do angular para tudo estão comentados.
    - Para ativá-los, remova os comentários e assegure-se de que a CLI do Angular e as dependências estejam instaladas antes de executar `ng test`.

Deploy Job

```
jobs:
  deploy:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: johnbeynon/render-deploy-action@v0.0.8
        with:
          service-id: ${{ secrets.RENDER_SERVICE_ID }}
          api-key: ${{ secrets.RENDER_API_KEY }}
```

Processo adicional para o deploy final:

- Deploy to Production: Este passo usa uma action do GitHub para implantar o aplicativo na plataforma especificada, neste caso, o Render.
- secrets: Utiliza secrets do repositório para autenticar no serviço de hospedagem (chaves de acesso do Render).

Docker Configuration

```
FROM node:alpine
WORKDIR /usr/src/app
COPY . .
RUN npm install -g @angular/cli
RUN npm install
RUN ng build
CMD ["ng", "serve", "--host", "0.0.0.0"]
```

A configuração do Docker constrói uma imagem contendo o aplicativo Angular:

- node:alpine: Uma imagem leve do Docker com Node.js instalado é usada como base.
- WORKDIR: Define o diretório de trabalho no container.
- COPY: Copia os arquivos do projeto para o container.
- npm install: Instala as dependências do projeto dentro do container.
- ng build: Compila o aplicativo Angular dentro do container.
- CMD: Define o comando para rodar o aplicativo Angular.

CI/CD Pipeline Aplicado no projeto

O pipeline de CI/CD aplicado funciona da seguinte forma:

- Integração Contínua (CI): No push para o branch principal ou em um pull request, o GitHub Actions executa o build para compilar o código e, opcionalmente, executar os testes unitários (test).

- Entrega Contínua (CD): Se o build e test passarem, o trabalho deploy é executado para implantar o aplicativo compilado em um ambiente de produção.

Este pipeline garante que cada alteração no código seja automaticamente construída e testada antes de ser entregue ao ambiente de produção, minimizando os riscos associados ao processo de deploy manual e aumentando a eficiência do ciclo de vida do desenvolvimento de software.

Dashboard em Produção

O dashboard da aplicação está atualmente implantado e em funcionamento no ambiente de produção. Ele é construído usando Angular e é hospedado em um ambiente de produção com Node.js e no Render.com

Tecnologias Utilizadas

a. **Angular:** Estrutura Typescript de plataforma única para construir aplicativos web dinâmicos.

b. **Node.js:** Ambiente de tempo de execução JavaScript que permite executar JavaScript no servidor.

Fluxo de Trabalho de Implantação

O processo de implantação do dashboard no ambiente de produção envolve os seguintes passos:

Compilação: A aplicação Angular é compilada para produção, resultando em arquivos estáticos otimizados. Isso inclui a minimização de arquivos, a remoção de código não utilizado e a otimização de recursos para garantir que a aplicação seja executada de forma eficiente no ambiente de produção.

Implantação: Os arquivos compilados são implantados em um ambiente de produção. A implantação é o último passo para disponibilizar a aplicação para os usuários finais, ela garante uma implantação suave e livre de erros para evitar interrupções no serviço e garantir uma experiência do usuário positiva.

Pipeline de CI/CD

- Configuração do Pipeline de CI/CD com GitHub Actions

O pipeline de CI/CD é configurado utilizando o GitHub Actions. Ele automatiza o processo de integração contínua e entrega contínua do seu dashboard. O pipeline inclui as seguintes etapas:

1. Compilação: Compilação da aplicação Angular para produção.
2. *Teste: Execução de testes unitários para garantir a integridade do código.
3. Implantação: Implantação do dashboard no ambiente de produção.

*Na execução de testes unitários, podemos garantir a integridade do código. Os testes são executados automaticamente para verificar se o código funciona conforme esperado, além disso é um parâmetro importante para medir a qualidade do código. Eles ajudam a identificar problemas e regressões no código antes da implantação.

Arquivos de Configuração

Arquivo do GitHub Actions (em formato yaml)

```yaml
# Exemplo de aplicação na branch develop

name: teste.Dashboard.WebApi

on:
  push:
    branches:
      - develop
  pull_request:
    branches:
      - develop

jobs:
  build:
    name: "Build"
    runs-on: ubuntu-latest

    steps:
    - name: "Checkout"
      uses: actions/checkout@v2

    - name: "Set Node.js"
      uses: actions/setup-node@v3
      with:
        node-version: '18.18.0'

    - name: "Install Angular CLI"
      run: npm install -g @angular/cli@latest

    - name: "Install dependencies"
      run: npm install

    - name: "Build"
      run: ng build --configuration=production
```

Dockerfile

```
FROM node:alpine

WORKDIR /usr/src/app

COPY . /usr/src/app/

RUN npm install -g @angular/cli@^17.2.0

RUN npm install

RUN ng build

CMD ["npm", "start", "--host", "0.0.0.0"]
```

Observações para rodar o pipeline:

- Certifique-se de ajustar o arquivo Dockerfile conforme necessário para corresponder à sua configuração específica de implantação.
- O pipeline precisa estar configurado para a ramificação da branch específica (denotada em cada um dos arquvios `yaml`)