FOURBANK POC

Documento criado por Jhonatan Silva da Costa - 11/04/2025

Objetivo

Este documento tem como objetivo apresentar tecnologias e bibliotecas sugeridas além de apresentar modos de uso de alguns componentes reutilizáveis presente na POC

Tecnologias e Bibliotecas

Node.js

Versão: 22.4.0

Angular

Versão: 19.2.0

Descrição: Proposto nessa POC o uso do angular na sua versão mais atualizada até o momento, desfrutando de todas as possibilidades de melhoria de performance e organização. Recursos como Stand alone components e Lazy Load de rotas e organização melhor das mesmas.

Bootstrap

Versão: 5.3.5

Descrição: Proposto o uso de bootstrap, pois essa biblioteca traz consigo o sistema de grid que deixa muito mais simples e prático a responsividade, além de ser mais leve e dar mais possibilidades de alterações de estilos.

Bootstrap Icons

versão: 1.11.3

Descrição: Proposto o uso de bootstrap icons, pois facilita o uso de ícones no código além de ter uma coleção ampla de ícones a disposição.

Chart.js e ng2-chart

Versões: chart.js - 4.4.8; ng2-chart: 8.0.0

descrição: Proposto o uso da biblioteca de gráfico o ng2-chart, que é feito pensando no Angular, traz o melhor em gráficos do chart.js para dentro da plataforma, com mais facilidade de adaptação a regras de negócios.

Padrão de código

Grid e Responsividade

Pensando na melhor solução para responsividade, foram criado os componentes com o sistema de grid no bootstrap. Consiste em usar linhas e colunas para criar o esqueleto que se adapta para outras telas.

Observação: Pode haver necessidade de ajustes dependendo do tipo da tela.

Exemplo de uso

```
<div class="container-fluid">
  <div class="row">
    <div class="col">
      <h1>Page title</h1>
    </div>
  </div>

  <div class="row">
    <div class="col">
      <p>First container</p>
    </div>
    <div class="col">
      <p>Second container</p>
    </div>
  </div>
</div>
```

Estruturas de pastas

Foi pensado na estrutura do projeto antes do início da codificação, ou seja, a estrutura apresentada irá separar cada função em pastas específicas. Foi adotado dessa forma para melhor organização e manutenções futuras.

app/
├── guards/
├── interceptors/
├── componentes/
├── interfaces/
├── pipes/
├── services/
├── shared/
├── views/
└── endpoints/

assets/
├── img/
├── Icons/
└── outros/

Observação: Pode ser necessário alteração ou adição de novas pastas dependendo da regra de negócio.

App

Pasta pai do sistema, nela conterá todas as outras pastas referente à features do sistema, como páginas, componentes e outros

Guards

Pasta onde terá a lógica de controle de acesso a páginas do sistema, como usuário logado, permissões de usuário entre outros.

Interceptors

Pasta onde ficará os interceptadores de requisições, usado para injetar tokens ou outras informações que uma ou mais requisições precisarem.

Componentes

Pasta onde se encontra todos os componentes reutilizáveis do sistema, como cards, gráficos e outros. Os componentes são feitos pensando em realização no sistema inteiro se necessário.
É separado em uma pasta específica onde cada componente tem seu html, scss e ts. Contendo lógica e seus próprios estilos. Os mesmos também tem seus próprios módulos, ou seja, só será chamado e carregado o que for necessário para o funcionamento do mesmo.

Interfaces

Pasta onde se encontram todas as interfaces do sistema, geralmente organizadas por pastas. Essas interfaces são a identificação dos tipos de dados que vai ser consumido dentro do sistema, é utilizadas no sistema todos, tipando variáveis, chamadas http e outros.

Pipes

Pasta que contém formatadores e/ou modificadores de elementos, como exemplo, um verificador de texto para status de transações. Pipes são criadas de forma a serem utilizadas em todo o sistema se necessário.

Services

Pasta que contém os serviços de consumo de dados, usado para armazenar chamadas http no sistema bem como tratamento específico de lógicas dentro do sistema, dependendo da regra pode ser utilizado métodos dentro do sistema, mediante a injeção via construtor.

Shared

Pasta onde contém arquivos compartilhados, diferente da pasta componentes, é geralmente utilizada para compartilhar arquivos únicos, não componentes com módulos. Um bom exemplo é scss de efeitos.

Views

Pasta que contém todas páginas de fato do sistema, ou seja, os componentes que possuem rotas e que serão renderizados para o usuário.

Assets

Pasta que guarda conteúdos estáticos do sistema, como imagens, documentos e outros.

imgs

pasta para guardar imagens do sistema, ao guardar uma imagem nessa pasta, deixo o conselho de passar a mesma em algum sistema de compressão antes, para ajudar no desempenho e carregamento das mesmas.

Icons

Pasta que guarda ícones personalizados do sistema, foi recomendado o uso da biblioteca bootstrap icons, porém, caso possua um ícone personalizado, será aqui que se encontrará o mesmo.

Outros

Pode conter outras pastas para arquivos diversos, como documentos e outros.

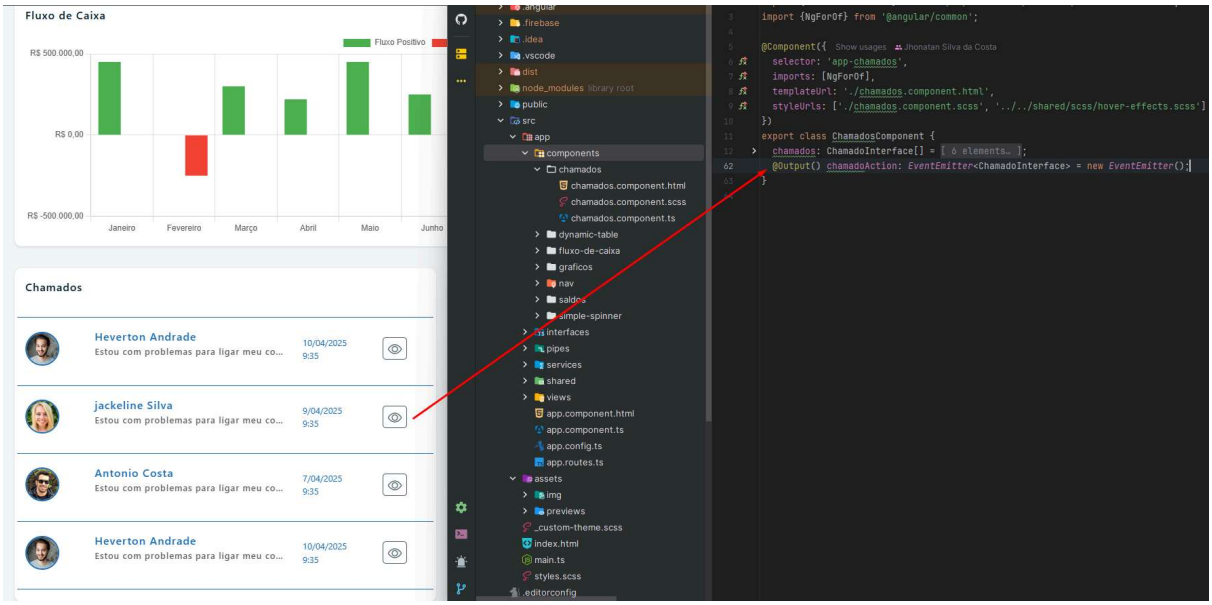
Componentes

Chamados

Foi criado um componente próprio para gerenciar os chamados, foi pensado caso precise ser reutilizado dentro do sistema.

Requisitos

Chamados: Precisa de uma Lista com os chamados, tem uma interface própria e será pensando em fazer o consumo da API via observables, ou seja, idenpendente de onde ele se encontra, se ouver mudanças na API de chamados, ele ouvirá e atualizará.
Ação de clique: Dentro do componente há um ícone de olho, que ao ser clicado dispará um emitter que passará o dado clicado no formato de chamado Interface.



O componente pai que chamar esse módulo deverá configurar uma função para tratar desse dado.

Interfaces:

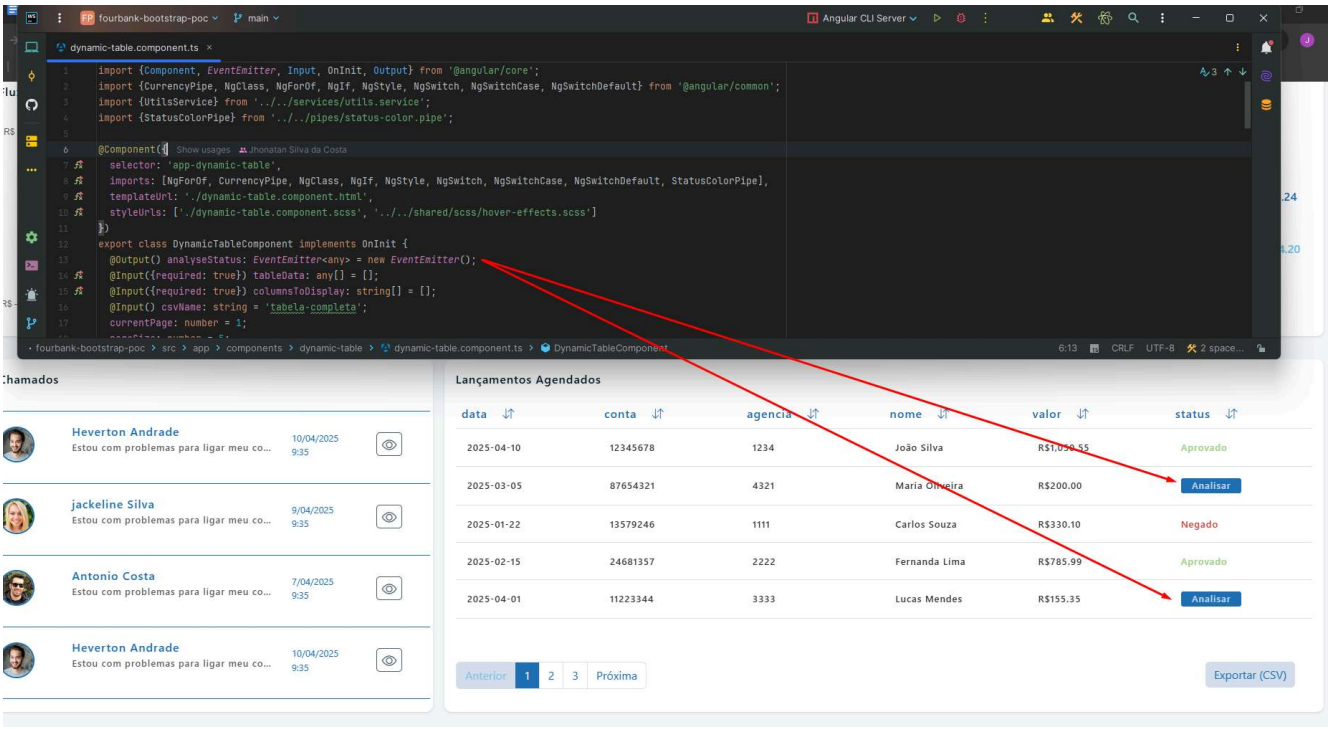
```
export interface ChamadoInterface {
  id: string;
  nome: string;
  descricao: string;
  hora: string;
  data: string
  foto?: string;
}
```

Tabela Dinâmica

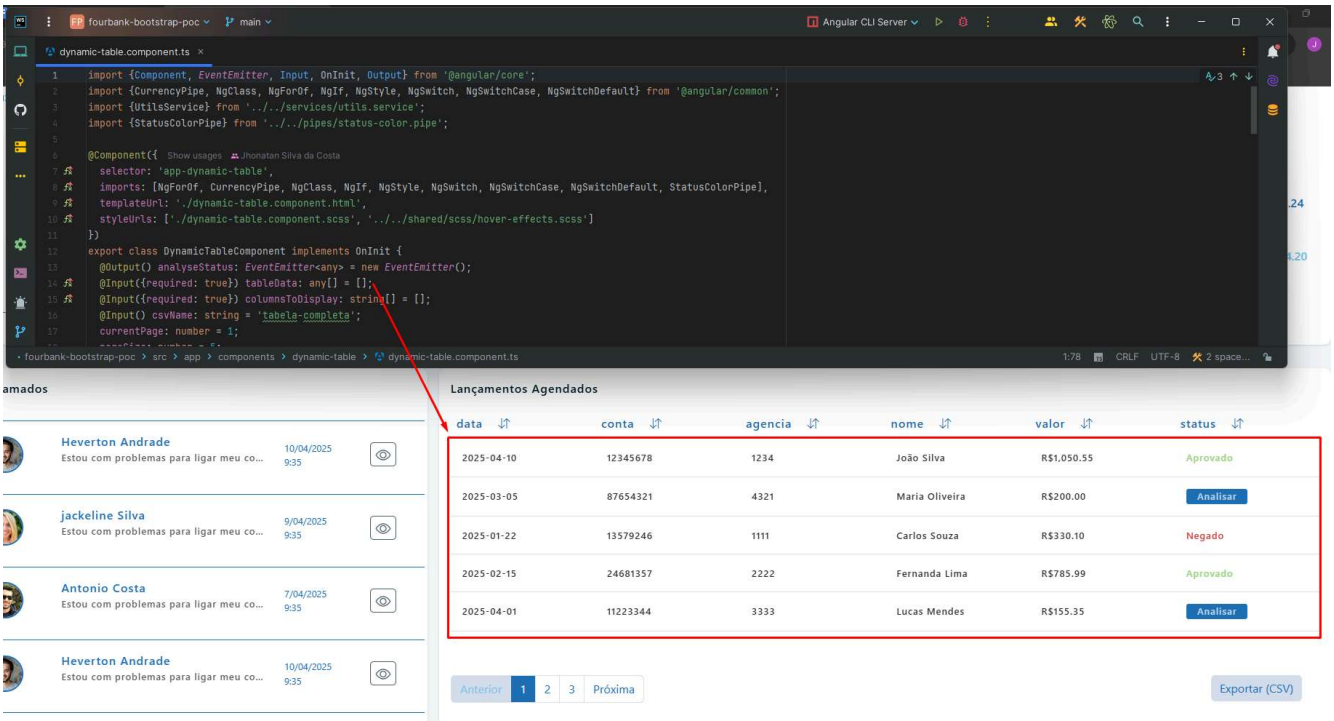
Foi criado um componente dinâmico, onde pode ser usado em qualquer lugar do sistema. Tabela completa que identifica status, também aplica sort e paginação automaticamente, além de uma lógica para exportar CSV.

Requisitos

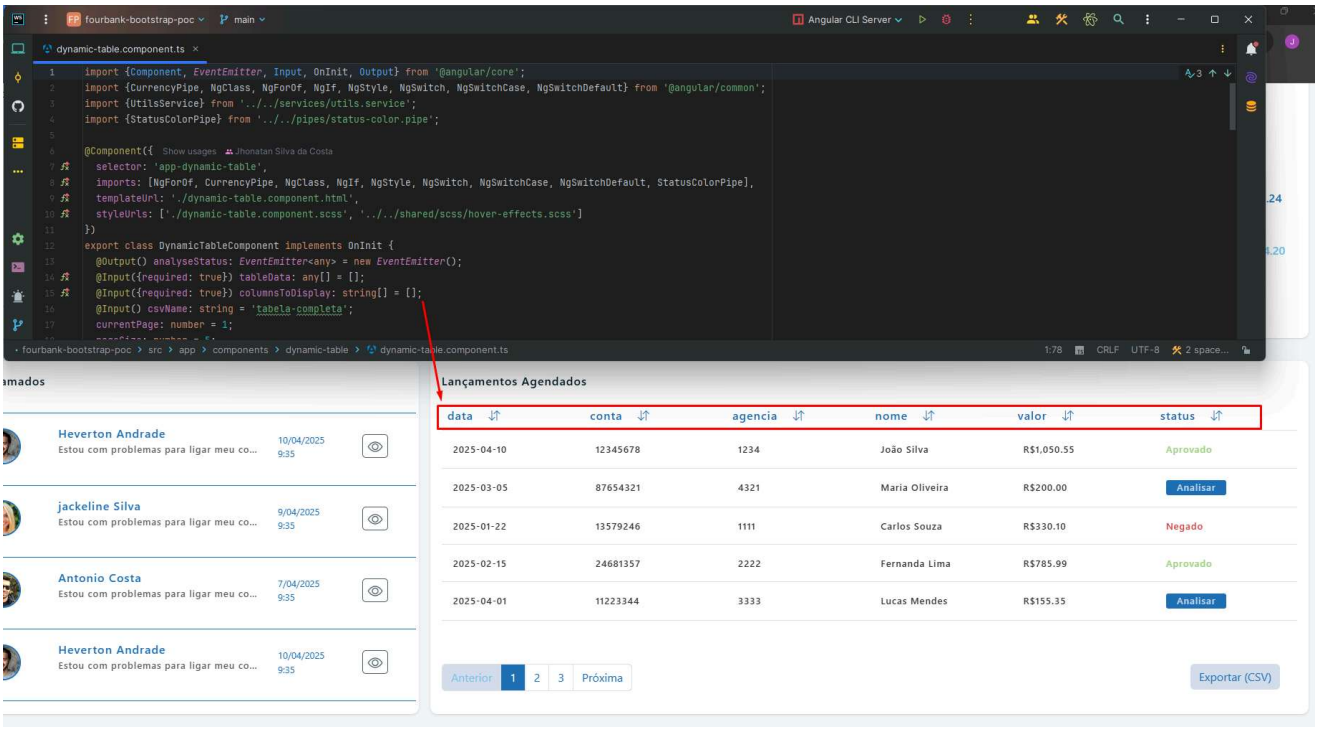
Ação de clique: Na tabela, tem uma opção automática que analisa os dados, se for encontrado uma coluna status e seu valor for analizar, será configurado um botão, que ao clicar passa o dado clicado daquela linha, ou seja, precisa ter uma função para captar esse clique, se necessário, dentro do componente pai.



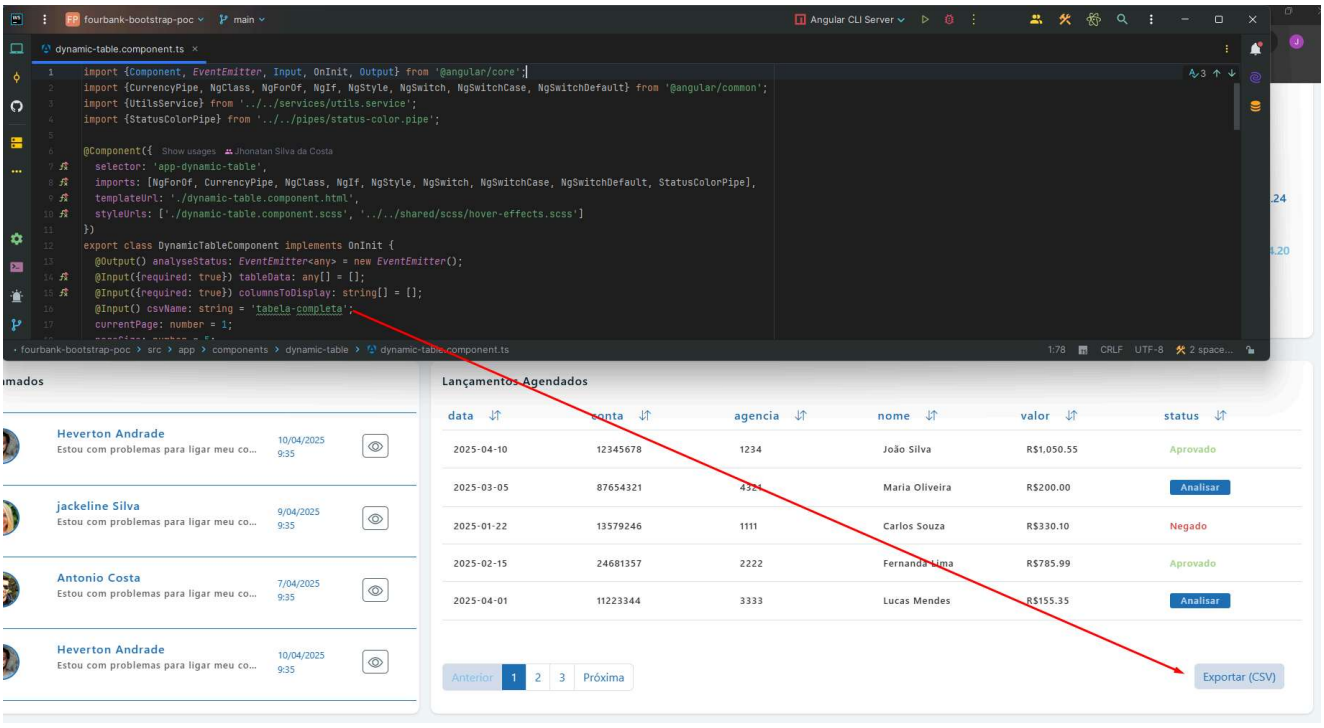
Dados da Tabela: tabela é montada dinamicamente, ou seja, basta passar um array de objetos para o componente que ele montará a tabela completa, com sort e paginação.



Colunas a serem mostradas: Como a tabela é dinâmica, deve se passar para o componente, em formato de array de string, quais colunas devem ser mostradas, os valores dentro do array de string deve ser, exatamente, os nomes das chaves do array de objetos que foi passado para o campo de dados da tabela.

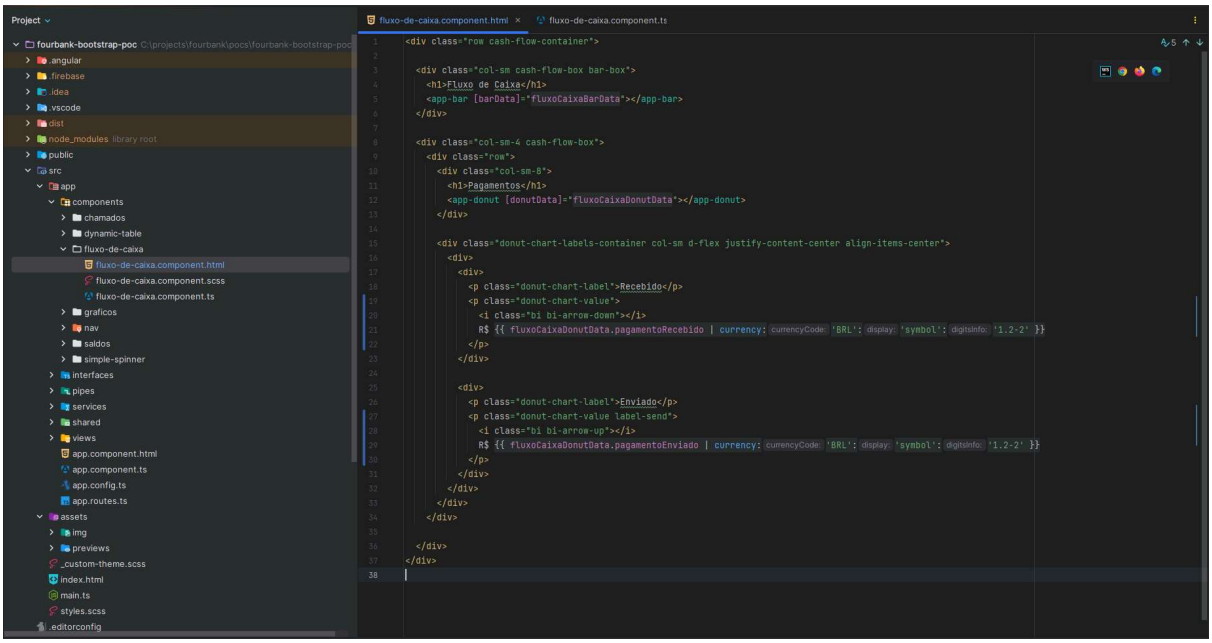


Nome do CSV(opcional): Na tabela tem um botão de exportação em formato CSV que tem a opção de consumir um nome para o arquivo a ser exportado, caso não seja informado, o padrão é “tabela-completa”.



Fluxo de Caixa

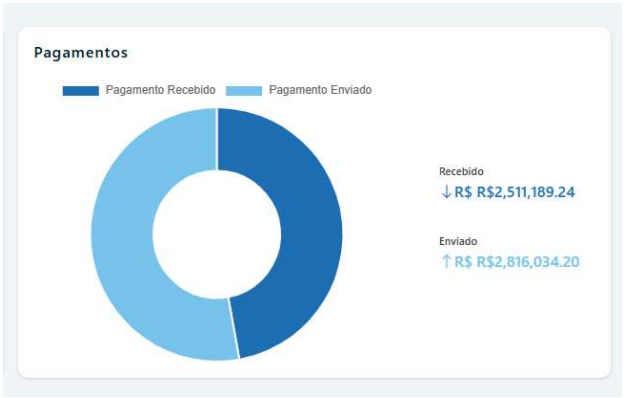
Componente responsável por montar os dois gráficos completos no sistema, pode ser usado em outros lugares do sistema.



Requisitos

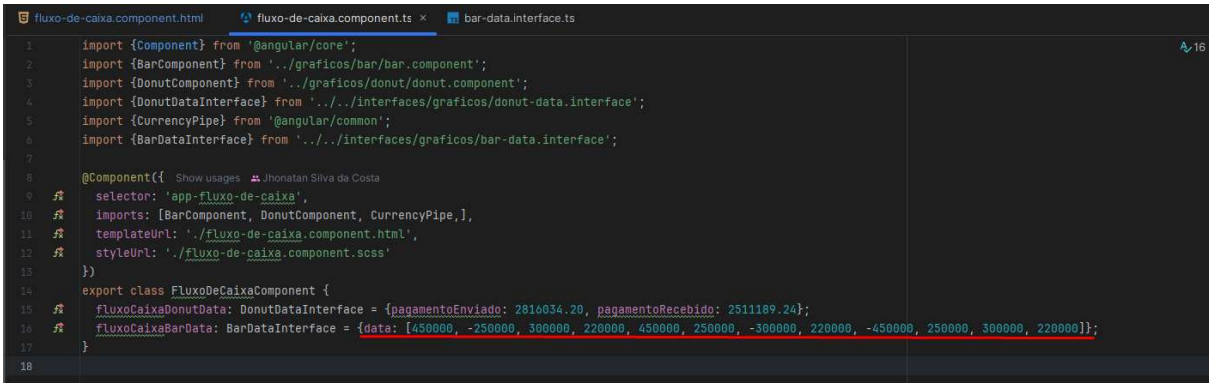
Dados do Fluxo de Caixa para gráfico donut: Precisa ser ter gerenciamento das chamadas http que alimentaram o gráfico de donut. Tem uma interface própria para esse tipo de dados.

```
export interface DonutDataInterface {  Show usages  Jhonatan Silva da Costa
  pagamentoRecebido: number;
  pagamentoEnviado: number;
}
```



Como mostrado na imagem, os dados irão ser montados e usados no gráfico e nos textos.

Dados do Fluxo de Caixa para gráfico bar: Precisa ter gerenciamento http das chamadas http que alimentam o gráfico de bar. Tem uma interface própria para o tipo de dado necessário para montagem.
Importante: Na POC foi criado o gráfico de bar pensando em dados durante 1 ano, inicialmente os dados são fictícios, mas ao iniciar o desenvolvimento, pode ser alinhado o uso de filtro de anos ou pegar de uma ano específico.
Pensando em ano, o gráfico espera um array de number com 12 valores, ou seja, de janeiro a dezembro.



Gráficos

Gráfico de Donut

Gráfico de donut foi criado para ser dinâmico e pode ser usado em outros lugares do sistema.

Requisitos

Dados do gráfico: Foi criado o donut pensando em fluxo de caixa, então espera dois valores para ser montado, o pagamento enviado e o recebido. Pode ser adaptado para outros dados também.

```
fluxo-de-caixa.component.html   fluxo-de-caixa.component.ts ×   bar-data.interface.ts
1   import {Component} from '@angular/core';
2   import {BarComponent} from '../graficos/bar/bar.component';
3   import {DonutComponent} from '../graficos/donut/donut.component';
4   import {DonutDataInterface} from '../../interfaces/graficos/donut-data.interface';
5   import {CurrencyPipe} from '@angular/common';
6   import {BarDataInterface} from '../../interfaces/graficos/bar-data.interface';
7
8   @Component({  Show usages  Jhonatan Silva da Costa
9     selector: 'app-fluxo-de-caixa',
10    imports: [BarComponent, DonutComponent, CurrencyPipe,],
11    templateUrl: './fluxo-de-caixa.component.html',
12    styleUrl: './fluxo-de-caixa.component.scss'
13  })
14  export class FluxoDeCaixaComponent {
15    fluxoCaixaDonutData: DonutDataInterface = {pagamentoEnviado: 2816034.20, pagamentoRecebido: 2511189.24};
16    fluxoCaixaBarData: BarDataInterface = {data: [450000, -250000, 300000, 220000, 450000, 250000, -300000, 220000, -450000, 250000, 300000, 220000]};
17  }
18
```

Gráfico de Barra

Gráfico de donut foi criado para ser dinâmico e pode ser usado em outros lugares do sistema.

Requisitos

Dados do Gráfico: Foi criado para ser dinâmico, representando os valores durante um período de um ano, podendo ser estudado a opção de filtro de ano ou pegar um ano específico.
Espera receber um array de números com 12 valores, ou seja, um para cada mês.

```ts
import {Component} from '@angular/core';
import {BarComponent} from '../graficos/bar/bar.component';
import {DonutComponent} from '../graficos/donut/donut.component';
import {DonutDataInterface} from '../../interfaces/graficos/donut-data.interface';
import {CurrencyPipe} from '@angular/common';
import {BarDataInterface} from '../../interfaces/graficos/bar-data.interface';

@Component({
  selector: 'app-fluxo-de-caixa',
  imports: [BarComponent, DonutComponent, CurrencyPipe,],
  templateUrl: './fluxo-de-caixa.component.html',
  styleUrl: './fluxo-de-caixa.component.scss'
})
export class FluxoDeCaixaComponent {
  fluxoCaixaDonutData: DonutDataInterface = {pagamentoEnviado: 2816034.20, pagamentoRecebido: 2511189.24};
  fluxoCaixaBarData: BarDataInterface = {data: [450000, -250000, 300000, 220000, 450000, 250000, -300000, 220000, -450000, 250000, 300000, 220000]};
}
```

Saldos

Foi criado dois cenários para saldos, o componente completo de saldo, onde vai ter os dados de aporte, saque, saldo atual, inicial e outros. E o card de saldo, que é um componente individual para mostrar um valor.

Card de saldo

Card dinâmico, pensado em mostrar um card com um saldo, um título, um subtítulo e algumas lógicas. Tem uma interface para esses dados.

```ts
import {SaldoCardMenuInterface} from './saldo-card-menu.interface';

export interface SaldoCardValueInterface {
  title: string;
  subTitle: string;
  menuActions: SaldoCardMenuInterface[];
  cardValue: number;
}
```

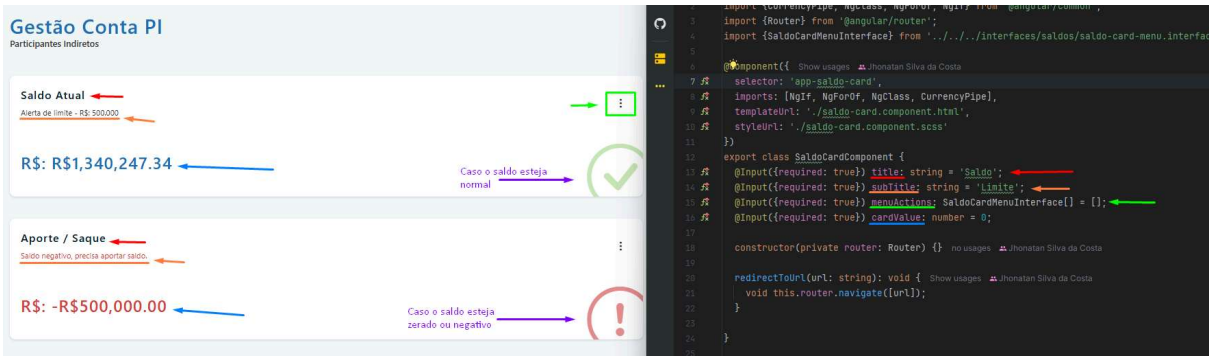
Requisitos

Título: Título principal superior,
Subtítulo: subtítulo que fica abaixo do título principal, existe uma lógica que, caso o saldo seja negativo ou zero, o subtítulo ficará com a cor vermelha, e mostrará um ícone de atenção ao invés do padrão.
Ação do menu: cada card tem uma opção de ação, um menu que pode ter ações, como, detalhes, editar algo e afins, isso é definido no componente pai, bem como a ação a ser tomada depois do clique. Tem uma interface para isso.

```typescript
export interface SaldoCardMenuInterface {  Show usages  Jhonatan Silva da Costa
  redirectLink: string,
  linkText: string,
}
```

```typescript
cardOne: SaldoCardValueInterface = {
  title: 'Saldo Atual',
  subTitle: 'Alerta de limite - R$: 500.000',
  menuActions: [
    {
      linkText: 'Editar Alerta de Limite',
      redirectLink: '#',
    },
    {
      linkText: 'Detalhes',
      redirectLink: '#',
    }
  ],
  cardValue: 1340247.34
};
```

Valor do Card: valor em número que será mostrado no cartão, o mesmo tem formatação de valor em BRL.



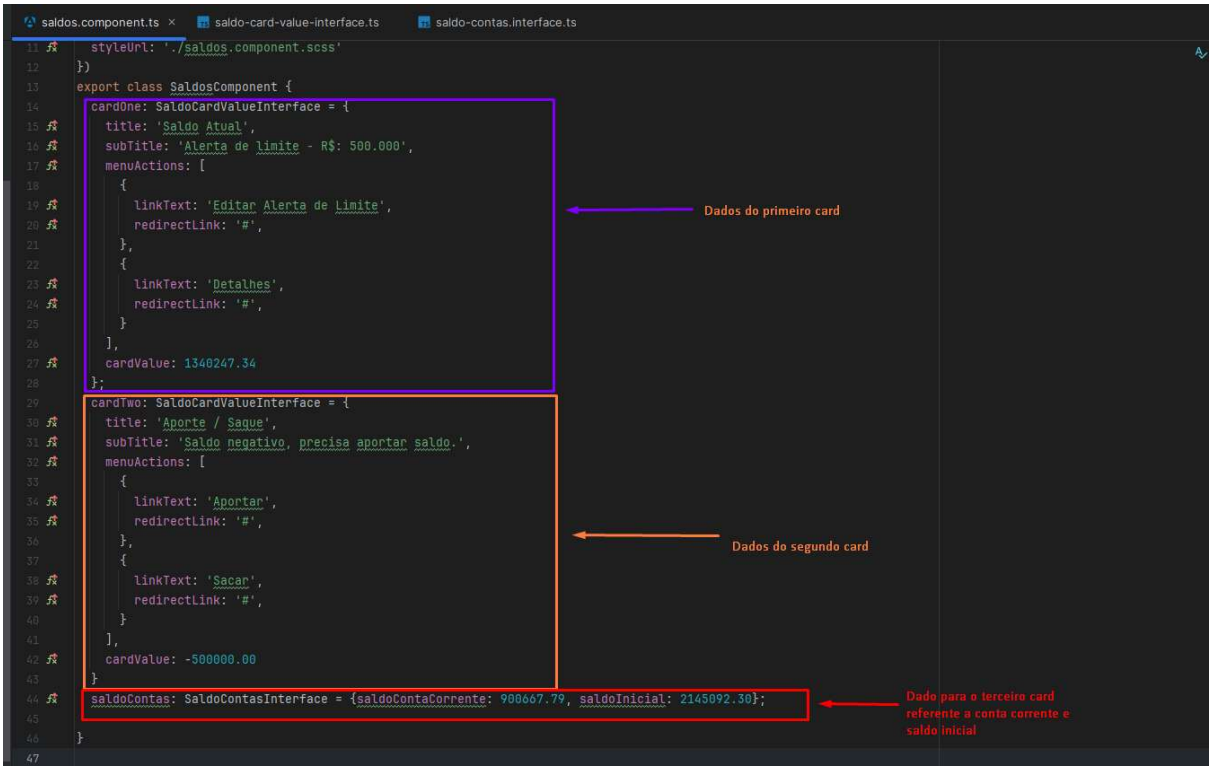
Saldo completo

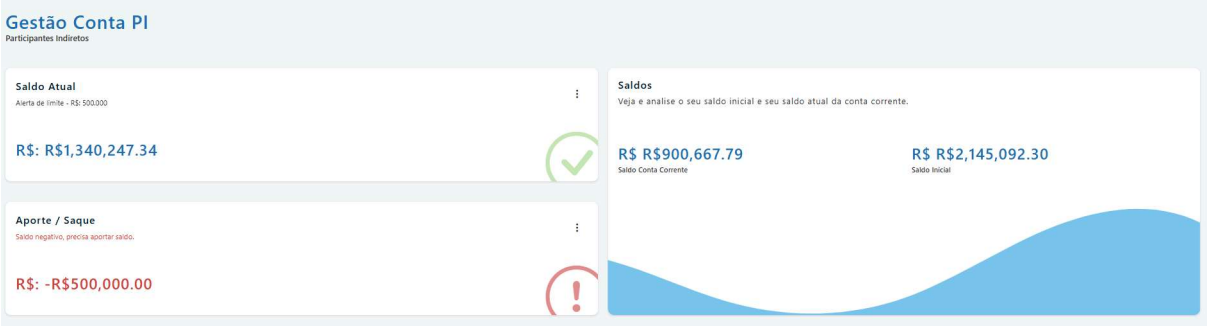
Componente completo usado na dashboard, pode ser usado em outros lugares, mas deve se passar os dados corretamente. Esse componente é responsável por mostrar dois cards de saldo citado acima e mais um de saldo de conta e saldo inicial.

Requisitos

Dados do primeiro card de saldo: Como orientado no item acima de card do saldo, precisa passar os dados que satisfazem o card.
Dados do segundo card de saldo: Como orientado no item acima de card do saldo, precisa passar os dados que satisfazem o card.
Dados das contas: Dado extra para o card de saldos da conta corrente e saldo inicial. Tem uma interface para isso.

```ts
export interface SaldoContasInterface {  Show usages  Jhonatan Silva da Costa
  saldoContaCorrente: number;
  saldoInicial: number;
}
```




Gestão Conta PI
Participantes Indiretos
Saldo Atual
Alerta de limite - R$: 500.000
R$: R$1,340,247.34
Aporte / Saque
Saldo negativo, precisa aportar saldo.
R$: -R$500,000.00
Saldos
Veja e analise o seu saldo inicial e seu saldo atual da conta corrente.
R$ R$900,667.79
Saldo Conta Corrente
R$ R$2,145,092.30
Saldo Inicial