

MAX LENNON MÜLLER
CONTROLE FINANCEIRO PESSOAL

Sumário

1	Objetivos.....	2
2	Funcionamento	2
3	Checklist	6
3.1	✓ Criar o repositório no GitHub com a estrutura do Gitflow, ou seja, branches main e develop.....	6
3.2	✓ Usar componentes de algum framework CSS (Bootstrap, Materialize ou outro).	7
3.3	✓ Apresentar as telas com layout responsivo usando ou não algum framework CSS.....	8
3.4	✓ Construir páginas web com o conceito de componentes.	9
3.5	✓ Criar o layout da aplicação com componentes, ou seja, o cabeçalho e rodapé precisam ser componentes.....	10
3.6	✓ Usar pelo menos dois tipos de data-binding (Interpolation, Property Binding, Event Binding e Two Way Data Binding).....	10
3.7	✓ Passar dados via hierarquia de componentes, ou seja, usando @Input ou @Output.....	11
3.8	✓ Mapear componentes às rotas no módulo de rotas.....	12
3.9	✓ Criar navegação entre páginas por meio de rotas.....	13
3.10	✓ Passar dados entre componentes que representam diferentes telas via parâmetros de rotas.	14
3.11	✓ Validar campos do formulário com REGEX e apresentar os erros ..	15
3.12	✓ Desabilitar o botão de submit enquanto o formulário está inválido..	16
3.13	✓ Fazer requisições a API com tratamento da resposta com Promises ou Observables.....	16
3.14	✓ Cadastrar uma entidade no JSON Server.	17
3.15	✓ Apresentar uma lista de dados com a diretiva estrutural ngFor.....	18
3.16	✓ Usar a diretiva ngIf	18
3.17	✓ Formatar a apresentação de dados com Pipes.....	19
3.18	✓ Build e deploy da aplicação.....	19

1 OBJETIVOS

O objetivo deste relatório é apresentar o projeto final da disciplina de Frameworks Web, e para isso foi desenvolvido uma aplicação web usando o framework Angular.


O objetivo do programa é gerenciar os gastos financeiros do usuário através do cadastro de suas receitas e despesas mensais e acompanhamento do saldo do mês corrente.

2 FUNCIONAMENTO

A aplicação está funcionando no ambiente de produção através do link: <https://maxlmuller.github.io/controle-financeiro-pessoal> e necessita da execução da API Fake (JSON Server) localmente para seu total funcionamento.

Na página inicial é mostrado o saldo referente a soma de todas as receitas menos a soma de todas as despesas cadastradas.

Nos formulários de Cadastro Rápido abaixo é possível passar um valor para ser preenchido automaticamente na página de cadastro de receitas ou despesas.

Saldo Receitas Despesas

Saldo do Mês:

R\$2,250.00

Cadastro Rápido de Receitas

Digite um valor para a receita

ENVIAR ➤

Cadastro Rápido de Despesas

Digite um valor para a despesa


ENVIAR ➤

Controle Financeiro Pessoal

Links:
Saldo
Receitas
Despesas

© 2023 - Todos os direitos reservados

Caso o saldo apresentado for negativo, o campo é estilizado em vermelho para indicar atenção ao controle financeiro.



Saldo Receitas Despesas

Saldo do Mês:

-R\$750.00

Na página Receitas é possível realizar o cadastro de uma nova Receita ou consultar na tabela as descrições e valores das Receitas já cadastradas.



Saldo Receitas Despesas

Receitas

Cadastro de Receitas

Descrição:

Valor (R\$):

ENVIAR >

Lista de Receitas

Total de Receitas: R\$5,000.00

Descrição	Valor
Salário	R\$3,000.00
Venda Notebook	R\$2,000.00




Controle Financeiro Pessoal

Links:
Saldo
Receitas
Despesas

© 2023 - Todos os direitos reservados

Na página Despesas é possível realizar o cadastro de uma nova Despesa ou consultar na tabela as descrições e valores das Despesas já cadastradas.



SaldoReceitasDespesas

Despesas

Cadastro de Despesas

Descrição:


Valor (R\$):

ENVIAR >

Lista de Despesas

Total de Despesas: R\$2,750.00

Descrição	Valor
Serviços de Streaming	R\$39.90
Supermercado	R\$560.00
Cartão de Crédito	R\$2,150.10



Controle Financeiro Pessoal

Links:

SaldoReceitasDespesas

© 2023 - Todos os direitos reservados

Os formulários contam com uma validação dos dados e só ativam o botão de Enviar caso os campos passem por essa validação.

Despesas

Cadastro de Despesas

Descrição:

Despesas Médicas

Valor (R\$):

3000

ENVIAR >

Após o cadastro da Receita ou Despesa, uma mensagem indica se o cadastro foi realizado com sucesso ou se apresentou um erro.

Despesas

Cadastro de Despesas

Despesa cadastrada com sucesso!

Descrição:

Valor (R\$):

ENVIAR ➤

Despesas

Cadastro de Despesas

Erro ao cadastrar despesa!

Descrição:

Valor (R\$):

ENVIAR ➤

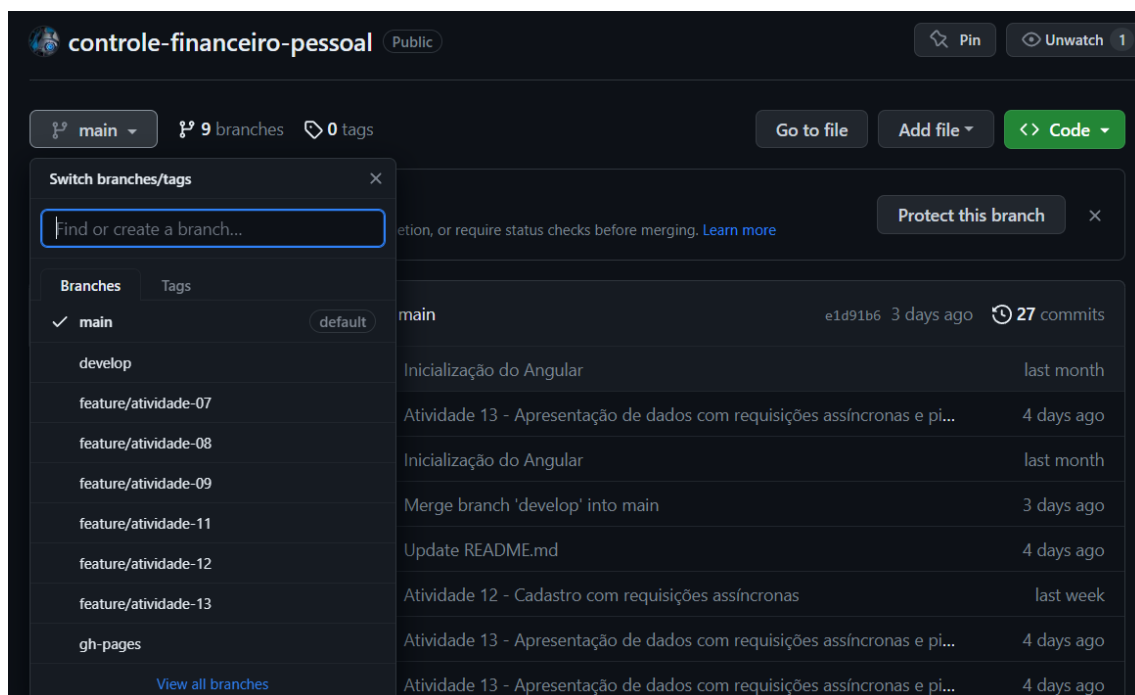
3 CHECKLIST

A seguir será apresentado a implementação de cada item do checklist, também disponível no repositório da aplicação:

<https://github.com/maxlmuller/controle-financeiro-pessoal>

3.1 ☒ Criar o repositório no GitHub com a estrutura do Gitflow, ou seja, branches main e develop.

No repositório da aplicação estão disponíveis as branches main, develop e aos features referentes as atividades da disciplina.



3.2 Usar componentes de algum framework CSS (Bootstrap, Materialize ou outro).

Na aplicação foram utilizados dois componentes do framework CSS Materialize, sendo eles o Navbar (<https://materializecss.com/navbar.html>) e o Footer (<https://materializecss.com/footer.html>) para os componentes menu e footer abaixo, respectivamente.

menu.component.html

```
<div class="navbar-fixed" style="z-index: 1001 !important">
<nav>
  <div class="nav-wrapper light-green-background">
    <div class="container">
      <a href="#" class="brand-logo"></a>
      <a href="#" data-target="mobile-demo" class="sidenav-trigger"><i
class="material-icons">menu</i></a>
      <ul class="right hide-on-med-and-down">
        <li><a routerLink="/saldo" class=dark-green-text>Saldo</a></li>
        <li><a routerLink="/receitas" class=dark-green-
text>Receitas</a></li>
        <li><a routerLink="/despesas" class=dark-green-
text>Despesas</a></li>
      </ul>
    </div>
  </div>
</nav>

<ul class="sidenav" id="mobile-demo">
  <div class="row">
    <div class="col s8 offset-s2">
      <div class="center">
        
      </div>
    </div>
  </div>
  <li><a routerLink="/saldo" class=dark-green-text>Saldo</a></li>
  <li><a routerLink="/receitas" class=dark-green-text>Receitas</a></li>
  <li><a routerLink="/despesas" class=dark-green-text>Despesas</a></li>
</ul>
</div>
```

footer.component.html

```
<footer class="page-footer light-green-background">
  <div class="container">
    <div class="row">
      <div class="col 16 s12">
        
        <h5 class="dark-green-text">Controle Financeiro Pessoal</h5>
      </div>
      <div class="col 14 offset-12 s12">
        <h4 class="dark-green-text">Links:</h4>
        <ul>
          <li><a routerLink="/saldo" class=dark-green-text>Saldo</a></li>
          <li><a routerLink="/receitas" class=dark-green-
text>Receitas</a></li>
          <li><a routerLink="/despesas" class=dark-green-
text>Despesas</a></li>
        </ul>
      </div>
    </div>
  </div>
  <div class="footer-copyright">
    <div class="container dark-green-text" style="font-size: small">
      © {{this.year}} - Todos os direitos reservados
    </div>
  </div>
</footer>
```

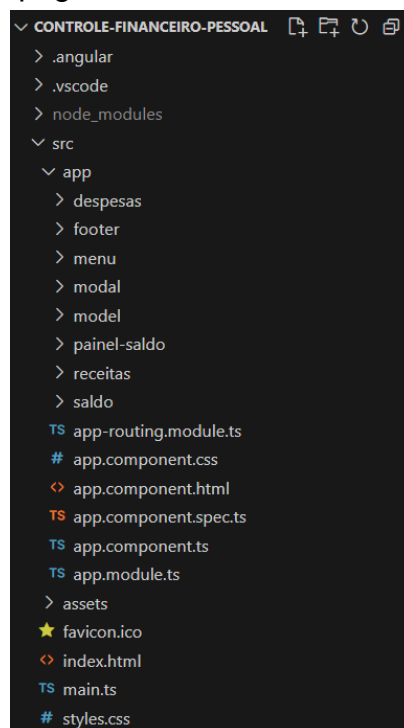
3.3 Apresentar as telas com layout responsivo usando ou não algum framework CSS.

Na aplicação foi utilizado o framework Materialize CSS para a implantação de um layout responsivo que por exemplo, mostra no menu o ícone de hambúrguer quando acessado por um dispositivo móvel e quando clicado mostra a navegação das páginas em um menu lateral.



3.4 Construir páginas web com o conceito de componentes.

A aplicação foi criada com o conceito de componentes para cada página utilizada. Por exemplo, as páginas saldo, receitas e despesas são componentes.



3.5 Criar o layout da aplicação com componentes, ou seja, o cabeçalho e rodapé precisam ser componentes.

Assim como mostrado na imagem do item anterior, o cabeçalho e o rodapé são, respectivamente, os componentes menu e footer da aplicação.

3.6 Usar pelo menos dois tipos de data-binding (Interpolation, Property Binding, Event Binding e Two Way Data Binding).

No trecho do código abaixo foi utilizado o Property Binding para associar a propriedade *hideBalance* do componente saldo para a propriedade *hidden* do elemento HTML.

Trecho do código do `saldo.component.html`

```
<!-- trecho do código omitido -->
<div id="painel-saldo" class="container
contact" [hidden]="this.hideBalance">
  <h5>Saldo do Mês:</h5>
  <app-painel-saldo [saldo]=saldo
(investmentsEvent)="this.onInvestmentsEvent($event)"></app-painel-saldo>
<!-- trecho do código omitido -->
</div>
```

No código abaixo foi utilizado o Event Binding para associar o evento de clique no botão com a chamada de um método *onClose()* do componente.

`modal.component.html`

```
<div class="modal" #modal1>
  <div class="modal-content">
    <h4>{{ content?.title }}</h4>
    <p>{{ content?.text }}</p>
  </div>
  <div class="modal-footer">
    <button class="modal-close btn-flat" (click)="onClose()">OK</button>
  </div>
</div>
```

3.7 Passar dados via hierarquia de componentes, ou seja, usando @Input ou @Output.

No componente-filho *painel-saldo* é usado o @Input para receber o valor do saldo do componente-pai *saldo*.

Também é usado o @Output para enviar para o componente-pai quando que deve ser disparado o evento de investimento como mostrado no código abaixo:

painel-saldo.component.html

```
import { Component, EventEmitter, Input, OnChanges, Output } from
'@angular/core';

@Component({
  selector: 'app-painel-saldo',
  templateUrl: './painel-saldo.component.html',
  styleUrls: ['./painel-saldo.component.css']
})
export class PaineSaldoComponent implements OnChanges{
  @Input() saldo: number = 0;
  @Output() investmentsEvent = new EventEmitter<boolean>();
  backgroundColor = 'light-green-background';

  constructor() {}

  ngOnChanges(): void {
    if (this.saldo < 0){
      this.backgroundColor = 'light-red-background';}
    else{
      this.backgroundColor = 'light-green-background';
    }
    if (this.saldo > 999)
      setTimeout(() => {this.investmentsEvent.emit(true);}, 4000);
  }
}
```

3.8 Mapear componentes às rotas no módulo de rotas.

O mapeamento das rotas aos componentes foi feito no código do `app-routing.module.ts`, como mostrado no código abaixo.

A rota `/saldo` mapeia para o componente `SaldoComponent`;

A rota `/receitas` mapeia para o componente `ReceitasComponent`;

A rota `/despesas` mapeia para o componente `DespesasComponent`;

A rota `/receitas/:valor` passa o parâmetro *valor* e o mapeia para o componente `ReceitasComponent`;

A rota `/despesas/:valor` passa o parâmetro *valor* e o mapeia para o componente `DespesasComponent`;

`app-routing.module.ts`

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { SaldoComponent } from '../saldo/saldo.component';
import { ReceitasComponent } from '../receitas/receitas.component';
import { DespesasComponent } from '../despesas/despesas.component';

const routes: Routes = [
  { path: '', redirectTo: '/saldo', pathMatch: 'full' },
  { path: 'saldo', component: SaldoComponent },
  { path: 'receitas', component: ReceitasComponent },
  { path: 'despesas', component: DespesasComponent },
  { path: 'receitas/:valor', component: ReceitasComponent },
  { path: 'despesas/:valor', component: DespesasComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

3.9 Criar navegação entre páginas por meio de rotas.

Toda a navegação entre as páginas foi feita por meio de rotas, como mostram os trechos dos códigos dos componentes menu e footer abaixo:

Trecho do código do menu.component.html

```
<nav>
  <div class="nav-wrapper light-green-background">
    <div class="container">
      <a href="#" class="brand-logo"></a>
      <a href="#" data-target="mobile-demo" class="sidenav-trigger"><i
class="material-icons">menu</i></a>
      <ul class="right hide-on-med-and-down">
        <li><a routerLink="/saldo" class=dark-green-text>Saldo</a></li>
        <li><a routerLink="/receitas" class=dark-green-
text>Receitas</a></li>
        <li><a routerLink="/despesas" class=dark-green-
text>Despesas</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Trecho do código do footer.component.html

```
<div class="col l4 offset-l2 s12">
  <h4 class="dark-green-text">Links:</h4>
  <ul>
    <li><a routerLink="/saldo" class=dark-green-text>Saldo</a></li>
    <li><a routerLink="/receitas" class=dark-green-
text>Receitas</a></li>
    <li><a routerLink="/despesas" class=dark-green-
text>Despesas</a></li>
  </ul>
</div>
```

3.10 Passar dados entre componentes que representam diferentes telas via parâmetros de rotas.

Na página principal o formulário *cadastroRapido* permite passar um valor para a tela correspondente através de parâmetros de rotas.

O valor enviado é preenchido automaticamente no campo valor do Cadastro de Receitas ou Despesas como mostrado no exemplo abaixo.

Trecho do código do `saldo.component.ts`

```
export class SaldoComponent implements OnInit {

  <!-- trecho do código omitido -->
  valorReceita!: number;
  valorDespesa!: number;

  constructor(private router: Router, private http: HttpClient) {}

  <!-- trecho do código omitido -->

  enviarReceita() {
    this.router.navigate(['/receitas', this.valorReceita]);
  }

  enviarDespesa() {
    this.router.navigate(['/despesas', this.valorDespesa]);
  }
}
```

Trecho do código do `receitas.component.ts`

```
export class ReceitasComponent {

  <!-- trecho do código omitido -->

  constructor(
    private receitasService: ReceitasService,
    private route: ActivatedRoute
  ) {}

  ngOnInit() {
    this.atualizarListaReceitas();
    this.route.params.subscribe(params => {
      this.valor = +params['valor'];
    });
  }

  <!-- trecho do código omitido -->
```

3.11 Validar campos do formulário com REGEX e apresentar os erros

Nos formulários de cadastro de receitas e despesas foram utilizados REGEX para a validação dos campos Descrição e Valor.

Para o campo Descrição o padrão `^[a-zA-Z0-9\sÀ-ÿ]{2,50}$` permite letras maiúsculas e minúsculas, números, caracteres com acento e espaços sendo válido quando estiver entre 2 e 50 caracteres.

Para o campo Valor o padrão `^[0-9]+(?:\.[0-9]{1,2})?$` permite números inteiros ou valores com no máximo 2 casas decimais para representação dos centavos separados por um ponto.

Caso os campos, após serem clicados, contiverem algum valor que seja considerado inválido um erro é apresentado informando o seu motivo como mostrado no trecho do código abaixo.

Trecho do código do `receitas.component.html`

```
<form (ngSubmit)="adicionarReceita()" #receitaForm="ngForm">
  <div>
    <label for="descricao">Descrição:</label>
    <input
      type="text"
      id="descricao"
      name="descricao"
      autocomplete="off"
      [(ngModel)]="descricao"
      pattern="^[a-zA-Z0-9\sÀ-ÿ]{2,50}$"
      required>
    </div>
    <div class="red-text" *ngIf="receitaForm.controls['descricao'].invalid
    && receitaForm.controls['descricao'].touched">
      Erro: A descrição é obrigatória e deve conter entre 2 e 50 caracteres.
    </div>
    <div>
      <label for="valor">Valor (R$):</label>
      <input
        type="number"
        id="valor"
        name="valor"
        autocomplete="off"
        [(ngModel)]="valor"
        pattern="^[0-9]+(?:\.[0-9]{1,2})?$"
        required>
      </div>
      <div class="red-text" *ngIf="receitaForm.controls['valor'].invalid &&
      receitaForm.controls['valor'].touched">
        Erro: O valor é obrigatório e deve conter no máximo 2 casas decimais.
      </div>
```

3.12 Desabilitar o botão de submit enquanto o formulário está inválido

Da mesma maneira, no botão de submit, caso o status do formulário não for válido, a propriedade *disabled* do botão é atribuída como *true*, desabilitando-o como mostrado no trecho do código abaixo.

Trecho do código do receitas.component.html

```
<button class="btn waves-effect waves-light" type="submit" name="action"
[disabled]="!receitaForm.valid">Enviar
  <i class="material-icons right">send</i>
</button>
</form>
```

3.13 Fazer requisições a API com tratamento da resposta com Promises ou Observables.

Quando uma receita é adicionada o método *adicionarReceita* do *receitas.service.ts* é chamado e nele é feita uma requisição a API através do trecho do código abaixo utilizando Observables.

receitas.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Movimentacao } from '../model/movimentacao';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class ReceitasService {

  constructor(private http: HttpClient) {}

  atualizarListaReceitas(): Observable<Movimentacao[]> {
    return
    this.http.get<Movimentacao[]>('http://localhost:3000/movimentacoes?tipo=receita');
  }

  adicionarReceita(receita: Movimentacao): Observable<void> {
    return this.http.post<void>('http://localhost:3000/movimentacoes',
    receita);
  }
}
```


Após isso, com a resposta do método *adicionarReceita* o trecho do código abaixo trata dessa resposta indicando ao usuário se a requisição a API foi sucedida ou obteve algum erro:

Trecho do código do `receitas.component.ts`

```
this.receitasService.adicionarReceita(receita).subscribe({  
  next: () => {  
    this.erroReceita = false;  
    this.mensagemReceita = 'Receita cadastrada com sucesso!';  
    this.atualizarListaReceitas();  
  },  
  error: () => {  
    this.erroReceita = true;  
    this.mensagemReceita = 'Erro ao cadastrar receita!';  
  }  
});
```

3.14 Cadastrar uma entidade no JSON Server.

Foi criada uma entidade nomeada *Movimentacao* e sempre que uma receita ou despesa é cadastrada através dos métodos *adicionarReceita* ou *adicionarDespesa* as mesmas são cadastradas no JSON Server.

`movimentacao.ts`

```
export class Movimentacao {  
  static readonly TIPO_RECEITA = 'receita';  
  static readonly TIPO_DESPESA = 'despesa';  
  descricao: string;  
  valor: number;  
  
  constructor(descricao: string, valor: number, public tipo: string) {  
    this.descricao = descricao;  
    this.valor = valor;  
    this.tipo = tipo;  
  }  
}
```

3.15 Apresentar uma lista de dados com a diretiva estrutural `ngFor`.

Tanto na página Receitas como Despesas é apresentado uma lista com todas as movimentações daquele tipo, mostrando sua descrição e seu valor, respectivamente.

O trecho de código abaixo mostra como foi implementado essa lista utilizando da diretiva estrutural `ngFor` na página de Receitas.

Trecho do código do `receitas.component.html`

```
<table>
  <tr>
    <th><h5>Lista de Receitas</h5></th>
    <th><h5>Total de Receitas: {{ calcularTotalReceitas() | currency:
"BRL" }}</h5></th>
  </tr>
  <tr>
    <th>Descrição</th>
    <th>Valor</th>
  </tr>
  <tr *ngFor="let receita of receitas">
    <td>{{ receita.descricao }}</td>
    <td>{{ receita.valor | currency: "BRL" }}</td>
  </tr>
</table>
```

3.16 Usar a diretiva `ngIf`

No formulário de cadastro, caso algum campo contiver algum valor inválido é apresentado um erro. A visibilidade desse erro é controlada por uma diretiva `ngIf` como mostrado no trecho do código abaixo.

Trecho do código do `receitas.component.html`

```
<!-- trecho do código omitido -->
<div class="red-text" *ngIf="receitaForm.controls['descricao'].invalid
&& receitaForm.controls['descricao'].touched">
  Erro: A descrição é obrigatória e deve conter entre 2 e 50 caracteres.
</div>
<!-- trecho do código omitido -->
<div class="red-text" *ngIf="receitaForm.controls['valor'].invalid &&
receitaForm.controls['valor'].touched">
  Erro: Utilize apenas números e ponto para separar os centavos.
</div>
<!-- trecho do código omitido -->
```

3.17 Formatar a apresentação de dados com Pipes.

Em todo o projeto, os valores de saldo, receitas e despesas são valores monetários e, portanto, foram apresentados utilizando o Pipe *currency* com a notação "BRL", que é específico para o Real Brasileiro, a moeda oficial do Brasil. Por exemplo, como mostrado no código abaixo.

painel-saldo.component.html

```
<div class="row">
  <div
    class="col s12 m12 card-panel"
    id="div-panel"
    [ngClass]="this.backgroundColor">
    {{ this.saldo | currency: "BRL" }}
  </div>
</div>
```

3.18 Build e deploy da aplicação.

Após o desenvolvimento da aplicação foi realizado o merge do branch *develop* no branch *main* e então realizado o build e deploy utilizando script *npm run deploy*, assim como configurado no *package.json* abaixo.

Trecho do código do package.json

```
{
  "name": "controle-financeiro-pessoal",
  "version": "1.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build --base-href=/controle-financeiro-pessoal/",
    "ghpages": "angular-cli-ghpages --dir=dist/controle-financeiro-pessoal",
    "deploy": "npm run build & npm run ghpages",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "json:server": "json-server --watch db.json",
    "json:server:routes": "json-server --watch db.json --routes routes.json"
  }
}
```

Por fim, a aplicação em sua versão final está disponível no seu ambiente de produção através do link:

<https://maxlmuller.github.io/controle-financeiro-pessoal/>