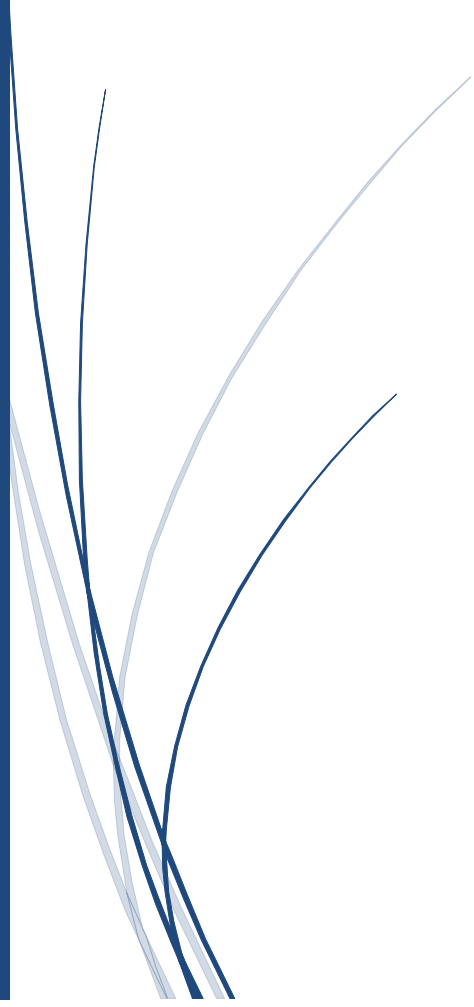
15/8/2019

Treinamento Angular 8

COTI INFORMATICA



WWW.COTIINFORMATICA.COM.BR
AV RIO BRANCO, 185 SALA 307 – CENTRO – RIO DE JANEIRO - RJ

Sumário

Routes

O Angular Router permite a navegação de uma view para a próxima conforme os usuários executam as tarefas da aplicação.

Visão geral

O navegador é um modelo familiar de navegação de aplicativos:

- Digite um URL na barra de endereços e o navegador navega para uma página correspondente.
- Clique nos links na página e o navegador navega para uma nova página.
- Clique nos botões Voltar e Avançar do navegador e o navegador navega para trás e para frente no histórico de páginas que você viu.

O Angular Router("o roteador") empresta esse modelo. Ele pode interpretar uma URL do navegador como uma instrução para navegar para uma visualização gerada pelo cliente. Ele pode passar parâmetros opcionais ao componente de visualização de suporte que o ajudam a decidir qual conteúdo específico apresentar. Você pode vincular o roteador a links em uma página e ele navegará para a visualização do aplicativo apropriada quando o usuário clicar em um link. Você pode navegar imperativamente quando o usuário clica em um botão, seleciona em uma caixa suspensa ou em resposta a algum outro estímulo de qualquer origem. E o roteador registra a atividade no diário de histórico do navegador para que os botões de voltar e avançar também funcionem.

Configuração

Um aplicativo Angular roteado tem uma instância singleton do Routerserviço. Quando o URL do navegador muda, esse roteador procura um correspondente Route do qual possa determinar o componente a ser exibido.

Um roteador não tem rotas até você configurá-lo. O exemplo a seguir cria cinco definições de rotas, configura o roteador através do RouterModule.forRoot()método, e adiciona o resultado ao AppModule's importsmatriz.

```
const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'hero/:id',      component: HeroDetailComponent },
  {
    path: 'heroes',
    component: HeroListComponent,
    data: { title: 'Heroes List' }
  },
  { path: '',
    redirectTo: '/heroes',
    pathMatch: 'full'
  },
  { path: '**', component: PageNotFoundComponent }
];
```

A appRoutesmatriz de rotas descreve como navegar. Passe para o RouterModule.forRoot()método no módulo importspara configurar o roteador.

Cada Routeum mapeia um URL pathpara um componente. Não há barras principais no caminho . O roteador analisa e constrói o URL final para você, permitindo que você use os caminhos relativo e absoluto ao navegar entre os modos de exibição do aplicativo.

A :idsegunda rota é um token para um parâmetro de rota. Em um URL como /hero/42"42" é o valor do idparâmetro. O correspondente HeroDetailComponent usará esse valor para encontrar e apresentar o herói cujo id42 é. Você aprenderá mais sobre os parâmetros de rota mais adiante neste guia.

A datapropriedade na terceira rota é um local para armazenar dados arbitrários associados a essa rota específica. A propriedade de dados é acessível dentro

de cada rota ativada. Use-o para armazenar itens como títulos de páginas, texto de trilhas e outros dados estáticossomente leitura . Você usará o protetor de resolução para recuperar dados dinâmicos posteriormente no guia.

O caminho vazio na quarta rota representa o caminho padrão para o aplicativo, o local a ser percorrido quando o caminho na URL está vazio, como normalmente é no início. Essa rota padrão redireciona para a rota da /heroesURL e, portanto, exibirá a HeroesListComponent.

O **caminho na última rota é um curinga . O roteador selecionará essa rota se a URL solicitada não corresponder a nenhum caminho para as rotas definidas anteriormente na configuração. Isso é útil para exibir uma página "404 - Não encontrado" ou redirecionar para outra rota.

A ordem das rotas na configuração é importante e isso é por design. O roteador usa uma estratégia de ganhos de primeira partida ao combinar rotas, portanto rotas mais específicas devem ser colocadas acima de rotas menos específicas. Na configuração acima, as rotas com um caminho estático são listadas primeiro, seguidas por uma rota de caminho vazia, que corresponde à rota padrão. A rota curinga vem por último porque corresponde a cada URL e deve ser selecionada apenas se nenhuma outra rota for correspondida primeiro.

Se você precisar ver quais eventos estão ocorrendo durante o ciclo de vida da navegação, há a opção enableTracing como parte da configuração padrão do roteador. Isso produz cada evento de roteador que ocorreu durante cada ciclo de vida de navegação no console do navegador. Isso só deve ser usado para fins de depuração . Você define a enableTracing: trueopção no objeto passado como o segundo argumento para o RouterModule.forRoot()método.

Router Link

As RouterLinkdiretivas nas tags de âncora dão ao roteador controle sobre esses elementos. Os caminhos de navegação são fixos, então você pode atribuir uma string à routerLink(uma ligação "one-time").

Se o caminho de navegação fosse mais dinâmico, você poderia ter vinculado a uma expressão de modelo que retornou uma matriz de parâmetros de link de rota (a matriz de parâmetros de link). O roteador resolve esse array em um URL completo.

Rota ativada

O caminho e os parâmetros da rota estão disponíveis através de um serviço de roteador injetado chamado de ActivatedRoute . Tem uma grande quantidade de informações úteis, incluindo:

Propriedade	Descrição
url	Um Observabledos caminhos da rota, representado como uma matriz de strings para cada parte do caminho da rota.
data	Um Observableque contenha o dataobjeto fornecido para a rota. Também contém todos os valores resolvidos do protetor de resolução .
paramMap	Um Observableque contenha um mapa dos parâmetros obrigatórios e opcionais específicos da rota. O mapa suporta a recuperação de valores únicos e múltiplos do mesmo parâmetro.
queryParamMap	Um Observableque contém um mapa dos parâmetros de consulta disponíveis para todas as rotas. O mapa suporta a recuperação de valores únicos e múltiplos do parâmetro de consulta.
fragment	Um Observabledos fragmentos de URL disponíveis para todas as rotas.
outlet	O nome do RouterOutletusado para renderizar a rota. Para uma tomada sem nome, o nome da saída é principal .
routeConfig	A configuração de rota usada para a rota que contém o

	caminho de origem.
parent	O pai da rota ActivatedRoutequando esta rota é uma rota secundária .
firstChild	Contém o primeiro ActivatedRoutena lista das rotas secundárias desta rota.
children	Contém todas as rotas secundárias ativadas na rota atual.

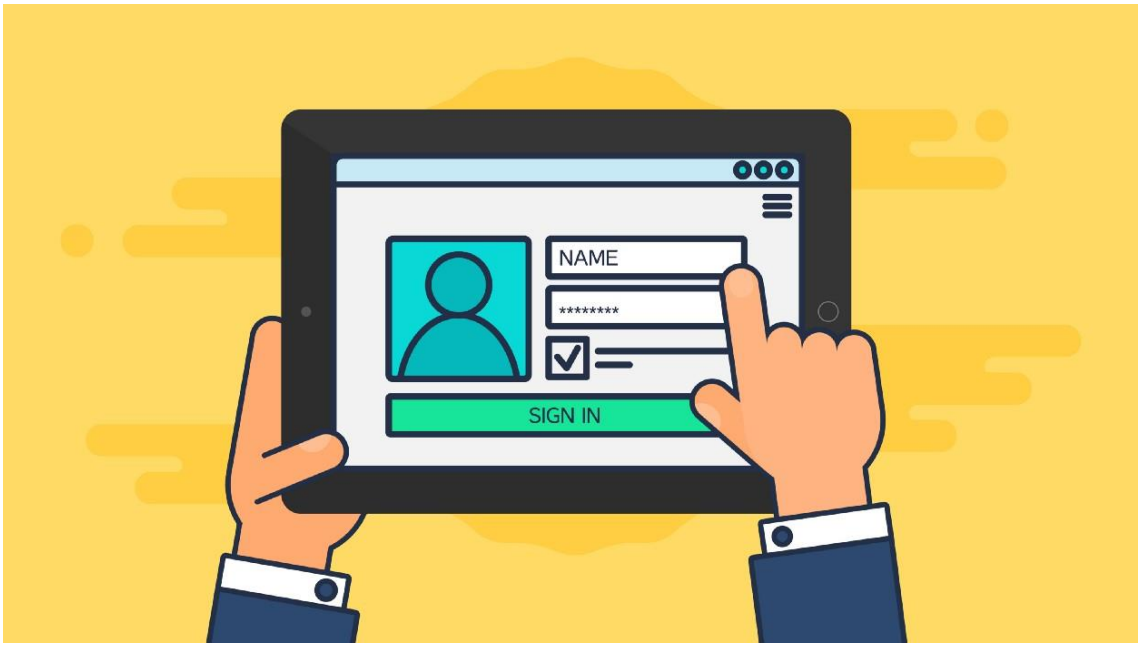
Eventos do roteador

Durante cada navegação, os Routereventos de navegação são emitidos pela Router.eventspropriedade. Esses eventos variam de quando a navegação começa e termina em muitos pontos entre eles. A lista completa de eventos de navegação é exibida na tabela abaixo.

Evento Router	Descrição
NavigationStart	Um evento disparado quando a navegação é iniciada.
RouteConfigLoadStart	Um evento acionado antes que o Router preguiçoso carregue uma configuração de rota.
RouteConfigLoadEnd	Um evento acionado depois que uma rota foi carregada com preguiça.
RoutesRecognized	Um evento acionado quando o roteador analisa o URL e as rotas são reconhecidas.

GuardsCheckStart	Um evento acionado quando o roteador inicia a fase de roteamento de guardas.
ChildActivationStart	Um evento acionado quando o roteador começa a ativar os filhos de uma rota.
ActivationStart	Um evento acionado quando o roteador começa a ativar uma rota.
GuardsCheckEnd	Um evento acionado quando o roteador conclui a fase de roteamento do Guards com êxito.
ResolveStart	Um evento acionado quando o roteador inicia a fase Resolver do roteamento.
ResolveEnd	Um evento acionado quando o roteador conclui a fase Resolver do roteiro com êxito.
ChildActivationEnd	Um evento acionado quando o roteador termina de ativar os filhos de uma rota.
ActivationEnd	Um evento acionado quando o roteador termina de ativar uma rota.
NavigationEnd	Um evento acionado quando a navegação termina com sucesso.
NavigationCancel	Um evento acionado quando a navegação é cancelada. Isto é devido a um Guarda de Rota que retorna falso durante a navegação.
NavigationError	Um evento acionado quando a navegação falha devido a um erro inesperado.
Scroll	Um evento que representa um evento de rolagem.

Route Guards



As proteções de rota da Angular são interfaces que podem informar ao roteador se ele deve ou não permitir a navegação para uma rota solicitada. Eles tomam essa decisão procurando por um valor trueou falseretorno de uma classe que implementa a interface de guarda fornecida.

Existem cinco tipos diferentes de guards e cada um deles é chamado em uma sequência específica. O comportamento do roteador é modificado de forma diferente dependendo de qual guarda é usada. Os guards são:

- CanActivatepara mediar a navegação para uma rota.
- CanActivateChildpara mediar a navegação para uma rota secundária.
- CanDeactivatepara mediar a navegação longe da rota atual.
- Resolvepara executar a recuperação de dados de rota antes da ativação da rota.
- CanLoadpara mediar a navegação para um módulo de recurso carregado de forma assíncrona .

CanActivate : exigindo link de autenticação

Os aplicativos geralmente restringem o acesso a uma área de recursos com base em quem é o usuário. Você pode permitir o acesso apenas a usuários autenticados ou a usuários com uma função específica. Você pode bloquear ou limitar o acesso até que a conta do usuário seja ativada.

O CanActivateguarda é a ferramenta para gerenciar essas regras de negócios de navegação.

Decisões de roteamento com base na validade do token

Se você estiver usando JSON Web Tokens (JWT) para proteger seu aplicativo Angular, uma maneira de decidir se uma rota deve ou não ser acessada é verificar o tempo de expiração do token. É provável que você esteja usando o JWT para permitir que seus usuários acessem recursos protegidos em seu back-end. Se esse for o caso, o token não será útil se expirar, portanto, isso é uma boa indicação de que o usuário deve ser considerado "não autenticado".

Crie um método no seu serviço de autenticação que verifique se o usuário está autenticado ou não. Para fins de autenticação sem estado com o JWT, isso é simplesmente uma questão de saber se o token está expirado.

O serviço injeta AuthServicee Routere tem um único método chamado canActivate. Este método é necessário para implementar corretamente a CanActivateinterface.

O canActivatemétodo retorna uma booleanindicação indicando se a navegação para uma rota deve ou não ser permitida. Se o usuário não for autenticado, ele será redirecionado para algum outro local, neste caso, uma rota chamada /login.

Em alguns casos, ainda há um forte desejo de bloquear completamente as rotas do lado do cliente. Embora não seja possível ter 100% de proteção de nada no lado do cliente, o Angular oferece algumas possibilidades interessantes através do roteamento assíncrono. Vamos dar uma olhada em como podemos usar roteamento assíncrono para nossa vantagem.

CanActivate

INTERFACE

Interface que uma classe pode implementar para ser um guarda decidindo se uma rota pode ser ativada. Se todas as proteções retornarem true, a navegação continuará. Se algum guarda retornar false, a navegação será cancelada. Se algum guarda retornar a UrlTree, a navegação atual será cancelada e uma nova navegação será retrocedida para o UrlTreeretorno da guarda.

Formulários Reativos

Formulários reativos fornecem uma abordagem orientada por modelo para manipular entradas de formulário cujos valores mudam com o tempo. Este guia mostra como criar e atualizar um controle de formulário simples, progredir para usar vários controles em um grupo, validar valores de formulário e implementar formulários mais avançados.

Formulários reativos usam uma abordagem explícita e imutável para gerenciar o estado de um formulário em um determinado ponto no tempo. Cada alteração no estado do formulário retorna um novo estado, que mantém a integridade do modelo entre as alterações. Formas reativas são construídas em torno de fluxos observáveis, onde entradas de formulário e valores são fornecidos como fluxos de valores de entrada, que podem ser acessados de forma síncrona.

Os formulários reativos também fornecem um caminho direto para o teste, pois você tem a garantia de que seus dados são consistentes e previsíveis quando solicitados. Todos os consumidores dos fluxos têm acesso para manipular esses dados com segurança.

As formas reativas diferem das formas controladas por modelos de maneiras distintas. Formulários reativos fornecem mais previsibilidade com acesso síncrono ao modelo de dados, imutabilidade com operadores observáveis e alteração de rastreamento por meio de fluxos observáveis. Se você preferir o acesso direto para modificar dados em seu modelo, os formulários orientados por modelo serão menos explícitos, pois dependem de diretivas incorporadas no modelo, além de dados mutáveis para acompanhar as alterações de forma assíncrona. Veja a Visão geral de formulários para comparações detalhadas entre os dois paradigmas.

A FormControlclasse é o bloco de construção básico ao usar formulários reativos. Para registrar um único controle de formulário, importe a FormControlclasse para seu componente e crie uma nova instância do controle de formulário para salvar como uma propriedade de classe.

Programação Reativa

Na computação , a programação reativa é um paradigma de programação declarativa que se preocupa com fluxos de dados e com a propagação de mudanças. Com este paradigma, é possível expressar facilmente fluxos de dados estáticos (por exemplo, matrizes) ou dinâmicos (por exemplo, emissores de eventos) e também comunicar

que existe uma dependência inferida dentro do modelo de execução associado , o que facilita a propagação automática dos dados alterados. fluxo.

Por exemplo, em um ambiente de programação imperativo, significaria que está sendo atribuído o resultado de no instante em que a expressão é avaliada e, posteriormente, os valores de e pode ser alterado sem efeito sobre o valor de . Por outro lado, na programação reativa , o valor de é atualizado automaticamente sempre que os valores de ou mudar, sem que o programa tenha que reexecutar a declaração para determinar o valor atualmente atribuído

Outro exemplo é uma linguagem de descrição de hardware , como Verilog , em que a programação reativa permite que as mudanças sejam modeladas à medida que se propagam pelos circuitos.

A programação reativa foi proposta como uma forma de simplificar a criação de interfaces de usuário interativas e animação de sistema quase em tempo real.

Por exemplo, em uma arquitetura de controlador de visualização de modelo (MVC), a programação reativa pode facilitar as alterações em um modelo subjacente que são refletidas automaticamente em uma exibição associada

O que é um Observable?

Por definição é uma coleção que funciona de forma unidirecional, ou seja, ele emite notificações sempre que ocorre uma mudança em um de seus itens e a partir disso podemos executar uma ação. Digamos que ele resolve o mesmo problema que a versão anterior do Angular havia resolvido com o $watch, porém sem usar força bruta. Enquanto no $watch verificamos todo nosso escopo por alterações após cada $digest cycle (o que tem um grande custo na performance), com Observable esta verificação não acontece, pois para cada evento é emitida uma notificação para nosso Observable e então tratamos os dados.

Mas quais são as vantagens de usar Observable e não Promises?

A grande vantagem está nos "poderes" que o Observable nos dá com seus operadores, por exemplo: Podemos "cancelar" requests para poupar processamento, ou até mesmo tentar fazer uma nova requisição caso algum problema como perda de conexão aconteça.

O usuário não precisa ver aquela tela de erro.

Operador Subscribe

Um dos operadores mais importantes do Observable é o subscribe, que é o equivalente ao thende uma promise ou ao $watch.

```
import { UserService } from 'user.service';

@Component({
  /* Propriedades do componente*/
})
class UserComponent {
  ...
  constructor(private userService: UserService){}

  private users: User[]; // Lista de usuários

  /* Neste método chamamos a função userService.getUsers, que nos retorna um Observable contendo um array de usuários, então atribuímos ao this.users */
  getUsers() {
    this.userService.getUsers()
                    .subscribe(
                      users => this.users = users,
                      error => /* Tratamos erros aqui :) */);

  }

  ...
}
```

Estamos chamando a função getUsers() criada anteriormente e deixamos um subscribe ali, ou seja, assim que a nossa resposta vier e for transformada em JSON, seremos notificados e o arrayusers receberá a lista de usuários, ou caso seja um erro, tratamos o erro onde deixei um comentário. Simples?
Um cenário ainda melhor seria uma aplicação que use web sockets, pois a cada atualização o subscribe iria atualizar o array users novamente, tudo de forma unidirecional e sem grande consumo de recursos com um $watch da vida.

Operador Retry

Vamos supor que você está tentando fazer o download de um arquivo, mas está em uma conexão com muita oscilação (o que convenhamos que é bem comum no Brasil), é de se esperar que o download falhe. Nestes casos podemos utilizar o operador retry, que é extremamente simples e funciona da seguinte forma: Se a operação falhar, será executada novamente para tentar completá-la (a quantidade de tentativas é especificada como parâmetro do retry).

```
@Injectable()
class downloadFileService {
  ...

  downloadFile(attachment: Attachment)  {
    this.http
        .get(/* URL de download do arquivo */, { responseType: ResponseContentType.Blob })
        .retry(3) // Aqui especificamos que a quantidade de tentativas caso o download falhe é 3.
        .map((response: any) => response.blob())
        /* Agora temos o arquivo e é só mandar para o usuário :) */
        }, (err:any) => /* Tratamos erros aqui :) */);
    }

  ...
}
```

Otimizando processo de busca com Observable

E se em nossa lista de usuários houvesse uma barra de pesquisa onde ao digitar qualquer tecla já fosse disparada a requisição para filtrar os usuários? Parece bem simples. Normalmente, no event keyup do campo colocaríamos uma função que filtraria a lista com base no que foi digitado, mas isso tem um problema: Para cada tecla digitada seria gerada uma nova requisição e toda a lista de usuários seria substituída.

Concordemos que substituir a lista a cada letra digitada não é o que queremos fazer. Em um mundo perfeito, a única request que deve ser considerada é a gerada pelo evento da última letra digitada, isto pode ser resolvido com dois métodos do Observable, o switchMap e o debounceTime.

Operador switchMap

Este operador soluciona um dos problemas, mesmo mais antigas. Por exemplo: Se você pesquisou por Paulo, serão 5 requisições, uma para cada letra, você irá construir os objetos e substituir em seu array 5 vezes, é algo custoso.

O switchMap é um dos operadores que você irá usar e se orgulhar infinitamente. Se você fez 5 requisições, ele irá processar apenas a última delas, que é a que realmente importa para nós, construiremos nosso objeto uma vez e substituiremos o array somente uma vez. Seu uso é simples, assim como todos os operadores que vimos até agora.

```
import { UserService } from 'user.service';

@Component({
  /* Propriedades do componente*/
})
class userComponent {
  ...

  constructor(private userService: UserService){}

  private users: User[]; // Nossa lista de usuários

  /* Variável que recebe o valor da função handleFilterChange */
  private filterString:Subject<string> = new Subject<string>();

  /* Função que recebe o valor digitado e coloca em nosso Subject */
  handleFilterChange(value: string) {
    this.filterString.next(value);
  }

  /* Fazemos um switchMap em nosso Subject filterString e atribuímos o
resultado à nossa lista de usuários */
  ngOnInit() {
    this.users = this.filterString
        .switchMap(value => this.userService.getUsers(value))
        .catch(error => /* Tratamos erros aqui :) */);
  }

  ...
}
```

Temos um pouco mais de código, mas nada muito complexo, a lógica é simples: No evento keyup da barra de pesquisa a função handleFilterChange é chamada e ela adiciona o valor ao Subject (que em tese, é uma versão bidirecional do Observable), após isso, temos no ngOnInit a variável users recebendo o resultado do switchMap, que busca os usuários filtrando com o valor passado. Sendo assim, quando digitarmos “Paulo”, apenas a última requisição será processada e terá seu valor atribuído à variável users, pois foi a última a ser enviada.

Mas ainda assim, podemos diminuir este número para apenas uma requisição, e poupar, além de processamento, consumo de dados!

O que é Promise?

As promessas funcionam com operações assíncronas e elas nos retornam um valor único (ou seja, a promessa é resolvida) ou uma mensagem de erro (ou seja, a promessa é rejeitada).

Outra coisa importante a lembrar sobre as promessas é que um pedido iniciado a partir de uma promessa não é cancelável. (Há algumas bibliotecas de implementação do Promise que nos permitem cancelar promessas, mas a implementação nativa do Promise do JavaScript não permite isso).

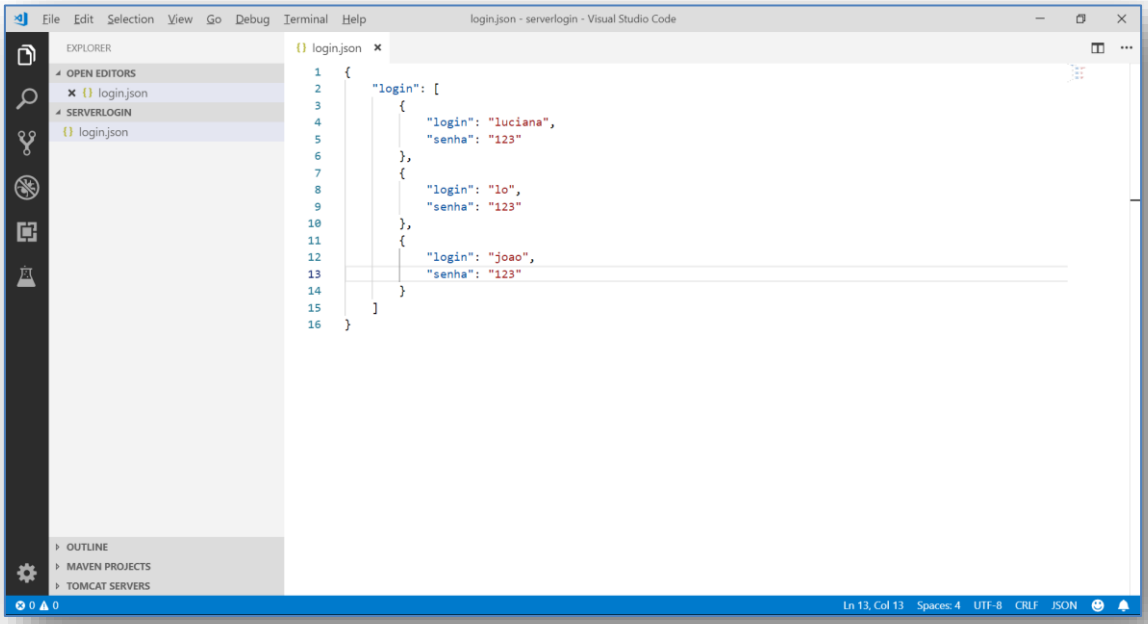
E therelin é uma das principais diferenças entre o uso de uma promessa ou um observável. Como não podemos cancelar uma promessa, uma requisição HTTP que faz uma busca por exemplo na chave seria executada quantas vezes pressionarmos a tecla.

O que é o encadeamento da promessa? O objeto promessa pode ainda ser passado para outro objeto de promessa, que pode então ser passado para outro objeto de promessa, que poderia então decidir resolver ou rejeitar a promessa. Esse processo é chamado de encadeamento .

Projeto Auth Guard

Projeto Servidor Local

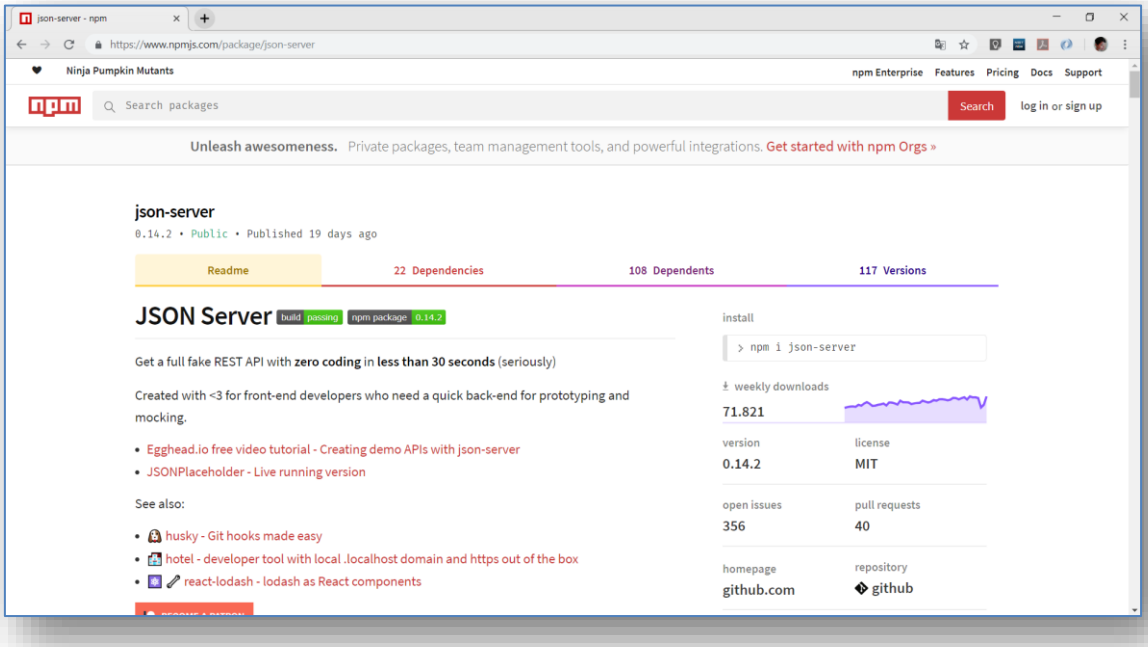
- Criar um diretório para armazenar o projeto
- Entrar no diretório com o Visual Code
- Criar o seguinte arquivo:
    - Login.json



Login.json

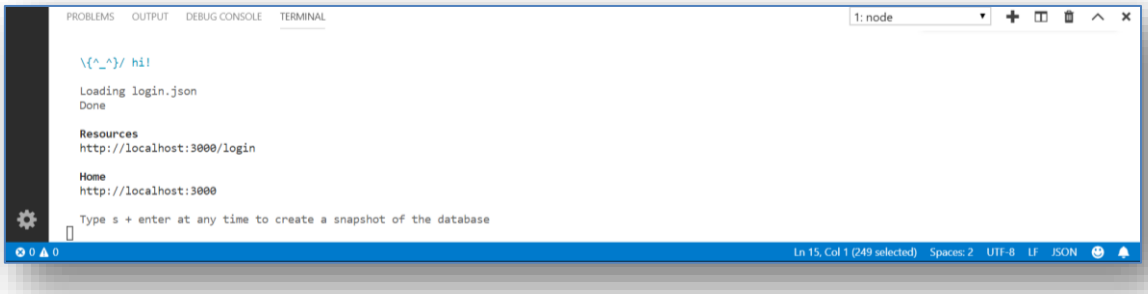
```
{
    "login": [
        {
            "login": "luciana",
            "senha": "123"
        },
        {
            "login": "lo",
            "senha": "123"
        },
        {
            "login": "joao",
            "senha": "123"
        }
    ]
}
```

Para instalar o json-sever:
https://www.npmjs.com/package/json-server

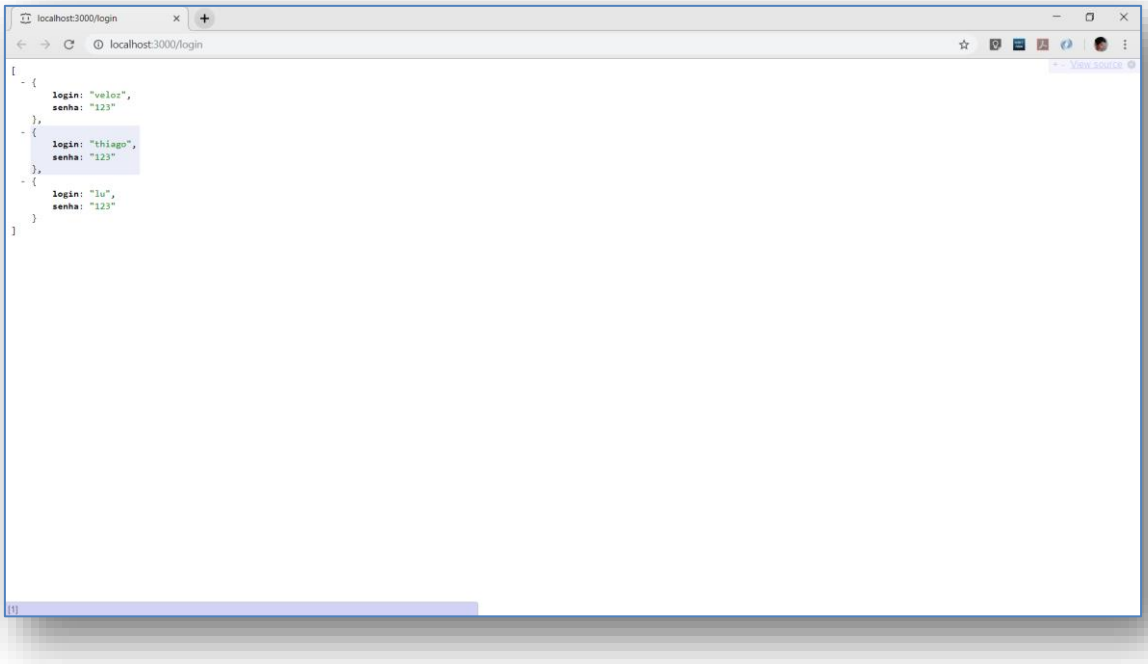


No terminal, digitar:
```
“npm install -g json-server”
```

Para iniciar o servidor:
No terminal, digitar:
“json-server login.json”

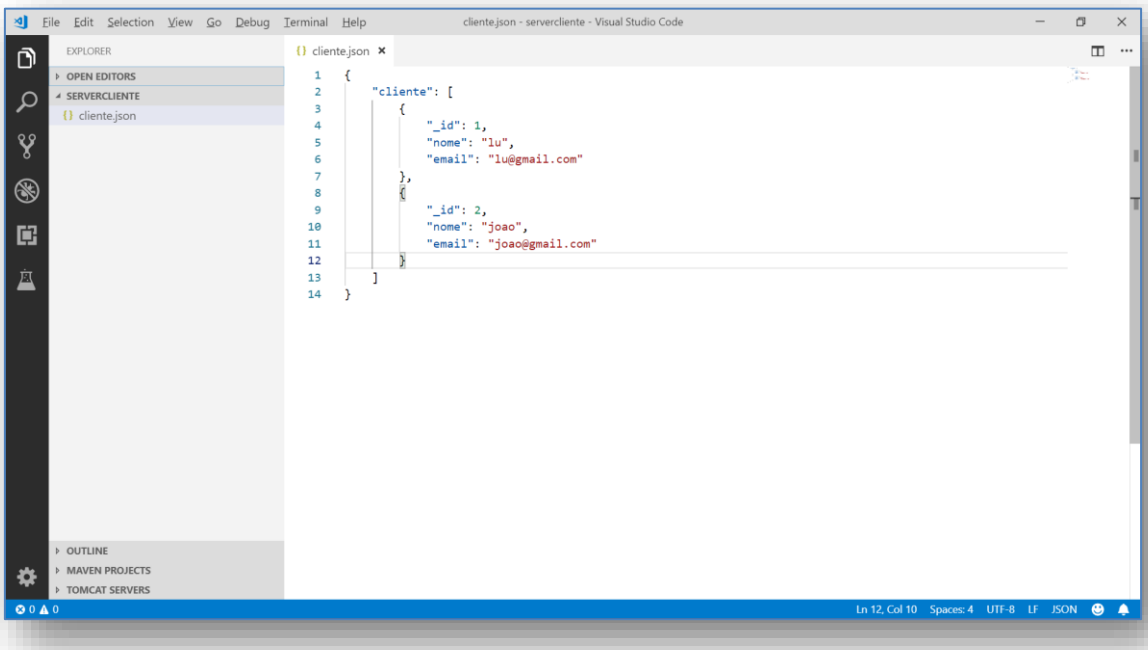


Clicar no link gerado ou digitar na URL:
http://localhost:3000/login



Projeto Servidor Local

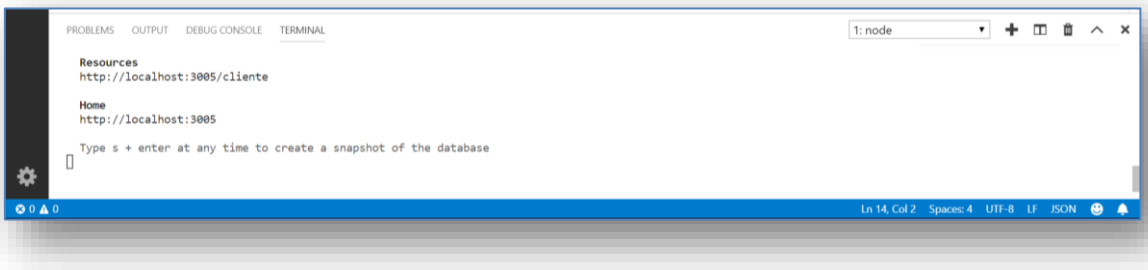
- Criar um diretório para armazenar o projeto
- Entrar no diretório com o Visual Code
- Criar o seguinte arquivo:
    - cliente.json



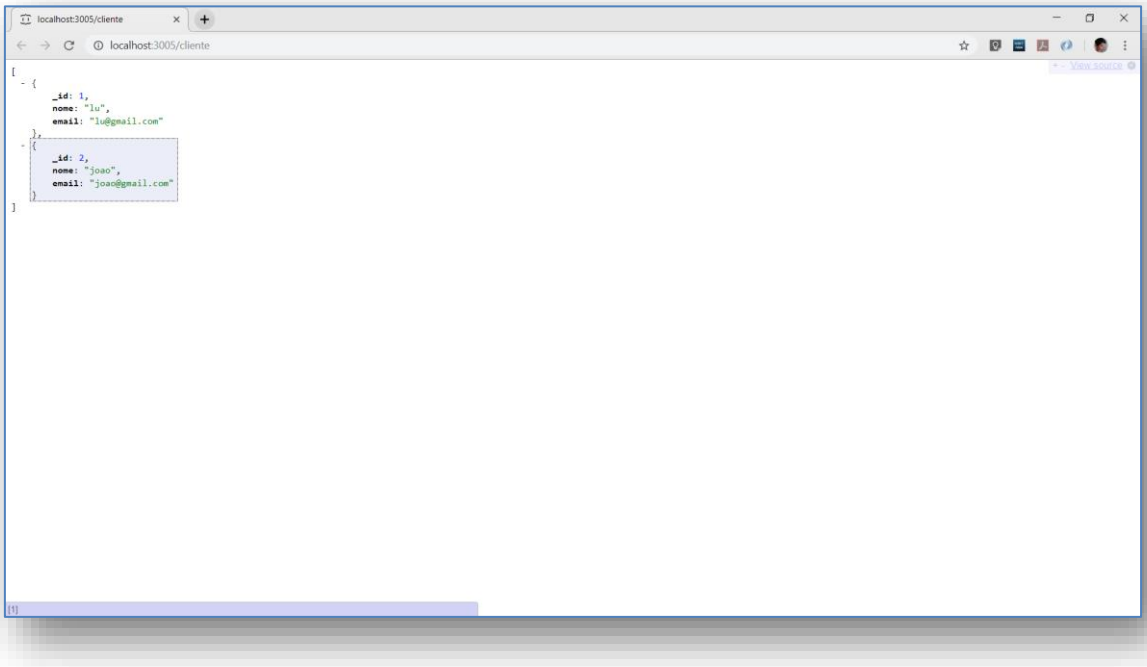
cliente.json

```
{
    "cliente": [
        {
            "_id": 1,
            "nome": "lu",
            "email": "lu@gmail.com"
        },
        {
            "_id": 2,
            "nome": "joao",
            "email": "joao@gmail.com"
        }
    ]
}
```

Para iniciar o servidor:
No terminal, digitar:
“json-server cliente.json –port 3005”

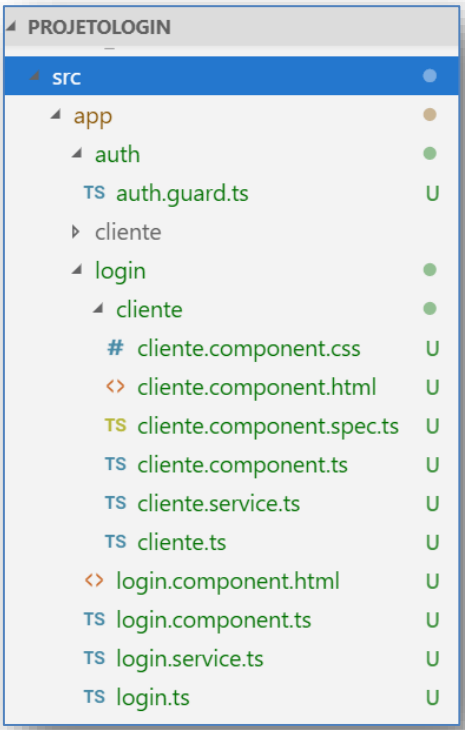


Clicar no link gerado ou digitar na URL:
http://localhost:3005/cliente



Projeto Client

Estrutura do projeto depois de finalizado:



login.ts

```
export class Login{

 login : string;
 senha : string;


  constructor(login ?: string,
    senha ?: string){
   this.login = login;
   this.senha = senha;
    }
}
```

Login.service.ts

```
import { Injectable } from "@angular/core";
import { HttpClient } from '@angular/common/http';
import { Router } from '@angular/router';
import { Observable } from 'rxjs';
import { Login } from "./login";
import { map } from 'rxjs/operators';

@Injectable()
export class LoginService {

    private api: string = "http://localhost:3000/login";

    //http utiliza endPoint, router mudanca de pagina
    constructor(public http: HttpClient,
        private router: Router ) {
    }
    //caminho é com crase
    public logar(authenticacao: any): Observable<Login[]> {
        return
this.http.get<Login[]>(`${this.api}?login=${authenticacao.login}
&senha=${authenticacao.senha}`).pipe(
            map((value) => {
                if (value.length == 0) {
```

```
                    throw new Error('Login ou Senha Invalido');
                }
                this.armazenaStorage(authenticacao);
                return value;
            })
        );
    }

    public armazenaStorage(authenticacao: any) {
        localStorage.setItem('logado',
JSON.stringify(authenticacao));
    }

    public isLogado() {
        try {
            let logado = localStorage.getItem("logado");
            console.log('verifica :', logado);
            if (logado) {
                return true;
            }
            return false;
        } catch (error) {
            console.log('error :', error);
            return false;
        }
    }

    public getLogado() {
        return JSON.parse(localStorage.getItem('logado'))[0];
    }

    public logout() {
        //apaga o storage  e ira retornar para o login
        localStorage.removeItem('logado');
        this.router.navigateByUrl('login');
    }
}
```

Login.component.ts

```
import { LoginService } from './login.service';
import { Component, ElementRef, ViewChild, OnInit }
    from '@angular/core';
import { Router } from '@angular/router';
import { Login } from './login';

declare var document;

@Component({
    selector: 'app-login',
    templateUrl: './login.component.html'
})
export class LoginComponent implements OnInit {

    message: string;
    login: Login;

    @ViewChild('senha')
    inputSenha: ElementRef;

    constructor(private service: LoginService,
        private router: Router
    ) {
        this.login = new Login();
    }

    ngOnInit(): void {
        document.getElementById('resposta').innerHTML
            = '<b>Bem vindo ao Angular6</b>'
    }

    //armazena a palavra logado em localStorage ...
    // e muda de pagina ...
    // o ato de chamar o serviço ou lança um erro
    // ou ele storage(armazena e entra na pagina do cliente)
    //javascript document.getElementByID

    logar() {
        this.service.logar(this.login).subscribe(() => {
```

```
            this.router.navigateByUrl("/cliente");
        }, (error) => {
            this.message = error.message;
            this.login.senha = '';
            this.inputSenha.nativeElement.focus();
        })
    }
    //#
    //ViewChield('senha')
    //inputSenha:ElementRef
}
```

Login.component.html

```
<div>
  <h2>Login </h2>

  <div id="resposta">
  </div>

  <br />
  <hr />
  <input type="text" [(ngModel)]="login.login" name="login"
placeholder="entre com Login" /><br />

  <input type="password" [(ngModel)]="login.senha" #senha
placeholder="entre com Senha" /><br />

  <button (click)="logar();">Login</button>
  <br /><br />
  {{message}}
</div>
```

Auth.guard.ts

```
import { LoginService } from './../login/login.service';
import {
    CanActivate, Router, ActivatedRouteSnapshot,
    RouterStateSnapshot
} from "@angular/router";
import { Injectable } from "@angular/core";

@Injectable({
    providedIn: 'root'
})
export class AuthGuard implements CanActivate {

    constructor(private login: LoginService, private router:
Router) {
    }
    //localStorage existir ele entra no cliente
    //sem logar nao entrar ...

    canActivate(next: ActivatedRouteSnapshot,
        state: RouterStateSnapshot
    ): boolean {
        if (!this.login.isLogado()) {
            this.router.navigateByUrl("login");
            return false; //so entra na pagina com true
        }
        return true;
    }

}
```

App.module.ts

```
import { AuthGuard } from './auth/auth.guard';
import { LoginService } from './login/login.service';
import { LoginComponent } from './login/login.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, Component } from '@angular/core';
import { AppComponent } from './app.component';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { Routes, RouterModule } from '@angular/router';

const ROTAS: Routes = [
  { path: 'login', component: LoginComponent },
  { path: '', pathMatch: 'full', redirectTo: 'login' }
}
]

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    RouterModule.forRoot(ROTAS)
  ],
  providers: [LoginService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Cliente.ts

```
export class Cliente {

    _id: number;
    nome: string;
    email: string;

    constructor(_id?: number,
        nome?: string,
        email?: string) {
        this._id = _id;
        this.nome = nome;
        this.email = email;
    }
}
```

Cliente.service.ts

```
import { HttpClient } from "@angular/common/http";
import { Router } from "@angular/router";
import { Observable } from 'rxjs';
import { Injectable } from "@angular/core";
import { Cliente } from "./cliente";

@Injectable()
export class ClienteService {

    caminho: string = "http://localhost:3005/cliente";

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

    public listar(): Observable<Cliente[]> {
        return this.http.get<Cliente[]>(this.caminho);
    }

    public gravar(cliente: Cliente): Observable<Cliente> {
```

```
        return this.http.post<Cliente>(this.caminho, cliente);
    }

}
```

Cliente.component.ts

```
import { LoginService } from './../login.service';
import { Component, OnInit } from '@angular/core';
import { Cliente } from './cliente';
import { ClienteService } from './cliente.service';

@Component({
  selector: 'app-cliente',
  templateUrl: './cliente.component.html',
  styleUrls: ['./cliente.component.css']
})
export class ClienteComponent implements OnInit {

 clientes: Cliente[] = [];

  constructor(private services: LoginService,
    private servicecliente: ClienteService) {
    this.listarCliente();
  }

  ngOnInit() {
  }

  public logout() {
    this.services.logout();
    //remove localStorage('logado')
    //retorna para a pagina anterior login
  }

  public listarCliente() {
    this.servicecliente.listar().subscribe(res => {
      this.clientes = res;
    });
```

```
  }
}
```

Cliente.component.html

```
<p>
  cliente works!
</p>
<br />
<button (click)="logout();">Logout</button>
<br />
<h2>Listar Cliente</h2>
<div *ngFor="let item of clientes">
  {{item._id}},{{item.nome}},{{item.email}}
</div>
```

App.module.ts

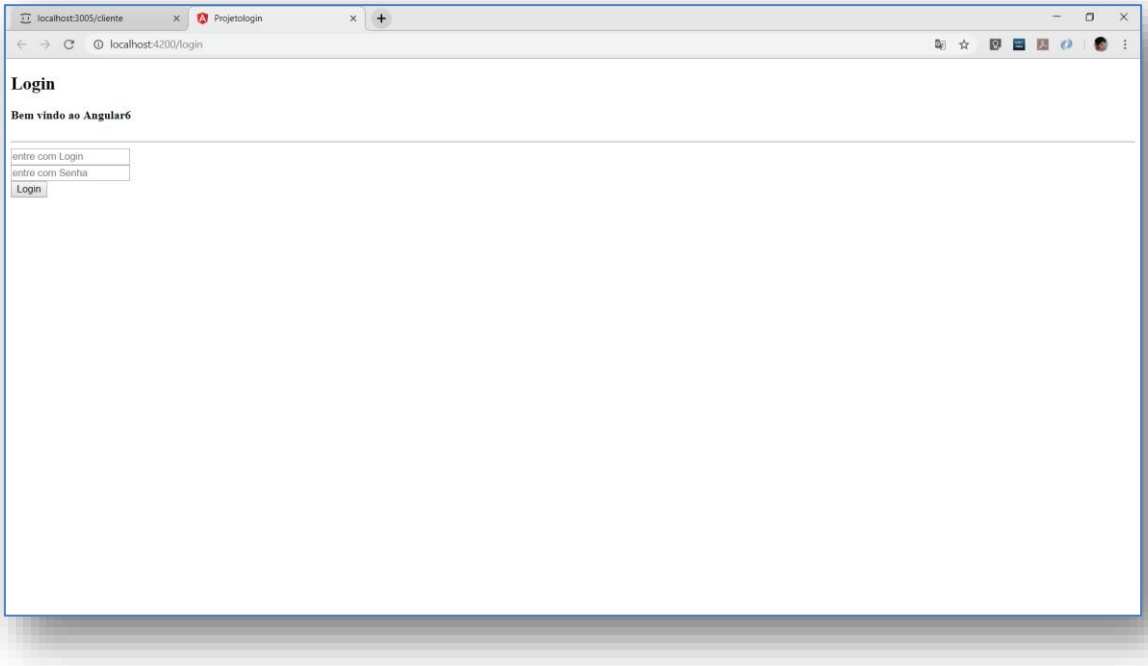
```
import { ClienteService } from
'./login/cliente/cliente.service';
import { AuthGuard } from './auth/auth.guard';
import { LoginService } from './login/login.service';
import { LoginComponent } from './login/login.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule, Component } from '@angular/core';
import { AppComponent } from './app.component';
import { ClienteComponent } from
'./login/cliente/cliente.component';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { Routes, RouterModule } from '@angular/router';

const ROTAS: Routes = [
  { path: 'login', component: LoginComponent },
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'cliente', canActivate: [AuthGuard], component:
ClienteComponent }
```

```
]

@NgModule({
  declarations: [
    AppComponent,
    ClienteComponent,
    LoginComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    RouterModule.forRoot(ROTAS)
  ],
  providers: [LoginService, ClienteService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

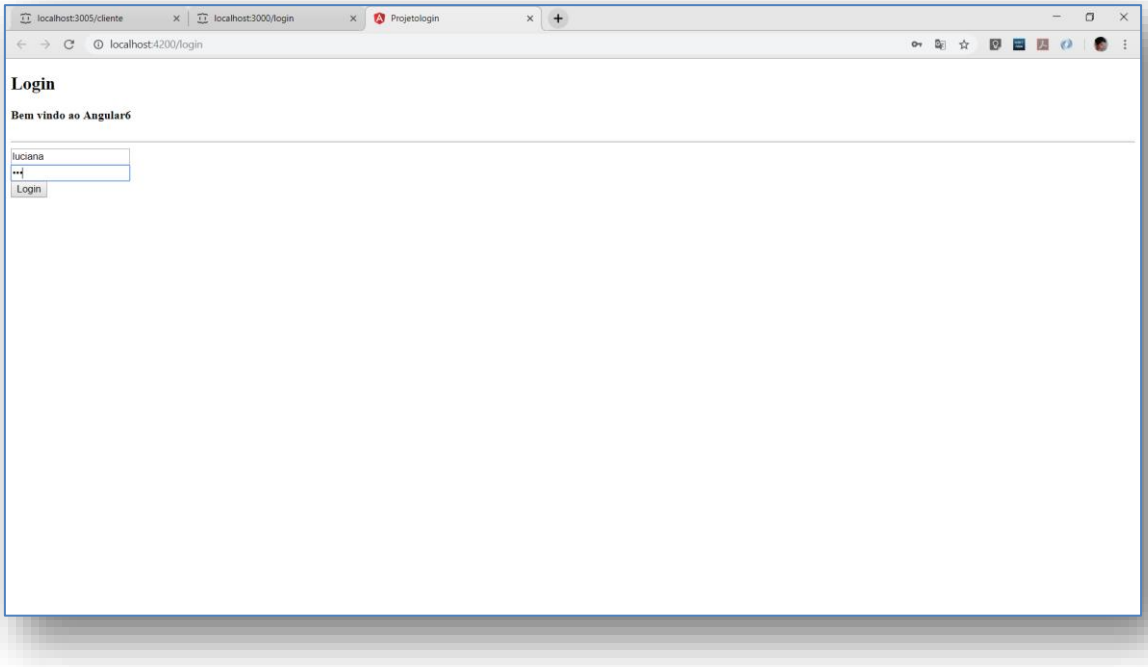
App.component.html

```
<router-outlet></router-outlet>
```

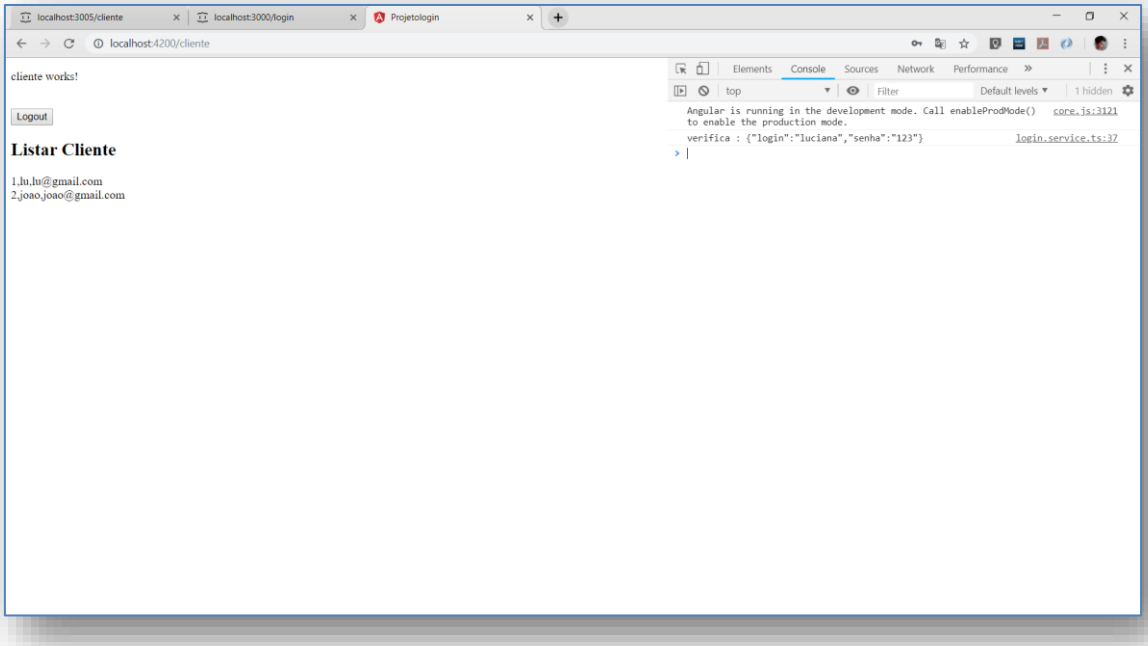
Rodando o projeto..
http://localhost:4200/login



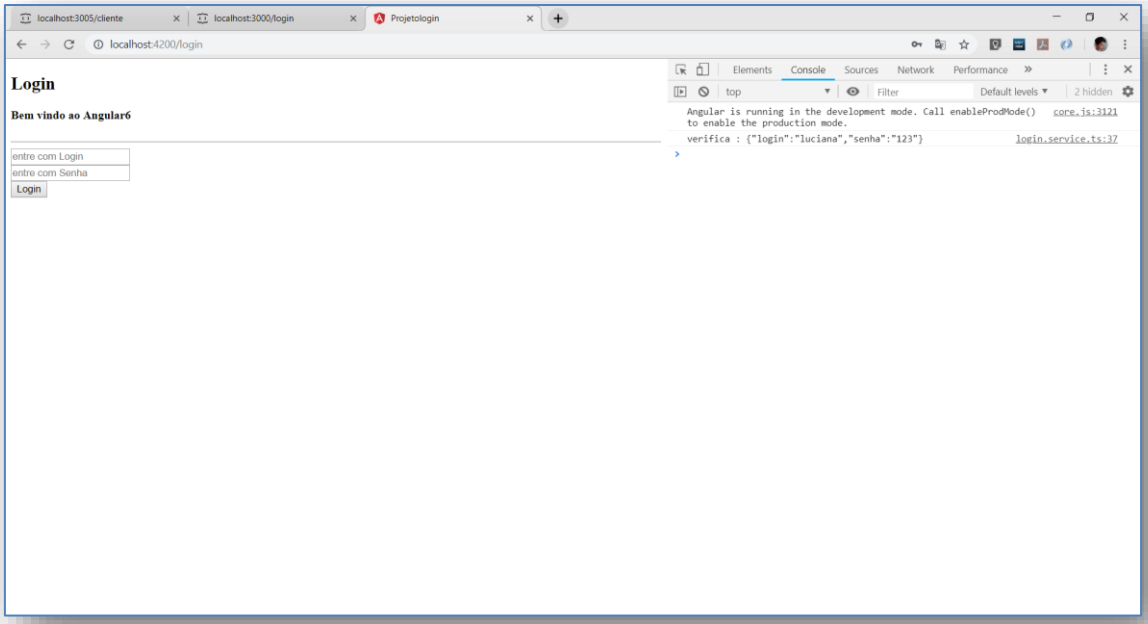
Logando...



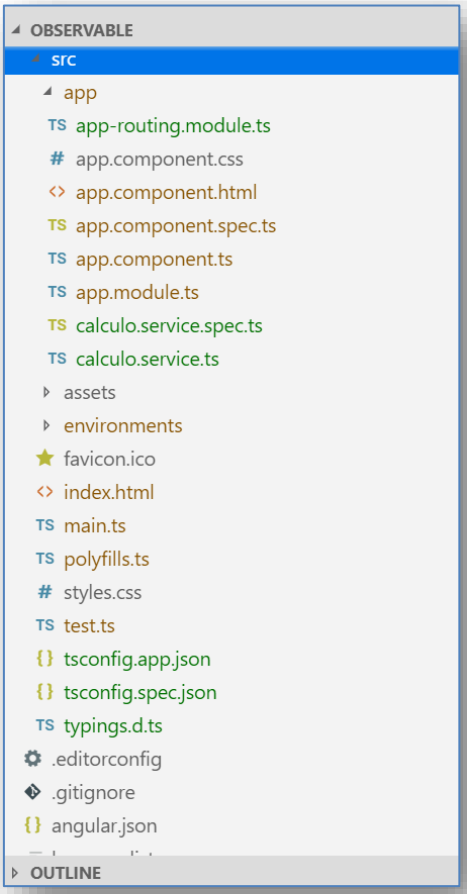
Logado...



Clicando em logout. Volta para login..



Projeto Observable



calculo.service.ts

```
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { from as fromPromise } from 'rxjs';

@Injectable()
export class CalculoService {

    constructor() { }

    generateNumber(): Promise<number> {
        let promise = new Promise<number>((resolve, reject) => {
            setTimeout(() => {
                let min = 2;
                let max = 10;
```

```
                let result = Math.random() * (max - min) + min;
                resolve(Math.round(result));
            }, 2000);
        });
        return promise;
    }

    generateNumberObservable(): Observable<number> {
        return fromPromise(this.generateNumber());
    }
}
```

---

app.component.ts

```
import { Component } from '@angular/core';
import { CalculoService } from './calculo.service';

@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css'],
    providers: [CalculoService]
})
export class AppComponent {

    multiplicador: number;
    resultado: number;
    numeroGerado: number;

    constructor(private service: CalculoService) {

    }

    calculaPromise() {
        this.service.generateNumber().then((num) => {
            this.calcula(num);
        }).then((n1) => {
            console.log('Dobro :', (this.numeroGerado * 2))
        }).then((n2) => {
            console.log('triplo :', (this.numeroGerado * 3))
```

```
        });
    }

    private calcula(num: number) {
        this.numeroGerado = num;
        this.resultado = this.multiplicador * num;
        return this.resultado;
    }

    consultaRX() {
    this.service.generateNumberObservable().subscribe((num) => {
            this.calcula(num);
            console.log('dobro ', (num * 2))
        })

    }
}
```

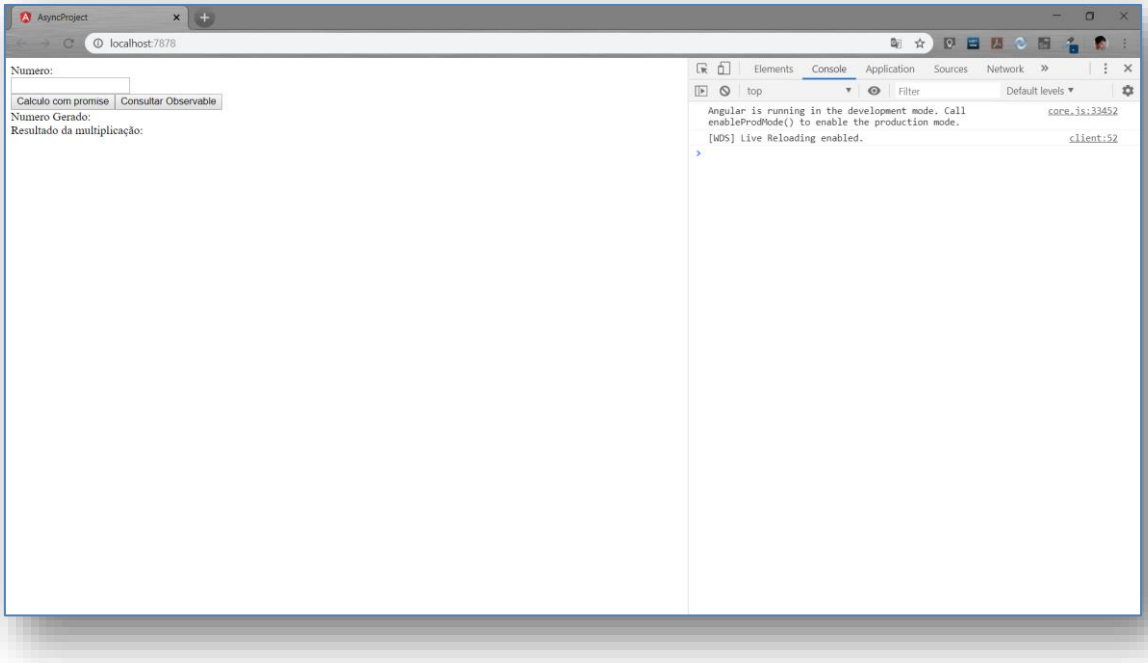
---

app.component.html

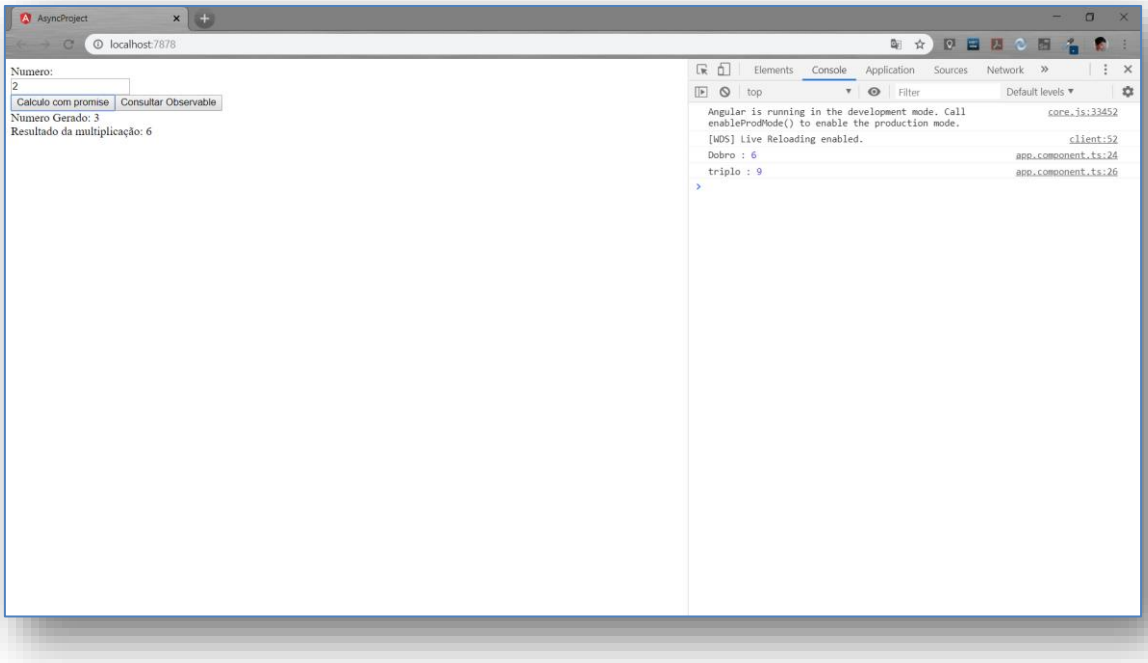
```
<form>

    <div>
        Numero: <br>
        <input type="number" [(ngModel)]="multiplicador"
name="mult">
    </div>

    <div>
        <button type="button" (click)="calculaPromise()">Calculo
com promise</button>
        <button type="button" (click)="consultaRX()">Consultar
Observable</button>
    </div>


    <div>
        Numero Gerado: {{numeroGerado}} <br>
        Resultado da multiplicação: {{resultado}}
    </div>
```

```
</form>
```

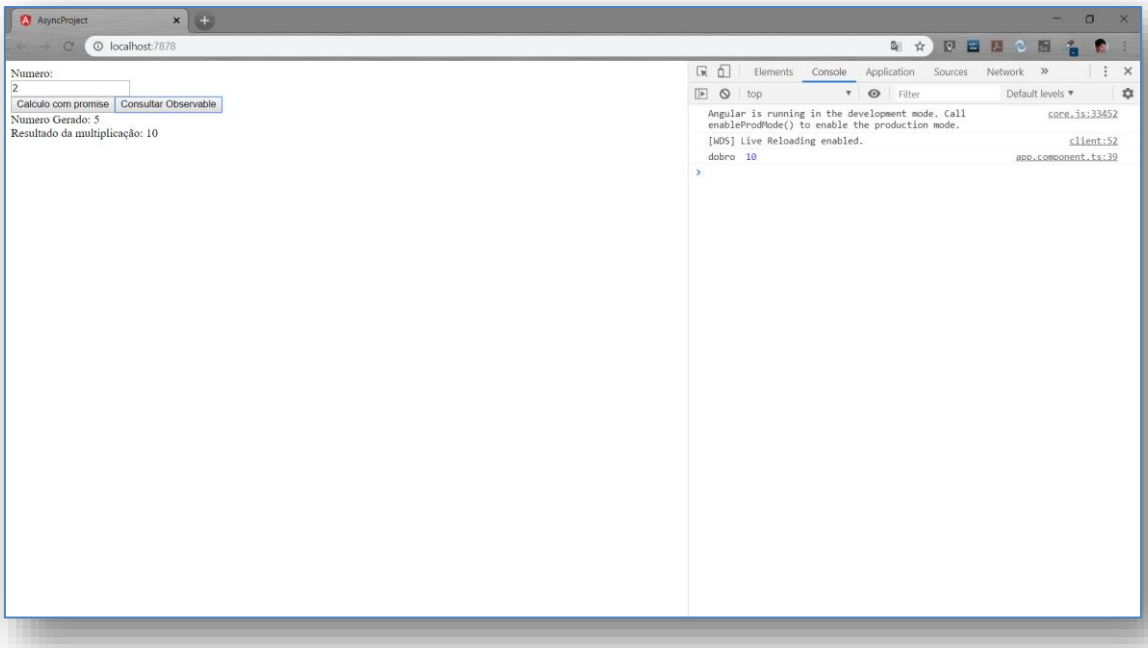
Rodando o projeto



Digitar um numero e clicar em promise



Digitar um numero e clicar em observable


AsyncProject
localhost:7878
Numero:
2
Calculo com promise
Consultar Observable
Numero Gerado: 5
Resultado da multiplicação: 10
Elements
Console
Application
Sources
Network
top
Filter
Default levels
Angular is running in the development mode. Call enableProdMode() to enable the production mode.
core.js:33452
[WDS] Live Reloading enabled.
client:52
dobro 10
app.component.ts:39