BIT PAGG LTDA

BOAS PRÁTICAS ANGULAR

Araxá
2018

Sumário

1. Boas Práticas Projeto

Para a utilização de angular em alguma aplicação existe algumas boas práticas que irão auxiliar no desenvolvimento do projeto, tendo uma melhor organização no decorrer do projeto.

Para iniciar um projeto angular é ideal utilizar o “prompt”, efetue instalação do “angular cli” globalmente com o comando “npm install -g @angular/cli” em seguida para gerar o projeto execute o seguinte comando “ng new ‘nome-do-projeto’”, e será criado as sequências de pastas da raiz do projeto, para a criação de componentes no decorrer do projeto e aconselhável que seja feito através do comando “ng g c ‘nome-do-componente’”, pois irá ser criado um componente com todas as dependências necessárias sem se preocupar se está faltando algo, e também na criação de uma sevice com o comando “ng g service ‘nome-service’”.

Em relação aos componentes crie um padrão para nomenclatura para os mesmos, e claro que isto se aplica para qualquer tipo de código, seja para evitarmos números mágicos arbitrários ou para evitar a tentação de abreviar variáveis. Em um mundo perfeito, seus componentes terão nomes que são bem legíveis e fluem como um texto quando colocados no seu código. Quando for registrar o nome dos variáveis e/ou funções, utilize o estilo **camelCase** para ser algo bem legível: “minhaVariavel”, “funcionalidadeTipo”. Já para Classes e Interfaces utilize o estilo **PascalCase** “MinhaClasse”, é simples, fácil e faz sentido do ponto de vista lógico. Você pode escolher outra convenção, mas o que importa é ser consistente e ter algum tipo de método no meio da loucura.

Para melhor organização de seus componentes e aconselhável fazer o seguinte, apenas um componente por arquivo, tentando manter este arquivo com menos de 400 linhas de código. Isto faz o seu código substancialmente mais fácil de ler, navegar e manter. Isto previne conflitos ao utilizar um SCM como o Git e também problemas com declaração de variáveis repetidas e coisas semelhantes.

Quando houver a necessidade de criar alguma classe ou serviço que será utilizado em toda a aplicação e em qualquer momento, e aconselhável que efetue a criação de uma pasta referente ao serviço ou a classe que irá utilizar, pois ficar mais fácil a localização dos mesmos e melhora a organização das pastas na raiz do projeto.

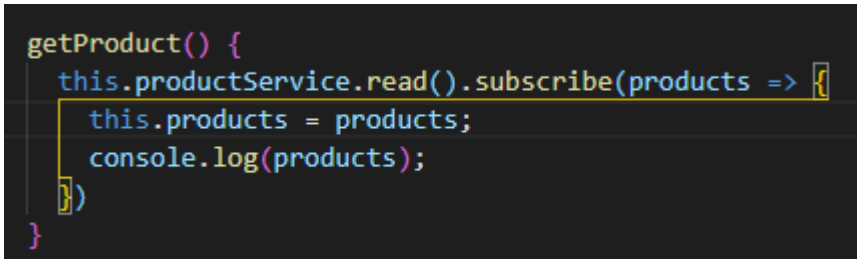
Sempre que possível utilize interface no lugar de classes, pois não precisam e nem podem ser instanciadas, o resultado da transpilação, “pegar seu

código e converter ele para outro em tempo de compilação", não gera código JavaScript isso é importante porque diminui o tamanho do build e melhora o tempo de processamento para tarefas como concat e minify), ou seja só usar classe, mesmo quando precisar fazer um parse dos dados vindos do servidor.

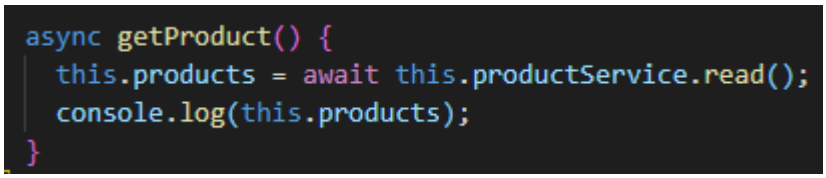
Crie pasta com nome da feature que ela representa, a estrutura fica mais legível e não existem nomes redundantes e repetitivos. O desenvolvedor consegue localizar rapidamente o código e identificar o que cada arquivo representa.

É sempre uma boa prática extrair sua lógica de negócios principal para os serviços. Dessa forma, fica muito mais fácil de manter, pois pode ser trocado e substituído por uma nova implementação em apenas alguns segundos. O mesmo vale para testes. Muitas vezes você precisa de serviços que buscam dados externos para falsificar os resultados em um ambiente de teste. Se você buscar seus dados nos serviços, isso é fácil. Se não, terá que mudar todas as linhas que precisam ser alteradas para isso.

Falando de serviços, quando for criar um serviço para consumo das APIs utilizaremos promisses com o async/await, pois com subscribe “observable”, pode ocorrer vazamento de memória e problemas de performance da aplicação.



Com async / await



O controle de erro “try/catch” pode ser feito no serviço, ou no componente quando utilizar várias requisições.

Mantenha o tamanho do seu aplicativo pequeno importando apenas o que você precisa, cada declaração de importação que você usa aumenta o tamanho do seu pacote, tendo algumas bibliotecas muito grandes e ao usar uma declaração de importação incorreta, você pode acabar com toda a biblioteca de seu aplicativo.

O Angular não usa diretivas tanto quanto o Angular.js, mas ainda tem algumas como: ngIf, ngFor. Então sempre que necessário usar estas diretivas pois com certeza irá aumentar o desempenho de sua aplicação.

Utilizar interpolação ao concatenar strings:

Trocar de “environment.API_URL + '/authenticator/ticket'” para

`${environment.API_URL}/authenticator/ticket`

2. Coisas que Não se Deve Fazer

Iremos falar um pouco do que não e aconselhável fazer em aplicações angular, para evitar erros e melhorar desempenho e manutenibilidade de sua aplicação angular.

Os componentes são os blocos de construção essenciais em um ecossistema angular, a ponte que conecta a lógica de nosso aplicativo com a visão. Mas às vezes os desenvolvedores ignoram fortemente os benefícios que um componente oferece.

Pode acontecer de criarem um código dentro de um componente que se repete em outros componentes ou até mesmo no mesmo componente, então o ideal e transformar essa parte do código que se repete em componente individual, para melhor acesso a aplicação.

Às vezes, você pode tender a pensar nos dados trazidos de um servidor / API como qualquer. Isso não é realmente o caso, você deve definir os tipos de dados que você recebe do seu back-end, para otimizar o uso dos dados em seus componentes por isso que o angular opta por ser usado principalmente no TypeScript.

Eu sugiro não fazer isso em um serviço, pois os serviços são para chamadas de API, compartilhando dados entre componentes e outros utilitários. As manipulações de dados devem pertencer aos componentes e modelos separados.

E ao criar um serviço crie o mesmo na pasta “app” da aplicação, pois todos os outros subcomponentes criados dentro da pasta “app” poderão utilizar a mesma informação que o serviço irá trazer, e com alerto que não e aconselhável criar mais de uma service na aplicação, e criar dentro das pastas dos subcomponentes com as mesmas funcionalidades do serviço principal, pois pode haver algum tipo de conflito, e a estrutura de pastas ficará desorganizada para futuras manutenções e implementações.

Cuidado ao declarar o seu componente no “AppModule”, pois ele não deve ser declarado mais de uma vez na module, por isso aconselho a criar o componente através de linhas de comando pelo prompt, pois ao criar automaticamente o comando já declara a instância do componente no “AppModule”.

3. Boas Práticas de Segurança

Iremos falara sobre algumas boas práticas em relação a desenvolvimento de sua aplicação.

Uma boa prática para a segurança, recomenda-se atualizar as bibliotecas Angular em intervalos regulares. Não o fazer pode permitir que invasores ataquem o aplicativo usando vulnerabilidades de segurança conhecidas presentes em versões mais antigas.

Em relação a comunicação do front-end e back-end pode-se usar o jwt, que e usado para codificar e decodificar token, usando este token para efetuar validações de usuário e transferência de informações via chamadas Rest.

Previna o script entre sites (XSS) que permite que invasores injetem códigos maliciosos em páginas da Web. Esse código pode, por exemplo, roubar dados do usuário (em particular, dados de login) ou executar ações para representar o usuário. Este é um dos ataques mais comuns na web.

Para bloquear ataques XSS, você deve impedir que códigos maliciosos entrem no DOM (Document Object Model). Por exemplo, se os invasores puderem induzi-lo a inserir uma tag <script> no DOM, eles poderão executar um código arbitrário em seu website. O ataque não está limitado a tags <script> - muitos elementos e propriedades no DOM permitem a execução de códigos, por exemplo, <img onerror = "..."> e <a href="javascript:...">. Se dados controlados pelo invasor entrarem no DOM, espere vulnerabilidades de segurança.

Previnir as APIs do navegador incorporado, pois não protegem você automaticamente contra vulnerabilidades de segurança. Por exemplo, documento, o nó disponível por meio do ElementRef e muitas APIs de terceiros contêm métodos não seguros. Da mesma forma, se você interagir com outras bibliotecas que manipulam o DOM, provavelmente não terá a mesma sanitização automática das interpolações angulares. Evite interagir diretamente com o DOM e, em vez disso, use modelos angulares sempre que possível.

Para casos em que isso é inevitável, use as funções de higienização angular incorporadas. Sanitize valores não confiáveis com o método DomSanitizer.sanitize e o SecurityContext apropriado. Essa função também aceita valores que foram marcados como confiáveis usando as funções bypassSecurityTrust ... e não os higienizará, conforme descrito abaixo.

Ter cuidado ao incluir código executável, exibir um <iframe> de algum URL ou construir URLs potencialmente perigosos. Para evitar a sanitização automática em qualquer uma dessas situações, você pode informar ao angular que você inspecionou um valor, verificou como ele foi gerado e garantiu que ele estivesse sempre seguro. Mas tenha cuidado, se você confia em um valor que pode ser mal-intencionado, está introduzindo uma vulnerabilidade de segurança em seu aplicativo

4. Parâmetros entre Páginas

Para efetuar a transição de parâmetros entre as páginas da aplicação existem maneiras diferentes de fazer isso, mais precisamente iremos abordar 3 dessas maneiras.

A primeira que iremos desenvolver será através de uma função que criaremos, nesta função usaremos a “queryParam” para passarmos os valores dos parâmetros para a outra da pagina de nossa aplicação, o único empecilho que está formato de solução e que ele além de passar os valores nas variáveis ele também exibe na URL da rota. Então não e aconselhado usar este método por não ter segurança das informações.

A principio devemos importar a biblioteca de rotas do angular e navegação.

```
import {Router, NavigationExtras} from "@angular/router";
```

Em seguida devemos declarar uma variável do tipo “Router” no construtor, que irá disponibilizar as opções de métodos da biblioteca.

```
// O CookieService e para ser usado no segundo metodo dentro do oninit()...
// O ActvatedRoute e para receber a rota de onde esta vindo a queryParam do primeiro metodo(Não aconselhavel)
// O nameModulo e usado no terceiro metodo de transferencia chamando um classe em modulo
constructor(public  loginService: AppService,
            private router: Router,
             public  cookieService: CookieService,
            private  NameModulo: nameModulo
             ) {}
```

Então criamos nosso método que irá pegar as variáveis colocadas em uma variável do tipo “NavigationExtras”, dentro do parâmetro “queryParams” onde o conteúdo das variáveis estará armazenado.

Em seguida informamos para qual rota vai ser passado os parâmetros que estão em “NavigationExtras”, com isso ao chamar esta função em um evento de tela qualquer ela mudará para a tela correspondente com a rota que foi colocada, e na URL aparecerá a rota concatenado com as informações dos parâmetros.

```
  public onTap() {
        var navigationExtras: NavigationExtras = {
            queryParams: {
              "firstname": "Nic",
                "lastname": "Raboy"
            }
        };
        this.router.navigate(['/login2'], navigationExtras);
    }
verifica() {
  console.log('ok');
  this.onTap();
```

Na segunda página para onde você direcionou os parâmetros você de importar a biblioteca “@angular/router” como na primeira página.

```
import {Router, NavigationExtras} from "@angular/router";
```

Em seguida declare as variáveis que irão receber os parâmetros vindos da página anterior dentro da classe do componente.

```
export class Login2Component implements OnInit {

   public firstname: string; // Usada no prmeiro metodo de passagem de parametros atraves de função (Não aconselhavel)
   public lastname: string; // Usada no prmeiro metodo de passagem de parametros atraves de função (Não aconselhavel)
```

Em seguida devemos declarar uma variável do tipo “Router” no construtor, que irá disponibilizar as opções de métodos da biblioteca.

```
// O CookieService e para ser usado no segundo metodo dentro do oninit()...
// O ActvatedRoute e para receber a rota de onde esta vindo a queryParam do primeiro metodo(Não aconselhavel)
// O nameModulo e usado no terceiro metodo de transferencia chamando um classe em modulo
public constructor( private route: ActivatedRoute,
                    public cookieService: CookieService,
                    private  NameModulo: nameModulo) {
```

Então para receber os parâmetros devemos chamara a “queryParams” de “route” e sobrescrever os parâmetros nas variáveis que determinamos para receber as informações.

```
    // Primeiro metodo não aconselhavel pois ele fica gravado as informações das variaveis na url...
    this.route.queryParams.subscribe(params => {
        this.firstname = params["firstname"];
        this.lastname = params["lastname"];
    });
    console.log(this.firstname);
    console.log(this.lastname);
}
```

A segunda opção para passar os parâmetros para outra página e usando o cookie do navegador, neste método armazenamos os parâmetros no cookie do navegador e depois damos um “get” para chamarmos os mesmos para utilizarmos.

Antes de começar a implementar transferência vai prompt de comando e execute o seguinte comando “npm install ngx-cookie-service --save” para utilizar a biblioteca de armazenamento no cookie, em seguida vá em “app. modules” e faça o seguinte:

Importe o “CookService” da biblioteca “ngx-cookie-service”.

```
import { CookieService } from 'ngx-cookie-service';
```

Em seguida coloque ele na lista de providers, para que possamos usa-la no resto da aplicação, pois “CookService” irá fornecer um serviço para o resto da aplicação.

```
@NgModule({
  declarations: [
    AppComponent,
    HeaderComponent,
    FooterComponent,
    LoginComponent,
    Login2Component,

  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    HttpClientModule,
    RouterModule.forRoot(ROUTES, {onSameUrlNavigation: 'reload'}),

  ],
  providers: [CookieService ],
  bootstrap: [AppComponent],
})
export class AppModule { }
```

Na primeira página onde será recolhido o parâmetro primeiramente devemos importar a biblioteca para utilizarmos os serviços do cookie do navegador.

```
import { CookieService } from 'ngx-cookie-service';
```

Em seguida declarar uma variável do tipo “CookieService” para utilizarmos suas dependências.

```
// O CookieService e para ser usado no segundo metodo dentro do oninit()...
// O ActvatedRoute e para receber a rota de onde esta vindo a queryParam do primeiro metodo(Não aconselhavel)
// O nameModulo e usado no terceiro metodo de transferencia chamando um classe em modulo
constructor(public  loginService: AppService,
            private router: Router,
            public  cookieService: CookieService,
            private  NameModulo: nameModulo
            ) {}
```

Em seguida vá em “ngOnInit()” e damos um “set” em “cookieService” para armazenarmos na variável “Test” a string “Hello world”, onde será gravada no cookie do navegador.

```
ngOnInit() {
  this.Token();
  // Usada no segundo metodo de passagem de parametros atraves de cookie no navegado,
  // esta variavel seta um valor para a variavel 'Test', ou seja 'Test' recebe 'Hello World'
  this.cookieService.set( 'Test', 'Hello World' );
  // Usada no segundo metodo de passagem de parametros atraves de cookie no navegado,
  // esta variavel recebe o que esta na variavel 'Test'
  //this.cookieValue = this.cookieService.get('Test');
  //console.log(this.cookieValue)

}
```

Na segunda página onde irá utilizar a variável que armazenou no cookie, primeiramente devemos importar a biblioteca para utilizarmos os serviços do cookie do navegador, como na primeira página.

```
import { CookieService } from 'ngx-cookie-service';
```

Em seguida declarar uma variável do tipo “CookieService” para utilizarmos suas dependências.

```
// O CookieService e para ser usado no segundo metodo dentro do oninit()...
// O ActvatedRoute e para receber a rota de onde esta vindo a queryParam do primeiro metodo(Não aconselhavel)
// O nameModulo e usado no terceiro metodo de transferencia chamando um classe em modulo
public constructor( private route: ActivatedRoute,
                    public cookieService: CookieService
                    private  NameModulo: nameModulo) {
```

Em seguida declare a variáveil que irá receber o parâmetro que foi gravado, dentro da classe do componente.

```
export class Login2Component implements OnInit {
    public cookieValue: any; // Usada no segundo metodo de passagem de parametros atraves de cookie no navegado
```

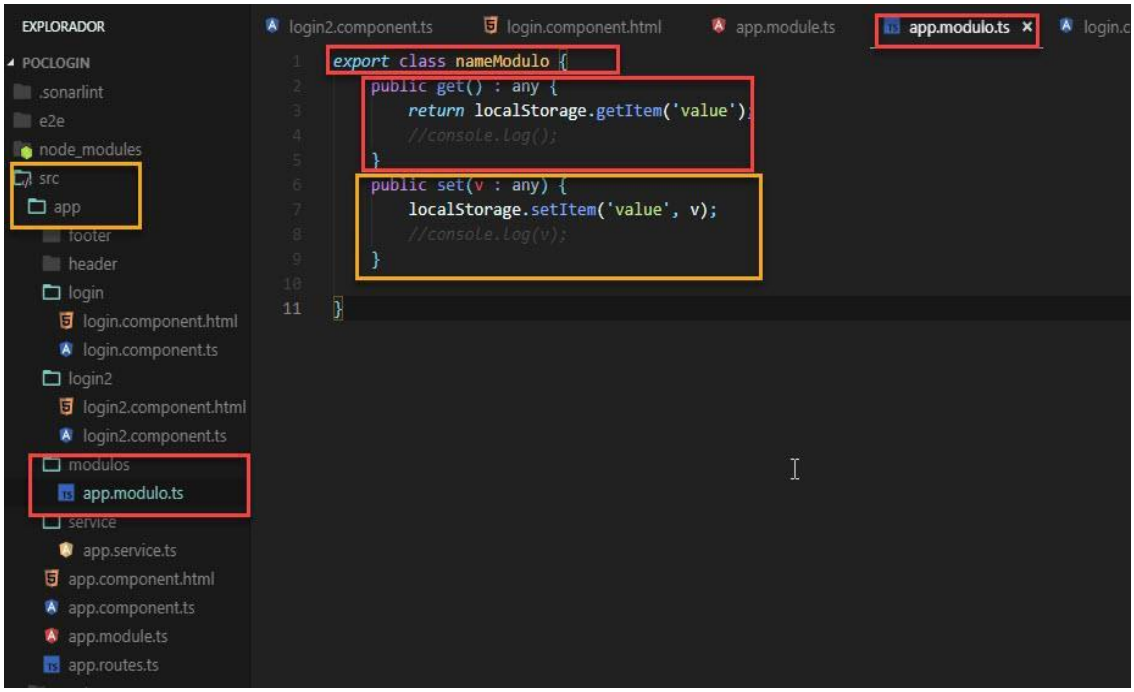
Em seguida vá em “ngOnInit()” e damos um “get” em “cookieService” com nome do parâmetro onde foi gravado no cookie, e armazene o retorno na variável que declarou.

```
  ngOnInit() {
    // Usada no segundo metodo de passagem de parametros atraves de cookie no navegado,
    // esta variavel recebe o que esta na variavel 'Test'
    this.cookieValue = this.cookieService.get('Test');
    // Usada no segundo metodo de passagem de parametros atraves de cookie no navegado
    console.log(this.cookieValue)
    }

}
```

O terceiro método para transferir parâmetros entre paginas e usar uma classe de variáveis para efetuar essa transição, ou seja, criando uma classe para ser utilizada como ponte, armazenando os valores necessários e depois buscando onde e quando necessário.

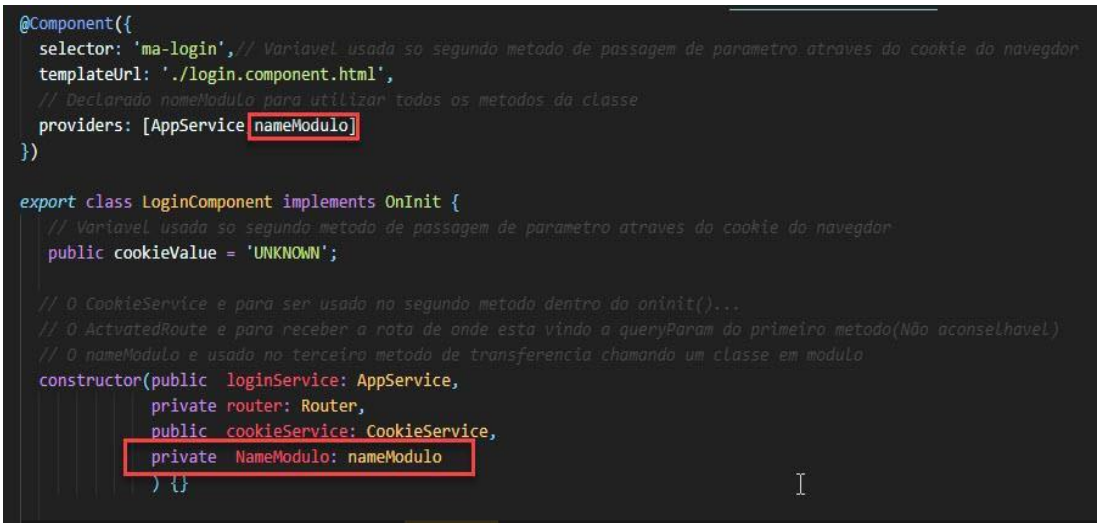
Primeiro e necessário criar uma pasta dentro de “app” e dentro desta pasta criar um arquivo “nomearquivo”.ts, e dentro deste arquivo será criada uma classe, dentro desta crie duas funções uma “get” e a outra “set”. A função “set” terá a função de armazenar os valores dentro de “value” e a “get” terá a função de fornecer o valor de “value” quando for chamado.



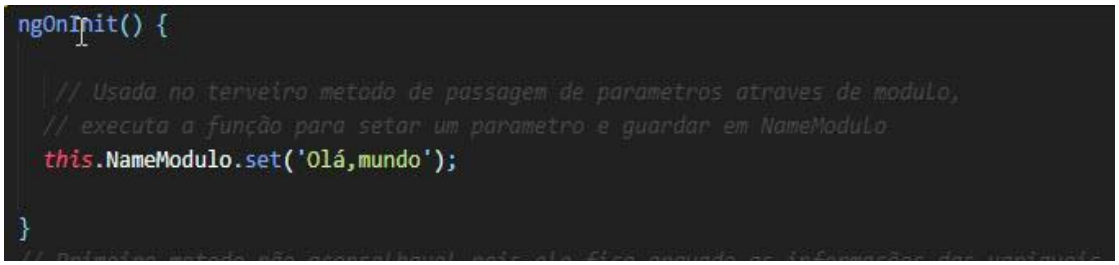
Em seguida na primeira página onde for recolher as variáveis, deverá ser importado a classe “nameModulo”, está na “app.modulo”.

```
import { nameModulo } from '../modulos/app.modulo';
```

Em seguida deve-se declarar o nome da classe na lista de “providers” do componente, pois iremos utilizar um serviço que vem de fora do componente, depois declarar no construtor do componente uma variável do tipo “nameModulo” para herdar todas as funções que estão na classe.



Em seguida na função “ngOnInit()” do componente da página onde irá recolher os dados, chame a variável que declarou no construtor do componente e atribuir a função “.set()” que e herdada da classe e colocar a variável que deseja armazenar para chamar em outra página.



Sendo assim no componente da segunda página onde pretende utilizar o valor da variável que foi guardada dentro da classe, e necessário primeiro importar a mesma classe que usamos na página anterior.

Em seguida repetir os mesmos passos que fizemos na página anterior, declarar o nome da classe na lista de “providers” do componente, pois iremos utilizar um serviço que vem de fora do componente, depois declarar no construtor do componente uma variável do tipo “nameModulo” para herdar todas as funções que estão na classe

```
@Component({
  selector: 'ma-login2',
  templateUrl: './login2.component.html'
   // Declarado nameModulo para utilizar todos os metodos da classe
  providers: [nameModulo]
})
export class Login2Component implements OnInit {

    public firstname: string; // Usada no prmeiro metodo de passagem de parametros atraves de função (Não aconselhavel
    public lastname: string; // Usada no prmeiro metodo de passagem de parametros atraves de função (Não aconselhavel)

    public cookieValue: any; // Usada no segundo metodo de passagem de parametros atraves de cookie no navegado

    // O CookieService e para ser usado no segundo metodo dentro do oninit()...
    // O ActvatedRoute e para receber a rota de onde esta vindo a queryParam do primeiro metodo(Não aconselhavel)
    // O nameModulo e usado no terceiro metodo de transferencia chamando um classe em modulo
    public constructor( private route: ActivatedRoute,
                        public cookieService: CookieService,
                        private  NameModulo: nameModulo) {
```

Em seguida na função “ngOnInit()” do componente da página onde irá recolher os dados, chame a variável que declarou no construtor do componente e atribuir a função “.get()” que é herdada da classe e armazene o retorno desta função na variável que deseja utilizar para manipular dentro da página.

```
  ngOnInit() {

    // Usada no terveiro metodo de passagem de parametros atraves de modulo,
    // executa a função para buscar o valor guardado na classe
    this.myItem = this.NameModulo.get()
    console.log(this.myItem);
    }

}
```

5. Controle de armazenamento de dados

Como vimos no tópico anterior podem ser usadas diferentes formas para armazenamento de variáveis dentro da aplicação, mas algumas delas devem ser tratadas ao término do uso do usuário e/ou ao fechamento da aplicação, para que não haja informações que possam prejudicar e expor o usuário.

A session e os cookies são muito utilizados nas aplicação para segurar o a seção do usuário dentro da aplicação, mas por padrão foi definido que ao fechamento da aba da aplicação ou em qualquer outro caso que haja retenção de dados do usuário ao termino da utilização da aplicação deve ser limpa e apagado qualquer tipo de dado retido na seção ou em cookies.

Para o armazenamento de dados na session usa-se o localStorage e SessionStorage, para a limpeza dos mesmos e necessário a utilização das seguintes funções, "**localStorage.removeItem('NomeDaVariavel')**","**localStorage.clear()**", "**sessionStorage.removeItem('NomeDaVariavel')**", "**sessionStorage.clear()**", com estas funções e possível a limpeza dos dados armazenados.

Para o armazenamento de dados utilizando os cookies no browser também e necessário a limpeza dos mesmos, para efetuar esta limpeza e utilizado as seguintes Bfunções, "**Cookie.delete('NomeDaVariavel');**","**Cookie.deleteAll();**", com estas funções e possível efetuar a limpeza dos cookies no navegador.

O armazenamento em cookie pode ser acessado pelo usuário a qualquer momento, então não e aconselhável o uso dos cookies para trafegar dados sensíveis na aplicação, porém a maneiras de efetuar a gravação nos cookies limitando o acesso aos dados gravados.

A utilização do **"Secure"**, parametro que garante o cookie só pode ser transmitido com segurança por HTTPS e não será enviado por conexões http não criptografadas:

```
document.cookie = 'name=Flavio; Secure;'
```

E também pode ser usado o **"HttpOnly"** que torna o cookie inacessível por meio do "document.cookie" API, portanto, eles são editáveis apenas pelo servidor:

```
document.cookie = 'name=Flavio; Secure; HttpOnly'
```

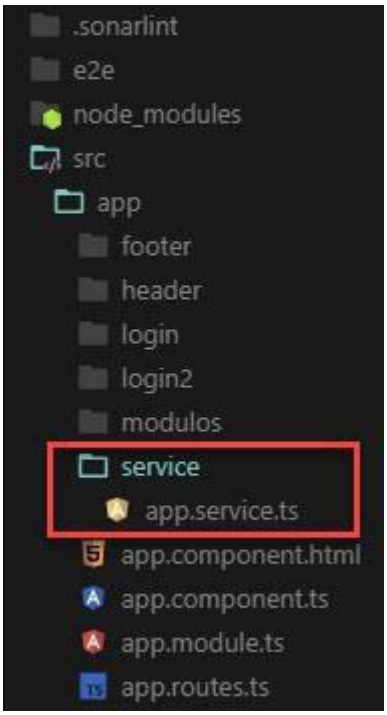
Objetivos das funções citadas:

Função	Funcionalidade
localStorage.removeItem('NomeDaVariavel')	Esta função tem a funcionalidade de apagar o conteúdo armazenado em localStorage com nome específico citado.
localStorage.clear()	Esta função tem a funcionalidade de apagar todos os conteúdos

	armazenado em localStorage, referente a todas as variáveis criadas.
sessionStorage.removeItem(‘NomeDaVariavel’)	Esta função tem a funcionalidade de apagar o conteúdo armazenado em sessionStorage com nome específico citado.
sessionStorage.clear()	Esta função tem a funcionalidade de apagar todos os conteúdos armazenado em sessionStorage, referente a todas as variáveis criadas.
Cookie.delete(‘NomeDaVariavel’)	Esta função tem a funcionalidade de deletar o conteúdo armazenado em Cookies com nome específico citado.
Cookie.deleteAll()	Esta função tem a funcionalidade de apagar todos os conteúdos armazenado em Cookies, referente a todas as variáveis criadas.

6. Consumo API pela Service

Para o consumo de uma API ou proxy, e necessário efetuar a criação de uma service, para a criação da service primeiro crie uma pasta com o nome “service” e navegue até a pasta usando o prompt, chegando dentro da pasta execute o seguinte comando “ng g service ‘nome_service’ --spec”, então será criado a service.



Após criado a service por default ela virá com os seguintes elementos. Já crida a classe onde será feita as chamadas externas.

app.service.ts

```
1  import { Injectable } from '@angular/core';
2
3  @Injectable({
4    providedIn: 'root'
5  })
6  export class AppService {
7
8    constructor() { }
9  }
10
```

Para efetuar as chamadas Http/Rest, GET, POST, PUT e DELETE, você deverá importar a biblioteca http do angular, para efetuar as chamadas via url.

```
import { HttpClient } from '@angular/common/http';
```

Em seguida declare uma variável no construtor da classe “AppService” que irá receber “HttpClient”, que herda as dependências da biblioteca http importada.

```
@Injectable()
export class AppService {
  constructor(private http: HttpClient) {}
```

Agora vamos criar as funções referentes as chamadas rest, para exemplificar a função GET, usaremos uma função chamada “ping()”, nesta função usamos a variável “http” que declaramos no construtor, para utilizarmos a função “.get()” que vem da biblioteca “http” do angular, e dentro do dos parênteses da função get e declarado a url, da API ou proxy que será feita a chamada. Então a função “ping()” retornará a resposta da chamada.

```
ping() {
  return this.http.get(this._endpoint.concat('autenticador/ping'));
}
```

Para exemplificar a função POST executa o processo que está no corpo como um subordinado da URL, usaremos uma função chamada “Login(inBody, inHeader)”, nesta função usamos a variável “http” que declaramos no construtor, para utilizarmos a função “.post()” que vem da biblioteca “http” do angular, e os parâmetros passados na função serão usados para transferir os dados no corpo da chamada e no cabeçalho, então as informações pertinentes a esses dois lugares deverão estar dentro dos parênteses quando chamada à função “Login()”, e dentro do dos parênteses da função “.post()” e declarado a url da API ou proxy que será feita a chamada, e após a url e onde fica os parâmetros de inbody e inHeader que receberão as informações na chamada da função. Então a função “Login()” retornará a resposta da chamada.

```
Login(inBody, inHeader) {
 return this.http.post(this._endpoint.concat('Autenticador/Token'), inBody, {headers : inHeader} );
}
```

Para exemplificar a função PUT cria ou atualiza uma informação identificada por uma URL, usaremos uma função chamada “CancelarVenda(inHeader, url)”, nesta função usamos a variável “http” que declaramos no construtor, para utilizarmos a função “.put()” que vem da biblioteca “http” do angular, e os parâmetros passados na função serão usados para transferir os dados na url da chamada e no cabeçalho, então as informações pertinentes a esses dois lugares deverão estar dentro dos parênteses quando chamada à função “CancelarVenda()”, e dentro do dos parênteses da função “.put()” e declarado a url da API ou proxy que será feita a chamada, e após a url e onde fica os parâmetros de inbody e inHeader que receberão as informações na chamada da função. Então a função “CancelarVenda()” retornará a resposta da chamada.

```
CancelarVenda(inHeader, url) {
  return this.http.put(this._endpoint.concat('AssistenciaMapfre/CancelarVenda/?CodOperacao=' + url),
                       {headers : inHeader });
}
```

Para exemplificar a função DELETE que deleta informação identificada por uma URL, usaremos uma função chamada “DeleteCpf(inHeader, url)”, nesta função usamos a variável “http” que declaramos no construtor, para utilizarmos a função “.delete()” que vem da biblioteca “http” do angular, e os parâmetros passados na função serão usados para transferir os dados na url da chamada e no cabeçalho, então as informações pertinentes a esses dois lugares deverão estar dentro dos parênteses quando chamada à função “DeleteCpf()”, e dentro do dos parênteses da função “.delete()” e declarado a url da API ou proxy que será feita a chamada, e após a url e onde fica os parâmetro inHeader que recebera as informações na chamada da função. Então a função “DeleteCpf()” retornará a resposta da chamada.

```
DeleteCpf(inHeader, url) {
return this.http.delete(this._endpoint.concat('NegocioComum/PessoaFisica?Cpf=' + url), {headers : inHeader });
}
```

Para receber o retorno das funções acima dentro dos componentes para podermos utilizar a informações que vem da API, fazemos da seguinte maneira, primeiro importamos a classe “AppService” de “app.service” para podermos chamar as funções que utilizamos para efetuar as chamadas rest.

```
import { AppService } from '../service/app.service';
```

Em seguida declaramos a classe “AppService” na lista de providers pois é um serviço externo ao componente, e em seguida declaramos uma variável do tipo “AppService” para herdar todas as funções que estão na classe.

```
@Component({
  selector: 'ma-login',// Variavel usada so segundo metodo de passagem de parametro atraves do cookie do navegdor
  templateUrl: './login.component.html',
  // Declarado nameModulo para utilizar todos os metodos da classe
  providers: [AppService,nameModulo]
})

export class LoginComponent implements OnInit {
   // Variavel usada so segundo metodo de passagem de parametro atraves do cookie do navegdor
   public cookieValue = 'UNKNOWN';

  // O CookieService e para ser usado no segundo metodo dentro do oninit()...
  // O ActvatedRoute e para receber a rota de onde esta vindo a queryParam do primeiro metodo(Não aconselhavel)
  // O nameModulo e usado no terceiro metodo de transferencia chamando um classe em modulo
  constructor(public  loginService: AppService,
              private router: Router,
              public  cookieService: CookieService,
              private  NameModulo: nameModulo
              ) {}
```

Em seguida para filtrarmos o retorno da função seja ela GET, PUT, POST ou DELETE, utilizamos o modelo a seguir:

```
Login() {
  try {
    this.loginService
      .Login(this.final, this.token)
      .subscribe(
        resp => {
          console.log(resp, 'res');
          this.retorno = JSON.stringify(resp);
        },
        error => {
          console.log(error, 'erro');
          this.retorno = JSON.stringify(error);
        }
      );
  } catch (e) {
    console.log(e);
  }
```

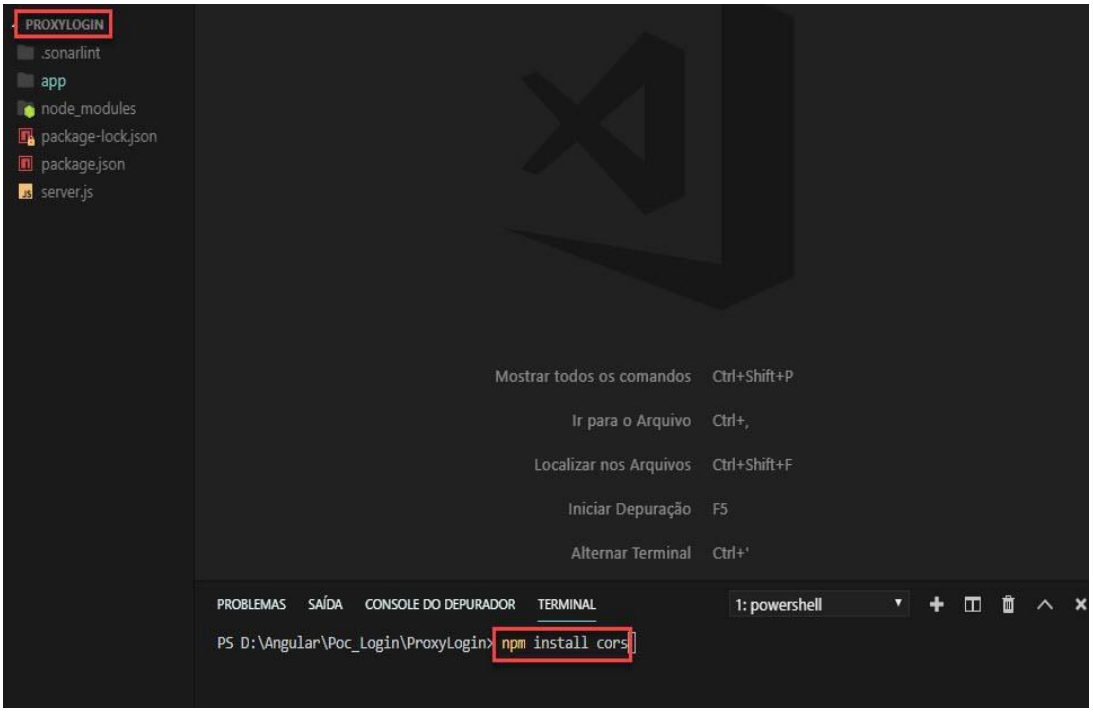
Onde criamos uma função e dentro dela criamos um “try” e “catch” para filtrar os erros, e dentro do “try” chamamos a “loginService” que foi declarada no construtor herdando as funções da classe “AppService”, então chamamos a função que desejarmos localizadas dentro da service, e dentro dos parênteses passamos as informações pertinente a função que declaramos, em seguida o “.subscribe” irá retornar a resposta da função, seja ela um erro ou não, nas variáveis “resp” e “error”.

7. Erro de Cors

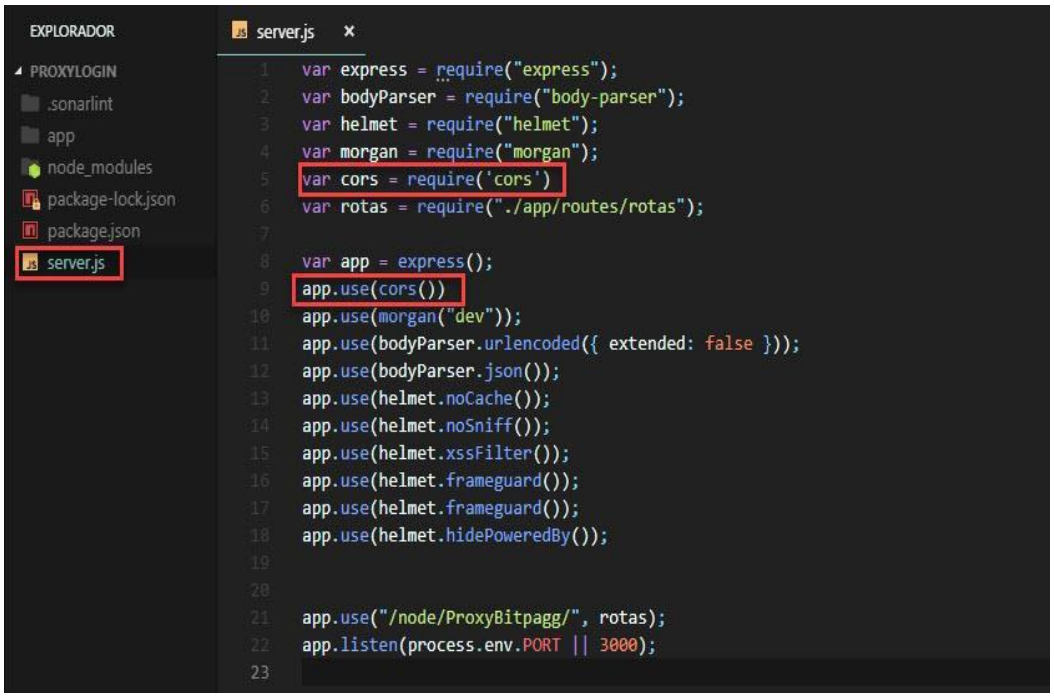
O compartilhamento de recursos de origem cruzada (CORS) permite que solicitações de recursos de acesso de hosts remotos. Mostrarei como ativar o suporte CORS no Express.

Para efetuar a ativação do suporte CORS irei utilizar da aplicação proxy que criamos como modelo, para facilitar o entendimento em relação a correção de erros referente a CORS da aplicação node.js.

Para efetuar a configuração de CORS primeiro acesse a sua aplicação pelo VSC (Visual Studio Code), no caso acessei a proxy que irei usar como exemplo, em seguida abra a central de comando do VS e digite “npm install cors”, para instalar as dependências de CORS.



Em seguida abra a “server.js” onde fica a configuração referente ao acesso a app, após aberto crie uma variável que irá herdar as dependências de CORS, após a declaração vá abaixo da variável app e declare que app use a variável de cors, assim ela utilizará a dependência de cors, resolvendo erros referente a compartilhamento de origens cruzadas.


EXPLORADOR
server.js
PROXYLOGIN
.sonarlint
app
node_modules
package-lock.json
package.json
server.js
var express = require("express");
var bodyParser = require("body-parser");
var helmet = require("helmet");
var morgan = require("morgan");
var cors = require('cors')
var rotas = require("./app/routes/rotas");
var app = express();
app.use(cors())
app.use(morgan("dev"));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(helmet.noCache());
app.use(helmet.noSniff());
app.use(helmet.xssFilter());
app.use(helmet.frameguard());
app.use(helmet.frameguard());
app.use(helmet.hidePoweredBy());
app.use("/node/ProxyBitpagg/", rotas);
app.listen(process.env.PORT || 3000);