

Projeto de Interfaces WEB

HTTP no Angular
GET, DELETE, POST e PUT
Aula 05

Introdução

- SPA – Single Page Application
 - São aplicações WEB com uma única página, onde o sistema é executado no navegador.
 - Páginas **não** são recarregadas em sistemas SPA.
 - O cliente faz uma requisição em HTTP (via GET) para um serviço que retorna os dados no formato JSON (geralmente).
 - O cliente, então, renderiza os dados no navegador.
 - HTTP é um protocolo com vários métodos: GET, POST, PUT, DELETE...

Preparando...

- Modifique o app.component.ts:

```
export class AppComponent {  
  contador:number = 5;  
  alunos:any[] =[  
    {id:1,nome:"Jefferson"},  
    {id:2,nome:"Fabrício"},  
    {id:3,nome:"Juliana"},  
    {id:4,nome:"Roberta"},  
  ]  
  
  adicionar(nome:string){  
    const temp = {id:this.contador,nome:nome};  
    this.contador++;  
    this.alunos.push(temp);  
  }  
  
  excluir(id:number){  
    alert(id);  
  }  
  
  atualizar(aluno:any){  
    alert(JSON.stringify(aluno))  
  }  
}
```

Preparando...

```
<div class="container">
  <div class="form-group">
    <label>Nome</label>
    <input type="text" class="form-control" #nomeInput>
  </div>

  <button type="button" class="btn btn-primary"
    (click)="adicionar(nomeInput.value);
    nomeInput.value="">Adicionar
  </button>

  <hr>
  <div class="row" style="font-weight: bold;">
    <div class="col-2">Id</div>
    <div class="col-10">Nome</div>
  </div>
  <hr>
```

...

```
<div class="row" *ngFor="let a of alunos" style="margin-top: 5px;">
  <div class="col-2">
    {{a.id}}
  </div>
  <div class="col-8">
    <div class="row">
      <div class="col-10" style="padding: 0;">
        <input type="text" class="form-control" value="{{a.nome}}">
      </div>
      <div class="col-2" style="padding: 0;">
        <button type="button" class="btn btn-primary">Atualizar</button>
      </div>
    </div>
    <div class="col-2">
      <button type="button" class="btn btn-danger">Excluir</button>
    </div>
  </div>
</div>
```

Preparando...

Nome

Adicionar

Id

Nome

1

Jefferson

Atualizar

Excluir

2

Fabício

Atualizar

Excluir

3

Juliana

Atualizar

Excluir

4

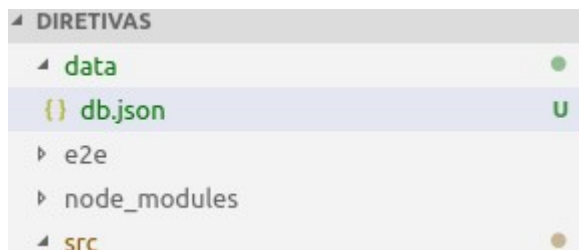
Roberta

Atualizar

Excluir

Json-server

- <https://github.com/typicode/json-server>
- Instalando:
 - `npm install -g json-server`
- Crie uma pasta “**data**”, no seguinte nível:



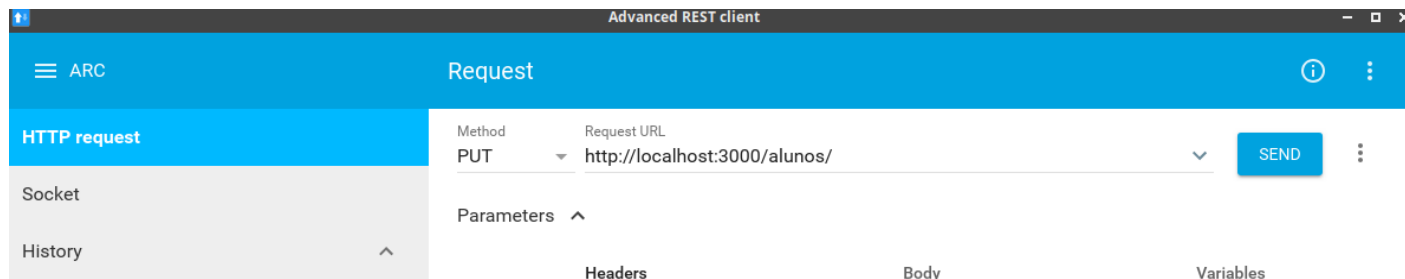
Json-server

- Dentro de “data”, crie o arquivo “db.json”:

```
{  
  "alunos": [  
    {"id": 1, "nome": "Jefferson"},  
    {"id": 2, "nome": "Fabrício"},  
    {"id": 3, "nome": "Juliana"},  
    {"id": 4, "nome": "Roberta"}  
  ]  
}
```

Json-server

- Inicie o json-server para que ele “escute” o arquivo criado:
 - `json-server --watch data/db.json`
 - Abra o seu navegador em “<http://localhost:3000/alunos/>”
 - Faça pequenos testes...
 - Dica: instale um cliente REST pro seu navegador.



Requisição GET

- Crie um serviço para fazer as requisições:
 - **ng n s aluno --spec=false**
 - Não esqueça de colocar em app.module:
 - providers: [AlunoService] *//caso não use o @Injectable*
 - imports: [HttpClientModule] de **import { HttpClientModule } from '@angular/common/http';**

Requisição GET

aluno.service.ts

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

```
@Injectable({  
  providedIn: 'root'  
})
```

```
export class AlunoService {
```

```
  constructor(private http: HttpClient) {}
```

```
  consultar(){  
    return this.http.get("http://localhost:3000/alunos");  
  }  
}
```

Injeção do serviço HTTP

Retorna um observável.

Requisição GET

- Em app.component

```
export class AppComponent implements OnInit{
```

```
  alunos:any[] =[]
```

```
  constructor(private alunoService:AlunoService){}
```

```
  ngOnInit(){  
    this.consultar();  
  }
```

```
  consultar(){  
    this.alunoService.consultar().subscribe((res : any[])=>{  
      this.alunos = res;  
    });  
  }  
}
```

Injeção do Serviço Aluno



Chamando o “consultar” do serviço.



Requisição POST

- Em aluno.service.ts, faça:

```
...  
adicionar(aluno:any){  
  return this.http.post("http://localhost:3000/alunos",aluno);  
}  
...
```

Requisição POST

- Em app.component.ts

```
...
adicionar(nome:string){
  this.alunoService.adicionar({nome:nome}).subscribe((res : any)=>{
    console.log(`Adicionado aluno ${res.nome} com id ${res.id}.`);
    this.consultar();
  });
}
...
```

Requisição DELETE

- Em aluno.service.ts

```
...  
excluir(id:number){  
  return this.http.delete(`http://localhost:3000/alunos/${id}`);  
}  
...
```

Template literals



Requisição DELETE

- Em app.component.ts

```
...  
excluir(id:number){  
  this.alunoService.excluir(id).subscribe((res : any)=>{  
    this.consultar();  
  });  
}  
...
```

Requisição PUT

- Em aluno.service.ts

```
...  
atualizar(aluno:any){  
  return this.http.put(`http://localhost:3000/alunos/${aluno.id}`,aluno);  
}  
...
```

Template literals



Requisição PUT

- Em app.component.ts

```
...
atualizar(aluno:any){
  this.alunoService.atualizar(aluno).subscribe((res : any)=>{
    console.log(res);
    this.consultar();
  });
}
...
```

Tratamento de erros...

- Force um erro em atualizar, no aluno.service.ts

```
...  
atualizar(aluno:any){  
  return this.http.put(`http://localhost:3000/aluno/${aluno.id}`,aluno);  
}  
...
```



Faltando o “s” em alunos, por exemplo. URL not Found.

Tratamento de erros...

- Em app.component.ts

```
atualizar(aluno:any){  
  this.alunoService.atualizar(aluno).subscribe(  
    (res : any)=>{  
      console.log(res);  
      this.consultar();  
    },  
    (error : any)=>{  
      console.log('oops',error)}  
  );  
}
```

Tratando o erro com a variável “errors”



Exercício

- Crie a mesma lógica deste slide mas agora para um produto como id, nome e preço.