

**UTFPR-UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

*Bacharelado em Engenharia de Software - 6o Período*

**DISCIPLINA:** *Programação Web 2 - IF67I*

**PROFESSOR:** *Willian Massami Watanabe*

## **Relatório**

# **Trabalho 1 da disciplina de Programação Web 2 Consumindo um serviço Rest com Json Server e Angula CLI.**

**Daniel Gonçalves da Silva RA: 1549898**

**Cornélio Procópio 2019**

# Sumário

|   |           |
|---|-----------|
| <b>Introdução</b>   | <b>3</b>  |
| <b>Contextualização</b>                                     | <b>3</b>  |
| <b>Desenvolvimento</b>                                      | <b>4</b>  |
| <b>Requisitos</b>   | <b>4</b>  |
| <b>Angular 8 e o HttpClient</b>                             | <b>4</b>  |
| <b>Benefícios do HttpClient</b>                             | <b>5</b>  |
| <b>Instalando o Angular CLI</b>                             | <b>6</b>  |
| <b>Criando projeto angular 8</b>                            | <b>6</b>  |
| <b>Rodando o projeto angular 8</b>                          | <b>6</b>  |
| <b>Criando uma API REST fake</b>                            | <b>7</b>  |
| <b>Configurando o HttpClient 8</b>                          |           |
| <b>Criando o model</b>                                      | <b>9</b>  |
| <b>Criando o serviço responsável pelas requisições http</b> | <b>9</b>  |
| <b>Usando o HttpClient</b>                                  | <b>9</b>  |
| <b>Entendendo nosso serviço</b>                             | <b>10</b> |
| <b>Usando nosso serviço</b>                                 | <b>11</b> |
| <b>Testando a aplicação</b>                                 | <b>11</b> |
| <b>Tela da aplicação</b>                                    | <b>12</b> |
| <b>Conclusão</b>  | <b>13</b> |
| <b>Referências:</b>   | <b>14</b> |

## **Introdução**

Este relatório foi produzido como trabalho para a disciplina de Programação Web 2 e objetiva a descrição de como aconteceu a implementação de um sistema web. O sistema Web proposto na disciplina consiste em uma aplicação que possa “consumir”, ou seja, acessar informações e realizar operações em dados vindos de uma outra aplicação. Este serviço de dados se comunica com a aplicação a ser desenvolvida por meio do serviço REST.

Iremos descrever neste relatório os passos realizados para desenvolver a aplicação proposta utilizando a tecnologia Angular.

## **Contextualização**

Nossa realidade está cercada de informações e de dados e o compartilhamento destas informações acontece a todo momento, assim podemos dizer que um dos grandes desafios que lidamos diariamente como desenvolvedores de softwares, é saber como lidar com toda essa massa de informação e de dados que nos bombardeiam a cada minuto na internet.

As aplicações REST estão “na moda”, pois, os benefícios de quem adere ao modelo de serviços são muitas, principalmente, quando o assunto está voltado a desenvolvimento Ágil e preocupações com qualidade e integração e desenvolvimento contínuo.

Partindo desta realidade vamos construir um relatório no formato passo-a-passo, ilustrando como criamos nossa aplicação de exemplo para

consumir dados vindo de uma API Rest “Fake”, fake porque a API estará sendo simulada pelo servidor Json Server, escolhemos esse formato de relatório com o intuito de que ela possa ser replicável por outras pessoas que queiram iniciar a sua primeira aplicação básica.

## **1 Desenvolvimento**

### **Requisitos**

Para iniciar a construção da nossa aplicação vamos criar um projeto angular, o projeto será um CRUD para nos ajudar de forma simples e básica a manipular funcionários, também criaremos uma API REST fake, essa api fake vai simular nosso back-end, então para isso vamos precisar de algumas tecnologias:

- Angular CLI (v8.15.0)
- Node.js (v10.9.0)
- Json-server

### **Angular 8 e o HttpClient**

O HttpClient é usado para fazer a comunicação entre cliente e servidor usando o protocolo HTTP. Ou seja, se você está querendo consumir dados de uma API externa o HttpClient facilitará essa comunicação, através métodos disponíveis como:

post()

get()

put()

delete()

patch()

request()

head()

jsonp()

options()

## **Benefícios do HttpClient**

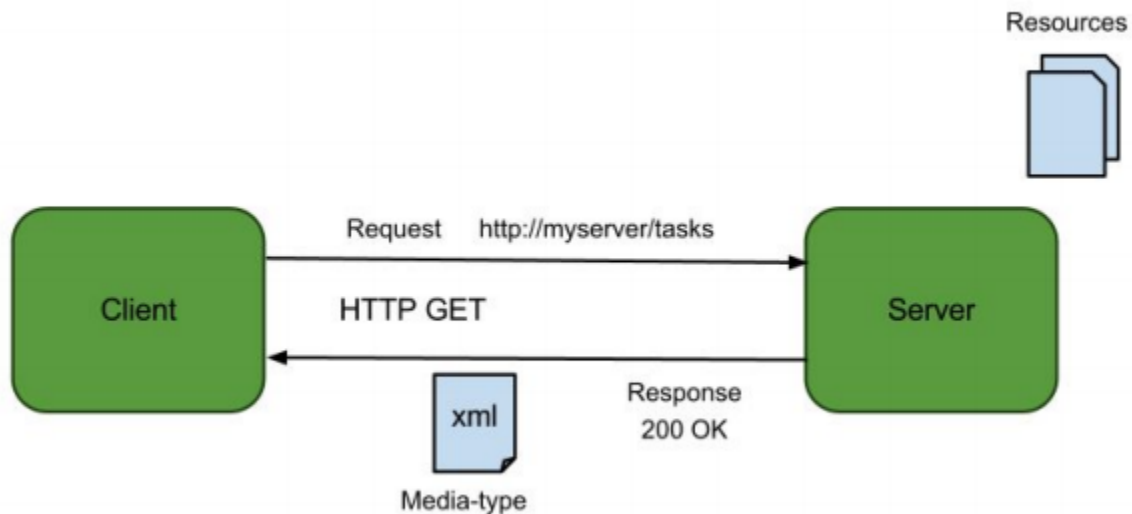
O HttpClient usa a interface XMLHttpRequest que também suporta a navegadores antigos, além de fácil de usar disponibiliza benefícios, como:

- Solicitações de request e response interceptadas
- Manipulação de erros simplificada
- Suporte a api Observable
- APIs e tratamentos de erros

Web Services, e estes, são soluções utilizadas para integração e comunicação entre sistemas. Como uma abstração da arquitetura HTTP, seu foco está direcionado para requisições e respostas entre cliente e servidor, onde: a) O cliente solicita um recurso disponível ao servidor através de informações, como o cabeçalho (header); e b) O servidor devolve ao solicitante uma resposta de acordo com o tratamento que deve realizar e seguindo informações recebidas por ele.

Essa maneira simplificada de comunicação, é uma das razões que tornam o

protocolo tão bem visto, deixando assim, todos os seus benefícios à tona.



## Instalando o Angular CLI

O projeto será criado usando o angular CLI, então para seguirmos com os passos, temos que tê-lo instalado em nossa máquina. Levando em consideração que temos o Node.js instalado, basta executar o seguinte comando:

```
sudo npm install -g @angular/cli
```

## Criando projeto angular 8

Com o angular CLI e o Node.js instalado, vamos criar o nosso projeto angular usando o CLI, para isso execute o comando abaixo no seu terminal:

```
ng new angular-http
```

Após executar o comando acima, o angular CLI fará algumas perguntas, na primeira digite "y" para criar o projeto com rota, em seguida escolha a opção CSS como formato de folha de estilo, isso criará o nosso projeto com os módulos NPM

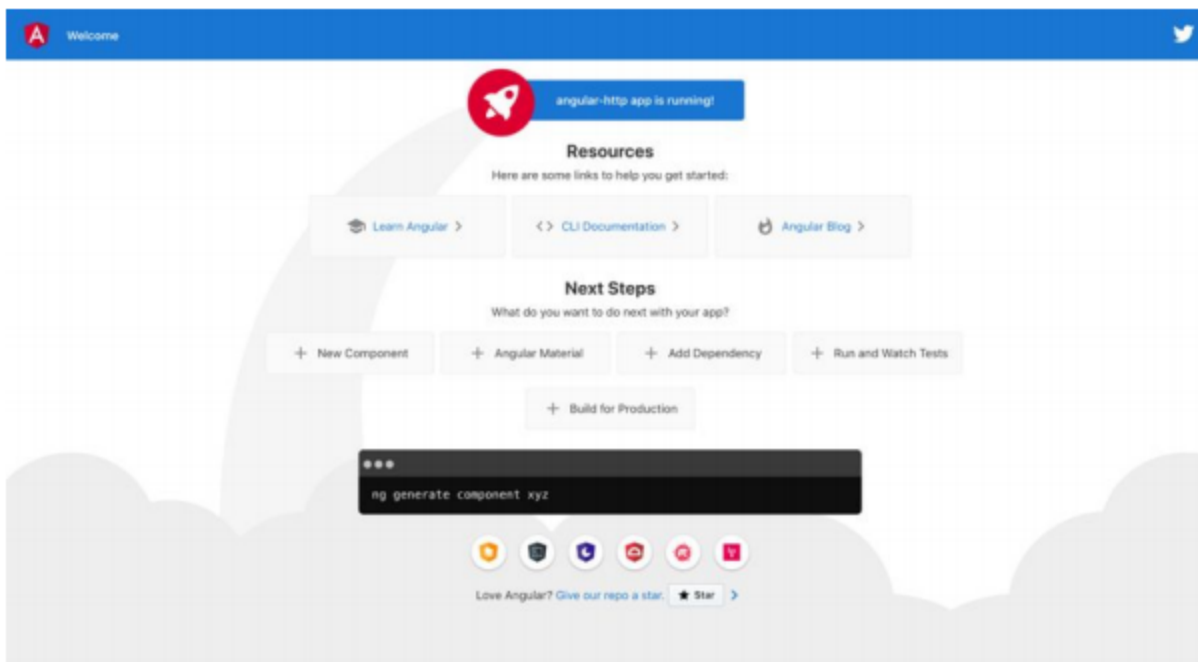
necessários.

## Rodando o projeto angular 8

Após criar o projeto, precisamos rodar e verificar se tudo foi criado corretamente, então entre na pasta do projeto e digite:

```
ng serve --open
```

O comando acima rodará o nosso projeto, e o parâmetro `--open` abrirá automaticamente o browser com o nosso projeto angular 8 em execução, como na imagem abaixo.



Tela do angular  
8m execução

## Criando uma API REST fake

Para simular o uso do HttpClient, precisamos de uma API REST, como o foco é o HttpClient não vamos nos preocupar em criar uma API REST, para isso

podemos usar o json-server, que faz uma API REST fake, assim focaremos no HttpClient.

Para mais detalhes sobre o json-server, podemos consultar seu github.

<<https://github.com/typicode/json-server>>

Para instalar o json-server, basta executar o seguinte comando:

```
sudo npm install -g json-server
```

Dentro do nosso projeto, vamos criar uma pasta chamada "data" dentro de "assets"

```
/src/assets/data
```

Agora crie um arquivo chamado db.json e jogue dentro da pasta "data" que acabamos de criar:

```
/src/assets/data/db.json
```

Vamos abrir o arquivo db.json e incluir o seguinte json:

O arquivo db.json para o json-server pode ser encontrado no github:

<[https://github.com/danielsilva83/trabalho\\_web2/src/assets/data/db.json](https://github.com/danielsilva83/trabalho_web2/src/assets/data/db.json)> Com esta

estrutura de pastas e com o arquivo db.json

Vamos rodar o json-server para simular nossa API REST, abra um novo terminal e

na raiz do projeto execute o seguinte comando:

```
json-server --watch src/assets/data/db.json
```



```
(→ angular-http git:(master) ✖ json-server --watch src/assets/data/db.json

\{^_^\}/ hi!

Loading src/assets/data/db.json
Done

Resources
http://localhost:3000/cars

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

json-server em execução

Tudo funcionará como na imagem acima, observe que nossa API REST fake está exposta no endereço: <http://localhost:3000>

## Configurando o HttpClient

Para usar o HttpClient, precisamos adicionar o módulo HttpClientModule no arquivo app.module.ts.

Para fazer isso, vamos abrir o arquivo app.module.ts

src/app/app.module.ts

Dentro de imports do decorator @NgModule, adicione o módulo HttpClientModule

Vamos nos atentar para não usar o pacote @angular/http, esse pacote estava

depreciado desde a versão 5 do angular, no angular 8 ele foi removido, o pacote

que utilizaremos é @angular/common/http.

Vamos aproveitar e adicionar o módulo FormsModule, esse módulo nos ajudará em nosso formulário para nossa tela de exemplo, porém não é necessário para o uso do HttpClient. O resultado do arquivo app.module.ts será:

Arquivo app.module.ts

## **Criando o model**

Vamos criar uma interface de modelo para os dados dos funcionários. Na raiz do projeto vamos executando o seguinte comando:

```
ng g interface models/fincionarios
```

O parâmetro g é uma abreviação de generate.

Será criado 2 arquivos dentro de uma pasta com o nome models, o arquivo funcionarios.spec.ts é um arquivo de teste, o segundo é o funcionarios.ts o model que representará nosso funcionário.

Para não criar o arquivo de teste, é só usar o parâmetro --skipTests=true

Agora vamos adicionar o seguinte conteúdo dentro de nosso model funcionarios.ts

Arquivo funcionarios.ts

## **Criando o serviço responsável pelas requisições http**

Vamos criar os métodos responsáveis pelas requisições http que faremos no json-server usando o HttpClient, mas antes devemos criar um arquivo service, no angular é recomendado criar services para os métodos que faz chamadas http.

Para criar o serviço, digite o seguinte comando:

```
ng g service services/funcionarios
```

O comando acima, criará um arquivo com o nome funcionarios.services.ts dentro da pasta services. Após criar nosso service, já podemos começar a utilizar o HttpClient

## Usando o HttpClient

Dentro do nosso service `funcionarios.services.ts`, vamos criar alguns métodos http utilizando o `HttpClient`.

Abra o arquivo `funcionarios.services.ts` dentro da pasta `services` e inclua o seguinte código:

Arquivo `funcionarios.service.ts`

## Entendendo nosso serviço

Injetamos o `HttpClient` e atribuímos a uma variável chamada `httpClient`, observe como é simples usar os métodos `get`, `post`, `put` e `delete` do nosso `httpClient`:

Chamando métodos http com `httpClient`

Note que passamos uma variável chamada “url” essa conterá o endereço

`http://localhost:3000/func.` disponibilizada pela API REST fake do `json-server`.

Adicionamos em nosso `HttpClient` um objeto do tipo `HttpOptions`, contendo nosso header através da classe `HttpHeaders`.

Alguns servidores podem exigir cabeçalhos para algumas operações como, `post`, `put` e `delete`, por exemplo um back-end onde as requisições dependem de um token de autorização.

Erros podem acontecer, por exemplo, um servidor back-end pode rejeitar nossa solicitação, ou um erro de rede no lado do cliente, esses erros produzem um objeto javascript do tipo `ErrorEvent`. Para manipularmos esses erros, criamos um método dentro do nosso serviço chamado `handleError`, esse método produzirá um `RxJS ErrorObservable`, caso nossas requisições contenha um erro, redirecionaremos para nossos serviços com uma mensagem mais amigável.

O angular recomenda que métodos de manipulação de erros deve ser nos services e não nos componentes.

Antes de chamar o método `handleError`, observe que adicionamos um operador chamado `retry`, esse é o mais simples da biblioteca `RxJS`, ele reexecutará a chamada em um numero específico de vezes caso aconteça um erro.

Após entendemos o uso do `httpClient`, concluímos que nosso serviço é responsável pelas seguintes ações:

- `getfunc()`: recupera todos os funcionários.
- `getCarById()`: recupera um funcionário específico pelo id.
- `saveCar()`: salva um funcionário.
- `updateCar()`: atualiza um funcionário.
- `deleteCar()`: exclui um funcionário específico pelo id.

## Usando nosso serviço

Agora vamos chamar nosso serviço através do component `app.component.ts`. Para isso vamos editar o component `app.component.ts` e adicionar o seguinte conteúdo:

Arquivo `app.component.ts`

Na linha 16 injetamos via construtor nosso serviço `func.ervice` e atribuímos a uma variável chamada `func.ervice`

Criamos um método `getfunc()`, esse será responsável por chamar a listagem de funcionários disponíveis através do nosso serviço `func.ervice`, incluímos no `onOnit()` para quando acessarmos `app.component.ts` ele nos trazer todos os funcionários, observe que logo após o método `getfunc()` do `func.ervice`, usamos o `subscribe`

```
getfunc() {
```

```
this.func.ervice.getfunc().subscribe((func.: Car[]) => {
```

```
this.func. = func.;  
});
```

} O subscribe é um dos operadores mais importantes do Observable da biblioteca RxJS, ele notificará assim que a resposta vier e for transformada em Json, nos retornando um array de funcionários.

No nosso exemplo, não esperamos retorno nas chamadas aos métodos saveCar(), updateCar() e deleteCar(), sendo assim, quando entrar no subscribe estamos limpando nosso formulário e consultando novamente o método getfunc. para listar todos os funcionários.

## **Testando a aplicação**

Modifiquei o HTML e CSS para ficar parecido com um projeto real, apenas para mostrar o uso do HttpClient em ação de forma simples e visual através de interações de tela.

Para isso, vamos incluir alguns trechos de código em nossa aplicação, primeiro vamos abrir o arquivo index.html e adicionamos o link CSS externo do bootstrap:

```
<link rel="stylesheet"
```

```
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
```

O código ficará:

Arquivo index.html

Agora vamos apagar todo o conteúdo do arquivo app.component.html e inserir o seguinte código:

No arquivo app.component.css vamos adicionar o seguinte CSS:

## Tela da aplicação

Já podemos notar o funcionamento do HttpClient através da listagem de funcionários que está no arquivo db.json. Assim que acessamos a tela da aplicação o método `onOnit()` foi disparado chamado o método `getfunc.()` que chama a listagem de funcionários do nosso serviço `func.ervice`.

Vamos adicionar um novo funcionário, uma Lamborghini Aventador

Adicionando um novo funcionário

Se abrir o terminal do json-server podemos verificar que requisições do tipo GET e POST foram realizadas:

POST /func. 201 45.079 ms — 97

GET /func. 200 4.980 ms - 407

A primeira foi uma requisição do tipo GET que nos retornou uma listagem contendo os funcionários, essa foi executada assim que acessamos a tela, ela foi chamada através do método `getfunc.()`, dentro do `onOnit()`, a segunda requisição foi do tipo POST, realizada através do método `savefunc.()`, invocada após clicar no botão

“Salvar”.

Se editarmos o modelo do funcionário para Lamborghini Huracan e sua cor para azul, podemos notar a requisição do tipo PUT e logo em seguida outra do tipo GET que é a listagem dos funcionários

PUT /func./4 200 5.340 ms — 99

GET /func. 200 3.185 ms — 409

E se excluirmos nossa Lamborghini Huracan, notamos outras requisições do tipo DELETE e outra do tipo GET .

DELETE /func./4 200 5.155 ms - 2

GET /func. 200 5.871 ms - 296

## **Conclusão**

Este trabalho é resultado de um projeto para a disciplina de Web 1 feito com dedicação e que exigiu análise, síntese e reflexão sobre todos os conceitos envolvidos durante o desenvolvimento. Um dos ganhos que consideramos importante foi o aprendizado de novos conhecimentos, principalmente sobre a integração entre back e front end.

O trabalho nos proporcionou a desenvolver a criatividade, incentivando a busca por encontrar soluções de softwares web, desta forma podemos entender na prática como integrar os sistemas.

Através desta aplicação modelo mostramos como consumir uma API REST usando o HttpClient , além de aprendemos como fazer requisições HTTP com os métodos GET, POST, PUT E DELETE, também aprendemos de forma básica como manipular erros e usar uma API REST fake com o json-server.

O código fonte do projeto está disponível no meu github

<[https://github.com/danielsilva83/trabalho\\_web2](https://github.com/danielsilva83/trabalho_web2)>

## **Referências:**

Documentação Angular. Disponível em: <<https://angular.io/docs>> acessado: em 12/2019

Documentação Nodejs. Disponível em: <<https://nodejs.org/pt-br/docs/>> acessado: em 12/2019

Criando aplicações Angular com Angular CLI. Disponível em: <<https://www.alura.com.br/artigos/criando-aplicacoes-angular-com-angular-cli>> acessado: em 12/2019

Angular CLI como criar artefatos do angular. Disponível em: <<https://www.devmedia.com.br/angular-cli-como-criar-artefatos-do-angular/38250>> acessado: em 12/2019