HTTP

Requisições HTTP

- HTTP é um protocolo de comunicação utilizado para transmissão de documentos hipermídia, como HTML
- Funciona basicamente através de requisições e respostas
- Contém basicamente três elementos:
  - Request line
  - Header
  - Body

Fluxo de uma comunicação HTTP

- O primeiro passo é a requisição, é executada toda vez que uma página é solicitada no browser:

```
Hypertext Transfer Protocol
  GET / HTTP/1.1\r\n
  Host: asus.edgebr.org\r\n
  Connection: keep-alive\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.75 Safari/537.36\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
  Accept-Encoding: gzip, deflate\r\n
  Accept-Language: en-US,en;q=0.9,pt-BR;q=0.8,pt;q=0.7\r\n
  Cookie: csrftoken=gObwHFAQxQA4AVVHckK8GDbW0deiElMuH4i2d0XQmmWVQTHyrRAxijrFypjzJDMU\r\n
  \r\n
  [Full request URI: http://asus.edgebr.org/]
  [HTTP request 1/2]
  [Response in frame: 2331]
  [Next request in frame: 2339]
```

Fluxo de uma comunicação HTTP

- Em seguida, o servidor responde a requisição, que acompanha o status, indicando se a requisição foi concluída com sucesso ou não, e o conteúdo:

```
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
  Date: Sun, 18 Oct 2020 05:11:38 GMT\r\n
  Server: Apache/2.4.29 (Ubuntu)\r\n
  Vary: Accept-Encoding\r\n
  Content-Encoding: gzip\r\n
  Content-Length: 409\r\n
  Keep-Alive: timeout=5, max=100\r\n
  Connection: Keep-Alive\r\n
  Content-Type: text/html;charset=UTF-8\r\n
  \r\n
  [HTTP response 1/2]
  [Time since request: 0.168405000 seconds]
  [Request in frame: 2317]
  [Next request in frame: 2339]
  [Next response in frame: 2348]
  Content-encoded entity body (gzip): 409 bytes -> 745 bytes
  File Data: 745 bytes
```

Verbos HTTP

- GET
  - Utilizado para leitura de dados
- POST
  - Utilizado para criação de novos dados
- PUT
  - Utilizado para atualização dos dados
- DELETE
  - Utilizado para exclusão de dados

APIs

- Uma API (*Application Programming Interface*) é uma abstração que permite acessar um serviço através de uma interface bem definida
- Exemplos de API:
  - Alguns elementos do próprio javascript (o *alert* é uma api para a caixa de diálogo)
  - APIs de sistemas operacionais: no windows, existem funções definidas na API Win32 para realizar diversos tipos de ações, não é necessário código específico para cada máquina
  - API REST: acesso a um serviço remoto através do HTTP, utilizando-se dos verbos
- Por exemplo, no caso de uma API REST, fazemos uma associação dos verbos HTTP mostrados anteriormente com ações a serem tomadas no nosso serviço

APIs REST

- É apenas boa prática, nada impede de se usar somente um verbo (GET, POST, PUT) para todas as operações, mas as *guidelines* recomendam que seja feito o uso conforme definido nas especificações do HTTP.

APIs REST

- Partindo do princípio que o verbo define a ação, podemos ter uma estrutura como a seguir, uma interface bem definida de como acessar os dados:

GET	http://events.asus.edgebr.org/api/tags/	Recupera todas as tags
POST	http://events.asus.edgebr.org/api/tags/	Cria uma nova tag
PUT	http://events.asus.edgebr.org/api/tags/id	Atualiza a tag de id <id>
DELETE	http://events.asus.edgebr.org/api/tags/id	Exclui a tag de id <id>

Angular

- Ao mudar de página no Angular, não é feita uma requisição HTTP, apenas o conteúdo da página é substituído via JavaScript
- Isso explica:
  - Angular demora um pouco mais pra carregar inicialmente (problema esse bem reduzido)
    - Payload JavaScript contendo todas as páginas
  - Browser não exibe indicação de carregamento ao alterar de página
  - Troca de páginas é muito rápida/instantânea
    - As páginas já vieram no payload javascript, não há necessidade de buscar no servidor

Angular

- Mas os dados não estão disponíveis, estes precisam ser carregados através de requisições HTTP
- Utilização do XmlHttpRequest, objeto do JavaScript que disponibiliza uma interface para fazer requisições HTTP diretamente do código *client-side*
    - XML: antigamente essas requisições eram muito usadas para carregar partes dinâmicas de algum site, por exemplo, ao atualizar um carrinho de compras, somente o HTML referente ao novo carrinho era trazido via XmlHttpRequest, não a página inteira
- Note que embora o nome, o XmlHttpRequest não tem limitações de formato, pode ser transferido qualquer tipo de dado

Angular

- Sendo assim, para obter dados no Angular, é necessário fazer requisições através do XmlHttpRequest
- O Angular disponibiliza uma interface sobre o XmlHttpRequest: o módulo Http
- O módulo Http foi criado pensando em APIs REST. O HttpClient, classe usada para fazer as requisições, contém todos os verbos HTTP como métodos

Formato de dados

- O HTTP não especifica qual o formato dos dados a serem enviados em seu *payload*, isso fica por conta da aplicação
- Por uma questão de compatibilidade com o JavaScript, o formato padrão usado pelo módulo Http e por outras bibliotecas é o JSON

JSON

- JavaScript Object Notation
- Uma maneira serializada de representar objetos JavaScript


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

JSON

- JavaScript traz em sua API os métodos JSON.parse() e JSON.stringify()
  - JSON.parse() lê uma string JSON e converte em um objeto JavaScript
  - JSON.stringify() recebe um objeto JavaScript e converte em uma string JSON
- Devido a essas funcionalidades nativas, normalmente é utilizado o JSON para transferir e enviar os dados da/para a API REST

Módulo Http

- Faz todas as conversões de JSON para objeto e vice-versa automaticamente, sendo assim, o desenvolvedor front-end apenas manipula objetos JSON

Módulo Http - Exemplo GET

```
getCourses(): Observable<Course[]> {
 return this.http.get('http://events.asus.edgebr.org/api/tags/');
}
```

Módulo Http - Exemplo GET

- `this.http` é um HttpClient injetado no serviço
- O método get() tem apenas um parâmetro obrigatório: a URL
- Note que o tipo de retorno é um Observable
  - A requisição não é feita no momento que essa função é definida, apenas na hora que algum outro código se increver neste Observable
  - Funciona da mesma forma que os parâmetros de roteamento que vimos anteriormente
- Podemos chamar subscribe() no componente e fornecer duas funções: a primeira é chamada caso a requisição ocorra com sucesso, a segunda caso ocorra um erro

Adendo: Observables

```
ngOnInit(): void {
 this.coursesService.getCourses()
    .pipe(take(1))
    .subscribe(
      data => this.courses = data,
      error => {
        this.error = true;
      }
    );
 this.courses[0] // ERRO. Por quê?
}
```

1ª função: caso a requisição der certo

2ª função: caso a requisição der errado

Adendo: Observables

- Ao chamar subscribe() os dados não são retornados imediatamente, o fluxo seria algo como:
  - Observable, estou me inscrevendo, quando terminar pode me chamar em uma dessas funções que estou passando (isso pode ser feito mais de uma vez)
  - O observable faz o seu trabalho, nesse caso uma requisição HTTP, e ao terminar, consulta quem está inscrito
  - O observable então chama a função que os inscritos passaram no primeiro passo, junto com os dados desejados

Observações:

- Questionário: https://forms.gle/UPCf2jK5aKYq1NeN6
- Código fonte: https://github.com/GabriellBP/Angular-course
- Dúvidas:
  - gabriel.pereira@edge.ufal.br
  - lucas.amorim@edge.ufal.br

Obrigado!