# TÓPICO 11 - INTEGRAÇÃO EXTERNA

Disciplina de Backend - Professor Ramon Venson - SATC 2025.2

# Web APIs

Uma Web API é um conjunto de interfaces de programação de aplicações (APIs) que são acessíveis por meio de solicitações HTTP.

É comum que a comunicação entre aplicações seja feita através de APIs.

# Integração com APIs

Um serviço web é comumente chamado a interagir com outros serviços dos quais possui dependência de dados e funcionalidades. Para isso, o serviço pode ser comunicar com outro usando a interface web, enviando e recebendo requisições no protocolo HTTP.

Ao usar um serviço de API externa, é importante estar atento a alguns detalhes:

- A API pode ser pública ou privada;
- A API pode ser gratuita ou paga;
- A API pode ser documentada ou não;
- A API pode exigir autenticação;

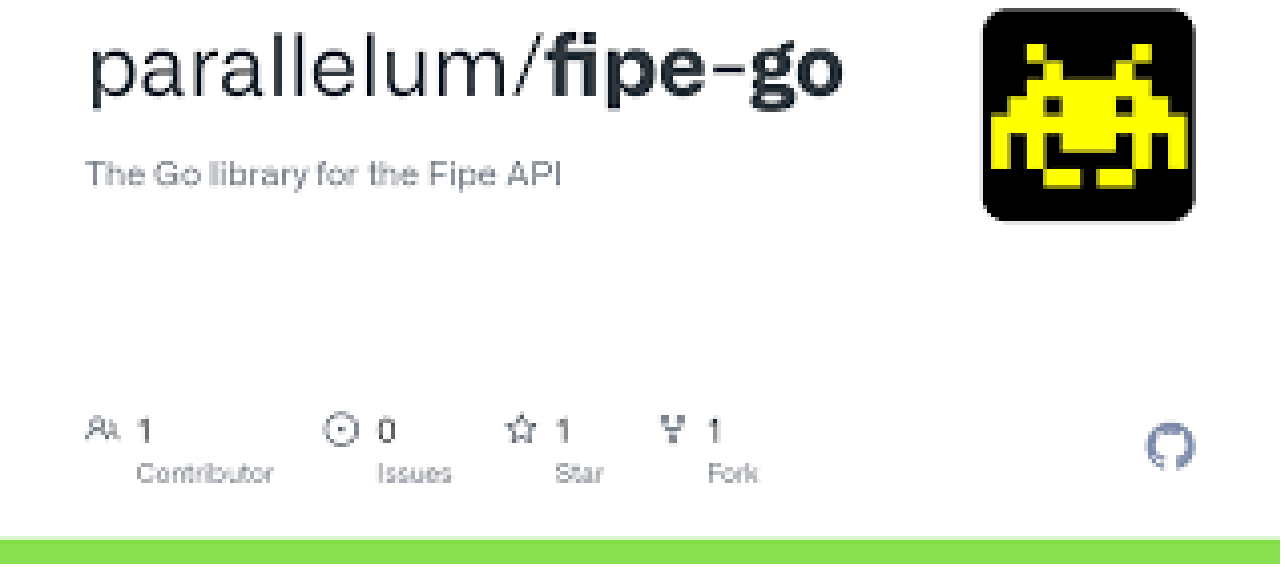
# Exemplos de APIs Famosas

Algumas APIs são bastante conhecidas e utilizadas por desenvolvedores, como:

- Google Maps - Mapas e geolocalização
- OpenWeather - Previsão do tempo
- Wikipedia - Informações sobre artigos da Wikipedia
- OpenAI - Chatbot e IA
- Gemini - Chatbot e IA
- GitHub - Gerenciamento de repositórios
- Last.fm - Informações sobre músicas
- GIPHY - Imagens e gifs

# Repositórios de APIs Públicas

- https://github.com/public-apis/public-apis
- https://publicapis.dev/
- https://publicapi.dev/




# Exemplo de Web API

Um serviço de Web API público e gratuito e que retorna informações de carros e valores da tabela FIPE é a Parallelum.

Essa API permite que você faça requisições HTTP para obter informações sobre marcas, modelos e veículos.

Esse serviço possui uma documentação que descreve as rotas disponíveis e os parâmetros que podem ser passados. Vamos utilizar a primeira versão como referência.

Essa versão possui os seguintes endpoints:

- `/marcas` : retorna uma lista de marcas de carros
- `/marcas/{id}/modelos` : retorna uma lista de modelos de carros de uma determinada marca
- `/marcas/{id}/modelos/{id}/anos` : retorna uma lista de anos de um modelo de carro

# Dados de Marcas

Ao acessar o primeiro endpoint, é retornado uma lista de marcas de carros:


```
[
  {
    "codigo": "1",
    "nome": "Acura"
  },
  {
    "codigo": "2",
    "nome": "Agrale"
  }
]
```


É importante saber que tipos de dados são esperados, pois isso pode influenciar na forma como os dados serão tratados.

# Mapeando o JSON

Para mapear o JSON, vamos criar uma classe que representa os dados da marca:

```
public class Marca {
  @JsonProperty("codigo")
  private String codigo;
  @JsonProperty("nome")
  private String nomeCompleto;
}
```

É importante incluir todos os atributos da classe. Também podemos utilizar a anotação `@JsonProperty` para mapear o nome do atributo do JSON para o nome do atributo da classe.

# Fazendo a requisição

Com a classe criada, podemos fazer a requisição para o endpoint. O objetivo será receber a lista de marcas da API externa (paralelum), repassando esses dados na requisição da nossa própria aplicação.

Usando o Spring, podemos utilizar diferentes métodos para gerar uma requisição HTTP:

- `RestClient`
- `RestTemplate`
- `WebClient`

Por questão de simplicidade, vamos utilizar o `RestClient`.

# RestClient

O `RestClient` é uma ferramenta integrada ao Spring Boot que permite realizar requisições HTTP para APIs REST.

Possui uma interface fluente que simplifica a construção de requisições HTTP.

> Essa classe foi introduzida no Spring Framework 6.0 e Spring Boot 3.0, substituindo o `RestTemplate` que está em desuso.

# Exemplo de Uso

```
@RestController
@RequestMapping("/api/marcas")
public class MarcasController {
  private RestClient cliente = RestClient.create("https://parallelum.com.br/fipe/api/v1/carros");

  @GetMapping
  public Marca[] buscaTodasAsMarcas() {
      return cliente.get().uri("/marcas").retrieve().body(Marca[].class);
    }
}
```

No exemplo acima, o `RestClient` é configurado para acessar a API de carros da Parallelum. O endereço passado é o endereço base da API.

Um mapeamento é feito em `buscaTodasMarcas`, que faz uma requisição GET para a API, passando o caminho `/marcas` e recebe uma lista de marcas como resposta.

É importante destacar alguns pontos no código anterior:

- O método `get()` é utilizado para realizar uma requisição GET.
- O método `uri()` é utilizado para adicionar um caminho ao endereço base.
- O método `retrieve()` é utilizado para obter a resposta da requisição.
- O método `body()` é utilizado para obter o corpo da resposta.
    - Os parâmetros em body são utilizados para definir o tipo de retorno esperado (no caso, uma lista).

# Manipulando a Resposta

Antes de retornar a lista de marcas, podemos manipular a resposta da requisição, de forma que apenas os nomes das marcas sejam retornados em uma lista. Para isso usaremos o método `map()` da classe `Stream`:

```
@GetMapping
public List<String> buscaTodasMarcas() {
  List<Marcas> listaDeMarcas = cliente.get().uri("/marcas").retrieve().body(ArrayList.class);
  List<String> nomesDasMarcas = listaDeMarcas.stream().map(Marca::getNome).toList();
}
```

## Mais sobre o `map()`

O método `map()` é uma operação de fluxo que permite transformar os elementos de um fluxo em outros elementos.

Pode parecer confuso de início, mas é muito útil para manipular dados. Uma outra forma de fazer o mesmo com a mesma lista seria:

```
List<String> nomesDasMarcas = new ArrayList<>();
for (Marca marca : listaDeMarcas) {
  nomesDasMarcas.add(marca.getNome());
}
```

# Bônus: Usando o `JsonNode`

O `RestClient` também permite que a resposta seja mapeada para um objeto `JsonNode`, que é uma representação genérica de um JSON. Isso pode ser útil quando não sabemos exatamente a estrutura do JSON que será retornado.

```
@GetMapping
public List<String> buscaTodasMarcas() {
  JsonNode resposta = cliente.get().uri("/marcas").retrieve().body(JsonNode.class);
  List<String> nomesDasMarcas = new ArrayList<>();
  for (JsonNode marca : resposta) {
    nomesDasMarcas.add(marca.get("nome").asText());
  }
  return nomesDasMarcas;
}
```

# Dicas Gerais

Podemos integrar facilmente outros serviços externos em nossa aplicação.

- Para isso, precisamos conhecer a documentação da API que queremos utilizar.
- Também precisamos conhecer os tipos de dados que a API espera e retorna.
- E por fim, precisamos saber como fazer requisições HTTP.
- Teste todas as requisições que você fizer para garantir que tudo está funcionando corretamente, tanto para o serviço externo quanto para a sua aplicação.

# Material de Apoio

- Documentação do Spring
- Guide to RestClient in Spring Boot