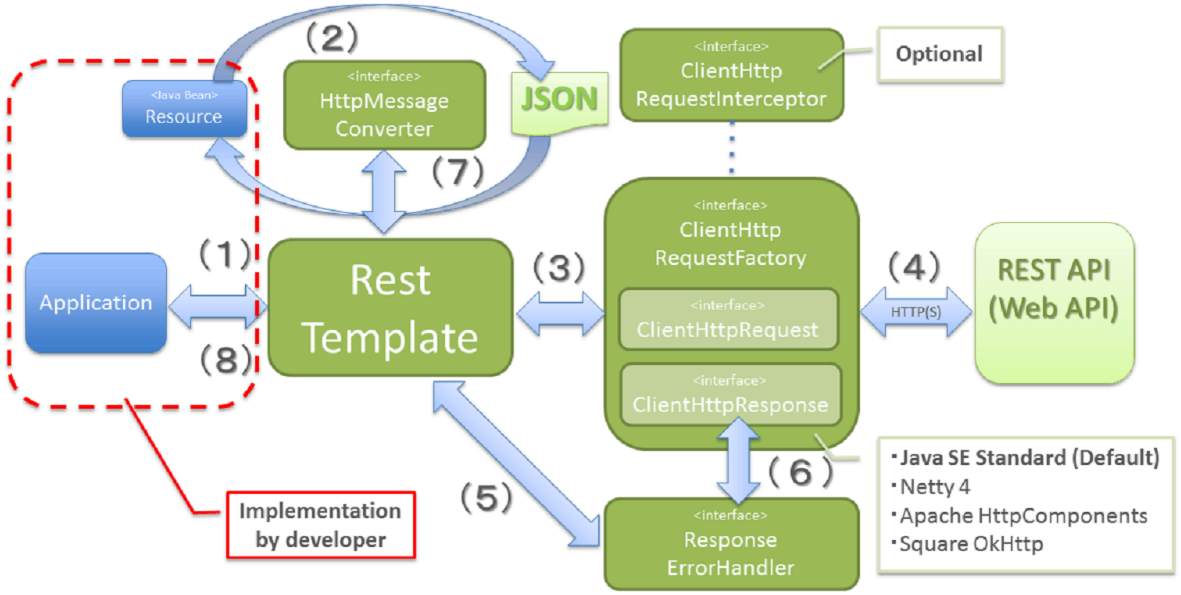
Aula 10 - Integrando APIs

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.

Repositórios de APIs Públicas

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



RestClient

- Introduzido no Spring 6.1 (2023)
- Permite realizar requisições com uma API fluente e funcional
- Converte dados além de validar e chamar outros componentes da `RestTemplate`

Criando o Controller

Tudo começa no controlador, então criaremos um controller básico por onde o usuário fará a requisição que chamará o serviço externo.

```
@RestController
public class CarrosController {
    @Autowired
    private IMarcasService marcasService;

    @GetMapping("/marcas")
    public ResponseEntity<Object> postagens() {
        return ResponseEntity.status(200).body(
          marcasService.getMarcas()
        );
    }
```

Interface

Vamos implementar também uma interface, pois a implementação das chamadas serviço externo pode ser feita de diferentes maneiras. Dessa forma isolamos nossa aplicação de mudanças na comunicação com outros servidores.

```
@RestController
public interface MarcasService {
    public Object getMarcas();
}
```

Service

Em seguida, vamos implementar o serviço que faz a comunicação com a API usando `RestClient`.

```
@Service
public class MarcasFipeServiceImpl implements CEPService {
    private RestClient client = RestClient.create("https://parallelum.com.br/fipe/api/v1/");

    @Override
    public CEP getMarcas() {
        String tipo = "carros";
        return client
                .get()
                .uri("{tipo}/marcas/", tipo)
                .retrieve()
                .body(Marca[].class);
    }
}
```

Model

Por fim, não esqueça de implementar um modelo (POJO) que possa mapear as respostas recebidas.

```
@Service
public class Marca {
    private int codigo;
    private String nome;
}
```

Não esqueça de adicionar o construtor e os métodos getter e setter.

Repare alguns detalhes da implementação do service:

- Inicializamos o `RestClient` com a URL base da API que queremos acessar. Não é necessário incluir o caminho completo aqui, visto que vamos completar com o método `.uri()`. Também não é necessário o uso de `@Autowired` aqui
- A variável `tipo` foi definida apenas para fins de exemplos, caso o controlador repasse o tipo informado na requisicao pelo usuário.
- `Marca[].class` representa um vetor de `Marca`, que é o conteúdo da resposta para essa requisição. Para apenas uma marca, utilize apenas `Marca.class`. Em outros casos use um POJO com o modelo da resposta completo para melhores resultados.

Extra: Adicionando um ID aos Objetos Criados

Imagine que você deseja retornar um objeto chamado Jogador na sua API. Cada jogador é único, e para facilitar a pesquisa dos jogadores, vamos adicionar o atributo id ao seu POJO;

```
public class Jogador {
  private UUID id = UUID.randomUUID();
  private String nome;
  private String posicao;
  private String clube;
}
```

Também usamos o `UUID.randomUUID()` para gerar um código único para o nosso objeto; Lembre-se de criar os getters para todos os atributos para que sejam acessíveis.

UUID

O UUID significa Identificador Universalmente Único e representa um valor de texto que tem como objetivo ser exclusivo para cada sistema e fração de tempo.

Cada UUID possui 16 bytes de comprimento e pode gerar números baseados no MAC do sistema, números aleatórios e na hora atual, provendo um número astronomicamente baixo de colisões (dois UUIDs iguais).

Cada versão do UUID possui particularidades na sua versão, por isso consulte antes de usar em ambientes de produção.

O que aprendemos hoje

- Integrar seu web service à uma outra API;
- Realizar requisições com o `RestClient`;
- Adicionar UUID aos objetos criados.