Consumo de API Externa

O consumo de APIs externas é uma prática essencial no desenvolvimento de software, permitindo que diferentes sistemas se comuniquem e compartilhem dados de forma eficiente. Uma API (Application Programming Interface) é um conjunto de regras e protocolos que permite a comunicação entre diferentes softwares. No desenvolvimento com Java, consumir uma API significa utilizar os serviços fornecidos por outra aplicação dentro do seu próprio projeto, facilitando a integração de funcionalidades como consultas de endereço, autenticação e pagamentos. Nesta aula, vamos entender como realizar o consumo de APIs no Java, explorando bibliotecas e frameworks que simplificam essa tarefa, e apresentando exemplos práticos para consolidar o aprendizado.

Conceito básico de Consumo de API

O consumo de APIs externas é uma prática essencial no desenvolvimento de software, permitindo que diferentes sistemas se comuniquem e compartilhem dados. Uma API (Application Programming Interface) é um conjunto de regras e protocolos que permite a comunicação entre diferentes softwares. No contexto do desenvolvimento, consumir uma API significa utilizar os serviços fornecidos por outra aplicação dentro do seu próprio projeto.

Para entender o consumo de APIs, considere um exemplo clássico: a API de consulta de CEP. Em um projeto que requer informações de endereço dos usuários, pode-se utilizar uma API externa para obter detalhes do CEP inserido pelo usuário, sem a necessidade de implementar essa funcionalidade do zero. Isso exemplifica como o consumo de APIs facilita o

desenvolvimento ao permitir a reutilização de serviços existentes, economizando tempo e recursos.

Existem diversas APIs públicas e gratuitas, bem como APIs pagas fornecidas por grandes empresas como Google e Amazon. APIs gratuitas geralmente oferecem funcionalidades básicas, enquanto APIs pagas podem fornecer serviços mais avançados e suporte técnico. É importante estar ciente das políticas de uso e custos associados ao consumo dessas APIs, para evitar surpresas durante o desenvolvimento.

O consumo de APIs é um componente fundamental em muitos softwares modernos, permitindo a integração de serviços como pagamentos, autenticação, notificações e muitos outros. No desenvolvimento com Java, esse consumo pode ser feito de maneira eficiente utilizando bibliotecas e frameworks específicos, que facilitam a implementação e a gestão das requisições e respostas das APIs.

APIs Falsas para Consumo

APIs falsas, ou fake APIs, são ferramentas essenciais para o desenvolvimento e teste de aplicações. Elas permitem simular o comportamento de uma API real sem a necessidade de uma infraestrutura completa. Isso é particularmente útil durante a fase de desenvolvimento, onde a integração com APIs reais pode não estar disponível ou ser muito custosa.

Fake APIs são utilizadas para validar a implementação de funcionalidades antes de realizar testes com dados reais. Elas permitem que desenvolvedores simulem a interação com uma API externa, garantindo que o código funcione conforme esperado. Isso é crucial para manter o fluxo de desenvolvimento sem interrupções, especialmente em projetos que dependem de múltiplos times trabalhando em paralelo.

Um exemplo popular de fake API é o JSON Placeholder, que fornece diversos endpoints para simular dados como usuários, posts e comentários. Utilizando uma fake API, desenvolvedores podem criar e testar funcionalidades sem a necessidade de configurar um servidor completo ou depender de APIs de terceiros ainda não prontas.

Para utilizar uma fake API, é necessário conhecer os endpoints disponíveis e as estruturas de dados que ela retorna. Por exemplo, ao simular uma lista de tarefas, a fake API pode retornar um conjunto de objetos com atributos como id, título e status de completude. Esses dados podem ser utilizados para testar a exibição de informações na interface do usuário e validar a lógica de negócios.

Bibliotecas e Frameworks para Consumo de APIs

Bibliotecas e frameworks são ferramentas essenciais para facilitar o consumo de APIs em projetos de software. Eles fornecem funcionalidades pré-construídas que simplificam a implementação de requisições HTTP, tratamento de respostas e gerenciamento de erros.

Uma biblioteca é um conjunto de módulos reutilizáveis que podem ser incorporados ao código conforme necessário. No contexto do consumo de APIs, bibliotecas como Apache HttpClient e OkHttp fornecem métodos prontos para realizar requisições HTTP, lidar com diferentes formatos de resposta (JSON, XML) e gerenciar conexões.

Um framework, por outro lado, é uma estrutura mais abrangente que define um padrão de desenvolvimento e oferece um conjunto de ferramentas integradas. No Java, o Spring Framework é um exemplo popular. Ele fornece diversas funcionalidades para a construção de aplicações web, incluindo o consumo de APIs. O Spring Boot, uma extensão do Spring Framework, simplifica ainda mais o desenvolvimento, configurando automaticamente componentes e fornecendo um ambiente pronto para o uso.

Consumir uma API no Projeto

Para consumir uma API em um projeto Java, é necessário seguir alguns passos básicos:

1. Conhecer a URL da API: Identificar os endpoints que serão consumidos.

2. Selecionar a Biblioteca: Escolher a biblioteca apropriada para realizar as requisições. No contexto do Spring, a biblioteca RestTemplate é amplamente utilizada.

3. Implementar o Cliente: Criar uma classe de serviço que utilizará a biblioteca para realizar as requisições HTTP e processar as respostas.

4. Configurar o Controlador: Expor endpoints no projeto que irão utilizar o serviço criado para consumir a API externa.

No exemplo a seguir, utilizaremos a API JSON Placeholder para consumir uma lista de tarefas (to-dos):

```java
// Classe de Serviço
@Service
public class TodoService {

    private final RestTemplate restTemplate;

    public TodoService(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    public List<Todo> getTodos() {
        String url = "https://jsonplaceholder.typicode.com/todos";
                ResponseEntity<Todo[]> response = restTemplate.getForEntity(url,
Todo[].class);
        return Arrays.asList(response.getBody());
    }
}
```

```
// Classe de Controlador
@RestController
@RequestMapping("/api/todos")
public class TodoController {

    private final TodoService todoService;

    public TodoController(TodoService todoService) {
        this.todoService = todoService;
    }

    @GetMapping
    public ResponseEntity<List<Todo>> getTodos() {


        List<Todo> todos = todoService.getTodos();
        return ResponseEntity.ok(todos);
    }
}
```

Neste exemplo, a classe `TodoService` utiliza o `RestTemplate` para realizar uma requisição GET ao endpoint da fake API. A resposta é processada e convertida em uma lista de objetos `Todo`. O `TodoController` expõe um endpoint `/api/todos` que utiliza o serviço para obter e retornar a lista de tarefas.

Essa abordagem modular facilita a manutenção e permite a substituição de fake APIs por APIs reais quando estiverem disponíveis, sem grandes alterações no código existente.

GitHub da Disciplina

Você pode acessar o repositório da disciplina no GitHub a partir do seguinte link:

https://github.com/FaculdadeDescomplica/Framework. Esse espaço é o seu portal para mergulhar fundo no universo da aprendizagem interativa. Nele,

você encontrará todos os códigos, além dos links para os arquivos e dados.

Conteúdo Bônus

Um excelente conteúdo bônus gratuito para alunos de graduação que estão aprendendo sobre Frameworks e Consumo de API Externa é o tutorial oficial do Spring Framework sobre consumo de APIs RESTful com Spring Boot.

Título: Consuming a RESTful Web Service

Plataforma: Spring.io

Este tutorial abrange os seguintes tópicos:

Introdução ao consumo de APIs RESTful.

Configuração inicial de um projeto Spring Boot para consumir APIs.

Utilização do `RestTemplate` e `WebClient` para realizar requisições a serviços REST externos.

Processamento de respostas de APIs e mapeamento para objetos Java.

Exemplos práticos de implementação, incluindo a configuração de cabeçalhos HTTP e autenticação.

Boas práticas para lidar com erros e exceções durante o consumo de APIs.

A documentação é clara, detalhada e rica em exemplos de código, facilitando o aprendizado e a aplicação prática dos conceitos no desenvolvimento de aplicações que consomem APIs externas.

Referência Bibliográfica

CARDOSO, L. da C. **Design de aplicativos**. Intersaberes: 2022

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 7.ed. Pearson: 2018.

JOÃO, B. do N. **Usabilidade e interface homem-máquina**. Pearson: 2017

LEAL, G. C. L. **Linguagem, programação e banco de dados**: guia prático de aprendizagem. Intersaberes: 2015.

MEDEIROS, L. F. de. **Banco de dados**: princípios e prática. Intersaberes: 2013.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**: implementação em SQL, PL/SQL e Oracle 11g. Pearson: 2013.

SETZER, V. W.; SILVA, F. S. C. **Bancos de dados**. Blucher: 2005.

VICCI, C. (Org.). **Banco de dados**. Pearson: 2014.

Ir para exercício