REST

Neste Módulo, vamos explorar os conceitos fundamentais de APIs RESTful. APIs RESTful (Representational State Transfer) são um padrão de arquitetura amplamente utilizado para a construção de serviços web. Veremos os princípios básicos, como recursos, métodos HTTP, e status codes. Entender esses conceitos é crucial para qualquer desenvolvedor que deseja trabalhar com APIs.

Conceitos Básicos de APIs RESTful

O que é uma API RESTful?

API RESTful é uma interface que permite a comunicação entre sistemas através de chamadas HTTP, seguindo os princípios da arquitetura REST.

Princípios Básicos de REST

1. Recursos: Tudo na REST é considerado um recurso, identificado por um URI (Uniform Resource Identifier).

2. Métodos HTTP: Os métodos HTTP principais são:

- **GET:** Recupera informações de um recurso.

- **POST:** Cria um novo recurso.

- **PUT:** Atualiza um recurso existente.

- **DELETE:** Exclui um recurso.

3. Status Codes: Indicadores do resultado de uma chamada HTTP. Exemplos:

- 200 OK: Requisição bem-sucedida.

- 201 Created: Recurso criado com sucesso.

- 400 Bad Request: Requisição inválida.

- 404 Not Found: Recurso não encontrado.

- 500 Internal Server Error: Erro no servidor.

Estrutura de uma URL RESTful

- Base URL: O endereço principal da API.

- Endpoint: A especificação do recurso. Ex: /users, /products.

Nesta aula, vamos aprender como consumir APIs REST em um aplicativo Flutter. Veremos como fazer requisições HTTP usando o pacote http e como lidar com as respostas da API. Aprender a consumir APIs é essencial para criar aplicativos que interagem com serviços web.

Consumindo APIs REST com Flutter

Adicionando Dependência ao pubspec.yaml

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.3
```

Fazendo uma Requisição GET

1. Importar o pacote http

```
import ‘package:http/http.dart’ as http;

import ‘dart:convert’;
```

2. Exemplo de Requisição GET

```
import ‘package:flutter/material.dart’;

import ‘package:http/http.dart’ as http;

import ‘dart:convert’;

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: HomeScreen(),

    );

  }

}
```

```
class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}
class _HomeScreenState extends State<HomeScreen> {
  Future<String> fetchData() async {
    final response = await http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts/1'));
    if (response.statusCode == 200) {
      return jsonDecode(response.body)['title'];
    } else {
      throw Exception('Falha ao carregar dados');
    }
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Consumindo API REST'),
```

```
      ),
      body: Center(
        child: FutureBuilder<String>(
          future: fetchData(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return CircularProgressIndicator();
            } else if (snapshot.hasError) {
              return Text('Erro: ${snapshot.error}');
            } else {
              return Text('Título: ${snapshot.data}');
            }
          },
        ),
      ),
    );
  }
}
```

Explicação: Este exemplo mostra como fazer uma requisição GET para um endpoint de teste, e exibir o resultado na interface do usuário.

Nesta aula, aprenderemos como lidar com dados JSON em Flutter. JSON (JavaScript Object Notation) é um formato leve de intercâmbio de dados, muito utilizado em APIs REST. Veremos como converter JSON em objetos Dart e vice-versa, e como estruturar nossos dados para fácil manipulação.

Tratamento de Dados JSON

Convertendo JSON para Objetos Dart

1. Modelo de Dados

```
class Post {
  final int userId;
  final int id;
  final String title;
  final String body;
    Post({required this.userId, required this.id, required this.title, required this.body});
  factory Post.fromJson(Map<String, dynamic> json) {
    return Post(
      userId: json['userId'],
      id: json['id'],
      title: json['title'],
```

```
      body: json['body'],
    );
  }
}
```

2. Exemplo de Conversão

```
Future<Post> fetchPost() async {
  final response = await http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts/1'));
  if (response.statusCode == 200) {
    return Post.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Falha ao carregar dados');
  }
}
```

Convertendo Objetos Dart para JSON

1. Método toJson

```
class Post {
  final int userId;
  final int id;
```

```
  final String title;

  final String body;

    Post({required this.userId, required this.id, required this.title, required this.body});

  factory Post.fromJson(Map<String, dynamic> json) {

    return Post(

      userId: json['userId'],

      id: json['id'],

      title: json['title'],

      body: json['body'],

    );

  }

  Map<String, dynamic> toJson() {

    return {

      'userId': userId,

      'id': id,

      'title': title,

      'body': body,

    };
```

```
  }
}
```

2. Exemplo de Conversão e Envio

```
Future<http.Response> createPost(Post post) async {
  final response = await http.post(
    Uri.parse('https://jsonplaceholder.typicode.com/posts'),
    headers: {'Content-Type': 'application/json; charset=UTF-8'},
    body: jsonEncode(post.toJson()),
  );
  if (response.statusCode == 201) {
    return response;
  } else {
    throw Exception('Falha ao criar post');
  }
}
```

Nesta aula, discutiremos as boas práticas e aspectos de segurança ao trabalhar com APIs REST. Veremos como proteger suas APIs, autenticação, validação de dados e práticas recomendadas para garantir a eficiência e segurança da comunicação entre cliente e servidor.

Boas Práticas e Segurança em APIs REST

Boas Práticas

1. Estrutura Clara de Endpoints

Use nomes claros e verbos HTTP apropriados para seus endpoints.

GET /api/v1/users

POST /api/v1/users

GET /api/v1/users/{id}

PUT /api/v1/users/{id}

DELETE /api/v1/users/{id}

2. Paginação e Limitação de Resultados

Implemente paginação para lidar com grandes volumes de dados.

GET /api/v1/users?page=1&limit=10

3. Documentação da API

Mantenha uma documentação clara e acessível para sua API, utilizando ferramentas como Swagger ou Postman.

Segurança

1. Autenticação e Autorização

Utilize OAuth 2.0, JWT ou outros métodos de autenticação robusta.

Authorization: Bearer {token}

2. Validação de Dados

Valide todos os dados de entrada no servidor para prevenir injeções de SQL e outros tipos de ataques.

```
{
  "userId": 1,
  "title": "Exemplo",
  "body": "Conteúdo do post"
}
```

3. CORS

Configure CORS (Cross-Origin Resource Sharing) para controlar o acesso a partir de diferentes origens.

```
Access-Control-Allow-Origin: *
```

4. HTTPS

Sempre utilize HTTPS para encriptar a comunicação entre cliente e servidor.

Esses exemplos e explicações fornecem uma base sólida para iniciantes em Flutter aprenderem a trabalhar com APIs RESTful. Para mais detalhes, consulte a documentação oficial do Flutter: Flutter Documentation.

Materiais Extras

Você pode realizar o download do arquivo contendo os materiais extras utilizados ao longo das aulas por meio do seguinte link: https://drive.google.com/file/d/1mg7lqMl8Pt2zl0rHIsFS0Qew00YN-sEX/view?usp=sharing.

Conteúdo Bônus

Para aprofundar seus conhecimentos em REST e compreender como as APIs RESTful funcionam, recomendamos o seguinte material:

Nome da obra: “O que é uma API RESTful?”

Autor: Amazon Web Services (AWS)

Plataforma: Site oficial da AWS

Este conteúdo gratuito oferece uma explicação detalhada sobre os princípios das APIs RESTful, seus componentes e como elas facilitam a comunicação entre serviços web. É uma leitura essencial para entender a aplicação prática de REST em arquiteturas modernas.

Referências Bibliográficas

BOYLESTAD, R. L.; NASHELSKY, L. Dispositivos Eletrônicos e Teoria de Circuitos. 11. ed. Pearson, 2013.

DEITEL, P. J.; DEITEL, H. M. Ajax, Rich Internet Applications e Desenvolvimento Web para Programadores. Pearson, 2008.

DUARTE, W. Delphi para Android e iOS: Desenvolvendo Aplicativos Móveis. Brasport, 2015.

FELIX, R.; SILVA, E. L. da. Arquitetura para Computação Móvel. 2. ed. Pearson, 2019.

LEE, V.; SCHNEIDER, H.; SCHELL, R. Aplicações Móveis: Arquitetura, Projeto e Desenvolvimento. Pearson, 2005.

MARINHO, A. L.; CRUZ, J. L. da. Desenvolvimento de Aplicações para Internet. 2. ed. Pearson, 2019.

MOLETTA, A. Você na Tela: Criação Audiovisual para a Internet. Summus, 2019.

SILVA, D. (Org.) Desenvolvimento para dispositivos móveis. Pearson, 2017.

Ir para exercício