



**FLUTTER**

**TEMA: CRUD COM WEBSERVICES**



# INTRODUÇÃO

# POKE APP

---

Crie um novo projeto **flutter**:

```
flutter create --org br.com.heiderlopes crud_app
```



**FLUTTER**

CRIANDO A ESTRUTURA DOS WIDGETS

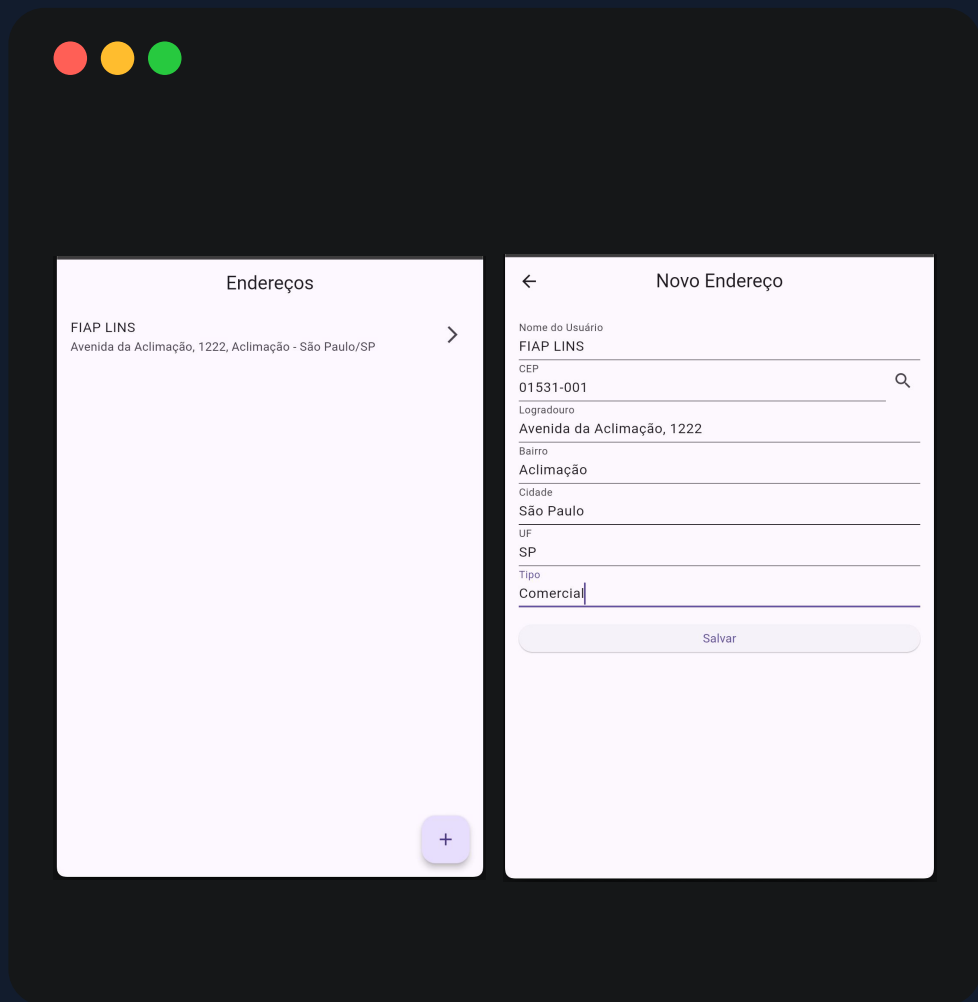


## CRUD COM WEBSERVICES

# VISÃO DO APP

Este é um app Flutter para gerenciar endereços que consome uma API REST. Funcionalidades principais:

- Lista endereços (Home).
- Formulário para criar/editar
- Consulta CEP para preencher automaticamente alguns dados.
- Excluir um endereço (com confirmação).



**FLUTTER**

ADICIONANDO LIB HTTP



## CRUD COM WEBSERVICES

# LIB HTTP

---

Abra o arquivo **pubspec.yaml** e adicione a lib **http**.

**http:** ^1.2.2



```
# Versions available, run flutter pub update  
dependencies:  
  flutter:  
    sdk: flutter  
  
  # The following adds the Cupertino Icons font to your application.  
  # Use with the CupertinoIcons class for iOS style icons.  
  cupertino_icons: ^1.0.8  
  http: ^1.2.2  
  
dev_dependencies:  
  flutter_test:  
    sdk: flutter
```

# FLUTTER

## CRIANDO MAIN





## CRUD COM WEBSERVICES

# MAIN


---

Crie o **main** do app.

**dart:convert:** usado para  
json.encode/json.decode.

**flutter/material.dart:** widgets Material  
(Scaffold, AppBar, etc).

**http:** pacote para fazer requisições  
HTTP.



```
import 'dart:convert';  
import 'package:flutter/material.dart';  
import 'package:http/http.dart' as http;  
  
//main() chama runApp() para inicializar o  
Flutter  
//e renderizar a árvore de widgets começando  
por AddressApp.  
  
void main() {  
  runApp(const AddressApp());  
}
```

**FLUTTER**

CRIANDO O WIDGET APP ADDRESS




## CRUD COM WEBSERVICES

# ADDRESS APP

---

**StatelessWidget:** porque é apenas configuração do app (tema, rota inicial).

**MaterialApp:** configura título, tema, página inicial (home) e remove o banner de debug com **debugShowCheckedModeBanner: false**.



```
class AddressApp extends StatelessWidget {  
  const AddressApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Address App',  
      theme: ThemeData(primarySwatch:  
Colors.blue),  
      home: const HomePage(),  
      debugShowCheckedModeBanner: false,  
    );  
  }  
}
```

**FLUTTER**

CRIANDO O API CONFIG




CRUD COM WEBSERVICES

## API CONFIG

---

Ter a base da API centralizada é uma boa prática. Se a URL mudar (dev → prod), altera-se apenas aqui.

Métodos ajudam a construir endpoints de maneira legível.



```
/// Centralização da URL da API

class ApiConfig {

    static const String baseUrl =

    "https://easy-address-app-15d989ca7c47.herokuapp.com";

    static String addresses() =>
    "$baseUrl/addresses";

    static String addressById(int id) =>
    "$baseUrl/addresses/$id";

    static String cep(String cep) =>
    "$baseUrl/cep/$cep";

}
```

**FLUTTER**

CRIANDO O MODELO DE ENDEREÇO



CRUD COM WEBSERVICES

## MODELO ADDRESS

---

Classe que representa um **endereço** do backend.

Ter modelos facilita tipagem, autocomplete e evita erros de string espalhados pelo código.




```
class Address {  
    final int id;  
    final String nomeUsuario;  
    final String cep;  
    final String logradouro;  
    final String bairro;  
    final String cidade;  
    final String uf;  
    final String tipo;  
    Address({  
        required this.id,  
        required this.nomeUsuario,  
        required this.cep,  
        required this.logradouro,  
        required this.bairro,  
        required this.cidade,  
        required this.uf,  
        required this.tipo,  
    });
```

CRUD COM WEBSERVICES

## MODELO ADDRESS

---

**fromJson:** transforma o  
Map<String,dynamic> (JSON) vindo  
da API em uma instância Address.



```
factory Address.fromJson(Map<String, dynamic> json) {  
  return Address(  
    id: json['id'],  
    nomeUsuario: json['nomeUsuario'],  
    cep: json['cep'],  
    logradouro: json['logradouro'],  
    bairro: json['bairro'],  
    cidade: json['cidade'],  
    uf: json['uf'],  
    tipo: json['tipo'],  
  );  
}
```



**FLUTTER**

CRIANDO A HOME PAGE




CRUD COM WEBSERVICES

## HOME PAGE

---

**HomePage** é um **StatefulWidget** porque precisa armazenar e atualizar a lista de endereços.




```
class HomePage extends StatefulWidget {  
  const HomePage ({super.key});  
  
  @override  
  State<HomePage> createState() => _HomePageState();  
}  
  
class _HomePageState extends State<HomePage> {  
  List<Address> addresses = [];  
  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

## HOME PAGE

---

**initState:** é chamado uma vez quando o widget é inserido na árvore; ideal para buscar dados iniciais.

**fetchAddresses():** carrega a lista da API.




```
@override
void initState() {
  super.initState();
  fetchAddresses();
}

Future<void> fetchAddresses() async {
  final response = await
http.get(Uri.parse(ApiConfig.addresses()));
  if (response.statusCode == 200) {
    final List<dynamic> data =
json.decode(response.body);
    setState(() {
      addresses = data.map((e) =>
Address.fromJson(e)).toList();
    });
  }
}
```

## HOME PAGE

---

**initState:** é chamado uma vez quando o widget é inserido na árvore; ideal para buscar dados iniciais.



```
@override  
void initState() {  
    super.initState();  
    fetchAddresses();  
}
```

## HOME PAGE

---

**fetchAddresses():** carrega a lista da API.

- Faz **GET** para /addresses.
- **json.decode** converte o body em lista dinâmica.
- **map(...).toList()** transforma cada item JSON em Address.
- **setState():** informa ao Flutter que a UI deve ser redesenhada com os novos dados.

```
Future<void> fetchAddresses() async {  
  final response = await  
http.get(Uri.parse(ApiConfig.addresses()));  
  if (response.statusCode == 200) {  
    final List<dynamic> data =  
json.decode(response.body);  
    setState(() {  
      addresses = data.map((e) =>  
Address.fromJson(e)).toList();  
    });  
  }  
}
```


## CRUD COM WEBSERVICES

# HOME PAGE

---

**Navigator.push:** abre uma nova tela (o formulário). **await** espera até o formulário fechar.

Ao voltar, chama **fetchAddresses()** para atualizar a listagem (**refetch**).



```
void goToForm({Address? address}) async {  
  await Navigator.push(  
    context,  
    MaterialPageRoute(builder: (_) =>  
      AddressFormPage(address: address)),  
  );  
  fetchAddresses();  
}
```

# HOME PAGE

## build() da Home

- **Scaffold** com **AppBar**.
- **ListView.builder**: eficiente para listas grandes, constrói apenas os itens visíveis.
- Cada item é um **ListTile** com **onTap** chamando **goToForm(address: addr)** para edição.
- **FloatingActionButton** abre o formulário em modo criação (sem address).



@override

```
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(title: const Text("Endereços")),  
    body: ListView.builder(  
      itemCount: addresses.length,  
      itemBuilder: (context, index) {  
        final addr = addresses[index];  
        return ListTile(  
          title: Text(addr.nomeUsuario),  
          subtitle: Text("${addr.logradouro}, ${addr.bairro} -  
${addr.cidade}/${addr.uf}"),  
          trailing: const Icon(Icons.arrow_forward_ios),  
          onTap: () => goToForm(address: addr),  
        ),  
      ),  
    floatingActionButton: FloatingActionButton(  
      onPressed: () => goToForm(),  
      child: const Icon(Icons.add),  
    ),  
  );  
}
```

**FLUTTER**

CRIANDO A ADDRESS FORM PAGE






CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Também é um **StatefulWidget** porque usa controllers e precisa reagir a mudanças.



```
class AddressFormPage extends StatefulWidget {  
  final Address? address;  
  
  const AddressFormPage({super.key, this.address});  
  
  @override  
  State<AddressFormPage> createState() =>  
    _AddressFormPageState();  
}  
  
class _AddressFormPageState extends  
  State<AddressFormPage> {  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Adicione o código ao lado ao  
**\_AddressFormPageState**.

Serve para validar e manipular o estado do  
**Form** (ex.: if  
(\_formKey.currentState!.validate())).


```
final _formKey = GlobalKey<FormState>();
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

**Controllers** controlam o texto dos **TextFormFields** (ler e escrever programaticamente).




```
final TextEditingController nomeController =  
TextEditingController();  
final TextEditingController cepController =  
TextEditingController();  
final TextEditingController logradouroController =  
TextEditingController();  
final TextEditingController bairroController =  
TextEditingController();  
final TextEditingController cidadeController =  
TextEditingController();  
final TextEditingController ufController =  
TextEditingController();  
final TextEditingController tipoController =  
TextEditingController();
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

**Prefill:** no `initState`, se `widget.address != null` então preenche os **controllers** com os valores do **endereço** (modo edição).




```
@override
void initState() {
  super.initState();
  if (widget.address != null) {
    nomeController.text = widget.address!.nomeUsuario;
    cepController.text = widget.address!.cep;
    logradouroController.text =
widget.address!.logradouro;
    bairroController.text = widget.address!.bairro;
    cidadeController.text = widget.address!.cidade;
    ufController.text = widget.address!.uf;
    tipoController.text = widget.address!.tipo;
  }
}
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

**Prefill:** no `initState`, se `widget.address != null` então preenche os **controllers** com os valores do **endereço** (modo edição).



```
@override
void initState() {
  super.initState();
  if (widget.address != null) {
    nomeController.text = widget.address!.nomeUsuario;
    cepController.text = widget.address!.cep;
    logradouroController.text =
widget.address!.logradouro;
    bairroController.text = widget.address!.bairro;
    cidadeController.text = widget.address!.cidade;
    ufController.text = widget.address!.uf;
    tipoController.text = widget.address!.tipo;
  }
}
```

## CRUD COM WEBSERVICES

# ADDRESS FORM PAGE

Monta um **Map** com os campos do formulário.

Se `widget.address == null` → **cria** (POST).

Senão → **atualiza** (PUT) usando id.

**mounted** é verificação para garantir que o **widget** ainda está na árvore antes de chamar **Navigator.pop**.

```
Future<void> saveAddress() async {
  if (_formKey.currentState!.validate()) {
    final Map<String, dynamic> data = {
      "nomeUsuario": nomeController.text,
      "cep": cepController.text,
      "logradouro": logradouroController.text,
      "bairro": bairroController.text,
      "cidade": cidadeController.text,
      "uf": ufController.text,
      "tipo": tipoController.text,
    };

    if (widget.address == null) {
      await http.post(
        Uri.parse(ApiConfig.addresses()),
        headers: {"Content-Type": "application/json"},
        body: json.encode(data),
      );
    } else {
```

## CRUD COM WEBSERVICES

# ADDRESS FORM PAGE

Monta um **Map** com os campos do formulário.

Se `widget.address == null` → **cria** (POST).

Senão → **atualiza** (PUT) usando id.

**mounted** é verificação para garantir que o **widget** ainda está na árvore antes de chamar **Navigator.pop**.

```
await http.put(
    Uri.parse(ApiConfig.addressById(widget.address!.id)),
    headers: {"Content-Type": "application/json"},
    body: json.encode(data),
);

if (mounted) Navigator.pop(context);
} else {
    ScaffoldMessenger.of(
        context,
    ).showSnackBar(SnackBar(content: Text('Falha ao salvar')));
}
```

## CRUD COM WEBSERVICES

# ADDRESS FORM PAGE

---

Deleta usando **DELETE** /addresses/{id}.

Se **sucesso**, volta para a lista.

Se **erro**, mostra **SnackBar**.

```
Future<void> deleteAddress() async {
    if (widget.address == null) return;

    final response = await http.delete(
        Uri.parse(ApiConfig.addressById(widget.address!.id)),
    );

    if (response.statusCode == 200) {
        if (mounted) Navigator.pop(context);
    } else {
        ScaffoldMessenger.of(
            context,
        ).showSnackBar(const SnackBar(content: Text("Erro ao
excluir endereço")));
    }
}
```



## CRUD COM WEBSERVICES

# ADDRESS FORM PAGE

Abre um **AlertDialog** e retorna **true/false** dependendo da escolha do usuário.

É usado para **confirmar exclusão** antes de chamar **deleteAddress()**.



```
Future<bool> showConfirmDialog(BuildContext context) async {  
  return await showDialog<bool>(  
    context: context,  
    builder:  
      (ctx) => AlertDialog(  
        title: const Text("Confirmação"),  
        content: const Text("Deseja realmente excluir  
este endereço?"),  
        actions: [  
          TextButton(  
            onPressed: () => Navigator.pop(ctx, false),  
            child: const Text("Cancelar"),  
          ),  
          TextButton(  
            onPressed: () => Navigator.pop(ctx, true),  
            child: const Text("Excluir"),  
          ),  
        ],  
      ),  
  ) ??  
  false; // se o usuário fechar o diálogo sem escolher  
}
```

CRUD COM WEBSERVICES

# ADDRESS FORM PAGE

**Scaffold** com **Form** e **ListView** para os campos dos formulário.

```
@override
Widget build(BuildContext context) {
  final isEditing = widget.address != null;

  return Scaffold(
    appBar: AppBar(
      title: Text(isEditing ? "Editar Endereço" : "Novo
Endereço"),
    ),
    body: Padding(
      padding: const EdgeInsets.all(16),
      child: Form(
        key: _formKey,
        child: ListView(
          children: [
            // CAMPOS DO FORMULARIO
          ],
        ),
      ),
    ),
  );
}
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Campo do nome com validação para campos vazios.

```
TextFormField(  
  controller: nomeController,  
  decoration: const InputDecoration(labelText: "Nome do  
Usuário"),  
  validator:  
    (v) => (v == null || v.isEmpty) ? 'Preencha o nome'  
    : null,  
)
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Linha com o campo do cep e o botão para realizar a busca.

```
Row(  
  children: [  
    Expanded(  
      child: TextFormField(  
        controller: cepController,  
        decoration: const  
InputDecoration(labelText: "CEP"),  
      ),  
    ),  
    IconButton(  
      icon: const Icon(Icons.search),  
      onPressed: fetchCep,  
    ),  
  ],  
)
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Adicione os campos de **Logradouro**, **Bairro** e **Cidade**.



```
TextFormField(  
  controller: logradouroController,  
  decoration: const InputDecoration(labelText:  
    "Logradouro"),  
  validator: (v) => (v == null || v.isEmpty)? 'Preencha o  
    logradouro': null,),
```

```
TextFormField(  
  controller: bairroController,  
  decoration: const InputDecoration(labelText: "Bairro"),  
  validator: (v) => (v == null || v.isEmpty) ? 'Preencha  
    o bairro' : null,),
```

```
TextFormField(  
  controller: cidadeController,  
  decoration: const InputDecoration(labelText: "Cidade"),  
  validator: (v) => (v == null || v.isEmpty) ? 'Preencha a  
    cidade' : null,),
```

## ADDRESS FORM PAGE

---

Adicione os campos de **UF** e **Tipo**.


```
TextFormField(  
  controller: ufController,  
  decoration: const InputDecoration(labelText: "UF"),  
  validator: (v) => (v == null || v.isEmpty) ? 'Preencha  
o UF' : null,),  
  
TextFormField(  
  controller: tipoController,  
  decoration: const InputDecoration(labelText: "Tipo"),  
  validator:  
    (v) => (v == null || v.isEmpty) ? 'Preencha o tipo'  
    : null,  
),
```

CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Adicione o botão para **Salvar**



```
const SizedBox(height: 20),  
ElevatedButton(  
  onPressed: saveAddress,  
  child: const Text("Salvar"),  
) ,
```


CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Adicione o botão para **Excluir**.

Ele só deverá ser mostrado caso seja edição.



```
if (isEditing) ...[
  const SizedBox(height: 10),
  ElevatedButton(
    style: ElevatedButton.styleFrom(backgroundColor:
Colors.red),
    onPressed: () async {
      final confirm = await showConfirmDialog(context);
      if (confirm == true) {
        deleteAddress();
      }
    },
    child: const Text("Excluir"),
  ),
],
```




CRUD COM WEBSERVICES

## ADDRESS FORM PAGE

---

Após o build adicione o **dispose**.

É importante sempre liberar os controllers ao finalizar o widget.



```
@override
void dispose() {
    nomeController.dispose();
    cepController.dispose();
    logradouroController.dispose();
    bairroController.dispose();
    cidadeController.dispose();
    ufController.dispose();
    tipoController.dispose();
    super.dispose();
}
```

**FLUTTER**

REVISANDO CONTEÚDO IMPORTANTE



## REVISÃO DE CONTEÚDO

---

**Widget:** elemento da UI (botão, texto, tela).

**StatelessWidget vs StatefulWidget:** use **StatefulWidget** quando precisa manter estado mutável (lista, loading, texto digitado).

**Async / await / Future:** operações de rede são assíncronas; usamos **await** para esperar o resultado sem bloquear a UI.

**setState():** quando você altera dados que afetam a interface, chame **setState()** para redesenhar.

**TextEditingController:** manipula o conteúdo de campos de texto programaticamente.

**Navigator:** gerencia pilha de telas; push adiciona, pop retorna.

**mounted:** booleano que indica se o **State** ainda faz parte da árvore; sempre checar antes de atualizar UI após await.

# FLUTTER

## EXERCÍCIO



## CRUD COM WEBSERVICES

# EXERCÍCIO

Adicione **Loading** e **tratamento de erro** no aplicativo.

Segue ao lado um exemplo. Faça as alterações necessárias.



```
Future<void> saveAddress() async {
  if (!_formKey.currentState!.validate()) return;
  setState(() => _loading = true);
  final data = { /* ... */ };
  try {
    final resp = widget.address == null
      ? await http.post(Uri.parse(ApiConfig.addresses()), headers:
        {...}, body: json.encode(data))
      : await
        http.put(Uri.parse(ApiConfig.addressById(widget.address!.id)),
          headers: {...}, body: json.encode(data));

    if (resp.statusCode == 200 || resp.statusCode == 201) {
      Navigator.pop(context, true);
    } else {
      throw Exception('Status ${resp.statusCode}');
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content:
      Text('Erro ao salvar: $e')));
  } finally {
    if (mounted) setState(() => _loading = false);
  }
}
```

# OBRIGADO



heider-lobes-a06b2869

Copyright © 2025 | Professor (a) Heider Lopes  
Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.