

Requisições HTTP - Parte 2

Usabilidade, Desenvolv. Web, Mobile e Jogos

Prof. Gustavo Custodio
gustavo.custodio@ulife.com.br



Requisições HTTP - Parte 2

Resolução Exercício

Resolução Exercício

- Modifique o aplicativo criado:
 - Adicione uma tela que é mostrada quando clicamos em uma das pizzas no ListView.
 - Essa tela deve mostrar uma imagem da pizza clicada, sua descrição e seu preço.

Resolução Exercício

- Crie um arquivo chamado `telapizza.dart` contendo o StatelessWidget `TelaPizza`.

```
1  class TelaPizza extends StatelessWidget {  
2    Pizza pizza;  
3  
4    // Esse Widget recebe como parâmetro a pizza mostrada  
5    TelaPizza(this.pizza);  
6  
7    Widget build(BuildContext context) {  
8      return Scaffold(  
9        appBar: AppBar(  
10         title: Text(pizza.nomePizza),  
11       ),  
12       body: Container(),  
13     );  
14   }  
15 }
```

Resolução Exercício

- Agora adicionamos as quatro imagens disponíveis em anexo **no site da disciplina**.
- No arquivo `pubspec.yaml`:

```
1  assets:  
2    - images/alho.png  
3    - images/mussarela.png  
4    - images/atum.png  
5    - images/calabresa.png
```

Resolução Exercício

- Criamos um *widget* para mostrar a tela com as informações da pizza:

```
1      body: infoPizza(context),
2      ...
3
4      Widget infoPizza(BuildContext context) {
5          return Column(
6              children: [
7                  //_imagem(context),
8                  //_descricao(context),
9                  //_preco(context),
10             ],
11         );
12     }
```

Resolução Exercício

```
1  Widget _imagem(BuildContext context) {  
2    return Container(  
3      width: MediaQuery.of(context).size.width,  
4      child: Image.asset('${pizza.urlImagem}'),  
5    );  
6  }
```

Resolução Exercício

```
1  Widget _descricao(BuildContext context) {  
2      return Padding(  
3          padding: EdgeInsets.only(top: 50, bottom: 50),  
4          child: Text(  
5              '${pizza.descricao}',  
6              style: TextStyle(  
7                  fontSize: 25,  
8                  fontFamily: 'Courier',  
9              ),  
10         ),  
11     );  
12 }
```


Resolução Exercício

```
1  Widget _preco(BuildContext context) {  
2      return Text(  
3          'R\$ ${pizza.preco.toStringAsFixed(2)}',  
4          style: TextStyle(  
5              fontSize: 40,  
6              fontFamily: 'Courier',  
7              fontWeight: FontWeight.bold,  
8          ),  
9      );  
10 }
```


Resolução Exercício

- Por fim, adicionamos a funcionalidade de navegar para a tela da pizza utilizando a propriedade onTap do ListTile.
- No arquivo cardapio.dart, altere o conteúdo do ListTile:

```
1   ListTile(  
2     ...  
3     onTap: () {  
4       Navigator.of(context).push(  
5         MaterialPageRoute(  
6           builder: (_) => TelaPizza(pizza),  
7         ),  
8       );  
9     },  
10  ),
```

Resolução Exercício





Requisições HTTP POST

Requisicoes hTTP ~ Parte 2

Requisições POST

- Agora vamos realizar requisições do tipo POST em *web services*.
- Requisições HTTP do tipo POST são adequadas para quando temos um *web service* que não só fornece dados,
 - mas também permite que o usuário realize alterações nos dados armazenados.

Requisições POST

- Requisições do tipo POST permitem enviar mais dados do que requisições do tipo GET.
 - O limite de tamanho das requisições POST é definido pela configuração do servidor).
 - Dados em requisições do tipo POST são armazenados no corpo do formulário ao invés da URL.
 - Mais discreto (para dados sensíveis).

Endpoint POST

- Vamos voltar ao *MockLab* e criar um *endpoint* para nossas requisições do tipo POST.

The screenshot displays the MockLab interface with two panels. The left panel shows a list of existing endpoints: 'AddPizza' (POST /addpizza), 'Pizzas' (GET /pizzalist), and 'GET a JSON resource' (GET /json/1). The right panel is the configuration page for the 'AddPizza' endpoint. It shows the method 'POST' and the path '/addpizza'. Under the 'Response' section, the status is set to '201'. The 'Body' tab is selected, showing a JSON response:

```
{ "mensagem": "Pizza adicionada com sucesso." }
```

. The 'Name' field is set to 'AddPizza'.

AddPizza

POST /addpizza

201

```
{ "mensagem": "Pizza adicionada com sucesso"
```

Endpoint POST

- Se a requisição do tipo POST for bem sucedida, é retornado uma mensagem em JSON:
 - {"mensagem": "Pizza adicionada com sucesso."}
 - Lembrando que esse *endpoint* apenas recebe requisições do tipo POST,
 - ele não trabalha com nenhum banco de dados.
 - não estamos adicionando uma pizza nova de verdade.

Formulário para POST

- Vamos adicionar o formulário para inserir informações da pizza.
 - Crie um arquivo chamado `formulario_post.dart`
 - Dentro dele, um `StatefulWidget` chamado `FormularioPost`:

Formulário para POST



← Adicionar Pizza DEBUG

Id

Nome

Descrição

Preço

URL da imagem

Adicionar

Formulário para POST

```
1 import 'http_helper.dart';
2 import 'model/pizza.dart';
3
4 class FormularioPost extends StatefulWidget {
5   const FormularioPost({super.key});
6
7   @override
8   State<FormularioPost> createState() => _FormularioPostState();
9 }
10
11 class _FormularioPostState extends State<FormularioPost> {
12   @override
13   Widget build(BuildContext context) {
14     return Scaffold(
15       appBar: AppBar(
16         title: Text("Adicionar Pizza"),
17       ),
18       body: Container(),
19     );
20   }
21 }
```

Formulário para POST

- Crie um método que será utilizado para criar campos de texto.

```
1  Widget _criarTextFormField(  
2      BuildContext context, TextEditingController controller,  
3      String label)  
4  {  
5      // Cria um campo de texto com o Controller correspondente  
6      return TextFormField(  
7          controller: controller,  
8          decoration: InputDecoration(labelText: label),  
9      );  
10 }
```

- Ele associa o *Controller* correspondente ao campo de texto.

Formulário para POST

- Criamos os Controllers correspondentes:

```
1    ...
2    class _FormularioPostState extends State<FormularioPost> {
3
4        final controllerId = TextEditingController();
5        final controllerNome = TextEditingController();
6        final controllerPreco = TextEditingController();
7        final controllerDescricao = TextEditingController();
8        final controllerImagem = TextEditingController();
9    ...
```

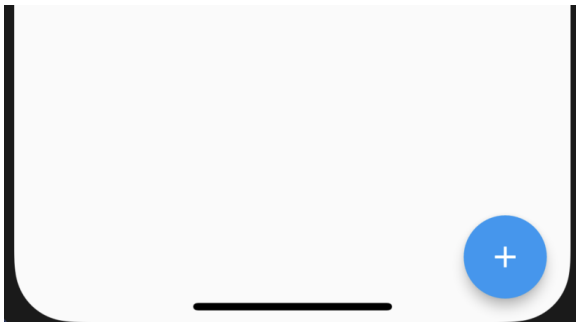
Formulário para POST

- E alteramos o método build:

```
1      body: Padding(  
2        padding: EdgeInsets.all(20),  
3        child: Column(  
4          mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
5          children: [  
6            _criarTextFormField(context, controllerId, "Id"),  
7            _criarTextFormField(context, controllerNome, "Nome"),  
8            _criarTextFormField(context, controllerDescricao, "Descrição"),  
9            _criarTextFormField(context, controllerPreco, "Preço"),  
10           _criarTextFormField(context, controllerImagem, "URL da imagem"),  
11           ElevatedButton(  
12             onPressed: () {  
13               //enviarForm();  
14             },  
15             child: Text('Adicionar'),  
16           ),  
17         ],  
18       ),  
19     ),
```

Floating Action Button

- Vamos adicionar um FAB (*Floating Action Button*).
 - Ele vai ser responsável por **abrir o formulário onde adicionamos** uma nova pizza.



Floating Action Button

- No método build da classe Cardapio (não esqueça de importar o `formulario_post.dart`):

```
1      ...
2      appBar: AppBar( title: Text("Pizzaria")),
3
4      floatingActionButton: FloatingActionButton(
5        onPressed: (){
6          Navigator.of(context).push(
7            MaterialPageRoute(
8              builder: (_) => FormularioPost(),
9            ));
10       },
11       child: Icon(Icons.add),
12     ),
13
```


Construindo a Requisição POST

- Agora conseguimos abrir o formulário.
- Vamos adicionar um método toJson dentro do model Pizza.
 - Ele tem a função oposto do fromJson.
 - Converte os dados de um objeto para o formato JSON.

Construindo a Requisição POST

```
1  Map<String, dynamic> toJson() {  
2      return {  
3          "id": id,  
4          "nomePizza": nomePizza,  
5          "descricao": descricao,  
6          "preco": preco,  
7          "urlImagem": urlImagem,  
8      };  
9  }
```

Construindo a Requisição POST

- Agora adicionamos um método para enviar os dados da pizza em formato JSON para nosso *endpoint*.
 - Modifique a classe HttpHelper:

```
1    final String endpointPost = "addpizza";
2
3    Future<String> postPizza(Pizza pizza) async {
4        String post = json.encode(pizza.toJson());
5        Uri url = Uri.https(dominio, endpointPost);
6        http.Response r = await http.post(
7            url, body: post
8        );
9        return r.body;
10    }
```

Construindo a Requisição POST

- Adicione o método `enviarForm` dentro da classe `_FormularioPostState`.
 - Ele é responsável por converter os dados do formulário pra o formato JSON
 - e enviar uma resposta para o usuário contendo o retorno do POST.

Construindo a Requisição POST

```
1 Future enviarForm() async {
2     HttpHelper helper = HttpHelper();
3     int id = int.parse(controllerId.text);
4     String nomePizza = controllerNome.text;
5     String descricao = controllerDescricao.text;
6     double preco = double.parse(controllerPreco.text);
7     String urlImagem = controllerImagem.text;
8
9     Pizza pizza = Pizza(id, nomePizza, descricao, preco, urlImagem);
10    String resultado = await helper.postPizza(pizza);
11
12    // Mostrar caixa de diálogo com o resultado do POST
13 }
```

Construindo a Requisição POST

- Ainda no `enviarForm`, adicione a caixa de diálogo, mostrando um *feedback* para o usuário.

```
1  showDialog(  
2    context: context,  
3    builder: (BuildContext context) {  
4      return AlertDialog(  
5        title: Text("POST realizado"),  
6        content: Text(resultado),  
7        actions: <Widget>[  
8          ElevatedButton(  
9            child: Text("Ok"),  
10           onPressed: () {Navigator.of(context).pop(); }  
11         ),  
12       ],  
13     );  
14   },  
15 );
```

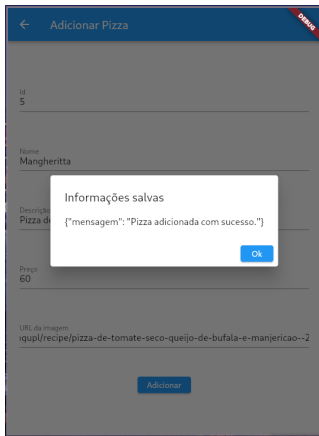
Construindo a Requisição POST

- Vamos adicionar uma imagem nova por meio de uma URL.
 - Então verificamos se a imagem existe na rede e, caso não, adicionamos por asset.

```
1  ...
2  width: MediaQuery.of(context).size.width,
3  child: carregarImagem('${pizza.urlImagem}'),
4  );
5  }
6
7  Image carregarImagem(String path) {
8    return Image.network(
9      path,
10     errorBuilder:
11       (BuildContext context, Object exception, StackTrace? stackTrace) {
12         return Image.asset(path);
13       },
14     );
15  }
```

Construindo a Requisição POST

- Descomente o `enviarForm` no `ElevatedButton` e seu botão de adicionar deve funcionar.



Construindo a Requisição POST

- Infelizmente **não estamos usando uma API de verdade**, então o resultado não é listado em nosso cardápio.

Exercício

- No app de pizzas adicione também as operações HTTP:
 - PUT (para alterar dados);
 - DELETE (para remover dados).

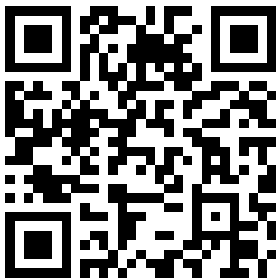
Referências

 Simone Alessandria and Brian Kayfitz.

Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart.

Packt Publishing Ltd, 2021.

Conteúdo



<https://gustavotcustodio.github.io/usabilidade.html>

Obrigado

gustavo.custodio@ulife.com.br