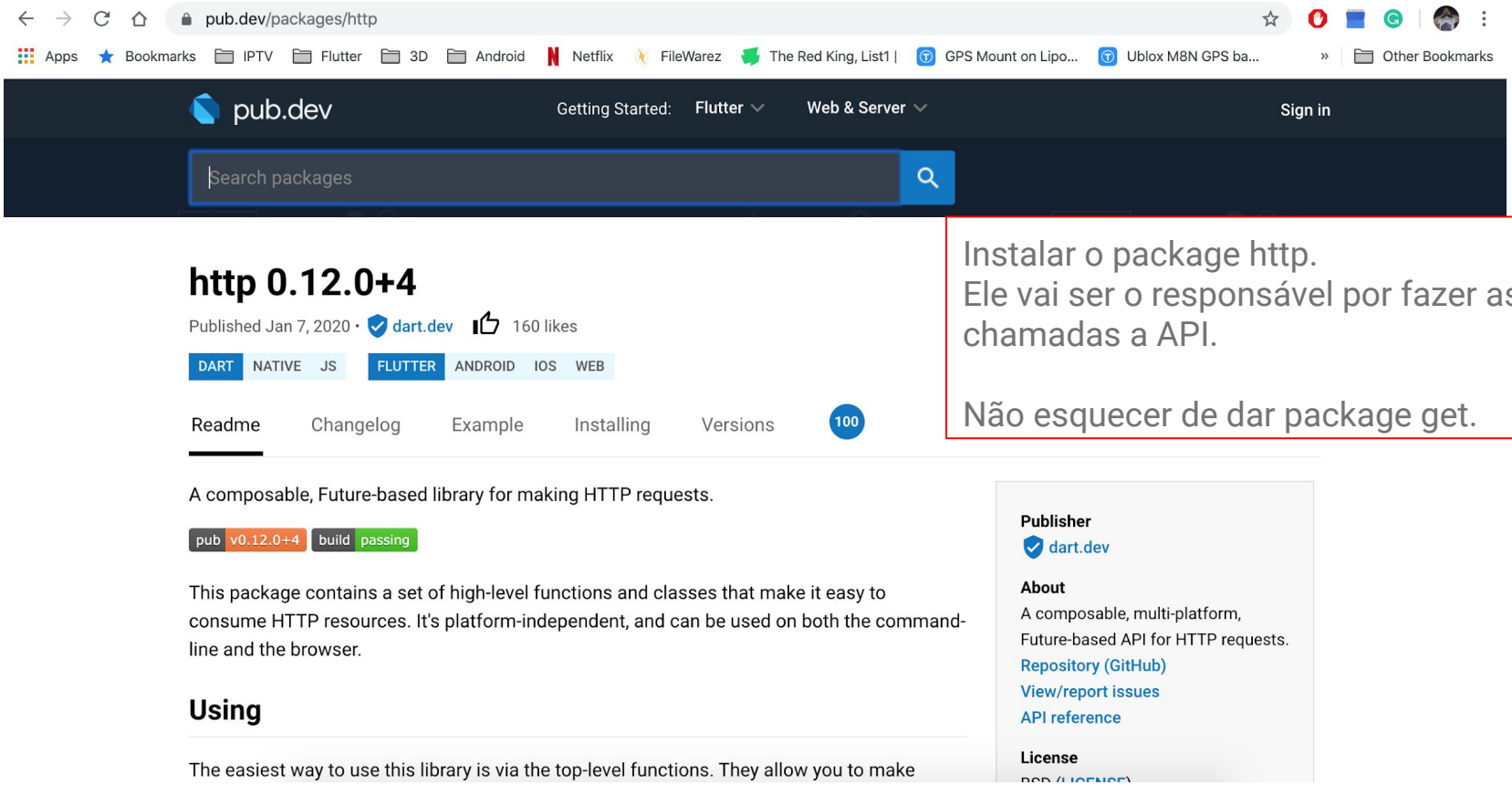
Flutter

Aula 4 - Ufrgs Mobile

Aula 4 - Index

1. Relembrando.
2. Callback Functions
3. Como adicionar fontes e imagens
4. Flutter Web
5. Como gerar o app?
6. Objetivo da aula: UFRGS Mobile

Relembrando - Http Dart package



Instalar o package http.
Ele vai ser o responsável por fazer as chamadas a API.

Não esquecer de dar package get.

Relembrando - Http Dart package - GET

```dart
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
```

```dart
void getApi() async {
  print('Fez chamada');
  var url = 'https://jsonplaceholder.typicode.com/posts';
  var response = await http.get(url);
  print('Response status: ${response.statusCode}');
  print('Response body: ${response.body}');
}
```

1. Importar o package como mostrado ao lado.
2. Criar função assíncrona para tratar o request.
3. Obter a resposta do endpoint.
4. No response.body tem uma string que representa o retorno. No caso de uma API Rest essa string vai ser um json.

Relembrando - Http Dart package - GET

```dart
import 'dart:convert' as JSON;

final json=JSON.jsonDecode(myJsonAsString);
```

Podemos utilizar o package convert para transformar a string de retorno em um Map (ou List<Map>) e acessar os atributos conforme foi visto na primeira aula.

Entretanto a maneira mais "elegante" seria desserializar a string de retorno e trata-la como um objeto dart.

Relembrando - Http Dart package - POST

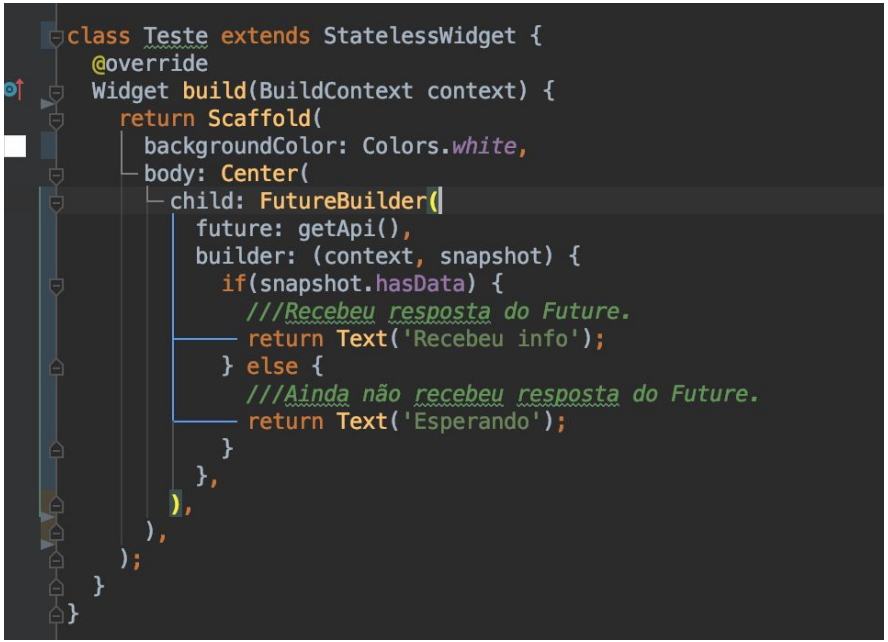
```dart
import 'package:http/http.dart' as http;

var url = 'https://example.com/whatsit/create';
var response = await http.post(url, body: {'name': 'doodle', 'color': 'blue'});
print('Response status: ${response.statusCode}');
print('Response body: ${response.body}');
```

Quando enviamos um post também podemos enviar um body que nada mais é que um Map representando um json. No exemplo acima o json enviado no body é:

```
{
      "name": "doodle",
      "color":"blue"
}
```

Relembrando - FutureBuilder

É um widget que se reconstrói baseado no último snapshot disponibilizado por um Future.



```
class Teste extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.white,
      body: Center(
        child: FutureBuilder(
          future: getApi(),
          builder: (context, snapshot) {
            if(snapshot.hasData) {
              ///Recebeu resposta do Future.
              return Text('Recebeu info');
            } else {
              ///Ainda não recebeu resposta do Future.
              return Text('Esperando');
            }
          },
        ),
      ),
    );
  }
}
```

Mesmo sendo um Stateless widget, quando o estado do Future muda ele chama o builder novamente, podendo assim redesenhar a parte de si que depende de uma chamada assíncrona.

Callback Functions

Objetivo: Comunicação entre widgets.
Exemplo: Widget filho precisa comunicar o pai que um botão foi pressionado.

```
void callback() {
  print('function callback');
}
```

```
RaisedButton _createButton(Function callbackFunction) {
  return RaisedButton(
    onPressed: () => callbackFunction(),
  );
}
```

```
_createButton(callback),
```

Neste exemplo quando criamos o botão **RaisedButton** invocamos a função que for passada como parâmetro quando o botão for apertado.

Neste caso, a função **callback**.

Callback Functions - Tipos

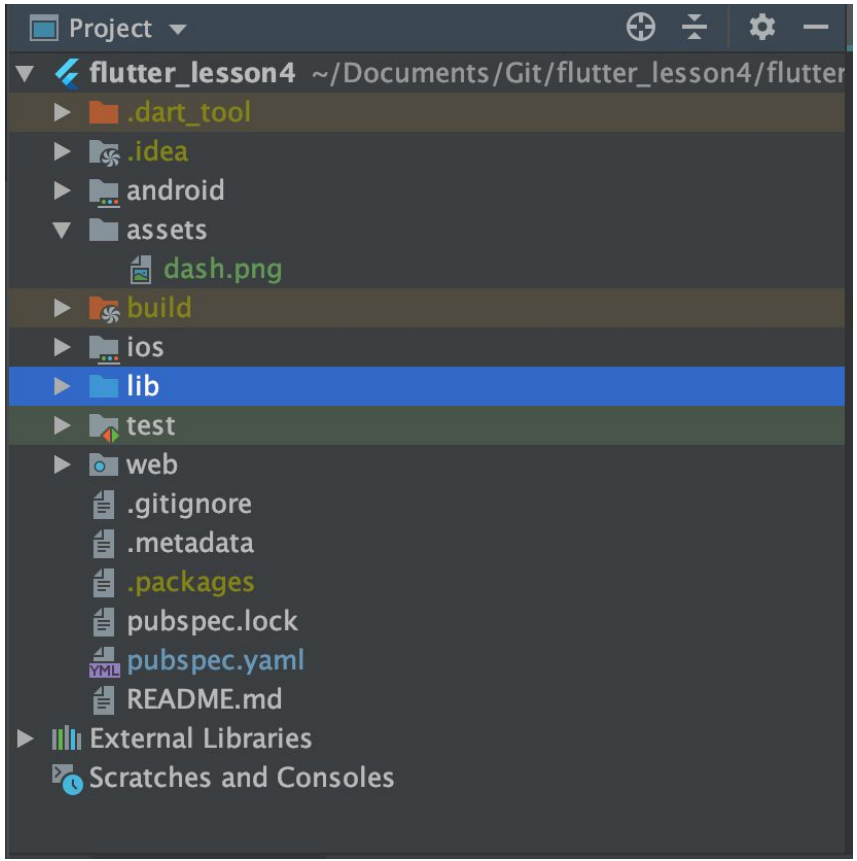
Existem dois tipos: **VoidCallback** e **Function(x)**

VoidCallback: São utilizados quando precisamos apenas notificar que um evento ocorreu e não retorna nenhum valor. Por exemplo: "Botão X foi apertado"

Function(x): Utilizado quando precisamos retornar um valor para o elemento pai. Por exemplo na aula passada quando utilizamos Function(Task) para retornar a task a ser modificada para o widget pai.

O **X** é o tipo do valor a ser retornado.

Como adicionar imagens



1- Criar a pasta **assets** na raiz do projeto.

2- Adicionar as imagens desejadas na pasta.

3- Adicionar a referencia das imagens no **pubspec.yaml** conforme a imagem abaixo

```yaml
flutter:
  uses-material-design: true
  assets:
    - assets/dash.png
```

Como usar imagens adicionadas

Após adicionar conforme o indicado no slide anterior de o comando get packages e realize uma fresh build. Assim os asset serão incluídos no app. Para utiliza-los basta utilizar o widget **Image.assets** e passar o caminho da imagem no src.

```
Image.asset('assets/dash.png'),
```

Como adicionar fontes

O processo é similar ao indicado anteriormente. A única diferença é no pubspec.yaml. Nele iremos indicar que o asset adicionado é uma fonte conforme a imagem abaixo.

```
fonts:
  - family: Schyler
    fonts:
      - asset: fonts/Schyler-Regular.ttf
      - asset: fonts/Schyler-Italic.ttf
        style: italic
```

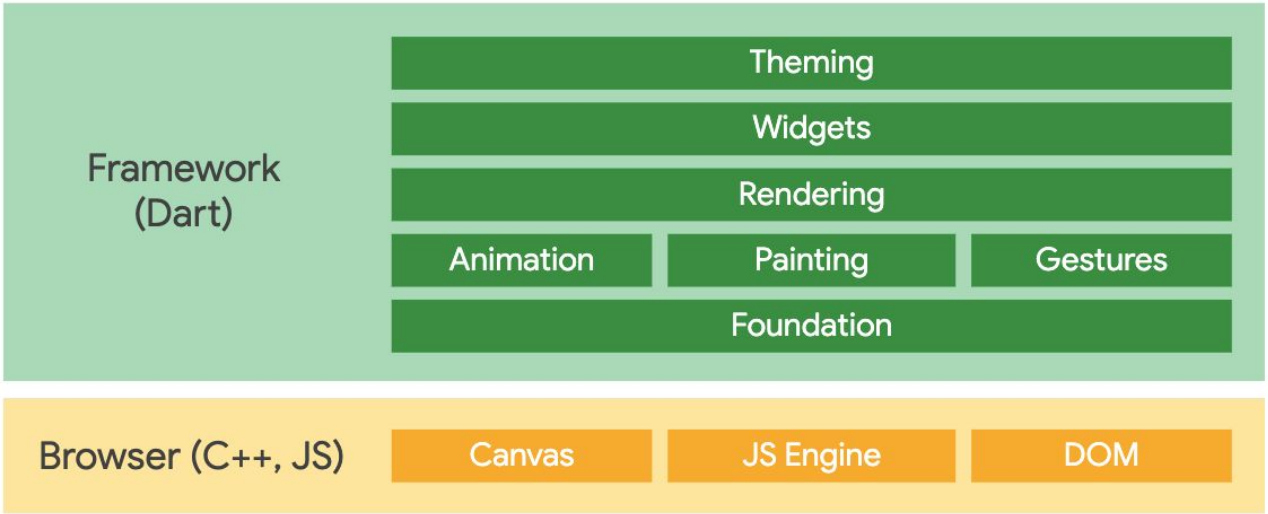
Usando as fontes adicionadas

Basta informar o nome da família criada no Text widget.
No exemplo abaixo o nome da família é **RobotoMono**.

```
Text(
  'Roboto Mono sample',
  style: TextStyle(fontFamily: 'RobotoMono'),
);
```

Flutter Web

Atualmente no flutter beta 1.12 é possível compilar o seu app para um PWA (Progressive Web App) que poderá ser executado em qualquer browser compatível.



Flutter Web - Exemplos

https://flutter.github.io/samples/

Como gerar o app?

Para gerar um apk do seu app basta digitar o seguinte comando no terminal do android studio:

Flutter build apk

O apk vai ser gerado na pasta: **build/app/outputs/apk/release/app-release.apk**

Caso deseje gerar uma versão assinada do app (necessária apenas para publicar na loja) siga o tutorial oficial da google.

https://flutter.dev/docs/deployment/android

Objetivo da aula: UFRGS Mobile

Desenvolva sua própria versão do UFRGS Mobile contendo apenas a parte referente ao RU.

Documentação da API: https://documenter.getpostman.com/view/476996/SWTD9cnU?version=latest