Programação na Internet

Turma i52d

Aula 12
Developing asynchronous API

Sync <versus> Async

Síncrono => Resultado de F() ==== retorno de F

Assíncrona => Resultado de F() != retorno de F

Assíncrono idiomas:

- Callbacks – génese do Javascript
- Promises (aka CompletableFuture em Java e Task em .net)
- async await (except em Java)
- Suspend functions (e.g. Kotlin)

Callbacks em Javascript

Requisito o resultado da chamada async => Sucesso ou Insucesso:

- **Callback** é uma função
- **Callback** é o ultimo parametro da função (convenção)
- **Callback** recebe como 1º argumento erro (convenção):

```
(err, data, …) => { if(err) return….  }
```

> SUCESSO – err é undefined

> ERRO – err refere um objecto Error

!=

Sync:

- Resultado = retorno da função.
- Erro = Excepção

Módulo pedidos HTTP

- request
- fetch – standard no Browser
- node-fetch
- urllib
- http – modulo standard do Node.js
- sync-request

FileSystem modulo fs

- `fs.readFileSync` – o sufixo Sync destaca o que não é normal, ou seja a API ser bloqueante.
- `fs.readFile` – por omisssão, assumimos sempre que a API é assíncrona.

NOTA: noutros ambientes (e.g. Java ou .Net) a abordagem é contrária:

- URL – tem API sincrona
- AsyncHttpClient – tem o prefix Async destacando que é assíncrona.

Sync versus Async

```
function fileSize(path) {
    console.log('Reading ' + path)
    const buffer = fs.readFileSync(path)
    const size = buffer.toString().length
    console.log(`>>>>> ${path}: ${size}`)
}
```

```
Reading Metamorphosis-by-Franz-Kafka.txt
>>>>> Metamorphosis-by-Franz-Kafka.txt: 141417
Reading The-History-of-Tom-Thumb-and-Others.txt
>>>>> The-History-of-Tom-Thumb-and-Others.txt: 41476
Reading The-Wizard-by-Rider-Haggard.txt
>>>>> The-Wizard-by-Rider-Haggard.txt: 346342
```

```
function fileSize(path) {
    console.log('Reading ' + path)
    fs.readFile(path, (err, buffer) => {
        if(err) console.log(err)
        else {
            const size = buffer.toString().length
            console.log(`>>>>> ${path}: ${size}`)

        }
    })
}
```

```
Reading The-Wizard-by-Rider-Haggard.txt
Reading Metamorphosis-by-Franz-Kafka.txt
Reading The-History-of-Tom-Thumb-and-Others.txt
>>>>> The-History-of-Tom-Thumb-and-Others.txt: 41476
>>>>> Metamorphosis-by-Franz-Kafka.txt: 141417
>>>>> The-Wizard-by-Rider-Haggard.txt: 346342
```

Exercício 1

Implementar uma função que recebe um array de nomes de ficheiros e **retorna** a soma dos tamanhos dos ficheiros.

```
function sumFilesSize()
```

Requisito:

- Tirar partido da função anterior fileSize (assíncrona) que pode ser modificada.
- Não usar variáveis globais.
- Resultados deixam de ser dados pelas funções em console.log()
  - E.g.

```
console.log(`>>>>> ${path}: ${size}`)
```

Exercício 1

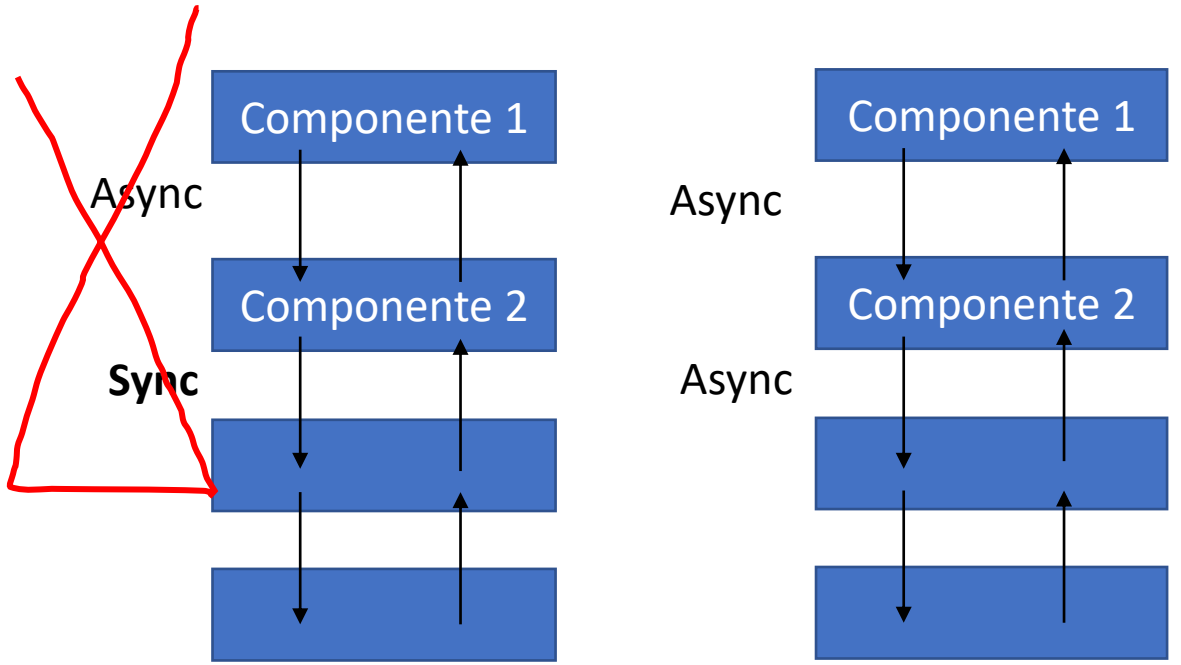
```
function fileSize(path) {
    console.log('Reading ' + path)
    fs.readFile(path, (err, buffer) => {
        if(err) console.log(err)
        else {
            const size = buffer.toString().length
            console.log(`>>>>> ${path}: ${size}`)

        }
    })
}
```

```
return size
```

Software Dev = Composing

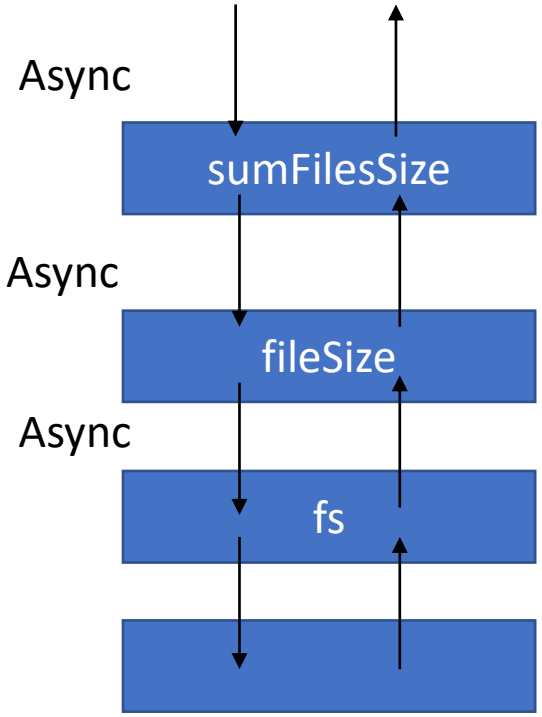
Desenvolvimento de Software

Composição
Composição de Classes
Composição de Funções
Composição de unidades de software
....



Cada componente tem que cumprir o idioma do componente que usa.

Exercício 1



Exercício 2 TPC

Implementar uma função que recebe um array de URLs e **retorna** a soma dos tamanhos dos corpos das respostas HTTP a esses URLs.

```
function sumBodiesLength()
```

Requisito:

- Tirar partido da função anterior bodyLength (assíncrona) que pode ser modificada.
- Não usar variáveis globais.
- Resultados deixam de ser dados pelas funções em console.log()
  - E.g.

```
console.log(`>>>>> ${path}: ${size}`)
```