Programação & Serviços Web
React – Aula 1
2019/2020

INTRODUÇÃO AO REACT

Aula 1 – Introdução ao React

- Fullstack Web Development: The Big Picture
- Instalação do Node
- Introdução ao React
- JSX
- Elementos
- Componentes

Web Design & Desenvolvimento

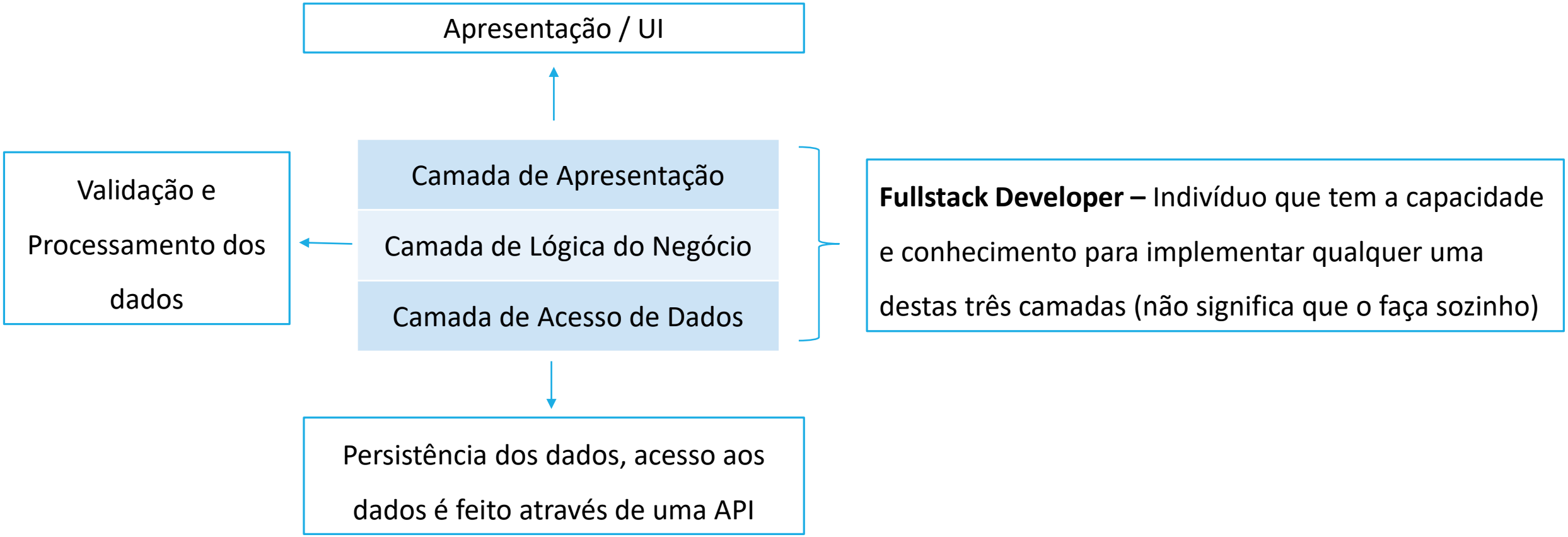
Design	Desenvolvimento
User Interface (UI) / User Experience (UX)	Biblioteca Javascript (React)
Design Visual	UI Framework (Bootstrap 4)
Prototipagem	ServerSide Developemnt (NodeJs + Express + MongoDB)
Cores, gráficos, animações	
Entre outros...	

Outras UCS

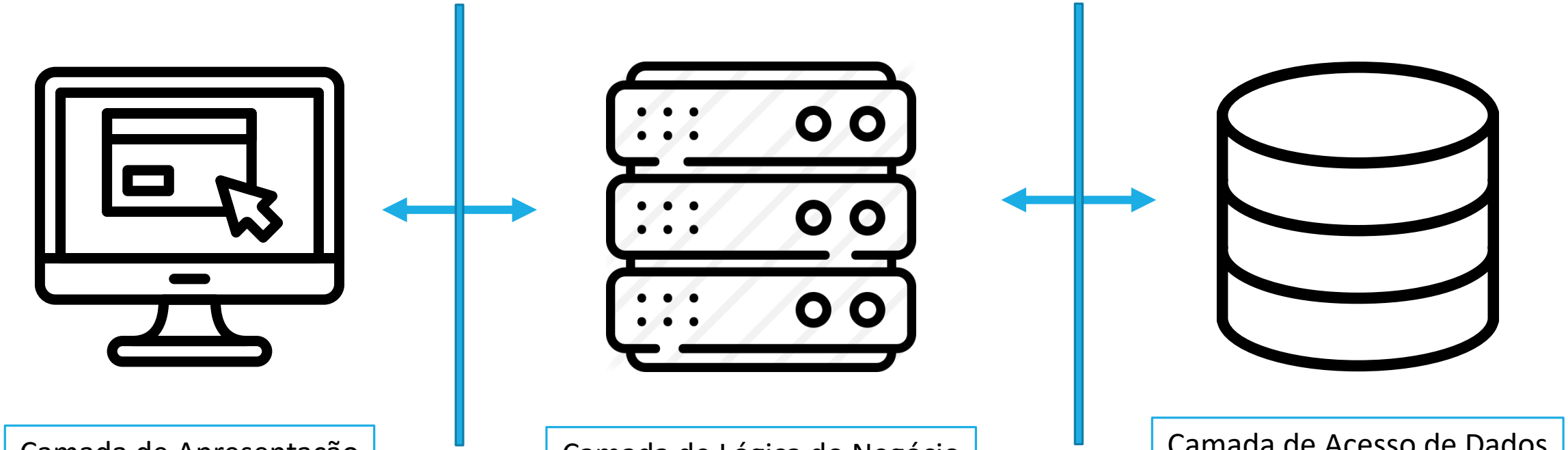
→ Trabalho de Casa

→ UC de TDW

Arquitetura Software 3 Camadas



Arquitetura 3 Camadas



Arquitetura 3 Camadas

Camada de Apresentação
Camada de Lógica do Negócio
Camada de Acesso de Dados

Este é apenas uma possibilidade. Esta será a arquitetura com que vai trabalhar na UC de PSW e TDW

Fullstack Web Development

- Frontend / Client-Side
  - Html, CSS e Javascript (Pode incluir Libraries ou Frameworks)
- Backend / Server-Side
  - Existem várias tecnologias (PHP, Java, C#...)
- Database Manager
  - Existem várias tecnologias (MySql, SQLServer, mongoDB...)

Node.Js

JAVASCRIPT, BUT BETTER

Node.JS

- Ambiente Javascript programado no motor Javascript Chrome V8
- Usa um modelo baseado em eventos que não bloqueia o I/O
  - Faz com que seja bastante leve e eficiente
- Neste momento, só vamos falar da utilização do Node a nível de cliente-side, utilizando o React
  - A nível de server-side vai ser feito na UC TDW

Node Package Manager (NPM)

- Gere o ecossistema dos módulos / pacotes do node.

- Cada um destes pacotes contem:
  - Ficheiros JS
  - Package.json (manifest)

- Resumidamente, um módulo/pacote é algo **feito por terceiros** que pode **adicionar ao seu projeto e utilizar**. Por exemplo:
  - Um input do tipo calendário
  - Um loader para mostrar quando carrega as páginas
  - Sistema de Traduções
  - Entre outros

Node.JS / NPM

- O ficheiro package.json serve para:
  - Documentação dos pacotes que o projeto precisa
  - Permite especificar a versão dos módulos a ser usados projeto
  - Faz com que a build seja reproduzível, de forma a ser mais fácil de partilhar com o resto da equipa

- **Build** – Processo de compactar todos os módulos/código do projeto num só ficheiro de forma a poder ser colocado num site (PDF 6)

Instalação do Node.JS / NPM

- Fazer o download e instalação do node.JS (LTS) em
  - https://nodejs.org/en/download/

- Para verificar a instalação do Node

```
node -v
```

Inicializar package.json

- Para inicializar o **package.json** no projeto, basta abrir a linha de comandos na pasta e fazer:

```
npm init
```

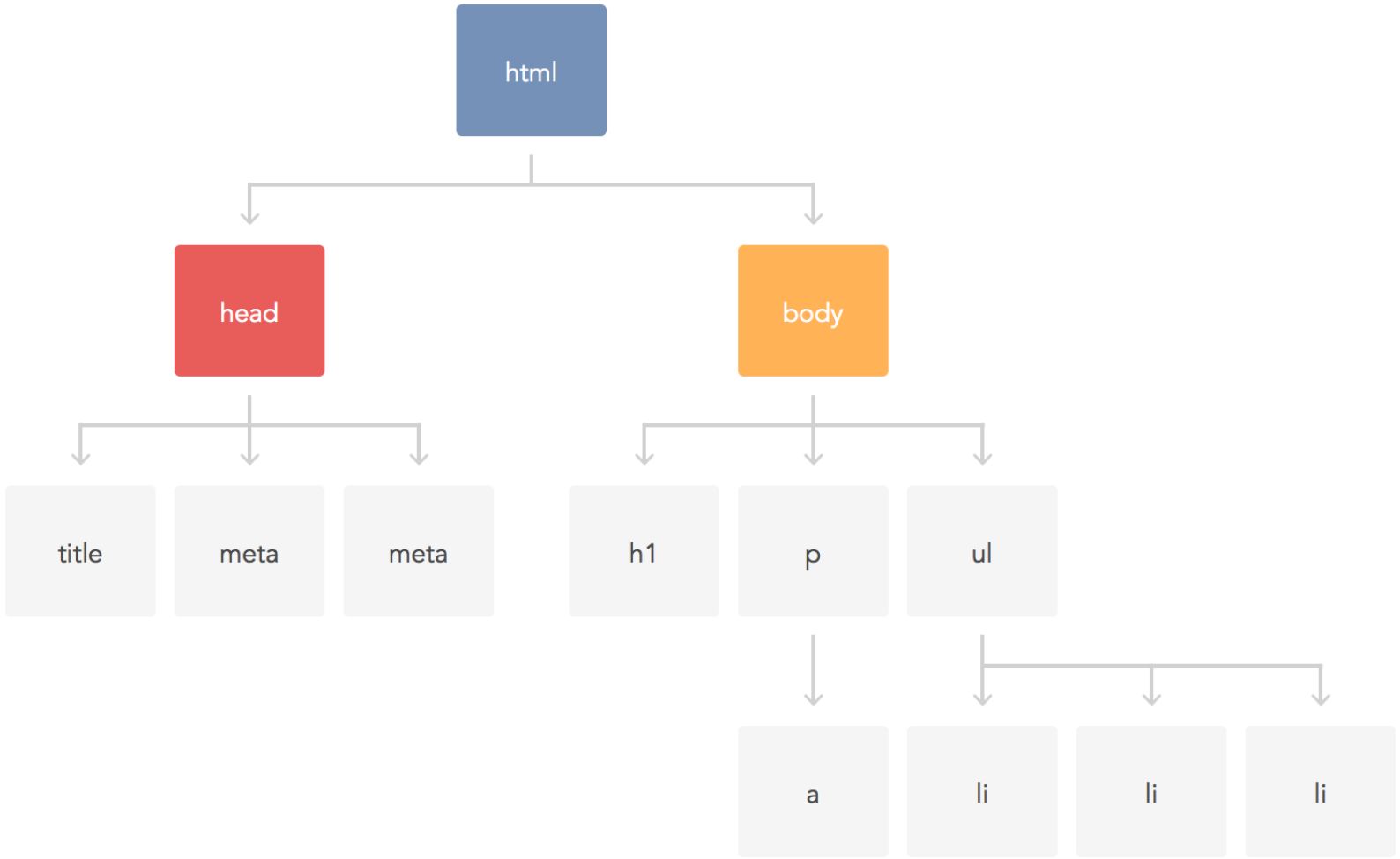
- São-lhe feitas umas perguntas relativamente ao projeto, informação essa que vai ficar gravada no ficheiro.
  - Se não souber o que introduzir em alguns dos campos (por exemplo do test), deverá deixar o campo vazio

Brower DOM

- **DOM** é o Modelo do Objeto de Documento, uma interface para HTML e XML.
  - Define a estrutura lógica do documento e a forma como o documento pode ser acedido e manipulado
  - Esta estrutura é uma árvore (invertida) em que na raiz está o html, depois como filhos está o body, head, e por aí adiante.

- Como viram no módulo de JS, um site complexo que requer muita atualização do DOM acarreta código demasiado complexo (JS / jQuery)
  - É aqui que as Bibliotecas / Frameworks ajudam

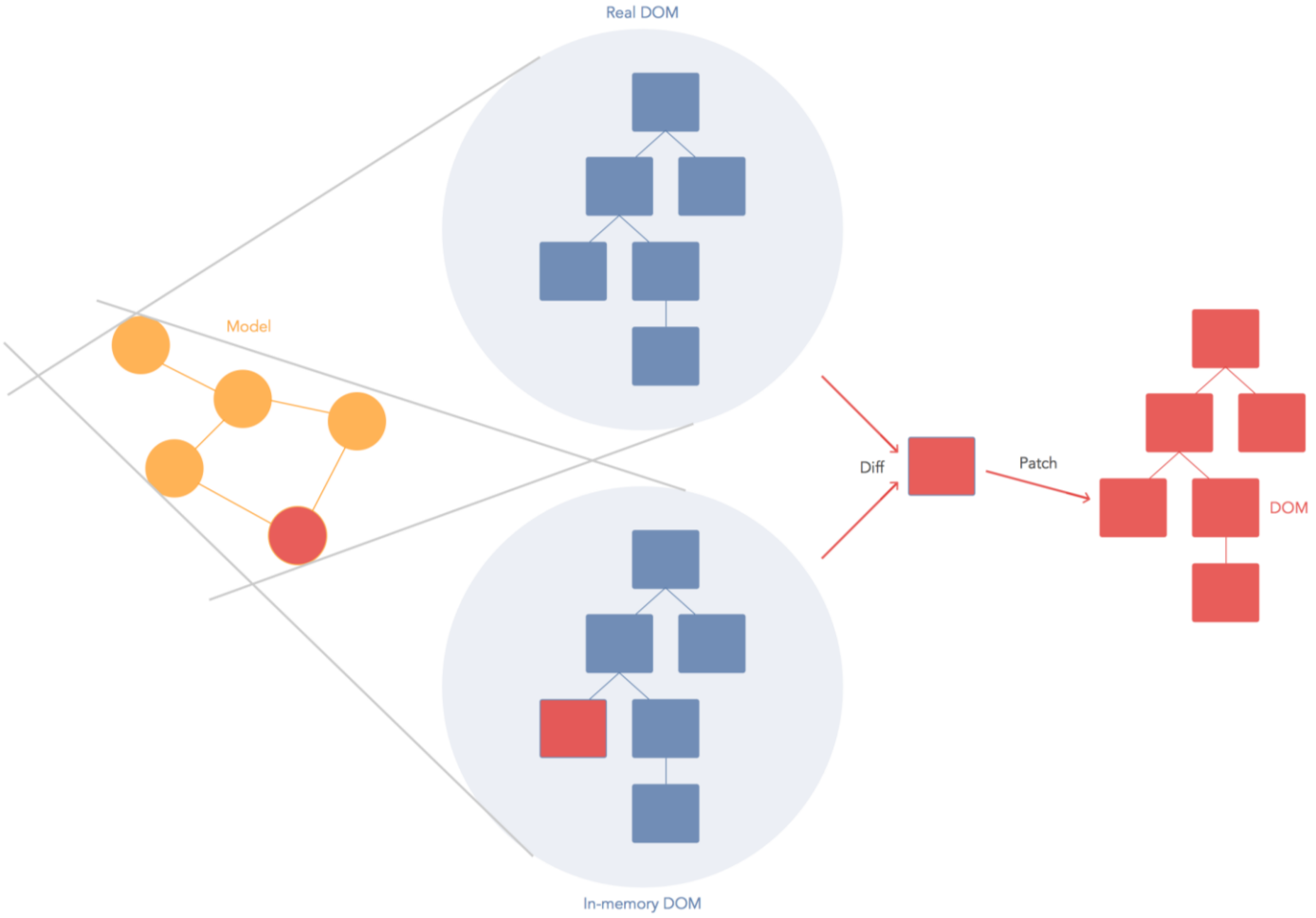
Browser DOM



Virtual DOM

- Nome que o React dá ao seu motor de manipulação do DOM.
- Composto por uma série de chamadas Javascript que dizem ao React como construir uma árvore do DOM Virtual, e actualiza-a quando a informação vai mudar.
- Após esta mudança, são calculadas as diferenças entre o estado anterior e o estado novo.
- O algoritmo garante assim o número mínimo de operações necessárias para actualizar o DOM real.

Virtual DOM



Biblioteca vs Framework

- **Biblioteca**
  - Coleção de funções que são uteis quando se está a programar web apps. O código está a mandar e chama a biblioteca apenas quando for preciso
- **Framework**
  - Uma implementação particular da web app, onde o nosso código preenche os detalhes. A Framework está a comandar e é chamada para dentro do código quando for necessário algo específico
  - **Hollywood Principle**: Don't call us, we call you! (Inversão do controlo)
- Existem uma série de bibliotecas JS
  - React, Angular, Vue, Meteor, Ember, etc

React

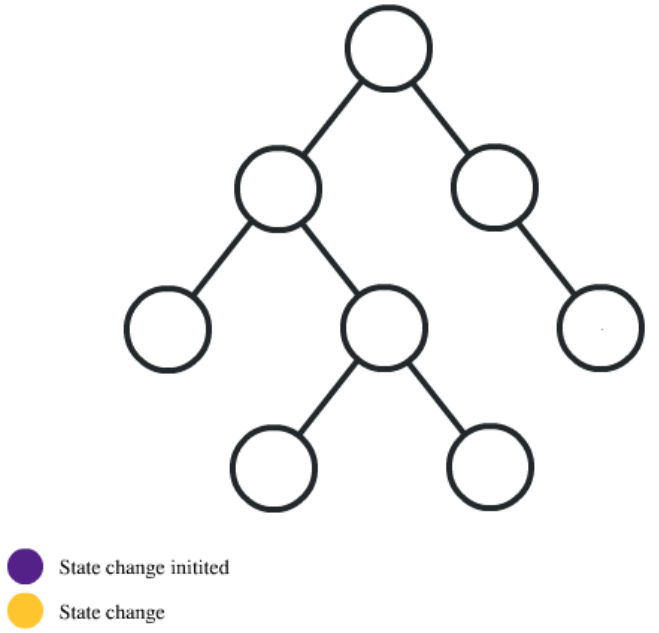
React

- Desenhada por **Jordan Walke**, usada pela primeira vez no newsfeed do **Facebook em 2011**.
- **Open-Source** desde **Maio 2013**
- Desenhada para velocidade, simplicidade e escalabilidade

- Framework JS usara para construir interfaces
  - O Facebook intitula o react como library
- Programação Declarativa
- Baseado em Componentes

Vocabulário React

- One way data flow
- JSX
- Element
- Components
- State
- Props



Criar uma aplicação com o React

- Para começar, instale o módulo que o irá ajudar a criar um projeto react (só necessário uma vez, pois a instalação é global (**-g**))

```
npm install –g create-react-app
```

- Abra a linha de comando num local onde quer criar a pasta do projeto

```
create-react-app NOME_DA_PASTA_A_CRIAR
```

- Para correr o projeto, mude para a pasta criada e corra o comando

```
npm start
```

Visão Global da App do React

- Pasta **Node_Modules**
  - contem todos os módulos do NPM que foram instalados e são necessários para o projeto correr
- Pasta **Public**
  - Contem o favicon, o index.html, o manifest.json e outros ficheiros.
  - Se abrir o index.html irá verificar que não está lá nada sem ser o HEAD, um noscript e uma **div** com o id **Root**
- Pasta **Src**
  - Se abrir o ficheiro index.js, irá ver a seguinte linha de código
    - ReactDOM.Render(<App />, document.getElementById('root'));
  - Isto faz com que o código do React seja integrado dentro daquela div

Imports

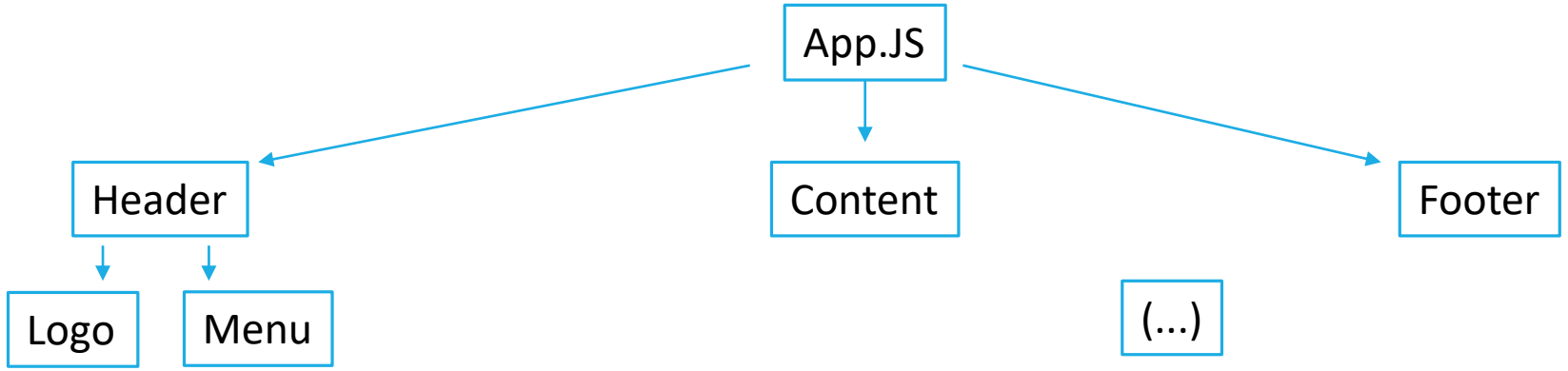
- No topo do ecrã, são feitos várias importações de outros módulos / ficheiros

```
import React, { Component } from 'react';
import './App.css';
```

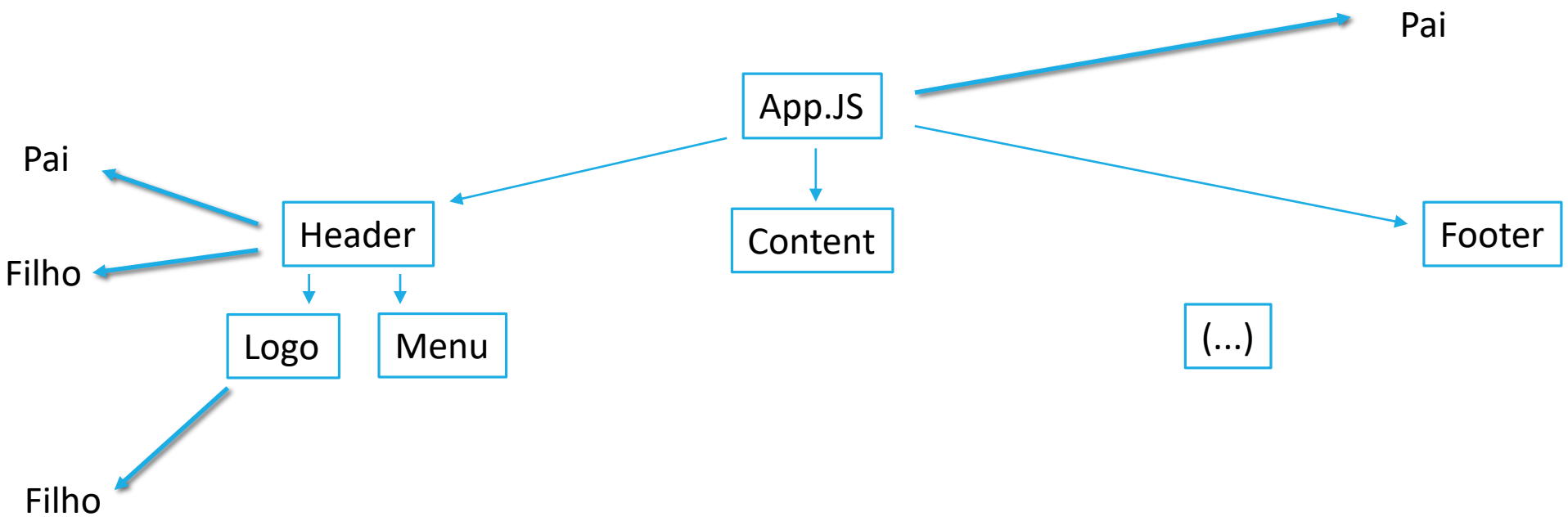
- O **Import** serve para ir buscar outros ficheiros, imagens, classes, para que possam ser usados nesse mesmo ficheiro
  - Neste caso está a importar-se o:
    - Ficheiro **App.css**
    - Classe **React** & classe **Component** do módulo React

Visão Global da App do React

- O ficheiro index.html indicava para editar o ficheiro App.js File
- O App.js é o ficheiro central do React
  - Pode ser mudado para outro ficheiro.
- Será a partir deste ficheiro que irá chamar todos os outros ficheiros



Componentes Filhos



Se o componente que estiver a usar estiver num ficheiro à parte, este terá de ser importado

Introdução ao JSX

```
const elemento = (
  <h1>Hello, World! </h1>
);
```

- Isto é uma sintaxe especial que o React utiliza.
  - JSX – Syntatic Extension to Javascript

- Evita a separação virtal da lógica de renderização de a lógica da UI

Utilização de expressões em JSX

Imaginemos que temos o seguinte objeto:

```
const pratos = {
  id : 0,
  name: 'Pizza',
  ...
};
```

Como é que podemos aceder a estes campos no JSX?

```
return (
    <div key={pratos.id} className="prato">
      <span>{pratos.name}</span>
    </div>
);
```

Utiliza o nome da variável, seguido de um ponto, e do nome do campo em questão

Elementos

- Um elemento é o bloco mais pequeno das aplicações React

```
const elemento = <h1 className="App-title">Welcome to React </h1>
```

- São objetos simples que ocupam pouca memoria a criar
- O React apenas atualiza o que é necessário
  - Ele compara o estado de um elemento com o do anterior e apenas aplica as atualizações aos elementos necessários

```
<div>
    <h1>Hello, world!</h1>
    <h2>It is {new Date().toLocaleTimeString()}.</h2>
</div>
```

Este h1 é estático e não precisa de atualizações

Este h2 vai mudar a cada segundo que passe. O valor é actualizado, tudo o resto fica igual

Exercício 1

Avaliar os seguintes elementos e verificar se são válidos

Elementos

var elemento = (...)

```
<h1>Este é o meu site </h1>
```

```
<span>Bem vindos ao meu site</div>
```

```
<span>Bem vindos ao meu site</span>
```

```
<span>
    Este é o meu site
</span>
<span>
    O meu nome é Nuno Carapito
</span>
```

```
<div>
    <span>
        Este é o meu site
    </span>
    <span>
        O meu nome é Nuno Carapito
    </span>
</div>
```

Elementos

```
return (
<span>
        Este é o meu site
</span>
<span>
        O meu nome é Nuno Carapito
</span>
)
```

```
return (
        <div>
                <span>
                        Este é o meu site
                </span>
                <span>
                        O meu nome é Nuno Carapito
                </span>
        </div>
)
```

Os exemplos seguintes resultam no mesmo

```
const elemento = <h1>Título</h1>)
return elemento
```

```
return (<h1>Título</h1>)
```

Componentes

Componentes

- São “feitos” a partir de 1 ou mais elementos

- Permitem dividir a UI em vários pedaços de código independentes e reutilizáveis.
  - Permite individualizar a programação de secções do projeto, o que facilita o trabalho em equipa.

- Podem ser de 2 tipos
  - Componentes de Função
  - Componentes de Classe

Componentes de Função

- Maneira simples de definir componentes de React
- **Função javascript** que devolve um elemento React ou uma colecção de elementos react que define a view
- Recebe o **objeto props** como parâmetro

  Não tem acesso ao state ou a lifecycle hooks

PDF 2

Componentes de Função

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

export default Welcome;
```

- Esta função é um componente de função válido porque:
  - Só aceita 1 parâmetro (objeto props, ou seja, objeto com as propriedades)
  - Devolve um elemento React

Exemplo de Componente de Função

```
function RenderContent(props){
        Return(
                <div .....>
                </div>
        );
}
```

```
Const RenderContent = (props) => {
        Return (
                <div .....>
                </div>
        )
}
```

No caso de os componentes serem criados em ficheiros à parte, **é importante** no final não esquecer de:

```
Export default RenderContent
```

Nome do Componente

Componentes (Classes)

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

- Estes 2 excertos de código são equivalentes. No entanto, o 1o permite mais funcionalidades, tais como o acesso ao state, e o acesso a funções do life cycle

Componentes (Classes)

- Estende o React.Component para ir buscar os seus métodos
  - É importada no topo do ficheiro
    - import React, { Component } from 'react';

- Quando se usa os **componentes de classes**, é obrigatório colocarem a função render(). É ela que vai fazer com que algo seja renderizado
  - Esta função tem de **devolver** um **elemento**

- Pode ter um state local e accesso a Lifecycle hooks

Exercício 2

Transformar o código do vosso app.js de **Componente de Função** em **Componente de Classe**

Componentes

- No **Html**, usamos class="XPTO". No **React**, className="XPTO"
  - Porquê? Repare que quando declara a classe App, usa a keyword "class". Para evitar confusões, no react usa-se className

- Os nomes dos componentes deve começar sempre por uma **maiúscula**
  - No react, quando inicializamos uma **tag** com uma letra **minúscula**, ele vai associar como DOM (H1 != h1)

Exercício 3

Crie a pasta **components** dentro da pasta **src** onde irá gravar os componentes criados por si

Exercício 4

Crie um novo componente com o nome **Header**

Mova o conteúdo do <header> do **App.js** para o **Header.Js**

Ficha 1 - Moodle

Para submissão no final da aula

Leitura Complementar

- https://reactjs.org/docs/hello-world.html
- https://reactjs.org/docs/introducing-jsx.html
- https://reactjs.org/docs/rendering-elements.html