

# Análise e Desenvolvimento de Sistemas

- Desenvolvimento de sistemas *front end*

# Aula 2 (parte 1)

- SPA → Aplicação de página única

# O que você vai aprender nessa aula

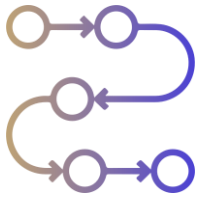


- Conceitos sobre SPA

## O que você vai precisar para acompanhar essa aula



- Nenhum recurso específico
- Atenção



## Dinâmica

- Exploração de conceitos
- Exemplificações



# Contexto



- Como as páginas web eram elaboradas
  - Originalmente eram puramente HTML
    - Objetivo era o compartilhamento de documentos entre pesquisadores
  - Com a popularização
    - Páginas foram estilizadas para melhorar a comunicação, empregando CSS
  - Animação
    - Com a entrada da Netscape propôs um recurso para dinamização das páginas, JS
  - Apesar da evolução
    - As páginas eram naturalmente estáticas
    - Ao acessar uma URL, um documento era retornado (literalmente um arquivo)

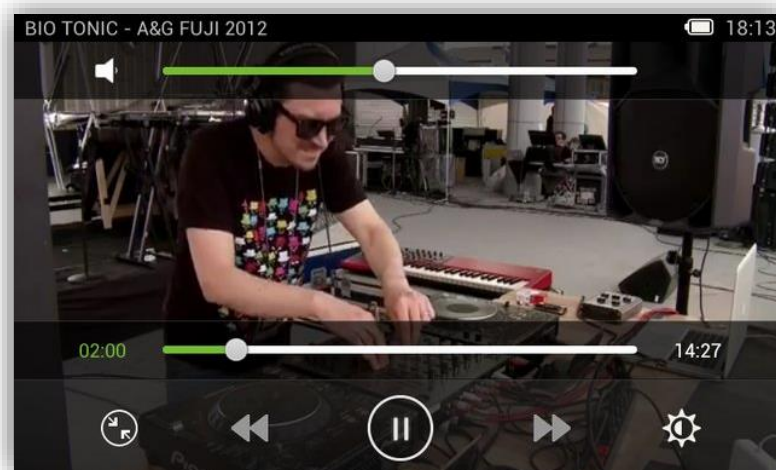
# Contexto



- Como as páginas web eram elaboradas
  - A sofisticação das páginas web levaram a exploração de novas propostas
    - Uma proposta inicial foi CGI (*Common Gateway Interface*)
      - Basicamente uma primeira versão de SSR (*Server Side Rendering*)
    - As requisições dos clientes eram tratadas pelo servidor
      - Ao invés de devolver um arquivo armazenado, um documento era gerado e devolvido
      - Documento era gerado baseado em parâmetros passados e lógica associada
  - Proposta contribuiu para a dinamização das páginas, mas....
    - Na prática os navegadores continuavam recebendo um arquivo HTML completo

# Contexto

- Outra proposta de front end
  - Aplicativo nativo (desktop ou mobile)
    - Todo aplicativo é baixado/instalado no dispositivo alvo
    - Todos recursos que precisam ser renderizados na tela já estão no dispositivo
    - O servidor, quando consultado, retorna tão somente os dados
    - Dá a aparência de solução mais rápida/responsiva comparada a soluções web



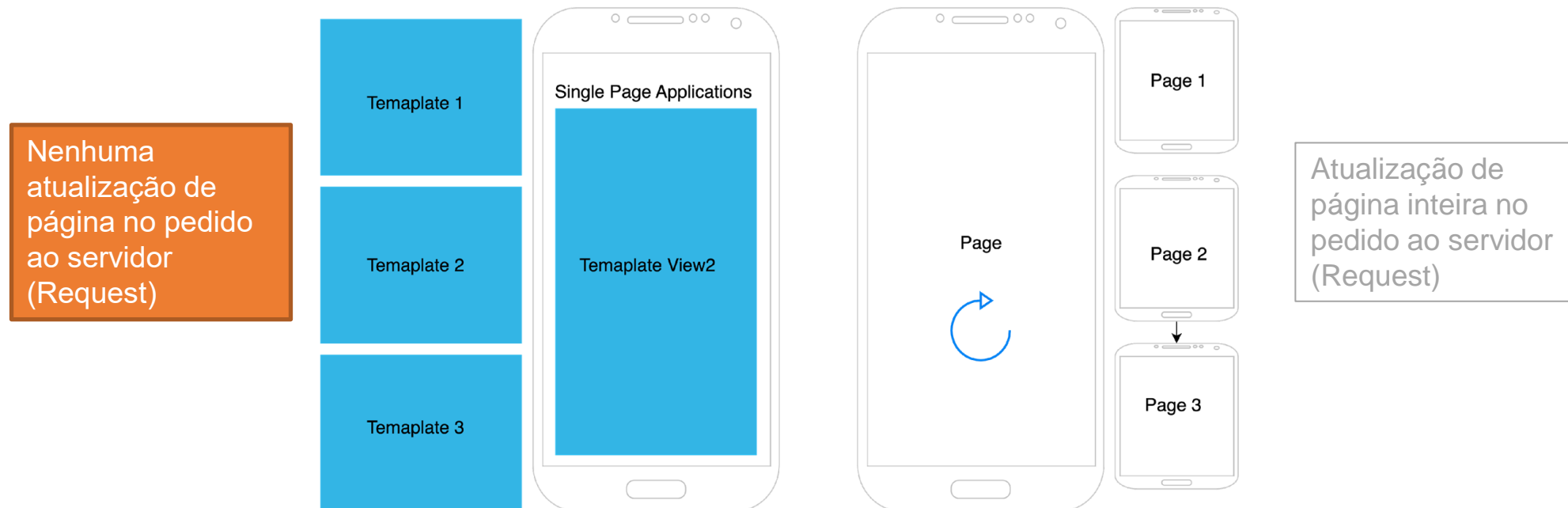


# Contexto

- SPA - Single Page Applications
  - Uma SPA ou *Single Page Application* é uma aplicação web que se baseia em uma única página
  - Página inicial a ser renderizada funciona como uma “casca”, um pedaço de HTML que irá conter as diferentes *views* parciais
  - Uma SPA se utiliza dos mecanismos de renderização parcial de páginas de forma assíncrona sem a necessidade de uma nova requisição completa a um servidor

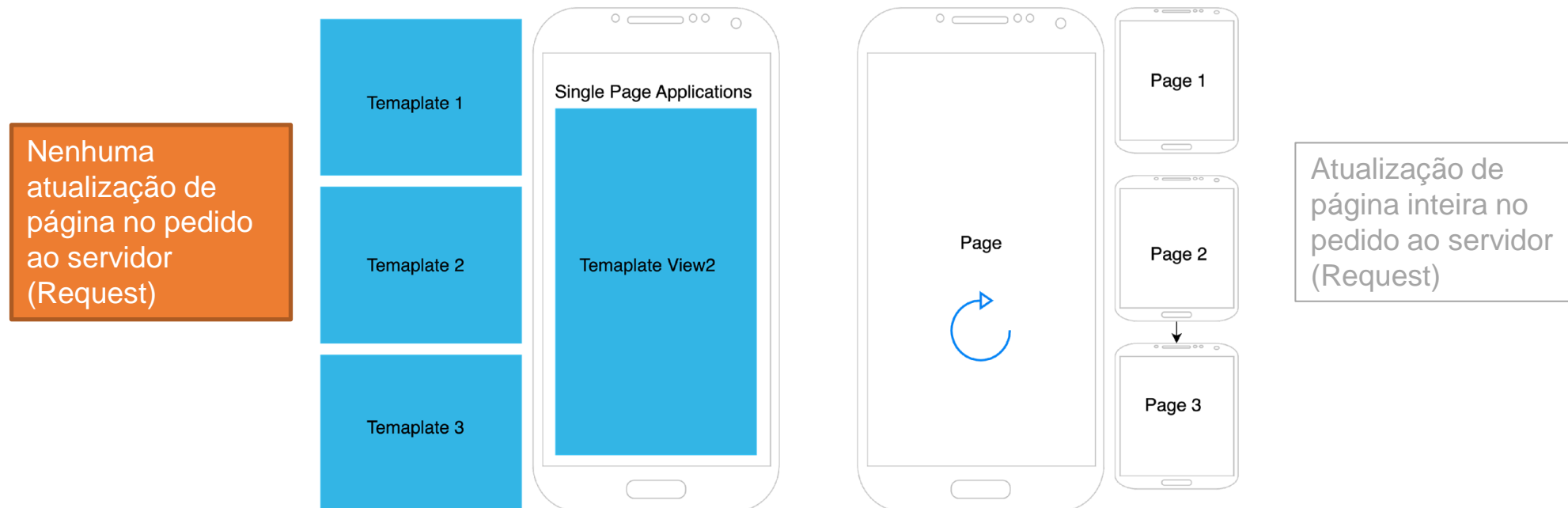
# Single Page Application

- Modelo web de página única
  - Conteúdo é atualizado dinamicamente
    - Não requer recarga completa da página



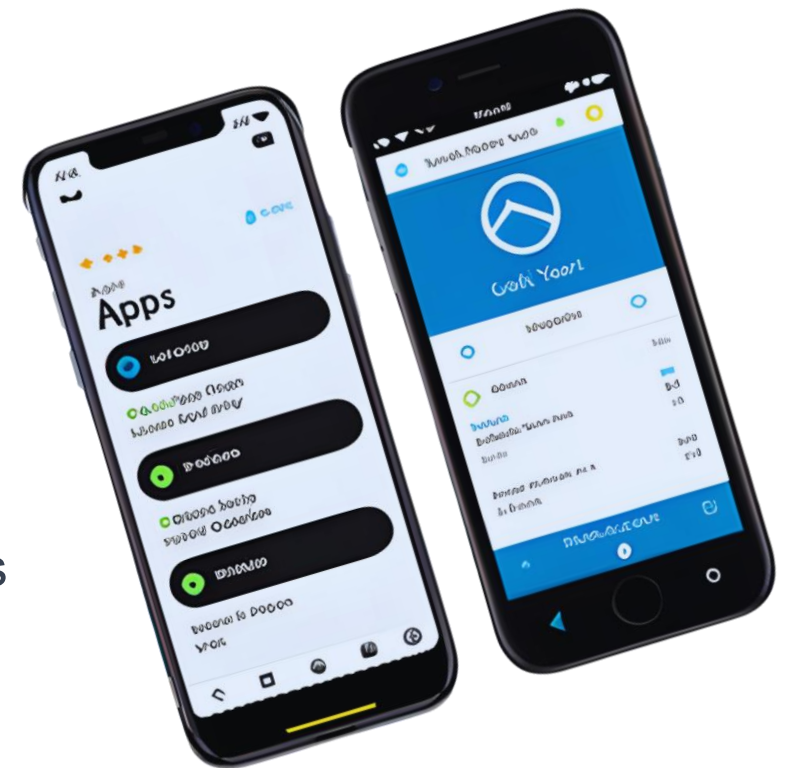
# Single Page Application

- Modelo web de página única
  - Interação do usuário ocorre principalmente no lado do cliente
    - A aplicação é executada no navegador
    - Utiliza tecnologias como HTML, CSS e JavaScript

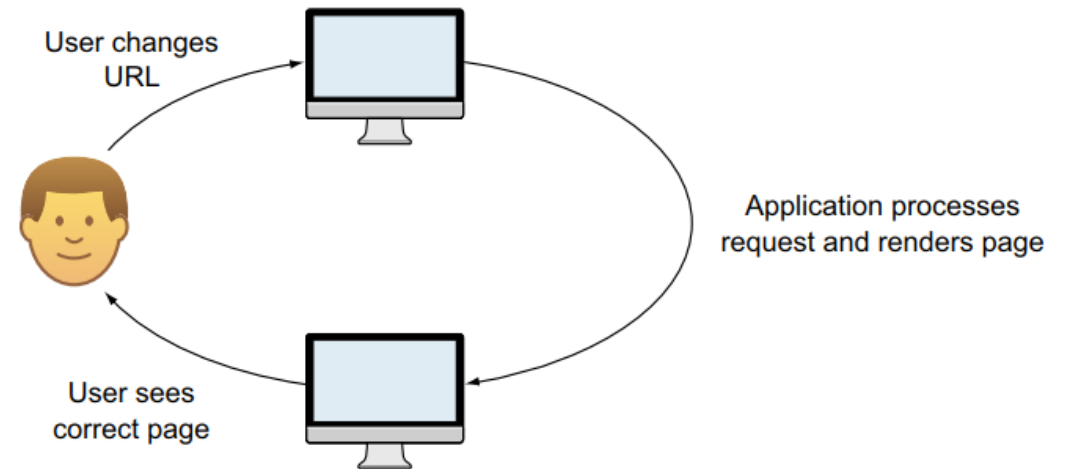
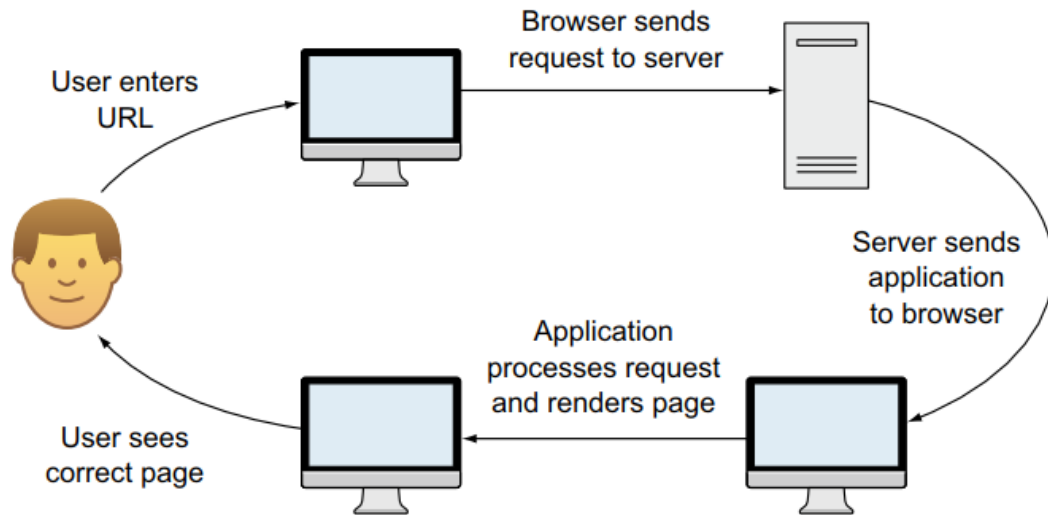


# Single Page Application

- **SPA** vs aplicação web tradicional
  - Navegação entre partes da aplicação é feita de forma assíncrona
    - Não necessita recarregar uma página completa
    - Apenas as partes que mudam na página são atualizadas
    - Experiência de usuário é mais fluída e responsiva
      - Similar àquela com aplicativo nativo.

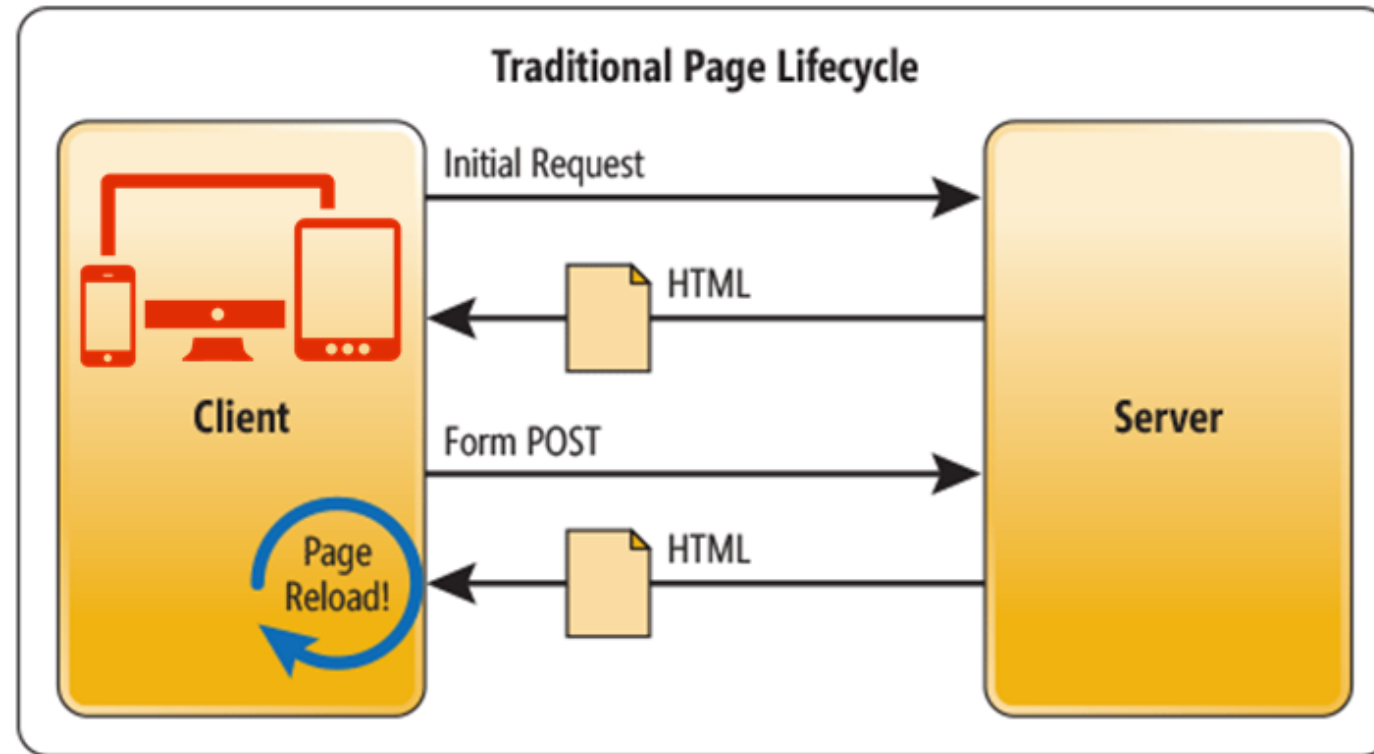


# Single Page Applications

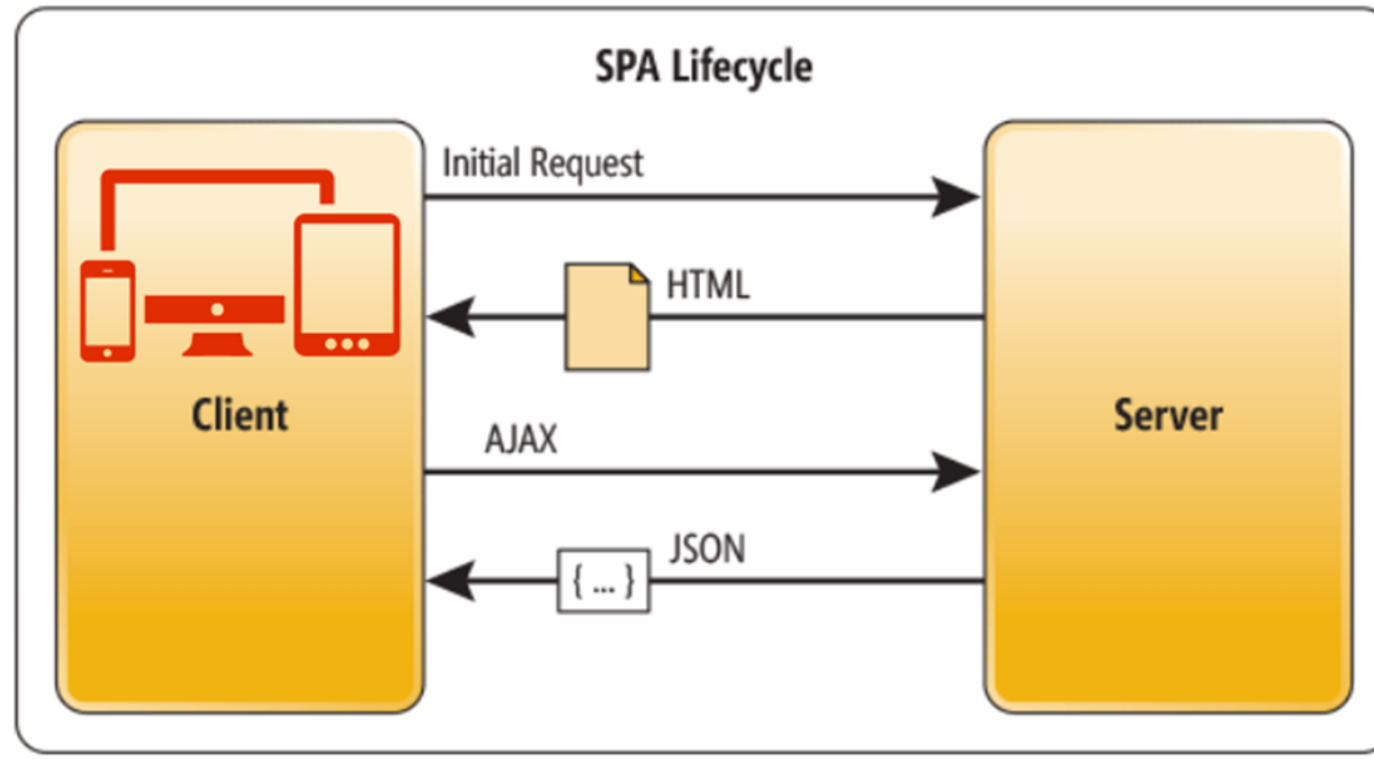




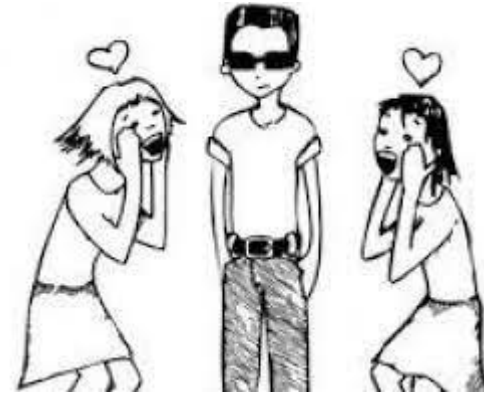
# Single Page Applications



# Single Page Applications

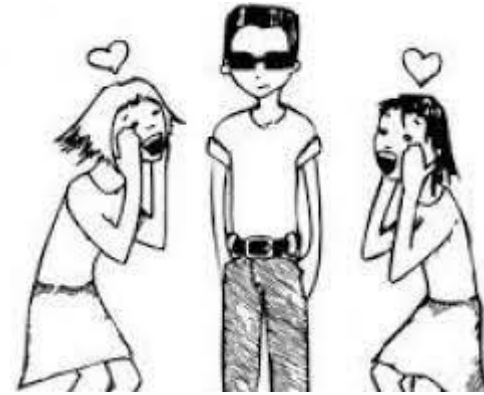


# Single Page Application



- As SPAs são populares por uma série de razões
  - Permitem melhoria na experiência do usuário
    - Solução potencialmente mais rápida quando comparada a soluções web convencionais
  - Contribuem para simplificar a lógica de negócios e o gerenciamento do estado
    - Parte da lógica de processamento pode ser transferida para o cliente
      - Como consequência reduzindo a carga no servidor

# Single Page Application

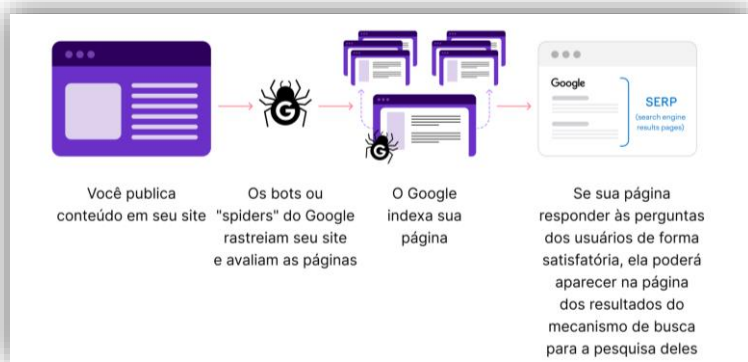


- As SPAs são populares por uma série de razões
  - Podem ser desenvolvidas de forma modular
    - Diferentes partes da aplicação são encapsuladas em componentes reutilizáveis
      - Facilitar a manutenção e o desenvolvimento ágil
  - Amplamente utilizadas em uma variedade de contextos
    - redes sociais, aplicativos de e-commerce, ferramentas de produtividade e etc.

# Single Page Application

- **Pontos de atenção no uso de SPA**

- Visibilidade da solução em mecanismos de busca buscadores
  - Sites convencionais exploram SEO para isto
    - SEO (*Search Engine Optimization* / otimização de mecanismos de busca)
    - Processo de melhoria de desempenho e experiência do site
      - Contribui o site ganhar mais visibilidade em resultados orgânicos em mecanismos de busca
- Problemas se forem desenvolvidas com frameworks SPA, mas...
  - SEO precisa conhecer o conteúdo do site, mas ele é elaborado de forma dinâmica e parcial
  - A maioria dos indexadores hoje conseguem entender aplicações SPA.





# Single Page Application

- **Pontos de atenção no uso de SPA**

- Lentidão inicial

- Aplicações SPA podem ser um pouco mais lentas, na inicialização.

- Isso ocorre porque todo o conteúdo precisa ser baixado de uma vez para o cliente no primeiro acesso

- Arquivos iniciais requeridos:

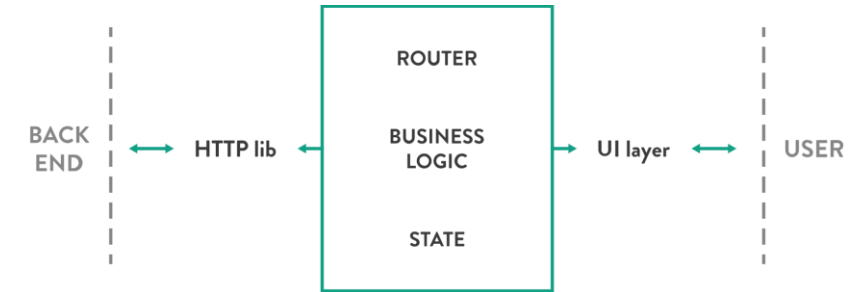
- Template HTML, arquivos JS e CSS, imagens, etc

- Cargas posteriores tendem a ter menor volume de dados transferidos

# Single Page Application

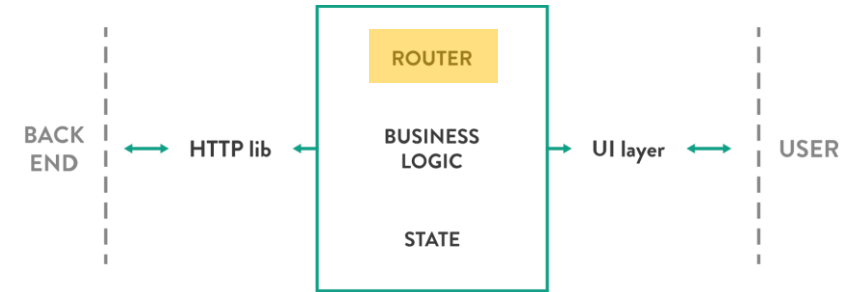
- **Pontos de atenção no uso de SPA**
  - Segurança da solução
    - Aplicações SPA menos seguras que aplicações multi-page renderizadas no servidor
  - Exemplo de ataques XSS (Cross-site Scripting)
    - Inserção de código malicioso
      - Inserem recursos visíveis ou não que capturam dados sensíveis do usuário
  - Qual a diferença se SPA ou multipágina renderizada no servidor
    - Código Malicioso é mantido na página em aplicações SPA
    - Podem ser “sanitizadas” uma vez que as páginas são completamente renderizadas

# Single Page Applications



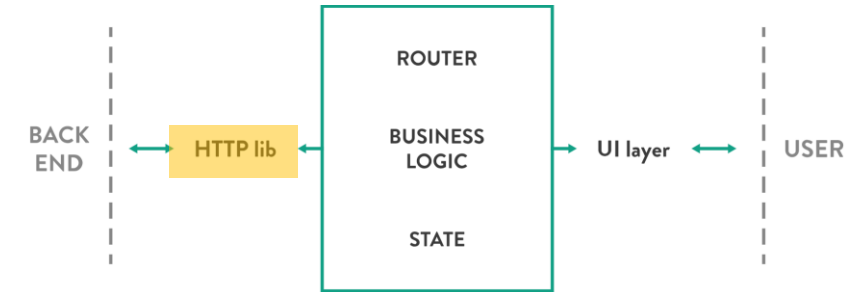
- Recursos frequentemente utilizados por frameworks para SPAs:
  - Roteamento entre views (Router)
  - Requisições assíncronas para o backend (HTTP lib)
  - Views baseadas em template (UI layer)
  - Gerenciamento de estado (State)
  - Lógica de negócio (Business logic)

# Single Page Applications



- Algumas características do **ROUTER**
  - Tratar as demandas do usuário dentro da aplicação
  - O roteador monitora a url da página
    - Sempre que ela muda, executa uma lógica personalizada
    - Geralmente renderiza uma visualização de componente/aplicativo
    - Fornece uma maneira de criar links dentro do seu aplicativo

# Single Page Applications

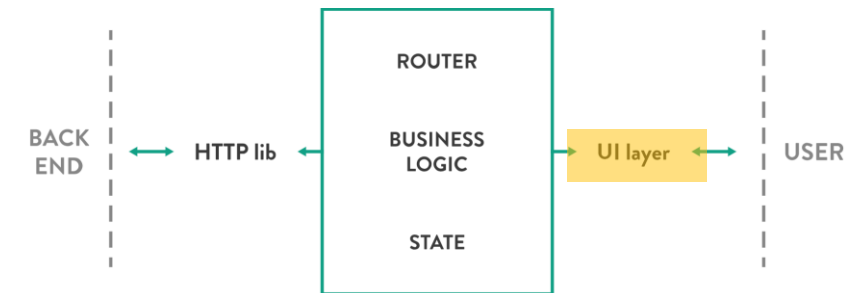


- Algumas características do **HTTP lib**
  - Tratar as demandas do aplicativo para o back end
    - Normalmente visa obter algum dado ou realizar outra ação específica
  - Quase todo aplicativo
    - Possui dados que persistem em um banco de dados
  - Aplicativos de página única
    - São compostos por uma única página que carrega o aplicativo via Javascript
    - Toda vez que você mudar de página devem ser solicitados novos dados
      - Ação realizada de forma assíncrona por meio de chamadas de API.

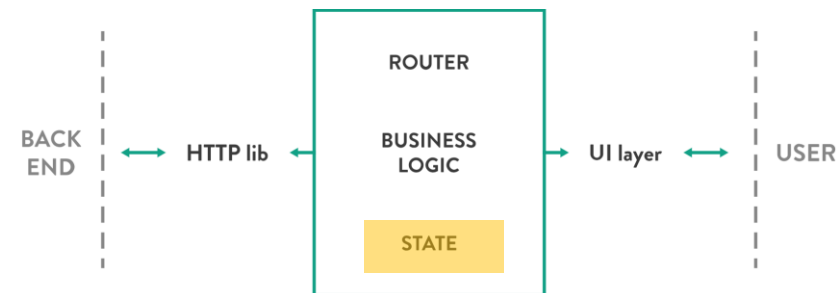


# Single Page Applications

- Algumas características do **UI layer**
  - Qualquer coisa renderizável no navegador
    - Layouts, texto, botões, qualquer coisa.
    - É uma interação bidirecional com o usuário
      - Inclui eventos acionados com a interação do usuário
- Relação muito próxima para as ferramentas que tratam
  - Gerência de estado (State)
  - Lógica da interface com o usuário (Business logic)

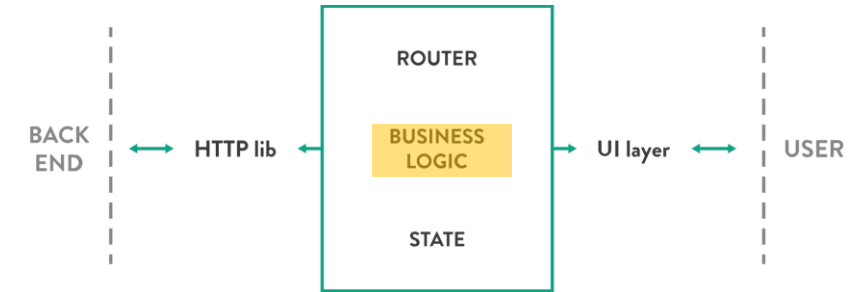


# Single Page Applications



- Algumas características do **State**
  - Gerenciamento de estados
    - Muito utilizados para a criação/adequação da interface com o usuário
    - Tudo não estético mostrado em seu aplicativo é, na verdade, dado.
- Cada tecnologia implementa uma forma de gerenciamento de estado
  - React
    - Permite criar componentes com estado
    - Fornece uma API útil para gerenciamento de estado

# Single Page Applications



- Algumas características do **Business logic**
  - É a ponte entre o front-end e o back-end
    - Qualquer lógica necessária a ser aplicada aos dados e interações.
  - Extremamente ligadas ao estado/modelo
    - Uma calculadora tem sua proposta de lógica associada aos estados e modelos
    - Um loja virtual tem sua proposta de lógica associada aos estados e modelos
    - Um jogo online tem sua proposta de lógica associada aos estados e modelos

# Tecnologias habilitadoras

- **Algumas tecnologias e frameworks**



- Angular:
  - <https://angular.io/tutorial>



- Single Page Apps for GitHub Pages:
  - <https://github.com/rafrex/spa-github-pages>



- Vue.js:
  - <https://vuejs.org/v2/examples/index.html>



- React:
  - <https://pt-br.reactjs.org/docs/create-a-new-react-app.html>

# Tecnologias habilitadoras

- **Algumas tecnologias e frameworks**



- Angular

- É um framework de JavaScript mantido pelo Google
  - Framework é
    - Recurso elaborado para abstrair várias etapas de construção de uma solução
    - Normalmente apoiado em biblioteca de funções
- Frequentemente usado para criar SPAs
- O site oficial do Angular inclui um tutorial passo a passo chamado Tour of Heroes
  - Demonstra como criar uma aplicação SPA completa



# Tecnologias habilitadoras

- **Algumas tecnologias e frameworks**



- Single Page Apps for GitHub Pages
  - É um repositório no GitHub
    - Inclui um exemplo de aplicação SPA
      - Usa o React para criar uma interface de usuário para exibir dados do GitHub.

# Tecnologias habilitadoras

- **Algumas tecnologias e frameworks**



- **Vue.js**

- É um framework popular para criar SPAs.
- O site oficial do Vue.js inclui um exemplo de aplicação SPA
  - Demonstra como usar o Vue.js para criar uma interface de usuário dinâmica e responsiva.

# Tecnologias habilitadoras

- Algumas tecnologias e frameworks



- React

- É uma biblioteca popular para criar interfaces de usuário em JavaScript
- É **frequentemente** usado para criar SPAs.
- A documentação oficial do React inclui
  - Exemplos práticos de como criar uma SPA usando o React
    - Aplicação de lista de tarefas (ToDo)
    - Aplicação de busca de filmes

## Resumo do que vimos até agora



# SPA

aplicações web que operam em uma única página, onde o conteúdo é atualizado dinamicamente sem que haja uma recarga completa da página.

criar uma experiência de usuário mais rápida e responsiva, carregando todo o conteúdo necessário em uma única página, em vez de recarregar a página inteira a cada interação do usuário.

# Análise e Desenvolvimento de Sistemas

- Desenvolvimento de sistemas *front end*



# Aula 2 (parte 2)

- Fundamentação da tecnologia React

**O que você  
vai aprender  
nessa aula**



- Fundamentos da tecnologia React

## Resumo do que vimos até agora



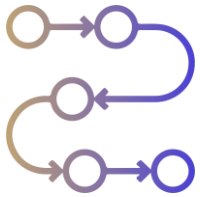
- Entendimento sobre a SPA



## O que você vai precisar para acompanhar essa aula



- Nenhum recurso específico
- Atenção



## Dinâmica

- Exploração de conceitos
- Exemplificações

# Conceitos

- React
  - É uma biblioteca JavaScript
    - Usada para criar interfaces com usuários
    - Baseada em blocos reutilizáveis (componentes) vindas de partes menores e mais simples.
  - Desafio da biblioteca
    - Melhorar atualização dos sites em resposta a eventos
    - Eventos normalmente requerem atualizações no site
    - Eventos que normalmente causam alguma atualização
      - Entrada de um usuário
      - Um novo dado vindo de outra fonte de dado
        - Um site, base de dados ou sensor (e.g. GPS)
- React propõem a atualização a partir de “*programação reativa*”

# Conceitos

- Interface com o usuário

- Aplicações web são normalmente construídas a partir do padrão MVC

- Elementos MVC

- Model → Camada de dados
      - Controller → Organiza/regra/facilita a comunicação com a camada de dados
      - View → o que é o usuário vê e interage

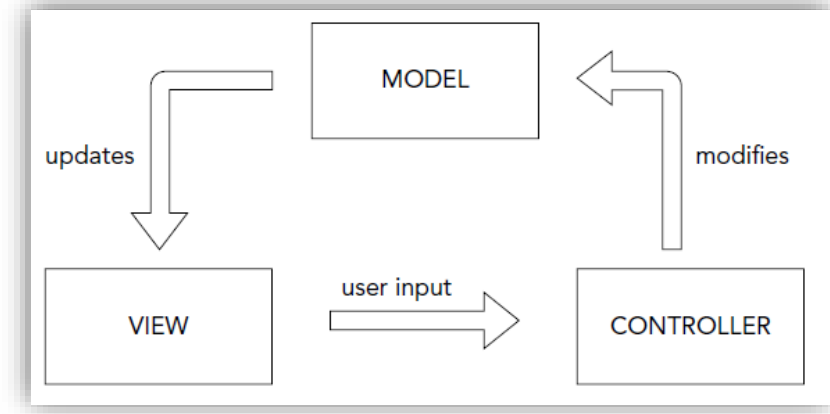
- Fluxo usual

- View envia *inputs* para o controller
      - Controller repassa dados entre a *view* e a camada de dados

- Foco da tecnologia React

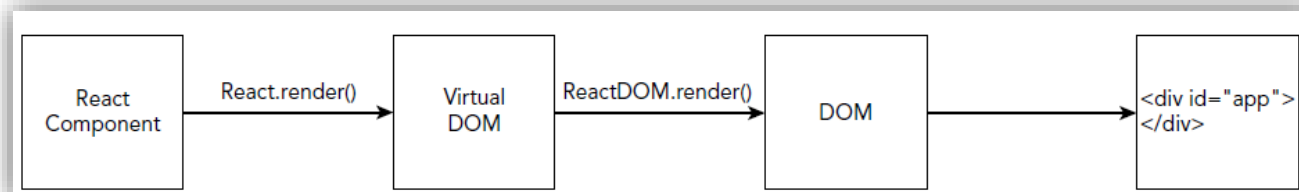
- View

- Capturar dados de entrada
    - Apresentar dados de alguma forma ao usuário



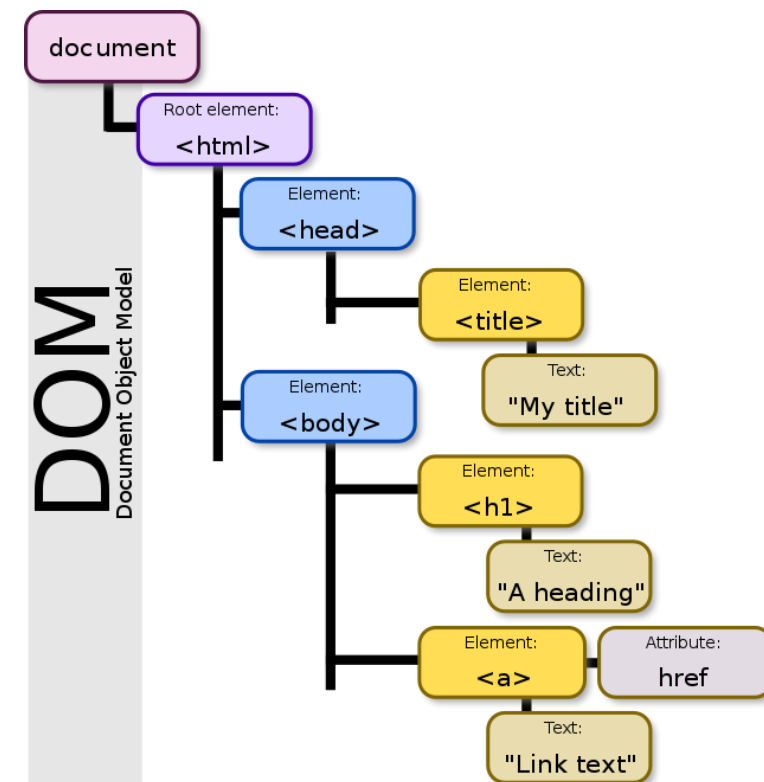
# Conceitos

- React
  - Biblioteca responsável pela renderização do componente
    - Independente do dispositivo alvo (mobile, tablet, web, etc)
    - Renderizar é diferente de apresentar algo
      - No contexto de React renderizar significa prepara para apresentar, para ser visualizado
- ReactDOM
  - Biblioteca responsável pela apresentação de componentes renderizados
    - Voltada para navegadores
  - Disponibiliza várias funções para interfacear React e navegadores
    - Principal função é ReactDOM.render
  - Opera sobre o Virtual DOM



# Conceitos

- DOM
  - Document Object Model
    - Representação interna de um documento web
  - Converte HTML, estilos e conteúdo em nodos
    - Nodos podem ser manipulados usando uma linguagem de programação
    - JavaScript é a linguagem adotada em projetos web
      - Exemplo funções para operação
        - getElementById
        - innerHTML
  - Modificar o DOM modifica a página
  - Modificações na página modificam o DOM



# Conceitos

- DOM
  - Manipulação do DOM com JS é lento e ineficiente
    - Sempre que o DOM for modificado, o navegador precisará
      - Investigar se a página precisará ser redesenhada
      - Em seguida redesenhar a página
    - Uso de algumas funções possuem nomenclaturas muito longas
      - e.g. `Document.getElementsByClassName`
      - Daí o surgimento de várias bibliotecas, tal como jQuery
        - Proposta facilitou o trabalho do programadores web mas....
          - Lhes entregou a responsabilidade de definir quando e como o DOM aconteceriam
          - Resultado levou a UI lentas para baixar ou mesmo reagir a interações com o usuário
        - React assumiu a responsabilidade de quando e como realizar tais modificações

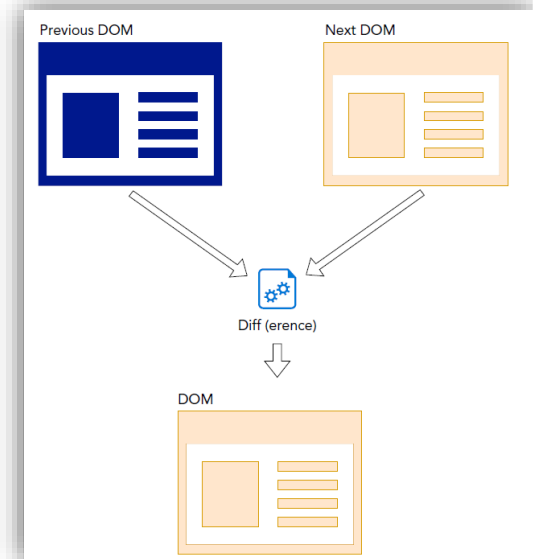


# Conceitos

- Virtual DOM

- Proposta de funcionamento

- (1) O programador elabora um código React para renderizar uma UI
      - Isto resulta em um elemento React sendo retornado
    - (2) O método de renderização do ReactDOM entra em ação
      - Cria uma representação simples e leve do elemento em memória (um virtualDOM - VDa)
    - (3) ReactDOM “escuta” a eventos que requerem atualização da página
      - Quando ocorre um evento
        - O método ReactDOM.render cria uma nova representação da página em memória (VDn)
        - Compara a nova Virtual DOM (VDn) com a antiga (VDa) e calcula a diferença
          - Sistema chamado de reconciliação
        - Se forem diferentes, aplica no DOM da forma mais eficiente o possível

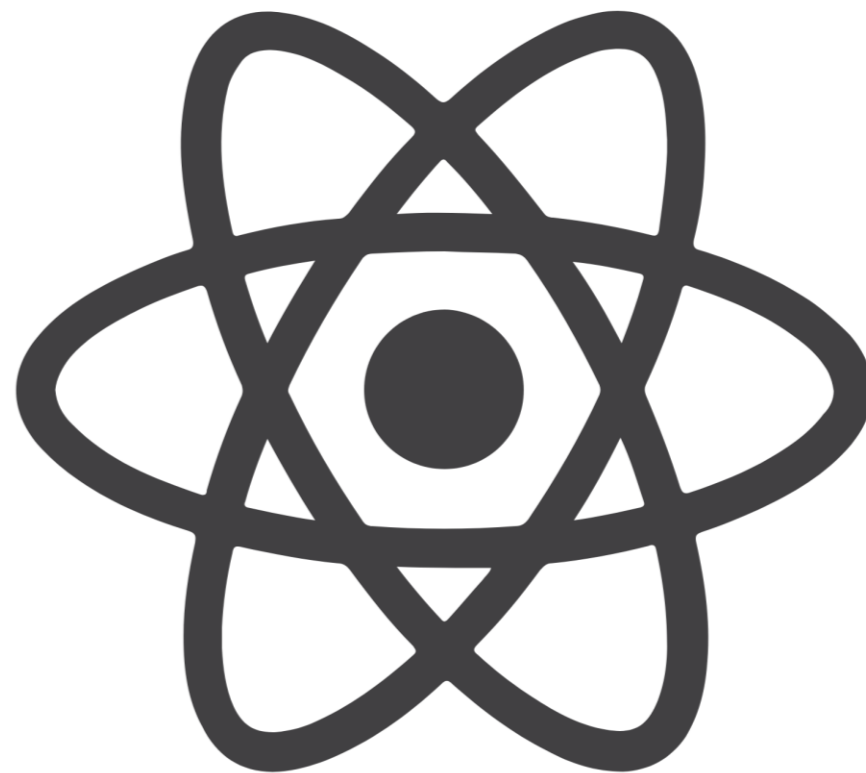




# Conceitos

---

- Filosofia da tecnologia
  - Focado em componentes
- Composição vs Herança
- React é declarativo
- React é Idiomático



# React JS

# Conceitos

[CLICK HERE](#)

- Filosofia da tecnologia
  - **Focado em componentes**
    - Biblioteca propicia a criação e composição de componentes para construir a UI
    - Componentes elaborados como peças independentes
      - Podem ser reusadas
      - Podem passar dados entre eles
  - Um componente
    - Um simples botão
    - Uma barra de navegação construída a partir de coleção de botões e dropdown
  - Decisões do desenvolvedor
    - Avaliar questões de estilização na elaboração do componente
    - Deve explorar o princípio de responsabilidade única (SOLID)
      - Sua existência deve ter uma tarefa específica a executar

# Conceitos

- Filosofia da tecnologia
  - **Composição vs Herança**
    - Herança é usada em POO para criar variações de classes
      - Uma subclasse herda as propriedades definidas em uma superclasse
  - Composição tem maior foco no reuso de um componente sem especializá-lo
    - Sua funcionalidade pode ser ajustada a partir da passagem de parâmetros

```
function Button(props){
  return(
    <button onClick={props.handleClick}>{props.label}</button>
  );
}

function Dialog(props){
  return(
    <div className="dialogStyle">{props.message}</div>
  )
}

function WelcomeDialog(props){
  return(
    <Dialog message="Welcome to our app!" />
  )
}
```

# Conceitos

- Filosofia da tecnologia
  - **React é declarativo**
    - Maioria das linguagens é Imperativa
  - Paradigma Imperativo
    - Descreve como algo tem de ser feito
  - Paradigma Declarativo
    - Descreve o que quer que seja feito



- Abordagem Imperativa
  - Pegue duas fatias de pão
  - Coloque as fatias de pão lado a lado
  - Escolha uma das fatias de pão
  - Sobre a fatia escolhida
    - Passe manteiga nela
    - Coloque uma fatia de presunto nela
    - Coloque uma fatia de queijo
    - Coloque a outra fatia de pão
  - Coloque o resultado em um prato

- Abordagem declarativa
  - Um sanduíche de presunto e queijo num prato

# Conceitos

- Filosofia da tecnologia

- **React é Idiomático**

- É uma biblioteca pequena

- Possui funcionalidades limitadas quando comparada a outras bibliotecas JS

- Possui vários conceitos e métodos intrínsecos a biblioteca

- Apesar disto, components React são JavaScript

- O lado bom é que

- Quanto mais souber JavaScript, melhor será sua habilidade com React
      - Quanto mais souber React, mais facilmente absorverá os conceitos de JavaScript



# Casos de uso da tecnologia



- Facebook - Desenvolveu o React para uso interno e agora o usa em seus aplicativos, incluindo o Facebook, Instagram e WhatsApp.



- Netflix - Usa React para construir sua plataforma de streaming e melhorar a experiência do usuário.



- Airbnb - Usa React em sua plataforma de reserva de viagens para fornecer uma experiência de usuário intuitiva e responsiva.



- Uber - Usa React em seus aplicativos de motorista e passageiro para fornecer uma experiência de usuário rápida e confiável.

# Recursos mínimos de um projeto React

- Arquivos de projeto
  - Arquivos React (JSX)
  - Arquivo HTML de entrada
- Ferramenta de transpilação
  - Ferramenta de tradução entre linguagens
- Ferramenta de empacotamento
  - Ferramenta de compactação de arquivos de projeto



**webpack**

# Arquivo JSX

- Tipo de arquivo explorado em projetos React
  - São opcionais, mas muito empregados
  - Origem do nome
    - Java XML
  - Agrupam recursos essenciais
    - lógica de marcação
      - Tags HTML
    - lógica de renderização
      - Comandos JavaScript like
  - Objetivo
    - Criação de interfaces com o Usuário

```
import React from "react";
function Login(){
  const handleSubmit = (e)=>{
    e.preventDefault();
    console.log(`logging in ${e.target[0].value}`);
  }
  return (
    <form id="login-form" onSubmit={handleSubmit}>
      <input
        type="email"
        id="email"
        placeholder="E-Mail Address"/>
      <input
        type="password"
        id="password"/>
      <button>Login</button>
    </form>
  );
}
export default Login;
```



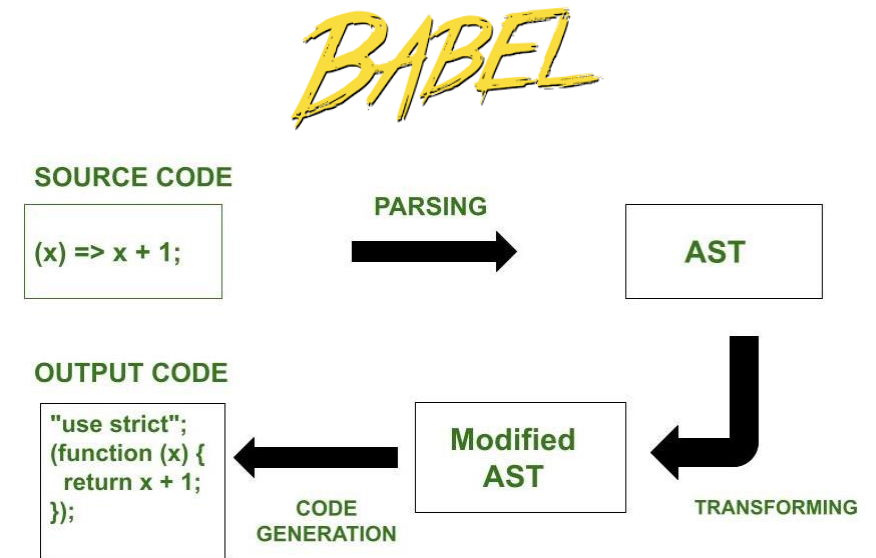
# O Babel

## • Características

- É um tipo de compilador
  - Tradutor de código fonte ou "**transcompiler**"
    - Converte o código fonte entre linguagens de programação

## • Transpilador vs Compilador

- Transpilador
  - Converte entre linguagens "semelhantes"
- Compilador tradicional
  - De código fonte
  - Para linguagem de baixo nível

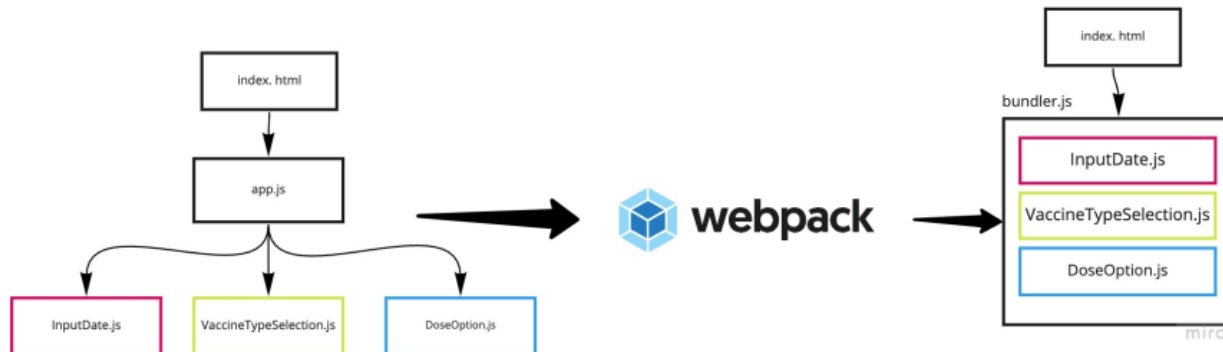


```

3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-
9         logo" alt="logo" />
10        Edit <code>src/App.js</code> and
11        save to reload.
12      </p>
13      <a
14        className="App-link"
15        href="https://reactjs.org"
16        target="_blank"
17        rel="noopener noreferrer"
18      >
19        Learn React
20      </a>
21    </header>
22  </div>
23
24
25
14 function App() {
15   return
16   /*#__PURE__*/React.createElement("div", {
17     className: "App"
18   },
19   /*#__PURE__*/React.createElement("header",
20   {
21     className: "App-header"
22   },
23   /*#__PURE__*/React.createElement("img", {
24     src: _logo.default,
25     className: "App-logo",
26     alt: "logo"
27   })), /*#__PURE__*/React.createElement("p",
28   null, "Edit ",
29   /*#__PURE__*/React.createElement("code",
30   null, "src/App.js"), " and save to
31   reload."),
32   /*#__PURE__*/React.createElement("a",
33   {
34     className: "App-link",
35     href: "https://reactjs.org",
  
```

# O Webpack

- Empacotador de módulos estáticos
  - Processa os recursos do projeto
    - Arquivos texto (html, css, js)
    - Imagens
  - Cria um gráfico de dependência
    - Mapeia os módulos necessários
    - Gera um ou mais pacotes (bundle)



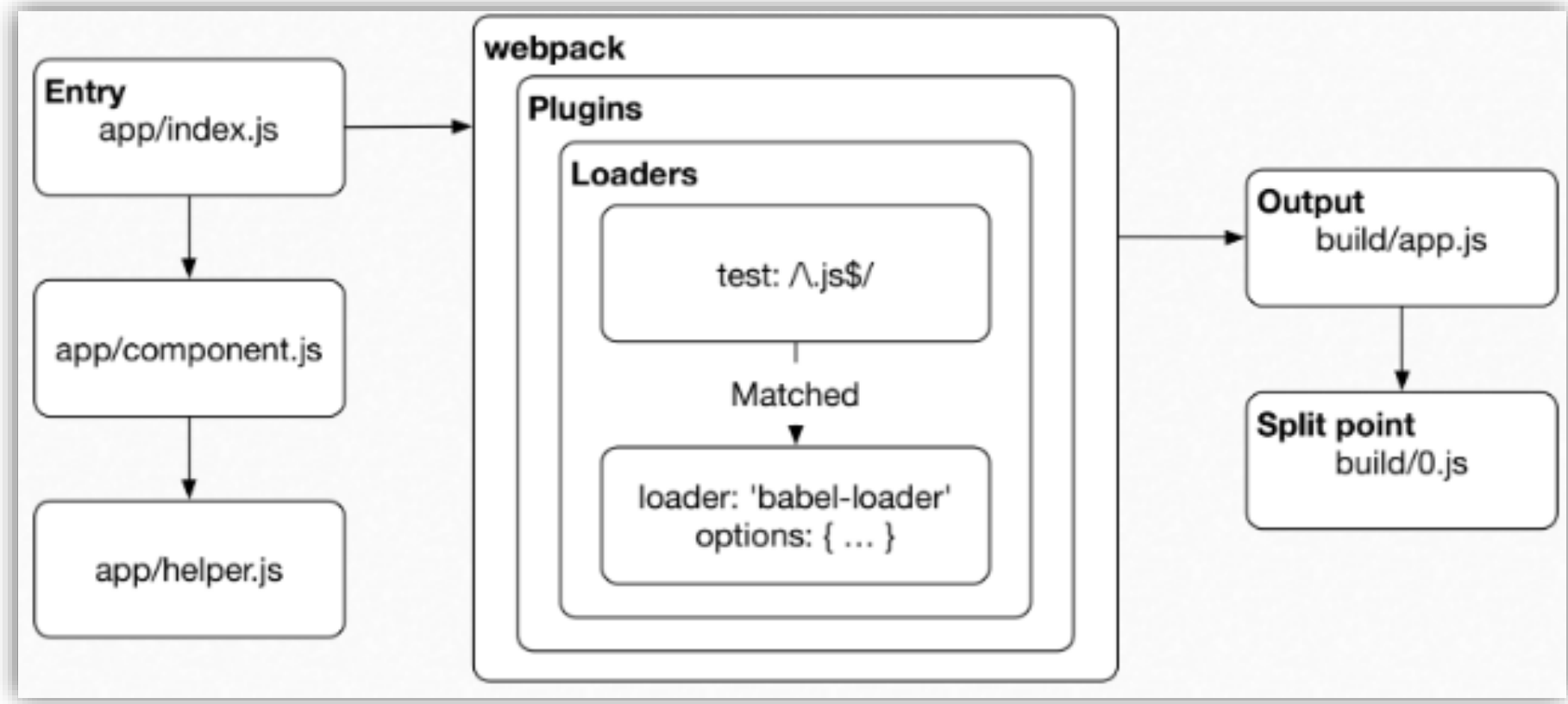
```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>React App</title>
5   </head>
6   <body>
7     <div id="root"></div>
8     <script src="./dist/bundle.js"></script>
9   </body>
10 </html>
  
```

```

1 /*! For license information please see bundle.js.LICENSE.txt */
2 (() =>{"use strict";var e={448:(e,n,t)>{var r=t(294),l=t(840);function a(e){for(var n=
"https://reactjs.org/docs/error-decoder.html?invariant="+e,t=1;t<arguments.length;t++)n+="%args["+
+encodeURIComponent(arguments[t])+"];return"}Minified React error #"+e+"; visit "+n+" for the full
message or use the non-minified dev environment for full errors and additional helpful warnings."}
var u=new Set,o={};function i(e,n){s(e,n),s(e+"Capture",n)}function s(e,n){for(o[e]=n,e=0;e<n.
length;e++)u.add(n[e])}var c={"undefined"==typeof window||void 0===window.document||void 0===
window.document.createElement(),f=Object.prototype.hasOwnProperty,d=
/^[:A-Z_a-z\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u02FF\u0370-\u037D\u037F-\u1FFF\u200C-\u200D\u2070-\u2
18F\u2C00-\u2FEF\u3001-\uD7FF\uF900-\uFDCF\uFDF0-\uFFFF]:A-Z_a-z\u00C0-\u00D6\u00D8-\u00F6\u00F8-\u
02FF\u0370-\u037D\u037F-\u1FFF\u200C-\u200D\u2070-\u218F\u2C00-\u2FEF\u3001-\uD7FF\uF900-\uFDCF\uF
DF0-\uFFFFD\-.0-9\u00B7\u0300-\u036F\u203F-\u2040]*$/;p={},m={};function h(e,n,t,r,l,a,u){this.
acceptsBooleans=2===n||3===n||4===n,this.attributeName=r,this.attributeNamespace=l,this.
mustUseProperty=t,this.propertyName=e,this.type=n,this.sanitizeURL=a,this.removeEmptyString=u}var g
={};"children dangerouslySetInnerHTML defaultValue defaultChecked innerHTML
suppressContentEditableWarning suppressHydrationWarning style".split(" ").forEach((function(e){g[e
]=new h(e,0,!1,e,null,!1,!1)})),["acceptCharset","accept-charset"],["className","class"],["
htmlFor","for"],["httpEquiv","http-equiv"]].forEach((function(e){var n=[0];g[n]=new h(n,1,!1,e[1
],null,!1,!1)})),["contentEditable","draggable","spellCheck","value"].forEach((function(e){g[e]=new
h(e,2,!1,e.toLowerCase(),null,!1,!1)})),["autoReverse","externalResourcesRequired","focusable",
"preserveAlpha"].forEach((function(e){g[e]=new h(e,2,!1,e,null,!1,!1)})),["allowFullScreen async
autoFocus autoPlay controls default defer disabled disablePictureInPicture disableRemotePlayback
formNoValidate hidden loop noModule noValidate open playsInline readOnly required reversed scoped
seamless itemScope".split(" ").forEach((function(e){g[e]=new h(e,3,!1,e.toLowerCase(),null,!1,!1
)})),["checked","multiple","muted","selected"].forEach((function(e){g[e]=new h(e,3,!0,e,null,!1,!1
)})),["capture","download"].forEach((function(e){g[e]=new h(e,4,!1,e,null,!1,!1)})),["cols","rows",
"size","span"].forEach((function(e){g[e]=new h(e,6,!1,e,null,!1,!1)})),["rowSpan","start"].forEach
((function(e){g[e]=new h(e,5,!1,e.toLowerCase(),null,!1,!1)}));var v=/[-:]{1,3}/g;function y(e){
return e[1].toUpperCase()}function b(e,n,t,r){var l=g.hasOwnProperty(n)?g[n]:null;(null!=l?!0!=l.
type:r||!(2<n.length)||"o"!=n[0]&&"O"!=n[0]||"n"!=n[1]&&"N"!=n[1])&&(function(e,n,t,r){if(null
==n||function(e,n,t,r){if(null!=t&&0===t.type)return!1;switch(typeof n){case"function":case
  
```

# Relação entre os recursos



## Resumo do que vimos até agora



React uma biblioteca JavaScript para criação de interfaces de usuário (UI).

O React foi projetado para ser declarativo

O Virtual DOM do React é usado para fazer essa atualização de forma mais eficiente.

Grandes corporações fazem uso do React:  
Facebook, Netflix, Airbnb, Uber, entre outras.



# Análise e Desenvolvimento de Sistemas

- Desenvolvimento de sistemas *front end*

# Aula 2 (parte 3)

- Explorando JSX

# O que você vai aprender nessa aula



- Sintaxe básica de JSX

## Resumo do que vimos até agora



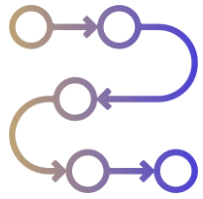
- Conceitos sobre SPA
- Como funciona a tecnologia React



## O que você vai precisar para acompanhar essa aula



- Ferramental de desenvolvimento
  - IDE
  - Node.JS
- Atenção



## Dinâmica

- Prática com exemplos

# Arquivos JSX

- O JSX (JavaScript XML)
  - É uma extensão de sintaxe usada no React
    - Descreve a estrutura e a aparência dos componentes de interface do usuário
  - Permite escrever código semelhante a HTML dentro do JavaScript
    - Tornando a construção de interfaces de usuário mais intuitiva e fácil de ler.
- Parte fundamental do React
  - Permite que você defina
    - A estrutura dos componentes
    - Os elementos visuais que compõem a interface do usuário

```
import React from 'react';

const App = () => {
  return <p>Hello, JSX!</p>;
};

export default App;
```

# Conceitos

- JSX não é HTML
  - Vendo o código
    - Conhecedores de HTML
      - Reconherão a estrutura a ser entregue
    - Conhecedores de JavaScript
      - Entenderão que o função resultara em é um erro
  - O que temos é um JSX
    - Um componente React bem elaborado
    - Os Markups internos não são HTML
    - São uma extensão de JavaScript

```
import React from "react";
function Login(){
  const handleSubmit = (e)=>{
    e.preventDefault();
    console.log(`logging in ${e.target[0].value}`);
    // do something else here
  }
  return (
    <form id="login-form" onSubmit={handleSubmit}>
      <input
        type="email"
        id="email"
        placeholder="E-Mail Address"/>
      <input
        type="password"
        id="password"/>
      <button>Login</button>
    </form>
  );
}
export default Login;
```

# Conceitos

- React tem componentes embutidos
  - Como dito, JSX não é primordial para a elaboração de componentes

Código HTML

```
<label class="inputLabel">Search:  
  <input type="text" id="searchInput">  
</label>
```

Código React sem o uso de JSX

```
using React.createElement;  
React.createElement("label", {className: "inputLabel"}, "Search:",  
  React.createElement("input", {type: "text", id: "searchInput"}));
```

# Conceitos

- React tem componentes embutidos
  - Elementos em HTML5
  - Pode-se usar qualquer nome de elemento HTML5
    - React produzirá esse elemento HTML5
  - Por exemplo
    - Componente React resultando na renderização da seguinte parte da marcação HTML

Código HTML

```
<label class="inputLabel">Search:  
  <input type="text" id="searchInput">  
</label>
```

Código JSX

```
<label className="inputLabel">Search:  
  <input type="text" id="searchInput">  
</label>
```

# Sintaxe básica de JSX

- JSX é XML para JavaScript
  - Regras JSX são equivalentes a XML
    - Todos elementos devem ser fechados
  - Elementos que não podem ter nodos filhos
    - Chamados elementos fechados
      - Devem ser fechados com *slash*
    - Exemplo: `<br />`, `<img />`, `<input />`, `<link />`
- *Atributos que são strings devem vir entre aspas*
- *Elementos HTML usados no JSX*
  - *Devem ser escritos em letras minúsculas*
    - `<input />` é um elemento HTML
    - `<Input />` deve ter sido elaborado como um componente

```
import React from 'react';

function App() {
  return (
    <div>
      
      <br />
      <input type="text" placeholder="Digite algo" />
      <br />
      <a href="https://www.example.com">Link</a>
    </div>
  );
}

export default App;
```

# Sintaxe básica de JSX

- Palavras reservadas

- Ao traduzir de JSX para JavaScript
  - Pode haver sobreposição na linguagem alvo
    - Casos comum
      - nome de elemento
      - nome de atributo usado
    - Resulta em erro do programa compilado

- Para evitar o erro

- Certos nomes de atributos HTML conflitantes são renomeados:
  - Exemplo
    - O atributo class torna-se className.
    - O atributo for torna-se htmlFor.

```
import React from 'react';

function App() {
  return (
    <div>
      <label htmlFor="my-input">Username:</label>
      <input type="text" id="my-input" />
      <div className="my-class">Conteúdo com classe CSS</div>
    </div>
  );
}

export default App;
```



# Sintaxe básica de JSX

- Uso de chaves {}
  - Utilizado quando se deseja
    - Incluir uma variável
    - Incluir um trecho de código JavaScript no JSX que não deve ser interpretado

```
import React from 'react';

function App() {
  const name = 'John Doe';

  return <h1>Hello, {name}!</h1>;
}

export default App;
```

```
import React from 'react';

function App() {
  const x = 5;
  const y = 10;

  return <p>A soma de {x} e {y} é igual a {x + y}.</p>;
}

export default App;
```

# Sintaxe básica de JSX

- Inserção de comentários
  - Comentários HTML não funcionam no JSX
    - Deve-se usar a sintaxe de comentário de bloco JavaScript (`/*` e `*/`)
    - Comentário só funciona se estiver entre chaves

```
function App(){  
  return (  
    <h1>  
      /* Todo: Make this header dynamic */  
      Welcome to My Website  
    </h1>  
  )  
}  
export default App;
```

# Sintaxe básica de JSX

- Retornando um resultado
  - Componentes devem retornar uma única estrutura

```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Título</h1>
      <p>Conteúdo</p>
    </div>
  );
}

export default App;
```

```
import React from 'react';

function App() {
  return (
    <section>
      <h1>Título</h1>
      <p>Conteúdo</p>
    </section>
  );
}

export default App;
```

```
import React from 'react';

function App() {
  return (
    <ul>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ul>
  );
}

export default App;
```

```
import React from 'react';

function App() {
  return (
    <table>
      <thead>
        <tr>
          <th>Coluna 1</th>
          <th>Coluna 2</th>
          <th>Coluna 3</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>Dado 1</td>
          <td>Dado 2</td>
          <td>Dado 3</td>
        </tr>
      </tbody>
    </table>
  );
}

export default App;
```


# Sintaxe básica de JSX

- Retornando um resultado
  - Mais de um resultado não é válido na linguagem

```
import React from 'react';

function App() {
  return (
    <h1>Título</h1>
    <p>Conteúdo</p>
  );
}

export default App;
```



```
import React from 'react';

function App() {
  return (
    <h1>Título</h1>
    <p>Conteúdo</p>
    <div>Outro elemento</div>
  );
}

export default App;
```



```
import React from 'react';

function App() {
  return (
    <div>
      <h1>Título</h1>
      <p>Conteúdo</p>
    </div>
    <div>Outro elemento</div>
  );
}

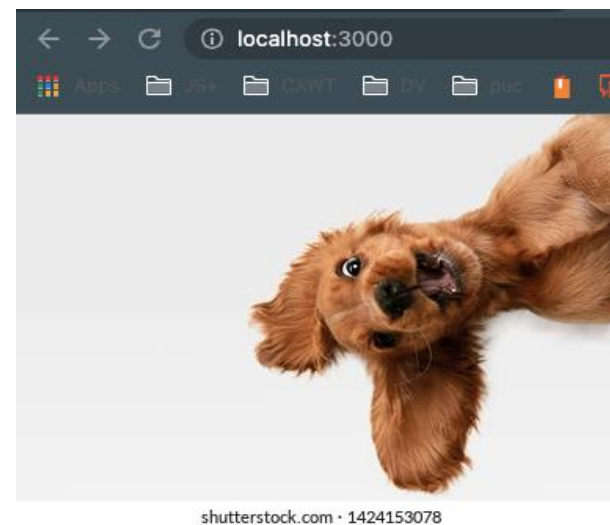
export default App;
```



# Sintaxe básica de JSX

- **Retornando um resultado**
  - Tags HTML são um bom mecanismo para unificar os elementos
  - Pode-se usar o conceito de fragment do React
    - Produz resultado equivalente

```
1 function App() {  
2   return (  
3     // Fragmento  
4     <>  
5         
11      <ul>  
12        <li>Cão 1</li>  
13        <li>Cão 3</li>  
14        <li>Cão 3</li>  
15      </ul>  
16    </>  
17  );  
18 }  
19 export default App;  
20
```



- Cão 1
- Cão 3
- Cão 3

# Sintaxe básica de JSX

- Resumindo
  - Todo elemento descrito em JSX deve ser fechado
    - Elaboração respeitadas as regras de XML
  - Elementos JSX devem ter uma única raiz
    - Ao retornar, deve haver apenas um elemento raiz envolvendo todos os outros elementos.
- Comentários em JSX
  - São escritos dentro de chaves
  - Exemplo: `{/* Comentário JSX aqui */}`
- Use chaves `{}` para incorporar expressões JavaScript
  - Permite incorporar expressões JavaScript, como variáveis, expressões ou chamadas de função.
  - Dentro das chaves, você pode escrever qualquer código JavaScript válido.

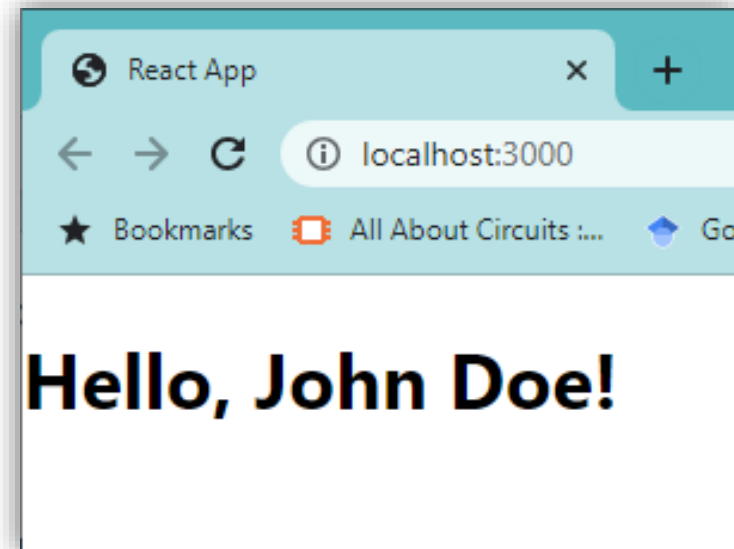
# Explorando conceitos na prática

- Vamos criar novamente um projeto
  - Faremos isso com o padrão create-react-app
    - Para criar o projeto use o comando `npx create-react-app nomedoprojeto`
    - Modificaremos o arquivo `App.js`
    - Para rodar, basta o comando `npm start`
- Exploraremos algumas alternativas de codificação
  - Uso de variáveis simples
  - Uso de lista de valores
  - Uso de funções de agregação

# Explorando casos

- Caso 1 – Uso de variáveis simples
  - Explorando o uso de variáveis

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4    const name = 'John Doe';
5
6    return <h1>Hello, {name}!</h1>;
7  }
8
9  export default App;
```

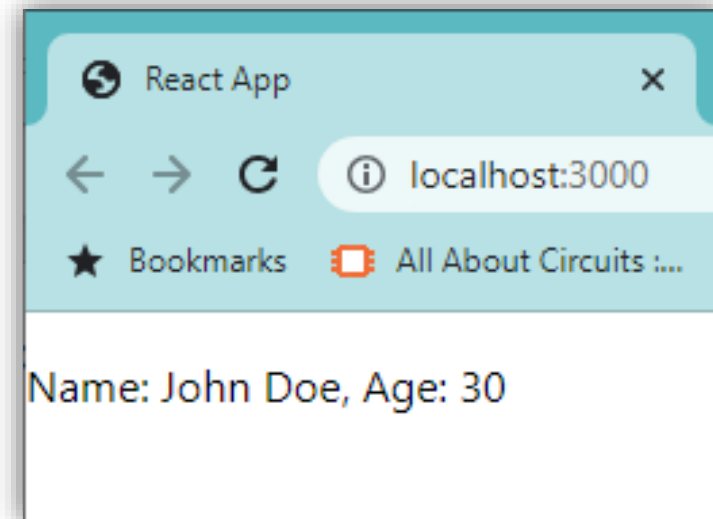




# Explorando casos

- Caso 2 – Uso de variáveis simples
  - Explorando o uso de mais de uma variável

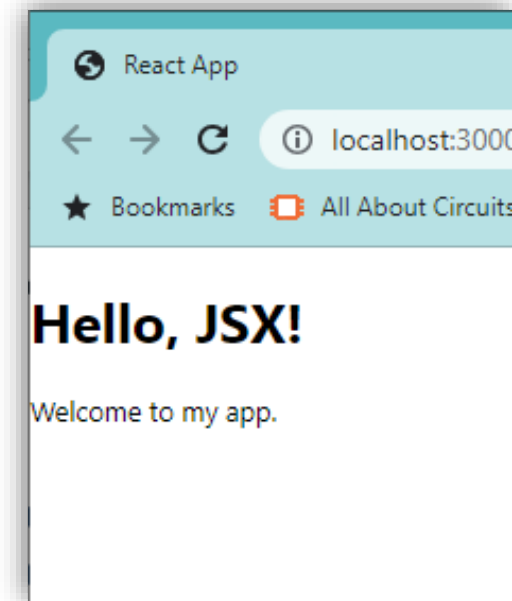
```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  const App = () => {
4    const name = 'John Doe';
5    const age = 30;
6
7    return <p>Name: {name}, Age: {age}</p>;
8  };
9
10 export default App;
```



# Explorando casos

- Caso 3 – Uso de tags em variáveis
  - Explorando o uso de mais de uma variável

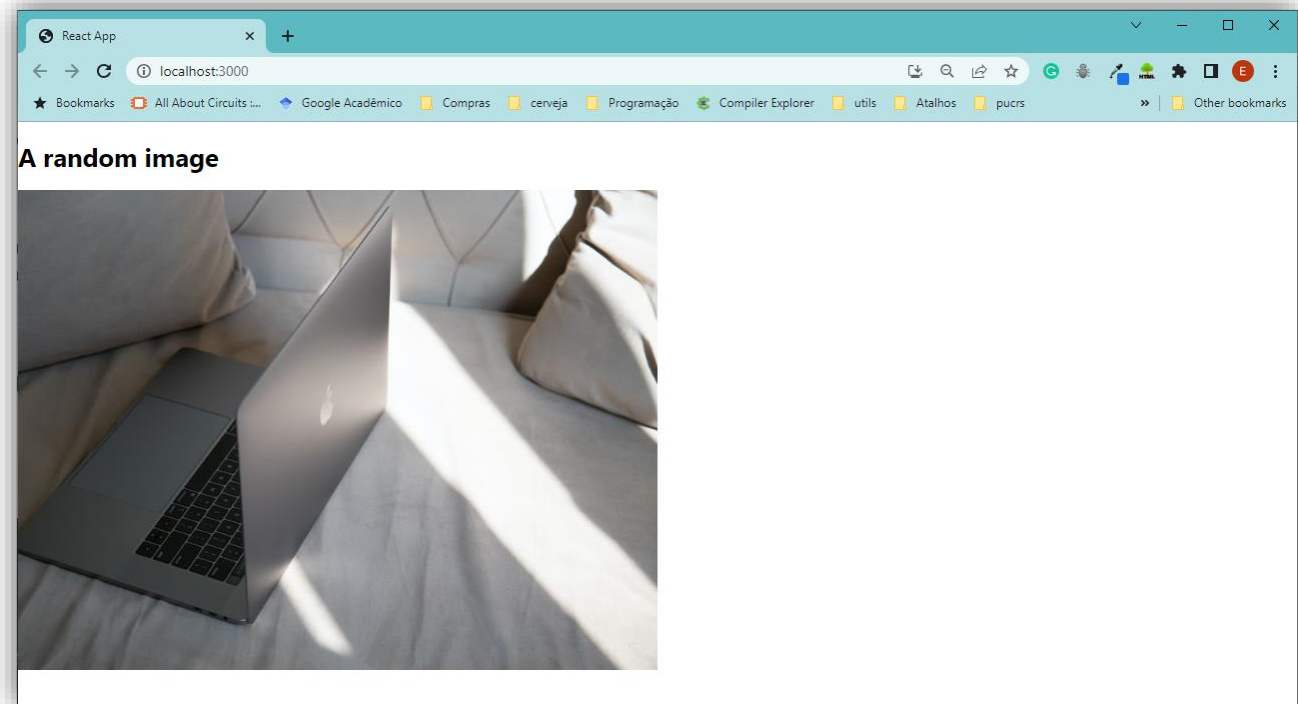
```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  const App = () => {
4    const heading = <h1>Hello, JSX!</h1>;
5    const message = <p>Welcome to my app.</p>;
6
7    return (
8      <div>
9        {heading}
10       {message}
11     </div>
12   );
13 };
14
15 export default App;
```



# Explorando casos

- Caso 4 – Uso de tags em variáveis
  - Explorando o uso de mais de uma variável

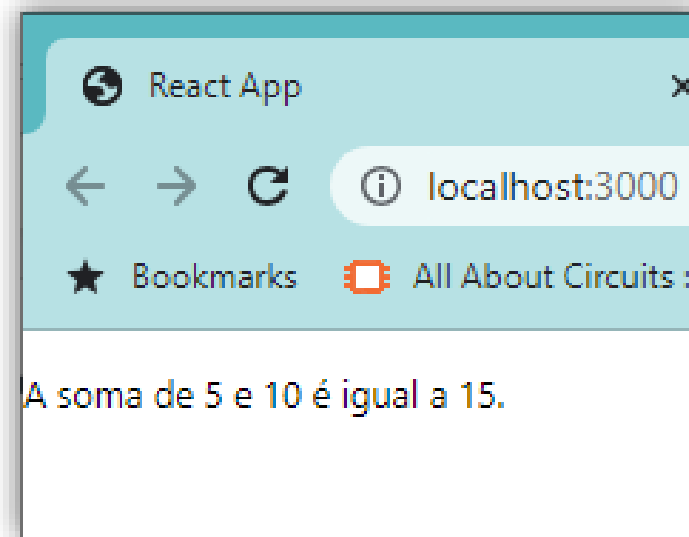
```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  const App = () => {
4    const imageUrl = 'https://source.unsplash.com/random/800x600/?react';
5    const altText = 'Example Image';
6
7    return(
8      <div>
9        <h1>A random image</h1>
10       <img src={imageUrl} alt={altText} />
11     </div>
12   );
13 };
14
15 export default App;
```



# Explorando casos

- Caso 5 – Uso de valores inteiros
  - Manipulando inteiros no contexto de React

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4    const x = 5;
5    const y = 10;
6
7    return <p>A soma de {x} e {y} é igual a {x + y}.</p>;
8  }
9
10 export default App;
```

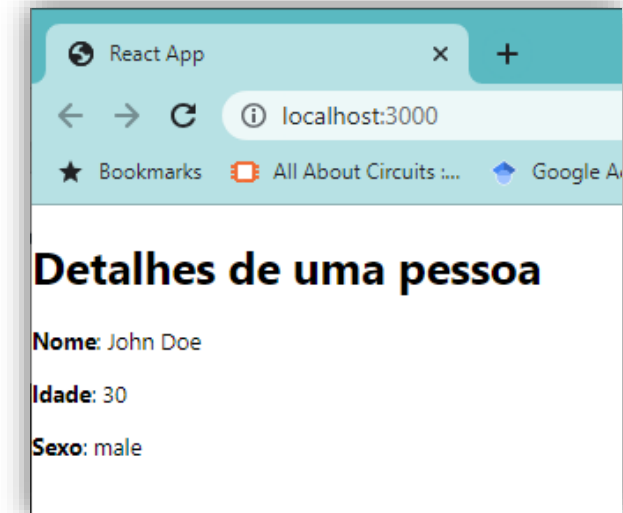


# Explorando casos

- Caso 6 – Uso de objetos para a construção da página
  - Declarando um objeto javaScript com campos
    - Criado arquivo a parte para armazenar o dado (abordagem opcional)

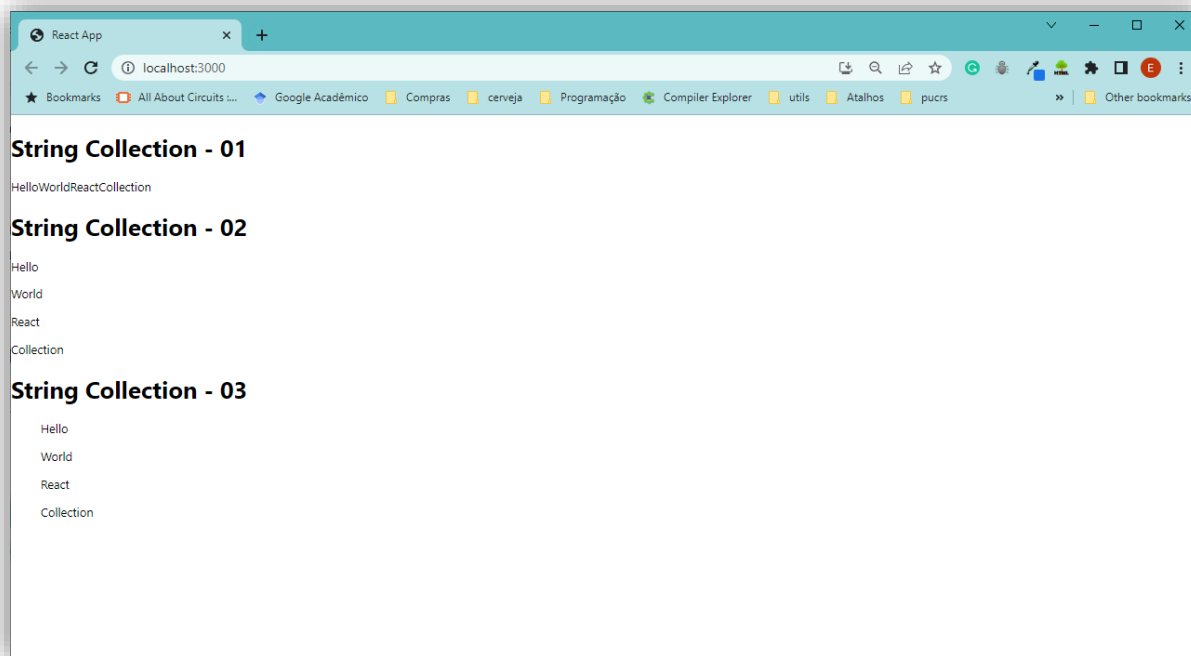
```
JS Data.js U X
src > JS Data.js > [🔗] default
1  const person = {
2    name: 'John Doe',
3    age: 30,
4    gender: 'male'
5  };
6
7  export default person;
```

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2  import person from './Data';
3
4  function App() {
5
6    return (
7      <header>
8        <h1>Detalhes de uma pessoa</h1>
9        <p><strong>Nome</strong>: {person.name}</p>
10       <p><strong>Idade</strong>: {person.age}</p>
11       <p><strong>Sexo</strong>: {person.gender}</p>
12     </header>
13
14   );
15
16 }
17
18 export default App;
```



# Explorando casos

- Caso 7 – Uso de coleções com tipos simples (string)



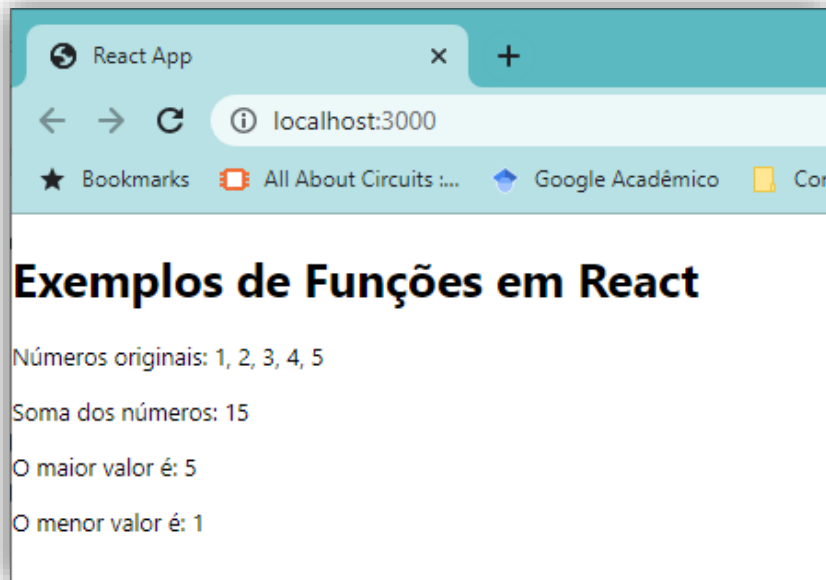
JS App.js M X

src &gt; JS App.js &gt; ...

```
1  import React from 'react';
2
3  const App = () => {
4    const strings = ["Hello", "World", "React", "Collection"];
5    let alist = []
6    for(let i=0; i<strings.length; i++)
7      alist.push(
8        <p>{strings[i]}</p>
9      );
10
11   return (
12     <div>
13       <h1>String Collection - 01</h1>
14       <p>{strings}</p>
15
16       <h1>String Collection - 02</h1>
17       <p>{alist}</p>
18
19       <h1>String Collection - 03</h1>
20       <ul>{strings.map((e)=><p>{e}</p>)}</ul>
21     </div>
22   );
23 };
24
25 export default App;
```

# Explorando casos

- Caso 8 – Uso de coleções com tipos simples (inteiros)



```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  const App = () => {
4    const numbers = [1, 2, 3, 4, 5];
5
6    // Exemplo de uso do forEach
7    numbers.forEach((number) => {
8      console.log(number);
9    });
10
11   // Exemplo de uso do reduce
12   const sum = numbers.reduce((accumulator, currentValue) => accumulator + currentValue, 0);
13
14   return (
15     <div>
16       <h1>Exemplos de Funções em React</h1>
17       <p>Números originais: {numbers.join(', ')}</p>
18       <p>Soma dos números: {sum}</p>
19       <p>O maior valor é: {Math.max(...numbers)}</p>
20       <p>O menor valor é: {numbers.reduce((min, current) => Math.min(min, current))}</p>
21     </div>
22   );
23 };
24
25 export default App;
```

## Resumo do que vimos até agora



- Entendimento sobre JSX
  - O que são
  - Ferramentas adicionais
  - Sintaxe básica



# Análise e Desenvolvimento de Sistemas

- Desenvolvimento de sistemas *front end*

# Aula 2 (parte 4)

- Renderização condicional

**O que você  
vai aprender  
nessa aula**



- Renderização condicional

## Resumo do que vimos até agora



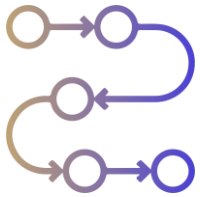
- Conceitos sobre SPA
- Como funciona a tecnologia React
- Sintaxe básica de JSX



## O que você vai precisar para acompanhar essa aula



- Ferramental de desenvolvimento
  - IDE
  - Node.JS
- O mesmo projeto base criado anteriormente
- Atenção



## Dinâmica

- Exploração de conceitos
- Exemplificações com códigos

# Conceitos

- Motivação
  - Por vezes um código precisa decidir sobre alguns detalhes
    - Apresentar um subconjunto de dados
    - Ocultar algum elemento
  - No JSX, existe forma especial de escrever JavaScript
    - Permite incorporar código JavaScript dentro dele usando chaves
    - Pode-se declarar uma variável e incorporá-la em uma tag `< >` HTML

# Conceitos

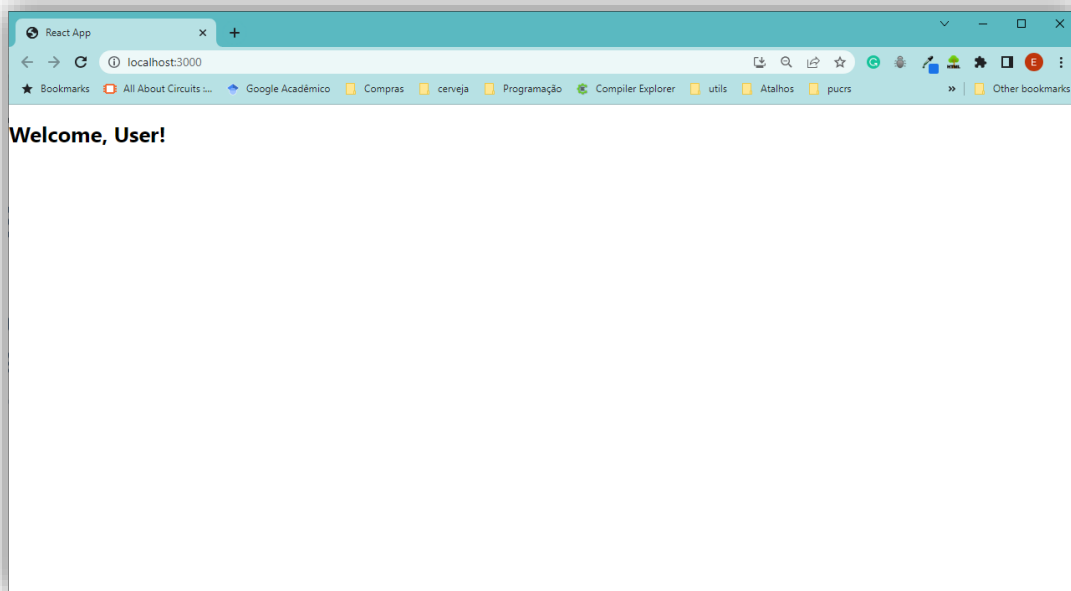
- **Renderização Condicional**

- A decisão é tomada baseada em resultado de expressões
  - São as chamadas *Renderizações Condicionais*
- Existem três formas de elaborar declarações condicionais
  - Operador if/else
  - Operador &&
  - Operador ternário ?:



# Renderização Condicional

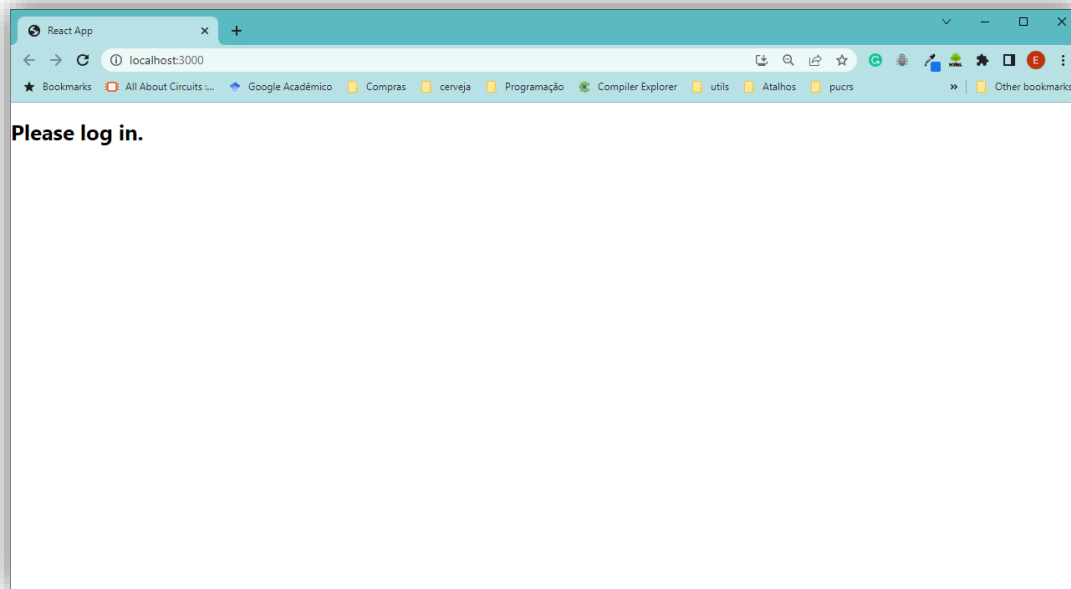
- Uso de if/else e variáveis
  - Elementos JSX podem ser atribuídos a variáveis
  - Variáveis podem ser substituídas pelos elementos dentro da instrução return



```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4      const isLoggedIn = true;
5
6      let greeting;
7
8      if (isLoggedIn) {
9          greeting = <h1>Welcome, User!</h1>;
10     } else {
11         greeting = <h1>Please log in.</h1>;
12     }
13
14     return (
15         <div>
16             {greeting}
17         </div>
18     );
19 }
20
21 export default App;
22
```

# Renderização Condicional

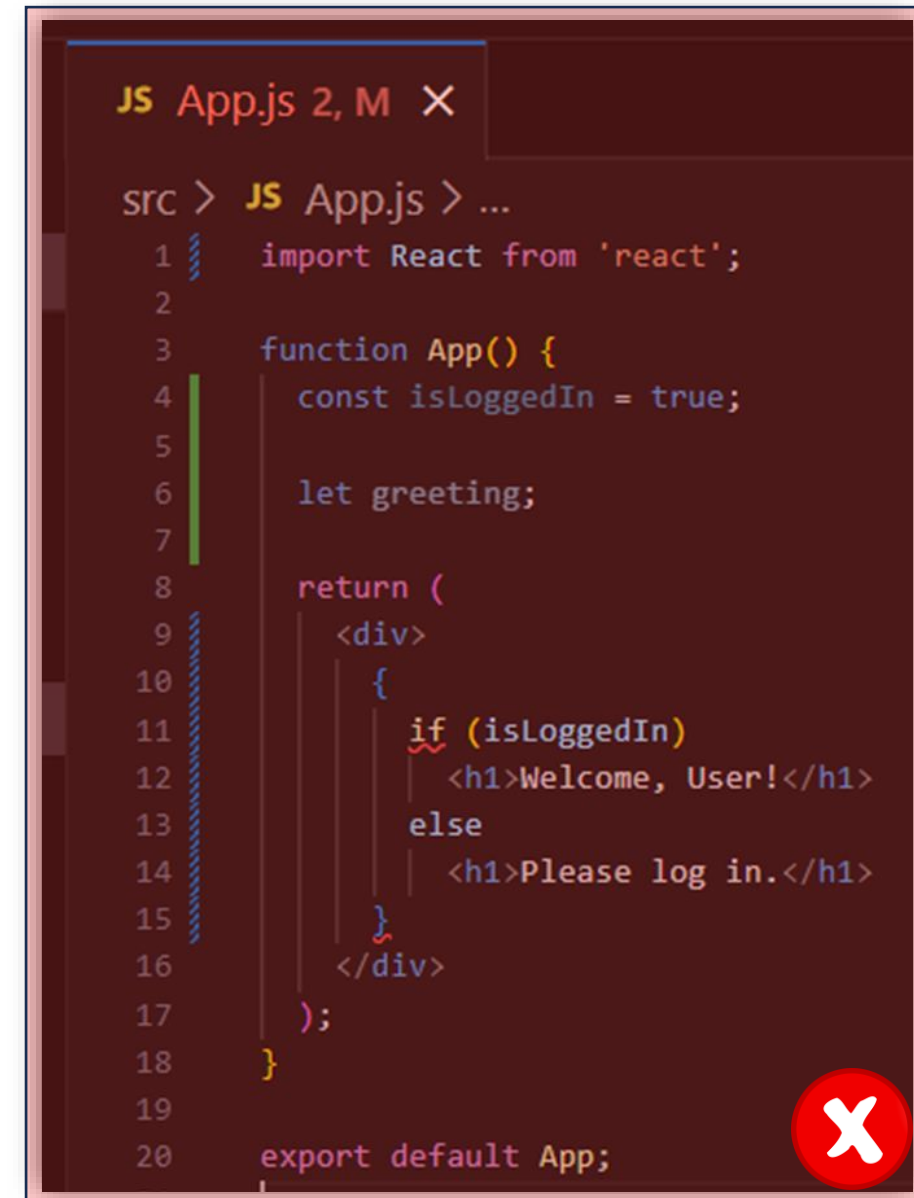
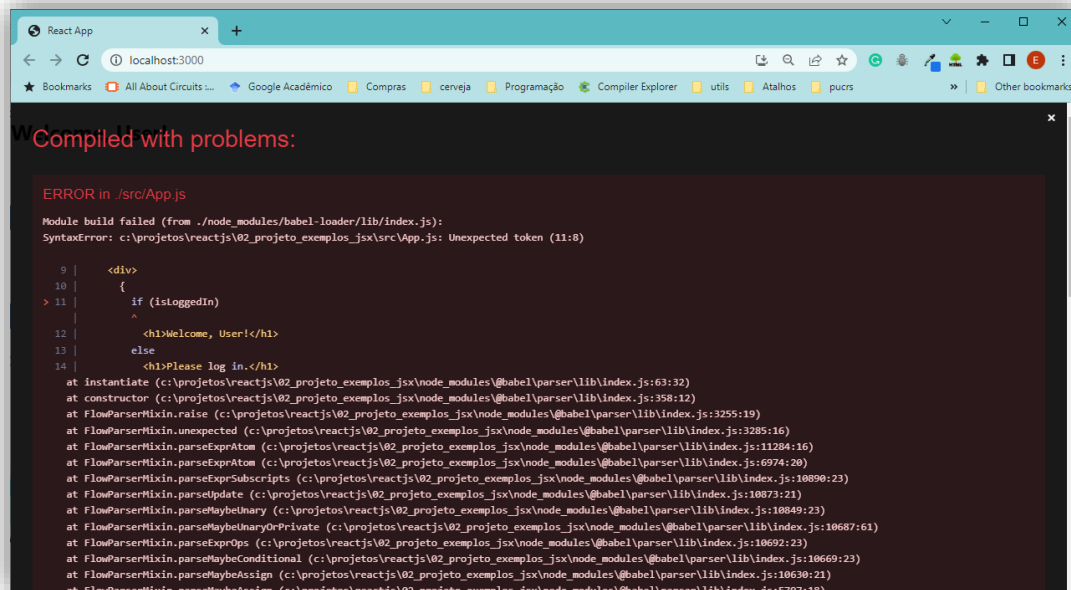
- Uso de if/else e variáveis
  - Elementos JSX podem ser atribuídos a variáveis
    - Fluxo de execução pode ser modificado



```
JS App.js M X
src > JS App.js > App > isLoggedIn
1  import React from 'react';
2
3  function App() {
4    const isLoggedIn = false;
5
6    if (isLoggedIn) {
7      return (
8        <div>
9          <h1>Welcome, User!</h1>
10         </div>
11       );
12     } else {
13       return (
14         <div>
15           <h1>Please log in.</h1>
16         </div>
17       );
18     }
19   }
20
21   export default App;
22
```

# Renderização Condicional

- Uso de if/else e variáveis
  - Mas....
    - Incluir no meio das tags não funciona



# Renderização Condicional

- Uso do operador &&
  - O operador && considera
    - Uma condição a esquerda
    - Uma expressão a direita
  - Se a condição for verdadeira
    - A expressão é executada
    - O elemento é incluído
  - Se a condição for falsa
    - A expressão não é executada
    - O elemento não é incluído

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4    const isLoggedIn = true;
5    const userName = "Ariovaldo"
6
7    return (
8      <div>
9        {isLoggedIn && (<>
10          <h1>Hi, {userName}!</h1>
11          <p>Welcome to our system</p>
12          </> )}
13      </div>
14    );
15  }
16
17  export default App;
```

# Renderização Condicional

- Uso do operador ternário
  - O operador ternário `__?__:`
    - `condicao ? valor_se_verdadeiro : valor_se_falso`
  - **Condição**
    - Equivalente àquele usado com if
  - **Valor\_se\_verdade**
    - Entregue se a condição for verdadeira
  - **Valor\_se\_falso**
    - Entregue se a condição for falsa

```
JS App.js M X
src > JS App.js > App
1  import React from 'react';
2
3  function App() {
4    const isLoggedIn = true;
5
6    return (
7      <div>
8        {
9          isLoggedIn
10         ? <h1>Welcome, User!</h1>
11         : <h1>Please log in.</h1>
12       }
13     </div>
14   );
15 }
16
17 export default App;
18
```

# Coleções

- Assunto recorrente em disciplinas de programação
  - Define uma estrutura de dados
    - Voltada para conjunto de dados
    - Muitas vezes de tipos equivalentes (mas não necessariamente)
    - Armazenar, manipular e organizar
- Em JavaScript
  - Existem tipos nativos para tratar coleções
    - Arrays
    - Objetos
    - Dicionários (mapas)



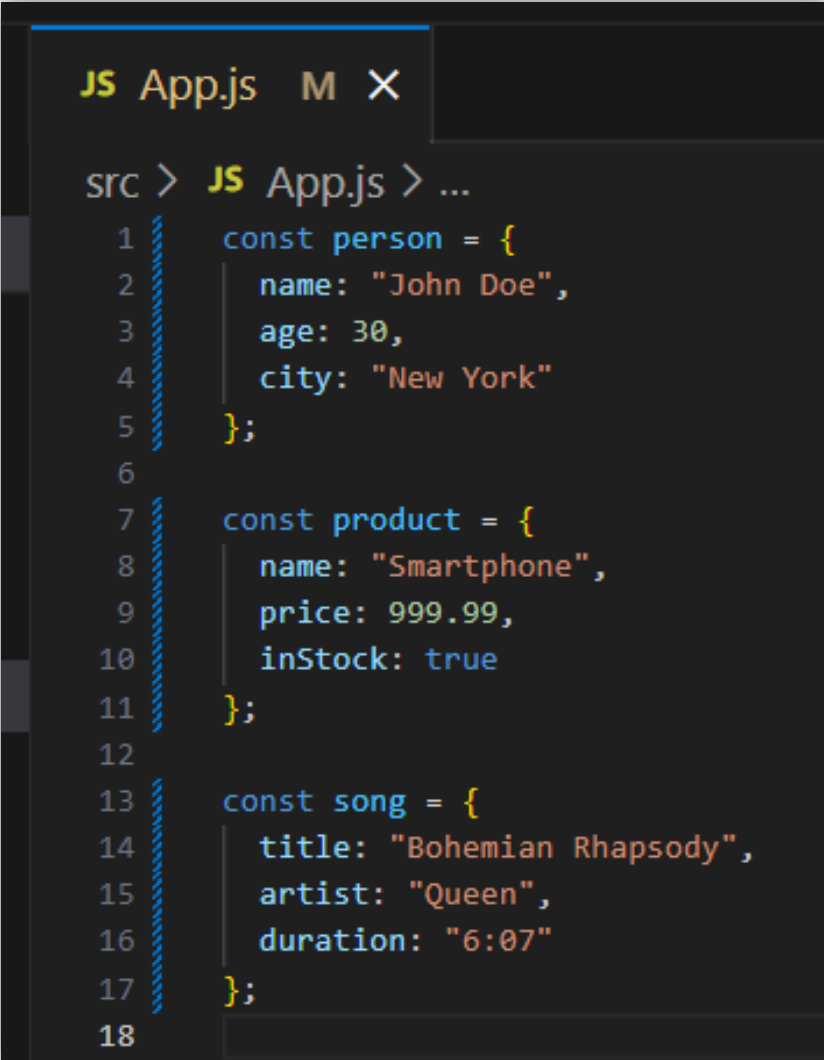
# Coleções

- Em JavaScript
  - Existem tipos nativos para tratar coleções
    - Arrays
      - Coleção ordenada de elementos
        - Pode armazenar
          - Qualquer tipo de dado, outros arrays, objetos
    - Elementos acessados por índice numérico
    - Podem ser adicionados ou removidos dinamicamente
    - Alguns métodos disponíveis
      - push e pop: Adiciona o remove elemento
      - forEach: Percorre cada elemento
      - Map: Transforma o elemento armazenado
      - Filter: retorna nova lista com o resultado da filtragem
      - Reduce: Usado para contagem, média, etc

```
JS App.js M X
src > JS App.js > ...
1 // Array de inteiros
2 const numbers = [1, 2, 3, 4, 5];
3
4 // Array de strings
5 const fruits = ["apple", "banana", "orange", "mango"];
6
7 // Array de objetos
8 const people = [
9   { name: "John", age: 30 },
10  { name: "Alice", age: 25 },
11  { name: "Bob", age: 35 }
12 ];
13
14 // Array de booleanos
15 const flags = [true, false, true, false];
16
17 // Array declarado de forma vazia
18 const emptyArray = [];
```

# Coleções

- Em JavaScript
  - Existem tipos nativos para tratar coleções
    - Objetos
      - coleção não ordenada de pares chave-valor
      - Para cada valor
        - É associado a uma chave exclusiva
        - Permite acessar e manipular o valor.
      - São usados para representar
        - Entidades com propriedades e comportamentos
        - As propriedades
          - podem ser adicionadas, modificadas ou removidas dinamicamente
          - O acesso a propriedades feito usando a notação de ponto objeto.propriedade

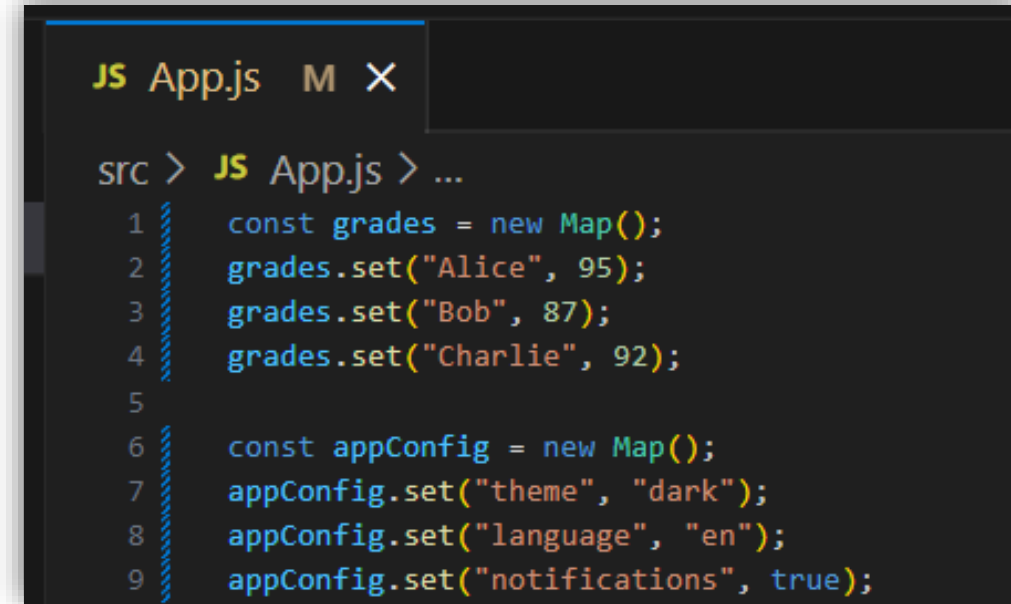
A screenshot of a code editor window titled 'JS App.js'. The editor shows three JavaScript objects defined with 'const'. The first object, 'person', has properties 'name', 'age', and 'city'. The second object, 'product', has properties 'name', 'price', and 'inStock'. The third object, 'song', has properties 'title', 'artist', and 'duration'. Line numbers 1 through 18 are visible on the left side of the code.

```
src > JS App.js > ...  
1  const person = {  
2    name: "John Doe",  
3    age: 30,  
4    city: "New York"  
5  };  
6  
7  const product = {  
8    name: "Smartphone",  
9    price: 999.99,  
10   inStock: true  
11  };  
12  
13  const song = {  
14    title: "Bohemian Rhapsody",  
15    artist: "Queen",  
16    duration: "6:07"  
17  };  
18
```



# Coleções

- Em JavaScript
  - Existem tipos nativos para tratar coleções
    - Dicionários (mapas)
      - Coleção de elementos que permite armazenar pares chave-valor
        - Tanto as chaves quanto os valores podem ser de qualquer tipo
        - Ao contrário dos objetos
          - Mapas mantêm a ordem de inserção dos elementos
          - Permitem usar qualquer tipo de valor como chave
    - Alguns métodos disponíveis
      - set, get: Permite modificar e capturar dados
      - Delete: Permite eliminar uma entrada
      - Has: Permite avaliar se um recurso está na coleção
      - forEach: Permite iterar sobre a coleção

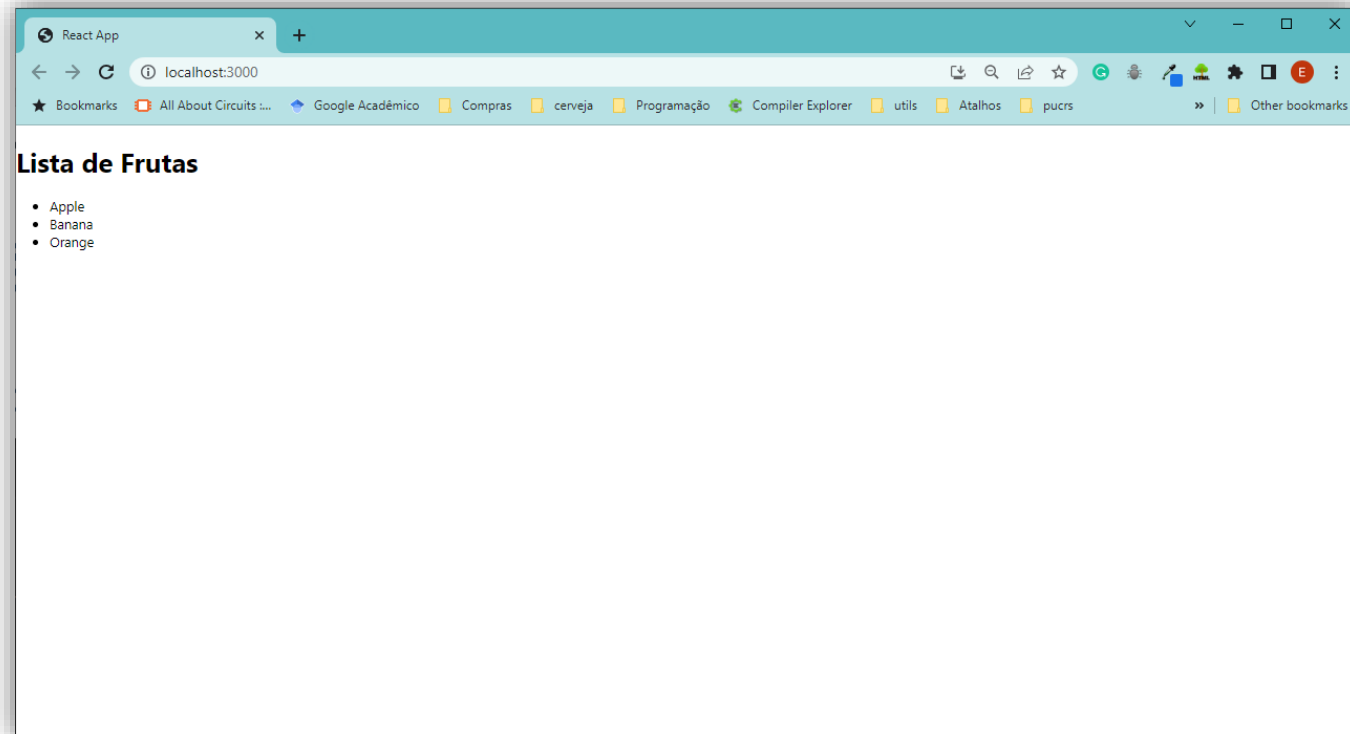


```
JS App.js M X
src > JS App.js > ...
1  const grades = new Map();
2  grades.set("Alice", 95);
3  grades.set("Bob", 87);
4  grades.set("Charlie", 92);
5
6  const appConfig = new Map();
7  appConfig.set("theme", "dark");
8  appConfig.set("language", "en");
9  appConfig.set("notifications", true);
```

# Exploração de coleções com React

- Caso 1 – Explorando Array de strings com renderização condicional

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4    const fruits = ['Apple', 'Banana', 'Orange'];
5    const showFruits = true;
6
7    return (
8      <div>
9        <h1>Lista de Frutas</h1>
10       {showFruits && (
11         <ul>
12           {fruits.map((fruit, index) => (
13             <li key={index}>{fruit}</li>
14           ))}
15         </ul>
16       )}
17     </div>
18   );
19 }
20
21 export default App;
```

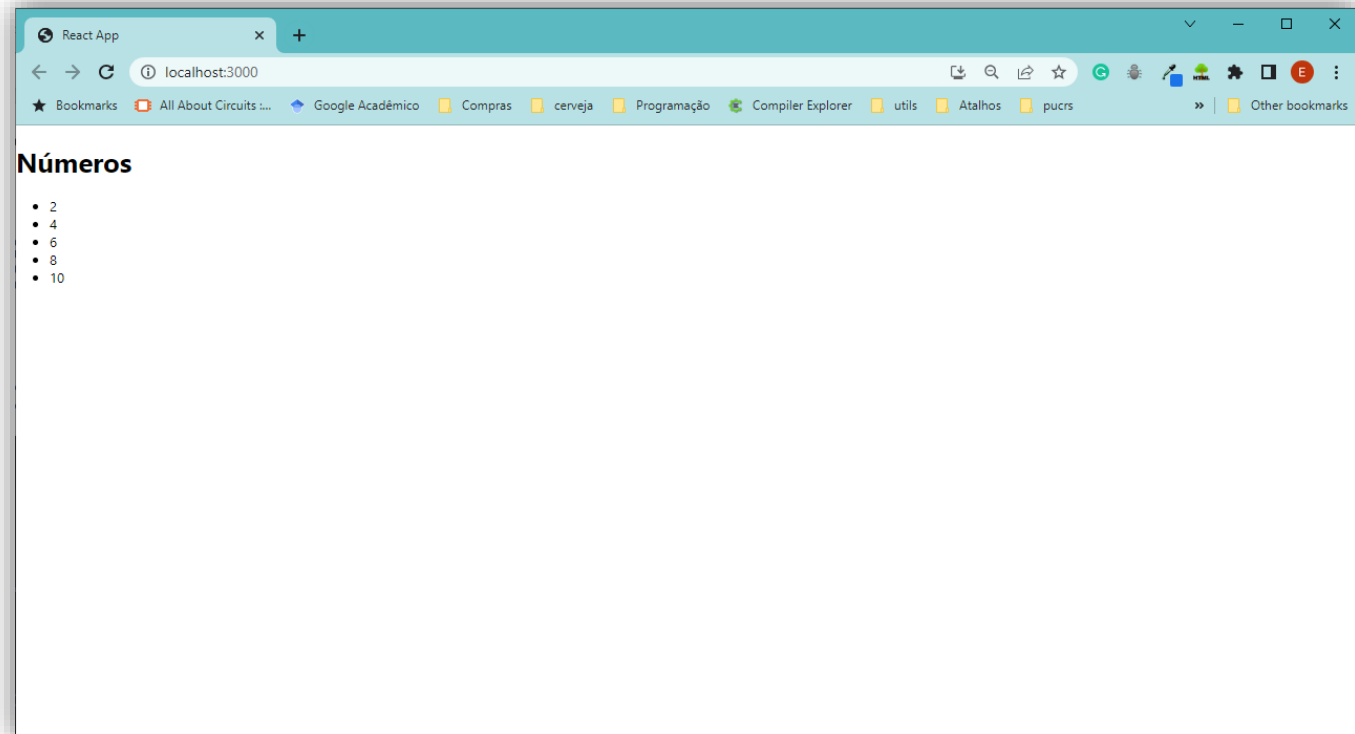


# Exploração de coleções com React

- Caso 2 – Explorando Array de inteiros com renderização condicional

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4    const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
5
6    return (
7      <div>
8        <h1>Números</h1>
9        <ul>
10         {
11           numbers.map(
12             (number, index) => {
13               if (number % 2 === 0) {
14                 return <li key={index}>{number}</li>;
15               }
16               else {
17                 return null;
18               }
19             }
20           )
21         }
22       </ul>
23     </div>
24   );
25 }
26
27 export default App;
28
```

Map



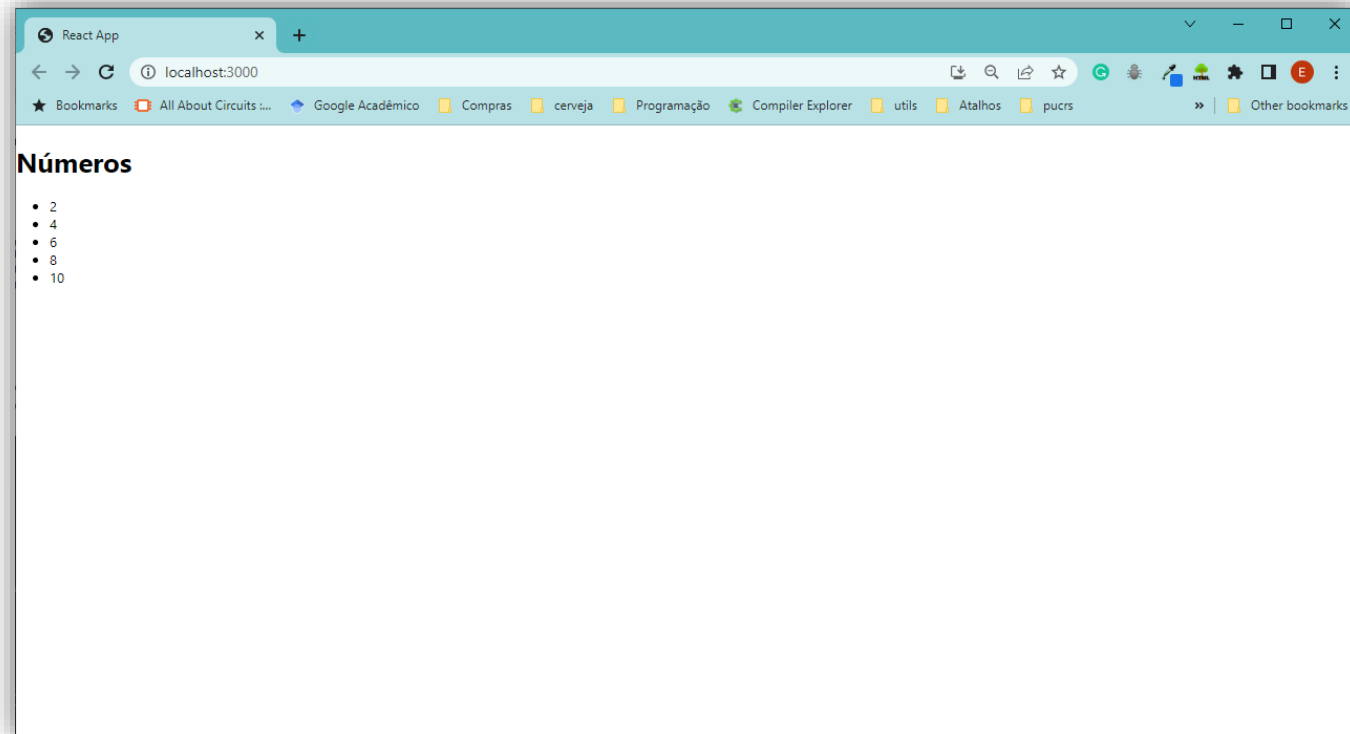
# Exploração de coleções com React

- Caso 3 – Explorando Array de inteiros com renderização condicional
  - Implementação equivalente a anterior, mas encadeando funções filter e map

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4    const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
5
6    return (
7      <div>
8        <h1>Números</h1>
9        <ul>
10         {
11           //number.filter.map
12           numbers.filter(
13             (e) => (e%2) === 0
14           ).
15           map(
16             (number, index) => {
17               return <li key={index}>{number}</li>;
18             }
19           )
20         }
21       </ul>
22     </div>
23   );
24 }
25
26 export default App;
```

Filter

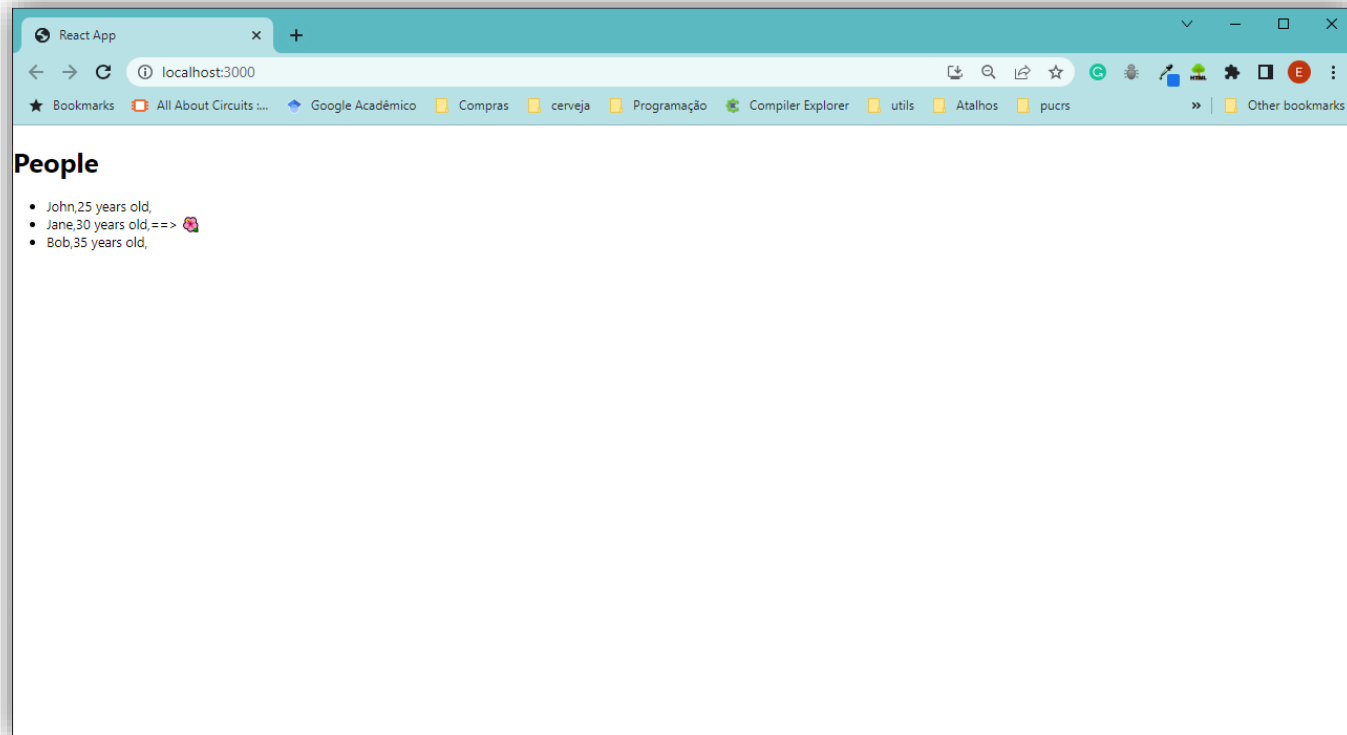
Map



# Exploração de coleções com React

- Caso 3 – Explorando Array de objetos com renderização condicional
  - Explorando o comando &&
  - Inclusão da flor só em caso de gênero feminino

```
JS App.js M X
src > JS App.js > ...
1  import React from 'react';
2
3  function App() {
4    const people = [
5      { id: 1, name: 'John', age: 25, gender: 'Male' },
6      { id: 2, name: 'Jane', age: 30, gender: 'Female' },
7      { id: 3, name: 'Bob', age: 35, gender: 'Male' }
8    ];
9
10   return (
11     <div>
12       <h1>People</h1>
13       <ul>
14         {people.map(person => (
15           <li key={person.id}>
16             {person.name},
17             {person.age} years old,
18             {(person.gender==='Female') && '==> \u{1F33A}'}
19           </li>
20         ))}
21       </ul>
22     </div>
23   );
24 }
25
26 export default App;
```



## Resumo do que vimos até agora



O uso de diferentes comandos para renderização condicional

O uso de coleções

# Análise e Desenvolvimento de Sistemas

- Desenvolvimento de sistemas *front end*



# Aula 2 (parte 5)

- Exemplos de solução explorando recursos vistos



# O que você vai aprender nessa aula

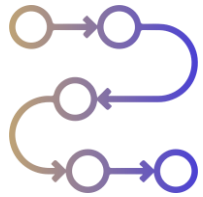


- Criação de projetos React
- Elaboração de código exemplo

## O que você vai precisar para acompanhar essa aula



- O ambiente de desenvolvimento funcionando
  - IDE
  - Node JS



## Dinâmica

- Construção de código



## Case

- Condução do caso 1
  - Criando um sistema para apresentação de um card de uma pessoa
- Passos
  - Elaboração de um objeto
  - Elaboração da solução
  - Estilização



## Case

- Condução do caso 2
  - Criando um sistema para apresentação de um card de pessoas
    - Estender cada objeto para conter peso e altura
- Passos
  - Elaboração de um array de objetos
  - Elaboração da solução
  - Indicar se a pessoa tem um risco dado pelo IMC
    - Apresentar ícone de alerta

## Resumo do que vimos até agora



- Exemplos básicos de manipulação de recursos em React JS