

Programação & Serviços Web

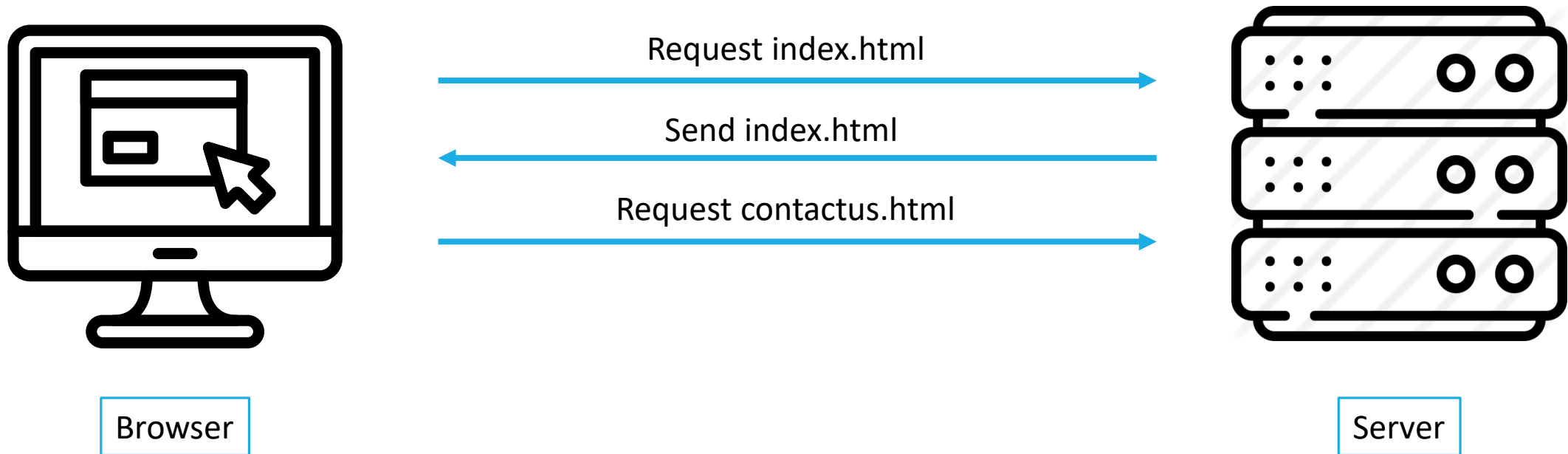
React – Aula 3

2019/2020

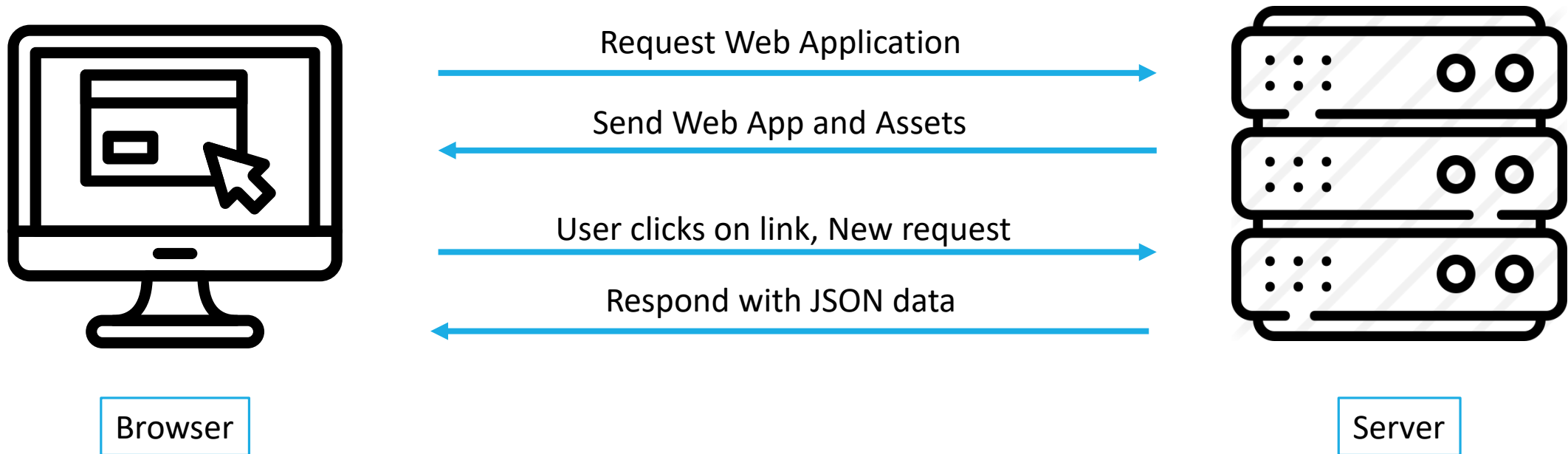
Ciclo de vida do react
React router

Single Page Application

Traditional Websites



Single Page Application



Single Page Application

- Aplicação Web ou Website que cabe todo dentro de uma só página
 - Faz com que não seja preciso fazer reload da página inteira, mas apenas de alguns componentes
 - Trabalhar a UX como se fosse uma aplicação desktop/nativa
 - A maioria dos recursos são acedidos através um load de página simples
 - Redesenha parte das páginas quando é preciso sem ser necessário uma “volta até ao servidor”

Ciclo de Vida do React

Ciclo de Vida

- Os componentes passam pelos seguintes estágios do ciclo de vida
 - Mounting
 - Updating
 - Umounting
- Existem vários métodos em cada estágio do ciclo de vida

Ciclo de Vida - Mounting

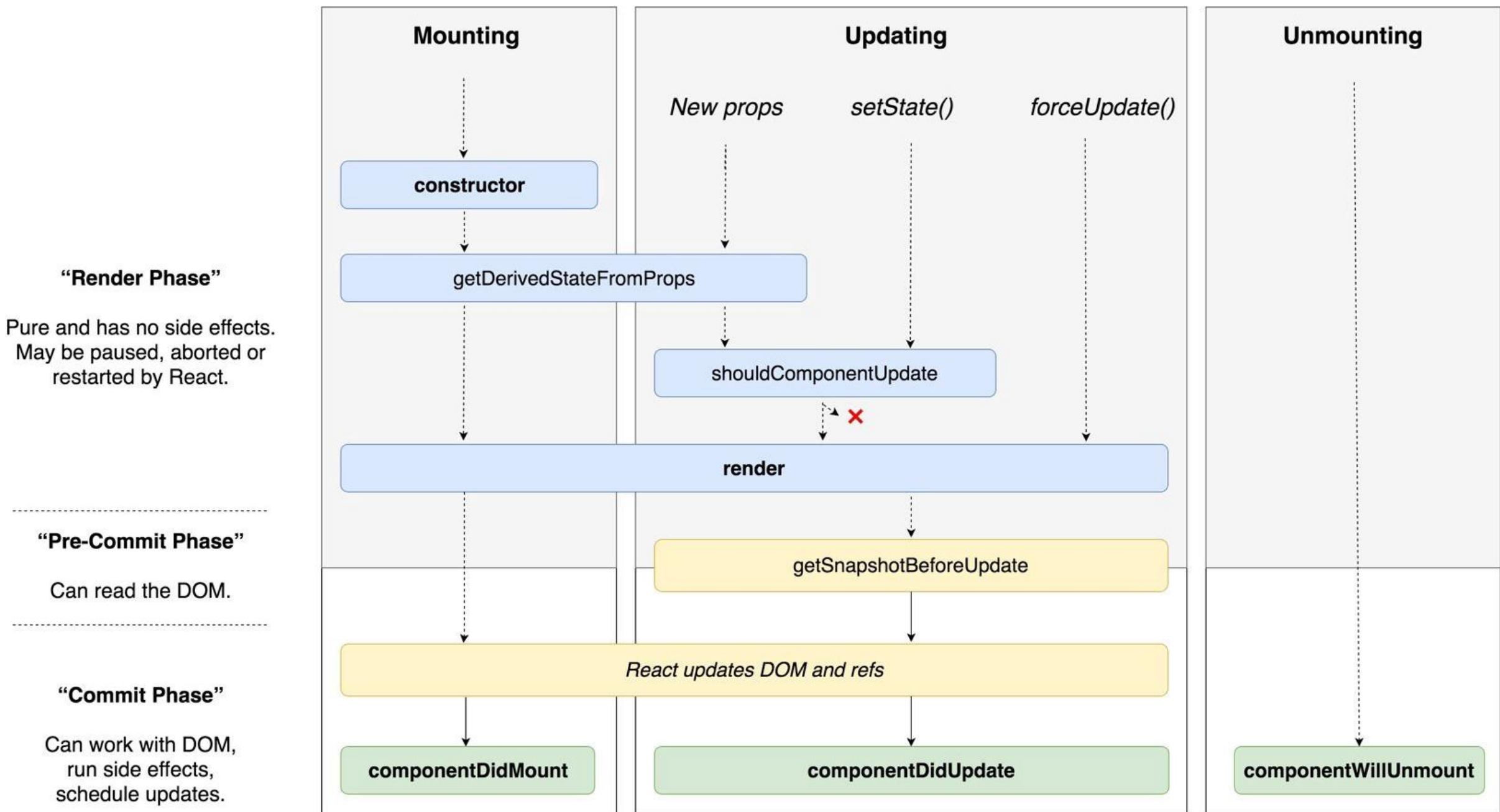
- Chamado quando uma instancia de um componente é criada e é inserida no DOM
 - constructor() – Opcional (mas convém colocar sempre)
 - getDerivedStateFromProps() - Opcional
 - render() - Obrigatório
 - componentDidMount() - Opcional
- Se procurarem no google, existe um outro método chamado `componentWillMount()`
 - Este método foi deprecated (não deve ser usado que vai ser removido numa versão futura)

Ciclo de Vida - Updated

- Chamado quando um componente vai ser renderizado novamente / atualizado
 - Pode ser causado por uma atualização ao props ou ao state
- Funções:
 - `getDerivedStateFromProps()` – igual ao do mount
 - `shouldComponentUpdate()` – devolve um `true` ou `false`, de acordo com se deve ser actualizado ou não
 - `render()` – igual ao do mount
 - `getSnapshotBeforeUpdate()`
 - `componentDidUpdate()`
- Duas funções deprecated:
 - `componentWillReceiveProps()`
 - `componentWillUpdate()`

Ciclo de Vida - Unmounting

- Só tem uma função:
 - `ComponentWillUnmount`
- Pode ser útil para quando se quer enviar uma mensagem a um servidor a dizer que aquela pessoa desligou a aplicação (por exemplo)



Exercício 1

Completar o esquema seguinte.

App.js

```
State:  
- clock  
  
function update  
- updateClock every 1 sec  
  
<MostraRelogio clock={this.  
state.clock} />
```

MostarRelógio.js

Funções necessárias?

Esta aplicação foi criada para mostrar as horas.

Complete a classe `MostrarRelogio`, indicando o tipo de componente e quais são as funções necessárias

Como os clientes desta aplicação são pessoas certas, a classe só deve actualizar quando os segundos for um número par

Aplicação - Relógio

Resolução

- O componente vai receber props novas a cada segundo que passa, devido ao timer.
- Há uma particularidade importante na classe MostrarRelogio, que é só atualizar a cada 2 segundos. Desta forma, é preciso usar a função **ShouldComponentUpdate**
- Como se têm de usar uma função do ciclo de vida, tem que se criar um **componente de classe**
- Pode-se também utilizar a função **Constructor** e converter o **props** recebido em **state** do actual

React Router

React Router

- Conjunto de componentes que ajuda a indicar para onde se pode navegar
 - Permite a navegação entre os vários componentes/páginas
- Utiliza URLs baseados no browser (para permitir adicionar aos favoritos)
 - Também se pode adicionar parâmetros opcionais

Instalação

- O React Router é um módulo à parte, pelo que tem que ser instalado

```
install react-router-dom
```

- Neste módulo, existem várias classes que podem (e vão) ser usadas:

- **BrowserRouter**

- Cria um objeto de história de navegação (para poder navegar para a frente e para trás)
 - Deve englobar a aplicação no App.js

BrowserRouter

```
<div className="App">
  <div>
    <Header my_name={this.state.my_name}
project_name={this.state.project_name} />
    <Content />
    <Footer my_name={this.state.my_name}
project_name={this.state.project_name} />
  </div>
</div>
```

```
<div className="App">
  <BrowserRouter >
    <div>
      <Header my_name={this.state.my_name}
project_name={this.state.project_name} />
      <Content />
      <Footer my_name={this.state.my_name}
project_name={this.state.project_name} />
    </div>
  </BrowserRouter>
</div>
```

Podia ser adicionado logo na raiz do elemento, ou a um nível mais fundo.
É importante englobar toda a aplicação (para que as outras classes possam ser usadas

Link

- A classe **Link** fornece uma navegação declarativa e acessível em qualquer lugar da aplicação (desde que dentro do Router)
 - Vai aparecer na aplicação como se fosse um <a>

```
<Link to="/about">About</Link>
```

- A opção “**to**” pode conter uma *string* (caminho do *url*) ou um objeto, em que podem ser definidos vários parâmetros
- Pode-se adicionar outros parâmetros, tais como o **id**, **title**, **className**, pois serão posteriormente passados para o <a>

Navlink

- **<NavLink>** é igual ao **Link**, mas adiciona a classe “**active**” quando o **url actual** for **igual** ao **url a que está associado**
 - Isto serve para definir alguma regra específica no CSS
- Uma propriedade do Navlink a ter em consideração é o “**exact**”
 - O exact valida se o link é exatamente igual ao que está definido, e só se for, é que adiciona o **active**
 - Caso contrário, ele vai verificar se existe. Ex:

```
<NavLink to="/">Home</NavLink>  
<NavLink to="/dashboard">Dashboard</NavLink>
```

Sem o exact, o “**to**” do **primeiro** link está presente no “**to**” do **segundo**.

Logo ele adiciona o “**active**” ao **1º Navlink** quer esteja no primeiro link ou no segundo

Switch

- O **Switch**, tal como o nome indica, funciona de forma semelhante a um **Switch** do Javascript.
 - Existem várias opções disponíveis e uma opção por defeito.
- Neste caso, o **Switch** vai ter associado a si **um ou mais Route**
 - Pode-se dizer que ele agrupa vários Route. Quando está à procura de um que faça **match** ao caminho definido, vai por ordem do 1º ao último Route/Redirect
- Ao longo de toda a aplicação, podem existir vários **Switch** (com um ou mais **Route** lá dentro)

Switch e Route

App.js

```
<Switch>
  <Route exact path="/" component={Dashboard} />
  <Route path="/mainPage" component={MainPage} />
</Switch>
```

Neste caso não é renderizado nada

MainPage.js

```
<Switch>
  <Route exact path="/mainPage" />
  <Route path="/mainPage/tab2" render={() => <div>Latest Novel</div>} />
  <Route path="/mainPage/tab1" render={() => <div>Create Novel</div>} />
</Switch>
```

Route

- Esta classe é uma das mais importantes no React Router. É ela que é responsável por renderizar o UI quando o **path** corresponde com o URL.

```
<Route path="/user/:username" component={User} />
```

↓
Caminho a comparar

↘
Componente a Renderizar quando o caminho corresponde

```
<Route path="/dashboard/add" render={props =>  
  (<SuperForm {...props} submit={this.submitForm} />)  
} />
```

No caso de se pretender enviar propriedades para o componente (ou associar mais que 1 componente), é preciso utilizar a propriedade `render`


Redirect

- A classe **Redirect** vai fazer um redireccionamento para o url definido.
- O **Redirect** pode servir para, por exemplo, ser útil para fazer um redirect para a **página 404**, no caso de o utilizador tentar abrir um url não válido.
- Outro exemplo é um utilizador tentar aceder a uma secção administrativa sem o login feito
 - O redirect iria enviar o utilizador para a página de login

Parâmetros

- Os caminhos são especificados como um URL. Podem também trazer parâmetros
 - Por exemplo: `‘/comida/42’` onde 42 é o parâmetro
- Os parâmetros são especificados no caminho como um token
 - Por exemplo, com base no caso anterior, seria `‘/comida/:id’`, onde o ID é o token
- Os parâmetros podem ser especificados quando estamos a especificar o link
 - `<Link to=‘/menu/42’>`

Objeto Match

- O objecto **Match** disponibiliza informação sobre um **<Route>** encontrado no url
 - **match.params**: um objeto que contem o par key/value a partir do url
 - Por exemplo
 - Se o caminho estiver especificado como **/menu/:id**
 - O caminho for **/menu/42** então o
 - **match.params.id** vai ser igual a **“42”**
- 

Parâmetros

- Imagine o seguinte link no **Route**:

```
<Route path="/dashboard/edit/:private_id" component={Form}>
```

- No componente Form, deveria procurar pelo campo private_id

```
props.match.params.private_id
```

Redirects via função

- Pode acontecer que durante a programação da aplicação, pretenda fazer um redirect no final de chamar uma função
 - Por exemplo, após clicar no botão de Save de um formulário, fazer redirect para o Dashboard
- No entanto, como se viu, o <Redirect> só pode ser chamado no Render..
Como fazer?
- Existe o componente WithRouter, que permite aceder ao campo History e, dessa forma, fazer o redirect no final da função

WithRouter

- Para poder utilizar o withRouter, é preciso atualizar o export do componente em que é para ser usado
 - No seguinte caso, o componente é o Content

```
export default Content;
```

```
export default withRouter(Content);
```

- Ao fazer isto, é adicionado ao props do Componente o objeto history, com o qual:

```
this.props.history.push('/dashboard');
```

Composição / Herança

Composição - Containment

- Em alguns componentes, não se sabe à partida quais são os elementos que eles vão ter. Isto acontece no Sidebar / Popups, que representam caixas genéricas.

```
function WelcomeDialog() {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title"> Welcome </h1>  
      <p className="Dialog-message">  
        Thank you for visiting our spacecraft!  
      </p>  
    </FancyBorder>  
  );  
}
```

```
function FancyBorder(props) {  
  return (  
    <div className={'FancyBorder FancyBorder-' + props.color}>  
      {props.children}  
    </div>  
  );  
}
```

Composição - Containment

```
function WelcomeDialog() {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title"> Welcome </h1>  
      <p className="Dialog-message">  
        Thank you for visiting our spacecraft!  
      </p>  
    </FancyBorder>  
  );  
}
```

```
function FancyBorder(props) {  
  return (  
    <div className={'FancyBorder FancyBorder-' + props.color}>  
      {props.children}  
    </div>  
  );  
}
```

Tudo o que seja colocado dentro do <FancyBorder> do painel da esquerda é enviado para o Componente FancyBorder como props children.

Composição

```
<SplitPane
  left={
    <Contacts />
  }
  right={
    <Chat />
  }
/>
```

```
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">
        {props.left}
      </div>
      <div className="SplitPane-right">
        {props.right}
      </div>
    </div>
  );
}
```

Embora este caso seja mais raro, também é possível dividir em 2 elementos à parte. No entanto, estes elementos (left, right), já não são adicionados ao props.children

Composição - Specialization

- Por vezes, alguns components são casos especiais de outros components. Por exemplo: o **Dialog** é o caso especial do **WelcomeDialog**.

```
function Dialog(props) {  
  return (  
    <FancyBorder color="blue">  
      <h1 className="Dialog-title"> {props.title} </h1>  
      <p className="Dialog-message">  
        {props.message}  
      </p>  
    </FancyBorder>  
  );  
}
```

```
function WelcomeDialog() {  
  return (  
    <Dialog  
      title="Welcome"  
      message="Thank you for visiting our spacecraft!"  
    />  
  );  
}
```

Composição - Specialization

- Outro exemplo, com classes mais especializadas

```
<Dialog title="Mars Exploration Program"
  message="How should we refer to you?">
  <input value={this.state.login}
    onChange={this.handleChange} />
  <button onClick={this.handleSignUp}>
    Sign Me Up!
  </button>
</Dialog>
```

```
function Dialog(props) {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        {props.title}
      </h1>
      <p className="Dialog-message">
        {props.message}
      </p>
      {props.children}
    </FancyBorder>
  );
}
```

Herança

- Props e a composição oferece toda a flexibilidade necessária para customizar um componente da forma que pretende.
- O Facebook utiliza milhares de componentes diferentes e nunca usou herança.

Exercício 2

No componente Content, adicione um botão que, ao clicar nele, faça o redirect para o “/adminhackermode”

Adicione um Route para que, quando for aberto um path não encontrado, faça redirect para um componente que mostre o erro 404

Ficha 3

Leitura Complementar

- <https://reactjs.org/docs/state-and-lifecycle.html> (A partir da secção Adding Lifecycle Methods to a Class)
- <https://github.com/ReactTraining/react-router/tree/master/packages/react-router/docs/api> (Ler a descrição de cada uma das classes usadas)