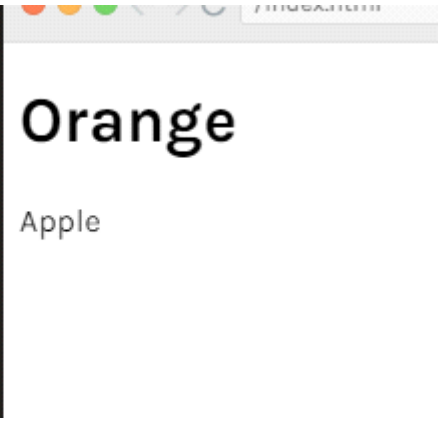
Props

sábado, 30 de setembro de 2023 13:21

Antes de começar: mapping components

React pode ler um array de componentes, basicamente copiando o conteúdo deste array e inserindo-os no retorno, como mostrado abaixo.

```
3
4
5  export default function App() {
6      const fruits = [<h1>Orange</h1>, <p>Apple</p>]
7      return (
8          <div>
9              {fruits}
10         </div>
11     )
12 }
```



Mas qual a utilidade disto? O que queremos de fato é pegar dados de um ambiente externo e criar novos componentes para cada nova informação que inserirmos no sistema.

Percorrendo um array de objetos

Vamos supor que temos um vetor que contém objetos com informações sobre piadas, identificadas pelas chaves "setup" e "punchline".

```
1  export default [
2      {
3          setup: "I got my daughter a fridge for her birthday.",
4          punchline: "I can't wait to see her face light up when she opens it."
5      },
6      {
7          setup: "How did the hacker escape the police?",
8          punchline: "He just ransomware!"
9      },
10     {
11         setup: "Why don't pirates travel on mountain roads?",
12         punchline: "Scurvy."
13     },
14     {
15         setup: "Why do bees stay in the hive in the winter?",
16         punchline: "Swarm."
17     },
18     {
19         setup: "What's the best thing about Switzerland?",
20         punchline: "I don't know, but the flag is a big plus!"
21     }
22 ]
```

```
    }
]
```

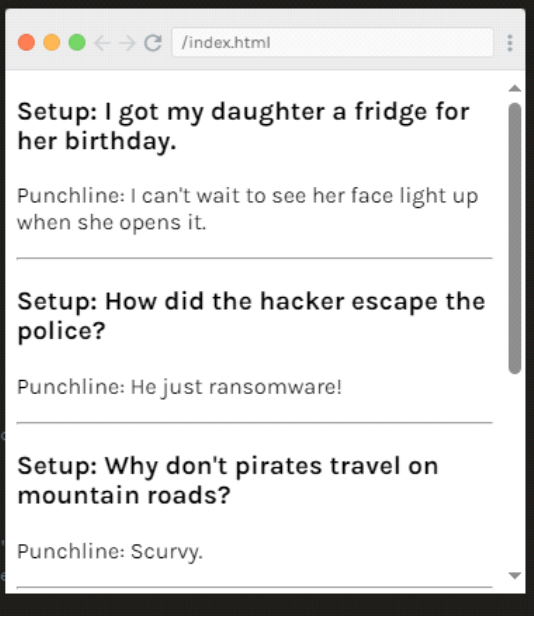
Como o vetor é exportado, podemos importá-lo no arquivo que vai gerar os componentes baseados nestes objetos. Já sabemos que é possível criar um vetor de componentes. Então, vamos gerar um novo array a partir do método map, que, para cada elemento do vetor jokesData (cada objeto presente nele), será criado um novo componente <Joke /> com os argumentos setup e punchline personalizados com os valores setup e punchline de cada um dos objetos presentes no vetor.

```
export default function App() {
    const jokeElements = jokesData.map(
        joke => <Joke setup={joke.setup} punchline={joke.punchline}/>
    )
```

Assim, ao retornar o novo array de componentes, o jokeElements, obtemos o seguinte resultado:

```
import React from "react"
import Joke from "./Joke"
import jokesData from "./jokesData"

export default function App() {
    const jokeElements = jokesData.map(
        joke => <Joke setup={joke.setup} punchline={joke.punchline}/>
    )
    return (
        <div>
            {jokeElements}
        </div>
    )
}
            // <Joke
            //     punchline="It's hard to explain puns to kleptomania
            things literally."
            // />
            // <Joke
            //     setup="I got my daughter a fridge for her birthday.
            //     punchline="I can't wait to see her face light up wh
            //     isPun={false}
```



Criando elementos com condicionais

Vamos supor que eu queira adicionar um badge de "sold out" em um card. Caso o número de vagas for igual a 0, este badge aparecerá sobre o card. Para isto, vou inicializar uma variável que vai conter o texto da mensagem que vai ser transmitida. Quando apenas declaro a variável, sem atribuir nenhum valor a ela, o seu conteúdo é "undefined". Assim posso verificar do seguinte modo.

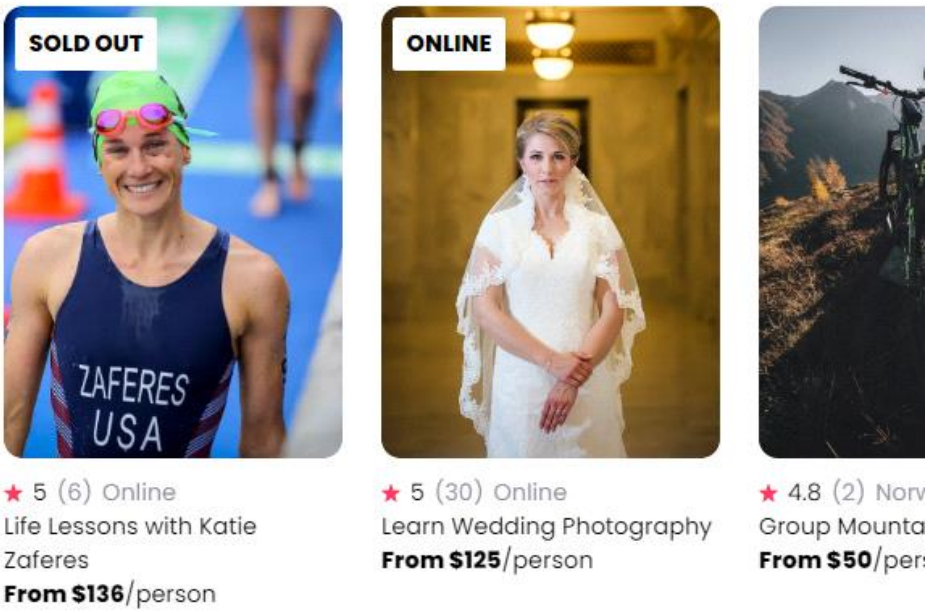
```
export default function Card(props) {
    let badgeText;

    if (props.openSpots === 0) badgeText = "SOLD OUT";
    else if(props.location === "Online") badgeText = "ONLINE";

    return <div className='card'>
        {badgeText && <div className="card--badge">{badgeText}</
        div>}
```

Assim, o resultado que obtemos é o seguinte:

Join unique interactive led by one-of-a-kind hosts - all without leaving home.



Mudando a forma como passamos valores em parâmetros

É grande, não é?

```
const cards = data.map(card => {
    return (
        <Card
            // Key ajuda a diferenciar o elemento com um ID
            key={card.id}
            img={card.coverImg}
            rating={card.stats.rating}
            reviewCount={card.stats.reviewCount} location=
            {card.location}
            title={card.title}
            price={card.price}
            // Para criação condicional de elementos:
            openSpots={card.openSpots}
            />
    )
})
```

Vamos reduzir isto:

```
const cards = data.map(card => {
    return (
        <Card
            // Key ajuda a diferenciar o elemento com um ID
            key={card.id}
            // Passando todo o objeto como argumento
            item={card}
        />
    )
})
```

Agora que todo o objeto foi passado para "item", preciso mudar o código do componente para que os valores acessados sejam consultados por meio deste "item".

```
export default function Card(props) {
    let badgeText;

    if (props.item.openSpots === 0) badgeText = "SOLD OUT";
    else if(props.item.location === "Online") badgeText =
    "ONLINE";

    return <div className='card'>
        {badgeText && <div className="card--badge">{badgeText}</
        div>}

        <img src={props.item.coverImg} className="card--image" />

        <div className="card--stats">
            <img src={star} alt="star image"
            className='card--star'/>
            <span>{props.item.stats.rating}</span>
            <span className="gray">({props.item.stats.
            reviewCount})</span>
            <span className="gray">{props.item.location}</span>
        </div>

        <p className="card--title">{props.item.title}</p>
        <p className="card--price"><strong>From ${props.item.
        price}</strong>/person</p>
    </div>
}
```

Só que este não é um único modo de simplificarmos a passagem de argumentos. Podemos passar todo o objeto, se aproveitando dos nomes já utilizados para cada propriedade dele:

```
return (
    <Card
        // Key ajuda a diferenciar o elemento com um ID
        key={card.id}
        // Passando todo o objeto como argumento
        // item={card}
        {...card}

        // Equivale a
        // id={card.id}
        // description={card.description}
        // ...
    />
```

Sim, são os object literals...

```
export default function Card(props) {
    let badgeText;

    if (props.openSpots === 0) badgeText = "SOLD OUT";
    else if(props.location === "Online") badgeText = "ONLINE";

    return <div className='card'>
        {badgeText && <div className="card--badge">{badgeText}</
        div>}

        <img src={props.coverImg} className="card--image" />

        <div className="card--stats">
            <img src={star} alt="star image"
            className='card--star'/>
            <span>{props.stats.rating}</span>
            <span className="gray">({props.stats.reviewCount})</
            span>
            <span className="gray">{props.location}</span>
        </div>

        <p className="card--title">{props.title}</p>
        <p className="card--price"><strong>From ${props.price}</
        strong>/person</p>
    </div>
}
```

Assim, voltamos para o ponto onde estávamos, com a diferença de que devemos utilizar o nome original das keys dos objetos.