

1. Birthday Reminder
2. 旅游推荐卡片

3. 卡片轮播 Reviews

4. Accordion (展开/收起信息)

5. 菜单 Menu (数据条件过滤)

6. Tab 切换信息(样式切换)

7. 轮播滑块 Slider

8. 点击计数栏

10. Grocery (Todolist)

 useState

11. 导航栏 navbar

12. 侧边栏 sidebar modal

13. Stripe submenu

14. Cart 购物车

15. react-router (cocktail)

 App.js

 context.js

 components

 — Cocktail.js

 — CocktailList.js

 — NavBar.js

 — SearchForm.js

 pages

 — Home.js

 — SingleCocktail.js

 — Error.js

 — About.js

16. Markdown Preview (预览)

18. Pagination 分页

20. Dark Mode 深色模式

23. 动态添加 input box

Tips

 StrictMode

<> </>

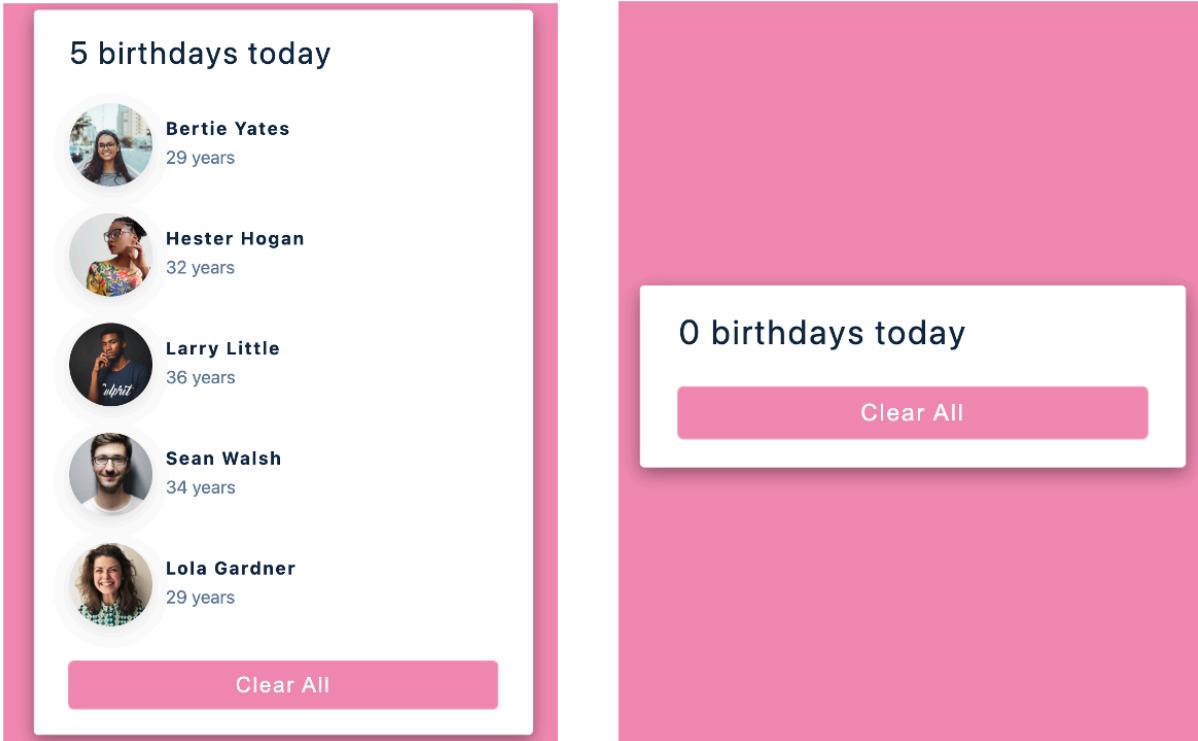
<https://course-api.com/react-tours-project>

<https://github.com/john-smilga/react-projects>

<https://react-projects.netlify.app/>

1. Birthday Reminder

页面状态有 2 个 : {1. 朋友的今日生日提醒 2. Clear All 后的提示 }



tips:

1. `<article>` 这个 HTML 标签会提供一种类似文章排列的感觉 .
2. `<section>` 表示一段专题性的内容，一般会带有标题。section 应用的典型场景有文章的章节、标签对话框中的标签页、或者论文中有编号的部分 . 比如对于评论区 , 就是一个 `<section>` , 这个 section 里会有很多 `<article>` 作为评论每条存在

```

<section>
  <h1>Comments</h1>
  <article>
    <p>Posted by: George Washington</p>
  </article>
  <article>
    <p>Posted by: George Hammond</p>
  </article>
  ...
</section>

```

关键代码

```

// List.jsx <- 组件首字母要大写
// <> 是 Fragments , 可以让你聚合一个子元素列表，并且不在 DOM 中增加额外节点。
const List = ({ people }) => { // 传入 {people} 这个 Object
  return (
    <>
      {people.map((person) => {
        const { id, name, age, image } = person; // 解包
        return (

```

```
<article key={id} className='person'>
  <img src={image} alt={name} />
  <div>
    <h4>{name}</h4>
    <p>{age} years</p>
  </div>
</article>
);
}
);
</>
);
};

export default List;
```

```
// App.js
import React, { useState } from 'react'
import data from './data'      // 伪造的 JSON 数据
import List from './List'      // 导入上面讲过的 List
function App() {
  const [people, setPeople] = useState(data)
  return (
    <main>
      <section className='container'>
        <h3>{people.length} birthdays today</h3>
        <List people={people} />
        <button onClick={() => setPeople([])}>clear all</button>
      </section>
    </main>
  )
}

export default App
```

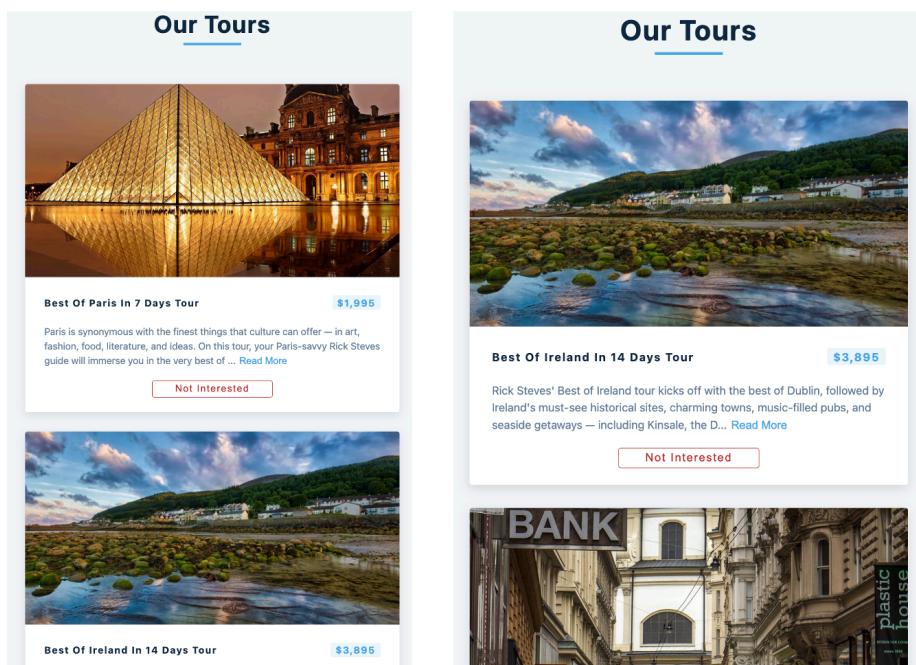
```
// data.js
export default [ {
  id: 1,
  name: 'Bertie Yates',
  age: 29,
  image:
    'https://res.cloudinary.com/diqqf3eq2/image/upload/v1595959131/person-2_ipcjws.jpg',
},
{
  id: 2, ...
}
...
];

```

2. 旅游推荐卡片

Loading...

数据 fetching 中...



都删没了之后, 会留一个 Refresh Button 重置

tips :

- ... 是 es6 中出现的扩展运算符, 作用是遍历当前使用的对象能够访问到的所有属性, 并将属性放入当前对象中
- 在 React 中, 每一个组件都默认有 key, key 不是给用户使用的, 而是给 React 自己使用的
- <component> \${price} </component>, 这里 \$ 不是什么魔术写法, 只是给 price 加一个 \$ 符号而已

模板字符串：

```
`${info.substring(0, 200)}...`  
> 这里是利用了模板文字 (Template literals) , ` ${.subx..(0, 200)} ` 是 JSX 表达式 ,  
`...` 就是省略号字符串:
```

如下例：

```
'string text line 1  
string text line 2'  
  
'string text ${expression} string text.....'
```

项目结构：

```
|--App.js  
|--Loading.js > 数据 fetching 的 Loading... 页  
|--Tour.js > 单个套餐卡片的实现  
└--Tours.js > 多个套餐卡片组合起来
```

关键代码

```
// Tour.js  
import React, { useState } from 'react';  
  
// 这里 Tour 组件，接受一个对象参数。这个对象参数需要能解包出来 {id, image, info, name, price, removeTour} 这些参数  
// 对象的参数多了没关系，但是不能少 !!  
// 比如你的 obj 有 { id0,id1,id, image, info, name, price, removeTour,..., } 这么多键值对，  
// 没关系 Tour 组件不管你的原始对象的 id0,id1 ，只要 id 即可满足  
const Tour = ({ id, image, info, name, price, removeTour }) => { // 对 object 对象解包  
  const [readMore, setReadMore] = useState(false); // 状态，是否被展开阅读  
  return (  
    <article className="single-tour">  
      <img src={image} alt={name} />  
      <footer>  
        <div className="tour-info">  
          <h4>{name}</h4> {  
            <!-- 套餐名称 -->  
            <h4 className="tour-price">${price}</h4> {  
              <!-- 套餐价格 -->  
            }  
          </div>  
          <p>  
            <!-- readMore 为 true : 展示完整 info -->  
            <!-- readMore 为 false: 展示完整 info 前 200 字-->  
          </p>  
        {readMore ? info : `${info.substring(0, 200)}...`}  
      </div>  
    </article>  
  );  
};
```

```

        <button onClick={() => setReadMore(!readMore)}>
            {readMore ? 'show less' : 'read more'}
        </button>
        <!-- 如果当前 readMore 为 true : 说明文本展开, button 的文字是 show less , 点击收起
文字 -->
        <!-- 如果当前 readMore 为 fasle: 说明文本收起, button 的文字是 read more , 点击展开
文字 -->

    </p>
    // 移除 Tour 卡片不是 Tour 的 property ,故不在这里实现.
    <button className="delete-btn" onClick={() => removeTour(id)}>
        not interested
    </button>
</footer>
</article>
);
};

export default Tour;

```

```

// Tours.js
import Tour from './Tour';

const Tours = ({ tours, removeTour }) => { // tours是对象, removeTour是一个方法
    return (
        <section>
            <div className="title">
                <h2>our tours</h2>
            </div>
            <div>
                {tours.map((tour) => {
                    // const Tour = ({ id, image, info, name, price, removeTour })
                    // ... 运算符 : 在函数的传参数过程。作用将剩余的参数放入一个数组中
                    // {...tour} 中 : {id, image, info, name, price}
                    // removeTour 只是传进来的一个方法。
                    // 注意 , tours 是一组对象的 json 数组, tour 是单个对象
                    return <Tour key={tour.id} {...tour} removeTour={removeTour} />;
                })}
            </div>
        </section>
    );
};

export default Tours;

```

```
// Loading.js
import React from 'react';
const Loading = () => {
  return (
    <div className="loading">
      <h1>loading...</h1>
    </div>
  );
};

export default Loading;
```

```
// App.js
import React, { useState, useEffect } from 'react'
import Loading from './Loading'
import Tours from './Tours'

function App() {
  const url = 'https://course-api.com/react-tours-project'
  const [loading, setLoading] = useState(true)
  const [tours, setTours] = useState([])

  const removeTour = (id) => {
    const newTours = tours.filter((tour) => tour.id !== id)
    setTours(newTours)
  }

  const fetchTours = async () => {
    setLoading(true)
    try {
      const response = await fetch(url) // fetch data...
      const tours = await response.json() // 转成 Json
      setLoading(false) // 一旦获取到数据，就设置 loading 状态为 false
      setTours(tours)
    } catch (error) {
      setLoading(false)
      console.log(error)
    }
  }
}

useEffect(() => { fetchTours() }, []) // 不依赖，直接执行 fetchTours()

if (loading) {
  return (
    <main>
      <Loading />
    </main>
  );
}

const TourList = () => {
  const {tours} = props
  return (
    <ul>
      {tours.map((tour) => (
        <li key={tour.id}>
          {tour.name}
        </li>
      ))}
    </ul>
  );
}
```

```
        )
    }
    if (tours.length === 0) { // tours 为空数组
      return (
        <main>
          <div className='title'>
            <h2>no tours left</h2>
            <button className='btn' onClick={() => fetchTours()}> <!--重新 fetch 获取数据-->
              refresh
            </button>
          </div>
        </main>
      )
    }
    return (
      <main>
        <Tours tours={tours} removeTour={removeTour} />
      </main>
    )
  }

export default App
```

3. 卡片轮播 Reviews

Our Reviews



Bill Anderson

THE BOSS

Edison bulb put a bird on it humblebrag, marfa pok pok heirloom fashion axe cray stumptown venmo actually seitan. VHS farm-to-table schlitz, edison bulb pop-up 3 wolf moon tote bag street art shabby chic.



Surprise Me

tips :

- 卡片式分页，`Surprise Me` 按钮可以提供随机的人来查看 .
- [React icon](#) 提供了一些有趣的 icon

```
// App.js
import React from 'react';
import Review from './Review';
function App() {
  return (
    <main> // Why a <main> ?
      <section className='container'>
        <div className='title'>
          <h2>our reviews</h2>
          <div className='underline'></div>
        </div>
        <Review />
      </section>
    </main>
  );
}
export default App;
```

```
import React, { useState } from 'react';
import people from './data'; // JSON 文件
import { FaChevronLeft, FaChevronRight, FaQuoteRight } from 'react-icons/fa';

const Review = () => {
  const [index, setIndex] = useState(0);
  const { name, job, image, text } = people[index];

  const checkNumber = (number) => {
    // if (number > people.length - 1 || number < 0 ) {
    //   return people.length - 1;
    // }
    return number;
  };
  const nextPerson = () => {
    setIndex((index) => {
      let newIndex = index + 1;
      return checkNumber(newIndex);
    });
  };
  const prevPerson = () => {
    setIndex((index) => {
      let newIndex = index - 1;
      return checkNumber(newIndex);
    });
  };
  const randomPerson = () => {
    let randomNumber = Math.floor(Math.random() * people.length) ;
    if (randomNumber === index) {
      randomNumber = index + 1;
    }
    setIndex(checkNumber(randomNumber));
  };

  return (
    <article className='review'>
      <div className='img-container'>
        <img src={image} alt={name} className='person-img' />
        <span className='quote-icon'>
          <FaQuoteRight /> /* React-icon 提供的 右引号”图标 */
        </span>
      </div>
      <h4 className='author'>{name}</h4>
      <p className='job'>{job}</p>
      <p className='info'>{text}</p>
      <div className='button-container'>
```

```

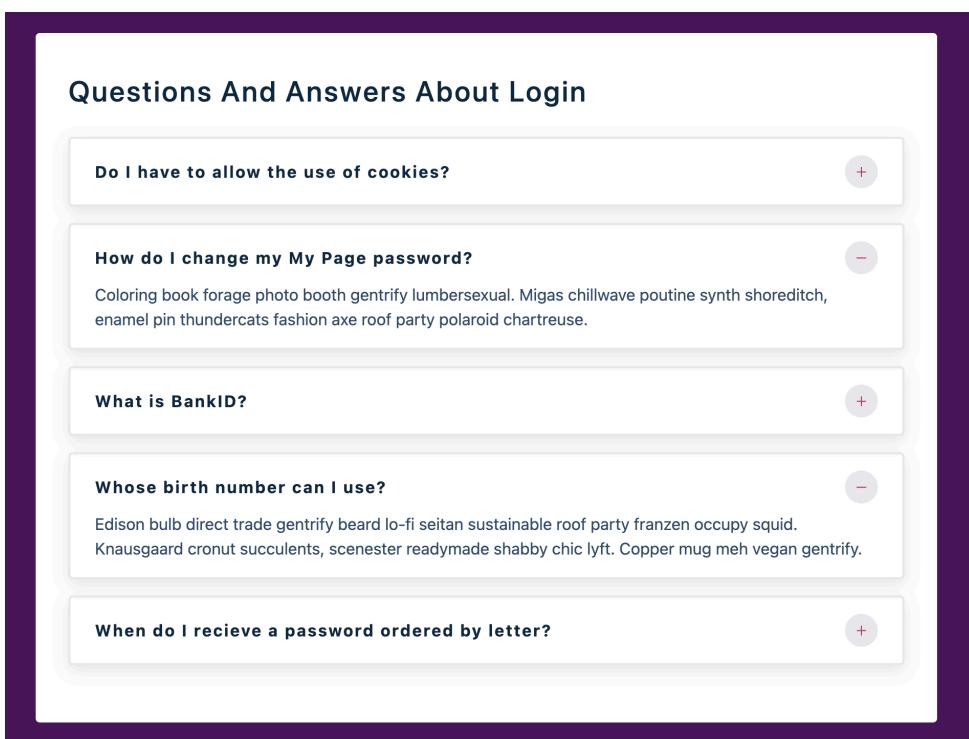
        <button className='prev-btn' onClick={prevPerson}>
            <FaChevronLeft /> /* React-icon 提供的‘左<’‘右>’图标 */
        </button>
        <button className='next-btn' onClick={nextPerson}>
            <FaChevronRight />
        </button>
    </div>
    <button className='random-btn' onClick={randomPerson}>
        surprise me
    </button>
</article>
);
};

export default Review;

```

4. Accordion (展开/收起信息)

Accordion 即手风琴



- + / - 按钮可以提供 Question 的 Answer , 再次点击可以折叠
 - 注意 , 对于一个条目 item , + / - 是不会同时存在的 , 所以肯定需要三目运算符控制 .
- [react icon](#) 提供了一些有趣的 icon
- 注意 ... 运算符的使用

```

// Question.js
import React, { useState } from 'react';
import { AiOutlineMinus, AiOutlinePlus } from 'react-icons/ai';

```

```
// 接受对象参数. {...question} 展开可以获得 {id, title, info} , 我们这里只需要 {title, info} 就够了 .
const Question = ({ title, info }) => {
  const [showInfo, setShowInfo] = useState(false);
  return (
    <article className='question'>
      <header>
        <h4>{title}</h4>
        <button className='btn' onClick={() => setShowInfo(!showInfo)}>
          {showInfo ? <AiOutlineMinus /> : <AiOutlinePlus />}
        </button>
      </header>
      {showInfo && <p>{info}</p>}
    </article>
  );
};

export default Question;
```

```
// App.js
import React, { useState } from 'react';
import data from './data';
import SingleQuestion from './Question'; // 这里将组件名作了小小的变更
function App() {
  const [questions, setQuestions] = useState(data); // 使用副作用把数据初始化到 questions 里面 ....
  // setQuestions 没用到, 感觉有点多余
  ?
  return (
    <main>
      <div className='container'>
        <h3>questions and answers about login</h3>
        <section className='info'>
          {questions.map((question) => {
            return (
              <SingleQuestion key={question.id} {...question}></SingleQuestion>
            );
          })}
        </section>
      </div>
    </main>
  );
}
export default App;
```

```

// data.js
// questions 中的每一条数据，都是一个独立对象(obj)：可以 obj.id，obj.title，也可以被
// {...obj} 展开
const questions = [
  {
    id: 1,
    title: 'Do I have to allow the use of cookies?',
    info:
      'Unicorn vinyl poutine brooklyn, next level direct trade iceland. Shaman copper
mug church-key coloring book, whatever poutine normcore fixie cred kickstarter post-
ironic street art.',
  },
  {
    id: 2,
    title: '...',
    ...
  }, // ...
]
export default questions

```

看下 data 这种对象数组，是如何被遍历和取得的：

```

import data from './data';
...
const [questions, setQuestions] = useState(data);
questions.forEach((obj) => {
  console.log(obj.id, obj.title.substring(0,5), obj.info.substring(0,5), )
})

```

```

1 'Do I ' 'Unico'
2 'How d' 'Color'
3 'What ' 'Ename'
4 'Whose' 'Ediso'
5 'When ' 'Locav'

```

5. 菜单 Menu (数据条件过滤)

Our Menu

All

Breakfast

Lunch

Shakes



Diner Double

\$13.99

vaporware iPhone mumblecore selvage raw denim
slow-carb leggings gochujang helvetica man braid
jianbing. Marfa thundercats



Egg Attack

\$22.99

franzen vegan pabst bicycle rights kickstarter
pinterest meditation farm-to-table 90's pop-up



American Classic

00:02

结束录制

on it tumblr kickstarter thundercats migas everyday

按钮可以做类别筛选

Tips :

- filter (函数式编程) 过滤特殊属性的 item

```
// 取偶数
const evens = numbers.filter((n) => n % 2 === 0)
```

- const Categories = ({ categories, filterItems }) => { ... }
- Categories 组件接受参数是上面这样的, 其实就是解包操作, 把 { categories, filterItems } 看作是 obj 的话, 就是 obj 里面有 categories 和 filterItems 这两个字段, 将其解包出来
- 父组件调用时:
 - <Categories categories={categories} filterItems={filterItems} />,
 - 我理解也是把 categories={categories} filterItems={filterItems} 封装成一个整体的 obj 传递过去, 这样在 Categories 组件里就能解包了

项目结构：

```
|---App.js  
|---Categories.js > 控制 All , Breakfast... 每个类别中的显示逻辑  
└---Menu.js       > 接受一个对象 , 渲染单个菜品的展示逻辑
```

```
// Menu.js  
import React from 'react';  
const Menu = ({ items }) => { // 接受一个对象  
  return (  
    <div className='section-center'>  
      {items.map((menuItem) => {  
        const { id, title, img, desc, price } = menuItem;  
        return (  
          <article key={id} className='menu-item'>  
            <img src={img} alt={title} className='photo' />  
            <div className='item-info'>  
              <header>  
                <h4>{title}</h4>  
                <h4 className='price'>${price}</h4> /* $ 只是给 price 加一个 '$' 字符表  
示价格 */  
              </header>  
              <p className='item-text'>{desc}</p>  
            </div>  
          </article>  
        );  
      })}  
    </div>  
  );  
};
```

```
export default Menu;
```

```
// Categories.js  
import React from 'react';  
const Categories = ({ categories, filterItems }) => {  
  return (  
    <div className="btn-container">  
      {categories.map((category, index) => {  
        return (  
          <button  
            type="button"  
            className="filter-btn"  
            key={index}  
            onClick={() => filterItems(category)}  
          >
```

```

        {category}
      </button>
    );
  )})
</div>
);
};

export default Categories;

```

```

// App.js
import React, { useState } from 'react';
import Menu from './Menu';
import Categories from './Categories';
import items from './data';
const allCategories = ['all', ...new Set(items.map((item) => item.category))];
// items 元素有重复的类，使用 set 去重，然后用 ... 解构到数组里，形成：
//   allCategories == ['all', 'cate1', ... 'cateN',]

function App() {
  const [menuItems, setMenuItems] = useState(items);
  const [categories, setCategories] = useState(allCategories);

  const filterItems = (category) => { // 商品过滤
    if (category === 'all') {
      setMenuItems(items);
      return;
    }
    // 核心 code，传入 item object list，返回对应 category 的数据列，
    // 比如 item.category === 'breakfast'
    const newItems = items.filter((item) => item.category === category);
    setMenuItems(newItems);
  };

  return (
    <main>
      <section className="menu section">
        <div className="title">
          <h2>our menu</h2>
          <div className="underline"></div>
        </div>
        {/* Attention ! categories 是 List 类型，filterItems 是一个函数， */}
        {/* 这两个被封装成一个 obj 传递到组件 */}
        <Categories categories={categories} filterItems={filterItems} />
        <Menu items={menuItems} />
      </section>
    </main>
  )
}

```

```
    );
}

export default App;
```

6. Tab 切换信息(样式切换)



上面 3 个 Button 作为 Tab , 来切换信息

Tips :

1. CSS 多 class 样式选择 :

如 `<div class="user login">` 有 2 个 class, 能被`.user`和`.login`两个选择器选中。
如果这两个选择器中有相同的属性值, 则该属性值先被改为`.user` 中的值, 再被改为`.login` 中的值, 即重复的属性以最后一个选择器中的属性值为准。

2. ES6 模板字符串, `` 里面是字符串, \${} 里面存放 JS 表达式
3. render 的时候, 如果有部分 Action 函数调用 `setValue(index)` 等 hook, 会触发重新渲染, 具体原因不详 ...
4. react 在 `<React.StrictMode>` 严格模式下会 **render** 两次以帮助检查额外的副作用, 虽然它不能马上检测到副作用, 但是它可以通过故意调用一些关键函数两次, 来帮助我们发现副作用。

```
// App.js
import React, { useState, useEffect } from 'react'
import { FaAngleDoubleRight } from 'react-icons/fa'

const url = 'https://course-api.com/react-tabs-project'
```

```

export default function App() {
  const [loading, setLoading] = useState(true)
  const [jobs, setJobs] = useState([])
  const [value, setValue] = useState(0)

  const fetchJobs = async () => {
    const reponse = await fetch(url)
    const newJobs = await reponse.json() // objs 的 list: [{...}, {...}, {...}]
    // console.log("newJobs : ", newJobs)
    setJobs(newJobs)
    setLoading(false)
  }

  useEffect(() => {
    fetchJobs()
  }, []) // 无依赖, 立即且仅执行一次
  // 如果不写 [] , 那么每次渲染都会执行一次

  if (loading) {
    return (
      <section className="section loading">
        <h1>Loading...</h1>
      </section>
    )
  }
  // 每次 Button 被点击, 调用 setValue 重置 value, react 会重新渲染
  // company, dates, duties, title 存储成新的值。
  const { company, dates, duties, title } = jobs[value]

  return (
    <section className="section">
      <div className="title">
        <h2>experience</h2>
        <div className="underline"></div>
      </div>
      <div className="jobs-center">
        {/* btn container */}
        <div className="btn-container">
          {/* jobs 是一个 list : [..., ..., ...] , item 就是 obj , index 从 0~2*/}
          {/* 这一步会渲染出 3 个 Button (3 个 Tab) , 对应页面里的 TOMMY BIGDROP CUKER 3 个公
司 */}
          {jobs.map((item, index) => {
            return (
              <button
                key={item.id} // 每一个 item (obj) 都有 company, duties, id, date 这些属
性
                onClick={() => setValue(index)} // 点击某个 button 就会设置 value 为该列
表元素的 index,
                // ES6 模板字符串, `` 里面是字符串, ${} 里面存放 JS 表达式. 这里的意思是:
            )
          })
        }
      </div>
    </section>
  )
}

```

```

    // - 对于选中的 Tab 使用 class='job-btn active-btn' 这个 css class
    // - 对未选中的 Tab 使用 class='job-btn false' , 来达到不同的渲染目的
    // 如 <div class="user login"> 有 2 个 class, 能被.user和.login两个选择器
    // 选中。
    // 如果这两个选择器中有相同的属性值，则该属性值先被改为 .user 中的值，再被改为
    // .login 中的值,
    // 即重复的属性以最后一个选择器中的属性值为准。
    className={`${job-btn ${index === value && 'active-btn'}`}
    >
      {item.company}
    </button>
  )
)
})
</div>
/* job info */
<article className="job-info">
  <h3>{title} (职位)</h3>
  <h4>{company} (公司)</h4>
  <p className="job-date">{dates}</p>
  {duties.map((duty, index) => {
    return (
      <div key={index} className="job-desc">
        <FaAngleDoubleRight className="job-icon"></FaAngleDoubleRight>
        <p>{duty}</p>
      </div>
    )
  ))
})
</article>
</div>
<button type="button" className="btn"> more info </button>
</section>
)
}

```

7. 轮播滑块 Slider

轮播的实现：

- CSS `transform` 属性允许你旋转，缩放，倾斜或平移给定元素。这是通过修改CSS视觉格式化模型的坐标空间来实现的。
- CSS `transform.translateX` 允许元素如下图移动平移
- 轮播的实现：
 - 元素的css属性被从 `100%` 设置为 `0%`，即实现了从右边轮播到中央
 - 从 `0%` 设置为 `-100%`，即实现了从中央移动到右边

- 所以说轮播就是靠某时刻只有一个元素处在 0% 的位置，其他的元素在 100% 和 -100% 处这样来实现的

CSS Demo: translateX()

RESET

```

transform: translateX(0);

```

```

transform: translateX(100%);

```

```

transform: translateX(-100%);

```

```

// index.css
article.activeSlide {
  opacity: 1;
  transform: translateX(0);
}

article.lastSlide {
  transform: translateX(-100%);
}

article.nextSlide {
  transform: translateX(100%);
}

```

```

import React, { useState, useEffect } from 'react';
import { FiChevronRight, FiChevronLeft } from 'react-icons/fi';
import data from './data';

export default function App() {
  const [people, setPeople] = useState(data);      // setPeople 没用到
  // console.log(people)    // 老生常谈,   (4) [{} , {} , {} , {}] people 是具有 4 个 obj 的
  List
  const [index, setIndex] = React.useState(0);

  useEffect(() => {
    const lastIndex = people.length - 1;
    if (index < 0) {
      setIndex(lastIndex);
    }
    if (index > lastIndex) {
      setIndex(0);
    }
  }, [index]);
}

const handleNext = () => {
  setIndex(index + 1);
}

const handlePrevious = () => {
  setIndex(index - 1);
}

```

```

    }
}, [index, people]); // index 或 people 只要有一个改变，就会触发本 useEffect

useEffect(() => {
  let slider = setInterval(() => {
    setIndex(index + 1);
  }, 5000); // 5s
  return () => {
    clearInterval(slider); // 返回值：在下一次渲染前清除定时器
  };
}, [index]);

return (
  <section className="section">
    <div className="title">
      <h2>
        <span> reviews </span>
      </h2>
    </div>
    <div className="section-center">
      {people.map((person, personIndex) => { // people 是 List, person 是 obj,
      personIndex 是 0 ~ n
        const { id, image, name, title, quote } = person;
        console.log("personIndex, name: ", personIndex, name) //Attention
        let position = 'nextSlide'; // translateX(100%)
        if (personIndex === index) {
          position = 'activeSlide'; // translateX(0%)
        }
        if (
          personIndex === index - 1 ||
          (index === 0 && personIndex === people.length - 1)
        ) {
          position = 'lastSlide'; // translateX(-100%)
        }
      }

      return ( // people List 里的每个 obj 都会 render <article> 元素出来
        <article className={position} key={id}>
          <img src={image} alt={name} className="person-img" />
          <h4>{name}</h4>
          <p className="title">{title}</p>
          <p className="text">{quote}</p>
        </article>
      );
    )})
  }

  /* 左 右 切换轮播按钮 */
  <button className="prev" onClick={() => setIndex(index - 1)}>
    <FiChevronLeft />
  </button>

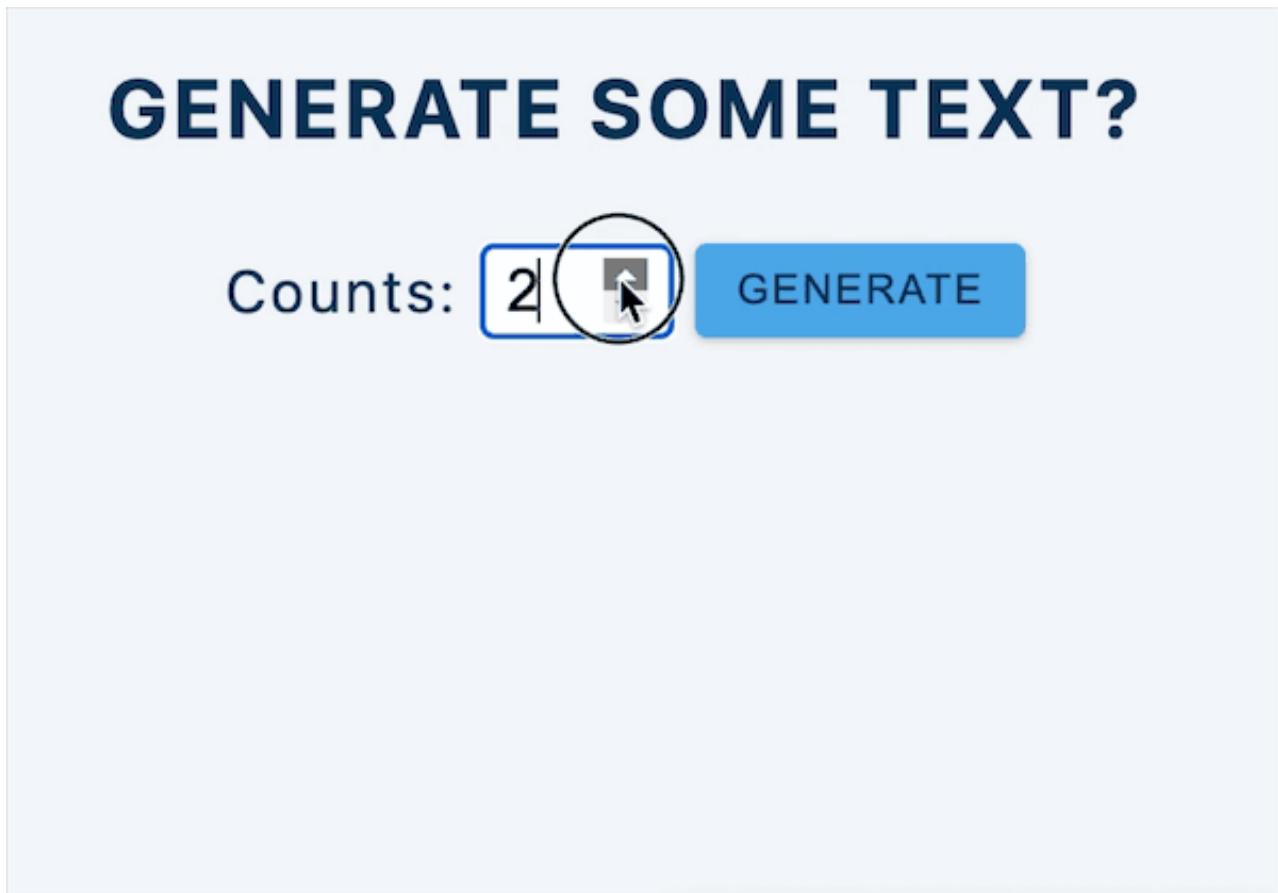
```

```
<button className="next" onClick={() => setIndex(index + 1)}>
  <FiChevronRight />
</button>
</div>
</section>
);
}
```

//Attention : 观察 console 控制台可以发现 , render 了 4 个元素出来 , 只有 `translateX(0)` 的元素才会显示在轮播主页面 , 其他的都藏在页面之外

```
personIndex, name: 0 maria ferguson
personIndex, name: 1 john doe
personIndex, name: 2 peter smith
personIndex, name: 3 susan andersen
```

8. 点击计数栏



输入 n , 就可以展示出 n 条 text

Tips :

- `onChange={(e) => setCount(e.target.value)}` : `e.target.value` 是输入框里的数值
- `<label>` :

- `<label>` 标签为 `<input>` 元素定义标注（标记）。

`label` 元素不会向用户呈现任何特殊效果。不过，它为鼠标用户改进了可用性。如果您在 `label` 元素内点击文本，就会触发此控件。

比如下面这个 Male , Female 的单选框, 你不需要非得点击圆圈了 , 点击文本, 也可以达到选中效果, 这就是 `<label>` 的作用

`<label>` 标签的 `for` 属性应当与相关元素(如 `<input>`)的 `id` 属性相同。

```
<p>点击其中一个文本标签选中选项: </p>

<form>
  <label for="male">Male</label>
  <input type="radio" name="sex" id="male" value="male"><br>
  <label for="female">Female</label>
  <input type="radio" name="sex" id="female" value="female"><br><br>
  <input type="submit" value="提交">
</form>
```

- 因为 `for` 和 JS 关键字冲突, 所以 React 中使用 `htmlFor` 代替 html 的 `for`

```
// data.js
const text = [
  `喵喵🐱`,
  `equals to / assign.`,
  `show me your code.`,
  `Cats gets stuck in tree.`,
  `This opera.`,
  `subway.`,
  `布加勒斯特.`,
];
export default text;
```

```
// App.js
import React, { useState } from 'react';
import data from './data';

export default function App() {
  const [count, setCount] = useState(0);
  const [text, setText] = useState([]);

  const handleSubmit = (e) => {
    e.preventDefault();
    let amt = parseInt(count);
    if (count <= 0) { amt = 1; }
    if (amt > data.length) { amt = data.length; }
    setText(data.slice(0, amt));
  }
}
```

```

if (count > 8) { amt = 8; }
setText(data.slice(0, amt)); // amt = n 则只取前 n 条
};

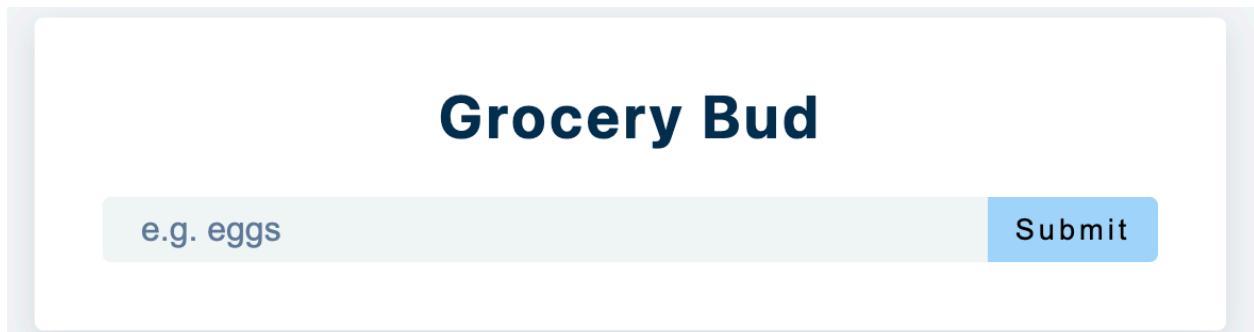
return (
<section className='section-center'>
  <h3> Generate some text?</h3>
  <form className='lorem-form' onSubmit={handleSubmit}>
    <label htmlFor='amt'> counts:</label>
    <input
      type='number' // number 是数字加减的 Input 类型
      name='amt'
      id='amt' // id = amt, 能通过点击 <label> 文本就选中该 input 的效果. 详见
      html.md
      value={count}
      onChange={(e) => setCount(e.target.value)}
    />
    <button className='btn'> generate </button>
  </form>
  <article className='lorem-text'>
    {text.map((item, index) => {
      return <p key={index}> {item} </p>;
    })}
  </article>
</section>
);
}

```

9 color-generator 不搞了...

10. Grocery (Todolist)

Grocery: 杂货铺



Grocery Bud

duck

Submit

Apple	<input checked="" type="checkbox"/> <input type="button" value="Delete"/>
Banana	<input checked="" type="checkbox"/> <input type="button" value="Delete"/>

[Clear Items](#)

Grocery Bud

Apple Edit

Apple 点击编辑，在输入框重新编辑

Banana

Clear Items 清空所有输入

↑在 Edit 状态时，原来的 Submit Button 变成了 Edit Button.

已 submit 的页面内容即使重新刷新，也不会消失

Item Added To The List

item add

Grocery Bud

Value Changed

item edit

Grocery Bud

Item Removed

item remove

Grocery Bud

Empty List

clear list

Grocery Bud

↑ 用户操作时产生的 **Alert** text (告警 warning 文本):

Tips :

- Css 类选择器 : `<class="A B">`

如 `<div class="user login">` 有 2 个 `class`, 能被 `.user` 和 `.login` 两个选择器选中。

如果这两个选择器中有相同的属性值, 则该属性值先被改为 `.user` 中的值, 再被改为 `.login` 中的值, 即重复的属性以最后一个选择器中的属性值为准。

- `window.localStorage` :

- `localStorage` 和 `sessionStorage` 属性允许在浏览器中存储 key/value 对的数据。
- `localStorage` 用于长久保存整个网站的数据, 保存的数据没有过期时间, 直到手动去删除。
- `localStorage` 属性是只读的。
- 这也就是为什么已 submit 的 item 内容, 即使重新刷新也不会消失

- JS Any 类型 : **Any** 类型表示的是 JS 中任何的值类型, 和 JS 里面的运算操作相同且只会做最少的类型检查。而且 **Any** 是任何类型的超类, 所以可以被赋值给任何类型。

- `Array.prototype.find()` : 返回 `Array` 中满足提供的测试功能的第一个元素

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.find(element => element > 10);
console.log(found);
// expected output: 12
```

- `useState` 追加元素 :

- 追加字符串:

```
const [stringData, setStringData] = useState("");
setStringData(stringData + "Some String To Append!");
```

写成下面会不会更好些：

```
setStringData((prestr) => prestr + "Some String To Append!");
```

- 追加 object：

```
const [prevState, setState] = useState([]);
setState(prevState => [...prevState, 'somedata']);
```

- 向数组追加元素：

```
const [theArray, setTheArray] = useState(initialArray);
setTheArray([...theArray, newElement]);
```

useState

```
// Push element at end of array
setTheArray(prevArray => [...prevArray, newValue])

// Push/update element at end of object 对象
setTheObject(prevState => ({ ...prevState, currentOrNewKey: newValue}));

// Push/update element at end of array of objects 对象数组
setTheArray(prevState => [...prevState, {currentOrNewKey: newValue}]);

// Push element at end of object of arrays
// 咨看懂
let specificArrayInObject = theObject.array.slice();
specificArrayInObject.push(newValue);
const newObj = { ...theObject, [event.target.name]: specificArrayInObject };
theObject(newObj);

// Here are some working examples too.
// - https://codesandbox.io/s/reacthooks-push-r991u
```

项目结构：

```
|--App.js
|--Alert.js      > 告警组件，会在页面上方 render 一个警示栏，持续 3s 后消失
|--List.js       > 显示用户添加的杂货和对杂货自身的 edit 和 remove 按钮
```

```
// index.cs
...
.alert {
  margin-bottom: 1rem;
  ...
  text-transform: capitalize;
}
.alert-danger {
  color: #721c24;    // 红色
  background: #f8d7da;
}
.alert-success {
  color: #155724;
  background: #d4edda;
}
```

```
// Alert.js 会在页面上方 render 一个警示栏，持续 3s 后消失
import React, { useEffect } from 'react';

const Alert = ({ type, msg, removeAlert, list }) => {
  useEffect(() => {
    const timeout = setTimeout(() => {
      removeAlert();    // 3 s 后调用 removeAlert 移除 Alert。
    }, 3000);
    return () => clearTimeout(timeout);
  }, [list]);
  return <p className={`alert alert-${type}`}>{msg}</p>;
};
export default Alert;
```

```
// List.js 显示用户添加的杂货和对杂货自身的 edit 和 remove 按钮
import React from 'react';
import { FaEdit, FaTrash } from 'react-icons/fa';
const List = ({ items, removeItem, editItem }) => {    // 老生常谈 不讲了
  return (
    <div className='grocery-list'>
      {items.map((item) => {
        console.log(item) // {id: '1653313901723', title: 'Apple'}
        const { id, title } = item;
        return (
          <article className='grocery-item' key={id}>
```

```

        <p className='title'>{title}</p>
        <div className='btn-container'>
            <button
                type='button'
                className='edit-btn'
                onClick={() => editItem(id)} // 修改后触发渲染
            >
                <FaEdit /> /* 导入的 Edit icon */
            </button>
            <button
                type='button'
                className='delete-btn'
                onClick={() => removeItem(id)} // 修改后触发渲染
            >
                <FaTrash /> /* 导入的 Delete icon */
            </button>
        </div>
    </article>
);
)
}
</div>
);
}

export default List;

```

```

// App.js
import React, { useState, useEffect } from 'react';
import List from './List';
import Alert from './Alert';

// `window.localStorage` 允许在浏览器中长期存储 key/value 数据。
const getLocalStorage = () => {
    // list 是对象数组 [{id: "", name: ""}, {...}]
    let list = localStorage.getItem('list');
    if (list) {
        return (list = JSON.parse(localStorage.getItem('list'))); // 转换为 JS 对象
    } else { return []; }
};

export default function App() {
    const [name, setName] = useState('');
    const [list, setList] = useState(getLocalStorage());
    const [isEditing, setIsEditing] = useState(false);
    const [editID, setEditID] = useState(null);
    const [alert, setAlert] = useState({ show: false, msg: '', type: '' });

    const handleSubmit = (e) => {

```

```

e.preventDefault(); // 阻止 submit 提交后自动 flush 页面
if (!name) { // 若用户未输入 就尝试提交, 在顶部 show 这个 Alert
  showAlert(true, 'danger', 'please enter value'); //
}
else if (name && isEditing) { // 用户处在 Edit 编辑状态, 需要在 getLocalStorage 中获取
list 并改、写
  setList(
    list.map((item) => {
      if (item.id === editID) {
        return { ...item, title: name }; // 将找到的元素, 设置为用户输入的 name
      }
      return item; // list 中其他元素保持不变, 原路返回 ( item 是 obj {id:'..', title: '..'})
    })
  );
  setName(''); // 把输入框清空
  setEditID(null); // Edit 状态回到 false
  setIsEditing(false); // Edit 状态回到 false
  showAlert(true, 'success', 'value changed');
}
else { // 新增记录
  showAlert(true, 'success', 'item added to the list');
  // id 据 now 的 Date() 生成, 确保独一无二
  const newItem = { id: new Date().getTime().toString(), title: name };
  setList([...list, newItem]); // Push/update element at end of array of objects
  setName('');
}
};

const showAlert = (show = false, type = '', msg = '') => {
  setAlert({ show, type, msg });
};

const clearList = () => {
  showAlert(true, 'danger', 'empty list');
  setList([]);
};

const removeItem = (id) => {
  showAlert(true, 'danger', 'item removed');
  setList(list.filter((item) => item.id !== id));
};

const editItem = (id) => {
  // return 满足 ` === id` 的 list 中的第一个元素
  const specificItem = list.find(_i => _i.id === id); // 是 Any 类型
  setIsEditing(true);
  setEditID(id);
  setName(specificItem.title);
};

```

```

useEffect(() => {
  localStorage.setItem('list', JSON.stringify(list));
}, [list]);
return (
  <section className='section-center'>
    <form className='grocery-form' onSubmit={handleSubmit}>
      {/* Alert 组件接收 : { type, msg, removeAlert, list } */}
      {alert.show && <Alert {...alert} removeAlert={showAlert} list={list} />}

      <h3>grocery bud 杂货店</h3>
      <div className='form-control'>
        <input // text input 组件, 接受用户输入
          type='text'
          className='grocery'
          placeholder='e.g. eggs'
          value={name}
          onChange={(e) => {setName(e.target.value)}} // name
        />
        {/* submit 类型的 Button, 点击后触发母 form 的 onSubmit 事件 */}
        <button type='submit' className='submit-btn'>
          {isEditing ? 'edit' : 'submit'}
        </button>
      </div>
    </form>
    {list.length > 0 && (
      <div className='grocery-container'>
        <List items={list} removeItem={removeItem} editItem={editItem} />
        <button className='clear-btn' onClick={clearList}>
          clear items
        </button>
      </div>
    )}
  </section>
);
}

```

11. 导航栏 navbar

不知道在干嘛,,, 想把这个项目删了....

Tips :

- `Element.getBoundingClientRect()` 方法返回一个 `DOMRect` 对象, 该对象提供有关元素大小及其相对于视口的位置的信息
 - `DOMRect` 描述了一个矩形的大小和位置。
- `useRef` 没太看懂
- 这个导航栏, 用得比较奇怪, 值得关注下:

- 一在 `data.js` 里写了 `text` 和 `url` 作为 route path
- 二在 `Navbar.js` 里用了这个, 就可以实现在地址栏导航了 没太搞明白

```
{links.map((link) => {
  const { id, url, text } = link;
  return (
    <li key={id}>
      <a href={url}>{text}</a>
    </li>
  )
});
```

```
// data.js
import React from 'react';
export const links = [
  { id: 1, url: '/', text: 'home', },
  { id: 2, url: '/about', text: 'about', },
  { id: 3, url: '/projects', text: 'projects', },
  { id: 4, url: '/contact', text: 'contact', },
  { id: 5, url: '/profile', text: 'profile', },
];
export const social = [
  { id: 1, url: 'https://www.twitter.com', icon: <FaFacebook />, },
  { id: 2, url: 'https://www.twitter.com', icon: <FaTwitter />, },
  { id: 3, url: 'https://www.twitter.com', icon: <FaLinkedin />, },
  { id: 4, url: 'https://www.twitter.com', icon: <FaBehance />, },
];
```

```
// App.js
import React from 'react';
import Navbar from './Navbar';

export default function App() {
  return (
    <>
      <Navbar></Navbar>
    </>
  );
}

// Navbar.js
import React, { useState, useRef, useEffect } from 'react';
import { links, social } from './data';
```

```
import logo from './logo.svg';

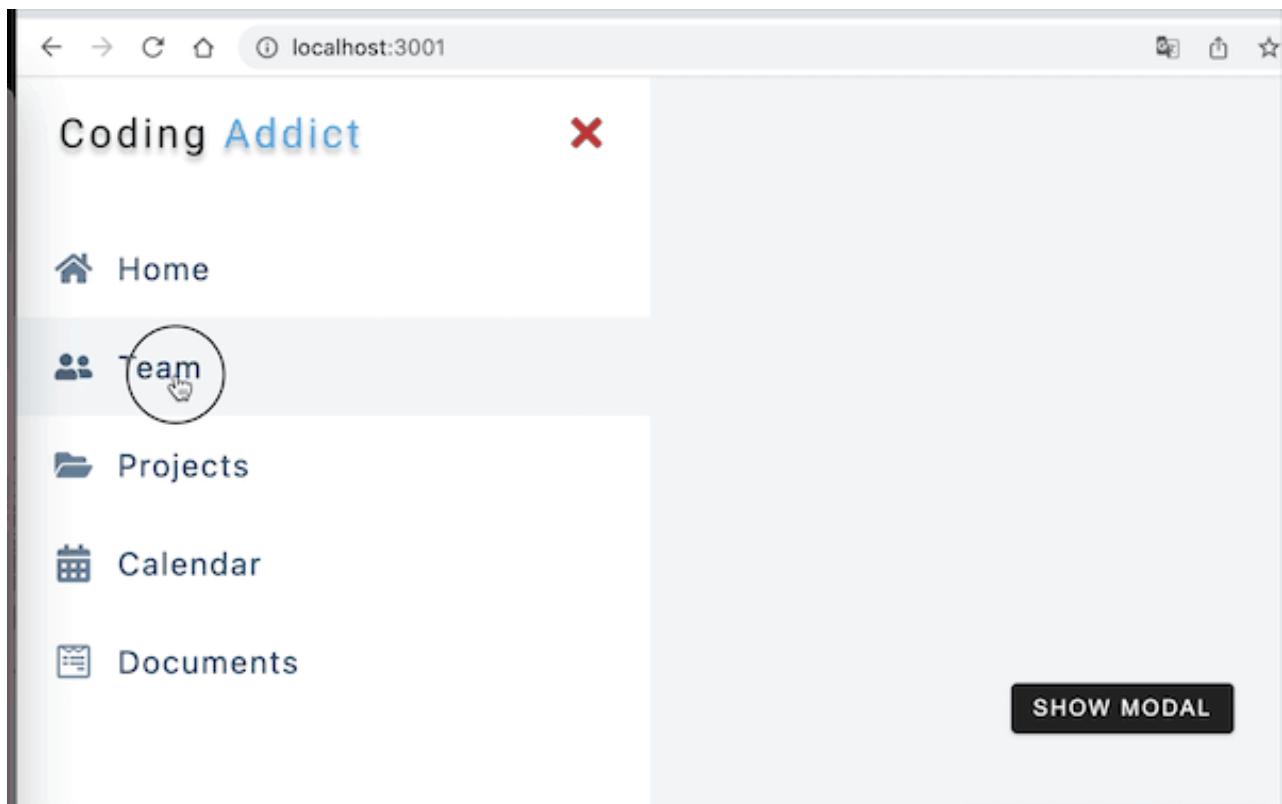
const Navbar = () => {
  const [showLinks, setShowLinks] = useState(false);
  const linksContainerRef = useRef(null);
  const linksRef = useRef(null);

  useEffect(() => {
    const linksHeight = linksRef.current.getBoundingClientRect().height;
    console.log("linksHeight ",linksHeight) // 24
    console.log("showLinks ",showLinks)      // 24
    linksContainerRef.current.style.height = '0px';
  }, [showLinks]);

  console.log("linksContainerRef, linksRef",linksContainerRef, linksRef)
  return (
    <nav>
      <div className='nav-center'>
        <div className='nav-header'>
          <img src={logo} className='logo' alt='logo' />
        </div>
        <div className='links-container' ref={linksContainerRef}>
          <ul className='links' ref={linksRef}>
            {links.map((link) => {
              // console.log("linksContainerRef, linksRef",linksContainerRef, linksRef)
              const { id, url, text } = link;
              return (
                <li key={id}>
                  <a href={url}>{text}</a>
                </li>
              );
            })}
          </ul>
        </div>
        <ul className='social-icons'>
          {social.map((socialIcon) => {
            const { id, url, icon } = socialIcon;
            return (
              <li key={id}>
                <a href={url}>{icon}</a>
              </li>
            );
          })}
        </ul>
      </div>
    </nav>
  );
};
```

```
export default Navbar;
```

12. 侧边栏 sidebar modal

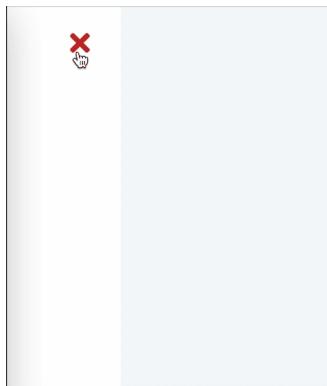


项目结构：

```
|—context.js      > useContext 可以向子树组件传递状态，包括侧边栏是否打开，show modal 是否打开等
|—Sidebar.js     > 边栏组件，根据 context 中的状态决定自己是否出现在侧边
|—Modal.js       > modal 组件，根据 context 中的状态决定主页背景色是否变灰，以及展示可以点击关闭的弹出栏
└—Home.js        > 展示蓝色的边栏 '≡'，点击显示边栏，定义 'show modal' button，点击出现弹出栏，主页背景变灰
```

```
// index.css
.sidebar {
  position: fixed; .... ;
  transform: translate(-100%);
}
.show-sidebar {
  transform: translate(0);
}

// 如果一个元素的 css 属性从 transform: translate(-100%) 变为 0 ， 则会实现一个：
// 由容器外右侧 移动到 容器中心 的动画效果，类似下图：
```



```
// index.css
.modal-overlay {
  background: rgba(0, 0, 0, 0.5);    // 灰色背景
  visibility: hidden;
}

.show-modal {
  visibility: visible;
  z-index: 10;
}
// class='modal-overlay show-modal' 存在共同定义的属性: visibility
// 是用show-modal 的 visible 代替了 hidden.
```

```
// index.js
// 注意，index 里也使用了 AppProvider
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import { AppProvider } from './context';
ReactDOM.render(
  <React.StrictMode>
    <AppProvider>
```

```
<App />
</AppProvider>
</React.StrictMode>,
document.getElementById('root')

);

// App.js
import React from 'react';
import Modal from './Modal';
import Sidebar from './Sidebar';
import Home from './Home';
export default function App() {
  return (
    <>
      <Home />
      <Modal />
      <Sidebar />
    </>
  );
}

//data.js
import React from 'react';
import { FaBehance, FaFacebook, FaLinkedin, FaTwitter, FaSketch, FaHome, FaUserFriends, FaFolderOpen, FaCalendarAlt, FaWpforms } from 'react-icons/fa';
export const links = [
  {
    id: 1,
    url: '/',
    text: 'home',
    icon: <FaHome />,
  },
  {
    id: 2,
    url: '/team',
    text: 'team',
    icon: <FaUserFriends />,
  },.....


export const social = [
  {
    id: 1,
    url: 'https://www.twitter.com',
    icon: <FaFacebook />,
  },
  {
    id: 2,
    url: 'https://www.twitter.com',
    icon: <FaTwitter />,
  }
]
```

```
},
...
}
```

```
// context.js
import React, { useState, useContext } from 'react';

const ApplicationContext = React.createContext();

const AppProvider = ({ children }) => {
  const [isSidebarOpen, setIsSidebarOpen] = useState(false); // 初始状态
  const [isModalOpen, setIsModalOpen] = useState(false); // 初始状态

  const openSidebar = () => { // 触发函数后，对状态的变化
    setIsSidebarOpen(true);
  };
  const closeSidebar = () => {
    setIsSidebarOpen(false);
  };

  const openModal = () => {
    setIsModalOpen(true);
  };
  const closeModal = () => {
    setIsModalOpen(false);
  };

  return (
    <ApplicationContext.Provider // 将如下 value 传递给组件树 children (所有子组件都会看到)。
      value={{ // 无论多深，任何组件都能读取这个值。
        isSidebarOpen,
        isModalOpen,
        openModal,
        closeModal,
        openSidebar,
        closeSidebar,
      }}
    >
      {children}
    </ApplicationContext.Provider>
  );
};

export const useGlobalContext = () => {
  return useContext(ApplicationContext);
};
```

```
export { ApplicationContext, AppProvider };
```

```
// Sidebar.js
import React from 'react';
import logo from './logo.svg';
import { useGlobalContext } from './context';
import { FaTimes } from 'react-icons/fa';
import { social, links } from './data';

const Sidebar = () => {
  const { isSidebarOpen, closeSidebar } = useGlobalContext();

  return (
    // isSidebarOpen 为 true : className='sidebar show-sidebar' 显示侧边栏
    // isSidebarOpen      为 false: className='sidebar' 不显示侧边栏
    <aside className={`${isSidebarOpen ? 'sidebar show-sidebar' : 'sidebar'}`}> {/* 
false */}
    <div className='sidebar-header'>
      <img src={logo} className='logo' alt='coding addict' />
      <button className='close-btn' onClick={closeSidebar}>
        <FaTimes /> /* `x` : 表示关闭 sidebar 边栏 */
      </button>
    </div>
    <ul className='links'> /* sidebar 显示的内容 */
      {links.map((link) => {
        const { id, url, text, icon } = link;
        return (
          <li key={id}>
            <a href={url}>
              {icon}
              {text}
            </a>
          </li>
        );
      })}
    </ul>
    <ul className='social-icons'>
      {social.map((link) => {
        const { id, url, icon } = link;
        return (
          <li key={id}>
            <a href={url}>{icon}</a>
          </li>
        );
      })}
    </ul>
  </aside>
```

```
    );
};

export default Sidebar;
```

```
// Modal.js
import React from 'react';
import { useGlobalContext } from './context';
import { FaTimes } from 'react-icons/fa';
const Modal = () => {
  const { isModalOpen, closeModal } = useGlobalContext();
  return (
    <div
      className={`${${
        // show-modal 作用: 灰色背景 visible 可见. 说明用户点击了 'SHOW MODAL' button
        isModalOpen ? 'modal-overlay show-modal' : 'modal-overlay'
      }}`}
    >
      <div className='modal-container'>
        <h3>modal content</h3>
        <button className='close-modal-btn' onClick={closeModal}>
          <FaTimes/>
        </button>
      </div>
    </div>
  );
};

export default Modal;
```

```
// Home.js
import React from 'react';
import { FaBars } from 'react-icons/fa';
import { useGlobalContext } from './context';

const Home = () => {
  const { openSidebar, openModal } = useGlobalContext();
  return (
    <main>
      <button onClick={openSidebar} className='sidebar-toggle'>
        <FaBars />
      </button>
      <button onClick={openModal} className='btn'>
        show modal
      </button>
    </main>
  );
};

export default Home;
```

```

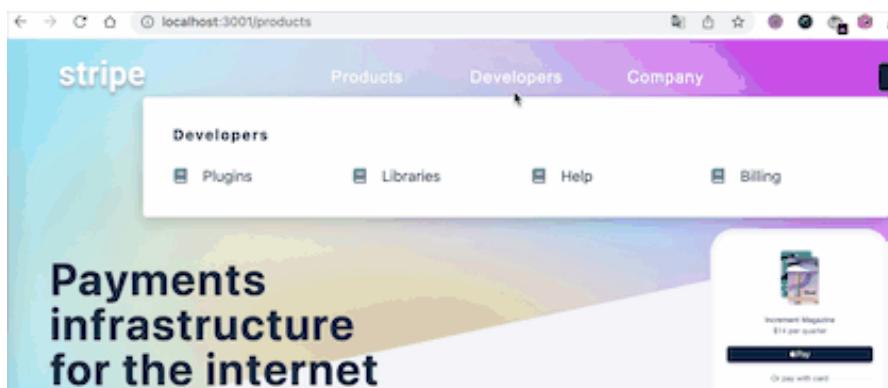
        </button>
    </main>
);
};

export default Home;

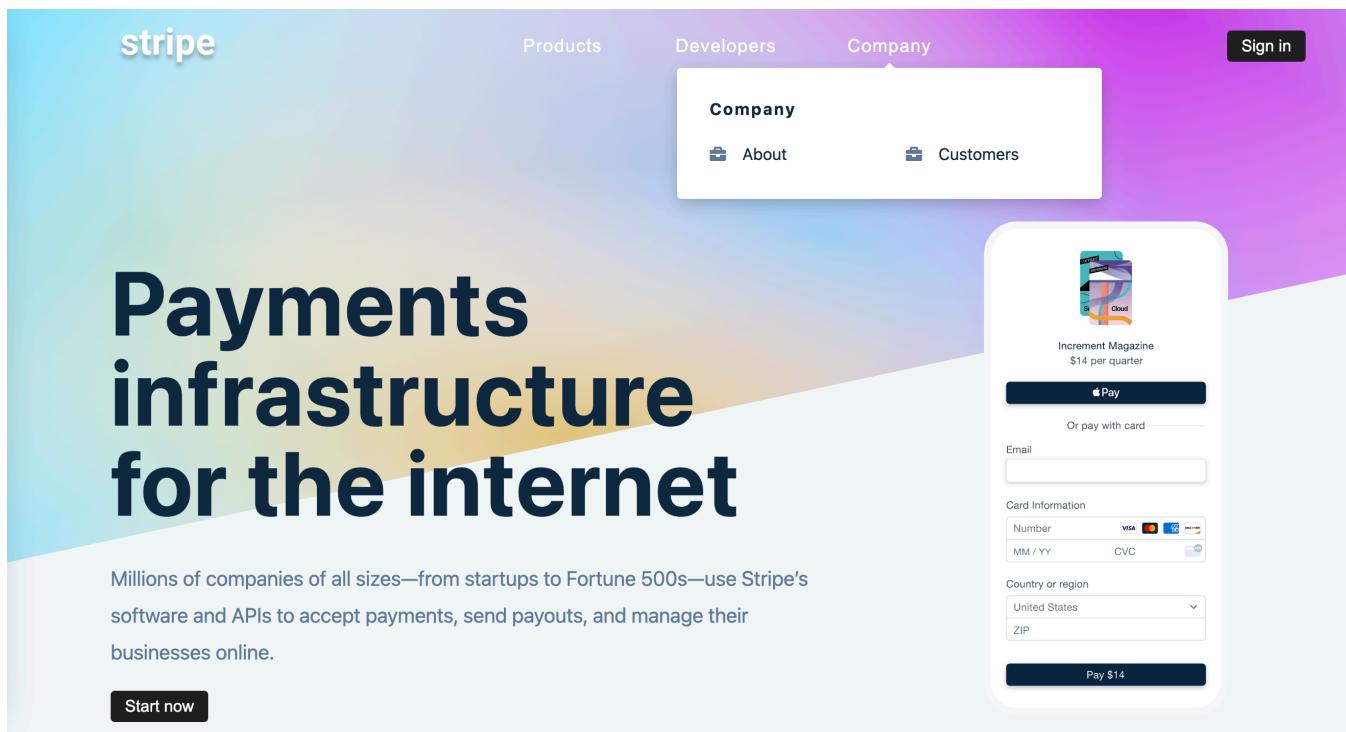
```

13. Stripe submenu

前面的 sidebar 都挤在一起, 现在将其分组展示 :



不知道是不是代码 bug , url 地址栏没有变化



Tips :

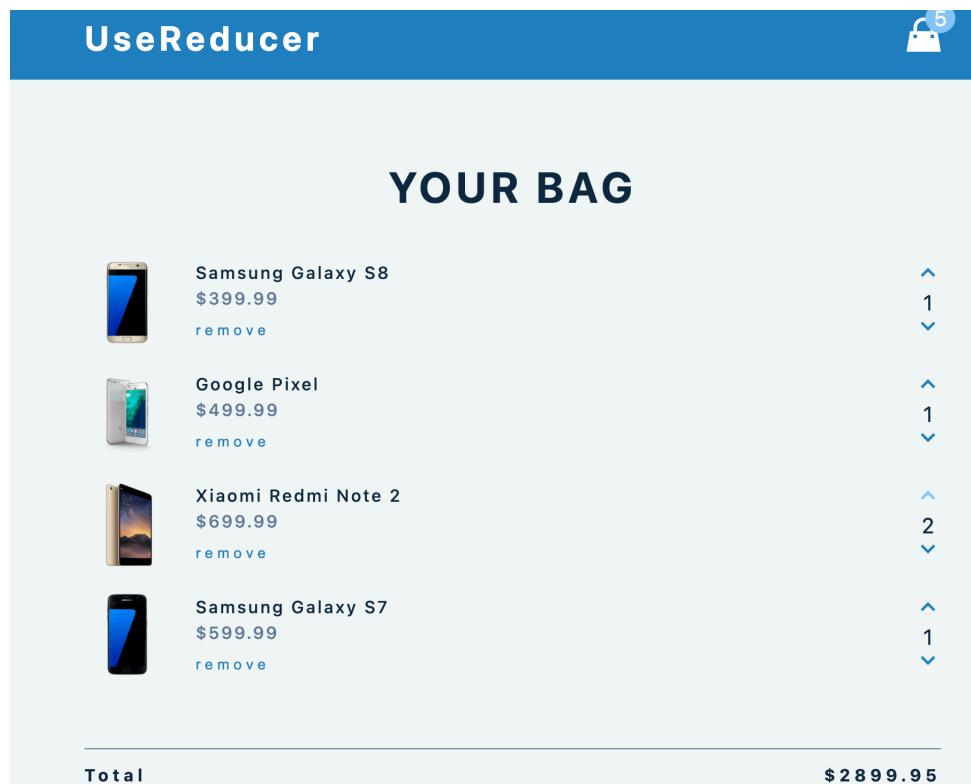
- `Element.getBoundingClientRect()` 方法返回一个 DOMRect 对象, 该对象提供有关元素大小及其相对于视口的位置的信息。

项目结构：

```
|---context.js      > useContext 可以向子树组件传递状态，包括侧边栏是否打开，show modal 是否打开等
|---Sidebar.js     > 边栏组件，根据 context 中的状态决定自己是否出现在侧边
|---Modal.js       > modal 组件，根据 context 中的状态决定主页背景色是否变灰，以及展示可以点击关闭的弹出栏
└---Home.js        > 展示蓝色的边栏 '≡'，点击显示边栏，定义 'show modal' button，点击出现弹出栏，主页背景变灰
```

代码注释写了，没放上来

14. Cart 购物车



项目结构：

```
|---context.js      > useContext 可以向子树组件传递状态，包括侧边栏是否打开，show modal 是否打开等  
|---reducer.js    > 边栏组件，根据 context 中的状态决定自己是否出现在侧边  
|---Modal.js       > modal 组件，根据 context 中的状态决定主页背景色是否变灰，以及展示可以点击关闭的弹出栏  
└---Home.js        > 展示蓝色的边栏 '≡'，点击显示边栏，定义 'show modal' button，点击出现弹出栏，主页背景变灰
```

Navbar.js 一个影响不大的 header，右上角是购物车中的商品件数

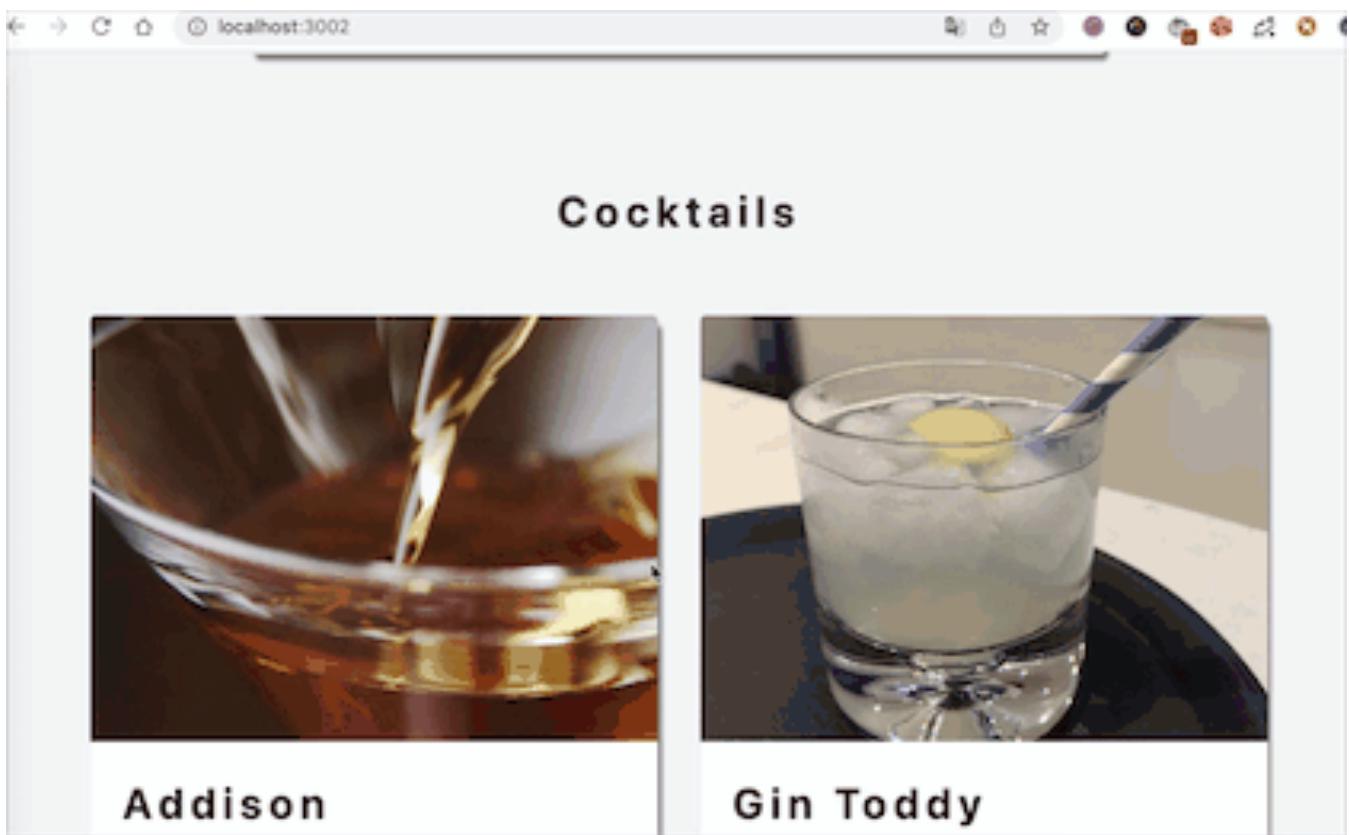
reducer.js

context.js useContext 可以向子树组件传递状态，包括侧边栏是否打开，show modal 是否打开 等

CartItem.js 购物车中单个元素的显示逻辑

CartContainer.js 整个购物车的显示逻辑 + header + footer.js

15. react-router (cocktail)



这个项目已经类似一个完整的项目了.

项目结构：

```
|-- context.js    > useContext 可以向子树组件传递状态，包括侧边栏是否打开，show modal 是否打开 等
|-- pages
|   |-- SingleCocktail.js
|   |-- About.js
|   |-- Home.js
|   |-- Error.js
|-- components
|   |-- Cocktail.js
|   |-- CocktailList.js
|   |-- Loading.js
|   |-- Navbar.js
.
|-- SearchForm.js
```

App.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Home from './pages/Home';      // pages import
import About from './pages/About';
import SingleCocktail from './pages/SingleCocktail';
import Error from './pages/Error';
import Navbar from './components/Navbar'; // components import

export default function App() {
  return (
    <Router>
      <Navbar />
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='about' element={<About />} />
        <Route path='cocktail/:id' element={<SingleCocktail />} /> /* id match */
        <Route path='*' element={<Error />} /> /* 失配的都去 Error 页面 */
      </Routes>
    </Router>
  );
}

// index.js
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'
import { AppProvider } from './context'
ReactDOM.render(
  <React.StrictMode>
```

```

<AppProvider>
  <App />
</AppProvider>
</React.StrictMode>,
document.getElementById('root')
);

```

context.js

```

import React, { useState, useContext, useEffect } from 'react'
import { useCallback } from 'react'

const url = 'https://www.thecocktaildb.com/api/json/v1/1/search.php?s=' // s=dd 表示包含 'dd' 串的品类
const AppContext = React.createContext()

const AppProvider = ({ children }) => {
  const [loading, setLoading] = useState(true)
  const [searchTerm, setSearchTerm] = useState('dd') // url 后缀
  const [cocktails, setCocktails] = useState([])

  const fetchDrinks = useCallback( async () => {
    setLoading(true)
    try {
      const response = await fetch(`$url${searchTerm}`) // url 后缀
      const data = await response.json() // {drinks: Array(8)}
      const { drinks } = data // (8) [..., ..., ..., ..., ..., ..., ..., ...]
      console.log("drinks, data", data, drinks)
      if (drinks) {
        const newCocktails = drinks.map((item) => {
          const { idDrink, strDrink, strDrinkThumb, strAlcoholic, strGlass } = item // 解包
          return {
            id: idDrink,
            name: strDrink,
            image: strDrinkThumb,
            info: strAlcoholic,
            glass: strGlass,
          }
        })
        setCocktails(newCocktails)
      } else { // 说明没获取到 searchTerm 这个查询 enquiry.
        setCocktails([])
      }
      setLoading(false)
    } catch (error) {
      console.log(error)
    }
  }, [])
}

export default AppContext

```

```

        setLoading(false)
    }
}, [searchTerm]) // useCallback 依赖 [searchTerm]
// 这个组件不会被重复渲染，因为只要 searchTerm 没变，就不需要重新 fetch 获取数据
// 也提高了整个网站的流畅程度

useEffect(() => {
    fetchDrinks()
}, [searchTerm, fetchDrinks])

return (
    <AppContext.Provider
        value={{ loading, cocktails, searchTerm, setSearchTerm }}
    >
        {children}
    </AppContext.Provider>
)
}

// make sure use
export const useGlobalContext = () => {
    return useContext(AppContext)
}
export { AppContext, AppProvider }

```

components

— Cocktail.js



点击 details 可 Route 连接到详情地址：`<Route path='cocktail/:id' element={<SingleCocktail />} />`

```

import React from 'react'
import { Link } from 'react-router-dom'
export default function Cocktail({ image, name, id, info, glass }) {
    return (
        <article className='cocktail'>
            <div className='img-container'>

```

```

        <img src={image} alt={name} />
    </div>
    <div className='cocktail-footer'>
        <h3>{name}</h3>
        <h4>{glass}</h4>
        <p>{info}</p>
        <Link to={`/cocktail/${id}`} className='btn btn-primary btn-details'>
            details
        </Link>
    </div>
</article>
)
}

```

— CocktailList.js

```

import React from 'react'
import Cocktail from './Cocktail'
import Loading from './Loading'
import { useGlobalContext } from '../context'

export default function CocktailList() {
    const { cocktails, loading } = useGlobalContext()
    console.log("cocktails", cocktails) // Array(8) [ {}, {} .... ]
    if (loading) { return <Loading/> }
    if (cocktails.length < 1) {
        return (
            <h2 className='section-title'>
                No cocktails matched your search criteria </h2>
        )
    }
    return (
        <section className='section'>
            <h2 className='section-title'>cocktails</h2>
            <div className='cocktails-center'>
                {cocktails.map((item) => {
                    return <Cocktail key={item.id} {...item} />
                ))}
            </div>
        </section>
    )
}

```

— Loading.js : 是一个 loading 的动画效果 :

```
import React from 'react'
const Loading = () => {
  return (
    <div className="loader">
    </div>
  )
}
export default Loading
```



— NavBar.js

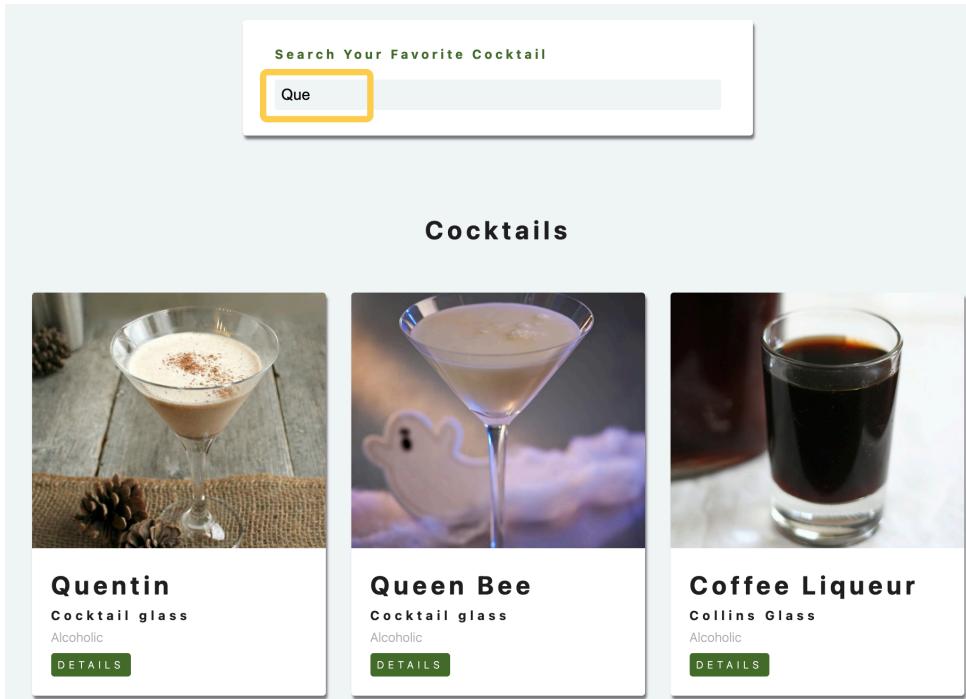


顶部导航栏

```
import React from 'react'
import { Link } from 'react-router-dom'
import logo from '../logo.svg'
export default function Navbar() {
  return (
    <nav className='navbar'>
      <div className='nav-center'>
        <Link to='/'>
          <img src={logo} alt='cocktail db logo' className='logo' />
        </Link>
        <ul className='nav-links'>
          <li>
            <Link to='/'>home</Link>
          </li>
          <li>
            <Link to='/about'>about</Link>
          </li>
        </ul>
      </div>
    </nav>
  )
}
```

```
</nav>
)
}
```

— SearchForm.js



每次 input `onChange` 都会触发 fetch 重新从接口获取数据 flush 整个 cocktails List .

```
import React from 'react'
import { useGlobalContext } from '../context'

export default function SearchForm() {
  const { setSearchTerm } = useGlobalContext()
  const searchValue = React.useRef('') // 点击 label 即可获取焦点

  React.useEffect(() => {
    searchValue.current.focus()
  }, [])

  function searchCocktail() {
    setSearchTerm(searchValue.current.value)
  }
  function handleSubmit(e) {
    e.preventDefault()
  }
  return (
    <section className='section search'>
      <form className='search-form' onSubmit={handleSubmit}>
        <div className='form-control'>
```

```

<label htmlFor='name'>search your favorite cocktail</label>
<input
  type='text'
  id='name' // id 和 htmlFor 链接起来
  ref={searchValue}
  onChange={searchCocktail} // 改变组件树的状态

  // context.js : const [searchTerm, setSearchTerm] = useState('dd')

  />
</div>
</form>
</section>
)
}

```

pages

— Home.js

The screenshot shows a web application interface for 'TheCocktailDB'. At the top, there is a navigation bar with links for 'Home' and 'About'. Below the navigation bar is a search bar labeled 'Search Your Favorite Cocktail' with a placeholder ' '.

The main content area is titled 'Cocktails' and displays three cocktail cards:

- Addison**: Martini Glass, Alcoholic. Image shows a dark liquid being poured into a glass.
- Gin Toddy**: Old-fashioned glass, Alcoholic. Image shows a clear drink with a lemon slice and a straw.
- Rum Toddy**: Old-fashioned glass, Alcoholic. Image shows a warm, yellowish drink with a lemon slice.

Each cocktail card has a 'DETAILS' button at the bottom.

主要是 搜索栏 和 下方 cocktail list 展示 . Navbar 不在这里 , Navbar 在 App.js 定义 .

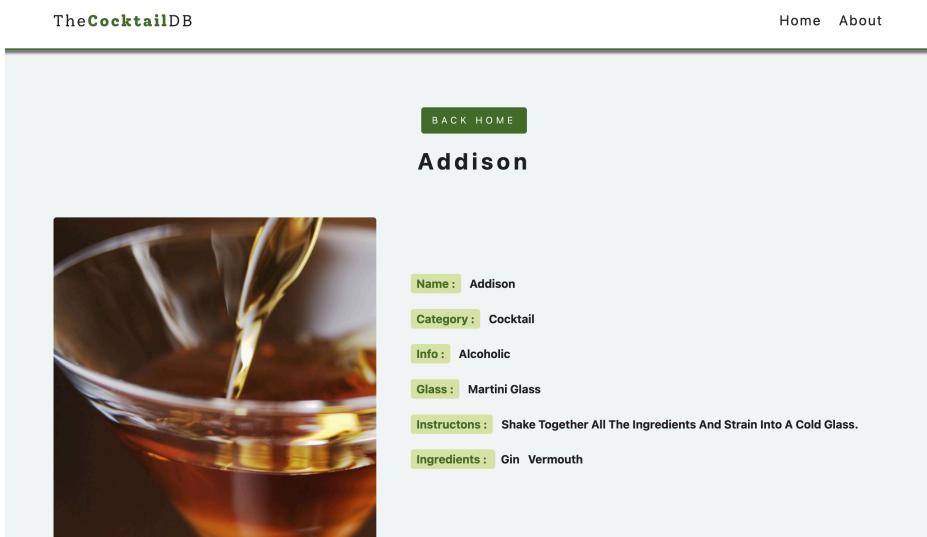
```

import React from 'react'
import CocktailList from '../components/CocktailList'
import SearchForm from '../components/SearchForm'
export default function Home() {
  return (
    <main>
      <SearchForm />
      <CocktailList />
    </main>
  )
}

```

— SingleCocktail.js

点击主页的 Details 后, 显示的鸡尾酒详情页 :



```

import React from 'react'
import Loading from '../components>Loading'
import { useParams, Link } from 'react-router-dom'

// cocktail Details
export default function SingleCocktail() {
  const { id } = useParams()
  const [loading, setLoading] = React.useState(false)
  const [cocktail, setCocktail] = React.useState(null)

  React.useEffect(() => {
    setLoading(true)
    async function getcocktail() {
      try {
        const response = await fetch(
          `https://www.thecocktaildb.com/api/json/v1/1/lookup.php?i=${id}`
        )
      }
    }
  }, [id])
}


```

```
        console.log("id    ", id)
        const data = await response.json()
        if (data.drinks) {
            const {
                strDrink: name,
                strDrinkThumb: image,
                strAlcoholic: info,
                strCategory: category,
                strGlass: glass,
                strInstructions: instructions,
                strIngredient1,
                strIngredient2,
                strIngredient3,
                strIngredient4,
                strIngredient5,
            } = data.drinks[0] // 都是接口里的 keys
            const ingredients = [ // 组成成分
                strIngredient1,
                strIngredient2,
                strIngredient3,
                strIngredient4,
                strIngredient5,
            ]
            const newCocktail = { // 显示在页面的详情信息
                name,
                image,
                info,
                category,
                glass,
                instructions,
                ingredients,
            }
            setCocktail(newCocktail)
        } else {
            setCocktail(null)
        }
    } catch (error) {
        console.log(error)
    }
    setLoading(false)
}
getCocktail()
}, [id]) // id 变化时, 说明用户点击了其他 鸡尾酒, 所以要 fetch 当前 cocktail 的信息

if (loading) {
    return <Loading/>
}
if (!cocktail) {
    return <h2 className='section-title'>no cocktail to display</h2>
```

```

} else {
  const {
    name,
    image,
    category,
    info,
    glass,
    instructions,
    ingredients,
  } = cocktail
  return (
    <section className='section cocktail-section'>
      <Link to='/' className='btn btn-primary'>
        back home
      </Link>
      <h2 className='section-title'>{name}</h2>
      <div className='drink'> /* cocktail 详情页 */
        <img src={image} alt={name}></img>
        <div className='drink-info'>
          <p> <span className='drink-data'>name :</span> {name} </p>
          <p> <span className='drink-data'>category :</span> {category} </p>
          <p> <span className='drink-data'>info :</span> {info} </p>
          <p> <span className='drink-data'>glass :</span> {glass} </p>
          <p> <span className='drink-data'>instructions :</span> {instructions} </p>
          <p>
            <span className='drink-data'>ingredients : </span>
            {ingredients.map((item, index) => {
              return item ? <span key={index}> {item}</span> : null
            ))}
          </p>
        </div>
      </div>
    </section>
  )
}
}

```

— Error.js

 localhost:3002/cocktail/不存在的 id

TheCocktailDB

No Cocktail To Display

```

import React from "react";
import { Link } from "react-router-dom";
export default function Error() {
  return (
    <section className="error-page section">
      <div className="error-container">
        <h1>oops! it's a 404 Error Page !</h1>
        <Link to="/" className="btn btn-primary">
          back home
        </Link>
      </div>
    </section>
  );
}

```

— About.js



```

import React from "react";

export default function About() {
  return (
    <section className="section about-section">
      <h1 className="section-title">about us</h1>
      <p>
        Lorem ipsum dolor sit amet consectetur adipisicing elit. Molestiae
        repudiandae architecto qui adipisci in officiis, aperiam sequi atque
        preferendis eos, autem maiores nisi saepe quisquam hic odio consectetur
        nobis veritatis quasi explicabo obcaecati doloremque? Placeat ratione
        hic aspernatur error blanditiis?
      </p>
    </section>
  );
}

```

16. Markdown Preview (预览)

```
# markdown preview
```

```
----
```

```
```python
```

```
def aa(1,):
 int a = 1
```
```

```
$$
```

```
ddd
```

```
$$
```

Markdown Preview

```
def aa(1, ):  
    int a = 1
```

\$\$ ddd \$\$

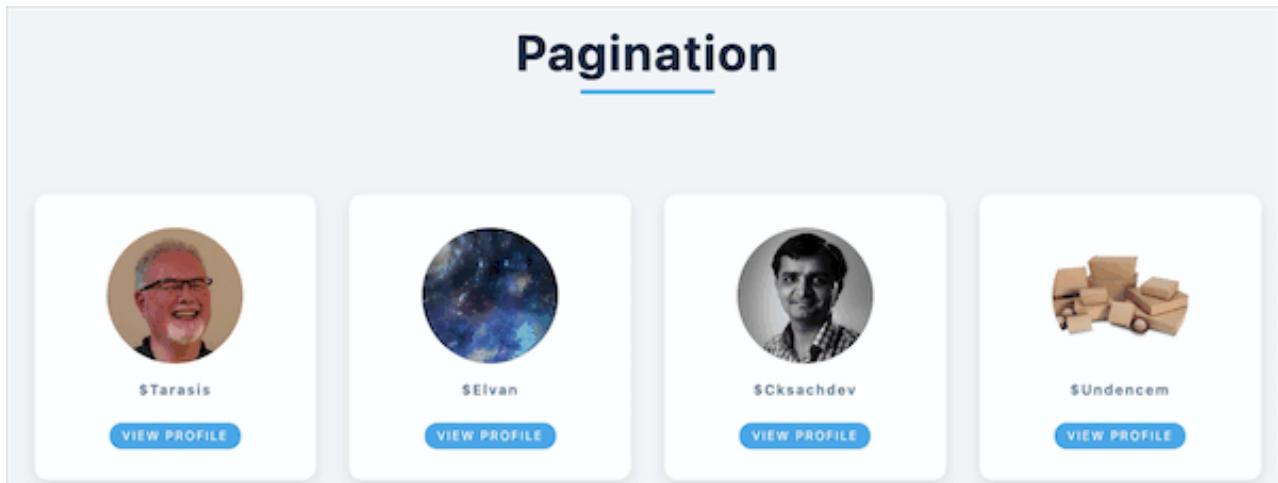
不好用, 丑拒

```
// App.js  
import React, { useState } from 'react'  
import ReactMarkdown from 'react-markdown'  
  
function App() {  
  const [markdown, setMarkdown] = useState('# markdown preview')  
  
  return (  
    <main>  
      <section className='markdown'>  
        <textarea  
          className='input'  
          value={markdown}  
          onChange={(e) => setMarkdown(e.target.value)}  
        ></textarea>  
        <article className='result'>  
          <ReactMarkdown>{markdown}</ReactMarkdown>  
        </article>  
      </section>  
    </main>  
  )  
}  
  
export default App
```

18. Pagination 分页

服务端渲染：

- 顾名思义，**服务端渲染**就是在浏览器请求页面URL的时候，**服务端**将我们需要的HTML文本组装好，并返回给浏览器，这个HTML文本被浏览器解析之后，不需要经过JavaScript脚本的执行，即可直接构建出希望的DOM树并展示到页面中。这个**服务端**组装HTML的过程，叫做**服务端渲染**。



```
// useFetch.js - 自定义 Hook
import { useState, useEffect } from 'react'
import paginate from './utils'
const url = 'https://api.github.com/users/john-smilga/followers?per_page=100'

export const useFetch = () => {
  const [loading, setLoading] = useState(true)
  const [data, setData] = useState([])
  const getProducts = async () => {
    const response = await fetch(url)
    const data = await response.json()
    setData(paginate(data)) // [Array(10), Array(10) , , , Array(10)]
    setLoading(false)
  }
  useEffect(() => {
    getProducts()
  }, [])
  return { loading, data }
}

// utils.js
const paginate = (followers) => {
  const itemsPerPage = 10      // 每页 10 人
  const numberOfPages = Math.ceil(followers.length / itemsPerPage) // 下取整
```

```

const newFollowers = Array.from({ length: numberOfPages }, (_, index) => {
  // ex: index == 3 ,  start = 3* 10 ,
  const start = index * itemsPerPage
  return followers.slice(start, start + itemsPerPage)
})
console.log(newFollowers) // (10) [Array(10), Array(10) , , , Array(10)]
return newFollowers // 返回 100 个 Users, 每页 10 个, 一共 10 页
}
export default paginate

```

```

//Follower.js - 人物卡片显示
import React from 'react'

const Follower = ({ avatar_url, html_url, login }) => {
  return (
    <article className='card'>
      <img src={avatar_url} alt={login} />
      <h4>${login}</h4>
      <a href={html_url} className='btn'>
        view profile
      </a>
    </article>
  )
}
export default Follower

```

```

// App.js
import React, { useState, useEffect } from 'react'
import { useFetch } from './useFetch' // 自定义的 fetch hook
import Follower from './Follower' // 用户显示组件

export default function App() {
  const { loading, data } = useFetch() // 解包 ! data 是 [Array(10) , , , Array(10)]
  const [page, setPage] = useState(0) // 当前页 (页角标颜色深蓝) 标识
  const [followers, setFollowers] = useState([])

  useEffect(() => {
    if (loading) return
    setFollowers(data[page])
  }, [loading, page])

  const nextPage = () => {
    setPage((oldPage) => { // oldvalue, 就是 page 状态中现在的值.
      console.log("oldPage ", oldPage)
      let nextPage = oldPage + 1
    })
  }
}
```

```
        if (nextPage > data.length - 1) {
            nextPage = 0
        }
        return nextPage
    })
}

const prevPage = () => {
    setPage((oldPage) => {
        let prevPage = oldPage - 1
        if (prevPage < 0) {
            prevPage = data.length - 1
        }
        return prevPage
    })
}

const handlePage = (index) => {
    setPage(index)
}

return (
    <main>
        <div className='section-title'>
            <h1>{loading ? 'loading...' : 'pagination'}</h1>
            <div className='underline'></div>
        </div>
        <section className='followers'>
            <div className='container'>
                {followers.map((follower) => {
                    return <Follower key={follower.id} {...follower} />
                })}
            </div>
            {!loading && (
                <div className='btn-container'>
                    <button className='prev-btn' onClick={prevPage}>
                        prev
                    </button>
                    {/* 10 个分页标记 */}
                    {data.map((_, index) => {
                        return (
                            <button
                                key={index}
                                className={`page-btn ${index === page ? 'active-btn' : null}`}
                                onClick={() => handlePage(index)} // 点击后设置为 current page
                            >
                                {index + 1}
                            </button>
                        )
                    ))}
                </div>
            )}
        </section>
    </main>
)
```

```

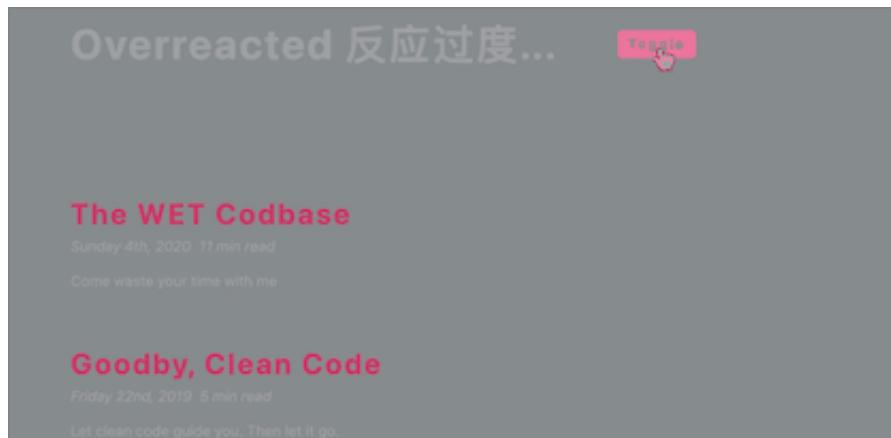
        <button className='next-btn' onClick={nextPage}>
          next
        </button>
      </div>
    )
  }
}

```

19. stock photos

没找到 `.env` 文件, 没法运行

20. Dark Mode 深色模式



初始化执行流程：

1. useState 调用 `getStorageTheme()` 赋初值 "light-theme"
2. `theme` 变化调用 useEffect , (向 Dom 插入元素) , 然后持久化到 localStorage

Toggle 执行流程：

1. `setTheme('xx-theme')` 设置 `theme`
2. 不知道为啥, 会调一下 `getStorageTheme`
3. `theme` 变化调用 useEffect ; 持久化到 `localStorage`

```

// App.js
import React, { useState, useEffect } from 'react';
import data from './data';
import Article from './Article';

const getStorageTheme = () => {
  let theme = 'light-theme';

```

```

if (localStorage.getItem('theme')) {
  theme = localStorage.getItem('theme');
}
return theme;
};

export default function App() {
  const [theme, setTheme] = useState(getStorageTheme());

  const toggleTheme = () => { // 切换浅、深色模式
    if (theme === 'light-theme') {
      setTheme('dark-theme');
    } else {
      setTheme('light-theme');
    }
  };

  useEffect(() => {
    document.documentElement.className = theme; // 向 Dom 插入元素
    localStorage.setItem('theme', theme);
  }, [theme]);
}

return (
  <main>
    <nav>
      <div className="nav-center">
        <h2> React </h2>
        <button className="btn" onClick={toggleTheme}>
          toggle
        </button>
      </div>
    </nav>
    <section className="articles">
      {data.map((item) => {
        return <Article key={item.id} {...item} />;
      })}
    </section>
  </main>
);
}

```

```

// Article.js
import React from 'react'
import moment from 'moment'
const Article = ({ title, snippet, date, length }) => {
  return (
    <article className='post'>

```

```

<h2>{title}</h2>
<div className='post-info'>
  <span>{moment(date).format('dddd Do, YYYY')}</span>
  <span>{length} min read</span>
</div>
<p>{snippet}</p>
</article>
)
}
export default Article

```

21. Movie db

没找到 `.env` 文件, 没法运行

23. 动态添加 input box

Service(s)

Remove

 Add a Service 

Remove

Output

上图可以通过 'Add a Service' Button 来动态插入 / 删除输入框

```

import { useState } from "react";
import "./App.css";

function App() {
  const [serviceList, setServiceList] = useState([{ service: "" }]);

  const handleServiceChange = (e, index) => {
    const { name, value } = e.target;
    const list = [...serviceList];
    list[index][name] = value;
    setServiceList(list);
  };

  const handleServiceRemove = (index) => {
    const list = [...serviceList];
    list.splice(index, 1);
    setServiceList(list);
  };
}

export default App;

```

```
};

const handleServiceAdd = () => {
  setServiceList([...serviceList, { service: "" }]);
};

return (
  <form className="App" autoComplete="off">
    <div className="form-field">
      <label htmlFor="service">Service(s)</label>
      {serviceList.map((singleService, index) => (
        <div key={index} className="services">
          <div className="first-division">
            <input
              name="service"
              type="text"
              id="service"
              value={singleService.service}
              onChange={(e) => handleServiceChange(e, index)}
              required
            />
            {serviceList.length - 1 === index && serviceList.length < 4 && (
              <button
                type="button"
                onClick={handleServiceAdd}
                className="add-btn"
              >
                <span>Add a Service</span>
              </button>
            )}
          </div>
          <div className="second-division">
            {serviceList.length !== 1 && (
              <button
                type="button"
                onClick={() => handleServiceRemove(index)}
                className="remove-btn"
              >
                <span>Remove</span>
              </button>
            )}
          </div>
        </div>
      ))}
    </div>
    <div className="output">
      <h2>Output</h2>
      {serviceList &&
        serviceList.map((singleService, index) => (

```

```
<ul key={index}>
  {singleService.service && <li>{singleService.service}</li>}
</ul>
)}
</div>
</form>
);
}

export default App;
```

Tips

StrictMode

React 的 StrictMode 是一种 helper component，可以帮助您编写更好的 react 组件，完全可以使用 `<StrictMode />` 包裹 a set of components，and it'll basically:

- 验证里面的组件是否遵循一些推荐的做法，否则就在控制台中，发出警告。
- 验证不推荐使用的方法没有被使用，如果它们被使用，严格模式会在控制台中警告你。
- 通过识别 potential risks (潜在风险) 来帮助您预防一些 side effects 。

严格模式是面向开发的，因此您不必担心它会影响您的生产构建。

`<> </>`

最近看一些代码，发现她们在定义组件的时候总是喜欢用 `<></>`

Fragments 可以让你聚合一个子元素列表，并且不在 DOM 中增加额外节点。也就是说

`React.Fragment` 组件能够在不额外创建 DOM 元素的情况下，让 `render()` 方法中返回多个元素

`<></>` 是 `<React.Fragment/>` 的语法糖。

具体解释：

一个常见的模式 (common pattern) 是组件返回一个子列表。以这个示例 React 片段为例：

```
class Table extends React.Component {
  render() {
    return (
      <table>
        <tr>
          <Columns />
        </tr>
      </table>
    );
  }
}
```

如上代码，`<Columns />` 需要返回多个 `<td>` 元素才能使呈现的 HTML 有效。如果在 `<Columns />` 的 `render()` 中使用了父 `div`，则生成的 HTML 将无效：

```
// 如下，在 <Columns /> 中，Mike 和 John 有父 <div>
class Columns extends React.Component {
  render() {
    return (
      <div>
        <td>Mike</td>
        <td>John</td>
      </div>
    );
  }
}

// 父 div 将会导致渲染失效：
<table>
  <tr>
    <div> // 此处的 <div> 不合理
      <td>Hello</td>
      <td>World</td>
    </div>
  </tr>
</table>
```

Fragments solve this problem :

```
class Columns extends React.Component {
  render() {
    return (
      <React.Fragment>
        <td>Hello</td>
        <td>World</td>
      </React.Fragment>
    );
  }
}
```

which results in a correct `<Table />` output of:

```
<table>
  <tr>
    <td>Hello</td>
    <td>World</td>
  </tr>
</table>
```

这也就是为什么说 Fragments 可以让你聚合一个子元素列表，并且不在 DOM 中增加额外节点。