**REACT 10: REACT HOOK**

A – LÝ THUYẾT

I – LIFECYCLE TRONG REACT

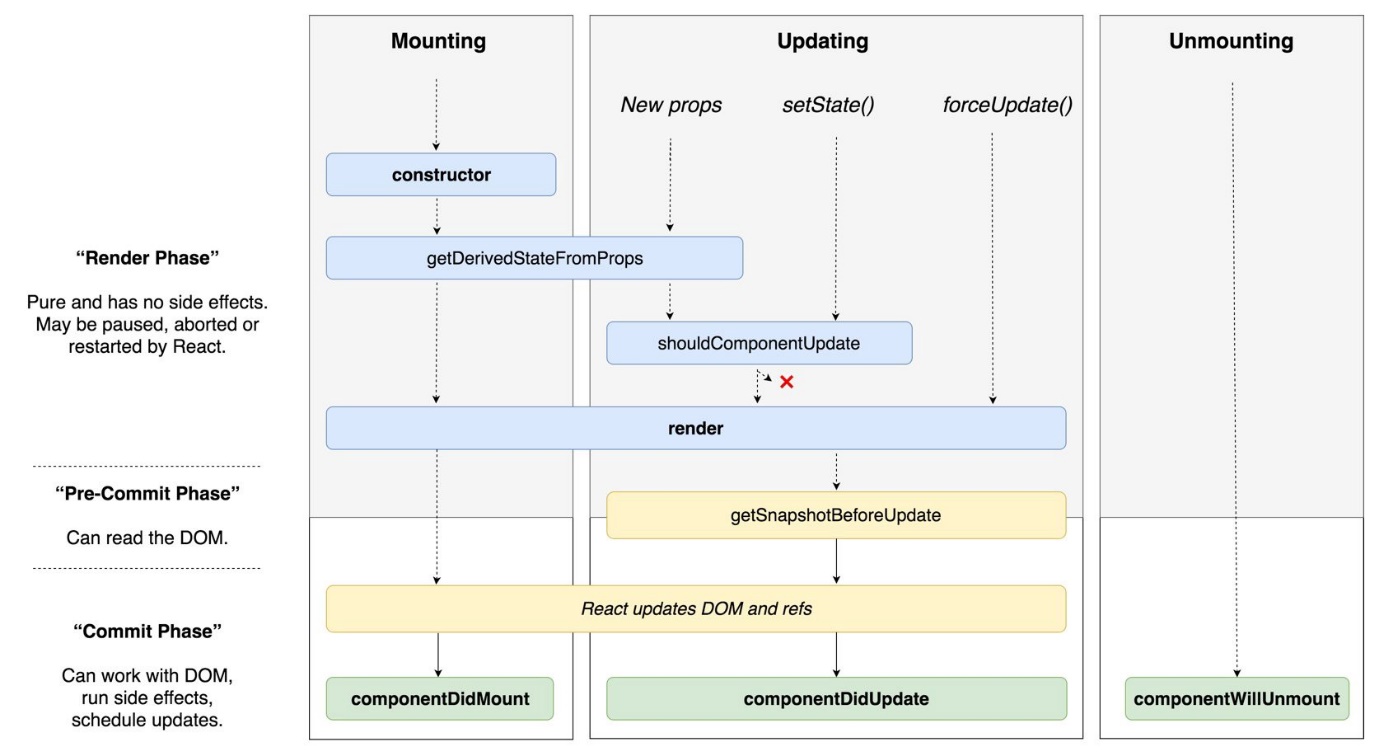
**1. Lifecycle là gì ?**

*Life là khái niệm nói về chu trình hoạt động của một Component từ lúc nó bắt đầu đến lúc nó kết thúc (thường hay được gọi là một vòng đời của Component)*

**2. Các giai đoạn chính trong một vòng đời của Lifecycle**

* Quá trình Component từ lúc tải lên đến khi trả một thành phần bất kỳ cho DOM (Mounting)
* Quá trình một Component từ lúc bắt đầu được cập nhật (có sự thay đổi giá trị của Props, State) đến khi một thành phần mới được cập nhật cho DOM (Updating)
* Quá trình cuối cùng của một vòng đời Component, đây là quá trình một Component loại bỏ một thành nào đó khỏi DOM (Unmounting)

Để hiểu rõ hơn về vòng đời của một Component (Life Cycle), chúng ta có thể xem qua ảnh mô tả rất chi tiết dưới đây



II – MỘT SỐ LIFE CYCLE HOOK PHỔ BIẾN SAU QUÁ TRÌNH RENDER

Ở mỗi trạng thái khác nhau React cung cấp cho Component những phương thức đã được xây dựng sẵn nhằm giúp Component xử lý dữ liệu một cách linh hoạt nhất, trong đó có 3 phương thức cực kỳ phổ biến được sử dụng để xử lý sau quá trình Render đó là **componentDidMount**, **componentDidUpdate** và **componentWillUnmount**

**1. componentDidMount**

Phương thức này sẽ tự động chạy và chạy ngay sau khi quá trình Render thực hiện xong của trạng thái Mounting

**2. componentDidUpdate**

Phương thức này sẽ tự động chạy và chạy ngay sau quá trình Render thực hiện xong của trạng thái Updating

**3. componentWillUnmount**

Phương thức này sẽ tự động chạy và chạy ngay sau khi một cấu trúc HTML bị loại bỏ khỏi giao diện

III – REACT HOOK

Các Life Cycle Hook trên chỉ sử dụng được đói với Component dạng Object, còn đối với Component dạng Function thì chúng ta không thể sử dụng được, do đó bắt đầu từ phiên bản 16.8 thì React đã cung cấp cho chúng ta một số React Hook để có thể làm việc với Component dạng Function, trong đó có những Hook dành riêng cho việc xử lý Life Cycle như useEffect mà chúng ta sẽ nghiên cứu trong bài này

**1. useEffect**

*useEffect là một Hook được sử dụng để thay thế cho các Life Cycle Hook trước đây như: componentDidMount, componentDidUpdate, componentWillUnMount,…*

**Code JS:**

|  |
| --- |
| import React from "react"  import ReactDom from "react-dom"  const App = ()=>{    React.useEffect(()=>{  console.log("DidMount")  })  React.useEffect(()=>{  console.log("DidUpdate")  })  React.useEffect(()=>{  console.log("WillUnMount")  })  return(  <button>Action</button>  )  }  ReactDom.render(<App />, document.getElementById("root")) |

**Trong đó:**

* **useEffect**: là một Hook, và nó sẽ tự động chạy mỗi khi Component Render xong

**2. Sử dụng useEffect như componentDidMount**

Để có thể biến userEffect Hook hoạt động như componentDidMount thì chúng ta cần phải sử dụng kỹ thuật như ví dụ dưới đây

**Cú pháp:**

|  |
| --- |
| React.useEffect(effectFunction, Array) |

**Code JS:**

|  |
| --- |
| import React from "react"  import ReactDom from "react-dom"  const App = ()=>{    const [data, setData] = React.useState(true)    React.useEffect(()=>{  console.log("DidMount")  }, [])  return(  <button onClick={()=>setData(!data)}>Action</button>  )  }  ReactDom.render(<App />, document.getElementById("root")) |

**Trong đó:**

* Trong ví dụ trên chúng ta đã sử dụng sự kiện Click để chuyển trạng thái từ Mouting sang Updating
* Đối số Array truyền vào sẽ được kiểm tra trước khi useEffect Hook tự động chạy, nếu giá trị của nó không đổi thì nó sẽ không thực thi effectFunction

**3. Sử dụng useEffect như componentDidUpdate**

Để có thể biến userEffect Hook hoạt động như componentDidUpdate thì chúng ta cần phải sử dụng kỹ thuật như ví dụ dưới đây

**Cú pháp:**

|  |
| --- |
| React.useEffect(effectFunction, Array) |

**Code JS:**

|  |
| --- |
| import react from "react";  import reactDom from "react-dom";  const App = ()=>{  const [data, updateData] = react.useState(true);  react.useEffect(()=>{  if(!data){  updateData(!data);  console.log("DidUpdate");  }  }, [data]);  return(  <button onClick={()=>updateData(!data)}>Action</button>  )  }  reactDom.render(<App />, document.getElementById("root")); |

**Trong đó:**

* Trong ví dụ trên chúng ta đã sử dụng sự kiện Click để chuyển trạng thái từ Mouting sang Updating
* Đối số Array truyền vào sẽ được kiểm tra trước khi useEffect Hook tự động chạy, nếu giá trị của thay đổi thì nó sẽ gọi tới tham số effectFunction để thực thi

**4. Sử dụng useEffect như componentWillUnMount**

Để có thể biến userEffect Hook hoạt động như componentWillUnMount thì chúng ta cần phải sử dụng kỹ thuật như ví dụ dưới đây

**Cú pháp:**

|  |
| --- |
| React.useEffect(effectFunction, Array) |

**Code JS:**

|  |
| --- |
| import React from "react"  import ReactDom from "react-dom"  const App = () => {  const [data, setData] = React.useState(true)  const header = () => {  return (  <>  <h3>Vietpro Academy</h3>  <hr />  </>  )  }  React.useEffect(() => {  if(!data){  console.log("WillUnMount")  }  })  return (  <>  {data && header()}  < button onClick = {()=> setData(!data)}> Action</button >  </>  )  }  ReactDom.render(<App />, document.getElementById("root")) |

**Trong đó:**

* Trong ví dụ trên chúng ta đã sử dụng sự kiện Click để chuyển trạng thái từ Mouting sang Unmounting
* Đối số Array truyền vào sẽ được kiểm tra trước khi useEffect Hook tự động chạy, nếu giá trị của thay đổi thì nó sẽ gọi tới tham số effectFunction để thực thi

III – MỘT SỐ KIẾN THỨC BỔ TRỢ VỀ JAVASCRIPT

**1. Hàm setTimeout()**

*Hàm setTimeout() cho phép chúng ta thực thi một hành động nào đó (thường là một hàm) sau thời gian cho trước*

**Code:**

|  |
| --- |
| setTimeout(()=>{  alert("Vietpro Academy !")  }, 1000) |

**Trong đó:**

* Tham số thứ nhất là một hàm
* Tham số thứ hai là thời gian trễ để thực thi hàm (thám số thứ nhất) và được tính theo mili giây

**2. Hàm splice()**

*Hàm splice() sử dụng để loại bỏ phần từ ra khỏ mảng*

**Code:**

|  |
| --- |
| var arr = ["HTML", "CSS", "Bootstrap", "Javascript"]  arr.splice(2, 1)  // ["HTML", "CSS", "Javascript"] |

**Trong đó:**

* Tham số thứ nhất là vị trí của phần tử muốn xóa
* Tham số thứ hai là số lượng phần tử muốn xóa tính từ tham số thứ nhất

**3. Spread Operator**

*Spread Operator là cú pháp cho phép chúng ta tải lại toàn bộ các phần tử của một mảng bất kỳ vào một mảng khác*

**Code:**

|  |
| --- |
| var arr = ["HTML", "CSS"]  var newArr = [...arr, "Bootstrap", "Javascript"]  // ["HTML", "CSS", "Bootstrap", "Javascript"] |

**Trong đó:**

* …arr: cho phép tải lại toàn bộ tất cả các phần tử của mảng arr vào mảng hiện tại

B – THỰC HÀNH

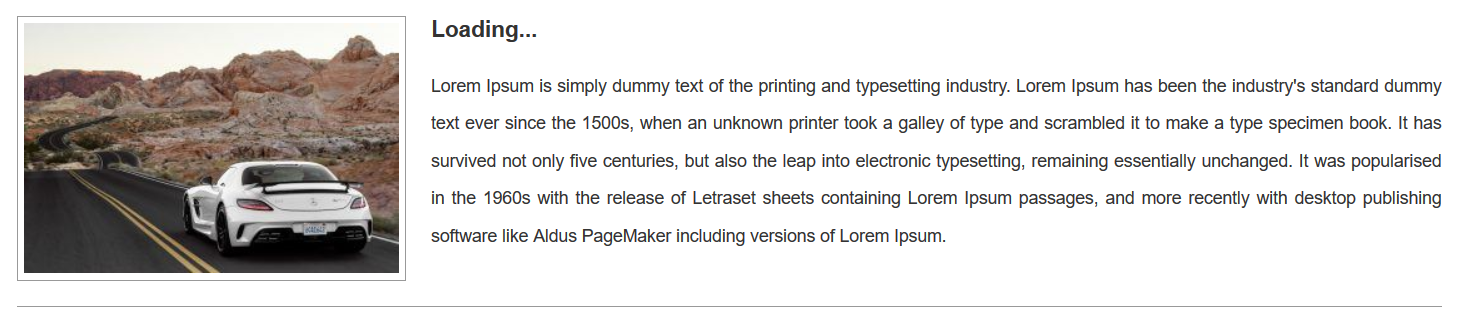
I – BÀI TẬP THỰC HÀNH

**Bài 1:**

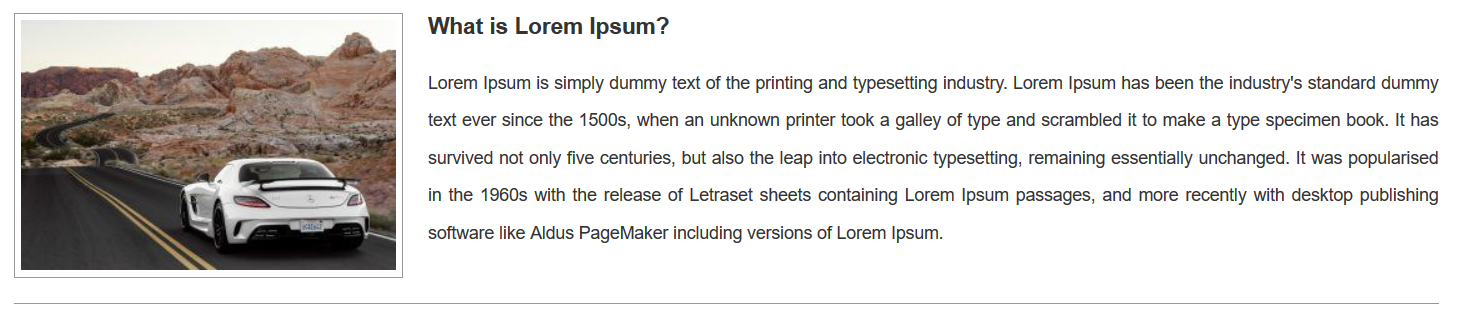
Sử dụng lại mã nguồn của bài Event & CSS để bổ sung thêm các tính năng sau

Khi mới tải giao diện, tiêu đề sẽ hiển thị chữ Loading…, sau đó 2s thì mới hiển thị tiêu đề đầy đủ

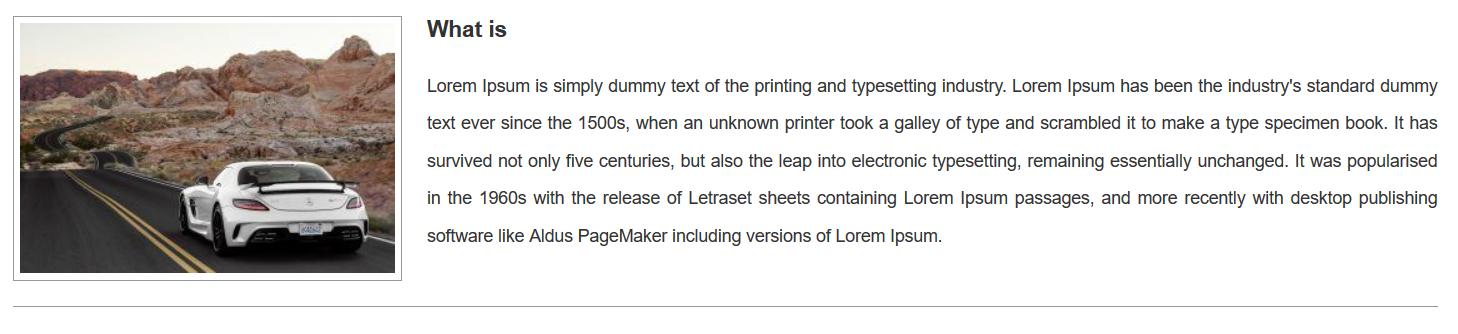
Giao diện mới mở

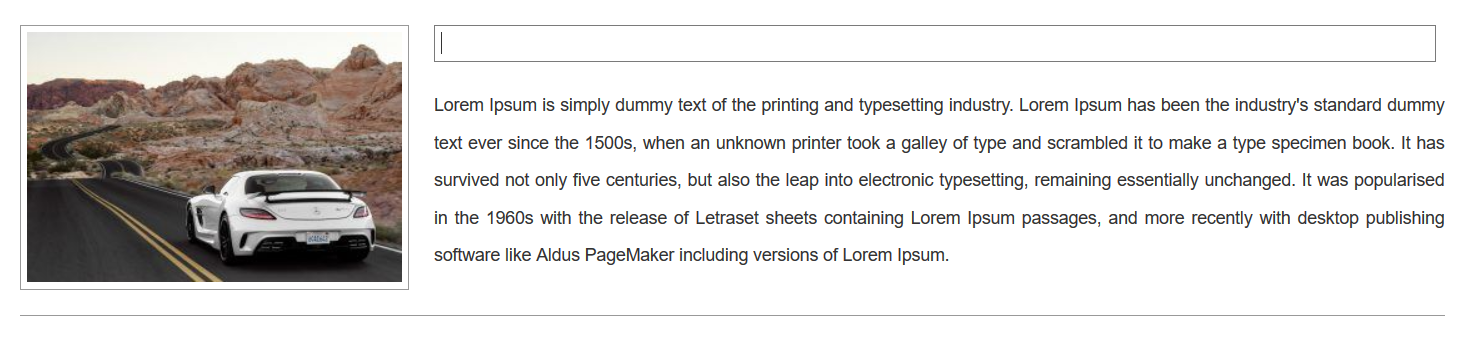


Giao diện sau 2s



Cho phép người dùng sửa trực tiếp nội dung tiêu đề thông qua Input Form (fix luôn lỗi khi nháy đúp vào tiêu đề nhưng không Focus Form ngay) và sau đó xác nhận tiêu đề mới thông qua việc bấm chuột ra bên ngoài hoặc sử dụng phím Enter

 Cuối cùng hoàn thiện ứng dụng khi kiểm tra nếu người dùng xóa hết tiêu đề thì sẽ tự động tải lại tiêu đề mặc định



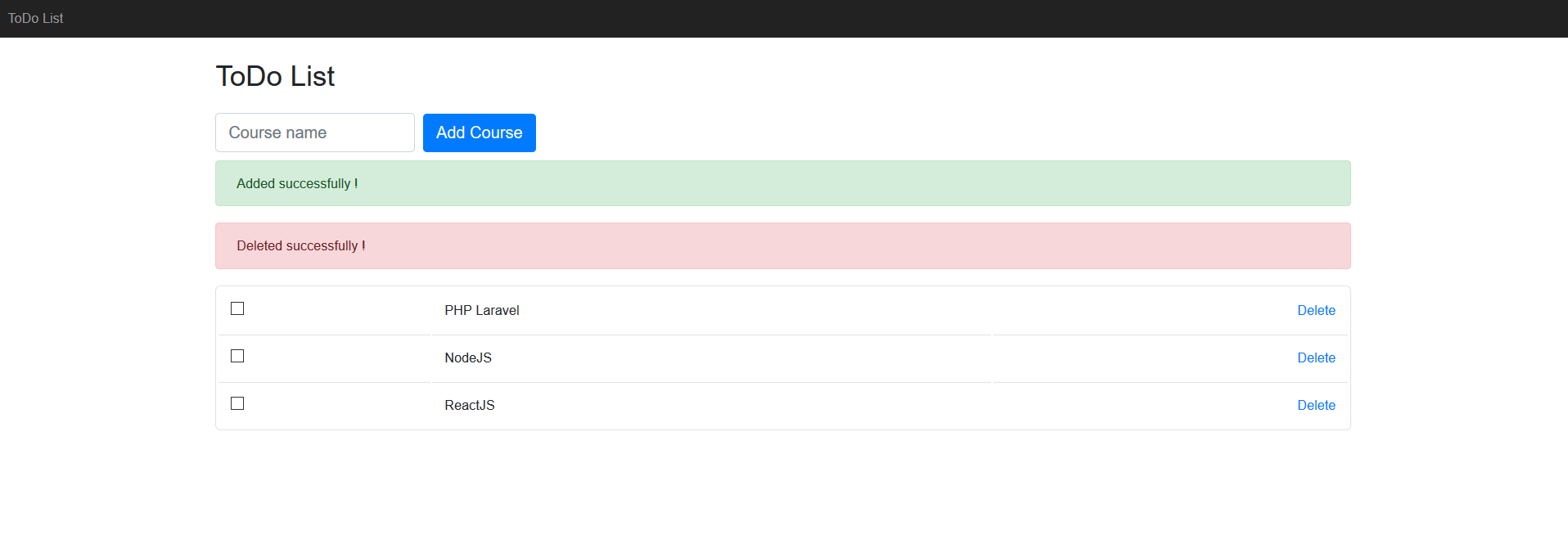
II – BÀI TẬP VỀ NHÀ

**Bài 1:**

Hoàn thiện phần nội dung mô tả tương tự như phần tiêu đề cho bài tập thực hành 1 ở trên lớp

**Bài 2:**

Sử dụng giao đã được cung cấp sẵn như hình dưới để xây dựng ứng dụng ToDo List như file demo ToDo-List.gif



**Bài 3:**

Nâng cấp ứng dụng ToDo List bằng cách tách Component

**Bài 4:**

Tiếp tục nâng cấp ứng dụng ToDo List bằng cách mỗi khi người dùng thêm mới một khóa học thành công thì phải xóa tên khóa học trong ô Textbox, ngoài ra người dùng muốn thêm mới có thể không cần phải bấm vào nút Button mà có thể sử dụng phím Enter