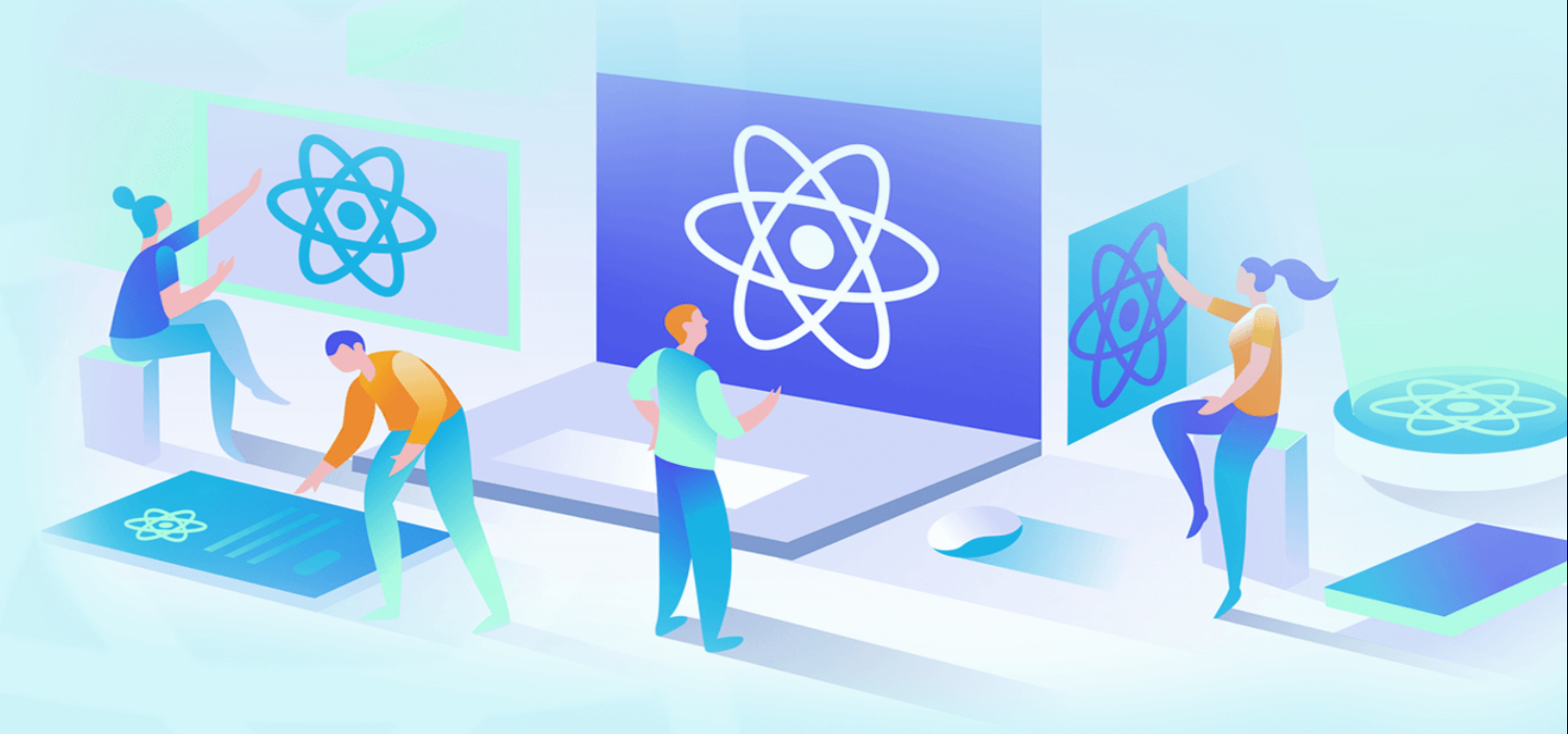


LẬP TRÌNH REACT THẬT ĐƠN GIẢN

Từng bước tạo ứng dụng từ A-Z

By VNTALKING VTA

MỤC LỤC

LỜI NÓI ĐẦU

Nói đến front-end là không thể không nhắc tới ReactJS. Một thư viện Javascript được chống lưng bởi ông lớn Facebook. Lướt qua các trang tuyển dụng lớn, bạn sẽ thấy nhu cầu tuyển người biết ReactJS rất lớn.

Bạn biết CMS nổi tiếng Wordpress không? Từ Wordpress 5.0, hẳn bạn đã nghe tới trình soạn thảo Gutenberg được tích hợp sẵn, nó cũng được xây dựng bằng React đó.

Thêm một ví dụ nữa, đó là Gatsby, một trình tạo trang web tĩnh nổi tiếng, một xu hướng mà nhiều nhà phát triển bắt đầu chuyển sang.

Và rất nhiều ứng dụng nổi tiếng khác cũng đang sử dụng React như: chotot, Shopee, Lazada, MyTV...

Điều này cho thấy rằng, việc lựa chọn ReactJS là một lựa chọn thông minh trong thời điểm hiện tại.

Với mục đích xây dựng nền tảng khi bắt đầu học ReactJS, mình cho ra đời cuốn sách này.

Trong cuốn sách này, bạn sẽ không chỉ học các nguyên tắc cơ bản của ReactJS, mà bạn còn có thể ứng dụng chúng để tự xây dựng các ứng dụng ReactJS hiện đại và triển khai miễn phí lên Internet.

Ngoài ra, trong cuốn sách, mình sẽ giới thiệu những kỹ thuật hỗ trợ bạn xây dựng ứng dụng ReactJS dễ dàng hơn. Ví dụ như quản lý mã nguồn với GIT, lựa chọn và cài đặt Code Editor phù hợp với ReactJS.v.v...

Nếu bạn là người thích sự chi tiết và tỉ mỉ thì cuốn sách này đích thị dành cho bạn.

Hãy tiếp tục đọc và nghiền ngẫm, bạn sẽ cảm thấy yêu thích cuốn sách này.

Mình đảm bảo!

NỘI DUNG CUỐN SÁCH

Chào mừng bạn đến với cuốn sách: "**Lập trình React thật đơn giản**". Mình tin chắc rằng, bạn được nghe không nhiều thì ít về ngôn ngữ lập trình web như HTML, Javascript, CSS.

Về cơ bản, để tạo một ứng dụng web (ám chỉ front-end), bạn chỉ cần biết 3 thứ: HTML, Javascript, CSS. Tất cả những công nghệ, thư viện, frameworks... dù đao to búa lớn thì cũng chỉ xoay quanh 3 ngôn ngữ đó mà thôi.

Thế giới web hiện đại ngày nay, chúng ta không chỉ đơn thuần xây dựng các website cung cấp thông tin một chiều mà còn có các ứng dụng web, khi mà người dùng có thể tương tác, thay đổi nội dung, thay đổi giao diện mà không phải tải lại trang (chính là các ứng dụng web dạng Single Single Page Applications - gọi tắt là SPA).

Và đây là mảnh đất để các thư viện như ReactJS tung hoành.

Trong cuốn sách này, bạn sẽ được học và thực hành:

- Cách cài đặt và bắt đầu làm việc với ReactJS (2 cách).
- Học các kiến thức nền tảng của React:
    - React component
    - Props và State
    - Class Component và Functional component
    - Truyền dữ liệu giữa các components.
    - Sử dụng Style trong ứng dụng React.
    - Xử lý sự kiện trong ứng dụng React.
    - React Hooks và cách sử dụng.
    - Redux - cách quản lý state.
- Cách lấy dữ liệu từ API và hiển thị lên ứng dụng React.
- Triển khai ứng dụng React lên Internet.
- Thực hành từng bước xây dựng ứng dụng Todos.
- Bài tập: Xây dựng ứng dụng Meme Generator.

Để đảm bảo các bạn tập trung và hiểu rõ những khái niệm, kiến thức nền tảng của ReactJS, mình sẽ không sử dụng bất kỳ thư viện 3rd nào khi xây dựng ứng dụng, ngoại trừ axios và redux.

AI NÊN ĐỌC CUỐN SÁCH NÀY?

Cuốn sách này phù hợp với tất cả những ai quan tâm đến việc phát triển ứng dụng ReactJS. Với những bạn đang định hướng sự nghiệp trở thành front-end developer thì cuốn sách này dành cho bạn.

Đây là cuốn sách "**No Experience Require**", tức là không yêu cầu người có kinh nghiệm ReactJS. Tất cả sẽ được mình hướng dẫn từ con số 0.

Yêu cầu trình độ

Do ReactJS là thư viện Javascript được xây dựng để tạo các ứng dụng web, nên sẽ tốt hơn nếu bạn đã có:

- Kiến thức cơ bản về HTML và CSS.
- Javascript cơ bản (Object, Arrays, điều kiện.v.v...)
- Đã biết tới Javascript ES6 (arrow function.v.v...)

Nếu bạn vẫn còn đang bỡ ngỡ với Javascript, cũng không sao! Đọc xong cuốn sách này bạn hiểu Javascript hơn.

Cách học đúng cách

Cuốn sách này mình chia nhỏ nội dung thành nhiều phần, mỗi phần sẽ giới thiệu một chủ đề riêng biệt, kèm thực hành. Mục đích là để bạn có thể chủ động lịch học, không bị dồn nén quá nhiều.

Với mỗi phần lý thuyết, mình đều có ví dụ minh họa và code luôn vào dự án. Vì vậy, cách học tốt nhất vẫn là vừa học vừa thực hành. Bạn nên **tự mình viết lại từng dòng code** và kiểm tra nó trên trình duyệt. Đừng copy cả đoạn code trong sách, điều này sẽ hạn chế khả năng viết code của bạn, cũng như làm bạn nhiều khi không hiểu vì sao code bị lỗi.

"Nhớ nhé, đọc đến đâu, tự viết code đến đó, tự build và kiểm tra đoạn code đó chạy đúng không"

Ngoài ra, cuốn sách này mình biên soạn theo mô hình: phần sau được xây dựng từ phần trước. Do vậy, bạn đừng đọc lướt mà bỏ sót đoạn nào nhé.

Tất cả mã nguồn trong cuốn sách sẽ được mình up lên Github: https://github.com/vntalking/ebook-reactjs-that-don-gian

Trong quá trình bạn đọc sách, nếu code của bạn không chạy hoặc chạy không đúng ý muốn, bạn có thể kiểm tra và so sánh với mã nguồn của mình trên Github. Nếu vẫn không được thì đừng ngần ngại đặt câu hỏi trên Group: Hỏi đáp lập trình - VNTALKING

Liên hệ tác giả

Nếu có bất kỳ vấn đề gì trong quá trình học, code bị lỗi hoặc không hiểu, các bạn có thể liên hệ với mình qua một trong những hình thức dưới đây:

- Website: https://vntalking.com
- Fanpage: https://facebook.com/vntalking
- Group: https://www.facebook.com/groups/hoidaplaptrinh.vntalking
- Email: support@vntalking.com
- Github: https://github.com/vntalking/ebook-reactjs-that-don-gian

GIỚI THIỆU

ReactJS (đôi lúc cũng có thể gọi là React.JS, React - cũng đều nó cả) là một thư viện Javascript giúp bạn nhanh chóng xây dựng giao diện ứng dụng (UI). React có thể xây dựng website hoàn toàn sử dụng Javascript (để thao tác với HTML), được tối ưu hiệu năng bởi kỹ thuật **VirtualDOM** (mình sẽ giải thích chi tiết ở phần dưới cuốn sách).

React được Facebook giới thiệu vào năm 2011, họ quảng bá đây là thư viện cho phép developer tạo các dự án ứng dụng web lớn, có giao diện UI phức tạp từ một đoạn mã nguồn nhỏ và độc lập.

Một số diễn đàn, React được coi là Framework vì khả năng và behaviors (hành vi) của nó. Nhưng về mặt kỹ thuật, nó chỉ là một thư viện (library). Không giống như Angular hay Vue được định nghĩa là một framework. Với React, bạn sẽ cần phải kết hợp sử dụng nhiều thư viện hơn để tạo thành một giải pháp hoàn chỉnh. Điều này khiến **React chỉ tập trung vào rendering** (kết xuất - nghĩa là xuất ra một thứ gì đó cho người dùng nhìn thấy) và đảm bảo nó được đồng bộ hóa với State của ứng dụng. Đó là tất cả mà React thực hiện.

Đọc mấy khái niệm này có vẻ cao siêu khó hiểu nhỉ?

Đừng hoảng nhé!

Bạn sẽ hiểu rõ hơn khi bắt tay vào thực hành xây dựng ứng dụng React ở phần tiếp theo cuốn sach - cứ bình tĩnh nhé.

Ưu điểm của React

Để tìm một thư viện/framework front-end, bạn chỉ cần vào Google gõ nhẹ một cái là ra hàng tá luôn, có thể kể những cái tên đình đám như: Angular, Vue, React, Jquery, Emberjs.v.v...

Do đó, khi phải sử dụng React, chắc hẳn bạn sẽ luôn băn khoăn: *Thư viện React này có ưu điểm gì mà mình phải dùng?*

Ở đây, mình sẽ chia thành từng mục và đưa ra lý do ngắn gọn tại sao sử dụng React là cần thiết nếu bạn đang cân nhắc đến việc xây dựng ứng dụng web hiện đại.

Ok bắt đầu nhé.

1. Component độc lập - có thể tái sử dụng

Trong React, giao diện được xây dựng từ việc kết hợp các components. Bạn có thể hiểu đơn giản component là một hàm, nó nhận giá trị bạn truyền vào và trả về một số đầu ra.

Và cũng giống như function, component có thể tái sử dụng ở bất kỳ đâu. Vì vậy, chúng ta có thể tái sử dụng lại, hợp nhất các component để tạo thành giao diện người dùng phức tạp hơn.

Trong cuốn sách này, bạn sẽ biết cách làm thế nào xây dựng giao diện ứng dụng phức tạp, nhiều tầng, nhiều lớp thông qua component.

2. React làm cho front-end javascript dễ sử dụng hơn, nhanh hơn bằng cách sử dụng Virtual DOM

Khi bạn làm việc với vanilla Javascript, bạn sẽ phải tự làm mọi nhiệm vụ khi thao tác với DOM. Nhưng với React thì khác.

Với React, bạn chỉ cần thay đổi state của UI (bạn sẽ biết khái niệm này ở phần 4 của cuốn sách), và React sẽ tự động cập nhật DOM để phù hợp với state đó. Kỹ thuật được React sử dụng đó là Virtual DOM. Điều này cho phép React chỉ cập nhật những cái cần thiết (kiểu như một phần trang web) thay vì phải tải lại toàn bộ trang.

Việc sử dụng Javascript để tạo ra mã HTML cho phép React có một cây đối tượng HTML ảo — VirtualDOM. Khi bạn tải trang web sử dụng React, một VirtualDOM được tạo ra và lưu trong bộ nhớ.

Mỗi khi state thay đổi, chúng ta sẽ cần phải có một cây đối tượng HTML mới để hiển thị lên trình duyệt. Thay vì tạo lại toàn bộ cây, React sử dụng một thuật toán thông minh để chỉ tạo lại các thành phần khác biệt giữa cây mới và cây cũ. Bởi vì cả hai cây HTML cũ và mới đều được lưu trong bộ nhớ, quá trình xử lý này diễn ra siêu nhanh.

3. Dễ dàng làm việc nhóm

Nếu bạn đang làm việc theo nhóm, thì React là thư viện tuyệt vời. Bởi vì, việc chia các task, chia màn hình ứng dụng sẽ rất đơn giản, mỗi người sẽ được chỉ định làm một thành phần. Các thành phần này độc lập với nhau, việc ghép nối các thành phần được thực hiện dễ dàng.

4. React được đảm bảo bởi Facebook

Khi bạn lựa chọn một thư viện/framework, việc đầu tiên là phải xem nó được bảo chứng bởi ai? Bởi vì một thư viện/framework không được bảo chứng bởi một tổ chức, công ty uy tín, nó có thể sẽ không được phát triển lâu dài, không được mở rộng hoặc maintain khi thư viện/framework phát sinh bugs...

Với React, bạn hoàn toàn yên tâm, vì nó được Facebook chống lưng, theo đó là một cộng đồng React tuyệt vời. Ngoài ra, tài liệu hướng dẫn chính chủ cũng rất chi tiết, đầy đủ.

Nói chung, về nguồn gốc, cộng đồng, tài liệu, bạn hoàn toàn có yên tâm gửi gắm dự án của mình cho React.

5. Nhu cầu tuyển dụng cao

Như mình đã đề cập ở trên, React là một kỹ năng web có nhu cầu tuyển dụng cao ở thời điểm hiện tại.

Hầu như nhà tuyển dụng nào khi cần tuyển một front-end developer đều yêu cầu phải biết React. Vì vậy, việc thành thạo React là một điểm sáng trong CV của bạn, cơ hội việc làm sẽ rộng mở hơn rất nhiều.

Ngoài ra, React có ít API hơn so với các thư viện/framework khác, do vậy việc học React cũng dễ hơn, đặc biệt nếu bạn đã thành thạo Javascript.

Trên đây là những ưu điểm của React mà mình tổng hợp và hệ thống lại. Giờ là lúc chúng ta cùng nhau chinh phục React thôi.

Let's go!

TỔNG QUAN REACT

Khi bạn bắt đầu tìm hiểu bất kỳ một nền tảng, thư viện hay framework nào đó, việc đầu tiên là bạn cần hiểu tư duy/triết lý của người viết ra nó. Hay nói một cách khác, bạn cần cùng hướng nhìn với tác giả của thư viện đó, khi mà cùng "hệ quy chiếu" thì việc học sẽ dễ dàng hơn.

Với React, có thứ mà bạn cần hiểu, đó là: Virtual DOM.

Virtual DOM

Hiểu được cách thức hoạt động của DOM, bạn sẽ nhanh chóng nắm bắt được concept đằng sau Virtual DOM.

Nếu bạn là một Javascript developer, đã từng xây dựng một trang web thì hẳn bạn đã tương tác với DOM. Tuy nhiên, mình vẫn muốn nhắc lại cách thức hoạt động của nó để chúng ta cùng có một hướng nhìn.

DOM là gì?

DOM (viết tắt của Document Object Model) là một giao diện lập trình ứng dụng, cho phép Javascript hoặc các một loại script khác đọc và thao tác nội dung của document (trong trường hợp này là HTML).

Bất cứ khi nào một HTML document được tải xuống trình duyệt dưới dạng một trang web, một DOM sẽ được tạo cho trang đó.

Theo cách này, Javascript có thể thao tác với DOM như thêm, sửa, xóa nội dung... hoặc thực hiện một hành động nào đó như ẩn/hiện một view.

Vấn đề của DOM là gì?

Một số developers tin rằng, việc thao túng DOM là không hiệu quả, có thể gây chậm ứng dụng. Đặc biệt là khi bạn thường xuyên phải cập nhật DOM.

Vấn đề phải cập nhật DOM sẽ còn phức tạp hơn nhiều với các ứng dụng dạng Single Page Applications (SPA). Khi mà DOM chỉ được tải về trình duyệt một lần, sau đó Javascript sẽ phải cập nhật DOM liên tục mỗi khi người dùng thao tác với ứng dụng.

Vấn đề ở đây là không phải do DOM chậm (bởi vì DOM chỉ là một object). Ứng dụng bị chậm là do trình duyệt phải xử lý khi DOM thay đổi. Mỗi khi DOM thay đổi, trình duyệt sẽ phải tính toán lại CSS, chạy bố cục và render lại trang web.

Như một logic dễ hiểu, để tăng tốc ứng dụng, chúng ta cần tìm cách giảm thiểu thời gian khi trình duyệt render lại trang. Đó là lý do mà nhà phát triển React nghĩ tới Virtual DOM.

Cơ chế hoạt động của Virtual DOM

Nói chung, Virtual DOM có cái gì đó giống với cơ chế Cache vậy. Mặc dù không hoàn toàn. Tức là, React sẽ làm việc trên Virtual DOM (được lưu trong bộ nhớ), nó đảm bảo DOM thật chỉ nhận nhưng thứ cần thiết mà thôi.

Cơ chế hoạt động của Virtual DOM cơ bản như sau:

- Bất cứ khi nào có yếu tố mới được thêm vào UI, một Virtual DOM sẽ được tạo.
- Khi state của bất cứ yếu tố nào thay đổi, React sẽ cập nhật Virtual DOM trong khi vẫn giữ phiên bản Virtual DOM trước để so sánh và tìm ra đối tượng Virtual DOM thay đổi. Khi tìm được sự thay đổi, React chỉ cập nhật đối tượng đó trên DOM thật.

Không giống như khi làm việc với vanilla JavaScript, việc cập nhật sẽ được React làm tự động. Tất cả công việc của bạn là thay đổi state của UI, phần còn lại cứ để React lo.

Bất cứ khi nào state thay đổi, React sẽ lập tức có hành động để cập nhật DOM, đó là lý do người ta gọi React là thư viện reactive.

CÀI ĐẶT REACT

Giống như mọi công nghệ web khác, bạn cần phải thiết lập môi trường phát triển trước khi có thể làm việc với React.

Yêu cầu môi trường phát triển:

- **Ubuntu v18.04**: Bạn có thể viết ứng dụng React trên Window hoặc Mac OS cũng được. Nhưng trong cuốn sách này mình sử dụng Ubuntu 18.04, một phần do mình quen dùng OS này rồi, một phần là khi đi làm ở công ty, các bạn cũng chủ yếu dùng Ubuntu để lập trình.
- **Node.js v12.18.3**: Các bạn cứ cài bản Node.js LTS mới nhất nhé. Cách cài đặt thì mình không trình bày trong cuốn sách này vì nó phổ biến quá rồi. Nếu cần có thể tham khảo lại trong bài viết này của mình: Hướng dẫn cài đặt Node.js chi tiết
- **NPM v6.14.6**: Là công cụ hỗ trợ quản lý và cài đặt các dependencies.
- **create-react-app**: Là công cụ giúp tạo dự án React mới nhanh chóng (mình sẽ trình bày về công cụ này ở phần dưới cuốn sách).

Có 2 cách để làm việc với React phổ biến nhất:

- Nhúng ngay thư viện React trong HTML.
- Sử dụng công cụ *create-react-app* (được tác giả khuyến khích sử dụng).

Mình sẽ hướng dẫn chi tiết cả hai cách sử dụng React này, bạn thấy cách nào phù hợp thì chọn lấy một cái cho dự án của mình nhé.

Đây chỉ là bản demo - phần này còn nữa
Đặt sách **tại đây** để đọc bản đầy đủ nhé

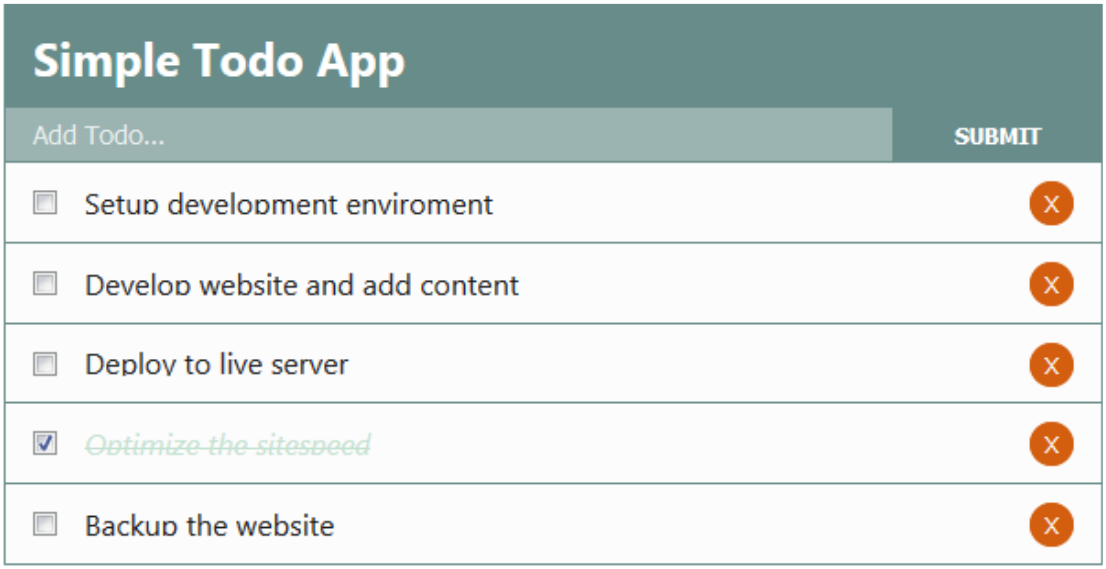
XÂY DỰNG ỨNG DỤNG TODOS

Sau khi hoàn thành xong phần 1 và 2 của cuốn sách này, bạn đã hiểu được khái niệm cơ bản của React, triết lý thiết kế của React cũng như đã chuẩn xong môi trường để phát triển ứng dụng React.

Đến phần này, chúng ta sẽ cùng nhau vừa tìm hiểu lý thuyết của React, vừa áp dụng kiến thức đó để xây dựng ứng dụng Todos.

Giới thiệu ứng dụng Todos

Đây là giao diện ứng dụng Todos mà chúng ta sẽ xây dựng trong phần này.



Hình 4.1: Giao diện ứng dụng Todo.

Giống như tên gọi, ứng dụng Todo dùng để lưu lại những công việc, những task cần phải thực hiện. Mỗi Todo được hiểu là một task, một công việc cụ thể mà người dùng cần phải thực hiện.

Ứng dụng này sẽ có những tính năng cơ bản như:

- Hiển thị danh sách các Todo được lấy từ API về.
- Thêm một Todo mới - Todo được thêm vào server thông qua API.
- Xóa một Todo - Todo được xóa trên server thông qua API.
- Đánh dấu một Todo đã hoàn thành.

Đầu tiên, các bạn checkout mã nguồn dự án rỗng tại đây:
https://github.com/vntalking/ebook-reactjs-that-don-gian/tree/master/phan4/todo-app-starter

Sau khi checkout về, bạn nhớ cài đặt các dependencies cần thiết bằng câu lệnh: `npm install`. Sau đó gõ: `npm start` để chạy dự án, nếu gặp lỗi: "*System limit for number of file watchers reached - react-native", cách sửa như sau link: https://github.com/facebook/create-react-app/issues/7612#issuecomment-565380404. Cụ thể: gõ lệnh: `$ echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf`. Sau đó gõ tiếp: `$ sudo sysctl -p`*

Hoặc bạn có thể code online tại đây (fork về tài khoản của bạn):
https://codesandbox.io/s/gallant-haze-rplk8

Mã nguồn dự án hiện tại được tạo bằng công cụ *create-react-app*, và chỉ có tệp *index.js* có nội dung như sau:

```
import React from "react";
import ReactDOM from "react-dom";

const element = <h1>Hello from Create React App</h1>;

ReactDOM.render(element, document.getElementById("root"));
```

Trong đoạn code trên, chúng ta thực hiện render trực tiếp một element vào cây DOM thực. Thực tế thì không ai làm thế này cả. Người ta sẽ render một React component (như mình đã giải thích trong phần Virtual DOM)

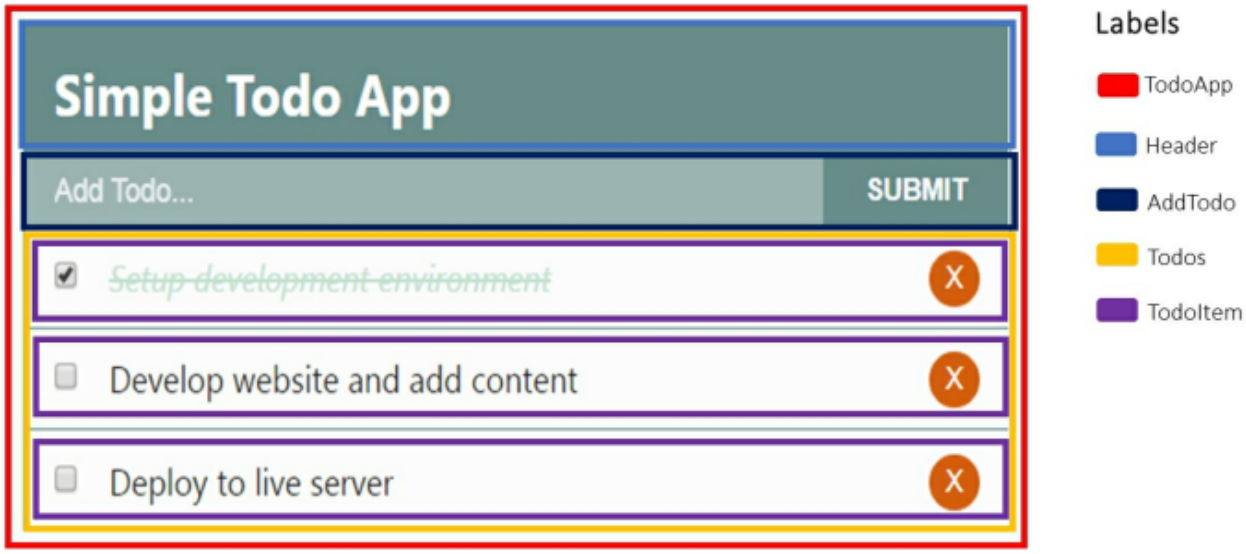
Những kiến thức React cơ bản nhất

1. React Component

Nhưng mình đã cập trước đó, mọi thứ bạn nhìn trên giao diện (UI) đều được chia thành các thành phần độc lập, gọi là Component. Điều này có nghĩa là, khi bạn nhận bản thiết kế giao diện từ đội UX Graphic, bạn cần suy nghĩa làm sao chia nhỏ giao diện đó thành nhiều phần độc lập với nhau, sau đó tái sử dụng chúng.

Nếu bạn nhìn hình ảnh giao diện ứng dụng dưới đây. Thoạt nhìn ban đầu, trong nó cũng "hầm hố" phết, đúng không?

Tư duy mà React muốn bạn làm quen ở đây đó chính là **chia tách.**



Hình 4.2: Giao diện ứng dụng Todos List

Để xây dựng các ứng dụng như này, thậm chí cả những ứng dụng phức tạp như Facebook, Twitter... Việc đầu tiên là bạn cần hình thành tư duy là chia tách. Bạn cần chia thiết kế UI thành nhiều phần nhỏ hơn, càng độc lập với các thành phần khác càng tốt.

Như hình trên, sau khi mình phân tích và khoanh vùng, mình tạo được 3 thành phần nhỏ nhất, có thể tái sử dụng: *header, AddTodo, TodoItem*.

Các thành phần nhỏ nhất này được tái sử dụng để tạo thành một thành phần phức tạp hơn. Ví dụ: nhiều thành phần TodoItem sẽ tạo ra thành phần Todos.

Về mặt code, React component không có gì đặc biệt cả, điểm nổi bật duy nhất là nó **độc lập và có thể tái sử dụng**.

Đây cũng chính là một trong những yếu tố giúp React phổ biến.

Có thể coi component như một tính năng đơn giản mà bạn có thể gọi ở bất kỳ đâu, chỉ cần bạn truyền giá trị đầu vào và nó sẽ hiển thị kết quả cho bạn.

React có 2 kiểu viết component: Functional component và Class component.

1. Functional component

Functional component là một hàm Javascript (hoặc ES6) trả về 1 React element. Theo tài liệu chính thức của React, hàm dưới đây là một component hợp lệ.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Function này là một component React hợp lệ vì nó nhận một "props" làm tham số và trả về 1 React element.

Vì vậy mình có thể định nghĩa 1 component như 1 hàm Javascript thông thường:

```
function exampleFunctionalComponent() {
  return ( <h1>Tôi là một functional component!</h1> );
};
```

Hoặc theo ES6 arrow function:

```
const exampleFunctionalComponent = () => {
  return ( <h1>Tôi là một functional component!</h1> );
};
```

Tóm lại, 1 React function component:

- Là một function Javascript / ES6 function
- Phải trả về 1 React element.
- Nhận props làm tham số nếu cần.

Functional component cũng được biết tới với cái tên là stateless components. Bởi vì chúng không thể làm nhiều thứ phức tạp như quản lý React State (data) hoặc xử lý vấn đề liên quan tới life-cycle trong functional components.

Tuy nhiên, từ phiên bản React 16.8, nhà phát hành giới thiệu tính năng React Hooks. Với Hooks, chúng ta có thể sử dụng state và những features khác trong functional components. Chúng ta sẽ tìm hiểu và thực hành về React Hooks ở phần 6 cuốn sách.

2. Class component

Các Class components là những class ES6. Chúng phức tạp hơn functional components một chút.

Đó là Class component còn có:

- Phương thức khởi tạo, có hàm về vòng đời component, hàm *render()*
- State (dữ liệu ứng dụng).

Ví dụ một class component:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Tóm lại, một Class component là:

- Là một class kế thừa từ *React.Component*
- Có thể nhận props (trong hàm khởi tạo) nếu cần.
- Phải có hàm *render()* và trong đó trả về 1 React element hoặc NULL.

Đây chỉ là bản demo - phần này còn nữa
Đặt sách **tại đây** để đọc bản đầy đủ nhé

FETCHING DỮ LIỆU TỪ API

Từ đầu tới giờ, ứng dụng Todo của chúng ta mới chỉ lưu dữ liệu vào biến State, đây có thể coi là biến tạm thôi. Vì khi reload lại ứng dụng, dữ liệu sẽ bị reset lại từ đầu.

Thực tế, dữ liệu sẽ được lưu vào cơ sở dữ liệu (database) và đặt tại một máy chủ nào đó. Các ứng dụng front-end như Todo App sẽ chỉ có nhiệm vụ request tới máy chủ đó để đọc và ghi dữ liệu. Máy chủ cần phải cung cấp REST API để ứng dụng front-end như Todo App sử dụng.

Trong trường hợp ứng dụng Todo, những tác vụ sau có thể request tới API của máy chủ:

- Lấy danh sách Todos.
- Thêm một Todo mới.
- Xóa một Todo.

Các request tới máy chủ được thực hiện trên giao thức HTTP, nên nhiều khi có thể gọi là HTTP request.

Để tạo HTTP request tới REST API, chúng ta có thể sử dụng native fetch API hoặc dùng thư viện ngoài, phổ biến nhất là thư viện axios.

Đây chỉ là bản demo - phần này còn nữa
Đặt sách **tại đây** để đọc bản đầy đủ nhé

Giới thiệu API cung cấp cho ứng dụng Todo

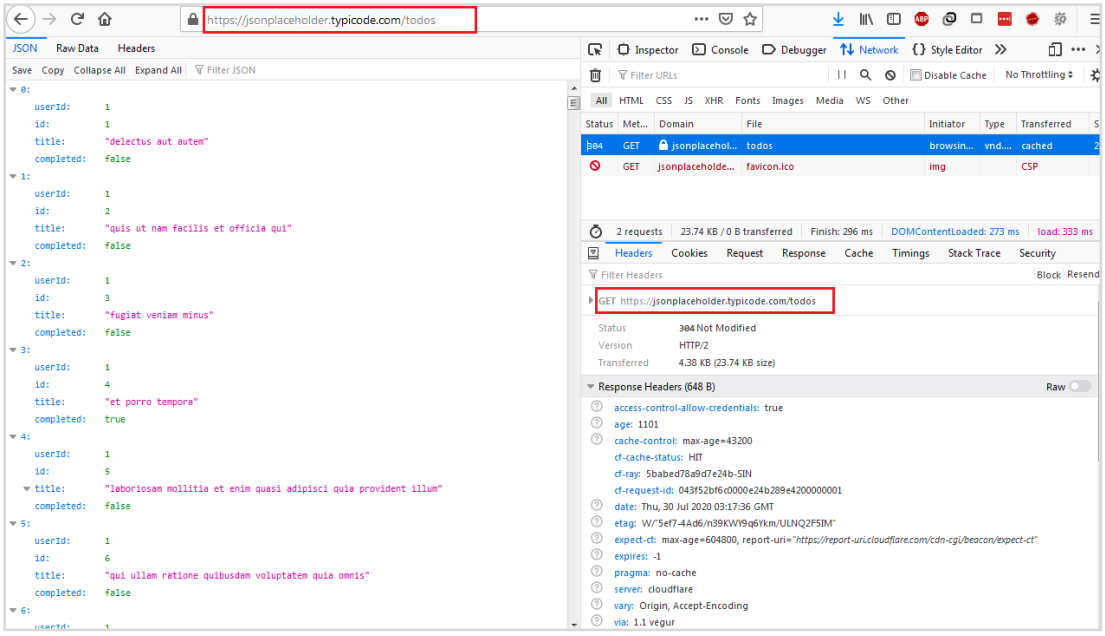
Về nguyên tắc thì chúng ta sẽ cần tự xây dựng một web service trên một máy chủ để lưu trữ và quản lý các Todo, sau đó cung cấp REST API để tương tác với ứng dụng Todo.

Trong khuôn khổ cuốn sách này, mình sẽ không hướng dẫn các bạn các xây dựng web service này, cũng như cách tạo các REST API. Thay vào đó, mình sẽ sử dụng luôn một dịch vụ fake REST API online: https://jsonplaceholder.typicode.com/

Dưới đây là các API sử dụng trong Todo:

Chức năng	Methods	URI
Lấy danh sách các Todos	GET	https://jsonplaceholder.typicode.com/todos
Thêm một Todo mới	POST	https://jsonplaceholder.typicode.com/todos
Xóa một Todo	DELETE	https://jsonplaceholder.typicode.com/todos/${id}

Bạn có thể kiểm tra các API này qua các công cụ như Postman. Với API có phương thức GET thì kiểm tra luôn trên trình duyệt cũng được. Ví dụ như API lấy danh sách todos nhé.



Hình 5.2: Kết quả trả về của API

Phần API coi như đã xong, phần tiếp theo, chúng ta sẽ cần fetching danh sách todos lúc bắt đầu vào ứng dụng Todo. Để làm điều này, chúng ta cần hiểu và sử dụng đến các hàm liên quan tới vòng đời component (life-cycle method).

Vòng đời Component trong React

Tất cả các component bạn tạo ra sẽ phải trải qua một loạt các events hoặc các giai đoạn từ khi được gọi tới khi bị hủy.

Cũng giống cuộc sống của con người vậy, ai cũng phải trải qua đủ các giai đoạn từ khi sinh ra, đến lúc trưởng thành rồi "về với đất mẹ".

Trong React, một component sẽ phải trải qua 3 giai đoạn chính:

1. Mounting: Đúng như tên gọi của nó, đây là giai đoạn mà Component được tạo và thêm vào cây DOM. Đây có thể coi là giai đoạn bắt đầu của một Component. Các methods được gọi trong giai đoạn này:

- *componentWillMount()*
- *render()*
- *componentDidMount()*.

2. Updating: Sau khi Component qua giai đoạn đầu tiên, tức là đã được thêm vào DOM, giai đoạn tiếp theo là cập nhập nếu có yêu cầu. Component sẽ được update khi giá trị state hoặc props thay đổi, lúc này Component sẽ được render lại. Các methods trong giai đoạn này gồm:

- *componentWillReceiveProps()*
- *shouldComponentUpdate()*
- *componentWillUpdate()*
- *render()*
- *componentDidUpdate()*

3. Unmounting: Đây là giai đoạn kết thúc vòng đời của Component, khi nó được xóa khỏi DOM. Chỉ có duy nhất 1 method trong giai đoạn này:

- *componentWillUnmount()*

Như bạn thấy, trong mỗi giai đoạn, React sẽ cung cấp các life-cycle methods để theo dõi và thao tác với những gì xảy ra trong component.

Từ đầu cuốn sách đến giờ, có lẽ bạn gặp nhiều nhất là hàm *render()*.

Đối với người mới tiếp cận React, ngoài *render()* là dùng nhiều nhất, bạn còn sử dụng hàm *componentDidMount()* nhiều không kém. Chủ yếu là lúc bạn cần tạo một HTTP request để lấy dữ liệu ban đầu.

Ok, giờ chúng ta quay trở lại với ứng dụng Todo nhé.

Đây chỉ là bản demo - phần này còn nữa
Đặt sách **tại đây** để đọc bản đầy đủ nhé

REACT HOOKS

Như trong phần tìm hiểu về Component, chúng ta biết rằng, có 2 kiểu viết component: Class Component và functional Component.

Trong đó, functional Component có những hạn chế nhất định như: không có state, không có life-cycle method...

Nhưng functional component có ưu điểm là dễ unit test hơn, phù hợp với trường phái viết code theo kiểu hướng function thay vì hướng đối tượng (OOP).

React Hooks ra đời để khắc phục những nhược điểm của functional component và còn hơn thế nữa.

React Hooks cho phép chúng ta sử dụng state và các tính năng khác của React mà không phải dùng đến Class**.**

Có thể thấy, các nhà phát triển React họ đang muốn hướng đến 1 tương lai **Functional Programming** thay vì sử dụng những Class mà chỉ nghe cái tên thôi là ta đã nghĩ ngay đến OOP (lập trình hướng đối tượng). Cộng với việc không sử dụng Class kế thừa từ React Component nữa nên giờ đây kích thước bundle sẽ được giảm đáng kể.

<Đây chỉ là bản demo - phần này còn nữa>

Refactoring mã nguồn Todo App sử dụng Hooks

Quay trở lại ứng dụng Todo, để áp dụng những kiến thức mới học về React Hooks, chúng ta sẽ cùng nhau thay đổi mã nguồn để sử dụng Hook nhé.

Xem lại mã nguồn của ứng dụng Todo hiện tại, chúng ta thấy có 2 component có sử dụng đến State là: *AddTodo* và *TodoApp*.

Chúng ta sẽ cùng nhau tiến hành chuyển 2 component trên thành functional component và sử dụng Hooks nhé. Còn các component khác thì để lại, nếu bạn thích thì có thể tự thực hành chuyển nhé.

Đầu tiên là *AddTodo.js*. Do component này chỉ có 1 state là title nên mình sẽ dùng cách khai báo thứ nhất.

```
import React, { useState } from "react";

function AddTodo (props) {
    const [title, setTitle] = useState("");

    const onInputChange = e => {
        setTitle(e.target.value)
    };

    const addTodo = e => {
        e.preventDefault();
        props.addTodo(title);
        setTitle("");
    };

    return (
            <form className="form-container" onSubmit={addTodo}>
                <input
                    type="text"
                    placeholder="Add Todo..."
                    className="input-text"
                    value={title}
                    onChange={onInputChange}
                />
                <input
                    type="submit"
                    value="Submit"
                    className="input-submit"/>
            </form>
    );

}
export default AddTodo;
```

Đây chỉ là bản demo - phần này còn nữa
Đặt sách tại đây để đọc bản đầy đủ nhé

QUẢN LÝ STATE VỚI REDUX

Redux là một thư viện Javascript giúp quản lý State của ứng dụng.

Khi mà ứng dụng ngày càng phức tạp, state ngày càng nhiều hơn... đó là lúc chúng ta cần phải quản lý state.

State có thể bao gồm:

- Dữ liệu trả về từ phía máy chủ và được cached lại.
- Hoặc dữ liệu được tạo ra và thao tác ở phía client nhưng chưa được đẩy lên phía server.
- UI state cho các ứng dụng phức tạp. Chúng ta cần quản lý việc active Routes, selected tabs, spinners, điều khiển phân trang.v.v...

Để giải quyết những khó khăn khi quản lý State, Redux ra đời. Redux nổi lên như 1 hiện tượng, nó thậm chí thay thế luôn kiến trúc Flux của Facebook dùng cho React. Hiện tại Facebook cũng khuyến cáo các lập trình viên chuyển qua dùng Redux vì nhiều ưu điểm được cải tiến so với Flux.

3 nguyên tắc của Redux

Khi sử dụng Redux, bạn cần ghi nhớ kỹ 3 nguyên tắc sau:

- **Chỉ có một nguồn dữ liệu tin cậy duy nhất**: Tức là State của toàn bộ ứng dụng được lưu trong trong 1 store duy nhất. Đó là 1 Object theo mô hình cây (tree).
- **State chỉ được phép đọc**: Tức là chỉ có 1 cách duy nhất để thay đổi giá trị state. Đó là tạo ra một ACTION (cũng là 1 object để mô tả những gì xảy ra).
- **Thay đổi chỉ bằng hàm JS thuần túy**: Để chỉ ra cách mà State được biến đổi bởi Action chúng ta dùng các pure function gọi là Reducer.

Khi nào thì sử dụng Redux?

Về bản chất thì Redux cũng chỉ là một công cụ giúp bạn quản lý State hiệu quả hơn. Do vậy, mỗi người, mỗi dự án lại có đặc điểm riêng mà bạn sử dụng Redux hợp lý.

Về cá nhân mình thì coi Redux để gom các State dùng chung về một "tổng kho". Tức là sẽ có một nơi để lưu các State được dùng trong toàn bộ ứng dụng. Ví dụ: trạng thái login, theme.v.v... Các state được lưu bằng redux có thể gọi là global state.

Nhiệm vụ của Redux đơn giản là lưu giá trị vào global state đó, hỗ trợ lấy giá trị ra để sử dụng. Còn việc sử dụng giá trị của global state đó như thế nào? dùng làm gì?... đó không phải phải nhiệm vụ của Redux.

Thành phần của Redux

Về cơ bản, Redux có những thành phần sau:

1. Actions

Trong Redux, Action là nơi mang các thông tin để gửi từ ứng dụng đến store.

Action là 1 đối tượng định nghĩa 2 thuộc tính:

- **type**: kiểu mô tả action.
- **payload**: giá trị tham số truyền lên (không bắt buộc).

Ví dụ một action:

```
{
  type: 'ADD_TODO',
  text: 'Optimize the site speed'
}
```

2. Reducers

Action có nhiệm vụ mô tả những gì xảy ra nhưng lại không chỉ rõ cụ thể phần state nào thay đổi. Việc này sẽ do Reducer đảm nhiệm.

Reducer nhận 2 tham số đầu vào:

- State cũ.
- Action được gửi lên.

Khi thao tác với state, reducer không được phép sửa đổi state. Thay vào đó, reducer dựa trên giá trị state cũ, cộng với dữ liệu đầu vào để tạo nên đối tượng state mới.

3. Store

Store thức chất là 1 đối tượng để lưu trữ state của toàn bộ ứng dụng.

Store có 3 phương thức sau:

- **getState():** Truy cập vào state và lấy ra state hiện tại.
- **dispatch(action):** Thực hiện gọi 1 action.
- **subscribible(listener):** Nó có vai trò cực quan trọng, luôn luôn lắng nghe xem có thay đổi gì ko rồi ngay lập tức cập nhật ra View.

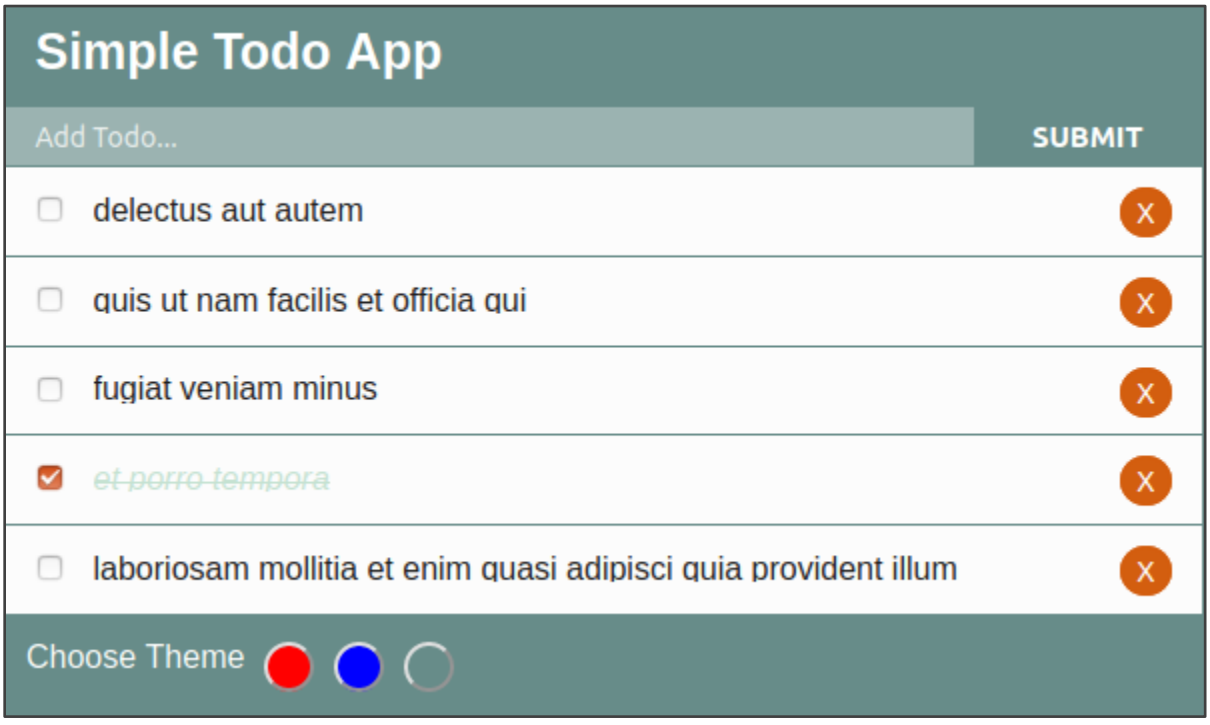
Đến đây, chúng ta cũng hiểu cơ bản về Redux rồi, giờ mình sẽ áp dụng Redux vào dự án xây dựng ứng dụng Todo.

Thực hành sử dụng Redux trong Todo App

Trong ứng dụng Todo, chúng ta thêm tính năng mới là thay đổi theme. Tức là có thể thay đổi màu của toàn ứng dụng.

Giá trị mã màu sẽ được lưu trong global state, và được quản lý bởi redux.

Giao diện ứng dụng khi thêm tính năng thay đổi theme như sau:



Hình 7.1: Tính năng thay đổi theme

Trước khi đi vào thực hành sử dụng Redux, chúng ta cần tạo sẵn giao diện phần Footer, nơi để người dùng thay đổi theme. Cách thực hiện tương tự như đã làm với các phần trước, chúng ta sẽ đi nhanh nhé.

Đây chỉ là bản demo - phần này còn nữa
Đặt sách **tại đây** để đọc bản đầy đủ nhé

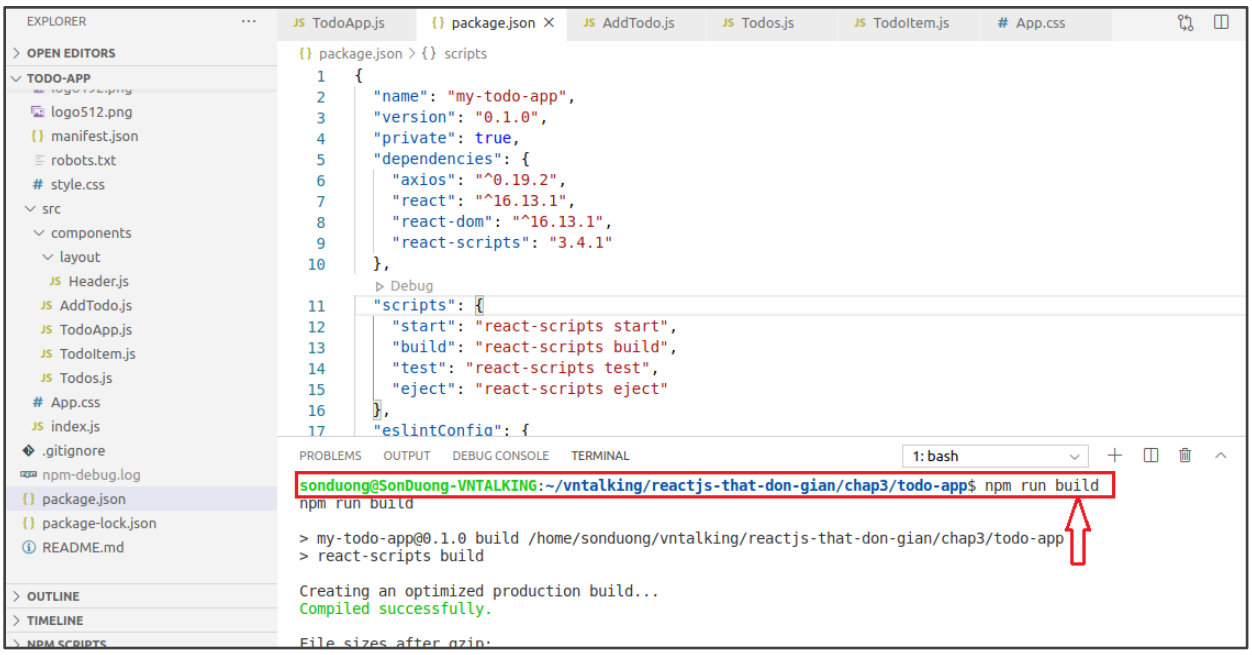
DEPLOY ỨNG DỤNG REACT

Về cơ bản, ứng dụng do React tạo ra là các website tĩnh, tức là chỉ gồm có HTML, CSS, JS. Sau khi xây dựng ứng dụng xong, bạn có thể build dự án và triển khai lên môi trường Internet.

Mình xin giới thiệu 2 cách để triển khai ứng dụng React:

Cách 1: Build dự án, sau đó triển khai lên bất kỳ static server nào.

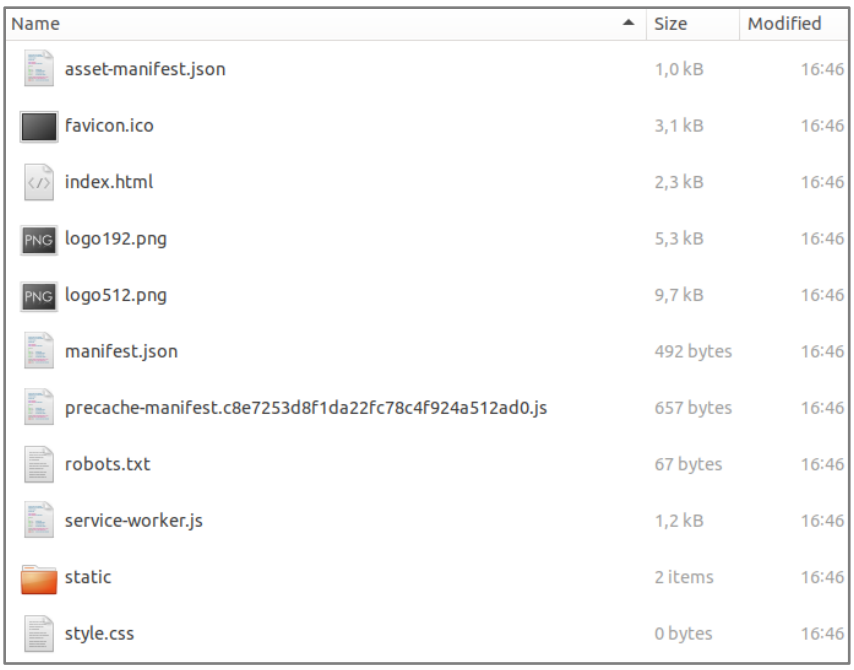
Tại màn hình terminal của dự án, gõ lệnh: `npm run build`



Hình 5.1: Lệnh build dự án

Khi lệnh build chạy thành công, nó sẽ tạo ra thư mục build trong dự án của bạn.

Hình dưới đây là nội dung thư mục build được tạo ra sau khi thực hiện lệnh build.



Hình 8.1: Nội dung của thư mục build của ứng dụng Todo

Bạn không thể chạy ứng dụng trực tiếp bằng cách mở tệp index.html bằng trình duyệt được. Bởi vì React sử dụng client-side routing nên nó không chạy với file:// URL

Công việc tiếp theo là bạn copy toàn bộ các files trong thư mục build và đẩy lên bất kì static server nào.

Static server có thể bạn tự xây dựng bằng cách sử dụng Nginx, hoặc Apache... Hoặc sử dụng dịch vụ static server như: Gatsby, AWS Amplify, Heroku...

Cách 2: Triển khai lên static server trực tiếp từ mã nguồn để trên github

Cách thứ 2 này cũng gần tương tự với cách thứ nhất, chỉ khác là bạn sẽ triển khai dự án trực tiếp từ mã nguồn trên Github. Ưu điểm nhất của phương pháp này là dự án có thể triển khai nhanh chóng và miễn phí.

Đây chỉ là bản demo - phần này còn nữa
Đặt sách **tại đây** để đọc bản đầy đủ nhé

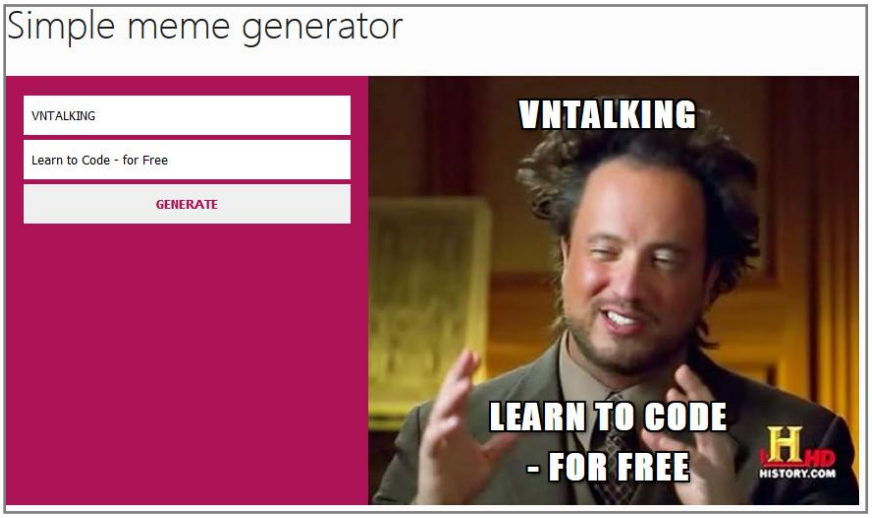
BÀI TẬP

Với mục đích để bạn tự thực hành và củng cố kiến thức React đã học ở các phần trước. Mình tạo ra một bài tập nhỏ nhỏ để bạn tự thực hành. Tại đây, mình sẽ hướng dẫn các bạn thực hành bằng cách chia dự án thành các task nhỏ để các bạn thực hiện. Trong các task, mình sẽ có gợi ý và giải thích cụ thể nhiệm vụ cần thực hiện.

Bạn có sẵn sàng để thực hiện dự án này không?

Nếu đã sẵn sàng thì bắt đầu thôi!

Đây là ứng dụng mà chúng ta sẽ xây dựng.



Hình 9.1: Giao diện ứng dụng Meme Generator

Ứng dụng này làm được gì?

Ứng dụng Meme Generator này nhận những dòng text được người dùng nhập vào, sau đó tạo một hình ảnh ngẫu nhiên khi nhấn vào nút "Generate". Ảnh sẽ được lấy từ trên Internet thông qua HTTP request.

Đây chỉ là bản demo - phần này còn nữa
Đặt sách tại đây để đọc bản đầy đủ nhé

Cám ơn bạn đã quan tâm tới sách "**Lập trình React thật đơn giản**".

Mời bạn truy cập **Đặt sách "Lập trình React thật đơn giản"** để đặt mua bản sách đầy đủ nhé. Đặc biệt ưu đãi cho các bạn học sinh - sinh viên.