|  |  |  |
| --- | --- | --- |
|  | TRƯỜNG CAO ĐẲNG CÔNG NGHỆ THỦ ĐỨC  **Khoa Công Nghệ Thông Tin** | Macintosh HD:Users:hoangnguyen:Documents:TDC:logoTDC_blue.png |
|  |  |  |

**CHUYÊN ĐỀ WEB 2** | HKI – 2021

PRJ – REPORT

**BÁO CÁO KẾT THÚC MÔN HỌC**

**NHÓM: D**

**THÀNH VIÊN:**

Trần Trọng Hiếu - 19211TT1201

Ngô Bảy Hên – 19211TT1120

Nguyễn Huỳnh Quốc Trung - 19211TT0744

**Mã lớp học phần:** 21111TNC10711102

Table of Contents

[DỰ ÁN CHUYÊN ĐỀ WEB 5](#_Toc90325143)

[2.1. Mô Tả Công Nghệ Được Sử Dụng 7](#_Toc90325144)

[2.1.1. ReactJS là gì? 7](#_Toc90325145)

[2.1.2. Redux là gì? 8](#_Toc90325146)

[2.1.3. NextJS là gì? 8](#_Toc90325147)

[2.2. Đặc Tả Dự Án 9](#_Toc90325148)

[2.2.1. Hiển thị confessions 10](#_Toc90325149)

[2.2.2. Phân trang hiển thị confessions 12](#_Toc90325150)

[2.2.3. Sửa confessions 16](#_Toc90325151)

[2.2.4. Đăng nhập confessions 18](#_Toc90325152)

[2.2.5. Đăng ký confessions 21](#_Toc90325153)

[2.2.6. Đăng xuất confessions 24](#_Toc90325154)

[2.2.7. Thêm confessions 26](#_Toc90325155)

[2.2.8. Xóa confessions 28](#_Toc90325156)

[2.2.9. Loading khi xử lý 30](#_Toc90325157)

[2.2.10. Tìm kiếm bài viết 32](#_Toc90325158)

[2.2.11. Trang chi tiết người dùng 33](#_Toc90325159)

[2.2.12. Trang chi tiết bài viết (bài viết liên quan) 35](#_Toc90325160)

[2.2.13. Trang chi tiết bài viết (nội dung) 38](#_Toc90325161)

[2.2.14. Đa ngôn ngữ 40](#_Toc90325162)

[2.2.15. Lọc bài viết 46](#_Toc90325163)

[2.2.16. Ranking user 49](#_Toc90325164)

[2.3. Hướng Dẫn Thêm Về Cấu Hình Dự Án 51](#_Toc90325165)

[2.3.1. Hướng dẫn cài đặt NodeJS. 51](#_Toc90325166)

[2.3.2. Hướng dẫn cài đặt NextJS. 54](#_Toc90325167)

[2.3.3. Hướng dẫn cài đặt Redux. 57](#_Toc90325168)

[2.3.3. Hướng dẫn cài đặt API Scanctum. 64](#_Toc90325169)

[2.3.3. Hướng dẫn tắt CORS API Scanctum. 68](#_Toc90325170)

[2.3.3. Hướng dẫn sử dụng API Scanctum. 70](#_Toc90325171)

Table of Caption

[Hình 1. Giao diện trang chủ của dự án khi hoàn thành. 5](#_Toc90327237)

[Hình 2. Hình ảnh Logo của ReactJS. 6](#_Toc90327238)

[Hình 3. Hình ảnh mô tả vòng đời (life cycle) của Redux. 7](#_Toc90327239)

[Hình 4. Hình ảnh Logo của NextJS. 7](#_Toc90327240)

[Hình 5. Trang chủ hiển thị confessions. 9](#_Toc90327241)

[Hình 6. Gọi useEffect để dispatch actions dữ liệu. 9](#_Toc90327242)

[Hình 7. Gọi API và đưa vào reducers của Redux. 10](#_Toc90327243)

[Hình 8. Sử dụng useSelector của Redux để lấy dữ liệu. 10](#_Toc90327244)

[Hình 9. Hiển thị dữ liệu bằng cách map qua mảng dữ liệu. 10](#_Toc90327245)

[Hình 10. Hiển thị phân trang trên giao diện. 11](#_Toc90327246)

[Hình 11. Lấy số trang hiện tại bằng useRouter, mặc định là 1. 11](#_Toc90327247)

[Hình 12. Sử dụng useEffect để xử lí bài viết hiển thị. 12](#_Toc90327248)

[Hình 13. Hiển thị pagination khi số lượng bài viết lớn hơn 0.. 12](#_Toc90327249)

[Hình 14. Hiển thị Pagination cho bài viết. 13](#_Toc90327250)

[Hình 15. Logic xử lý hiển thị pagination. 14](#_Toc90327251)

[Hình 16. Hiển thị sửa khi người dùng đăng nhập và đúng bài viết. 15](#_Toc90327252)

[Hình 17. Hiển thị modal sửa khi người dùng nhấp vào nút sửa. 15](#_Toc90327253)

[Hình 18. Khi click vào nút sửa sẽ gọi đến function sửa. 16](#_Toc90327254)

[Hình 19. Xử lý sửa bài viết và load lại dữ liệu. 16](#_Toc90327255)

[Hình 20. Nơi hiển thị button Sign In. 17](#_Toc90327256)

[Hình 21. Hiện model để đăng nhập. 17](#_Toc90327257)

[Hình 22. Hiện model để đăng ký. 18](#_Toc90327258)

[Hình 23. Hàm Sign Up action ở Redux. 19](#_Toc90327259)

[Hình 24. Model Sign In. 20](#_Toc90327260)

[Hình 25. Sau đăng nhập thành công. 20](#_Toc90327261)

[Hình 26. Hàm Redux xử lý đăng nhập. 21](#_Toc90327262)

[Hình 27. Hàm sẽ thực hiện khi người dùng reload trang. 21](#_Toc90327263)

[Hình 28. Hàm Redux loading đăng nhập. 22](#_Toc90327264)

[Hình 29. Hiện button Sign Out. 23](#_Toc90327265)

[Hình 30. Ở hàm authReducer. 23](#_Toc90327266)

[Hình 31. Hàm Logout ở Redux. 24](#_Toc90327267)

[Hình 32. HIện thông báo. 25](#_Toc90327268)

[Hình 33. Model Thêm Confessions. 25](#_Toc90327269)

[Hình 34. Khi click vào nút thêm sẽ gọi đến function thêm. 26](#_Toc90327270)

[Hình 35. Xử lý thêm bài viết và load lại dữ liệu. 26](#_Toc90327271)

[Hình 36. Hiển thị xóa khi người dùng đăng nhập và đúng bài viết. 27](#_Toc90327272)

[Hình 37. Hiển thị Model xóa khi người dùng click vào nút xóa. 27](#_Toc90327273)

[Hình 38. Khi click vào nút xóa sẽ gọi đến function xóa. 28](#_Toc90327274)

[Hình 39. Xử lý xóa bài viết và load lại dữ liệu. 28](#_Toc90327275)

[Hình 40. Giao diện loading. 29](#_Toc90327276)

[Hình 41. Tạo 2 case action. 29](#_Toc90327277)

[Hình 42. Dispatch action trong một function xử lý khác. 30](#_Toc90327278)

[Hình 43. Gọi giá trị loading từ Redux và sử dụng. 30](#_Toc90327279)

[Hình 44. Giao diện sau khi tìm kiếm. 31](#_Toc90327280)

[Hình 45. Khai báo onSearch. 31](#_Toc90327281)

[Hình 46. useEffect hiển thị mục tìm kiếm. 31](#_Toc90327282)

[Hình 47. Lọc các bài viết của người dùng đã tạo. 32](#_Toc90327283)

[Hình 48. Sử dụng redux mà trước đó đã lưu thông tin người dùng. 32](#_Toc90327284)

[Hình 49. Nhúng thư viện mã hóa md5 vào dự án. 32](#_Toc90327285)

[Hình 50. Hiển thị hình ảnh người dùng sau khi hash ra mã md5. 33](#_Toc90327286)

[Hình 51. Giao diện hiển thị của bài viết liên quan. 34](#_Toc90327287)

[Hình 52. Function trộn thứ tự của các bài viết. 35](#_Toc90327288)

[Hình 53. Khai báo và gọi hàm bài viết liên quan. 35](#_Toc90327289)

[Hình 54. Kế thừa từ thằng cha của bài viết liên quan. 36](#_Toc90327290)

[Hình 55. Click vào tiêu đề bài viết. 37](#_Toc90327291)

[Hình 56. Giao diện trang chi tiết bài viết. 37](#_Toc90327292)

[Hình 57. useEffect và useState trang chi tiết bài viết. 38](#_Toc90327293)

[Hình 58. HIển thị thông tin trong trang chi tiết bài viết. 38](#_Toc90327294)

[Hình 59. Thư mục chứa nội dung bản dịch và hình ảnh quốc kỳ. 39](#_Toc90327295)

[Hình 60. File nội dung của bản dịch Tiếng Anh. 39](#_Toc90327296)

[Hình 61. Sử dụng thẻ Link của NextJS để set locale của ứng dụng. 40](#_Toc90327297)

[Hình 62. Kiểm tra thư viện đã cài đặt trong package.json. 40](#_Toc90327298)

[Hình 63. Cấu hình tại file next-i18next.config.js. 41](#_Toc90327299)

[Hình 64. Import file cấu hình vào next.config.js. 41](#_Toc90327300)

[Hình 65. Thêm cấu hình vào file \_app.js. 41](#_Toc90327301)

[Hình 66. Sử dụng serverSideTranslations cho từng trang. 42](#_Toc90327302)

[Hình 67. Sử dụng hook useTranslation để gọi bản dịch. 42](#_Toc90327303)

[Hình 68. Để hiển thị ta chỉ cần gọi ra mã nội dung của nó. 43](#_Toc90327304)

[Hình 69. Nội dung được hiển thị bằng Tiếng Anh. 44](#_Toc90327305)

[Hình 70. Nội dung được hiển thị bằng Tiếng Việt. 44](#_Toc90327306)

[Hình 71. Lọc bài viết 45](#_Toc90327307)

[Hình 72. Load bài viết mới nhất. 45](#_Toc90327308)

[Hình 73. Load bài viết ngẫu nhiên. 46](#_Toc90327309)

[Hình 74. Code mới nhất và ngẫu nhiên. 46](#_Toc90327310)

[Hình 75. Khai báo hàm random. 47](#_Toc90327311)

[Hình 76. Gọi hàm random. 47](#_Toc90327312)

[Hình 77. Nội dung được hiển thị của Ranking User. 48](#_Toc90327313)

[Hình 78. Hàm đếm và Sort bài post của Ranking User. 48](#_Toc90327314)

[Hình 79. Nơi được gọi hàm countAndSort. 49](#_Toc90327315)

[Hình 80. Ở lớp cha gọi PostsUserRanking. 49](#_Toc90327316)

[Hình 81. Phiên bản tải nodejs 50](#_Toc90327317)

[Hình 82. Tải về máy gói install 50](#_Toc90327318)

[Hình 83. Chấp nhận điều kiện của nodejs 51](#_Toc90327319)

[Hình 84. Install Nodejs vào máy 51](#_Toc90327320)

[Hình 85. Thời gian chờ install nodejs 52](#_Toc90327321)

[Hình 86. Kiểm tra version và kiểm tra xem đã thành công chưa 52](#_Toc90327322)

[Hình 87. Khởi tạo dự án với npm. 53](#_Toc90327323)

[Hình 88. Cài đặt các thư viện để cấu hình NextJS. 53](#_Toc90327324)

[Hình 89. Thêm scripts vào package.json để chạy NextJS. 54](#_Toc90327325)

[Hình 90. Thêm nội dung giao diện để chạy dự án. 55](#_Toc90327326)

[Hình 91. Ta gọi câu lệnh yarn dev để chạy server. 55](#_Toc90327327)

[Hình 92. Giao diện sau khi chạy lệnh. 55](#_Toc90327328)

[Hình 93. Cài đặt các thư viện cho Redux. 56](#_Toc90327329)

[Hình 94. Chỉnh sửa lại giao diện cho counter redux. 56](#_Toc90327330)

[Hình 95. Giao diện có được sau khi chỉnh sửa. 57](#_Toc90327331)

[Hình 96. Cấu hình thư mục và import counter vào rootReducer. 57](#_Toc90327332)

[Hình 97. Cấu hình file counterReducer. 58](#_Toc90327333)

[Hình 98. Cấu hình file counterActions. 59](#_Toc90327334)

[Hình 99. Cấu hình file store. 60](#_Toc90327335)

[Hình 100. Đưa file store vào file gốc của NextJS để hoàn tất. 61](#_Toc90327336)

[Hình 101. Ta gọi useSelector và useDispatch đến counter redux. 61](#_Toc90327337)

[Hình 102. Giao diện sau khi cấu hình redux đã hoạt động. 62](#_Toc90327338)

[Hình 103. Clone code API 63](#_Toc90327339)

[Hình 104. Check out nhánh laravel-8x/4-scanctum 63](#_Toc90327340)

[Hình 105. Cấu hình lại file môi trường ở laravel-8x 64](#_Toc90327341)

[Hình 106. Update lại compser bằng cmd 65](#_Toc90327342)

[Hình 107. Thêm Schema vào để hoạt động nếu ko chạy được 65](#_Toc90327343)

[Hình 108. Start serve của php API 66](#_Toc90327344)

[Hình 109. Import api để hiển thị giao diện quản lý của Postman 66](#_Toc90327345)

[Hình 110. Chạy thử API xem có hoạt động được không 67](#_Toc90327346)

[Hình 111. Hình chạy câu lệnh composer require fruitcake/laravel-cors 67](#_Toc90327347)

[Hình 112. Chỉnh sửa file config\cors.php 68](#_Toc90327348)

[Hình 113. Chạy câu lệnh yarn add axios 69](#_Toc90327349)

[Hình 114. Cấu hình file môi trường 69](#_Toc90327350)

[Hình 115. Chỉnh lại file môi trường sẽ không commit nó lên git, hãy tạo một file .gitignore 70](#_Toc90327351)

[Hình 116. Viết hàm call API 70](#_Toc90327352)

[Hình 117. Import hàm call API để sử dụng 72](#_Toc90327353)

[Hình 118. Call Api và đỗ dữ liệu ra màn hình 72](#_Toc90327354)

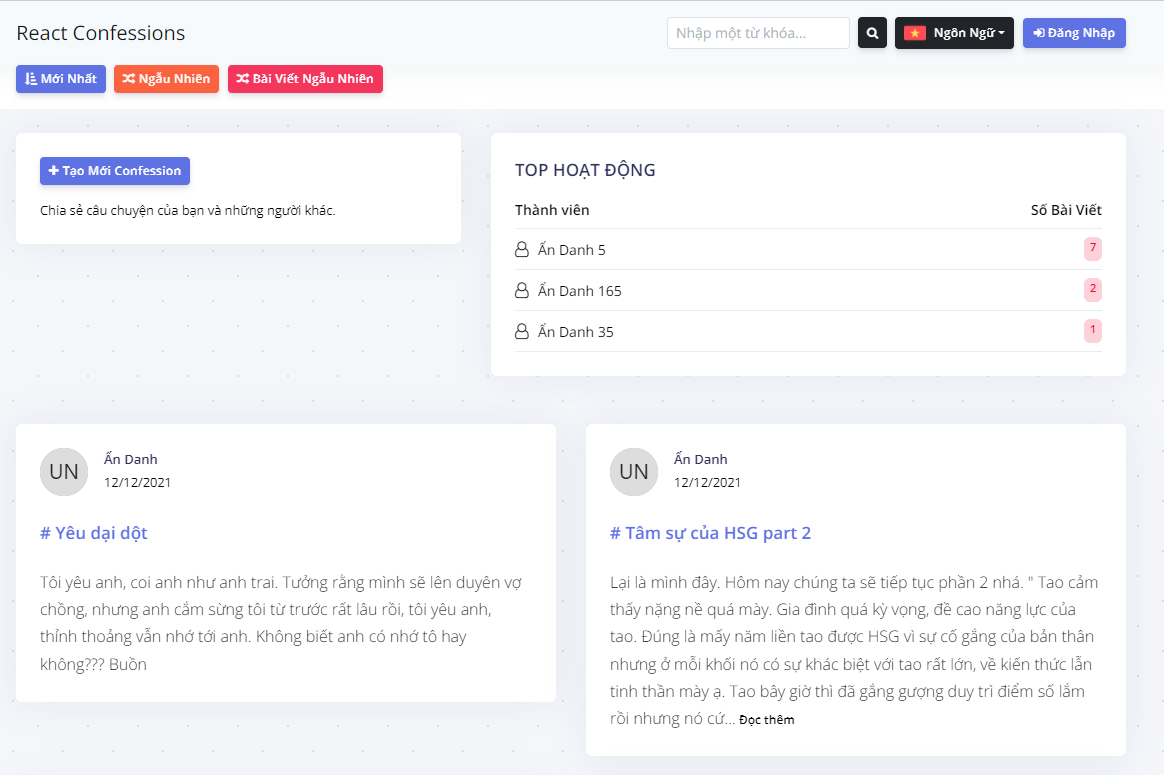
# DỰ ÁN CHUYÊN ĐỀ WEB

Ứng dụng Confessions Vietnam.

Trong thời buổi công nghệ hiện nay, việc chia sẻ thông tin là rất quan trọng. Những vấn đề trong cuộc sống khó khăn càng khiến người ta mệt mỏi và cần người tâm sự. Ứng dụng được tạo ra giúp các bạn chia sẻ và giải tỏa những điều trong lòng, những niềm vui, nỗi buồn một cách hoàn toàn ẩn danh, hoàn toàn bảo mật. Chia sẻ và kết nối cùng những người khác.

Source: <https://github.com/tronghieu60s/chuyen-de-web/tree/cdptw2>

Demo: <https://react-confessions.vercel.app/>



Hình 1. Giao diện trang chủ của dự án khi hoàn thành.

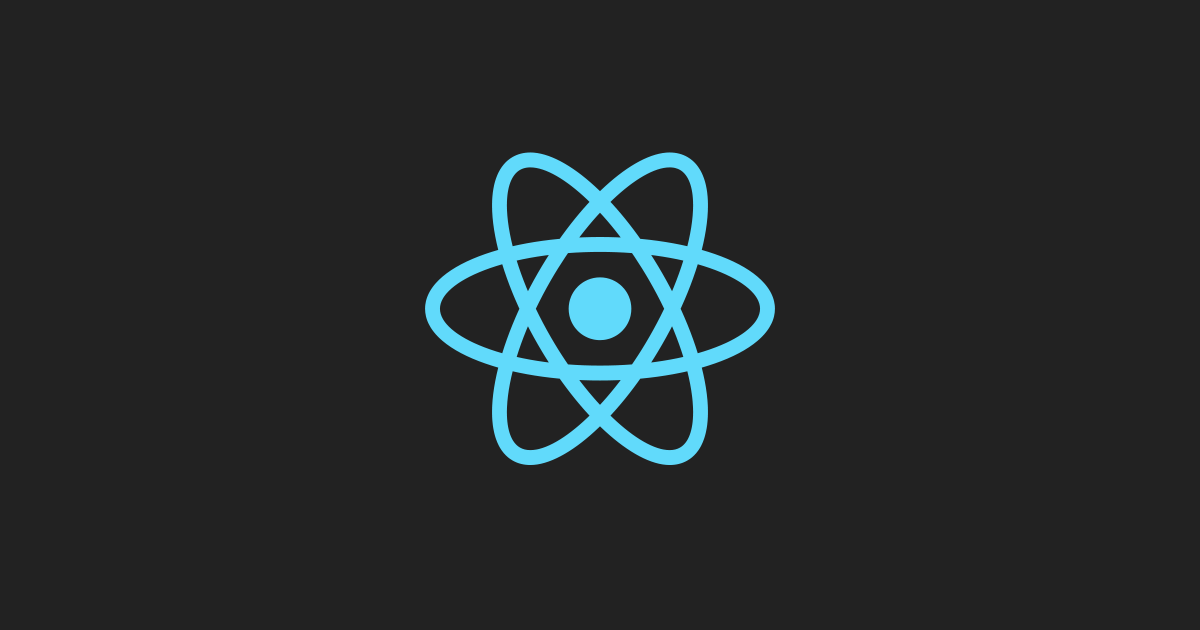
## 2.1. Mô Tả Công Nghệ Được Sử Dụng

Công nghệ được sử dụng: ReactJS (Redux, NextJS)

### 2.1.1. ReactJS là gì?

ReactJs là một thư viện viết bằng **Javascript**, dùng để xây dựng giao diện người dùng (UI). ReactJS được duy trì bởi Facebook và cộng đồng developer hùng mạnh. React cơ bản được sử dụng để phát triển các các **Single Page Application**.

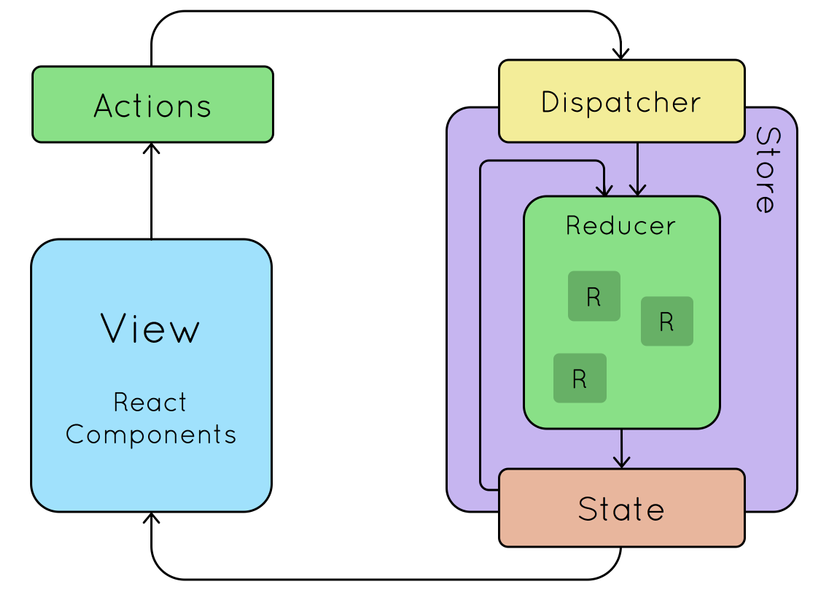
ReactJS sử dụng để tạo giao diện và quản lý cấu trúc cực tốt, và được đứng sau bởi một ông lớn trong ngành IT. Ngoài ra ReactJS được kế thừa một kho thư viện đồ sộ trên NPM. Không thể không kể đến Redux.



Hình 2. Hình ảnh Logo của ReactJS.

### 2.1.2. Redux là gì?

Redux là một thư viện Javascript của **ReactJS**, giúp tạo ra thành một lớp quản lý state của ứng dụng. Mặc định ReactJS cũng có thể quản lý state chung (**React Context**) nhưng cũng khó có thể thay cho Redux vì khả năng quản lý state rõ ràng của nó.



Hình 3. Hình ảnh mô tả vòng đời (life cycle) của Redux.

### 2.1.3. NextJS là gì?

NextJS được tạo ra bởi Vercel là một framework của ReactJS được xây dựng trên nền NodeJS (**Server Side Rendering SSR**) khác với ReactJS là **Client Side Rendering CSR** giúp tạo ra một trang viết bằng ReactJS nhưng được render phía Server, giúp tăng cải thiện SEO đối với một ứng dụng ReactJS.



Hình 4. Hình ảnh Logo của NextJS.

## 2.2. Đặc Tả Dự Án

Các chức năng và phân công công việc CDPTW1:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Thành Viên** | **Nội Dung Được Phân Công** | **Hoàn Thành** |
| 1 | Ngô Bảy Hên | Đăng nhập,  Đăng ký,  Đăng xuất (1) | 100% |
| 2 | Trần Trọng Hiếu | Sửa confessions (3),  Hiển thị confessions,  Phân trang confessions (2) | 100% |
| 3 | Nguyễn Huỳnh Quốc Trung | Thêm confessions (3),  Xóa confessions (4) | 100% |

Các chức năng và phân công công việc CDPTW2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Thành Viên** | **Nội Dung Được Phân Công** | **Thời Gian** | **Hoàn Thành** |
| 1 | Ngô Bảy Hên | Loading khi xử lý API  Trang chi tiết bài viết (bài viết liên quan) | 10 ngày | 100% |
| 2 | Trần Trọng Hiếu | Đa Ngôn Ngữ  Trang Profile người dùng | 14 ngày | 100% |
| 3 | Nguyễn Huỳnh Quốc Trung | Tìm kiếm bài viết  Trang chi tiết bài viết (nội dung) | 10 ngày | 100% |

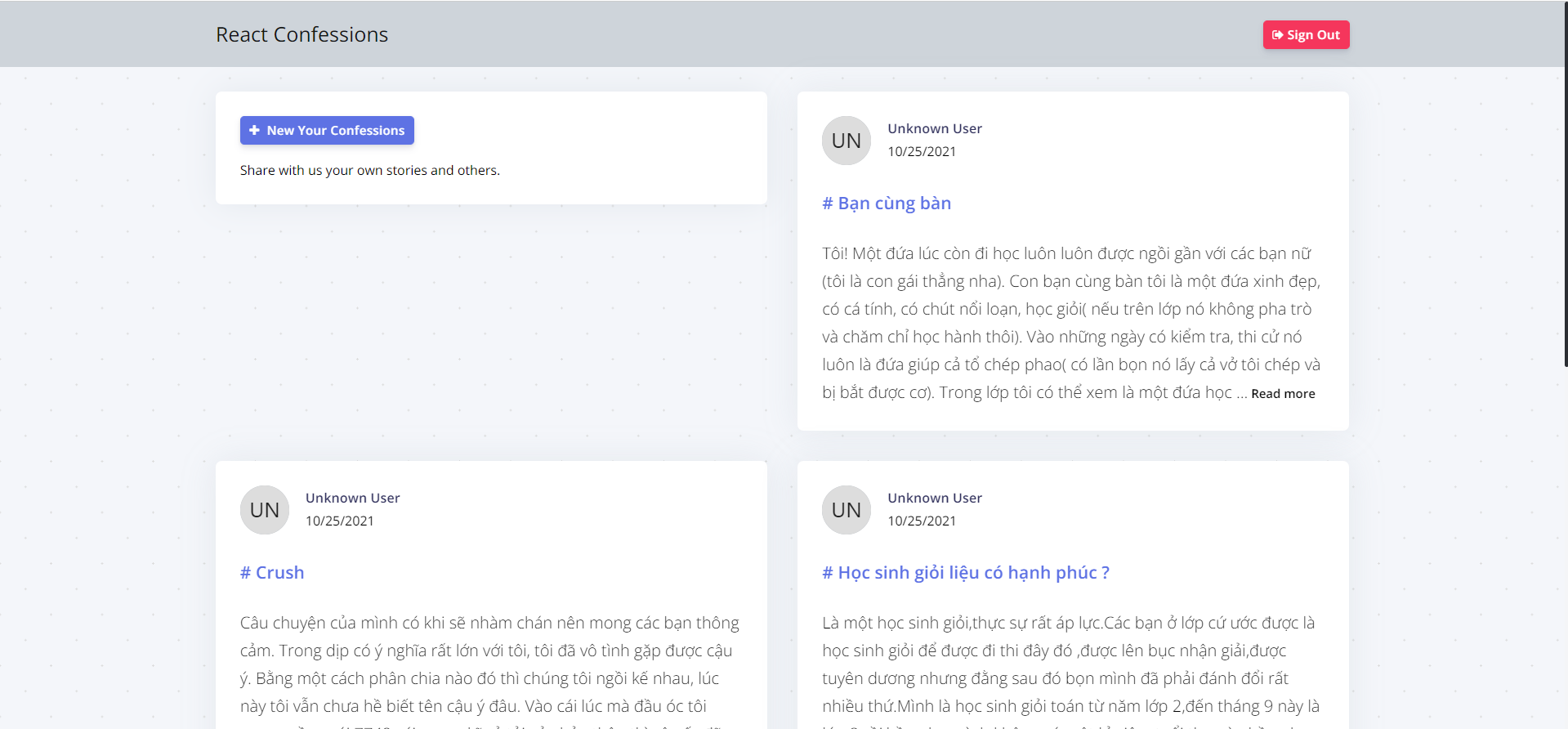
Các chức năng và phân công công việc CDPTW2 (BCKT):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **STT** | **Thành Viên** | **Nội Dung Được Phân Công** | **Bắt Đầu** | **Kết Thúc** | **Hoàn Thành** |
| 1 | Ngô Bảy Hên | Trang chi tiết bài viết (bài viết liên quan) (12)  Ranking user (16) | 25/11 | 11/12 | 100% |
| 2 | Trần Trọng Hiếu | Đa Ngôn Ngữ (14) | 13/11 | 20/11 | 100% |
| 3 | Nguyễn Huỳnh Quốc Trung | Trang chi tiết bài viết (nội dung) (13)  Lọc bài viết (15) | 25/11 | 11/12 | 100% |

### 2.2.1. Hiển thị confessions

Đầu tiên là hiển thị confession. Đây là giao diện hiển thị của confessions bao gồm các phần chính mà dữ liệu được lấy từ API:

* Thời gian của bài viết.
* Tiêu đề của bài viết.
* Nội dung của bài viết.



Hình 5. Trang chủ hiển thị confessions.

Khi vào trang có chứa Posts thực hiện gọi dispatch actions bằng useEffect, và thực hiện gọi API trong Redux, dữ liệu được đưa vào reducers của Redux.

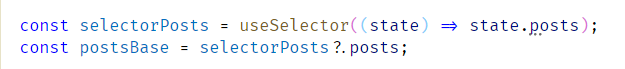


Hình 6. Gọi useEffect để dispatch actions dữ liệu.

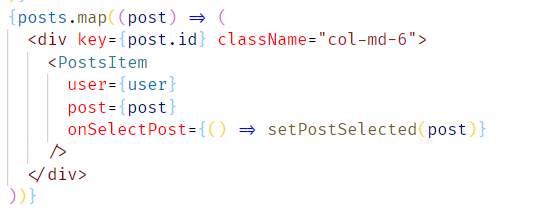


Hình 7. Gọi API và đưa vào reducers của Redux.

Sau khi lấy dữ liệu từ API, ta tiến hành lấy dữ liệu ra từ Redux và hiển thị ra màn hình của người dùng (PostsItem là components giao diện hiển thị).



Hình 8. Sử dụng useSelector của Redux để lấy dữ liệu.

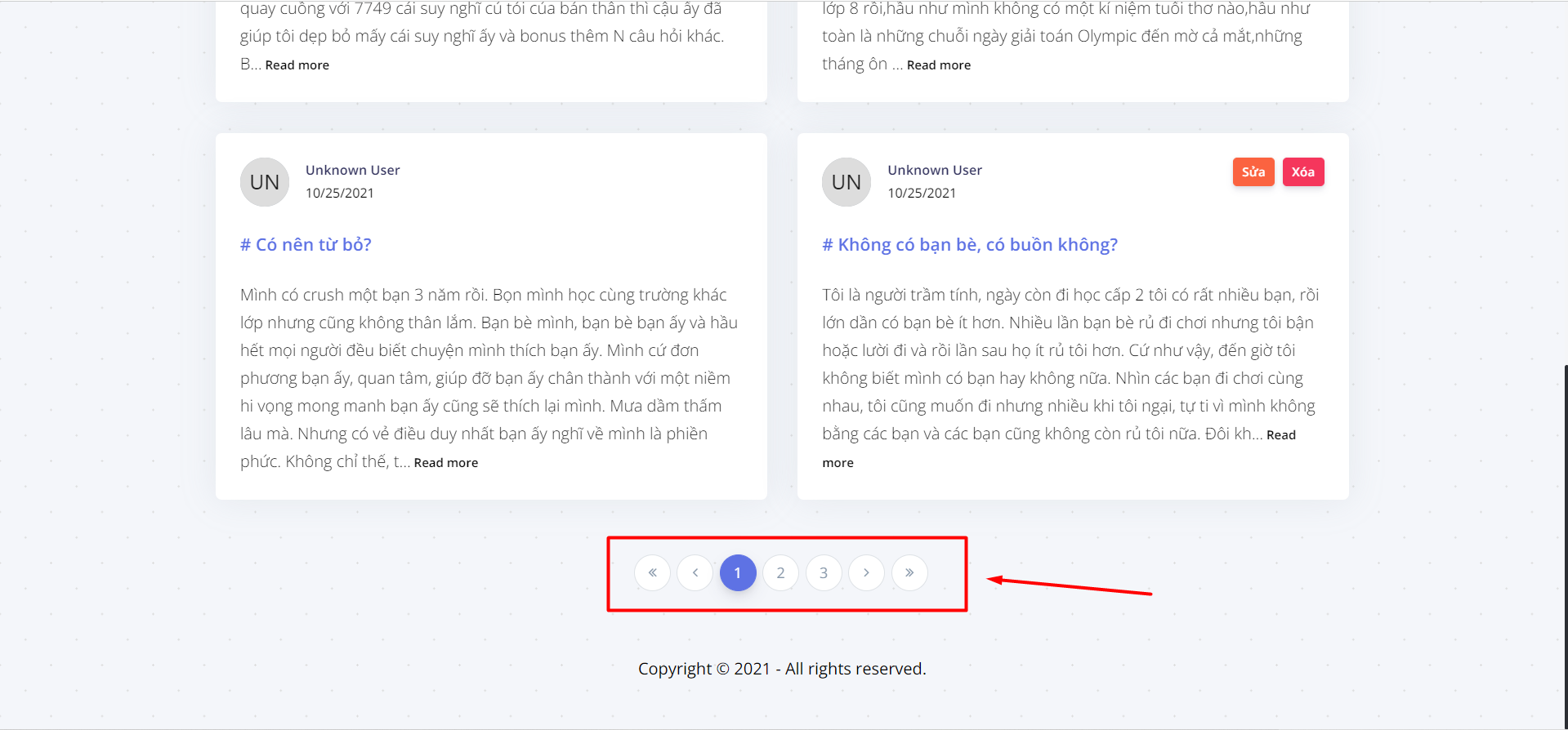


Hình 9. Hiển thị dữ liệu bằng cách map qua mảng dữ liệu.

### 2.2.2. Phân trang hiển thị confessions

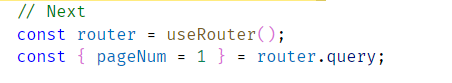
Hiển thị phân trang ở phía dưới của bài viết, giao diện bao gồm:

* Nút back về trang thứ 1.
* Nút back về trang trước đó.
* Hiển thị số trang.
* Nút next đến trang tiếp theo.
* Nút next đến trang cuối cùng.



Hình 10. Hiển thị phân trang trên giao diện.

Để phân trang ta cần lấy số trang hiện tại, vì sử dụng NextJS nên ta có useRouter.



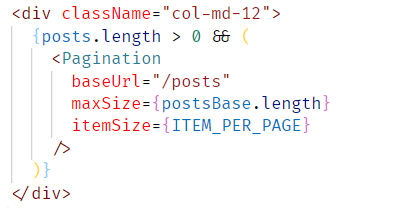
Hình 11. Lấy số trang hiện tại bằng useRouter, mặc định là 1.

Dữ liệu Posts được lấy từ API sẽ được để trong biến postsBase, và dữ liệu hiển thị theo trang hiện tại (posts) sẽ được cắt mảng ra theo chiều dài được xác định trước (ITEM\_PER\_PAGE).



Hình 12. Sử dụng useEffect để xử lí bài viết hiển thị.

Hiển thị Pagination ở trang chủ bằng components Pagination.

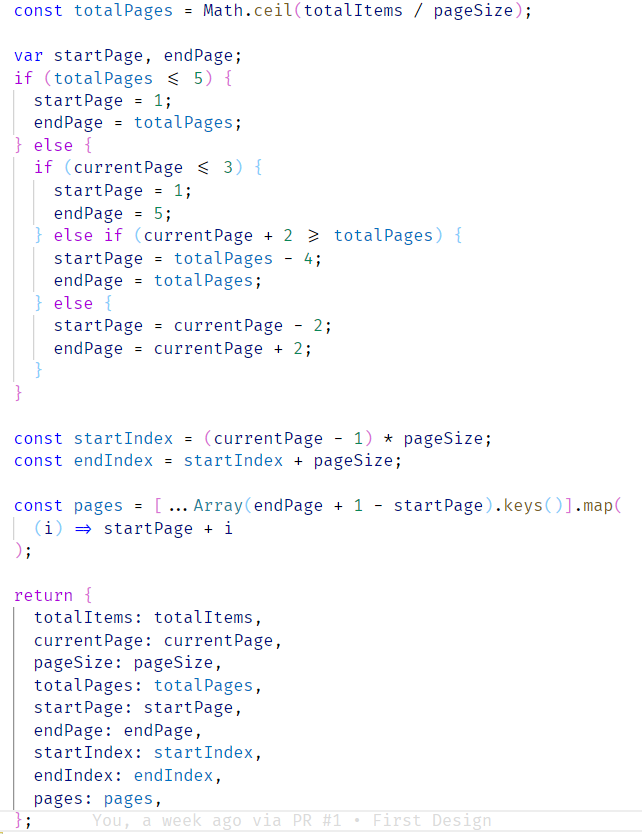


Hình 13. Hiển thị pagination khi số lượng bài viết lớn hơn 0..

Xử lý logic của Pagination được gọi trong hàm getPaginate và hiển thị số của paginate bằng cách map qua pages.



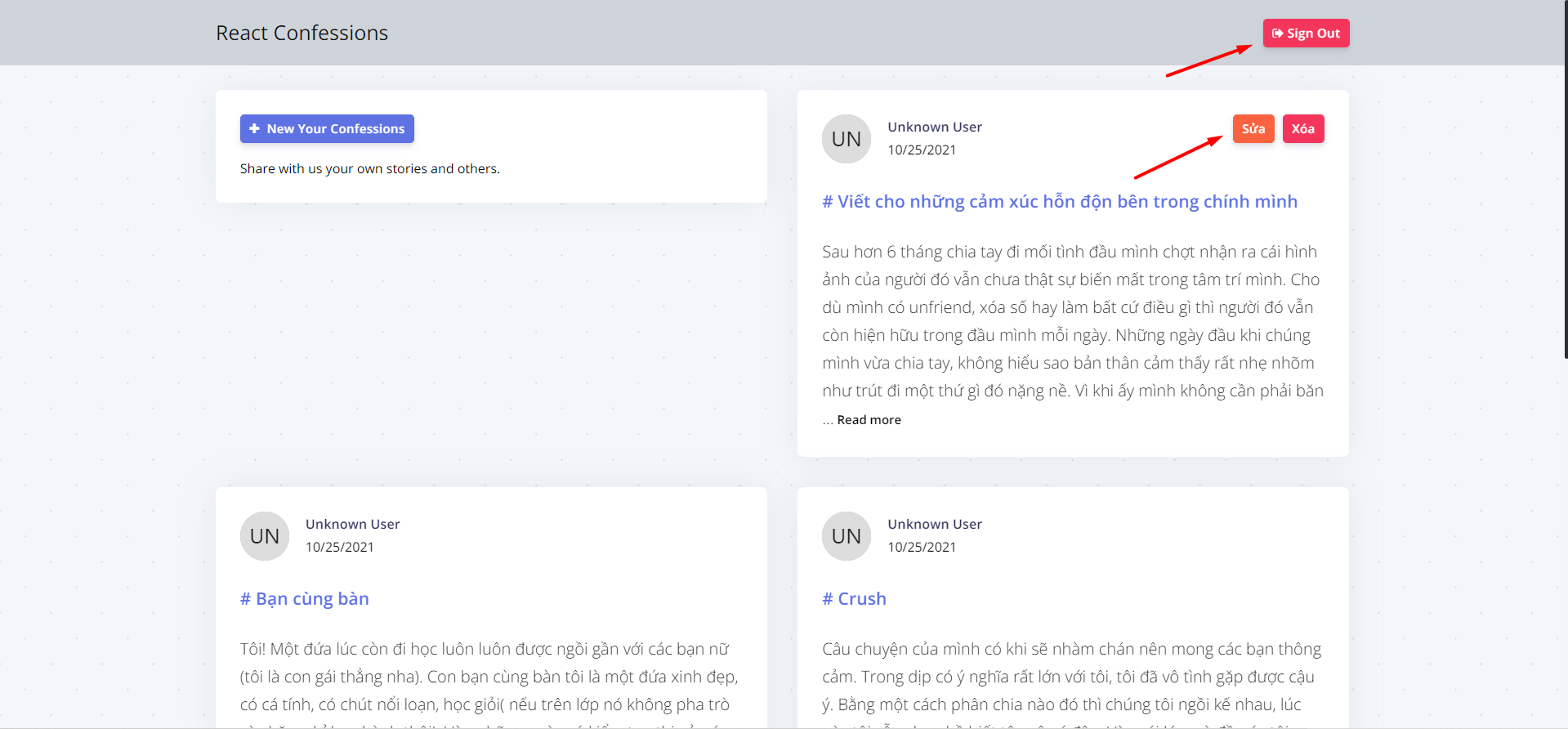
Hình 14. Hiển thị Pagination cho bài viết.



Hình 15. Logic xử lý hiển thị pagination.

### 2.2.3. Sửa confessions

Giao diện sửa confession chỉ cho người đã đăng nhập và tạo bài viết, khi người dùng nhấn vào nút sửa, một modal sẽ hiện ra và cho phép sửa bài viết.

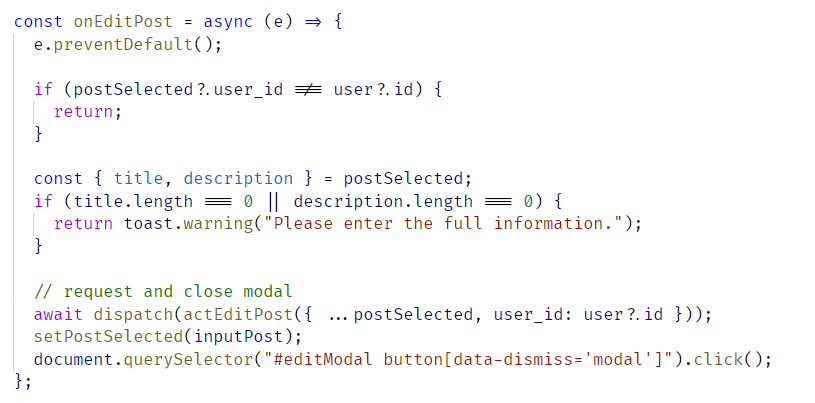


Hình 16. Hiển thị sửa khi người dùng đăng nhập và đúng bài viết.



Hình 17. Hiển thị modal sửa khi người dùng nhấp vào nút sửa.

Khi click vào nút sửa sẻ gọi đến function sửa, đầu tiên sẽ check đúng user id hiện tại. Sau đó check đang input hiện tại có rỗng hay không. Tiếp tục gọi dispatch actions sửa bài viết của Redux, sau đó đóng modal lại.



Hình 18. Khi click vào nút sửa sẽ gọi đến function sửa.

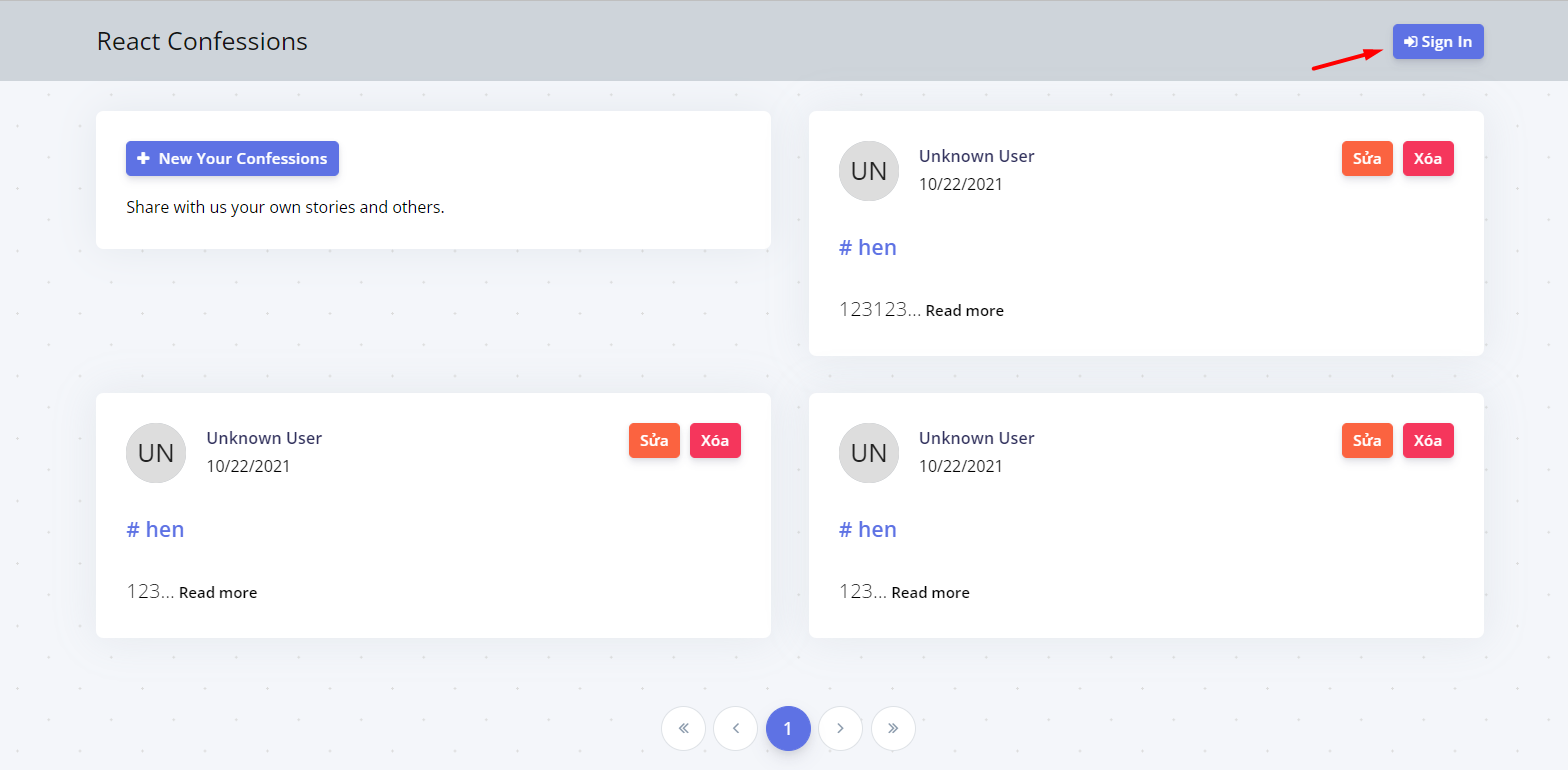
Ở trong Redux sẽ xử lý chức năng sửa bài viết, và load lại tất cả bài viết hiện tại.



Hình 19. Xử lý sửa bài viết và load lại dữ liệu.

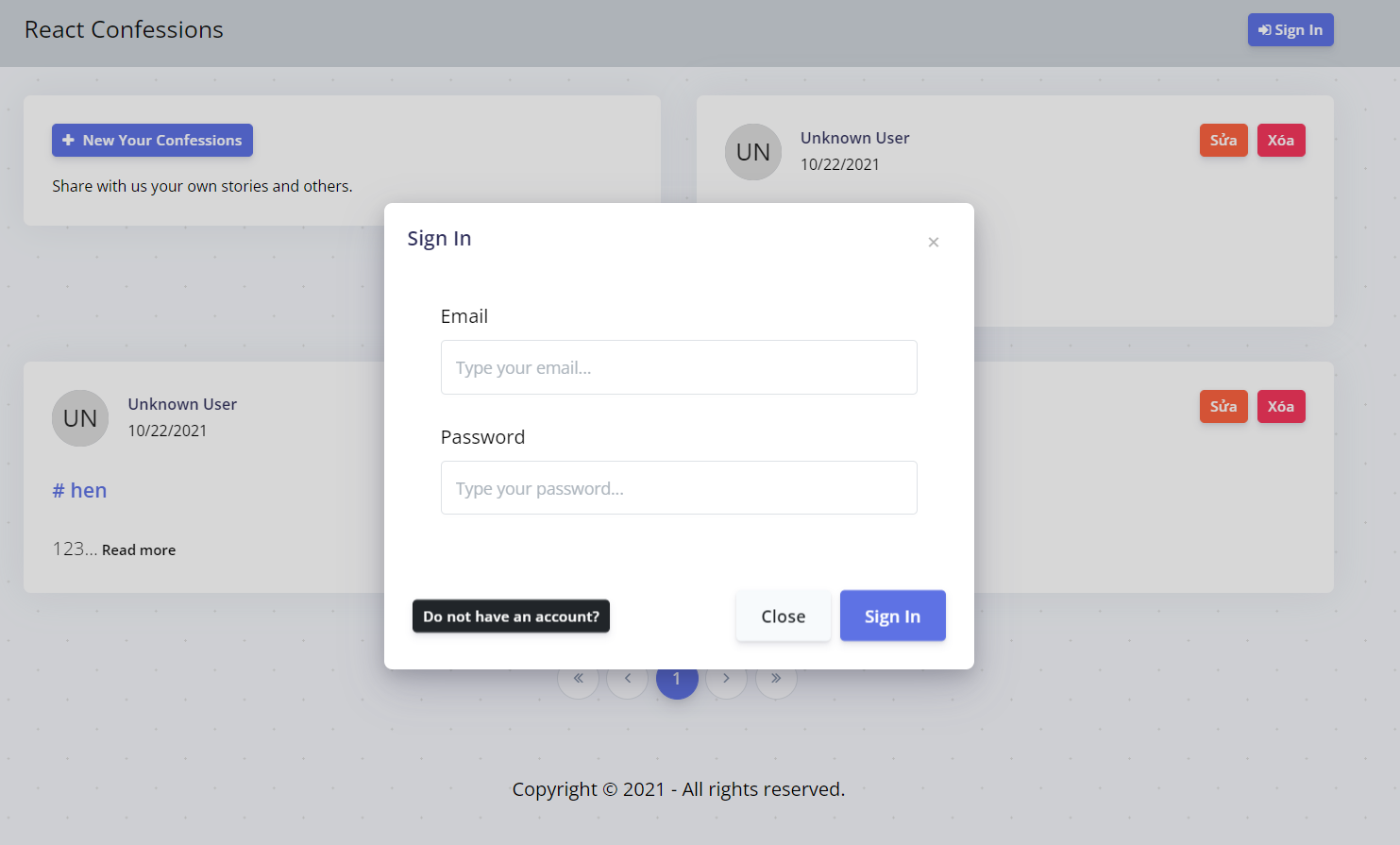
### 2.2.4. Đăng nhập confessions

Ở giao diện chính của web confessions ở trên cùng bên phải của web sẽ có một button Sign In để cho người dùng có thể đăng ký tài khoản.



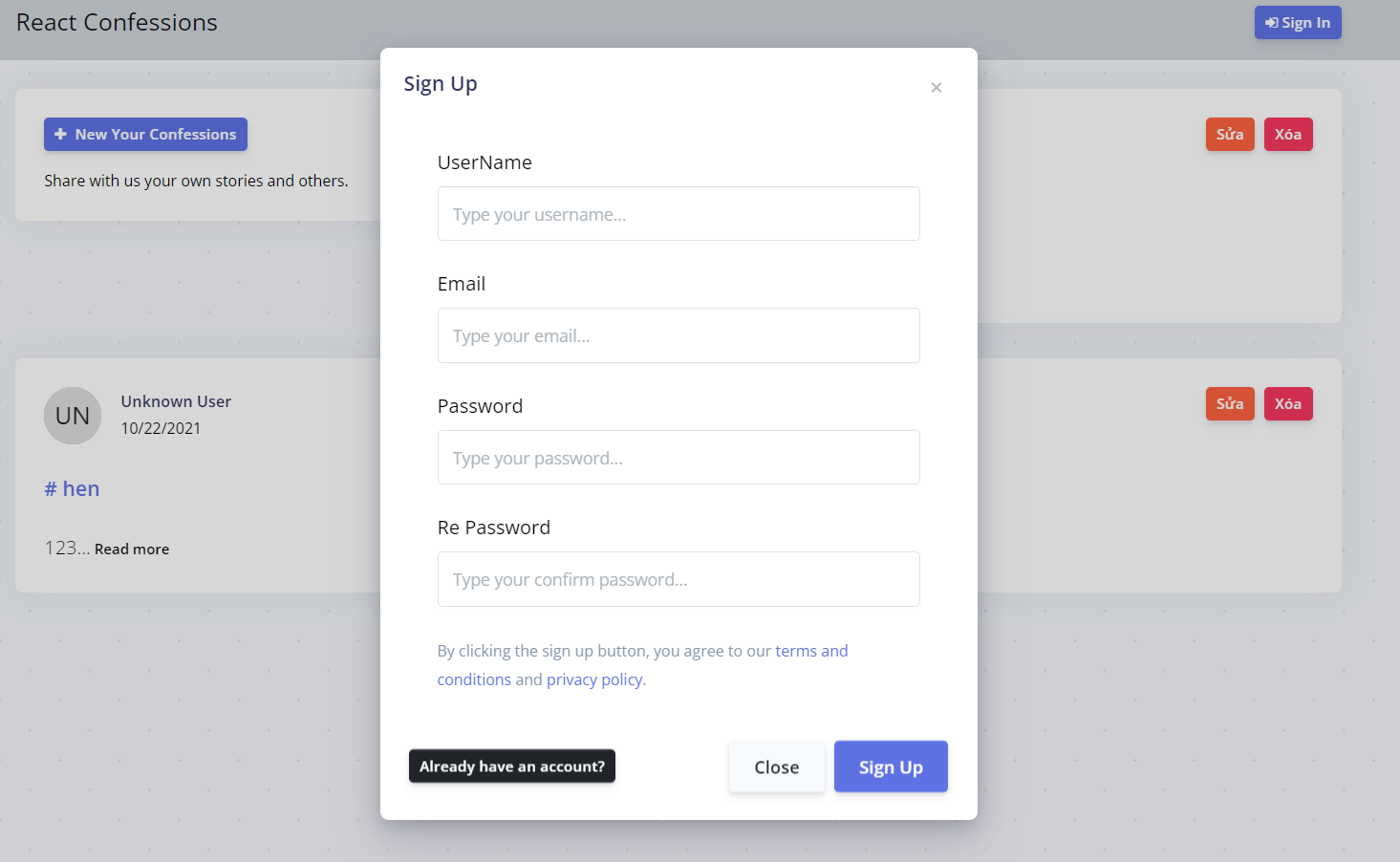
Hình 20. Nơi hiển thị button Sign In.

Khi click vào button Sign In web sẽ hiện lên cho người dùng một model để có thể đăng nhập hoặc đăng ký tài khoản.



Hình 21. Hiện model để đăng nhập.

Ở đây khi người dùng chưa có tài khoản thì học sẽ click vào button “Do not have an account” để web xử lý là hiện ra model đăng ký để người dùng có thể đăng ký tài khoản. Ở model này nếu người dùng có tài khoản thì chỉ cần nhấn button “Already have an account ? ” để có thể đăng nhập.



Hình 22. Hiện model để đăng ký.

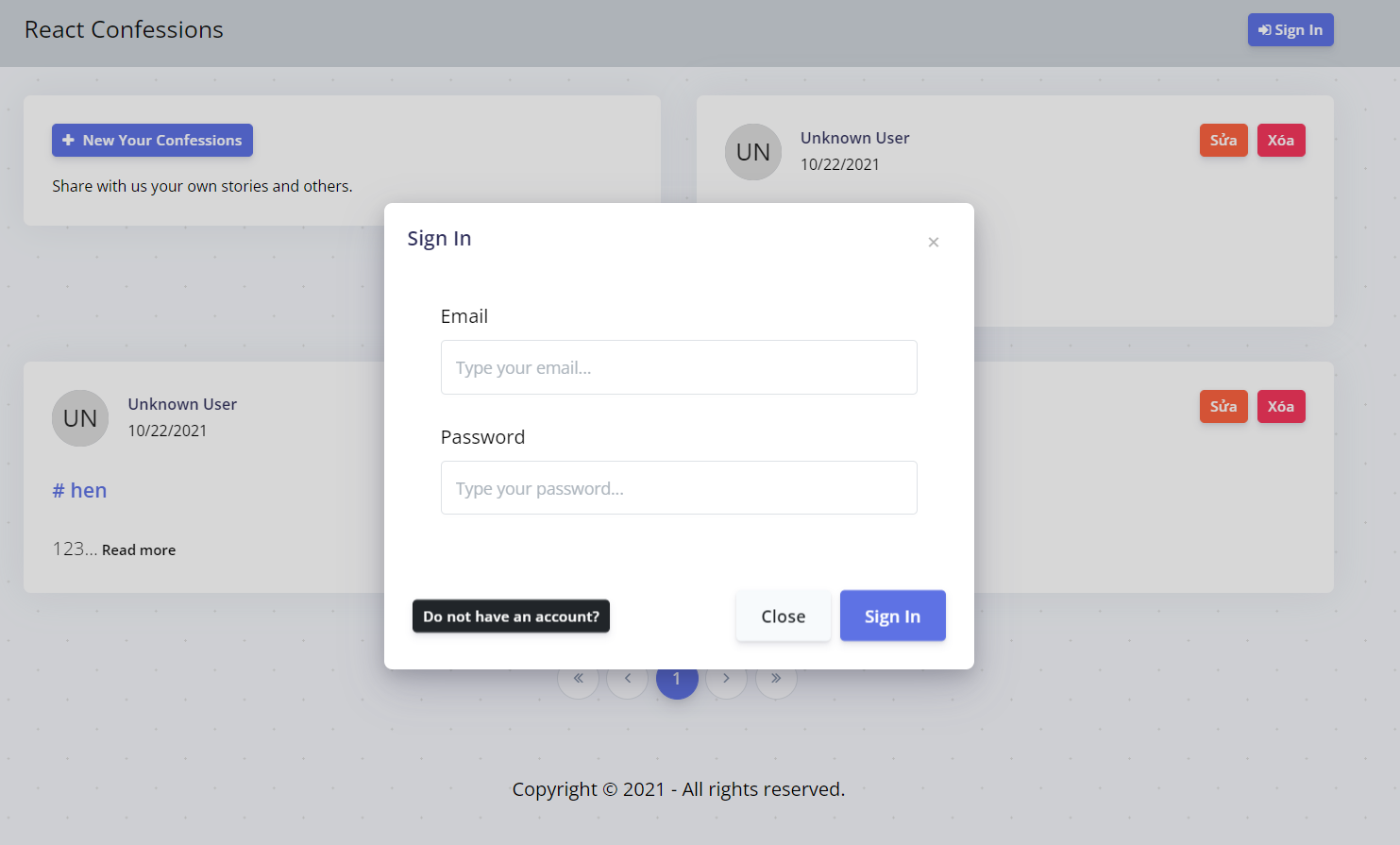
Ở model này người dùng chỉ nhập đầy đủ và đúng thông tin và nhấn button Sign Up web sẽ vào Redux và callApi để thực hiện chức năng đăng ký và thông báo kết quả. Nếu đăng ký thành công thì sẽ web sẽ navigation qua model Sign In.



Hình 23. Hàm Sign Up action ở Redux.

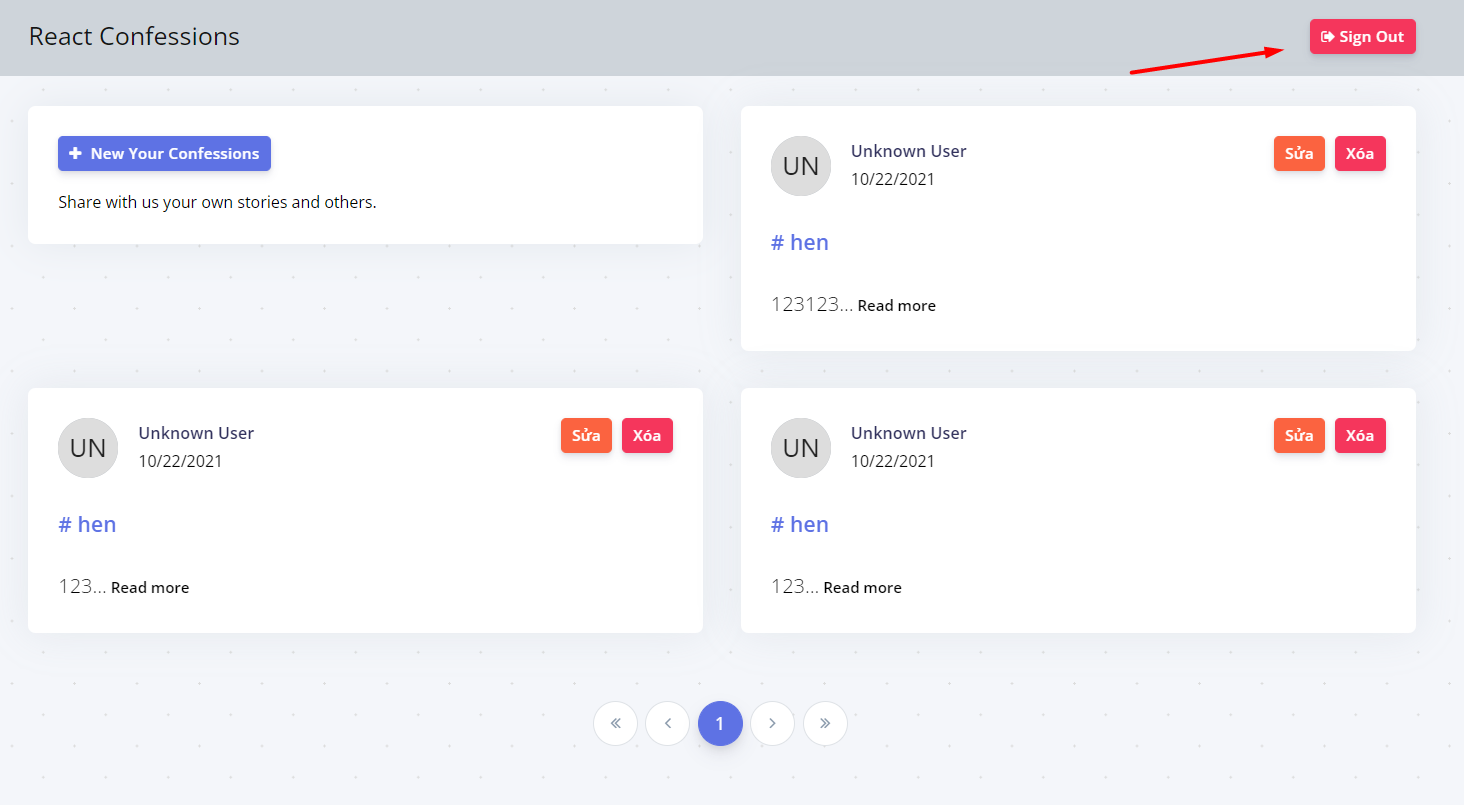
### 2.2.5. Đăng ký confessions

Ở model đăng nhập sau khi click vào button Sign In để hiện ra thì người dùng nếu đã có tài khoản thì chỉ cần đăng nhập điền đầy đủ thông tin và bấm button Sign In để đăng nhập.



Hình 24. Model Sign In.

Ở đây khi người dùng điền thông tin đầy đủ và nhấn button Sign In thì web sẽ vào Redux và callApi để xử lý đăng nhập nếu đăng nhập thành công sẽ đóng model và thông báo đăng nhập thành công chuyển button Sign In thành Sign Out.



Hình 25. Sau đăng nhập thành công.

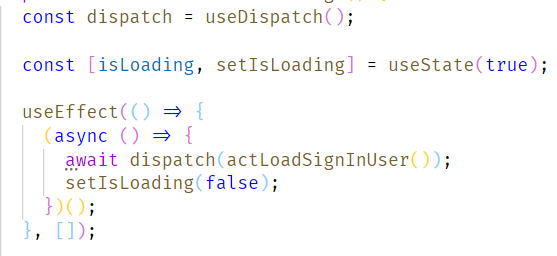


Hình 26. Hàm Redux xử lý đăng nhập.

Sau khi chuyển button thành Sign Out ta phải ghi thêm hàm loading để có thể lưu được trạng thái đăng nhập.



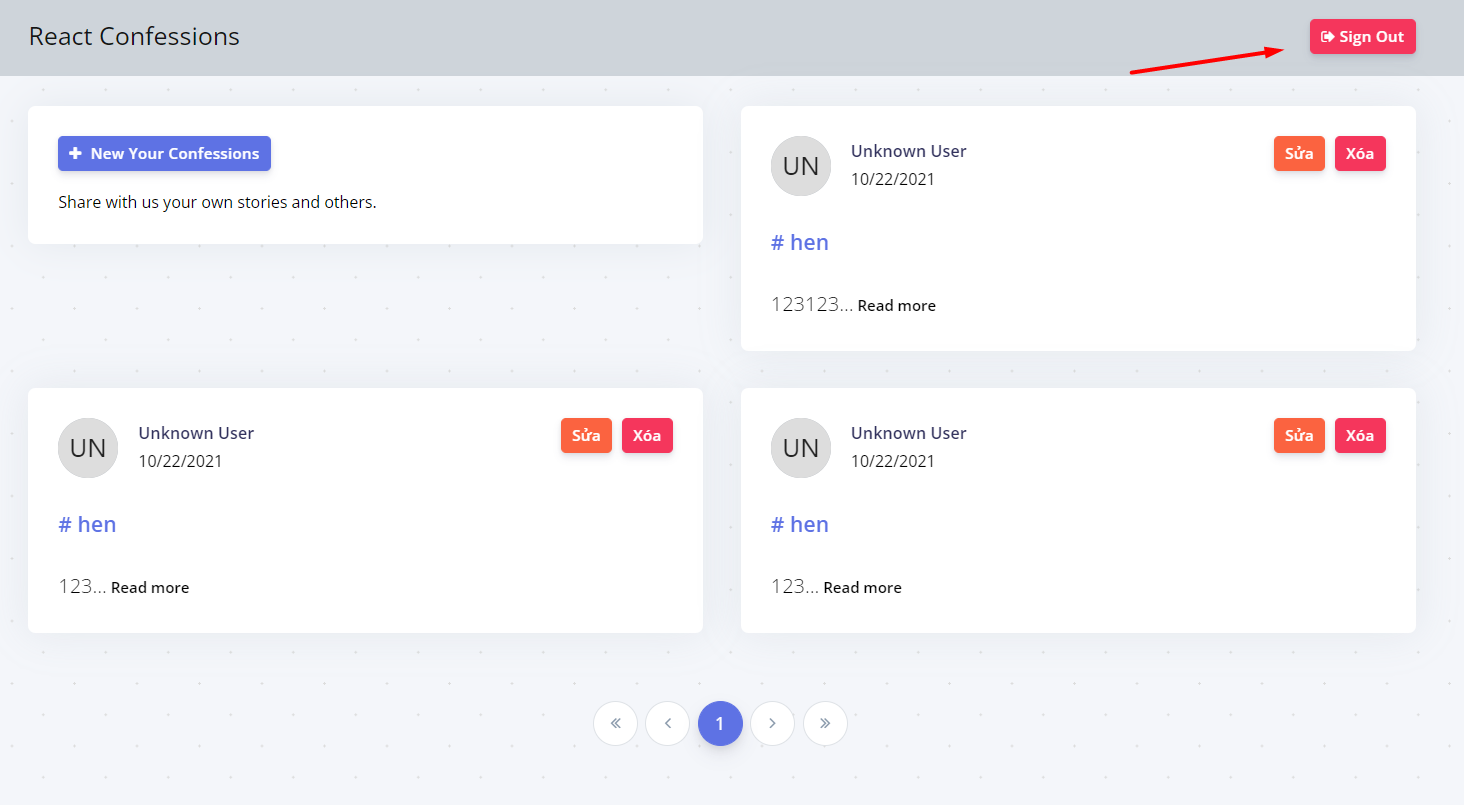
Hình 27. Hàm sẽ thực hiện khi người dùng reload trang.



Hình 28. Hàm Redux loading đăng nhập.

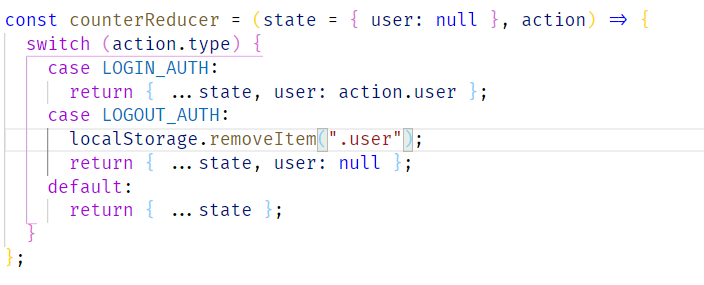
### 2.2.6. Đăng xuất confessions

Khi người dùng đã đăng nhập và muốn Sign out tài khoản của mình ra ngoài thì người dùng sẽ click vào nút button Sign Out để có thể đăng xuất tài khoản ra khỏi web confessions.

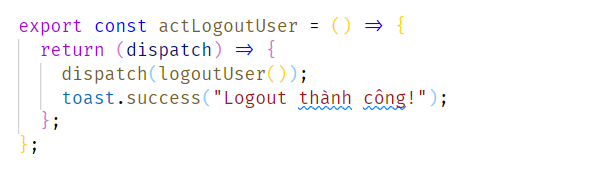


Hình 29. Hiện button Sign Out.

Sau khi click vào button Sign Out thì token ở LocalStorage sẽ bị xóa và button sẽ chuyển về lại thành Sign In và để thực hiện được chuyện đó chúng ta sẽ viết hàm ở Redux tạo action để có thể xóa token ở LocalStorage.



Hình 30. Ở hàm authReducer.

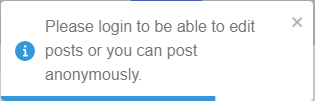


Hình 31. Hàm Logout ở Redux.

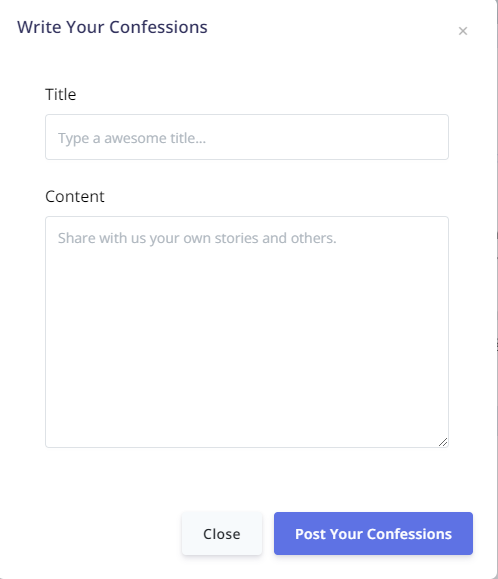
### 2.2.7. Thêm confessions

Khi người dùng Click vào button “New Your Confessions” thì một model sẽ hiện ra cho phép người dùng thêm bài viết.

Trường hợp không đăng nhập thì sẽ có thông báo:

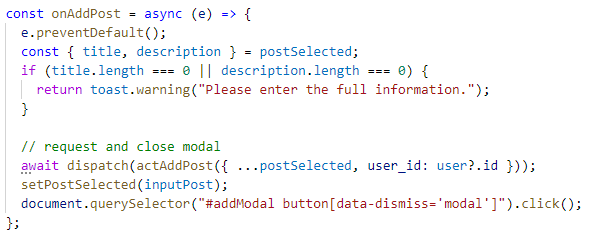


Hình 32. HIện thông báo.



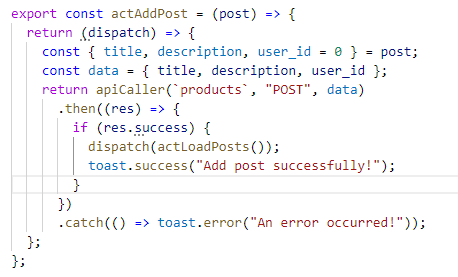
Hình 33. Model Thêm Confessions.

Khi click vào nút “Post Your Confessions”sẻ gọi đến function thêm. Đầu tiên check đang input hiện tại có rỗng hay không. Tiếp tục gọi dispatch actions thêm bài viết của Redux, sau đó đóng model lại.



Hình 34. Khi click vào nút thêm sẽ gọi đến function thêm.

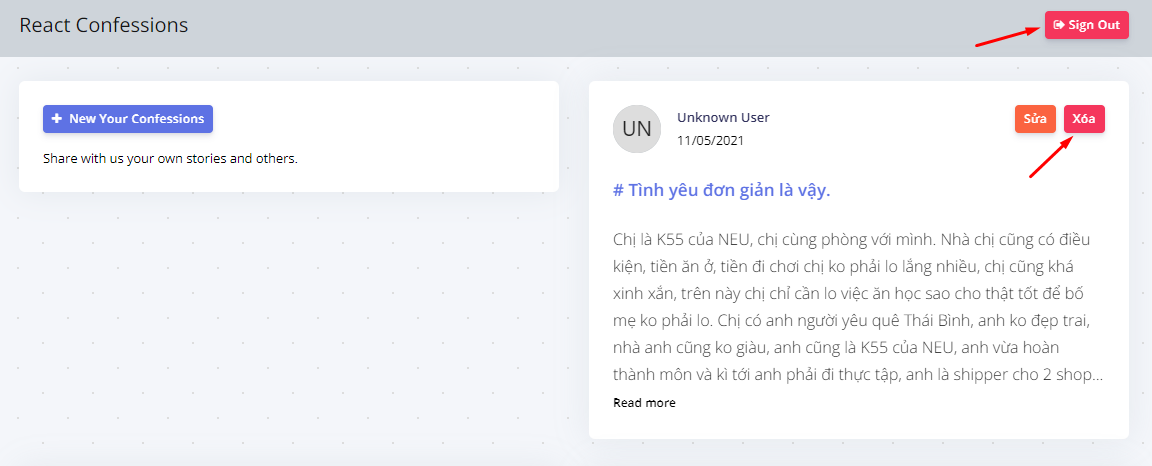
Ở trong Redux sẽ xử lý chức năng thêm bài viết, và load lại tất cả bài viết hiện tại.



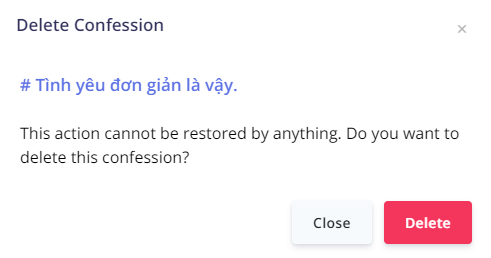
Hình 35. Xử lý thêm bài viết và load lại dữ liệu.

### 2.2.8. Xóa confessions

Giao diện xóa confession chỉ cho người đã đăng nhập và tạo bài viết, khi người dùng nhấn vào nút sửa, một modal sẽ hiện ra và cho phép xóa bài viết.

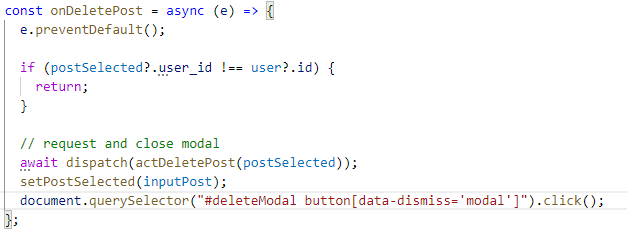


Hình 36. Hiển thị xóa khi người dùng đăng nhập và đúng bài viết.



Hình 37. Hiển thị Model xóa khi người dùng click vào nút xóa.

Khi click vào nút xóa sẻ gọi đến function xóa, đầu tiên sẽ check đúng user id hiện tại. Sau đó check đang input hiện tại có rỗng hay không. Tiếp tục gọi dispatch actions xóa bài viết của Redux, sau đó đóng modal lại.



Hình 38. Khi click vào nút xóa sẽ gọi đến function xóa.

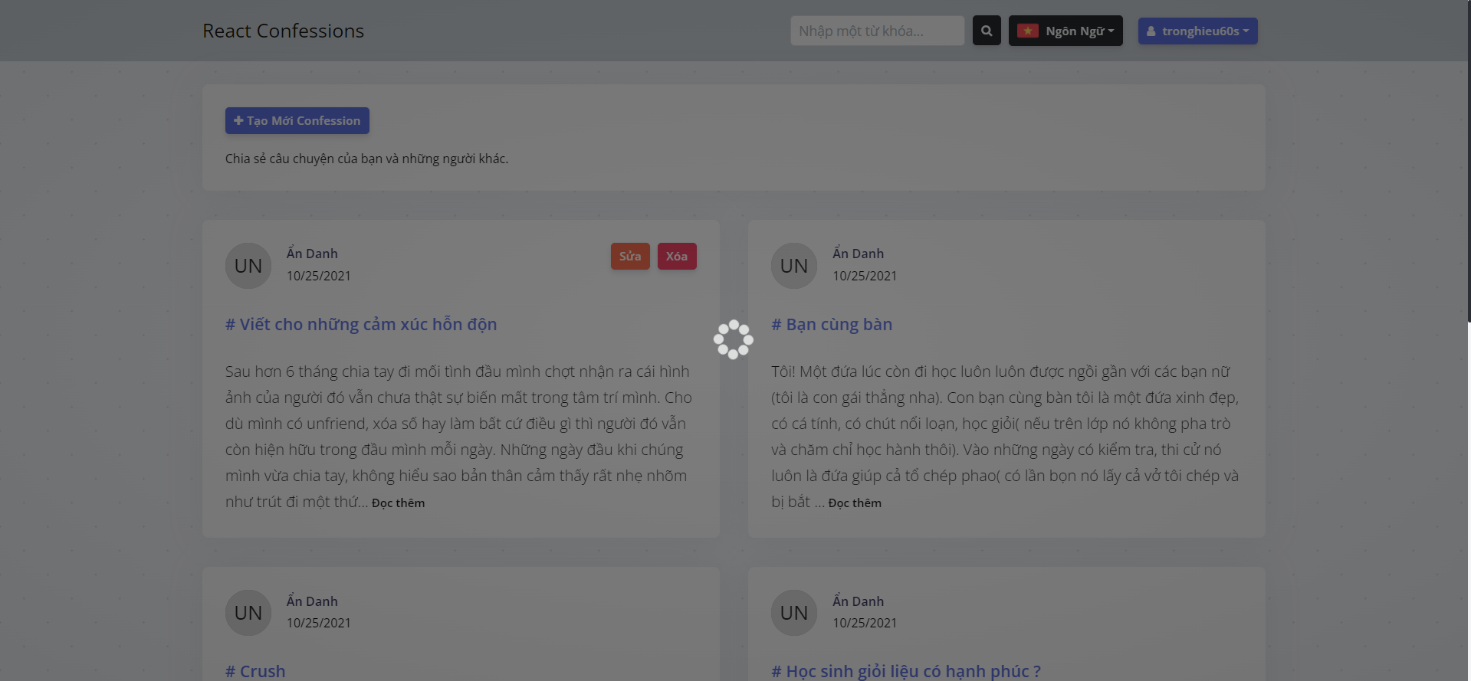
Ở trong Redux sẽ xử lý chức năng xóa bài viết, và load lại tất cả bài viết hiện tại.



Hình 39. Xử lý xóa bài viết và load lại dữ liệu.

### 2.2.9. Loading khi xử lý

Loading sẽ được hiện lên khi người dùng có hành động xử lý và gọi đến API của ứng dụng, tăng UI/UX người dùng.



Hình 40. Giao diện loading.

Để có thể xử lý được loading ở đây chúng ta sẽ dùng Redux để xử lý đầu tiên chúng ta vào commonReducer để set state loading bằng false và tạo 2 case action START\_LOADING VÀ STOP\_LOADING để có thể thay đổi giá trị của loading khi chúng ta gọi làm để chạy action.



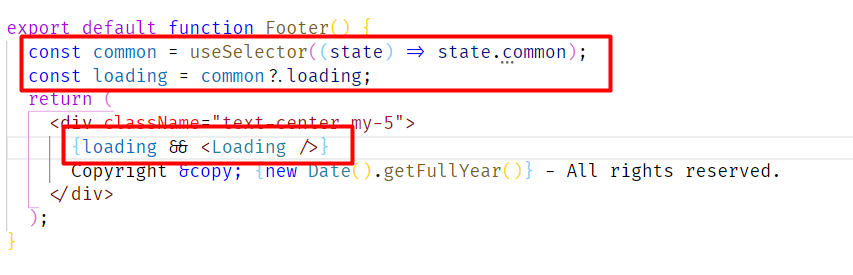
Hình 41. Tạo 2 case action.

Sau khi đã thêm xong commonReducer thì chúng ta sẽ đem xử lý này để gắn vào những nơi có gọi API để hiển thị loading trong thời gian gọi API



Hình 42. Dispatch action trong một function xử lý khác.

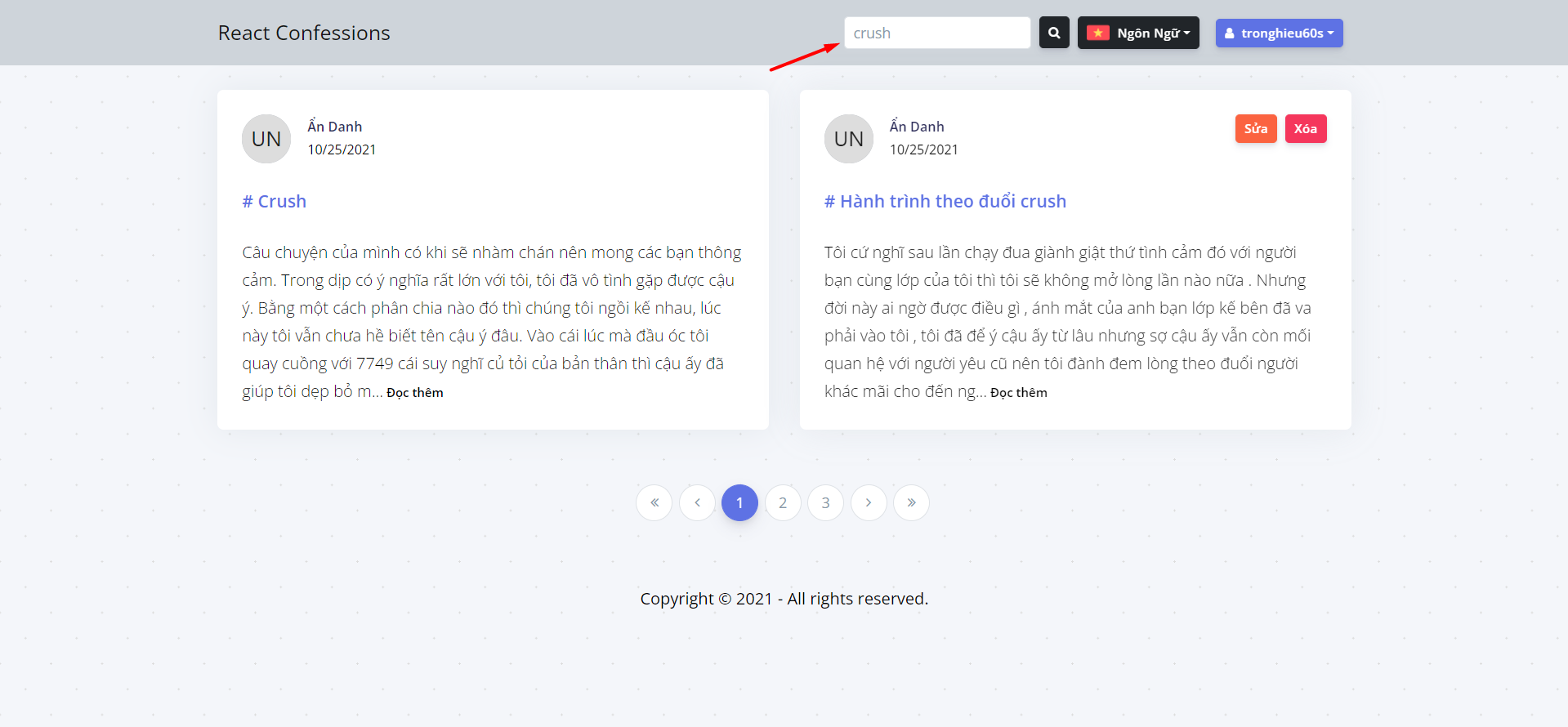
Khi đã gắn vào trong hàm call API thì sau đó chúng tra sẽ ghi làm để set trạng thái cho loading là gì để có thể hiển thị ra màn hình



Hình 43. Gọi giá trị loading từ Redux và sử dụng.

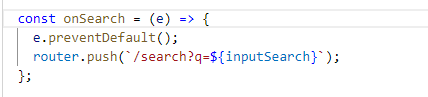
### 2.2.10. Tìm kiếm bài viết

Người dùng có thể tìm kiếm được cái bài viết trong ứng dụng nhanh chóng.



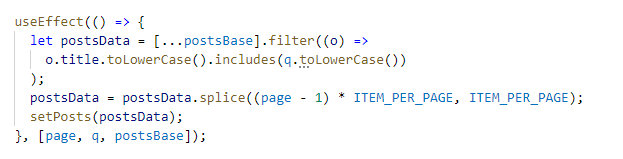
Hình 44. Giao diện sau khi tìm kiếm.

Khai báo nút Search:



Hình 45. Khai báo onSearch.

Khi Search thì màn hình sẽ chỉ hiển thị những mục mà ta tìm kiếm:



Hình 46. useEffect hiển thị mục tìm kiếm.

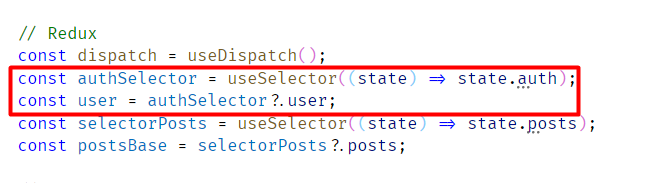
### 2.2.11. Trang chi tiết người dùng

Ở trang chi tiết người dùng, phần bên trái là hiển thị danh sách bài viết, cũng giống như trang chủ, ta chỉ cần lọc đi phần hiển thị cho người dùng hiện tại.



Hình 47. Lọc các bài viết của người dùng đã tạo.

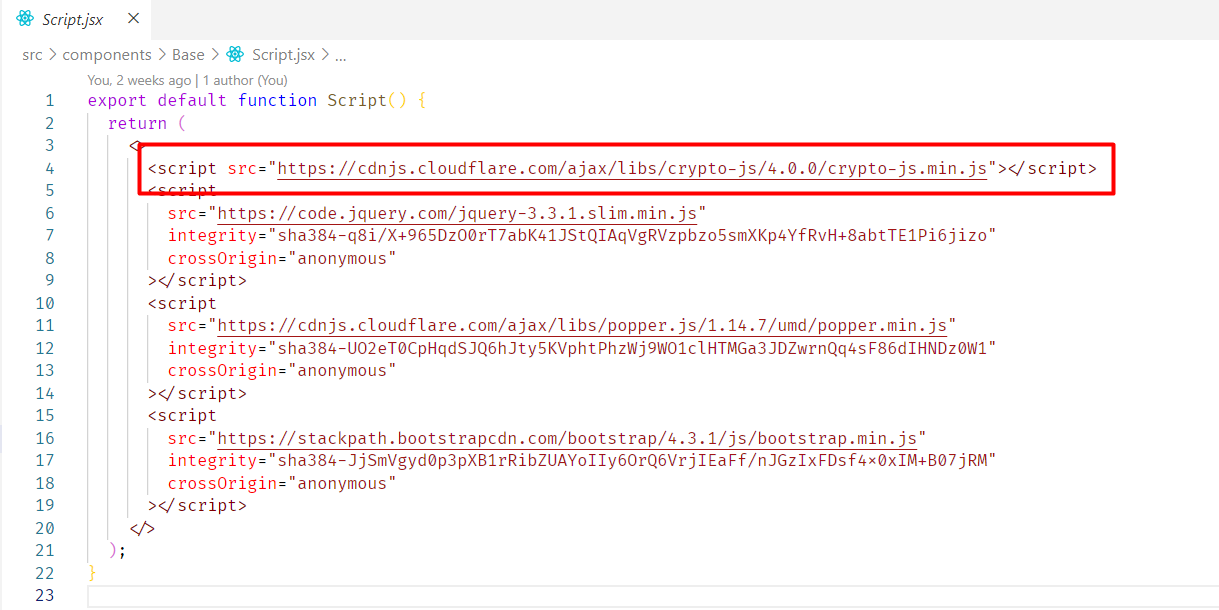
Để hiển thị thông tin cá nhân người dùng ra ta chỉ cần gọi redux mà đã lưu người dùng đăng nhập trước đó.



Hình 48. Sử dụng redux mà trước đó đã lưu thông tin người dùng.

Ảnh đại diện được sử dụng thông qua gravatar, ta chỉ cần lấy thông tin email người dùng hash md5 để lấy đường dẫn hình ảnh.

Để mã hóa md5, hãy nhúng một thư viện khác.



Hình 49. Nhúng thư viện mã hóa md5 vào dự án.

Sau đó ta sẽ hash email người dùng bằng useEffect hook, vậy là đã có đường dẫn hình ảnh rồi, chỉ cần hiển thị ra là xong.



Hình 50. Hiển thị hình ảnh người dùng sau khi hash ra mã md5.

### 2.2.12. Trang chi tiết bài viết (bài viết liên quan)

Khi người dùng click vào một bài viết để xem nội dung bài viết thì ở page này chúng ta sẽ có thêm một cột bài viết liên quan để có thể gợi ý thêm cho người đọc những bài viết khác.



Hình 51. Giao diện hiển thị của bài viết liên quan.

Để làm bài viết liên quan đầu tiên ta phải viết một function để trộn thứ tự hiển thị của các bài viết.



Hình 52. Function trộn thứ tự của các bài viết.

Sau khi tạo xong hàm trộn các bài viết thì chúng ta sẽ khai báo useState và dùng Redux và useEffect để có thể sử dụng và hiển thị nó ra màn hình.



Hình 53. Khai báo và gọi hàm bài viết liên quan.

Sau cùng khi đã gọi hàm, khai báo và xử lý ở thằng cha xong việc cuối cùng là mình sẽ cho thằng con kế thừa thằng cha là mọi chuyện đã xong.



Hình 54. Kế thừa từ thằng cha của bài viết liên quan.

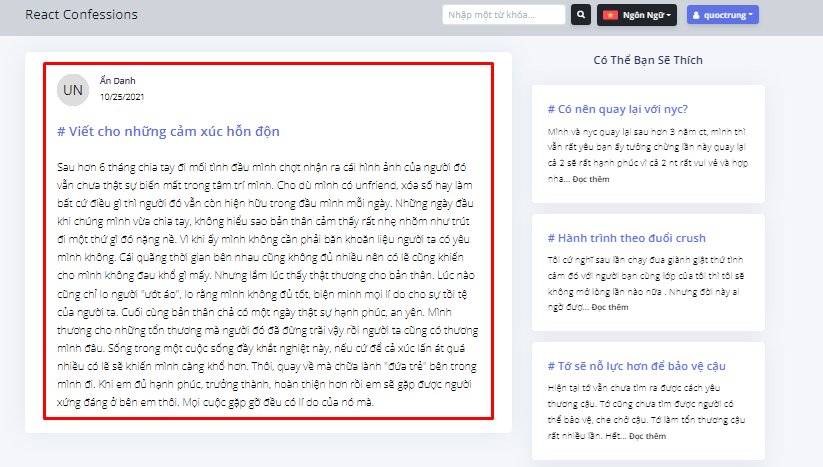
### 2.2.13. Trang chi tiết bài viết (nội dung)

Muốn xem đầy đủ nội dung bài viết nào thì ta cần click vào tiêu đề của bài viết đó:



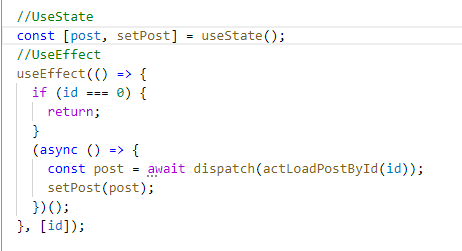
Hình 55. Click vào tiêu đề bài viết.

Sau khi click vào tiêu đề, trang sẽ chuyển sang giao diện mới:



Hình 56. Giao diện trang chi tiết bài viết.

Để hiện thị ra, ta cần dùng useEffect và useState:



Hình 57. useEffect và useState trang chi tiết bài viết.

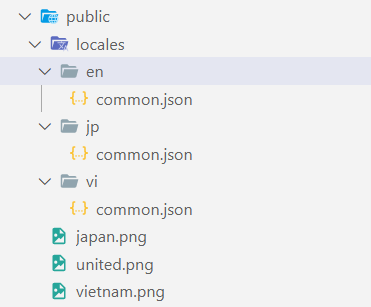
Sau đó gọi qua để hiển thị là xong:



Hình 58. HIển thị thông tin trong trang chi tiết bài viết.

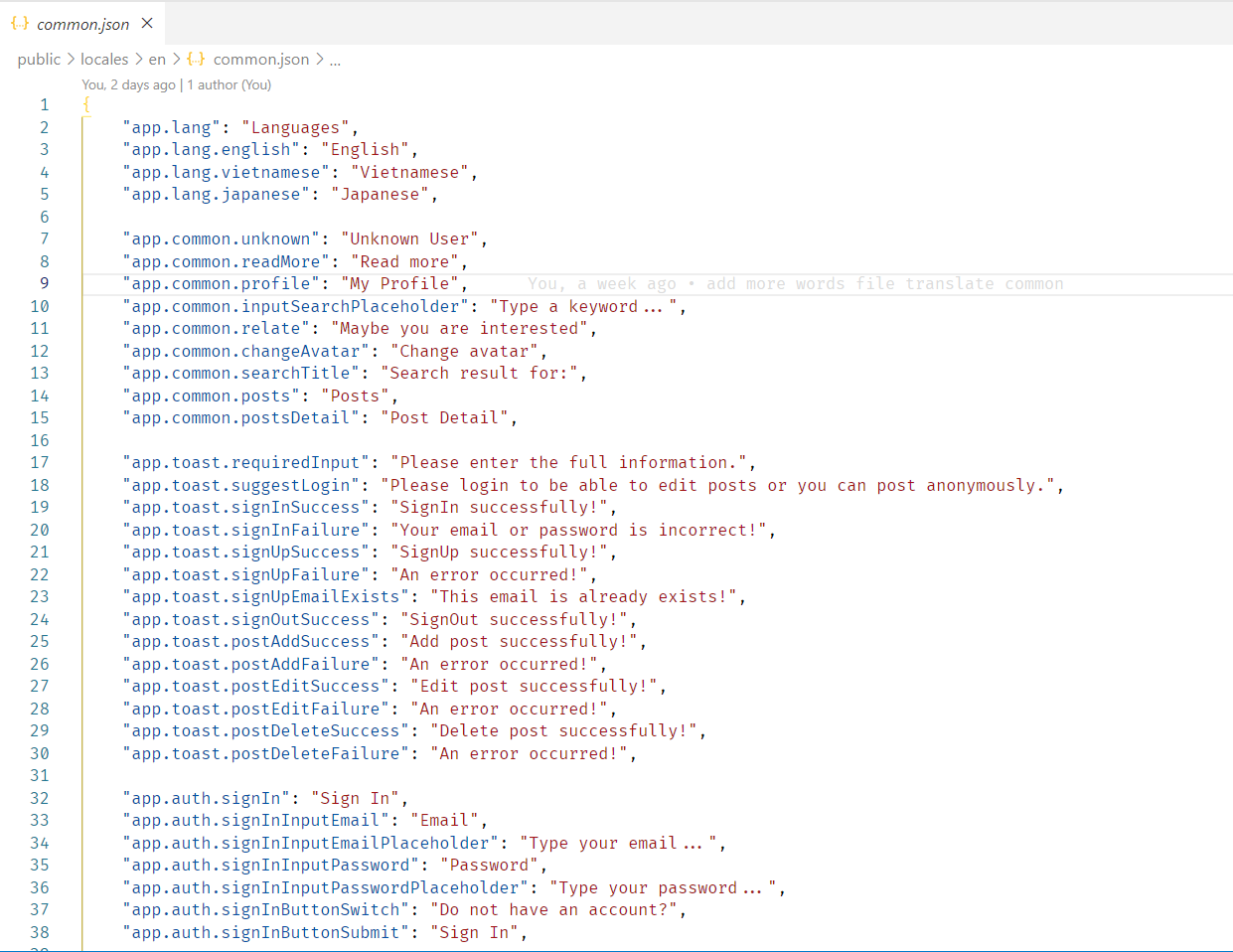
### 2.2.14. Đa ngôn ngữ

Ta cần chuẩn bị một số tài nguyên để hiển thị đa ngôn ngữ, thư mục ngôn ngữ chứa các loại ngôn ngữ khác nhau, ở đây ta có 3 ngôn ngữ Anh, Việt, Nhật. Ta cũng cần một vài hình ảnh để hiển thị.



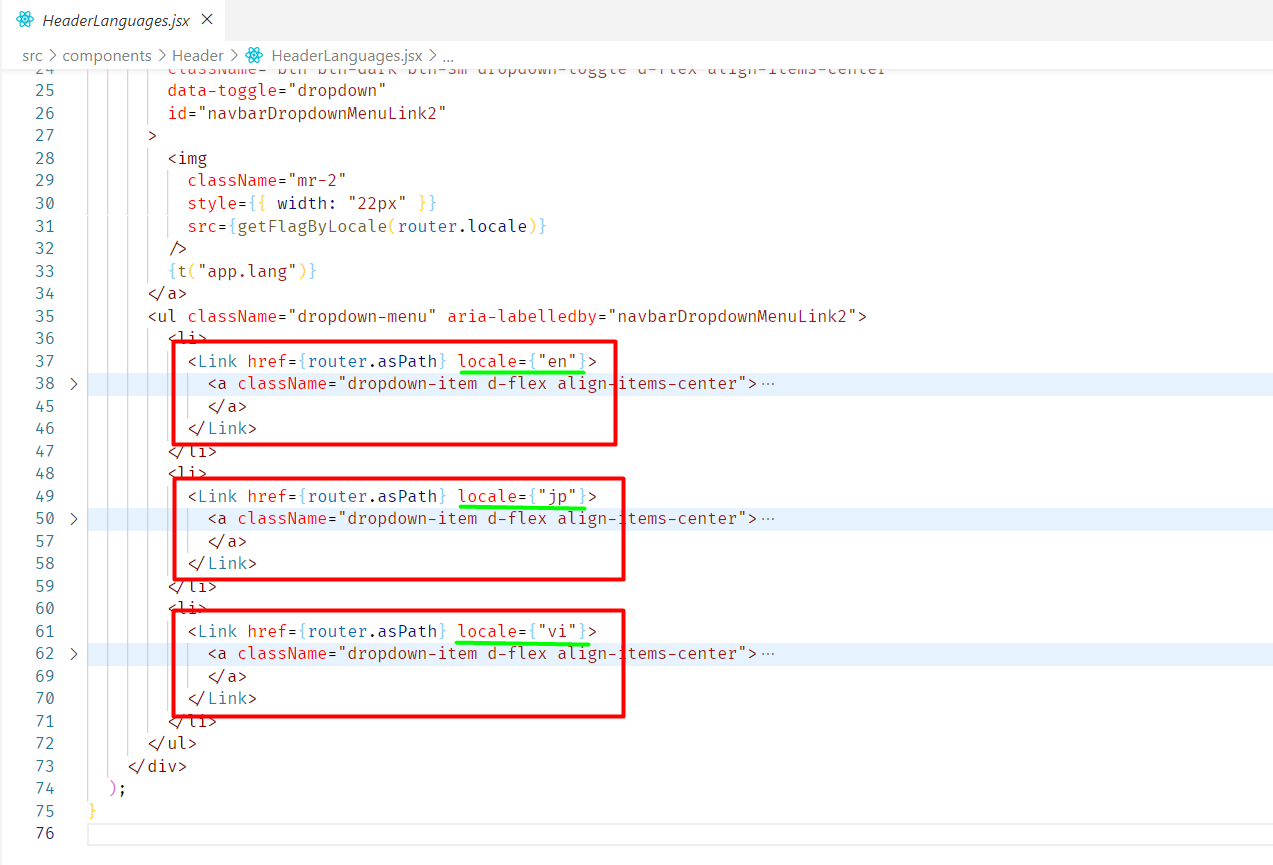
Hình 59. Thư mục chứa nội dung bản dịch và hình ảnh quốc kỳ.

Mỗi ngôn ngữ có file common chứa ngôn ngữ đó, ta sẽ cho hiển thị ngôn ngữ bằng mã code, ví dụ về một file như sau:



Hình 60. File nội dung của bản dịch Tiếng Anh.

Ở phần giao diện, ta cho hiển thị thẻ Link, thẻ này dùng để set lại locale của ứng dụng, từ đó có thể nhận biết được ngôn ngữ nào đang sử dụng.



Hình 61. Sử dụng thẻ Link của NextJS để set locale của ứng dụng.

Ở ứng dụng này ta sẽ sử dụng thư viện next-i18next để cài đặt hãy gõ câu lệnh sau: npm install –save-dev next-i18next

Sau khi cài đặt ta sẽ kiểm tra tại file package.json:



Hình 62. Kiểm tra thư viện đã cài đặt trong package.json.

Tại file next-i18next.config.js ta cấu hình như sau (nếu chưa có hãy tạo ở thư mục gốc):



Hình 63. Cấu hình tại file next-i18next.config.js.

Tiếp tục nhúng file config vào file next.config.js:



Hình 64. Import file cấu hình vào next.config.js.

Tại file pages/\_app.js tiếp tục cấu hình thêm appWithTranslation.



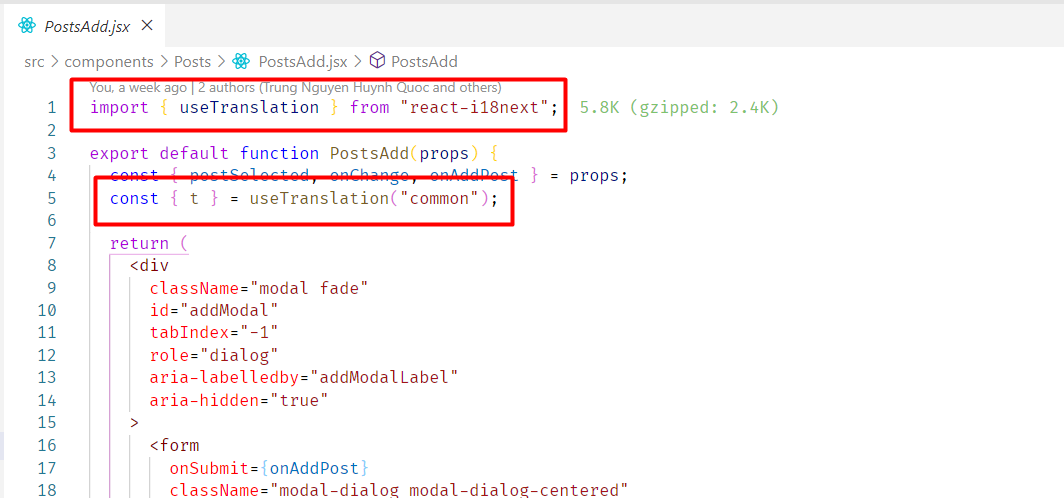
Hình 65. Thêm cấu hình vào file \_app.js.

Tại từng files từng trang trên folder pages cấu hình thêm serverSideTranslations.



Hình 66. Sử dụng serverSideTranslations cho từng trang.

Để sử dụng ta gọi hook useTranslation, và truyền vào tên file đã config trước đó.



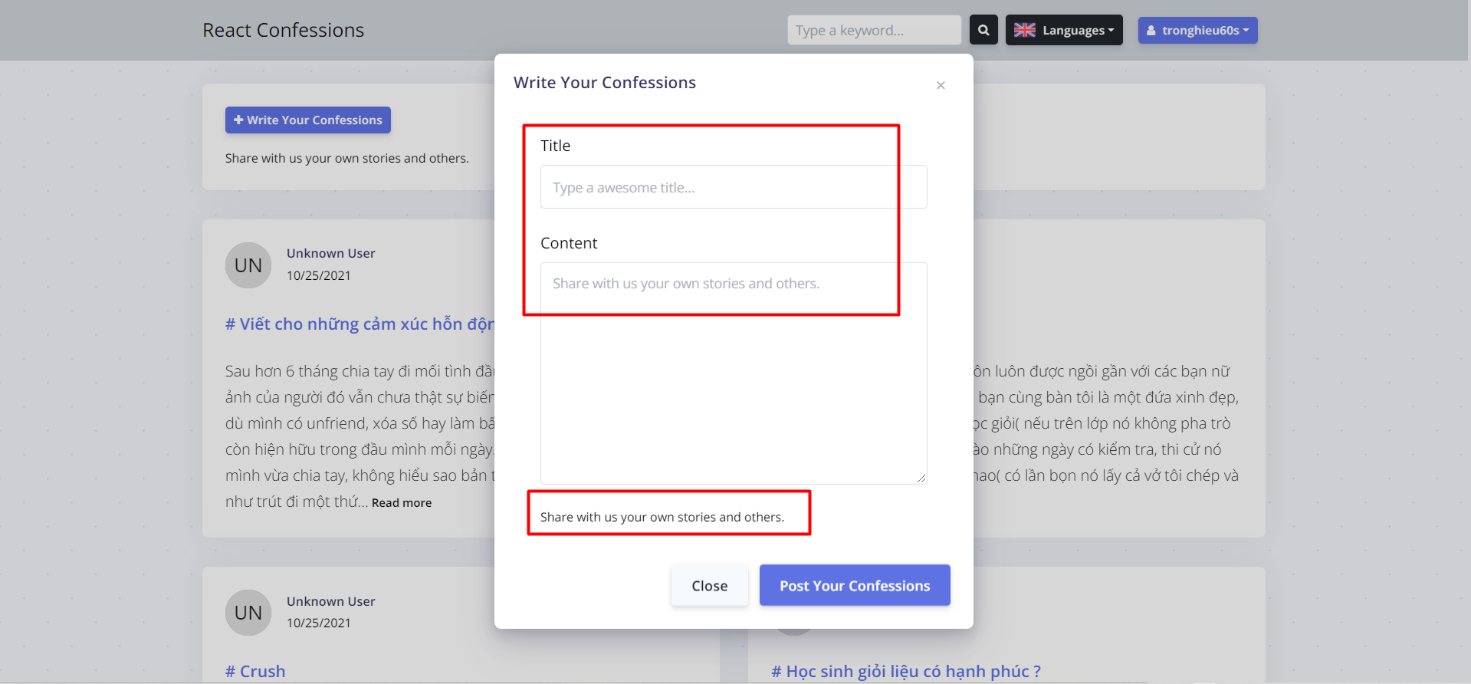
Hình 67. Sử dụng hook useTranslation để gọi bản dịch.

Để lấy nội dung, ta sẽ truyền mã nội dung mà ta cần hiển thị.



Hình 68. Để hiển thị ta chỉ cần gọi ra mã nội dung của nó.

Nội dung hiển thị đã xuất hiện, khi đổi ngôn ngữ, ta cũng sẽ lấy được ngôn ngữ đó.



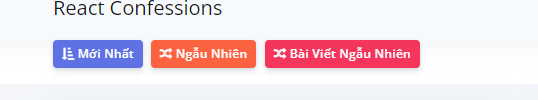
Hình 69. Nội dung được hiển thị bằng Tiếng Anh.



Hình 70. Nội dung được hiển thị bằng Tiếng Việt.

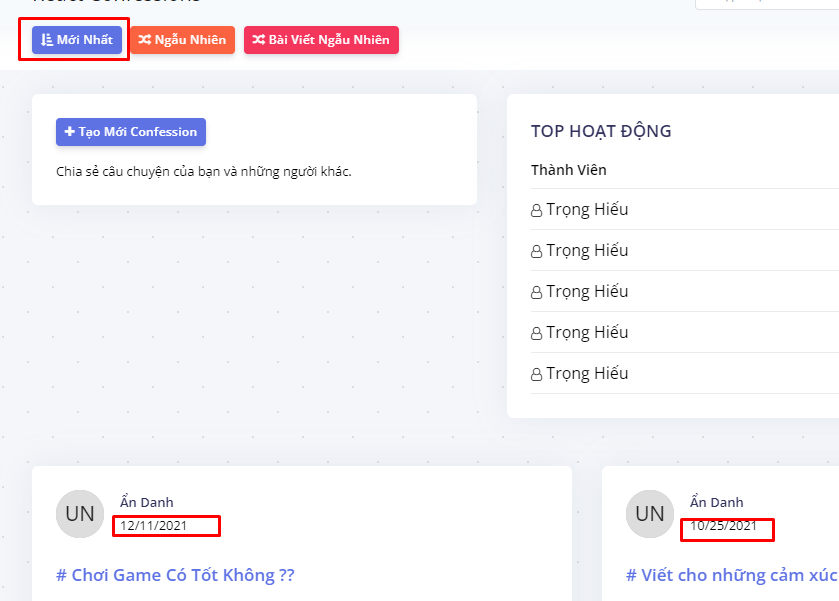
### 2.2.15. Lọc bài viết

Có 3 cách lọc bài viết: “Mới Nhất” “Ngẫu Nhiên” “Bài Viết Ngẫu Nhiên”



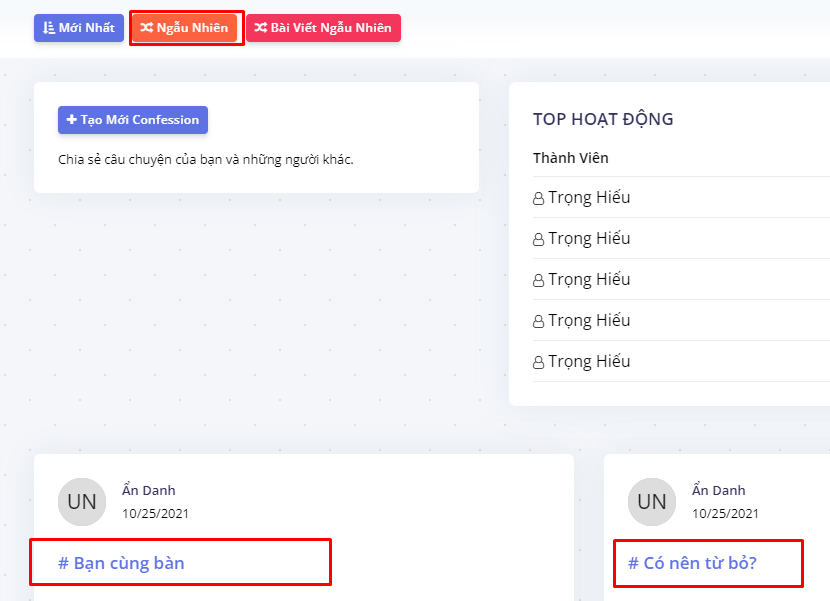
Hình 71. Lọc bài viết

Khi Click vào “Mới Nhất” thì trang sẽ load lại những bài viết mới nhất lên đầu:



Hình 72. Load bài viết mới nhất.

Khi Click vào “Ngẫu Nhiên” thì trang sẽ load lại những bài viết ngẫu nhiên:



Hình 73. Load bài viết ngẫu nhiên.

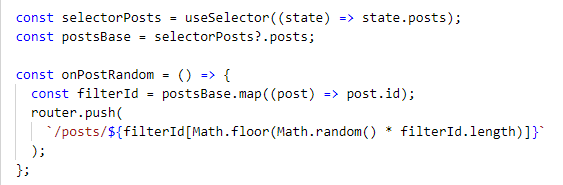
Muốn lọc bài viết như trên, chỉ cần đưa Link vào:



Hình 74. Code mới nhất và ngẫu nhiên.

Khi Click vào “Bài Viết Ngẫu Nhiên” thì sẽ chuyển sang trang chi tiết nội dung bài viết, tất nhiên là theo ngẫu nhiên.

Khai báo “onPostRandom”:



Hình 75. Khai báo hàm random.

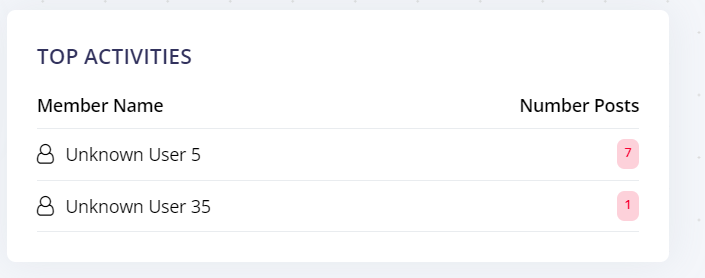
Gọi hàm để hiển thị ra màn hình:



Hình 76. Gọi hàm random.

### 2.2.16. Ranking user

Chúng ta sẽ tạo ra một bảng để hiển thị người có số lượng bài viết cao và đây là giao diện.



Hình 77. Nội dung được hiển thị của Ranking User.

Việc đầu tiên sẽ viết hàm để đếm số lượng bài viết và rút gắn lại tìm ra top 5 người có bài viết nhiều nhất.



Hình 78. Hàm đếm và Sort bài post của Ranking User.

Sau khi chúng ta ghi xong hàm ở trên thì chúng ta sẽ gọi hàm ở PostsUserRanking.jsx và đồng thời sẽ khỏi tạo biến để dùng và gán nó vào hàm trên.



Hình 79. Nơi được gọi hàm countAndSort.

Sau khi thêm và xử lý PostsUserRanking.jsx thì chúng ra sẽ gọi hàm ra lớp cha để nó có thể hiển thị ra giao diện.



Hình 80. Ở lớp cha gọi PostsUserRanking.

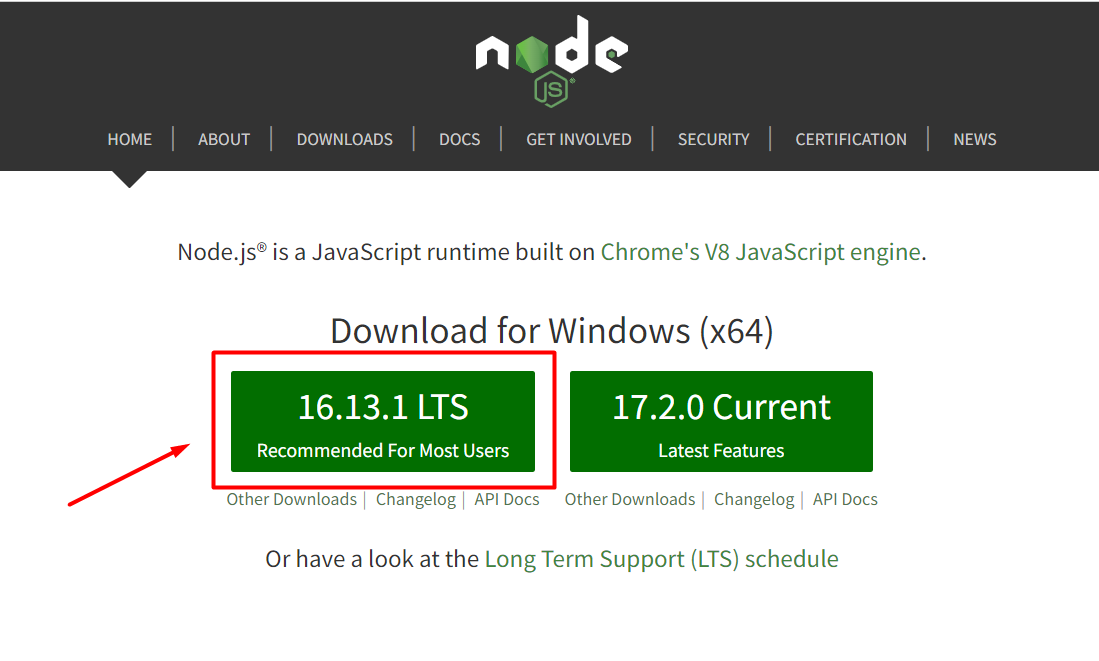
## 2.3. Hướng Dẫn Thêm Về Cấu Hình Dự Án

### 2.3.1. Hướng dẫn cài đặt NodeJS.

Đầu tiên để chạy được NextJs ta cần khởi tạo môi trường NodeJS.

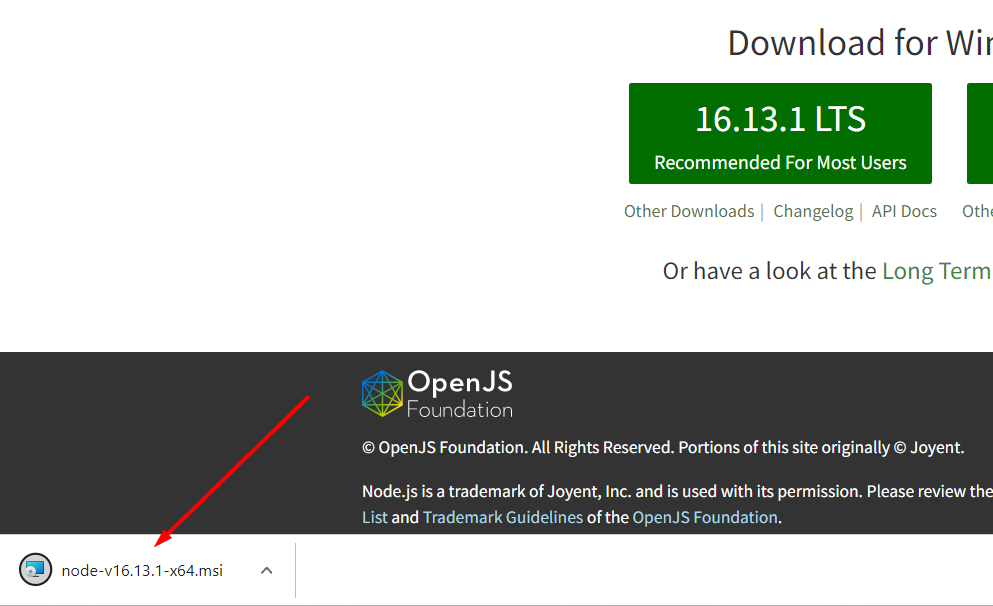
Tải và cài đặt NodeJS tại địa chỉ: <https://nodejs.org/en/>

Click vào nút dưới để tải bản LTS.



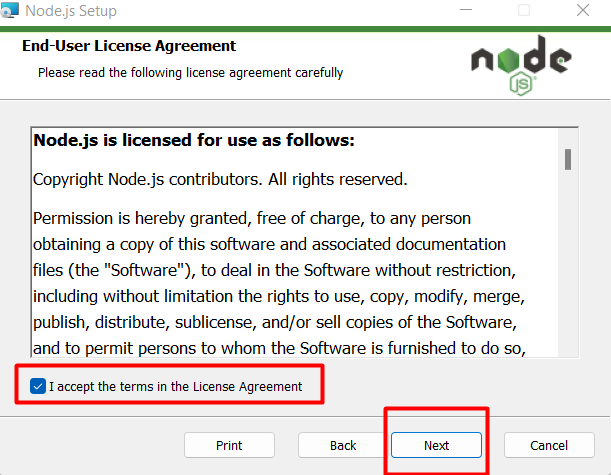
Hình 81. Phiên bản tải nodejs

Click vào bên dưới để cài đặt NodeJS.

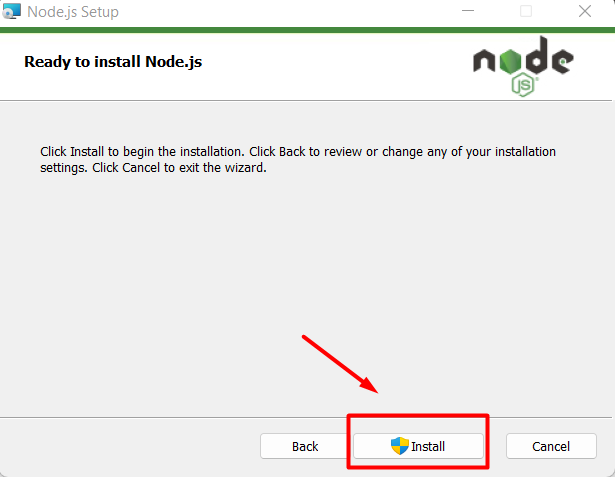


Hình 82. Tải về máy gói install

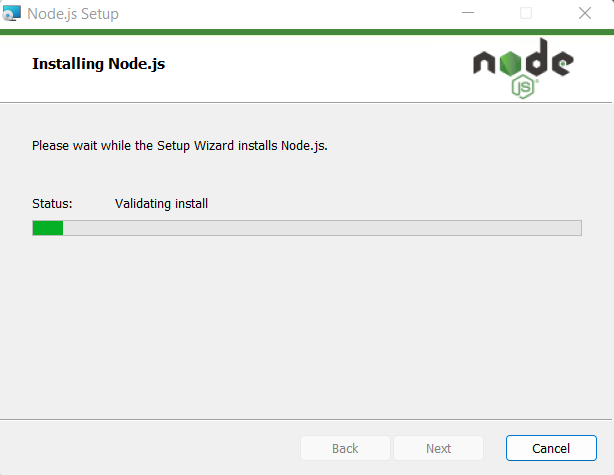
Ta tiếp tục next cho đến khi quá trình cài đặt kết thúc:



Hình 83. Chấp nhận điều kiện của nodejs



Hình 84. Install Nodejs vào máy

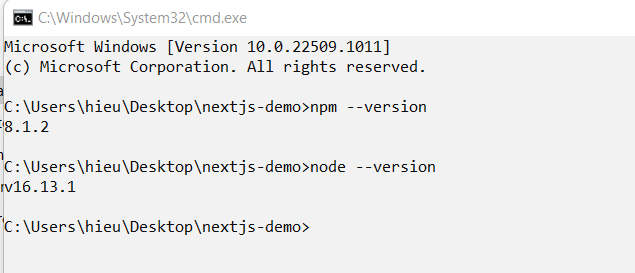


Hình 85. Thời gian chờ install nodejs

Để kiểm tra quá trình cài đặt có thành công hay không, hãy chạy 2 câu lệnh sau:

npm –version

node –verison

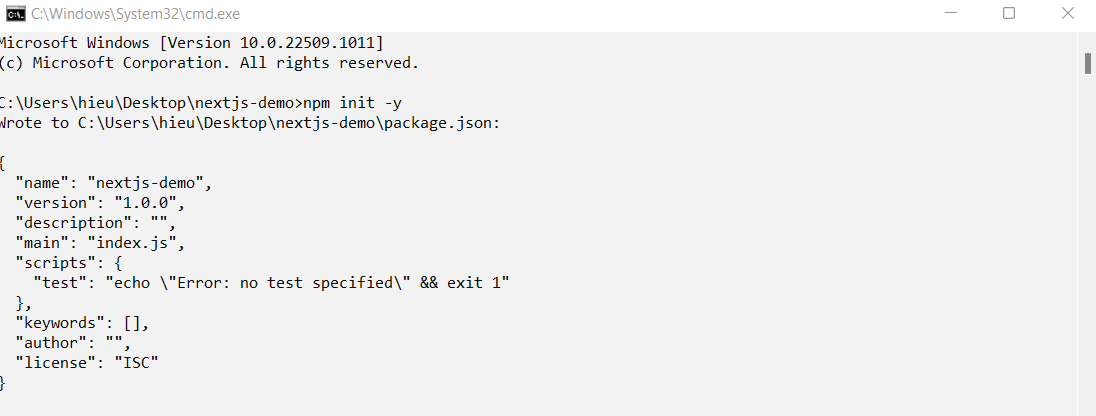


Hình 86. Kiểm tra version và kiểm tra xem đã thành công chưa

### 2.3.2. Hướng dẫn cài đặt NextJS.

Đầu tiên mình tạo thư mục nextjs-demo. Mở terminal và thực hiện như sau.

Ta sử dụng câu lệnh npm init -y để khởi tạo dự án.

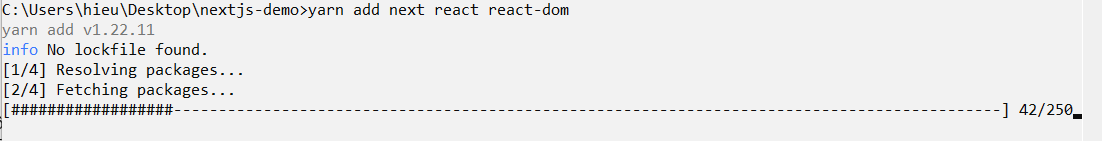


Hình 87. Khởi tạo dự án với npm.

Tài liệu cài đặt nextjs được mô tả chi tiết ở đây bạn có thể làm theo bên dưới hoặc truy cập để cài đặt: <https://nextjs.org/docs/getting-started>

Ở đây mình sử dụng yarn, nếu chưa cài, bạn hãy chạy câu lệnh npm install -g yarn.

Tiếp tục chạy câu lệnh sau để cài thư viện: yarn add next react react-dom



Hình 88. Cài đặt các thư viện để cấu hình NextJS.

Câu lệnh sẽ cài đặt 3 thư viện chính:

* next: thư viện chính thức của nextjs dùng để chạy môi trường nextJs.
* react: thư viện của react.
* react-dom: thư viện xử lý dom của react.

Tiếp tục thực hiện các bước tiếp theo.

Mở file package.json và thêm nội dung sau vào scripts để có thể chạy NextJs.

"scripts": {

    "dev": "next dev",

    "build": "next build",

    "start": "next start",

    "lint": "next lint"

},



Hình 89. Thêm scripts vào package.json để chạy NextJS.

Giải thích các câu lệnh chính:

* dev: khởi chạy dự án môi trường develop, các thao tác sửa code sẽ được hiển thị ngay lập tức.
* build: xây dựng dự án để triển khai production.
* start: chạy dự án với môi trường production, dữ liệu đã được lệnh build xây dựng sẽ không thể chỉnh sửa.

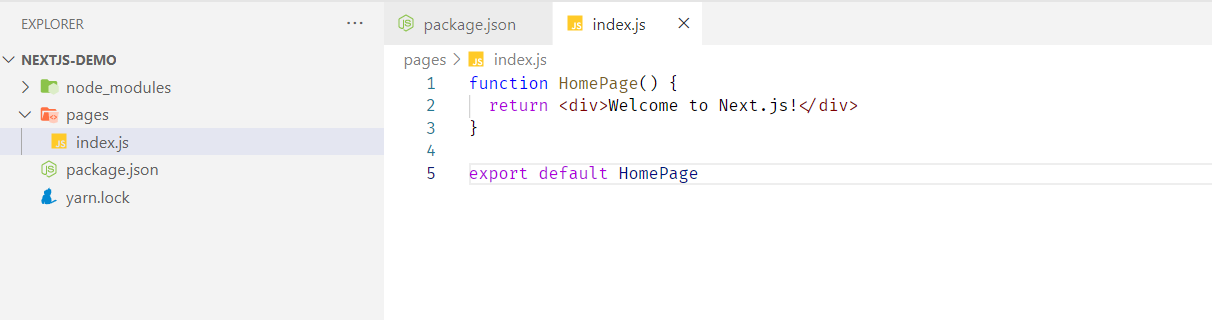
Để chạy dự án đầu tiên thêm một file index.js vào thư mục pages với nội dung như sau:

function HomePage() {

  return <div>Welcome to Next.js!</div>

}

export default HomePage



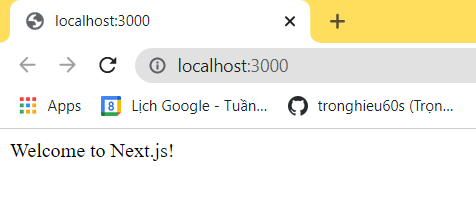
Hình 90. Thêm nội dung giao diện để chạy dự án.

Để chạy dự án, ta chỉ cần gọi câu lệnh yarn dev trong command lines thôi:



Hình 91. Ta gọi câu lệnh yarn dev để chạy server.

Mở trang <http://localhost:3000> ta đã thấy NextJs đã hoạt động.



Hình 92. Giao diện sau khi chạy lệnh.

### 2.3.3. Hướng dẫn cài đặt Redux.

Để cài đặt redux, ta thực hiện cài đặt các thư viện sau, chạy câu lệnh yarn add sau đây để tiếp tục: yarn add redux react-redux redux-devtools-extension next-redux-wrapper redux-thunk



Hình 93. Cài đặt các thư viện cho Redux.

Trong lúc chờ cài đặt, ta sửa file pages/index.js để tạo giao diện một chút, ta sẽ tạo một giao diện đơn giản cộng trừ một số bằng redux:

function HomePage() {

  return (

    <div>

      <p>Welcome to Next.js!</p>

      <div>1</div>

      <button>+</button>

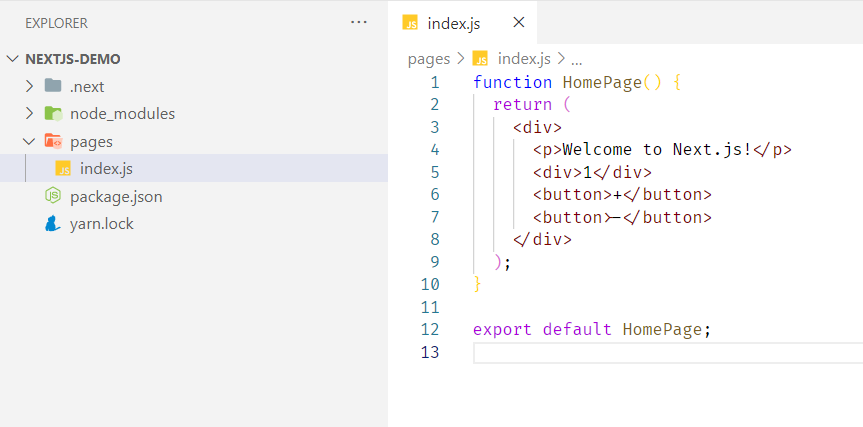
      <button>-</button>

    </div>

  );

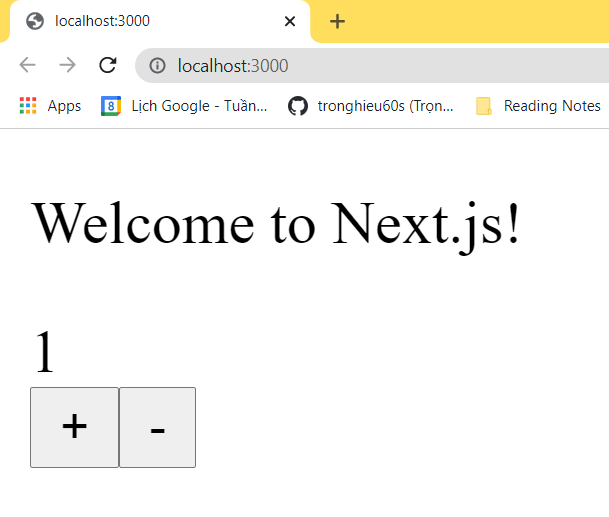
}

export default HomePage;



Hình 94. Chỉnh sửa lại giao diện cho counter redux.

Giao diện sau khi chỉnh sửa sẽ như thế này.



Hình 95. Giao diện có được sau khi chỉnh sửa.

Ta thực hiện tạo cấu trúc thư mục như sau, và tại file rootReducer ta thêm nội dung:

import { combineReducers } from "redux";

import counterReducer from "./counterReducer";

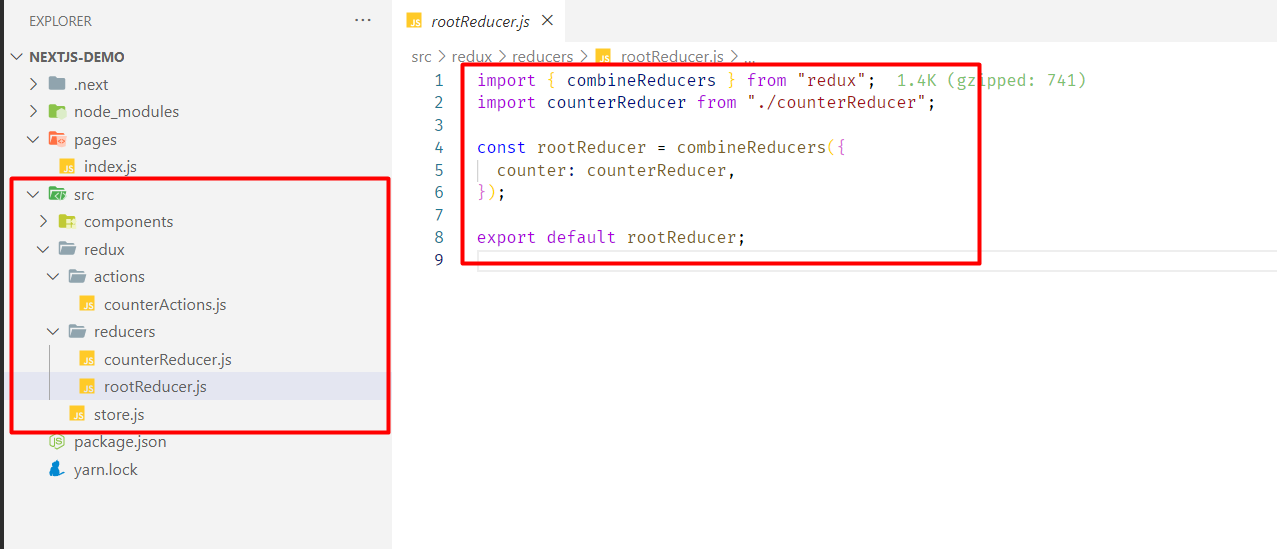
const rootReducer = combineReducers({

  counter: counterReducer,

});

export default rootReducer;

Khi này reducer của couter đã được import vào rootReducer. Sau này nếu có reducer nào bạn chỉ cần import vào là xong.



Hình 96. Cấu hình thư mục và import counter vào rootReducer.

Tại couterReducer ta sẽ có nội dung như sau:

import {

  DECREMENT\_COUNTER,

  INCREMENT\_COUNTER,

} from "../actions/counterActions";

const counterReducer = (state = { value: 0 }, action) => {

  switch (action.type) {

    case INCREMENT\_COUNTER:

      return { ...state, value: state.value + 1 };

    case DECREMENT\_COUNTER:

      return { ...state, value: state.value - 1 };

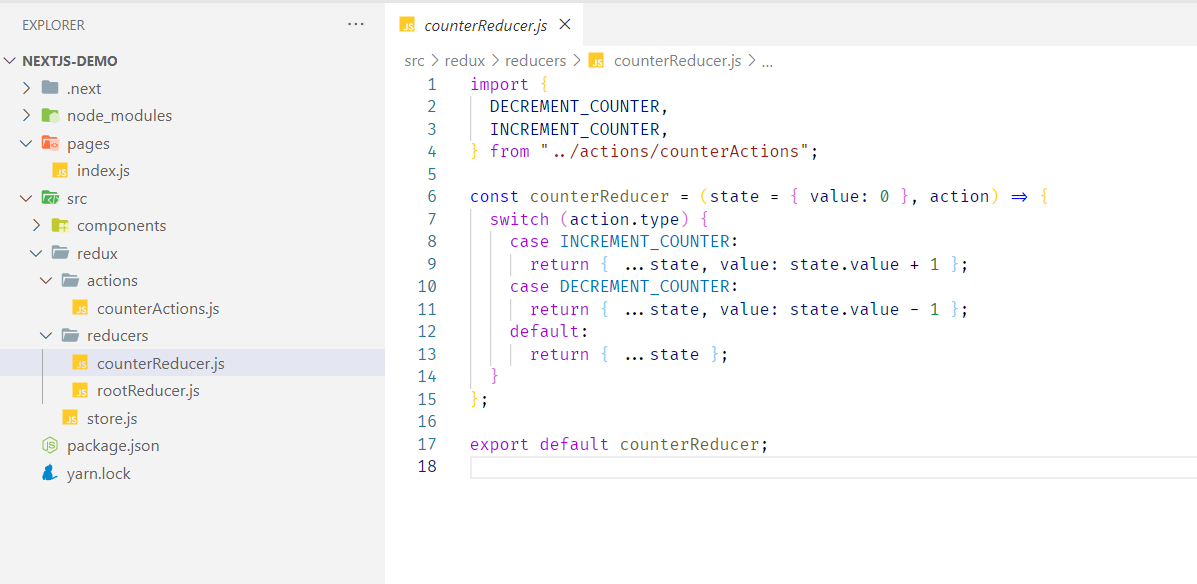
    default:

      return { ...state };

  }

};

export default counterReducer;



Hình 97. Cấu hình file counterReducer.

Ở đây ta switch case đến 2 actions của redux để thực hiện hành động là INCREMENT\_COUNTER, DECREMENT\_COUNTER.

Tiếp tục tại file couterAction ta định nghĩa 2 actions này và function cần thiết để gọi nó.

export const INCREMENT\_COUNTER = "INCREMENT\_COUNTER";

export const DECREMENT\_COUNTER = "DECREMENT\_COUNTER";

export const incrementCounter = () => ({

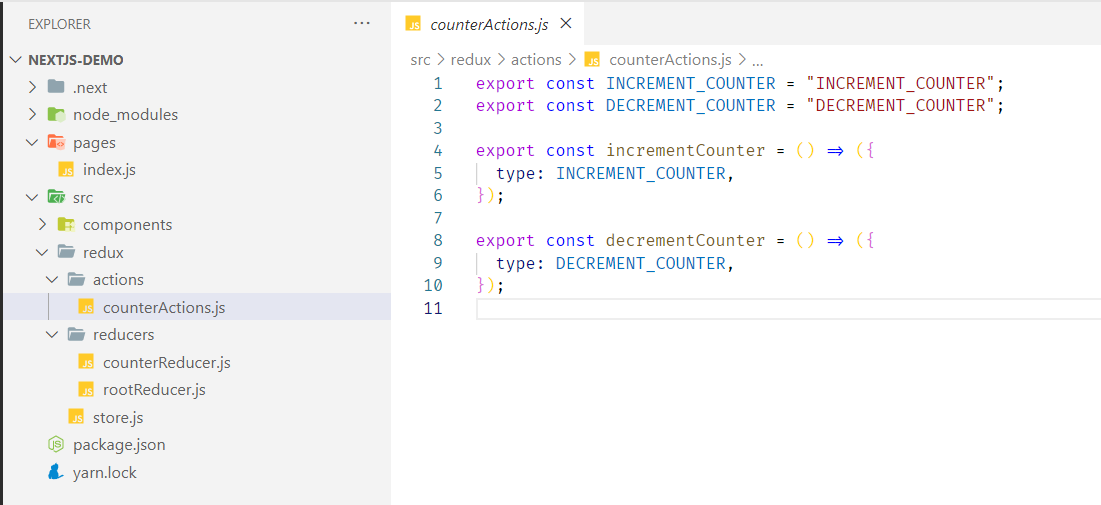
  type: INCREMENT\_COUNTER,

});

export const decrementCounter = () => ({

  type: DECREMENT\_COUNTER,

});



Hình 98. Cấu hình file counterActions.

Hai function incrementCounter, decrementCounter sau này sẽ được gọi để thực hiện hành động đến reducer, nên ta chủ yếu sẽ dùng 2 function này.

Tại file store.js, đây là file chính mà ta sẽ gọi ra ngoài để cấu hình, nội dung files này sẽ cấu hình redux devtool để debug trên chrome, cấu hình redux thunk để gọi async function, next redux wrapper để cấu hình riêng redux cho next.

import { createWrapper, HYDRATE } from "next-redux-wrapper";

import { applyMiddleware, createStore } from "redux";

import { composeWithDevTools } from "redux-devtools-extension";

import thunkMiddleware from "redux-thunk";

import rootReducer from "./reducers/rootReducer";

const bindMiddleware = (middleware) => {

  if (process.env.NODE\_ENV !== "production")

    return composeWithDevTools(applyMiddleware(...middleware));

  return applyMiddleware(...middleware);

};

const reducer = (state, action) => {

  if (action.type === HYDRATE) {

    const nextState = {

      ...state, // use previous state

      ...action.payload, // apply delta from hydration

    };

    return nextState;

  }

  return rootReducer(state, action);

};

const initStore = () => {

  return createStore(reducer, bindMiddleware([thunkMiddleware]));

};

export default createWrapper(initStore);



Hình 99. Cấu hình file store.

Tại file pages/\_app.js ta thực hiện wrapper redux vào nextjs, vậy là quá trình cấu hình đã hoàn tất.

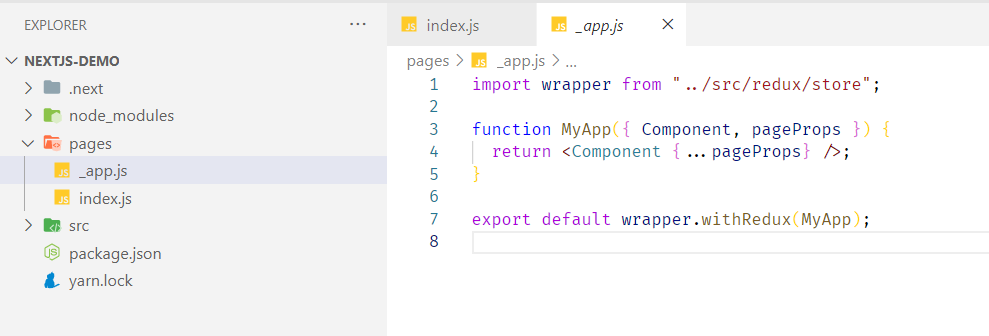
import wrapper from "../src/redux/store";

function MyApp({ Component, pageProps }) {

  return <Component {...pageProps} />;

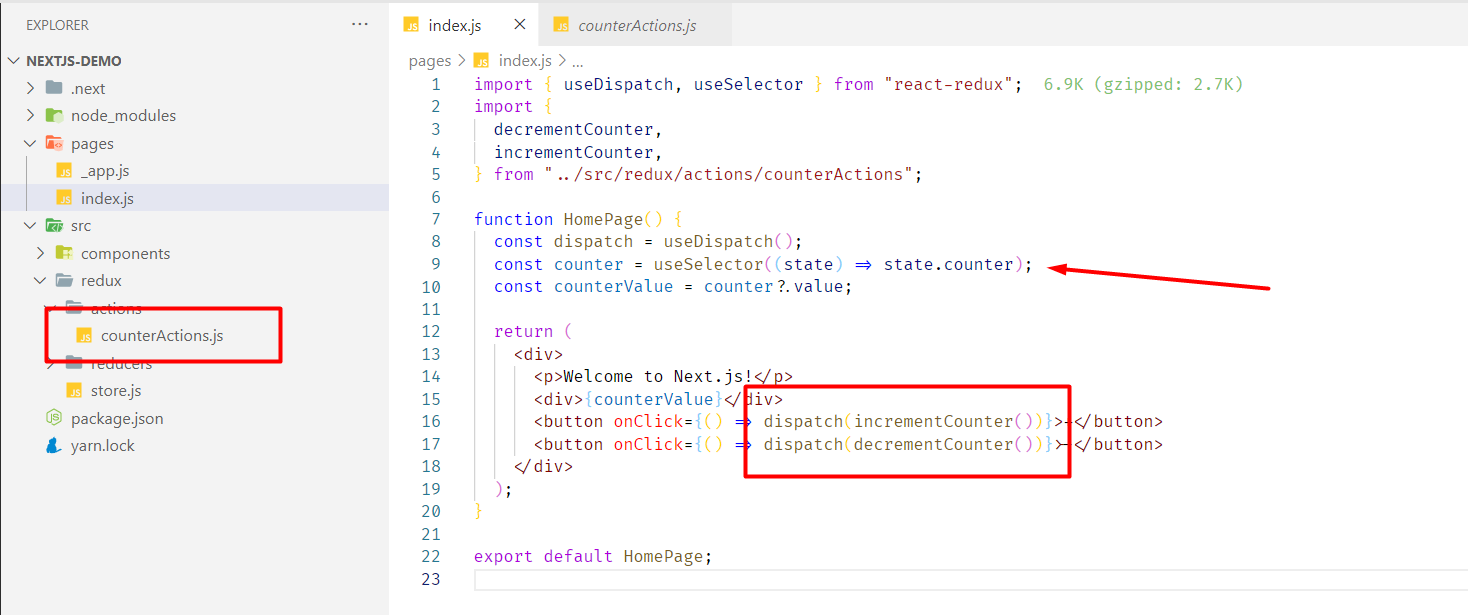
}

export default wrapper.withRedux(MyApp);



Hình 100. Đưa file store vào file gốc của NextJS để hoàn tất.

Để sử dụng ta thực hiện sử dụng useSelector để lấy giá trị của counter, ta sử dụng useDispatch để gọi function xử lý từ file actions.



Hình 101. Ta gọi useSelector và useDispatch đến counter redux.

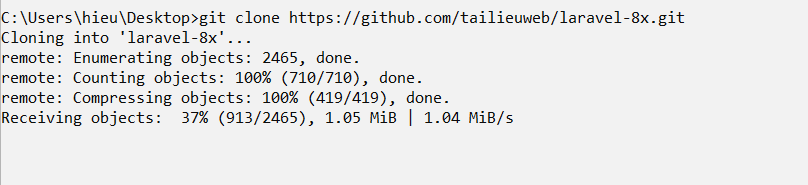
Và sau đây là thành quả:



Hình 102. Giao diện sau khi cấu hình redux đã hoạt động.

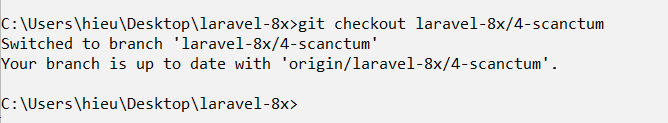
### 2.3.3. Hướng dẫn cài đặt API Scanctum.

Clone dự án từ trang web: <https://github.com/tailieuweb/laravel-8x/>



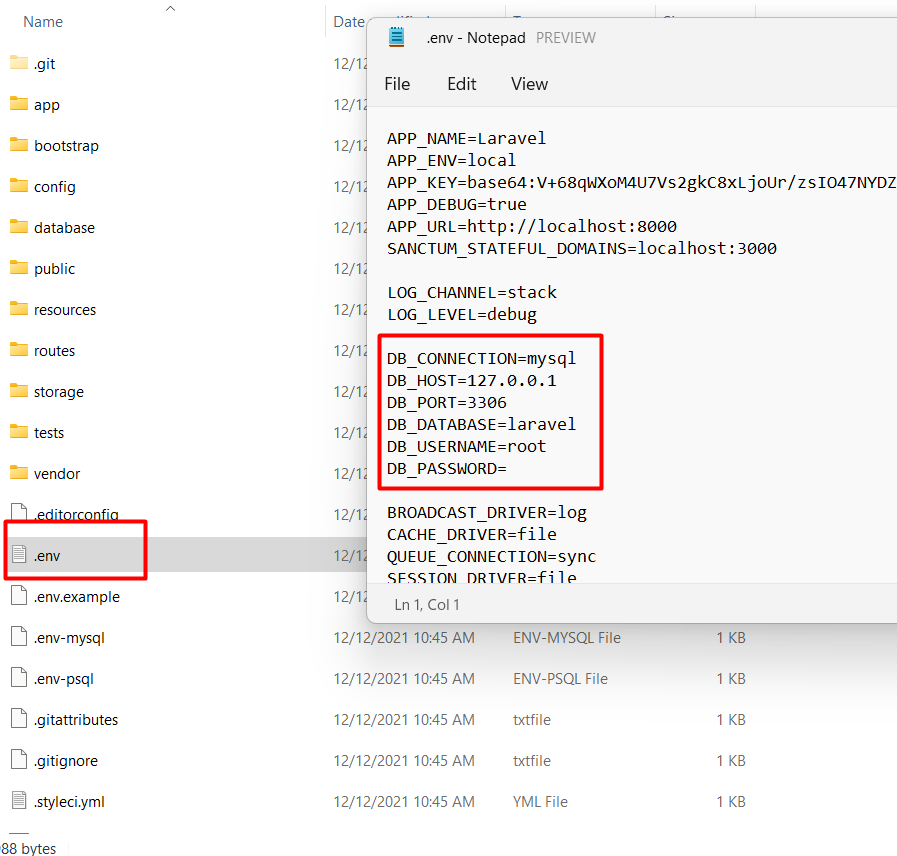
Hình 103. Clone code API

Ta tiếp tục tiến hành checkout sang nhánh laravel-8x/4-scanctum.



Hình 104. Check out nhánh laravel-8x/4-scanctum

Cấu hình database tại file .env của dự án:

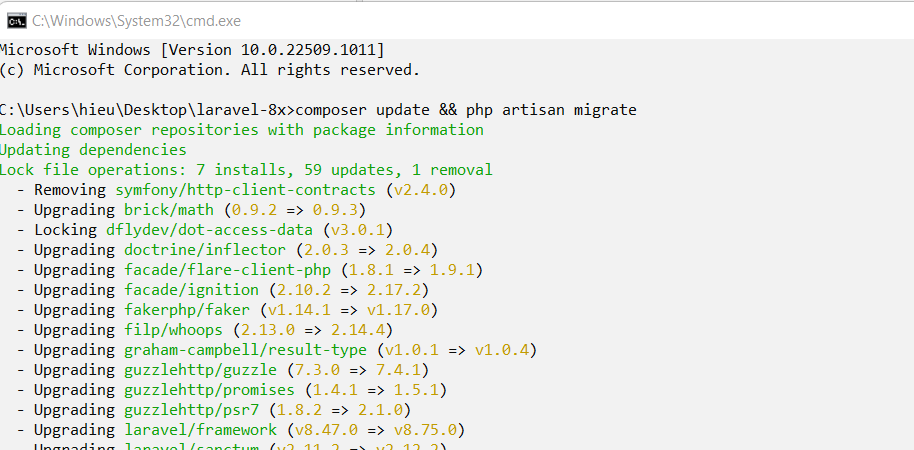


Hình 105. Cấu hình lại file môi trường ở laravel-8x

Tại thư mục gốc của scanctum, ta chạy file sau: 1-clear.bat

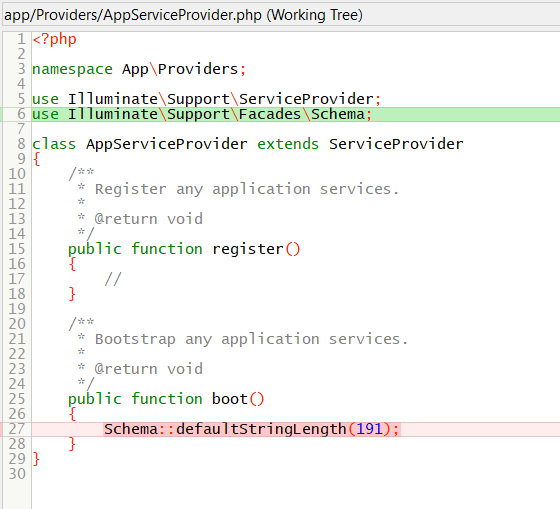
Tiếp tục chạy câu lệnh sau để cài đặt thư viện cho scanctum (lưu ý trước khi chạy hãy cài đặt [composer](https://getcomposer.org/) để chạy php và bật web servies (xampp, wamp)):

composer update && php artisan migrate



Hình 106. Update lại compser bằng cmd

Nếu chạy không thành công hãy thêm giống như sau vào file: app\Providers\AppServiceProvider.php



Hình 107. Thêm Schema vào để hoạt động nếu ko chạy được

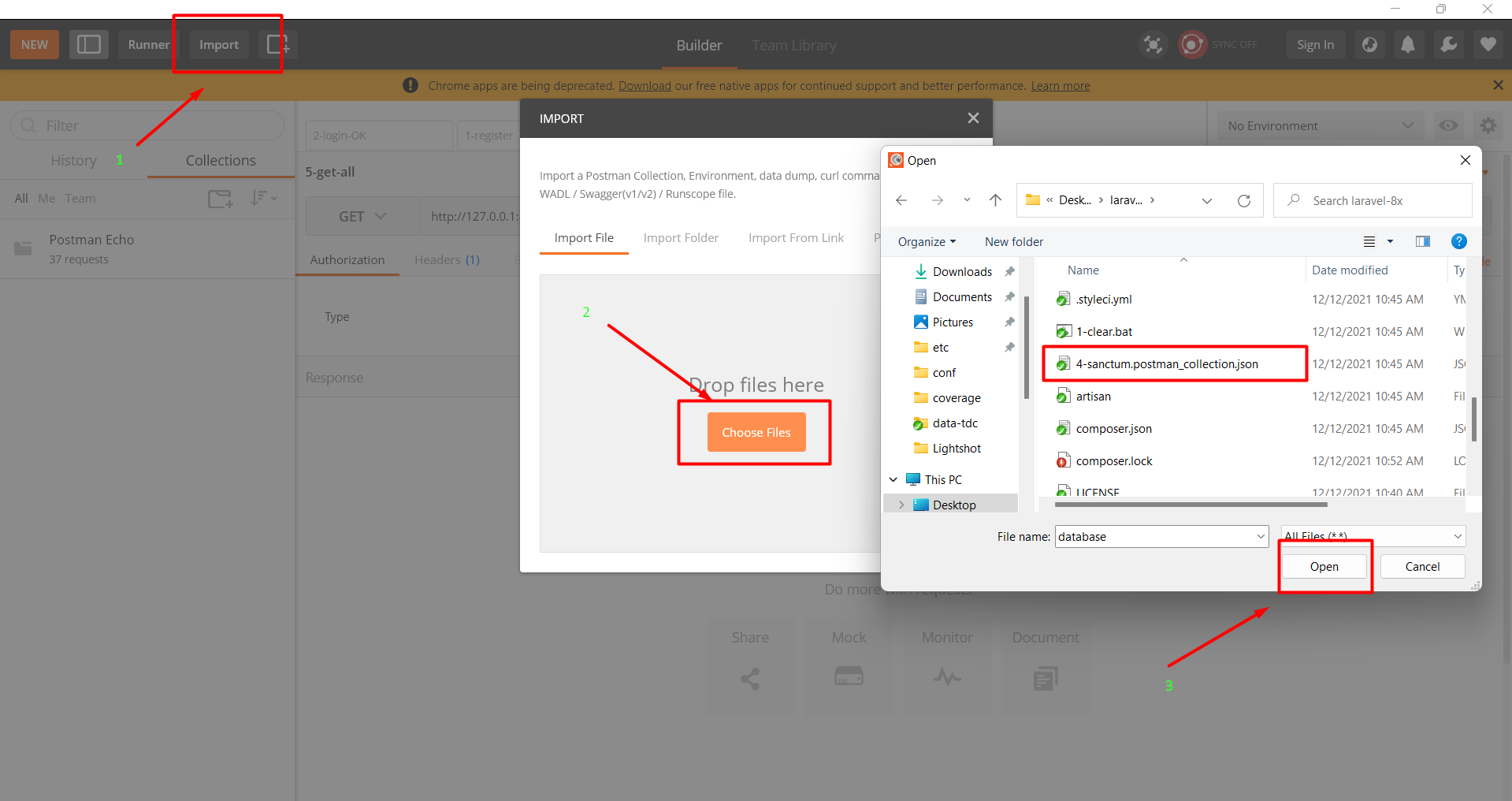
Để chạy server ta hãy gọi câu lệnh: php artisan serve



Hình 108. Start serve của php API

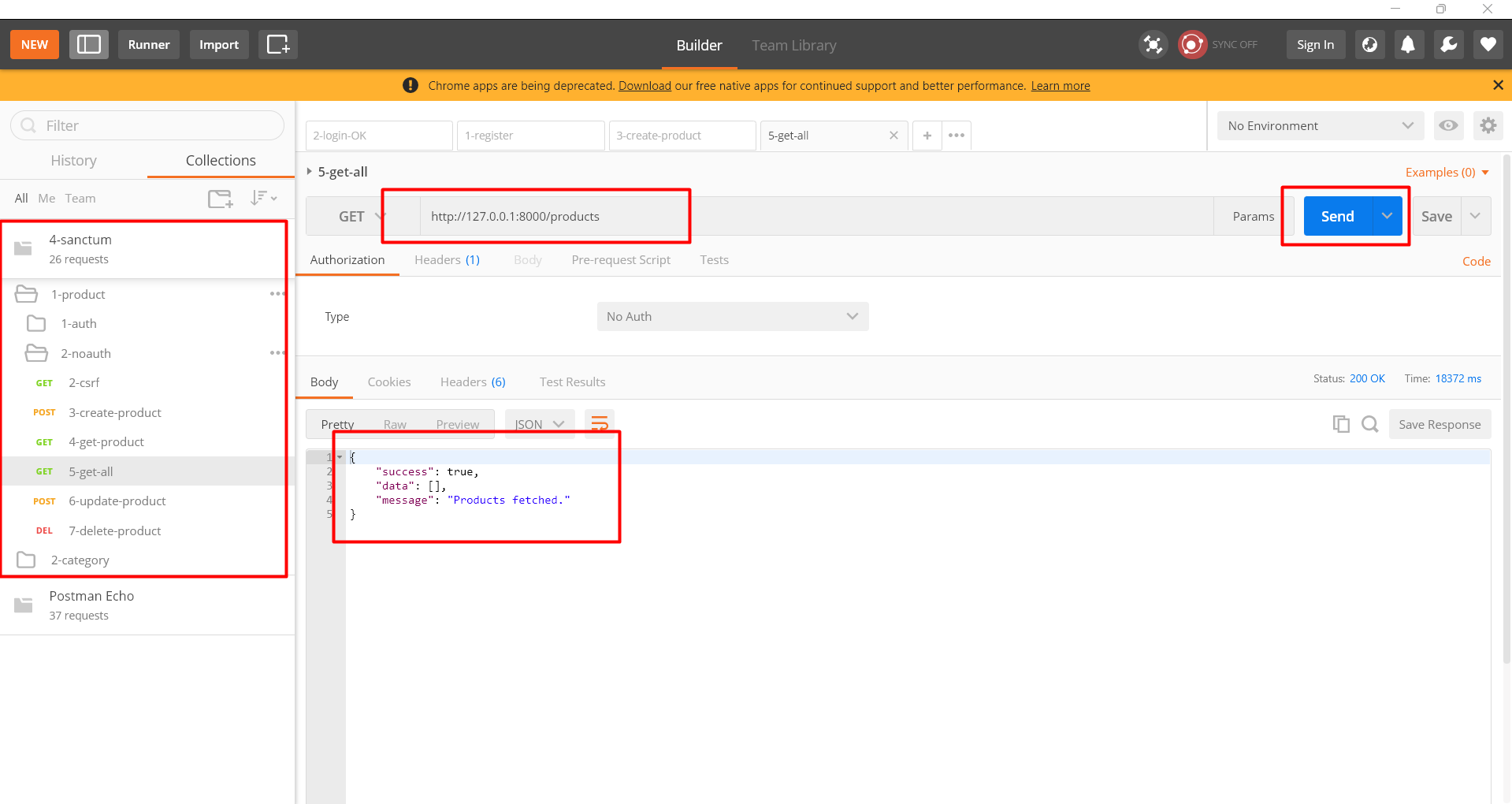
Để thực hiện test API, ta cài đặt một ứng dụng lên máy là postman tải nó tại đây: <https://www.postman.com/downloads/>

Sau khi tải và cài đặt ta có giao diện như sau, ta cũng tiến hành import api example của scanctum vào dự án:



Hình 109. Import api để hiển thị giao diện quản lý của Postman

Để sử dụng, ta chỉ cần coi trong danh sách API và chọn ra cái cần thiết để gọi, ở đây mình gọi API products, nó sẽ hiển thị ra được như sau:



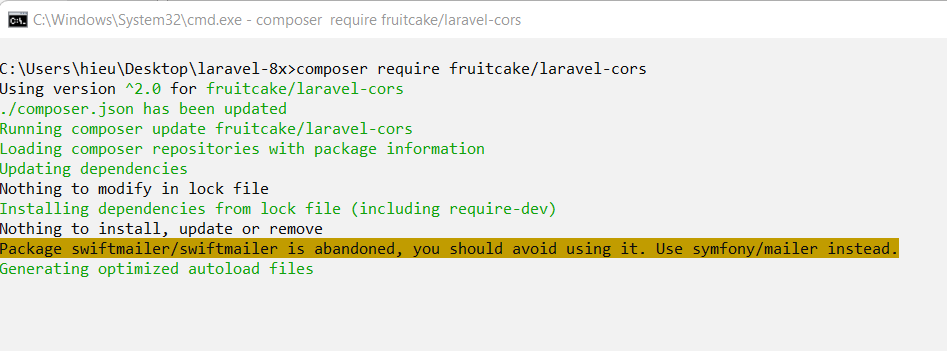
Hình 110. Chạy thử API xem có hoạt động được không

### 2.3.3. Hướng dẫn tắt CORS API Scanctum.

Lưu ý:

* Chỉ tắt CORS ([Cross Origin Resource Sharing](https://viblo.asia/p/cors-la-gi-Qbq5Q0j3lD8)) trong quá trình phát triển và không có nhu cầu bảo mật,
* Không nên tắt CORS vì có nguy cơ mất dữ liệu từ các trang web bên ngoài.

Chạy câu lệnh sau để cài đặt laravel-cors: composer require fruitcake/laravel-cors



Hình 111. Hình chạy câu lệnh composer require fruitcake/laravel-cors

Tại file config\cors.php ta chỉnh sửa như sau:



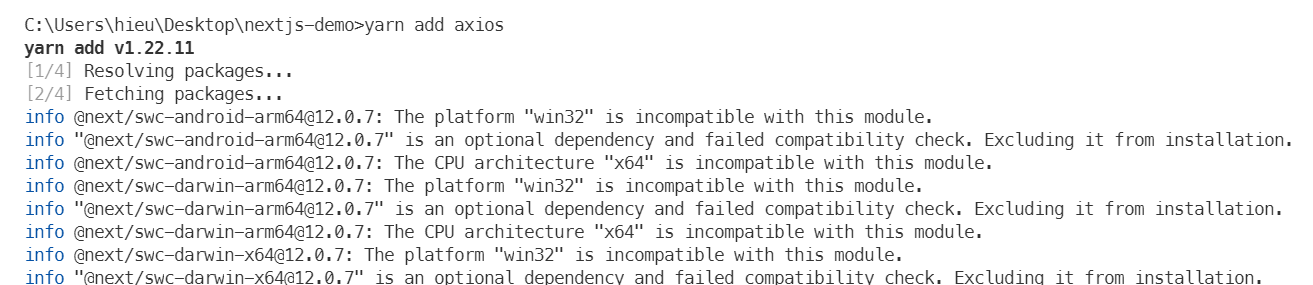
Hình 112. Chỉnh sửa file config\cors.php

Thực hiện khởi động lại server, ta đã tắt được CORS.

### 2.3.3. Hướng dẫn sử dụng API Scanctum.

Để gọi API ta cần một thư viện khác để xử lý, ở đây ta sẽ sử dụng axios.

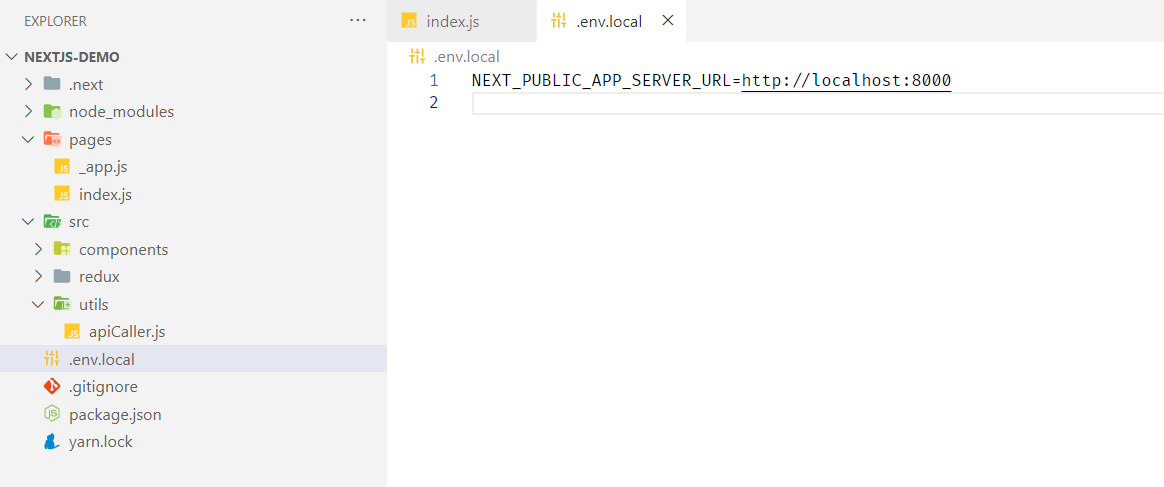
Để cài đặt, bạn hãy gõ câu lệnh: yarn add axios



Hình 113. Chạy câu lệnh yarn add axios

Tiếp tục ta cần tạo file môi trường cho dự án, ở đây mình sẽ cấu hình server đến trang <https://localhost:8000> là đường dẫn của API hiện tại. Tạo một file .env.local và cấu hình như sau:

NEXT\_PUBLIC\_APP\_SERVER\_URL=http://localhost:8000



Hình 114. Cấu hình file môi trường

Để bảo mật file môi trường, ta sẽ không commit nó lên git, hãy tạo một file .gitignore và thêm đoạn sau:

# Environments Variables

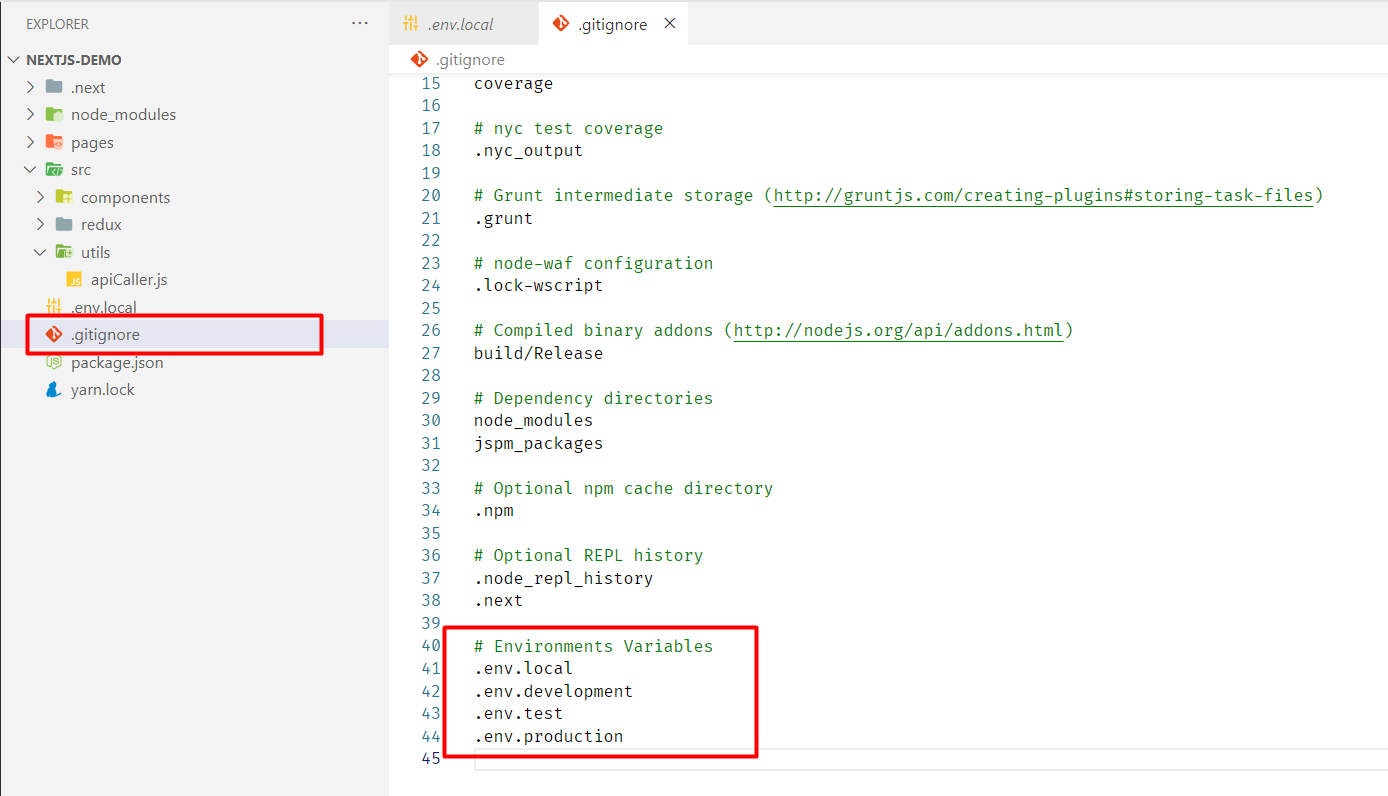
.env.local

.env.development

.env.test

.env.production

Để bảo mật các thuộc tính khác, tải file gitignore example ở đây: <https://github.com/akiran/nextjs-demo/blob/master/.gitignore>



Hình 115. Chỉnh lại file môi trường sẽ không commit nó lên git, hãy tạo một file .gitignore

Tiếp tục trong thư mục src/utils ta tạo một file apiCaller, và cấu hình như sau:

import axios from "axios";

const APP\_SERVER\_URL = process.env.NEXT\_PUBLIC\_APP\_SERVER\_URL;

export default function apiCaller(endpoint, method = "GET", data = null) {

  return axios({

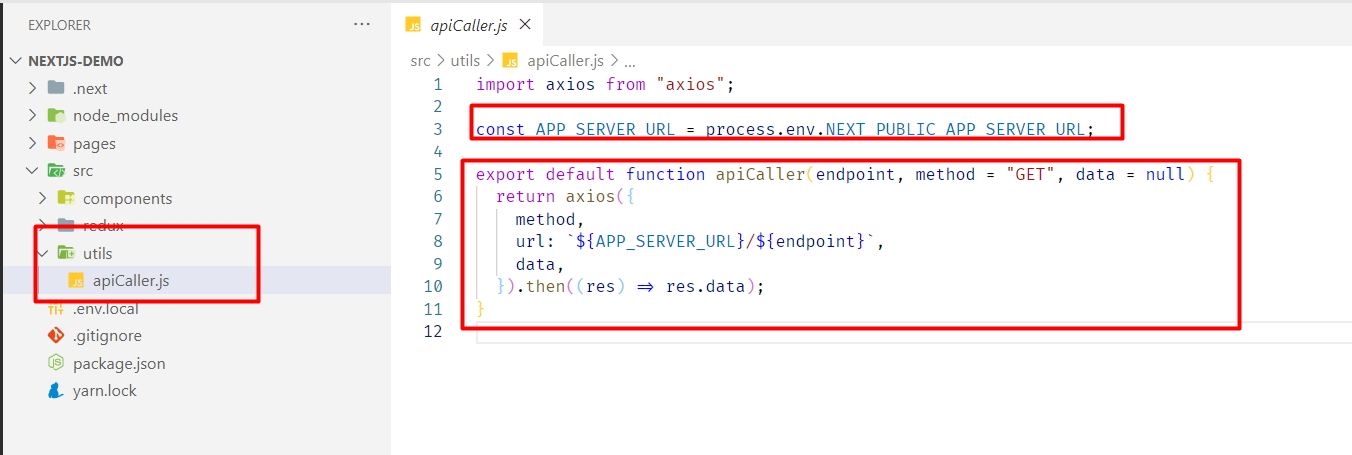
    method,

    url: `${APP\_SERVER\_URL}/${endpoint}`,

    data,

  }).then((res) => res.data);

}



Hình 116. Viết hàm call API

Đây là function chính mà ta sẽ dùng để gọi API scanctum. Để sử dụng nó, ta chỉ cần import vào là xong. Ta chỉnh sửa lại pages/index.js như sau:

import { useEffect, useState } from "react";

import { useDispatch, useSelector } from "react-redux";

import {

  decrementCounter,

  incrementCounter,

} from "../src/redux/actions/counterActions";

import apiCaller from "../src/utils/apiCaller";

function HomePage() {

  const dispatch = useDispatch();

  const counter = useSelector((state) => state.counter);

  const counterValue = counter?.value;

  const [products, setProducts] = useState([]);

  useEffect(() => {

    apiCaller("products", "GET").then((res) => {

      console.log(res);

      setProducts(res.data);

    });

  }, []);

  return (

    <div>

      <p>Welcome to Next.js!</p>

      <div>{counterValue}</div>

      <button onClick={() => dispatch(incrementCounter())}>+</button>

      <button onClick={() => dispatch(decrementCounter())}>-</button>

      {products.map((product) => (

        <p key={product.id}>{product.title}</p>

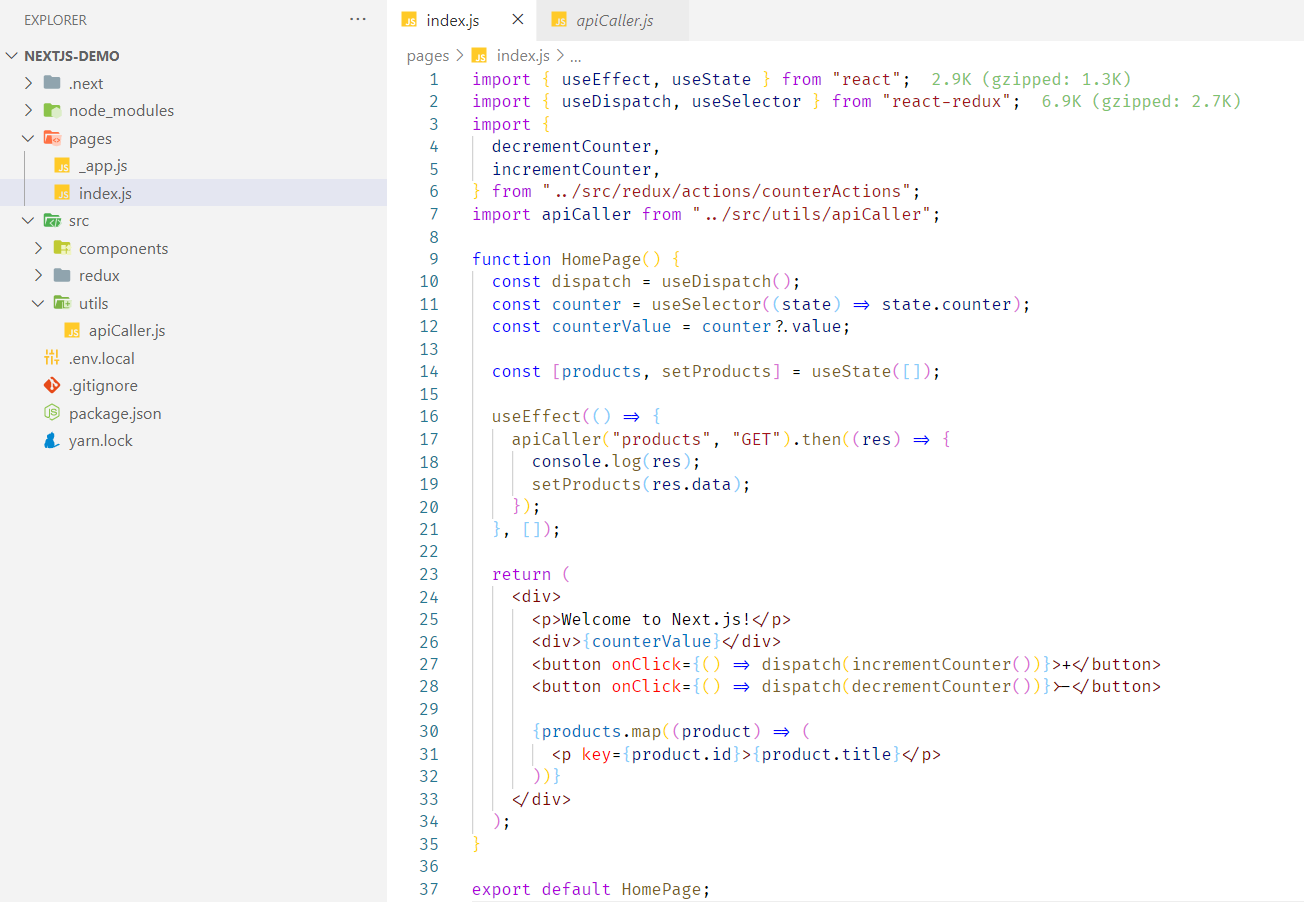
      ))}

    </div>

  );

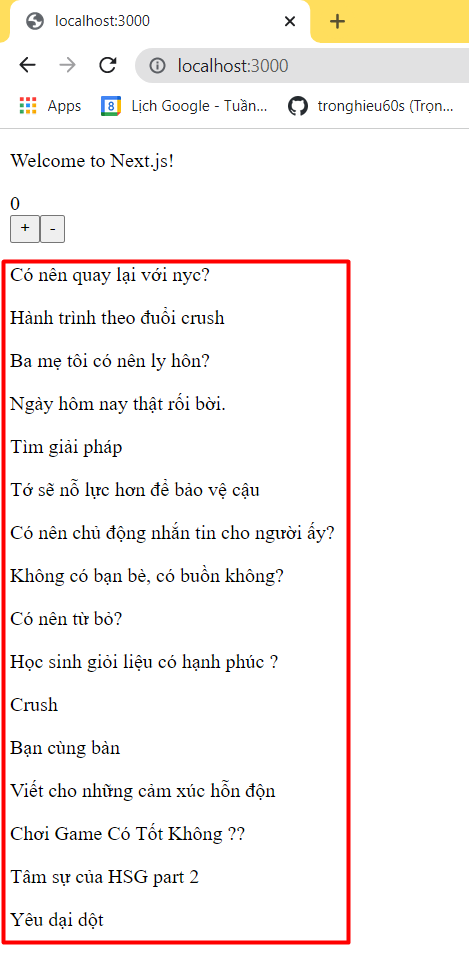
}

export default HomePage;



Hình 117. Import hàm call API để sử dụng

Ở đây trong useEffect ta sẽ gọi apiCaller từ file mà ta đã tạo trước đó, ở đây useEffect không được truyền dependencies, mục đích để gọi 1 lần duy nhất, sau khi có kết quả ta sẽ đưa nó cho products state, khi có kết quả sẽ được hiển thị bên dưới.



Hình 118. Call Api và đỗ dữ liệu ra màn hình

Lưu ý: có thể khi gọi sẽ bị lỗi 403, hay 404 bạn hãy kiểm tra lại đường dẫn hoặc server có thể đang chặn CORS.