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

**CHUYÊN ĐỀ WEB 1** | 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

[1. DESIGN PATTERN 4](#_Toc86929601)

[1.1. Factory 4](#_Toc86929602)

[1.1.1. Factory là gì? 4](#_Toc86929603)

[1.1.2. Ví dụ thực tiễn. 4](#_Toc86929604)

[1.1.3. Minh họa lược đồ Factory. 5](#_Toc86929605)

[1.2. Repository 6](#_Toc86929606)

[1.2.1. Repository là gì? 6](#_Toc86929607)

[1.2.2. Ví dụ thực tiễn. 6](#_Toc86929608)

[1.2.3. Minh họa lược đồ Repository. 8](#_Toc86929609)

[1.3. Proxy 8](#_Toc86929610)

[1.3.1. Proxy là gì? 9](#_Toc86929611)

[1.3.2. Ví dụ thực tiễn. 9](#_Toc86929612)

[1.3.3. Minh họa lược đồ Proxy. 10](#_Toc86929613)

[2. DỰ ÁN CHUYÊN ĐỀ WEB 1 11](#_Toc86929614)

[2.1. Phân Công 11](#_Toc86929615)

[2.2. Đặc Tả Đồ Án 11](#_Toc86929616)

[2.2.1. Hiển thị confessions 12](#_Toc86929617)

[2.2.2. Phân trang hiển thị confessions 14](#_Toc86929618)

[2.2.3. Sửa confessions 17](#_Toc86929619)

[2.2.4. Sign Up confessions 18](#_Toc86929620)

[2.2.5. Sign In confessions 20](#_Toc86929621)

[2.2.6. Sign Out confessions 23](#_Toc86929622)

[Hình 1. Gọi và khởi tạo userModel và bankModel. 4](#_Toc86431535)

[Hình 2. Khởi tạo một lớp FactoryModel có nhiệm vụ khai báo. 4](#_Toc86431536)

[Hình 3. Để sử dụng ta chỉ cần gọi một lớp Factory duy nhất. 5](#_Toc86431537)

[Hình 4. Trang chủ hiển thị confessions. 7](#_Toc86431538)

[Hình 5. Gọi useEffect để dispatch actions dữ liệu. 7](#_Toc86431539)

[Hình 6. Xử lí gọi API và đưa vào reducers của Redux. 8](#_Toc86431540)

[Hình 7. Sử dụng useSelector của Redux để lấy dữ liệu. 8](#_Toc86431541)

[Hình 8. Hiển thị dữ liệu bằng cách map qua mảng dữ liệu. 8](#_Toc86431542)

[Hình 9. Hiển thị phân trang trên giao diện. 9](#_Toc86431543)

[Hình 10. Lấy số trang hiện tại bằng useRouter, mặc định là 1. 9](#_Toc86431544)

[Hình 11. Sử dụng useEffect để xử lí bài viết hiển thị. 10](#_Toc86431545)

[Hình 12. Hiển thị pagination khi số lượng bài viết lớn hơn 0. 10](#_Toc86431546)

[Hình 13. Hiển thị Pagination cho bài viết. 11](#_Toc86431547)

[Hình 14. Logic xử lí hiển thị pagination. 11](#_Toc86431548)

[Hình 15. Hiển thị sửa khi người dùng đăng nhập và đúng bài viết. 12](#_Toc86431549)

[Hình 16. Hiển thị modal sửa khi người dùng nhấp vào nút sửa. 12](#_Toc86431550)

[Hình 17. Khi click vào nút sửa sẽ gọi đến function sửa. 13](#_Toc86431551)

[Hình 18. Xử lí sửa bài viết và load lại dữ liệu. 13](#_Toc86431552)

# 1. DESIGN PATTERN

## 1.1. Factory

### 1.1.1. Factory là gì?

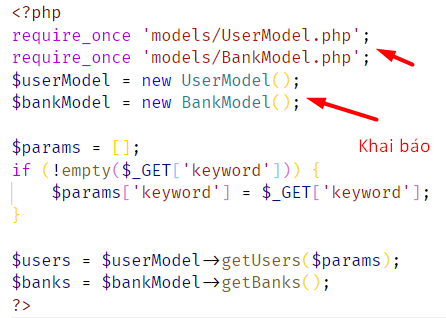
Factory design pattern rất phổ biến và không thể thiếu trong lập trình hướng đối tượng (OOP). Factory là pattern thuộc nhóm gọi là mẫu khởi tạo, giúp quản lý và cung cấp một cách ngắn gọn để khởi tạo đối tượng.

### 1.1.2. Ví dụ thực tiễn.

**Vấn đề:**

Để sử dụng một lớp đối tượng, ta cần gọi và khởi tạo để sử dụng nó, nhưng có một bất lợi là ta sử dụng nhiều lớp trong một lần, rất bất tiện khi phải cứ gọi và khởi tạo.

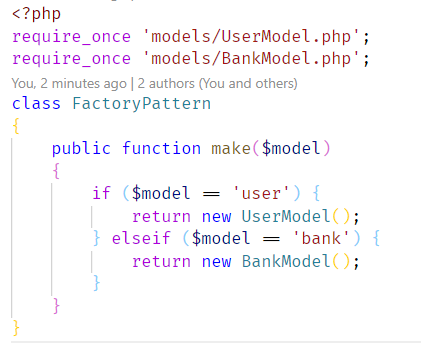
Dưới đây là lớp Users và Banks được gọi vào và khởi tạo để sử dụng, ta có thể thấy nó khá rườm rà và không tối ưu.



Hình 1. Gọi và khởi tạo userModel và bankModel.

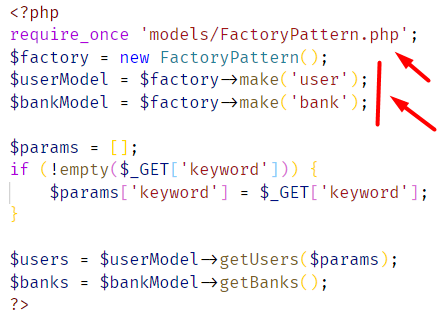
**Giải pháp:**

Để khắc phục điều này, ta sẽ gọi một lớp khác, có nhiệm vụ gọi thay cho lớp gốc và sử dụng rất dễ dàng. Ta cần khai báo thêm một lớp như sau.



Hình 2. Khởi tạo một lớp FactoryModel có nhiệm vụ khai báo.

Để sử dụng, ta chỉ cần gọi và khai báo duy nhất lớp FactoryModel và mọi thứ sẽ ngắn gọn và tối ưu hơn.



Hình 3. Để sử dụng ta chỉ cần gọi một lớp Factory duy nhất.

### 1.1.3. Minh họa lược đồ Factory.

Hình minh họa dưới đây mô tả lớp ShapeFactory được khởi tạo, và gọi đến phương thức getShape(), phương thức getShape sẽ được truyền vào 3 giá trị “circle”, “rectangle”, “square” để lấy được lớp tương ứng mà không cần phải tự gọi và khai báo từng phần.

Các lớp tương ứng có cùng một phương thức draw() được kế thừa từ Shape Interface.



Hình 4. Lược đồ UML Factory Pattern của Shape.

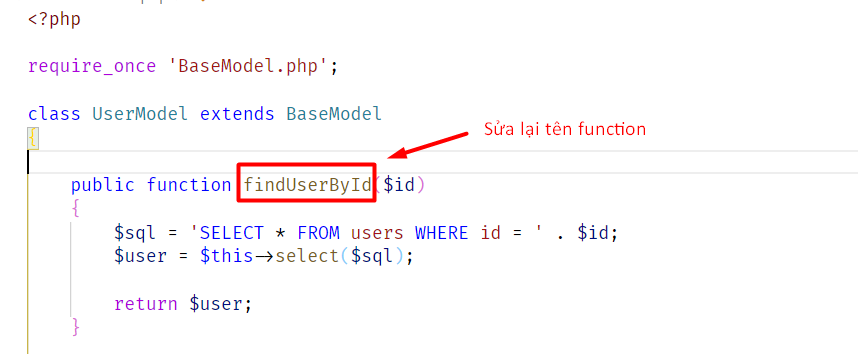
## 1.2. Repository

### 1.2.1. Repository là gì?

Repositoty là nơi làm việc và xử lý của tầng Data và tầng Logic. Hiểu đơn giản là chúng ta sẽ tạo một file có tên là Repository và ở đây chúng ta sẽ gọi các phương thức hoặc hàm xử lý truy data . Và chúng ta sẽ gọi các hàm để sử dụng thay vì gọi thẳng controller những việc trên giúp chúng ta chỉ cẩn sửa code ở một nơi không cần đi sửa ở nhiều nơi.

### 1.2.2. Ví dụ thực tiễn.

**Vấn đề:**

Nếu ta thay đổi code ở tầng Data thì chúng ta phải đi thay đổi lại tất cả các hàm mà chúng ta vừa sửa ở tầng Data mà được gọi trên tầng Logic việc đó sẽ làm chúng ta mất thời gian và nhiều khi còn có thể xóa thiếu hay là sửa dư hoặc không chính xác làm ảnh hưởng tới code.

Hình 5. Sửa tên function ở tầng Data



Hình 6. Nơi phải sửa lại ở tầng Logic

**Giải pháp:**

Để khắc phục trường hợp trên chúng ta chỉ cần tạo ra một file repository làm nơi trung gian để xử lý. Khi chúng ta cần thay đổi hay sửa một hàm gì đó chỉ cần vào file repository để sửa nó sẽ tiết kiệm cho chúng ta nhìu thời gian và tránh trường hợp xóa thiếu hay sửa dư.



Hình 7. Thêm file repository

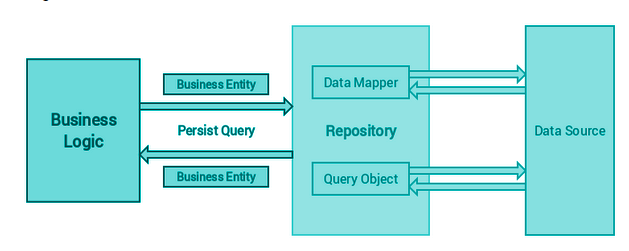
Và cách gọi hàm để sử dụng cũng vô cùng đơn giản như cách gọi các controller bình thường



Hình 8. Khai báo và sử dụng Repository

### 1.2.3. Minh họa lược đồ Repository.

Lược đồ dưới đây minh họa mô tả Repository là một nơi để xử lý các yêu cầu về truy xuất Data và xử lý Logic. Ở đây nó sẽ nhận các câu truy xuất data và xử lý phù hợp để đưa lên tầng Business Logic và ngược lại. Nó sẽ giúp chúng ta bảo mật và bảo trì code tốt hơn.



Hình 9. Lược đồ minh họa về Repository

## 1.3. Proxy

### 1.3.1. Proxy là gì?

Proxy là một đối tượng sẽ đại diện cho một đối tượng khác.

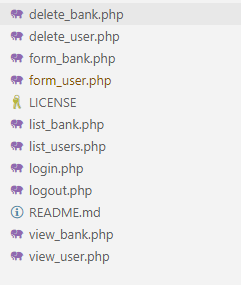
* Kiểm soát quyền truy xuất các phương thức của đối tượng.
* Bổ sung thêm chức năng trước khi thực thi phương thức.
* Tạo ra đối tượng mới có chức năng nâng cao hơn đối tượng ban đầu.
* Giảm chi phí khi có nhiều truy cập vào đối tượng có chi phí khởi tạo ban đầu lớn.

### 1.3.2. Ví dụ thực tiễn.

**Vấn đề:**

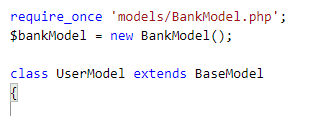
Nếu chúng ta có 1 đồ án lớn mà có những bảng giống nhau, chúng ta cứ code thì sẽ bất tiện và mất thời gian

Dưới đây là Users và Banks, ta có thể thấy chúng ta sẽ code rất nhiều và rườm rà, dù đây là 2 bảng giống nhau

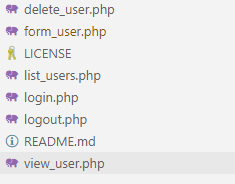


**Giải pháp:**

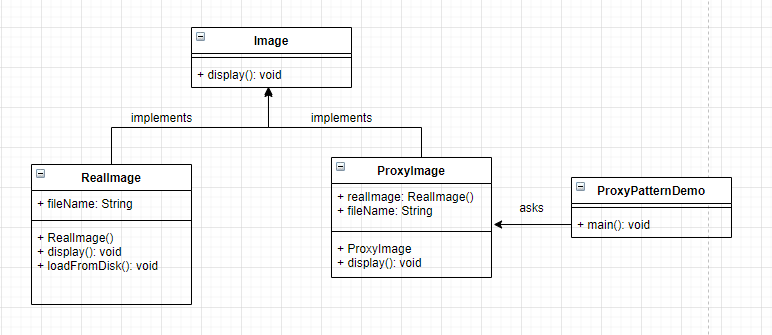
1. Cô lập UsersModel và BankModel
2. UserModel sẽ gọi BankModel và làm những việc của BankModel



1. Để sử dụng, ta chỉ cần gọi vào User thì Bank cũng sẽ chạy, và mọi thứ sẽ ngắn gọn và tối ưu hơn.



### 1.3.3. Minh họa lược đồ Proxy.



# 2. DỰ ÁN CHUYÊN ĐỀ WEB 1

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.

- Hoàn thiện đặc tả yêu cầu đồ án theo hướng tính năng rõ ràng

- Hoàn thiện đặc tả yêu cầu đồ án theo hướng hình minh họa phát thảo rõ ràng

- Hoàn thiện đặc tả yêu cầu đồ án theo hướng kế hoạch (phân công, thời gian rõ ràng, git)

## 2.1. Phân Công

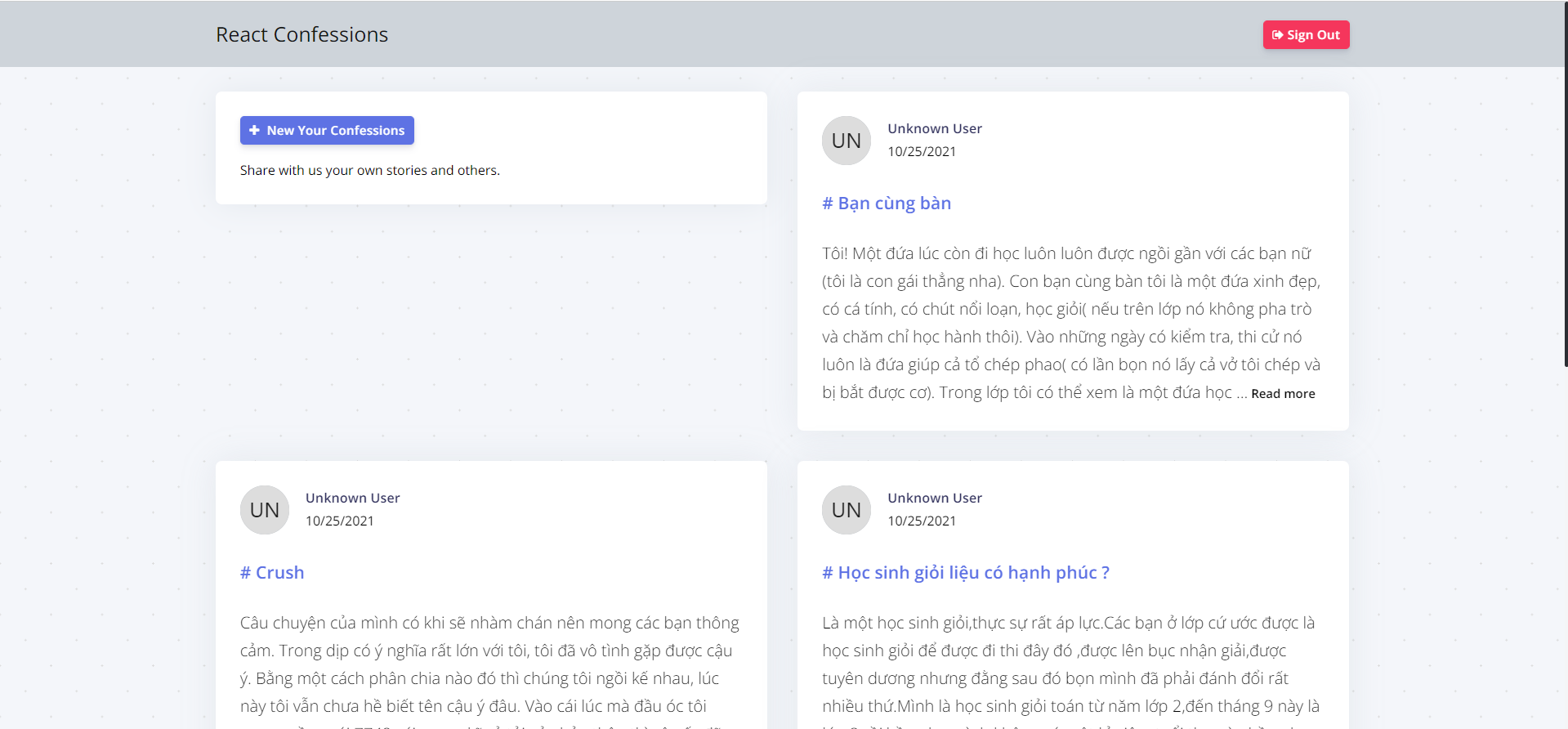
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Thành Viên** | **Nội Dung Phân Công** | **Branch (feature/)** | **Thời Gian** |
| 1 | Ngô Bảy Hên | Đăng nhập | Sign-In | 5 ngày |
| Đăng ký | Sign-Up | 5 ngày |
| Đăng xuất người dùng | Sign-Out | 2 ngày |
| 2 | Trần Trọng Hiếu | Hiển thị confessions | Show-Confession | 4 ngày |
| Phân trang hiển thị confessions | Pagination-Confession | 4 ngày |
| Sửa confessions | Edit-Confession | 4 ngày |
| 3 | Nguyễn Huỳnh Quốc Trung | Thêm confessions | Add-Confession | 5 ngày |
| Xóa confessions | Delete-Confession | 3 ngày |

## 2.2. Đặc Tả Đồ Án

### 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.

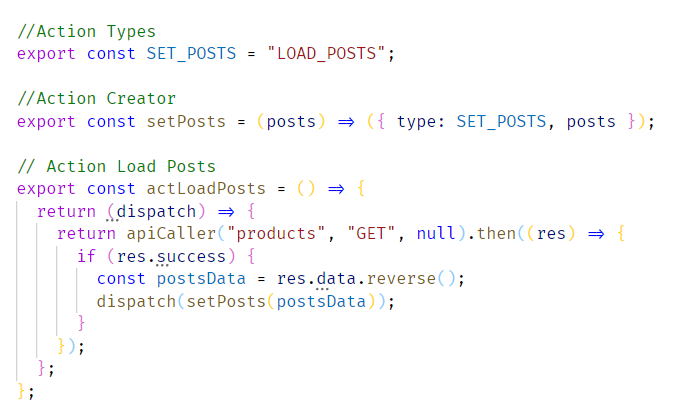


Hình 10. Trang chủ hiển thị confessions.

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.

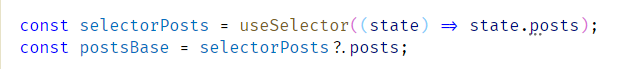


Hình 11. Gọi useEffect để dispatch actions dữ liệu.

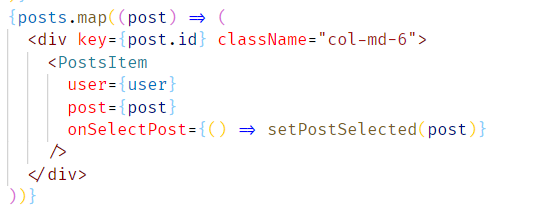


Hình 12. Xử lí gọi API và đưa vào reducers của Redux.

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 13. Sử dụng useSelector của Redux để lấy dữ liệu.

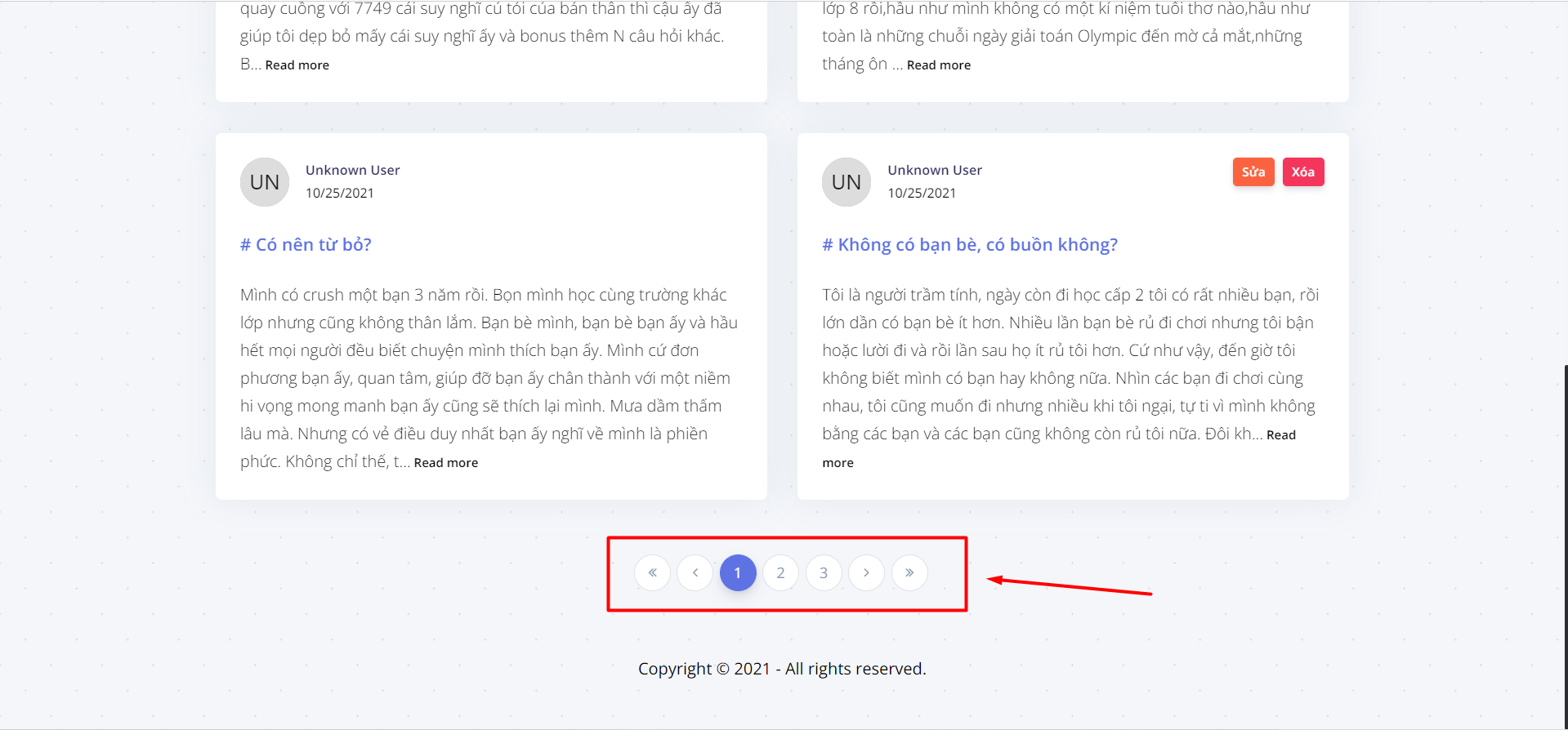


Hình 14. Hiển thị dữ liệu bằng cách map qua mảng dữ liệu.

### 2.2.2. Phân trang hiển thị confessions

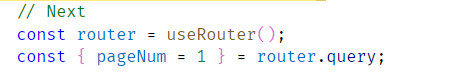
Hiển thị phân trang ở phía dưới của bài viết, giao diện bao gồm:

* 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 15. 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.



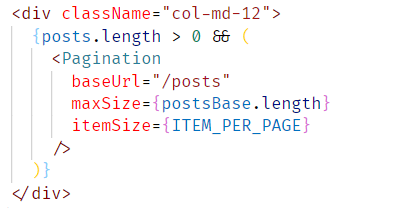
Hình 16. Lấy số trang hiện tại bằng useRouter, mặc định là 1.

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 17. Sử dụng useEffect để xử lí bài viết hiển thị.

Hiển thị Pagination ở trang chủ bằng components Pagination.

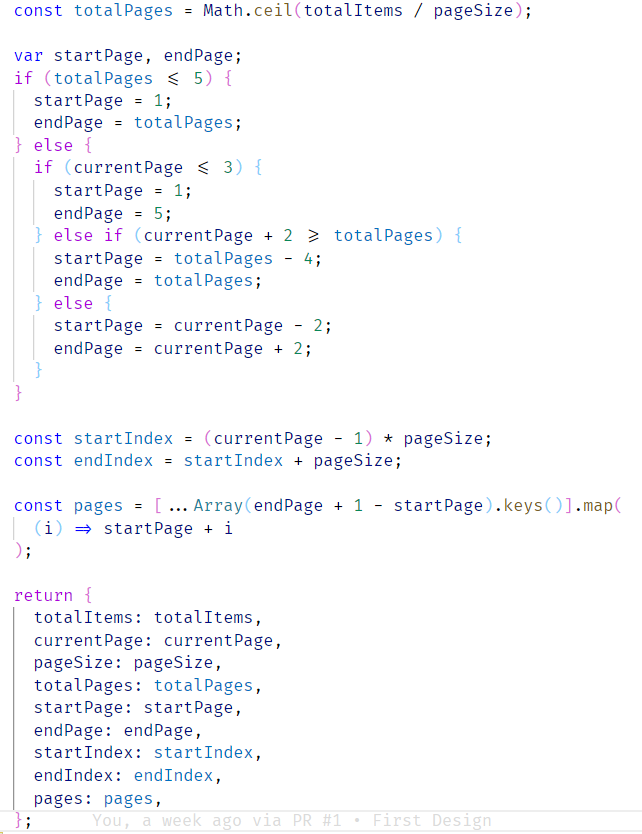


Hình 18. Hiển thị pagination khi số lượng bài viết lớn hơn 0.

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



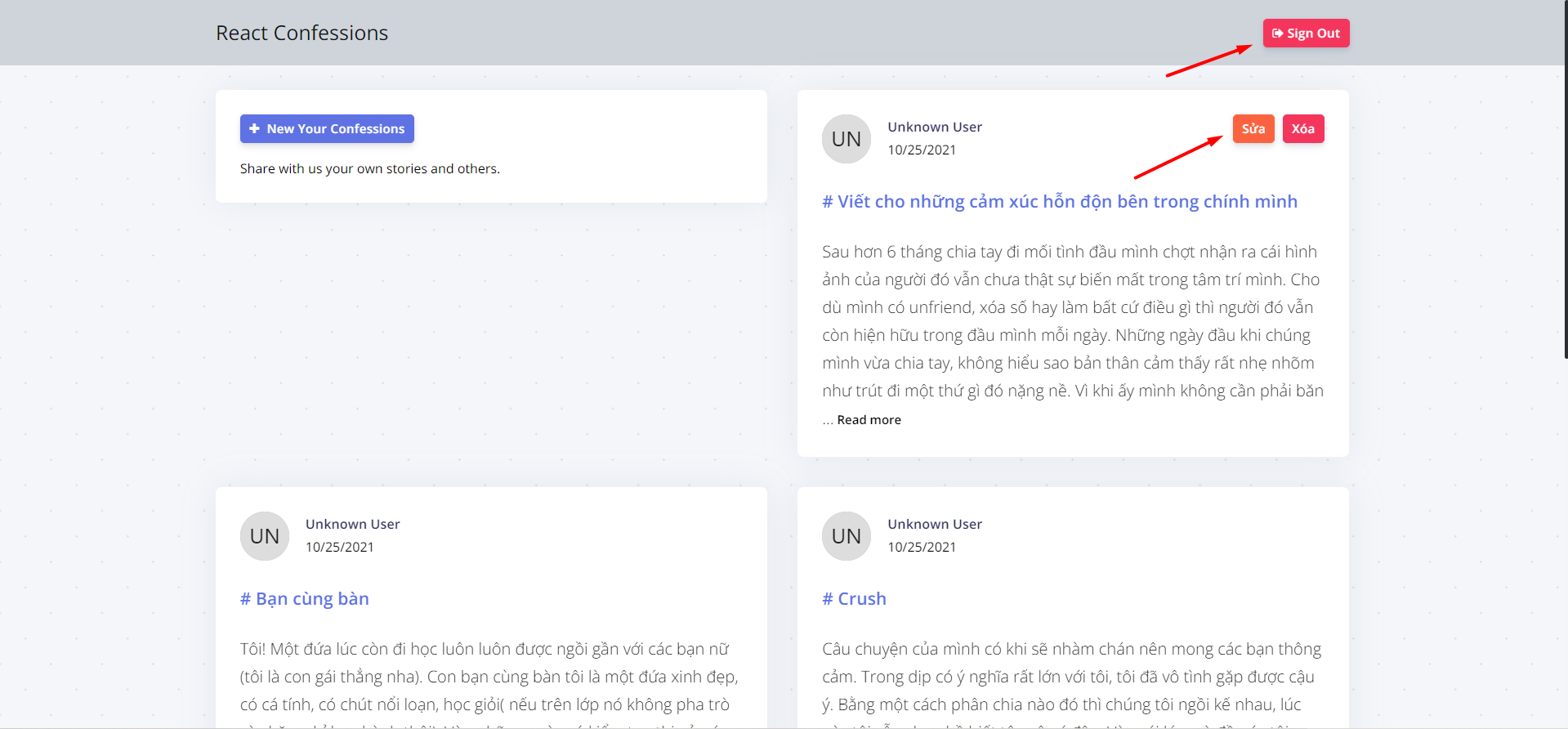
Hình 19. Hiển thị Pagination cho bài viết.



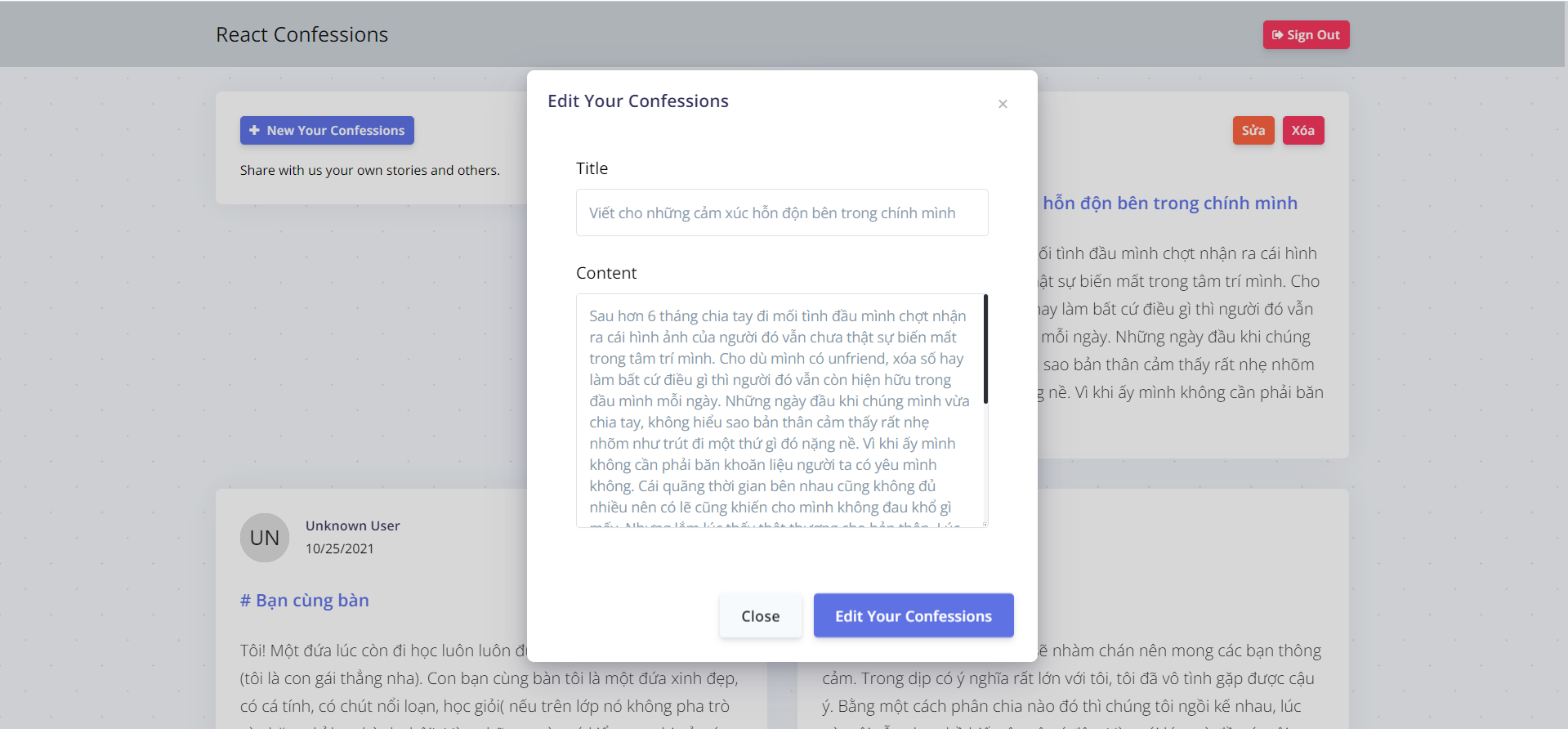
Hình 20. Logic xử lí hiển thị pagination.

### 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.

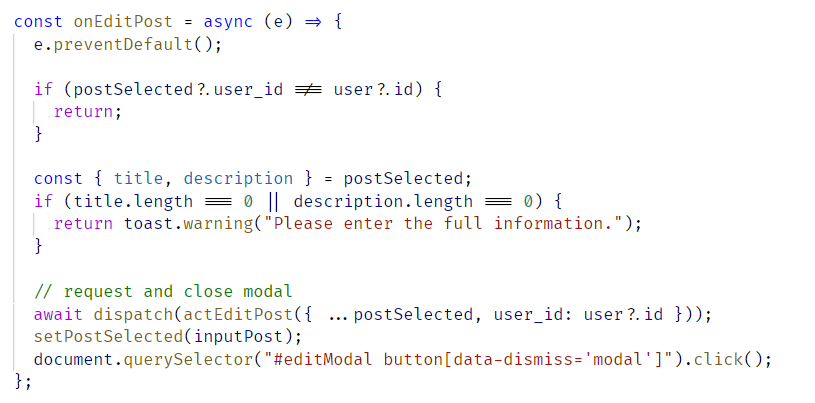


Hình 21. Hiển thị sửa khi người dùng đăng nhập và đúng bài viết.



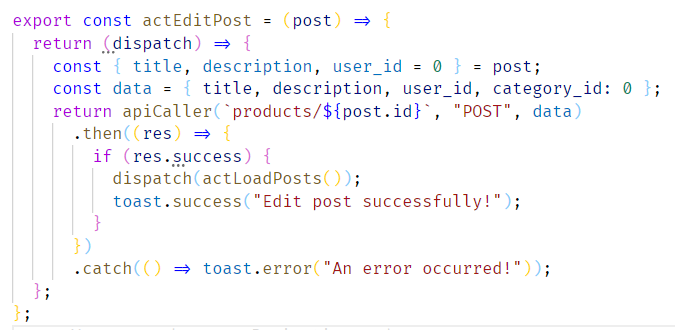
Hình 22. Hiển thị modal sửa khi người dùng nhấp vào nút sửa.

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 23. Khi click vào nút sửa sẽ gọi đến function sửa.

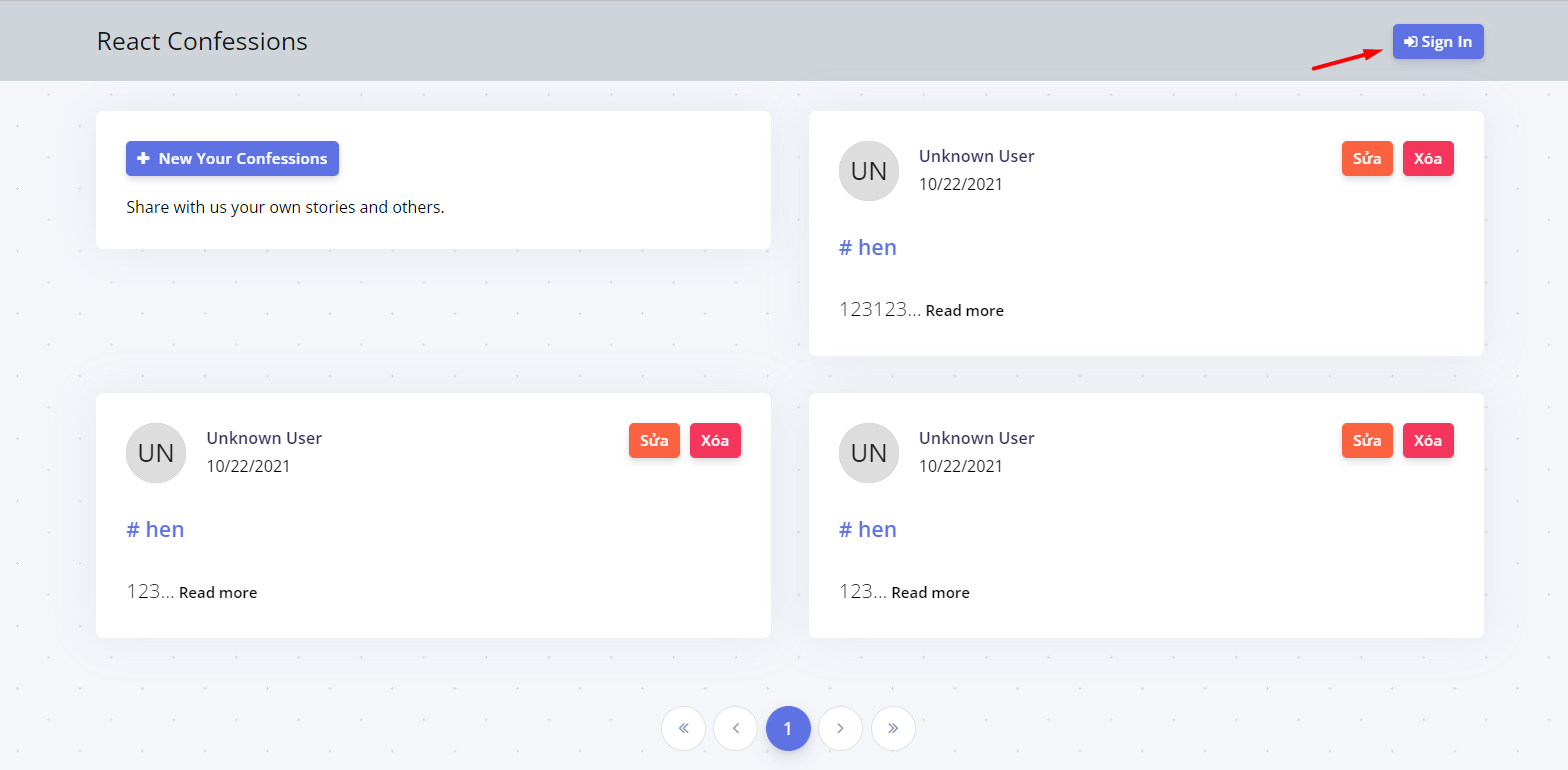
Ở trong Redux sẽ xứ lí chức năng sửa bài viết, và load lại tất cả bài viết hiện tại.



Hình 24. Xử lí sửa bài viết và load lại dữ liệu.

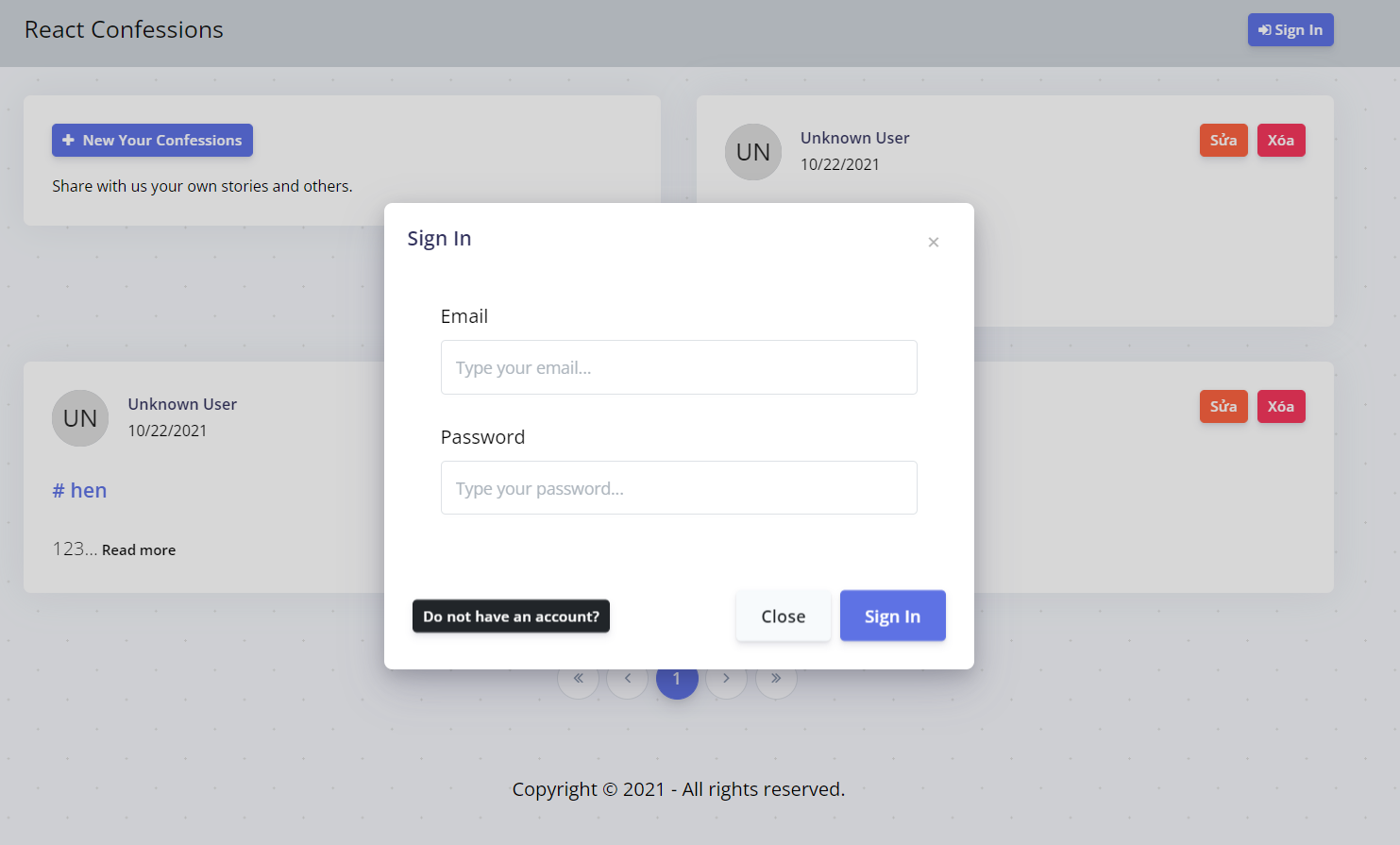
### 2.2.4. Sign Up confessions

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.



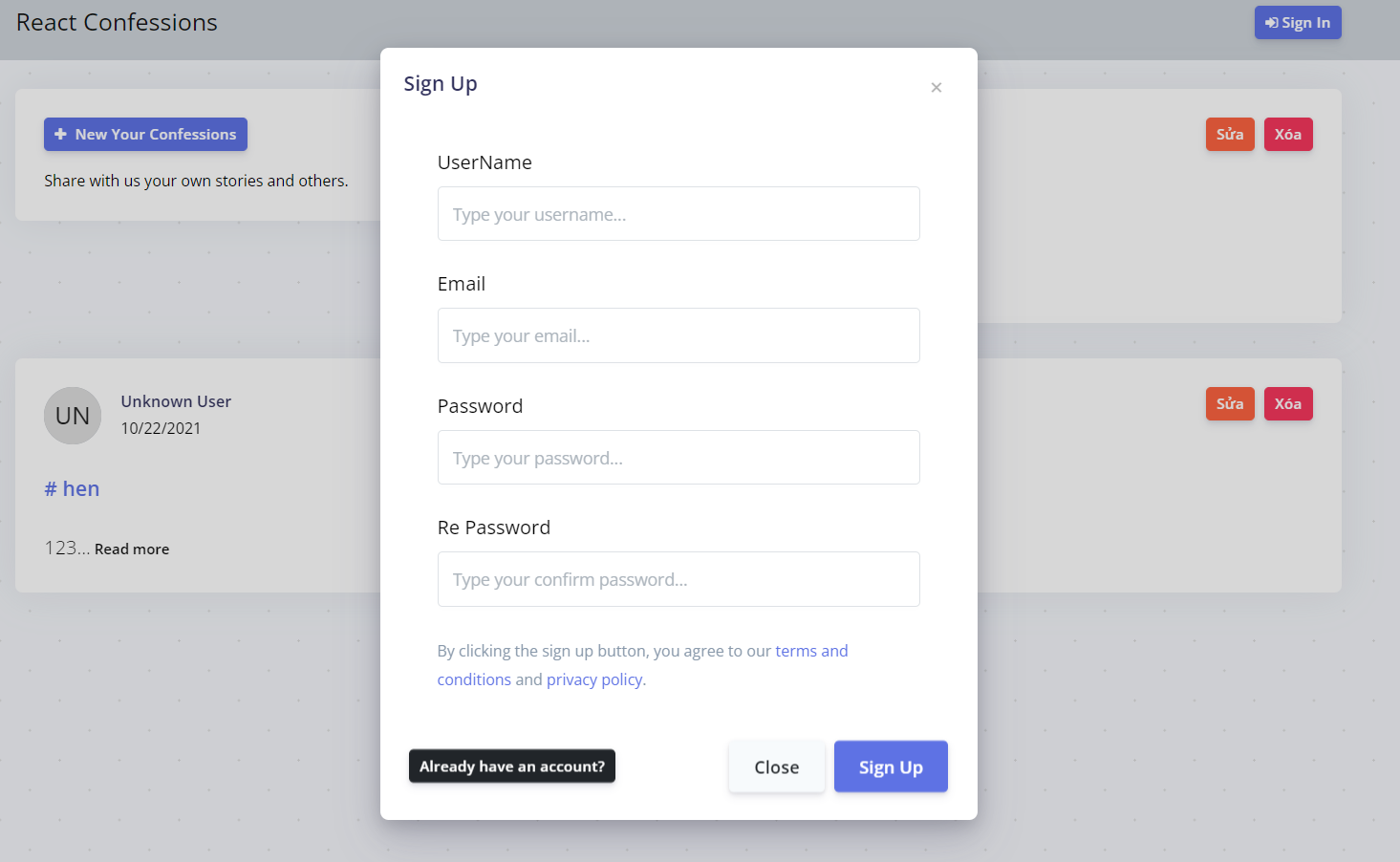
Hình 25. Nơi hiển thị button Sign In

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.



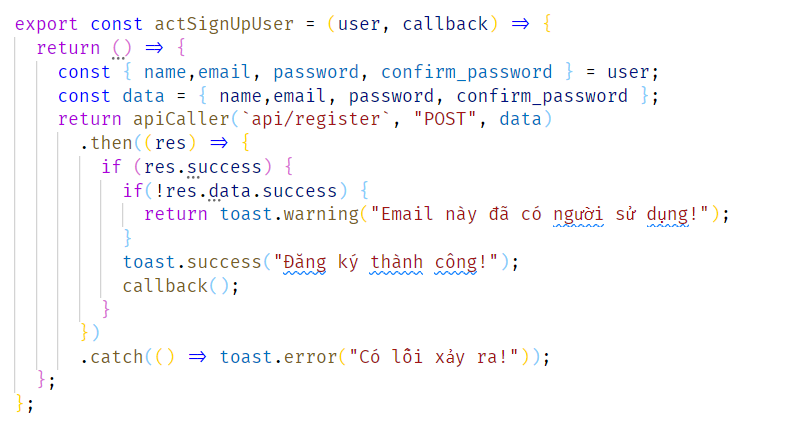
Hình 26. Hiện model để đăng nhập.

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 27. Hiện model để đăng ký.

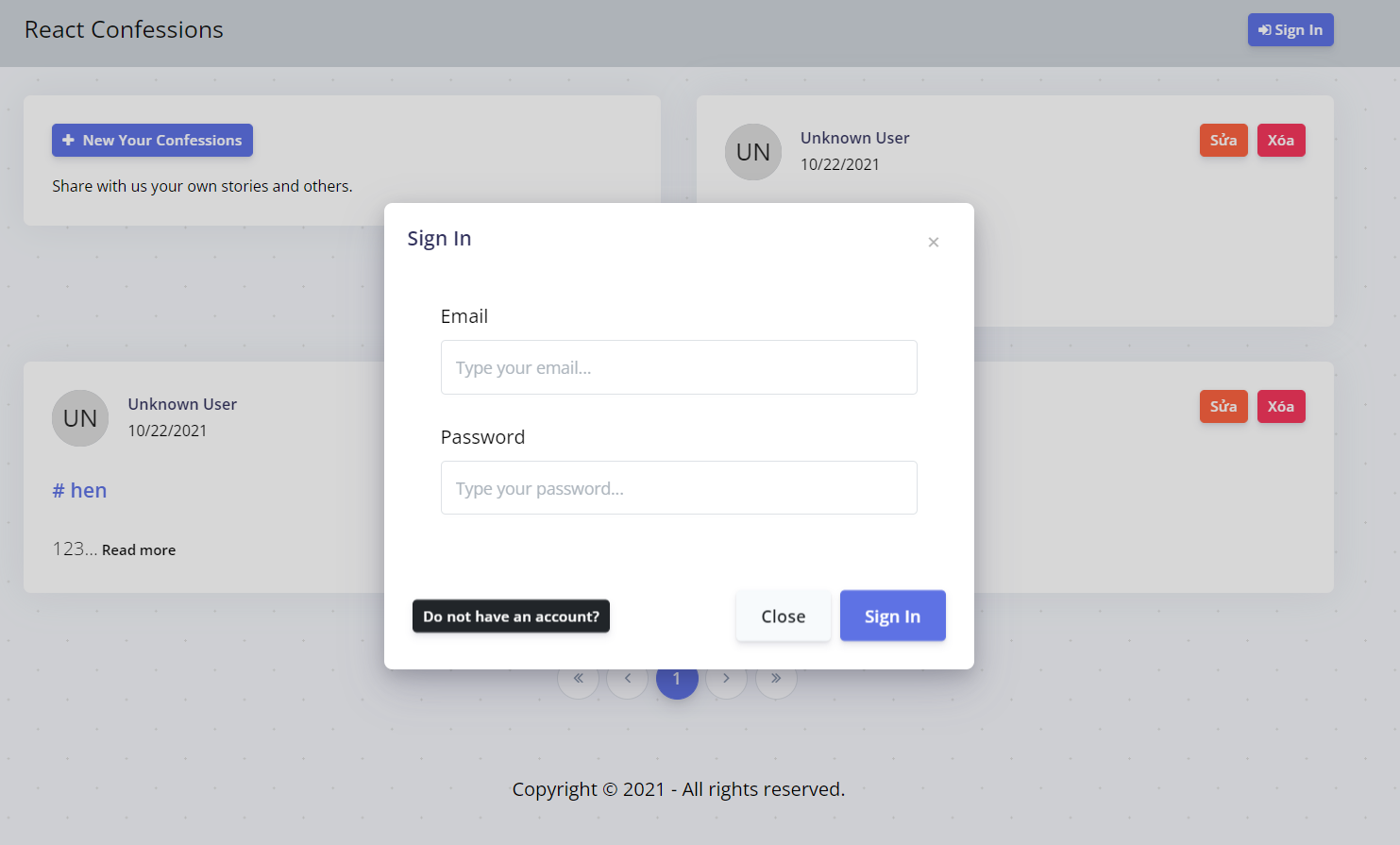
Ở model này người dùng chỉ nhập đầy đủ và đúng thông tin và nhấn button Sign Up web sẽ vào Redux và callApi để thực hiện chức năng đăng ký và thông báo kết quả. Nếu đăng ký thành công thì sẽ web sẽ navigation qua model Sign In.



Hình 28. Hàm Sign Up action ở redux.

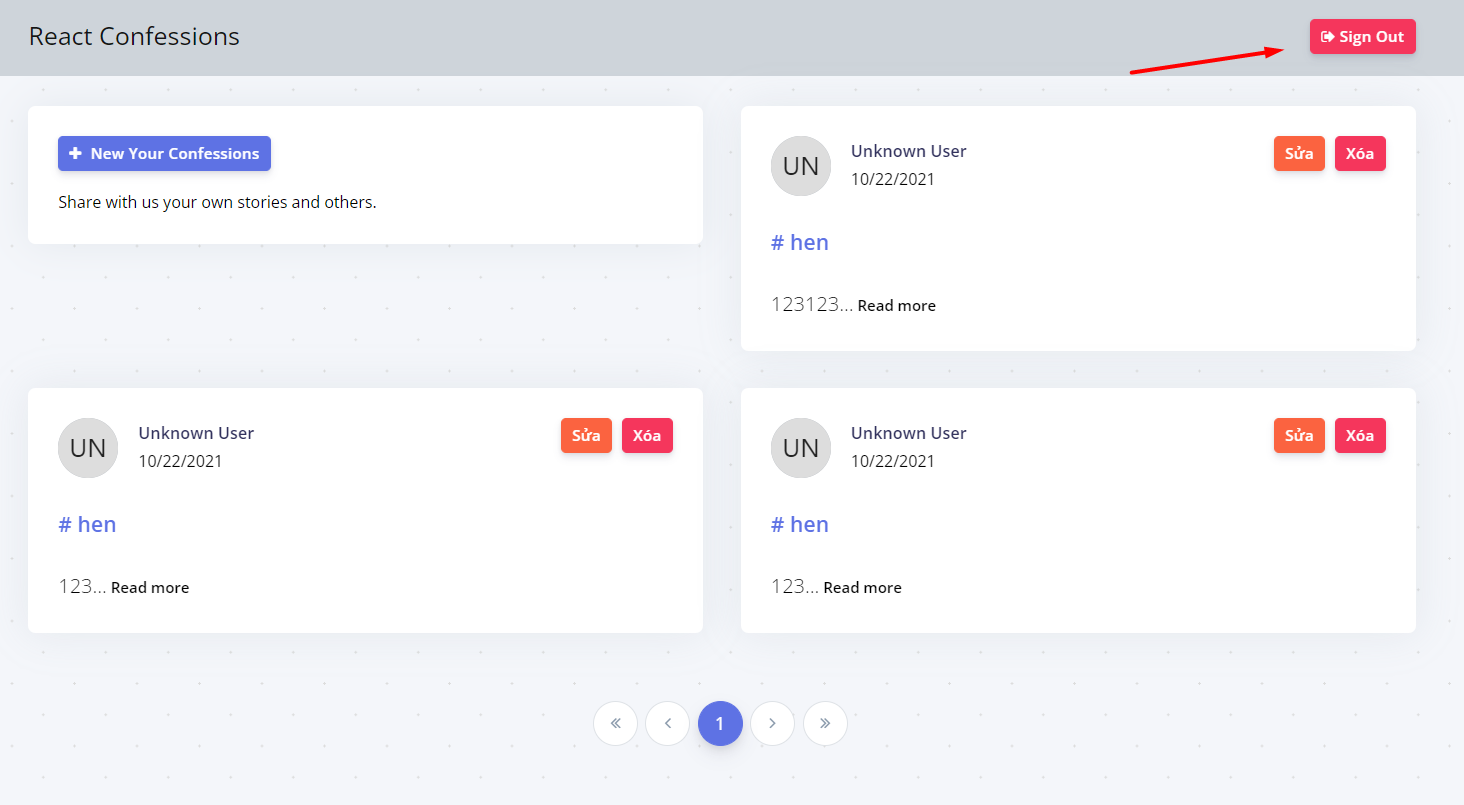
### 2.2.5. Sign In confessions

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

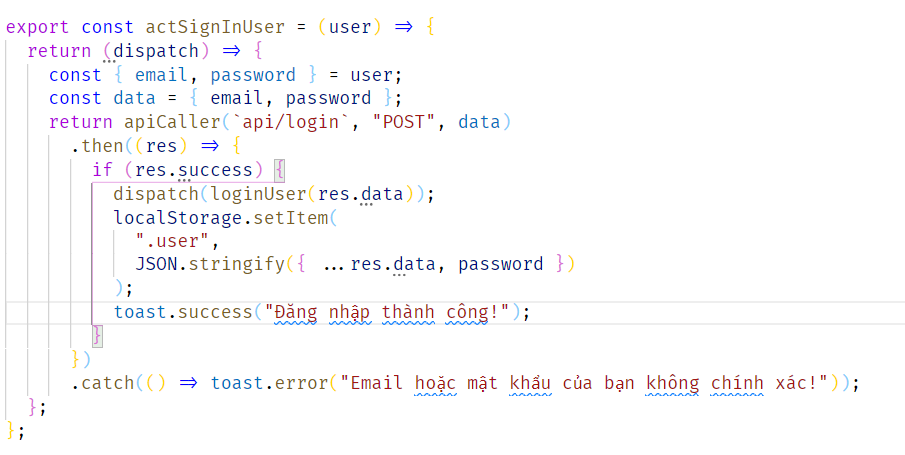


Hình 28. Model Sign In.

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 bào đăng nhập thành công chuyển button Sign In thành Sign Out.



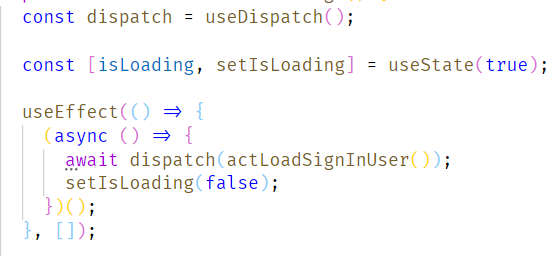
Hình 29. Sau đăng nhập thành công.



Hình 30. 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.

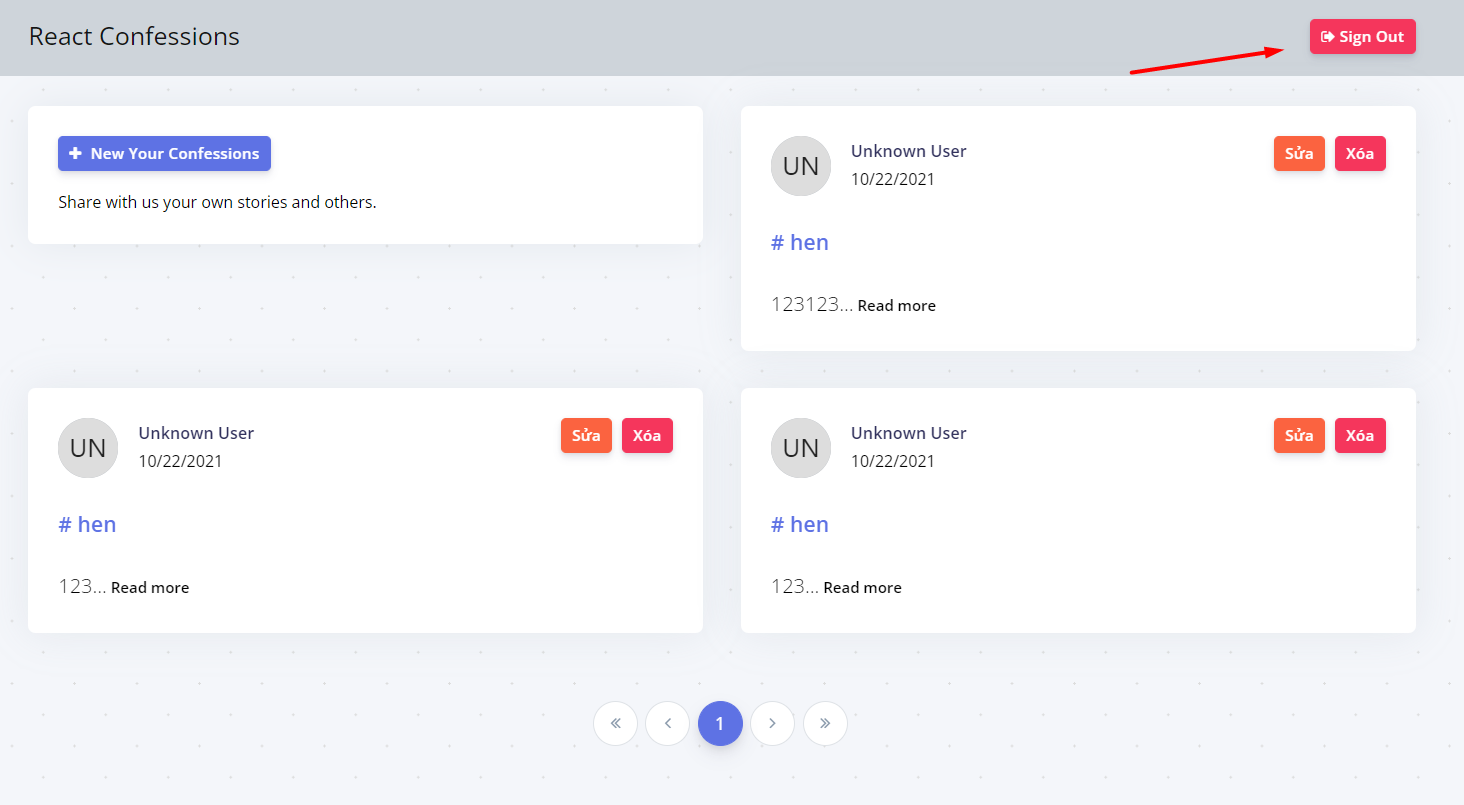




Hình 31. Hàm Redux loading đăng nhập.

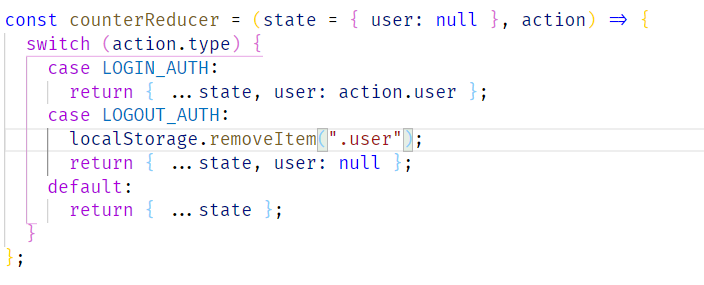
### 2.2.6. Sign Out confessions

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.

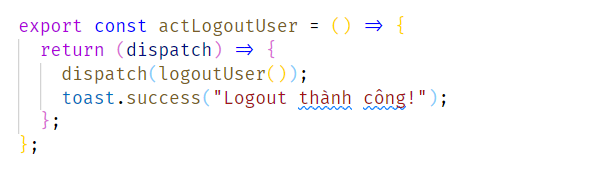


Hình 32. Hiện button Sign Out.

Sau khi click vào button Sign Out thì token ở LocalStorage sẽ bị xóa và button sẽ chuyền 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 33. Ở hàm authReducer.



Hình 34. Hàm Logout ở Redux.