**BÁO CÁO 5: TÌM HIỂU VỀ AXIOS, ROUTER, ANT DESIGN VÀ REDUX**

1. **Axios**
   1. **Get request**
      * Đây là phương thức để lấy toàn bộ data về, chính vì thế get request thường được đặt trong life cycle componentDidMount để lấy dữ liệu :

  componentDidMount() {

    axios

      .get("https://60b0f8b91f26610017fff943.mockapi.io/api/v1/todo\_data")

      .then((res) => {

        this.setState({ todoItems: res.data });

      })

      .catch((err) => {

        console.error(err);

      });

  }

* 1. **Post request**
     + Post request dùng để tạo dữ liệu mới

Axios.post("https://60b0f8b91f26610017fff943.mockapi.io/api/v1/todo\_data", {

      id: n,

      title: event.target.value,

      isEditing: false,

      isCompleted: false,

    })

    .then((res) => {

      console.log(res);

    })

    .catch((err) => {

      console.log(err);

});

* 1. **Put request** 
     + Put request dùng để cập nhật lại dữ liệu theo id

axios.put("https://60b0f8b91f26610017fff943.mockapi.io/api/v1/todo\_data/" + item.id,

{

title: text,

})

.then((res) => {

console.log(res.data);

})

.catch((err) => {

console.log(err);

});

* 1. **Delete request**
     + Delete request dùng để xóa dữ liệu theo id

axios.delete("https://60b0f8b91f26610017fff943.mockapi.io/api/v1/todo\_data/" +

item.id

)

.then((response) => {

console.log(response);

});

* 1. **Vấn đề về too many request**
     + Khi thực hiện gửi đồng loạt nhiều request trong một lúc cho server thì sẽ gặp hiện tượng lỗi too many request (error 429). Để khắc phục vấn đề này ta sẽ setTimeout cho mỗi lần lặp gửi request.

for (let i = 0; i < todoItem.length; i++) {

  setTimeout(() => {

    axios.put(

      "https://60b0f8b91f26610017fff943.mockapi.io/api/v1/todo\_data/" +

        todoItems[i].id,

      {

        isCompleted: todoItems[i].isCompleted,

      }

    )

    .then((res) => {

      console.log(res.data);

    })

    .catch((err) => {

      console.log(err);

    });

  }, 5000);

}

* + - Tuy nhiên trong JS, hàm setTimeout chỉ delay được duy nhất 1 lần trườc khi thực hiện hàm chính vì thế vẫn xuất hiện lỗi too many request
    - Để giải quyết vấn đề trên, ta sẽ tạo luôn một hàm để lặp và gọi lại hàm đó trong mỗi lần lặp.

function myLoop() {

      setTimeout(function () {

        axios

          .put(

            "https://60b0f8b91f26610017fff943.mockapi.io/api/v1/todo\_data/" +

              todoItems[i].id,

            {

              isCompleted: todoItems[i].isCompleted,

            }

          )

          .then((res) => {

            console.log(res.data);

          })

          .catch((err) => {

            console.log(err);

          });

        i++; //  increment the counter

        if (i < n) {

          myLoop();

        }

      }, 700);

    }

* + - Khi đó, hàm myLoop sẽ được lặp lại sau 700ms
    - Tuy nhiên cách làm này chưa được tối ưu, bởi vì khi muốn thay đổi thời gian timeout phải sửa lại toàn bộ hàm. Chính vì thế ta sẽ sửa lại hàm này với async await.

async function wait(ms) {

      return new Promise((resolve) => {

        setTimeout(resolve, ms);

      });

    }

    async function updateAPI(todoList) {

      for (const item of todoList) {

        await axios

          .put(

            "https://60b0f8b91f26610017fff943.mockapi.io/api/v1/todo\_data/" +

              item.id,

            {

              isCompleted: item.isCompleted,

            }

          )

          .then((res) => {

            console.log(res.data);

          })

          .catch((err) => {

            console.log(err);

          });

        await wait(5000);

      }

    }

* + - Với hàm wait, ta có thể tùy ý cài đặt thời gian timeout mà không cần viết lại hàm

1. **Router**
   1. **Cài đặt**
      * Thực hiện lệnh sau trong terminal: npm install react-router-dom
      * Để điều hướng chúng ta sử dụng các thẻ Router, Switch, Route, Link. Ví dụ:

<Router>

  <div>

    <Link to="/">Home</Link>

    <Link to="/about">About</Link>

    <Link to="/topics">Topics</Link>

  </div>

  <Switch>

    <Route exact path="/about">

      <About />

    </Route>

    <Route exact path="/topics">

      <Topics />

    </Route>

    <Route exact path="/">

      <Home />

    </Route>

  </Switch>

</Router>;

* + - Tất cả các phần để điều hướng được đặt trong thẻ Router. Thẻ Switch dùng để điều hướng khi nhấn vào các thẻ Link
  1. **Sử dụng trong Todo**
     + Trong Todo có 3 loại filter là All (tất cả các công việc), Active (các công việc chưa hoàn thành), Completed (các công việc đã hoàn thành). Vậy nên ta sẽ chuyển 3 nút này thành 3 route.

**Trong app.js**

<Router>

<TodoInput onKeyUp={this.onPressEnter} chooseAll={this.chooseAll} />

   <Switch>

    <Route exact path="/">

    <TodoGeneral path="" todoState={this.state} />

    </Route>

    <Route exact path="/active">

    <TodoGeneral path="active" todoState={this.state} />

    </Route>

    <Route exact path="/completed">

    <TodoGeneral path="completed" todoState={this.state} />

    </Route>

   </Switch>

   <TodoFilter numItem={numItem} clearCompleted={this.clearCompleted} />

</Router>

**Trong TodoFilter.js**

<div className="TodoFilter">

<p className="numItem">{numItem} left</p>

   <div className="filter" onClick={routeFunc}>

    <Link to="/">All</Link>

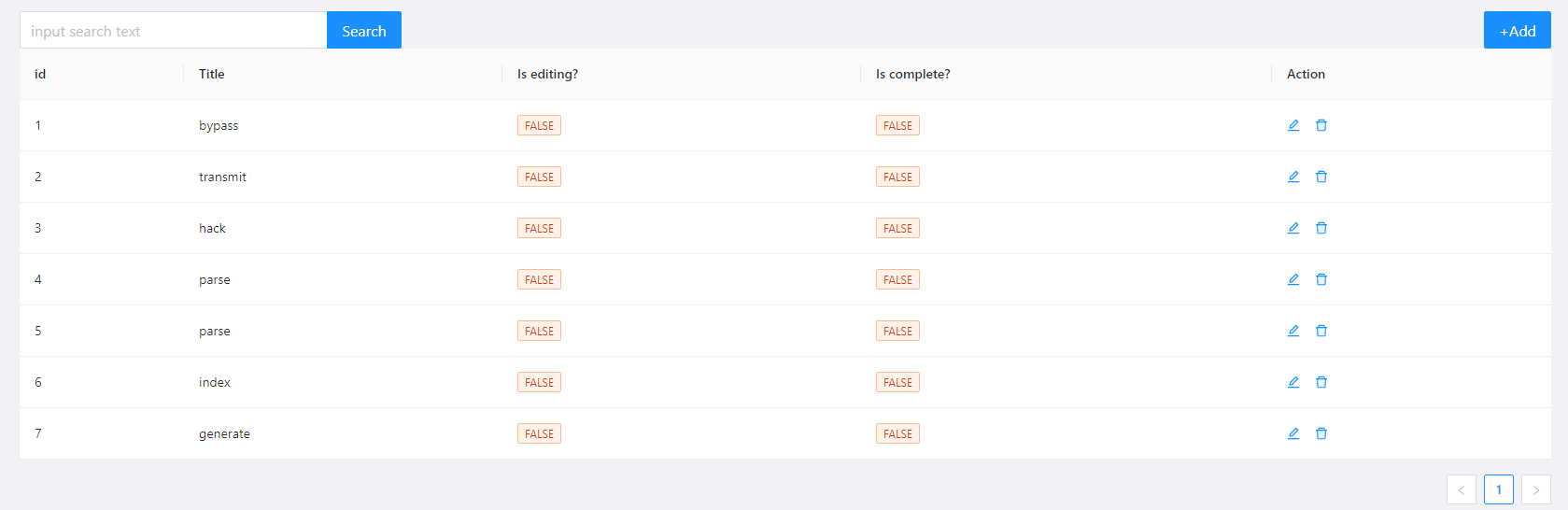
        <Link to="/active">Active</Link>

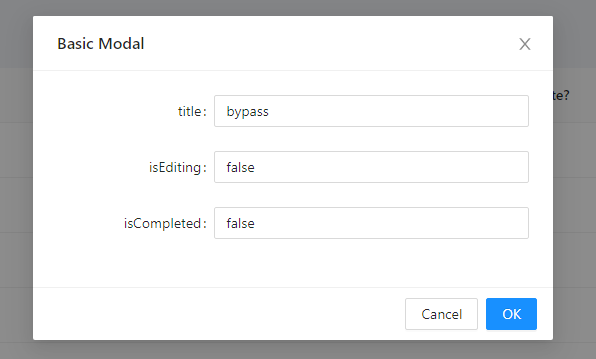
        <Link to="/completed">Completed</Link>

   </div>

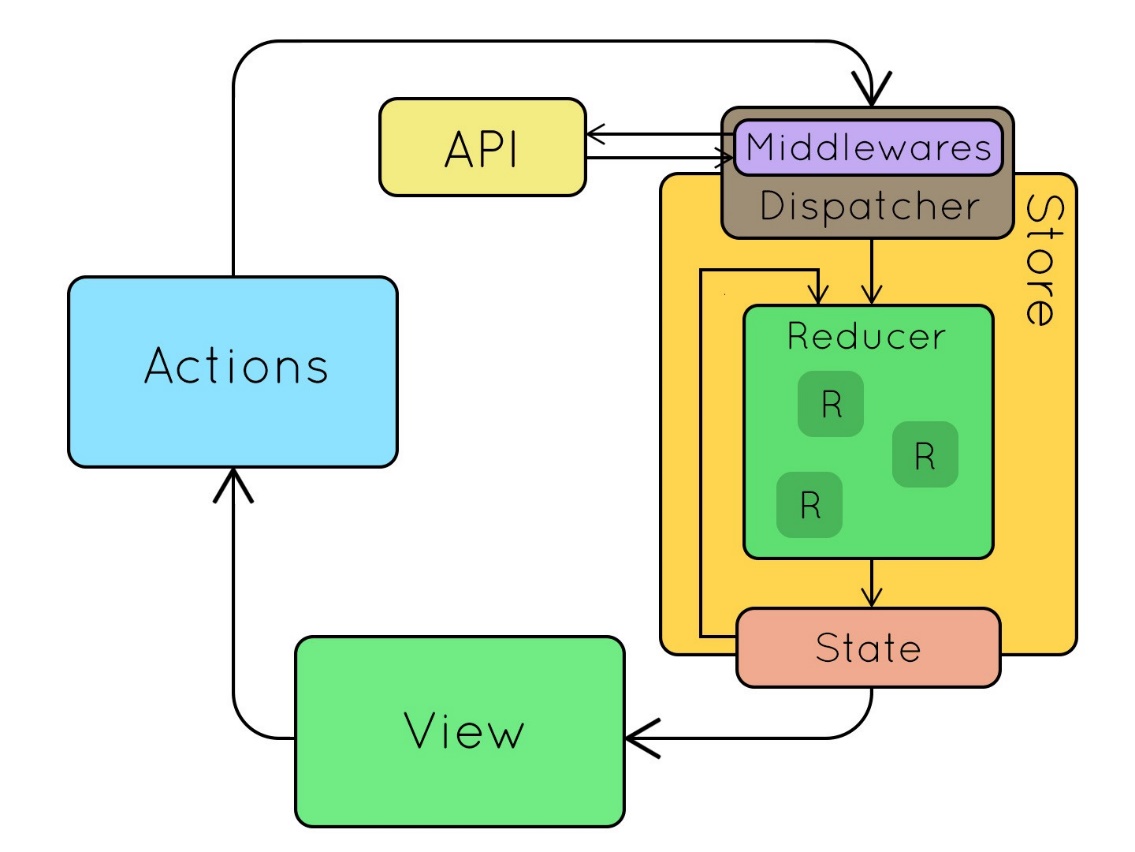
   <button className="clearCompleted" onClick={clearCompleted}>Clear completed</button>

</div>

1. **Ant design**
   1. **Cài đặt vào project**
      * Ta sẽ sử dụng Ant Design để tạo table và navbar điều hướng cho app.
      * Table ở đây sẽ lấy data từ Mock api về và cho phép người dùng thêm xóa sửa cũng như tìm kiếm trên table
      * Trước hết ta định nghĩa table gồm các cột sau :
        + ID : mã todo
        + Title : tên todo
        + Is editing ? : trạng thái isEditing của todo (true/false)
        + Is completed ?: trạng tháng isCompleted của todo (true/false)
        + Action : sửa hay xóa todo
      * Sau đó ta tạo các nút thêm và tìm kiếm trong table
      * Cuối cùng ta tạo các modal để sửa và thêm todo
   2. **Demo**

****

1. **Redux**
   1. **Giới thiệu**
      * Redux ra đời với mục đích quản lý state có thể dự đoán được. cung cấp một vùng chứa state duy nhất và các quy tắc nghiêm ngặt về cách có thể thay đổi state.
      * Có thể sử dụng Redux cho React, Angular, …



* 1. **Concept chính**
     + Store : đây là nơi lưu trữ state và chỉ nên có 1 store trong 1 app
     + Action: một đối tượng JS đơn thuần. Action mô tả rõ ràng lý do tại sao thay đổi state và có thể thực hiện ở bất kỳ đâu trong app. Ví dụ:

{

  type: "ADD\_STUDENT",

  name: "Thu Hương"

}

* + - Reducer: là thành phần để liên kết store với action. Nó nhận state và action làm đối số và trả về state tiếp theo của ứng dụng.
    - Một reducer có thể xử lý nhiều action chính vì thế thường sử dụng cấu trúc switch trong reducer
    - Tốt nhất là cho mỗi reducer quản lý 1 phần state
    - Ví dụ:

function studentsApp(state, action) {

  // Câu lệnh switch

  // Kiểm tra type và trả về sate mới

  switch (action.type) {

    case "ADD\_STUDENT":

      return [ ...state,  action.name ]

    default:

      // Không có case nào khớp

      // thì trả về state hiện tại

      return state

}