**Hướng dẫn làm website todo-app bằng nextjs**

1. **Khởi tạo dự án**

Mở terminal lên ngay thư mục bạn muốn tạo dự án.

Dùng `npx` tạo một dự án nextjs (version 13.4.1) bằng lệnh sau `npx create-next-app@^13.4.1`

Sau đó bổ sung thêm tên dự án và các thông tin khác như sau:

```

npx create-next-app@^13.4.1

√ What is your project named? ... todoapp

√ Would you like to use TypeScript with this project? ... No / Yes

√ Would you like to use ESLint with this project? ... No / Yes

√ Would you like to use Tailwind CSS with this project? ... No / Yes

√ Would you like to use `src/` directory with this project? ... No / Yes

√ Use App Router (recommended)? ... No / Yes

√ Would you like to customize the default import alias? ... No / Yes

```

Nếu thấy dòng chữ `Success! Created <tên dự án> at...` thì bạn đã cài đặt thành công rồi đó!

1. **Setup thư viện**

Chúng ta sẽ cài đặt các thư viện hỗ trợ với npm như sau:

```dependency

npm install @flaticon/flaticon-uicons@latest react-hook-form@^7.48.2 sharp@latest lodash@latest --save

```

Và sau đó, cài đặt các devDependencies bằng câu lệnh sau:

```devDependency

npm install [sass@^1.69.5](mailto:sass@%5e1.69.5) @types/lodash --save-dev

```

1. **Chạy app**

Navigate tới thư mục của dự án bằng: `cd <tên dự án>` ví dụ `cd todoapp`

Nếu là development, chạy `npm run dev`

Để build dự án, chạy `npm run build`

1. **Setup cấu trúc thư mục**

1. Tạo 1 thư mục assets trong public để chứa các file tĩnh như hình ảnh, font chữ, ...v.v...

2. src là thư mục chính của dự án để làm việc. Trong src có 1 folder con là src/app, router của nextjs sẽ phụ thuộc vào

   file system của thư mục này.

3. SSR: Tất cả file nằm trong app đều mặc định là SSR, nếu muốn nó là CSR để sử dụng những cái như hook, modules thì phải

   thêm tag 'use client' lên đầu file (trên import).

4. Trong `/src` tạo các thư mục như sau:

   - components: theo atom design, nhằm tạo các component riêng biệt và dễ tái sử dụng.

   - utils: Chứa các helper và validator. Trong validator có 1 thư mục là `/yup` là nơi cấu hình Yup cho việc validation.

5. Tạo 1 route mặc định (route '/') bằng group như sau: Tạo 1 folder `(default)` bên trong app, tạo 1 file `page.tsx` trong

   `(default)`. File `page.tsx` được xem như file `index.tsx`.

   Okay tạm thời là thế đã, giờ hãy dùng kiến thức HTML CSS (tailwind css) để tạo nên 1 UI cơ bản nào.

1. **Cấu hình context lưu dữ liệu tasks**

**NOTE: Các file trong context phải mark là ‘use client’, vì chỉ có phía CSR mới sử dụng được các hooks.**

Ở đây chúng ta sẽ chỉ dùng **useContext** và **useReducer** có sẵn của react thôi nhé! Tạo 1 thư mục ‘context’ và hãy làm theo các bước sau:

* Trong context, tạo 1 file `type.ts`, trong đó, tạo type cho state và action cũng như context tương ứng nhé. Type này sẽ giúp chúng ta được define rõ ràng và recommend tự động khi code nữa, rất là tiện.



* Tiếp theo đó, tạo 1 context bằng React.context, rồi truyền nó vào useContext để tạo 1 instance cho context đó, mọi thao tác ta sẽ dùng tới nó đó.



* Okay tiếp theo là nó cần 1 reducer, thế thì tạo 1 reducer đơn giản có 4 action là được, 1 là set mới, 1 là update tình trạng isComplete của 1 task, 1 là filter và cuối cùng là add nhé.

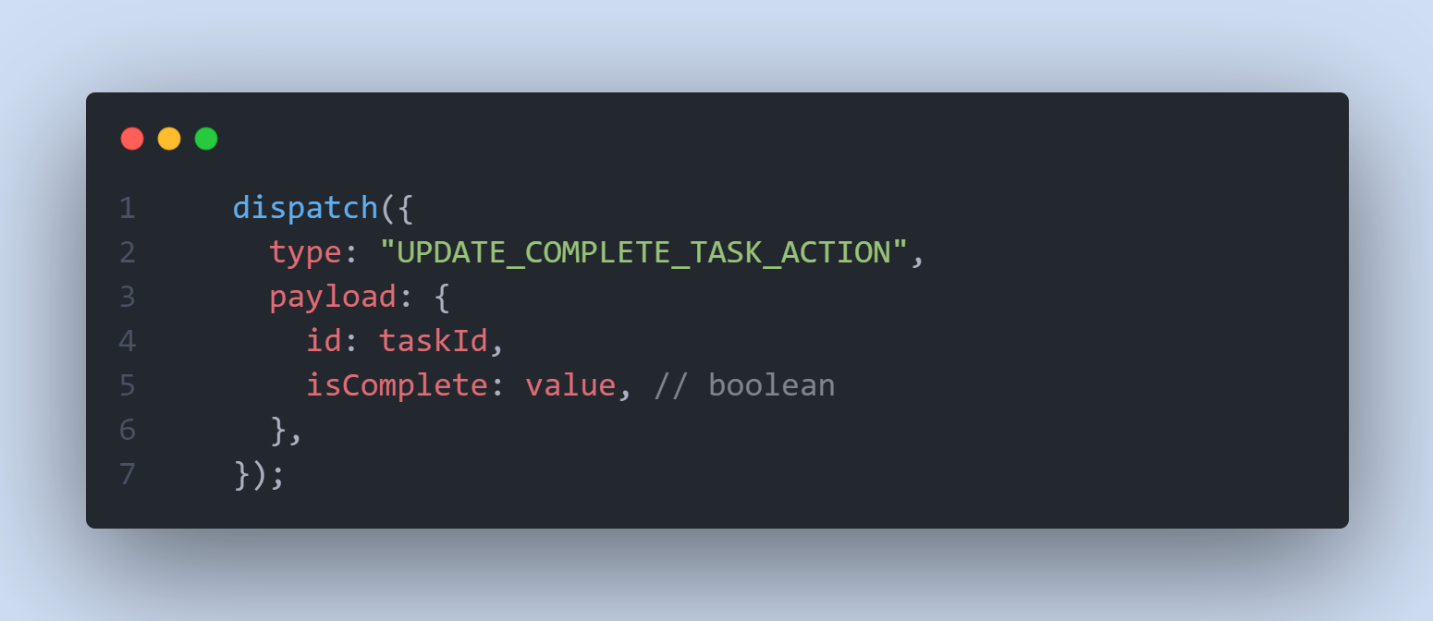




* Okay cuối cùng cũng là đơn giản nhất, tạo 1 provider, cung cấp cho nó state và dispatch (từ useReducer nhằm mục đích giúp cho các component có thể dựa vào dispatch này mà tương tác với context). Cuối cùng là bọc vào toàn bộ app là xong, ở đây mình sẽ bọc vào file layout.tsx của app nhé.



Phần còn lại, để hiển thị task trong component như nào, ta có thể dùng trực tiếp state.taskList từ useTaskContext, hoặc có thể màu mè hơn là sử dụng 1 state lưu trữ nó như sau (không khuyến nghị nếu không có lý do cụ thể):  


**Okay done!!!** Giờ ví dụ ta muốn khi bấm vào checkbox thì task được complete ư, thế thì chỉ cần gọi như sau:

Quá đơn giản phải không, vậy thì mình sẽ skip qua phần này nhé.

1. **Thực hiện filter**

Ta sẽ dùng bối cảnh sau để filter theo tên task: Khi người dùng gõ bất kì chữ cái nào lên ô tìm kiếm, ta sẽ dùng dispatch để gọi action Filter lúc nãy ta đã cài đặt. Ủa sao dễ vậy nhỉ? Vậy thì nâng cấp lên 1 chút, thay vì user gõ 1 chữ ta gọi dispatch 1 lần, thế thì cực kì không tối ưu, do đó mình khuyến nghị sử dụng debounce (có thể tự viết hoặc dùng của lodash cho lẹ). Do đó ở đây mình sẽ dùng của lodash nhé! (debounce nhận 2 tham số, tham số đầu là callback, tham số thứ 2 là timeout)



Okay thế là done filter rồi nhé.

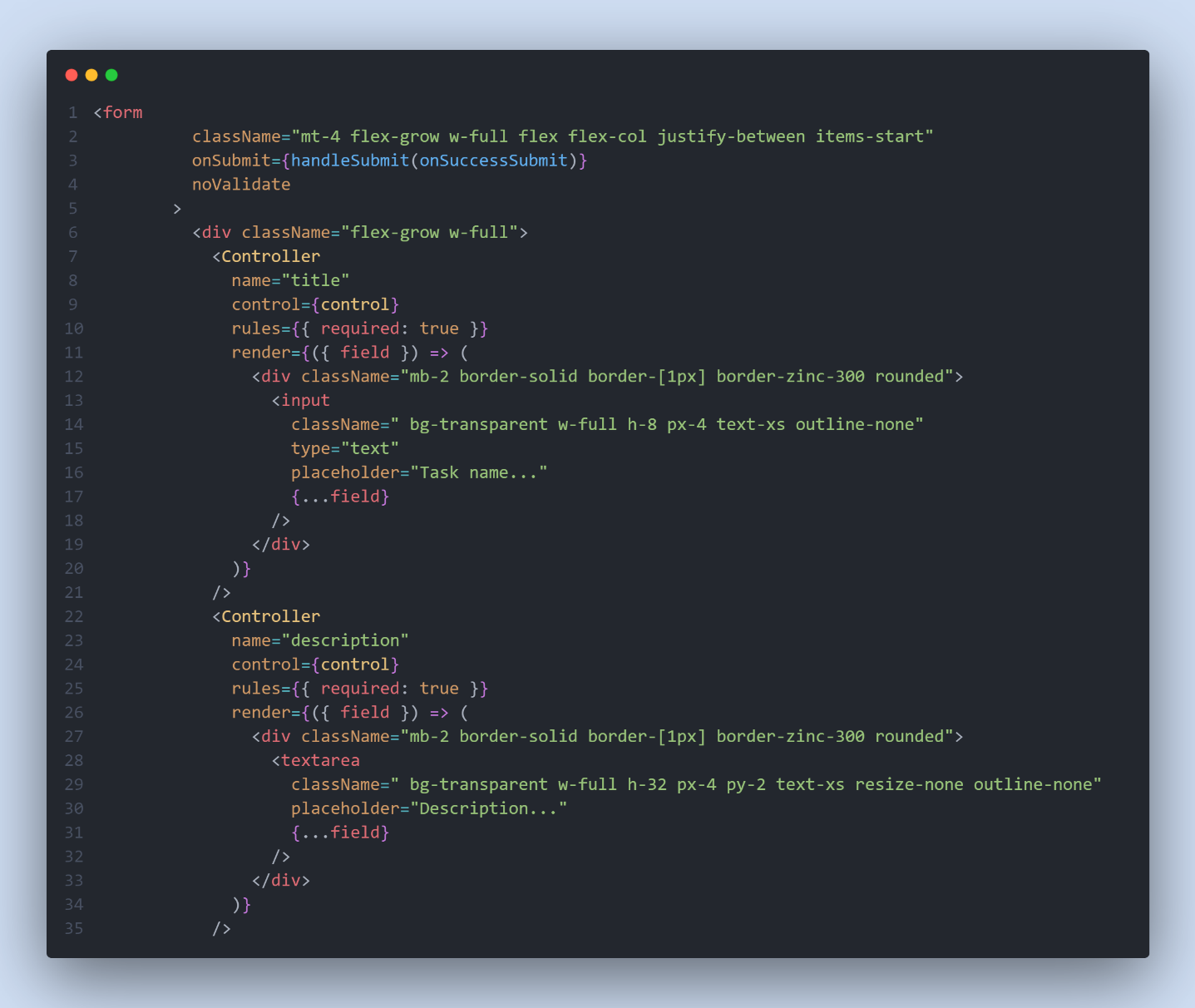
1. **Thực hiện thêm task**

Thêm task thì tương tự, chỉ cần truyền vô dispatch action ADD\_TASK\_ACTION một cái title và description là xong phải không? Thế thì không có chuyện gì để nói cả. Cái quan trọng ở đây là làm thế nào ta quản lý được việc user nhập liệu title, description và truyền nó cho dispatch như thế nào phải không? Vậy thì giờ là sân chơi của react-hook-form rồi, okay Let him cook!

Ta dùng useForm của react-hook-form để định nghĩa 1 form có control (Truyền vô các Controller giúp form quản lý được các giá trị), reset (là phương thức clear toàn bộ các input mà nó quản lý) và cuối cùng là 1 hàm handleSubmit (nhận vô 2 tham số lần lượt là hàm khi submit thành công và hàm khi submit lỗi do user input không đúng rules ta đã đề ra).



Tiếp đó ta phải gán handleSubmit này cho Form, trong form đó có các input, ta phải bao các input đó bằng các Controller của react-hook-form, trong Controller đó phải có name, rules (optional để validate thôi), control (đúng rồi, là cái bạn lấy từ useForm đó) và hàm render bao các input như sau:



Cuối cùng, cần có 1 button bất kì có type là submit để trigger onSubmit của form là xong rồi. Cảm ơn thầy và các bạn đã đọc tới dòng này.

Trân trọng.