# Frontend

## Pages

* Dùng để tạo các trang hiển thị lên frontend

## Redux

### Api

* Dùng để tạo endpoint API với backend
* Kết nối chặt chẽ với Routes

### Constanst

* Dùng để định nghĩa các đường dẫn API cố định

# Backend

## Routes

* Dùng để định nghĩa các đường dẫn API

## Models

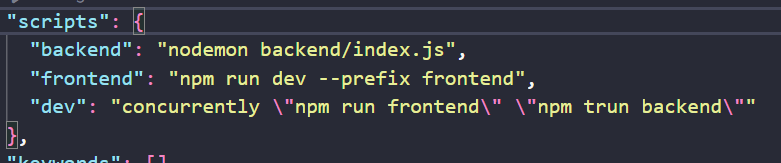
* Dùng để định nghĩa CSDL

## Controllers

* Dùng để định nghĩa, viết code các phương thức

# Các bước Backend

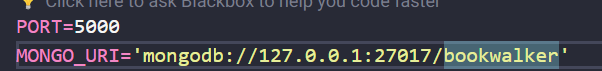
### Thay đổi trong package.json (ngoài cùng)

Đổi mục script:  


### Tạo các folder và file trong backend

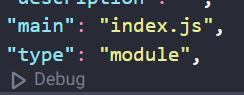
### Trong config cấu hình db (Backend)

* Tạo file .env bên ngoài thư mục



### Cấu hình index.js

* Bên ngoài cùng package.json, tạo type module



## Tạo User

### Tạo userModel

File này xác định schema (cấu trúc) cho tài liệu người dùng trong cơ sở dữ liệu MongoDB. Schema bao gồm các trường như username, email, password và isAdmin.

### Tạo userRoutes

Xác định các route (đường dẫn URL) cho các hoạt động liên quan đến người dùng và ánh xạ chúng với các hàm xử lý tương ứng trong userController.js. Nó cũng sử dụng các middleware authenticate và authorizeAdmin để kiểm tra xác thực và quyền truy cập cho các route nhất định.

Một số route chính bao gồm:

* /api/users (POST): Tạo một người dùng mới.
* /api/users (GET): Lấy danh sách tất cả người dùng (chỉ dành cho admin).
* /api/users/auth (POST): Xác thực người dùng (đăng nhập).
* /api/users/logout (POST): Đăng xuất người dùng hiện tại.
* /api/users/profile (GET): Lấy thông tin người dùng hiện tại.
* /api/users/profile (PUT): Cập nhật thông tin người dùng hiện tại.
* /api/users/:id (DELETE): Xóa một người dùng theo id (chỉ dành cho admin).
* /api/users/:id (GET): Lấy thông tin của một người dùng theo id (chỉ dành cho admin).
* /api/users/:id (PUT): Cập nhật thông tin của một người dùng theo id (chỉ dành cho admin).

### Tạo userController

File này chứa các hàm xử lý logic cho các hoạt động liên quan đến người dùng. Một số chức năng chính bao gồm:

* createUser: Tạo một người dùng mới với tên, email và mật khẩu được mã hóa bằng bcrypt.
* loginUser: Xác thực người dùng bằng cách so sánh mật khẩu nhập vào với mật khẩu đã mã hóa trong cơ sở dữ liệu.
* logoutCurrentUser: Đăng xuất người dùng hiện tại bằng cách xóa JWT khỏi cookie.
* getAllUsers: Lấy danh sách tất cả người dùng (chỉ dành cho admin).
* getCurrentUserProfile: Lấy thông tin của người dùng hiện tại đang đăng nhập.
* updateCurrentUserProfile: Cập nhật thông tin người dùng hiện tại (tên, email, mật khẩu).
* deleteUserById: Xóa một người dùng theo id (chỉ dành cho admin).
* getUserById: Lấy thông tin của một người dùng theo id (chỉ dành cho admin).
* updateUserById: Cập nhật thông tin của một người dùng theo id (chỉ dành cho admin).

### Phụ trợ

Trong userController:

* asyncHandler: được sử dụng để đơn giản hóa việc xử lý lỗi trong các hàm xử lý request của Express.js.

Khi sử dụng asyncHandler, nếu có bất kỳ lỗi nào xảy ra trong các hàm này (ví dụ: lỗi khi truy cập cơ sở dữ liệu, lỗi xác thực, lỗi logic, v.v.), asyncHandler sẽ tự động bắt lỗi và gửi một response với status code 500 (Internal Server Error) và thông tin về lỗi đó trong error.message.

* createToken: Việc sử dụng JWT làm phương thức xác thực và lưu trữ JWT trong HTTP-Only cookie là một cách thực hành phổ biến trong các ứng dụng web hiện đại. Khi người dùng đăng nhập thành công, một JWT sẽ được tạo ra và gửi đến client dưới dạng một HTTP-Only cookie. Sau đó, trong mọi request tiếp theo, client sẽ gửi kèm cookie này để server xác thực và lấy thông tin người dùng từ JWT. Bằng cách sử dụng HTTP-Only cookie, JWT được bảo vệ khỏi các rủi ro bảo mật liên quan đến client-side, đồng thời vẫn cho phép người dùng duy trì trạng thái đăng nhập trong một khoảng thời gian nhất định (30 ngày trong trường hợp này).
* Bcrypt: Mã hoá mật khẩu

# Các bước frontend

* Các thư mục components, pages, redux

## Redux

### API

* Chứa ApiSlice.js và usersApiSlice.js
* ApiSlice.js: là nơi bạn cấu hình RTK Query cho ứng dụng React của mình, bao gồm việc cấu hình baseQuery với URL gốc cho các API request, và định nghĩa các tagTypes cho các thực thể dữ liệu khác nhau trong ứng dụng. Nó cũng chuẩn bị cho việc định nghĩa các endpoint API trong endpoints
* usersApiSlice.js: định nghĩa các endpoint API liên quan đến người dùng bằng cách sử dụng RTK Query (Redux Toolkit Query)

Các endpoint API được định nghĩa bao gồm:

* login: Endpoint để đăng nhập người dùng (POST /auth).
* register: Endpoint để đăng ký người dùng mới (POST /).
* logout: Endpoint để đăng xuất người dùng (POST /logout).
* profile: Endpoint để cập nhật thông tin người dùng hiện tại (PUT /profile).
* getUsers: Endpoint để lấy danh sách tất cả người dùng (GET /).
* deleteUser: Endpoint để xóa người dùng (DELETE /${userId}).
* getUserDetails: Endpoint để lấy thông tin chi tiết của một người dùng (GET /${id}).
* updateUser: Endpoint để cập nhật thông tin người dùng (PUT /${userId}).

1. providesTags, invalidatesTags, keepUnusedDataFor: Đây là các tùy chọn của RTK Query để quản lý cache và vô hiệu hóa dữ liệu trong Redux store khi cần thiết.
2. export const { ... } = userApiSlice;: Export các hook tương ứng với các endpoint API đã định nghĩa, cho phép sử dụng chúng trong các thành phần React để gọi API và truy cập dữ liệu.

Tóm lại, userApiSlice.js là nơi định nghĩa tất cả các endpoint API liên quan đến người dùng trong ứng dụng, sử dụng RTK Query. Các endpoint này sau đó có thể được sử dụng trong các thành phần React khác nhau thông qua các hook tương ứng đã được export.

### Features

* authSlice.js: Sử dụng authSlice.js, bạn có thể quản lý trạng thái xác thực và thông tin người dùng trong ứng dụng Redux của mình. Khi người dùng đăng nhập thành công, bạn có thể gọi action setCredentials để lưu thông tin người dùng vào Redux store và localStorage. Khi người dùng đăng xuất, bạn có thể gọi action logout để xóa thông tin người dùng khỏi Redux store và localStorage.