I. Cấu trúc cơ bản 1 solutions:

- Namespace.MVC hoặc Namespace.Web → Project này chứa giao diện dành cho người dùng, gọi tắt là FrontEnd

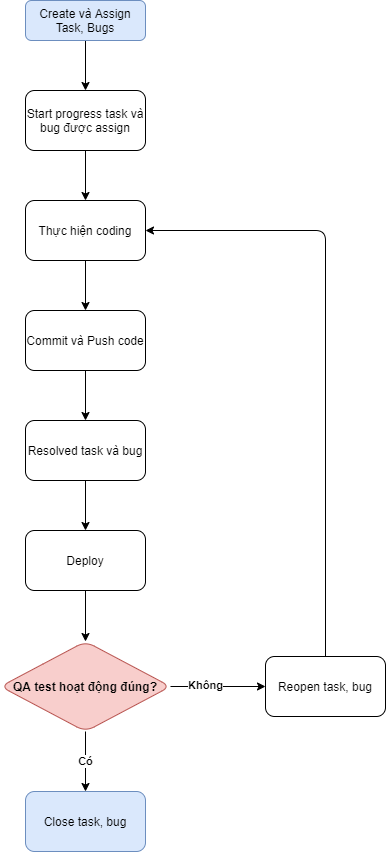
- Namespace.CMS → Project này chứa giao diện và chức năng dành cho quản trị, cập nhật dữ liệu hiển thị ra FrontEnd, gọi tắt là Backend.

- Namespace.Business → Project này chứa các model, views model, các helpers sử dụng chung cho toàn bộ Solutions

- Namespace.DataAccess → Project này chứa các model, views model liên quan đến phần Identity xác thực người dùng.

II. Các thao tác cơ bản khi bắt đầu với solutions

1. Nắm rõ quy trình thực hiện task



2. Đối với các Project FrontEnd hoặc Backend sẽ sử dụng webpack để bundle styles và js, cần cài đặt nodejs để sử dụng.

Trước khi code chạy lệnh npm install để cài đặt các package của giao diện

- Code frontend sẽ nằm trong thư mục src ở ngoài cùng project, các khung dự án đều đã có sẵn Controller, Action và Views dành cho frontend.

III. Một số quy cách đặt tên biến, tệp

- Tất cả các tên tệp đều phân biệt hoa thường nên khi đặt tên phải chú ý.

- Với tên tệp của C#, đặt dạng Camel case, viết hoa chữ cái đầu mỗi từ, ví dụ: DocumentController, UtilityHelper, StringExtesions.

- Quy cách đặt tên biến C#:

+ Tên biến phải tường minh, ko đc đặt kiểu aa, bb, cc, dd…;

+ Với Constant, Enum → Viết HOA toàn bộ, ví dụ: const PI = 3.144444;

+ Với biến toàn cục → Đặt dạng camel case kèm tiền tố \_, ví dụ: \_dbFactory, \_layout…;

+ Với biến nội bộ → Đặt dạng camel case, có kiểu dữ liệu rõ ràng, hạn chế sử dụng var, ví dụ: int a, b; DateTime dt, bool isTrue?;

- Quy cách đặt tên biến trong js:

+ với biến toàn cục → Viết dạng camelCase, kèm tiền tố g\_, ví dụ: g\_Popup, g\_Window, g\_Dom…

+ với biến nội bộ → Khai báo bằng let, viết dạng camel case, kiểu dữ liệu rõ ràng, ví dụ: let numA = 0; let bWrong = true;

- Quy cách viết RestAPI:

+ Tất cả các response đều dùng chung model: RestBase, RestData và RestError

+ Với tham số của action controller, phân biệt rõ từng loại theo [FromBody] hay là [FromForm]

- Quy cách coding:

+ Ưu tiên dùng linq

+ Null check đầy đủ

+ Tối ưu vòng lặp

+ Không đc hardcode

3. Cấu trúc code frontEnd:

- Mỗi giao diện đều có 1 file html để chứa template của html và 1 file js controller để xử lý phía sau, tuy nhiên nếu cần thì cũng có thể chèn thêm file scss vào giao diện đó.

- Tên thư mục chứa template html và js phải trùng với tên controller trên C#, và viết thường

- Tên file html và js phải trùng với Action trả về Views, viết thường.

- Khi tạo mới 1 action trả về Views, phải khai báo trong Webpack theo dạng sau:

const Bundles = [{

hash: true,

title: 'Home',

template: './src/views/pages/home/index.html',

chunkPath: "./src/views/pages/home/index.js",

chunk: 'home.index',

viewName: "Index.cshtml", ←-- tên file view tách riêng

viewPath: "../Views/Home/", ←-- đường dẫn tách riêng

}];

- Lưu ý tên file phải chính xác, hoa thường rõ ràng.

- Để sử dụng multilanguage (i18n) cho các đoạn text trong file view thì làm như sau (lưu ý viết trong file template html), ví dụ đoạn cần translate sang EN:

--→ <%= \_\_('Về di tích') %>

--→ viết theo cú pháp: <%= \_\_(‘nội dung’) %>

- Để sử dụng multilanguage (i18n) cho các đoạn text trong file js thì làm như sau:

--→ \_\_('Về di tích')

--→ viết theo cú pháp: \_\_(‘nội dung’)

- Sau khi đã cấu hình xong, tìm đến file src/static/i18n/en.json, chuyển ngữ như trong file.