Roadmap book

Chương 1. Khai quát dòng thời gian của .NET Core và Angular

Chương 2. Tìm hiểu về workspace của project, caching

Chương 3. Giới thiệu về unit test, ICMP test

Chương 4.

## Getting Ready

### Two players, one goal

Từ bối cảnh của ứng dụng dựa trên nền web với đầy đủ tính năng, chúng ta có thể nói rằng Web API interface với ASP.NET Core framework là a programmatic set of server-side handlers được sử dụng bởi server để phơi một vài hooks and/or endpoints tới a defined request-response massage system. Cái này được biểu diễn bởi cấu trúc XML, JSON hoặc GrapQL. Như chúng ta đã nói, cái này được lấy bởi việc phơi ra **application programming interfaces (APIs)** thông qua giao thức HTTP and/or HTTPS thông qua web server available pbliecly như là IIS, Node.js, Apache, Nginx...

Tương tự như vậy, Angular có thể được mô tả như là một client-side framework hiện đại, giàu tính năng

### The ASP.NET Core revolution

Để tóm lược cá đã xảy ra trong thế giới ASP.NET trong 4 năm trước là một điều không dễ dàng, chúng ta có thể nói rằng chúng ta đã chứng kiến một series về sự thay đổi quan trọng nhất trong .NET Framework từ khi nó sinh ra. Đây là một cuộc cách mạng cái mà đã thay đổi toàn bộ cách tiếp cận của Microsoft tới software development. Để mà hiểu đúng cái mà đã xảy ra xuốt suốt vài năm, nó có thể hữu ích để nhận định một vài key frames đặc biệt, cái mà đã cho phép một công ty biết về its proprietary software, licenses, and patents để trở thành một lực lượng định hướng cho open source development trên toàn thế giới (ý là Microsoft chuyển mình từ 1 thằng chỉ quan tâm độc quyền phần mềm, bằng sáng chế... thành thằng định hướng open source)

Vào tháng 4 năm 2014, tại hội nghị hàng năm Microsoft Build. Nó ở đây, trong một bài phát biểu đáng nhớ, Anders Hejlsberg – bố của Delphi và là lead architect của C# - đã released version đầu tiên của .NET Compiler Platform như là một open source project. Nó cũng là cái mà Scott Guthrie – phó giám đốc điều hành của Microsoft Cloud và AI đã thông báo sự ra mắt chính thức của .NET Foundation một tổ chức phi lợi nhuận nhắm vào việc cải tiến open source sofware development và hợp tác làm việc bên trong hệ thống .NET

Từ cái ngày then chốt đó, .NET development team đã phát hành một loạt các open source project trên nền tảng GitHub, bao gồm: EF Core, TypeScript, .NET Core...

### ASP.NET Core 1.x

Thành tích quan trọng nhất mang lại sự hướng đến open source development là đã release ASP.NET Core 1.0 Q3 2016. Nó thực thi lại một cách hoàn toàn ASP.NET Framework cái mà chúng ta đã biết từ tháng 1-2002 và đã tiến hóa, không có sự thay đổi đáng kể trong kiến trúc nguồn của lên tới version 4.6.2. Framework mới này đã liên kết toàn bộ các công nghệ web app từ trước như là MVC, Web API và Web pages vào bên trong một module duy nhất.

Nhiều bạn tự hỏi rằng cái gì đã xảy ra đối với ASP.NET 5 và Web API 2 vì chúng được sử dụng khác phổ biến cho đến giữa năm 2016

ASP.NET 5 đã được đổi tên để nhấn mạnh thực tế rằng nó viết lại hoàn toàn.

Web API 2, nó đã là một framework phục vụ việc xây dựng HTTP services cái mà trả về JSON hoặc XML data thay vì web pages. Ban đầu nó sinh ra để thay thế MVC platform, nó vũng đã gộp với các bản cũ vào trọng một bản mới

The 1.0 final release được nói gắn gọn bởi ASP.NET Core 1.1, cái mang lại nhiều tính năng mới và cải tiến hiệu năng, và nhiều bug cũng đã được thảo luận. Nhiều tính năng mới được thêm vào để có thể config middleware như filter, built-in, host-independent URL module được viết lại

### ASP.NET Core 2.x

### ASP.NET Core 3.x

Được release vào tháng 9 năm 2019 với việc cải thiện hiệu năng và bảo mật và các tính năng mới, chẳng hạn như Window desktop application hỗ trợ với khả năng importing được cải tiến cho Winform và WPF applications, C# 8 support,

### SPAs, NWAs, and PWAs

* Single – page application

Một SPA là một ứng dụng dựa trên nền web, cung cấp các tính năng và trải nghiệm như trên desktop. Một SPA tốt, 2 thứ là contents và resources – HTML, JS, CSS và hơn nữa có thể lấy được bên trong một single page load khi cần. Cái này cũng có nghĩa rằng page không reload hoặc refresh, nó chỉ thay đổi các thành phần chịu trách nhiệm tới user actions.

* Native web application

Multi-pages applications, cái này được phổ biến từ năm 2015, chúng được gọi chung là NWAs bởi vì chúng có xu hướng thực thi a number of small-scale single-page bọc cùng nhau

* Progressive web application

Trong năm 2015, web development pattern khác đã được phát hành, thằng này có lợi thế của các tính năng quan trọng được hỗ trợ bởi browsers hiện đại: servoce workers và web manifest files. Đây là 2 cải tiến quan trọng có thể được sử dụng một cách thành công để cung cấp nhiều tính năng thường chỉ có trên mobile apps – push các thống báo, offline mode, permission-based hardware access, và nhiều hơn nữa – sử dụng cũng dựa trên web-based development tools giống như HTML,CSS, và JS

* Mặc dù có nhiều điểm tương đồng, PWAs và SPAs là 2 nội dung khác nhau, có yêu cầu khác nhau và các khía cạnh quan trọng khác nhau. Như chúng ta có thể thấy không có yêu cẩu nào của PWA đã đề cập từ trước liên quan tới Single-Page Applications hoặc server-side round tríp. Một Progressive Web App có thể làm việc bên trong một single HTML page và AJAX-based request, nhưng nó cũng có thể request server-rendered pages khác và thể hiện standard HTTP GET hoặc POST requests, giống như một MPA. Bất cứ SPA nào cũng có thể thực thi một PWA, phụ thuộc vào sản phẩm được yêu cầu ntn, thẻ server-side và client-side framworks đã gây dựng và mục tiêu cuối cùng của developer.

Khi chúng ta sử dụng Angular, nó là tất cả về developing SPAs...

### Product owner expectations

### A sample SPA project

### Preparing the workspace

Điều đầu tiên chúng ta làm là set up workstation của chúng ta; nó không khó bởi vì chúng ta chỉ cần một vài tools thiết yếu. Những cái này thì bao gồm VS 2019, Node.js , a development web server (giống như built-in IIS Express), một hệ thống điểu khiển source code như Git

### Disclaimer – do (not) try this at home

Có vài cái quan trọng cần hiểu trước khi xử lý.

Cuốn sách này sẽ sử dụng sâu về một số các tools programming khác nhau, Components mở rông, thư viện bên thứ 3 và nhiều hơn nữa. Phần lớn trong số họ là TypeScript, NPM, NuGet, most .NET Core frameworks/packages/runtimes và chúng cùng được shipped tới VS 2019, trong khi đó các cái khác như Angular, nó yêu cầu JS dependecies và server-side và client-side packages bên thứ 3 sẽ được lấy từ các repositoryes chính thức của họ.

### Setting up the project

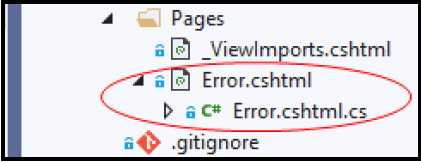
1. Download and install the .NET Core SDK
2. Check that the .NET CLI will user that SDK version
3. Create a new .NET Core and Angular project
4. Check out the newly created project within Visual Studio
5. Update all the packages and libraries to our chosen versions

## Chapter 2- Looking Around

### The .NET Core back-end

Razor Pages

Razor Pages được giới thiệu trong .NET Core 2.0 và đại diện cho một cách thay thế tới ASP.NET Core MVC pattern. Một Razor Pages tương tự một Razor view, với cũng syntax và tính năng, nhưng nó cũng chứa controller source code – Cái đã được đặt trong một file



Controllers

Nếu một Razor Page bao gồm controller thì tại sao chúng ta vẫn có một /Controller/ folder ? Lý do khá đơn giản: không phải tất cả controllers thì có nghĩa là để serve server-rendered HTML pages. Ví dị, họ có thể output một JOSM output (REST APIs), XML-based response (SOAP web services), một static hoặc dynamically-created resource (JPG, JS, and CSS files), hoặc thậm chí một HTTP response đơn giản mà không cần content body.

Thực tế, một trong những lợi ích quan trọng nhất của Razor Pages là chúng cho phép tách rời giữa HTML content – cái mà ta gọi là pages và toàn bộ HTTP response – cái có thể được định nghĩa là services APIs. .NET Core + Angular template của chúng ta hỗ trợ đầy đủ việc chia này, cái này có 2 lợi ích chính.

* Separation of concern: việc sử dụng các pages bắt buộc chia giữa cách cách mà chúng ta load server side pages (1%) và cách chúng ta serve APIs (99%) của chúng ta. Tỉ lệ % đã show hợp lệ cho kịch bản xác định của chúng ta.
* Single responsibility

Configuration files

Bây giờ chúng ta cũng nhìn vào root-level với các configuration files và xem xem mục đính của chúng là gì: Program.cs, Startup.cs, appsetting.json. Các files này chứa configuration của ứng dụng web, bao gồm các modules và middlewares, settings biên dịch và publishing rules.

WeatherForecast.cs file nó chỉ là một class được thiết kế để deserialize the JSON objects đã trả về bởi WeatherForecastController, nói cách khác thì nó là JSON View Model – một view model để chứa deserialized JSON objects.

Program.cs

Program.cs file có mục đích chính là tạo một WebHostBuilder, một object cái sẽ được sử dụng bởi .NET Core framework để set up và build IwebHost và cái này sẽ host web app của chúng ta.

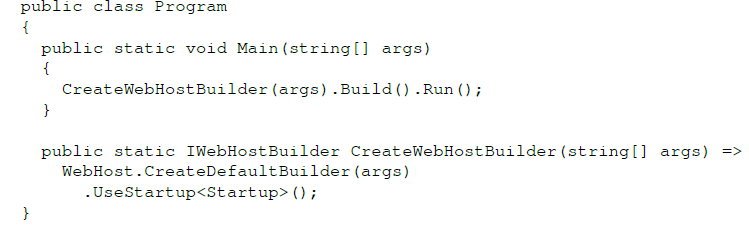
Web host vs web server

Web host là gì? Một host là the execution context của bất cứ ASP.NET Core app. Trong một application dựa trên web-based, host phải thực thi IwebHost interface, cái này phơi một collection của web-related features và services và cũng có một method là Start. The web host references the server cái mà sẽ xử lý request.

Web host chịu trách nhiệm cho application startup và lifetime management, trong khi đó server chịu trách nhiệm cho việc chấp nhận HTTP requests. Một phần trách nhiệm của host bao gồm việc chắn chắn rằng application’s services và server thì đã có và được confiured đúng.

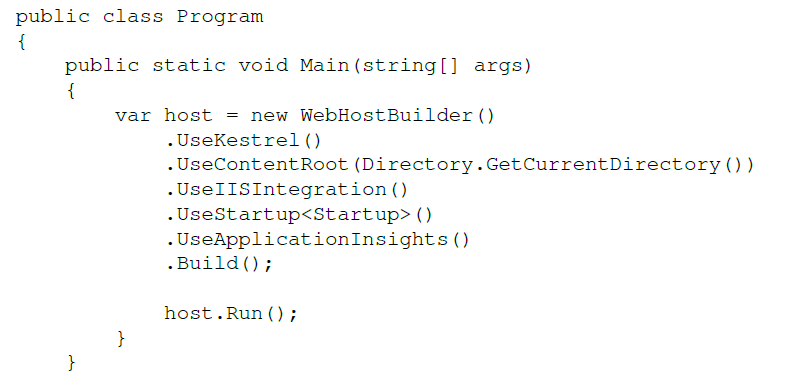
Chúng ta có thể nghĩa là host thì đang gói xung quanh server: the host is configured to user a particular server, while the server is unaware of its host.

Nếu chúng ta mở Program.cs file



WebHost.CreateDefaultBuilder(args) đã được giới thiệu trong .NET Core 2 và là một cải tiến tuyệt vời so với các phiên bản của nó, vì nó đơn giản hóa source code được yêu cầu để set up base use cases, vì vậy việc makling nó dễ dàng hơn để bắt đầu với một new project.

Để hiểu tốt hơn thì hãy xem một sample Program.cs trong .NET Core 1.x:



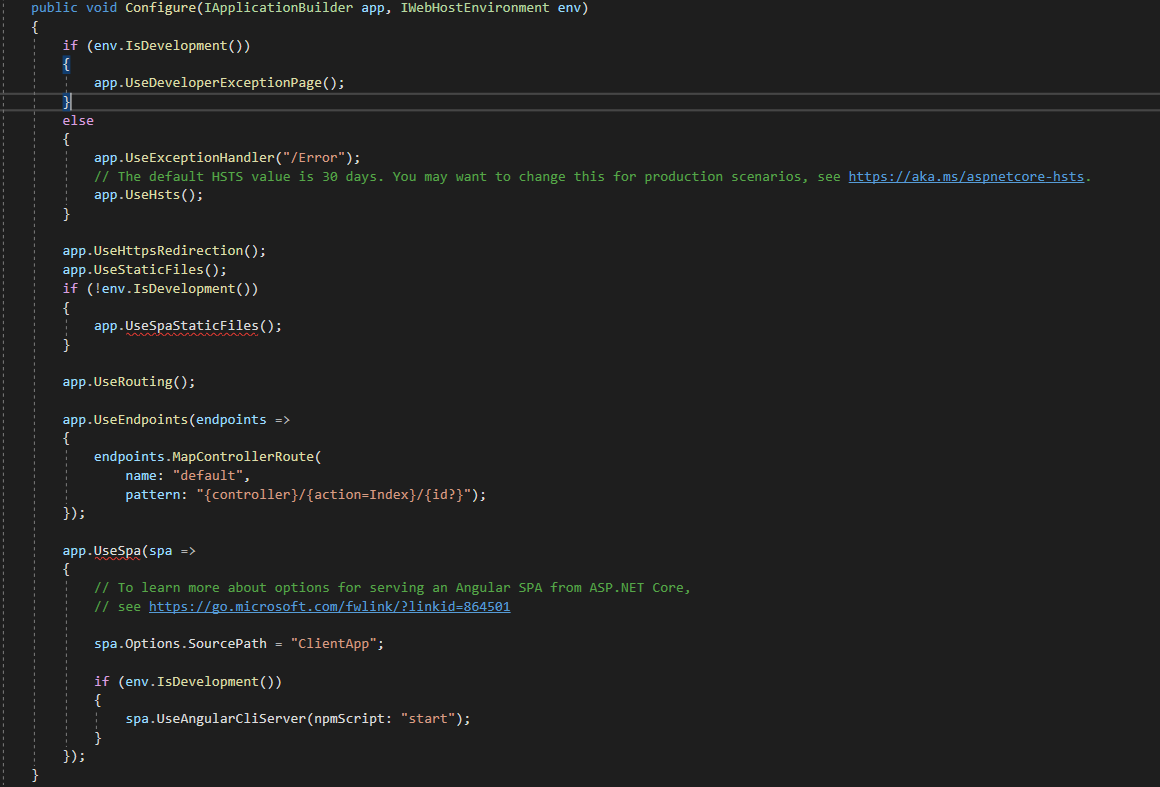
Cái này đã thực hiện các bước như sau:

1. Setting up the Kestrel web server
2. Setting up the content root folder
3. Setting IIS Integration
4. Defining the Startup class to use
5. Finally, executong Build and Run on now configured IwebHost

Startup.cs

Trong .NET Core, the Startup.cs file là nơi mà đặt chúng ta có thể làm theo như sau:

* Add và configure Services và Dependency Injection, trong ConfigureService () method
* Configure một HTTP request pipeline bởi việc thêm the required middleware, trong Configure () method



Code trên khá dễ đọc, chúng ta có thể hiểu các vấn đề như sau:

* Config đầu tiên là để xử lý runtime exeptions trong development và production
* Ngay sau đó, chúng ta có thể nhìn first block of middlewares: HttpsRedirection, để xử lý HTTP –to-HTTPS redirects;StaticFiles, để serve static files đặc ở dưới /wwwroot/ folder; và SpaStaticFiles, để serve static files trong /ClientApp/src/assets/ folder. Không có 2 thằng middlewares này, chúng ta không thể serve locally hosted assets như là JS,CSS và images; đây là lý do mà họ trong pipeline. Chú ý một điều là các method được gọi với không parameters: điều này chỉ có nghĩa chúng là setting mặc đinh là đủ cho chúng ta
* Ngay sau đó, Endpoints middleware, cái này sẽ add the required routing rule(s) để map HTTP requests tới Web API controllers của chúng ta
* Cuối cùng là UseSpa middleware, cái này được add tới HTTP pipeline với 2 configuration settings. Đầu tiên khá dễ hioeeur: nó chỉ là the source path của Angular app’s root folder. Nó là /ClientApp/ folder. Phần thứ 2 sẽ chỉ được executed trong development scenarios, UseAngularClientServer () method nói với .NET Core để pass thông qua tất cả các requests addressed tới Angular app

How the HTTP request received by our web application will be handled

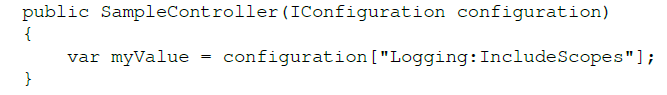
1. Mỗi request sẽ được nhaanbj bởi the .NET Core back-end, cái sẽ thử để xử lý nó tại server-side level bởi việc checking các middlewares registered trong HTTP pipeline: trong kịch bản cụ thể của chúng ta, chúng ta sẽ check đầu tiên static files trong /wwwroot/ folder, sau đó static files trong /ClientApp/src/assets/ folder, sau đó routes mapped tới Web API controllers/endpoints
2. Nếu một trong các middlewares đã đề cập bên trên phù hợp và xử lý request, the .NET Core back-end sẽ take care nó; ngược lại, the Spa middleware sẽ pass request tới Angular client-side app, cái này sẽ xử lý nó băng việc sử dụng client-side routing rules của nó.

Startup.cs

appsetting.json

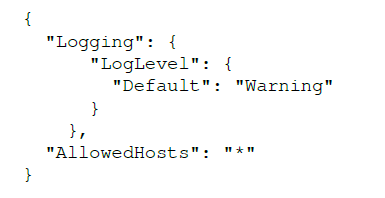
The appsetting.json file chỉ là nơi cho Web.config file cũ; với syntax là XML.

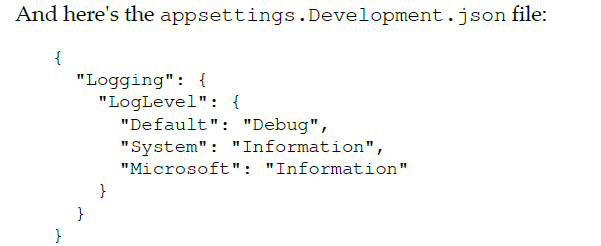
Một khi đã lấy được, chúng có thể dễ dàng được truy cập vào bên trong code của chúng ta bằng việc sử dụng DI thông qua một vài strings



Có một điều đáng chú ý là có một appsettings.Development.json file được lồng bên dưới.

Tóm lại, các file này được add thêm vào để được sử dụng tới configuration xác định cho các environments xác định.





Như ta có thể nhìn. The Loggin.LogLevel.Default value được set là Warning trong file đầu tiên, tuy nhiên khi mà app của chúng ta chạy trong development mode thì file thứ 2 sẽ ghi đè giá trị và setting nó là Debug

### The Angular front-end

Workspace

The Angular workspace là hệ thống file đặt tại nơi đang chưa the Angular files: một tập hợp của các file, libraries, assets và hơn nữa. Trong template của chúng ta, worksapce được đặt bên trong /ClientApp/ folder cái này thì được định nghĩa là workspace root.

The workspace thường được tạo và khởi tạo bởi CLI command được sử dụng để tạo app, bất cứ CLI commands operating trên app hoặc libraries của họ sẽ được executed từ bên trong workspace folder.

angular.json

Vai trò quan trọng nhất bên trong workspace được điều khiển bởi angular.json file, được tạo bởi CLI trong workspace root: đây là workspace configuration file và chứa workspace-wide và project-specific configuration defaults cho tất cả build và development tools được cung cấp bởi Angular CLI.

Các properties tại top của file định nghĩa workspace và project configuration

* version: The configuration file version
* newProjectRoot: The path nơi mà new projects đã được tạo, relative với workspace root folder. Giá trị này set tới projects folder, cái mà thậm chí k tồn tại. Điều này là hoàn toàn bt vì workspace của chúng ta đang chứa 2 Angular projects trong 2 folder đã định nghĩa: HealthCheck Angular app của chúng ta, vị trí trong /ClientApp/src folder, và end-to-end tests, vị trí trong /ClientApp/e2e folder. Vì vậy, ở đây không cần định nghĩa newProjectRoot – và nó cũng quan trọng để không sử dụng mooetj existing folder để tránh rủi ro của việc ghi đè existing.
* projects: một container item cái mà hosts một sub-section cho mỗi peoject trong workspace, chứa configuration-options của project xác định
* defaultProject: the default project name – bất cứ CLI command cái mà không có thêm optional xác định project name thì sẽ executed trên project này

Tip: angular.json file follow tiêu chuẩn generic-to-specific cascading rule. Tất cả configuration values set tại workspace level sẽ là default values cho bất kì projects, và có thể được override bởi set tại the project level. Các cái này lại có thể được overridden bởi command-line values khi sử dụng CLI

package.json

The package.json file là the Node Package Manager (npm) configuration file; nó chứa danh sách npm packages cái mà developer cần để restored trước khi project starts.

npm started its life as the default package manager for the JS runtime environment known as Nodejs (môi trường quản lý của js). Trong vài năm gần đấy, nó cũng được sử dụng để host a number of independent JS projects, libraries, and frameworks of any kind, including Angular. Cuối cùng nó cũng trở thành the de facto package manager cho JS framework và tooling. Người nào chưa bao giờ sử dụng nó có thể nghĩ nó là NuGet cho thế giới của JS

Thông qua npm phần lớn là command-line tool, cách dễ nhất để sử dụng nó từ VS là configure package.json file chính xác việc chứa tất cả npm packages chúng ta cần để get, restore và keep up-to-date later on. Các packages đã nhận được từ downloaded nằm ở trong /node\_modules/ folder, cái này thì bị ẩn bởi mặc định vủa VS; tuy nhiên, tất cả packages đã lấy có thể được nhìn thấy từ npm virtual folder. Miễn là chúng ta add, delete, hoặc update package.json file, VS sẽ tự động update folder này theo.

Trong Angular SPA template chúng ta đã sử dụng package.json chứa một số lớn các packages – tất cả Angular packages – cộng thêm một bó các dependencies, tools, và utilities thứ 3 như là Karma.

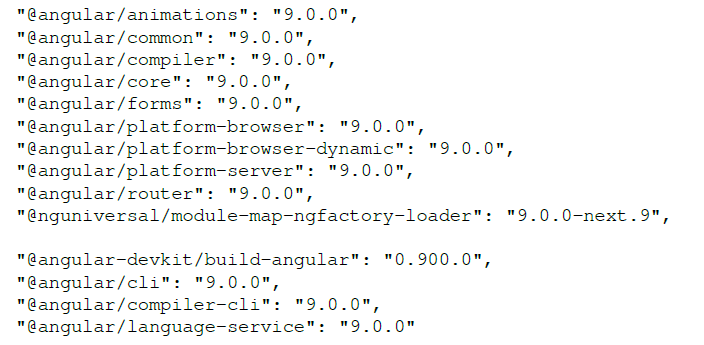
Trước khi di chuyển đến phần đầu, hãy nhìn lại package.json file, chúng ta có thể nhìn thấy tất cả packages đã được liệt kê bên trong một standard JSON object theo kiểu key-value; the package name là key, trong khi đó value thì được sử dụng để xác định version number. Chúng ta có thể input chính xác build numbers hoặc sử dụng standard npmJS syntax để xác định auto-update rules xung quanh custom version rages.

* Dấu ngã (~): một giá trị của “~1.1.4” sẽ phù hợp tất cả 1.1.x versions, bao gồm 1.2.0,1.0.x,...
* Dầu mũ (^): một giá trị của “^1.1.4” sẽ phù hợp mọi cái bên trên 1.1.4, bao gồm 2.0.0...

Upgrading (or downgrading) Angular

The Angular SPA template sử đụng các version đã fix cho tất cả Angular-related packages; đây chắc chắn là lựa chọn sáng suốt vì chúng ta k đảm bảo được rằng version mới hơn sẽ có thể tích hợp hiền mạch với code cũ mà không gặp lỗi nào. Chắc chắn các phiên bản sẽ tăng theo thời gian.

Dưới đây lkaf Angular packages and releases quan trọng nhất cái sẽ được sử dụng trong cuốn sách này



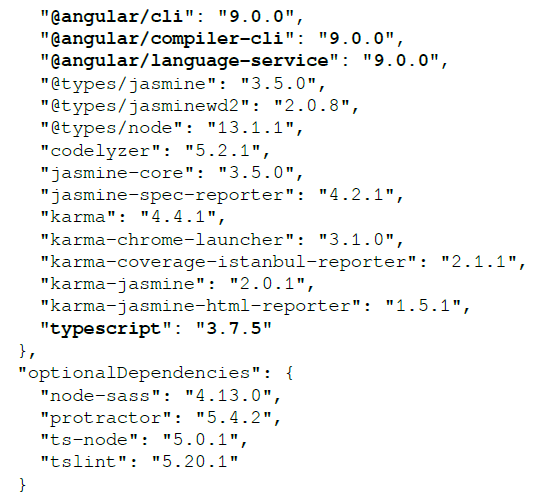
Chúng tôi có thể dễ dàng thực hiện upgrade – hoặc downgrade – bởi việc tahy đổi version, miễn sao chúng ta lưu the file, VS sẽ tự động fetch new versions thông qua npm.

Upgrading (or downgrading) the other packages

Nếu chúng ta upgrade (or downgrade) Angular to 5.0.0, chúng ta cũng cần take care một danh sách các npm packages cái mà cần updated.

Ở đây là danh sách đầy đủ (bao gồm cả Angular packages) chúng ta sẽ sử dụng trong package.json file , chia nó vào bên trong dependencies, devDependencies và optionalDependencies sections: packages quan trọng được highlight trong nippet





tsconfig.json

The tsconfig.json file là file configuration TypeScript.

TS free, open source, hoạt động như một JS superset; có nghĩa rằng bất cứ JS program thì cũng hợp lệ TS program. TS cũng compile tới JS. Lý di chính để sử dụng nó để vượt qua giới hạn syntax của JS và thiếu sót khi phát triển các dự án lớn và phức tạp.

Một số lý do sử dụng: một trong những điểm quan trọng nhất:

* TS có một số tính năng vượt qua JS, như kiểu static, classes, và interfaces. Sử dụng nó trong VS cũng cho chúng ta cơ hội tới tính hiệu quả từ built-in IntelliSense,
* Cho một project lớn về client-side, TS sẽ cho phép chúng ta tăng tính mạnh mẽ của code.

Quay trở lại với tsconfig.json file, ở đây không có nhiều cái để nói; các giá trị option được sử dụng bởi Angular SPA template thì nhiều hoặc ít hơn cái chúng ta cần configure với VS và TS compiler (TSC). Chúng ta có thể điều chỉnh:



Chúng ta đã thêm mợ angularCompilerOptions section, cái này có thể được sử dụng đẻ config hành vi của Angular AoT compiler. strictMetadataEmit setting chúng ta đã thêm sẽ nói compiler để report syntax ngay lập tức hơn là tạo ra một .metadata.json eror log file.

Other workspace-level files

Có vài file được tạo bởi CLI trong workspace root. Chúng ta sẽ không thay đổi chúng, chúng ta chỉ tóm tắt chúng theo danh sách:

* . editorconfig: config cho code editors.
* .gitignore:
* README.md:
* package-lock.json: cung cấp thông tin version cho tất cả packages đã cài trong /nodes\_module/ folder bởi npm client. Nếu bạn có kết hoạc thay thế npm với Yarn, bạn có thể xóa file này và yarn.loack file sẽ được tạo thay thế
* tslint.json: Default TSLint config opetions cho toàn bộ projects trong workspace. Đây là các rules sẽ được tích hợp hoặc ghi đè với project-specific tslint.json fle bao gồm trong project root folder (TSLint là một extensible static analysis tool cái mà checks TS code phục vụ việc dễ đọc, dễ maintain và chuecs năng errors).

The /ClientApp/src/folder

Bằng việc mở rộng /ClientApp/src/ directory, chúng ta có thể nhìn thấy các sub-folders

* /ClientApp/src/app folder chứa tất cả TypeScript files có liên quan tới Angular app của chúng ta: nói cách khác, toàn bộ client-side application source code thì được đặt tại đây.
* /ClientApp/src/app/asets/ folder lưu tất cả images của ứng dụng và các file tài sản khác: Các files này sẽ được copied hoặc updated as-í tring the /wwwroot/ folder mỗi khi application build
* /ClientApp/src/app/environment/ folder chứa build configuration options cái mà mục đích để xác định environments

Cũng có một bó các files ở root-level

* browserslist: Configures việc chia sẻ của browsers đich và Node.js version bên trong các front-end tools
* index.html: The main HTML page cái mà được phục vụ khi một ai đó ghé thăm site của chúng ta. CLI tự động add tất cả JS và CSS files khi xây dựng app, vì vậy bạn không cần add bất cứ <script> hoặc <link> tags
* karma.còn.js: Application-specific Karama configuration. Karma là một tool được sử dụng để chạy Jasmine-based tests: chúng ta có thể bỏ qua một cách an toàn trong toàn bộ topic
* main.ts: The main entry point cho your application.
* polyfills.ts: Cung cấp polyfill scripts hỗ trợ browser
* styles.css: một danh sách của CSS files cái hỗ trợ styles cho project
* test.ts: The main entry point cho unit test của project
* tsconfig.\*.json: Project-specific configuration options cho các khía cạnh ứng dụng của chúng ta: .app.json cho application-level, .server.json cho server-level, và .specs.json cho tests. Các options này sẽ được khi đè những set trong tsconfig.json file trong workspace root
* tslint.json: The TSLint configuration cho project hiện hành

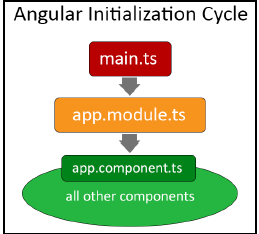
The /app/ folder

/ClientApp/src/app/ folder follows Angular folder structure và chứa logic và data của project của chúng ta, vì vậy việc bao gồm Angular Modules, Services, và Components. Nó cũng chỉ là sub-folder

AppModule

Như chúng ta đã tóm tắt trong Chapter 1, the basic building blocks của một Angular application là NgModules, cái mà cung cấp a compilation context for Components. Vai trò của NgModules là để thu thập code có liên quan vào bên trong một bộ chức năng: vì vậy, toàn bộ Angular app thì được ddienhj nghĩa bởi một bộ của một hoặc nhiều NgModules

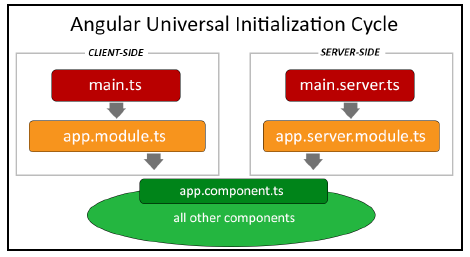
Một Angular app yêu cầu một root module – được gọi theo quy tắc là AppModule – cái mà nói với Angular cách để lắp ráp application, vì vậy cho phép việc khởi động (bootstrapping) và starting the initialization life cycle. Các modules còn lại được biết là modules tính nawg và phục vụ các mục đích khác nhau. The root module cũng chứa một reference list của tất cả Component available



Như chúng ta có thể nhìn thấy, main.ts file bootstrap app.module.ts (AppModule), cái mà sau đó load app.component.ts file (AppComponent); sau đó như chúng ta sẽ thấy sẽ là load tất cả các Components mỗi khi app cần chúng.

Server-side AppModule for SSR

Như chúng ta có thể nhìn, the /ClientApp/src/app/ folder cũng chứa một app.server.module.ts file cái mà sẽ được sử dụng để cho phép Angular Universal Serverside Rendering (SSR) – một công nghệ cái mà renders Angular application trên server, được cung cấp tại back-end framewrok hỡ trợ nó. The template đã gereated ở file này bởi vì .NET Core hỗ trợ như tính năng tiện lợi



### AppComponent

Nếu NgModules là Angular building blocks, Components có thể được định nghĩa như là các viên gạch được sử dụng để đặt cùng nhau trong app, tóm lại thì Angular app là dựa trên tree of Components đang làm việc cùng nhau

Components định nghĩa views, cái mà là một tập hợp các elements screen, cái mà Angular có thể chọn và chỉnh sử theo logic và dữ liệu chương trình của chúng ta, và sử dụng services, cái mà cung cấp tính năng xác định không có mối liên hệ trực tiếp với views. Service providers cũng có thể được inject vào trong Components như dependencies, vì vậy việc app code modular, reusable, and efficient.

Nền tảng của các Components này thì được gọi quy ước là AppComponent

### Other components

khác với AppComponet, trong template có chứa 4 Components, mỗi một thằng được nuôi trong một folder

* CounterComponet: Đặt trong counter subfolder
* FetchDataComponent
* HomComponent
* NavMenuComponent

### Our first app test

Trước khi chạy test, nó có lẽ hữu ích cho voeecj hiểu một vài thông tin về Jamine và Karma (test framework). Nếu bạn không biết về bất cứ cái gì ở chúng, không cần lo lắng – bạn sẽ sớm biết. Bây giờ, chỉ cần biết rằng Jasmine là một open source testing framework cho JS cái mà có thể được sử dụng để define tests, trong khi Karma là một test runner tool cái mà tự động sản sinh một web server cái mà sẽ eexecute JS soure code trống lại Jasmine-made tests và output kết quả tương ứng của chúng trên command line.

Trong lần test nhanh này, chúng ta sẽ giới thiệu cơ bản Karma tới execute the source code của Angular sample của chúng ta đối với Jasmine tests đã định nghĩa bởi template trong counter.component.spec.ts file.

Mở Command Prompt, navigate tới <project>/ClientApp/ folder sau đó execute theo command:

* npm run ng test

## Getting to work

### Static file caching

Hãy bắt đầu với server-side task. Bạn có nhớ rằng /wwwroot/test.html file chúng ta đã thêm khi chúng ta muốn check cách mà StaticFiles middleware làm việc ? Chúng ta sẽ sử dụng nó để mô tả nhanh cách mà ứng dụng của chúng ta sẽ cache nội bộ static files.

Việc cache là chúng ta đã tạo một client-cached version trên page, nên khi ta vào lại địa chỉ đó chúng ta không thấy thay đổi mặc dù code đã có thay đổi

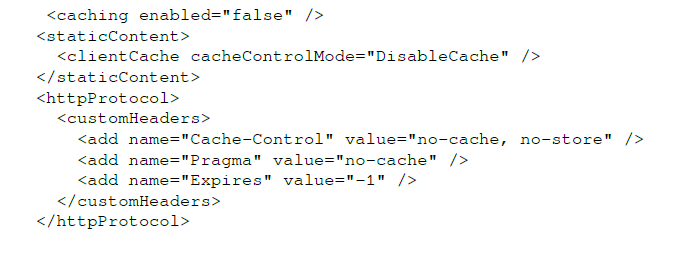
Việc caching này sẽ là tốt trên server production nhưng với development thì khá khó chịu. May mắn thay, trong memory AngularCliServer đã cung cấp Spa middleware trong khi dev sẽ tự động fix lỗi này cho tất cả TypeScript files, và cũng cho tất cả static assets. Tuy nhiên cái gì được phục vụ thông qua back-end? Chúng ta sẽ có một vài static HTML files, favicons, image file, audio files, và các thứ khác cái mà chúng ta sẽ phục vụ trực tiếp trên server.

Vậy có cách nào để tinh chỉnh hành vi caching cho static files? nếu có, chúng ta có thể set up các hành vi khác nhau cho debug/dev and release/production ?

Câu trả lời là có cho cả 2 câu hỏi. Hãy nhìn vào cách chúng ta có thể làm

### A blast from the past

Quay lại ASP.NET 4, chúng ta có thể dễ dàng disable static file caching bởi việc thêm một vài dòng tới Web.config file



Với cái này, chúng ta thậm chí có thể restrict hành vi tới debug environment bởi việc thêm các dòng tới Web.debug.config file

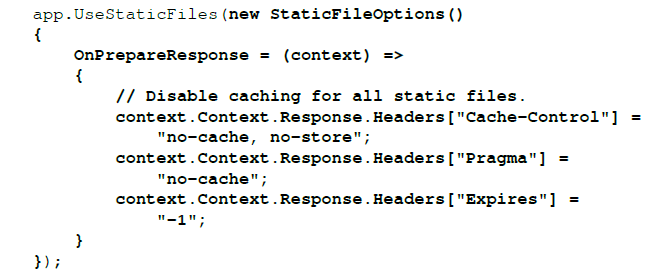
Chúng ta không sử dụng cách này tới .NET Core, việc config hệ thống đã được thiết kế lại và khá là khác với các phiên bản trước.

Và các phương pháp giải quyết bây giờ sẽ được trình bày như sau.

### Back to the future

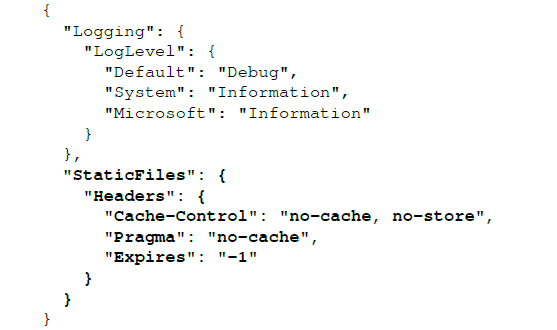
Cái đầu tiên làm là để hiểu cách mà chúng ta có thể chỉnh sử default HTTP headers cho static files. Thực tế, chúng ta có thể làm cái này bởi việc thêm một custom danh sách config options tới app.UseStaticFiles() method

Để làm điều này chúng ta sử lại một chút trong Startup.cs như sau

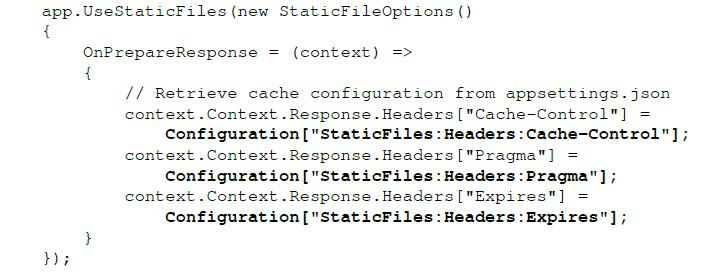


Theo convenient references

Đặt cách giá trị vào trong appsettings.Development.json



Sau đó thay lại code vào Startup.cs như sau



## Client app cleanup

### Trimming down the Component list

Đầu tiên chúng ta có một việc là xóa Angular Components chúng ta không muốn sử dụng.

Đi tới /ClientApp/src/app/ folder và xóa counter và fetch-data folder, cùng với tất cả các files họ chứa, sau đó VS sẽ báo lỗi ở tất cả những nơi mà import component này và ta chỉ việc xóa chúng ở các nơi đó là xong.

### The AppModule source code

AppModule thường chia vào trong 2 code blocks chính:

* Một list of import statements, trỏ tới tất cả references (trong form của TypeScript files) được yêu cầu bởi application.
* The root NgModule block, cái là một mảng của các mảng đã đặt tên, mỗi một thằng chứa một set of Angular objects cài mà phục vụ mục đích chung: directives, components, pipes, modules, providers,... Thằng cuối cùng chứa Component chúng ta muốn phát động, trong phần lớn scenarios thì nó là AppComponent.

## Front-end và Back-end Interactions

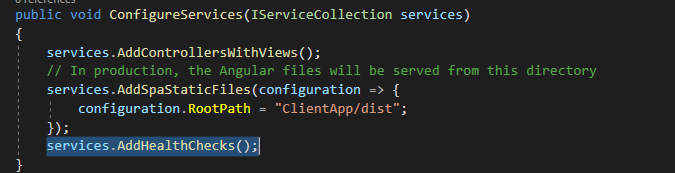
* Introducing .NET Core health checks
* HealthCheckMiddleware: Cách để thực thi nó vào trong .NET Core back-end, config nó bên trong web application’s pipelien của chúng ta và output một JSON –structured message cái mà có thể được sử dụng bởi Angular app của chúng ta
* HeathCheckCOmponent: Cách để xây dựng một Angular Component để fetch the HealthCheck structured data từ .NET Core back-end

### Introducing .NET Core health checks

Chúng ta đã gọi the first project HealthCheck vì: the web app chúng ta sắp build sẽ như là một dịch vụ báo cáo và theo dõi kiểm tra tình trạng sứa khỏe của một server, và hạ tầng của nó và hiển thị theo real time.

Để làm điều này, chúng ta sẽ sử dụng Microsoft.AspNetCore.Diagnostics.HealthChecks package, là một built-in tính năng của .NET Core framework được giới thiệu trong bản 2.2, và sau đó được cái tiến cho bản 3.0. Package này được sử dụng để cho phép một service giám sát để kiểm tra trạng thái của service khác đang chạy.

### Adding the HealthChecks middleware



### 

The /hc parameters chúng ta đã passed là để UseHealthChecks middleware sẽ tạo một server-side route cho health checks. Chú ý là chúng ta cần add thằng này ngay phía trước UseEnpoints vì toute mới của chúng ta không bị ghi đè bởi Controller route đã xác định

### Adding an Internet Control Mesage Protocol (ICMP) check

Việc Check đầu tiên chúng ta sẽ thực thi là một trong những cái chung nhất: ICMP request check tới một external host, cũng được biết như là PING

PING request là cách cơ bản để kiểm tra sự hiện diện – và vì vậy sự có sẵn – của một server cái mà chúng ta biết chúng ta sẽ có thể với tới bên trong một LAN hoặc WAN connection.

Tóm lại, nó làm việc theo cách: the machine cái mà thực hiện the PING gửi một hoặc nhiều ICMP request tới target host và chờ phản hồi; nếu nó nhận được, nó báo cáo round-trip time của toàn bộ task; không thì nó time outs và báo cáo một host không thể với tới

The host not reachable error có thể do một vài khả năng như sau:

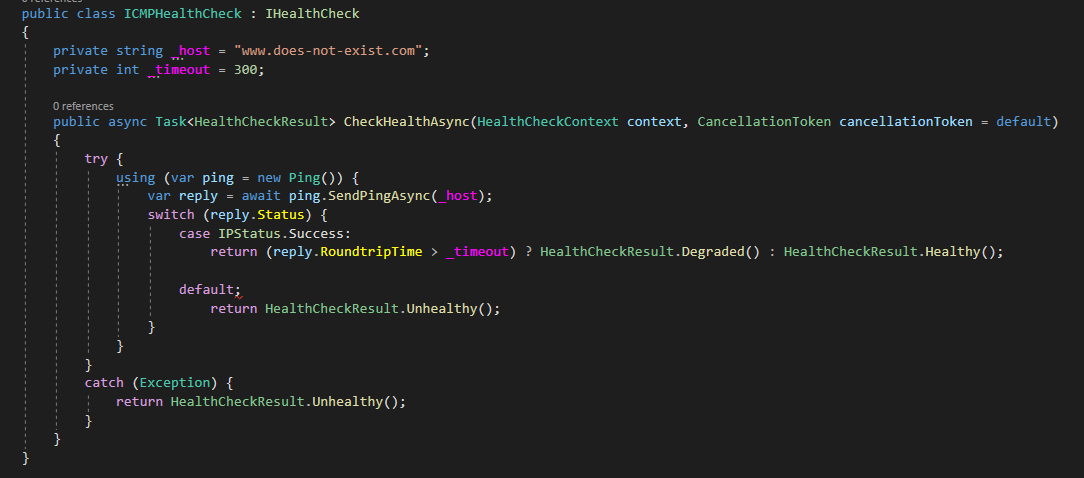
* Target host is not available
* The target host is available, but chủ động từ chối kết nối TCP/IP
* Có cả 2 thằng trên nhưng nod đã config để từ chối ICMP request và không gửi lại phản hồi ICMP lại
* Có cả 3 cái trên nhưng kết nối thì quá lâu hoặc bị cản trở bởi lý do không rõ (performance, heavy load,...) vì vậy round-trip time quá dài dẫn tới time out

### Possible outcomes

Danh sách có thể trả ra:

* Healthy: PING thành công với không lỗi và timeouts
* Degraded: PING thành công nhưng round-trip quá dài
* Unhealthy: PING failed

### Creating một ICMPHealthCheck class



### Adding the ICMPHealthCheck to the pipeline



Sau khi kiểm tra, kết quả như mong muốn của ta. Nhưng ở đây có 3 sai sót lớn

* Hardcoded values: the Host và the Timeout variables sẽ được passed nhưng là các parameters vì chúng ta có thể set chúng một cách tự động.
* Uniformative responsive: Healthy và Unhealthy không phải là điều tuyệt vời, chúng ta sẽ tìm cách để có một custom output message thay thế
* Untyped output: phản hồi hiện tại đang gửi dưới dạng văn bản thuần túy – nếu chúng ta muốn fetch nó với Angular, a JSON content-type sẽ cần định nghĩa tốt hơn.

### Improving the ICMPHealthCheck class

Bỏ hardcoded, thay thế host và timeout bởi biến truyền vào khi khởi tạo

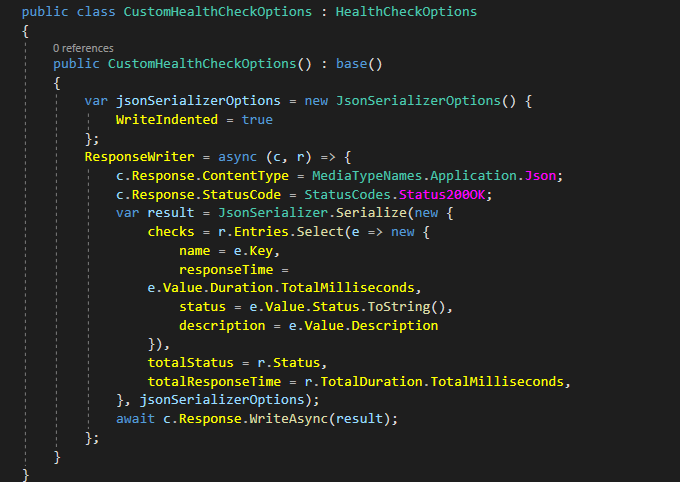
### 

Sửa lại middleware setup

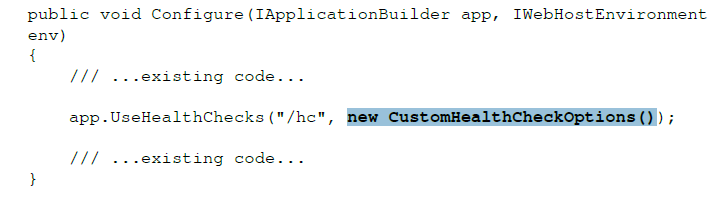


### Implementing a custom output message

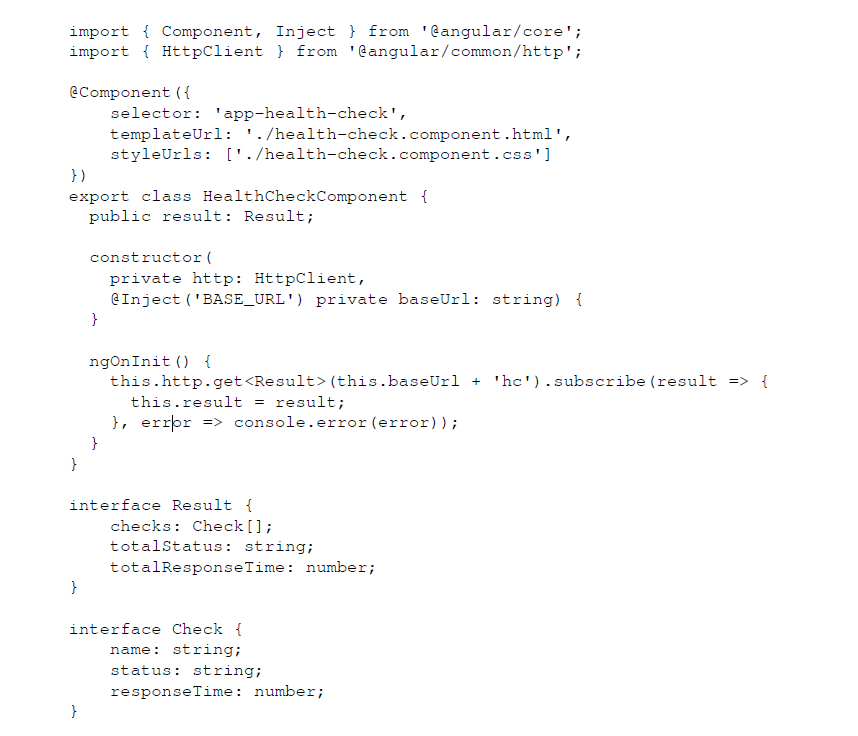
Để thực thi một custom output message, chúng ta cần ghi đè HealthCheckOptions class



### Configuring the output message



### Health check in Angular



Tạo component health-check như trên

* Tại đầu của file, chúng ta import tất cả Angular directives, pipes, services, và Components , modules – cái mà chúng ta cần toàn bộ class
* Trong Component’s constructor, chúng ta khởi tạo một HttpClient service và một baseUrl variable đang sử dụng DI; baseUrl value thì đang set bởi việc sử dụng BASE\_URL provider, đã được định nghĩa trọng /ClientApp/src/main.ts file

Trước khi đi xa hơn, chúng ta sẽ cần tìm hiểu một số topic quan trọng như:

* Imports and modules
* DI
* ngOnInit
* Constructor
* HttpClient
* Observables
* Interfaces

### Import and modules

Mộ modules là một collection of variables, functions, classes,... được nhóm bên trong một class: mỗi module được executed bên trong scope của chính nó, không trong the global scope, có nghĩa rằng tất cả elements đã khai báo bên trong nó thì không thể nhìn thấy từ bên ngoài trừ khi chúng được exported một cách rõ ràng bằng việc sử dụng export statement. Ngươcj lại, để sử dụng một variable, function, class, interface,... đã chứa bên trong một module, that module has to be imported using the import statement.

Thực tết, tất cả Angular directives, pipes, and Components thì cũng được đóng gói bên trong một collections of JavaScript modules, cái mà chúng ta có thể import vào trong bất cứ một TS class nào mà chúng ta cần. Các collections này là các libraries of modules: chúng ta có thể dễ dàng nhìn nhận chúng khi mà tên của chúng bắt đầu với @angular prefix. /ClientApp/packages.json file chứa phần lớn chúng.

Chú ý: JavaScript modules không nên bị nhầm lẫn với Angular module, cái mà dựa trên @NgModule decorator. Angular’s NgModules là building blocks – cái này là các containers for a cohesive block of code đeicated to an application domain, a workflow, or a common feature set.

Không may rằng hệ thống module của JS và Angular NgModule thì sử dụng khá nhiều từ vựng giống nhau (import vs imports, export vs exports) cái này có lẽ gây ra sự rối rắm – xem xét đặc biệt Angular apps yêu cầu developer sử dụng 2 trong số chúng tại cùng một lần



DI

....

# 4 Data Model with Entity Framework Core

Để cover chapter này, chúng ta sẽ làm những điều sau

* Create a brand-new .NET Core and Angular web application project called WorldCities: a database of cities from all over the world
* Chọn nguồn dữ liệu phù hợp
* Define and implement a data model using EF Core
* Config and deploy DBMS engine
* Create the database using EF Core’s Data Migrations feature
* Implement a data seeding strategy to load the data source to the database
* Read and write data with .NET Core using the Object – Relational Mapping (ORM)

Chúng ta chỉ cần một nguồn dữ liệu và một model dữ liệu cái mà có thể được truy cập thông qua back-end Web API để đạt được các dữ liệu này: nói cách khác, một Data Server

### Reasons to use a Data Server

Thực tế, chúng ta có thể tránh làm việc này và skip luôn chương này vì Angular cũng cấp một In-memory Web API packages cái mà thay thế the HttpClient module’s HttpBackend và môt phỏng các hoạt động CRUD trên một RESTful API; việc mô phỏng này được thực hiện bởi việc ngăn lại the Angular HTTP requests and redirecting họ tới an in-memory data source mà chúng ta kiểm soát.

Packages này khá là tuyệt, làm việc thực sự ngon cho phần lớn các kịch bản test, như sau

* Để mô phỏng hoạt động chống lại data collection cái mà không được thực thi trên dev/test server
* Đề viết unit test apps cái mà đọc và viết dữ liệu ngoài sự đánh chặn multiple HTTP calls và trình tự sản xuất phản hồi
* Để thực hiện end-to-end tests mà không cần real database, cái này là điều tuyệt vời cho CI builds

### The data source

Chúng ta có vài nguồn dữ liệu

The following is the free world cities database by DSpace-CRIS:

* **URL**: https:/ / dspace- cris. 4science. it/ handle/ 123456789/ 31
* **Format**: CSV
* **License**: Free to use

The following is GeoDataSource's world cities database (free edition):

* **URL**: http:/ / www. geodatasource. com/ world- cities- database/ free
* **Format**: CSV
* **License**: Free to use (registration required)

The following is the world cities database by simplemaps.com:

* **URL**: https:/ / simplemaps. com/ data/ world- cities
* **Format**: CSV, XLSX
* **License**: Free to use (CC BY 4.0, https:/ / creativecommons. org/ licenses/by/ 4. 0/ )

### The data model

### Intro EF Core

....

Bạn có lẽ tự hỏi tại sao chúng ta chọn việc nuôi một SQL-based mà không phải là NoSQL; có nhiều sản phẩm tốt từ NoSQL như MongoDB, RavenDB, và CouchDB cái xảy ra là để có một thư viện kết nối của C#.

Câu trả lời đơn giản: do thư viện bên thứ 3 cung cấp, họ k được includeed trong EF Core Database provide list chính thức.

### The SQL Server Data Provide

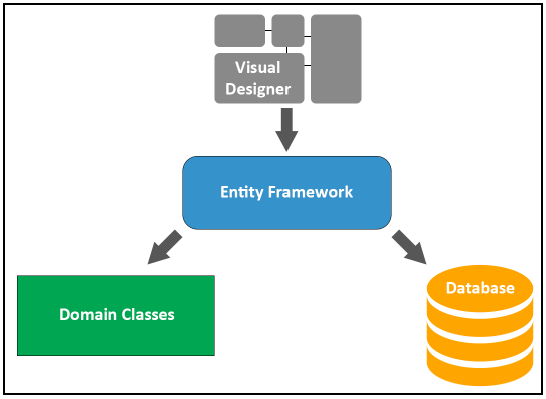
Thằng này cung cấp một interface cho toàn bộ họ nhà Microsoft SQL Server database

### Data modeling approaches

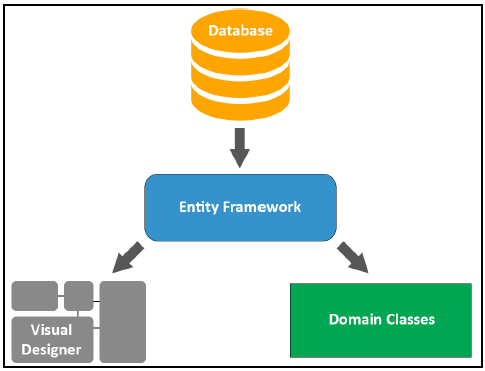
Có 3 cái là : Model-First, Database-First, Code-First

* Model-First

Cách này là ta design một visual diagram, diagram này sau đó được sử dụng bởi framework để tự động generate SQL script và data model source code files

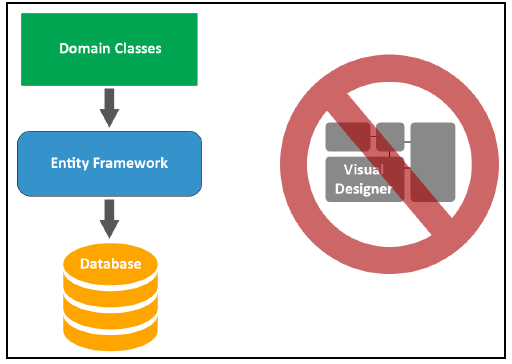


* Ưu điểm:
* Chúng ta có thể tạo database schema và class diagram toàn bộ bằng việc sử dụng visual design tool, cái này có thể tuyệt vời khi mà cấu trúc dữ liệu khá lớn
* Mỗi khi database thay đổi, model có thể được update mà không mất dữ liệu
* Nhược:
* Diagram-driven, tự động generate SQL scripts có thể dẫn tới mất dữ liệu trong trường hợp update. Xử lý trường hợp này là vào sql tạo script execute cơm
* Database-First



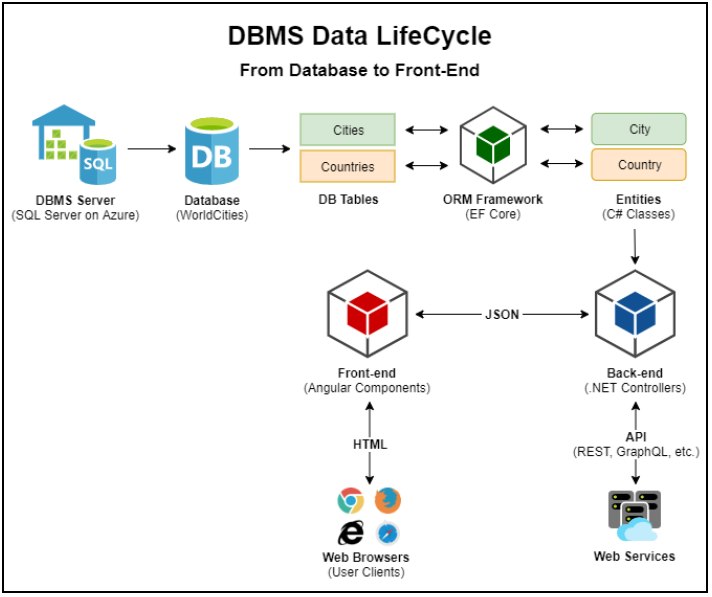
* Code-First

Cách tiếp cận hàng đầu từ bản thứ 4 của EF



* Ưu:
* Không cần diagrams và visual tools, cái này tuyệt vời cho dự án nhỏ đến vừa, giúp tiết kiệm phần lớn thời gian
* Nhược:
* Hiểu biết về C# và EF updated conventions được yêu cầu
* Việc bảo trì database có thể khó khăn, cũng như cập nhật mà không mất dữ liệu

### Creating the entities



.....

Sau một loạt cách thao tác tạo entity và relationship giờ ta tiến hành load theo mối quan hệ

### EF Core loading pattern

Có 3 kiểu load

* Eager Loading: The related data được load từ database như một phần của initial query
* Explicit Loading: The related data được loaded rõ ràng từ database vào lần sau.
* Lazy Loading: The related data được load một cách trong suốt từ database khi navigation property được truy cập cho lần đầu tiên. Mô hình này phức tạp nhất trong 3 thằng, sẽ gặp vấn đề nghiêm trọng nếu không thực thi một cách hợp lý.
* Related data chỉ được load khi mà chúng ta nói với EF Core để load chúng, nếu không chúng sẽ ở trạng thái với giá trị là NULL

## Getting a SQL Server

### Installing SQL Server 2019

### Creating a SQL Database on Azure

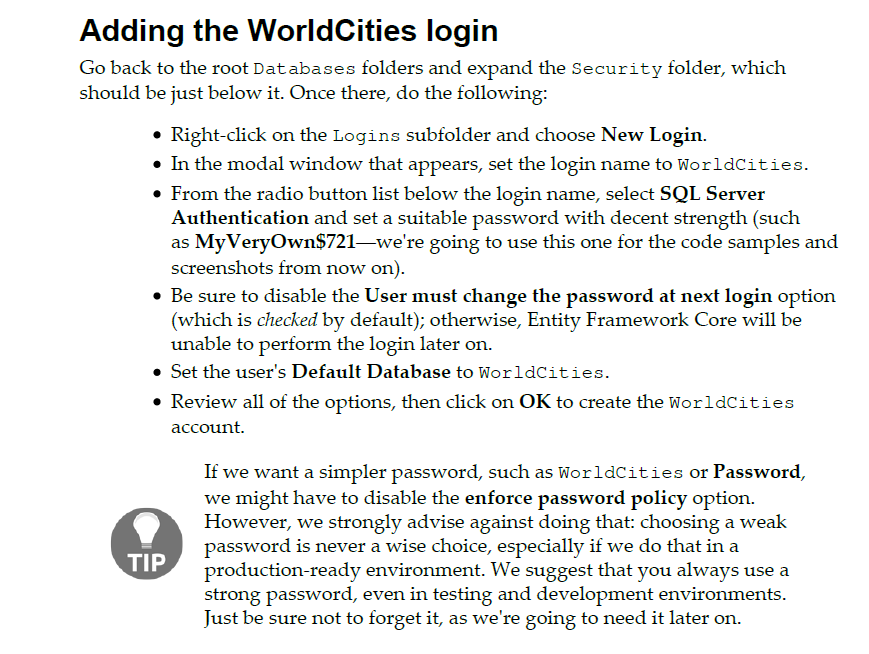
* SQL Database: This is a fully-managed SQL Database engine based on SQL Server Enterprise Edition. Với option này nó cho phép chúng ta set up và quản lý một hoặc nhiều single relational database hosted in the Azure clound with a Platform-as-a-Service (Paas) mô hình sử dụng và thanh toán: chúng ta có thể định nghĩa nó như một Database-as-a-Service (DbaaS) . Option này cung cấp buit-in high avaibility, intelligence, and management, ý nghĩa tuyệt với cho người muốn một giải pháp đa năng mà không muốn rắc rối về việc config, manage và chi trả cho toàn bộ server host.
* SQL manager instance: Cái này là một instance quản lý SQL một cách riêng trên Azure, đây là dịch vụ cung cấp SQL Server một cách tiêu chuẩn và có thể mở rộng
* SQL virtual machine

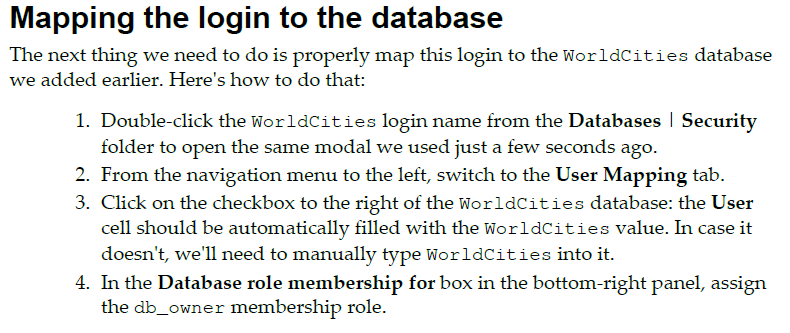
SQL Database được cho là rẻ nhất vì nó free cho 12 tháng, tiếp đó là SQL mi và SQL vm

##### Setting up a SQL Database

### Configurating the database

Bất chấp path có như nào, chúng ta làm với local instance hay là Azure chúng ta sẽ bắt đầu để quản lý SQL Database chúng ta đã tạo





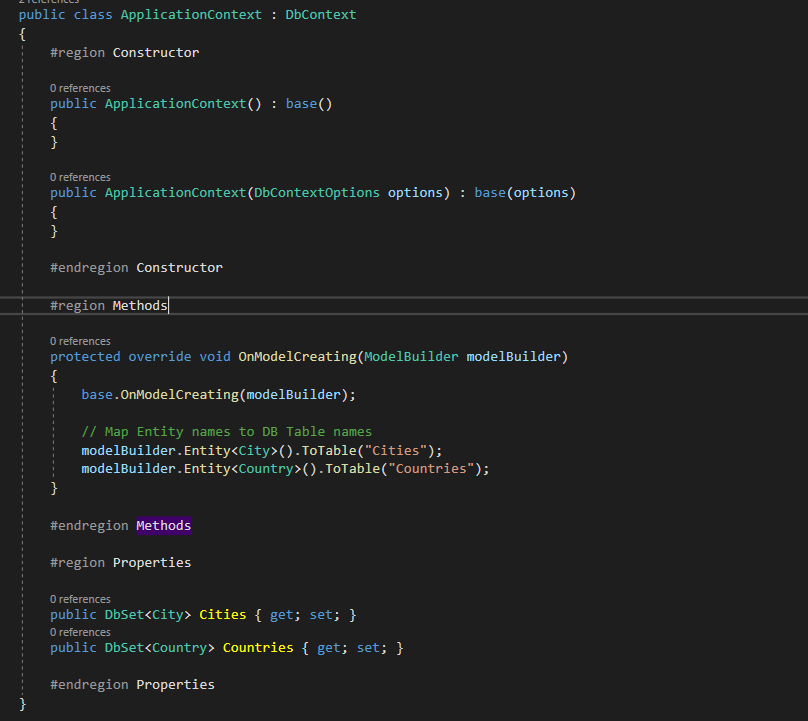
### Creating the database using Code-First

Chúng ta cần take care 2 cái:

* Set up an appropriate Database Context
* Enable Code-First Data Migrations support within our project

##### Setting up the DbContext

Để tương tác với data như là object/entity classes, EF Core sử dụng Microsoft.EntityFrameworkCore.DbContext class, cũng được gọi tắt là DbContext hoặc đơn giản là Context.



Có vài điểm quan trọng đã làm ở đây:

* Chúng ta đã ghi đè OnModelCreating method để định nghĩa cơm các mối quan hệ model dữ liệu của chúng ta cho các classes entity của chúng ta. Chú ý rằng, chúng ta đã configured cơm tables names cho mỗi entity sử dung modelBuilder.Entity<TentityType>().ToTable method
* Chúng ta đã added DbSet<T> property cho mỗi entities để chúng ta có thể dễ dàng access tới chúng sau này

##### Database initialization strategies

Với cách Code-First này chúng ta cần: CreateDatabaseIfNotExits, DropCreateDatabaseIfModelChanges, DropCreateDatabaseAlways, or MigrateDatabaseToLatestVersion.

Bật PowerShell -> trỏ tới đường dẫn chứa file project

**dotnet ef migrations add MigrationName [options]**

*dotnet ef migrations add "Initial" -o "Data\Migrations"*

the optional –o parameter có thể được sử dụng để thay đổi location nơi mà migration code-generated files

Đây mới chỉ là bưới tạo migration, sau đó để thực sự nó tạo database thì ta cần apply nó

**dotnet ef database update [options]**

Ta nhận thấy các trường của ta như Lat và Lon chưa có kiểu dữ liệu xác định => ta cần phải vào sửa lại, do lúc này database chưa được apply nên ta có thể xóa luôn migration kia đi.

##### Updating the database

Applying a data migration (di chuyển dữ liệu/ di cư dữ liệu) có nghĩa là việc tạo (hoặc update) the database để đồng bộ hóa các nội dung của nó (tables structure, constrains, and so on) với rules cái mà đã xác định trong DbContext và data annotations bên trong các Entity classes.

##### Hiểu về migrations

Trước khi chúng ta rời đi, Code-First Migrations thực sự là cái gì, và những lợi ích mà ta thu được khi sử dụng chúng.

Mỗi khi chúng ta đang phát triển ứng dụng và đang định nghĩa a data model, chúng ta có thể chắc chắn rằng nó sẽ thay đổi theo thời gian như thêm trường, thêm bảng,...

Mỗi lần chúng ta làm điều này, chúng ta cũng sẽ đặt Data Model của chúng ta out khỏi sự đồng bộ, cái mà Code-First đã generated database. Điều này không phải là vấn đề khi mà chúng ta đang debugging trong mối trường development. Nhưng khi chuyển sang môi trường production, chúng ta sẽ đối mặt với toàn bộ những thứ như: miễn là chúng ta đang xử lý với real data thì dropping and recreating database của chúng ta sẽ không có thêm bất cứ sự lựa chọn nào thêm. Đây là cái mà tính năng Code-First Migrations thì có ý nghĩa để bàn tới: trao cho developer cơ hội để thay thế the database schema mà không cần drop/recreate toàn bộ mọi cái.

##### Populating the database

Chúng ta đã có database, có DbContext và có data source. Giờ là lúc mà chúng ta sẽ đưa nó vào trong database hay còn gọi là **data seeding**

Chúng ta có thể sử dụng một trong những cách sau mà EF Core hỗ trợ

* Model data seed
* Manual migration customization
* Custom initialization logic

*https://docs.microsoft.com/en-us/ef/core/ modeling/data-seeding*

Link trên có mô tả chi tiết về cả 3 thằng này

##### Implementing SeedingController

Custom initialization sẽ được nôi trong Controller, cái này chúng ta sẽ gọi lad SeedController

Importing the Excel file

Chungs ta khá may mắn khi có một thư viện bên thứ 3 đọc (thậm chí ghi) Excel file sử dụng định dạnh Office Open XML khá là tuyệt vời, nó là EPPlus

## 5 Fetching and Displaying Data

### Requests and response

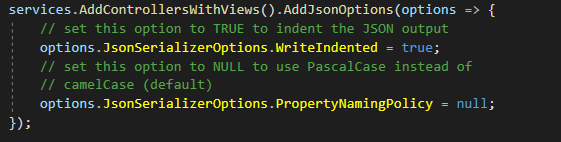
### JSON conventions and defaults



Như chúng ta có thể nhìn, JSON is a serialization of our City entity, với built-in conventions như sau:

* CamelCase instead of PascalCase: Chúng ta đã nhận *name* thay vì *Name*, *countryId* thay vì *ConuntryId*, điều này có nghĩa rằng tất cả PascalCase .NET class names và properties sẽ được converted tự động vào trong camelCase khi mà chúng được chuỗi hóa sang JSON
* Không có thụt dòng, dòng trắng: Mọi thứ được stacked vào trong một dòng

Chúng ta có thể thay đổi conventions này theo ý muốn với services ở middleware



### A (very) long list

Từ client-side chúng ta lấy được dữ liệu, nhưng nó lên tận 13k dòng, điều này là huge peformance stress for both .NET Core and Angular, hay với trải nghiệm khách hàng khi cố gắng tìm thành phố của mình trong từng ấy dòng.

Để fix vấn đề này, chúng ta cần thực thi một vài tính năng quan trọng cái mà được sử dụng thường xuyên để deal với HTML tables: paging, sorting, and filtering

### Serving data with Angular Material

Chúng ta có thể sử dụng component trong material để có Paging, nhưng k cần phải là thiên tài cũng hiểu là tất cả các row vẫn ở trên page: chúng ta không bao giờ cần server để support một paginated request, vì vậy chúng ta vẫn fetch tất cả cities từ Data Provide (thông qua .NET Core API Controller), thực tế chúng ta chỉ ẩn bởi front-end.

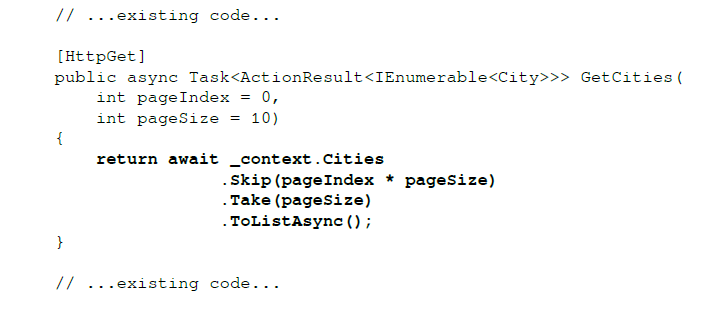
Điều này vẫn làm ảnh hưởng đến hiệu năng của cả server và client. Để giảm thiểu vấn đề này, chúng ta cần đưa paging tới server-side

### Server-side paging

* Thay đổi CitiesController để support paged HTTP GET request
* Create a new ApiResult .NET class cái mà chúng ta có thể sử dụng để cải thiện JSON response của .NET Core Controller
* Thay đổi cities.component.ts

##### CitiesController

Chúng ta sử lại API GetCities với 2 biến là pageIndex và pageSize



Nhưng nếu trả thế này thì phân trang làm sao biết tổng số trang và phân kiểu gì được

Chúng ta cần tìm một cách để nói với Angular app về các thông tin thêm này như sau:

* Tổng số pages có thể có
* The current page
* The number of records on each page

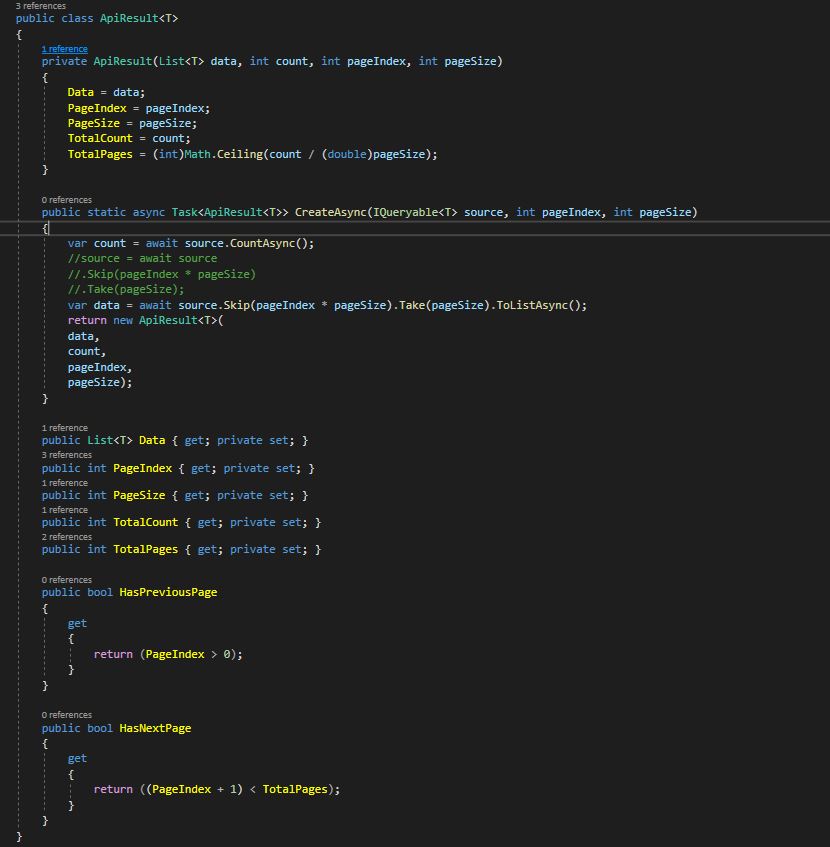
Thực sự mà nói chúng ta chỉ cần thông tin đầu tiên, Angular client có thể keep track 2 thằng còn lại, tuy nhiên, khi mà chúng ta thực thi thằng đầu, chúng ta có lẽ tốt hơn là trả về tất cả.

Để làm điều này, cách tốt nhất là chúng ta tạo một response-type class cái mà sẽ sử dụng cho những cái như thế này.

##### ApiReuslt

Là generic class phục vụ cho việc trả về danh sách dữ liệu theo page, và trạng thái của page như:

* Data
* PageIndex
* PageSize
* TotalCount
* TotalPages
* HasNext
* HasPrev



Và việc còn lại là ta sẽ thay đổi lại logic trong controller

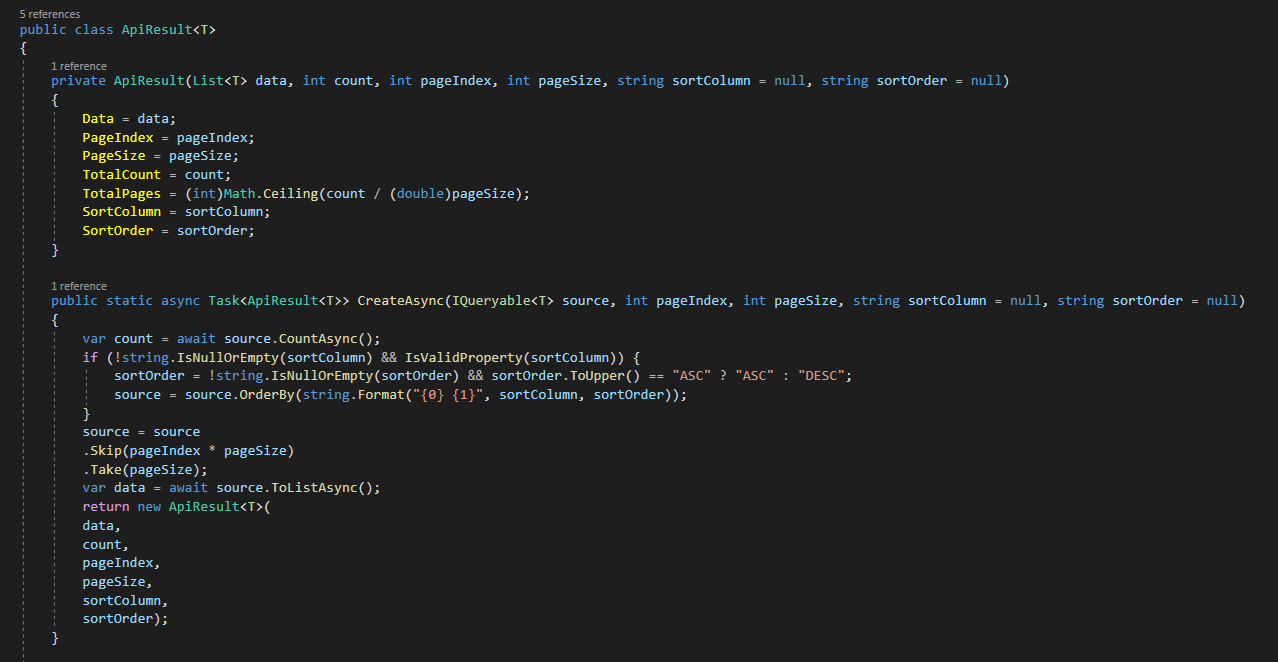
Nếu ta thêm sorting nữa thì sẽ mở rộng ApiResult

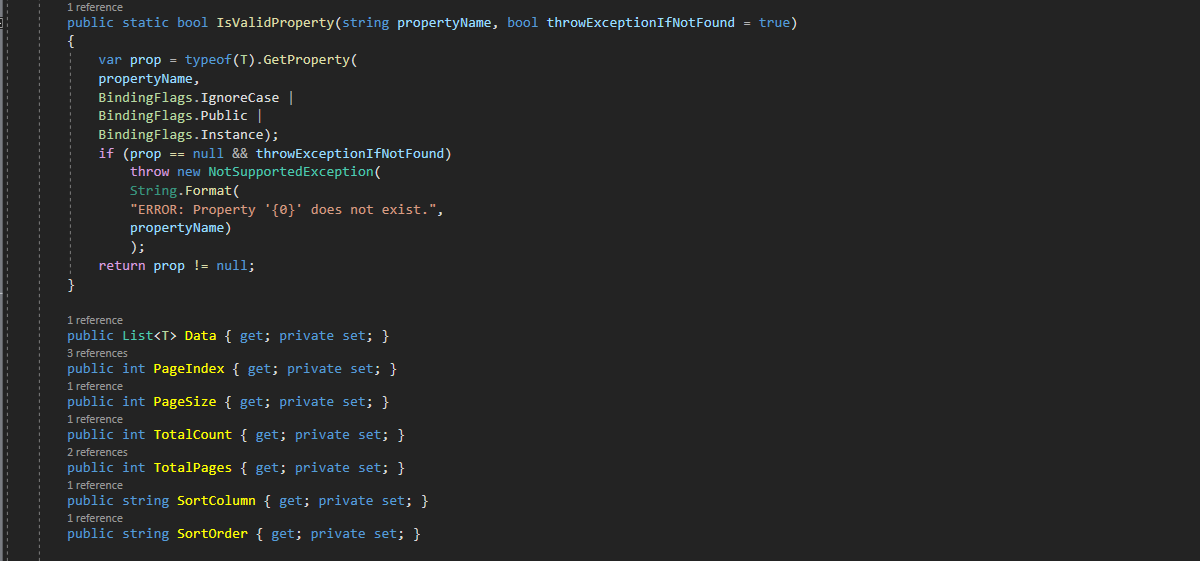
### Extending ApiResult

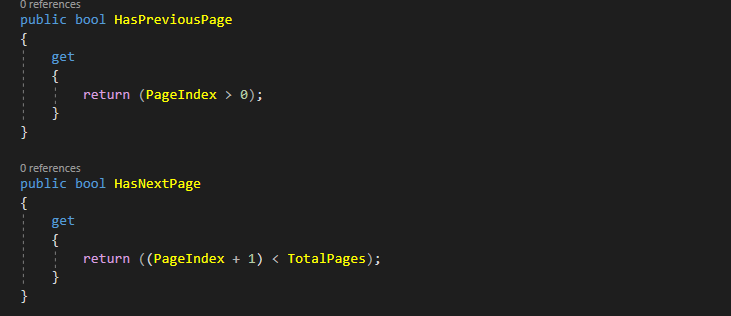
Để mà có thể có được dữ liệu mà đã sort thì tất nhiên khi request bên client ta cũng phải đưa các biến vào để sort.

Chúng ta thêm 2 attributes là sortColumn và sortOrder

Chúng ta sẽ cải tiến ApiResult của chúng ta như sau







Chúng ta đang sử dụng LINQ, nhưng là bản Dynamic LINQ

##### LINQ là gì

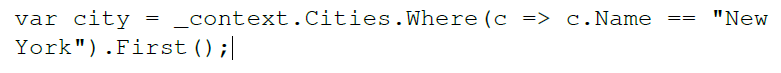
Trước khi mà chúng ta đi tiếp, hãy giành vài phút để nói về LINQ

Viết tắt của Language-Integrated Query, LINQ là tên của một Framework với một bộ các công nghệ query dữ liệu, cái mà có khả năng sử dụng trong .NET như C# hoặc VB.NET . Nó được ra mắt năm 2007.

Mục đích chính của LINQ là để developer có thể biểu diễn cấu trúc queries đới với data bằng việc sử dụng một cấu trúc ngôn ngữ duy nhất, mà không cần học thêm các ngôn ngữ query khác cho mỗi kiểu data source ( SQL, XML, CSV,...). Mỗi kiểu data source, LINQ cung cấp trải nghiệm query giống nhau cho các objects (LINQ to Objects).

Các queries cấu trúc LINQ có thể được biểu diễn bằng việc sử dụng 2 thay thế

* Lambda expressions



* Query expressions

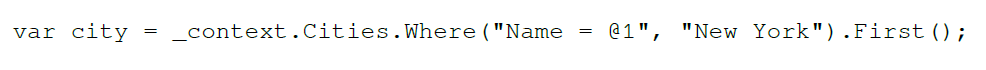


Cả 2 cho hiệu năng và kết quả giống nhau, Query expression sẽ được convert sang Lambda trước khi biên dịch

##### Linq.Dynamic.Core pros and cons

Tại sao chúng ta có thằng này.

Với 2 ví dụ về Lambda và Query expression làm việc, mỗi khi chúng ta query một object của bất cứ kiểu nào sử dụng LINQ, the source type cũng với tất cả properties chúng ta muốn query phải được biết bởi compiler. Điều này có nghĩa rằng chúng ta sẽ không thể sử dụng các kỹ thuật với generic objects (objects) hoặc types (<T>). Đây chính là nơi mà Linq.Dynamic xử lý vấn để này, nó cho phép developer viết lambda expressions và query expressions với chuỗi strings và translate chúng tới kiểu dữ liệu bằng việc sử dụng reflection.



Tuy nhiên với cách này, do kiểu dữ liệu chỉ xác định trong runtime, kiểm tra trong runtime nên dễ lỗi vì 2 lý do

* Chúng ta sẽ chỉ cần một chuỗi string từ query, các lỗi này sẽ là nguyên nhân dẫn tới crashes hạng nặng
* Rủi ro của việc queries không moing muốn (bao gồm cả SQL injection) có thể tăng lên theo cấp số nhân, phụ thuộc vào cách chúng ta xây dựng các queries này và nơi chúng ta nhận các string dynamic của chúng ta

##### Việc tránh SQL injection

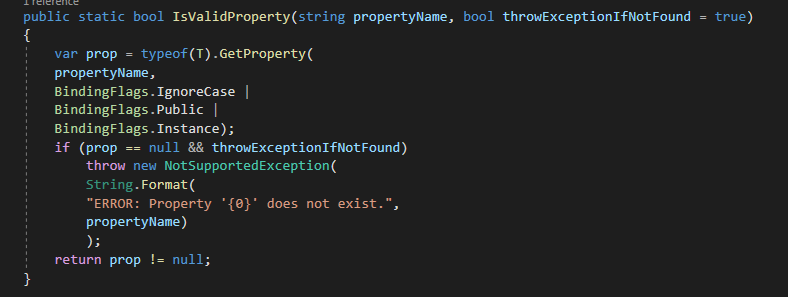
May mắn thay, chúng ta không cần làm điều này: mặc dù chúng ta nhận 2 biến strings từ client là sortColumn và sortOrder – chúng ta đã đưa các biện pháp đối phó cả 2 trước source code của ApiResult

Đối với sortOrder

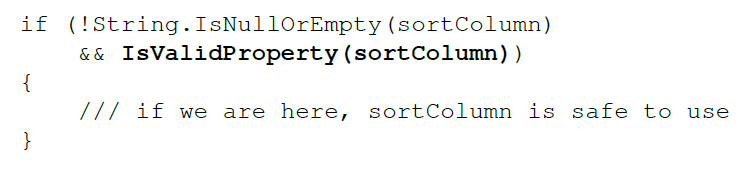


Với đoạn code này, chúng ta check giá trị null và chuẩn hóa nó về dạng dữ liệu an toàn.

Với sortColumn thì phức tạp hơn



Với cách này chúng ta luôn chắc chắn với property name sẽ nhận được theo đúng chuẩn của ta.



Trong dấu {} chính là khu vực an toàn SQL injection: miễn là chúng ta làm việc với sortColumn bên trong chúng, chúng ta không lo lắng vấn đề gì

Việc bảo mật này đòi hỏi code của chúng ta có sự chặt chẽ cao trong các mục xử lý truy vấn

### Adding filtering

Chúng ta sẽ đặt một textbox lên trên đầu, mỗi khi ta gõ thì nó sẽ request tới back-end và lấy toàn bộ dự liệu mới về

* Chúng ta phải thay đổi khác nhiều thứ, từ Controller, hàm GetCities, ApiResult

Chúng ta chả có cách chó nào cả, đây là cách tiếp cận duy nhất

# 6 Forms and Data Validation

Trong chương này, chúng ta sẽ làm việc với forms, data input, và kỹ thuật validation.

Như chúng ta đã biết, HTML forms thì là một trong những khía cạnh quan trọng và tính tế nhất của bất cứ ứng dụng nào. Ngày nay, forms được sử dụng hầu hết với bất cứ nhiệm vụ mà người dùng submit data, như là việc đăng lý hoặc logging trong website, thanh toán, đặt mua sản phẩm,...

Trong chapter này, chúng ta sẽ cover các topic sau:

* Angular forms, đây là nơi mà chúng ta sẽ làm việc với Template-Driven Forms cũng như Reactive Forms, hiểu được ưu và nhược của 2 cách tiếp cận, và các tình huống sử dụng sao cho phù hợp
* Data Validation, nơi mà chúng ta sẽ học cách double-check our user’ input data trong front-end và back-end, cũng như các kỹ thuật để hiển thị feedback khi người dùng nhập sai hoặc giá trị không hợp lệ
* Form builder, nơi mà chúng ta sẽ thực thi Reactive Form khác sử dụng các factory methods thay vì tạo từng thằng cho các model elements

### Lý do để sử dụng form

Nếu giả dụ chúng ta không sử dụng form thì sao:

* Chúng ta không theo dõi được trạng thái toàn cục của form, không có cahs nào nói với input text là hợp lệ hay không.
* Chúng ta không dễ dàng để có thể display error message tới users để họ biết rằng họ làm hợp lệ
* Chúng ta không xác minh input data trong bất cứ cách nào, chúng ta chỉ thu thập và ném nó với server mà không nghĩ gì thêm.
* Nói chung thì form sinh ra là để chúng ta kiểm chứng dữ liệu, xác minh dữ liệu nhập và chuẩn hóa dữ liệu nhập của chúng ta trước khi đưa lên server để hạn chế các mã độc.

Tất nhiên mà nói, về mặt nào đó ta không nhất thiết cần form, ta có thể validate tay, có thể thông báo theo điều kiện, logic xuất hiện các thứ, tự custom hoàn toàn. Nhưng làm thế để làm gì khi đã có những công cụ như Angular thay thế ta làm, nếu làm như vậy việc dùng Angular là không cần đến nữa.

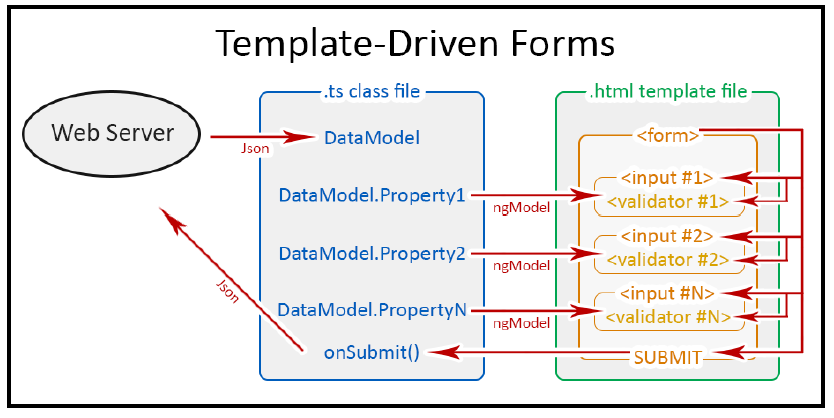
Angular cung cấp cho chúng ta một vài chiến lược để làm việc với các kịch bản liên quan đến form như thế này.

* Template-Driven Forms
* Model-Driven Forms, gọi khác là Reactive Forms

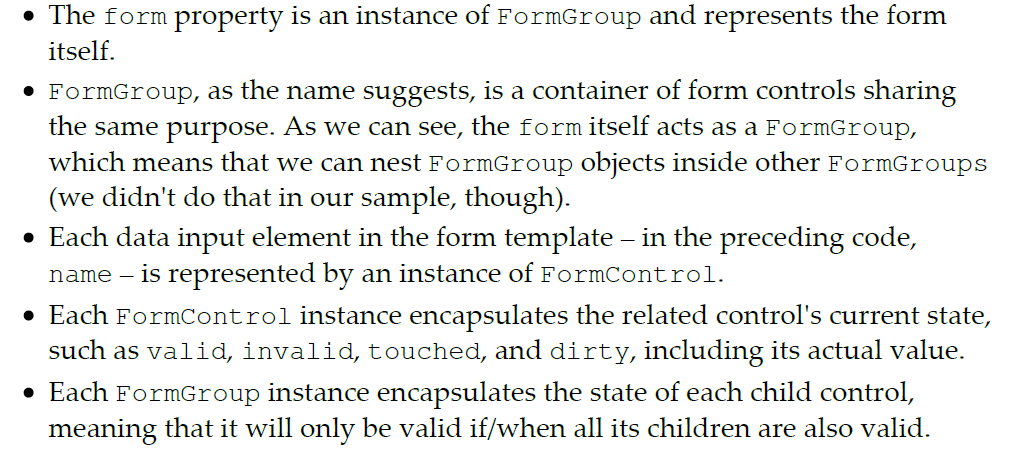
##### Template-Driven Forms

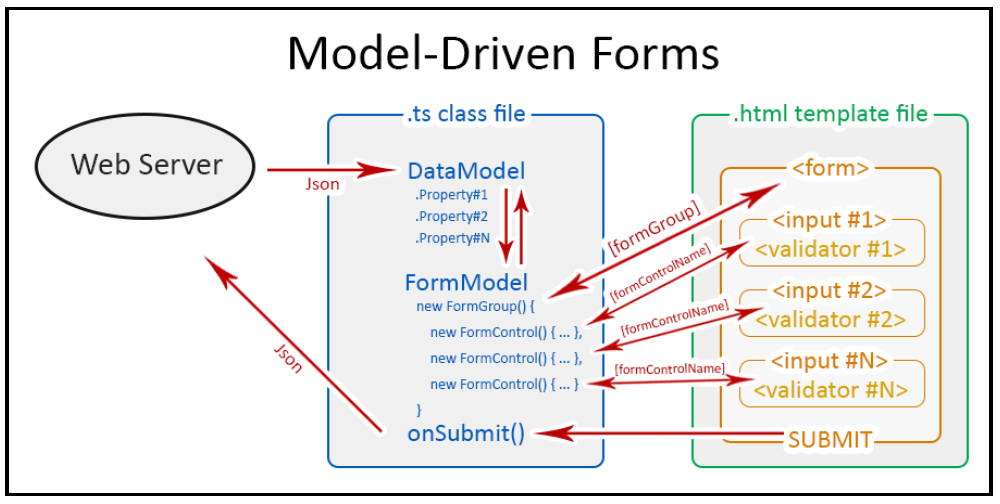
Làm việc với Template-Driven Form nghĩa là xây dự form trong .html template file, liên kết dữ liệu tới các fields sử dụng ngModel instance, và sử dụng một object ngForm để chứa toàn bộ các inputs, mỗi cái được truy cập thông qua tên của chúng để thực hiện các việc checks

Sử dụng để build form vừa và nhỏ, đơn giản không phức tạp



##### Model-Driven/Reactive Forms





### Hiểu về data validation

Việc thêm dữ liệu validation tới một form hầu như không phải là option: nó là tính năng bắt buộc để kiểm tra độ chính xác dữ liệu người dùng input.

Chúng ta thiết lập validation các input trên form theo như rule và tính năng của Angular trên front-end

Cái này sẽ được tìm hiểu chi tiết ở phần front-end

### Server-side validation

Là quá trính kiểm tra các errors (và xử lý chúng) trên phía server-side, điều này làm sau khi data đã được gửi đến back-end. Cái này khác với cách tiếp cận của phía client-side.

Việc xử lý errors trên client-side có rất nhiều lợi thế về tốc độ và hiệu năng bởi vì user biết ngay lập tức liệu dữ liệu input là hợp lệ hay không mà không cần query từ server. Tuy nhiên, server side validation là phải làm cho bất cứ web application nào bởi vì nó phòng ngừa đối với các kịch bản có hại như:

* Implement errors của client-side validation process
* Client-side hacks được thực hiện bởi các users có nhiều kinh nghiệm, browser extensions, hoặc plugins cái mà có lẽ muốn cho phép user để gửi các input với giá trị không hỗ trợ tới back-end
* Request forgery, là các request HTTP sai hoặc độc hại

Vì vậy, với client-side validation là một tính năng tiện lợi, trong khi với server-side validation là bắt buppcj cho bất cuws web app nào cái mà quan tâm về caahst lượng của dữ liệu nhập vào

Sau khi dữ liệu input nhập vào mà đã hợp lệ, ta sẽ gửi một request tới server và xem nó có đúng vs logic của ta hay k, cái mà trên front-end không thể làm được. Nếu k đúng logic sẽ báo

Vấn đề hiệu năng: Tác vụ validate này sẽ chỉ làm sau khi cả form đã hợp lệ, và việc check này là cuối cùng sau khi client-side đã hợp lệ

# 7 Code Tweaks and Data Services

Chương này chúng ta sẽ làm các mục sau:

* Optimizations and tweaks, nơi chúng ta sẽ thực thi nhiều high-level source code và UI refinements
* Bug fixes and improvements, chúng ta sẽ tận dụng việc tinh chỉnh trước đó để tăng sự nhất quán của app và thêm nhiều tính năng mới
* Data Services, chúng ta sẽ học cách di rời raw HttpClient service vào trong Components khác

## Optimizations and tweaks

Trong chương trình máy tính, khái niệm **code bloat** được sử dụng để mô tả source code dài, chậm không cần thiết. Cách hiệu quả nhất để trống lại việc code sẽ trở lên cồng kềnh là tuân thủ nguyên tắc **DRY**

## Template improvements

Nếu chúng ta nhìn lại CityEditComponent và CountryEditComponent template files, chúng ta có thể nhìn rõ một lượng lớn code bloat. Form.get() method được gọi không dưới 10 lần mỗi form.

Thực tế, mỗi khi chúng ta cảm thấy như là chúng ta đang viết quá nhiều code hoặc lặp lại một task complex quá nhiều lần, chúng ta có thể tạo một hoặc nhiều methods helper bên trong Component class của chúng ta để qui về một chỗ.

Tương tự như các ngôn ngữ server-side thì với TypeScript chúng ta cũng có thể kế thừa các class với nhau, thằng template thì tách riêng ra không chấp. Từ đó ta có thể sử dụng các nguyên lý và các mô hình của design pattern để xử lý cho chúng

## Bug fixes and improvements

Chúng ta có 3 bug

* City Detail view chưa validate lat và lon input values, ví dụ, chúng ta vẫn cho phép nhập chữ thay vì số
* Countries view không show danh sách các cities, cái mà mỗi country thực sự chứa
* Cities view của chúng ta không show country name

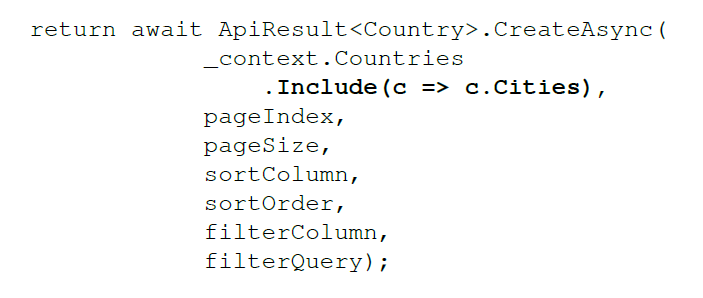
## Validating lat and lon

Một form cái có thể bị phá vỡ từ front-end là mọi cái chúng ta luôn tránh, thậm chí nếu input types đã được kiểm tra trong back-end

* Chúng ta cần validation thêm cho 2 thằng này nữa, làm giống như gì đã làm

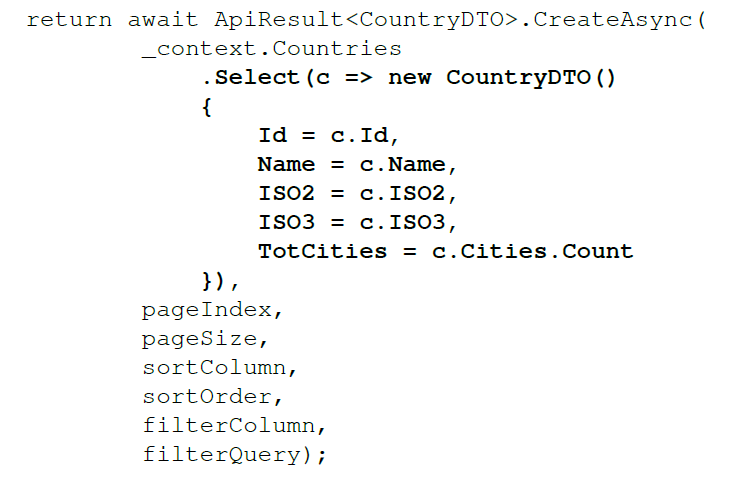
## Adding the number of cities

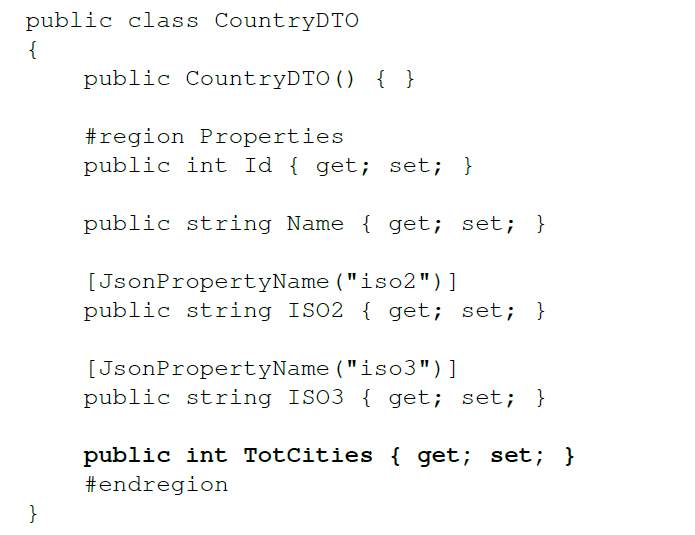
Việc thêm một cột về số lượng của cities trong contry thì có nghĩa ở back-end chúng ta sẽ trả về thêm một trường giữ số lượng của cities khi query country, bình thường chúng ta sẽ dùng Eagload



Nhưng đéo phải thiên tài cũng hiểu cái này ảnh hưởng đến hiệu năng, tài nguyên...

Chúng ta sẽ sử dụng cách khác





## Data Services

2 web app mà chúng ta đã tạo thì front-end và back-end giao tiếp với nhau bằng giao thức HTTP(S), và để thành lập giao thức này chúng ta đã sử dụng HttpClient class, một built-in Angular HTTP API client shipped with the @angular/common/http package cái nằm trên XMLHttpRequest interface

The Angular’s HttpClient class có nhiều lợi ích.

### XMLHttpRequest vs Fetch

Như đã nói Angular’s HttpClient class dựa trên nền XMLHttpRequest (XHR) là một API được cung cấp bởi browser thông qua JavaScript engine, cái mà có thể được sử dụng để transfer data giữa web browser và web server một cách bất đồng bộ, thằng này vừa được kỷ niệm 20 năm cho đnế khi thay thế năm 2017, khi mà Fetch API thực sự ra đời

Fetch API là interface khác cho việc fetching tài nguyên cái mà nhắm tới một sự thay thế hiện đại cho XMLHttpRequest API, việc cung cấp nhiều tính năng mạnh mẽ và linh động hơn. Chúng ta cùng xem ưu và nhược của cả 2 thằng này

##### XMLHttpRequest

Đầu tiên, với khái nueemj AJAX, nó được viết tắt bởi Async JavaScript + XML: một bộ các kỹ thuật của web-development cái được sử dụng để tạo async web app từ client-side, nơi mà XMLHttpRequest đòng vai trò then chốt.

5/4/2006 World Wid Web Consorrtium (W3C) đã phát hành bản thảo đầu tiên cho XMLHttpRequest, dự thảo W3C đã mở đường để áp dụng rộng rãi AJAX development. Tuy nhiên, việc triển khai đầu tiên khó hơn cho phần lớn developer web, do sự khác biệt liên quan đến việc thực thi của các trình duyện . May mắn thay, nhiều cái trở thành dễ hơn nhờ nhiều cross-browser JavaScript libraries như jQuery, Axios...

Trong suốt thời gian này, the XHR data format đã nhanh chóng chuyển từ XML sang JSON, HTML và plain text, cái mà phù hợp để làm việc hơn với DOM page. Khi RxJS library came out, XMLHttpRequest object có thể dễ dàng đặt dưới Observable, do đó đạt được rất nhiều lợi thế (giống như có thể mix nó với observables khác, subcribe/unsubcribe, pipe/map,...)

Đây là ý tưởng chính bên dưới Angular’s HttpClient class, cái được mô tả như Angular way to deal with XMLHttpRequest, một gói rất thuận tiên cho phép developers sử dụng hiệu quả Observable pattern.

##### Fetch

Trong vài năm trở lại đây, việc sử dụng raw XMLHttpRequest object khó hơn cho phần lớn web dev, và có thể dẫn tới một lượng lớn JS source code, vấn đề này được giải quyết bởi superstructures cái đã được mang lại bởi các thư viện như jQuery

### Building a Data Services

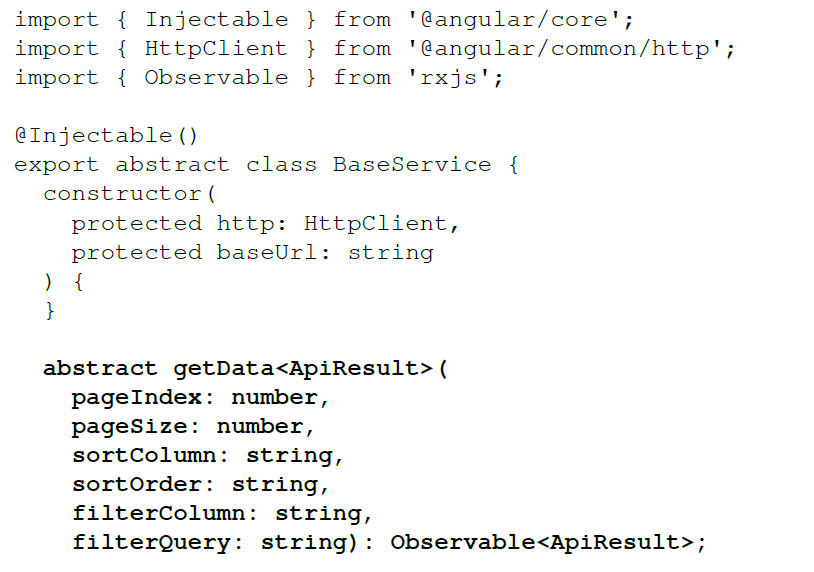
Chúng ta sẽ tách việc request dữ liệu sang một layer khác để phục vụ nhiều mục đích xử lý sau này hơn, vấn là áp dụng nguyên tắc SOC.

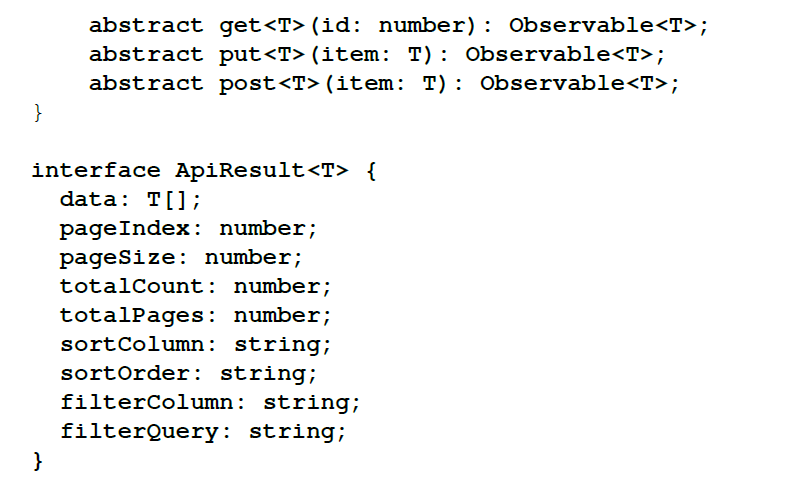


Tạo base services cho cities services và country services

@Injectable được sử dụng trước khai báo class để nói với Angular rằng, class này sẽ cung cấp một injectable service cái mà được sử dụng thông qua DI, bởi các classes và Components khác

Chúng ta sẽ add thêm một common interface method





* Cuối cùng thì với chương 7, đơn giản là chúng ta refactor code, thực hiện nguyên lý SOC, sử dụng DTO để an toàn dữ liệu

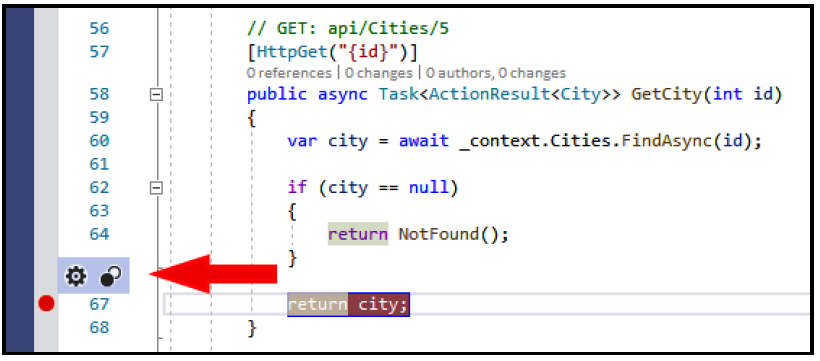
# 8 Debugging

## Back-end Debugging

### Conditional breakpoints

The conditional breackpoint hữu dụng cho tính năng debugging, cái anyf thường không được biết bởi phần lớn các developer; nó giống normal breakpoint, nhưng chỉ trigger khi gặp đúng điều kiện.

Để set điều kiện cho breakpoint, chỉ cần clink trên Setting contextual icon



### Actions

Là tính năng được sử dụng để show một custom message trong Output window

### The Output window

Được sử dụng để show các messages trạng thái cho các tính năng trong IDE

## Front-end debugging

# 10 Authentication and Authorization (xác thực và ủy quyền)

Nói chung, khái niệm authentication (xác thực) refers tới bất cứ quá trình xác minh một ai đó, có thể là một người hoặc một hệ thống tự động. Điều này đúng bên trong context của W3

Authentication sẽ không bao giờ bị nhầm lẫn với authorization, cái này là một xử lý khác và phụ trách một nhiệm vụ rất khác. Để định nghĩa nhanh, chúng ta có thể nói rằng mục đích của authorization là để xác nhận rằng the requesting user thì được cho phép để truy cập tới action họ muốn thực hiện. Nói cách khác, authentication là về ai đó, authorization là về cái mà họ được phép làm.

Để hiểu tốt hơn sự khác nhau giữa 2 nội dung này, chúng ta có thể nghĩ với 2 kịch bản thực như sau:

* Một free chưa được đăng kí tài khoản đang cố để giành quyền truy cập tới một service hoặc tính năng premium; đây là ví dụ chung của đã xác thực, truy cập không được ủy quyền, chúng ta biết họ là ai, họ không được phép tới chỗ này
* Một user nặc danh đang có giành quyền truy cập tới một page publicly; đây là một ví dụ không được xác thực, truy cập được ủy quyền; chúng ta không biết họ là ai, nhưng họ có thể truy cập tài nguyên public như mọi người khác

Bản chất, Authentication là xác nhận người dùng tương tác với hệ thống, Authorization là cấp quyền cho người dùng được phép tương tác với phần nào của hệ thống.

Như vậy, muốn phân quyền thì trước tiên cần xác thực, có 2 loại người dùng là người dùng ẩn danh và người dùng có danh tính, khu vực vip muốn vào bắt buộc phải có đăng kí với hệ thống và khai báo danh tính.

* Discuss some typical scenarios nơi authentication và authorization có thể được yêu cầu hoặc hkoong
* Intro ASP.NET Core Identity, hệ thống thành viên hiện đại cái cho phép developers add tính năng login tới app của họ như là IdentityServer, middleware được thiết kế để add OIDC và Oauth 2.0 endpoints tới bất cứ ASP.NET Core app
* Implement ASP.NET Core Identity và IndetityServer để add tính năng login và registration tới app của chúng ta
* Nghiên cứu the Angular authorization API được cung cấp bởi .NET Core và Angular Visual Studio project template, cái mà thực thi the oidc-client npm package để tương tác với URI endpoints được cung cấp bởi ASP.NET Core Identity system, như là tính năng key Angular, giống như Route Guards và HTTP interceptors để xử lý toàn bộ luồng authorization
* Integrate the aforementioned back-end và front-end APIs tới project của chúng ta để đưa cho các user được trải nghiệm thoải mái authen và author

## To auth, or not to auth

Thực tế, việc thực hiện authentication hoặc authorization logic không bắt buộc cho phần lớn wep app hoặc services; có vài websites không cần làm cái này, bởi vì họ phục vụ content cái mà có thể được truy cập từ bất cứ nơi nào.

Chúng ta không cần là một expert dev cũng biết là WWW đã thay đổi trong vài năm trở lại đây; mỗi website bất kể mục đích gì đều muốn tăng tính năng theo dõi người dùng của họ, và chúng không thể nào được làm nếu không dùng cơ chế authentication

Hàng tỉ websites và services yêu cầu authentication để làm việc, phần lớn content hoặc intents của họ phục thuộc vào hành động của user đã đăng ký. Nếu mục đích của chúng ta là sản phầm về SPA các tính năng tương tác với người dùng, chúng ta muốn xác định, biết về người dùng của chúng ta với names, and email address. Nó chỉ là cách để nhận định ai sẽ có thể nhìn, add, update, or delete records của chúng ta.

### Authentication

Khi chưa xác thực, người dùng đi đến 1 trang HTML trống và điền thông tin vào, sau đó sẽ gửi lên server bằng POST command, web app nhận dc dữ liệu từ POST data và gọi server-side thực hiện việc xác thực user bằng dữ liệu vừa nhận và trả ra kết quả.

Nếu kết quả thành công thì web app sẽ xác thực user và lưu dữ liệu liên quan một vài nơi, phụ thuộc vào việc chọn phương thức xác thưc: sessions/cookies, tokens, signatures,...

## Proprietary auth with .NET Core

The authentications patterns được cung cấp bởi ASP.NET Core về cơ bản là giống với các phiên bản ASP.NET trước

* No authentication, nếu chúng ta không thích thực thi bất cứ cái gì
* Individual User Accounts, khi chúng ta set up một database nội bộ để lưu trữ dữ liệu user sử dụng ASP.NET Core Identity Interface
* Azure Active Directory
* Windows authentication

Bắt đầu với .NET Core 3.0, ASP.NET Core Identity đã được tích hợp với một cơ chế API authorization mới để xử lý authentication trong SPAs: tính năng mới này dựa trên IndentityServer, một phần của open source OIDC và Oauth 2.0 middleware

Với ASP.NET Core Identity, chúng ta có thể dễ dàng thực hiện một cơ chế login

* Introduce the ASP.NET Core Identity Model, the framework cung cấp bởi ASP.NET Core để quản lý và lưu user accounts
* Set up một ASP.NET Core Identity implementation bởi việc installing the NuGet packages được yêu cầu
* Extend the ApplicationDbContext
* Configure the Identity service
* Update the existing SeedController

Sau tất cả, chúng ta sẽ xem xét về ASP.NET Core Task Async Programming (TAP) model

### The ASP.NET Core Identity Model

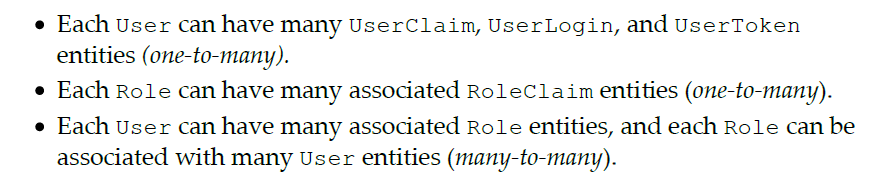
ASP.NET Core cung cấp một framework thống nhất để quản lý và luuw trữ user accounts cái mà có thể được sử dụng một cách dễ dàng trong .NET Core application; framework này được gọi là ASP.NET Core Identity và chúng cấp một bộ APIs cho phép developer xử lý các tasks sau:

* Design, set up, and implement user registration and login functionalities
* Manage users, passwords, profile data, roles, claims, tokens, email confirmation and so on
* Support external (third-party) login providers such as Facebook, Google, Microsoft Account, Twitter, and more

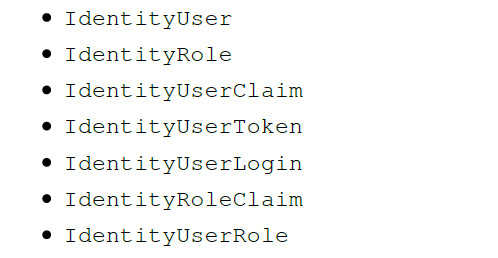
##### Entity types

The ASP.NET Core identity platform dựa trên theo entity types, mỗi một trong số chúng thể hiện một set of records xác định

* User: the Users of our application
* Role: the roles that we can assign to each user
* UserClaim: the claims that a user possesses
* UserToken: The authentication token that a user might user to perform auth-based tasks (ssuch as logging in)
* UserLogin: the login account associated with each user
* RoleClaim: the claims that granted to all users within ta given role
* UserRole:The lookup table to store relationship between users and their assigned roles



ASP.NET Core Identity cung cấp các default Common Language Runtime (CLR) types for each one of them



Các types này có thể được sử dụng như base classes cho việc thực thi của chúng ta, mỗi khi chúng ta cần định nghĩa chính xác một model là identity-related, hầu hết trong số chúng k được thực hiện trong phần lớn các kịch bản phổ biến

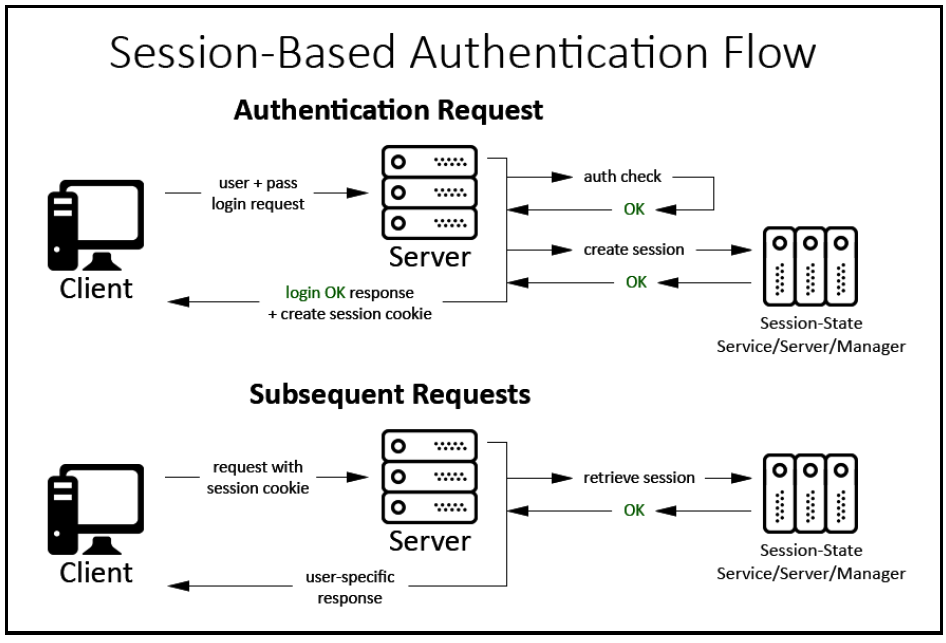
##### Authentication methods

Bây giờ chúng ta đã update cho database để nó có thể hỗ trợ ASP.NET Core Identity authentication workflow and patterns, chúng ta sẽ mất một chút thời gian để lựa chọn authentication method;một cách chính xác hơn, khi chuings ta đã thực thi .NET Core IdentityServer, để hiểu default authentication method cái mà cung cấp cho SAP – JWT tokens – là đủ an toàn để sử dụng hoặc chúng ta có thể thay đổi nó trong một vài cơ chế

Như chúng ta đã biết, giao thực HTTP là stateless, nghĩa là bất cứ cái gì chúng ta làm trong chu trình request/response sẽ bị mất trước khi yêu cầu tiếp theo, bao gồm cả authentication result. Cách duy nhất để khác phục điều này là chúng ta lưu kết quả đấy ở đâu đó, cùng với tất cả những dữ liệu liên quan tới nó, như user ID, login date/time.

##### Sessions

Từ vài năm trước đây, phương pháp truyền thống và phổ biến nhất để làm điều này là lưu data trên server sử dụng memory-based hoặc disk-based hoặc external session manager. Mỗi session có thể nhận được và sử dụng một ID duy nhất cái mà client nhận được với authentication response. Thường bên trong một session cookie, cái mà sẽ được truyền đến máy chủ cho mỗi yêu cầu tiếp theo



Kỹ thuật này vẫn rất phổ biến được sử dụng bởi phần lớn web app. Chẳng có gì sao với cách tiếp cận này, miễn là chúng ok với nó và vẫn thừa nhận nhược điểm sau

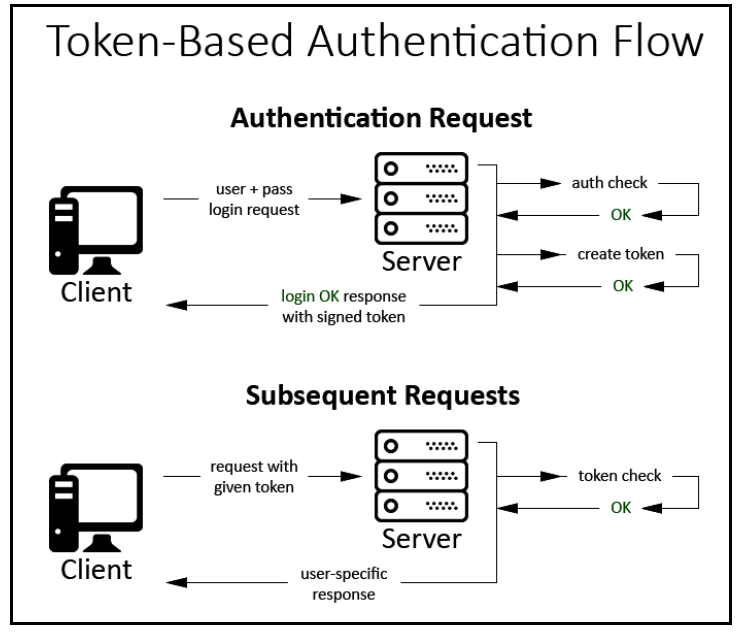
* Memory issues: Bất cứ khi nào có nhiều người dùng xác thực, web server sẽ tiêu thục nhiều bộ nhớ hơn. Thậm chí nếu chúng ta sử dụng một file-based hoặc external sesion provide, dù sao cái này sẽ có một I/O, TCP, hoặc socket ở bên trên
* Scalability issues: Việc nhân rộng một session provide trong một kiến trúc có khả năng mở rộng có lẽ không dễ dàng và sẽ thường dẫn tới tắc nghẽn hoặc lãng phí tài nguyên..
* Cross-domain issues: Session cookies hoạt động chỉ giống như standard cookies, vì chúng không dễ dàng shared giữa origins/domains khác nhau. Có vài vấn đề đã được giải quyết, nhưng giải quyết chúng lại gây ra bất ổn với thằng khác
* Security issues:

Tokens

Token-based authentication được sử dụng bởi SPAs và mobile apps trong vài năm vì một số lý do tốt

Điểm khác nhau quan trọng giữa thằng này với session-based là that the latter í stateless, có nghĩa rằng chugns ta không chứa bất cứ thông tin nào trong server memory, database, session provider...

Chúng ta không có sessions nên không bị tăng overhead; chúng ta không cần session provide vì vật scaling sẽ dễ hơn. For browser supporting LocalStorage, chúng ta thậm chí không sử dụng cookies, vì vậy chúng ta không nhaatnj blocked bởi cross-origin restrictive policies



Trong khái niệm về client-side interaction, ta thấy diagram không có sự khác nhau nhiều, chỉ có khác là chúng ta sẽ checking token thay vì việc tạo và nhận sessions.

##### Signatures

Được sử dụng bởi hầu hết các máy tính và lưu trữ đám mây

##### Two-factor

Cách này là tiêu chuẩn được sử dụng bởi phần lớn tài khoản ngân hàng và tài chính, được cho là an toàn nhất

Nó luôn theo một base workflow như sau:

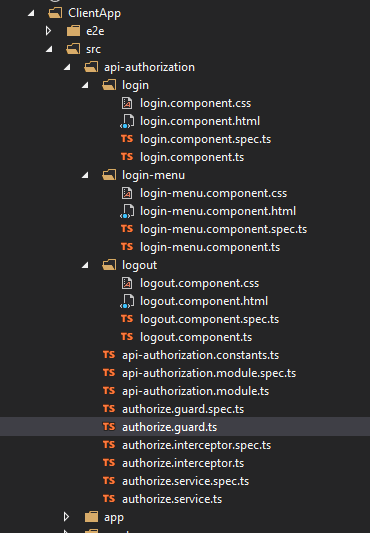
* Người dùng thực hiện login theo tiêu chuẩn với username và pass
* Server xác định người dùng và nhắc họ với một bổ sung, user-specific request cái chỉ có thể được thỏa mản bởi việc đạt đuộc cái gì đó: một OTP password sent bởi SMS,...
* Nếu user nhập đúng, họ được xác thực bằng việc sử dụng một standard session-based hoặc token-based method

### Implementing authentication in Angular

##### Exploring the Angular authorization APIs

Trong phần này, chúng ta sẽ có cái nhìn bao quát về việc cấp quyền APIs được cung cấp bởi .NET Core và Angular Visual Studio project template: một bộ các tính năng dự trên oidc-client library cái mà cho phép Angular app tương tác với URI endpoints được cung cấp bởi ASP.NET Core Identity system

Sau khi tạo project ta có một template của authenrization trên angular với một số component to vcl



* Api-authorization.module.ts file là một NgModule, là một container chứa bộ tính năng chung về authoriaztion APIs, giống như AngularMaterialModule chúng ta đã tạo



* Api-authorization.constants.ts fiel chứa một đống interfaces và constants
* Authorize.fuard.ts file giới thiệu khái niệm của Route Guards, đây là cái mà chúng ta chưa học
* Authorize.interceptor.ts file thực thi một HTTP interceptor class – cơ chế khác cái mà chúng ta chưa biết
* Authorize.service.ts file chưa Data Service cái sẽ xử lý tất cả HTTP request và responses

##### Route Guards

Là cơ chế để thực thi đúng yêu cầu về hạn chế, cấp quyền truy cập tới API. Chúng ta có thể thêm vào các config route để trả về các giá trị cái mà có thể control router’s behavior theo cách:

* Nếu Route Guard trả về true, the navigation process continues
* Nếu nó trả về false, the navigation process dừng lại
* Nếu trả về UrlTree, the navigation process bị hủy bỏ và thay thế bởi một navigation mới tới UrlTree

# Các chú ý hay sử dụng

1. Cách để request với parameter

