**Angular JS**

# MVC (Model – View – Controller)

## Khái Niệm

Các khái niệm cốt lõi đằng sau Angular JS Framework là mô hình kiến trúc MVC. Mô hình Model - View – Controller (hay MVVM – Model- View – View – Model) đã được phát triển như một cách để tách các đơn vị logic và những lo ngại khi phát triển các ứng dụng lớn hơn. MVC thể hiện tính chuyên nghiệp trong lập trình, phân tích thiết kế.

Mô hình kiến trúc **MVC** chia 1 ứng dụng thành 3 phần khác nhau *Model – View – Controller* , mỗi thành phần có 1 nhiệm vụ riêng biệt và độc lập với các thành phần khác nên giúp phát triển ứng dụng nhanh, đơn giản, dễ nâng cấp bảo trì. Đối với *Angular*, **View** sẽ là DOM, **Controller** là các lớp Javascript, còn **Model** sẽ là dữ liệu được lưu ở thuộc tính của các đối tượng trong JS.

1. **Models:**  trong các ứng dụng dựa trên **MVC, Model** là thành phần tương tác với cơ sở hoặc hệ cơ sở dữ liệu ,có nhiệm vụ lưu trữ thông tin, trạng thái của các đối tượng, thông thường nó là một lớp được ánh xạ từ một bảng trong **CSDL**. Nó sẽ bao gồm các class/function xử lý nhiều nghiệp vụ: kết nối database, truy vấn dữ liệu, thêm-sửa-xoá dữ liệu,...

Ví dụ chúng ta có lớp Product được sử dụng để mô tả dữ liệu từ bảng Products trong cơ sở dữ liệu…

1. **Views**: là các thành phần chịu trách nhiệm hiển thị các thông tin dữ liệu thông qua giao diện và giúp người dùng tương tác với hệ thống. Thông thường, các thông tin cần hiển thị được lấy từ thành phần **Model**.
2. **Controllers** : là nơi tiếp nhận những yêu cầu xử lý được gửi từ người dùng, nó sẽ gồm những class/ function xử lý nhiều nghiệp vụ logic giúp lấy đúng dữ liệu thông tin cần thiết nhờ các nghiệp vụ lớp **Model** cung cấp và hiển thị dữ liệu đó ra cho người dùng nhờ lớp **View.**

Một lợi điểm của **MVC** là nó bắt buộc phải tạo ra một sự rành mạch trong liên kết giữa 3 thành phần **Models**, **Views** và **Controllers** trong ứng dụng. Duy trì được mối quan hệ rành mạch rõ ràng này khiến việc kiểm tra ứng dụng trở nên dễ dàng hơn, đồng thời trách nhiệm của từng thành phần được định nghĩa rõ ràng và cả ba đều làm việc ăn ý với nhau.

## Hoạt động

Khi có một yêu cầu từ phía client gửi đến server, Bộ phận **Controller** có nhiệm vụ nhận yêu cầu, xử lý yêu cầu đó. Và nếu cần, nó sẽ gọi đến phần **Model**- bộ phận làm việc với Database.

Sau khi xử lý xong, toàn bộ kết quả được đẩy về phần **View**. Tại **View**, sẽ tạo ra mã html giao diện, và trả toàn bộ html về trình duyệt để hiển thị.

## Sự tương tác giữa các lớp trong mô hình MVC

* **Controller – View**sẽ lấy những hình ảnh, nút bấm…hoặc hiển thị dữ liệu được trả ra từ **Controller** để người dùng có thể quan sát và thao tác. Trong sự tương tác này cũng có thể không có dữ liệu được lấy từ **Model** và khi đó nó chỉ chịu trách nhiệm hiển thị đơn thuần như hình ảnh, nút bấm...
* **Controller – Model** là luồng xử lý khi controller tiếp nhận yêu cầu và các tham số đầu vào từ người dùng, **Controller** sẽ sử dụng các class/function trong **Model** cần thiết để lấy ra những dữ liệu chính xác.
* **View – Model** có thể tương tác với nhau mà không qua **Controller**, nó chỉ đảm nhận hiển thị dữ liệu chứ không phải qua bất kỳ xử lý nghiệp vụ logics nào. Nó giống như các vùng dữ liệu hiển thị tĩnh trên các website như block slidebar…

1. Dependence Injection (DI)

Đây có lẽ là kỹ thuật hay nhất của Angular JS. DI là một mẫu thiết kế, có thể hiểu đơn giản cách làm giảm sự phụ thuộc giữa các object khi xây dựng ứng dụng.

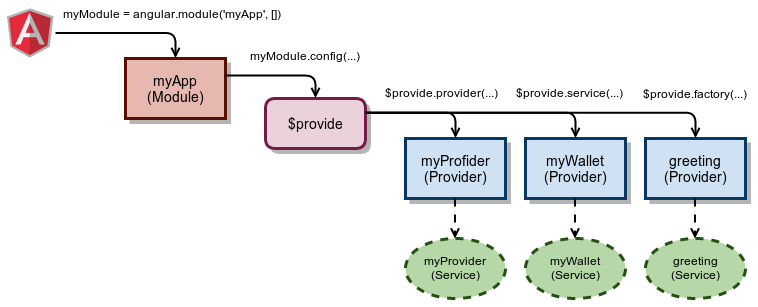
Nếu thiết kế một object A phụ thuộc vào nhiều object khác thì dẫn tới object A sẽ khó được quản lý cũng như khó thực hiện Unit Test – là kỹ thuật đòi hỏi phải cô lập các thành phần phụ thuộc.

**Các thuật ngữ quy ước**

* **Inject** : khái niệm để chỉ hành động lấy giá trị hoặc tham chiếu của một đối tượng (A), gán cho thuộc tính của một đối tượng khác (B) theo một cách nào đó.
* **Injectable object** là những đối tượng được AngularJS tạo ra từ khai báo của người dùng, do AngularJS quản lý và có thể inject vào bên trong các hàm hoặc đối tượng khác.
* **Injected object** : là đối tượng có chứa một số chỗ đánh dấu (thường là tham số của hàm khởi tạo) để AngularJS có thể inject các đối tượng vào bên trong nó.
* **Provider** là đối tượng do AngularJS tạo ra để chứa các định nghĩa về **Service** . AngularJS cung cấp một số provider tạo sẵn cũng như cho phép nhà phát triển ứng dụng định nghĩa thêm các provider của riêng mình.
* **Service :**  khái niệm này được AngularJS sử dụng để nói đến những đối tượng do AngularJS tạo ra, quản lý, và có thể được inject vào cho những thành phần khác.
* **Instance of Service** : khái niệm tương tự như **Service** và được dùng lẫn lộn với khái niệm này trong AngularJS.

Hai nhân tố cơ bản tạo dựng nên DI trong Angular JS chính là hai đối tượng dịch vụ cơ bản trong Module auto: $provide và $injector.

#### Đối tượng dịch vụ $provide

****

Đối tượng dịch vụ $provide có nhiệm vụ chỉ dẫn cho AngularJS cách thức tạo ra các injectable object - các đối tượng này còn được gọi là đối tượng dịch vụ của AngularJS. Những chỉ dẫn của $provide được “ghi lại” dưới dạng một loại đối tượng được gọi chung là provider. Định nghĩa provider được thực hiện thông qua hàm provider() của đối tượng dịch vụ $provide. $provide được AngularJS tạo ra khi khởi tạo ứng dụng và chúng ta có thể lấy nó ra bằng cách gọi hàm config() của module.

**var myModule = angular.module('myApp', []);**

**myModule.config(function($provide) {**

**$provide.provider('myProvider', function() {**

**this.$get = function() {**

**return function(name) {**

**alert("Hello, " + name);**

**};**

**};**

**});**

**});**

Đoạn mã trên tạo ra một provider có tên là myProvider. Với provider này, AngularJS sẽ chịu trách nhiệm: tạo ra một service bằng cách gọi hàm  $get()  định nghĩa bên trong myProvider; inject service vừa tạo này vào bất cứ **Injectable object** (đối tượng có thể inject) (chẳng hạn là một  Controller ) .  **Injectable object**  là những đối tượng được tạo ra từ một provider nào đó và chứa tham số có tên myProvider trong hàm định nghĩa provider của nó.

##### **Hàm factory()**

Để đơn giản hóa mã lệnh định nghĩa provider trong hàm provider(), AngularJS cung cấp hàm khai báo factory() là phiên bản giản lược của provider(), bằng cách giản lược

$provide.provider(‘myProvider’, function(){this.$get=function(){…}; );

thành

$provide.factory(‘myProvider’, function() { … });

Với cách này, ta đã giảm bớt được một cấp khối lệnh trong khai báo.

**myModule.config(function($provide) {**

**$provide.factory('myProvider', function($window) {**

**return function(name) {**

**$window.alert("Hello, " + name);**

**};**

**});**

**});**

##### **Hàm service()**

Hàm service() cho phép tạo ra đối tượng bằng cách sử dụng hàm khai báo làm hàm khởi tạo đối tượng. Giá trị trả lại của service() không phải là kết quả thực thi hàm khai báo như trong factory() mà là kết quả áp dụng toán tửnew đối với hàm khai báo đó

Hàm service() cũng là trường hợp đặc biệt của provider().

#### Đối tượng dịch vụ $injector

Đối tượng dịch vụ $injector có nhiệm vụ tạo ra các đối tượng thể hiện của dịch vụ (instances of services) từ các khai báo provider được định nghĩa bởi $provide, đồng thời inject các service đã có vào cho instances of services vừa mới tạo. Mỗi ứng dụng AngularJS có một đối tượng $injector được tự động tạo ra khi ứng dụng khởi động

Làm thế nào để lấy đối tượng $injector tương ứng với một ứng dụng? Chúng ta sử dụng hàm angular.injector(), với các tham số là danh sách module của ứng dụng cần nạp.

Giốngnhư các đối tượng **Service** của AngularJS và các đối tượng service do chúng ta tạo ra, $injector có thể được inject vào bất cứ hàm nào muốn có đối tượng này để thao tác:

**// $injector is an injectable object!**

**myModule.controller('myController', function($scope, $injector, myProvider) {**

**$scope.onClick = function() {**

**// do something with $injector**

**myProvider("AngularJS's users");**

**};**

**});**

##### **Hàm $injector.get()**

Khi đã có đối tượng $injector, chúng ta có thể lấy đối tượng thể hiện của các service đã được khai báo trước đó bằng cách sử dụng hàm $injector.get(). Nếu là lần đầu gọi đến, đối tượng thể hiện sẽ được $injector tạo mới, lưu lại và trả tham chiếu về cho người dùng. Trong những lần gọi instances of services tiếp theo, $injector.get() sẽ tìm trong danh sách có sẵn và trả về tham chiếu đến đối tượng có sẵn cho người dùng. Như vậy, 2 lần gọi liên tiếp của hàm $injector.get() sẽ cho ra cùng một instances of services .

##### **Hàm $injector.invoke()**

Đối tượng dịch vụ $injector còn có khả năng inject các instances of services của AngularJS cho các hàm bên ngoài thông qua hàm $injector.invoke().

Rõ ràng là chúng ta có thể inject các instances of services vào bên trong bất cứ hàm nào được kích hoạt bởi *$injector.invoke()*. Các hàm định nghĩa trong *AngularJS* được kích hoạt bằng *$injector.invoke()*(ngầm định bởi AngularJS) cũng có khả năng được *inject*. Các hàm này bao gồm:

* Các hàm định nghĩa *controller*
* Các hàm định nghĩa *directive*
* Các hàm định nghĩa *filter*
* Các hàm định nghĩa *factory*
* Các hàm định nghĩa *service*
* Các hàm *$get* trong định nghĩa *provider*

### Ưu điểm của Dependency Injection

* Tách ra các thành phần liên quan.
* Gimar code trong application class, tất cả việc khởi tạo dependency được tạo ra bởi các injector
* Ứng dụng dễ dàng mở rộng.
* Dễ viết unit test với các đối tượng giả lập.

### Nhược điểm

* Nếu quá lạm dụng sẽ dẫn đến đến gặp vấn đề bảo trì vì ảnh hưởng của các thay đổi là lúc runtime.
* Dependency injection ẩn các service class phụ thuộc, cho nên có thể chương trình sẽ lỗi lúc runtime thay vì chúng ta có thể bắt được ngay từ lúc biên dịch.

# Deep Linking

**Deep Linking:** Liên kết sâu cho phép bạn mã hoá trạng thái của ứng dụng trong các URL để nó có thể sử dụng lịch sử của trình duyệt và đánh dấu (bookmark) được với công cụ tìm kiếm.