# Wen Architecture

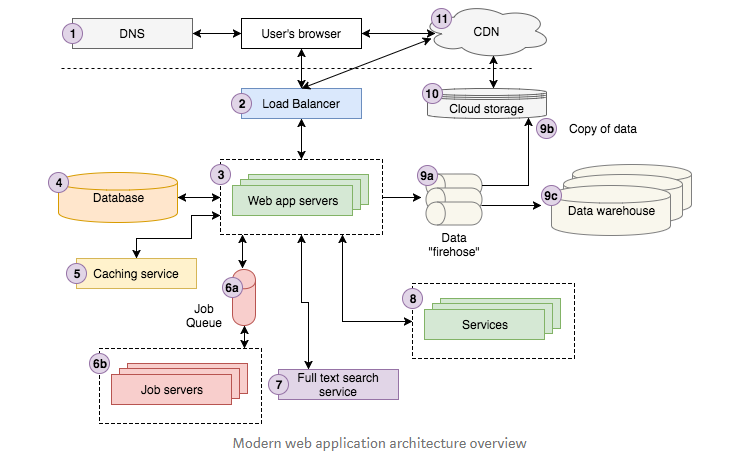


Diagram bên trên đại diện cho kiến trúc của chúng ta tại Storyblocks. Nếu bạn không phải là web developer có kinh nghiệm, bạn sẽ thấy phức tạp. Vì vậy chúng ta sẽ đi qua các phần dưới đây trước khi đi sâu vào các phần.

Một user tìm kiếm trên google với cụm từ “Strong Beautiful Fog And Sunbeams In The Forest”. Kết quả ta gặp được là trang Storyblocks. Khi mà user bấm vào kết quả dẫn đến việc điều hướng browser của chúng ta tới page đó. Bên dưới nó hoạt động như thế nào? Thì browser của chúng ta gửi một request tới DNS server để tra cứu tìm cách liên hệ với Storyblocks, và sau đó gửi request.

The request đến load balance của chúng ta, thằng này chọn ngẫu nhiên một trong 10 hoặc hơn mà web servers chúng ta đang chạy site tại thời điểm xử lý request. The web server tìm kiếm các thông tin về image từ caching service của chúng ta và fetches data còn lại ở database. Chúng ta nhận thấy rằng color profile cho image chưa được tính toán, vì vậy chúng ta gửi một “color profile” job tới job queue của chúng ta, cái mà job servers của chúng ta sẽ xử lý bất đồng bộ, updating database một cách phù hợp với kết quả.

Tiếp theo, chúng ta tìm kiếm các photos tương tự bằng việc gửi một request tới full text search service của chúng ta bằng việc sử dụng title của photo là input. User gặp phải việc logged into Storyblocks như một member để nhìn thấy thông tin tài khoản của họ từ account service. Cuối cùng chúng ta kích hoạt page view event để dữ liệu firehouse của chúng ta được ghi lại trên cloud storage system và được load vào trong data warehouse, cái mà phân tích việc sử dụng để giúp trả lời các câu hỏi về business.

The server bây giờ renders the view là HTML và gửi quay lại tới browser. The page chứa JS và CSS tài nguyên mà chúng ta load vào trong cloud storage system, cái này được connect tới CDN, vì vậy browser liên hệ với CDN để nhận được content. Cuối cùng, browser hiện page render tới user.

## DNS

DNS vai trò cho “Domain Name System” và nó là công nghệ xương sống cái mà làm lên the world wide web possible. Tại DNS mức độ cơ bản nhất cung cấp một cặp key/value để lookup một domain name (ví dụ: google.com) tới IP address (ví dụ: 85.129.83.120), cái này được yêu cầu để computer có thể định tuyết một request tới server thích hợp. Bạn cần DNS để look up the IP address cho một domain. Nôm na thì thằng này dùng để xem IP dựa trên domain đã có domain-IP

## Load Balancer

Trước khi lặn sâu vào chi tiết về load balancing, chúng ta cần quay lại thảo luận về horizontal vs. vertical application scaling. CHúng là cái gì và khác nhau như thế nào? Rất đơn giản, horizontal scaling có nghĩa rằng bạn sclae bởi việc thêm nhiều hơn machines vào trong pool of resources trong khi đó “vertical” là cái mà bạn thêm sức mạnh cho machine có sẵn như CPU, RAM

Trong web development, bạn thường muốn scale horizontal bởi vì nó đơn giản. Các Servers crash một cách ngẫu nhiên. Networks suy thoái. Toàn bộ data center thi thoảng offline. Việc có nhiều hơn một server cho phép bạn lên kế hoạc cho application tiếp tục chạy. Nói cách khác, app của ta thì tránh trường hợp server chết app chết theo. Thứ hai, horizontal cho phép bạn chia nhỏ các backend application trên các servers khác nhau. Và cuối cùng thì do thằng vertical không thể thêm nữa. Không có computer nào đủ lớn trên thế giới để làm tất cả các tác vụ tích toán. Storyblocks chạy từ 150 tới 4000 AWS EC2 instances.

OK, quay lại load balancers. Chúng định tuyến các requests đến tới một trong các application servers. Bất kì 1 thằng nào trong số chúng đều xử lý request giống nhau vì vậy chỉ là vấn đề việc phân phát request tới server để chúng không bị quá tải.

## Web Application Servers

Tại bậc cao web application servers tương đối đơn giản để mô tả. Chúng execute logic cái mà xử lý request của user và gửi HTML lại tới browser. Để làm job của chúng, chúng giao tiếp với các backend infracture khác như database, cachung layers, job queues, search services, other microservices,...

Bạn nên biết rằng app server triển khai thì chọn một ngôn ngữ xác đinh (Node.js, Ruby, PHP, Scala, Java, C# .NET,etc) và một web MVC framewornk cho ngôn ngữ (Express cho Node.js, Ruby cho Rails, Play cho Scale, Laravel cho PHP, etc.). Tuy nhiên, đi sâu vào chi tiết của các ngôn ngữ và frameworks thì vượt quá phạm vi của bài viết này.

## Database Servers

## Caching Service

Một caching service cung cấp một key/value data store cái mà có thể lưu và xem thông tin. Applications tận dụng caching services để lưu kết quả của tính toán phức tạp vì cái mà nó có thể nhận kết quả từ cache thay vì tính toán lại một lần nữa. Một application có lẽ cache các kết quả từ một database query, gọi tới external serivces, HTML cho việc trao URL, và nhiều hơn thế. Ở đây có các ví dụ thực tế:

* Google caches search kết quả cho các tìm kiếm chung như là “dog” hoặc “Taylor Swift” mà không cần tính toán lại chúng mỗi lần
* Facebook caches nhieeud dữ liệu bạn nhìn thấy như khi bạn log in, như bạn post data, friends,etc. Chi tiết về kĩ thuật mà facebook cache <https://medium.com/@shagun/scaling-memcache-at-facebook-1ba77d71c082>
* Storyblocks caches HTML output từ server-side

## Job Quêu & Servers

Phần lớn web applications cần làm việc một cách bất đồng bộ bên dưới nền.

Trong khi có nhiều kiến trúc khác nhau có thể làm bất đồng bộ, thì phần lớn chúng gọi là “job queue” architecture. Nó bao gồm 2 phần: một queue của các jobs cái mà cần run và một hoặc nhiều hơn job servers (thương được gọi lad worker) chịu trách nhiệm run các jobs trong queue

Job queue lưu giữa một list các jobs cái mà cần chạy bất đồng bộ. Đơn giản là firt-in-first-out (FIFO) queues

## Full-text Search Service

## Services

## Data

Ngày nay, các công ty sống hoặc trên đều dựa trên cách mà họ khai thác data. Phần lớn mọi app khi mà đã có quy mô nhất định, tận dụng data pipeline để chắc chắn rằng data có thể được thu thập, được lưu và được phân tích. Một pipeline điển hình có 3 giai đoạn:

1. App gửi data, thường là các events về sự tương tác của user, data “firehouse” cung cấp streaming interface để ăn và xử lý data. Thỉnh thoảng raw data thì được chuyển đổi hoặc tăng cường và được pass tới firehose khác. AWS Kinesis và Kafka là 2 công nghệ phổ biến cho việc này.
2. The raw data cũng như data được chuyển đổi được lưu trên cloud storage. AWS Kinesis cung cấp một setting được gọi là “firehose” cái mà làm việc lưu trữ raw data tới cloud config một cách dễ dàng
3. The data transformed/augmented data thường được laod vào trong một data warehouse cho việc phân tích

## Cloud Storage

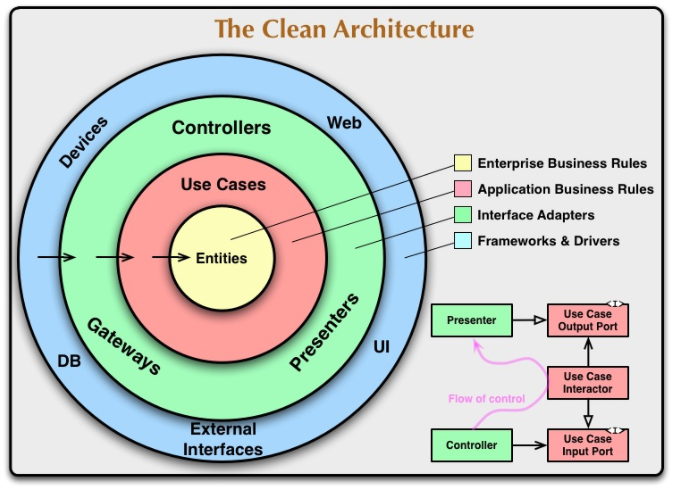
## CDN

CDN viết tắt bởi “Content Delivery Network” là công nghệ cung cấp phục vụ tài nguyên như HTML, CSS, JS và images làm cho web nhanh hơn. Nó hoạt động bằng cách phân phối nội dung trên nhiều “edge” servers trên khắp thế giới để user tải assets từ “edge” server thay vì origin server

# Better Sofware Design with Clean Architecture

## How Clean Architecture works

Chìa khóa bên dưới Clean Architecture là: **The Dependency Rule.** Các ý chính của thằng này thì đơn giản là các dependencies được đóng với vào mỗi “ring” của architecture model và các dependencies này chỉ có thể trỏ vào bên trong



### The Dependency Rule

Các vòng tròn đồng tâm đại diện cho các khu vực khác nhau của software. Nói chúng, vòng tròn càng xa thì càng ở level cao của software. Các vòng tròn ngoài là các cơ chế. Các vòng tròng trong là các chính sách.

Architecture này làm việc là *The Dependency Rule*. Rule này nói rằng *source code dependencies* can only *inwards.* Không một vòng tròn bên trong nào có thể biết bất cứ cái gì ở vòng tròn bên ngoài. Đặc biệt, tên của một cái gì đó được khai báo ở vòng tròn ngoài không bao giờ được đề cập bởi code ở trong vòng tròn trong. Cái này bao gồm functions, classes, variables hoặc bất cứ software entity được đặt tên.

Vì lẽ ấy, data forrmats được sử dụng trong một vòng tròn ngoài sẽ không được sử dụng bởi một vòng tròn trong, đặc biệt nếu các formats được generate bởi một framework trong một vòng tròn ngoài. Chúng ta không muốn bất cứ một cái gì ở vòng tròn ngoại tác động vòng tròn trong.

### Entities

Entities đóng gói *Enterprise wide* business logic rules. Một Entity có thể là một object với các methods hoặc nó có thể là một set of data structures and functions. Các entities có thể được sử dụng bởi nhiều applications trong doanh nghiệp.

Nếu bạn không có một enterprise mà chỉ viết một application đơn, các entities này là business objects của application. Chúng ít khi thay đổi khi cái gì đó bên ngoài thay đổi.

### Use Cases

The software trong layer này chứa *application specific* business rules. Nó đóng gói và thực thi tất cả các trường hợp sử dụng của hệ thống. Các use cases này theo flow of data from entities và các entities này trực tiếp sử dụng *enterprise wide* business rules để đạt được mục tiêu của use case

Chúng ta không mong muốn khi mà có sự thay đổi trong layer này sẽ gây ảnh hưởng tới the entities. Chúng ta cũng không mong mong muốn layer này thayu đổi làm cho thay đổi các yếu tố bên ngoài khác như database, UI hoặc bất cứ cái nào khác.

### Interface Adapters

The software trong layer này là một set of adapters cái mà convert data từ forrmat này tới format khác

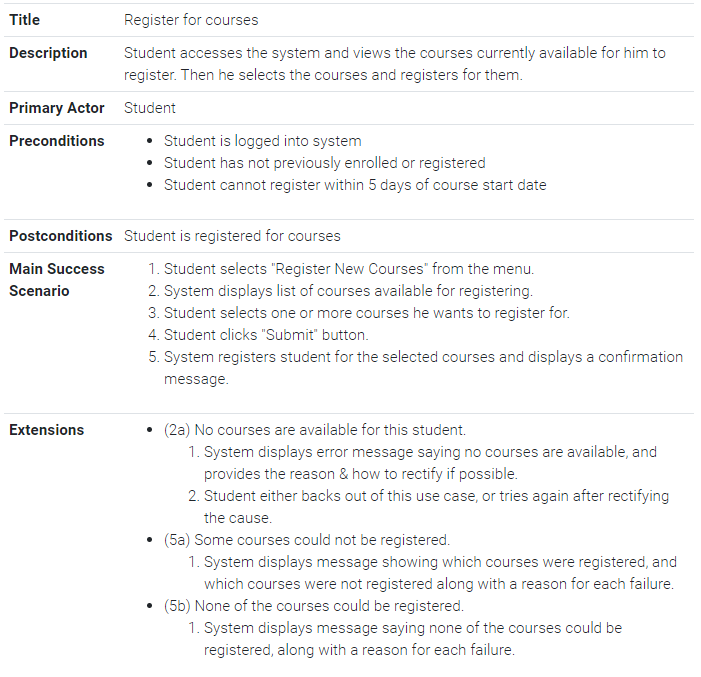
Quay lại với Clean Architecture, nó giữ các chi tiết như web frameworks và database trong outer layers trong khi business rule quan trọng và policies thì được đặt trong inner và không biết bất cứ cái gì ngoài chúng

### Implementing the “Course Registration” use case (đăng kí lớp học)

* Title – Register of Courses
* Description – Student truy cập vào hệ thống và xem các lớp đang available cho anh ta. Sau đó đăng kí lớp mà anh ta muốn
* Primary Actor – Student
* Preconditions – Student phải đăng nhập vào hệ thống, không thể đăng kí lớp khi mà nó đã bắt đầu được 5 ngày
* Postconditions – Student được đăng kí các khóa học
* Main Success/ Scenario

1. Student chọn “Register New Courses” từ menu
2. Hệ thống hiển thị danh sách lớp học có thể đăng kí
3. Student chọn một hoặc nhiều lớp anh ta muốn đăng kí
4. Student click submits button
5. Hệ thống xác nhận các lớp đã đăng kí

* Extensions



#### Creating the Entities

Entities là trái tim của clean architecture và chứa *enterprise-wide* business rules and logic. Chìa khóa của thằng này là chúng không thể chứa *application specific* – vì vậy bất cứ cái gì là global và có thể dùng chung trong các application khác thì sẽ được đóng gói trong một entity.

Chúng ta có 2 entities : Student and Course



#### Use Cases

Di chuyển lên trên ta sẽ có Use Case Layer. Các classes này chịu trách nhiệm cho:

* CHưa application specific business rules
* Đóng gói và thực thi tất cả các trường hợp của hệ thống. Một good rule thì bắt đầu với một class mỗi use case
* Dữ liệu đến từ các entities và có thể dựa trên business của nó để đạt được mục đích của use case
* Không phụ thuộc database, UI,...
* Gần như chắc chắn sẽ refactoring nếu chi tiết về use case thay đổi

Các class thường theo sau bởi cụm **Interactor**