1. **Introduce your self:**

Good afternoon, thank you for having me here. My name’s Viet. I’m glad to for this opportiunity to introduce my self. I have graduated Hust last month. In university, my major is automaion control engineering. I have been self-learning IOS development IOS for 8 months through searching and reading documents. I’m also a friendly person.

Strength:

I’m a self-taught. So I’m good at learning and researching and how to resolve problem by my self. I always try to learn about new technich. Beside, I have gratued Hust with very good degree, it demonstrate I’m a responsible person and always try to complete group project to the best of my ablitity.

Weak:

Although, I have ablitity for reading almost documents in English, I think speaking skills is a little bit low, and I know that English is important for programming. So I know that I need to improve it in near future.

**2. OOP in swift**

* Encapsulation: Là cách ta đóng gói các dữ liệu liên thành 1 class, điều này giúp ẩn đi dữ liệu với các đối tượng bên ngoài thông qua sử dụng các access modifier như open access, public, inernal, private access.
* Abstraction: Là cách ta hiển thị dữ liệu và phương thức mong muốn của 1 đối tượng, và ẩn đi các logic phức tạp bên trong. Điều này làm code dễ đọc, dễ maintain và sử dụng trong tương lai.
* Inheritance: Cho phép 1 class kế thừa 1 class khác.
* Polymorphism: Với cùng 1 method ở superclass, các subclass khác nhau có thể ovveride method đó để triển khai các cách khác nhau.

1. **Protocol là gì:**

Protocol: định nghĩa 1 blueprint cho các methods, propertie. Protocol có thể được adopted bởi class, struct, enum. Bất kì kiểu dữ liệu nào mà đáp ứng những methors và properties đó thì được gọi là đã **comform protocol.**

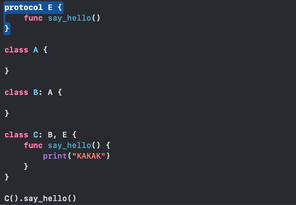
Ta thường dùng protocol cho các trường hợp như:

* Khi ta muốn đảm bảo 1 kiểu dữ liệu phải có đủ các methods và properties mong muốn.
* Delegate
* Đa kế thừa

Trong swift, delegate là 1 design pattern cho phép 1 đối tượng gửi message tới 1 đối tượng khác khi có 1 sự kiện xảy ra. Khi sử dụng delegate, ta khai báo nó với kiểu tham chiếu weak để tránh retail cycle. Delegate là kiểu quan hệ 1 1

1. **Kế thừa trong Swift là đơn kế thừa hay đa thừa kế? Muốn đa thừa kế thì phải làm thế nào? Swift có cho phép kế thừa nhiều mức không ?**

Trong swift, 1 class chỉ có thể kế thừa 1 super class, nên nó là đơn kế thừa. Nhưng ta có thể sử dụng **protocol** để có thể đạt được cơ chế như đa kế thừa(**multiple inheritance**).



Như ở VD trên, ta cũng thấy được trong swift cho phép **multiple level inheritance.** Cái này hiển nhiên thoi vì với TableView và CollectionView, thì 2 View đó kế thừa UIScrollView, sau đó ta tự tạo 1 class khác để kế thừa 2 View đó.

1. **GCD là gì**

Trước khi nhắc đến GCD thì ta cần phải nhắc tới Queue. Queue là 1 hàng đợi block code để đợi các Thread thực thi. GCD là 1 low level API được cung cấp để quản lý Queue, công việc chính của GCD là: Tạo Queue, Đưa các block code vào trong Queue. GCD cho phép ta tạo 2 loại Queue:

* Serial Queue: Các block code được thực thi lần lượt
* ConcurrentQueue: Các block code được thực thi đồng thời.

1. **So sánh Class và Struct.**

Giống nhau:

* Đều có thể định nghĩa các thuộc tính để lưu giá trị, định nghĩa các method để thực thi các chức năng, định nghĩa các hàm khởi tạo init. Đều có thể kế thừa protocol.

**Khác nhau:**

* Điểm khác biệt lớn nhất là **Class là reference Type còn Struct và Enum là Value Type.**
* Điểm khác biệt thứ 2 là **Class cho phép kế thừa còn Struct thì không.**

1. **Có bao nhiêu mức Access Control trong Swift? Hãy liệt kê chúng theo thứ tự giảm dần**

Trong swift có 5 mức Access Control đó là: **Open, Public, Internal, fileprivate, private**

* Với **Open Access:** Thì class đó có thể được truy cập và kế thừa từ mọi source file mà import module này, các phương thức hoặc thuộc tính của class Open có thể được **Override từ các source file đó.**
* Với **Public Access: Thì class Public đó chỉ có thể được truy cập từ các source file mà import module này, chứ ko thể kế thừa hoặc override các thuộc tính các class public.**
* Với **Internal Access:** Chỉ các entity trong Module đó mới có quyền truy cập vào class Internal
* Với **fileprivate Access:** Chỉ các entity trong file đó mới có quyền truy cập vào fileprivate class
* Với **Private Access:** Chỉ có thể được truy cập ở bên trong class khai báo nó.

1. **Quản lý bộ nhớ trong IOS thế nào**

Swift sử dụng ARC để theo dõi và quản lý memory. ARC sẽ theo dõi số lượng refercence tới instance. Khi số lượng reference về 0, vùng nhớ đó sẽ được thu hồi.

1. **ARC và Non-ARC**

Khi sử dụng các project Non-ARC, người dùng sẽ phải tự mình quản lý vòng đời của bộ nhớ.

**ARC là Automatic Reference Counting**. Swift sử dụng ARC để theo dõi và quản lý bộ nhớ của ứng dụng. ARC theo dõi số lượng tham chiếu đến vùng nhớ của Class. Khi số lượng tham chiếu bằng 0, ARC sẽ tự động thu hồi vùng nhớ này.

. ARC là 1 thuộc tính của trình biên dịch, thay vì người dùng phải tự mình thu hồi các vùng nhớ ko sử dụng thì compiler sẽ làm nó

1. **Strong và weak là gì?**

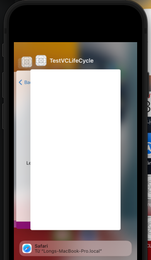
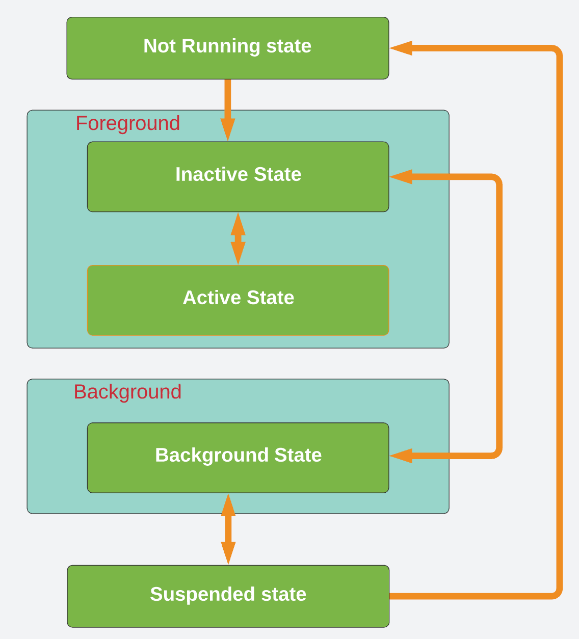
Strong Reference là 1 kiểu tham chiếu được sử dụng để giữ đối tượng trong bộ nhớ. Weak Reference không làm ăng . Weak thường được sử dụng để tránh retain cycles

Cái này tự chém thêm 😊

1. **IOS Application Life Cycle**

Các trạng thái của 1 IOS Application là: **Not Running,** **InActive, Active, Background, Suspended.**

* **Not Running:** Khi chúng ta mở máy lên thì chưa có ứng dụng nào được chạy, và tất cả các App ở trạng thái **Not Running**
* **Active:** Sau khi ta nhấn vào biểu tượng App trên điện thoại chúng ta, thì App chúng ta vào trạng **thái InActive(Kiểu lúc nó mới Lauch thì ra màn hình trắng, và ko thể tương tác với UI của App vì nó chưa được load xong UI 😊)) rồi về trạng thái** **Active và chạy trên Foregroud,** lúc này người dùng có thể tương tác với giao diện của App. Khi App đã được mở sẵn nhưng chạy trên **Backgroud,** thì khi người dùng mở lại App thì App đang từ trạng Background sẽ về trạng **InActive rồi về Active**
* **In-Active:** Khi người dùng làm như ảnh bên dưới hoặc trong các trường hợp như có các cuộc gọi hoặc tin nhắn tới, thì lúc này App rơi vào trạng thái **In-Active. Lúc này App** vẫn chạy trên ForeGround, nhưng không nhận bất kì sự kiện tương tác nào của người dùng.

* **Background:** Khi người dùng nhấn home để về màn hình chính hoặc sử dụng 1 App khác thìApp hiện tại sẽ chạy dưới background nhưng vẫn thực thi code. **Khi người dùng nhấn home, App từ trạng thái Active sẽ về In-Active rồi về trạng thái Backgroud**
* **Suspended:** App chạy dưới Backgroud nhưng không thực thi code.
* **Cái này nói về trong Code:** Ta có 1 hàm sceneWillResignActive(). Hàm này sẽ được gọi khi App đang từ trạng thái Active về trạng thái In-Active

***Khi nào App chạy trên background nhưng thực thi code, và khi nào ko thực thi code ?***

* Thông thường app thực thi code và chạy trên background trong các trường hợp dưới đây: Các App Audio, Location Updates,..
* Tuy nhiên khi App không thực hiện các TH trên, nó sẽ rơi vào trạng thái suspended bởi hệ thống. Cần lưu ý rằng có những hạn chế bổ sung về những gì ứng dụng có thể thực hiện trong backgroud, tùy thuộc vào cài đặt thiết bị của người dùng, thời lượng pin và các yếu tố khác.

1. **Sự khác nhau của AppDelegate và SceneDelegate**

**Tạm bỏ**

1. **Trình bày mô hình MVC và MVVM trong swift.**

Mô hình MVC gồm 3 thành phần đó là Model, View, Controller::

* Model: Đại diện cho dữ liệu, chứa thông tin và loginc của ứng dụng
* View: Đại điện cho giao diện người dùng của ứng dụng, hiển thị dữ liệu từ model, cho phép người dùng tương tác với ứng dụng
* Controller: Đại diện cho login điều khiển của ứng dụng, là cấu nối giữa model và view, xử lý các yêu cầu từ view sau đó update view và model.

Mô hình MVVM gồm 3 thành phần đó là Model, View và ViewModel

* ViewController: Chỉ thực hiện những công việc liên quan đến giao diện: Như là lấy và hiển thị thông tin
* View Model: Nhận thông tin từ ViewController, xử lý tất cả các thông tin này, sau đó gửi lại cho VC
* Model: Đại diện cho dữ liệu của chúng ta

Flow dữ liệu của MVVM:

* Đầu tiên 1 ViewController sẽ có 1 tham chiếu tới ViewModel
* ViewController sẽ nhận 1 vài hành động của user, và sau đó sẽ gọi tới viewModel
* viewModel sẽ request APIService, sau đó trả response ngược lại cho viewModel
* Khi viewModel nhận response, ViewModel sẽ thông báo cho ViewController thông qua Binding
* View sẽ update UI với Data vừa nhận được.

1. **PhotosKit**

Các App Photos quản lý tài nguyên của ảnh thông qua tập hợp APIs gọi là PhotoKits. Ta đa số làm việc với assets trong photoKits. Mỗi Asset chứa các thông tin meta data của image như thời gian tạo, location, favourite,…

* Đầu tiên ta cần fetchAsset(), với các option ta truyền vào như sort, lọc
* Sau khi fetch được Asset(), ta tiến hành fetch Imgae.

1. **AVFoundation:**

Capture Session là 1 đối tượng quản lý các hành động capture, điều phối luồng dữ liệu từ input tời ouput.

1. Closure:

Closure là một khối chức năng mà có thể được sử dụng trong code. Closure có thể capture và store reference. Trailing colusre cho phép ta viết block code bên ngoài dấu ngoặc đơn khi nó là argument cuối cùng của function.

1. Core Data:

Core Data là 1 framework được sử dụng để quản lý model object layer. Chúng ta sử dụng CoreData để filter, save and track data mà không cần học hay sư dụng tới SQL.

1. Git và github

Git là source control, cho phép hệ thống lưu trữ mọi thay đổi của source code, hỗ trợ nhiều người làm việc 1 lúc, các tính năng của git có thể kể đến là: Revert các thay đổi, đưa code về version cũ hơn. Có 3 khai niệm quan trọng đó là:

* Repo: Là nơi chứa toàn bộ source code và commit, repo có 2 loại: repo trong máy và cloud
* Commit:
* Branch

1. **Core data**

**+Chốt cơ bản lại:**

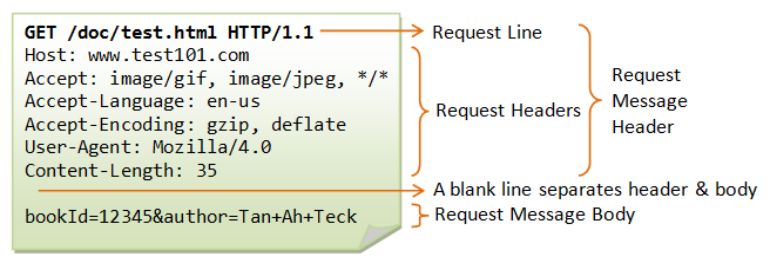
* NSManagedObjectModel : Là thằng tạo ra các relation trong database quy định các trường, kiểu dữ liệu, tên biến trong database.
* NSPersistentStoreCoordinator: Là thằng có nhiệm vụ ghi dữ liệu vào ổ cứng, là cầu nối trung gian giữa database với app.
* NSManagedObjectContext: Là đối tượng mà người lập trình sử dụng để tương tác, lưu giá trị,…

1. **HTTP**

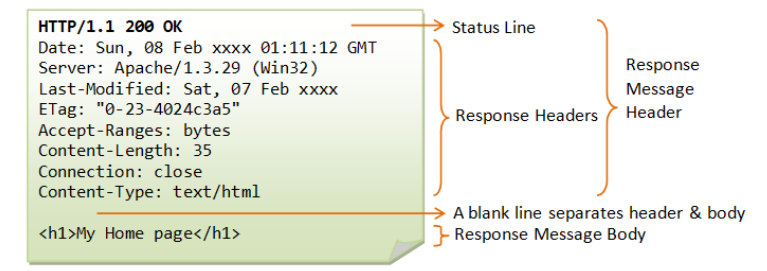
Http là 1 protocol mà cho phép truyền dữ liệu qua internet. Dữ liệu có thể là: text, video, âm thanh.

Để có thể nhận được resource từ server, client gửi 1 bản tin http request tới server, server sẽ gửi lại 1 http response.

Cấu trúc của 1 bản tin http request là:



HTTP Response:



* Get: Được sử dụng để requet 1 tài nguyên từ server
* Post: Được sử dụng để tạo mới hoặc thêm tài nguyên (resource) vào server. HTTP Post thường có body, và các thành phần header như: content-type, content-length.
* Put: Được sử dụng để modify resource, Put sẽ update **toàn bộ** data mà được truyền vào phần body, nếu ko có tài nguyên nào ứng với request, nó sẽ khởi tạo tài nguyên mới.
* Patch: Được sử dụng để update **1 phần** resource.
* Delete: Được sử dụng để xóa resource.

So sánh các phương thức:

* Post và Put: Mỗi lần gọi Post thì sẽ tạo ra 1 tài nguyên mới, dẫn đến gọi nhiều lần Post sẽ sinh ra nhiều tài nguyên. Còn với Put thì cái này chỉ sửa resource đã có, dẫn đến khi gọi nhiều lần cũng chỉ ra cùng 1 kết quả.
* Put và Patch: Với Put sẽ tạo mới tài nguyên nếu chưa có tài nguyên, hoặc sẽ sửa lại toàn bộ tài nguyên. Còn với Patch chỉ update 1

RESTFULApi: Là 1 Api mà thỏa mãn được các rang buộc của kiến trúc Rest. Các rang buộc đó là:

* Sử dụng kiến trúc client-server
* Stateless: Nghĩa là các thông tin của client sẽ không lưu trữ lại tại server, mỗi request là độc lập
* Uniform Interface: Mỗi 1 resource chỉ có 1 identifier(URI)
* Cachable: Liên quan tới khả năng lưu lại và sử dụng lại responses của server. Điều này làm cải thiện hiệu năng ứng dụng, Khi nào server response lại header là cachable thì thông tin đó mới có thể được lưu lại.
* Layer system:: client có thể kết nối tới bên ủy quyền trung gian, và vẫn có thể nhận được response từ server. Chúng ta có thể thiết kế nhiều server.
* Code on demand: Server có thể custom các chức năng của client bằng cách gửi 1 đoạn code tới client. VD khi user register 1 form, brosewr có thể hightlights lỗi luôn nhờ server.