**MỤC LỤC**

LỜI MỞ ĐẦU 4

CHƯƠNG 1: GIỚI THIỆU iOS 6

1. Khái niệm iOS 6
2. Thiết lập môi trường lập trình iOS 6
3. Tài khoản Apple Developer 12
4. Chạy ứng dụng (Build App) 12

CHƯƠNG 2: GIỚI THIỆU XCODE 14

1. Xcode IDE (Môi trường phát triển tích hợp) 14
2. Cách tạo ứng dụng trên Xcode 14
3. Giao diện chương trình (Interface Builder) 17
4. Chạy ứng dụng (Simulator) 18
5. Trình biên dịch (Apple LLVM) 20

CHƯƠNG 3: GIỚI THIỆU SWIFT 22

1. Giới thiệu 22
2. Cơ bản về lập trình Swift 23
3. Toán tử cơ bản 31
4. Kiểu String và kiểu Character 37
5. Các kiểu tập hợp (Collection Type) 39
6. Vòng lặp (Control Flow) 47
7. Hàm (Function) 54
8. Kiểu liệt kê (Enumeration) 56
9. Lớp và cấu trúc (Classes And Structures) 58
10. Thuộc tính (Properties) 62
11. Phương thức (Method) 62
12. Các hàm con (Subscript) 64
13. Kế thừa (Inheritance) 65
14. Khởi tạo (Initialization) 68
15. Hàm huỷ (Deinitialization) 70
16. Optional Chaining 71
17. Xử lý lỗi (Handling Error) 72
18. Một số control quan trọng 74
19. Đa luồng (Multi Thread) 84

CHƯƠNG 4: MỘT SỐ CÔNG NGHỆ PHỤ TRỢ 88

1. Firebase Platform 88
2. Mô hình MVC 100

CHƯƠNG 5: GIỚI THIỆU BÀI TOÁN HỖ TRỢ TÌM ĐƯỜNG ĐI NGẮN NHẤT GIỮA NHIỀU CẶP ĐIỂM 104

1. Đặt vấn đề 104
2. Giải quyết vấn đề 104

CHƯƠNG 6: THIẾT KẾ & XÂY DỰNG HỆ THỐNG 105

1. Mô hình UML 105
2. Thiết kế cơ sở dữ liệu (No SQL) 125
3. Thiết kế giao diện 132
   1. Đối với chủ Shop 132
      1. Đăng nhập 132
      2. Đăng kí 132
      3. Màn hình chính 133
      4. Đăng bài 136
      5. Quản lý bài đăng 141
      6. Quản lý tài khoản 142
   2. Đối với Shipper 143
      1. Đăng nhập 143
      2. Đăng kí 145
      3. Xem danh sách bài đăng 146
      4. Nhận bài đăng 147
      5. Quản lý bài nhận 148
      6. Xem đường đi 149
      7. Quản lý tài khoản 150
4. Hiện thực ứng dụng 151

CHƯƠNG 7: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 152

1. Tổng kết kết quả đạt được: 152
2. Hạn chế 152
3. Hướng phát triển 153

TÀI LIỆU THAM KHẢO 154

**LỜI MỞ ĐẦU**

Đầu tiên chúng em xin gửi lời cảm ơn tới các thầy cô trong trường Đại Học Sư Phạm Kỹ Thuật TP. Hồ Chí Minh, đặc biệt là các thầy cô trong Khoa Công Nghệ Thông Tin, đã nhiệt tình giảng dạy, trao dồi, vun đắp kiến thức cho chúng em trong nhiều năm ngồi trên ghế nhà trường.

Chúng em chân thành cảm ơn Thầy Nguyễn Thiên Bảo trong thời gian qua đã cung cấp cho chúng em những kiến thức về cũng như tài liệu rất hữu ích và hướng dẫn chi tiết trong quá trình chúng em làm để tài. Thầy luôn giải đáp tận tình những vướng mắc khi chúng em gặp phải.

Trong phạm vi khả năng cho phép, chúng em đã rất cố gắng để hoàn thành đề tài một cách tốt nhất. Tuy nhiên không thể tránh khỏi nhiều thiếu sót, chúng em kính mong nhận được sự cảm thông và những ý kiến đóng góp của quý thầy cô và các bạn.

[IOS](https://www.dienmayxanh.com/dien-thoai?g=ios-iphone) là một hệ điều hành [điện thoại di động](https://www.dienmayxanh.com/dien-thoai) được phát triển bởi Apple và phân phối độc quyền cho phần cứng của Apple. Ban đầu, iOS được công bố năm 2007 cho [iPhone](https://www.dienmayxanh.com/dien-thoai-apple-iphone), sau đó được mở rộng để hỗ trợ các thiết bị khác của Apple như iPod Touch (tháng 9 năm 2007), iPad (tháng 1 năm 2010), iPad Mini (tháng 11 năm 2012) và thế hệ thứ hai của Apple Ttivi trở đi (tháng 9 năm 2010).

Đến cuối năm 2011, iOS chiếm 60% thị phần [điện thoại thông minh](https://www.dienmayxanh.com/tag/dien-thoai-smartphone) và máy tính bảng. Cuối năm 2012, iOS chiếm 21% thị phần hệ điều hành điện thoại thông minh trên thị trường và 43,6% thị trường hệ điều hành máy tính bảng. Và đến tháng 6 năm 2014, App Store của Apple chứa hơn 1.200.000 ứng dụng iOS, và được tải về hơn 60 tỷ lần.

Chính vì khả năng chiếm lĩnh thị trường lớn như vậy, nên thế giới cũng như ở Việt Nam đã có nhiều lập trình viên theo học và lập trình ứng dụng cho hệ điều hành iOS. Hiện tại trên thị trường, tài liệu để lập trình được trên hệ điều hành iOS rất nhiều nhưng đa phần là tiếng nước ngoài, tài liệu bằng tiếng Việt rất ít. Hơn nữa, iOS là hệ điều hành còn rất mới so với sinh viên. Cho nên việc học lập trình ứng dụng trên hệ điều hành iOS còn gặp rất nhiều khó khăn.

Vì vậy, nhóm tiểu luận chuyên ngành sẽ nghiên cứu, tìm hiểu về hệ điều hành iOS và ngôn ngữ Swift dùng để lập trình ứng dụng trên iOS, đồng thời đưa ra một ứng dụng kết nối các Shipper và các chủ Shop lại với nhau và gợi ý cho Shipper giao hàng ít tốn chi phí nhất có thể. Tiểu luận bao gồm các chương như: Tìm hiểu hệ điều hành iOS, tìm hiểu công cụ lập trình Swift, tìm hiểu ngôn ngữ Swift, tìm hiểu các hỗ trợ khác như: cơ sở dữ liệu được lưu trữ bằng Firebase, và mô hình MVC.

Về mặt kiến thức, đồ án sẽ mang đến nhiều kiến thức tổng hợp từ cơ bản đến nâng cao về lập trình bằng ngôn ngữ swift. Đồng thời nâng cao được khả năng lập trình, rèn luyện được khả năng tư duy và suy luận để đưa ra những giải pháp an toàn khi tổ chức một ứng dụng.

Về mặt ứng dụng, mong rằng đây sẽ là một ứng dụng giải quyết 2 vấn đề: tạo thêm cơ hội việc làm cho Shipper, nhu cầu kiếm Shipper của các chủ Shop và giảm thiểu chi phí vận chuyển cho Shipper.

Báo cáo gồm 3 phần :

* Nghiên cứu lập trình ứng dụng trên hệ điều hành iOS gồm các chương: chương 1, chương 2, chương 3, chương 4.
* Ứng dụng tìm đường đi ngắn nhấn cho Shipper và kết nối Shipper và chủ Shop được trình bày ở các chương: chương 5 và chương 6
* Sau đó chương 7 tổng kết ưu nhược điểm, hướng phát triển của đề tài.

**CHƯƠNG 1: GIỚI THIỆU iOS**

1. **Khái niệm iOS**

iOS viết tắt của từ Internetwork Operating System là hệ điều hành trên các thiết bị di động của Apple.

*Ưu điểm:*

* Nền tảng ổn định, ứng dụng có khả năng tương thích cao. Ứng dụng trên iOS có thể hoạt động mượt mà vì được tối ưu hóa cho các thiết bị của Apple.
* Độ tin cậy và bảo mật cao.
* Ứng dụng phong phú, chất lượng và cập nhật nhanh khi có phiên bản mới.

*Nhược điểm:*

* Không thể chạy nhiều ứng dụng cùng một lúc trên màn hình.
* Giá thành cao.

1. **Thiết lập môi trường lập trình iOS**
   1. **Hệ điều hành**

* Để lập trình được ứng dụng chạy trên các thiết bị của iOS, các nhà phát triển đã đưa ra hệ điều hành Mac OS X (trước đây là Mac OS)  là một dòng hệ điều hành độc quyền, được phát triển và phân phối bởi [Apple](https://vi.wikipedia.org/wiki/Apple_Inc.), được cài đặt sẵn trên các máy tính [Macintosh](https://vi.wikipedia.org/wiki/Macintosh). Mac OS X là thế hệ tiếp nối của [Mac OS](https://vi.wikipedia.org/wiki/Mac_OS), hệ điều hành ban đầu của Apple từ năm 1984. Không như Mac OS, Mac OS X là một hệ điều hành [kiểu Unix](https://vi.wikipedia.org/wiki/Unix) được xây dựng trên công nghệ được phát triển tại [NeXT](https://vi.wikipedia.org/w/index.php?title=NeXT&action=edit&redlink=1) trong nửa đầu những năm 1980 và cho đến khi Apple mua công ty này vào đầu năm 1997.

|  |
| --- |
| macOS.PNG |

*Hình 1.1. Hệ điều hành Mac OS*

* Trước năm 2005, hệ điều hành Mac OS X dường như chỉ dành cho các máy tính dùng vi xử lý PowerPC (trong đó có Apple và một vài hãng khác), nhưng giờ đây, với việc chuyển đổi sang sử dụng chip Intel Duo Core, các máy tính PowerPC cũng có thể chạy được hệ điều hành [Windows](https://vi.wikipedia.org/wiki/Microsoft_Windows) và ngược lại, một số máy PC chạy được Mac OS X với bản vá đặc biệt.
  1. **Thiết bị**

|  |  |
| --- | --- |
| iPAd.PNG | iphone.PNG |
| *Hình 1.2. iPad* | *Hình 1.3. IPhone* |

* Các thiết bị làm mẫu chạy ứng dụng như là: các iPhone: 4, 4s, 5, 5s, 6, 6s. Ngoài ra còn có các loại iPAd
  1. **Công cụ hỗ trợ lập trình**
* Apple cung cấp Xcode để lập trình ứng dụng, công cụ này cung cấp 2 ngôn ngữ chính để sử dụng: Swift và Objective C.

|  |
| --- |
| xcode.PNG |

*Hình 1.4. Xcode – Công cụ lập trình*

* 1. **Các thư viện hỗ trợ lập trình**

|  |  |  |
| --- | --- | --- |
| **Tên** | **Tiền tố** | **Mô tả** |
| Acclerate.framework | Cblas, vDSP | Thông tin toán học |
| Accounts.framework | AC | Có giao diện để quản lý, truy cập vào tài khoản của người sử dụng một hệ thống. |
| AdressBook.framework | AB | Chứa các chức năng để truy cập vào CSDL danh bạ của người dùng trực tiếp |
| AdressBookUI.framework | AB | Chứa các lớp để hiển thị hệ thống xác định người dùng và các giao diện trình soạn thảo |
| AssetsLibrary.framework | AL | Chứa các lớp truy cập hình ảnh và video của người dùng |
| AudioToolbox.framework | AU,  Audio | Chứa các giao diện để xử lý các dòng dữ liệu âm thanh |
| AudioUnit.framework | AU,  Audio | Chứa các giao diện tải và đơn vị sử dụng âm thanh |
| AVFoundation.framework | AV | Chứa giao diện để chơi và ghi lại âm thanh, video |
| CFNetwork.framework | CF | Chứa giao diện để truy cập mạng thông qua Wi-fi và radio di động |
| CoreAudio.framework | Audio | Cung cấp các loại dữ liệu được sử dụng trong cả âm thanh Core |
| CoreBlutooth.framework | CB | Cung cấp sự truy cập phần cứng Blutooth năng lượng thấp |
| CoreData.framework | NS | Thông tin giao diện quản lý ứng dụng mô hình dữ liệu |
| CoreFoundation.framework | CF | Cung cấp các dịch vụ phần mềm cơ bản, bao gồm khái niệm trừu tượng cho các loại dữ liệu phổ biến, chuỗi tiện ích, tiện ích thu thập, quản lý tài nguyên |
| CoreImage.framework | CI | Chứa giao diện để thao tác với video và hình ảnh tĩnh |
| CoreLocation.framework | CL | Chứa các giao diện để xác định vị trí người dùng |
| CoreMedia.framework | CM | Chứa giao diện để thao tác với âm thanh và video |
| CoreMotion.framework | CM | Chứa giao diện để truy cập dữ liệu tốc độ |
| CoreTelephony.framework | CT | Chứa giao diện truy cập thông tin liên quan đến điện thoại |
| CoreText.framework | CT | Chứa giao diện thao tác với văn bản và vẽ |
| CoreVideo.framework | CV | Chứa giao diện cho các thao tác âm thanh và video |
| EventKit.framework | EK | Chứa giao diện để truy cập dữ liệu sự kiện lịch của người dùng |
| EventKitUI.framework | EK | Chứa các lớp để hiển thị giao diện hệ thống lịch tiêu chuẩn |
| ExternalAccessory.framework | EA | Chứa giao diện để giao tiếp với các phụ kiện phần cứng |
| Foundation.framework | NS | Chứa giao diện cho các chuỗi quản lý, bộ sưu tập và các loại dữ liệu khác ở mức độ thấp |
| GLKit.framework | GLK | Chứa các lớp Objective-C tiện ích để xây dựng các ứng dụng OpenGL ES phức tạp |
| GSS.framework | gss | Cung cấp một bộ tiêu chuẩn các dịch vụ liên quan đến an ninh |
| iAd.framework | AD | Chứa các giao diện để hiển thị các quảng cáo trong ứng dụng |
| ImageIO.framework | CG | Chứa các giao diện để đọc và ghi dữ liệu hình ảnh |
| IOKit.framework | N/A | Chứa các giao diện được sử dụng bởi thiết bị |
| MapKit.framework | MK | Chứa các giao diện để nhúng một giao diện bản đồ vào ứng dụng và cho tọa độ mã hóa địa lý đảo ngược |
| MediaPlayer.framework | MP | Chứa giao diện để chơi video và xem toàn màn hình |
| MessageUI.framework | MF | Chứa giao diện để viết tin nhắn và email |
| NewsstandKit.framework | NK | Cung cấp các giao diện để tải về nội dung tạp chí và tờ báo ở chế độ nền |
| OpenAL.framework | EAGL,GL | Chứa các giao diện OpenGL ES, là một phiên bản nhúng của các nền tảng OpenGL 2D và 3D, thư viện đồ họa |
| QuartzCore.framework | CA | Chứa giao diện Core Animation |
| QuickLook.framework | QL | Chứa giao diện cho các tập tin xem trước |
| Security.framework | CSSM,Sec | Chứa giao diện cho các chứng chỉ quản lý, bảo mật |
| SytemConfiguration.framework | SC | Chứa giao diện để xác định cấu hình mạng của một thiết bị |
| Twitter.framework | TW | Chứa giao diện để gửi tweets thông qua dịch vụ Twitter |
| UIKir.framework | UI | Chứa các lớp và phương pháp cho các lớp ứng dụng giao diện người sử dụng iOS |

*Bảng 1.1. Thư viện hỗ trợ lập trình*

1. **Tài khoản Apple Developer**

* Hiện nay, Apple cung cấp một số loại tài khoản dành cho các lập trình viên phát triển ứng dụng cho iOS:
* iOS Developer Program Standard 🡪 99$
* iOS Developer Program Enterprise 🡪 299$
* iOS University Program 🡪 Miễn phí (Free)

Đăng ký tài khoản tại: developer.apple.com

|  |
| --- |
| taikhoanDeveloper.PNG |

*Hình 1.5. Các tài khoản của Developer Apple*

1. **Chạy ứng dụng (Build App)**

* Có 2 hình thức chạy ứng dụng:
* Chạy trên Simmulator( Máy ảo trên Mac).
* Chạy trên thiết bị thật (iphone hoặc ipad).

|  |
| --- |
| builtApp.PNG |

*Hình 1.6. Chạy ứng dụng trên máy ảo và thiết bị thật*

* Khi chọn chạy ứng dụng trên máy ảo:

|  |
| --- |
| Snip20160622_4.png |

*Hình 1.7. Giao diện chạy ứng dụng trên máy ảo*

* Chạy ứng dụng trên thiết bị thật:

Khi phát triển xong các ứng dụng, việc tiếp theo đó là triển khai ứng dụng đó trên App Store. Quá trình này được thực hiện thông qua iTune Connect, cổng kết nối các nhà phát triển với thế giới.

**CHƯƠNG 2: GIỚI THIỆU XCODE**

1. **Xcode IDE (Môi trường phát triển tích hợp)**

* Được thiết kế dựa trên các công nghệ mới nhất của Apple. Xcode IDE tích hợp tất cả các công cụ cần thiết và cửa sổ làm việc như: giao diện, soạn mã nguồn, gỡ lỗi.
* Trong quá trình soạn thảo, Xcode kiểm tra giá trị của một biến bằng cách:
* IDE Xcode xác định những lỗi trong cú pháp dễ dàng và dự đoán lỗi chính xác đồng thời tự sửa lỗi.
* IDE (Integrated Development Enviroment) trong Xcode cung cấp nhiều công cụ và tính năng giúp người dùng dễ dàng lập trình hơn. Một trong các tính năng đó là hiển thị một danh sách, các lớp trong thư viện và các thuộc tính, phương pháp của lớp đó:

|  |
| --- |
| Snip20160624_26.png |

*Hình 2.1. Các hỗ trợ của IDE Xcode.*

1. **Cách tạo ứng dụng trên Xcode**

* Bước 1: Khởi động xCode 🡪 “Create a new Xcode project”

|  |
| --- |
| Khởi động xcode |

*Hình 2.2. Giao diện khởi động xCode*

* Bước 2:
* Cây thư mục trái: Chọn **Appliction** của mục iOS.
* Màn hình bên phải: chọn **Single View Application.**
* Nhấn nút next.

|  |
| --- |
| Tạo một giao diện người dùng đơn trên Xcode. |

*Hình 2.3. Chọn loại ứng dụng.*

* Bước 3: Nhập các thông tin:
* Product Name: Tên ứng dụng của người dùng.
* Organization Name: tên cơ quan, tổ chức, hoặc tên lập trình viên.
* Organization Identifier: tên định danh của cơ quan, tổ chức hoặc của lập trình viên
* Language: Chọn ngôn ngữ lập trình là Objective C hoặc Swift.
* Devices: thiết bị mà người dùng sẽ lập trình để ứng dụng chạy. Có thể là iPhone, iPad và Universal
* Use Core Data: Đây là bộ cơ sở dữ liệu cục bộ, nếu người dùng có nhu cầu sử dụng thì check vào.
* Điền đúng thông tin và nhấn nút Next.

|  |
| --- |
| Cấu hình chung cho ứng dụng Xcode |

*Hình 2.4. Thông tin của ứng dụng được tạo.*

* Bước 4: Khi nhấn next sẽ hiện ra giao diện chọn thư mục lưu ứng dụng:

|  |
| --- |
| Lựa chọn nơi lưu ứng dụng của bạn |

*Hình 2.5. Chọn thư mục lưu ứng dụng*

1. **Giao diện chương trình (Interface Builder)**

|  |
| --- |
| http://2.bp.blogspot.com/-hnu8pbNmEgY/VHBXcH3v-2I/AAAAAAAAAAM/kmfxdi1tlkQ/s1600/2014-02-06-16_43_09-tai-lieu-huong-dan-mobile-pc-laptop-pdf-foxit-reader.jpg |

*Hình 2.6. Giao diện chương trình*

* Giao diện Xcode có 5 thành phần chính: Toolbar, Editor area, Navigator area, Debug area, Utility area.
* **Debug area**: vùng hỗ trợ người dùng trong quá trình debug lỗi của chương trình.
* **Toolbar area**: vùng chưa các công cụ tiện ích giúp người dùng đơn giản trong việc chạy, debug ứng dụng, lựa chọn iOS Simulator, đóng mở các vùng khác, …
* **Editor area**: vùng để người dùng thiết kế giao diện, viết và chỉnh sửa code của chương trình.
* **Utility area**: vùng cho phép người dùng tùy chỉnh các tham số, giá trị của các đối tượng trên giao diện, cũng như cho phép người dùng kéo thả và sử dụng các đối tượng có sẵn của Xcode như Button, Label, Slider,… hay các đoạn code mẫu (If, Switch).

|  |  |
| --- | --- |
| Tuỳ chỉnh cho đối tượng | Kéo label từ thanh công cụ bên phải sang giao diện hiển thị |

*Hình 2.7. Utility area*

* **Navigation area**: cung cấp người dùng một cách nhìn trực quan, tiện lợi cho việc quản lý ứng dụng, xem thông báo lỗi, tìm kiếm một đoạn code trong chương trình hay kiểm tra mức độ hoạt động của RAM, CPU khi chạy ứng dụng,…

|  |
| --- |
| Màn hình bên trái ứng dụng |

*Hình 2.8. Navigation area.*

1. **Chạy ứng dụng (Simulator)**

* Để biên dịch và chạy ứng dụng, người dùng sử dụng Simulator. Đây là một giải pháp chạy ứng dụng để kiểm tra khi người dùng chưa có thiết bị thật để chạy.
* IOS Simulator có sẵn các loại thiết bị khác nhau như: iPad, Iphone, phù hợp với các phiên bản màn hình khác nhau.
* Các bước chạy ứng dụng trên Simulator.
* Bước 1: Chọn iPhone 6 từ Scheme pop – up menu trong thanh Xcode toolbar

|  |
| --- |
| Screen Shot 2015-07-23 at 2.26.52 PM |

*Hình 2.9. Mô phỏng ứng dụng*

* Bước 2: Nhấn nút Run hoặc chọn Product > Run (hoặc kích Command +R). Màn hình mô phỏng sẽ hiện ra:

|  |
| --- |
| Screen Shot 2015-07-23 at 2.31.47 PM |

*Hình 2.10. Màn hình mô phỏng.*

* Các tính năng của iPhone Simulator:

|  |
| --- |
| simulator.PNG |

*Hình 2.11. Thanh Toolbar cho màn hình mô phỏng*

* Xcode tự động cài đặt các ứng dụng trên iphone Simulator. Simulator iPhone có thể mô phỏng các phiên bản khác nhau của hệ điều hành iPhone
* Các tính năng có thể thử nghiệm trên Simulator iPhone bao gồm :
* Xoay màn hình : bên trái, bên phải,trên đầu.
* Hỗ trợ cho hành động : Touch and Hold (chạm và giữ), vẽ, tap, …
* Tuy nhiên cũng có 1 số tính năng không có sẵn trên iPhone Simulator :
* Gọi điện thoại
* Gửi và nhận tin nhắn
* Cài đặt các ứng dụng từ App Store
* Máy ảnh
* Microphone

1. **Trình biên dịch (Apple LLVM)**

* Trình biên dịch trong Xcode bao gồm một tập hợp các thư viện được tối ưu hóa, dễ dàng mở rộng.
* Xcode IDE còn có tính năng sửa chữa các lỗi tự động. Ngoài việc báo cáo lỗi, IDE rất thông mình, trong nhiều trường hợp Xcode sẽ không chỉ thông báo một lỗi mà còn trình bày một giải pháp tốt để khắc phục bằng cách nhập chuột vào lỗi để sửa chữa.
* Ví dụ, sửa chữa một biểu tượng sai chính tả hoặc thêm một dấu chấm phẩy còn thiếu.

|  |
| --- |
| Snip20160626_1.png |

*Hình 2.12. Tác dụng của trình biên dịch*

**CHƯƠNG 3: GIỚI THIỆU SWIFT**

1. **Giới thiệu**

* Swift là một ngôn ngữ lập trình hướng đối tượng của Apple dành cho việc phát triển iOS và OS X được giới thiệu bởi Apple tại hội nghĩ WWDC 2014. Swift được mong đợi sẽ tồn tại song song cùng Objective – C, ngôn ngữ lập trình hiện tại dành cho các hệ điều hành của Apple. Swift được thiết kế là một ngôn ngữ có khả năng phòng chống lỗi cao. Swift được biên dịch với trình biên dịch LLVM (Low Level Virtual Machine).

|  |
| --- |
| Ngôn ngữ Swift 2 với nhiều ưu điểm vượt trội |

*Hình 3.1. Swift*

* Swift là ngôn ngữ script hiện đại, mang tới hiệu năng cao hơn Object C tới 35%: Trong số thử nghiệm benchmark, Swift cho hiệu năng nhanh hơn Python và thậm chí là nhanh hơn cả Objective-C. Trên bộ IDE (phần mềm lập trình) **Xcode**của Apple, mã nguồn Swift sẽ được đồ thị hóa theo thời gian thực. Điều này có nghĩa rằng lập trình viên có thể chạy và kiểm lỗi mã nguồn Swift một cách dễ dàng không kém gì Python.
* **Ưu điểm**
* Tiết kiệm thời gian biên dịch: swift là ngôn ngữ “kịch bản (Scripting)” thực thi trực tiếp từng dòng lệnh nên việc kiểm tra kết quả của các lập trình viên trở nên nhanh và dễ dàng hơn rất nhiều khi người dùng đang viết. Người lập trình viên không phải ngồi chờ biên dịch.
* Các dòng code rất ngắn và dễ đọc. Ba dòng code Objective C có thể gói gọn vào một dòng code Swift. Đều này sẽ đẩy nhanh quá trình phát triển ứng dụng và tăng hiệu quả cho quá trình bảo trì và vá lỗi trong tương lai.
* Việc lập trình với Swift cũng trực quan hơn nhờ phương thức sắp xếp hợp lý ở giao diện sử dụng và kiểu hiển thị kết quả theo thời gian thực trong Xcode Runtime. Nghĩa là, trong lúc viết mã, kết quả sẽ ngay lập tức hiện ra để tiện việc chỉnh sửa và hình ảnh cũng có thể xem ngay từ bên trong IDE. Ngoài ra Xcode cũng dựng các kịch bản thực hiện trong các ứng dụng, tính năng này giúp lập trình viên có thể quản lý ứng dụng khỏi bị sự cố tràn bộ nhớ.
* Cuối cùng là Swift tạo ra ứng dụng có sức ép phần cứng thấp hơn Objective – C, vì vậy người dùng sẽ tận hưởng trải nghiệm game ấn tượng hơn trên các thiết bị iOS
  + - **Yếu điểm**
* Swift là ngôn ngữ mới
* Các nhà phát triển ứng dụng iOS sẽ mất một khoảng thời gian đáng kể để làm chủ ngôn ngữ mới của Apple.
* Giảm tốc độ phát triển của hệ sinh thái ứng dụng Apple.
* Một mối lo ngại khác có thể kể đến là các ứng dụng rác. Do swift giúp quá trình phát triển ứng dụng trở nên dễ dàng hơn trước đây, nên số lượng ứng dụng chất lượng kém xuất hiện trên gian hàng App Store có thể sẽ gia tăng. Các cấu trúc, hàm, biến hằng và các control trong Swift.

1. **Cơ bản về lập trình Swift**
   * + ***Hằng số và biến***
2. *Hằng số và biến*

* Hằng và biến được khai báo trước khi sử dụng. Khai báo hằng với từ khóa **let** và biến với từ khóa **var**

**Cú pháp:**

**Khai báo hằng**

Khai báo không cần kiểu

*let <tên hằng> = <giá trị>*

Khai báo có kiểu

*let <tên hằng>:<kiểu dữ liệu> = <giá trị>*

**Khai báo biến**

Khai báo không cần kiểu

*var <tên hằng> = <giá trị>*

Khi báo có kiểu

*var <tên hằng>:<kiểu dữ liệu> = <giá trị>*

Ví dụ

|  |
| --- |
| //Hằng số  let hangso = 10  //Biến  var bien = 20 |

Có thể khai báo nhiều hằng hoặc biến như sau:

|  |
| --- |
| var x = 0.0, y = 0.0, z = 0.0 |

Có thể cung cấp kểu khi khai báo biến hoặc hằng, để rõ rang hơn cho kiểu giá trị

|  |
| --- |
| var welcomeMessage: String |

Sau đó có thể gán bất kỳ giá trị cho biến:

|  |
| --- |
| welcomeMessage = "Hello" |

Có thể khai báo nhiều biến có cùng kiểu như sau:

|  |
| --- |
| var red, green, blue: Double |

1. *Tên hằng và tên biến*

Tên hằng và tên biến có thể chứa hầu hết bất kỳ character( kí tự), bao gồm cả kí tự Unicode.

|  |
| --- |
| let π = 3.14159  let ten = "trinh"  let 🐶🐮 = "dogcow" |

**Lưu ý:**

* Tên hằng và tên biến không thể chứa các kí tự khoảng trắng, ký hiệu toán học, các mũi tên… và không thể bắt đầu bằng một con số
* Khi người dùng khai báo một hằng hoặc một biến có kiểu nhất định, người dùng không thể khia báo lại nữa với cùng tên, hoặc thay đổi kiểu khác.

Người dùng có thể thay đổi giá trị của biến có cùng kiểu.

|  |
| --- |
| var friendlyWelcome = "Hello!"  friendlyWelcome = "Bonjour!"  // friendlyWelcome is now "Bonjour!" |

Với giá trị của hằng thì không thể thay đổi khi đã được khai báo

|  |
| --- |
| let languageName = "Swift"  languageName = "Swift++"  // this is a compile-time error - languageName cannot be changed |

1. *In hằng và biến*

Người dùng có thể print trước kết quả của hằng hoặc biến để xem hoặc kiểm tra.

|  |
| --- |
| print(friendlyWelcome)  // Prints "Bonjour!" |

Ở Swift 1.0, có 2 hàm in là print và println nhưng với Swift 2.0 chỉ còn hàm print. Có thể dùng print để in ra những thông điệp phức tập hơn. Bao gồm những giá trị hiện tại của hằng và biến.

|  |
| --- |
| print("Giá trị hiện tại của friendlyWelcome là \(friendlyWelcome)")  // Prints "Giá trị hiện tại của friendlyWelcome là Bonjour!" |

* + - ***Số nguyên – Integers***

Là kiểu số nguyên. Swift cung cấp Sign và Unsign integer trong mẫu 8, 16, 32 và 64 bit. Những số nguyên theo một qui ước đặt tên tương tự như C, trong đó một số nguyên không dấu 8 – bit có kiểu UInt8, và mộ số nguyến có dấu 32-bit có kiểu Int32. Giống như tất cả các kiểu trong Swift, các kiểu số nguyên có tên viết bằng chữ in hoa.

1. *Giá trị thấp nhất, cao nhất trong Integer*

Có thể kiểm tra giá trị cao nhất của từng kiểu integer với thuộc tính Max, thấp nhất của integer với thuộc tính Min

|  |
| --- |
| let minValue = UInt8.min // giá trị thấp nhất của UInt8 là 0  let maxValue = UInt8.max // giá trị cao nhất của UInt8 là 255 |

1. *Int*

Kiểu Int thay thế hết tất cả kich thước cụ thể và phù hợp với từng nền tảng.

* Trong nền tảng 32 bit, kiểu Int có kích thước Int32.
* Trong nền tảng 64 bit, kiểu Int có kích thước Int64.
* Miền giá trị của Int trong nền tảng 32-bit: -2147483648 đến 147438647.

1. *UInt*

* Swift cũng cung cấp một loại số nguyên không dấu, uint, trong đó có kích thước tương tự như bản kích thước từ các nền tảng hiện tại:
* Trên nền tảng 32-bit, UINT có kích thước giống như UInt 32
* Trên một nền tảng 64-bit, Uint có kích thước giống như UInt64.

*Chú ý:* sử dụng UINT chỉ khi người dùng đặc biệt cần một loại số nguyên không dấu với kích thước tương tự như bản kích thước của nền tảng. Nếu không phải trường hợp trên, thì Int được ưa thích hơn.

* + - ***Kiểu Float và kiểu Double***

Là một dạng phân số, Có thể lưu trữ những số thập phân mà kiểu integer không lưu trữ được. Swift cung cấp 2 kiểu là:

* Float (32-bit).
* Double(64-bit).

|  |
| --- |
| //Giá trị lưu trữ lớn nhất của kiểu float  var float: Float = FLT\_MAX  //Kết quả: 3.402823e+38  //Giá trị lưu trữ lớn nhất của kiểu double  var double:Double = DBL\_MAX  //Kết quả: 1.797693134862316e+308 |

* + - ***An toàn kiểu và kiểu suy luận - Kiểu Safe và Kiểu Inference***
* Swift là một ngôn ngữ lập trình an toàn. Vì thế Swift muốn người dùng phải khai thác rõ ràng về các kiểu giá trị. Nếu người dùng khai báo kiểu String thì không thể truyền giá trị kiểu Int hoặc ngược lại. Bởi vì Swift thực hiện kiểm tra kiểu (type check) khi biên dịch và báo hiệu những kiểu không phù hợp và lỗi. Điều này cho phép người dùng nắm bắt và sửa lỗi càng sớm càng tốt.
* Kiểm tra kiểu giúp người dùng tránh được lỗi khi họ đang làm việc với các kiểu giá trị khác nhau. Tuy nhiên, không có nghĩa là người dùng phải xác định kiểu của mỗi hằng và biến mà người dùng khai báo. Nếu người dùng không chỉ định các kiểu giá trị, Swift sẽ sử dụng kiểu Inference để tìm ra kiểu thích hợp dựa trên giá trị mà người dùng đã cung cấp.
* Kiểu Inference đặt biệt hữu ích khi người dùng khai báo một hằng hoặc một biết có giá trị banđầu:

|  |
| --- |
| let meaningOfLife = 42  // meaningOfLife được hiểu là kiểu Int  let pi = 3.14159  // pi được hiểu là kiểu Double Double  let anotherPi = 3 + 0.14159  // anotherPi Được hiểu là kiểu Double |

* + - ***Chuyển đổi kiểu số (Numeric Type Conversion)***

1. *Chuyển đổi số nguyên (Integer Conversion)*

Ở ví dụ dưới hằng so1 và hằng so2 có 2 kiểu số nguyên khác nhau, vì thế ta phải ép kiểu của hằng so1 sang cùng kiểu với hằng so2 là kiểu Int16

|  |
| --- |
| let so2: UInt16 = 2\_000  let so1: UInt8 = 1  let tong = so2 + UInt16(one) |

1. *Chuyển đổi kiểu Int với kiểu Float Double*

Ở ví dụ dưới hằng so3 có kiểu Int và hằng so4 có kiểu Double, vì thế ta phải ưu tiên ép kiểu Int sang Double.

|  |
| --- |
| let so3 = 3  let so4 = 0.14159  let pi = Double(so3) + so4  // pi có giá trị là 3.14159, và được hiểu là kiểu Double |

* + - ***Kiểu bí danh - Kiểu Aliases***

Thay thế một kiểu hiện có. Được khai báo với từ khóa typealias

|  |
| --- |
| typealias alias = UInt32  var so5:alias = 5  // so5 có giá trị là 5 |

Ta có thể thấy kiểu alias có tên *alias* được nhận kiểu UInt32 và biến *so5* có kiểu *alias*. Thực chất biến *so5* nhận kiểu UInt32 thông qua *alias*. Từ ví dụ trên ta thấy kiểu *alias* hữu ích khi muốn đề cập đến một kiểu đã tồn tại với tên khác.

* + - ***Kiểu Boolean***

Đây là kiểu trả về 2 giá trị True và False

|  |
| --- |
| let dung = true  let sai = false |

Thường được sử dụng nhiều trong trường hợp so sánh, câu lệnh if

|  |
| --- |
| if sai {  print("Giá trị true!")  } else {  print(“Giá trị false")  }  // Prints "Giá trị false" |

Trong trường hợp người dùng dùng câu lệnh if để so sánh giá trị kiểu số:

|  |
| --- |
| let i = 1  if i {  // trường hợp này chương trình sẽ không chạy và báo lỗi  }  if i == 1 {  // trường hợp đúng  } |

* + - ***Kiểu Tuple***

Đây là một kiểu mới trong Swift. Kiểu Tuple có thể nhóm các giá trị và các giá trị đó không nhất thiết phải cùng kiểu

|  |
| --- |
| let http404Error = (404, "Not Found")  // http404Error is of type (Int, String), and equals (404, "Not Found") |

Người dùng có thể nhóm nhiều giá trị khác kiểu lại với nhau, Và ta có thể lấy từng giá trị ra như sau:

|  |
| --- |
| let (statusCode, statusMessage) = http404Error  print("The status code is \(statusCode)")  // Prints "The status code is 404"  print("The status message is \(statusMessage)")  // Prints "The status message is Not Found" |

Ngoài ra để truy cập các phần tử trong một tuple có thể sử dụng chỉ số như sau

|  |
| --- |
| print("The status code is \(http404Error.0)")  // Prints "The status code is 404"  print("The status message is \(http404Error.1)")  // Prints "The status message is statusMessage " |

Người dùng cũng có thể khai báo như sau:

|  |
| --- |
| let http200Status = (statusCode: 200, description: "OK")  print("The status code is \(http200Status.statusCode)")  // Prints "The status code is 200"  print("The status message is \(http200Status.description)")  // Prints "The status message is OK" |

* + - ***Optional***

Người dùng có thể sử dụng Optional trong tình huống có giá trị hoặc không có giá trị.

|  |
| --- |
| var serverResponseCode: Int? = 404  // serverResponseCode có giá trị kiểu int là 404  serverResponseCode = nil  // serverResponseCode không có giá trị |

Nếu người dùng định nghĩa một hằng số hoặc biến optional mà không cung cần cung cấp một giá trị mặc định, hằng số hoặc biến được tự động thiết lập để nil cho người dùng:

|  |
| --- |
| var surveyAnswer: String?  // surveyAnswer tự động có giá trị là nil |

* + - ***Error Handling***

Error Handling mới được apple nâng cấp vào Swift 2.0. Người dùng có thể sử dụng Error Handling để xử lý những lỗi mà người dùng có thể gặp phải. Khi một func gặp phải một tình trạng lỗi nó sẽ ném ra lỗi và người dùng có thể bắt lỗi và xử lý thích hợp.

|  |
| --- |
| do {  try canThrowAnError()  // thực hiện khi không có lỗi  } catch {  // thực hiện khi có lỗi  } |

1. **Toán tử cơ bản**

Toán tử là một ký hiện đặc biệt hoặc cụm từ mà người dùng sử dụng để kiểm tra, thay đổi, hoặc kết hợp các giá trị. Ví dụ, các toán tử cộng (+) thêm hai số với nhau (ví dụ let I = 1 + 2). Ví dụ phức tạp hơn bao gồm toán tử logic AND && (ví dụ: if enteredDoorCode @@ passedRetinaScan) và toán tử logic tăng dần ++ i, đó là một lệnh tắt để tăng giá trị của i lên 1.

Swift hỗ trợ chuẩn toán tử C và cải thiện một số tính năng để loại bỏ các lỗi mã hóa phổ biến. Toán tử gán (=) không trả về một giá trị, để ngăn chặn toán tử gán này được sử dụng lẫn lộn với toán tử so sánh (==). Toán tử số học (+, -, \*, /, % và vv) phát hiện và không cho phép tràn giá trị, để tránh những kết quả bất ngờ khi làm việc với các con số lớn hơn hay nhỏ hơn so với phạm vi.

* + - ***Terminology***
* Hầu hết các toán tử một ngôi, hai ngôi, ba ngôi:
* Toán tử một ngôi hoạt động trên một phần tử dữ liệu duy nhất (như -a). Các tiền tố toán tử một ngôi xuất hiện ngay trước biến của chúng(như !b) và các hậu tố toán tử một ngôi xuất hiện ngay sau biến của chúng (như i++).
* Toán tử 2 ngôi hoạt động trên 2 phần tử dữ liệu (như 2 + 3) và trung tố vì chúng xuất hiện ở giữa hai phần tử dữ liệu của chúng.
* Toán tử 3 ngôi hoạt động trên ba phần tử dữ liệu. Giống như C, Swift chỉ có duy nhất một toán tử 3 ngôi, các điều kiện của toán tử ba ngôi (a?b:c).
* Các giá trị mà toán tử ảnh hưởng là các toán hạng. Trong biểu thức 1 + 2, biểu tượng + là một toán tử nhị phân và hai toán hạng là các giá trị 1 và 2.
  + - ***Arithmetic Operators(Toán tử toán học)***

Toán tử gán (a = b) khởi tạo hay cập nhật giá trị của a với giá trị của b

|  |
| --- |
| let b = 10  var a = 5  a = b  // kết quả a = 10,b = 10 |

Nếu phía bên phải của phép gán là một bộ với nhiều giá trị, các yếu tố của phép gán có thể được phân tách ra thành nhiều hằng hoặc biến cùng một lúc.

* + - ***Toán tử số dư (Remainder)***

Swift hỗ trợ bốn toán tử toán học tiêu chuẩn cho tất cả các kiểu số:

* Phép cộng (+)

|  |
| --- |
| let (x, y) = (1, 2)  // x is equal to 1, and y is equal to 2 |

* Phép trừ (-)
* Phép nhân (\*)
* Phép chia (/)

|  |
| --- |
| 1 + 2 // = 3  5 - 3 // = 2  2 \* 3 // = 6  10.0 / 2.5 // = 4.0 |

* Toán tử cộng còn hỗ trợ nối chuỗi String

|  |
| --- |
| "hello, " + "world" // kết quả: "hello, world" |

* + - ***Toán tử số dư thập phân (Floating – Point Remainder Calculations)***

Toán tử số dư (a%b) tính toán ra bao nhiêu bộ số của b sẽ phù hợp với bên a và trả về giá trị được để lại (gọi là phần còn lại)

Ví dụ các toán tử số dư hoạt động: 9 % 4

|  |
| --- |
| image: ../Art/remainderInteger_2x.png  9 % 4 // kết quả 1 |

* + - ***Toán tử tăng và giảm dần (Increment and Decrement Operators)***
* Toán tử tăng dần (++) có nghĩa là i = i + 1
* Toán tử giảm dần (--) có nghĩa là i = i – 1

|  |
| --- |
| var a1 = 0  let b1 = ++a1 //b1=1 và a1=1  let c1 = a1++ //c1=1 và a1=2 |

Trong ví dụ trên, *let b1 = ++a1* tăng giá trị của a1 trước khi trả về giá trị của chính nó. Đây là lý do tại sao cả a1 và b1 đều có giá trị mới là 1

Tuy nhiên, *let c1 = a1++* tăng giá trị của a1 sau khi trả về giá trị của c1. Đều này có nghĩa là c1 lấy giá trị cũ là 1 và sau đó a1 được cập nhật giá trị bằng 2.

* + - ***Toán tử gán phức hợp – Compound Assignment Operators.***

Giống như C, Swift cung cấp các toán tử gán phức hợp kết hợp với phép gán (=) với các toán tử khác. Ví dụ như +=

|  |
| --- |
| var a = 1  a += 2  // a có kết quả là 3 |

Biểu thức a += 2 viết tắt cho a = a + 2. Hiệu quả cho việc bổ sung và gán được kết hợp thành một toán tử mà thực hiện cả hai tác vụ cùng một lúc.

* + - ***Toán tử so sánh (Comparison Operators)***

Swift hỗ trợ tất cả chuẩn toán tử so sánh trong C:

* Bằng (a==b)
* Không bằng (a!=b)
* Lớn hơn (a>b)
* Nhỏ hơn (a<b)
* Lớn hơn hoặc bằng (a >= b)
* Nhỏ hơn hoặc bằng (a <= b)

Mỗi toán tử so sánh trả về một giá trị Bool để cho biết hay không câu lệnh là true

|  |
| --- |
| 1 == 1 // true because 1 is equal to 1  2 != 1 // true because 2 is not equal to 1  2 > 1 // true because 2 is greater than 1  1 < 2 // true because 1 is less than 2  1 >= 1 // true because 1 is greater than or equal to 1  2 <= 1 // false because 2 is not less than or equal to 1 |

Toán tử so sánh thường được sử dụng trong câu lệnh if.

* + - ***Toán tử điều khiển tam phân (Ternary Conditional Operator)***

Toán tử điều kiện tam phân là một toán tử đặc biệt với ba phần, trong đó có dạng question? Answer: answer2. Đây là một phím tắt để đánh giá một trong hai biểu thức dựa trên một trong hai giá trị question là true hay fasle.

*if question {*

*answer1*

*} else {*

*answer2*

*}*

Ví dụ dưới đây: “(Bool ? 40 : 10)” có nghĩ là nếu Bool là true thì lấy 40 ngược lại Bool là false lấy 10. Ở đây Bool bằng false nên kết quả là 30

|  |
| --- |
| let tao = 20  let Bool = false  let soluong = tao + (Bool ? 40 : 10)  // soluong = 30 |

* + - ***Toán tử kết hợp nil (Nil Coalescing Operator)***

|  |
| --- |
| let red = "red"  var chuoi:String? //mặc định là nil  var Mau = chuoi ?? red //nếu chuoi bằng nil thì xuất ra red  //Kết quả: "red" |

* + - ***Toán tử phạm vi (Range Operators)***

Swift bao gồm 2 loại toán tử là: Closed Range Operator và Half-Open Eange Onperator

1. *Toán tử phạm vi kín (Closed Range Operator)*

Phạm vi toán tử khép kín (a…b) định nghĩa một phạm vi đó chạy từ a đến b, vào bao gồm các giá trị a và b. Giá trị của a không lớn hơn b.

Toán tử phạm vi khép kín hữu ích khi lặp qua một dãy mà người dùng muốn tất cả các giá trị được sử dụng, chẳng hạn như với một vòng lặp for – in:

|  |
| --- |
| for index in 1...5 {  print("\(index) times 5 is \(index \* 5)")  }  // 1 times 5 is 5  // 2 times 5 is 10  // 3 times 5 is 15  // 4 times 5 is 20  // 5 times 5 is 25 |

1. *Toán tử bán phạm vi (Half-Open Range Operator)*

Toán tử bán phạm vi ( a…<b) định nghĩa một phạm vi chạy từ a đến b, nhưng không bao gồm b. Toán tử bán phạm vi được cho là mở một nửa (half-open) bởi vì nó có chứa các giá trị đầu tiên, nhưng không có giá trị cuối cùng. Giống như với toán tử phạm vi khép kín, giá trị của a không được lớn hơn b.

Toán tử bán phạm vi đặc biết hữu ích khi người dùng làm việc với danh sách dạng zero – based như mạng, nơi mà nos thuận tiện để đếm độ dài của danh sách:

|  |
| --- |
| let names = ["Anna", "Alex", "Brian", "Jack"]  let count = names.count  for i in 0..<count {  print("Person \(i + 1) is called \(names[i])")  }  // Person 1 is called Anna  // Person 2 is called Alex  // Person 3 is called Brian  // Person 4 is called Jack |

* + - ***Toán tử logic AND (Logical AND Operator)***

Toán tử logic AND (a && b) tạo ra các biểu thức logic mà cả hai giá trị phải true cho kết quả cộng chung cũng là true.

Nếu một trong hai giá trị là false, kết quả cộng chung cũng sẽ là false. Trong thực tế, nếu giá trị đầu tiên là false, giá trị thứ 2 thậm chí sẽ không được đánh giá, bởi vì nó không làm thay đổi được việc cộng chung tương đương với true. Điều này được gọi là đánh giá ngắn mạch – short – circuit evaluation.

|  |
| --- |
| let enteredDoorCode = true  let passedRetinaScan = false  if enteredDoorCode && passedRetinaScan {  print("Welcome!")  } else {  print("ACCESS DENIED")  }  // Prints "ACCESS DENIED" |

* + - ***Toán tử logic OR (Logical OR Operator)***

Toán tử logic OR (a || b) là một toán tử được viết bở hai dấu gạch liền kề. Người dùng sử dụng nó để tạo ra các biểu thức logic trong đó chỉ có một trong hai giá trị là true thì những kết quả cộng chung là true.

Giống như toán tử logic AND, toán tử logic OR sử dụng đánh giá ngắn mạch để xem xét biểu thức của nó. Nếu phía bên trái của một biểu thức logic OR là true, bên phải không được đánh giá, bởi vì nó không thể thay đổi kết quả của biểu thức tổng thể.

|  |
| --- |
| let hasDoorKey = false  let knowsOverridePassword = true  if hasDoorKey || knowsOverridePassword {  print("Welcome!")  } else {  print("ACCESS DENIED")  }  // Prints "Welcome!" |

1. **Kiểu String và kiểu Character**

Một chuỗi là một tập hợp có sắp xếp của những ký tự, như là “hello word” hay “albatross”. Chuỗi trong Swift biểu diễn bằng kiểu String, chúng lần lượt biểu diễn một tập hợp các giá trị của kiểu ký tự (Character).

Kiểu String và Character của Swift cung cấp nhanh, cách phù hợp với mã thống nhất (Unicode) để làm việc với văn bản trong mã code của người dùng. Cú pháp tạo ra các chuỗi và thao tác trên đây đỡ nặng nề hơn và dễ đọc hơn, với cú pháp chuỗi ký tự cũng tương tự như C. Nối chuỗi cũng đơn giản như cách cộng hai chuỗi với nhau với toán tử +. Và tính biến đổi chuỗi được quản lý bằng cách chọn giữa một hằng hoặc một biến, giống như bất kỳ giá trị nào khác trong Swift.

* + - ***String Literal (Chuỗi ký tự)***

Chuỗi ký tự là một chuỗi cố định của văn bản bao quanh bở một cặp dấu ngoặc kép (“”)

|  |
| --- |
| let someString = "Kiều Trinh" |

* + - ***Khởi tạo chuỗi***

Cách kiểm tra chuỗi rỗng:

|  |
| --- |
| var emptyString = "" // có giá trị “”  var anotherEmptyString = String() // khởi tạo chuỗi có giá trị “”  // these two strings are both empty, and are equivalent to each other  if emptyString.isEmpty {  print("Nothing to see here")  }  // Prints "Nothing to see here" |

* + - ***Biến đổi chuỗi (String Mutability)***

Người dùng có thể thay đổi chuỗi cho một biến có kiểu String bằng cách gán vào cho biến đó một giá trị kiểu String.

|  |
| --- |
| var variableString = "Horse"  variableString += " and carriage"  // variableString có giá trị là "Horse and carriage"  let constantString = "Highlander"  constantString += " and another Highlander"  // chương trình sẽ báo lỗi vì khai báo biến là constantString là hằng số |

* + - ***Ký tự (Character)***

Là một ký tự trong một chuỗi.

|  |
| --- |
| for character in "Dog@1".characters {  print(character)  }  // D  // o  // g  //@  //1 |

* + - ***Interpolation (chuỗi nội suy)***

Là một cách xây dựng chuỗi String mới từ hỗn hợp các hằng, biến,…

|  |
| --- |
| let multiplier = 3  let message = "\(multiplier) times 2.5 is \(Double(multiplier) \* 2.5)"  // message is "3 times 2.5 is 7.5" |

* + - ***Counting Character***

Để lấy số lượng giá trị trong một chuỗi hàm sau:

|  |
| --- |
| let unusualMenagerie = "Koala 🐨, Snail 🐌, Penguin 🐧, Dromedary 🐪"  print("unusualMenagerie has \(unusualMenagerie.characters.count) characters")  // Prints "unusualMenagerie has 40 characters" |

Để so sánh 2 chuỗi với nhau, ta dùng toán tử bằng (==), toán tử không bằng (!=)

|  |
| --- |
| let quotation = "swift is a language"  let sameQuotation = "swift is a language"  if quotation == sameQuotation {  print("These two strings are considered equal")  }  // Prints "These two strings are considered equal" |

1. **Các kiểu tập hợp (Collection Type)**

Swift cung cấp cho chúng ta 3 kiểu Collection: Array, Set, Dictionary

* Array lưu trữ số thứ tự của các giá trị có cùng kiểu
* Set thứ tự các giá trị duy nhất
* Dictionary lưu trữ các giá trị cùng kiểu có thứ tự, truy xuất thông qua một định danh duy nhất.

|  |
| --- |
| image: ../Art/CollectionTypes_intro_2x.png |

* + - ***Mảng (Array)***

Lưu trữ các giá trị cùng kiểu theo một danh sách có thứ tự. Các giá trị có thể truy xuất theo vị trí trong Array.

Tạo một mảng rỗng

|  |
| --- |
| var mang = [Int]()  print("someInts is of type [Int] with \(someInts.count) items.")  // Prints "someInts is of type [Int] with 0 items." |

Thao tác gán giá trị cho mãng

|  |
| --- |
| mang.append(3)  // mang chứa một giá trị là 3  mang = []  // mang bây giờ là mãng rỗng nhưng vần được mặc định làm mảng kiểu int |

Các thao tác trên mãng

|  |
| --- |
| //Cách thêm phần tử cho mang  mang.append(4)  //Kết quả: [1,2,3,4]  //Lấy giá trị  print("Số: \(mang[1])")  //Kết quả: "Số: 2"  //Xoá vị trí phần tử trong mang  mang.removeAtIndex(2)  //Vị trí 2 là số 3  print(mang)  //Kết quả: [1,2,4]  //Thêm một phần tử vào mang tại vị trí nhất định mang.insert(5, atIndex: 2)  //Kết quả: [1,2,5,4] |

Tạo một mãng bằng cách khai báo số phần tử và giá trị mặc định

|  |
| --- |
| var anotherThreeDoubles = [Double](count: 3, repeatedValue: 2.5)  // anotherThreeDoubles is of type [Double], and equals [2.5, 2.5, 2.5]  var threeDoubles = [Double](count: 3, repeatedValue: 0.0)  // threeDoubles is of type [Double], and equals [0.0, 0.0, 0.0]  var sixDoubles = threeDoubles + anotherThreeDoubles  // sixDoubles is inferred as [Double], and equals [0.0, 0.0, 0.0, 2.5, 2.5, 2.5] |

*Iterating Over an Array(Duyệt qua một mãng)*

|  |
| --- |
| var shoppingList = ["Eggs", "Milk"]  for item in shoppingList {  print(item)  }  // Eggs  // Milk |

Cách lấy vị trí và giá trị các phần tử trong mãng

|  |
| --- |
| for (index, value) in shoppingList.enumerate() {  print("Item \(index + 1): \(value)")  }  // Item 1: Eggs  // Item 2: Milk |

* + - ***Set***

Set là một kiểu Collection mới có trong Swift 2.0.

Set lưu trữ những giá trị khác nhau có cùng kiểu nhưng không có thứ tự rõ rang. Người dùng có thể sử dụng để thay thế Array khi thứ tự của các giá trị là không quan trọng, hoặc khi người dùng muốn các giá trị chỉ xuất hiện một lần.

1. *Thao tác cơ bản*

Khởi tạo một set rỗng

|  |
| --- |
| var letters = Set<Character>()  print("letters is of type Set<Character> with \(letters.count) items.")  // Prints "letters is of type Set<Character> with 0 items." |

Thêm giá trị vào cho set hoặc gán giá trị rỗng lại

|  |
| --- |
| letters.insert("a")  // letters chứ 1 ký tự a  letters = []  // letters khởi tạo rỗng lại và vẫn là kiểu Character |

Cách khởi tạo khác và kiểm tra set có rỗng hay không

|  |
| --- |
| var favoriteGenres: Set<String> = ["Rock", "Classical", "Hip hop"]  // favoriteGenres được khởi tạo ban đầu với 3 giá trị  if favoriteGenres.isEmpty {  print("Không có")  } else {  print("Có tồn tại")  }  // Prints "Có tồn tại" |

Xóa phần tử ra khỏi set dùng từ khóa “remove”

|  |
| --- |
| if let removedGenre = favoriteGenres.remove("Rock") {  print("\(removedGenre)? I'm over it.")  } else {  print("I never much cared for that.")  }  // Prints "Rock? I'm over it." |

Kiểm tra tồn tại của một phần tử nào đó trong set

|  |
| --- |
| if favoriteGenres.contains("Funk") {  print("I get up on the good foot.")  } else {  print("It's too funky in here.")  }  // Prints "It's too funky in here." |

1. *Biểu diễn hoạt đông của Set (Performing Set Operation)*

Người dùng có thể sử dụng Set theo các toán tử cơ bản như kết hợp hai Set, xác định những giá trị giống trong 2 Set…

|  |
| --- |
| image: ../Art/setVennDiagram_2x.png |

*Hình 3.3. Các phép toán giữa 2 set*

* Sử dụng *a.intersect(b)*: Giao giữa hai tập tập hợp a và b
* Sử dụng *a.exclusiverOr(b):* Bỏ đi phần giống nhau giữa hai set a và b.
* Sử dụng *a.uinion(b):* Lấy tất cả các phần tử của set a và set b.
* Sử dụng *a.subtract(b)*:Set a sẽ xóa đi những phần tử của set b mà chứa trong a. (Lấy những phần tử chỉ thuộc set a.)

Ví dụ:

|  |
| --- |
| let oddDigits: Set = [1, 3, 5, 7, 9]  let evenDigits: Set = [0, 2, 4, 6, 8]  let singleDigitPrimeNumbers: Set = [2, 3, 5, 7]  oddDigits.union(evenDigits).sort()  // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  oddDigits.intersect(evenDigits).sort()  // []  oddDigits.subtract(singleDigitPrimeNumbers).sort()  // [1, 9]  oddDigits.exclusiveOr(singleDigitPrimeNumbers).sort()  // [1, 2, 9] |

1. *Các tập Set giao và thuộc nhau (Set Membership và Equality)*

|  |
| --- |
| image: ../Art/setEulerDiagram_2x.png |

*Hình 3.4. Mối quan hệ giữa hai set*

Hình trên cho ta thấy có 2 Set a, b, c chồng chéo lên nhau. Set b nằm trong set a và Set a giao nhau với Set c. Ta sẽ có những mối quan hệ như sau

Sử dụng toán tử “==” để kiểm tra s Set có giống nhau hay không

* *b.isSubsetOf(a)* : Kiểm tra b có nằm trong a không.
* *a.isSupersetOf(a)*: Kiểm tra có bao quát b không.
* *b.isDisjointWith(c):*Kiểm tra b có giao với c không.

Ví dụ:

|  |
| --- |
| let houseAnimals: Set = ["🐶", "🐱"]  let farmAnimals: Set = ["🐮", "🐔", "🐑", "🐶", "🐱"]  let cityAnimals: Set = ["🐦", "🐭"]  houseAnimals.isSubsetOf(farmAnimals)  // true  farmAnimals.isSupersetOf(houseAnimals)  // true  farmAnimals.isDisjointWith(cityAnimals)  // true |

* + - ***Dictionary***

Lưu trữ nhiều giá trị cùng kiểu, giá trị của các kiểu giống như Array nhưng không có thứ tự nhất định. Mỗi giá trị được liên kết thông qua một key duy nhất. Người dùng sử dụng một từ điển khi người dùng cần tìm ra giá trị dựa trên nhận dạng của chúng, theo cách giống như một từ điển thực tế được sử dụng để tìm định nghĩa cho một từ cụ thể.

Khai báo dictionary

|  |
| --- |
| var namesOfIntegers = [Int: String]()  // namesOfIntegers là một dictionary rỗng có các phần tử kiểu:[Int: String]  namesOfIntegers[16] = "sixteen"  // namesOfIntegers chứa 1 cặp key – value 16: “sixteen”  namesOfIntegers = [:]  // khởi tạo lại namesOfIntegers có kiểu [Int: String]  // các khởi tạo khác  var airports: [String: String] = ["YYZ": "Toronto Pearson", "DUB": "Dublin"]  var airports = ["YYZ": "Toronto Pearson", "DUB": "Dublin"] |

Đếm số giá trị dictionary

|  |
| --- |
| print("The airports dictionary contains \(airports.count) items.")  // Prints "The airports dictionary contains 2 items." |

Kiểm tra rỗng trong Dictionary

|  |
| --- |
| if airports.isEmpty {  print("The airports dictionary is empty.")  } else {  print("The airports dictionary is not empty.")  }  // Prints "The airports dictionary is not empty." |

Cách thêm và cập nhật 1 cặp key – value vào trong dictionary

|  |
| --- |
| airports["LHR"] = "London"  // the airports dictionary now contains 3 items  airports["LHR"] = "London Heathrow"  // the value for "LHR" has been changed to "London Heathrow"  if let oldValue = airports.updateValue("Dublin Airport", forKey: "DUB") {  print("The old value for DUB was \(oldValue).")  }  // Prints "The old value for DUB was Dublin." |

Cách xóa phần tử trong dictionary

|  |
| --- |
| // APL has now been removed from the dictionary  if let removedValue = airports.removeValueForKey("DUB") {  print("The removed airport's name is \(removedValue).")  } else {  print("The airports dictionary does not contain a value for DUB.")  }  // Prints "The removed airport's name is Dublin Airport." |

Truy xuất phần tử trong dictionary

Truy xuất từng cặp (key, value)

|  |
| --- |
| for (airportCode, airportName) in airports {  print("\(airportCode): \(airportName)")  }  // YYZ: Toronto Pearson  // LHR: London Heathrow |

Có thể truy câp riêng lẽ key, value

|  |
| --- |
| for airportCode in airports.keys {  print("Airport code: \(airportCode)")  }  // Airport code: YYZ  // Airport code: LHR  for airportName in airports.values {  print("Airport name: \(airportName)")  }  // Airport name: Toronto Pearson  // Airport name: London Heathrow |

1. **Vòng lặp (Control Flow)**
   * + ***For in***

Người dùng có thể dử dụng vòng lặp để duyệt các phần tử, các phần tử trong mãng hay các ký tự trong chuỗi, …

Ví dụ vòng lặp từ 1 tới 5:

|  |
| --- |
| for index in 1...5 {  print("\(index) times 5 is \(index \* 5)")  }  // 1 times 5 is 5  // 2 times 5 is 10  // 3 times 5 is 15  // 4 times 5 is 20  // 5 times 5 is 25 |

Vòng lặp duyệt qua các phần tử trong mãng:

|  |
| --- |
| let names = ["Anna", "Alex", "Brian", "Jack"]  for name in names {  print("Hello, \(name)!")  }  // Hello, Anna!  // Hello, Alex!  // Hello, Brian!  // Hello, Jack! |

Vòng lặp duyệt qua các ký tự đặc biệt trong dictionary:

|  |
| --- |
| let numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]  for (animalName, legCount) in numberOfLegs {  print("\(animalName)s have \(legCount) legs")  }  // ants have 6 legs  // cats have 4 legs  // spiders have 8 legs |

* + - ***While loops***

1. *While*

**Cú pháp:**

while *condition* {

*statements*

}

|  |
| --- |
| var so = 0  var ketqua = 6  while so < ketqua {  // roll the dice  so += 1  print(so)  }  //Kết quả: 1  // Kết quả: 2  //Kết quả : 3  //Kết quả : 4  //Kết quả : 5  //Kết quả : 6 |

1. *Repeat-While*

Repeat-While là vòng lặp sẽ hoạt động khi điều kiện While vẫn còn thỏa

Cú pháp

repeat {

*statements*

} while *condition*

Ví dụ

|  |
| --- |
| var a = 0  repeat {  a += 1  print(a)  } while a < 4  //Kết quả: 1  //Kết quả: 2  //Kết quả : 3  //Kết quả : 4 |

1. *Câu lệnh If*

Câu lệnh điều kiện đơn giản. Khi điều kiện đúng nó sẽ thực hiện một tập lệnh này, khi điều kiện sai thì thực hiện một tập lệnh khác

Cú pháp:

if <điều kiện 1>{

//các câu lệnh

}

Câu lệnh if có từ khóa else

if <điều kiện> {

//các câu lệnh

}

else{

//các câu lệnh

}

Ví dụ:

Câu lệnh if đơn

|  |
| --- |
| var temperatureInFahrenheit = 30  if temperatureInFahrenheit <= 32 {  print("It's very cold. Consider wearing a scarf.")  }  // Prints "It's very cold. Consider wearing a scarf." |

Câu lệnh if có else

|  |
| --- |
| temperatureInFahrenheit = 40  if temperatureInFahrenheit <= 32 {  print("It's very cold. Consider wearing a scarf.")  } else {  print("It's not that cold. Wear a t-shirt.")  }  // Prints "It's not that cold. Wear a t-shirt." |

Câu lệnh if, else lồng nhau:

|  |
| --- |
| temperatureInFahrenheit = 90  if temperatureInFahrenheit <= 32 {  print("It's very cold. Consider wearing a scarf.")  } else if temperatureInFahrenheit >= 86 {  print("It's really warm. Don't forget to wear sunscreen.")  } else { print("It's not that cold. Wear a t-shirt.")}  // Prints "It's really warm. Don't forget to wear sunscreen." |

1. *Switch*

Một lệnh switch xem xét giá trị và so sánh nó với một số mô hình phù hợp có thể. Sau đó nó thực hiện một khối mã code thích hợp. Một lệnh switch cung cấp một thay thế cho lệnh if để hồi đáp cho nhiều trạng thái tiềm năng. Mỗi Switch gồm nhiều Case.

Cú pháp:

switch *some value to consider* {

case *value 1*:

*respond to value 1*

case *value 2*,

*value 3*:

*respond to value 2 or 3*

default:

*otherwise, do something else*

}

Ví dụ:

|  |
| --- |
| let someCharacter: Character = "e"  switch someCharacter {  case "a", "e", "i", "o", "u":  print("\(someCharacter) is a vowel")  case "b", "c", "d", "f", "g", "h", "j", "k", "l", "m",  "n", "p", "q", "r", "s", "t", "v", "w", "x", "y", "z":  print("\(someCharacter) is a consonant")  default:  print("\(someCharacter) is not a vowel or a consonant")  }  // Prints "e is a vowel" |

1. *Where*

Một switch case có thể dùng where để kiểm tra điều kiện bổ sung:

|  |
| --- |
| let yetAnotherPoint = (1, -1)  switch yetAnotherPoint {  case let (x, y) where x == y:  print("(\(x), \(y)) is on the line x == y")  case let (x, y) where x == -y:  print("(\(x), \(y)) is on the line x == -y")  case let (x, y):  print("(\(x), \(y)) is just some arbitrary point")  }  // Prints "(1, -1) is on the line x == -y" |

* + - ***Control transfer Statements***

Gồm có: continue, break, fallthrough, return, throw.

1. *Continue*

Câu lệnh continue nói với một vòng lặp dừng hành động nó đang thực hiện và bắt đầu lại tại điểm bắt đầu của việc duyệt qua vòng lặp tiếp theo.

|  |
| --- |
| let puzzleInput = "great minds think alike"  var puzzleOutput = ""  for character in puzzleInput.characters {  switch character {  case "a", "e", "i", "o", "u", " ":  continue  default:  puzzleOutput.append(character)  }  }  print(puzzleOutput)  // Prints "grtmndsthnklk" |

1. *Break*

Lệnh break kết thúc việc thực hiện của toàn bộ lệnh chuyển điều khiển ngay lập tức. Lệnh break có thể được sử dụng bên trong lệnh switch, hoặc lệnh lặp khi người dùng muốn chấm dứt việc thực hiện của lệnh switch.

|  |
| --- |
| let puzzleInput = "great minds think alike"  var puzzleOutput = ""  for character in puzzleInput.characters {  switch character {  case "a", "e", "i", "o", "u", " ":  puzzleOutput.append(character)  default: break  }  }  print(puzzleOutput) // Prints "ea i i aie" |

1. *Fallthrough*

|  |
| --- |
| let integerToDescribe = 5  var description = "The number \(integerToDescribe) is"  switch integerToDescribe {  case 2, 3, 5, 7, 11, 13, 17, 19:  description += " a prime number, and also"  fallthrough  default:  description += " an integer."  }  print(description)  // Prints "The number 5 is a prime number, and also an integer." |

1. **Hàm (Function)**
   * + ***Khai báo và gọi function***

|  |
| --- |
| func sayHello(personName: String) -> String {  let greeting = "Hello, " + personName + "!"  return greeting  }  print(sayHello("Anna"))  // Prints "Hello, Anna!"  print(sayHello("Brian"))  // Prints "Hello, Brian!" |

* + - ***Tham số truyền và giá trị trả về của function***

Function không tham số

|  |
| --- |
| func sayHelloWorld() -> String {  return "hello, world"  }  print(sayHelloWorld())  // Prints "hello, world" |

Function nhiều tham số

|  |
| --- |
| func sayHello(personName: String, alreadyGreeted: Bool) -> String {  if alreadyGreeted {  return sayHello(personName)  } else {  return sayHello(personName)  }  }  print(sayHello("Tim", alreadyGreeted: true))  // Prints "Hello again, Tim!" |

Function không có giá trị trả về:

|  |
| --- |
| func sayGoodbye(personName: String) {  print("Goodbye, \(personName)!")  }  sayGoodbye("Dave")  // Prints "Goodbye, Dave!" |

Function với nhiều giá trị trả về

|  |
| --- |
| func minMax(array: [Int]) -> (min: Int, max: Int) {  var currentMin = array[0]  var currentMax = array[0]  for value in array[1..<array.count] {  if value < currentMin {  currentMin = value  } else if value > currentMax {  currentMax = value  }  }  return (currentMin, currentMax)  }  let bounds = minMax([8, -6, 2, 109, 3, 71])  print("min is \(bounds.min) and max is \(bounds.max)")  // Prints "min is -6 and max is 109" |

Tham số In-Out

|  |
| --- |
| func swapTwoInts(inout a: Int, inout \_ b: Int) {  let temporaryA = a  a = b  b = temporaryA  }  var someInt = 3  var anotherInt = 107  swapTwoInts(&someInt, &anotherInt)  print("someInt is now \(someInt), and anotherInt is now \(anotherInt)")  // Prints "someInt is now 107, and anotherInt is now 3" |

* + - ***Biến bằng function***

|  |
| --- |
| func addTwoInts(a: Int, \_ b: Int) -> Int {  return a + b  }  func multiplyTwoInts(a: Int, \_ b: Int) -> Int {  return a \* b  }  var mathFunction: (Int, Int) -> Int = addTwoInts  print("Result: \(mathFunction(2, 3))")  // Prints "Result: 5"  mathFunction = multiplyTwoInts  print("Result: \(mathFunction(2, 3))")  // Prints "Result: 6" |

1. **Kiểu liệt kê (Enumeration)**

Khai báo cách 1:

Cú pháp:

*enum <tên enum>{*

*case <phần tử 1>*

*case <phần tử 2>*

*…*

*}*

Ví dụ

|  |
| --- |
| enum CompassPoint {  case North  case South  case East  case West  } |

Khai báo cách 2:

Cú pháp:

*enum <tên enum>{*

*case <phần tử 1>, <phần tử 2>, <phần tử 3>, …*

*}*

Ví dụ:

|  |
| --- |
| enum Planet {  case Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune  } |

* + 1. *Kết hợp các giá trị kiểu Enumeration với câu lệnh Switch*

|  |
| --- |
| var directionToHead = CompassPoint.West  switch directionToHead {  case .North:  print("Lots of planets have a north")  case .South:  print("Watch out for penguins")  case .East:  print("Where the sun rises")  case .West:  print("Where the skies are blue")  }  // Prints “Where the skies are blue" |

1. **Lớp và cấu trúc (Classes And Structures)**
   * + ***Định nghĩa:***

Cú pháp:

*class <tên class>{*

*// các thuộc tính và phương thức*

*}*

*struct <tên struct>{*

*//các thuộc tính và phương thức*

*}*

|  |
| --- |
| class SomeClass {  // class definition goes here  }  struct SomeStructure {  // structure definition goes here  } |

Khai báo các thuộc tính trong class và struct

|  |
| --- |
| struct Resolution {  var width = 0  var height = 0  }  class VideoMode {  var resolution = Resolution()  var interlaced = false  var frameRate = 0.0  var name: String?  } |

Cách khai báo và lấy giá trị từ class và struct

|  |
| --- |
| let someResolution = Resolution()  print("The width of someResolution is \(someResolution.width)")  // Prints "The width of someResolution is 0"  let someVideoMode = VideoMode()  print("The width of someVideoMode is \(someVideoMode.resolution.width)")  // Prints "The width of someVideoMode is 0" |

* + - ***So sánh giữa class và struct:***

Class và Struct trong Swift có nhiều điểm chung:

* Thuộc tính (Properties): struct và class đều có thuộc tính.
* Method (Phương thức): struct và class đều có method.
* Hàm khởi tạo (Initializier): struct và class hỗ trợ các phương thức khởi tạo mặc định, hoặc các phương thức khởi tạo tùy chỉnh theo mục đích và logic của lập trình viên.
* Subscript: Struct và class hỗ trợ các subscript syntax (các câu lệnh con bên trong một thuộc tính, hoặc một hàm).
* Extension: Extension là một khái niệm mới trong Swift, nó giúp lập trình viên có thể mở rộng các struct hoặc class sẵn có hoặc đã được xây dựng từ trước.

Sự khác biệt giữa struct và class:

* Type (kiểu): struct là value type còn class và reference type.
* Inheritance (kế thừa): struct không kế thừa, class thì có thể kế thừa.
* Deinitializers: Struct không có hàm hủy (destructor trong java/C++), chỉ có hàm khởi tạo initializer, class thì có đầy đủ
* Multiple reference (đa tham chiếu): người dùng có thể có nhiều đối tượng cùng tham chiếu đến 1 class, còn struct thì không thể.

*Giải thích Value type và reference type.*

* Reference type: là dạng tham chiếu – ánh xạ bộ nhớ.
* Value type: kiểu sao chép và và để lại một bản.

Ví dụ 1:

|  |
| --- |
| Screen Shot 2016-03-23 at 14.14.40.png |

Hình. Ví dụ 1: sự khác biệt giữ struct và class

Qua ví dụ trên ta thấy được Struct copy và tạo mới, còn class thì giữ tham chiếu, vẫn lấy giá trị từ địa chỉ gốc.

Ví dụ 2:

|  |
| --- |
| Screen Shot 2016-03-23 at 15.04.59.png |

*Hình. Ví dụ 2: sự khác biệt giữ struct và class*

Qua ví dụ trên chúng ta có thể thấy rõ ràng, khi dòng lệnh chạy trong hàm changeName, property “name” của cả struct và class đề thay đổi tương ứng. Nhưng đến khi ra khỏi hàm, property “name” trong struct vẫn giữ nguyên giá trị, trong khi class lại thay đổi.

Trường hợp như vậy được giải thích như sau: Struct không hề truyền địa chỉ của mình, mà thay vào đó, hàm changeName sẽ copy lại myStruct truyền vào. Đến khi hàm kết thúc, giá trị copy đó bị giải phóng đi (release), và nó không hề ảnh hưởng đến myStruct truyền vào. Còn class, khi trong hàm changeName giá trị được gán mới trỏ tới địa chỉ của myClass để thay đổi nên giá trị của thuộc tính “name” sẽ thay đổi theo khi ra khỏi hàm changeName.

1. **Thuộc tính (Properties)**

Properties là các thuộc tính của một class hoặc một struct hoặc một enumeration cụ thể. Thuộc tính lưu trữ là một hằng số hoặc một biến mà được lưu trữ như một phần của một lớp hoặc một cấu trúc cụ thể.

|  |
| --- |
| struct FixedLengthRange {  var firstValue: Int  let length: Int  }  var rangeOfThreeItems = FixedLengthRange(firstValue: 0, length: 3)  // the range represents integer values 0, 1, and 2  rangeOfThreeItems.firstValue = 6  // the range now represents integer values 6, 7, and 8 |

Thể hiện của FixedLengthRange có một thuộc tính lưu trữ biến gọi là firstValue và một thuộc tính lưu trữ hằng gọi là length. Trong ví dụ trên length được khởi tạo khi một phạm vi mới được tạo ra và không thể bị thay đổi sau đó, bởi vì nó là một thuộc tính hằng số.

1. **Phương thức (Method)**

Method là các hàm thuộc về thể hiện của một lớp, một cấu trúc, hoặc một liệt kê nào đó. Chúng hỗ trợ các chức nawgn cho những thể hiện đó, hoặc cung cấp những các để truy cập và sửa đổi các thuộc tính. Các phương thức có cấu trúc tương tự như các hàm.

|  |
| --- |
| class Counter {  var count = 0  func increment() {  count += 1  }  func incrementBy(amount: Int) {  count += amount  }  func reset() {  count = 0  }  } |

Lớp Counter định nghĩa 3 phương thức thể hiện:

* Increment tang bộ đếm lên 1
* incrementBy (amount: Int) tang bộ đếm bởi một số nguyên xác định.
* Reset đặt lại bộ đếm bằng 0.

Lớp Counter cũng khai báo một thuộc tính biến, count, để theo dõi các giá trị truy cập hiện tại.

Cách gọi phương thức thể hiện với dấu chấm cùng một cú pháp như là thuộc tính:

|  |
| --- |
| let counter = Counter()  // the initial counter value is 0  counter.increment()  // the counter's value is now 1  counter.incrementBy(5)  // the counter's value is now 6  counter.reset()  // the counter's value is now 0 |

Gán self trog một phương thức biến đổi

* + 1. Phương thức biến đổi có thể gán một thể hiện hoàn toàn mới đối với thuộc tính self.

Ví dụ

|  |
| --- |
| struct Point {  var x = 0.0, y = 0.0  mutating func moveByX(deltaX: Double, y deltaY: Double) {  self = Point(x: x + deltaX, y: y + deltaY)  }  } |

Khi thực hiện phương thức này thì sẽ tạo ra một Point mới với các giá trị x và y tương ứng với các tham số truyền vào.

* + 1. Tham số cục bộ và tham số mở rộng cho phương thức (Local and External Parameter Name for Methods)

Swift đặt mặc định tham số được truyền đầu tiên trong method là tham số địa phương, các tham số theo sau nó gọi là tham số mở rộng.

Ví dụ

|  |
| --- |
| class Counter {  var count: Int = 0  func incrementBy(amount: Int, numberOfTimes: Int) {  count += amount \* numberOfTimes  }  } |

Phương thức incrementBy có hai tham số mặc định là amount và numberOfTime. Theo mặc định, Swift coi amount như là một tên địa phương, nhưng coi numberOfTime là tham số mở rộng

Khi gọi hàm sẽ có dạng như sau:

|  |
| --- |
| let counter = Counter()  counter.incrementBy(5, numberOfTimes: 3)  // counter value is now 15 |

1. **Các hàm con (Subscript)**

Lớp (class), cấu trúc(structure), và kiểu liệt kê (enumeration) có thể định nghĩa subscript, đó là các phím tắt để truy cập và các phần tử của một tập hợp, danh sách, hoặc chuỗi. Người dùng sử dụng subscripts để thiết lập và lấy giá trị của index mà không cần phương thức riêng biệt cho thiết lập và phục hồi.

*Cú pháp Suupsript*

Subscripts cho phép người dùng truy vấn các thể hiện của một kiểu bằng cách viết một hoặc nhiều giá trị trong dấu ngoặc vuông sau tên thể hiện. Người dùng viết định nghĩa subscript với từ khóa subscript, và chỉ định một hoặc nhiều tham số đầu vào và một kiểu trả về, trong cùng một cách như các phương thức thể hiện. Điều không giống như các phương thức là script đó có thể đọc – ghi hoặc chỉ đọc.

Hoạt động này được thể hiện bở một cặp getter và setter trong cùng một script.

|  |
| --- |
| subscript(index: Int) -> Int {  get {  // return an appropriate subscript value here  }  set(newValue) {  // perform a suitable setting action here  }  } |

Ví dụ subscript chỉ đọc

|  |
| --- |
| struct TimesTable {  let multiplier: Int  subscript(index: Int) -> Int {  return multiplier \* index  }  }  let threeTimesTable = TimesTable(multiplier: 3)  print("six times three is \(threeTimesTable[6])")  // Prints "six times three is 18" |

1. **Kế thừa (Inheritance)**

* Một class có thể kế thừa (inherit) method, property, và các đặc tính khác từ các class khác. Khi một class kế thừ từ một class khác, class kế thừa được gọi là một class con (subclass), và class được nó kế thừa gọi là class cha (superclass) của nó. Kế thừa là một hoạt động cơ bản để phân biệt các lớp từ các kiểu khác nhau trong Swift.
* Class trong Swift có thể gọi và truy cập các method, property, và subscript thuộc class cha của chúng và có thể cung cấp thêm các hình thức ghi đè trong lớp con lên những phương thức, thuộc tính và subscript để tinh chỉnh hay thay đổi hoạt động trong chúng. Swift giúp đảm bảo việc ghi đề của người dùng chính xác bằng cách kiểm tra các định nghĩa ghi đề đã có một định nghĩa phù hợp vs class cha.
* Cách định nghĩa một lớp cha

|  |
| --- |
| class Vehicle {  var currentSpeed = 0.0  var description: String {  return "traveling at \(currentSpeed) miles per hour"  }  func makeNoise() {  // do nothing - an arbitrary vehicle doesn't necessarily make a noise  }  } |

* + - ***Lớp con***
* Lớp con là một thể hiện của một lớp mới dựa trên một lớp hiện có. Các lớp con thừa hưởng những đặc tính từ lớp hiện có, sau đó người dùng có thể tinh chỉnh hoặc có thể thêm các thuộc tính, phương thức mới cho lớp con đó.
* Cách định nghĩa một lớp con:

|  |
| --- |
| class Bicycle: Vehicle {  var hasBasket = false  } |

* Có thể chỉnh sửa thuộc tính từ lớp Vehicle như sau:

|  |
| --- |
| bicycle.currentSpeed = 15.0  print("Bicycle: \(bicycle.description)")  // Bicycle: traveling at 15.0 miles per hour |

* + - ***Overriding (ghi đè)***

1. *Ghi đè lên thuộc tính*

Người dùng có thể ghi đè lên thuộc tính lớp thừa kế để cung cấp getter và setter tùy chỉnh của riêng người dùng cho thuộc tính đó, bằng cách để từ “override ” trước thuộc tính đó.

|  |
| --- |
| class Car: Vehicle {  var gear = 1  override var description: String {  return super.description + " in gear \(gear)"  }  } |

Việc ghi đè lên thuộc tính description bắt đầu bằng cách gọi super.description, trả thuộc tính description của lớp Vehicle. Phiên bản lớp Car của description sau đó bổ sung thêm một số văn bản mở rộng vào cuối của mô tả này.

1. *Ghi đè lên phương thức*

Với cùng một phương thức ở lớp cha, thì đứng ở lớp con ta có thể ghi đè lên phương thức đó bằng cách thay đổi tham số (về kiểu của tham số hoặc số lượng của tham số.). Bằng cách dùng từ override trước hàm.

|  |
| --- |
| class cricket {  func print() {  println("Welcome to Swift Super Class")  }  }  class tennis: cricket {  override func print() {  println("Welcome to Swift Sub Class")  }  }  let cricinstance = cricket()  cricinstance.print()  let tennisinstance = tennis()  tennisinstance.print()  // print : “Welcome to Swift Super Class”  // print : “Welcome to Swift Sub Class” |

Từ ví dụ trên ta thấy phương thức print ở lớp tennis được ghi đè lên phương thức print ở class cricket. ở 2 phương thức có nội dung khác nhau sẽ cho ra kết quả khác nhau, tương tự chúng ta có thể thay đổi tham số cho các phương thức về kiểu và số lượng các tham số vẫn được.

* *Ngoài ra có thể ghi đè lên supscript.* 
  + 1. Preventing Overrides (Ngăn ghi đè)

Người dùng có thể ngăn chặn một phương thức, thuộc tính, hoặc subscript bị ghi đè bằng cách đánh dấu nó như là final. Làm đều này bằng cách viết các sửa đổi final trước từ khóa giới thiệu của phương thức, thuộc tính, hoặc subscript (như final var, final func, final class func, final subsript).

1. **Khởi tạo (Initialization)**

Hàm khởi tạo là hàm tạo ra một thể hiện mới cho một lớp, một cấu trúc, hoặc liệt kê để sử dụng. Quá trình này bao gồm việc thiết lập lại giá trị ban đầu cho mỗi thuộc tính được lưu trữ.

Hàm khởi tạo được viết bằng từ khóa “init”

*Hàm khởi tạo không tham số:*

|  |
| --- |
| struct Fahrenheit {  var temperature: Double  init() {  temperature = 32.0  }  }  var f = Fahrenheit()  print("The default temperature is \(f.temperature)° Fahrenheit")  // Prints "The default temperature is 32.0° Fahrenheit" |

Cấu trúc định nghĩa một bộ khởi tạo duy nhất, init không có tham sô, khởi tạo nhiệt độ lưu trữ với giá trị 32.0.

Hàm khởi tạo có tham số:

|  |
| --- |
| struct Color {  let red, green, blue: Double  init(red: Double, green: Double, blue: Double) {  self.red = red  self.green = green  self.blue = blue  }  init(white: Double) {  red = white  green = white  blue = white  }  }  // Khi gọi hàm  let magenta = Color(red: 1.0, green: 0.0, blue: 1.0)  let halfGray = Color(white: 0.5) |

Cả hai bộ khởi tạo có thể được sử dụng để tạo thể hiện Color mới, bằng cách cung cấp các giá trị được đặt tên cho từng tham số khởi tạo

Lưu ý: Không thể gọi các bộ khởi tạo có nhiều hơn 1 tham số truyền vào mà không sử dụng tên tham số mở rộng. Tên mở rộng phải luôn được sử dụng trong một bộ khởi tạo nếu chúng được định nghĩa, và việc bỏ qua chúng sẽ xảy ra lỗi trong quá trình biên dịch.

1. **Hàm huỷ (Deinitialization)**

* Một hàm hủy được khởi tạo được gọi ngay trước khi một thể hiện của lớp được giải phóng.
* Hàm hủy được viết với từ khóa “*deinit*”, tương tự như cách bộ khởi tạo – initializers – được viết với từ khóa init. Deinitializers chỉ có sẳn trên cấu trúc lớp.

Ví dụ:

|  |
| --- |
| class Nguoi {  var ten:String = ""  var tuoi:Int = 0  init(Ten:String, Tuoi:Int){  ten = Ten  tuoi = Tuoi  }  deinit{  print("Giá trị nil")  }  }  var nguoi1:Nguoi = Nguoi(Ten: "Le Lam", Tuoi: 26)  var nguoi2:Nguoi? = Nguoi(Ten: "KhoaPham", Tuoi: 28)  //Nguoi2 có giá trị  nguoi2 = nil  //đã được giải phóng và thực thi các câu lệnh trong deinit  //print : “Giá trị nil” |

1. **Optional Chaining**

Optional Chaining là một quá trình để truy vấn và gọi các property, method, và subscript vào một optional mà hiện tại nó có thể là nil. Nếu optional có chứa một giá trị, thuộc tính, phương thức, hoặc kí hiệu gọi trả về giá trị nào đó. Nếu optional là nil, thuộc tính, phương thức, hoặc subscrip gọi trả về là nil.

|  |
| --- |
| class Person {  var residence: Residence?  }  class Residence {  var numberOfRooms = 1  }  let john = Person()  let roomCount = john.residence!.numberOfRooms  // this triggers a runtime error  if let roomCount = john.residence?.numberOfRooms {  print("John's residence has \(roomCount) room(s).")  }  else  {  print("Unable to retrieve the number of rooms.")  }  // Prints "Unable to retrieve the number of rooms." |

Nếu ta thực hiện câu lệnh let roomCount = john.residence!.numberOfRooms sẽ xảy ra lỗi vì join.residence! là nguyên nhân gây ra lỗi, dấu “!” có nghĩa là residence không được rỗng, nhưng trong khai báo thì nó được rỗng. Vì thế ta phải dùng câu lệnh: roomCount = john.residence?.numberOfRooms.

1. **Xử lý lỗi (Handling Error)**

Là quá trình xử lý lỗi và khắc phục từ các lỗi trong chương trình. Swift đã cung cấp first-class hỗ trợ việc xuất lỗi, bắt lỗi, tuyên truyền và khắc phục lỗi khi chạy.

* + - ***Biểu diễn lỗi (Representing Errors)***

Enumeration đặc biệt thích hợp để mô hình hóa một nhóm các điều kiện lỗi với các giá trị liên quan để biết thêm về bản chất của một lỗi được truyền vào.

|  |
| --- |
| enum VendingMachineError: ErrorType {  case InvalidSelection  case InsufficientFunds(coinsNeeded: Int)  case OutOfStock  } |

* + - ***Throwing Errors***

Chỉ ra function hoặc method có thể ném ra lỗi

|  |
| --- |
| var inventory = [  "Candy Bar": Item(price: 12, count: 7),  "Chips": Item(price: 10, count: 4),  "Pretzels": Item(price: 7, count: 11)]  func vend(itemNamed name: String) throws {  guard let item = inventory[name] else {  throw VendingMachineError.InvalidSelection  }  guard item.count > 0 else {  throw VendingMachineError.OutOfStock  }  guard item.price <= coinsDeposited else {  throw VendingMachineError.InsufficientFunds(coinsNeeded: item.price -coinsDeposited)  } |

* + - ***Bắt và xử lý lỗi (Catching and Handling Errors)***

Cú pháp:

do {

try *expression*

*statements*

} catch *pattern 1* {

*statements*

} catch *pattern 2* where *condition* {

*statements*

}

Ví dụ

|  |
| --- |
| var vendingMachine = VendingMachine()  vendingMachine.coinsDeposited = 8  do {  try buyFavoriteSnack("Alice", vendingMachine: vendingMachine)  } catch VendingMachineError.InvalidSelection {  print("Invalid Selection.")  } catch VendingMachineError.OutOfStock {  print("Out of Stock.")  } catch VendingMachineError.InsufficientFunds(let coinsNeeded) {  print("Insufficient funds. Please insert an additional \(coinsNeeded) coins.")  }  // Prints "Insufficient funds. Please insert an additional 2 coins." |

1. **Một số control quan trọng**
   * + ***UILabel***
2. *Chức năng:*

* UILabel giưps chúng ta hiển thị một một tiêu đề, một đoạn giới thiệu, một chuỗi text.

1. *Cách cấu hình và cách sử dụng:*

* Cấu hình UILabel rất đơn giản, chúng ta có thể tạo ra một UIButton bằng cách kéo thả hoặc viết code khai báo. Bên cạnh định dạng mặc định, chúng ta cũng dễ dàng tuỳ chỉnh lại UILabel theo ý tưởng của chúng ta.
* Việc sử dụng UILabel chỉ đơn giản là truyền vào một giá trị kiểu chuỗi (String) để hiển thị trong Label.

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.15.16%20AM.png |

*Hình 3.5. Control Label*

Sử dụng code để tạo và định dạng lable

|  |
| --- |
| var lblTuaDeMoTa:UILabel! //khai báo label  //khởi tạo label  lblTuaDeMoTa = UILabel(frame: CGRect(x: 0, y: view2.frame.size.height - heightTabBar - heightLabel, width: view2.frame.size.width , height: heightLabel))    //gán text cho lable  dinhDangLable(lblTuaDeMoTa, text: "Mô tả ") |

1. *Ứng dụng trong đồ án:*

Label được sử dụng để hiển thị các thông tin tiêu đề, thông tin thuộc tính của món ăn trong các dòng danh sách. Ngoài ra, Label còn được ứng dụng để hiển thị một số thông tin chức năng của ứng dụng.

* + - ***UITextfile***

1. *Chức năng:*

UITextField giúp ứng dụng tương tác với các dữ liệu được nhập từ bàn phím, bắt các giá trị đó để những đoạn mã thực thi của ứng dụng có thể sử dụng thực hiện những công việc xử lý cần thiết.

1. *Cách cấu hình và cách sử dụng:*

UITextField có thể được cấu hình bằng cách kéo thả hoặc viết code, tương tự như UILabel.

UITextField thường được sử dụng để nhập dữ liệu thông tin người dùng, các thông tin cần tìm kiếm, thông tin về sản phẩm, dịch vụ để từ đó lưu trữ lên cơ sở dữ liệu.

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.15.57%20AM.png |

*Hình 3.6. Control Textfile*

Khởi tạo bằng code

|  |
| --- |
| var txtMoTa: UITextView! //khai báo textfile  //khởi tạo textfile  let heightMoTa = view2.frame.size.height \* 95 / 342  txtMoTa = UITextView(frame: CGRect(x: 8 , y: 4, width: view2.frame.size.width - 16, height: heightMoTa )) |

1. *Ứng dụng trong đồ án:*

Trong đồ án này, UITextField được ứng dụng để tạo form đăng kí/ đăng nhập người dùng, tạo thanh tìm kiếm một món ăn, điền các thông tin để thêm món ăn. Ngoài ra UITextFiel còn được ứng dụng để tạo giao diện cho trang thêm thông tin một món ăn,…

* + - ***UIButton***

1. *Chức năng:*

UIButton tạo ra một block giúp thực hiện một đoạn mã lệnh khi chúng ta nhấn chọn nút UIButton.

1. *Cách cấu hình và cách sử dụng:*

UIButton cũng được cấu hình bằng cách kéo thả, tuỳ chỉnh các thuộc tính hoặc sử dụng code để cấu hình. Việc tuỳ chỉnh cấu hình UIButton cũng khá đơn giản và dễ thực hiện với các hàm đã được Swift xây dựng sẵn. Ngoài ra chúng ta cũng có thể viết thêm các hàm để tuỳ chỉnh cấu hình cho UIButton nói riêng và các đối tượng khác nói chung.

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.15.35%20AM.png |

*Hình 3.7. Control button*

Cách tạo button trong code:

|  |
| --- |
| //khai báo button  var btn:UIButton!  //khởi tạo button  btn = UIButton()  //đặt tự đề cho button  btn.setTitle("Click me", forState: .Normal) |

1. *Ứng dụng trong đồ án:*

Trong đồ án này, UIButton được sử dụng để tạo các nút thực hiện các chắc năng như đăng kí, đăng nhập, upload món ăn, … Đồng thời, UIButton cũng được ứng dụng trong các dòng của danh sách món ăn giúp thực hiện các thao tác chuyển màn hình đến các giao diện như bình luận, thông tin chi tiết món ăn,…

* + - ***UIImageView***

1. *Chức năng:*

UIImageView được sử dụng với mục đích chính là hiển thị dữ liệu hình ảnh cho người dùng quan sát. Ngoài ra với việc kết hợp các Gesture, UIImage trong Swift càng mạnh mẽ hơn bao giờ hết, có thể tạo nên rất nhiều hiệu ứng đẹp mắt và thú vị.

1. *Cách cấu hình và cách sử dụng:*

Cấu hình UIImageView quan trọng nhất là xử lý độ phân giải cho hình ảnh, việc đổ dữ liệu vào để hiển thị ra thì đã có các hàm có sẵn của Swift giải quyết.

UIImageView được dùng chủ yếu hiển thị hình ảnh sản phẩm, ảnh đại diện của user, các background ứng dụng,…

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.16.15%20AM.png |

*Hình 3.8.Control Imgae*

Cách thiết lập bằng code:

|  |
| --- |
| // khai báo hình ảnh  var imgHinhMonAn: UIImageView!  // định dạng cho ảnh  imgUserDangNhap.layer.cornerRadius = 20  imgUserDangNhap.clipsToBounds = true |

1. *Ứng dụng trong đồ án:*

Tất cả các hình ảnh của món ăn, ảnh đại diện của người dùng, ảnh nền các màn hình,… trong đồ án này đều sử dụng UIImageView để giải quyết.

* + - ***UITabBar***

1. *Chức năng:*

UITabBar đơn giản là giúp chúng ta tạo ra một thanh công cụ gồm các nút chuyển tới các màn hình khác nhau.

1. *Cách cấu hình và cách sử dụng:*

Chúng ta có thể cấu hình số lượng nút cho UITabBar để chuyển tới số lượng màn hình mà chúng ta mong muốn, đồng thời có thể tuỳ chỉnh hình ảnh cho mỗi nút chuyển màn hình của UITabBar.

UITabBar thường được sử dụng trong giao diện màn hình chính, từ đó sẽ gọi được các màn hình khác một cách dễ dàng.

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.16.40%20AM.png |

*Hình 3.9. Control tabBar*

1. *Ứng dụng trong đồ án:*

Trong đồ án này, UITabBar được sử dụng để gọi đến các màn hình thông tin chức năng cơ bản nhất của ứng dụng như: Nấu ăn, Sức khoẻ, Nhà hàng, Tin tức, Tìa khoản.

* + - ***UITableView***

1. *Chức năng:*

UITableView có chức năng hiển thị dữ liệu dưới dạng danh sách các dòng, mỗi dòng vừa hiển thị thông tin vừa cho phép người dùng thao tác với thông tin đó. UITableView thường kết hợp với các đối tượng cơ bản bên trên để hiển thị dữ liệu phong phú và đẹp mắt hơn.

1. *Cách cấu hình và cách sử dụng:*

Một UITableView được cấu hình qua 3 bước:

* Thứ nhất: khai báo số lượng section và số lượng dòng trong mỗi section.
* Thứ hai: cấu hình giao diện và đổ dữ liệu vào từng dòng trong TableView. Bước này đòi hỏi xử lý rất nhiều về Layout để có một giao diện thực sự bắt mắt.
* Cuối cùng: cấu hình các hành động sẽ thực hiện khi người dùng tương tác với một dòng dữ liệu. Bước này đòi hỏi nắm rõ code và các luồng xử lý chuyển màn hình.

Cách sử dụng: UITableView được sử dụng như một màn hình chính load danh sách các sản phẩm, dữ liệu lên cho người dùng xem, bắt sự kiện và xử lý khi người dùng tương tác với các dòng.

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.16.54%20AM.png |

*Hình 3.10. Control TableView*

Các hàm quan trọng trong table view

|  |
| --- |
| func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  // #warning Incomplete method implementation.  // trả về số dòng của table  print(mangTuDe.count)  return mangTuDe.count  }      // hàm gán các giá trị, định dạng cho các dòng  func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {  let cell = tableView.dequeueReusableCellWithIdentifier(mangviTri[indexPath.row], forIndexPath: indexPath) as! MenuSongKhoeTableViewCell    // gán các nội dung cho các dòng  cell.lbl.text = mangTuDe[indexPath.row]  cell.lbl.font = cell.lbl.font.fontWithSize(17)      cell.img.image = mangHinhLoaiBaiVietLocal[indexPath.row]    return cell  } |

1. *Ứng dụng trong đồ án:*

UITableView được sử dụng để load dữ liệu danh sách về loại món ăn, load dữ liệu danh sách món ăn, … Ngoài ra UITableView còn được sử dụng để load dữ liệu về bình luận của người dùng đối với các các món ăn.

* + - ***UICollectionView***

1. *Chức năng:*

Tương tự như UITableView, chức năng chính là hiển thị danh sách dữ liệu dưới dạng các ô cửa sổ, đồng thời cho phép tương tác sự kiện với người dùng.

1. *Cách cấu hình và cách sử dụng:*

UICollectionView cũng gồm 3 bước cấu hình tương tự như UITableView.

Cách sử dụng cũng gần tương tự như UITableView, tuy nhiên UICollectionView sẽ được ưu tiên sử dụng hơn khi dữ liệu là các mảng hình ảnh.

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.17.12%20AM.png |

*Hình 3.11. Control CollectionView*

Các hàm quan trọng của collectionView

|  |
| --- |
| func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {  //hàm trả về số lượng phần tử trong collectionview  return mangBaiViet.count  }    func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {  let cell = collectionView.dequeueReusableCellWithReuseIdentifier("Cell", forIndexPath: indexPath) as! BaiVietCollectionViewCell    //dinh dang cell (mỗi phần tử)    cell.frame.origin.x = collectionBaiViet.frame.size.width \* 16 / 375  cell.frame.size.width = view.frame.size.width \* 344 / 375  cell.frame.size.height = view.frame.size.height \* 300 / 667    cell.lblTextView.text = mangBaiViet[indexPath.row].noiDung  cell.lblTuaDe.text = mangBaiViet[indexPath.row].tenBaiViet  cell.imgBaiViet.image = mangBaiViet[indexPath.row].hinhBaiViet    return cell  } |

1. *Ứng dụng trong đồ án:*

UICollectionview được sử dụng trong đồ án tạo ra danh sách các loại món ăn dạng một mảng hình ảnh, bắt sự kiện người dùng tương tác và xử lý các hành động tương ứng.

* + - ***UIScrollView***

1. *Chức năng:*

UIScrollView thường được sử dụng để tạo các hiệu ứng chuyển các UIView và đối tượng trong 1 màn hình.

1. *Cách cấu hình và cách sử dụng:*

* UIScrollView được cấu hình qua 3 bước:
* Xác định vị trí cho ScrollView
* Xác định các đối tượng chứa trong ScrollView
* Xác định contentSize cho ScrollView
* UIScrollView thường được dùng trong các thanh cuộn các chức năng: ví dụ như cuộn giữa chức năng và đọc tin tức, …

1. *Ứng dụng trong đồ án:*

Trong đồ án này, UIScrollView được dùng để trượt qua lại giữa các thông tin chi tiết về món ăn: thông tin công thức, videos về món ăn, tin tức trong UIWebView,…

Ngoài những class đối tượng cơ bản kể trên thì ứng dụng còn sử dụng một số class đối tượng hỗ trợ khác:

* + - ***UIWebView:***

Hỗ trợ load URL html và hiển thị ra trong ứng dụng.

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.22.05%20AM.png |

*Hình 3.12. Control WebView*

* + - ***UINavigation***

Tạo hiệu ứng trở về trang trước đó trong quá trình chuyển trang

|  |
| --- |
| Screen%20Shot%202016-06-17%20at%202.22.21%20AM.png |

*Hình 3.13. Control Navigation*

1. **Đa luồng (Multi Thread)**

* Một ứng dụng luôn tồn tại một tiểu trình (thread) chính thực thi ứng dụng đó, việc thực thi bao gồm những xử lý về giao diện, đọc dữ liệu liệu, tính toán, tất cả code người dùng mặc định được thực thi trong tiểu trình chính của ứng dụng, còn gọi là **main thread.**
* Tuy nhiên, khi một chương trình có quá nhiều chức năng như: tải ảnh, xử lý dữ liệu, tính toán trong cùng 1 thao tác thì sẽ dẫn đến vấn đề về thời gian xử lý mất nhiều hơn do chỉ thực hiện trên cùng 1 hàng đợi, đều này sẽ gây sự khó chịu cho người dùng.
* Đều này có thể được giải quyết bằng 1 số cách như GCD, NSThread, …
* GCD viết tắt của Grand Central Dispatch. GCD hỗ trợ việc quản lý một hàng đợi công việc. Mặc định, một ứng dụng iOS luôn luôn tồn tại một hàng đợi được tạo sẵn gọi là main queue, và main thread được tạo ra để xử lý những công việc trong hàng đợi này. Có một hàng đợi khác gọi là globel queue là hàng đợi của hệ thống, quản lý thread đang thực thi trên các ứng dụng khác nhau. Hoặc có thể tạo ra hàng đợi của chính người dùng bằng GCD.
* Có các hàm dispatch như sau:
  + - Dispatch\_queue\_create(UnsafePointer<Int8>, dispatch\_queue\_attr\_t!)
    - Dispatch\_async(dispatch\_queue\_t),{ Các việc thực hiện}).
    - Dispatch\_group\_create().
    - Dispatch\_group\_enter(dispatch\_group\_t) .
    - Dispatch\_group\_leave(dispatch\_group\_t) .
    - Dispatch\_group\_wait(dispatch\_group\_t, dispatch\_time\_t).
    - ***Dispatch\_queue\_create(UnsafePointer<Int8>, dispatch\_queue\_attr\_t!)***
* Mô tả: Tạo 1 queue(hàng đợi).
* Tham số 1: Tên cho hàng đợi
* Tham số 2: Chọn kiểu hàng đợi

Có 2 kiểu hàng đợi:

* DISPATCH\_QUEUE\_CONCURRENT : tạo ra các luồng song song trong CPU, tận dụng hết tất cả các luồng trong CPU
* DISPATCH\_QUEUE\_SERIAL: Tạo một luồng dữ liệu: sự việc 1 thực hiện xong mới đến sự việc 2, …
  + - ***Dispatch\_async(dispatch\_queue\_t),{ Các việc thực hiện})***
* Mô tả: Tạo một luồng dữ liệu mới, code xử lý nằm trong cặp {}, tham số thứ nhất truyền vào là 1 hàng đợi được tạo từ ở trên. Hoặc các queue khác như:
* Dispatch\_get\_main\_queue(): Gọi luồng chính
* Dispatch\_get\_global\_queue(Int, Int):Gọi một hàng đợi được xây dựng sẵn.
  + - ***Dispatch\_group\_create(): để tạo một group mới***
    - ***Dispatch\_group\_enter(dispatch\_group\_t)*** :
* Mở luồng dữ liệu mới với group vừa tạo
  + - ***Dispatch\_group\_leave(dispatch\_group\_t)***.
* Đóng luồng dữ liệu group.
  + - ***Dispatch\_group\_wait(dispatch\_group\_t, dispatch\_time\_t):***
* Chờ luồng dữ liệu trong group thực hiện xong với một khoảng thời gian nào đó

|  |
| --- |
| func forgetPassword(email:String) ->Bool{  var isSuccess = false  let ref = Firebase(url: DuongDan.User.fullDuongDanUrl())  let group = dispatch\_group\_create()  dispatch\_group\_enter(group)  ref.resetPasswordForUser(email, withCompletionBlock: { error in  if error != nil {  isSuccess = false  } else {  isSuccess = true  }  dispatch\_group\_leave(group)  })    dispatch\_group\_wait(group, DISPATCH\_TIME\_FOREVER) //chờ đến khi những câu lệnh phía trên thực hiện xong mới thực hiện tiếp    return isSuccess  }  @IBAction func ForgetPass(sender: AnyObject) {  if( txtTenDangNhap.text != ""){  dispatch\_async(dispatch\_get\_global\_queue(0, 0), {    let thongbao = self.xulyUser.forgetPassword(self.txtTenDangNhap.text!)    dispatch\_async(dispatch\_get\_main\_queue(), {  if(thongbao == true){  let alter\_controllerf:UIAlertController = self.xulyUser.thongbaoThanhCong("THÔNG BÁO", smessage: "Chúng tôi đã gửi mã đăng nhập vào email của bạn", titlebutton: "OK")  self.presentViewController(alter\_controllerf, animated: true, completion: nil)  }  else{  let alter\_controllerf:UIAlertController = self.xulyUser.thongBaoThatBai("THÔNG BÁO", smessage: "Quá trình reset password thất bại \n Hãy kiểm tra email của bạn!", titlebutton: "Cancel", error: "")  self.presentViewController(alter\_controllerf, animated: true, completion: nil)    }    })  })  }  else{  let alter\_controllerf:UIAlertController = xulyUser.thongBaoThatBai("THÔNG BÁO", smessage: "Bạn hãy nhập email để chúng tôi gửi password!", titlebutton: "Cancel", error: "")  self.presentViewController(alter\_controllerf, animated: true, completion: nil)  }    } |

Đầu tiên tạo một hàng đợi xây dựng sẵn nhờ vào hàm *dispatch\_asyn(,).* Khi đó, tham số thứ nhất truyền vào là *dispatch\_get\_global\_queue* để xây dựng một hàng đợi thực hiện các công việc trong nó.

Khi gọi hàm *forgetPassword* trong *dispatch\_get\_global\_queue* thì hàm này sẽ thực hiện xong thì trở về main chính nhờ hàm *dispatch\_get\_main\_queue*.

Hàm *dispatch\_async* được truyền hai tham số, tham số thứ nhất là tên hàng đợi đã tạo ra trước đó, tham số thứ 2 là danh sách các công việc cần được xử lý. Khi hàm forgetPassword thực hiện xong kết quả phải được đưa về main thread nhờ vào hàm *dispatch\_async* khi truyền tham số vào là main queue.

**CHƯƠNG 4: MỘT SỐ CÔNG NGHỆ PHỤ TRỢ**

1. **Firebase Platform**
   * + ***Giới thiệu firebase.***

**Firebase** là một dịch vụ cơ sở dữ liệu thời gian thực hoạt động trên nền tảng đám mây điện toán được cung cấp bởi Google nhằm giúp các lập trình viên phát triển nhanh các ứng dụng bằng cách đơn giản hóa các thao tác với cơ sở dữ liệu.

1. *Các chức năng chính của Firebase*

|  |
| --- |
| https://lh3.googleusercontent.com/-2EtNzOAQtOs/VnkPtp9FBPI/AAAAAAAACzQ/mnKNh-3Pii8/w426-h352/45ae6488-2b1a-4cf3-91d4-6f7f3b411865 |

*Hình 4.1. Firebase hỗ trợ đa nền tảng*

* *Realtime Database – Cơ sở thời gian thực:* Firebase lưu trữ dữ liệu database dưới dạng JSON và thực hiện đồng bộ database tới tất cả các client theo thời gian thực. Cụ thể hơn là người dùng có thể xây dựng được client đa nền tảng (cross-platform client) và tất cả các client này sẽ cùng sử dụng chung 1 database đến từ Firebase và có thể tự động cập nhật mỗi khi dữ liệu trong database được thêm mới hoặc sửa đổi.
* *Firebase Authentication – Hệ thống xác thực của Firebase (Hay nói cách khác là quản lý user về việc đăng ký đăng nhập):* Với Firebase, người dùng có thể dễ dàng tích hợp các công nghệ xác thực của Google, Facebook, Twitter, … hoặc một hệ thống xác thực mà người dùng tự mình tạo ra vào trong ứng dụng của người dùng ở bất kì nền tảng nào như Android, iOS hoặc Web.
* *Firebase Hosting*: Các dữ liệu được lưu trữ đám mây đồng thời được bảo mật thông qua giao thức truy cập SSL.
* Các ứng dụng sẽ được cấp 1 tên miền dạng \*.firebaseapp.com hoặc người dùng có thể trả tiền để sử dụng tên miền riêng của mình.

1. *Những lợi ích từ việc sử dụng Google Firebase*

* Triển khai ứng dụng nhanh:
* Giảm thời gian viết các dòng code để quản lý đồng bộ cơ sở dữ liệu, mọi việc sẽ diễn ra hoàn toàn tự động với các API của Firebase.
* Firebase hỗ trợ đa nền tảng nên người dùng đỡ mất thời gian rất nhiều cho một ứng dụng đa nền tảng.
* Google Firebase giúp người dùng đơn giản hóa quá trình đăng ký và đăng nhập vào ứng dụng bằng hệ thống xác thực do chính Firebase cung cấp.
* Bảo mật: Firebase hoạt động trên nền tảng đám mây điện toán và thực hiện kết nối thông qua giao thức bảo mật SSL, chính vì vậy người dùng sẽ bớt lo lắng rất nhiều về việc bảo mật của dữ liệu cũng như đường truyền giữa client và server.

**Giao thức bảo mật SSL là gì?**

SSL là viết tắt của Secure Socket Layer. Đây là một tiêu chuẩn an ninh công nghệ toàn cầu tạo ra một liên kết được mã hóa giữa máy chủ và trình duyệt. Liên kết này đảm bảo tất cả dữ liệu trao đổi giữa máy chủ web và trình duyệt luôn được bảo mật và an toàn. SSL đảm bảo rằng tất cả các dữ liệu được truyền giữa các máy chủ web và các trình duyệt được mang tính riêng tư, tách rời. SSL là một chuẩn công nghiệp được sử dụng bởi hàng triệu trang web trong việc bảo vệ các giao dịch trực tuyến với khách hàng của họ.

*Chế độ làm việc của SSL:*

|  |
| --- |
| https://www.matbao.net/App_Themes/MatBaoNet2014/img/banner-ssl.jpg |

*Hình 4.2. Cơ chế làm việc của SSL*

Khi có một máy tính kết nối với một Website đã được chứng thực.

* Tính linh hoạt và khả năng mở rộng: Việc firebase sử dụng NoSQL giúp cho database của họ sẽ không bị bó buộc trong các bảng và các trường mà người dùng có thể tùy ý xây dựng database theo cấu trúc riêng của người dùng.
* Sự ổn định: Firebase hoạt động dựa trên nền tảng đám mây điện toán đến từ Google vì vậy hầu như người dùng không bao giờ phải lo lắng về việc sập server, tấn công mạng như DDOS, tốc độ kết nối lúc nhanh lúc chậm, …Ngoài ra việc nâng cấp, bảo trì server cũng diễn ra rất đơn giản mà không cần phải dừng sever để nâng cáp như truyền thống.
* Giá thành: Google Firebase có rất nhiều gói dịch vụ với các mức dung lượng lưu trữ như băng thông khác nhau với mức giá dao động từ Free đến $1500 đáp ứng được nhu cầu của tất cả các đối tượng.
  + - ***Cách kết nôi Firebase với Xcode***

1. *Tạo tài khoản*

Người dùng đăng ký một tài khoản. Ứng dụng firebase sẽ tạo cho người dùng một URL là duy nhất trong firebaseio.com. Người dùng sử dụng chính URL vừa đăng ký để lưu trữ và đồng bộ dữ liệu.

|  |
| --- |
| Cài đặt và tìm hiểu về kiểu dữ liệu của FireBase |

*Hình 4.3. Cơ sở dữ liệu Firebase*

Ấn vào một ứng dụng Firebase cho phép người dùng xem và chỉnh sửa dữ liệu của ứng dụng trong realtime. Trong bảng điều khiển người dùng có thể thiết lập chế độ bảo mật và một số các quy tắc cho Firebase.

1. *Cài đặt bằng CocoaPods*

CocoaPods là cách dễ nhất để bắt đầu với Firebase IOS SDK.

Để khởi tạo CocoaPods trong project người dùng chạy lệnh dưới đây:

|  |
| --- |
| $ cd đường-dẫn-đến-project-của-người dùng  $ pod init  $ open –a Xcode Podfile |

Sau khi chạy xong CocoaPods, người dùng thêm Firebase Pod đến Podfile vừa tạo với đoạn mã sau:

|  |
| --- |
| Pod ‘Firebase’. ‘>=2.5.1’ |

Sau đó quay trở lại màn hình terminal chạy đoạn mã sau:

|  |
| --- |
| $ pod install  $ open your-project.xcworkspace |

1. *Cài đặt bằng cách thêm thư viên trực tiếp*

* Bước 1: tạo một ứng dụng trên Xcode.
* Bước 2: Tải FirebaseOSX framework theo đường dẫn:

<https://www.firebase.com/docs/ios/alternate-setup.html>

* Bước 3: Giải nén tập tin vừa tải, sau khi giải nén tập tin có tên là: Firebase.framework
* Bước 4: Kéo tâp tin vừa giải nén kéo vảo ứng dụng đã tạo ở bước 1.
* Bước 5: Ở mục Build Pharse \ Link Binary With Librarise, thêm vào:
* *libicucore.dylib*
* *libc++.dylib*
* *CFNetwork.framework*
* *Security.framework*
* *SystemConfiguration.framework*

|  |
| --- |
| *Snip20160624_27.png* |

*Hình 4.4. Các thư việc sử dụng cho Xcode khi làm việc với firebase*

* Bước 6: Chạy ứng dụng và tiến hành làm việc với Firebase.

1. *Cách sử dụng*

* Để sử dụng firebase người dùng cần import vào tất cả các lớp mà mình sử dụng:

|  |
| --- |
| import Firebase |

* Cấu trúc File JSON trong firebase: không có bản ghi.

|  |
| --- |
| {  "users": {  "techmaster ": {  "address": "Nguyen Dinh Chieu",  //Tất cả các key và value của file JSON ở “categories” được thêm.  "categories": { "one": true, "three": true }  },  "brinchen": { ... },  "hmadi": { ... }  }  } |

* Ở file này thì Swift có thể chuyển về các dạng như sau để lưu trữ:

|  |
| --- |
| *NSString*  *NSNumber*  *NSDictionary*  *NSArray*  *Với kiểu BOOL nó có thể đại diện bời NSNumber numberWithBool:(BOOL)* |

* Các thao tác liên kết giữa Firebase và Swift:

Đầu tiên, tạo tham chiếu đến cơ sở dữ liệu Firbase:

|  |
| --- |
| var myRootRef = Firebase(url:"https://docs-examples.firebaseio.com/") |

* Các method để lấy dữ liệu:

|  |
| --- |
| * ***setValue****: viết dữ liệu đến một path được định nghĩa ....* * ***updateChildValues****: Cập nhật giá trị đã tồn tại trong dữ liệu trong cơ sở dữ liệu của người dùng.* * ***childByAutoId****: Tự động tạo ra một id là duy nhất trong cơ sở dữ liệu. Mỗi lần người dùng gọi childByAutoId cơ sở dữ liệu của người dùng tạo ra một id duy nhất  URL dạng messages/users/<unique-user-id>/<username>* * ***runTransactionBlock****: sử dụng  func này khi làm việc với dữ liệu phức tạp có thể bị ảnh hưởng bởi việc cập nhật đồng thời.* |

Sử dụng setValue:

Tạo một cây dữ liệu có cấu trúc:

|  |
| --- |
| var ref = Firebase(url: "https://docs-examples.firebaseio.com/web/saving-data/fireblog")  var alanisawesome = ["full\_name": "Alan Turing", "date\_of\_birth": "June 23, 1912"]  var gracehop = ["full\_name": "Grace Hopper", "date\_of\_birth": "December 9, 1906"]  var usersRef = ref.childByAppendingPath("users")  var users = ["alanisawesome": alanisawesome, "gracehop": gracehop]  usersRef.setValue(users) |

Kết quả :

|  |
| --- |
| database:  {  "users": {  "alanisawesome": {  "date\_of\_birth": "June 23, 1912",  "full\_name": "Alan Turing"  },  "gracehop": {  "date\_of\_birth": "December 9, 1906",  "full\_name": "Grace Hopper"  }  }  } |

Cập nhật dữ liệu đã lưu

|  |
| --- |
| var hopperRef = usersRef.childByAppendingPath("gracehop")  var nickname = ["nickname": "Amazing Grace"]  hopperRef.updateChildValues(nickname) |

Khi sử dụng từ khóa “**updateChildValues**” thì nó sẽ delete cả full\_name và date\_of\_birth từ “**hopperRef**”

Kết quả:

|  |
| --- |
| "gracehop": {  "date\_of\_birth": "December 9, 1906",  "full\_name": "Grace Hopper",  "nickname": "Amazing Grace"  } |

Lưu danh sách với childByAutoID

Sử dụng từ khóa childByAutoId thì firebase sẽ tự sinh ra một id duy nhất khi 1 nhánh con gọi và thêm vào tham chiếu của database.

|  |
| --- |
| let postRef = ref.childByAppendingPath("posts")  let post1 = ["author": "gracehop", "title": "Announcing COBOL, a New Programming Language"]  let post1Ref = postRef.childByAutoId()  post1Ref.setValue(post1) |

Kết quả :

|  |
| --- |
| "posts": {  "-JRHTHaIs-jNPLXOQivY": {  "author": "gracehop",  "title": "Announcing COBOL, a New Programming Language"  }  } |

Cách lấy dữ liệu từ firebase

|  |
| --- |
| // Url là đường dẫn của database của chúng ta  var ref = Firebase(url:"https://docs-examples.firebaseio.com/web/saving-data/fireblog/posts")  // Chúng ta phải truyền vào Event và một block, block này sẽ được callback khi dữ liệu thay đổi  ref.observeEventType(.Value, withBlock: { snapshot in  print(snapshot.value)  }, withCancelBlock: { error in  print(error.description)  }) |

Nếu người dùng chạy đoạn mã này, người dùng sẽ có được một đối tượng chứa tất cả các bài mà người dùng đã post sau đó in ra console.

Các kiểu sự kiện:

* **.Value**: Sự kiện này được sử dụng để đọc một snapshot của nội dung tại một đường dẫn. Nó được kích hoạt ngay lập tức với dữ liệu được khởi tạo và lặp lại quá trình khởi tạo mỗi khi dữ liệu thay đổi.
* **.ChildAdded**: Sự kiện này được sử dụng khi muốn lấy một danh sách của các item trong Firebase database của người dùng. Không giống như .Value trả về toàn bộ nội dung tại đường dẫn đó, với ChildAdded được kích hoạt ngay lập tức cho mỗi nhánh con và làm lại điều đó khi một nhánh con được thêm vào đường dẫn đó.
* **.ChildChanged**: Sự kiện này được kích hoạt mỗi khi một nhánh con được chỉnh sửa và bao gồm các thành phần ở trong nhánh con đó.
* **.ChildRemove**: Sự kiện này được kích hoạt khi một nhánh con bị xóa.

**Các câu lênh sắp xếp**

Ví dụ để lấy được các con khủng long theo cân nặng

|  |
| --- |
| let ref = Firebase(url:"https://dinosaur-facts.firebaseio.com/dinosaurs")  ref.queryOrderedByChild("height").observeEventType(.ChildAdded, withBlock: { snapshot in  if let height = snapshot.value["height"] as? Double {  print("\(snapshot.key) was \(height) meters tall")  }  }) |

1. *Bảo mật thông tin người dùng*

Các hình thức đăng ký tài khoản cho người dùng:

|  |
| --- |
| **firbaseUser2.PNG** |

*Hình 4.5. Các hình thức đăng ký tài khoản cho người dùng.*

Khi người dùng login ta dùng câu lệnh:

|  |
| --- |
| ref.authUser("jenny@example.com", password: "correcthorsebatterystaple") {  error, authData in  if error != nil {  // an error occured while attempting login  } else {  // user is logged in, check authData for data  }  } |

Khi người dùng đăng xuất

|  |
| --- |
| ref.unauth() |

Khi thêm 1 người dùng mới:

* Bằng cách đăng ký 1 email và password mới. Firebase chủ động kiểm tra xem email này có hợp lệ không, có trùng email không rồi mới cho thêm vào firebase.
* Trong quá trình thêm vào firebase, firebase cung cấp cho người dùng 1 uid để làm khóa được lưu trong authData.
* Sau đó, lấy uid đó làm khóa chính (nút) để lưu thông tin của người dùng ở mục Database.

|  |
| --- |
| ref.authUser("jenny@example.com", password:"correcthorsebatterystaple") {  error, authData in  if error != nil {  // Something went wrong. :(  } else {  // The logged in user's unique identifier  println(authData.uid)  // Create a new user dictionary accessing the user's info  // provided by the authData parameter  let newUser = [  "provider": authData.provider,  "displayName": authData.providerData["displayName"] as? NSString as? String  ]  // Create a child path with a key set to the uid underneath the "users" node  // This creates a URL path like the following:  // - https://<YOUR-FIREBASE-APP>.firebaseio.com/users/<uid>  ref.childByAppendingPath("users")  .childByAppendingPath(authData.uid).setValue(newUser)  }  } |

|  |
| --- |
| {  "users": {  "6d914336-d254-4fdb-8520-68b740e047e4": {  "displayName": "alanisawesome",  "provider": "password"  },  "002a448c-30c0-4b87-a16b-f70dfebe3386": {  "displayName": "gracehop",  "provider": "password"  }  }  } |

1. **Mô hình MVC**
   * + ***Giới thiệu mô hình***

Với mỗi ngôn ngữ đều có các chuẩn chung trong quá trình thiết kế phần mềm, việc đó được gọi là Design patterns. Design pattern giúp người dùng giải quyết nhiều vấn đề khi phát triển phần mềm như: viết code dễ đọc và hiểu, tái sử dụng hoặc có thể thay đổi các thành phần trong code mà không có quá phức tạp.

**Model-View-Controller** (MVC) là một trong những design pattern thường hay sử dụng và nhiều người biết đến nhất. Nó phân loại đối tượng theo những quy tắc chung và khuyến khích chia code theo chức năng của từng phần: Model, view, controller. Trong đó:

* **Model**: Là đối tượng có vai trò định nghĩa, tổ chức lưu trữ và thao tác trực tiếp với dữ liệu.
* **View**: là đối tượng chịu trách nhiệm hiển thị giao diện hay nói cách khác đó là tầng mà người dùng quan sát và tương tác trực tiếp.
* **Controller**: Là bộ điều khiển trung gian có vai trò quản lý, điều phối tất cả các công việc. Nó truy cập dữ liệu từ Model và quyết định hiển thị nó trên View. Khi có một sự kiện được truyền xuống từ View, Controller quyết định cách thức xử lý: Có thể tương tác với dữ liệu, thay đổi giao diện trên view,… Một cách cài đặt tốt nhất theo nguyên tắc design pattern này đó là mỗi một object sẽ rơi vào một trong những thành phần này.

Mô hình:

|  |
| --- |
| Model-View-Controller1 |

*Hình 4.6. Mô hình MVC*

Có thể liên tưởng đến vị trí của từng thành phần trong file của 1 ứng dụng trong swift như sau:

|  |
| --- |
| mvc1.PNG |

*Hình 4.7. Các đối tượng nhận biết của mô hình VMC.*

* Các file có đuôi chấm .storyboard và .xib là các view.
* Các file có tên .Swift là các Controller,
* Các file được tạo từ File Swift có thể là kiểu struct hoặc class là Model.

Các thông tin được truyền giữa các phần như sau:

|  |
| --- |
| mvc2.PNG |

*Hình 4.8. Truyền thông điệp giữa các phần trong MVC*

* View sẽ truyền các tác động của người dùng vào controller như: yêu cầu hiển thị dữ liệu, tác động các control như là button hoặc textfile, …
* Controller nhận các yêu cầu đấy sau đó đưa ra các hình thức xử lý đồng thời lấy thông tin và các thuộc tính từ Model lên làm việc, khi xử lý xong sẽ trả ra kết quả lên các Outlet (Label, textbox) của View
* Model chứa các thông báo, các thuộc tính của đối tượng, đồng thời là nơi lấy các dữ liệu từ xa.
  + - ***Lợi ích của mô hình***
* Giúp ứng dụng dễ bảo trì
* Dễ dàng thêm các tính năng mới hoặc các tính năng cũ có thể dễ dàng thay đổi.
* Có thể thay đổi 1 trong 3 phần mà không làm ảnh hưởng đến các phần còn lại.
* Dễ phân công làm việc theo từng chắc năng của ứng dụng khi làm việc nhóm.

**Tóm tắt phần 1:** phần 1 bao gồm các nội dung lớn: giới thiệu hệ điều hành iOS, giới thiệu công cụ lập trình Xcode và giới thiệu ngôn ngữ lập trình Swift, cách kết nối cơ sở dữ liệu đám mây điện toán - Firebase. Trong đó đi sâu tìm hiểu các cú pháp của ngôn ngữ Swift. Đây là phần quan trọng nhất của phần. Người dùng cần nắm rõ những đặc điểm tính chất về cách tổ chức các hằng, các biến, các kiểu một cách hợp lý để xây dựng ứng dụng một cách tốt nhất. Bên cạnh những mặt cơ bản, người dùng có thể kết hợp với một số mô hình tổ chức code như MVC để người dùng dễ dàng kiểm tra và sửa lỗi cũng như cải tiến chương trình.

**CHƯƠNG 5: GIỚI THIỆU BÀI TOÁN HỖ TRỢ TÌM ĐƯỜNG ĐI NGẮN NHẤT GIỮA NHIỀU CẶP ĐIỂM**

1. **Đặt vấn đề**

Ngày nay thương mại điện tử phát triển rất nhanh, các cửa hàng online mọc lên rất nhiều thông qua Facebook, Zalo, Instagram,… đặc thù của bán hàng online là việc giao hàng và nhận hàng tại nhà, không mất thời gian đi lại. Việc giao hàng giữ vai trò rất quan trọng đối với những Shop có mô hình kinh doanh nhỏ lẻ, đơn hàng ít, chi phí để thuê 1 Shipper ruột để vận chuyển riêng rất là cao. Nhiều khi những Shop có mô hình kinh doanh lớn đội ngũ Shipper của họ không đáp ứng đủ nhu cầu vận chuyển và họ phải tìm thêm nguồn nhân lực từ bên ngoài. Các công ty trung gian thường bắt Shipper cọc tiền mởi được trở thành Shipper cho công ty nên nhiều khi những bạn muốn làm Shipper để kiếm thêm ít chi phí nhưng không có vốn cũng chẳng thể nào làm được, giảm đi 1 số lượng Shipper tiềm ẩn cho thị trường.

1. **Giải quyết vấn đề**

Để giải quyết nhu cầu việc làm cho Shipper và nhu cầu tìm Shipper của các chủ Shop.Nhóm em sẽ tạo ra 1 cầu nối để kết nối chủ shop và Shipper lại với nhau trên nền tảng iOS để đáp ứng nhu cầu Shipper tìm việc và Shop tìm Shipper.

Hệ thống sẽ được chia thành 2 App:

* App dành cho các chủ Shop đăng bài tìm Shipper
* App dành cho các Shipper nhận bài từ các Shop, xem đường đi, ***đặc biệt App sẽ gợi ý cho Shipper đường đi giao hàng trong ngày ít tốn chi phí nhất***

**CHƯƠNG 6: THIẾT KẾ & XÂY DỰNG HỆ THỐNG**

1. **Mô hình UML**
   1. **Mô hình Usecase**

|  |
| --- |
|  |

*Hình 1.1. Mô hình user case tổng quát*

* 1. **Đặc tả Usecase**
     1. **Danh sách các Usecase**

|  |  |  |  |
| --- | --- | --- | --- |
| Số Thứ Tự | Code | Tên Usecase | Mô Tả |
| 1 | UC01 | Đăng ký thành viên | Shop và Shipper muốn sử dụng App phải đăng kí thành viên. |
| 2 | UC02 | Đăng nhập | Shipper/Shop khi đăng ký đã trở thành viên. Lần đầu tiên mở App cần phải đăng nhập mới vào được hệ thống, các lần sau không cần đăng nhập lại. |
| 3 | UC03 | Đăng bài | Shop khi đăng nhập thành công vào hệ thống có thể đăng bài để kiếm Shipper giao hàng cho sản phẩm đã đăng. |
| 4 | UC04 | Xem lịch sử bài đăng | Shop có thể xem lại tất cả các bài mà Shop đã đăng trước đó, Shop có thể biết được tình trạng Sản phẩm mình có Shipper nào nhận và giao thành công hay chưa. |
| 5 | UC05 | Quản lý tài khoản | Shop/Shipper có thể xem lại các thông tin của mình khi đăng kí trước đó. |
| 6 | UC06 | Xem danh sách các sản phẩm của các Shop | Shipper xem các bài đăng của các Shop trong khu vực gần mình để lựa và chọn nên nhận hàng hay không. |
| 7 | UC07 | Nhận bài đăng | Sau khi xem danh sách các sản phẩm của các Shop, Shipper nhấn vào từng bài và chọn nhận đơn hàng, đơn hàng này sẽ không hiển thị ra cho các Shipper khác nhận cho tới khi Shipper này chủ động huỷ đơn hàng/ sau 24h nhận hàng nhưng chưa giao thành công. |
| 8 | UC08 | Xem lưu trữ các bài đã nhận | Shipper sau khi nhận 1 loại bài đăng của các Shop, các bài đăng này được hiển thị hết vào mục Lưu trữ. |
| 9 | UC09 | Xem đường đi ngắn nhất | Shipper sau khi xem lưu trữ những đơn hàng chưa giao, Shipper chọn tìm đường đi ngắn nhất App sẽ gợi ý nên giao và nhận hàng ở những địa điểm nào trên bản đồ mà chi phí tiền xăng Shipper bỏ ra là ít nhất. |
| 10 | UC10 | Xem lịch sử các bài đã nhận | Shipper có thể xem lại tất cả các bài đã nhận từ các Shop, xem đơn hàng nào đã phát thành công hay chưa phát. |

* + 1. **Đặc tả**

***UC01: Đăng ký thành viên***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Đăng ký thành viên | **Code** | UC01 |
| **Description** | Shipper/Shop có thể đăng ký để có một tài khoản để được sử dụng App | | |
| **Actor** | Shop/Shipper | **Trigger** |  |
| **Pre-condition** | Shipper/Shop chưa có tài khoản trước đó trong hệ thống | | |
| **Post condition** |  | | |

***UC02: Đăng nhập***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Đăng nhập | **Code** | UC02 |
| **Description** | Shipper/Shop khi đăng ký đã trở thành viên. Thành viên có thể đăng nhập vào App để sử dụng App | | |
| **Actor** | Shop/Shipper | **Trigger** |  |
| **Pre-condition** | * Người dùng mở App, hệ thống sẽ chuyển tới trang Đăng nhập. * Nhập thông tin tài khoản, mật khẩu | | |
| **Post condition** |  | | |

***UC03: Đăng bài***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Đăng bài | **Code** | UC03 |
| **Description** | Shop khi đăng nhập thành công vào hệ thống có thể đăng bài để kiếm Shipper giao hàng cho sản phẩm đã đăng. | | |
| **Actor** | Shop | **Trigger** | Click Create Item |
| **Pre-condition** | * Shop click vào Tabar Item Tạo mới * Chọn tạo bài viết * Điền đầy đủ điểm nhận, điểm giao sẽ chuyển tới trang điền chi tiết đơn hàng. * Nhấn nút đăng bài sau khi điền đầy đủ các thông tin trong Form. | | |
| **Post condition** |  | | |

***UC04: Xem lịch sử bài đăng***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Xem lịch sử bài đăng | **Code** | UC04 |
| **Description** | Shop có thể xem lại tất cả các bài mà Shop đã đăng trước đó, Shop có thể biết được tình trạng Sản phẩm mình có Shipper nào nhận và giao thành công hay chưa. | | |
| **Actor** | Shop | **Trigger** |  |
| **Pre-condition** | * Shop click vào Tabar Item Lưu trữ * Hiển thị danh sách các bài đăng của Shop | | |
| **Post condition** |  | | |

***UC05: Quản lý tài khoản***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Quản lý tài khoản | **Code** | UC05 |
| **Description** | Shop/Shipper có thể xem lại các thông tin của mình khi đăng kí trước đó. | | |
| **Actor** | Shipper/Shop | **Trigger** |  |
| **Pre-condition** | * Shop click vào Tabar Item Account * Hiển thị tất cả thông tin của user đã đăng kí trước đó | | |
| **Post condition** |  | | |

***UC06: Xem danh sách bài đăng của các Shop***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Xem danh sách bài đăng của các Shop | **Code** | UC06 |
| **Description** | Shipper xem các bài đăng của các Shop trong khu vực gần mình để lựa và chọn nên nhận hàng hay không. | | |
| **Actor** | Shipper | **Trigger** |  |
| **Pre-condition** | * Shipper click vào Tabar Item News * Hiển thị tất cả danh sách đơn hàng gần với Shipper | | |
| **Post condition** |  | | |

***UC07: Nhận bài đăng***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Nhận bài đăng | **Code** | UC07 |
| **Description** | Sau khi xem danh sách các sản phẩm của các Shop, Shipper nhấn vào từng bài và chọn nhận đơn hàng, đơn hàng này sẽ không hiển thị ra cho các Shipper khác nhận cho tới khi Shipper này chủ động huỷ đơn hàng/ sau 24h nhận hàng nhưng chưa giao thành công. | | |
| **Actor** | Shipper | **Trigger** | Click vào Cell cần nhận hàng |
| **Pre-condition** | * Shipper click vào Cell sản phẩm của Shop * Chuyển tới Storyboard Chi tiết bài đăng của Shop đó * Nhấn nhận đơn hàng * Chuyển Shipper tới màn hình Storage | | |
| **Post condition** |  | | |

***UC08: Xem các đơn hàng đã nhận chưa giao.***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Xem các đơn hàng đã nhận chưa giao | **Code** | UC08 |
| **Description** | Shipper sau khi nhận 1 loại bài đăng của các Shop, các bài đăng này được hiển thị hết vào mục Lưu trữ. | | |
| **Actor** | Shipper | **Trigger** |  |
| **Pre-condition** | * Shop click vào Tabar Item Storage * Hiển thị tất cả các đơn hàng chưa giao. | | |
| **Post condition** |  | | |

***UC09: Xem đường đi ngắn nhất***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Xem đường đi ngắn nhấn | **Code** | UC09 |
| **Description** | Shipper sau khi xem lưu trữ những đơn hàng chưa giao, Shipper chọn tìm đường đi ngắn nhất App sẽ gợi ý nên giao và nhận hàng ở những địa điểm nào trên bản đồ mà chi phí tiền xăng Shipper bỏ ra là ít nhất. | | |
| **Actor** | Shipper | **Trigger** |  |
| **Pre-condition** | * Shop click vào Tabar Item Tạo mới * Chọn tạo bài viết * Điền đầy đủ điểm nhận, điểm giao sẽ chuyển tới trang điền chi tiết đơn hàng. * Nhấn nút đăng bài sau khi điền đầy đủ các thông tin trong Form. | | |
| **Post condition** |  | | |

***UC10: Xem lịch sử bài nhận***

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | Xem lịch sử bài nhận | **Code** | UC10 |
| **Description** | Shipper có thể xem lại tất cả các bài đã nhận từ các Shop, xem đơn hàng nào đã phát thành công hay chưa phát. | | |
| **Actor** | Shipper | **Trigger** |  |
| **Pre-condition** | * Shop click vào Tabar Item Lưu trữ * Hiển thị danh sách các bài đăng của Shop | | |
| **Post condition** |  | | |

* 1. **Sequence diagram**
  2. *Sơ đồ tuần tự cho quá trình Đăng nhập*

|  |
| --- |
|  |

*Hình 1.10. Quá trình Đăng nhập*

* 1. *Sơ đồ tuần tự cho quá trình Đăng kí*

|  |
| --- |
|  |

*Hình 1.11. Quá trình Đăng kí*

* 1. *Sơ đồ tuần tự cho quá trình Đăng bài, Xem lịch Sử, Quản lý tài khoản của chủ Shop*

|  |
| --- |
|  |

*Hình 1.12. Quá trình Đăng bài, Xem lịch Sử, Quản lý tài khoản của chủ Shop*

* 1. *Sơ đồ tuần tự cho quá trình Xem list bài đăng, Nhận bài đăng, Xem kho lưu trữ, Xem đường đi ngắn nhất và Thông tin của Shipper*

|  |
| --- |
|  |

*Hình 1.13. Quá trình Xem list bài đăng, Nhận bài đăng, Xem kho lưu trữ, Xem đường đi ngắn nhất và Thông tin của Shipper*

* 1. **Activity Diagram:**

**Chức năng đăng nhập:**

|  |
| --- |
|  |

**Chức năng đăng kí:**

|  |
| --- |
|  |

**Chức năng Xem danh sách các đơn hàng:**

|  |
| --- |
|  |

**Chức năng nhận bài**

|  |
| --- |
|  |

**Chức năng xem đường đi**

|  |
| --- |
|  |

**Xem thông tin tài khoản**

|  |
| --- |
|  |

**Đổi thông tin tài khoản**

|  |
| --- |
|  |

**Chức năng đăng bài**

|  |
| --- |
|  |

1. **Thiết kế cơ sở dữ liệu (No SQL)**

Cây thư mục tổng quát:

|  |
| --- |
|  |

*Hình 1.14. Cây thư mục tổng quát của sơ sở dữ liệu.*

Dữ liệu được lưu những đối tượng chính như: Loại tài khoản, Tài khoản, Người nhận, Toạ độ, Bài viết, Chi tiết bài viết, sản phẩm của chi tiết bài viết.

* 1. **Loại tài khoản**
* Có 2 loại tài khoản được quy định 1 là Shop, 2 là Shipper

|  |
| --- |
|  |

* 1. **Tài khoản**
* Tài khoản gồm các thông tin: tên người dùng, hình đại diện, số điện thoại người dùng, chứng minh thư,…

|  |
| --- |
|  |

* 1. **Thông tin khách hàng**
* Khách hàng nhận hàng gồm có 2 thông tin là tên và số điện thoại, thông tin này sẽ được chủ Shop cung cấp

|  |
| --- |
|  |

* 1. **Toạ độ**
* Mỗi toạ độ gồm các thuộc tính: kinh độ, vĩ độ, tên địa điểm

|  |
| --- |
|  |

* 1. **Bài đăng**
* Thông tin bài đăng bao gồm: mã người đăng, mã chi tiết bài đăng, thời gian đăng bài.

|  |
| --- |
|  |

* 1. **Chi tiết bài đăng**
* Thông tin chi tiết bài đăng gồm: mã điểm nhận, mã điểm giao, mã Shipper nhận bài, mã sản phẩm, mã khách hàng, tiền ứng, tiền công của Shipper.

|  |
| --- |
|  |

* 1. **Sản phẩm chi tiết bài đăng**
* Thông tin sản phẩm bao gồm: hình sản phẩm, khối lượng, tiêu đề, nội dung.

|  |
| --- |
|  |

* 1. **Bảo mật đăng nhập người dùng**
* Nơi lưu trữ email và password mà người dùng đăng ký

|  |
| --- |
|  |

1. **Thiết kế giao diện**
   1. **Đối với chủ Shop**
      1. **Đăng nhập**

* Khi mở app người dùng phải đăng nhập mới vào được bên trong hệ thống, để đăng nhập người dùng cần điền email và mật khẩu đã đăng kí trước đó

|  |
| --- |
|  |

* + 1. **Đăng kí**
* Khi đăng kí tài khoản đối với chủ Shop cần nhập email, mật khẩu và 1 số thông tin cá nhân như họ tên, số điện thoại, chứng minh thư và hình đại diện.

|  |
| --- |
|  |

* + 1. **Màn hình chính**
* Lần đầu đăng nhập thành công vào App sẽ yêu cầu cung cấp quyền để App lấy location của người dùng

|  |
| --- |
|  |

* Hiển thị location trên bản đồ

|  |
| --- |
|  |

* Hiển thị location qua góc nhìn vệ tinh

|  |
| --- |
|  |

* + 1. **Đăng bài**
* Tạo bài viết

|  |
| --- |
|  |

* Nhập thông tin cho điểm giao, điểm nhận

|  |
| --- |
|  |

|  |
| --- |
|  |

* Sau khi nhập đủ điểm giao, điểm nhận sẽ nút đăng bài sẽ được bật lên để nguời dùng điền chi tiết bài

|  |
| --- |
|  |

* Điền đầy đủ thông tin vào Form chi tiết bài đăng và đăng bài viết

|  |
| --- |
|  |

* + 1. **Quản lý bài đăng**
* Tất cả các bài Shop đã đăng được lưu vào mục Lưu trữ, tại đây Shop có thể xem được tình trạng của đơn hàng đã được Shipper nào book.

|  |
| --- |
|  |

* + 1. **Quản lý tài khoản**
* Thông tin tài khoản gồm: họ tên, email, mật khẩu, chứng minh thư,….Người dùng có thể xem và đổi thông tin ở màn hình này

|  |
| --- |
|  |

* 1. **Đối với Shipper**
     1. **Đăng nhập**
* Người dùng nhấn vào button Sign in và cho phép truy cập vào google sign in để vào hệ thống.

|  |
| --- |
|  |

* Người dùng có 2 tuỳ chọn là sử dụng tài khoản đã qua đăng nhập rồi hoặc chọn tài khoản khác để đăng nhập vào hệ thống.

|  |
| --- |
|  |

* + 1. **Đăng kí**
* Người dùng nhập đầy đủ thông tin chọn thành phố để giao hàng, nhập họ tên, số điện thoại, email, mật khẩu và chứng minh nhân dân để đăng ký tài khoản.

|  |
| --- |
|  |

* + 1. **Xem danh sách bài đăng**
* Hiển thị thông tin thời gian đăng hàng, tiền ứng, tiền giao hàng, địa chỉ nhận hàng, địa chỉ giao hàng, người đăng của những shop đã đăng bài.

|  |
| --- |
|  |

* + 1. **Nhận bài đăng**
* Hiển thị chi tiết thời gian đăng bài, họ tên người đăng, số điện thoại, nơi giao hàng, nơi nhận hàng, khối lượng sản phẩm, tiền ứng, tiền giao hàng, hình ảnh của sản phẩm của một đơn hàng

|  |
| --- |
|  |

* + 1. **Quản lý bài nhận**
* Hiển thị thời gian đăng bài, tiền ứng, tiền giao hàng, nơi nhận hàng, nơi giao hàng và họ tên người nhận của những đơn hàng mình đã nhận.

|  |
| --- |
|  |

* + 1. **Xem đường đi**
* Hiển thị đường đi ngắn nhất để người giao hàng nhận hàng và giao hàng ít tốn kém chi phí thời gian và nhiên liệu nhất.

|  |
| --- |
|  |

* + 1. **Quản lý tài khoản**
* Hiển thị thông tin cần quản lý của người giao hàng như họ tên người dùng, hình ảnh hiển thị, số điện thoại, chứng minh nhân dân.

|  |
| --- |
|  |

1. **Hiện thực ứng dụng**

* Cơ sở dữ liệu: FirebaseDatabase.
* Ngôn ngữ lập trình: Swift 4.2.
* Các thư viện hỗ trợ: UIkit, Mapkit, FirebaseAuth, FirebaseStorage, CoreLocation.
* Mô hình: MVC.
* Các Design Pattern sử dụng trong App: SINGLETON DESIGN PATTERN, FACTORY DESIGN PATTERN.

**CHƯƠNG 7: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

Bằng những kiến thức ***nền tảng*** được học ở trường và tự tìm hiểu qua sách, Internet nhóm đã hoàn thành Tiểu luận chuyên ngành. Tuy nhiên, do những khó khăn nhất định về thời gian, thiết bị hỗ trợ và những trở ngại trong quá trình cài đặt nên bên cạnh những mặt làm được thì đồ án cũng còn tồn tại một vài thiếu sót nhỏ. Dưới đây là phần tự nhận xét và đánh giá của nhóm đối với đồ án đã làm.

1. **Tổng kết kết quả đạt được:**

* Hoàn thành được những mặt cơ bản về lý thuyết tương đối đầy đủ.
* Tự tìm hiểu và sử dụng được ngôn ngữ lập trình Swift.
* Bài báo cáo có sự phân chia các mục rõ ràng, trình bày từng bước giúp người đọc dễ hiểu và tưởng tượng khi có kèm theo các ví dụ minh họa
* Tìm hiểu thêm một số công nghệ hỗ trợ về lưu trữ và kỹ thuật lập trình tiện ích hơn: firebase, mô hình MVC.
* Sử dụng được những chế độ tự động trình chiếu ở các màn hình iPhone, iPad.
* Ứng dụng tạo được cầu nối cho Shipper và Shop, gợi ý được đường đi ngắn nhất cho Shipper.
* Ngoài ra, ứng dụng chọn Firebase để lưu trữ nên mang tính bảo mật cao về cơ sở dữ liệu và thông tin người dùng.
* Ứng dụng thành công mô hình **MVC** làm cho code của chương trình dễ phân chia chức năng, dễ kiểm soát và sửa lỗi

1. **Hạn chế**

* Tốc độ dữ liệu tải về phụ thuộc vào Internet.
* Chưa có quản trị ứng dụng.
* Do phải đi thực tập bên công ty nên thời gian dành cho Tiểu luận không được nhiều nên còn 1 số tính năng chưa được hoàn thiện
* Giao diện không được đẹp và thân thiện với người dùng vì nhóm không có Design riêng.
* Dữ liệu chỉ ở mức của ứng dụng mô phỏng.
* Ứng dụng chỉ hoạt động khi có mạng Internet
* Code chưa được **Clean.**

1. **Hướng phát triển**

Vì ứng dụng ở mức mô phỏng nên dữ liệu lưu trữ ít, và xây dựng những chức năng chính nên cần hoàn thiện và khắc phục thêm ở những mặt sau:

*Về mặt lý thuyết:*

* Tìm hiểu cách tạo Webservice, cách kết nối giữa firebase và ngôn ngữ PHP để phục vụ cho quá trình quản lý dữ liệu.
* Nghiên cứu cách tạo hiệu ứng cho các control
* Đi sâu tìm hiểu từng hàm chức năng nâng cao của các control TableView, CollectionView, SearchBar Controller, …

*Về mặt ứng dụng:*

* Chọn hình thức lưu trữ dữ liệu mới và cách lấy dữ liệu ảnh để hiển thị nhanh nhất có thể.
* Tạo thêm nhiều hình thức chia sẽ, kết nối trên mạng xã hội.
* Tạo quản trị ứng dụng để dễ dàng thao tác trong việc thêm, sửa, xóa dữ liệu.
* Phân quyền chức năng ứng dụng khi có quản trị
* Tạo hiệu ứng cho chương trình được sinh động và đẹp mắt hơn
* Áp dụng những thuật toán tìm kiếm nhanh và hữu hiệu đối với bài viết trên nền dữ liệu lớn.
* Nâng cấp mô hình lên VIPER.

**TÀI LIỆU THAM KHẢO**

**[1].** Gary Bennett, Brad Lees, Swift for Absolute Beginners, Apress, 11/2018.

**[2].** David Mark, Jack Nutting, Kim Topley, Fredrik Olsson, Jeff LaMarche, Beginning iPhone Development Exploring the iOS SDK, Apress, ISBN: 978-1-484202-00-5, tái bản lần thứ 7 vào năm 2014.

**[3].** Matthew Campbell, Swift Quick Syntax Reference, Apress, 2018.

**[4].** https://www.firebase.com

**[5].** <https://developer.apple.com>