BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC HOA SEN

KHOA KHOA HỌC VÀ CÔNG NGHỆ

KHÓA LUẬN TỐT NGHIỆP

Tên đề tài: Quản lý nhà hàng không dây

Giảng viên hướng dẫn : Ths Nguyễn Kim Long

Nhóm sinh viên thực hiện: Lê Đức Huy - 070302

Hà Bảo Ngọc - 070729

Phạm Duy Nguyên - 070200

Lớp : QL071

Tháng 12 /năm 2010

Trích yếu

iOS là một nền tảng di động ra mắt sớm và có tốc độ phát triển mạnh mẽ thật sự là một đề tài hấp dẫn với các nhà phát triển ứng dụng. Với mong muốn tìm hiểu và vận dụng những điểm mạnh của nền tảng iOS nhóm chúng tôi đã được khoa Khoa học và Công nghệ đồng ý cho phép thực hiện đề tài "**Hệ thống quản lý phục vụ nhà hàng không dây**".

Qua việc thực hiện đề tài trên, nhóm chúng tôi đã có thể hiểu được tổng quan hệ điều hành iOS, mô hình tương tác, các công nghệ được hỗ trợ và các framework tương ứng. Và bằng những kiến thức đã tìm hiểu được chúng tôi đã có những so sánh ở mức tổng quan nền tảng iOS và các nền tảng di động phổ biến khác. Nhóm chúng tôi đã nắm được cách thức phát triển ứng dụng cho HĐH iOS bằng ngôn ngữ lập trình Objective-C, hiểu được một số design pattern dùng để nâng cao hiệu suất, tăng khả năng tái sử dụng cũng như làm tăng tính dễ hiểu của ứng dụng. Qua đó vận dụng những kiến thức tìm hiểu được để xây dựng giải pháp iQsine giúp quản lý phục vụ nhà hàng. Ngoài ra nhóm chúng tôi đã tích lũy, xây dựng được một bộ thư viện gồm một tập các lớp (class) được định nghĩa bằng Objective-C được sử dụng trong ứng dụng iQsine vàcó thể tái sử dụng được trong các ứng dụng tương lai. Cả quá trình thực hiện khóa luận cũng như những kiến thức mà nhóm chúng tôi đã tích lũy đều được thể hiện trong cuốn báo cáo này.

Bố cục báo cáo khóa luận

Nội dung cuốn báo cáo khóa luận được chia thành năm phần chính như sau:

Phần 1: Tổng quan đề tài, giới thiệu chung về đề tài, ý nghĩa và mục tiêu của đề tài, trình bày các vấn đề nghiên cứu, nội dung nghiên cứu của đề tài và tóm tắt kết quả đạt được.

Phần 2: Tổng quan về hệ điều hành iOS, các công nghệ hỗ trợ cũng như các tính năng, đặc trưng của thiết bị sử dụng hệ điều hành iOS.

Phần 3: Vấn đề phát triển ứng dụng trên nền tảng iOS. Phần này giới thiệu tổng quan về bộ công cụ phát triển ứng dụng cho HĐH iOS, các design-pattern phổ biến cũng như các đặc trưng của việc phát triển ứng dụng cho HĐH iOS.

Phần 4:Ứng dụng hệ điều hành iOS xây dựng giải pháp iQsine. Phần này giới thiệu việc vận dụng các kiến thức đạt được trong quá trình nghiên cứu để xây dựng giải pháp iQsine.

Phần 5: Tổng kết và đánh giá. Phần này đúc kết lại những kết quả mà đề tài đã đạt được và hướng phát triển trong tương lai.

Phần phụ lục

Thuật ngữ chuyên môn

Tài liệu tham khảo

Mục lục

Danh mục hình ảnh

Danh mục bảng

Lời cám ơn

Chúng em xin chân thành cám ơn khoa Khoa học và Công nghệ trường đại học Hoa Sen đã tạo điều kiện cho chúng em thực hiện đề tài tốt nghiệp này.Chúng em xin cám ơn thầy Nguyễn Kin Long đã tận tình hướng dẫn và giúp đở nhóm hoàn thành đề tài.Chúng em cũng xin cám ơn các thầy cô của khoa đã sẵn sàng trả lời mọi thắc mắccủa chúng em về vấn đề học thuật, cũng như các bạn sinh viên đã góp tay hỗ trợ chúng em trong suốt thời gian vừa qua.Mặc dù cố gắng hoàn thành đề tài trên với tất cả nỗ lực của bản thân nhưng chắc chắn không thể tránh khỏi những thiếu sót nhất định, kính mong sự chỉ dạy tận tình của quý thầy cô.

Thành phố Hồ Chí Minh 12/2010

Nhóm thực hiện

Lê Đức Huy

Hà Bảo Ngọc

Phạm Duy Nguyên

Nhận xét của giáo viên hướng dẫn

Nhóm đã hoàn tất khóa luận với nội dung đầy đủ, rõ ràng, đi sát với yêu cầu đề tài nêu ra. Tuy ban đầu có nhiều khó khăn do phải làm quen với một nền tảng công nghệ hoàn tòan mới, một môi trường lập trình mới với nhiều hạn chế về thiết bị, cũng như gặp rất nhiều những lỗi lặt vặt, những tình huống tưởng như đơn giản nhưng lại mất rất nhiều thời gian để tìm hiểu và sửa lỗi. Nhóm đã cố gắng rất nhiều để vượt qua các trở ngại đó.

Trong quá trình tìm hiểu lý thuyết hệ điều hành iOS, nhóm đã đưa ra được nhiều giải pháp cho việc xây dựng một ứng dụng trên nền tảng iOS, vừa để làm cẩm nang cho công việc tương lai, vừa đem lại cho bộ môn tài liệu nghiên cứu hữu ích.

Song song với ứng dụng iQsine, Nhóm đã đưa ra các giải pháp sát với thực tế đáp ứng được yêu cầu mà người dùng mong đợi, nghiên cứu các giải pháp mang tính "framework" cho lập trình ứng dụng trên iOS để các ứng dụng khác có thể kế thừa. mặc dù sản phẩm chưa thật sự hòan chỉnh như mong muốn do hạn chế về thời gian nhưng nhóm cũng đã hòan thành những mục tiêu mà đề tài yêu cầu.

Lời mở đầu

Kể từ lần đầu tiên xuất hiện vào năm 1973, chiếc điện thoại di động đã trởthành một phần không thể thiếu của người dân thuộc mọi tầng lớp. Với tất cả mọi tầng lớp, lứa tuổi, chiếc máy bé nhỏ ấy đã đi sâu vào đời sống hằng ngày. Những chiếc điện thoại di động cũng không chỉ còn để tán gẫu, gửi tin nhắn nữa mà đang từng ngày một mang trong mình thêm những chức năng mới mẻ hơn, "thông minh" hơn. Thuật ngữ "**Smartphone**" từ đó đã xuất hiện.

Đón đầu xu hướng phát triển của smartphone, được sự ủng hộ và giúp đỡ của **Khoa Khoa học và Công nghệ-Đại học Hoa Sen**, nhóm chúng tôi đã thực hiện đề tài **"Phát triển ứng dụng trên nền tảng Android"**. Qua việc thực hiện đề tài trên nhóm đã tích lũy được những kiến thức căn bản về nền tảng di động Android.Một nền tảng di động mở có tốc độ tăng trưởng mạnh mẽ.Bằng cách vận dụng những kiến thức đạt được trong quá trình nghiên cứu để phát triển ứng dụng **aStudent** như là một thực nghiệm những gì mà nhóm đã tìm hiểu được.

Tuy nhiên để có được một cái nhìn tổng quan về thị trường smartphone, thực trạng và xu hướng phát triển của thị trường smartphone hiện nay nhóm đã được sự ủng hộ của khoa tiếp tục nghiên cứu công nghệ di động bằng cách thực hiện đề tài **"Nghiên cứu nền tảng và phát triển ứng dụng thực nghiệm trên iOS"**, một trong những nền tảng di động phổ biến và phát triển nhất hiện nay.

1 Giới thiệu lĩnh vực và ý nghĩa của đề tài

Khi nhóm tiến hành thực hiện đề án chuyên ngành A với đề tài "**Phát triển ứng dụng trên nền tảng Android**" sự phát triển của thị trường smartphone đang từng ngày một rõ nét hơn. Và tính đến thời điểm hiện tại đã xuất hiện rõ nét những cuộc chạy đua dành miếng bánh thị trường smartphone đang ngày một lan rộng. Hàng loạt các đại gia trong lĩnh vực tin học trên thế giới lần lượt tham gia cuộc đua bằng cách lần lượt tung ra các nền tảng di động của riêng mình như: Bada, Megoo, Windows phone 7…. Tuy nhiên, giữa những nền tảng trên nổi bật lên hai nền tảng có tốc độ phát triển rất mạnh mẽ trong năm 2009 là **Android** và **iOS**. Cả hai nền tảng trên đều có những tính năng, những đặc trưngcủa riêng mình. Ra đời trước và nắm giữ những công nghệ độc quyền, **iOS** thật sự tạo ra một cuộc cách mạng vĩ đại với các các thiết bị sử dụng nền tảng này.**Apple Inc** đã rất thành công khi đưa ranhững trải nghiệm người dùng hoàn toàn mới và nhanh chóng thống lĩnh thị trường smartphone ngay khi vừa ra đời. Tuy ra đời sau nhưng nền tảng**Android** với sức mạnh là một hệ điều hành di động mã nguồn mở được xây dựng không chỉ dành cho các nhà phát triển ứng dụng mà cả những nhà phát triển hệ thống đã tạo nên một cuộc rượt đuổi ngoạn mục với gã khổng lồ **iOS**. Cả hai nền tảng tạo thành hai thái cực thống trị thị trường smartphone và đang dần tấn công ra các thị trường thiết bị di động khác.

Ngay từ những ngày đầu mới ra đời, những tích năng và đặc trưng của nền tảng **iOS** đã giúp **Apple Inc** nắm được một lượng thị phần smartphone lớn và có tốc độ phát triển mạnh mẽ nhất hiện nay. Ngoài ra nó còn kéo theo cả một thị trường ứng dụng cho nền tảng iOS có tốc độ phát triển chóng mặt. Chỉ trong bốn năm (2007-Cuối năm 2010) thị trường ứng dụng cho nền tảng iOS đã nắm giữ con số gần 300.000 ứng dụng và mỗi năm đem về cho Apple Inc khoản lợi nhuận gần 500 triệu đô la(Số liệu 2009). Hơn thế nữa, thị trường ứng dụng cho nền tảng iOS vẫn tiếp tục phát triển mạnh mẽ, khai thác các tính năng mới được bổ sung cho các nền tảng iOS mới cũng như các thiết bị **iDevice**(Thiết bị di động sử dụng iOS) mới ra đời.Chính vì những thế mạnh trên mà nền tảng iOS trở thành một đề tài đầy hấp dẫn và tiềm năng dành cho các nhà phát triển.

Với mong muốn có một cái nhìn tổng quan, có sự so sánh với các hệ điều hành di động hiện tại để có thể cung cấp một khả năng đánh giá nhu cầu, tính năng công nghệ để lựa chọn một nền tảng phù hợp với nhu cầu khi mong muốn phát triển một ứng dụ cho nền tảng di động. Chính vì những lý do trên mà nhóm đã quyết định thực hiện đề tài khóa luận tốt nghiệp trên nền tảng iOS với các mục tiêu sau:

- Tìm hiểu kiến trúc tổng quan HĐH iOS, có một sự so sánh ở mức tổng quan với các nền tảng di động phổ biến.
- Tìm hiểu các công nghệ được hỗ trợ bởi iOS.
- Tìm hiểu các đặc trưng, tính năng của thiết bị sử dụng iOS.
- Tìm hiểu cách thức, quy trình xây dựng ứng dụng cho nền tảng di động nói chung và nền tảng iOS nói riêng.
- Xây dựng giải pháp iQsine giúp quản lý phục vụ nhà hàng.

2 Tổng quan lý thuyết hệ điều hành iOS

2.1 Lịch sử và thị trường

2.1.1 Lịch sử ra mắt của hệ điều hành iOS

Ngày 9 tháng 1 năm 2007, Steve Jobs, CEO của Apple Inc, ra mắt chiếc điện thoại thông minh **iPhone** tại hội nghị Macworld (Macworld Conference & Expo). Sử dụng giao diện người dùng hoàn toàn dựa trên công nghệ cảm ứng chạm đa điểm, chiếc iPhone đầu tiên đã gây ấn tượng mạnh ở cả người dùng cuối lẫn các nhà sản xuất phần mềm máy tính. Tuy nhiên, bấy giờ Steve Jobs gần như không hề đề cập gì đến hệ điều hành được dùng trên iPhone, mà chỉ nói ngắn gọn rằng "iPhone sử dụng OS X", tức ngầm ý Mac OS X - hệ điều hành của các máy tính Apple Inc.

Khi được đưa ra thị trường vào tháng 6/2007, iPhone không hề hỗ trợ việc phát triển hay cài đặt phần mềm của các hãng thứ 3. Steve Jobs mong muốn iPhone sẽ là một chiếc điện thoại Internet thuần túy, và tin rằng các ứng dụng web sẽ dần được xây dựng để có thể "hoạt động hệt như các ứng dụng thuần trên iPhone". Tuy vậy, đến ngày 17/10/2007, Apple Inc công bố sẽ sớm phát hành bộ SDK phát triển ứng dụng cho iPhone vào tháng 2 năm sau. Vào ngày 6/3/2008, phiên bản beta đầu tiên của bộ SDK được lưu hành rộng rãi, và cũng là lần đầu tiên, hệ điều hành chạy trên iPhone chính thức được đặt tên: **iPhone OS.**

Tháng 9/2007, Apple Inc tung ra chiếc **iPod Touch**, với gần như đầy đủ các chức năng của iPhone được bán với giá rẻ hơn nhiều. Ngày 27/1/2010, Apple Inc tiếp tục ra mắt **iPad**, chiếc máy tính bảng (tablet computer) với màn hình cảm ứng chạm lớn hơn iPhone và iPod Touch. iPad sử dụng hệ điều hành iPhone OS với một số khác biệt nhỏ, nhưng vẫn chạy được tất cả các ứng dụng được viết cho iPhone.

Tháng 6/2010, Steve Jobs công bố iPhone OS được đổi tên thành **iOS**, coi đó là minh chứng cho sự trưởng thành của hệ điều hành.

Tháng 9/2010, Apple Inc công bố thế hệ thứ 2 của đầu thu kỹ thuật số Apple Inc TV. Mặc dù không được đề cập đến, nhưng thiết bị này cũng sử dụng 1 phiên bản rút gọn của iOS.

2.1.2 Các phiên bản quan trọng của hệ điều hành iOS

Bảng 1- Các phiên bản quan trọng của iOS

Phiên bản	Những chức năng quan trọng
1.0	Ra mắt cùng chiếc iPhone đầu tiên.
1.1	Ra mắt cùng chiếc iPod Touch đầu tiên. Hỗ trợ mua nhạc MP3 với ứng dụng iTunes Music Store.
2.0	Ra mắt cùng iPhone 3G. Hỗ trợ mua ứng dụng hãng thứ 3 với ứng dụng App Store. Hỗ trợ 3G.
2.1	Ra mắt cùng iPod Touch thế hệ 2.
3.0	Ra mắt cùng iPhone 3GS. Hỗ trợ Copy - Cut - Paste. Hỗ trợ quay phim.
3.1	Ra mắt cùng iPod Touch thế hệ 3.
3.2	Ra mắt cùng chiếc iPad đầu tiên. Ngừng hỗ trợ iPhone và iPod Touch thế hệ đầu tiên.
4.0	Ra mắt cùng iPhone 4. Chỉ dành cho iPhone và iPod Touch thế hệ 2 trở lên. Hạn chế 1 số chức năng trên iPhone 3G và iPod Touch thế hệ 2. Hỗ trợ đa nhiệm (multitasking.) Hỗ trợ đọc và mua sách với ứng dụng iBooks. Hỗ trợ điện thoại video với Facetime. Hỗ trợ quảng cáo trên iAd. Một phiên bản rút gọn được dùng cho Apple Inc TV thế hệ 2.
4.1	Ra mắt cùng iPod Touch thế hệ 4. Hỗ trợ kết bạn - chia sẻ âm nhạc với mạng xã hội Ping của Apple Inc.

	Hỗ trợ chơi game online và chia sẻ thánh tích trong game với ứng dụng.
4.2	Đem những chức năng của iOS 4.0 tới iPad. Hỗ trợ truyền tin hiệu video từ các thiết bị iDevice tới Apple Inc TV với AirPlay. Hỗ trợ in ấn từ xa từ các thiết bị iDevice với AirPrint.

2.1.3 Thị trường các thiết bị và ứng dụng iOS:

Doanh số bán thiết bị iDevice trên thị trường Mỹ (đến tháng 04/2010):

- 50 triệu iPhone.
- 35 triệu iPod Touch.
- 2 triệu iPad.

Các số liệu của App Store:

- Có 300.000 ứng dụng iOS, trong đó:
  - 85% là ứng dụng iPhone/iPod Touch.
  - 7% là ứng dụng iPad.
  - 8% là ứng dụng universal, tức vừa iPhone vừa iPad.
- 1/3 số các ứng dụng là miễn phí, trong đó một số lượng lớn là các phiên bản rút gọn để khuyến khích người dùng chi tiền cho phiên bản chính thức.
- Giá bán trung bình các ứng dụng: 2.43 USD.
- Tổng số lượt download ứng dụng: 5 triệu.
- Tổng lợi nhuận của nhà phát triển ứng dụng (sau khi đã chia 30% cho Apple Inc): 1 tỷ USD.
- Số lượng nhà phát triển ứng dụng iOS: 62.126.
- Mỗi tuần có 15.000 ứng dụng mới được đưa vào App Store, với hơn 97% được Apple Inc chấp thuận trong 7 ngày đầu.

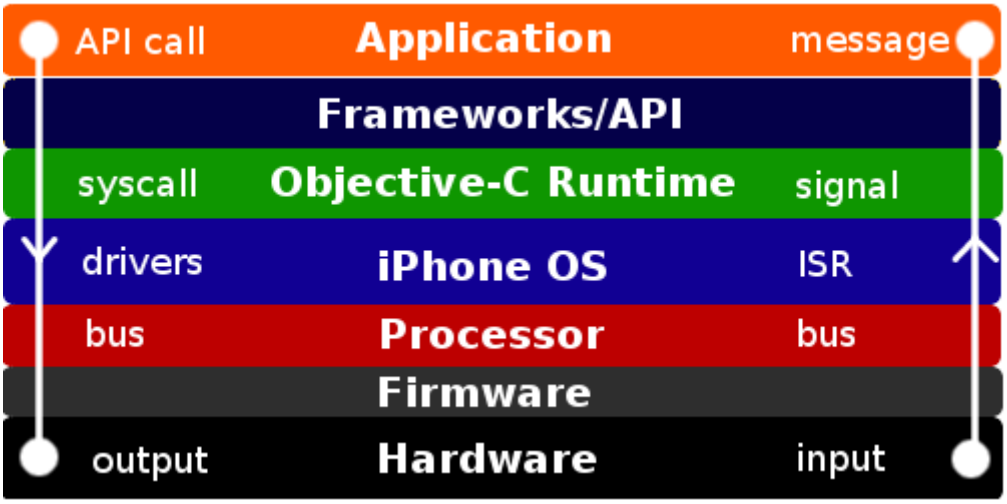
2.2 Tổng quan về kiến trúc HĐH iOS

Vì lý do bảo mật mà phần nhân hệ điều hành của iOS được giấu kín khỏi sự hiểu biết của người phát triển ứng dụng cho iOS. Để tìm hiểu cấu trúc tổng quan của hệ điều hành iOS cần phải xem xét dưới hai góc nhìn khác nhau: Mô hình tương tác giữa ứng

dụng và phần cứng và mô hình các lớp thể hiện tổng quan hệ điều hành iOS dưới góc nhìn của một nhà phát triển ứng dụng.

2.2.1 Mô hình tương tác giữa ứng dụng và phần cứng

Về kiến trúc hệ điều hành thì **iOS** được phát triển dựa trên bộ nhân **Mach** (Bộ nhân được đại học **Carnegie Mellon** phát triển từ **UNIX**).



Hình 1 - Biểu đồ mô hình tổng quan hệ điều hành iOS

Biểu đồ trên thể hiện mô hình kiến trúc hiện điều hành iOS được mô tả cụ thể như sau:

Application: Đây là ứng dụng hiện tại đang chạy trên hệ điều hành iOS.

Frameworks/API: Những framework được tích hợp sẵn trên hệ điều hành iOS (Sau này có thêm một số framework được phát triển bởi một hãng thứ ba).

Objective-C runtime: Môi trường thực thi Objective-C. Đây là lớp bao gồm cả thư viện DLL Objective cũng như các thư viện C. Các bộ thư viện C sẽ thiết lập một môi trường thực thi Objective-C.

iPhone OS: Tên cũ của hệ điều hành iOS bao gồm kernel, driver và dịch vụ tích hợp trên hệ điều hành iOS. Đây được xem là bộ phận trung gian giữa ứng dụng và phần cứng thiết bị nói chung.

Processor: Lớp này không phải là chip xử lý ARM (Được bao gồm trong lớp Hardware), được dùng để đại diện cho tập các lệnh vi xử lý chip ARM hỗ trợ.

Firmware: Lớp chứa đựng các trình điều khiển thiết bị.

Hardware: Đây là lớp đại diện cho phần cứng thiết bị bao gồm các thiết bị như: màn hình thiết bị, các cảm biến…

Để làm rõ mô hình tổng quan hệ điều hành iOS ta tiến hành phân tích theo hai hướng, từ phần cứng đi lên ứng dụng và từ ứng dụng đi xuống phần cứng.

2.2.1.1 Mô hình tương tác từ ứng dụng xuống phần cứng

Để làm rõ mô hình tương tác từ ứng dụng xuống phần cứng ta xem xét một ví dụ: ứng dụng yêu cầu phần cứng thiết bị iDevice thể hiện một hình ảnh đồ họa lên màn hình thiết bị.

1. Ứng dụng được nạp lên bộ nhớ và bắt đầu thực thi. Trong quá trình thực thi ứng dụng cần thể hiện một hình ảnh đồ họa lên màn hình thiết bị.
2. Yêu cầu này thể hiện một hình ảnh đồ họa được ứng dụng thực hiện bằng cách gọi các hàm API tích hợp sẵn trên iOS (**-(void)setBackgroundImage**) bằng cách truyền đường dẫn hình ảnh muốn hiển thị lên thiết bị trong lời gọi hàm.
3. Framework nhận được lời gọi hàm và dịch lời gọi hàm trên từ mức cao xuống các hàm API ở mức thấp hơn (Mô tả kĩ hơn ở phần mô hình 4 lớp) bằng cách gọi đến môi trường thực thi Objective-C (**Objective-C runtime**). Bộ thực thi Objective-C sẽ gọi đến các hàm ở mức thấp hơn được định nghĩa trong các bộ thư viện C.
4. Bộ thư viện liên kết động (dynamically-linked-library in C) mà nhận được một loạt các lời gọi hàm từ Objective-C runtime, các Framework hoặc API tương ứng sẽ tiếp tục biên dịch những lời gọi hàm mà nó nhận được thành dạng ngôn ngữ ở mức thấp hơn (ngôn ngữ máy) dùng để thực thi trên trên nhân iOS.
5. Tại iOS kernel, nơi nhận được các lời gọi hệ thống ở dạng ngôn ngữ máy từ các bộ thư viện C sẽ gọi xuống các trình điều khiển thích hợp (Touchscreen display driver…) để tương tác với phần cứng. Trình điều khiển phần cứng sẽ điều khiển thiết bị để thực hiện các yêu cầu như: hiện thiện hình ảnh, âm thanh, lấy thông tin từ cảm biến…
6. Tại lớp phần cứng, màn hình hiển thị vật lý sẽ thay đổi để hiện thị hình ảnh, âm thanh, lấy thông tin cảm biến…

2.2.1.2 Mô hình tương tác từ phần cứng lên ứng dụng

1. Phần cứng phát hiện những thay đổi hệ thống dưới dạng biến đổi vật lý.
2. Firmware phát hiện những thay đổi của phần cứng và gởi một processor interrupt thông qua system bus.

3. Bộ vi xử lý ARM nhận interrupt và gọi một in-memory interrupt service routine (ISR) thuộc hệ điều hành iOS.
4. ISR là một hàm (một đoạn chương trình con) đặt trong hệ điều hành iOS có nhiệm vụ tiếp nhận interrupt và gởi một tín hiệu (signal) đến ứng dụng thích hợp.
5. Tín hiệu được gởi đi ở bước trước sẽ được bắt giữ thông qua một phần cụ thể của môi trường thực thi Objective-C. Tại đây, môi trường thực thi Objective-C sẽ gởi một tín hiệu đến một ứng dụng cụ thể bằng cách xác định những ứng dụng hiện tại nào được thiết kế để bắt giữ tín hiện tại.
6. Nếu sự kiện đã được khai báo sẽ bắt giữ bởi một hàm cụ thể trên ứng dụng thì sự kiện trên sẽ được bắt giữ, xử lý và thay đổi giao diện để thông báo sự kiện đến người sử dụng ứng dụng.

2.2.2 Mô hình các lớp thể hiện tổng quan hệ điều hành iOS dưới góc nhìn của một nhà phát triển ứng dụng

Nền tảng iOS bao gồm hệ điều hành và các công nghệ sử dụng để chạyứng dụng trên các thiết bị như củaApple Inc như: iPad, iPhone, iPod touch. Mặc dù kiến trúc của nền tảng iOS chia sẻ những công nghệ cơ sở với hệđiều hành Mac OS X, iOS được thiết kế đểđápứng những nhu cầu của một nền tảng di động, nơi mà nhu cầu của người dùng khácđôi chút so với những nền tảngđiện toán truyền thống. Những công nghệ trên nền tảng iOS được kế thừa phần lớn từ Mac OS X và cũng bổ sung những công nghệ riêng biệt mới của nền tảng di động (Multi-Touch và Công nghệ cảm biến).

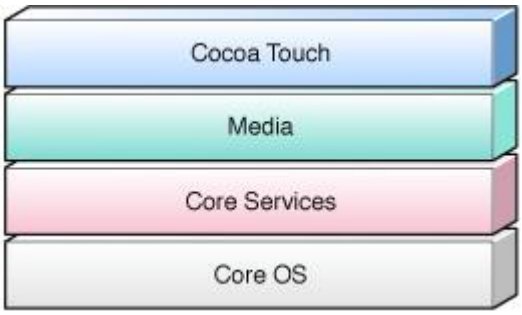
iOS là hệ điều hành sử dụng trên các thiết bị của Apple Inc như: iPhone, iPod touch hoặc iPad. Hệ điều hành iOS quản lý phần cứng thiết bị và qua đó cung cấp những công nghệ cần thiết để hiện thực những ứng dụng tận dụng tối đa sức mạnh phần cứng iDevice (Thiết bị di động của Apple Inc nói chung: iPhone, iPod touch, iPad).

iOS SDK bao gồm bộ source code mẫu, tài liệu cũng như những công cụ cần thiết để test, run, debug ứng dụng cho nền tảng iOS. Ứng dụng của nền tảng iOS được phát triển bằng những framework và ngôn ngữ Objective-C chạy trực tiếp trên thiết bị iDevice.

Nhân của nền tảng iOS dựa vào nhân của hệ điều hành Mac OS X. Trên cùng của bộ nhân là mô hình 4 lớp cơ bản cung cấp những dịch vụ cần thiết để hiện thực những ứng dụng cho nền tảng này. Nền tảng iOS đóng vai trò là thành phần trung gian giữa phần cứng thiết bị và những ứng dụng được chạy trên thiết bị. Những ứng dụng được phát triển cho thiết bị iDevice của App ít khi nào tương tác trực tiếp với phần cứng thiết bị. Thay vào đó, ứng dụng sẽ giao tiếp với phần cứng thông qua những system interface nhằm bảo vệ ứng dụng khỏi sự thay đổi của phần cứng thiết bị. Và

mặc dù ứng dụng được xây dựng cho nền tảng iOS được bảo vệ tránh khỏi những thay đổi của thiết bị nhưng nhà phát triển ứng dụng phần phải tính toán trước để phù hợp với sự khác biệt giữa các thiết bị iDevice trong khi lập trình. Để đảm đảm bảo sự đáp ứng của phần cứng thiết bị iDevice cần khai báo những tính năng phần cứng cần thiết cho ứng dụng khi hiện thực ứng dụng.

Các công nghệ được cung cấp bởi iOS dùng để phát triển ứng dụng cho nền tàng này được thể hiện qua mô hình bốn lớp cơ bản sau:



Hình 2 - Mô hình bốn lớp của nền tảng iOS

Mỗi lớp trong mô hình trên cung cấp một số công nghệ nhất định đến người dùng thông qua các framework cụ thể. Mỗi **framework** là một thư mục chứa một thư viện chia sẻ động (**dynamic shared library**) và những tài nguyên khác (header file, hình ảnh, helper application…). Để sử dụng những framework này người phát triển ứng dụng iOS chỉ cần liên kế chúng vào project của ứng dụng đang xây dựng. Ngoài những framework đặc trưng của iOS, Apple Inc còn cung cấp những công nghệ ở dạng thư viện chia sẻ chuẩn (**standar shared library**). Do iOS được phát triển dựa vào nền tảng UNIX nên nhiều công nghệ ở mức thấp của hệ điều hành được Apple Inc phát hành dưới dạng công nghệ mở (open source technology). Những lớp phía dưới (Core OS, Core Services) cung cấp những công nghệ và dịch vụ nền tảng mà mọi chương trình cần đến. Những lớp ở phía trên (CoCoa Touch, Media) cung cấp những công nghệ và dịch vụ phức tạp hơn cho việc xây dựng một ứng dụng trên nền tảng iOS.

Đối với phát triển ứng dụng trên nền tảng iOS, người phát triển thường sử dụng những framework ở mức cao hơn là sử dụng những framework ở mức thấp. Những framework ở mức cao được xây dựng từ những framework ở mức thấp nhằm cung cấp một sự " **trừu tượng hóa hướng đối tượng**" (object-oriented abstractions) cho những công nghệ cung cấp ở các framework mức thấp.

Sự trừu tượng hóa các khái niệm và tính năng cung cấp ở những framework mức thấp giúp giảm bớt số lượng dòng code phải viết cũng như gói gọn những tính năng phức tạp (Socket, thread) trong những khái niệm hướng đối tượng, bằng cách đó nó khiến việc sử dụng những tính năng ở mức thấp trở nên dễ dàng hơn. Mặc dù những tính năng ở mức thấp được trừu tượng hóa hướng đối tượng trong những framework ở mức cao hơn nhưng nó vẫn sẵn sàng cho những nhà phát triển hệ thống, những người mong muốn sử dụng những công nghệ ở mức thấp để phát triển những tính năng ở mức hệ thống.

Lớp Cocoa touch

Lớp Cocoa touch chứa những framework chủ đạo để xây dựng một ứng dụng trên nền tảng iOS. Lớp này định nghĩa những cơ sở hạ tầng ứng dụng cơ bản và hỗ trợ những công nghệ chủ đạo như Multitasking, touch-based input, push notification và những dịch vụ hệ thống ở mức cao. Khi bắt đầu thiết kế một ứng dụng cho nền tảng iOS thì nhà phát triển nên tham khảo những công nghệ mà lớp này cung cấp.

Tính năng hỗ trợ

+ Multitasking: Công nghệ đa nhiệm (Xuất hiện từ phiên bản iOS 4.0).

+ Printing: Hộ trợ công nghệ in ấn không dây.

+ Data protection: Công nghệ mã hóa và giải mã dữ liệu.

+ Apple Inc Push Notification Service (Mới xuất hiện từ phiên bản iOS 3.0): Cung cấp một cách thức hoàn toàn mới nhằm đưa một thông báo về một thông tin nào đó đến người dùng theo một cách hoàn toàn mới và thực sự phù hợp với một thiết bị di động cảm ứng.

+ Local Notifications.

+ Gesture Recognizers (iOS 3.2 trở lên): Hỗ trợ khả năng ghi nhận những tương tác theo nhiều cách khác nhau của người dùng trực tiếp lên màn hình thiết bị (gõ, double-touch, kéo thả, xoay, chạm và giữ, trượt…), đây chính là một trong những tính năng đặc biệt của iOS đánh bại mọi đối thủ khi ra mắt đồng thời tạo nên những chuẩn mực giao diện tương tác trực quan, sinh động giữa người và màn hình thiết bị.

+ File-Sharing: Hỗ trợ khả năng đồng bộ hóa tập tin giữa iTune và thiết bị.

+ Peer-To-Peer Services: Hỗ trợ khả năng giao tiếp thông qua mạng ngang hàng.

+ Standard System View Controller: Hỗ trợ những giao diện chuẩn dùng để thao tác với Contact, thao tác tệp tin, tạo sự kiện theo lịch, chụp ảnh, xem clip từ YouTube. Tính năng nhằm giúp giảm bớt thời gian hiện thực ứng dụng bằng cách tái sử dụng những giao diện chuẩn của các ứng dụng mặc định của hệ điều hành iOS.

+ External Display Support: Hỗ trợ khả năng xuất hình ảnh từ iDevice ra các thiết bị hiện thị bên ngoài.

Framework

+ Address Book UI: Framework cung cấp các giao diện chuẩn dùng tương tác với Contact lưu trữ trên thiết bị.

+ Event Kit UI: Framework cung cấp các giao diện chuẩn dùng tương tác với các sự kiện theo lịch.

+ Game Kit UI: Framework cung cấp khả năng xây dựng tính năng kết nối mạng ngang hàng dùng hỗ trợ những game nhiều người chơi.

+ iAd: Framework hỗ trợ dịch vụ gắn quảng cáo trên ứng dụng dưới sự hỗ trợ của dịch vụ quảng cáo của Apple Inc (Gắn banner quảng cáo trên ứng dụng và bán quyền quảng cáo cho Apple Inc).

+ Map Kit: Hỗ trợ truy vấn và sử dụng hệ thống bản đồ trực tuyến của Google.

+ Message UI: Hỗ trợ giao diện chuẩn để gởi nhận email.

+ UIKit: Đây là framework quan trọng nhất của lớp Cocoa touch. Đây là framework cung cấp cơ sở hạ tầng chủ đạo để phát triển một **ứng dụng đồ họa theo sự kiện** (graphical, event-driven application) trên nền tảng iOS. Mọi ứng dụng đều sử dụng framework này để cài đặt những tính năng sau:

- Quản lý ứng dụng.
- Quản lý giao diện người dùng.
- Đồ họa và cửa sổ ứng dụng.
- Đa nhiệm.
- In ấn.
- Hỗ trợ bắt giữ các sự kiện dựa vào cử động tương tác trên màn hình thiết bị.
- Hỗ trợ khả năng tương tác với các đối tượng của hệ thống (Contact…).
- Hỗ trợ nội dung văn bản và nội dung web.
- Hỗ trợ các thao tác copy, cut và paste.
- Hỗ trợ hiệu ứng giao diện người dùng.
- Hỗ trợ tương tác với các ứng dụng khác thông qua URL.
- Hỗ trợ dịch vụ Push Notification.

- Hỗ trợ tạo file PDF.
- Hỗ trợ khả năng nhập liệu thông qua tương tác chạm.
- Hỗ trợ khả năng truy xuất dữ liệu của các cảm biến trên thiết bị (Camera, GPS…
- Hỗ trợ truy xuất thư viện hình ảnh, âm thanh của thiết bị.
- Hỗ trợ thông tin thiết bị…

Lớp Media

Lớp Media cung cấp những công nghệ liên quan đến xử lý hình ảnh, âm thanh và video.

Các công nghệ xử lý hình ảnh

Đồ họa chất lượng cao là một phần rất quan trọng của những ứng dụng được xây dựng trên nền tảng iOS. Cách đơn giản nhất (và cũng là hiệu quả nhất) để tạo ra một ứng dụng có giao diện đồ họa tốt là sử dụng phương pháp chèn hình nền dưới những control mặc định được iOS cung cấp. Tuy nhiên đôi lúc cũng có những tình huống nhà phát triển ứng dụng mong muốn xây dựng những control đặc biệt hoặc đơn giản là muốn thể hiện nội dung đồ họa thì lớp Media sẽ cung cấp những công nghệ giúp nhà phát triển quản lý công việc thể hiện nội dung này.

Core Graphic (Còn được biết đến với tên Quartz) cung cấp khả năng vẽ vector 2D hoặc sự thể hiện dựa vào nội dung ảnh.

Core Animation (Một phần của Quartz Core Framework) cung cấp khả năng hỗ trợ hiệu ứng cho các khung nhìn (View- Có thể hiểu là Control container) và những nội dung khác.

OpenGL ES: Cung cấp khả năng trình diễn 2D và 3D sử dụng sự tăng tốc bằng phần cứng.

Core Text: cung cấp khả năng tạo một bố cục văn bản phức tạp.

Assets Library: Cung cấp khả năng truy xuất thư viện hình ảnh, video từ thư viện của media của người dùng.

Các công nghệ xử lý âm thanh

Các công nghệ xử lý âm thanh trên hệ điều hành iOS được thiết kế để giúp nhà phát triển xây dựng những ứng dụng có khả năng cung cấp những trải nghiệm tốt nhất có thể về âm thanh.

Hệ điều hành iOS cung cấp nhiều cách để trình diễn và ghi lại nội dung âm thanh. Những Framework ở danh sách dưới được sắp xếp theo thứ tự từ mức level

cao xuống level thấp (Từ mức độ trừu tượng hướng đối tượng cao nhất xuống mức gần hệ thống nhất). Việc hiểu rõ mức độ trừu tượng hóa của một framework giữ một vai trò rất quan trọng khi lựa chọn một framework để xử lý âm thanh trên ứng dụng được phát triển cho nền tảng iOS. Những framework ở mức cao sẽ cung cấp các API phổ biến và dễ dàng sử dụng hơn cả. Những framework ở mức thấp cung cấp những khả năng linh hoạt hơn nhưng cũng đòi hỏi nhà phát triển phải làm việc nhiều hơn.

Media Player: Cung cấp khả năng truy xuất thư viện iTune cũng như hỗ trợ khả năng trình diễn file âm thanh và playlist.

AV Foundation: Cung cấp một tập các Objective-C API có thể sử dụng để quản lý việc phát lại và ghi âm âm thanh.

OpenAL: (Tham khảo OpenGL)

Core Audio: Framework này không những cung cấp các API đơn giản mà còn cung cả những API phức tạp được dùng để trình diễn hoặc ghi lại nội dung âm thanh.Những định dạng âm thanh mà iOS hỗ trợ:AAC, ALAC, A-law, IMA/ADPCM(IMA4), Linear PCM, µ-law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10, AES3-2003.

Các công nghệ xử lý video

Để chơi một đoạn phim từ tệp tin hoặc tải về từ một nơi nào đó thì iOS cung cấp một vài công nghệ có thể sử dụng để trình diễn những tệp tin định dạng video số. Ngoài ra nếu thiết bị iDevice hỗ trợ cảm biến hình ảnh thì những công nghệ này còn có thể sử dụng để ghi trực tiếp những nội dung này lên ứng dụng. iOS hỗ trợ một vài công nghệ có thể sử dụng để trình diễn video mà nhà phát triển có thể cân nhắc lựa chọn khi xây dựng ứng dụng. Khi lựa chọn một công nghệ để xử lý video trên ứng dụng nhà phát triển cần hiểu rõ những framework thì cung cấp những API dễ sử dụng nhưng không sẽ không đủ tính linh hoạt để tùy biến. Những framework ở mức level thấp thì linh hoạt dễ tùy biến nhưng lại đòi hỏi nhiều thời gian làm việc hơn. Danh sách các Framework dưới đây liệt kê theo mức level từ cao xuống thấp (Từ mức độ trừu tượng hướng đối tượng cao nhất xuống mức gần hệ thống nhất).

UIImagePickerController: Lớp này thuộc framework UIKit cung cấp khả năng ghi lại video bằng thiết bị hỗ trợ cảm biến hình ảnh.

Media framework:Hỗ trợ khả năng trình diễn video.

AV Foundation: Cung cấp một tập các Objective-C API có thể sử dụng để quản lý việc phát lại và ghi âm âm thanh.

Core Media: Cung cấp những API mức level thấp được sử dụng để xây dựng những API ở mức cao hơn cũng đồng thời cho phép xử lý media ở mức thấp.

Các định dạng Video iOS hỗ trợ:

M4V, MP4, MOV, MPEG-4 và một số định dạng phổ biến.

Framework của lớp Media:

+ Assets Library.

+ AV Foundation.

+ Core Audio.

+ Core Graphics.

+ Core MIDI.

+ Core Text.

+ Core Video.

+ Image I/O.

+ Media Player.

+ OpenAL.

+ OpenGL ES.

+ Quartz Core.

Lớp Core Services

Lớp Core Services cung cấp những system service nền tảng mà phần lớn mọi ứng dụng trên nền tảng iOS sử dụng. Mặc dù khi phát triển ứng dụng nhà phát triển không trực tiếp sử dụng các system service trực tiếp nhưng trong bản thân các hàm API có thể gọi các system service.

Các tính năng ở mức cao

Dưới đây là một số công nghệ chủ đạo có mặt trong hệ điều hành iOS.

Block Objects (Được giới thiệu ở phiên bản iOS 4.0)**:**Block object là một cấu trúc C (C-level language construct) mà nhà phát triển ứng dụng có thể kết hợp vào phần code C hoặc Objective-C. Một block object đơn giản là một hàm nặc danh (anonymous funtion) đi kèm với dữ liệu cần sử dụng trong đoạn chương trình trên, đôi khi với những ngôn ngữ khác nó được biết đến với tên **closure** hoặc **lambda**. Block rất hữu dụng trong trường hợp cần callback (Tham khảo thêm tại[đây](#)) hoặc tại những nơi mà nhà lập trình cần một cách dễ dàng để kết hợp phần code thực thi và dữ liệu đi kèm. Trên hệ điều hành iOS block thường được sử dụng trong các trường hợp sau:

+ Thay thế cho delegate và delegate methods.

+ Hiện thực một bộ xử lý hoàn thiện (completion handler) cho những tác vụ chỉ dùng một lần.

+ Hiện thực những tác vụ không đồng bộ.

+ Tạo callback.

GrandCentralDispatch (Được giới thiệu ở phiên bản iOS 4.0)**:** là một công nghệ được phát triển bởi **Apple Inc** cho các hệ thống đa lõi hoặc hệ thống đa xử lý đối xứng tối ưu hóa việc thực thi ứng dụng. Trong trường hợp chương trình muốn chạy đồng thời các tác vụ, GCD sắp xếp chúng vào một hàng đợi theo thứ tự. Tùy thuộc vào tính sẵn sàng của bộ vi xử lý mà lên kế hoạch cho các tác vụ này thực thi trên các lõi vi xử lý đang rảnh (availble processor cores). *(Tham khảo thêm tại đây:http://en.wikipedia.org/wiki/Grand_Central_Dispatch.)*

In-App Purchase (Được giới thiệu ở phiên bản 3.0): Cung cấp khả năng để buôn bán nội dung số hoặc dịch vụ từ ngay bên trong ứng dụng mà không cần phải tích hợp thêm thành phần thanh toán trực tuyến cho ứng dụng. Tính năng này được hiện thực trong Store KIT framework nhằm cung cấp một cơ sở hạ tầng cần thiết để xử lý các giao dịch thương mại bằng cách sử dụng tài khoản iTunes.

SQLite: Cung cấp một DBMS cho phép khởi tạo và quản lý database cục bộ (Được tạo và truy xuất giới hạn trong phạm vi ứng dụng).

XML: Cung cấp khả năng thao tác trên dữ liệu dạng XML.

Framework

Address Book: Cung cấp khả năng truy xuất và thao tác trên Address book của iOS.

CFNetwork: Là một tập hợp các API bằng ngôn ngữ C, có hiệu suất cao và đã được trừu tượng hóa hướng đối tượng để làm việc với các giao thức mạng (Trừu tượng hóa các công việc thực hiện với các giao thức mạng thành các đối tượng).

Core Data: Là framework cung cấp khả năng quản lý lớp Data Model trong mô hình 3 lớp MVC. Thay vì định nghĩa cấu trúc dữ liệu bằng cách lập trình, với framework Core Data các nhà phát triển ứng dụng có thể sử dụng công cụ đồ họa để để tạo lược đồ mô tả đối tượng data-model cần sử dụng trong chương trình. Khi chương trình thực thi, các đối tượng data-model được khởi tạo và truy vấn thông qua framework Core Data. Bằng cách quản lý các đối tượng data-model cho nhà phát triển ứng dụng, framework đã giúp nhà phát triển ứng dụng giảm bớt số lượng dòng code mà họ phải viết khi muốn xây dựng những ứng dụng cần sử dụng các đối tượng data-model. Framework cung cấp những tính năng sau:

+ Lưu trữ đối tượng data-model trong cơ sở dữ liệu SQLite nhằm tối ưu hóa thực thi.

+ Quản lý việc undo/redo các thao tác chỉnh sửa dữ liệu trên các đối tượng data-model.

+ Hỗ trợ kiếm chứng giá trị thuộc tính.

+ Hỗ trợ quản lý sự thay đổi của các đối tượng data-model và đảm bảo mối quan hệ giữa các đối tượng được giữ nguyên.

+ Hỗ trợ nhóm, lọc và tổ chức dữ liệu trên bộ nhớ.

Core Foundation: Framework Core Foundation là một tập các interface bằng ngôn ngữ C cung cấp những tính năng quản lý dữ liệu. Framework này hỗ trợ những tính năng sau:

+ Kiểu dữ liệu tập hợp (Array, set…).

+ Bundles.

+ Quản lý String.

+ Quản lý ngày tháng (Date, time).

+ Quản lý tham chiếu (Preferences management).

+ URL và stream.

+ Threads và vòng lặp.

+ Giao tiếp port và socket.

Core Location: Framework này cung cấp những API cho phép lấy vị trí của thiết bị thông qua chip GPS, định vị Cell ID hoặc định vị sóng Wifi.

Core Media: cung cấp các API xử lý media ở mức level thấp được sử dụng để xây dựng nên framework **AV Foundation**. Phần lớn các chương trình ít khi sử dụng framework này nhưng nó vẫn sẵn sàng dành cho những nhà phát triển mong muốn sử dụng các API xử lý media ở mức thấp.

Core Telephony: Cung cấp các API tương tác với các thông tin dựa trên điện thoại trên các thiết bị có tế bào vô tuyến (cellular radio). Ứng dụng sử dụng framework này có thể truy vấn thông tin về nhà cung cấp dịch vụ thoại…. Bằng các sử dụng framework này ứng dụng có thể bắt giữ các sự kiện thoại (Nhận cuộc gọi, tin nhắn…).

Event Kit: Framework cung cấp một interface dùng để truy xuất các sự kiện theo lịch trên thiết bị. Các nhà phát triển ứng dụng có thể sử dụng framework này để truy xuất các sự kiện có sẵn trên thiết bị, thêm mới một sự kiện vào lịch của người dùng trên thiết bị.

Foundation: Cung cấp một tập bao gói gồm các API trên Objective-C cho các tính năng được tìm thấy trong framework Core Foundation:

+ Collection data types (arrays, sets, and so on)

+ Bundles

+ String management

+ Date and time management

+ Raw data block management

+ Preferences management

+ URL and stream manipulation

+ Threads and run loops

+ Bonjour

+ Communication port management

+ Internationalization

+ Regular expression matching

+ Cache support

Mobile Core Services (Được giới thiệu ở iOS 3.0): Framework này định nghĩa các low-level type được sử dụng trong Uniform Type Indentifiers (UTIs).

Quick Look (Được giới thiệu ở iOS 4): Cung cấp các API để xem trước trực tiếp nội dung các tệp tin không được hỗ trợ trực tiếp bằng ứng dụng.

Store Kit: Cung cấp một khả năng buôn bán những nội dung số hoặc dịch vụ ngay trong bản thân ứng dụng. Bằng cách sử dụng framework này nhà phát triển ứng dụng có thể bán những tính năng mở rộng ngay trong ứng dụng mà không cần phải tích hợp thêm bất kì một module thanh toán nào.

System Configuration: Framework này cung cấp khả năng truy xuất các thông tin cấu hình hệ thống (Cấu hình mạng, cấu hình phone…)

Lớp Core OS

Lớp Core OS cung cấp những tính năng ở mức level thấp mà từ đó mọi công nghệ ở mức cao hơn được xây dựng nên. Mặc dù các nhà phát triển ứng dụng gần như rất ít sử dụng các công nghệ ở lớp này một các trực tiếp nhưng nó sẽ được sử dụng ở các framework khác. Và trong trường hợp nhà phát triển muốn trực tiếp sử dụng các công nghệ ở mức thấp thì có thể sử dụng các công nghệ ở mức level này.

Accelerate: Framework chưa đựng các interface phục vụ cho việc tính toán toán học, xử lý các con số lớn, tính toán DSH và những xử lý khác.

External Accessory (Được giới thiệu ở phiên bản iOS 3.0): Framework này hỗ trợ việc giao tiếp với các thiết bị phần cứng gắn ngoài kết nối thông qua cáp kết nối 30 chân hoặc kết nối không dây dựa vào sóng Bluetooth. Bằng cách sử dụng framework này các nhà phát triển ứng dụng có thể truy vấn thông tin từ các thiết bị gắn ngoài cũng như gọi các lệnh được hỗ trợ bởi thiết bị gắn ngoài.

Security (Được giới thiệu ở phiên bản iOS 3.0): Cung cấp các interface mã hóa hoặc giải mã dữ liệu.

System: Cung cấp khả năng truy xuất các thành phần ở mức sâu như driver, interface UNIX mức level thấp của hệ điều hành.

2.3 Các thành phần cơ bản của HĐH iOS

2.3.1 Window và View

Trên hệ điều hành iOS, window và view là hai thành phần dùng thể hiện nội dung của ứng dụng lên màn hình thiết bị iDevice. Window bản thân nó không có bất kì nội dung nào có thể hiển thị lên màn hình nhưng nó lại cung cấp một thùng chứa (container) chính để chứa view. Đối tượng View chiếm giữ một hình chữ nhật của window dùng để hiển thị nội dung. View được dùng như một container để chứa view con (UIButton,UITextView…).

Mỗi ứng dụng có duy nhất một window và có ít nhất một view dùng để thể hiện nội dung của ứng dụng. UIKit và những framework khác định nghĩa các kiểu view mà các nhà phát triển có thể sử dụng để hiển thị nội dung mà ứng dụng muốn truyền tải đến người dùng. Các bộ framework tích hợp trong hệ điều hành iOS cung cấp các kiểu view từ đơn giản như button, text label hoặc những kiểu view phức tạp hơn như table view, picker view, scroll view. Ngoài những kiểu view được hệ điều hành iOS hỗ trợ, các nhà phát triển ứng dụng có thể tự định nghĩa các kiểu view riêng.

2.3.1.1 Tổng quan về Window và View

Window định vị vùng hiển thị của view

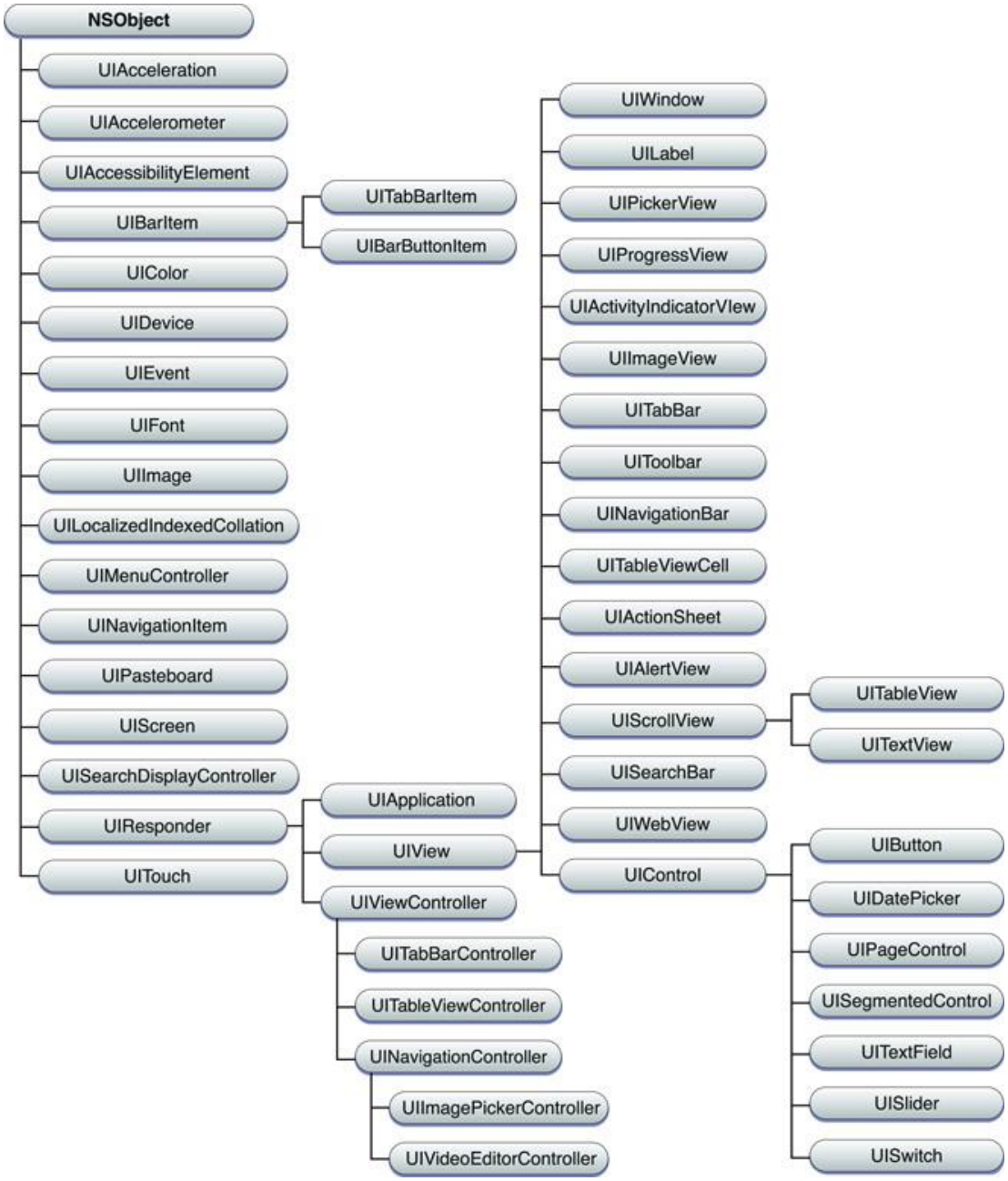
Một đối tượng window là một thể hiện của UIWindow có nhiệm vụ điều khiển mọi công việc thể hiện giao diện đồ họa. Đối tượng window hợp tác với view và các view controller tương ứng để quản lý việc hiển thị nội dung và xử lý tương tác với người dùng. Gần như đối tượng window của một ứng dụng sẽ không thay đổi trong suốt khoảng thời gian sống của ứng dụng. Mọi ứng dụng chỉ có duy nhất một đối tượng window dùng để hiện thị giao diện người dùng trên màn hình chính của thiết bị. Nếu

một thiết bị hiện thị gắn ngoài (external display) kết nối vào thiết bị iDevice, ứng dụng có thể tạo một đối tượng window thứ hai dùng để thể hiện nội dung trên màn hình của thiết bị gắn ngoài này.

View quản lý nội dung mà ứng dụng muốn hiển thị

Một đối tượng view là một thể hiện của lớp UIView (Hoặc lớp con của UIView) và làm nhiệm vụ quản lý một vùng chữ nhật trên cửa sổ của ứng dụng. Đối tượng view chịu trách nhiệm thể hiện nội dung, đón nhận các sự kiện đa chạm (Multi-touch) và quản lý việc sắp đặt các view con nằm trong nó. Một view đón nhận sự kiện tương tác lên vùng hiển thị mà nó quản lý thông qua đối tượng Gesture Recognizer hoặc bằng cách trực tiếp bắt lấy sự kiện chạm (touch-event) lên đối tượng view (Mỗi loại view hỗ trợ một số sự kiện chạm nhất định, với những kiểu view do người dùng định nghĩa thì cần sử dụng các đối tượng Gesture Recognizer để bắt sự kiện tương tác với view).

Dưới đây là cây cấu trúc các class thuộc framework UIKit:



Hình 3 - Danh sách Class được hỗ trợ trong framework UIKit

Danh sách các Class kế thừa UIView và chức năng tương ứng:

UIControl: Lớp trừu tượng dùng kế thừa khi xây dựng control, lớp này hỗ trợ khả năng xử lý các sự kiện tương tác lên control (Chạm, gỏ, búng, trượt…).

UIWebView: Dùng hiển thị nội dung có định dạng HTML.

UISearchBar: Thanh tìm kiếm.

UIScrollView: View hỗ trợ khả năng hiện thị nội dung vượt ngoài kích thước của màn hình thiết bị.

UIAlertView: View hỗ trợ hiển thị một cảnh báo (alert) đến người dùng.

UIActionSheet: View hỗ trợ hiện thị một tập các lựa chọn.

UINavigationBar: Thanh điều hướng.

UIToolBar: Thanh công cụ, nơi đặt các nút tác vụ tương ứng với cửa sổ view.

UITabBar: Thanh tabbar của cửa sổ view.

UIImageView: View hỗ trợ hiển thị hình ảnh.

UIActivityIndicatorView: View chuyển động thể hiện trạng thái load dữ liệu.

UIProcessView: View hỗ trợ hiển thị thanh tiến trình.

UIPickerView: View hỗ trợ hiện thị Picker cho phép lựa chọn ngày tháng hoặc các thông tin khác.

UILabel: View hỗ trợ hiển thị nhãn.

UIWindow: Dùng để tạo đối tượng window cho ứng dụng.

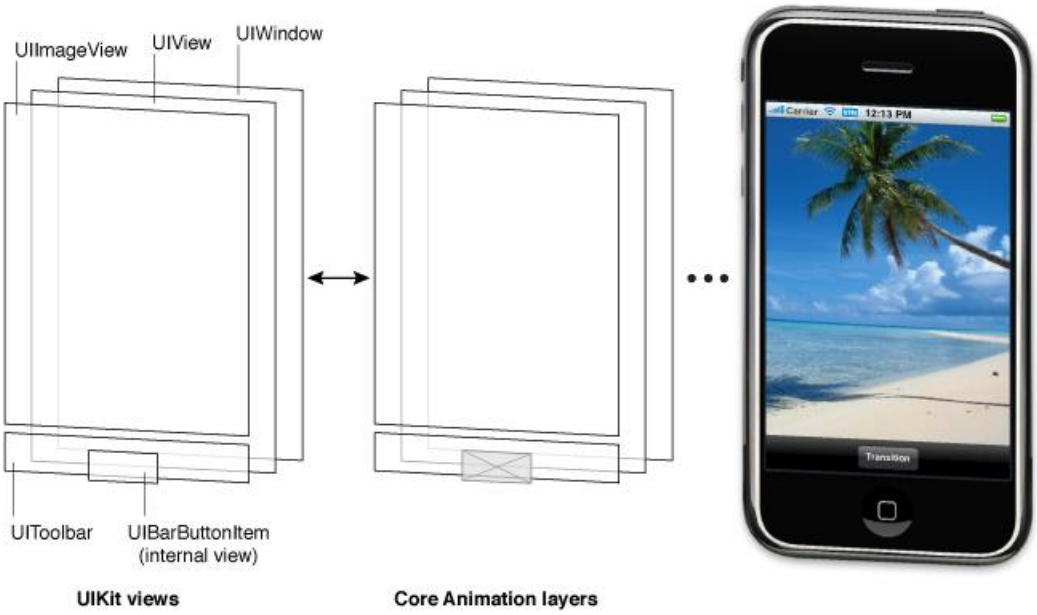
Core Animation

Hệ điều hành iOS hỗ trợ framework Core Animation cung cấp khả năng phản hồi sự thay đổi của giao diện đồ họa ứng dụng đến người dùng. Qua đó iOS định nghĩa một số hiệu ứng cho việc hiển thị view, chuyển đổi giữa các view, thay đổi vị trí view…. Tuy nhiên bản thân view cũng có nhiều thuộc tính có thể tự tạo hiệu ứng chuyển đổi một cách trực tiếp (Thay đổi độ mờ của view, vị trí view, kích thước, hình nền và một số thuộc tính khác) không cần thông qua Core Animation.

Interface Builder

Interface Builder là một công cụ có giao diện đồ họa kéo thả nằm trong bộ công cụ X-Code tích hợp trong iOS SDK cung cấp khả năng xây dựng giao diện đồ họa cho ứng dụng trên iOS. Bằng cách sử dụng Interface Builder, các nhà phát triển ứng dụng có thể đóng gói các đối tượng view vào file nib, đây là một file nén dùng để đóng gói các đối tượng view cũng như các đối tượng thuộc giao diện đồ họa (view controller). Khi ứng dụng nạp một file nib, những đối tượng bên trong nó sẽ được giải nén thành các đối tượng có thể sử dụng trong chương trình mà không cần khởi tạo bằng code.

Dưới đây là cấu trúc View của một ứng dụng trên nền iOS:



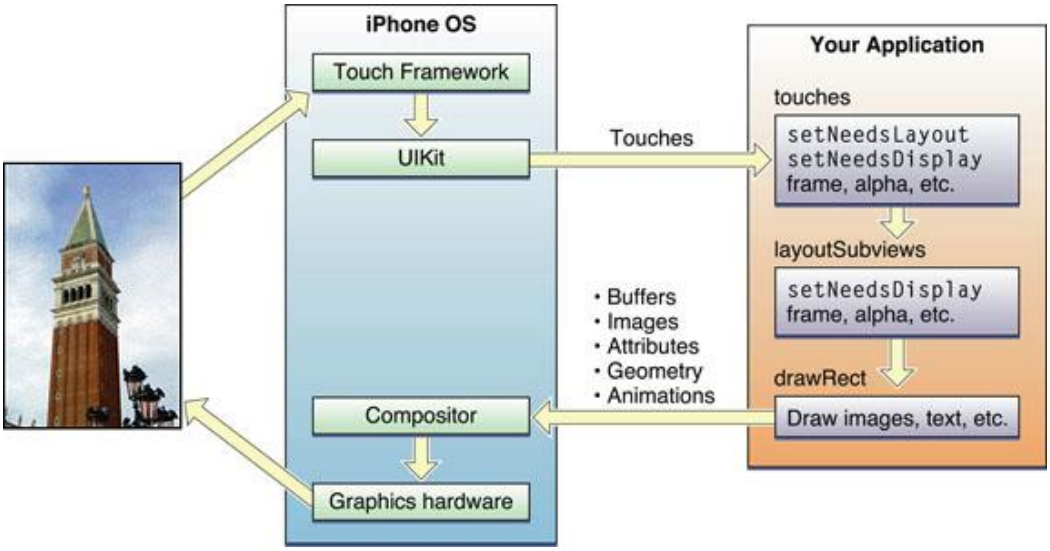
Hình 4 - Cấu trúc view của một ứng dụng iOS

Một đối tượng view quản lý một vùng hình chữ nhật trên màn hình của thiết bị, nó sẽ quản lý việc vẽ và bắt giữ các sự kiện chạm tác dụng lên vùng chữ nhật này. Ngoài ra view còn có thể dùng như một container để chứa các loại view khác. Mỗi đối tượng view hoạt động trong một mối quan hệ cộng tác với một đối tượng Core Animation layer để điều khiển việc thực thi các hiệu ứng thay đổi nội dung của view đó. Đằng sau mỗi đối tượng view thuộc framework UIKit là một đối tượng layer (là một thể hiện của CALayer) có nhiệm vụ quản lý nền phía sau của một đối tượng view cũng như điều khiển các hiệu ứng liên quan đến view. Việc sử dụng đối tượng Core Animation layer có ý nghĩ rất lớn với hiệu suất làm việc của ứng dụng. Để nâng cao hiệu suất của ứng dụng thì việc sử dụng mã lệnh của đối tượng view để vẽ lại nội dung của view lên màn hình thiết bị iDevice được giảm thiểu một cách tối đa. Mỗi khi thực hiện việc vẽ nội dung của một đối tượng view lên màn hình thiết bị thì kết quả vẽ được sẽ được cache lại bởi đối tượng Core Animation layer dưới dạng hình BITMAP và được tái sử dụng khi có thể. Việc tái sử dụng những nội dung mà đối tượng layer đã cache được trước đó sẽ không những làm giảm chi phí cần thiết để cập nhật nội dung cho một view mà còn giúp đơn giản hóa việc tạo một hiệu ứng cho view.

2.3.1.2 Mô hình quản lý tương tác

Bất kì khi nào có sự tương tác với giao diện người dùng của chương trình (hoặc một sự thay đổi đến từ mã lệnh của chương trình), một dãy các sự kiện tuần tự diễn ra ngay bên trong của UIKit dùng để đón nhận các tương tác này. Tại một điểm xác định trong dãy các sự kiện này, UIKit sẽ gọi đối tượng view của chương trình yêu cầu đối tượng view này thay mặt ứng dụng xử lý sự kiện tương tác.

Dưới đây là mô hình tương tác giữa hệ điều hành và đối tượng view.



Hình 5 - Mô hình tương tác giữa hệ điều hành và đối tượng view

Những bước sau đây mô tả lại dãy các sự kiện tuần tự diễn ra khi một tương tác chạm sảy ra trên một màn hình thiết bị iDevice.

Người dùng chạm lên màn hình thiết bị.

Hệ điều hành báo cáo xuất hiện một sự kiện chạm cho UIKit framework.

Framework UIKit đóng gói sự kiện chạm vào một đối tượng kiểu UIEvent và phân phối đến đối tượng view phù hợp (Đối tượng view mà người dùng chạm vào)

Câu lệnh đón nhận sự kiện chạm của đối tượng view có thể phản ứng lại sự kiện trên bằng cách thực hiện những việc sau:

Thay đổi giá trị thuộc tính của view (frame, bounds, alpha…)

Gọi phương thức setNeedsLayout để thông báo đối tượng view hiện tại cần cập nhật layout.

Gọi phương thức setNeedsDisplay hoặc setNeedsDisplayInRect để thông báo đối tượng view này (hoặc đối tượng view con của nó) cần vẽ lại lên màn hình thiết bị.

Thông báo cho đối tượng controller biết sự thay đổi dữ liệu có trong đối tượng view.

Nếu hình dạng của đối tượng view thay đổithì UIKit sẽ cập nhật các view con của nó (đối tượng view có hình dạng thay đổi) theo những luật sau:

Nếu luật autoresize có tác dụng thì UIKit thay đổi từng view con theo luật này.

Nếu đối tượng view có hiện thực phương thức layoutSubview thì UIKit sẽ gọi phương thức này.

Nếu bất kì một phần nào của view được đánh dấu là cần vẽ lại thì UIKit yêu cầu đối tượng view tự vẽ lại chính nó.

Bất kì đối tượng view được cập nhật nào cũng được hợp với phần còn lại củaứng dụng và gởi đến thiết bị hiển thị đồ họa (màn hình thiết bị iDevice) để hiển thị.

Thiết bị hiển thị đồ họa xuất nội dung kết xuất ra màn hình.

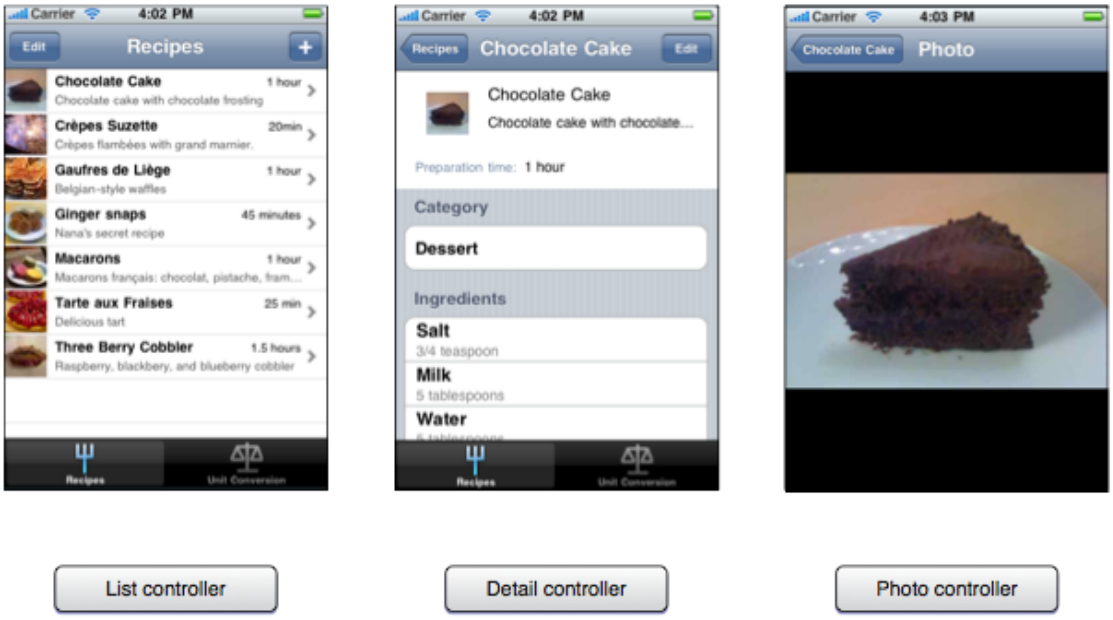
2.3.2 View controller

2.3.2.1 Tổng quan về View controller

View controller cung cấp một cơ sở hạ tầng cơ bản và cần thiết để xây dựng một ứng dụng trên nền tảng iOS. Trong mô hình MVC, đối tượng controller giữ vai trò cầu nối giữa giữ liệu của ứng dụng và giao diện hiển thị (View). Trong ứng dụng trên nền tảng iOS, một đối tượng view controller là một thể hiện của lớp kế thừa từ UIViewController (Định nghĩa trong UIKit). Đối tượng view controller giữ một vai trò qua trọng trong thiết kế và hiện thực ứng dụng trên nền tảng iOS.

Mọi ứng dụng chạy trên thiết bị iDevice luôn bị giới hạn kích thước màn hình dùng để hiển thị nội dung, vì vậy các nhà phát triển luôn phải sáng tạo trong việc truyền đạt thông tin đến người dùng. Một ứng dụng có quá nhiều thông tin cần hiển thị thì cần phải chia nhỏ thành nhiều phần hiển thị trên nhiều màn hình hoặc hiển thị từng phần tại những thời điểm thích hợp. Lúc này đối tượng view controller cung cấp cơ sở hạ tầng để quản lý nội dung muốn hiển thị của view thông qua việc hiển thị hoặc ẩn chúng. Nếu View là đối tượng nắm giữ những thuộc tính (Kích thước view, độ trong của view so với các view khác…) quyết định cách thức hiển thị một thông tin nào đó thì đối tượng View controller là đối tượng điều khiển việc hiển thị thông tin đó lên đối tượng View.

2.3.2.2 Các kiểu View controller



Hình 6 - Một số các View controller mặc định của iOS

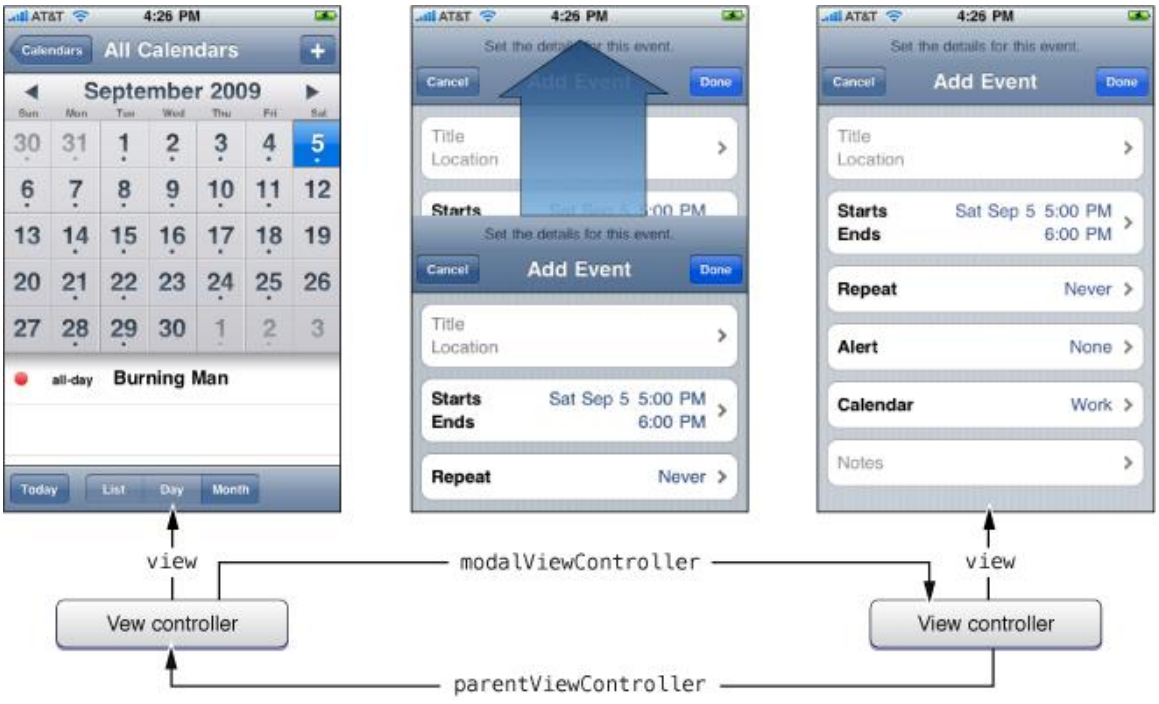
Các view controller của iOS được phân thành 3 nhóm chính: custom, container và modal:

Custom view controller: Là đối tượng controller được định nghĩa nhằm mục đích hiển thị nội dung trên màn hình. Các ứng dụng iOS thường trình bày dữ liệu bằng nhiều tập hợp các view khác nhau. Ví dụ trong cùngmột ứng dụng: một tập hợp các view để hiển thị một list các item trong một table, và một tập hợp khác để hiển thị thông tin chi tiết từng item đó.

Container view controller: Là một loại View controller đặc biệt nhằm quản lý các view controller khác và thiết lập phương thức di chuyển giữa chúng. Một số ví dụ của loại view controller này là các thành phần tab bar, navigational bar,.... Loại view controller này không thể được tùy chỉnh và nhà phát triển ứng dụng phải sử dụng nguyên bản do hệ thống cung cấp.

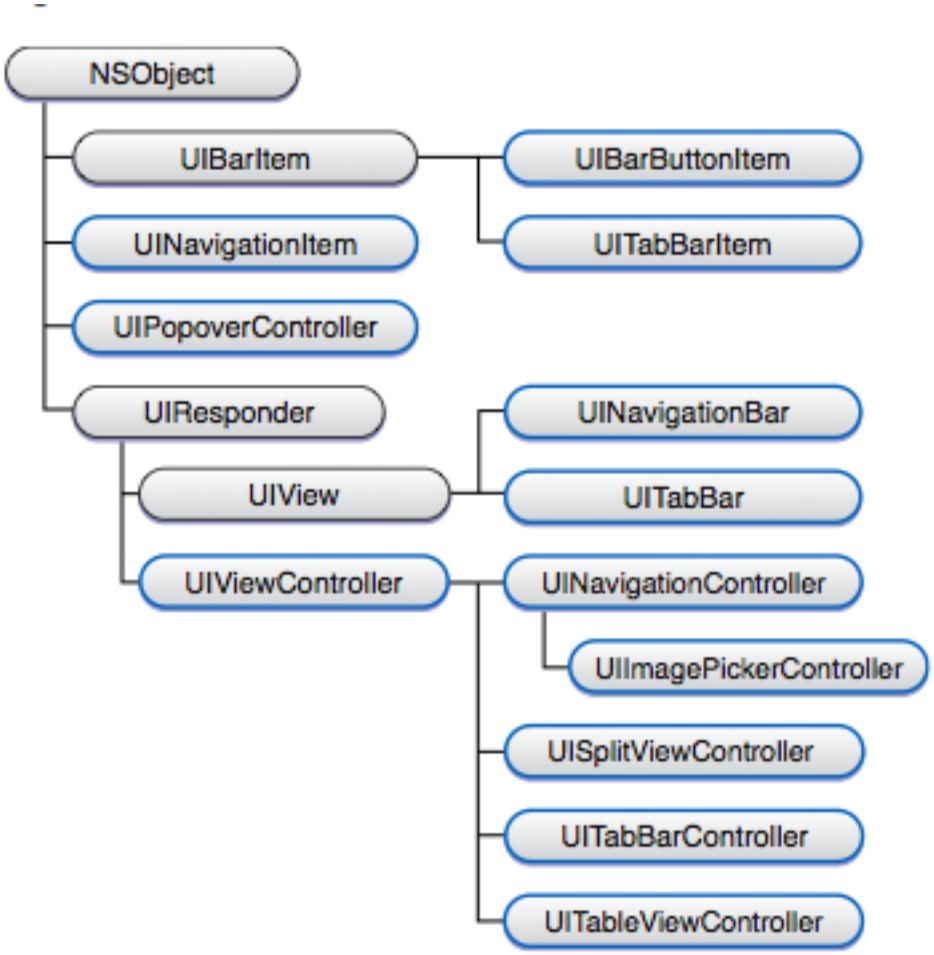
Hình 7 - Tab bar view controller

Modal view controller: Là loại view controller được hiển thị theo cách riêng bởi một view controller khác. Thường được sử dụng để yêu cầu người dùng nhập liệu hay thực hiện một sự lựa chọn rồi nhanh chóng dấu đi trả người dùng về view ban đầu.



Hình 8 - Modal view controller

Sau đây sẽ trình bày chi tiết hơn từng view controlelr quan trọng của iOS:



Hình 9 - Các lớp View controller trong UIKit

Custom View Controller:

Custom View Controller là các đối tượng định hướng chủ đạo của dữ liệu cho mọi ứng dụng iOS. Chúng hỗ trợ các tương tác giữa dữ liệu và các view trình bày dữ liệu đó.

Mỗi custom view controller được tạo nên đều mang trong mình nhiệm vụ quán lý tất cả các view trong một hệ thống trật tự các view (view hierachy). Cần lưu ý:

- Trong iPhone/iPod Touch: các view trong một view hierachy chiếm hết toàn màn hình.
- Trong iPad: các view có thể chỉ chiếm một phần màn hình.

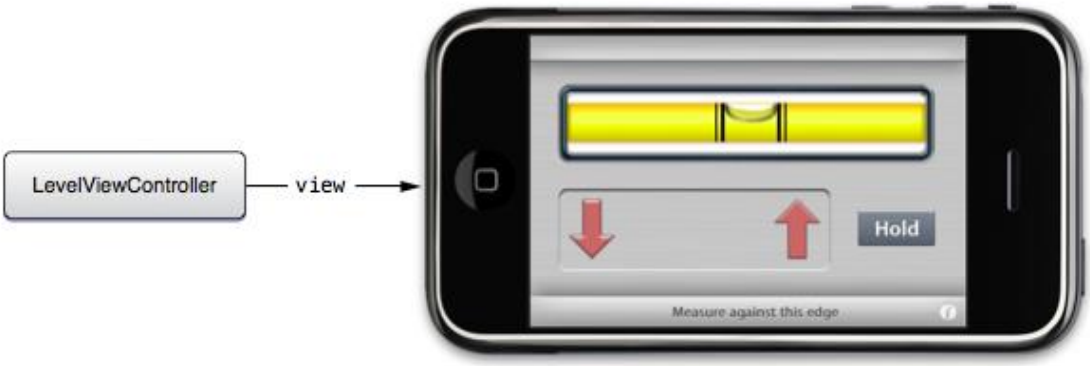
Điểm quan trọng nhất cần ghi nhớ chính là sự liên lạc một-đối-một giữa view controller và các view trong view hierachy. Apple Inc khuyên nhà phát triển ứng dụng không nên sử dụng hàng loạt các view controller để quản lý từng phần khác nhau của

chung một view hierachy,và ngược lại, không nên dùng duy chỉ một view controller cho nhiều màn hình khác nhau.

Nhà phát triển ứng dụng khởi tạo một custom view controller bằng cách subclass lớp UIViewController, với mỗi lớp con chứa các thành phần sau:

- Các biến chỉ tới các đối tượng chứa dữ liệu chuẩn bị được hiển thị.
- Các biến (hay outlet) chỉ đến các đối tượng view chính mà view controller này sẽ tương tác.
- Các phương thức hành động cho các nút nhấn và control trong view hierachy.
- Bất kỳ môt phương thức nào khác cần thiết để hiện thực các hành vi của ứng dụng.

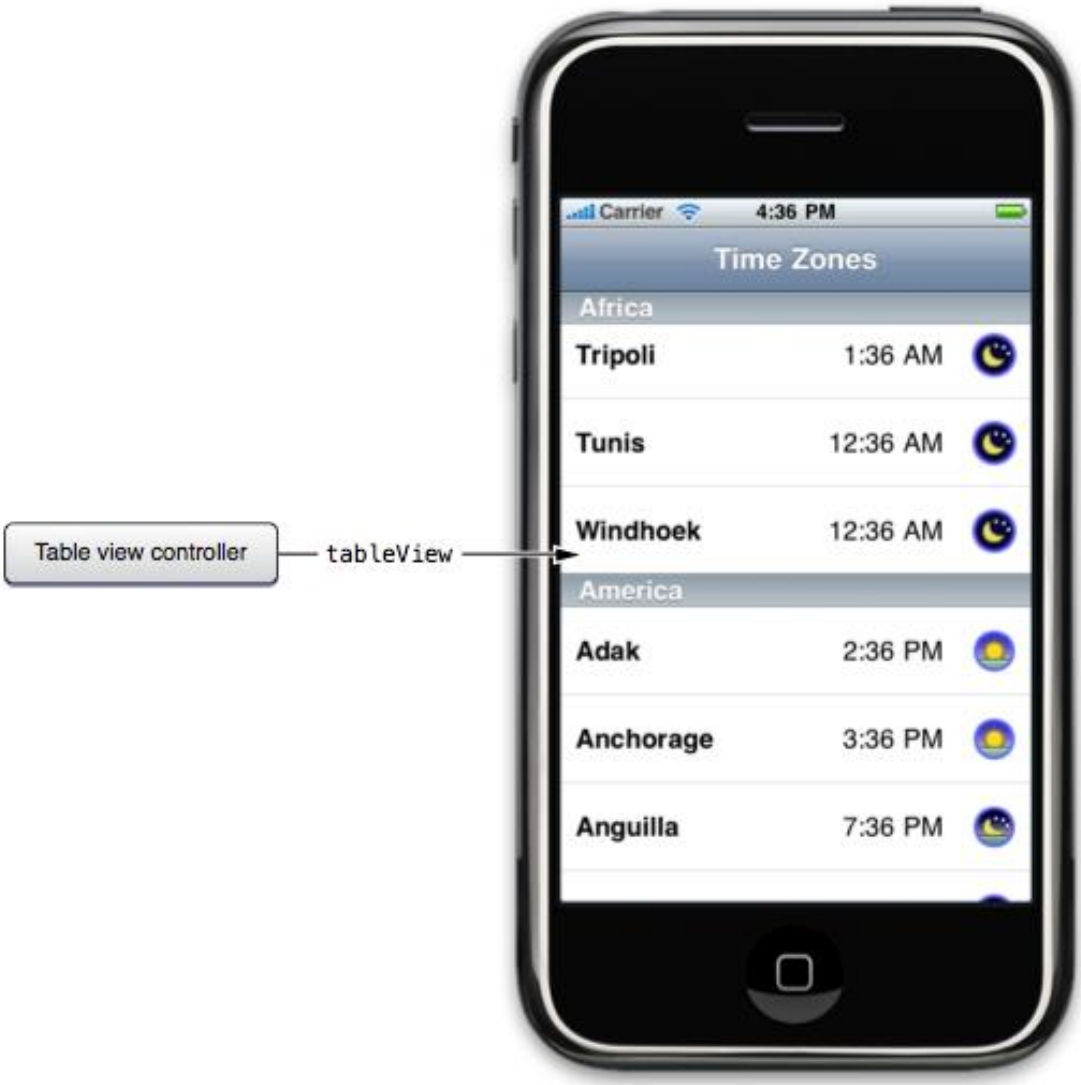
Hình dưới minh họa một custom view controler mẫu. Lớp LevelViewController là lớp con của UIViewController và có nhiệm vụ đo lường độ nghiêng của thiết bị (thông qua accelerometer) và dùng dữ liệu thu được đó để cập nhật đối tượng view tương ứng.



Hình 10 - Custom View Controller LevelViewController

Table View Controller:

Table View Controller, hay UITableViewController, là một dạng custom view controller được dùng để chứa và quản lý dự liệu dạng bảng. Mặc dù nhà phát triển ứng dụng vẫn có thể quản lý các bảng mà không dùng view controller này, Table View Controller giúp giảm bớt công việc phải code các thao tác trên bảng như quản lý chọn item, chỉnh sửa dòng,... ViewController này cung có thể được subclass để tinh chỉnh hay thêm các chức năng mới.

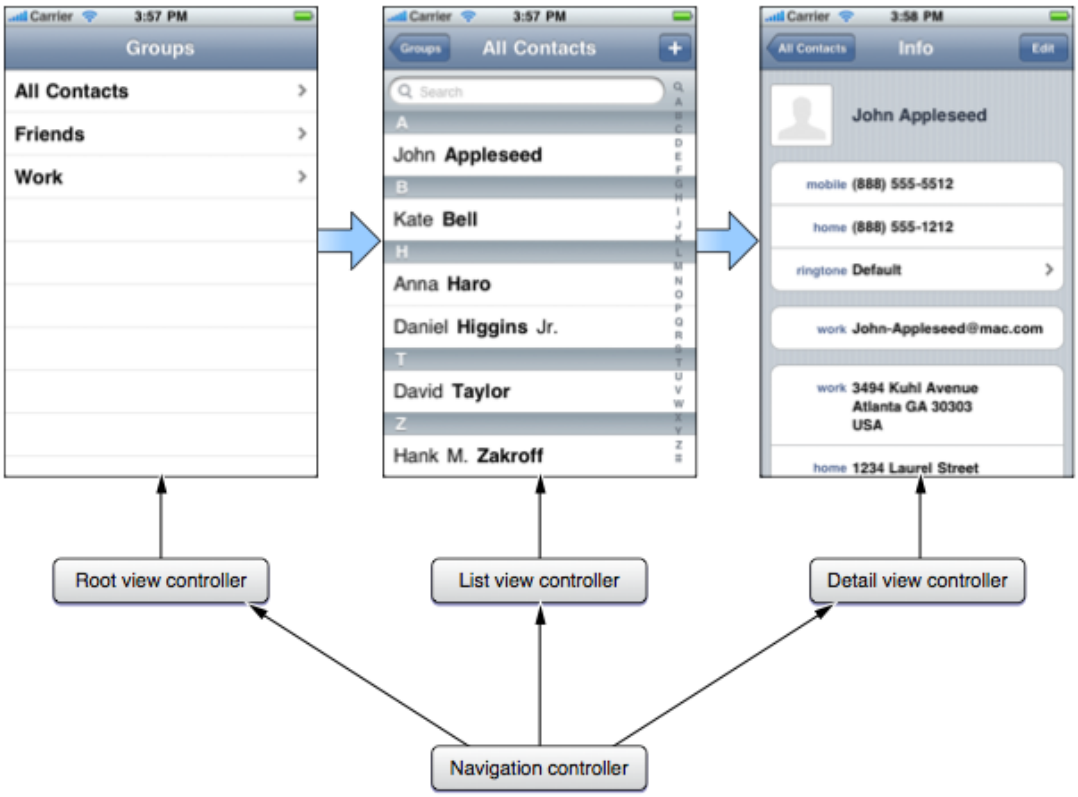


Hình 11 - Quản lý dữ liệu dạng bảng

Navigation controller:

Navigation controller là một loại custom view controller cho phép trình bày dữ liệu dưới dạng tuần tự nhau. Đây là một instance của lớp UINavigationController (iOS quy định không được phép subclass lớp này). Về cơ bản, lớp này cung cấp các phương thức để quản lý một tập hợp các view controller dưới dạng stack, với đáy của stack được coi như điểm bắt đầu và đỉnh trên cùng stack là vị trí hiện hành của người dùng.

Ngoài ra, Navigation controller còn quản lý một navigation bar hiển thị vị trí hiện hành của người dùng, một nút Back để trở về màn hình trước đó, và bất kỳ một control nào khác do nhà phát triển ứng dụng đặt vào.



Hình 12 - Navigation Controller

Tab bar controller:

Tab bar controller là một custom view controller cho phép chia ứng dụng thành nhiều khu vực hay phương thức hoạt động khác nhau. Nó là m6o5t instance của lớp UITabBarCOntroller (Tương tự UINavigationController, chỉ có thể dùng trực tiếp chứ không được subclass).

Lưu ý là nếu số lượng tab trong tab bar vượt quá 5, tab bar sẽ tự động gộp chung cá tab ngoài cùng thành một nhóm và dấu đi trong một tab riêng tên là “More”.

Hình 13 - Tab bar controller trong ứng dụng Clock

Split View Controller:

Split view controller là một custom view controller thường được dùng trong ác ứng dụng iPad có giao diện kết hợp hai phần Tổng quát-Chi tiết. Nó là một instance của lớp UISplitViewController. Nội dung của giao diện chia đôi này được lấy từ 2 view controller do nhà phát triển ứng dụng cung cấp.

Khi thiết bị đang ở phương ngang, một split view controller sẽ hiển thị dữ liệu của cả 2 view controller kia cạnh nhau.

Khi ở phương thẳng đứng, chỉ có một view controller được trình bày, view controller còn lại được dấu đi và chỉ hiện ra khi người dùng nhấn vào một nút popover.

Hình 14 - Split View controller ở phương thẳng đứng và phương ngang

Modal View Controller:

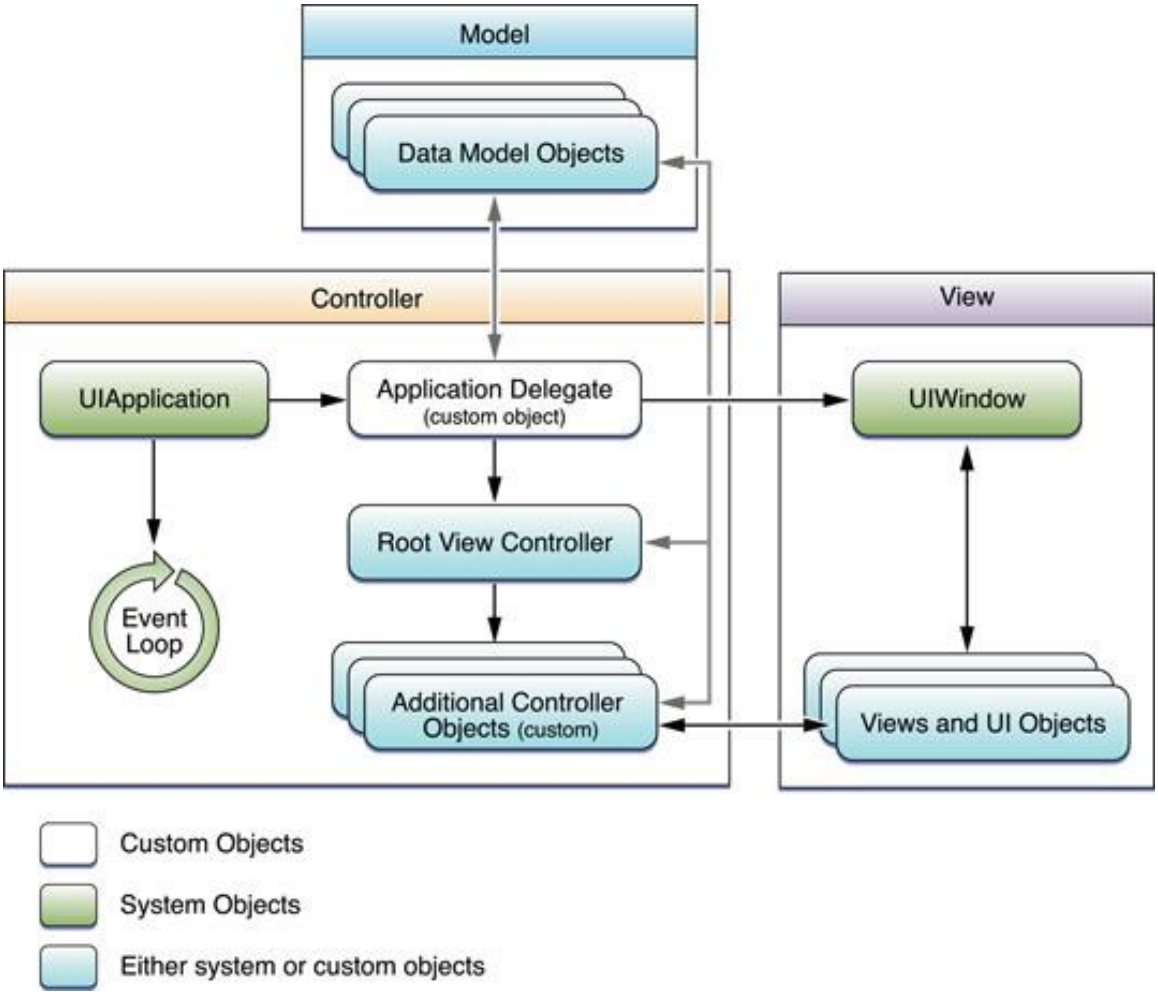
Một modal view controller không hẳn là view controller xác định, mà chỉ là một cách hiển thị bất kỳ một view controller khác tới người dùng. Như thế, bất kỳ một đối tượng view controller nào cũng có thể hiển thị một đối tượng view controller tùy ý khác. Các view controller được trình bày bằng cách này sau khi được người dùng đáp trả xong sẽ được loại đi khỏi màn hình thiết bị và trở về view ban đầu.

Hình 15 - Hiển thị một modal view controller

2.4 Vòng đời một ứng dụng iOS

2.4.1 Vòng đời một ứng dụng iOS

Ngay sau khi người dùng mở ứng dụng, framework UIKit của iOS sẽ đảm nhận gần như toàn bộ các hành vi chủ đạo của ứng dụng. Một ví dụ là cách một ứng dụng iOS liên tục nhận và phản hồi lại các sự kiện từ phía hệ thống. Đối tượng UIApplication đón nhận các sự kiện, nhưng chính các đoạn code do nhà phát triển ứng dụng viết sẽ đảm nhận công việc đáp trả lại phù hợp. Tương tự, trong phần lớn các trường hợp khác khi lập trình iOS, các đối tượng có sẵn của hệ thống sẽ quản lý tiến trình tổng quát và nhà phát triển ứng dụng tự viết code hiện thực các hành vi đặc trưng riêng của ứng dụng.



Hình 16 - Các đối tượng chính của một ứng dụng iOS

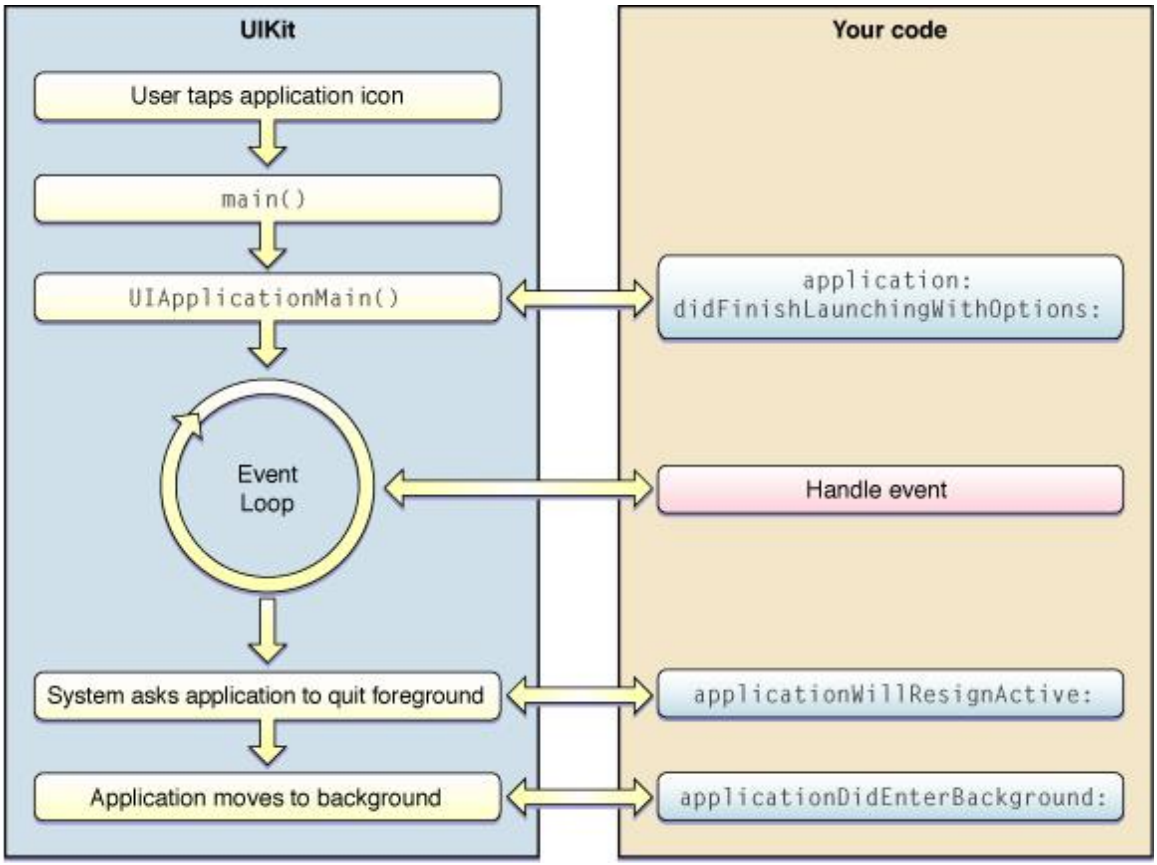
Bảng 2 - Vai trò của các đối tượng trong một ứng dụng iOS

Đối tượng	Mô tả
UIApplication	Quản lý các vòng lặp sự kiện và chỉ định các hành vi cấp độ cao của ứng dụng. Chủ yếu được dùng để tùy biến giao diện bên ngoài của ứng dụng.
Application delegate	Là một đối tượng do nhà phát triển ứng dụng cung cấp vào thời điểm chạy ứng dụng, nằm trong file nib chính. Nhiệm vụ chính là khởi tạo ứng dụng và hiển thị cửa sổ lên màn hình. Application delegate cũng liên tục được lớp UIApplication thông báo khi xảy ra bất kỳ một sự kiện ở cấp độ ứng dụng (chẳng hạn khi tin nhắn SMS gửi tới khiến ứng dụng phải tạm ngừng,…)
Data model	Lưu trữ dữ liệu ứng dụng, bao gồm từ dữ liệu tài chính của một ứng dụng ngân hàng tới một tấm hình của một ứng dụng vẽ hình, hay thậm chí từng nét vẽ trong tấm hình ấy.
View controller	Quản lý việc hiển thị dữ liệu ứng dụng. Về cơ bản lớp này sẽ tạo các view và quản lý các tương tác giữa view và các đối tượng data model. UIViewController là lớp cha của mọi đối tượng view controller. Nó cung cấp các hàm mặc định để xử lý các sự kiện như xoay ngang thiết bị, hiệu ứng chuyển view,… UIKit và một số frameworks khác lại định nghĩa các lớp quản lý giao diện chuẩn của iOS như các nút bấm, tab bar,…
UIWindow	Chỉ định việc trình bày một hay nhiều view trên màn hình thiết bị. Một ứng dụng thay đổi nội dung bên trong một cửa sổ bằng cách thay đổi các view của nó. Ngoài ra, cửa sổ còn có nhiệm vụ chuyển giao các sự kiện tới các view và các view controller tương ứng.
Các view, controls, layers	View là đối tượng sẽ vẽ lên trên một khu vực hình chữ nhật trên màn hình và đáp trả lại các sự kiện xảy ra trong khu vực đó. Control là dạng view đặc biệt để hiển thị các đối tượng giao diện nhất định như nút bấm, text fields,... Bên cạnh các view mặc định do UIKit cung cấp, nhà phát triển ứng dụng có thể tự định nghĩa view bằng cách subclass UIView. Ngoài các view và control, các ứng dụng iOS có thể sử dụng các layer Core Application. Các đối tượng layer thực chất là các đối tượng dữ liệu (data objects) có thể hiển thị các yếu tố đồ họa. View thường xuyên sử dụng các đối tượng layers để vẽ ra màn hình. Nhà phát triển ứng dụng có thể tự tạo các đối tượng

	layer mới để hiện thực các hiệu ứng đồ họa phức tạp.

Vòng đời của một ứng dụng:

Vòng đời của một ứng dụng bao gồm một chuỗi sự kiện xảy ra từ khi ứng dụng vừa được mở tới khi người dùng đóng ứng dụng lại.



Hình 17 - Vòng đời ứng dụng

Trong iOS, vòng đời ứng dụng được sơ lược như sau:

- Người dùng nhấn vào icon ứng dụng tên màn hình Home Screen.

- Hệ thống hiển thị một số hiệu ứng đồ họa chuyển tiếo (transitional graphic effects).

- Hệ thống gọi hàm main, và ứng dụng được khởi động.

- UIKit bắt đầu load file nib chính và chuẩn bị chạy các vòng lặp sự kiện.

Như hình trên miêu tả, ở một số thời điểm quan trọng của ứng dụng, UIKit sẽ gửi các thông điệp tới đối tượng application delegate để cho biết những gì đang xảy ra

trong ứng dụng. Trong suốt vòng lặp sự kiện, UIKit đồng thời cũng chuyển giao các sự kiện tới các view và view controllers do nhà phát triển ứng dụng chuẩn bị sẵn.

Trước phiên bản iOS 4.0, khi người dùng tắt ứng dụng (nhấn nút Home, ứng dụng coi như bị hủy hoàn toàn, không còn được lưu chút nào trong bộ nhớ. Từ 4.0 trở đi, Apple Inc công bố chức năng Multitasking cho phép ứng dụng chạy nền hay được lưu lại ở nền dưới hệ thống (background). Sự thay đổi này dẫn tới mô hình vòng đời phức tạp hơn nhưng lại dễ dùng hơn nhiều cho người dùng và cho phép nhà phát triển ứng dụng mở rộng tiềm năng của ứng dụng.

Hàm main

Cũng như mọi ứng dụng C thông thường, điểm bát đầu của một ứng dụng iOS chính là hàm main. Trong iOS, hàm main được sử dụng tương đối ít, chủ yếu đảm nhận vai trò giao lại quyền kiểm soát cho framework UIKit. Nhà phát triển ứng dụng được khuyên không nên tinh chỉnh hàm main quá nhiều.

```
#import <UIKit/UIKit.h>

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, nil);
    [pool release];
    return retVal;
}
```

Hình 18 - Hàm main của một ứng dụng iOS

Trọng tâm của toàn hàm main trên chính là UIApplicationMain. Hàm này sử dụng 4 tham số để khởi tạo ứng dụng:

- argc và argv: Chứa những mệnh đề ở launch-time được hệ thống chuyển tới ứng dụng.

- Tham số thứ 3: Chỉ định tên của lớp chính của ứng dụng. Đây là lớp sẽ mang trọng trách chạy toàn bộ ứng dụng. Được khuyên gán giá trị là nil để UIKit mặc định sử dụng lớp UIApplication.

- Tham số thư 4: Chỉ định lớp của application delegate. Delegate này có nhiệm vụ quản lý những tương tác cấp độ cao giữa hệ thống và code ứng dụng. Với giá trị là nil, UIKit sẽ tự hiểu là đối tượng application object được đặt trong file nib chính của ứng dụng.

Bên cạnh đó, UIApplication còn đảm nhận công việc load file nib chính dựa trên khóa NSMainNibFile trong file Info.plist của mỗi ứng dụng.

Application Delegate

Đối tượng application delegate có nhiệm vụ theo dõi các hành vi cấp cao của ứng dụng. Như đã nói ở design pattern, delegation là một cơ chế giúp tránh việc phải subclass các đối tượng UIKit phức tạp cũng như override các phương thức trong đó. Giờ đây, nhà phát triển ứng dụng có thể sử dụng đối tượng đó một cách bình thường và đặt code của riêng mình vào trong đối tượng delegate.

Khi bất kỳ một sự kiện đặc biệt nào đó xảy ra, đối tượng phức tạp trên sẽ gửi thông điệp tới các đối tượng delegate, và nhà phát triển ứng dụng có thể sử dụng mối liên hệ này để hiện thực các hành vi như ý muốn.

Các trạng thài của ứng dụng:

- **Not running:**Ứng dụng chưa được khởi động hay vốn đang chạy mà đã bị hệ thống bắt hủy.
- **Inactive:**Ứng dụng đang chạy trên màn hình nhưng chưa nhận được sự kiện nào. THông thường ứng dụng chỉ ở trạng thái inactive một khoảng thời gian rất ngắn trong khi chuyển sang trạng thái khác. Trường hợp ứng dụng inactive lâu hơn là khi người dùng tắt màn hình, hoặc khi có cuộc gọi hay tin nhắn mới tới.
- **Active:**Ứng dụng đang chạy tên nền và đang nhận cái sự kiện được gửi tới.
- **Background**: Ứng dụng ở dưới nền và vẫn đang chạy code. Trừ phi được thiết kế để xử lý một số công việc đặc biệt (download, kết nối,...), còn lại thì một ứng dụng chỉ ở trạng thái này một thời gian ngắn trước khi bị chuyển sang trạng thái suspended. Cần lưu ý là trạng thái này chỉ tồn tại nếu thiết bị chạy iOS 4.0 trở lên. Ngược lại ứng dụng sẽ bị hủy và đưa về trạng thái Not running.

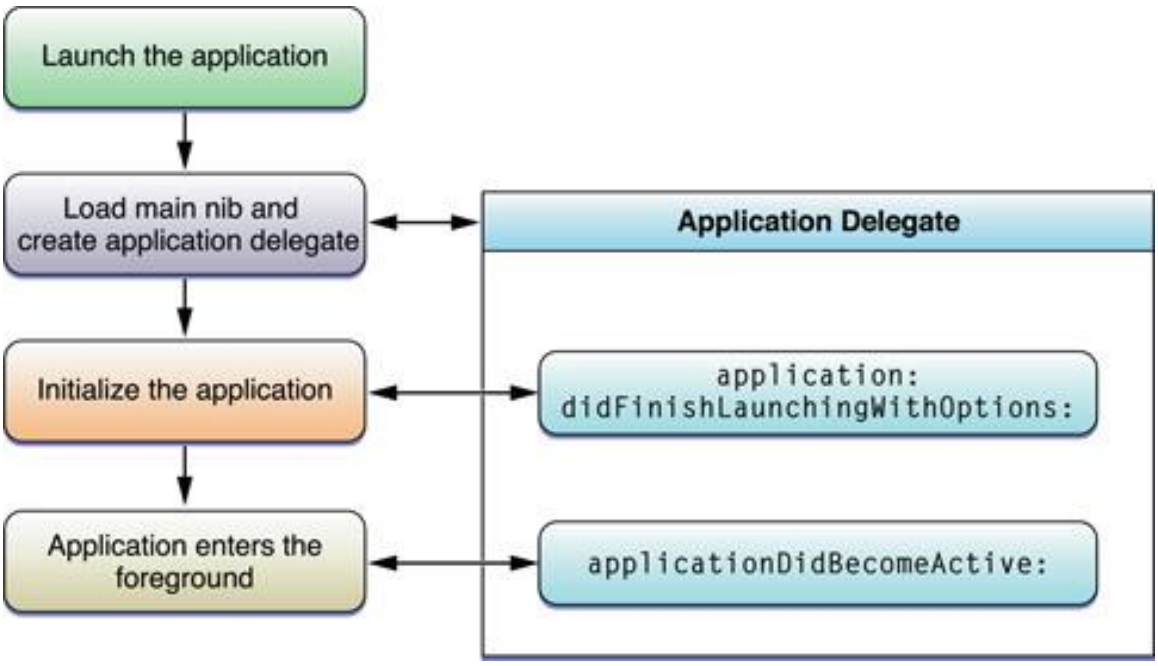
- **Suspended**: Ứng dụng nằm ở dưới nền nhưng đang không chạy bất kỳ một dòng code nào. Ở một số thời điểm nhất định. hệ thống tự động đưa ứng dụng về trạng thái này, và khi điều này xảy ra, ứng dụng gần nhu bị "đóng băng" hoàn toàn. Khi thiết bị bị thiếu bộ nhớ, các ứng dụng ở trạng thái này có thể sẽ bị buộc kết thúc để dành bộ nhớ cho ứng dụng trên nền. Cũng như Background, Suspended chỉ tồn tại với phiên bản iOS 4.0 trở lên.

Để có thể chuyển đổi giữa các trạng thái, hệ thống sử dụng các phương thức như sau trong application delegate:

- application:didFinishLaunchingWithOptions:
- applicationDidBecomeActive:
- applicationWillResignActive:
- applicationWillEnterBackground:
- applicationDidEnterBackground:
- applicationWillEnterForeground:
- applicationWillTerminate:

Sự chuyển giao trạng thái trong một ứng dụng:

- **Mở ứng dụng:**Khi mới được mở, ứng dụng chuyển từng trạng thái Not running sang Active hoặc Background. Trong quá trình khởi động, ứng dụng sẽ gọi các phương thức application:didFisnishLaunchingWithOptions: và một trong hai phương thức applicationDidBecomeActive: hay applicationDidEnterBackground: (Tùy theo trạng thái ứng dụng đang chuyển sang là foreground hay background).



Hình 19 - Đưa một ứng dụng ở trạng thái Active sang Foreground

Trong applicationd delegate trên, phương thức **application:didFinishLaunchingWithOptions:** đảm nhiệm một số trọng trách sau:

- Khởi tạo cấu trúc dữ liệu của ứng dụng.

- Tạo mới hay load cửa sổ chính cũng như các view của ứng dụng.

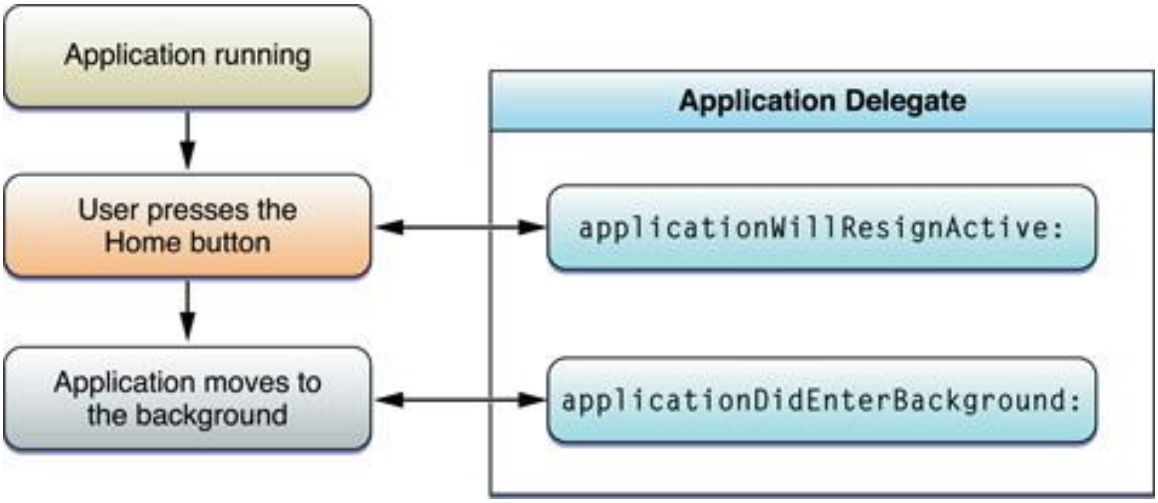
- Kiểm tra các ứng dụng.

- Kiểm tra xem ứng dụng có lưu lại thông số hay tùy biến gì để hồi lại trạng thái cũ trước khi đóng ứng dụng được không.

Do phương thức này được chạy vào thời điểm úng dụng mới được mở, Apple Inc khuyến cáo nên tránh phức tạp hóa **application:didFinishLaunchingWithOptions:** để giảm thiểu thời gian người dùng phải chờ ứng dụng khởi động. Để làm việc này, nhà phát triển ứng dụng có thể tiến hành thực thi các tiến trình không đồng bộ (asynchronously) hoặc đưa các tiến trình phức tạp hơn sang các thread phụ.

Đưa xuống dưới nền:

Khi người dùng nhấn nút Home hay khi hệ thống mở một ứng dụng khác, ứng dụng đang chạy trên nền (**foreground**) sẽ được chuyển sang trạng thái **Inactive** rồi sau đó là trạng thái Background.



Hình 20 - Đưa ứng dụng từ foreground xuống background

Khi một ứng dụng chuyển sang trạng thái background, tất cả các đối tượng core application của ứng dụng đó đều được lưu xuống bộ nhớ thiết bị và luôn sẵn sàng để được sử dụng. Các đối tượng này bao gồm tất cả các đối tượng và cấu trúc dữ liệu cho

lập trình viện thiết kế, cũng như các đối tượng controller, view, window,... Tuy vậy, hệ thống vẫn sẽ giải phóng một số các đối tượng sau:

- Phần nền của tất cả các layer Core Animation, khiến cho các layer này không hiển thị trên màn hình mà vẫn giữ được các thuộc tính của layer. Các đối tượng layer không hề bị giải phóng khi việc này xảy ra.

- Bất kỳ một tham chiếu nào tới các file hình ảnh được lưu trong cache.

- Các cache dữ liệu do hệ thống quản lý.

Phương thức **applicationDidEnterBackground** của delegate sẽ có chính xác 5 giây để thực hiện một tác vụ nào. Quá 5 giây này, ứng dụng sẽ bị hủy ra khỏi bộ nhớ. Tuy có thể sử dụng hàm **beginBackgroundTaskWithExpirationhandler** để đưa tiến trình xuống một luồng thứ hai ở background, phương thức **applicationDidEnterBackground** vẫn nên trở về càng sớm càng tốt.

Khi được đưa xuống **background**, tất cả mọi ứng dụng có hỗ trợ multitasking đều nên thực hiện hai điều sau:

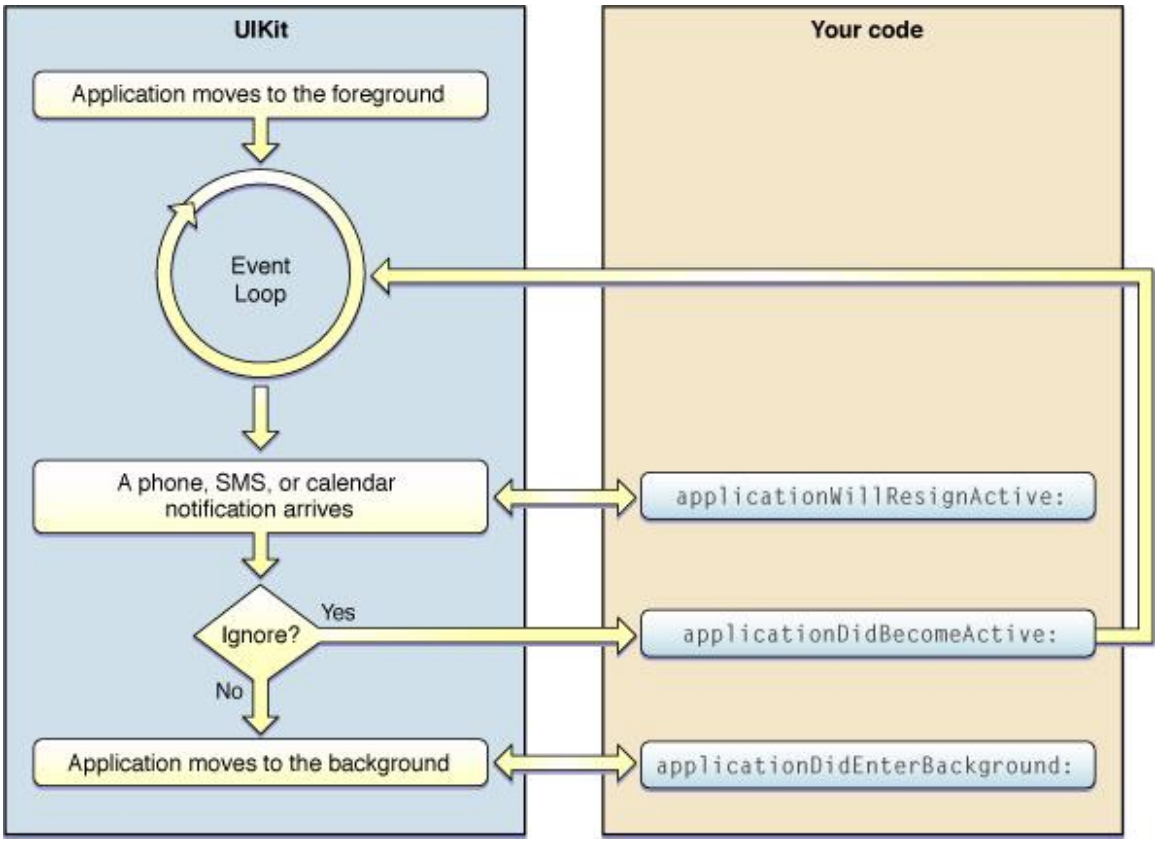
- Chuẩn bị chụp lại màn hình ứng dụng: Khi phương thức **applicationDidEnterBackground** trả về kết quả, hệ thống sẽ chụp một tấm hình giao diện người dùng của ứng dụng và sử dụng hình này cho hiệu ứng chuyển tiếp. Vì thế, nếu bất kỳ một view nào của ứng dụng đang hiển thị thông tin nhạy cảm thì nhà phát triển ứng dụng cần phải giấu đi hay thay đổi nội dung hiển thị.
- Lưu lại dữ liệu người dùng và thông tin trạng thái: Mọi thay đổi chưa được lưu lại nên được ghi lên địa khi ứng dụng được đưa xuống background. Việc này hết sức quan trọng, bởi úng dụng có thể bị hủy đi khi đang ở background.

Trường hợp bị ngắt đoạn:

Khi thiết bị iDevice nhận một cuộc gọi tới hay thông báo **Push**, ứng dụng sẽ buộc bị ngắt đoạn tạm thời, được đưa về trạng thái **Inactive** và sẽ ở mãi trạng thái này tới khi người dùng quyết định đáp trả hay bỏ qua sự kiện ngắt đoạn (ví dụ nghe hay không nghe điện thoại).

- Nếu người dùng đáp trả lại sự ngắt đoạn: ứng dụng được đưa xuống background.

- Nếu người dùng bỏ qua: ứng dụng được khởi tạo lại và có thể trở lại hoạt động ban đầu.



Hình 21 - Xử lý trường hợp ứng dụng bị ngắt đoạn

Từng bước chi tiết của quy trình trên:

- Một sự kiện ngắt đoạn xảy ra (cuộc gọi tới, đồng hồ hẹn giờ,...)
- Hệ thống gọi phương thức applciationWillResignActive của applciation delegate, đồng thời vô hiệu hóa các sự kiện cảm ứng chạm của ứng dụng.
- Hệ thống hiển thị thông báo lên màn hình. Người dùng có thể chọn bỏ qua hay đáp trả lại.
- Nếu người dùng bỏ qua, hệ thống gọi phương thứ **applicationDidBecomeActive** và trở lại tiếp nhận các tương tác cảm ứng chạm.Ngược lại, hệ thống gọi hàm **applicationDidEnterBackground**, ứng dụng được đưa xuống background, lưu lại dữ liệu người dùng và các thông tin khác để có thể hồi lại trạng thái ban đầu sau này.

Tùy theo quyết định của người dùng mà hệ thống có thể sẽ trở về ứng dụng ban đầu khi sự ngắt đoạn kết thúc.

Ví dụ:

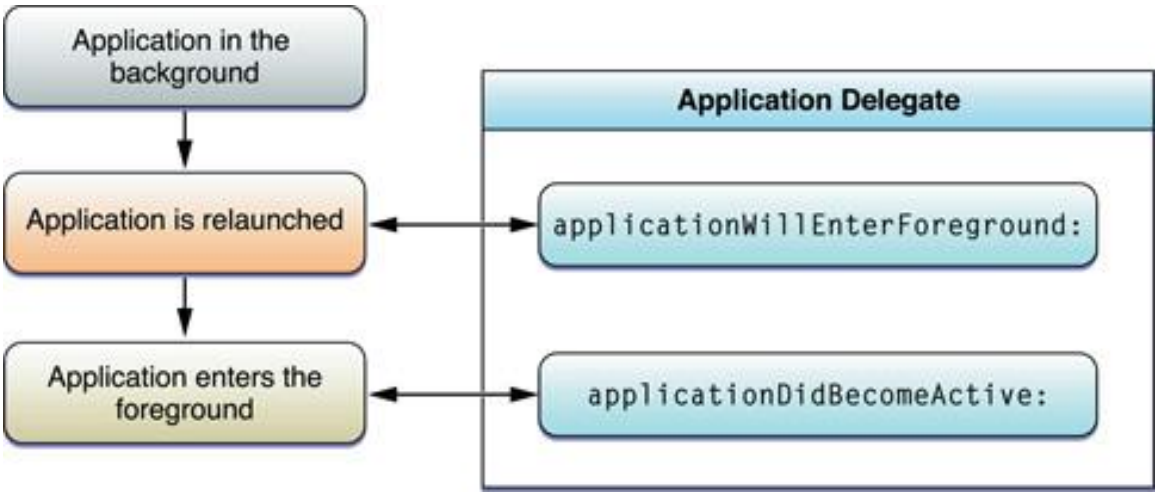
Nếu người dùng nghe điện thoại rồi ngắt cuộc gọi, hệ thống sẽ mở lại ứng dụng ban đầu..

Nếu khi đang nghe mà người dùng trở về Home Screen hay mở một ứng dụng khác, hệ thống sẽ không mở lại ứng dụng ban đầu.

Nếu người dùng nhấn nút Sleep/Wake khi đang sử dụng ứng dụng, hệ thống sẽ gọi hàm **applicationWillResignActive**, ngừng lại việc tiếp nhận sự kiện cảm ứng chạm, và cuối cùng khóa màn hình lại. Khi người dùng mở màn hình trở lại, phương thức **applicationDidBecomeActive** được gọi. Lưu ý là trong khi màn hình đang bị khóa, mọi ứng dụng trên foreground lẫn dưới background vẫn tiếp tục chạy.

Trường hợp phục hồi lại Foreground:

Khi người dùng mở lại một ứng dụng đang chạy dưới Background, hệ thống đưa nó sang từ trạng thái Inactive sang Active, dẫn tới việc các phương thức **applicationWillEnterForeground** và **applicationDidBecomeActive** được gọi tới.



Hình 22 - Chuyển trạng thái từ Background lên Foreground

Khi ứng dụng đang ở trạng thái **Suspended**, hệ thống theo dõi và lưu lại các sự kiện có thể ảnh hưởng lại ứng dụng khi nó được mở lại. Nhờ vậy, ngay khi được khởi động lại, mọi cấu trúc bên trong ứng dụng nên đáp trả lại phù hợp.

Ví dụ: Nếu khi đưa xuống background, thiết bị iDevice đang ở phương dọc, nhưng khi đưa lên ứng dụng lên foreground thì thiết bị lại ở phương ngang, thì giao diện ứng dụng phải ngay lập tức chuyển sang phương ngang một cách tự động.

Trường hợp ứng dụng bị hủy:

Trong một số trường hợp, ứng dụng sẽ bị tự động hủy khỏi bộ nhớ thay vì đưa về background:

- Ứng dụng được xây dựng dựa trên phiên bản iOS thấp hơn phiên bản 4.0.
- Ứng dụng được khởi động trên một thiết bị đang chạy phiên bản iOS thấp hơn 4.0.
- Thiết bị hiện hành không hỗ trợ multitasking.
- Ứng dụng có kèm key **UIApplicationExitsOnSuspend** trong file **Info.plist**.

Nếu ứng dụng đang ở trạng thái foreground hay background, hệ thống sẽ gọi phương thức **applicationWillTerminate**. Nhà phát triển ứng dụng có thể tận dụng phương thức này để tiến hành lưu dữ liệu người dùng hay thông tin trạng thái ứng dụng để chuẩn bị cho lần chạy sau. Phương thức này có 5 giây để thực hiện các tiến trình trên. Sau 5 giây này, ứng dụng sẽ bị buộc hủy khỏi bộ nhớ.

Ngay cả khi ứng dụng được viết với iOS SDK 4 trở lên, thì vẫn có một số trường hợp khác có thể dẫn đến ứng dụng bị hủy mà nhà phát triển ứng dụng cần chuẩn bị trước:

- Người dùng cố tình hủy ứng dụng thông qua giao diện quản lý multitasking.
- Thiết bị cần thêm bộ nhờ cho các ứng dụng khác.
- Các tình huống khác...

Multitasking:

Kể từ phiên bản iOS 4.0 trở về sau, một lúc nhiều ứng dụng có thể được lưu trong bộ nhớ và chạy đồng thời, với một ứng dụng chạy trên foreground và các ứng dụng còn lại nằm dưới background. Để có thể tận dụng hết các ưu điểm của multitasking, các ứng dụng phải được thiết kế cho việc chuyển đổi giữa foreground và background.

Sơ lược:

Thực chất thì ngay từ phiên bản đầu tiên, iOS đã là một hệ điều hành hỗ trợ multitasking, nhưng chỉ đối với các ứng dụng cho chính Apple Inc phát triển.

Ví dụ: Người dùng có thể mở ứng dụng nghe nhạc và sau khi tắt đi chuyển sang ứng dụng khác, ứng dụng vẫn nằm ở dưới background chơi nhạc.

Đến phiên bản 4.0, iOS chính thức hỗ trợ chức năng multitasking cho tất cả các ứng dụng hãng thứ 3. Tuy vậy, multitasking trên iOS vẫn còn mang nhiêu hạn chế, với chỉ 7 dịch vụ cho nhà phát triển ứng dụng sử dụng:

Background audio: Cho phép ứng dụng tiếp tục phát ra âm thanh ở dưới background trong khi các ứng dụng khác đang chạy trên foreground.

Voice over IP: Dành cho các ứng dụng gọi điện thoại qua internet. Với dịch vụ người dùng có thể vừa nghe điện thoại VoIP vừa sử dụng một ứng dụng khác hay khi thiết bị đang bị lock/tắt màn hình.

Background location: Cho phép các ứng dụng định vị bằng sóng GPS có thể tiếp tục chỉ đường cho người dùng ngay cả khi ứng dụng khác đang được dùng trên foreground.

Push notifications: Cho phép các ứng dụng hãng thứ 3, cho dù cò đang được mở hay không, có thể gửi các thông báo đến người dùng gần như tức thời và thông qua các server riêng.

Local notifications: Tương tự Push notifications nhưng các thông báo này không cần server, chẳng hạn đồng hồ báo thức.

Task finishing: Nếu ứng dụng đang thực hiện chưa xong một tiến trình mà bị tắt đi, nó sẽ tiếp tục chạy cho xong ở dưới background.

Fast app switching: Cho phép ứng dụng bị "đóng băng" khi người dùng tắt đi, và hồi trở lại đúng trạng thái ban đầu khi được mở lại.

Với multitasking, tất cả các ứng dụng khi bị tắt đi đều được tự động đưa xuống background và chỉ bị hủy khỏi bộ nhớ nếu hệ thống thiếu bộ nhớ hay người dùng có tình kết thúc thông qua giao diện quản lý multitasking.

Lưu ý là khi ở background, ứng dụng chỉ có thể sử dụng các dịch vụ trên. Bất kỳ một tác vụ nào khác không nằm trong số 7 dịch vụ trên đều bị tạm dừng. Như thế, multitasking của iOS chưa đạt tới mức hoàn chỉnh như các hệ điều hành hiện đại trên máy vi tính. Phươn thức multitasking này, tuy còn mang nhiều hạn chế, lại phù hợp với các thiết bị iDevice:

- Cấu hình máy chưa cao, nhất là bộ nhớ.
- Bản chất của iOS: Ở mỗi thời điểm chỉ có duy nhất một ứng dụng chạy trên màn hình.

Những yêu cầu khi hiện thực ứng dụng multitasking:

- **Bắt buộc:** Đáp trả phù hợp với từng sự chuyển đổi trạng thái. Một ứng dụng multitasking cần phải theo dõi những chuyển biến này để có thể lưu lại trạng thái ban đầu và luân phiên hành vi của nó ứng với từng trường hợp foreground hay background. Không tuân theo quy tắc này sẽ dẫn đến hiện tượng mất dữ liệu và bất ổn định.

- **Bắt buộc:** Làm đúng theo các hướng dẫn (guidelines) do Apple Inc đưa ra cho các úng dụng multitasking.
- **Nên làm theo:** Không bắt buộc: Nếu nhà phát triển ứng dụng muốn ứng dụng thực thi code khi ở dưới background thì cần phải xin quyền được phép tiếp tục chạy.

Để là một ứng dụng multitasking hoàn thiện:

- Các ứng dụng chạy trên foreground luôn được ưu tiên tài nguyên hệ thống. Do đó khi chuyển tiếp giữa foreground và background, ứng dụng cần phải được tinh chỉnh hành vi cho phù hợp. Cụ thể hơn là:
- Không gọi hàm OpenGL ES: Không nên tạo các đối tượng EAGLContext hay thực hiện bất kỳ một lệnh đồ họa OpenGL ES nào khi ứng dụng đang ở background. Nếu làm trái nguyên tắc này, ứng dụng sẽ bị hủy khỏi bộ nhờ ngay lập tức.
- Hủy mọi dịch vụ liên quan tới Bonjour trước khi chuyển sang trạng thái Suspended. Do một ứng dụng bị suspended cũng không thể đáp trả các yêu cầu dịch vụ mạng, tốt nhất nhà phát triển ứng dụng nên đóng các dịch vụ này lại. Nếu không, hệ thống sẽ tự động đóng giùm. (Bonjour: một protocol của Apple Inc cho phép kết nối chia sẽ dữ liệu giữa các thiết bị với nhau)
- Sẵn sàng đối phó với việc mất kết nối trong các socket mạng: Khi ứng dụng đang bị suspended, hệ thống có thể tự động kết thúc các kết nối socket vì nhiều lý do khác nhau. Miễn là nhà phát triển ứng dụng chuẩn bị trước cho việc rớt mạng th ì sẽ không nảy sinh vấn đề gì quá nghiêm trọng. Khi được hồi lại, ứng dụng có thể dễ dàng tạo lại kết nối mới.
- Lưu lại các dữ liệu ứng dụng trước khi chuyển xuống background: Nhằm đề phòng trường hợp ứng dụng bị hủy để giành bộ nhớ cho ứng dụng foreground.
- Giải phóng bộ nhớ không cần thiết khi chuyển xuống background: Do khi thiếu bộ nhớ, hệ thống sẽ kiểm tra xem ứng dụng nào đang lưu cache lớn nhất để ưu tiên kết thúc, nhà phát triển ứng dụng nên giải phóng bớt các cache trong ứng dụng. Như thế sẽ giảm bớt khả năng ứng dụng bị tự động hủy sớm.
- Trước khi bị suspended, nên ngừng ngay việc sử dụng những tài nguyên chia sẻ của hệ thống : Những tài nguyên chia sẻ như thông tin trong Sổ danh bạ (Address Book) hay cơ sở dữ liệu ngày tháng trong Lịch (Calendar) phải được ưu tiên dành cho ứng dụng trên foreground.

- Không nên cập nhật các window và view khi đang ở background: Nhất là khi cả hai đều đang bị dấu đi người dùng không thấy.
- Đáp trả lại các thông báo kết nối và ngừng kết nối đối với các thiết bị ngoại vi: Đối với các ứng dụng có thiết lập kết nối với các thiết bị ngoại vi (thông qua Bluetooth, Wifi,...), hệ thống sẽ tự động cắt những kết nối này khi đưa ứng dụng xuống background. Ứng dụng phải đăng kí nhận những thông báo này để dùng cho việc kết thúc session với các thiết bị khác. Khi được hồi trở lại foreground, một thông báo kết nối tương ứng sẽ được gửi tới và cho phép ứng dụng tạo kết nối trở lại.
- Giải phóng tài nguyên của các cảnh báo dạng popup (alert) không cần thiết khi ứng dụng chuyển xuống background.
- Loại bỏ khỏi view những thông tin nhạy cảm trước kih chuyển xuống background: Khi một ứng dụng chuyển xuống background, hệ thống tự động chụp màn hình bấy giờ của ứng dụng để dùng cho hiệu ứng chuyển tiếp. Cần tránh trường hợp những file hình chụp này chứa những thông tin quan trọng của người dùng.
- Tránh thực thi những tác vụ nặng nề khi đang ở background: So với ở foreground, các ứng dụng ở background được giao ít thời gian hơn để xử lý code. Nếu ứng dụng có nhiệm vụ chính là chơi nhạc hay định vị GPS thì nhà phát triển ứng dụng nên code cho ứng dụng chỉ tập trung làm việc đó, để tránh trường hợp bị hệ thống giảm độ ưu tiên hay thậm chí hủy khỏi bộ nhớ hoàn toàn.

Đối phó với những thay đổi của hệ thống khi ứng dụng đang ở background:

Một ứng dụng background mặc dù không nhận được tất cả các sự kiện quan trọng xảy ra trong hệ thống, nhưng hệ thống vẫn lưu lại các sự kiện có liên quan và xếp vô hàng đợi để sau này gửi tới ứng dụng.Để tránh việc ứng dụng, khi được hồi lại foreground, bị quá tải các thông báo này, hệ thống sẽ tổng hợp các sự kiện lại thành chung một sự kiện duy nhất ứng với mỗi loại.

Ví dụ:

Khi ứng dụng bị suspended, thiết bị đang được để ở phương thẳng đứng.Trong khi ứng dụng đang bị suspended, thiết bị được quay ngang từ bên trái, thẳng đứng trở lại, đảo ngược trên dưới là cuối cùng trở lại phương ngang. Khi được hồi lại, ứng dụng sẽ nận một thông báo duy nhất về sự kiện thay đổi phương chiều của thiết bị rằng: Thiết bị đang ở phương ngang.

Cần lưu ý rằng nếu dùng framework UIKit có sẵn, nhà phát triển ứng dụng không phải quan tâm đến vấn đề phương hướng thiết bị này.

Bảng 3 - Các thông báo được gửi tới ứng dụng chuẩn bị được hồi lại

Sự kiện	Cách thức thông báo
Code ứng dụng đánh dấu một view bị "dơ".	Các cú gọi hàm setNeedsDisplay hoặc setNeedsDisplayInRect: cho view được tổng hợp và lưu lại cho tới khi ứng dụng được trở lại foreground.
Thiết bị ngoại vi được kết nối hoặc ngừng kết nối.	EAAccessoryDidConnectNotification EAAccessoryDidDisconnectNotification
Thiết bị thay đổi phương chiều.	UIDeviceOrientationDidChangeNotification Đồng thời các view controller sẽ tự động cập nhật lại phương chiều của giao diện.
Có sự thay đổi thời gian đáng kể.	UIApplicationSignificantTimeChangeNotification
Thay đổi trạng thái sử dụng pin (từ pin sang sạc, sắp hết pin,...)	UIDeviceBatteryLevelDidChangeNotification UIDeviceBatteryStateDidChangeNotification
Thay đổi trạng thái các file được bảo vệ.	UIApplicationProtectedDataWillBecomeUnavailable UIApplicationProtectedDataDidBecomeAvailable
Một màn hình gắn ngoài được kết nối hay ngừng kết nối.	UIScreenDidConnectNotification UIScreenDidDisconnectNotification
Thay đổi trạng thái màn hình.	UIScreenModeDidChangeNotification
Thay đổi trên những tùy chỉnh của ứng dụng trong Settings.	NSUserDefaultsDidChangeNotification
Thay đổi ngôn ngữ hệ điều hành.	NSCurrentLocaleDidChangeNotification

Các thông báo này sẽ được gửi tới ứng dụng trước khi xảy ra bất kỳ một sự kiện chạm hay tương tác nào từ phía người dùng.Với phần lớn trường hợp, ứng dụng sẽ xử lý các sự kiện này nhanh tới mức người dùng không cảm thấy phải chờ đợi.

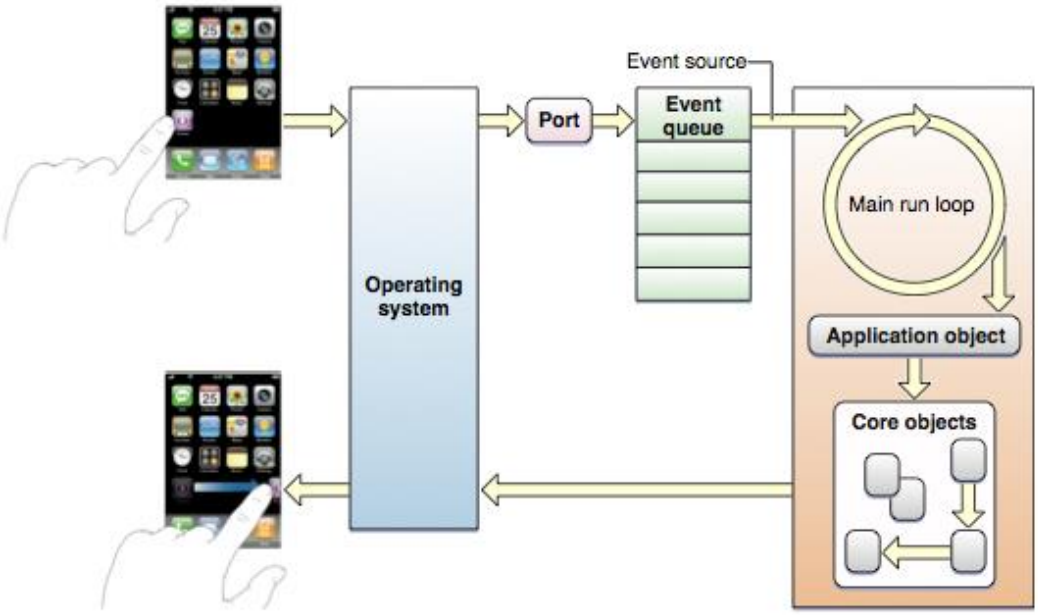
Đặc biệt, tuy không được khuyên làm, nhưng nếu muốn, al65p trình viên cũng có thể chỉ định ứng dụng của mình không tham gia vào quy trình multitasking của iOS. Khi đó, ứng dụng nếu bị tắt thì sẽ ngay lập tức bị hủy khỏi bộ nhớ. Công việc này được thực hiện bằng cách thêm key UIApplicationExitsOnSuspend vào file Info.plist của ứng dụng và gán cho giá trị YES.

Hệ thống xử lý sự kiện:

Hệ thống xử lý sự kiện trong iOS đảm nhiệm trọng trách theo dõi – nhận diện các tương tác cảm ứng chạm từ phía người dùng và chuyển tới các ứng dụng. Phần lớn các sự kiện này được gửi thông qua đối tượng UIApplication, từ đó chúng được xếp vô các hàng đợi và phân phối tới các thành phần khác của ứng dụng.

Khi bắt đầu mở một ứng dụng, hệ thống sẽ tạo ra một tiến trình và một luồng duy nhất cho ứng dụng. Đây cũng là luồng chính của ứng dụng. Chính tại luồng này mà đối tượng UIApplication sẽ khởi tạo vòng lặp chính (main run loop) và tinh chỉnh các đoạn code xử lý sự kiện.

Khi các sự kiện tương tác cảm ứng chạm xảy ra, chúng sẽ được xếp hàng đợi cho tới khi ứng dụng xử lý xong từng sự kiện một. Chi tiết hơn, việc xử lý này sẽ xảy ra trong các đối tượng (view, view controller,…) khác nhau vàứng với từng loại sự kiện.



Hình 23 - Quy trình xử lý sự kiện trong vòng lặp chính

Phần lớn các sự kiện gửi tới một ứng dụng sẽ được bao gói trong một đối tượng sự kiện, bản thân nó là một thực thể của lớp UIEvent. Nếu là sự kiện cảm ứng chạm thì

đối tượng sự kiện sẽ bao gồm một hay nhiều đối tượng UITouch tượng trưng cho các ngón tay người dùng chạm lên màn hình.

Đảm nhận công việc chuyển giao các sự kiện này là các đối tượng responder, cũng là các thực thể của lớp UIResponder. Thực tế thì tất cả các lớp UIApplciation, UIViewController, UIWindow và UIView đều là lớp con của UIResponder. Khi m6o5t sự kiện xảy ra, ứng dụng sẽ gửi sự kiện tới đối tượng UIWindow, nơi mà sự kiện xảy ra. Sau đó đối tượng window này sẽ gửi tiếp tới responder đầu tiên.

Ví dụ: Với sự kiện cảm ứng chạm, responder đầu tiên chính là đối tượng view (UIVIew) tương ứng.

Nếu responder đầu tiên không đươc tìm thấy, sự kiện sẽ được gửi tới responde tiếp theo, thường là một view cha hay view controller.Nếu vẫn không thể giải quyết được thì sự kiện sẽ tiếp tục được gửi tới responder tiếp theo, tạo thành chuỗi responder. Nếu cuối cùng vẫn không xong thì sự kiện đó sẽ bị loại bỏ đi và coi như chưa hề xảy ra.

2.4.2 Vòng đời View controller

Vòng đời View controller

Cũng như các controller khác, một View controller bắt đầu vòng đờicủa nó với hàm alloc và khởi tạo. Dưới đây là hàm khởi tạo mặc định của UIViewController:

(id)initWithNibName:(NSString *)nibName bundle:(NSBundle *)aBundle;

Theo đó:

UIViewController sẽ bắt đầu tạo view từ file xib có tên là giá trị củanibName. Nếu nibName có giá trị nil thì UIViewController sẽ tự động sử dụng file xib nào trùng tên với class hiện hành.

Bundle cho phép khai báo sẽ dùng một hay nhiều file xib (Giả sử nếu ứng dụng có riêng hai giao diện tiếng Anh và tiếng Việt).

Gán giá trị nil cho aBundle để tự động tìm file xib trong thư mục Resources mặc định của một dự án Xcode.

Phần lớn các ứng dụng iOS sẽ khởi tạo UIVIewController bằng hàm init với nibName và aBundle đều là nil.

Sau khi khởi tạo UIViewController, tiếp theo hàm viewDidLoad sẽ được gọi, cũng là nơi nhà phát triển ứng dụng có thể thực thi các đoạn mã cài đặt ban đầu cho ứng dụng:

(void)viewDidLoad;

Trước khi view xuất hiện trên màn hình thiết bị, hệ thống sẽ báo trước với hàm viewWill Appear:

(void)viewWillAppear:(BOOL)animated;

Khi gọi hàm này, các bounds của ứng ụng sẽ được cài đặt

Cần lưu ý là mặc dù view của ứng dụng thường chỉ được load một lần, nó có thể liên tục được dấu đi và hiển thị lại. Do đó, không nên đặt vào đây các phương thức lẽ ra

Tương tự, khi view sắp bị đưa ra khỏi màn hình, hệ thống cũng sẽ thông báo với hàm viewWillDisappear:

(void)viewWillDisappear:(BOOL)animated

Tương ứng với viewWillAppear và viewWill Disappear là hai hàm “Did”:

(void)viewDidAppear:(BOOL)animated;

(void)viewDidDisappear:(BOOL)animated;

Chi tiết các hàm trên được mô tả trong bảng sau:

Bảng 4 - Các hàm cấp phát và giải phóng bộ nhớ

Chức năng	Tên phương thức	Ghi chú
Cấp phát cấu trúc dữ liệu tối quan trọng của view controller.	Các phương thức khởi tạo.	Phương thức khởi tạo dosử dụng phảinhà phát triển ứng dụng.
Tạo các đối tượng view.	loadView	Chỉ cần override phương thức loadView này nếu nhà phát triển ứng dụng muốn tự code các view một cách thủ công. Ngược lại, nếu nhà phát triển ứng dụng dùng file nib để load các view thì chỉ cần gọi phương thức initWithNibName: bundle: để khởi tạo view controller tương ứng.
Cấp phát bộ nhớ hay nạp dữ liệu để hiển thị lên view.	viewDidLoad	Bất cứ dữ liệu nào có tham chiếu tới các đối tượng view đều phải được tạo hay nạp ở phương thức này.

Giải phóng các tham chiếu tới các đối tượng view.	viewDidUnload dealloc	Nếu có retain bất kỳ một đối tượng view hay biế số nào trong lớp của ứng dụng thì phải giải phóng ngay khi không cần nữa. Chắc chắn nhất nên đặt các biến hay outlet về giá trị nil.
Giải phóng dữ liệu không cần thiết nữa.	viewDidUnload	Dành cho những dữ liệu không còn cần thiết và có thể được tạo lại dễ dàng.
Đáp trả lại các thông báo thiếu bộ nhớ.	didReleaseMemory Warning	Dùng để giải phóng bất kỳ các cấu trúc dữ liệu nào có liên kết với view controller mà vẫn còn lại sau hàm viewDidUnload.
Giải phóng các cấu trúc dữ liệu quan trọng mà view controller cần tới.	dealloc	Dùng trong trường hợp view controller vẫn còn các outlet hay biến số với giá trị khác nil ma chưa được giải phóng.

2.5 Các mẫu thiết kế (design pattern) được sử dụng trong hệ điều hành iOS

2.5.1 Tổng quan các design patterns cơ bản

Do có nền tảng thừa hưởng từ các API Cocoa của Mac OS X, framework UIKit của iOS sử dụng khá nhiều các design pattern của Cocoa. Để có thể thiết kế các ứng dụng iOS hoàn chỉnh, việc hiểu rõ các design patterns này là tối quan trọng. Dưới đây là danh sách các designpatterns được dùng trong iOS:

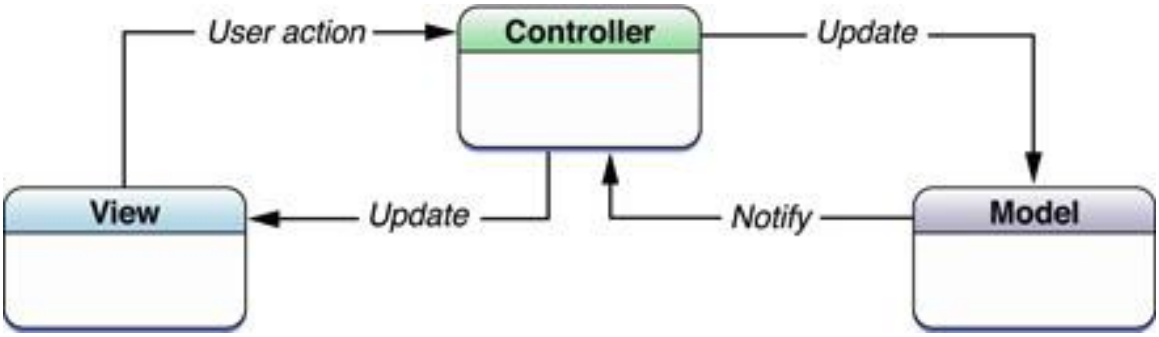
- Model-View-Controller (MVC).
- Block objects.
- Delegation.
- Target-Action.
- Managed memory model.
- Threads và lập trình đồng thời.

2.5.2 Chi tiết các design pattern

2.5.2.1 Model-View-Controller (MVC)

MVC pattern là một mô hình xây dựng phần mềm (Định nghĩa pattern bắt nguồn từ mẫu kiến trúc trong ngành xây dựng) được thiết kế để phân tách các đối tượng trong ứng dụng thành ba lớp (layer): lớp Model, lớp View, lớp Controller. Mô hình ba lớp trên dùng để thiết lập sự tách biệt trong phát triển (Từng lớp được phát triển riêng biệt), kiểm tra (Kiểm tra tách biệt từng đối tượng) và bảo trì (Bảo trì tách biệt từng lớp đối tượng).

Việc áp dụng mô hình MVC pattern trao quyền cho mỗi đối tượng tồn tại trong ứng dụng một trong ba vai trò tách biệt nhau: model, view hoặc controller. Đồng thời mô hình MCV cũng định ra phương thức liên lạc, truyền tải thông tin giữa các đối tượng. Việc áp dụng mô hình MVC sẽ giúp các đối tượng có thể dễ dàng tái sử dụngvà giảm bớt những khó khăn khi muốn mở rộng ứng dụng. Hơn thế nữa do các công nghệ nền tảng của iOS được thiết kế dựa trên mô hình MVC nên việc hiểu rõ và áp dụng tốt mô hình MVC là cực kì quan trọng với các nhà phát triển ứng dụng iOS.



Hình 24 - Mô hình Model-View-Controller

Đối tượng Model

Các đối tượng thuộc lớpModel có nhiệm vụ quản lý hành vi và dữ liệu của miền ứng dụng, đối tượng thuộc lớp Model có trách nhiệm trả lời những yêu cầu lấy thông tin về trạng thái (Giá trị các thuộc tính) của đối tượng Model (Thường thì yêu cầu này sẽ đến từ các đối tượng thuộc lớp View), trả lời những yêu cầu thay đổi trạng thái của đối tượng (Yêu cầu này thường đến từ đối tượng thuộc lớp Controller).

Các đối tượng thuộc lớp này có thể có quan hệ một-nhiều và nhiều-nhiều với nhau. Nếu dữ liệu là một phần của trạng thái tĩnh của mỗi ứng dụng thì người phát triển ứng dụng nên tập trung thành các đối tượng model sau khi các dữ liệu này được nạp vào ứng dụng.

Tốt nhất thì một đối tượng model không nên có mối liên hệ xác định với các đối tượng view có nhiệm vụ hiển thị và cho phép người dùng thay đổi dữ liệu đó.

Vấn đề liên lạc: Ở view layer, những tác động từ người dùng có gây nên sự thay đổi dữ liệu sẽ được liên lạc thông qua một đối tượng controller và dẫn tới sự hình

thành hay cập nhật một đối tượng model. Khi có thay đổi gì, đối tượng model sẽ báo cho đối tượng controller biết để cập nhật các đối tượng view phù hợp.

Đối tượng View

Mỗi đối tượng view là một đối tượng mà người dùng có thể tận mắt nhìn thấy (Tham khảo thêm phần các thành phần chính của một ứng dụng iOS). Đối tượng view này có nhiệm vụ vẽ chính nó lên màn hình thiết bị và đáp trả lại các tương tác từ người dùng.

Mục tiêu chính của của đối tượng view là hiển thị dữ liệu từ các đối tượng model và cho phép thay đổi dữ liệu đó. Thoạt trông có liên hệ mật thiết với nhau, nhưng trong một ứng dụng ứng dụng mô hình MVC, các đối tượng view và model thường được tách biệt với nhau khá rõ.

Do thường xuyên được tái sử dụng và tái tinh chỉnh, các đối tượng view giúp tạo nên sự nhất quán giữa các ứng dụng. Framework UIKit của iOS cung cấp rất một tập hợp các lớp (Class) view dùng để hiện thị dữ liệu trên màn hình thiết bị. Các nhà phát triển ứng dụng iOS có thể sử dụng các kiểu view mà framework UIKit cung cấp, tùy biến các kiểu view sẵn có hoặc tự định nghĩa một kiểu view riêng.

Vấn đề liên lạc: Các đối tượng view nhận biết các tương tác làm thay đổi trên dữ liệu được hiện thị trên nó và thông báo sự thay đổi dữ liệu đến đối tượngmodel thông qua các đối tượng controller, và chuyển giao các thay đổi do người dùng, cũng thông qua các đối tượng controller, tới các đối tượngmodel.

Đối tượng Controller

Trong một ứng dụng áp dụng mô hình MVC, mỗi đối tượng controller đóng vai trò trung gian giữa các đối tượng view và model. Nói cách khác, chính nhờ thông qua các đối tượng controller này mà hai loại đối tượng còn lại có thể biết được sự thay đổi của nhau. Ngoài ra, các đối tượng controller còn làm công việc cài đặt, điều phối các tác vụ và quản lý vòng đời các đối tượng khác.

Vấn đề liên lạc: đối tượng controller biên dịch tác động của người dùng trên các đối tượng view và chuyển giao dữ liệu mới được thêm vào hay sửa đổi tới các đối tượng thuộc lớp model. Ngược lại, khi các đối tượng model thay đổi thì đối tượng controller sẽ đưa dữ liệu của đối tượng model tới các đối tượng view để hiển thị lên màn hình thiết bị.

2.5.2.2 Block object

Block object là một cấu trúc C (C-level language construct) mà nhà phát triển ứng dụng có thể kết hợp vào phần code C hoặc Objective-C. Một block object đơn giản là một hàm nặc danh (anonymous funtion) đi kèm với dữ liệu cần sử dụng trong đoạn chương trình trên, đôi khi với những ngôn ngữ khác nó được biết đến với tên closure hoặc lambda. Block rất hữu dụng trong trường hợp cần tạo callback (Tham khảo thêm tạiđây) hoặc tại những nơi mà nhà lập trình cần một cách dễ dàng để kết hợp phần code thực thi và dữ liệu đi kèm. Trên hệ điều hành iOS block thường được sử dụng trong các trường hợp sau:

+ Thay thế cho delegate và delegate methods.

+ Hiện thực một bộ xử lý hoàn thiện (completion handler) cho những tác vụ chỉ dùng một lần.

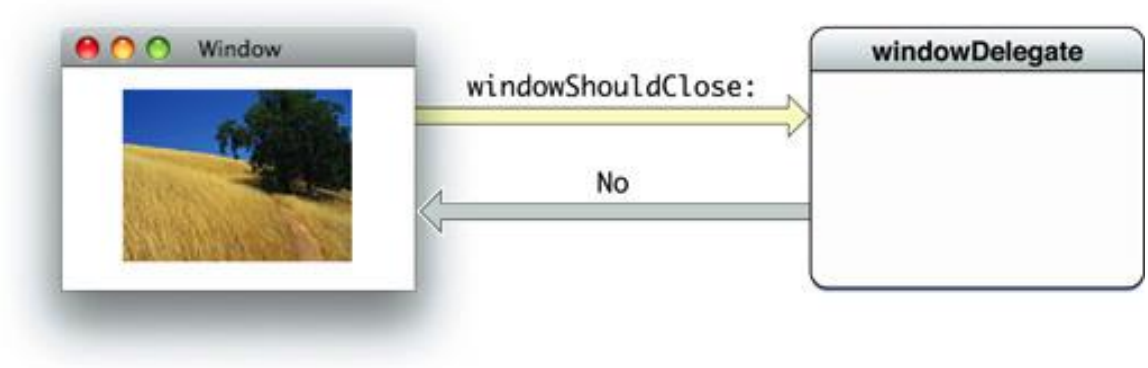
+ Hiện thực những tác vụ không đồng bộ.

+ Tạo callback.

2.5.2.3 Delegation

Delegation là một design pattern đơn giản nhưng đem lại hiệu quả cao. Theo đó, trong một ứng dụng, một đối tượng sẽ thực thi một công việc thay mặt một đối tượng hoặc phối hợp cùng một đối tượng khác. Đối tượng delegating (Đối tượng cần nhượng quyền) sẽ ghi nhớ lại đối tượng còn lại, gọi là đối tượng delegate (Đối tượng được nhượng quyền), và tới thời điểm thích hợp sẽ gửi một thông điệp tới nó. Thông điệp này sẽ cho đối tượng delegate biết về một sự kiện mà đối tượng delegating sắp sửa xử lý hay đã xử lý. Delegate có thể đáp trả lại bằng cách cập nhật giao diện hay trạng thái của nó hay của một đối tượng trong ứng dụng.

Một ví dụ của đối tượng delegating là một thể hiện của lớp NSWindow trong framework AppKit của Mac OS X. NSWindow khái báo một protocol, trong đó có một phương thức là windowShouldClose. Khi người dùng nhấn chuột vào nút X để đóng cửa sổ, đối tượng window sẽ gửi thông điệp windowShouldClose tới deletagte của nó để yêu cầu xác nhận sự kiện đóng cửa số. Delegate sẽ trả về một giá trị Boolean, thông qua đó kiểm soát hành vị của đới tượng window đó.



Hình 25 - Delegation với NSWindow

Lợi ích chính của việc dùng delegation là cho phép dễ dàng tùy biến hành vi của hàng loạt đối tượng từ bên trong một đối tượng duy nhất.

Delegation và các framework Cocoa

Đối tượng delegating thường sẽ là một đối tượng framework (UIKit, AppKiit,...) và delegate là một đối tượng controller do nhà phát triển ứng dụng tạo nên.

Data source

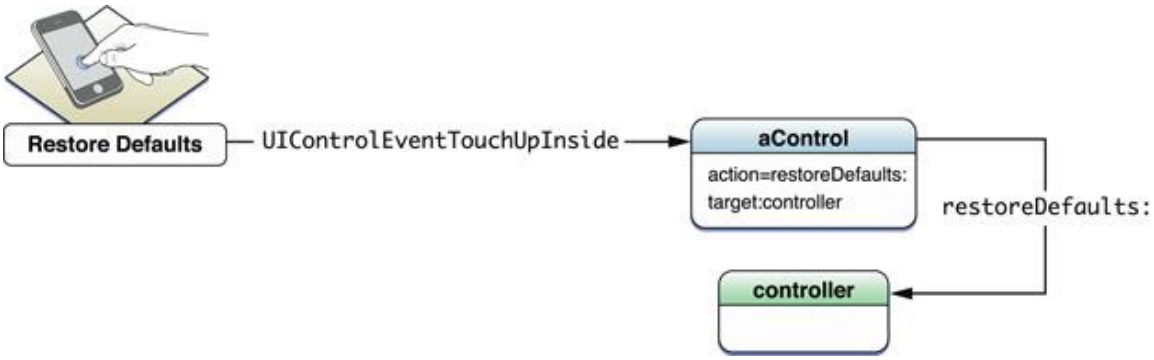
Data source gần như giống hệt delegate, nhưng khác ở mối quan hệ với đối tượng delegating: Thay vì được giao quyền kiểm soát giao diện người dùng, một data source được giao qyền kiểm soát dữ liệu.

Đối tượng delegating, thường là một đối tượng view như table view (Một dạng view dùng để thể hiển dữ liệu ở dạng danh sách trê bảng (table)), tham chiếu tới data source của nó và thỉnh thoảng sẽ yêu cầu dữ liệu để hiển thị.

Một data source, cũng như một delegate, phải tuân theo một protocol và thực thi đầy đủ các phương thức của protocol đó. Các data source cũng đảm nhiệm vai trò quản lý bộ nhớ của các đối tượng model được giao tới delegating view.

2.5.2.4 Target-Action

Trong design pattern Target-Action, một đối tượng sẽ lưu trữ thông tin cần thiết để gửi thông điệp tới một đối tượng khác khi một sự kiện nào đó xảy ra. Thông tin được lưu trữ bao gồm 2 loại dữ liệu: một action selector xác định phương thức sẽ được gọi, và một target (mục tiêu), tức đối tượng sẽ nhận thông điệp. Thông điệp gửi đi được gọi là một action message, và có thể là bất kỳ một đối tượng nào, kể cả đối tượng của framework, nhưng thường thì thông điệp sẽ là một custom controller.



Hình 26 - Ví dụ Target-Action

Một phương thức action phải tuân theo kiểu khai báo như sau:

-(IBAction)doSomething:(id)sender;

Tiếp theo, người nhà phát triển ứng dụng sẽ gán các action và các target nhất định với công cụ Interface Builder hoặc viết code bằng tay, nhưng trước hết cần phải khai báo action này trong file header của lớp mà thể hiện của nó sẽ nhận thông điệp.

Nếu target của thông điệp được gán là nil, ứng dụng sẽ tự quyết định target tại runtime và gửi thông điệp trước tiên tới responder đầu tiên rồi lên dần theo thứ tự responder tới khi được xử lý hoàn chỉnh.

2.5.2.5 Memory Management

Với bất kì nhà phát triển ứng dụng nào thì việc quản lý bộ nhớ thực thi của chương trình luôn là một yêu cầu cần thiết để đảm bảo ứng dụng sử dụng một cách hiệu quả nhất bộ nhớ thiết bị. Để đảm bảo được điều này, bất kì một ứng dụng viết bằng ngôn ngữ Objective-C luôn phải đảm bảo rằng mọi đối tượng được xóa khỏi bộ nhớ ngay khi mà nó không cần được dùng đến nữa.

Trong những hệ thống phức tạp, thật sự rất khó để xác định được chính xác khi nào không còn cần một đối tượng nào đó nữa. Lớp **Cocoa touch** định nghĩa một số luật và nguyên lý giúp quản lý bộ nhớ hiệu quả hơn.

Luật cơ bản

- Chỉ **release** hoặc **autorelease** những đối tượng mình sở hữu.
    - Một con trỏ được cho là chiếm quyền sở hữu với một đối tượng nếu nó tạo ra đối tượng đó với các phương thức bắt đầu bằng “alloc” hoặc “new” hoặc các phương thức chứa các từ “copy” hoặc gởi thông điệp retain yêu cầu giữ một đối tượng.

- Sử dụng **release** hoặc **autorelease** để từ bỏ quyền sở hữ với một đối tượng. **autorelease** có nghĩa là gởi thông điệp **release** đến đối tượng tại một thời điểm nào đó trong tương lai.

Những luật dưới đây phát sinh từ luật cơ bản ở trên

- Hệ quả của luật cơ bản: Nếu muốn con trỏ sở hữu một đối tượng như là một thuộc tính được lưu trữ dưới dạng một biến thì phải gởi thông điệp **retain** hoặc **copy** đối tượng đó.
- Khi một đối tượng được tạo ra (Bằng các phương thức tạo mới đối tượng) thì chắc chắn nó sẽ được giữ lại trong phạm vi phương thức mà đối tượng đó được tạo ra, và phương thức (nơi nhận đối tượng mới được tạo ra) có thể trả đối tượng này về cho đối tượng đã gọi phương thức (nơi nhận đối tượng được tạo ra) này.

Quyền sở hữu đối tượng và hủy đối tượng

Một thách thức lớn với các nhà phát triển ứng dụng là sử dụng càng ít bộ nhớ hệ thống càng tốt. Nền tảng iOS định nghĩa những kỹ thuật và nguyên tắc cho phép người phát triển ứng dụng quản lý bộ nhớ chương trình. Trong một trương trình viết bằng ngôn ngữ Objective-C, một đối tượng luôn được tạo ra và hủy khi cần thiết. Để đảm bảo ứng dụng không sự dụng nhiều bộ nhớ hơn mức cần thiết. Tuy nhiên những kĩ thuật và nguyên tắc này phải giúp đảm bảo một đối tượng sẽ được hủy chỉ khi mà nó không còn cần đến nữa. Và để đảm bảo điều này, Cocoa định nghĩa một kĩ thuật với tên gọi **object-ownership**bằng cách xác định khi nào cần một đối tượng và khi nào không cần đối tượng đó nữa.

Nguyên tắc **Object Ownership**

Bất kì đố tượng nào cũng có một hoặc nhiều **owner** (Con trỏ). Bằng cách đó mỗi đối tượng phải có ít nhất một con trỏ sở hữu nó (**owner**), và nếu đạt điều kiện này thì nó sẽ tiếp tục được giữ lại trong bộ nhớ. Nếu đối tượng không có bất kì con trỏ nào sở hữu nó thì hệ thống sẽ tự động xóa nó khỏi bộ nhớ. Để xác định khi nào một con trỏ sở hữu hoặc khi nào nó không sở hữu một đối tượng Cocoa touch đưa ra một tập các nguyên tắc sau:

+ Một con trỏ được cho là sở hữu một đối tượng khi nó nhận về đối tượng được tạo ra bằng cách gọi một phương thức nào đó.

Để tạo một đối tượng cần phải sử dụng các phương thức có tên bắt đầu bằng “alloc” hoặc “new” hoặc chứa từ “copy”.

+ Một con trỏ nhận được quyền sở hữu một đối tượng khi gởi thông điệp **retain** cho một đối tượng.

Lưu ý rằng một đối tượng có thể được sở hữu bởi nhiều hơn một con trỏ. Việc yêu cầu sở hữu một đối tượng tương đương với việc giữ đối tượng đó tồn tại trong bộ nhớ.

+ Phải từ bỏ quyền sở hữu của những đối tượng khi không cần đến đối tượng đó nữa.

+ Không từ bỏ quyền sở hữu với những đối tượng mình không sở hữu.

Retain counts

Nguyên tắc**Object-Ownership** được thực hiện thông qua việc đếm số lần **retain** một đối tượng. Mỗi đối tượng có một chỉ số **retain count** và sẽ thay đổi khi số lượng con trỏ sở hữu nó thay đổi.

Khi một đối tượng được tạo ra, chỉ số retain count được gán bằng 1.

Khi gởi một thông điệp retain để một đối tượng thì chỉ số retain count tăng một đơn vị.

Khi một đối tượng nhận thông điệp **release** thì chỉ số **retain count** giảm đi 1 đơn vị.

Khi một đối tượng nhận thông điệp **autorelease** thì chỉ số **retain count** sẽ giảm đi 1 đơn vị tại một thời điểm nào đó trong tương lai (Khi kết thúc phương thức mà tại đó thông điệp **autorelease** được gởi đến đối tượng).

Nếu một đối tượng có chỉ số **retain count** bằng 0 thì nó sẽ bị hủy khỏi bộ nhớ.

3 Vấn đề phát triển ứng dụng trên HĐH iOS

3.1 Các ngôn ngữ dùng phát triển ứng dụng trên nền tảng iOS

3.1.1 Phát triển bằng ngôn ngữ Objective-C

Ngôn ngữ Objective-C là ngôn ngữ lập trình chính thống được Apple Inchỗ trợ dùng để xây dựng các ứng dụng, game cho nền tảng iOS thông qua việc sử dụng các bộ framework được cung cấp trong nền tảng iOS.

Tổng quan về Objective-C

Objective-C là một ngôn ngữ lập trình máy tính đơn giản được thiết kế để cho phép lập trình hướng đối tượng . Objective-C được xây dựng như là một tập các phần mở rộng của chuẩn ANSI C. Những phần bổ sung được thêm vào chủ yếu từ ngôn ngữ lập trình Smalltalk, một trong những ngôn ngữ lập trình hướng đầu tiên. Objective-C

được thiết kế để bổ sung khả năng lập trình hướng đối tượng một cách đầy đủ cho C theo một cách đơn giản và dễ dàng nhất có thể.

Môi trường phát triển hướng đối tượng trên nền tảng iOS bao gồm các phần sau

+ Ngôn ngữ lập trình hướng đối tượng (Objective-C).

+ Thư viện các đối tượng.

+ Bộ công cụ phát triển (iOS SDK).

+ Một môi trường thực thi (Hệ điều hành iOS).

Các khái niệm cơ bản của Objective-C

+ Đối tượng, lớp, truyền thông điệp.

+ Định nghĩa lớp.

+ Cấp phát vùng nhớ và khởi tạo đối tượng.

+ Protocol.

+ Khai báo thuộc tính (Property).

+ Category và Extension.

+ Associative reference.

+ Fast Enumeration.

+ Hàm static.

+ Selector.

+ Bắt lỗi (Exeption handling).

+ Luồng.

+ Remote messaging.

3.1.2 Phát triển ứng dụng bằng ngôn ngữ C/C++

3.2 Các công cụ hỗ trợ phát triển ứng dụng cho nền tảng iOS

Cùng với việc phát hành nền tảng iOS, Apple Inc cũng cung cấp kèm theo đó là bộ công cụ iOS SDK chạy trên nền tảng Mac OS X. Đây là bộ công cụ phát triển ứng dụng, game chính thức được Apple Inc phát triển và hỗ trợ.

3.2.1 iOS SDK

iOS SDK cung cấp đầy đủ mọi tài liệu, công cụ và nguồn tài nguyên cần thiết để phát triển ứng dụng cho nền tảng iOS từ máy Macintosh.

iOS SDK cung cấp những thành phần thiết yếu sau:

3.2.1.1 Bộ công cụ Xcode

Xcode: Integrated development enviroment dùng để quản lý project, biên dịch, chạy và gở rối ứng dụng. Xcode tương thích một cách hoàn toàn với những công cụ khác.

Interface Builder: Công cụ trực quan dùng để xây dựng giao diện ứng dụng.

Instruments: Công cụ phân tích thực thi và gở rối (debug).

3.2.1.2 iOS Simulator

Ứng dụng giả lập thiết bị **iDevice** dùng để chạy thử ứng dụng iOS trên máy tính.

3.2.1.3 iOS Developer Library

Tài liệu thao khảo một cách toàn diện về những công nghệ và quy trình xây dựng ứng dụng trên nền tảng iOS.

3.2.2 Các bộ công cụ SDK cho nền tảng iOS khác

Ngoài bộ công cụ iOS SDK được Apple Inc chính thức hỗ trợ phát triển cũng có nhiều bộ công cụ SDK dành cho iOS được phát triển cho các nền tảng ngoài Mac OS.Những bộ công cụ này do các nhà phát triển tự xây dựng và không được hỗ trợ bởi Apple Inc nên khả năng sử dụng rất hạn chế. Để phân biệt các bộ công cụ SDK được phát triển dành cho nền tảng iOS ta phân chia chúng thành hai nhóm cơ bản:

SDK trung gian: Đại diện của nhóm này phải kể đến: **DragonFireSDK**… Đây là nhóm các bộ công cụ SDK cung cấp cho các nhà phát triển ứng dụng xây dựng ứng dụng, game cho HĐH iOS dưới dạng các ngôn ngữ trung gian như: C,C++ thông qua việc sử dụng các framework được cung cấp bởi từng nhà phát triển riêng biệt. Sau khi hoàn tất quá trình phát triển ứng dụng các nhà phát triển chuyển phần dự án về **DragonFireSDK**để dịch thành ứng dụng cho nền tảng iOS. **DragonFireSDK**đơn giản đứng vị trí trung gian xây dựng các bộ công cụ phát triển ứng dụng, game và các framework kèm theo để đơn giản hóa việc xây dựng ứng dụng cho nền tảng iOS. Các bộ công cụ này thường không cung cấp đầy đủ các công nghệ mà iOS hỗ trợ, hoạt động thiếu ổn định.

SDK dạng đa nền tảng: Đại diện của nhóm này phải kể đến: **PhoneGap**… Đây là nhóm các bộ công cụ cung cấp các bộ framework dùng để xây dựng ứng dụng đa nền tảng (PhoneGap hỗ trợ iOS, Android, Symbian, WebOS…). Các bộ framework trên

thường được xây dựng bằng HTML (5) và Javascript nhưng vẫn tận dụng được các tính năng cao cấp ở các nền tảng di động như iOS, Android, Symbian…

3.3 Framework của hãng thứ ba

Ở các phiên bản trước iOS 4.0, Apple Inc nghiêm cấm việc sử dụng các framework của hãng thứ ba trong phát triển ứng dụng cho nền tảng iOS. Tuy nhiên kể từ phiên bản iOS 4.0 Apple Inc đã cho phép các nhà phát triển ứng dụng sử dụng framework được cung cấp bởi các hãng thứ ba. Lần lượt các bộ framework ra đời giúp giảm bớt sự khó khăn của nhà phát triển ứng dụng trong phát triển ứng dụng trên nền tảng iOS. Dưới đây là danh sách một số bộ framework được các nhà phát triển ứng dụng sử dụng và đánh giá cao:

1. Three20: Bộ framework của Facebook phát triển cung cấp một loạt kiểu View mới, các framework xử lý dữ liệu (md5, XML parser bằng DOM…), một loạt framework xử lý kết nối mạng. Đây là bộ framework được đánh giá cao nhất trên iOS.
2. Facebook connect Library for iPhone: framework hỗ trợ kết nối tài khoản Facebook.
3. Route me: frameworkhỗ trợ các xử lý liên quan đến tọa độ địa lý.
4. CHDataStructures.framework: framework hỗ trợ các kiểu dữ liệu: Queue, heap, LinkedList, SearchTree, Stack...
5. ElementParser: Hỗ trợ parse XML bằng đối tượng DOM.

Tuy vậy, đến thời điểm viết báo cáo này, iOS vẫn chưa hỗ trợ lập trình ứng dụng bằng ngôn ngữ Flash. Đồng thời, trình duyệt web của iOS cũng không hỗ trợ chạy các ứng dụng hay các phim ảnh ở định dạng Flash.

4 Ứng dụng hệ điều hành iOS xây dựng giải pháp iQsine

4.1 Giới thiệu

Trong một lần tình cờ làm việc với anh Nguyễn Hùng Ân (Tổ trưởng tổ phục vụ-Nhà hàng Bốn mùa)nhóm chúng tôi được anh chia sẽ những khó khăn trong việc quản lý phục vụ ở nhà hàng Bốn mùa nơi anh đang làm việc. Bằng những phân tích ban đầu, nhóm chúng tôi nhận thấy khả năng ứng dụng tính năng di động của các thiết bị di động hiện tại để giải quyết bài toán nghiệp vụ ở nhà hàng Bốn mùa cũng như các mô hình nhà hàng, quán ăn tương tự. Nhóm chúng tôi nhận thấy đây là cơ hội tốt để thực nghiệm những gì đã nghiên cứu được trong thời gian tìm hiểu về nền tảng iOS. Và

chính vì vậy mà nhóm đã quyết định xây dựng giải pháp iQsine ứng dụng nền tảng iOS giải quyết bài toán nghiệp vụ ở nhà hàng Bốn mùa.

4.2 Mô tả nghiệp vụ

Phần phân tích hệ thống nghiệp vụ được nhóm thực hiện thông qua việc phân tích hệ thống nghiệp vụ của của nhà hàng **Bốn Mùa** (**Đường 20-Làng đại học Quận Thủ Đức-TP Hồ Chí Minh**) dưới sự hướng dẫn của anh NguyễnHùngÂn (Tổ trưởng tổ Phục vụ).

4.2.1 Giới thiệu sơ lược về nhà hàng Bốn Mùa

Cách trung tâm thành phố Hồ Chí Minh 20km, một nhà hàng mang đậm hương vị dân tộc từ món ăn đến phong cách phục vụ, phong cách bày trí,….Nhà hàng BốnMùa tọa lạc tại số 120 đường 20 Làngđại học Quận Thủ Đức trên một diện tích rộng rãi. Với phong cách dân dã, những mái nhà tranh, lu nước, cầu dừa và những khoảng sân sau hèđưa thực khách về một không gian đậm chất Nam bộ. Nhà hàng gồm có hai khu vực: ngoài trời và trong nhà tạo cho thực khách nhiều lựa chọn khi đến đây thưởng thức cácmón ăn mang đậm chất dân dã theo đúng tiêu chí “Mùa nào thức nấy”.

4.2.1.1 Sơ lược hệ thống phục vụ

Để phục vụ bữa ăn cho **Khách hàng**, đầu tiên **Lễ tân** sẽ hướng dẫn khách đến **Khu vực** thích hợp. Tiếp sau đó bồi bàn sẽ giúp khách chọn **Bàn** và giới thiệu với khách thực đơn của ngày hiện tại. Bồi bàn sẽ ghi lại chi tiết **Phiếu ghi Order** và chuyển một bản về nhà bếp, và một bản chuyển về**Quầy thu ngân** để tính tiền sau khi khách dùng bữa xong. Bếp trưởng có nhiệm vụ phân chia món ăn cho các đầu bếp thực hiện sao cho đảm bảo món ăn được hoàn thành trong thời gian thích hợp để phục vụ cho khách. Khi khách đã dùng xong các **Món ăn** trên bàn, bồi bàn sẽ yêu cầu nhân viên tiếp thực dọn bàn và mang món mới lên cho khách. Khi khách yêu cầu thanh toán, bồi bàn sẽ mang **Phiếu ghi Order** về quầy thu ngân để xuất **Hóa đơn thanh toán**. Bồi bàn mang hóa đơn thanh toán để khách kiểm tra, nhận tiền thanh toán. **Thu ngân** sẽ kiểm tra số tiền cùng với hóa đơn và gửi lại số tiền thừa để bồi bàn mang lại cho khách hàng.

4.2.2 Chi tiết về hệ thống nghiệp vụ hiện tại

4.2.2.1 Đặt bàn

Khách hàng gọi điện đến cửa hàng yêu cầu đặt bàn, lễ tân sẽ hướng dẫn khách chọn **Bàn** theo yêu cầu. Khi đã thống nhất được bàn, lễ tân ghi nhận lại các thông tin cần thiết vào **Danh sách đặt bàn** như **Họ tên**, **Số điện thoại**, **Số thực khách dùng bữa**, **Thời gian đến của khách**… Lễ tân yêu cầu khách đến đúng giờ nếu không đơn

đặt hàng sẽ bị hủy. (Sau 15p chưa thấy khách tới thì hủy đặt bàn. Phải đặt bàn trước ít nhất 3 tiếng).

4.2.2.2 Tìm bàn

Mỗi khi có **Khách hàng** bước vào nhà hàng, nhân viên **Lễ tân** sẽ hỏi xem khách đã đặt bàn trước hay chưa. Nếu khách đã đặt bàn trước, lễ tân sẽ kiểm tra danh sách đặt bàn và hướng dẫn khách tới bàn đã đặt. Đối với khách chưa đặt bàn, lễ tân sẽ hướng dẫn khách đến khu vực thích hợp. Sau đó, bồi bàn sẽ hướng dẫn khách chọn **Bàn**.

4.2.2.3 Ghi order

Sau khi khách đã ngồi vào **Bàn**, **Bồi bàn** sẽ hướng dẫn khách chọn **Món ăn**theo **Thực đơn** của ngày. Sau khi khách đã chọn món ăn xong, bồi bồi ghi vào **Phiếu ghi Order** , Trong phiếu **Phiếu ghi Order** ,mỗi món ăn sẽ được đặt trong một **Nhóm các phần ăn**nhất định. Mỗi **Nhóm các phần ăn** sẽ có một số thứ tự nhất định, số thứ tự này sẽ tương ứng với thứ tự mà nhóm những món ăn này được phục vụ cho thực khách.

Ghi chú:

Việc phân nhóm có 2 loại. Với những nhà hàng phục vụmón ăn kiểu châu Á việc phân nhóm sẽ dựa theo những nhóm cơ bản như: Món lẫu, món nước, món khô, đồ uống, món tráng miệng. Với những nhà hàng phục vụ mónăn kiểu châu Âu thì sẽ có những nhóm cơ bản: Món khai vị, món chính, món tráng miệng, đồ uống… Việc phân nhóm phụ thuộc vào kiểu nhà hàng và phong cách phục vụ của nhà hàng đó. Mỗi nóm món ăn có một thứ tự tương ứng với thứ tự mà nhóm những món ăn này được phục vụ cho thực khách. Việc phân **Nhóm các phần ăn**dựa vào kinh nghiệm của bồi bàn và yêu cầu đặt biệt của thực khách (***Với nhà hàng châu Á thì các món đồ uống sẽ được phục vụ trước tiên, sau đó là các món khô và cuối cùng mới đến các món lẫu…***).

Sau khi khách chọn món ăn xong bồi tạo một bản và gởi về cho nhà bếp và giữ một bản để gởi về**Quầy thu ngân** để tính tiền sau khi khách đã dùng bữa xong.

Trong khi ăn, khách có thể gọi thêm món, thay đổi bàn ăn. Mọi thay đổi bồi bàn phải có nhiệm vụ cập nhật**Phiếu ghi Order** .

4.2.2.4 Thực hiện món ăn

Khi có **Order** gửi vào bếp, **Bếp trưởng** sẽ xem xét danh sách các **Nhóm các phần ăn** trong **Order** sẽ phân chia kế hoạch thực hiện. Khi **Phiếu ghi Order** mới được gởi vào **Bếp trưởng** sẽ lên kế hoạch để thực hiện**Nhóm các phần ăn** đầu tiên.

Mỗimón ăn thuộc một**Nhóm các phần ăn** phục vụ cho một**Bàn** nhất định sẽ được đặtở một khu vực nhấtđịnh trên **Bàn bếp chính**. Mỗi khu vực sẽ chỉ dùng để đặt những món ăn riêng biệt phục vụ cho một**Bàn xác định** và thuộc một**Nhóm các phần ăn** xác định. **Mỗi nhóm các phần ăn** được phân biệt với các**Nhóm các phần ăn** khác thông qua **Cờ chỉ bàn**. Khi thực hiện xong một**Phần ăn** bếp trưởng sẽ phân chia vào các**Nhóm các phần ăn**thích hợp.

Ghi chú:

Cờ chỉ bàn *là dạng bảng nhựa có gắn với bệ đứng bằng một thân kim loại có khả năng gập xuống và mở lên. Bảng nhựa của* ***Cờ chỉ bàn*** *dùng cho bếp trưởng ghi tên bàn, đây chính là định danh xác định* ***Nhóm các phần ăn*** *tại một khu vực thuộc bàn nào trong nhà hàng. Mặc khác cờ chỉ bàn cũng chính là nơi để bếp trưởng kẹp* ***Phiếu ghi Order****.*

Trước khi tất cả các phần ăn của một ***Nhóm các phần ăn*** *được thực hiện hoàn tất thì cờ chỉ bàn sẽ được gập xuống. Lúc này Cờ chỉ bàn đơn giản dùng để Bếp trưởng sắp xếp các Phần ăn sau khi thực hiện xong.*

Khi tất cả các**Phần ăn** thuộc**Nhóm các phần ăn** được thực hiện xong, **Bếp trưởng** sẽ dựng cờ đánh dấu bàn vào **Nhóm các phần ăn** đã thực hiện xong để nhân viên **Tiếp thực** mang ra bàn khi có yêu cầu của **Bồi bàn**.

Nếu một món ăn không còn (hết nguyên liệu,...) **Bếp trưởng** sẽ yêu cầu nhân viên **Tiếp thực** báo lại cho **Bồi bàn** để tránh việc khách chọn các món ăn đã hết.

Nếu món ăn chưa hết hẳn nhưng không còn đủ để nấu theo số lượng mà khách đã yêu cầu, bếp trưởng sẽ yêu cầu nhân viên **Tiếp thực** báo lại cho **Bồi bàn** để bồi bàn gởi lời xin lỗi đến**Thực khách**.

4.2.2.5 Phục vụ món ăn

Sau khi chuyển **Phiếu ghi Order**vào bếp một thời gian nhất định, bồi bàn sẽ yêu cầu tiếp thực mang **Nhóm các phần ăn** đầu tiên trong **Phiếu ghi Order**lên.

Đểđảm bảo cácmón ăn sau khi chuẩn bị sẽ được phục vụ cho **Thực khách** ngay khi còn nóng cũng như tránh việc thực khách phải chờ quá lâu sau khi ăn xong một**Nhóm các phần ăn** thì bồi bàn phải có nhiệm vụ quan sát việc dùng bữa của**Thực khách**. Khi thực khách gần dùng xong một**Nhóm các phần ăn** thì**Bồi bàn** phải có nhiệm vụ báo nhân viên **Tiếp thực** yêu cầu **Bếp trưởng** chuẩn bị**Nhóm các phần ăn** còn lại để có thể phục vụ**Thực khách** nhanh nhất có thể. Quá trình này sẽ được lặp đi lặp lại cho tới khi khách dùng bữa xong.

4.2.2.6 Thanh toán

Khi khách yêu cầu thanh toán, bồi bàn sẽ mang **Phiếu ghi Order**về cho **Thu ngân** để **Thu ngân** ghi nhận và tính hóa đơn. Sau khi in**Hóa đơn**, **Bồi bàn** sẽ kẹp **Hóa đơn** vào **Sổ tính tiền** mang ra cho khách. Trên mỗi tờ **Hóa đơn** sẽ có một dòng trống để khách có thể điền số tiền boa.

Khách hàng sẽ kiểm tra và thanh toán tiền cho bồi bàn. **Tiền** và **Hóa đơn** tính tiền sẽ được đem về cho thu ngân kiểm tra, nếu có tiền thừa sẽ được bồi bàn mang trả lại cho khách.

4.2.2.7 Dọn bàn và tiễn khách

Sau khi khách đã hoàn tất thanh toán, bồi bàn sẽ yêu cầu tiếp thực ra dọn bàn để đón khách mới.

4.2.2.8 Lên thực đơn và chuẩn bị nguyên vật liệu cho ngày hôm sau

Cuối ngày bếp trưởng sẽ kiểm tra số nguyên liệu còn trong kho của nhà hàng để lên danh sách thực đơn cũng như danh sách các nguyên liệu cần chuẩn bị cho ngày hôm sau. Mỗi mónăn sẽ được chuẩn bị một số lượng phần xácđịnh tùy theo quyết định của bếp trưởng. Bếp trưởng ra quyết định chọn thựcđơn cho ngày hôm sau dựa vào các yếu tố:

Số lượng và chủng loại nguyên liệu còn lại trong kho của nhà hàng.

Thựcđơn theo mùa của nhà hàng.

Doanh số củamón ăn những ngày trước đó hoặc mùa trướcđó.

4.2.3 Những vấn đề đang tồn tại trong hệ thống nghiệp vụ hiện tại

4.2.3.1 Khó khăn trong quản lý Phiếu ghi Order

Mỗi khi có một**Phiếu ghi Order** mới được tạo ra Bồi bàn luôn phải tạo 2 bản, một bản gởi vào nhà bếp để thực hiệnmón ăn và một bản Bồi bàn chuyển về Quầy thu ngân để tính tiền sau khi khách dùng bữa xong. Nhưng do đặc trưng của thực khách là người Việt Nam có thói quen gọi trước một số phần ăn và dùng trước sau đó mới gọi thêm các phần ăn khác nên việc quản lý **Phiếu ghi Order** tương đối khó khăn. Vào những lúc đông khách việc cập nhật **Phiếu ghi Order** gây tốn rất nhiều thời gian cũng như rắc rối cho **Bồi bàn**. Nhà hàng **Bốn mùa** mong muốn rằng ngay khi **Phiếu ghi Order** được tạo xong nó sẽ được gởi ngay về**Quầy thu ngân** để**Nhân viên Thu Ngân** có thể kiểm soát một**Phiếu ghi order** ngay khi nó mới được tạo ra.

4.2.3.2 Khó khăn trong việc thông báo thay đổi thực đơn khi hết một món ăn

Mặc dù kinh nghiệm dự đoán của bếp trước rất chính xác nhưng vẫn có thể xảy ra những trường hợp sai sót ngoài mong đợi (Nguyên liệu hao hụt, hư hỏng trong quá trình vận chuyển, bảo quản, dự đoán số lượng phần ăn không chính xác). Hơn nữa do đặc trưng của nhà hàng **Bốn mùa** nằm trên một khuôn viên khá rộng nên việc thông báo hết mộtmón ăn đến tất cả**Bồi bàn** trong quán vào những thời điểm đông đúc luôn là một vấn đề lớn với nhà hàng. Chính vìđiểm nàyđôi lúc nhà hàng thường xuyên nhận được các**Phiếu ghi Order** yêu cầu những món ănđã hết khiến tiếp thực phải liên tục thông báo với**Bồi bàn**.

4.2.3.3 Khó khăn trong việc gọi làm hay gọi phục vụ một Nhóm các phần ăn

Do bồi bàn luôn đứngở khu vực họ quản lý nên mỗi khi muốn gọi làm hay gọi phục vụ một**Nhóm các phần ăn** Bồi bàn có thể thực hiện bằng 2 cách:

+ Bấm chuông gọi**Tiếp thực** lên khu vực của mình để gọi làm hay gọi phục vụ một **Nhóm các phần ăn**.

+ Yêu cầu nhân viên **Tiếp thực**đang có mặtở khu vực của mình báo bếp trưởng làm hoặc báo yêu cầu phục vụ một**Nhóm các phần ăn**.

+ Ở những thời điểm đông khách hoặc cần thiết**Bồi bàn** có thể trực tiếp chuyển yêu cầu đến**Bếp trưởng**.

Cũng do các thức tổ chức trên mà***yêu cầu làm hay yêu cầu phục vụ*** được truyềnđạt thông qua **Tiếp thực** và không có biện pháp ghi chú phù hợp nên có một số trường hợp phải để thực khách chờ vì***yêu cầu gọi làm hay gọi phục*** vụ bị**"đè"** lên bởi những yêu cầu từ những bàn khác gọi sau nhưng lại được nhân viên **Tiếp thực** mang đến**Bếp trưởng** trước. Vào những thời điểm nhà hàng đông thực khách, nhân viên tiếp thực phải liên tục phụ vụmón ăn và dọn bàn dẫn đến một số trường hợp yêu cầu làm hay yêu cầu phục vụ bị**"lãng quên"** và phải đợi đến khi **Bồi bàn** nhắc lại yêu cầu.

4.2.3.4 Khó khăn trong việc thống kê doanh thu

Với hệ thống quản lý cũ, nhà hàng **Bốn mùa** sử dụng một máy**POS**đơn giản của Lý Phú Vinh để thực hiện việc tính tiền gây khó khăn trong việc thống kế doanh thu trong ngày.

4.2.4 Yêu cầu đề ra

Trước những khó khăn trên nhà hàng Bốn mùađã đề nghị nhóm thiết kế xây dựng một giải pháp đơn giản, linh hoạt với chi phí hợp lý để giải quyết bài toán quản lýcũng như tăng sự hài lòng của khách hàng về dịch vụ của nhà hàng cung cấp. Giải pháp quản lý nhà hàng được đề nghị phải thỏa mãn được những yếu tố sau:

+ Giải quyết các khó khăn hiện tại nhà hàng đang gặp phải.

+ Hệ thống mới phải có khả năng tương thích với mô hình quản lý hiện tại của nhà hàng.

+ Hệ thống mới phải có khả năng nâng cấp, tích hợp và bổ sung các tính năng mới khi cần thiết.

+ Hệ thống phải có khả năng lưu trữ và backup dữ liệu tự động.

4.3 Mô tả giải pháp iQsine

Bằng những phân tích trên bài toán nghiệp vụ được đại diện nhà hàng Bốn mùa cung cấp, nhóm đã tiến hành phân tích thiết kế một giải pháp giúp giải quyết bài toán nghiệp vụ kể trên.

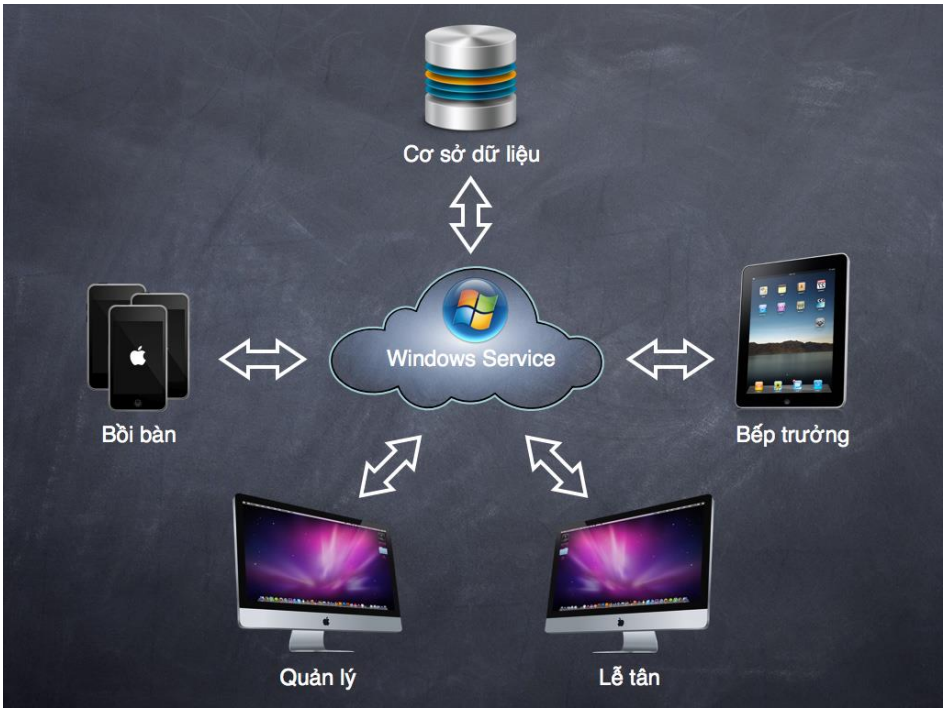
4.3.1 Sơ lược

Giải pháp iQsine được xây dựng dựa trên những nhận định có được từ quá trình phân tích hệ thống nghiệp vụ của nhà hàng **Bốn mùa** do anh NguyễnHùngÂn hướng dẫn. Giải pháp iQsinekhai thác tính tiện dụng và dễ di chuyển của thiết bị di động đa chức năng, mà cụ thể là các máy iPod Touch và iPad của hãng Apple Inc để cung cấp một giải pháp quản lý phục vụ nhà hàng. Trong giới hạn của đề tài khóa luận tốt nghiệp, giải pháp iQsine được ra đời với những mục tiêu sau:

- Thực nghiệm những kiến thức đã học được trong quá trình thực hiện đề tài.
- Ứng dụng nền tảng iOS, thiết bị **iDevice** và những kiến thức nhóm chúng tôi tích lũy được để giải quyết bài toán nghiệp vụ của nhà hàng **Bốn mùa**.

4.3.2 Các thành phần của giải pháp iQsine

Các thành phần của giải pháp iQsine được thể hiện bằng lược đồ sau:



Hình 27 - Lược đồ các thành phần của giải pháp iQsine

4.3.2.1 iQsine waiter

Phần mềm **iQsine waiter** chạy trên iPod hỗ trợ bồi bàn ghi nhận **Phiếu ghi Order**, thêm,xóa, chỉnh sửa, ghi chú **Nhóm phần ăn**, **Phần ăn**. Yêu cầu thực hiện hoặc yêu cầu phục vụ một **Nhóm phần ăn**. Thanh toán và in hóa đơn từ xa.

4.3.2.2 iQsine chef

Phần mềm **iQsine chef**: Hỗ trợ bếp trưởng quản lý danh mục các **Nhóm các phần ăn**. Các chức năng được phần iQsine chef hỗ trợ:

- Thêm, xóa, chỉnh sửa **Nhóm các phần ăn** được yêu cầu phục vụ hoặc yêu cầu thực hiện.
- Từ chối một **Phần ăn** đã hết nguyên liệu.
- Lên thực đơn và danh mục nguyên vật liệu cho ngày hôm sau.

4.3.2.3 iQsine manager

Phần mềm **iQsine manager** cho phép quản lý doanh số, doanh thu, quản lý nguyên vật liệu… của nhà hàng.

4.3.2.4 iQsine receptionist

iQsine receptionist iPad: Phần mềm iQsine receptionist cho máy tính bảng/PC cung cấp những chức năng sau:

- Quản lý Khu vực, Danh sách bàn.
- Quản lý đặt bàn, hủy đặt bàn.
- Quản lý danh sách đen.

4.3.2.5 iQsine service

Đây là thành phần trung tâm của giải pháp iQsine giữ nhiệm vụ trung chuyển thông tin giữa các thành phần khác trong giải pháp (Chuyển Phiếu ghi Order để lưu trữ vào Database, chuyển yêu cầu gọi thực hiện hoặc gọi phục vụ từ iQsine waiter đến iQsine chef, chuyển yêu cầu đặt bàn từ iQsine receptionist đến iQsine waiter phù hợp).

4.3.2.6 iQsine database

Trung tâm lưu trữ dữ liệu của giải pháp iQsine.

4.3.3 Mô tả chi tiết giải pháp iQsine

4.3.3.1 Đặt bàn

Khách hàngđặt bàn, **Lễ tân** sẽ ghi nhận lại các thông tin đặt bàn bằng ứng dụng **iQsine Receptionist**trên PC.

4.3.3.2 Tìm bàn

Mỗi khi có **Khách hàng** bước vào nhà hàng, nhân viên **Lễ tân** tra cứu thông tin danh sách khu vực, danh sách bàn bằng ứng dụng **iQsine receptionist** và hướng dẫn khách đến khu vực thích hợp.

4.3.3.3 Ghi order

Khi khách hàng đến khu vực do **Bồi bàn** phụ trách, **Bồi bàn** có nhiệm vụ đón khách và hướng dẫn khách lựa chọn bàn phù hợp. Khi khách đã chọn **Bàn**, **Bồi bàn** sẽ dùng phần mềm **iQsine waiter** chuyển trạng thái bàn thành "**Đang được sử dụng**". Yêu cầu chuyển trạng thái bàn sẽ được gởi về **iQsine service**, tại đây hệ thống sẽ thay đổi trạng thái của bàn trong cơ sở dữ liệu đồng thời tạo mới một **Order** và trả thông tin về cho **iQsine waiter**. Nếu **iQsine service** trả về một thông báo lỗi (Bàn đã được đặt…) thì **Bồi bàn** sẽ hướng dẫn khách chọn bàn khác. Ngược lại bồi bàn hướng dẫn khách chọn **Món ăn**theo **Thực đơn** của ngày.

Sau khi khách đã chọn **Món ăn**, bồi bàn ghi vào nhận lại danh sách các **Phần ăn** khách gọi bằng phần mềm **iQsine waiter**. Để dễ dàng quản lý, thao tác trên **Phần ăn** giải pháp iQsine đưa ra các định nghĩa sau:

- Mỗi **Món ăn** sẽ thuộc một **Nhóm món ăn** nhất định do quy định phân nhóm của nhà hàng. Việc phân nhóm **Món ăn** có 2 loại. Với những nhà hàng phục vụmón ăn kiểu châu Á việc phân nhóm sẽ dựa theo những nhóm cơ bản như: Món lẫu, món nước, món khô, đồ uống, món tráng miệng. Với những nhà hàng phục vụmón ăn kiểu châu Âu thì sẽ có những nhóm cơ bản: Món khai vị, món chính, món tráng miệng, đồ uống… Việc phân nhóm phụ thuộc vào kiểu nhà hàng và phong cách phục vụ của nhà hàng đó. Mỗi món **Món ăn** có một độ ưu tiên tương ứng với thứ tự mà nhóm những món ăn này được phục vụ cho thực khách (Món khô luôn được phục vụ trước, sau đó mới đến các món nướng, món lẫu thường được phục vụ cuối cùng).

- **Order item(OI)** là một thể hiện cụ thể của Món ăn, Order item có các thông tin như:

+ Độ ưu tiên (Bằng độ ưu tiên của món ăn).

+ Số lượng phần ăn

+ Ghi chú yêu cầu đặt biệt với phần ăn.

+ Trạng thái: Có 4 trạng thái sau: Đang đợi, đang làm, đang phục vụ, đã phục vụ.

- **Order item group(OIG)** là khái niệm trừu tượng hóa của **Nhóm các phầnăn** được mô tả ở phần phân tích hệ thống nghiệp vụ. Mỗi **Order item group** chứa các thông tin:

+ Độ ưu tiên tương ứng với thứ tự mà **OIG** này sẽ được phục vụ thực khách. Độ ưu tiên của **OIG** được chính là độ ưu tiên của các **OI**chứa trong **OIG**.

+ Trạng thái: Có 4 trạng thái sau: Đang đợi, đang làm, đang phục vụ, đã phục vụ.

Khi thêm một **OI** vào **Order** nếu không tồn tại một **OIG** nào có độ ưu tiên bằng độ ưu tiên của **OI** thì ứng dụng **iQsine waiter** sẽ mặc định tạo một **OIG** có độ ưu tiên bằng bằng độ ưu tiên của **OI** mới thêm vào. Ngược lại nếu tồn tại một **OIG** có độ ưu tiên bằng độ ưu tiên của **OI** thì hệ thống sẽ mặc định thêm vào **OIG** tìm được. Ngoài ra **Bồi bàn** có thể thay đổi **OIG** của **OI** bằng cách kéo thả **OI** qua **OIG** khác trên ứng dụng **iQsine waiter**.

Sau khi hoàn tất việc tạo **Order** bồi bàn sẽ sử dụng ứng dụng **iQsine waiter** để gởi yêu cầu thực hiện một **OIG** vào bếp.

Trong khi ăn, khách có thể gọi thêm món, thay đổi bàn ăn. Mọi thay đổi bồi bàn phải có nhiệm vụ cập nhật**Order bằng** ứng dụng **iQsine waiter**.

4.3.3.4 Thực hiện món ăn

Đểđảm bảo cácmón ăn sau khi chuẩn bị sẽ được phục vụ cho **Thực khách** ngay khi còn nóng cũng như tránh việc thực khách phải chờ quá lâu sau khi ăn xong một**Nhóm các phần ăn** thì bồi bànngoài nhiệm vụ đón khách mới vào còn có nhiệm vụ quan sát khách dùng bữa. Khi khách dùng gần hết những phần ăn được mang ra trước đó thì bồi bàn có nhiệm vụ nhấn nút trên ứng dụng **iQsine waiter** để gởi yêu cầu thực hiện một **OIG** về **iQsine service**. **iQsine service** sẽ làm nhiệm vụ thay đổi trạng thái của OIG ở **iQsine database** và thông báo yêu cầu thực hiện **OIG** đến bếp trưởng. Việc thông báo thực hiện một **OIG** đến bếp trưởng bằng cách gởi yêu cầu thực hiện OIG đến **iQsine service** để chuyển giao lại cho ứng dụng **iQsine chef** của bếp trưởng.

Mỗi phần ăn thuộc một**OIG** phục vụ cho một**Bàn** nhất định sẽ được đặtở một khu vực nhấtđịnh trên **Bàn bếp chính**. Mỗi khu vực sẽ chỉ dùng để đặt những món ăn riêng biệt phục vụ cho một**Bàn** xác định và thuộc một**OIG** xác định. Mỗi **OIG** được phân biệt với các**OIG** khác thông qua **Cờ chỉ bàn**. Khi thực hiện xong mộtphần ăn bếp trưởng sẽ phân chia vào các**OIG** thích hợp.

Ghi chú:

Cờ chỉ bàn *là dạng bảng nhựa có gắn với bệ đứng bằng một thân kim loại có khả năng gập xuống và mở lên. Bảng nhựa của* ***Cờ chỉ bàn*** *dùng cho bếp trưởng ghi tên bàn, đây chính là định danh xác định* ***OIG*** *tại một khu vực thuộc bàn nào trong nhà hàng.*

Trước khi tất cả các phần ăn của một **OIG** được thực hiện hoàn tất thì cờ chỉ bàn sẽ được gập xuống. Lúc này Cờ chỉ bàn đơn giản dùng để Bếp trường sắp xếp các phần ăn sau khi thực hiện xong.

Khi tất cả cácphần ăn thuộc**OIG** được thực hiện xong, **Bếp trưởng** sẽ dựng cờ đánh dấu bàn vào **OIG** đã thực hiện xong để nhân viên **Tiếp thực** mang ra bàn khi có yêu cầu của **Bồi bàn**.

Nếu một món ăn không còn (hết nguyên liệu,...) **Bếp trưởng** sẽ sử dụng ứng dụng **iQsine chef** để thay đổi số lượng phần ăn của **Món ăn** (Số lượng phần ăn tối đa có thể phục vụ tương ứng với số lượng nguyên liệu chuẩn bị trong ngày).

Nếu món ăn chưa hết hẳn nhưng không còn đủ để nấu theo số lượng mà khách đã yêu cầu, bếp trưởng sẽ gởi thông báo cho **Bồi bàn** để bồi bàn gởi lời xin lỗi đến**Thực khách** thông qua ứng dụng **iQsine chef**.

4.3.3.5 Phục vụ món ăn

Sau khi chuyển **OIG**vào bếp một thời gian nhất định, bồi bàn sẽ yêu cầu tiếp thực phục vụ một **OIG** bằng cách nhấn nút trên ứng dụng **iQsine waiter**. Yêu cầu phục

vụ một OIG sẽ được gởi về **iQsine service** để thay đổi trạng thái của OIG dưới cơ sở dữ liệu và chuyển yêu cầu đến nhà bếp thông bằng các gởi thông báo đến bếp trường thông qua ứng dụng **iQsine chef**.

4.3.3.6 Thanh toán

Khi khách yêu cầu thanh toán, bồi bàn sẽ mang nhấn nút thanh toán trên ứng dụng **iQsine waiter**. Yêu cầu thanh toán sẽ được gởi về **iQsine service** để thực hiện việc tình toán và in **Hóa đơn**. Sau khi in **Hóa đơn**, **Bồi bàn** sẽ kẹp **Hóa đơn** vào **Sổ tính tiền** mang ra cho khách. Trên mỗi tờ **Hóa đơn** sẽ có một dòng trống để khách có thể điền số tiền boa.

Khách hàng sẽ kiểm tra và thanh toán tiền cho bồi bàn. **Tiền** và **Hóa đơn** tính tiền sẽ được đem về cho thu ngân kiểm tra, nếu có tiền thừa sẽ được bồi bàn mang trả lại cho khách.

4.3.3.7 Dọn bàn và tiễn khách

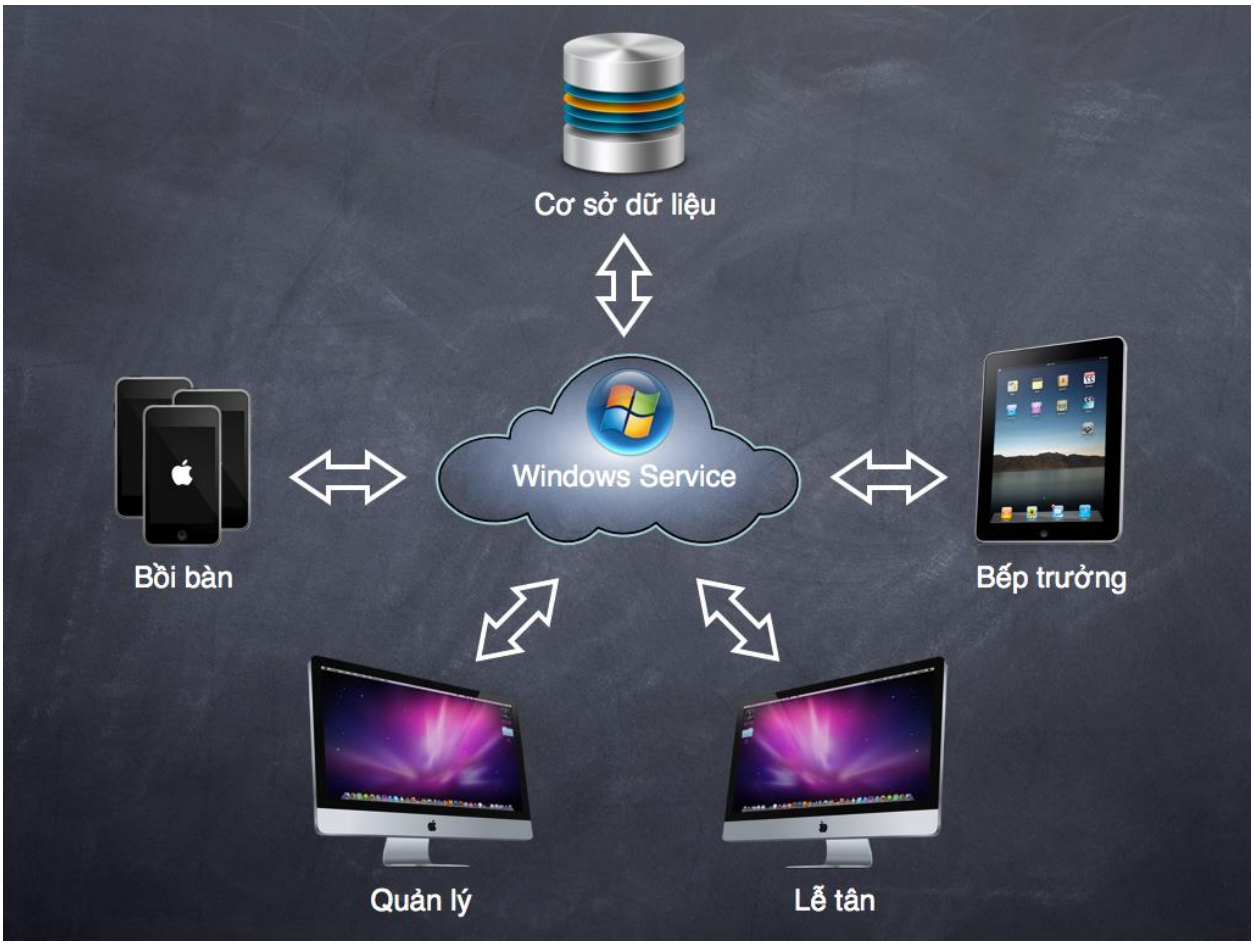
Sau khi khách đã hoàn tất thanh toán, bồi bàn sẽ nhấn nút chuyển trạng thái bàn thành “Đang dọn” và yêu cầu tiếp thực ra dọn bàn để đón khách mới. Sau một thời gian nhất định (5 phút) bàn sẽ tự chuyển về trạng thái đang trống.

Như đã trình bày ở phần trên, mục đích chính của sự ra đời giải pháp iQsine là thực nghiệm những kiến thức tìm hiểu được về nền tảng iOS nên việc thực hiện hết toàn bộ giải pháp iQsine vượt ra khỏi giới hạn của khóa luận. Chính vì vậy nhóm chúng tôi quyết định chỉ thực hiện một phần giới hạn giải pháp iQsine bao gồm các thành phần sau:

- **iQsine waiter.**
- **iQsine service.**
- **iQsine database.**
- **iQsine receptionist.**

4.4 Phân tích thiết kế giải pháp iQsine

Từ các phân tích tích lũy được trong quá trình làm việc với anh Nguyễn Hùng Ân nhóm chúng tôi đã đưa ra được mô hình tổng quan của giải pháp iQsine ở dưới đây:



Hình 28 - Mô hình tổng quan giải pháp iQsine

Trong toàn bộ giải pháp thành phần **iQsine service** là thành phần trung tâm của toàn giải pháp. **iQsine service** sẽ giữa vai trò trung gian, điều phối liên lạc giữa các thành phần trong giải pháp. Khi một tin nhắn được gởi đi từ các thành phần còn lại của giải pháp nó sẽ được chuyển tới **iQsine service**, nơi nó sẽ được phân loại, lưu vết xuống **iQsine database** và chuyển tiếp đến các thành phần tương ứng. Chính vì vậy **iQsine service** được nhóm chọn làm điểm bắt đầu cho việc phân tích và thiết kế giải pháp iQsine.

Thành phần **iQsine service** được nhóm định nghĩa như là một dịch vụ chạy trên máy chủ của giải pháplàm nhiệm vụ trung chuyển dữ liệu giữa các thành phần khác trong hệ thống. **iQsine service** phải có khả năng nhận dữ liệu từ các thành phần và tạo kết nối để gởi dữ liệu đến các thành phần tương ứng. Tuy nhiên với đặc trưng riêng của các thiết bị di động, hệ điều hành cho thiết bị di động cần phải đáp ứng được những yêu cầu riêng không giống bất kì nền tảng điện toán trước đây như:

- Thiết bị nhỏgọn, màn hình hạn chế.

- Bộ nhớ thiết bị nhỏ, tốc độ bộ vi xử lý trung tâm thấp (500MHz, 600 MHz, 1GHz).
- Thời gian sử dụng hạn chế do dung lượng pin nhỏ.
- Sử dụng kết nối không dây.

Chính vì những lý do trên mà các hệ điều hành di động thường bị giới hạn các tính năng, các công nghệ đi kèm. Nằm trong những hạn chế đó, nền tảng iOS chỉ hỗ trợ hai giao thức kết nối mạng là:

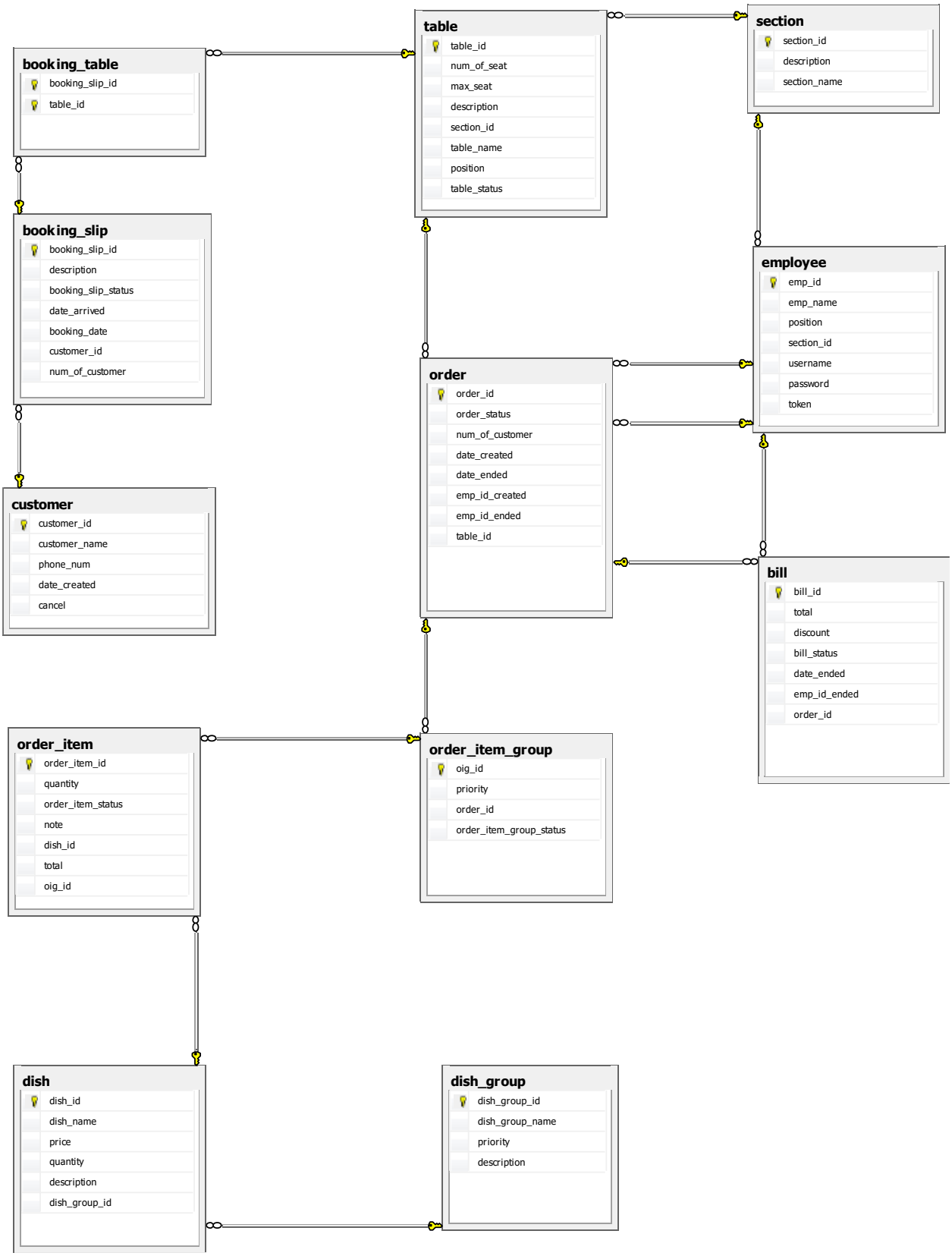
- Socket.
- HTTP.

Với mô hình tổng quan giải pháp iQsine rõ ràng giao thức Socket là phù hợp hơn cả. Tuy nhiên, để có thể phát triển ứng dụng sử dụng kết nối Socket trên nền tảng iOS người phát triển phải sử dụng thiết bị thật để deploy ứng dụng mới có thể lấy được IP thật của thiết bị. Đây thực sự là một khó khăn của nhóm khi hiện thực giải pháp iQsine vì nhóm không sở hữu đủ thiết bị iDevice để xây dựng giải pháp. Nhưng với mong muốn thực nghiệm những kiến thức tìm hiểu được nhóm đã quyết định sử dụng giao thức HTTP để hiện thực giải pháp iQsine. Tuy thật sự theo cách giải quyết ở trên thì giải pháp iQsine sẽ không thật sự hoàn hảo theo ý nghĩa ứng dụng thực tiễn nhưng vẫn cho nhóm chúng tôi cơ hội để trải nghiệm những kiến thức đã tìm hiểu được. Chính vì vậy nhóm vẫn quyết định thực hiện giải pháp iQsine với thiết kế như sau:

- iQsine database sử dụng SQL server 2008.
- iQsine service là webservice.
- Giao tiếp giữa iQsine service và iQsine waiter bằng cách thực hiện trao đổi dữ liệu dạng xml (Được đặc tả ở phần sau).

4.4.1 Thiết kế Database

Từ những mô tả nghiệm vụ, mô tả giải pháp, nhóm chúng tôi đã tiến hành phân tích thiết kế cơ sở dữ liệu của ứng dụng bằng cách phân tách các thực thể, xác định mối quan hệ giữa chúng cũng như làm rõ các ràng buộc cần thiết. Dựa vào mô tả giải pháp ở trên nhóm chúng tôi đã thiết kế cơ sở dữ liệu chính của giải pháp **iQsine** như sau:



Hình 29 - Database diagram

Dưới đây là danh sách table kèm đặc tả chi tiết của chúng:

section

Table description

- Lưu trữ thông tin của từng khu vực trong nhà hàng.

Field	Type	Null	Key	Default	Extra
Section_id	Int		PK	Identity(1,1)	
Section_name	Nvarchar(50)				
Description	Nvarchar(100)	√			

table

Table description

- Lưu trữ thông tin của mỗi bàn ăn.
- Một bàn thuộc một khu vực duy nhất.
- Table_status(tình trạng bàn): 0: trống, 1: đang đặt, 2: đang dọn, 3: đang dùng, 4:ghép bàn.

Field	Type	Null	Key	Default	Extra
Table_id	Int		PK	Identity(1,1)	
Table_name	Nvarchar(50)				
Table_status	Int			0	
Num_of_seats	Int				
Max_seats	Int				
Position	Int				
Section_id	Int		FK(Section.section_id)		
Description	Nvarchar(300)	√			

customer

Table description

- Lưu trữ thông tin khách hàng.

Field	Type	Null	Key	Default	Extra
Customer_id	Int		PK	Identity(1,1)	
Customer_name	Nvarchar(50)				
Phone_num	Nvarchar(20)				

Date_created	Datetime			CURRENT_TIME	
Cancel	Int			0	

booking_slip

Table description

- Lưu trữ thông tin phiếu đặt bàn của khách.
- Mỗi phiếu đặt bàn thuộc một khách duy nhất.
- Booking_slip_status (tình trạng phiếu đặt bàn): 0: đang đặt, 1: đã đến, 2: hủy tự động, 3:hủy bằng tay.

Field	Type	Null	Key	Default
Booking_slip_id	Int		PK	Identity(1,1)
Booking_slip_status	Int			0
Booking_date	Datetime			CURRENT_TIME
Date_arrived	Datetime			
Num_of_customer	Int			
Customer_id	Int		FK (customer.customer_id)	
Description	Nvarchar(300)	√		

booking_bable

Table description

- Lưu trữ thông tin bàn được đặt thuộc phiếu đặt bàn nào.

Field	Type	Null	Key	Default	Extra
Booking_slip_id	Int		PK		
Table_id	Int		PK		

employee

Table description

- Lưu trữ thông tin nhân viên.

Field	Type	Null	Key	Default	Extra

Employee_id	Int		PK	Indentity(1,1)	
Employee_name	Nvarchar(50)				
Position	Nvarchar(30)	√			
Username	Varchar(40)				
Password	Varchar(40)				
Token	Varchar(32)	√			
Section_id	Int	√	FK (section.section_id)		

order

Table description

- Đơn đặt hàng của khách tại mỗi bàn.
- Order_status (tình trạng của đơn đặt hàng): 0: đang phục vụ, 1:đã kết thúc.

Field	Type	Null	Key	Default	Extra
Order_id	Int		PK	Indentity(1,1)	
Order_status	Int			0	
Num_of_customer	Int	√		0	
Date_created	Datatime				
Date_ended	Datetime	√			
Emp_created	Int		FK (employee.employee_id)		
Emp_ended	Int	√	FK (employee.employee_id)		
Table_id	Int		FK (table.table_id)		

dish_group

Table description

- Món ăn được phân loại và lưu thành từng nhóm nhằm dễdàng sắpxếp độ ưu tiên khi thực hiện món ăn trong order.
- Độ ưu tiên có số càng nhỏ thì mức ưu tiên càng lớn.

Field	Type	Null	Key	Default	Extra
Dish_group_id	Int		PK	Indentity(1,1)	
Dish_group_name	Nvarchar(30)				
Priority	Int				
Description	Nvarchar(300)	√			

dish

Table description

- Thông tin phần ăn.

Field	Type	Null	Key	Default	Extra
Dish_id	Int		PK	Indentity(1,1)	
Dish_name	Nvarchar(50)				
Price	Float				
Quantity	Int				
Dish_group_id	Int		FK (dish_group.dish_group_id)		
Description	Nvarchar(300)	√			

order_item_group

Table description

- Từng phần ăn sẽ được gom nhóm tùy theo mức độ ưu tiên để gửi vào bếp.
- Tình trạng của nhóm phần ăn: 0: đang đợi, 1: đang làm, 2: đang phục vụ, 3: đã phục vụ.

Field	Type	Null	Key	Default	Extra
OIG_id	Int		PK	Indentity(1,1)	
Priority	Int				
Order_item_group_status	Int			0	
Order_id	Int		FK (order.order_id)		

order_item

Table description

- Lưu trữ thông tin từng suất ăn
- Tình trạng của suất ăn: 0: đang đợi, 1:đang làm, 2:đang phục vụ , 3:đã phục vụ

Field	Type	Null	Key	Default	Extra
Order_item_id	Int		PK	Indentity(1,1)	
Quantity	Int				
Order_item_status	Int			0	
Total	Float	√			
Dish_id	Int		FK (dish.dish_id)		
OIG_id	Int		FK (order_item_group.OIG_id)		
Description	Nvarchar(300)	√			

bill

Table description
- Lưu trữ thông tin hóa đơn - Tình trạng hóa đơn: 0: đang thanh toán 1: đã thanh toán

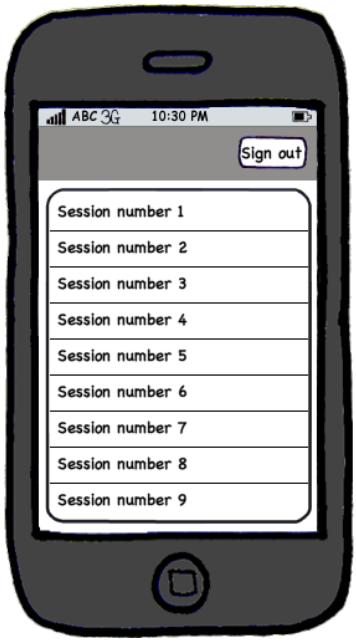
Field	Type	Null	Key	Default	Extra
Bill_id	Int		PK	Identity(1,1)	
Total	Float	√			
Discount	Float	√			
Bill_status	Int			0	
Date_ended	Datetime	√			
Emp_id_ended	Int	√	FK (employee.employee_id)		
Order_id	Int		FK (order.order_id)		

4.4.2 Thiết kế giao diện

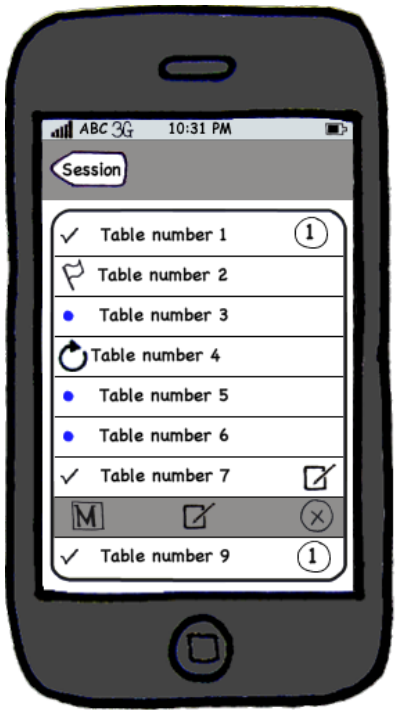
Do mục tiêu chính của đề tài là nghiên cứu nền tảng iOS nên những công việc liên quan đến phân tích thiết kế sẽ được làm rõ ở phần phụ lục.

4.4.2.1 iQsine waiter

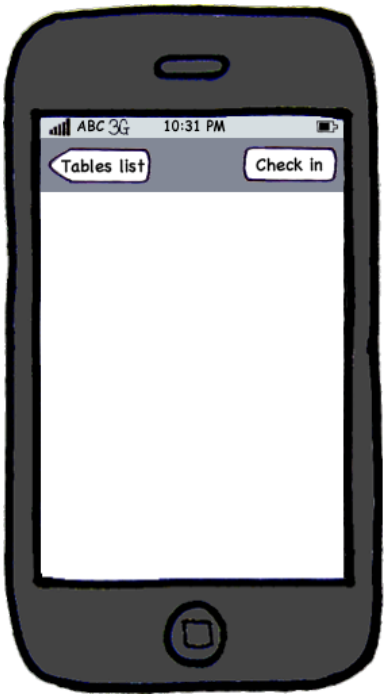
Hình 30 - Giao diện đăng nhập



Hình 31- Giao diện xem danh sách khu vực có trong nhà hàng



Hình 32 - Giao diện xem danh sách bàn trong khu vực



Hình 33 - Check-in một bàn mới

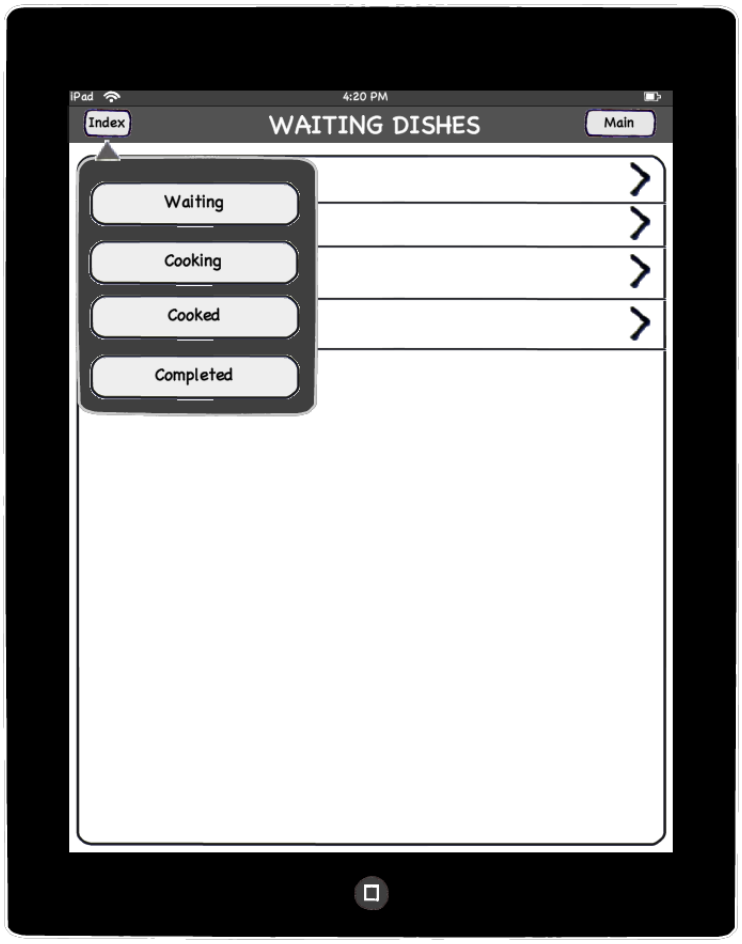
Hình 34 - Giao diện xem thực đơn trong ngày

Hình 35 - Xem danh sách sản phẩm có trong một mục

4.4.2.2

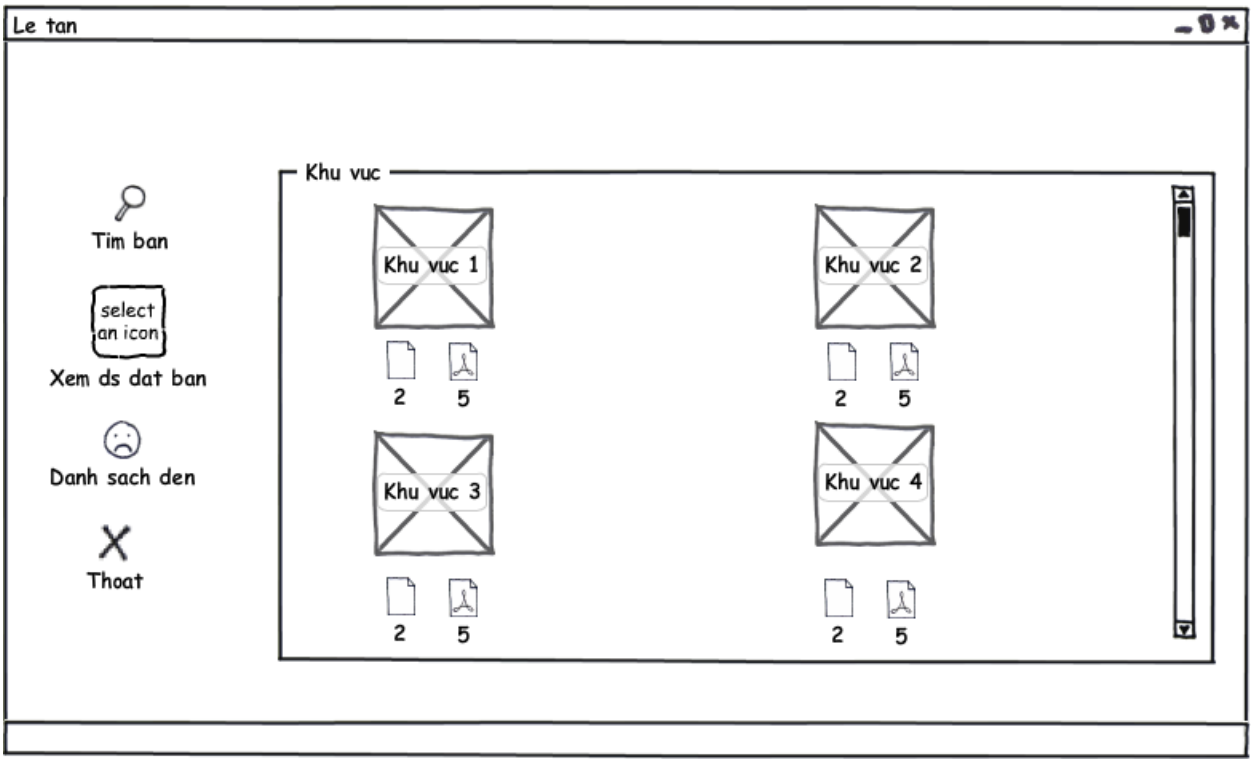
4.4.2.3 iQsine chef

Hình 36 – Giao diện xem danh sách Order Item

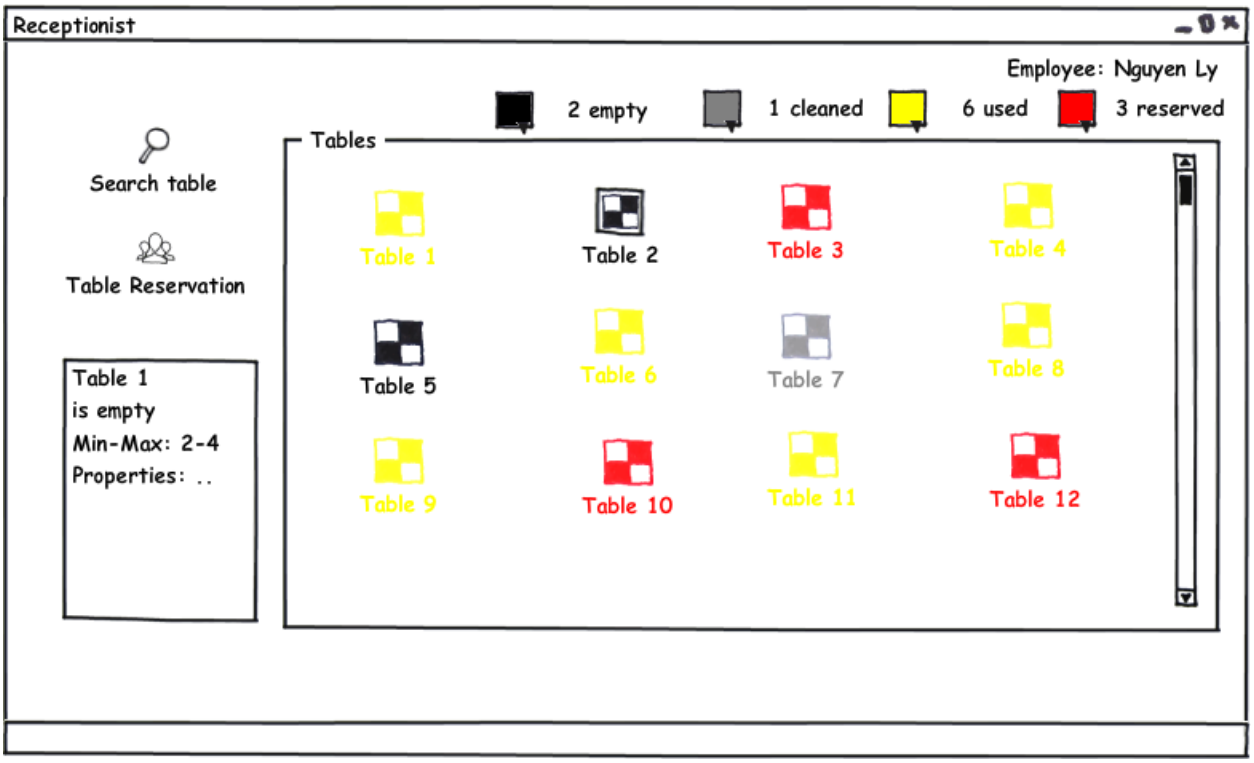


Hình 37 – Giao diện đứng của chương trình iQsine chef

4.4.2.4 iQsine receptionist



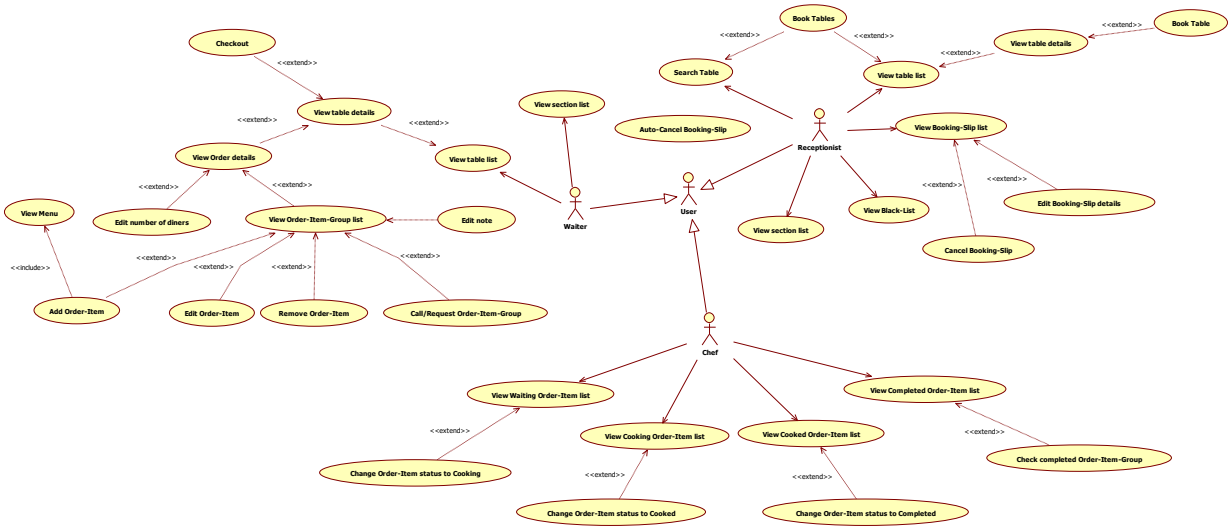
Hình 38 – Giao diện xem danh sách khu vực



Hình 39 – Giao diện xem danh sách bàn có trong khu vực

4.4.3 Phân tích thiết kế chương trình (UC Diagram&Class Diagram)

Dưới đây là mô hình Use case của giải pháp iQsine mô tả hai phần chính của giải pháp là iQsine waiter và iQsine receptionist.



Hình 40 - Mô hình Usecase

(Xem hình lớn ở phần phụ lục)

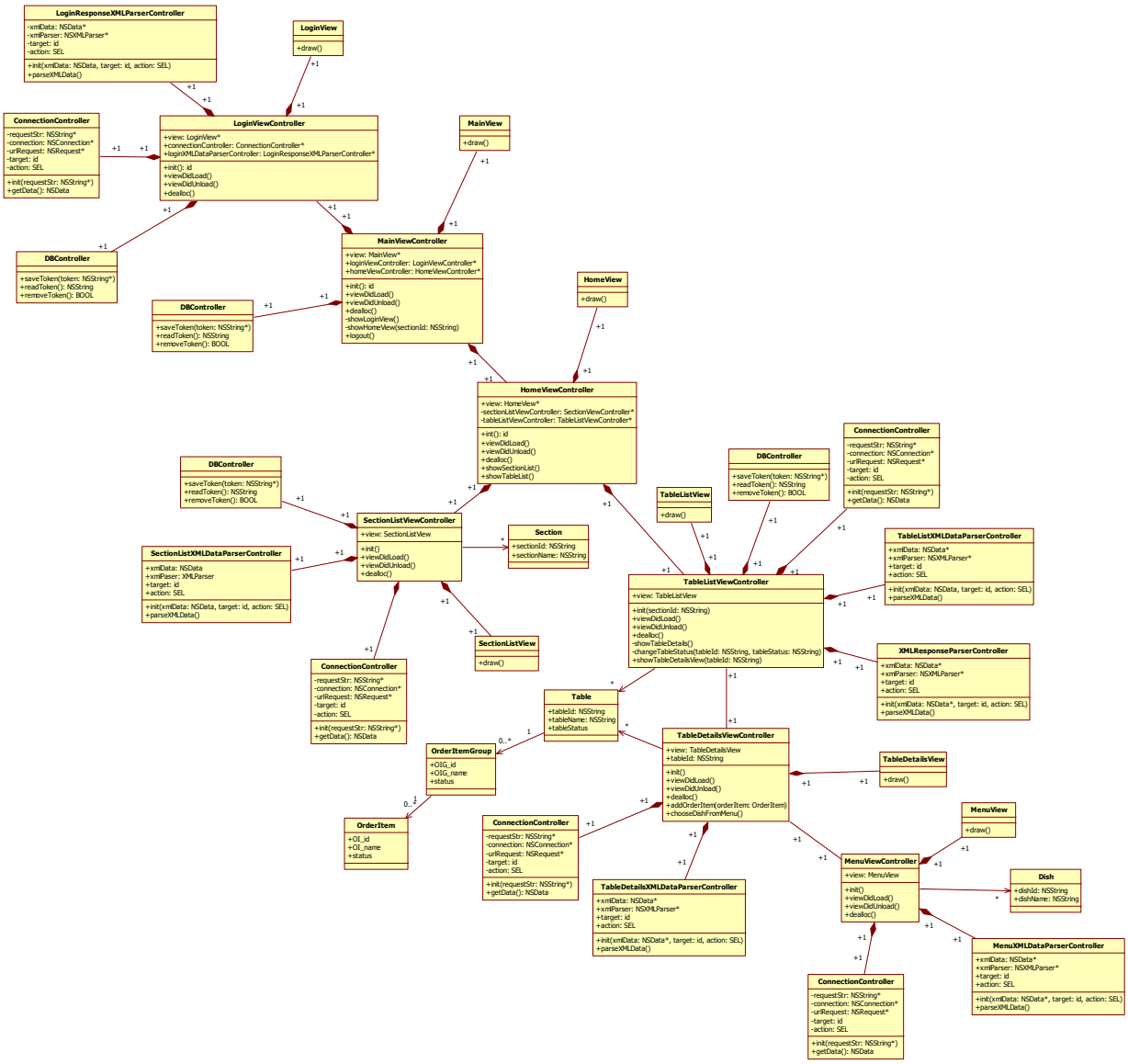
4.4.3.1 iQsine service

iQsine service được thiết kế như là một dịch vụ trung tâm làm nhiệm vụ trung chuyển thông tin giữa các thành phần trong giải pháp cũng như với cơ sở dữ liệu chính (iQsine database). Việc trao đổi thông tin được thực hiện thông qua việc gởi nhận dữ liệu với định dạng XML. Do phần này không nằm trong mục tiêu chính của đề tài nên phần trình bày dưới đây không nhằm mục đích phân tích rõ thiết kế của iQsine service. Phần này sẽ được trình bày rõ hơn trong phần phụ lục của cuốn báo cáo này.

4.4.3.2 iQsine waiter

iQsine waiter là thành phần chính được nhấn mạnh trong giải pháp iQsine và được xem phần thực nghiệm những kiến thức mà nhóm chúng tôi đã nghiên cứu được trong suốt quá trình hiện thực đề tài này. Thiết kế của ứng dụng **iQsine waiter** áp dụng hai pattern chính làMVC và Target-Action.

Hình dưới thể hiện sơ lược Domain của ứng dụng iQsine:



Hình 41 - Sơ đồ Domain của ứng dụng iQsine

(Xem hình lớn ở phần phụ lục)

Sơ lược về phần thiết kế của ứng dụng iQsine:

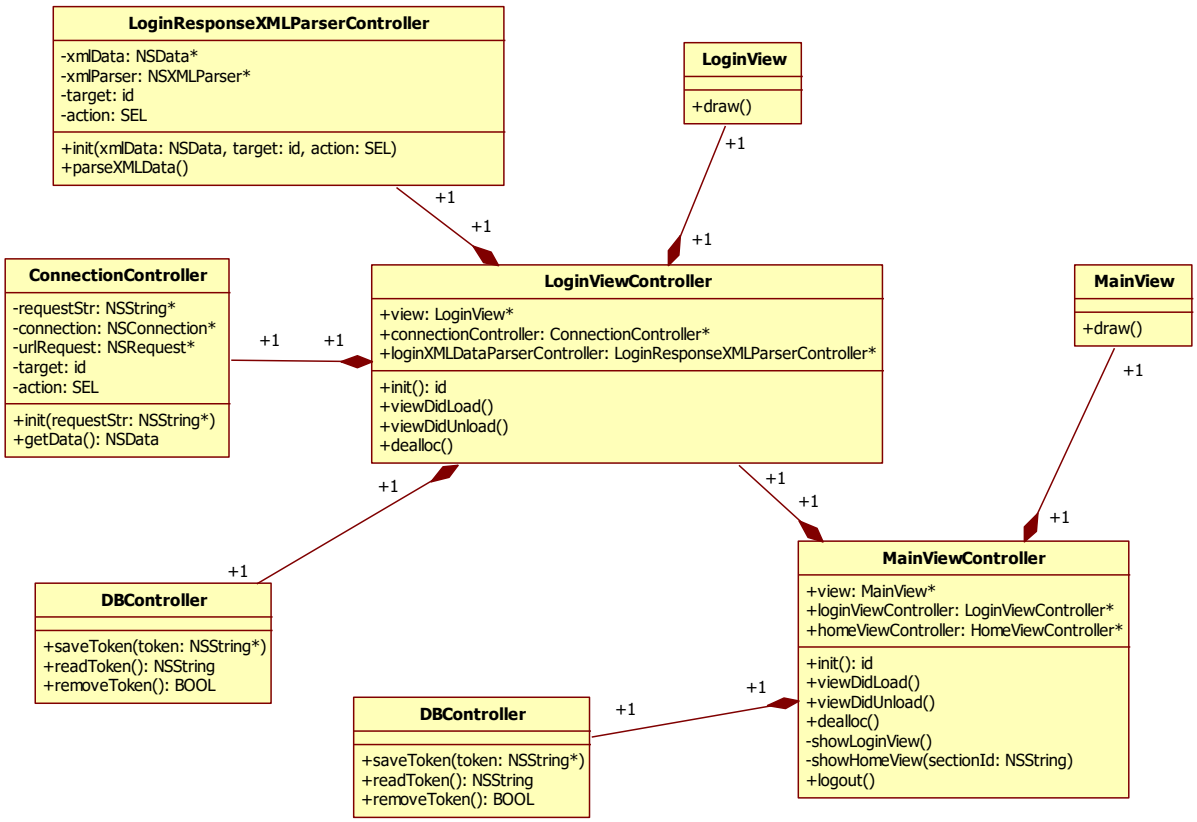
Giao diện chính của ứng dụng là đối tượng MainView được MainViewController quản lý. Mọi nội dung của ứng dụng đều được thể hiện thông qua đối tượng MainView.

Khi ứng dụng iQsine được mở lên, chương trình sẽ khởi tạo đối tượng MainViewControllervà nạp đối tượng View mà nó có nhiệm vụ quản lý lên màn hình chính của ứng dụng (MainView).

Do yêu cầu đặc trưng của ứng dụng nên mỗi lần mở ứng dụng người dùng phải đăng nhập hệ thống. Để thực hiện được việc đăng nhập, chương trìnhsẽ khởi tạo và

nạp đối tượng LoginViewControllervào **NavigationController** (Đã trình bày ở phần trên) (LoginViewControllerquản lý LoginView-dùng để thể hiện giao diện đăng nhập ứng dụng iQsine). Sau khi nạp LoginViewController vào NavigationController thì đối tượng này sẽ nạp View mà nó có nhiệm vụ quản lý lên màn hình chính của ứng dụng. Đây cũng chính là giao diện đăng nhập của chương trình.

Phần mô hình Domain dưới đây thể hiện thiết kế quản lý giao diện chính và giao diện đăng nhập của ứng dụng iQsine:



Hình 42 - Mô hình domain thể hiện thể thiết kế quản lý giao diện chính và giao diện đăng nhập của ứng dụng iQsine

Như đã trình bày ở mô hình trên thể hiện một phần thiết kế của ứng dụng bao gồm các lớp chính như sau:

MainView (Kế thừa UIView): Lớp dùng khởi tạo đối tượng thể hiện giao diện chính của ứng dụng (Quản lý vùng hiện thị giao diện của ứng dụng-Nơi dùng để nạp các phần giao diện khác của ứng dụng).

MainViewController (Kế thừa UIViewController): Lớp dùng khởi tạo đối tượng controller của MainView (Đối tượng điều khiển việc thể hiện nội dung của đối tượng MainView).

LoginView (Kế thừa UIView): Lớp dùng khởi tạo đối tượng thể hiện giao diện đăng nhập của ứng dụng (Quản lý giao diện đăng nhập).

LoginViewController (Kế thừa UIViewController): Lớp dùng khởi tạo đối tượng controller của LoginView (Đối tượng điều khiển việc thể hiện nội dung của đối tượng LoginView).

DBController: Lớp chứa các phương thứcstatic dùng để lưu trữ dữ liệu xuống thư mục chính của ứng dụng iQsine.

ConnectionController:Lớp tự định nghĩa làm nhiệm vụ khởi tạo kết nối với giao thức HTTP, gởi và nhận hồi đáp từ webserver.Lớp này được thiết kế sử dụng để gởi và nhận hồi đáp (Kết quả dữ liệu, lỗi, yêu cầu xác thực quyền…) theo mẫu design-pattern **Target-Action**.Khi khởi tạo thể hiện từ lớp này phải cung cấp cho nó ba tham số là:

- Request cần gởi đi.
- Target-Action: Đích đến và hành động cần thực thi mỗi khi kết thúc việc gởi nhận dữ liệu. Nếu gởi yêu cầu và nhận dữ liệu thành công thì trả về dữ liệu nhận được. Ngược trả về một thể hiện của lớp **Error** (Lớp tự định nghĩa) cung cấp các thông số chi tiết của lỗi. Việc áp dụng mẫu design-pattern tăng khả năng tái sử dụng ở những phần sau của ứng dụng cũng như xây dựng thành một framework riêng.

LoginResponseXMLDataParserController:Lớp tự định nghĩa làm nhiệm vụ parse dữ liệu có định dạng XML. Lớp này được thiết kế sử theo mẫu design-pattern **Target-Action**. Khi khởi tạo thể hiện từ lớp này phải cung cấp cho nó ba tham số là:

- Dữ liệu cần parse.
- Target-Action: Đích đến và hành động cần thực thi mỗi khi kết thúc việc parse dữ liệu. Nếu parse thành công thì trả về dữ liệu phân tích được. Ngược trả về một thể hiện của lớp Error (Lớp tự định nghĩa) cung cấp các thông số chi tiết của lỗi.Việc áp dụng mẫu design-pattern tăng khả năng tái sử dụng ở những phần sau của ứng dụng cũng như xây dựng thành một framework riêng.

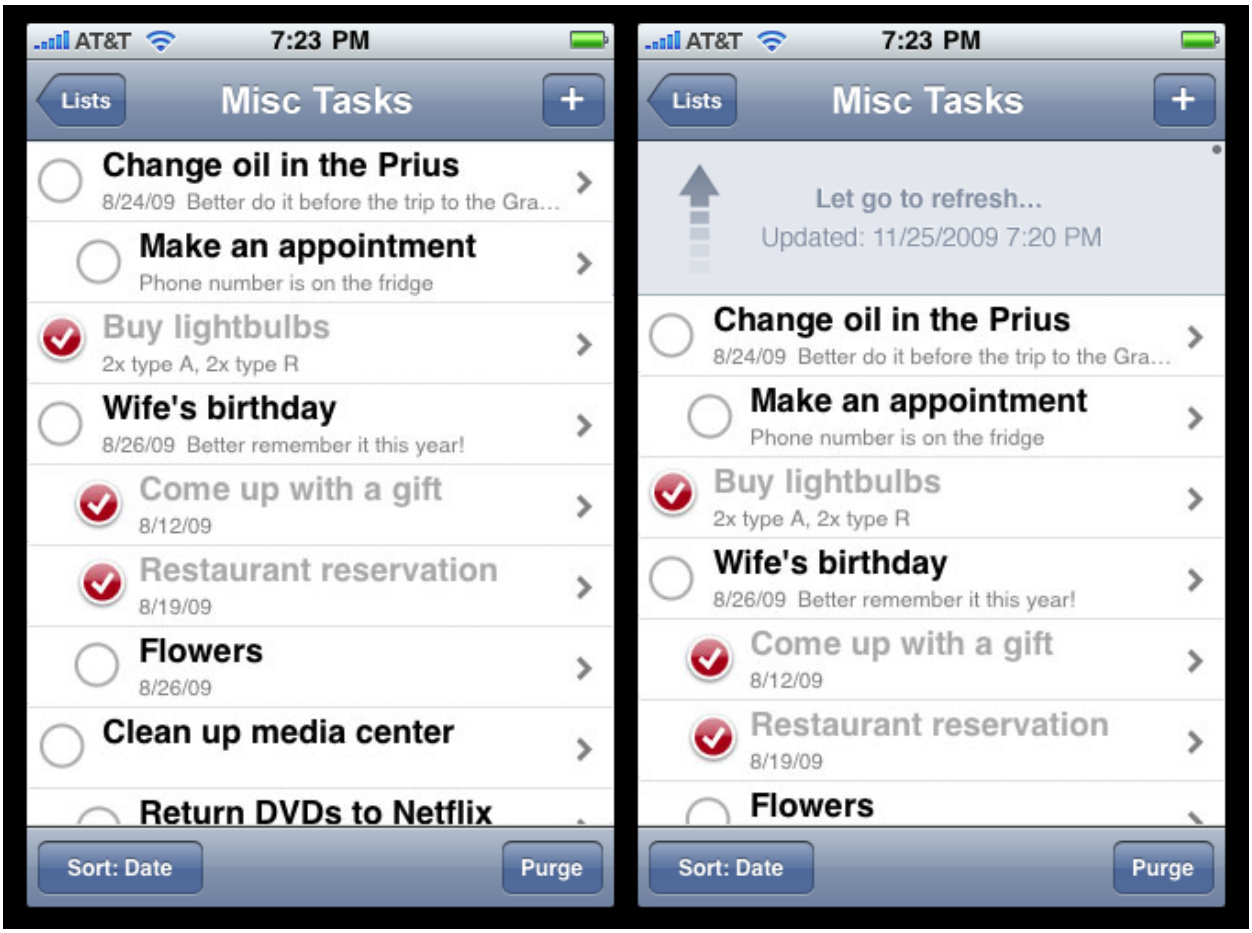
Phần thiết kế trên sử dụng hai design-pattern chính là: MVC model và Target-Action model(Đã được trình bày ở phần Tổng quan lý thuyết). Mô hình trên cũng là mô hình chính của toàn bộ ứng dụng iQsine. Bằng các thiết kế ứng dụng iQsine sử dụng các design pattern nhóm chúng tôi mong muốn tự định nghĩa ra một thư viện nhỏ bao gồm các lớp có khả năng tái sử dụng tốt dùng ứng dụng trong các ứng dụng xây dựng sau này. Những phần còn lại của thiết kế ứng dụng iQsine tuân thủ chặt chẽ mô hình

trên, tái sử dụng triệt để các lớp đã được định nghĩa giúp giảm thời gian thiết kế và cài đặt.

Ngoài ra, nhóm chúng tôi còn sử dụng hai bộ thư viện open-source là:

EGOTableViewPullRefresh:https://github.com/enormego/EGOTableViewPullRefresh

- Đây là bộ thư viện dùng xây dựng giao diện Kéo xuống-để-refresh một UITableView (Đã định nghĩa ở mục trên) giống ứng dụng Facebook trên iOS.

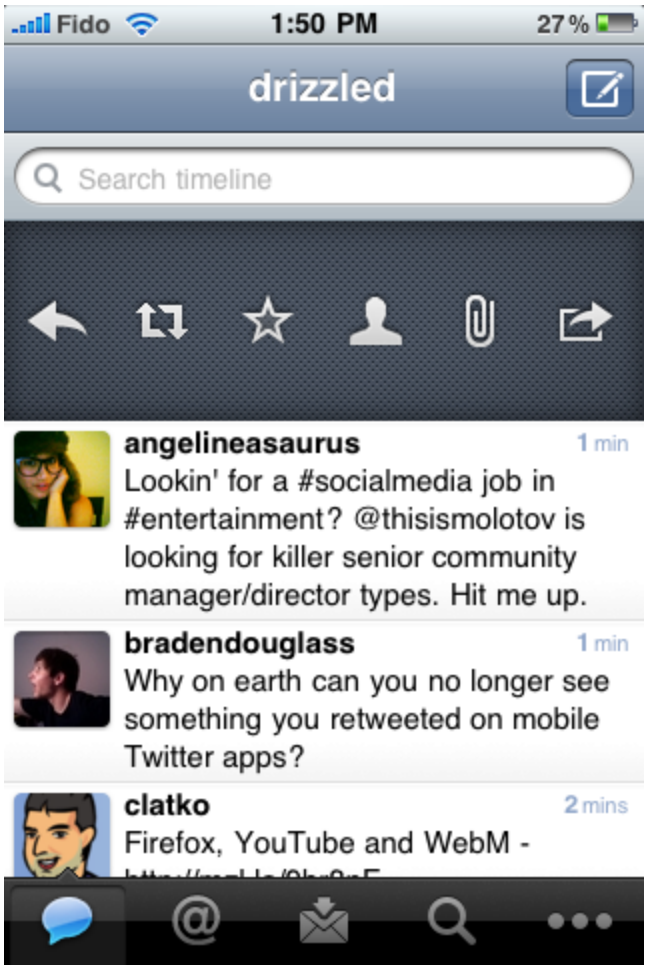


Hình 43 - Giao diện Kéo xuống và thả ra để refresh một UITableView

TISwipeableTableView:http://thermoglobalnuclearwar.com/opensource/

- Đây là bộ thư viện dùng xây dựng giao diện menu được mở bằng cách trượt ngang trên một dòng thuộc một UITableView.

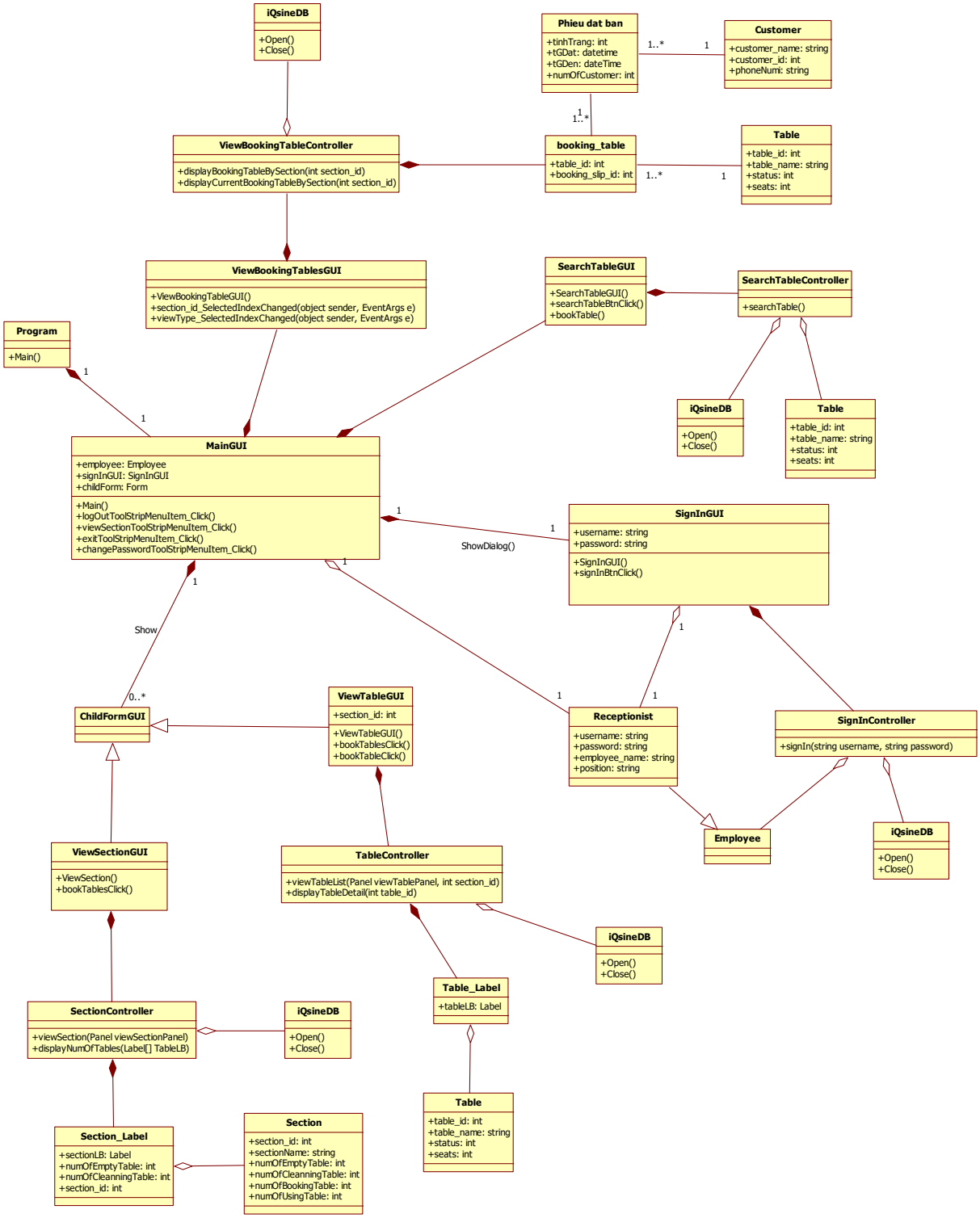
Cả hai bộ thư viện trên đều là mã nguồn mở, được xây dựng để tạo ra hai loại giao diện đặc biệt chỉ tồn tại trên thiết bị sử dụng màn hình cảm ứng.



Hình 44 - Giao diện "Trượt ngang" trên một dòng để mở menu

4.4.3.3 iQsine receptionist

Sơ lược về Class Diagram của ứng dụng iQsine receptionist:



Đối với ứng dụng iQsine receptionist, MainGUI là giao diện chính của chương trình. Khi ứng dụng được bật lên, MainGUI sẽ khởi tạo Form SignInGUIyêu cầu người dùng đăng nhập vào hệ thống. Nếu việc đăng nhập thành công, MainGUI sẽ khởi tạo một Form con là ViewSectionGUI, lớp SectionController sẽ kết nối cơ sở dữ liệu để lấy

lên thông tin các khu vực phục vụ trong nhà hàng. Từ Form ViewSectionGUI, nhân viên có thể xem thông tin bàn ăn bằng cách chọn một khu vực để MainGUI khởi tạo Form ViewTableGUI. Để đặt bàn, nhân viên lễ tân có thể chọn nút Đặt bàn từ form ViewTableGUI hoặc ViewSectionGUI để khởi tạo form SearchTableGUI và thự hiện các thao tác tìm bàn để đặt được bàn thích hợp.

4.5 Cài đặt

4.5.1 Cài đặt database

4.5.1.1 STORED PROCEDURE

4.5.1.1.1 iQisine receptionist

Tên		Usp_bookTable
Mô tả	Mục đích	Lưu thông tin đặt bàn khi có khách muốn đặt bàn
	Input	Tên khách hàng: @customer_name nvarchar(50), Số điện thoại: @phone_num varchar(20), Số lượng khách đến:@num_of_customer int, Yêu cầu của khách: @description nvarchar(300), Thời gian khách đến: @date_arrived datetime, ID của bàn được đặt: @table_id int
	Output	Không có

Tên		usp_changePass
Mô tả	Mục đích	Thay đổi mật khẩu của nhân viên
	Input	@emp_id int, @oldpassword varchar(40), @newpassword varchar(40)
	Output	Không có

Tên		usp_getEmptyTableInfo
Mô tả	Mục đích	Lấy thông tin bàn đang được đặt
	Input	@table_id int
	Output	Tên bàn: table_name, Tình trạng bàn: [status], Tên khách hàng: customer_name, Số điện thoại: phone_num, Số lượng ghế: num_of_seat, Số ghế max:max_seat,

		Ngày đặt bàn: booking_date, Thời gian khách đến:bs.date_arrived, Yêu cầu của khách:[description], Số lượng khách đến: num_of_customer

Tên		**usp_GetNumTable**
Mô tả	Mục đích	Lấy ra thông tin về số lượng bàn trong một khu vực
	Input	@section_id int
	Output	num_of_empty_table, num_of_cleaning_table, num_of_booking_table , num_of_using_table

Tên		**usp_getSection**
Mô tả	Mục đích	Lấy ra các khu vực hiện có
	Input	Không có
	Output	section_id, section_name

Tên		**usp_GetSectionInfo**
Mô tả	Mục đích	Xem thông tin khu vực
	Input	Không có
	Output	s.section_id , section_name, num_of_empty_table, num_of_cleaning_table

Tên		**usp_getTableInfoBySection**
Mô tả	Mục đích	Xem thông tin bàn trong khu vực
	Input	@section_id int
	Output	table_id, [status], t.[description], table_name, position

Tên	**usp_getUsingTableInfo**

Mô tả	Mục đích	Xem thông tin bàn đang sử dụng
	Input	@table_id int
	Output	table_name, [status], num_of_seat, max_seat, num_of_customer, date_created, [description]

Tên		**usp_viewNowBookingTable**
Mô tả	Mục đích	Xem thông tin những bàn đang được đặt
	Input	@section_id int
	Output	t.table_id, t.table_name, [status], seats, bs.[description]

Tên		**usp_signIn**
Mô tả	Mục đích	Kiểm tra đăng nhập
	Input	@username nvarchar(40), @password nvarchar(40)
	Output	emp_id, emp_name

Tên		**usp_update_booking_slip_status**
Mô tả	Mục đích	Hủy phiếu đặt bàn đã quá thời gian nhưng khách chưa tới.
	Input	Thời gian chờ khách, nếu giờ hiện tại + thời gian chờ lớn hơn thời gian đặt bàn thì phiếu đặt bàn sẽ được hủy:@delay_time time
	Output	Không có

Tên		**usp_viewBookingTable**
Mô tả	Mục đích	Xem thông tin bàn đã và đang được đặt
	Input	@section_id int
	Output	t.table_id, t.table_name,

		[status], seats, bs.[description]

Tên		**usp_searchTable**
Mô tả	Mục đích	Tìm bàn trống để đặt
	Input	@num_of_customer int, @time_arrived datetime, @section_id int, @before time, @after time
	Output	table_id, table_name, seats

Tên		**usp_searchTable**
Mô tả	Mục đích	Tìm bàn trống để đặt
	Input	@num_of_customer int, @time_arrived datetime, @section_id int, @before time, @after time
	Output	table_id, table_name, seats

4.5.1.1.2 iQisine waiter

Tên		**usp_getSectionList**
Mô tả	Mục đích	Lấy danh sách khu vực có trong nhà hàng
	Input	Không
	Output	Mã khu vực: section_id Tên khu vực: section_name

Tên		**usp_getSectionList**
Mô tả	Mục đích	Lấy danh sách khu vực có trong nhà hàng

	Input	Không
	Output	Mã khu vực: section_id Tên khu vực: section_name

Tên		**usp_login**
Mô tả	Mục đích	Đăng nhập tài khoản iQsine.
	Input	Tên tài khoản: @username VARCHAR(40) Mật khẩu: @password VARCHAR(40)
	Output	Chuỗi token củaphiên đăng nhập.

Tên		**usp_login**
Mô tả	Mục đích	Đăng nhập tài khoản iQsine.
	Input	Tên tài khoản: @username VARCHAR(40) Mật khẩu: @password VARCHAR(40)
	Output	Chuỗi token củaphiên đăng nhập.

Tên		**usp_checkInTable**
Mô tả	Mục đích	Chekin bàn. Store procedure này sẽ chuyển trạng thái đặt bàn và tạo order.
	Input	Mã phiếu đặt bàn: @bookingSlipId INT Mã bàn: @tableId INT Số lượng khách: @numberOfDinners INT @employeeCreated INT
	Output	Không.

Tên		**usp_getDishList**
Mô tả	Mục đích	Lấy danh sách món ăn trong ngày.
	Input	Không.
	Output	ID của món ăn: dish_id Tên món ăn: dish_name Độ ưu tiên của món ăn: priority Số lượng: quantity

Tên		**usp_addOrderItemGroup**
Mô tả	Mục đích	Tạo OIG mới.
	Input	Order Item cần tạo OIG: @orderId VARCHAR(40) Độ ưu tiên của OIG: @priority INT

	Output	1: Tạo thành công. 2: Tạo thất bại.

Tên		usp_addOrderItem
Mô tả	Mục đích	Tạo OI mới.
	Input	Order Item Group cần tạo OI: @orderItemGroupId VARCHAR(40) Mã món ăn: @dishId VARCHAR(4) Số lượng phần: @numberOfServings INT Ghi chú: @note VARCHAR(300)
	Output	1: Tạo thành công. 2: Tạo thất bại.

Tên		usp_callOrderItemGroup
Mô tả	Mục đích	Gọi chuẩn bị hay gọi làm một OIG.
	Input	OIG muốn gọi: @orderItemGroupId INT Trạng thái mới: @newStatus INT
	Output	1: Chuyển trạng thái thành công. 2: Chuyển trạng thái thất bại.

Tên		usp_editOrderItem
Mô tả	Mục đích	Gọi chuẩn bị hay gọi làm một OIG.
	Input	ID của Order Item: @orderItemId INT Số lượng phần ăn: @quantity INT Ghi chú: @note NVARCHAR(300)
	Output	1: Thay đổi OI thành công. 2: Thay đổi OI thất bại.

4.5.2 Cài đặt chương trình

5 Tổng kết và đánh giá

Sức hấp dẫn của thị trường di động đang lan tỏa khắp nơi với hai đại diện ưu tú là: Android và iOS. Android với sức mạnh là một HĐH di động mã nguồn mở mang lại khả năng mở rộng và phát triển cực kì mạnh mẽ. HĐH iOS với sức mạnh được hỗ trợ bởi nền tảng iOS được phát triển bởi Apple Inc. Trong đó nền tảng iOS bao gồm hệ điều hành iOS, các thiết bị sử dụng HĐH iOS với các tính năng độc quyền (Cảm ứng đa điểm) cùng với bộ công cụ phát triển được phát triển hoàn toàn bởi Apple Inc cung cấp một khả năng phát triển, kiểm thử và cài đặt ứng dụng một cách thống nhất, tương

thích đầy đủ giữa các thành phần trong bộ công cụ phát triển. Nền tảng iOS cung cấp một khả năng phát triển ứng dụng dễ dàng thông qua bộ công cụ đầy đủ cũng như bộ tài liệu phát triển được chuẩn bị rất kĩ lưỡng bởi Apple Inc. Trái ngược với Android cũng như các nền tảng di động mã nguồn mở khác, nền tảng kiến trúc của HĐH iOS được Apple Inc giữ kín tránh xa khỏi mọi sự can thiệp của người phát triển. Bằng cách cung cấp các công nghệ hỗ trợ thông qua các Framework, Apple Inc đẩy sự phức tạp của các khái niệm liên quan đến hệ thống thông qua việc trừu tượng hóa chúng trong các công nghệ được cung cấp thông qua các bộ framework. Bằng cách phân chia nền tảng iOS thành mô hình bốn lớp và cung cấp sự tiện lợi cho nhà phát triển ứng dụng bằng cách hướng họ sử dụng các framework được cung cấp ở các lớp mức trên. Tuy nhiên, Apple Inc vẫn cho phép nhà phát triển sử dụng các công nghệ ở gần mức hệ thống hơn thông qua các framework ở mức thấp hơn với một sự hạn chế nhất định. Bằng cách này mà nền tảng HĐH iOS được bảo vệ trách xa sự can thiệp trực tiếp không qua kiểm soát.

Ngoài ra nền tảng iOS còn đi kèm với các thiết bị sử dụng công nghệ độc quyền (Cảm ứng đa điểm) cung cấp một trải nghiệm đặc biệt mà chỉ sản phẩm của Apple Inc mới có thể cung cấp cho người sử dụng. Hơn thế nữa, ngoài smartphone sử dụng HĐH iOS, Apple Inc còn cung cấp các thiết bị sử dụng iOS khác như iPad, iPod cung cấp những trải nghiệm phù hợp với nhu cầu của người sử dụng.

Nhận thấy một tiềm năng rất lớn trong việc ứng dụng thiết bị iPod để giải quyết bài toàn quản lý phục vụ nhà hàng nhóm chúng tôi đã quyết định sử dụng những kiến thức đã tích lũy được trong quá trình tìm hiểu, nghiên cứu để xây dựng giải pháp iQsine nhằm mục đích thực nghiệm.

Trong suốt quá trình tìm hiểu, nghiên cứu và thực nghiệm, nhóm chúng tôi đã rút ra được những kết quả như sau:

- Hiểu được tổng quan hệ điều hành iOS, mô hình tương tác, các công nghệ hỗ trợ và framework tương ứng. Có thể chỉ rõ được cách thực hợp tác của các thành phần cơ bản của hệ điều hành iOS, các thành phần cơ bản của một ứng dụng và cách thức hợp tác giữa chúng.
- Nắm được cách thức phát triển ứng dụng cho HĐH iOS bằng ngôn ngữ lập trình Objective-C. Chúng tôi đã nắm được các cú pháp cơ bản trong lập trình bằng Objective-C, cách thức quản lý bộ nhớ của ứng dụng, tối ưu hóa thực thi. Có thể áp dụng các design-pattern mặc định của HĐH iOS để thiết kế ứng dụng.
- Tìm hiểu được một số **design pattern** dùng để nâng cao hiệu suất, tăng khả năng tái sử dụng cũng như làm tăng tính dễ hiểu của ứng dụng.

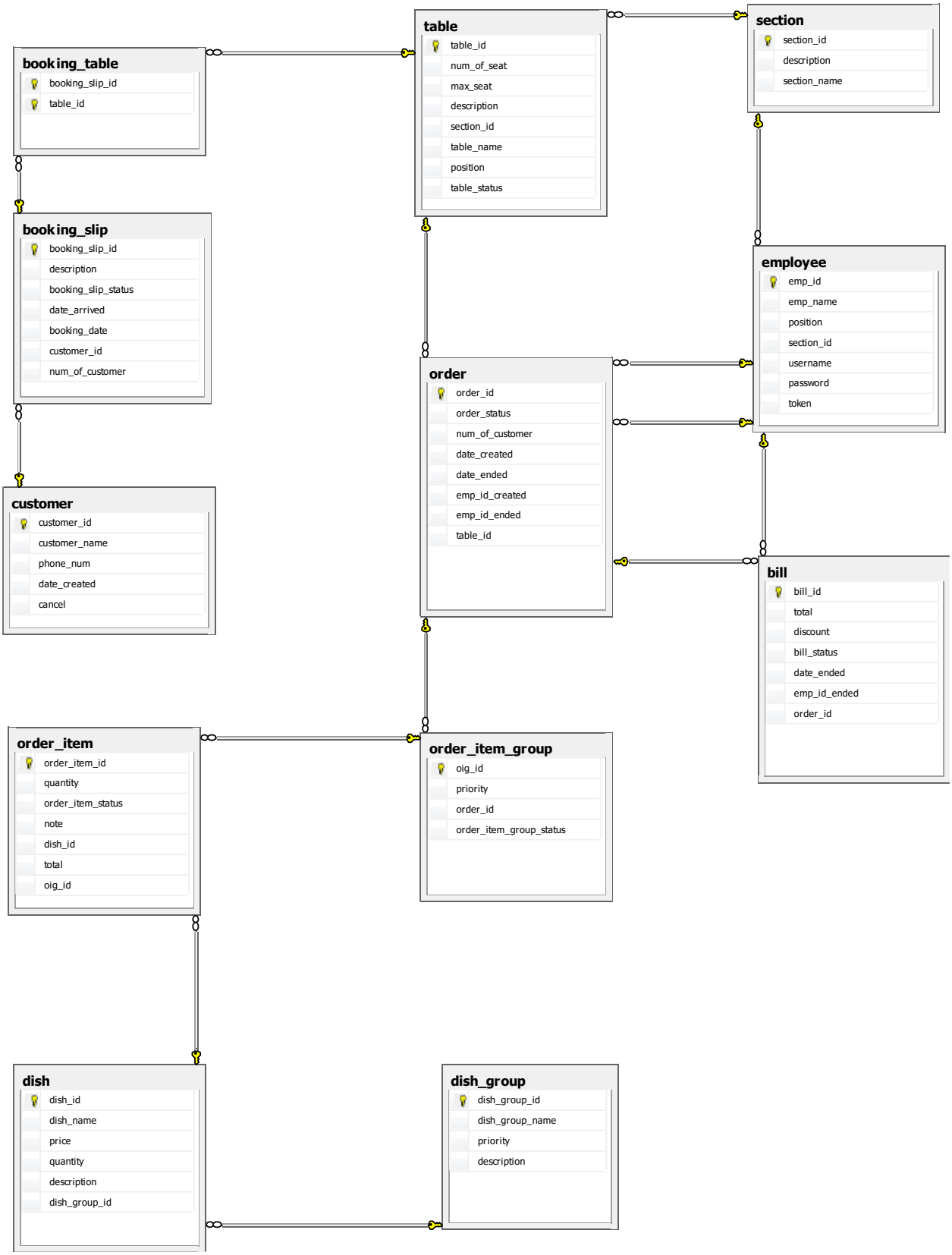
- Vận dụng những kiến thức tìm hiểu được để xây dựng giải pháp iQsine trên nền tảng iOS.
- Tích lũy và xây dựng được một bộ thư viện gồm một tập các lớp (class) được định nghĩa bằng Objective-C được sử dụng trong ứng dụng iQsine và có thể tái sử dụng được trong các ứng dụng tương lai.
- Tìm hiểu và phát triển một quy trình xây dựng ứng dụng cho thiết bị smartphone phù hợp với yêu cầu làm việc nhóm. Nhóm chúng tôi đã tự xây dựng được một quy trình làm việc phù hợp với thời gian biểu cũng như phù hợp với môi trường phát triển ứng dụng thực nghiệm (Cắt giảm một số bước không cần thiết).

Do thời gian và sức lực hạn chế nên nhóm làm đề tài đã giới hạn lại giải pháp iQsine ở nhiều mặt. Dựa trên tính thực tiễn cao và tiềm năng lớn của giải pháp, nhóm hy vọng trong tương lai sẽ có thể tiếp tục cải thiện và hoàn thiện giải pháp iQsine với những bổ sung sau:

- Quản lý nhân sự, kho chứa hàng, chế độ khuyến mãi-ưu đãi khách hàng thân thiết.
- Hệ thống phân tích và gợi ý món ăn cho thực đơn ngày hôm sau dựa trên nguyên vật liệu ở thời điểm hiện tại.
- Mở rộng chức năng phân tích-thống kê dữ liệu nhà hàng.
- Hỗ trợ xuất báo cáo chi tiết các dữ liệu theo ngày, tuần, tháng, quý, năm với mẫu báo cáo có sẵn hoặc tùy chỉnh theo ý nhà hàng.
- Mở rộng hỗ trợ các hệ điều hành khác nhau (Windows, Mac, Linux), có thể phát triển một giao diện website.

6 Phụ lục

6.1 Database Diagram


booking_table
booking_slip_id
table_id
table
table_id
num_of_seat
max_seat
description
section_id
table_name
position
table_status
section
section_id
description
section_name
booking_slip
booking_slip_id
description
booking_slip_status
date_arrived
booking_date
customer_id
num_of_customer
employee
emp_id
emp_name
position
section_id
username
password
token
order
order_id
order_status
num_of_customer
date_created
date_ended
emp_id_created
emp_id_ended
table_id
customer
customer_id
customer_name
phone_num
date_created
cancel
bill
bill_id
total
discount
bill_status
date_ended
emp_id_ended
order_id
order_item
order_item_id
quantity
order_item_status
note
dish_id
total
oig_id
order_item_group
oig_id
priority
order_id
order_item_group_status
dish
dish_id
dish_name
price
quantity
description
dish_group_id
dish_group
dish_group_id
dish_group_name
priority
description

7 Tài liệu tham khảo

Mark, D&LaMarche, J , 2009,*Beginning iPhone Development: Exploring the iPhone SDK*, 3rd edition, Apress, New York.

Dalrymple, M &Knaster, S, 2009, *Learn Objective-C on the Mac*, Apress, New York.

Apple Inc, 2010, *iOS overview*, Apple Inc, viewed 01, September 2010, http://developer.Apple Inc.com/library/ios/#referencelibrary/GettingStarted/URL_iPhone_OS_Overview/index.html#//Apple Inc_ref/doc/uid/TP40007592

Buchanan, M, 2010, *How Multitasking Works on a Phone,*Gizmodo,viewed 01, September 2010, http://gizmodo.com/5527407/giz-explains-how-multitasking-works-on-a-phone

Anthony, 2010, *An overview of iPhone architecture,* The Coffee Desk 2010 ,viewed 01, September 2010, http://thecoffeedesk.com/news/index.php/2009/05/17/iphone-architecture/

Apple Inc developer website: http://developer.Apple Inc.com

The iPhone Blog:

June 2010: http://www.tipb.com/2010/06/07/100000000-ios-devices-5-billion-apps-wwdc-numbers/

April 2010: http://www.tipb.com/2010/04/08/50-million-iphones-sold-35-million-ipod-touches-85-million-iphone-os-devices/

July 2010:http://www.tipb.com/2010/06/07/100000000-ios-devices-5-billion-apps-wwdc-numbers/

App of the day, Decemeber 2010: http://appoftheday.com/infographic

Distimo, November 2010: http://www.distimo.com/report