**TRƯỜNG ĐẠI HỌC ĐIỆN LỰC**

Khoa Công Nghệ Thông Tin

-----🙞🙜🕮🙞🙜-----



**ĐỒ ÁN MÔN HỌC**

**PHÂN TÍCH THIẾT KẾ HƯỚNG ĐỐI TƯỢNG**

**Đề Tài: Ứng dụng quản lí bất động sản cho thuê**

Giảng viên hướng dẫn : Ths. Nguyễn Thị Hồng Khánh

Sinh viên thực hiện : Phạm Xuân Duy

Lớp : D9-CNPM

Mã sv : 1481310017

***Hà Nội 5/2017***

Mục lục

[**I. Nắm bắt yêu cầu** 4](#_Toc482829326)

[**I.1. Mô tả bài toán** 4](#_Toc482829327)

[**I.1.1.Bài toán** 4](#_Toc482829328)

[**I.1.2. Thực trạng và vấn đề cần giải quyết** 4](#_Toc482829329)

[**I.1.3. Hệ thống đã tin học hóa** 5](#_Toc482829330)

[**I.1.4. Vấn đề cần giải quyết** 7](#_Toc482829331)

[**I.2. Mục tiêu của hệ thống** 9](#_Toc482829332)

[**I.3.Các chức năng hệ thống** 9](#_Toc482829333)

[**I.4. Các thuộc tính của hệ thống** 9](#_Toc482829334)

[**I.5. Xác định các khái niệm** 10](#_Toc482829335)

[**I.6. Mô hình khái niệm** 10](#_Toc482829336)

[**I.7. Xác định các tác nhân, các ca sử dụng và mô tả các ca sử dụng** 11](#_Toc482829337)

[**I.7.1. Xác định các tác nhân** 11](#_Toc482829338)

[**I.7.2. Xác định các ca sử dụng** 12](#_Toc482829339)

[**I.7.3 Biểu đồ ca sử dụng gói** 13](#_Toc482829340)

[**I.7.4 Mô tả các ca sử dụng** 15](#_Toc482829341)

[**II. Phân tích** 20](#_Toc482829342)

[**II.1. Biểu đồ tuần tự hệ thống** 20](#_Toc482829343)

[**II.2. Ghi nhận các thao tác hệ thống** 21](#_Toc482829344)

[**II.3. Hợp đồng cho các thao tác hệ thống** 22](#_Toc482829345)

[**II.4. Mô hình phân tích** 26](#_Toc482829346)

[**II.4.1. Gói cập nhật sinh viên** 26](#_Toc482829347)

[**II.4.2. Quản lí phí đoàn** 26](#_Toc482829348)

[**II.4.3 Quản lí khen thưởng, kỷ luật** 27](#_Toc482829349)

[**II.4.4. Xét lên Đảng viên** 28](#_Toc482829350)

[**II.4.5. Quản lí hoạt động đoàn** 28](#_Toc482829351)

[**III. Thiết kế** 29](#_Toc482829352)

[**III.1. Biểu đồ tuần tự đối tượng** 29](#_Toc482829353)

[**III.2. Biểu đồ cộng tác** 31](#_Toc482829354)

[**III.3. Biểu đồ lớp** 34](#_Toc482829355)

[**IV. Kết luận** 34](#_Toc482829356)

**Lời cảm ơn**

Ngày nay, việc ứng dụng công nghệ thông tin đã trở nên phổ biến trong hầu hết mọi cơ quan, doanh nghiệp, trường học, đặc biệt là việc áp dụng các giải pháp tin học trong công tác quản lý.

Với tốc độ phát triển của công nghệ thông tin, làm cho cuộc sống của con người trở nên thú vị và đơn giản hơn. Vì vậy, để bắt kịp với nhịp độ phát triển của xã hội, những kiến thức được học trên trường là vô cùng quan trọng đối với mỗi sinh viên chúng em.

Trong quá trình thực hiện bài tập này, chúng em luôn nhận được sự hướng dẫn, chỉ bảo tận tình của cô Nguyễn Thị Hồng Khánh, giảng viên khoa công nghệ thông tin trường Đại học Điện lực, cô đã giành nhiều thời gian hướng dẫn, giúp đỡ tận tình chúng em trong quá trình học tập.

Em xin chân thành gửi lời cảm ơn sâu sắc tới cô, cô đã tận tình giảng dạy và truyền đạt cho chúng em những kiến thức, kinh nghiệm quý báu trong suốt thời gian học tập và rèn luyện.

Hà nội, tháng 11 năm 2018

Sinh viên thực hiện:Phạm Xuân Duy.

**Chương 1. Tổng quan lập trình di động.**

**1.1.** Giới thiệu về lập trình di động.

Ngày nay, điện thoại thông minh (smart phone) đã trở thành 1 thiết bị hết sức phổ biến và không thể thiếu trong cuộc sống của chúng ta. Chính vì vậy, nhu cầu xây dựng những ứng dụng và hệ sinh thái trên các nền tảng di động như Android hay iOS đã trở nên cấp thiết hơn bao giờ hết. Chỉnh bởi thị trường ứng dụng di động đang là 1 mảnh đất béo bở nên rất nhiều doanh nghiệp IT đã và đang cùng nhảy vào mảng sản xuất này. Điều này đã dẫn đến thiếu hụt 1 lượng lớn nhân lực có chất lượng trong lĩnh vực phát triển ứng dụng di động. Theo Vietnamworks, dự tính đến năm 2020, Việt Nam cần khoảng 20.000 lập trình viên cho mảng này.

Lập trình viên Ứng dụng di động, ngoài những kỹ năng về lập trình nền tảng, thành thạo các ngôn ngữ & công nghệ trên các nền tảng di động (Android, iOS) thì còn phải có các kỹ năng về thiết kế trải nghiệm người dùng (UX), xây dựng Back-End cho các Ứng dụng di động nữa.

Công việc liên quan đến Lập trình Ứng dụng di động:

* Lập trình viên Ứng dụng di động nền tảng Android
* Lập trình viên Ứng dụng di động nền tảng iOS
* Lập trình viên Back-End cho Ứng dụng di động

Theo thống kê và đánh giá của Cổng thông tin việc làm MyWork: “Mức lương lương trung bình của các bạn làm trong lĩnh vực Lập trình Ứng dụng di động thuộc mức cao của nhóm tuyển dụng lập trình, vào khoảng từ 7 – 12 triệu”. Đặc biệt, có một số bạn có những sản phẩm thành công tự phát hành trên Store thì có đạt mức thu nhập lên tới 1000USD / tháng.

Với doanh số bán điện thoại thông minh (smartphone) dự đoán sẽ đạt 1,4 tỷ chiếc trong năm 2016, chưa bao giờ có một thời điểm tốt hơn để nhảy vào lĩnh vực phát triển ứng dụng di động. Kể từ khi iPhone ra mắt năm 2007, ứng dụng di động đã trở thành một ngành công nghiệp lớn với nguồn lợi nhuận khổng lồ.

Những năm sau đó, đã có rất nhiều đối thủ khác muốn truất ngôi vương của Apple, nổi bật nhất là [Android.](https://techmaster.vn/khoa-hoc/25513/lap-trinhandroid" \t "_blank) Hai nền tảng này chiếm hơn 90% thị trường điện thoại di động, và đến năm 2015, đã có 2,6 tỷ điện thoại thông minh và gần 3 triệu ứng dụng đang được sử dụng

2.2. Các phương pháp lập trình trên di động.

- Java : Năm 2015 là năm Java kỷ niệm sinh nhật lần thứ 20 của mình, có rất nhiều lý do để ăn mừng. Java là một trong những ngôn ngữ lập trình được sử dụng rộng rãi nhất trên thế giới với ước tính khoảng 9 triệu nhà phát triển.

+ Hệ điều hành Android của Google sử dụng Java như là cơ sở cho tất cả các ứng dụng Android. Trong khi Android Java là không hoàn toàn giống như Java thông thường, nhưng nó cũng có nhiều điểm chung - vì vậy học Java sẽ giúp ích rất nhiều khi bạn phát triển ứng dụng cho Android.

Android hiện đang chạy trên khoảng 60% trong tất cả các thiết bị di động, có nghĩa là nếu bạn muốn học một ngôn ngữ lập trình với tiềm năng lớn nhất, thì Java là ngôn ngữ mà bạn nên lựa chọn.

* Swift : Các ứng dụng iOS trước đây hoàn toàn được viết bằng ngôn ngữ Objective-C. Trong một nỗ lực để đơn giản hóa việc học và phát triển ứng dụng, Apple đã phát hành ngôn ngữ lập trình riêng của họ dành cho iOS và OS X gọi là Swift.

+, Swift được thiết kế để cung cấp những điều kiện tốt nhất cho phát triển ứng dụng iOS và OS X, và chính sự đơn giản của Swift giúp cho bạn học lập trình được dễ dàng hơn.

Vì đây là một ngôn ngữ lập trình mới với rất nhiều sự quảng cáo ngày càng tăng xung quanh nó, Swift có thể là một trong những ngôn ngữ có giá trị nhất mà bạn nên học để giúp bạn dễ dàng kiếm được việc làm. Đây là một tin tuyệt vời nếu bạn đang tìm kiếm một nghề nghiệp trong lĩnh vực phát triển ứng dụng cho iOS hay OS X

## - [HTML5 + JavaScript](https://techmaster.vn/khoa-hoc/25487/web-co-ban-html5-css3-va-javascript" \t "_blank)

HTML là ngôn ngữ đánh dấu được sử dụng để làm các trang web. Bạn không thể làm ứng dụng bằng nó, nhưng bạn có thể kết hợp HTML5 với JavaScript để tạo ra các ứng dụng di động và web.

Việc phát triển ứng dụng bằng HTML5 có những ưu điểm khi so sánh với việc xây dựng ứng dụng native vì bạn thường chỉ phải viết code ứng dụng một lần, và ứng dụng có thể chạy trên nhiều nền tảng khác nhau: các thiết bị iOS, Android, desktop, v.v... Điều này sẽ giúp giảm thời gian phát triển và chi phí bảo trì của ứng dụng.

Tuy nhiên, ngay cả với những lợi ích này, sẽ có một sự đánh đổi về tốc độ thực thi. Ứng dụng native có quyền truy cập vào toàn bộ framework phát triển của hệ điều hành, được tối ưu hóa để cung cấp hiệu suất tốt nhất. Các ứng dụng HTML5 có thể là tuyệt vời cho sự linh hoạt và đa nền tảng, nhưng có thể hiệu suất ứng dụng sẽ ở dưới mức trung bình.

Nếu ứng dụng của bạn cần truy cập vào phần cứng trên thiết bị, như camera, thì rất khó để có thể sử dụng HTML5.

Có một cách để khắc phục nhược điểm trên đó là bằng cách tạo ra một ứng dụng dạng "lai", một ứng dụng HTML5 được wrapper trong một ứng dụng native. Điều này tận dụng được tính linh hoạt và bảo trì thấp của HTML5, nhưng vẫn có thể sử dụng được các tính năng của ứng dụng native tạo ra trải nghiệm tốt hơn cho người dùng.

1.3. Kết luận chương 1.

**Chương 2. Phân tích hệ thống quản lí bất động sản cho thuê**

* 1. Giới thiệu về hệ thống quản lí bất động sản cho thuê.

Trong những năm gần đây dịch vụ cho thuê nhà trọ là một trong những ngành có độ tăng trưởng cao nhất cả nước. Rất nhiều nhà trọ đua nhau phát triển liên tục và nhanh chóng theo sự phát triển của xã hội về qui mô và chất lượng.

Hiện nay, các nhà trọ phải trực tiếp tiếp nhận, quản lý một khối lượng lớn và thường xuyên nhiều loại khách, cùng với hàng loạt dịch vụ phát sinh theo nhu cầu của khách hàng. Do đó, công việc quản lý hoạt động kinh doanh của nhà trọ ngày càng phức tạp hơn.

Hơn nữa, công tác quản lý không chỉ đơn thuần là quản lý về lưu lược khách đến với nhà trọ, sử dụng các loại hình dịch vụ … mà công việc quản lý còn phải đáp ứng nhu cầu về việc báo cáo các loại hình doanh thu, tình hình kinh doanh của nhà trọ … để từ đó có thể đưa ra định hướng và lập kế hoạch phát triển cho công việc kinh doanh đó. Nhưng với việc lưu trữ và xử lý bằng thủ công như hiện nay thì sẽ tốn rất nhiều thời gian và nhân lực mà không đem lại hiệu quả cao. Do đó cần phải tin học hóa hình thức quản lý, cụ thể là xây dựng một phần mềm để đáp ứng nhu cầu quản lý toàn diện, thống nhất và đạt hiệu quả cao nhất cho hoạt động kinh doanh của nhà trọ.

Do những nhu cầu trên nên nhóm chúng em quyết định chọn đề tài là “Quản lý bất động sản cho thuê” như là một chính yếu cho nhu cầu ứng dụng công nghệ thông tin vào kinh doanh.

Xây dựng phần mềm quản lý nhà trọ cung cấp đầy đủ thông tin về người thuê, về loại phòng thuê, tình trạng phòng thuê và các dịch vụ đi kèm với phòng thuê đó

Giúp cho việc quản lí nhà trọ trở nên dễ dàng hơn, giảm thiểu lao động tay chân. Mang lai tính chính và hiệu quả cao trong việc thu chi mỗi tháng.

* 1. Phân tích yêu cầu về việc thuê bất động sản.
  2. Các công nghệ sử dụng trong hệ thống.

Sử dụng web ASP core để làm API

Sử dụng ngôn ngữ Swift ,

Hệ thống database mysql, realm database

* 1. Kết luận chương 2.

**Chương 3. Phát triển ứng dụng quản lí bất động sản cho thuê**

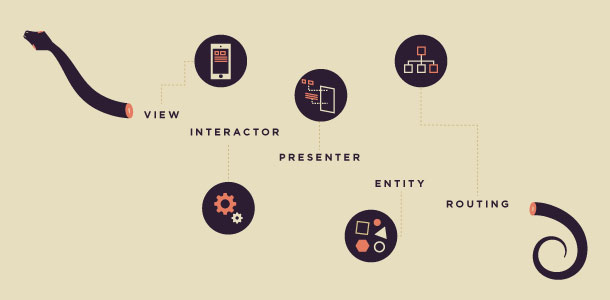
* 1. Mô hình áp dụng vào lập trình IOS.
* Mô hình VIPER :
* Với mỗi Developer, một điều quan trọng và gần như là mối quan tâm hàng đầu khi bắt đầu viết bất kỳ dòng code nào đó là làm cách nào để có thể thiết kế code của chúng ta sao cho mỗi phần trong đó phải thật dễ hiểu, dễ xác định, phải có mục đích rõ ràng, hiển nhiên, và phải phù hợp với các phần khác theo cách hợp lý. Một kiến trúc tốt không thể làm nên thành công của 1 sản phẩm, nhưng nó sẽ giúp chúng ta dễ dàng hơn trong việc bảo trì, cải tiến.
* VIPER là một sự áp dụng [Clean Architecture](http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html) vào iOS/Android app. Từ **VIPER** được viết tắt bởi **View, Interactor, Presenter, Entity và Routing**. Clean Architecture phân chia kiến trúc logic của một ứng dụng thành các lớp đảm nhận những trách nhiệm riêng biệt. Điều này làm cho nó dễ dàng cô lập các thành phần phụ thuộc (dependencies) như: database,… và kiểm thử (test) sự tương tác (interaction) giữa các lớp.
* VIPER là một mô hình mới để thiết kế các ứng dụng iOS, nó được giới thiệu vào năm 2014. VIPER dựa trên nguyên lý cốt lõi là “Single Responsibility”, trong đó mỗi class, layer chỉ đảm nhận duy nhất một nhiệm vụ, điều này giúp giải quyết vấn đề của Massive View Controller trong mô hình MVC cổ điển.

### Ứng Dụng Thiết Kế Dựa Trên Use Cases:

Các ứng dụng thường được hiện thực như là tập hợp các Use Case. Các Use Case cũng được biết đến như là các hành vi và mô tả những gì mà ứng dụng muốn làm. Một Use case là một tầng (layer) của ứng dụng chịu trách nhiệm cho business logic. Các use case nên độc lập với User Interface hiện thực chúng. Chúng cũng nên nhỏ và được định nghĩa rõ ràng. Việc quyết định làm cách nào để phân chia một ứng dụng phức tạp thành các use case nhỏ là một thách thức và cần phải luyện tập, nhưng nó là cách hữu ích để hạn chế tầm vực của mỗi vấn đề mà bạn phải giải quyết và mỗi class mà bạn viết ra.

Xây dựng một ứng dụng với mô hình VIPER liên quan đến việc hiện thực một tập các thành phần để hoàn thiện mỗi Use Case. Logic của ứng dụng là một phần trong việc hiện thực một Use Case, nhưng nó không phải là phần duy nhất. Use Case cũng tác động đến User Interface (UI). Thêm nữa, nó rất quan trọng để xem xét các Use Case phù hợp với nhau và với thành phần cốt lõi khác của ứng dụng, ví dụ như networking và data persistence, như thế nào. Các thành phần hoạt động như là các plugin của các Use Case, và VIPER là một cách mô tả

## Các Phần Chính Của Mô Hình VIPER

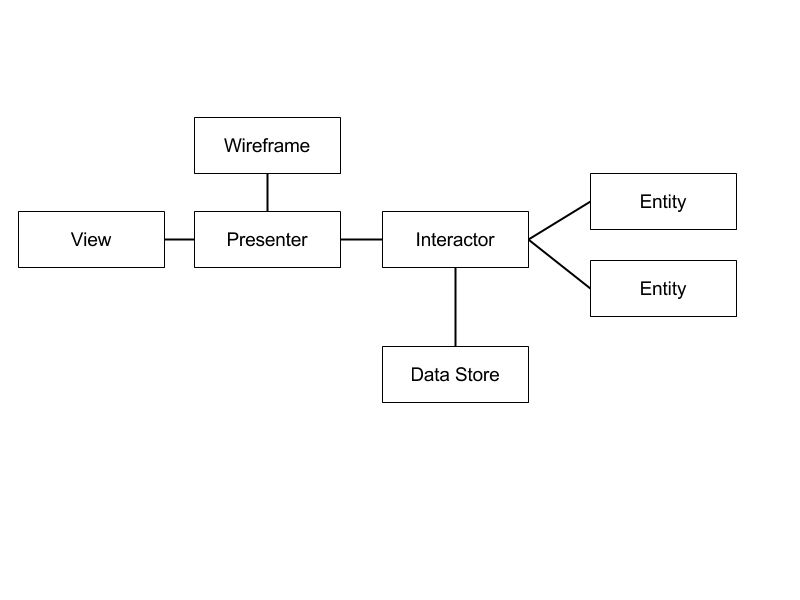


Các phần chính của mô hình VIPER bao gồm:

1. View: hiển thị những gì nó được Presenter thông báo tới và gởi lại các sự kiện tương tác của user đến Presenter.
2. Interactor: bao gồm business logic mà được xác định bởi 1 Use Case.
3. Presenter: bao gồm view logic để hiển thị các dữ liệu nhận được từ Interactor và xử lý user input (bằng cách yêu cầu dữ liệu mới từ Interactor).
4. Entity: bao gồm các model object cơ bản được sử dụng bởi Interactor.
5. Routing: bao gồm logic chuyển đổi giữa các màn hình với nhau theo thứ tự thiết kế trước.

Sự phân chia này cũng tuân theo [Single Responsibility Principle](http://www.objectmentor.com/resources/articles/srp.pdf). **Interactor** chịu trách nhiệm phân tích business, **Presenter** chịu trách nhiệm thiết kế các tương tác, và **View** có trách nhiệm thiết kế hình ảnh.

Dưới đây là sơ dồ của các thành phần trong VIPER và sự kết nối giữa chúng:



ác thành phần trong VIPER có thể được hiện thực trong một ứng theo bất kỳ thứ tự nào, chúng tôi chọn giới thiệu các thành phần này theo thứ tự mà chúng tôi khuyến khích bạn hiện thực chúng. Bạn sẽ nhận thấy rằng thứ tự này là tương đối phù hợp với quá trình xây dựng một ứng dụng đầy đủ, bắt đầu bằng việc thảo luận những gì sản phẩm cần làm, theo sau bởi người dùng sẽ tương tác với nó như thế nào.

### 1. Interactor

Một Interactor đại diện cho một Use Case duy nhất trong ứng dụng. Nó bao gồm các Business Logic điều khiển, sử dụng các đối tượng Model (Entities) để thực hiện một thao tác cụ thể. Công việc được thực hiện trong Interactor nên độc lập với các đối tượng UI.

### 2. Entity

Các Entity là các đối tượng Model được thao tác bởi một Interactor. Các Entity chỉ được thao tác bởi Interactor. Interactor không bao giờ truyền các Entity đến lớp Presenter.

### 3. Presenter

Presenter bao gồm các logic để điều khiển UI. Nó biết khi nào để hiển thị UI. Nó thu thập thông tin từ các tương tác của người dùng nên nó có thể cập nhập UI và gửi các yêu cầu đến một Interactor.

Presenter cũng nhận kết quả từ Interactor và chuyển đổi dữ liệu thành một khuôn mẫu để hiển thị hiệu quả nhất trên View.

Entity không bao giờ được truyền từ Interactor đến Presenter. Thay vào đó, các cấu trúc dữ liệu đơn giản, không có bất kỳ hành vi nào sẽ được truyền từ Interactor đến Presenter. Điều này ngăn không cho bất kỳ “Công việc thật sự” nào được thực hiện trong Presenter. Presenter chỉ có thể chuẩn bị dữ liệu để hiển thị trên View.

### 4. View

View luôn bị động. Nó chờ Presenter cho nó dữ liệu để hiện thị, nó không bao giờ yêu cầu dữ liệu từ Presenter. Các phương thức được định nghĩa trong View nên chấp nhận một Presenter có thể giao tiếp ở mức trừa tượng cao nhất, thể hiện trong nội dung của nó, chứ không phải là nội dung đó được hiển thị như thế nào. Presenter không biết về sự tồn tại của các đối tượng View như Label, Button,… Presenter chỉ biết về nội dung mà nó duy trì và khi nào thì nó nên được hiển thị. Còn việc nội dung hiển thị như thế nào phụ thuộc hoàn toàn View.

View là một lớp trừu tượng (abstract interface). UIViewController hay lớp con của nó sẽ hiện thực View interface.

Các View và ViewController cũng xử lý các tương tác và input từ người dùng. Thật dễ dàng để hiểu tại sao các ViewController thường xuyên trở nên quá lớn, do chúng là nơi dễ dàng nhất để đặt code xử lý các input để thực hiện một vài hành động nào đó. Để giữ cho các ViewController nhỏ gọn chúng ta cần phải cung cấp cho chúng một cách để thông báo cho các bên quan tâm (Presenter) khi người dùng thực hiện các hành động xác định. ViewController không nên quyết định dựa trên những hành động này, nhưng nó nên truyền đi những hành động này cùng với một vài thứ khác mà có thể giúp Presenter đưa ra những quyết định dựa trên các hành động này.

### 5. Routing

Chuyển đổi từ một màn hình đến màn hình khác như đã được thiết kế trong các wireframe của các designer. Trong VIPER, trách nhiệm của Routing được chia sẻ giữa 2 object: Presenter và Wireframe (Router). Wireframe có trách nhiệm tạo ra một View/ViewController và cài đặt chúng vào trong window.

Vì Presenter chứa đựng các logic để phản hồi input của người dùng nên nó biết khi nào thì chuyển đến màn hình khác và màn hình chuyển đến là màn hình nào. Trong khi đó, Wireframe biết làm cách nào để chuyển đến các màn hình khác. Do đó, Presenter sẽ sử dụng Wireframe để thực hiện việc chuyển đổi màn hình. Cùng nhau, chúng mô tả một luồng từ một màn hình đến màn hình kế tiếp.

Wireframe cũng là nơi hiển nhiên để xử lý các animation.

3.2. Cấu trúc dự án và cơ sở dữ liệu.

Realm database:

Ngày nay, việc phát triển ứng dụng di động đang trở thành xu hướng và phổ biến khắp mọi nơi. Trong mỗi ứng dụng thì phần quan trọng không kém chính là Cơ sở dữ liệu. CSDL phổ biến nhất được sử dụng hiện nay trên hầu hết các thiết bị là SQLite bởi vì nó khá quen thuộc với đại đa số các lập trình viên do sử dụng câu truy vấn SQL. Tuy nhiên, SQLite cũng có những mặt hạn chế nhất định như tốc độ truy vấn khá chậm khi mà dữ liệu phình to ra cũng như khi mà thực hiện phép JOIN. Trên cơ sở đó, Realm Mobible Database ra đời với mục đích cung cấp cho lập trình viên một lựa chọn có thể thay thế cho SQLite hiện nay nhưng vẫn đảm bảo mọi chức năng cần thiết của một CSDL thông thường:

* Realm Moblie Database ( gọi tắt là RMD) là một NoSQL ( Not Only SQL). Nó hướng tới việc xây dựng một ứng dụng theo hướng Offline database first. Điều này có nghĩa là ứng dụng vẫn có thể hoạt động dù cho không có kết nối mạng, dữ liệu sẽ được lưu trực tiếp trên thiết bị, người dùng vẫn có thể tiến hành mọi việc một cách thuận lợi.
* RMD lưu trữ dữ liệu dưới dạng Object và nó cũng cung cấp các hàm và phương thức để có thể truy vấn dữ liệu mà không cần thông qua câu truy vấn SQL.
* Phần core của RMD được viết bằng C++ và là mã nguồn mở, người dùng có thể tùy chỉnh lại theo ý muốn cá nhân.
* Cross-flatform và đã có phiên bản cho các ngôn ngữ sau: Swift, Java, Objective – C, Xamarin, React Native.
* Cung cấp miễn phí.

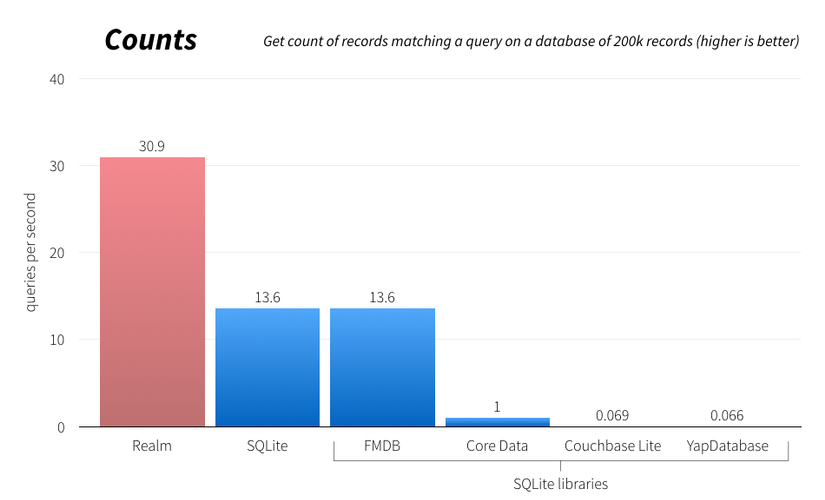
## **Điểm mạnh của Realm Mobile Database**

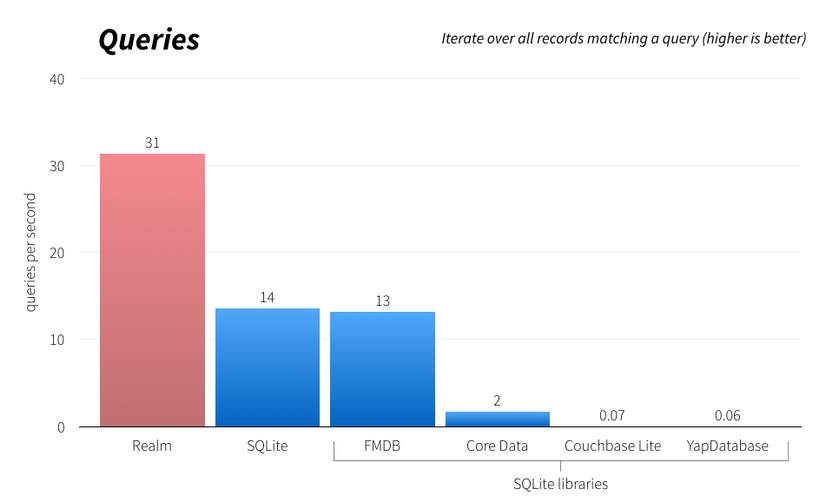
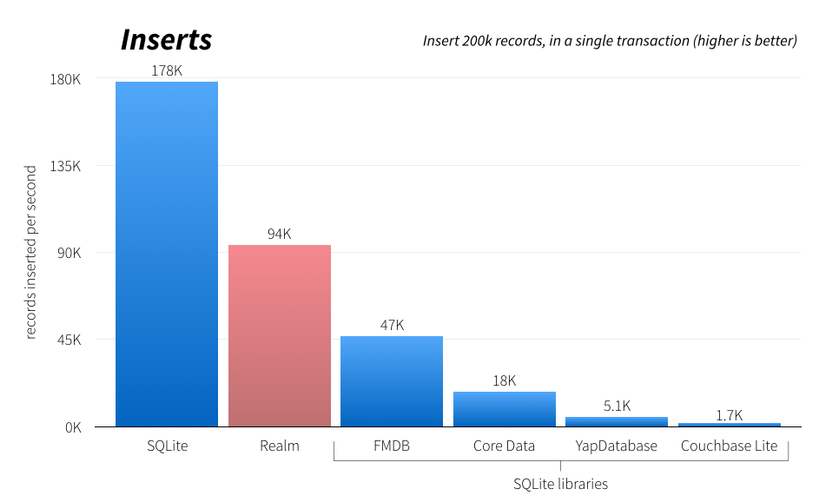
1. Dễ cài đặt và sử dụng

* RMD khá dễ cài đặt và sử dụng. Đối với IOS, chúng ta có thể sử dụng thư viện quản lý CocoaPods để cài đặt và sử dụng.
* Để sử dụng RMD, chúng ta chỉ cần tạo một class như bình thường, và kế thừa từ class “Object” của RMD.

1. Tốc độ truy vấn nhanh.

* RMD được tối ưu hóa để sử dụng bộ nhớ một cách ít nhất nhưng hiệu suất vẫn vượt trội so với các CSDL khác.
* Không giống như một số các database khác, bạn có thể sử dụng trực tiếp bên trong ứng ựng IOS của ban (hoặc là với những ứng dụng Android) để lưu và truy vấn dữ liệu cục bộ trên thiết bị, cho phép bạn xây dựng các ứng dụng nhanh hơn. Chúng ta đã thấy nhiều thư viện cố gắng cung cấp các chức năng và tốc độ giống SQLite.Nhưng trái lại Realm nhanh hơn một số xử lý SQLite.





Hiệu xuất của Realm đến từ việc nhiều năm phát triển trên một thiết kế lõi C++ được chỉnh sử để phù hợp với nhu cầu của nhiều thiết bị, nhờ áp dụng bit-packing, caching, vectorization và zero-copy architecture để nhận được lời ích tuyệt vời về việc sử dụng bộ nhớ và tốc độ