Lập trình iOS 101

Huy Phạm

About mee

- 2011 freelancer.
- 2012 RoR developer at Zigexn Ventura.
- 2013 CTO at Lozi Singapore (gọi vốn thành công series A).
- Skill: Back-end, Front-end, Mobile, Analytics, Team leader…
- Tools: Golang, Nodejs, Ruby, RabbitMQ, Kafka, Storm, ElasticSearch, PostgresSQL, MongoDB, Redis…

Table of content

1. Why iOS 101

- Xây một ngôi nhà thì dễ nhưng lâu đài thì khó.

- Hiểu rõ được các công việc mình làm hằng ngày từ đó tăng hiệu quả làm việc.

- Rèn luyện khả năng tư duy, kỷ luật và tự build cho mình các công cụ, mã nguồn cần thiết.

- Không ngừng cải tiến cách làm việc và tiến xa hơn trong công việc.

- Sản phẩm là thứ để đánh giá năng lực bản thân của các nhà tuyển dụng vậy nên ta nên xây nó theo cách mà ta tự hào nhất.

- Giới hạn về slide và tránh dài dòng, đối với từng mục thường chỉ đưa ra một số ví dụ cụ thể mà thôi. Việc học là tự bản thân nên nếu thấy hứng thú các bạn hãy tự tìm hiểu thêm.

2. Language and tools

Objective-C hay Swift?

- Objective-C và Swift (đã stable) đều tốt như nhau nên hãy chọn ngôn ngữ mà bạn am hiểu nhất để bắt đầu.

- Phần lớn thời gian làm việc với iOS là làm việc với framework, nên dành thời gian tìm hiểu: UIKit, Foundation,…

- Sử dụng Swift vẫn có thể dùng chung Objective-C.

Còn ngôn ngữ nào khác?

- Ngoài Objecitive-C và Swift ra thì để code được native app cho iOS cũng có thể dùng C++, C# trên Mono framework của Xamarin và gần đây là Javascript trên react-native.

- Bởi vì ngôn ngữ cũng chỉ là công cụ thế nên hãy chọn công cụ mà mình cảm thấy thoải mái nhất.

- Ngoài ra ta cũng cần thêm các công cụ soát lỗi của Objective-C như OCLint hoặc của Swift như swiftlint.

Học thế nào?

- Việc học ngôn ngữ cũng quan trọng như học framework iOS.
- Trước khi bắt đầu ta nên tìm hiểu về ngôn ngữ trước: singleton, parallels, multi thread, code block, function pointer, property attributes,… Hầu hết chúng là các khái niệm của lập trình hướng đối tượng, bạn sẽ lợi thế hơn rất nhiều nếu nắm được các khái niệm về nguyên lý ngôn ngữ và design pattern.
- Việc hiểu sẽ giúp ta dễ dàng xây dựng cấu trúc project cũng như dễ dàng hiểu được cách thức hoạt động và debug các lỗi xảy ra.
- Học cách sử dụng Xcode bài bản…

3. Coding conventions

What and Why

- Là quy ước chung khi làm việc trong một team, hoặc một project. Tuỳ vào team mà có thể khác nhau về các quy ước này.

- Quy định về cú pháp code, cách đặt tên biến, tên hàm, padding trong code, comment code, chia files, chia thư mục,…

- Làm cho project sạch sẽ dễ đọc, không dư thừa. 80% thời gian là đọc code vậy nên ta sẽ vui vẻ hơn khi nó được tuân thủ tốt.

Examples

- Đặt tên biến theo camelCase, đặt theo mục đích sử dụng hay đặt theo tên class: sendButton, titleLabel,…

- Comment chức năng function theo chuẩn của java và comment trước các đoạn code work around hoặc hotfix, cấu trúc cây thư mục phân cấp.

4. Best practice

What and Why

- Là cách làm đúng nhất trong số những cách làm.
- Làm đúng sẽ giảm được bug, hiệu suất cao.
- Làm đúng sẽ tăng chất lượng project.
- Tuy nhiên không phải lúc nào cách đúng nhất cũng là cách tốt nhất. Cách làm tốt nhất còn là cách làm phù hợp với hoàn cảnh.

Code vs Interface Builder

- Quá trình tạo một đối tượng bằng IB: xml -> parse -> code -> object.

- Quá trình tạo một đối tượng: code -> object.

- Mỗi cách làm đều có lợi thế riêng. Tuy nhiên trong từng cách sẽ phù hợp hơn trong một số hoàn cảnh.

- Một số tiêu chí đưa ra để so sánh: tốc độ xây dựng, chi phí xây dựng, sử dụng lại, sự phụ thuộc, làm việc nhóm, tính kế thừa, opensource, tính linh hoạt…

asynchronous vs synchronous

- Hầu hết thời gian làm việc đều là các đoạn code đồng bộ, tuy nhiên đôi lúc ta cần các hàm xử lý các hàm bất đồng bộ.

- Các phương pháp gọi hàm: protocol, pointer, code block, key-observer,…

- Các vấn đề thường mắc phải: retain cycle, strong vs weak pointer,…

5. Design pattern

What and Why

- Nó là những bản mẫu thiết kế được cho là tốt nhất trong lập trình mà người dùng nên follow khi xây dựng một thành phần.

- Hiểu và nắm design pattern sẽ giúp tăng tốc độ phát triển ứng dụng, tính năng, hoàn thành tốt các mô hình test và tối ưu hoá được hiệu năng.

- Lập trình iOS sử dụng tư tưởng lập trình hướng đối tượng, design patter chỉ xoay quanh các vấn đề về khởi tạo, cấu trúc và hành vi của các đối tượng.

Examples

- Singleton dùng trong trường hợp chỉ muốn có 1 thực thể của class: networking, file utils, data base connector, config file,…

- Observer là pattern cho phép các đối tượng phản ứng lại khi có tương tác từ một đối tượng khác.

- Flyweight là pattern cho phép ta giảm số lượng các đối tượng được khởi tạo.

- MVC thì quá nổi tiếng rồi. Thế còn MVVM?

6. Project structure

Beautiful project

- Một project có code đẹp, design pattern đẹp nhưng sẽ hoàn hảo hơn nếu có cả một structure đẹp.

- Các thành phần chung: constants, configs, enviroment, common macros, resource, ultils, extensions, dependency manager, readme, tools…

- Sự sắp xếp các thành phần: cấu trúc thư mục, đặt tên file.

Examples

- Tham khảo code tại: https://github.com/huyphams/Maverick

7. Common components

What and Why

- Là các thành phần thường sử dụng trong project
- Việc xây dựng sẽ giúp giảm thời gian xây dựng lại các thành phần này trong các project khác nhau.
- Nếu thành phần được xây dựng stable thì giảm thiểu được bug.

Common componenets

- Networking: thư viện network
- Queue: thực thi các tasks
- File utils: giao tiếp với files
- Extension: các hàm mở rộng cho class
- Macro: các hàm inline, tiện ích hệ thống, logger.
- Constant, models, ViewKit,….

Thanks