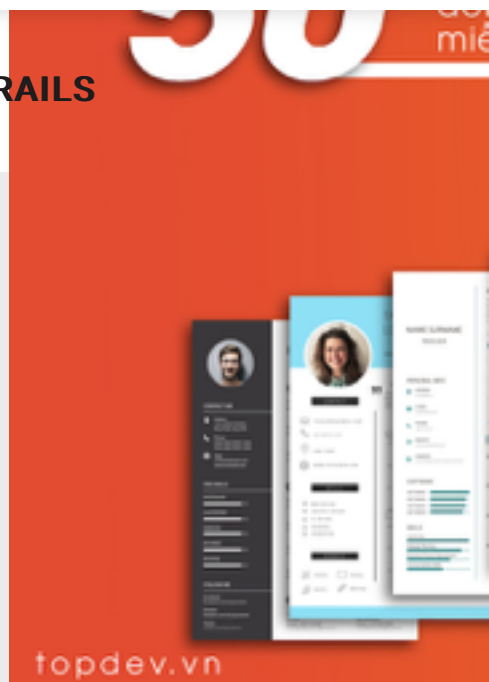


[JAVASCRIPT](#)[RUBY](#)[PHP](#)[IOS](#)[ANDROID](#)[SWIFT](#)[RAILS](#)[LARAVEL](#)[REACTJS](#)[RUBY ON RAILS](#)[RAILS](#)[LARAVEL](#)[REACTJS](#)[RUBY ON RAILS](#)[Home](#) >[iOS Concurrency - Phần 4:](#)[Operation và](#)[OperationQueue](#)**IOS**

iOS Concurrency - Phần 4: Operation và OperationQueue



NGÔ QUANG TUẤN ANH

[HTTPS://VIBLO.ASIA/P/IOS-CONCURRENCY-PHAN-4-
OPERATION-VA-OPERATIONQUEUE...](https://viblo.asia/p/ios-concurrency-phan-4-operation-va-operationqueue...)

Bài viết liên quan

- trình mô phỏng ios trên steroids: các thủ thuật trong xcode 9
- cảm nhận đầu tiên của bạn về swift với playgrounds
- tìm hiểu về kiểu result trong swift 5
- điểm nổi bật, hạn chế và sự thay đổi lớn nhất trong phát triển swift của ios
- chương 1: các công cụ phát triển, phương pháp học tập và ý tưởng app

50⁺
đợt
miễn



topdev.vn



📖 Đầu mục bài viết

iOS 9

By Tuananh Steven

Operation và OperationQueue

Giới thiệu chung

Chắc hẳn ai trong chúng ta cũng đã từng có trải nghiệm khi nhấn một button hay nhập một số đoạn text trong ứng dụng iOS hay Mac OS thì giao diện người dùng bị đứng, không còn tương tác (responsive) nữa. Trên Mac OS, chúng ta bắt gặp hình ảnh đồng hồ cát hay bánh xe đủ màu quay tròn cho tới khi người dùng có thể tương tác với UI lại. Trong ứng dụng iOS, giao diện người dùng bị đứng và không thể tương tác được. Những app như vậy chúng ta hay gọi là unresponsive app, người dùng sẽ cảm thấy bức mình khi thường xuyên gặp phải trường hợp này.



Sở dĩ chúng ta gặp phải trường hợp này là bởi vì chúng ta đã thao tác quá nhiều công việc nặng, đòi hỏi thời gian lâu trên **main thread**. Bản chất main thread là một serial queue do đó chúng ta sẽ phải đợi thực hiện xong công việc nặng này rồi mới tiến hành các công việc khác, điều này làm ứng dụng chúng ta không mượt và bị lag do thời gian đợi quá lâu.

Giải pháp mà những developer non tay mới vào nghề là chuyển hết các công việc nặng ấy sang background global queue và nhường main queue lại cho thành phần xử lý UI. Global queue sẽ chạy những task này đồng thời dưới background làm cho người dùng cảm thấy ứng dụng chúng ta responsive hơn. Mình đã trình bày giải pháp này ở bài **"iOS Concurrency**

- **Phần 3.2: Grand Central Dispatch**
phần **“Xử lý background tasks”**, bạn
có thể xem lại tại link này :

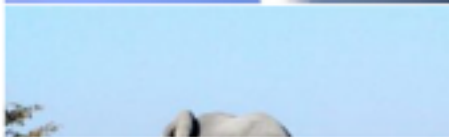
<https://viblo.asia/p/ios-concurrency-phan-32-grand-central-dispatch-YWOZrMyyKQ0>

Apple cung cấp cho chúng ta một api khác để hiện thực các tác vụ đồng thời đó là **Operation và OperationQueue**. Trong bài này chúng ta sẽ học cách sử dụng chúng thông qua một ví dụ mà tất hần ai cũng từng gặp phải. Thật sự ra ví dụ này được lấy từ một bài trên trang raywenderlich.com, đây là một bài rất hay tuy nhiên đã lâu rồi tác giả không cập nhật lại nó từ hồi xcode 6.3 và swift 1.2. Nay mình xin phép viết lại sang tiếng Việt, cập nhật xcode 8.3.3 và Swift 3.1. Mình hy vọng rằng với cách trình bày bình dân của mình cũng như ngôn ngữ là Tiếng Việt sẽ giúp các bạn nắm được cách sử dụng Operation và Operation Queue trong vấn đề này.

Vấn đề gặp phải

Mục tiêu của project này là hiển thị một table view của những bức hình. Những bức hình này được download

từ internet về và được apply filter rồi được hiển thị lên tableView. Filter image ở đây mình sử dụng thư viện Core Image để làm cho bức hình có một màn màu nâu nhạt phủ bên ngoài làm cho nó như một bức hình cổ điển.



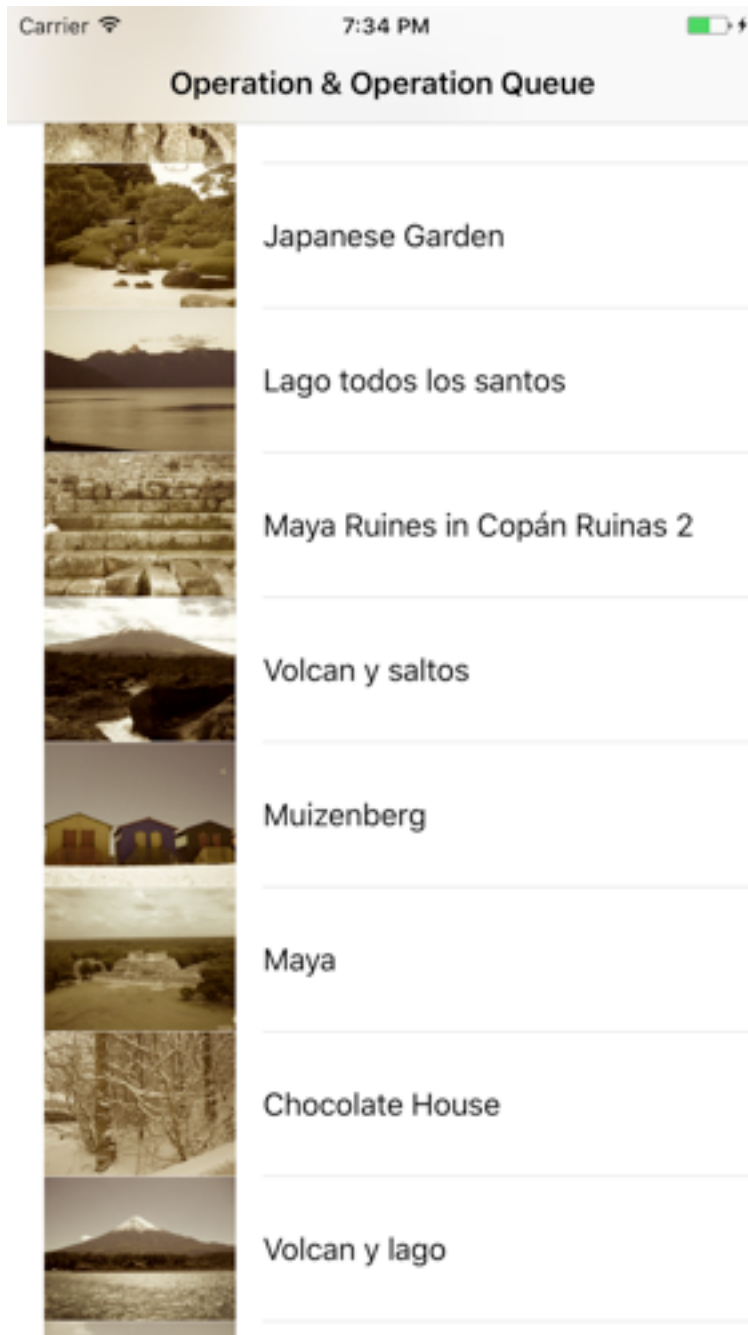
Hình ảnh trước và sau khi filter

Các bạn download starter project này để cùng làm với mình:

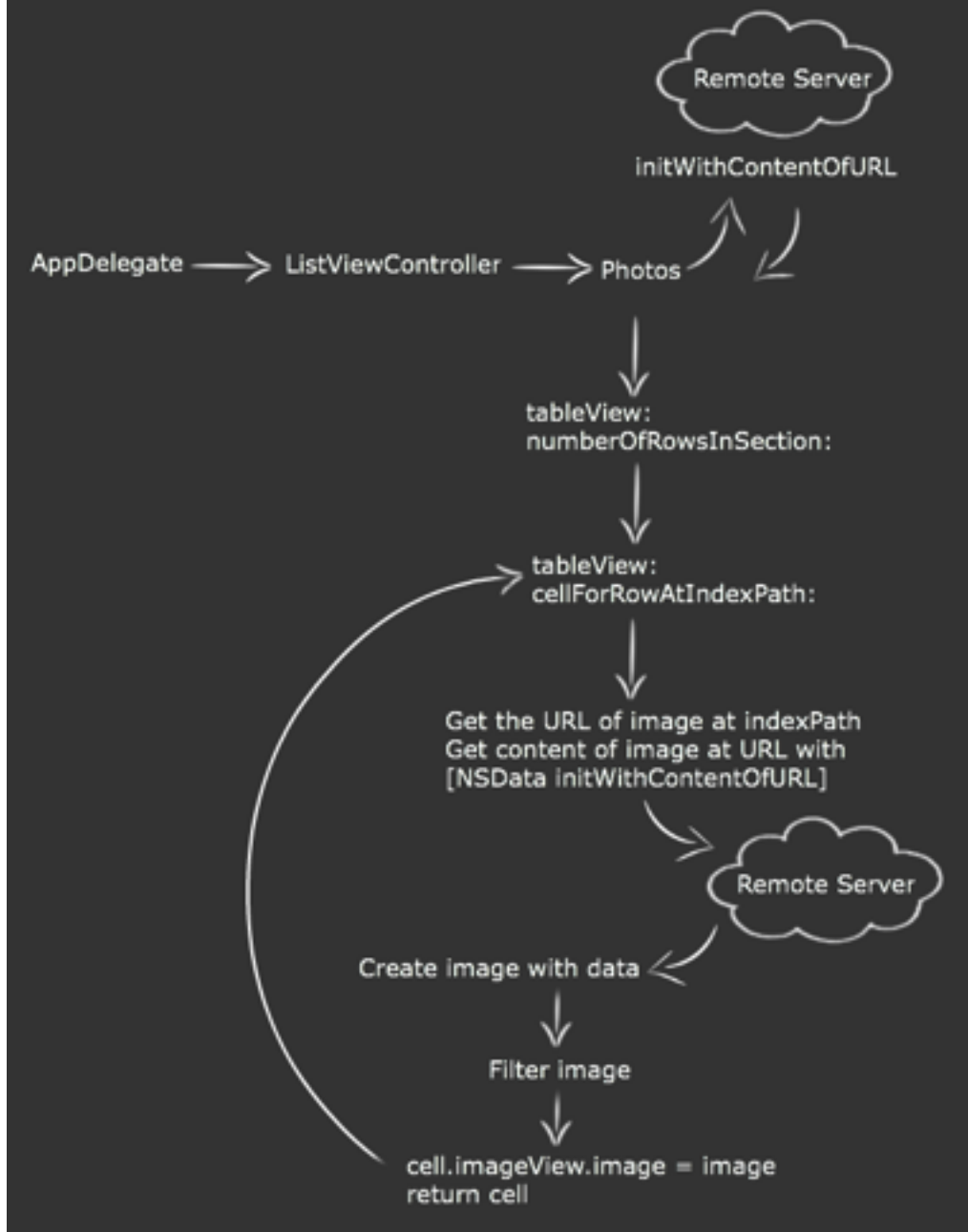
<https://github.com/TuananhSteven/ClassicPhotos->

Starter

Build và run project chúng ta sẽ thấy app hiển thị danh sách những bức hình. Khi trượt (scroll) tableView, nó lắc ko thể tả nổi.



Dưới đây là mô hình hoạt động của app.



Đầu tiên từ AppDelegate chúng ta sẽ chạy vào ListViewController, tại đây chúng ta load danh sách url của những bức hình (data source) từ trên mạng về và thả vào Dictionary Photos (1).

Tiếp theo chúng ta vào hàm `numberOfRowsInSection` để đặc tả số lượng hàng của tableView.

Ở hàm `cellForRowAtIndexPath` chúng ta thực hiện 2 tác vụ nặng đó là:



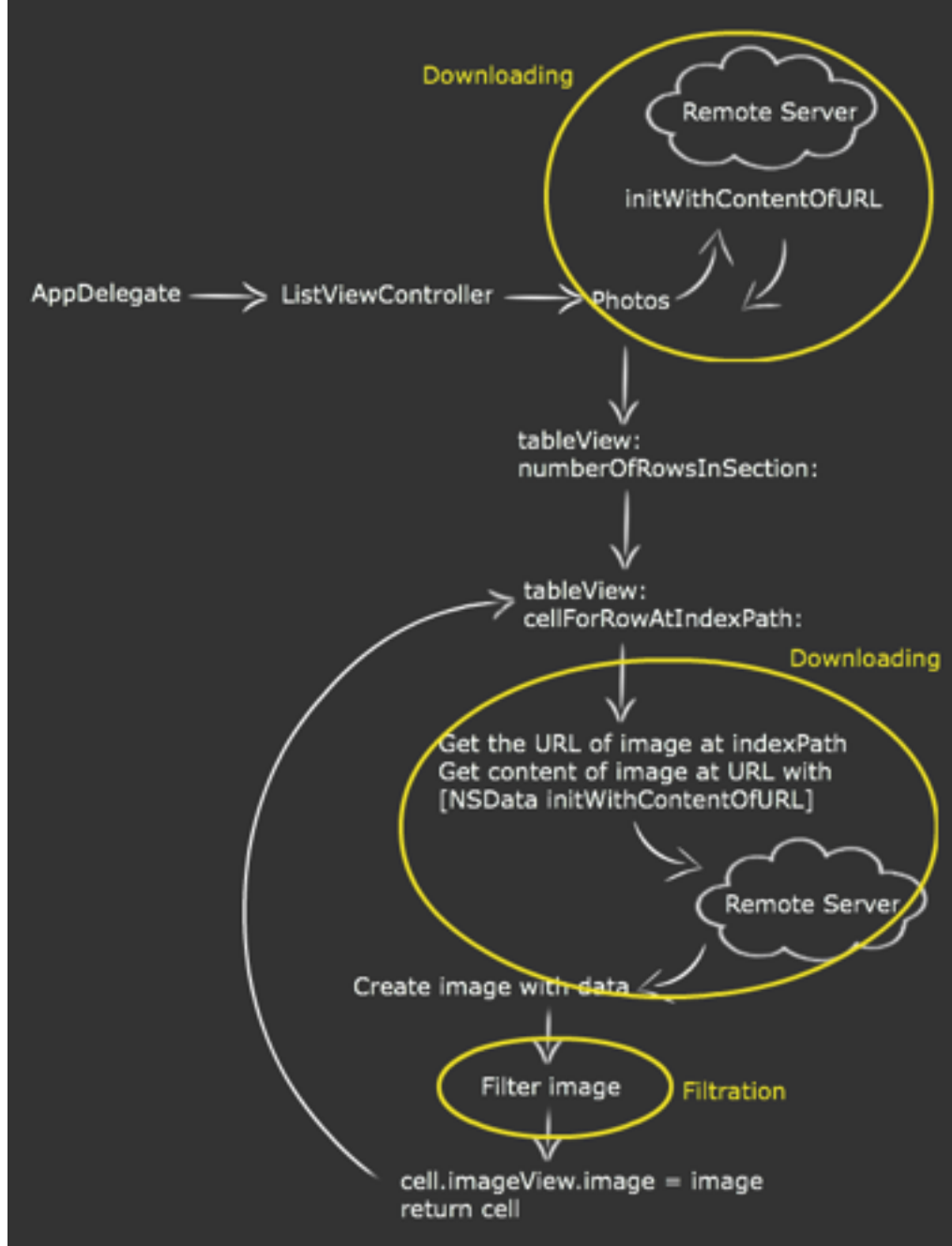
- Download image data bằng URL của image ở `indexPath` hiện tại (2)
- Filter image sử dụng thư viện Core Image (3)

Sau đó chúng ta gán hình ảnh vào trong imageView để hiển thị.

Tất cả 3 tác vụ nặng (1), (2), (3) đều diễn ra trên main thread của ứng dụng. Bởi vì nhiệm vụ chính của main thread là tương tác người dùng nên việc chúng ta làm nó chạy 3 tác vụ này đã vô tình giết đi tính responsive của nó.

Giải pháp

Vấn đề trên sử dụng mô hình non-threaded, tất cả các tác vụ đều chạy ở main thread làm cho ứng dụng bị lag. Cùng xem xét lại vấn đề này, có 3 khu vực mà chúng ta cần nâng cấp về luồng được khoanh tròn màu vàng ở hình dưới đây. Bằng việc tách rời 3 khu vực này và đặt chúng trong những thread riêng, main thread sẽ được giải phóng và ứng dụng trở nên tương tác hơn với người dùng.



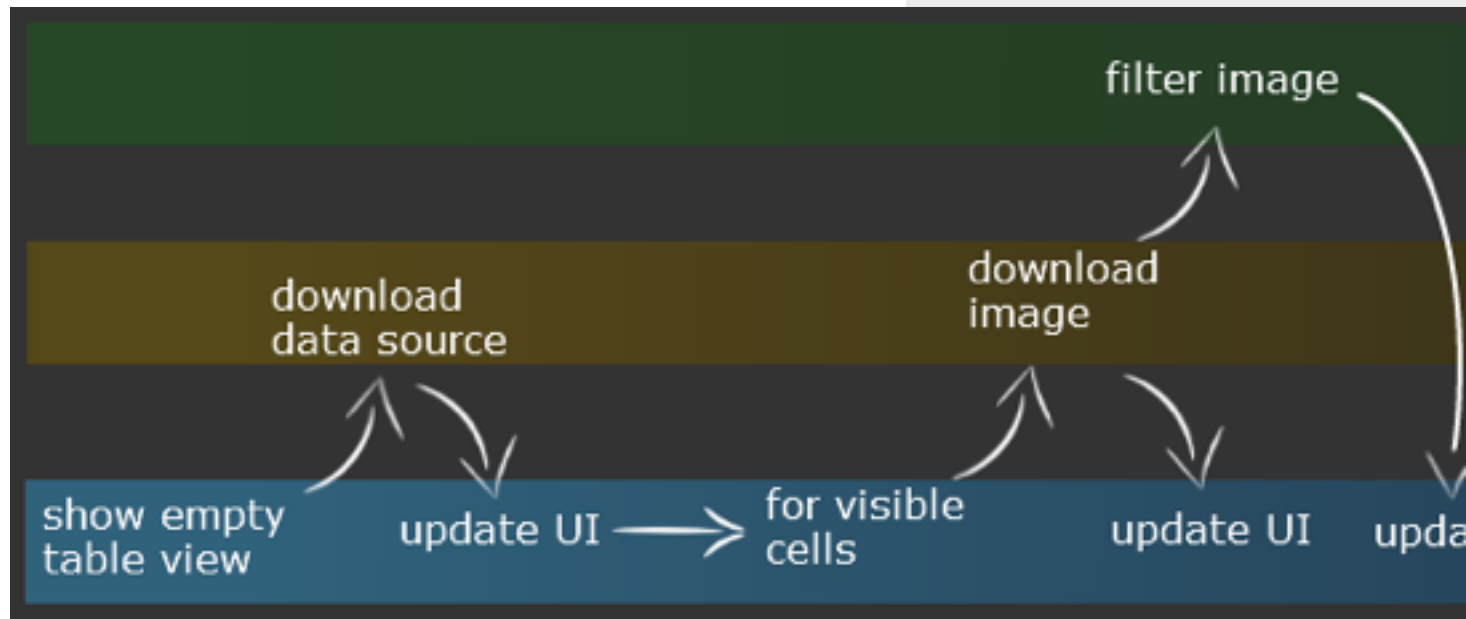
Như vậy ứng dụng của chúng ta sẽ cần 1 thread chịu trách nhiệm tương tác người dùng (main thread), một thread dành riêng cho việc download datasource và hình ảnh, và một thread cho việc filter hình ảnh. Trong mô hình mới, ứng dụng sẽ bắt đầu trên main thread và load một table view trống. Cùng thời điểm đó,

ứng dụng sẽ cho chạy một thread thứ 2 để download datasource. Một khi datasource đã được download xong, bạn sẽ reload lại table view. Dĩ nhiên việc reload này sẽ được thực hiện trên main thread vì nó liên quan tới tương tác người dùng. Ở thời điểm này, table view biết có bao nhiêu hàng và url của những image cần hiển thị nhưng nó vẫn chưa thực sự là hình ảnh vì chúng ta chưa down chúng. Nếu bạn tiến hành download ngay lập tức tất cả hình ảnh vào lúc này, nó sẽ không hiệu quả bởi vì chưa chắc gì người dùng coi hết tất cả hình đó.

Giải pháp ở đây là chúng ta chỉ download những hình ảnh ở những hàng mà người dùng mong muốn thấy. Bên cạnh đó việc filter hình ảnh không thể bắt đầu khi hình ảnh chưa được download xong. Do đó chúng ta chỉ bắt đầu tiến trình filter hình ảnh khi hình ảnh đã được download hoàn thành.

Để làm cho app trở nên tương tác hơn, chúng ta sẽ hiển thị hình ảnh ngay khi được download xong. Cùng lúc đó chúng ta sẽ kích hoạt tính năng filter hình ảnh và update lại UI khi hình

ảnh được filter xong. Hình vẽ dưới đây sẽ cho ta thấy những luồng cần xử lý.



Hiện thực

Để đạt được những mục tiêu trên, bạn sẽ cần phải theo dõi để biết tấm hình hiện tại có đang được download, đã download xong hay đã được apply filter. Bạn cũng cần theo dõi trạng thái của mỗi hoạt động (operation) bao gồm download operation và filter operation để mà có thể cancel, dừng (pause) hay chạy lại (resume) khi người dùng scroll tableView.

Chúng ta tạo một swift file tên là PhotoOperations.swift và thêm đoạn code dưới đây:

```
import UIKit
```

```
// This enum contains all the possible states
enum PhotoRecordState {
    case New, Downloaded, Filtered, Failed
}

class PhotoRecord {
    let name:String
    let url:NSURL
    var state = PhotoRecordState.New
    var image = UIImage(named: "Placeholder")

    init(name:String, url:NSURL) {
        self.name = name
        self.url = url
    }
}
```

Class **PhotoRecord** là một lớp mà nó trình bày thông tin của hình ảnh bao gồm tên, url, trạng thái của bức hình và hình ảnh của nó. Những trạng thái có thể của bức hình được biểu diễn trong enum **PhotoRecordState**. Mặc định trạng thái bức hình là **.New** và hình ảnh được gán tới tấm hình có tên là “Placeholder”.

Để theo dõi trạng thái của mỗi hoạt

động (Operation) diễn ra, chúng ta cần một class riêng. Chúng ta thêm đoạn code dưới đây vào trong file PhotoOperations.swift mới này.

```
class PendingOperations {  
    lazy var downloadsInProgress = [NSIndex  
    lazy var downloadQueue: OperationQueue  
        var queue = OperationQueue()  
        queue.name = "Download queue"  
        queue.maxConcurrentOperationCount  
        return queue  
    }()  
  
    lazy var filtrationsInProgress = [NSIndexP  
    lazy var filtrationQueue: OperationQueue =  
        var queue = OperationQueue()  
        queue.name = "Image Filtration queue"  
        queue.maxConcurrentOperationCount  
        return queue  
    }()  
}
```

Class **PendingOperations** bao gồm 2 dictionary để nắm giữ những hoạt động (Operation) bao gồm download và filter cho mỗi hàng trong tableView và 2 OperationQueue cho mỗi loại Operation.

Tất cả các giá trị được tạo lazy, điều này có nghĩa là chúng sẽ không được khởi tạo cho tới khi lần đầu chúng được truy cập, điều này làm nâng cao performance của ứng dụng.

Việc tạo OperationQueue cho việc download và filter rất rõ ràng. Chúng ta khởi tạo, đặt tên cho queue và set số lượng Concurrent Operation tối đa. Ở đây chúng ta set chúng bằng 1 để mỗi operation kết thúc lần lượt.

Tiếp theo chúng ta tiến hành tạo những Operation cho hoạt động download và filter. Cùng thêm đoạn code sau vào cuối file

PhotoOperations.swift:

```
class ImageDownloader: Operation {  
    //1  
    let photoRecord: PhotoRecord  
  
    //2  
    init(photoRecord: PhotoRecord) {  
        self.photoRecord = photoRecord  
    }  
  
    //3  
    override func main() {  
        //4
```

```

    if self.isCancelled {
        return
    }
    //5
    let imageData = NSData(contentsOf:se

    //6
    if self.isCancelled {
        return
    }

    //7
    if (imageData?.length)! > 0 {
        self.photoRecord.image = UIImage(d
        self.photoRecord.state = .Downloade
    }
    else
    {
        self.photoRecord.state = .Failed
        self.photoRecord.image = UIImage(r
    }
}
}
}

```

Chúng ta tạo lớp **ImageDownloader** thừa kế từ lớp trừu tượng (abstract class) **Operation**.

Mình sẽ giải thích những gì diễn ra ở

những dòng comment phía trên:

//1: Chúng ta thêm một thuộc tính hằng của lớp** PhotoRecord** để lưu thông tin hình cần download

//2: Tại đây chúng ta tạo constructor có tên là init để khởi tạo giá trị ban đầu cho thuộc tính trên

//3: Class **ImageDownloader** thừa kế từ lớp trừu tượng **Operation** nên chúng ta phải override lại hàm main để chỉ rõ những việc cần làm trong lớp này.

//4: Kiểm tra cancel trước khi bắt đầu. Những Operations nên được kiểm tra thường xuyên nếu chúng đã bị cancel trước khi bắt đầu công việc.

//5: Chúng ta tiến hành download data bức hình.

//6: Kiểm tra cancel một lần nữa

//7: Chúng ta kiểm tra thử dữ liệu có được down thành công không. Nếu có dữ liệu chúng ta tạo một UIImage Object và thêm nó vào thuộc tính image của PhotoRecord đồng thời thay đổi trạng thái cho nó. Ngược lại, chúng ta đánh dấu .Failed và set tấm hình thất bại tương ứng.

Tiếp theo chúng ta tạo một lớp

Operation khác để quản lý việc filter hình ảnh. Thêm đoạn code dưới đây vào file **PhotoOperations.swift**:

```
class ImageFiltration: Operation {
    let photoRecord: PhotoRecord

    init(photoRecord: PhotoRecord) {
        self.photoRecord = photoRecord
    }

    override func main () {
        if self.isCancelled {
            return
        }

        if self.photoRecord.state != .Downloaded {
            return
        }

        if let filteredImage = self.applySepiaFilter(photoRecord.image) {
            self.photoRecord.image = filteredImage
            self.photoRecord.state = .Filtered
        }
    }
}
```

Class** ImageFiltration** trên tương tự như class **ImageDownloader**, chỉ có

điều thay vì download thì chúng ta gọi một hàm khác để filter hình ảnh. Các bạn hãy thêm hàm sau vào trong lớp **ImageFiltration**:

```
func applySepiaFilter(image:UIImage) -> UIImage {
    let inputImage = UIImage(data:image.data)

    if self.isCancelled {
        return nil
    }

    let context = CIContext(options:nil)
    let filter = CIFilter(name:"CISepiaTone")
    filter?.setValue(inputImage, forKey: kCIInputImageKey)
    filter?.setValue(0.8, forKey: "inputIntensity")
    let outputImage = filter?.outputImage

    if self.isCancelled {
        return nil
    }

    let outImage = context.createCGImage(outputImage!, from: CGRectZero)
    let returnImage = UIImage(cgImage: outImage)
    return returnImage
}
```

Hàm filter trên được hiện thực tương tự như hàm cũ trong **ListViewController**. Nó được duy

chuyển ra đây để có thể chạy trong một Operation tách rời trong background. Một lần nữa bạn nên kiểm tra cancel thường xuyên. Một kinh nghiệm là nên kiểm tra cancel trước và sau khi thực hiện tác vụ nặng. Khi hình ảnh được filter xong, chúng ta set giá trị vào trong instance của PhotoRecord.

Ok, chúng ta vừa mới chuẩn bị xong những công cụ và nền tảng cần thiết để xử lý Operation như thể là background task. Bây giờ chúng ta trở lại **ListViewController.swift** và chỉnh sửa lại như sau:

Chúng ta xóa thuộc tính lazy var photos và thay bằng dòng dưới đây:

```
var photos = [PhotoRecord]()  
let pendingOperations = PendingOperations
```

Chúng ta khai báo một mảng PhotoRecord để lưu thông tin về những bức hình và khởi tạo đối tượng của lớp PendingOperation cho việc quản lý những operations.

Tiếp theo, chúng ta thêm mới một method cho việc download danh sách

thông tin của hình.

```
func fetchPhotoDetails() {
    let request = URLRequest(url:dataSource.url)
    guard UIApplication.shared.isNetworkActivityIndicatorVisible == false else {
        return
    }
    let connection = NSMutableURLRequest(request: request, cachePolicy: .useProtocolCachePolicy)
    connection.timeoutInterval = 10
    let dataTask = URLSession.shared.dataTask(with: connection) { data, response, error in
        if data != nil {
            let datasourceDictionary = try! PhotoRecordParser.parse(data!)
            for (key,value) in datasourceDictionary {
                let name = String(describing: value)
                print("key : \(name) , value: \(value)")
                let url = NSURL(string:value.value!)
                if name != nil && url != nil {
                    let photoRecord = PhotoRecord(name: name, url: url)
                    self.photos.append(photoRecord)
                }
            }
            self.tableView.reloadData()
        }
        if error != nil {
            let alert = UIAlertView(title:"Oops!", message:error.localizedDescription, cancelButtonTitle:"OK")
            alert.show()
        }
    }
    UIApplication.shared.isNetworkActivityIndicatorVisible = true
}
```



```
}
```

Method trên tạo một web request async mà khi kết thúc sẽ chạy completion block trên main queue. Khi việc download được hoàn tất, danh sách thuộc tính được trích xuất vào trong NSDictionary và sau đó được xử lý lần nữa vào trong một mảng của những object PhotoRecord. Bạn vẫn chưa sử dụng NSOperation trực tiếp ở đây, nhưng thay vì bạn đã truy cập main queue sử dụng NSOperationQueue.mainQueue(). Thật sự ra method trên của raywenderlich hơi cũ kĩ bởi vì NSURLConnection và UIAlertView đã bị deprecated trong iOS9 và chúng đã được Apple thay bằng NSURLSession và UIAlertController. Có thể bạn sẽ lúng túng khi đọc method trên nhưng đừng lo, nó chỉ là đoạn code để lấy danh sách thông tin của hình và url của chúng. Bởi vì raywenderlich cung cấp cho chúng ta một file thông tin .plist trên server để chúng ta đọc dữ liệu. Bản chất của file này bao gồm một danh sách những

key và value. Trên những dự án thực tế chúng ta sẽ làm việc với dữ liệu dạng json hay xml. Một trong những thư viện Networking nổi tiếng trong iOS bạn có thể sử dụng để thay thế đoạn code trên có thể kể đến là AFNetworking và Alamofire. Tuy nhiên project này mình chỉ muốn demo cách sử dụng Operation và OperationQueue khi làm việc với tableView, do đó mình tạm thời sai lại đoạn code cũ trên. Nếu bạn không hiểu nó, bạn có thể bỏ qua để đọc phần tiếp theo.

Tiếp theo chúng ta gọi method mới ở trong viewDidLoad:

```
fetchPhotoDetails()
```

Tiếp theo, ở hàm tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) chúng ta thay thế tất cả bằng đoạn code dưới đây:

```
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)  
  
    //1  
    if cell.accessoryView == nil {
```

```

        let indicator = UIActivityIndicatorView()
        cell.accessoryView = indicator
    }

    let indicator = cell.accessoryView as! UIActivityIndicatorView

    //2
    let photoDetails = photos[indexPath.row]

    //3
    cell.textLabel?.text = photoDetails.name
    cell.imageView?.image = photoDetails.image

    //4
    switch (photoDetails.state){
    case .Filtered:
        indicator.stopAnimating()
    case .Failed:
        indicator.stopAnimating()
        cell.textLabel?.text = "Failed to load"
    case .New, .Downloaded:
        indicator.startAnimating()
        self.startOperationsForPhotoRecord(photoDetails)
    }

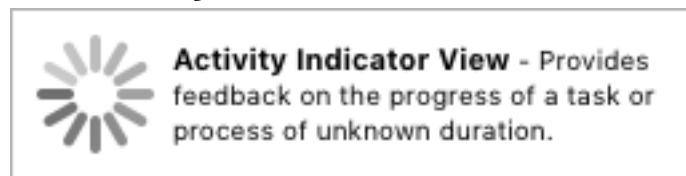
    return cell
}

```

Sau đây mình sẽ giải thích những

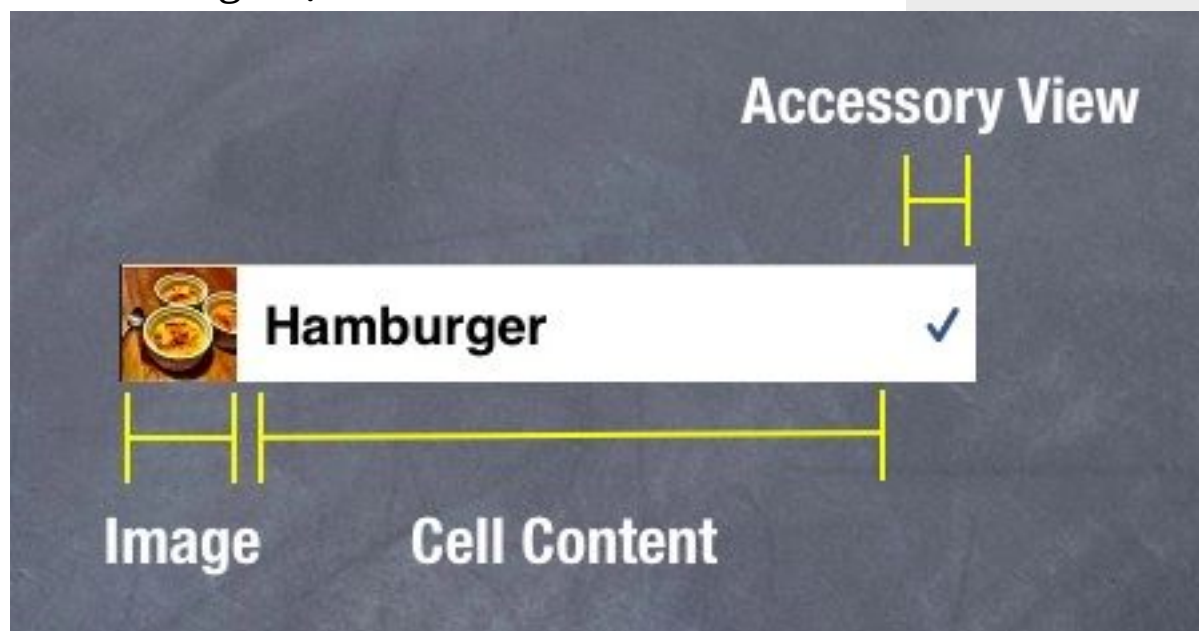
đồng comment phía trên:

1. Để cung cấp cho người dùng biết được hình ảnh đang được download hay filter, chúng tạo một `UIActivityIndicatorViewView` và set nó vào accessory view của table view cell.



Hình bên trên chính là

`UIActivityIndicatorView` thường được dùng để cho người dùng biết là họ cần phải đợi để ứng dụng thực hiện một task nào đó. Hình phía dưới cho chúng ta biết được vị trí của accessory view trong một table view cell.



Ứng dụng của chúng ta sẽ hiển thị `UIActivityIndicatorView` như thế này. Hình dưới cho ta thấy rằng ứng dụng đang filter hình ảnh.



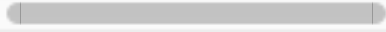
2. Lấy thông tin hình ảnh ở dòng hiện tại từ data source.

3. Set tên và hình ảnh vào trong cell hiện tại

4. Kiểm tra trạng thái hình ảnh, nếu nó đã được filter hoặc failed, chúng ta tiến hành dừng animation quay tròn tròn của Activity Indicator View. Nếu nó là hình mới hoặc chỉ mới được download và chưa được filter, chúng ta cho Activity Indicator View quay tròn tròn và gọi hàm startOperationsForPhotoRecord. Bạn có thể xóa hiện thực applySepiaFilter bởi vì chúng ta không còn sử dụng nữa. Thêm mới method dưới đây để start những operation:

```
func startOperationsForPhotoRecord(photoDetails: PhotoDetails) {  
    switch (photoDetails.state) {  
        case .New:  
            startDownloadForRecord(photoDetails)  
        case .Downloaded:  
            startFiltrationForRecord(photoDetails)  
        default:  
            NSLog("do nothing")  
    }  
}
```

```
}  
}
```



Tại đây chúng ta truyền vào instance của PhotoRecord cùng với indexpath của chúng. Dựa vào trạng thái của photo record mà bạn sẽ kích hoạt bước download hay filter.

Method cho việc download và filter hình ảnh được hiện thực tách rời như thế có 1 khả năng rằng trong khi một hình ảnh đang được download, người dùng có thể scroll đi và bạn không cần apply image filter. Vì thế thời gian kế tiếp, nếu người dùng quay trở lại dòng đó, bạn không cần download lại hình ảnh, bạn chỉ cần apply image filter.

Rất hiệu quả phải không nào.

Bây giờ bạn cần hiện thực những phương thức mà bạn gọi trong method trên. Nhớ rằng bạn đã tạo một custom class là PendingOperations để giữ những Operation. Giờ đây bạn sẽ sử dụng nó. Thêm method dưới đây vào tới class:

```
func startDownloadForRecord(photoDetails  
    //1  
    if pendingOperations.downloadsInProgress
```

```

        return
    }

    //2
    let downloader = ImageDownloader(photoURL)

    //3
    downloader.completionBlock = {
        if downloader.isCancelled {
            return
        }
        DispatchQueue.main.async(execute: {
            self.pendingOperations.downloadImage(photoURL)
            self.tableView.reloadRows(at: [indexPath],
                                        withRowAnimation: .Fade))
        })
    }

    //4
    pendingOperations.downloadsInProgress += 1

    //5
    pendingOperations.downloadQueue.add(downloader)
}

func startFiltrationForRecord(photoDetail: PhotoDetail) {
    if pendingOperations.filtrationsInProgress == 0 {
        return
    }

    let filterer = ImageFiltration(photoRecord: photoDetail)
    filterer.completionBlock = {
        if filterer.isCancelled {
            return
        }
        DispatchQueue.main.async(execute: {
            self.pendingOperations.filtrationsInProgress -= 1
            self.tableView.reloadRows(at: [indexPath],
                                        withRowAnimation: .Fade))
        })
    }
    pendingOperations.filtrationsInProgress += 1
    pendingOperations.filtrationQueue.add(filterer)
}

```



```

        return
    }
    DispatchQueue.main.async(execute: {
        self.pendingOperations.filtrationsInProgress -= 1
        self.tableView.reloadRows(at: [indexPath], with: .fade)
    })
}
pendingOperations.filtrationsInProgress -= 1
pendingOperations.filtrationQueue.add(operation)
}

```

Mình sẽ giải thích những gì được comment phía trên:

1: Đầu tiên kiểm tra cho indexPath hiện tại xem không biết là đã hoàn toàn có một operation trong downloadsInProgress cho nó chưa.

Nếu có rồi, chúng ta return ngay lập tức.

2: Nếu không có, tạo một instance của ImageDownloader bằng việc sử dụng constructor. Tại đây, hàm main trong class ImageDownloader sẽ được kích hoạt để tiến hành download hình ảnh.

3: Thêm một completion block để chạy một vài đoạn code khi operation download được hoàn thành.

Completion block là một nơi tốt để

phần còn lại của ứng dụng của bạn biết rằng một operation đã kết thúc. Thật là quan trọng để ghi chú rằng completion block sẽ được chạy thậm chí nếu operation được gọi cancelled, do đó bạn phải kiểm tra thuộc tính này trước khi làm bất cứ thứ gì. Bạn không có gì đảm bảo rằng những luồng nào mà completion block được gọi, vì thế bạn cần sử dụng GCD để trigger việc reload tableView trên main thread.

4: Thêm operation tới
downloadsInProgress

5: Thêm operation tới download queue. Đây là cách bạn thật sự lấy những operation để bắt đầu chạy. Queue sẽ take care việc lập lịch cho bạn khi bạn đã thêm operation.

Phương thức để filter hình ảnh theo sau mẫu phía trên, ngoại trừ nó sử dụng ImageFilteration và filterarionsInProgress để giữ operations.

Build và Run ứng dụng để xem những gì được thay đổi. Khi bạn scroll tableView, app không bị lag nữa.



Classic Photos



Cheetah



Santorini



Mozambique 2



Big Fella



Muizenberg



Cải tiến

Ứng dụng của chúng ta đã trở nên responsive hơn so với bản gốc ban đầu. Tuy nhiên vẫn còn một số chi tiết nho nhỏ mà bạn đã bỏ quên.

Bạn chú ý rằng khi bạn scroll table view, những cell đi qua vẫn còn tiếp tục diễn ra tiến trình download và

filter. Nếu bạn scroll table view nhanh chóng, app của chúng ta sẽ trở nên bận rộn trong việc download và filter hình ảnh từ những cell mà bạn đã scroll qua nhưng bạn không cần thiết thấy nó. Một ý tưởng hay là ứng dụng nên cancel filter những cell đã bị lướt qua đó và ưu tiên cho cell hiện tại mà nó cần được hiển thị với người dùng. Quay trở lại với

ListViewController.swift, trong phần hiện thực của tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath), bạn đóng gói việc gọi startOperationsForPhotoRecord trong mệnh đề if như sau:

```
if (!tableView.isDragging && !tableView.isDecelerating) {  
    self.startOperationsForPhotoRecord(photoRecord)   
}
```

Ở đoạn code trên, bạn nói với table view start những operations chỉ khi table view không scroll. Những thuộc tính isDragging và isDecelerating thật sự là của UIScrollView và bởi vì UITableView là lớp con của UIScrollView nên bạn tự động thừa hưởng những thuộc tính này.

Để biết được table view có đang scroll hay không, chúng ta cần hiện thực protocol UIScrollViewDelegate. Khái niệm Protocol trong Swift tương tự như interface ở trong ngôn ngữ lập trình Java. Protocol này bao gồm những hàm chưa được hiện thực giúp chúng ta trigger được khi nào table view scroll và dừng. Cùng thêm đoạn code sau vào cuối file ListViewController.swift:

```
extension ListViewController: UIScrollViewDelegate {
    func scrollViewWillBeginDragging(_ scrollView: UIScrollView) {
        // 1
        suspendAllOperations()
    }

    func scrollViewDidEndDragging(_ scrollView: UIScrollView,
                                   withVelocity velocity: CGPoint) {
        // 2
        if !decelerate {
            loadImagesForOnscreenCells()
            resumeAllOperations()
        }
    }

    func scrollViewDidEndDecelerating(_ scrollView: UIScrollView) {
        // 3
        loadImagesForOnscreenCells()
    }
}
```

```
resumeAllOperations()
```

```
}
```

```
}
```

Mình sẽ giải thích những đoạn comment trong code phía trên:

1. Hàm `scrollViewWillBeginDragging` sẽ được gọi khi table view bắt đầu scroll. Ngay lúc này, chúng ta sẽ tạm dừng tất cả những operations và chỉ tập trung vào những gì người dùng chuẩn bị muốn xem. Method `suspendAllOperations` sẽ được hiện thực ở phía sau.

2. Ở hàm này, nếu giá trị của `decelerate` là `false` có nghĩa rằng người dùng dừng việc trượt table view. Vì thế, chúng ta sẽ resume những operations cũ đã bị suspend trước đó mà bây giờ người dùng đang thấy, cancel những operation của những cell ẩn đi và start operations của những cell mới mà người dùng thấy. Mình sẽ hiện thực 2 phương thức: `loadImagesForOnscreenCells()`, `resumeAllOperations()` ở phía sau.

3. Phương thức này cũng nói rằng table view dừng scroll do đó chúng ta

sẽ làm giống như số 2 ở trên

Chúng ta vào viewDidLoad để thiết lập thuộc tính delegate của tableView để ListViewController sử dụng được 3 hàm trên này.

```
tableView.delegate = self
```

Cùng thêm những methods còn thiếu vào ListViewController.swift

```
func suspendAllOperations () {  
    pendingOperations.downloadQueue.isSuspended = true  
    pendingOperations.filtrationQueue.isSuspended = true  
}  
  
func resumeAllOperations () {  
    pendingOperations.downloadQueue.isSuspended = false  
    pendingOperations.filtrationQueue.isSuspended = false  
}  
  
func loadImagesForOnscreenCells () {  
    //1  
    if let pathsArray = tableView.indexPathsForVisibleCells {  
        //2  
        var allPendingOperations = Set<Operation>()  
        allPendingOperations.formUnion(Set<Operation>())  
        //3
```



```
var toBeCancelled = allPendingOperations
let visiblePaths = Set(pathsArray)
toBeCancelled.subtract(visiblePaths)
```

```
//4
```

```
var toBeStarted = visiblePaths
toBeStarted.subtract(allPendingOperations)
```

```
// 5
```

```
for indexPath in toBeCancelled {
    if let pendingDownload = pendingOperations[indexPath]
        pendingDownload.cancel()
    }
    pendingOperations.remove(indexPath)
    if let pendingFiltration = pendingOperations[indexPath]
        pendingFiltration.cancel()
    }
    pendingOperations.remove(indexPath)
}
```

```
// 6
```

```
for indexPath in toBeStarted {
    let indexPath = indexPath as NSIndexPath
    let recordToProcess = self.photos[indexPath]
    startOperationsForPhotoRecord(recordToProcess)
}
}
```



Hai hàm `suspendAllOperations` và `resumeAllOperations` được hiện thực một cách đơn giản.

`NSOperationQueues` có thể được suspend bằng việc set thuộc tính `suspended` bằng `true`. Điều này sẽ dừng tất cả những operation trong queue, bạn không thể suspend operation một cách riêng rẽ tại đây.

Phương thức

`loadImagesForOnscreenCells` được hiện thực phức tạp hơn, mình sẽ giải thích những dòng comment như sau:

1.Lấy một array bao gồm `indexPath` của những dòng người dùng thấy trên table view.

2.Hình thành một tập hợp `indexPath` của những operation đang chạy trước đó (pending operations) bằng cách kết hợp tất cả keys của những downloads in progress và the filters in progress.

3.Hình thành một tập hợp tất cả những `indexPaths` với operations để cancel. Bắt đầu với `indexPath` tất cả những operations đang chạy trừ cho `index path` của những hàng thấy. Điều này sẽ để lại tập hợp những operations của những hàng không

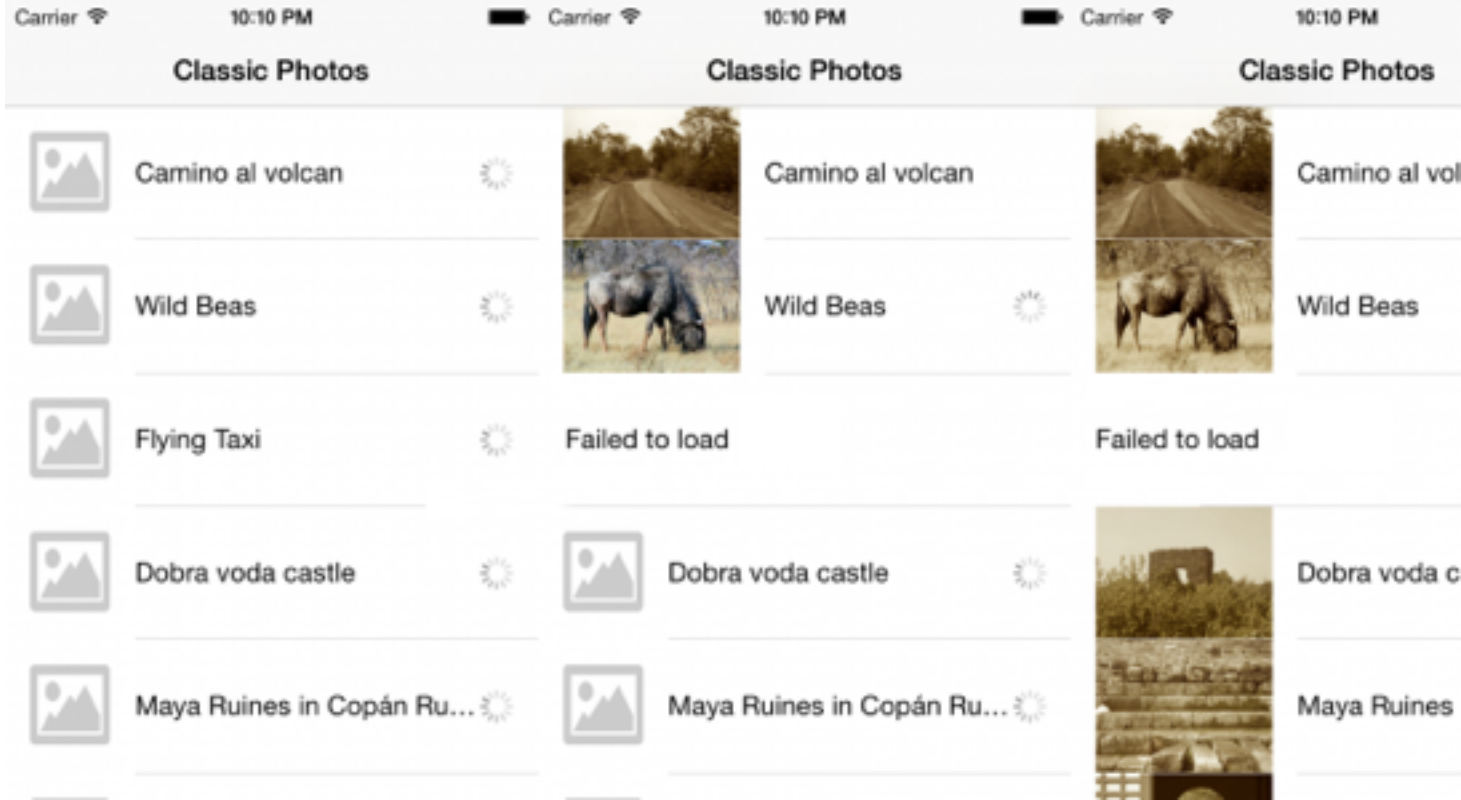
hiển thị.

4.Hình thành tập hợp những index paths mà operations của chúng cần bắt đầu chạy. Bắt đầu với indexpaths của tất cả những hàng nhìn thấy và sau đó remove những cái operations đã hoàn toàn đang chạy (pending).

5.Lập thông qua những cái cần được cancel, cancel chúng và remove reference của chúng từ PendingOperations.

6.Lập thông qua những cái cần được start, và gọi startOperationsForPhotoRecord cho chúng

Build và run ứng dụng, bạn sẽ thấy chúng responsive hơn và quản lý tài nguyên tốt hơn.



Các bạn có thể download completed source code ở link này:

<https://github.com/TuananhSteven/ClassicPhotos-Finished> . Nhớ bấm **Star** cho mình với nha. Cảm ơn nhìu.

Những gì mình sẽ nói tiếp?

Thông qua bài giới thiệu nho nhỏ này, mặc dù nó hơi dài nhưng chứa đựng đầy đủ những kỹ thuật liên quan đến **Operation** và **OperationQueue** trong việc load image từ internet để hiển thị lên table view. Đây là bài mình rất thích và tâm đắc. Hy vọng rằng nó sẽ giúp các bạn giải quyết được phần nào đó những vấn đề gặp phải khi làm

project. Ở bài tiếp theo mình sẽ so sánh giữa Grand Central Dispatch với Operation và OperationQueue. Đây là một câu hỏi thường hay có trong những buổi phỏng vấn về iOS. Hẹn gặp các bạn ở những bài viết kế tiếp trong chuỗi series về iOS concurrency.

Tham khảo

<https://www.raywenderlich.com/76341/use-nsoperation-nsoperationqueue-swift>

Những việc làm hấp dẫn



AI Engineer

Viettel Cyber Security 📍 *Ha Noi*

📶 *Up to \$1,500*

AI



WEB DEVELOPER (CSS, JavaScri...

Công Ty TNHH OKXE Việt Nam

📍 *Ho Chi Minh* 📶 *Negotiable*

CSS

JavaScript

HTML

Web

UI



Front-end Web Developer (Junio...

Aimesoft 📍 *Ha Noi* 📶 *\$500 - \$1,000*

CSS

JavaScript

HTML

Front-End

IOS

SWIFT

MULTI THREADING

CONCURRENCY



The advertisement for Clear TV Key features a bright orange and yellow background. At the top left is the 'clearTVKey' logo. Below it, Vietnamese text reads: 'KHÔNG PHẢI MUA GÓI CƯỚC TRUYỀN HÌNH VẪN XEM ĐƯỢC HÀNG TRĂM KÊNH HD CHẤT LƯỢNG CAO'. To the right is an image of a family in a living room watching TV, with a 'Clear TV Key' device shown in the foreground. A '1080p FULL HD' badge is also present. The main headline in large white letters says 'ĂNG-TEN TIVI XEM CẢ TRĂM KÊNH HD MIỄN PHÍ'. Below this is a red button with white text 'TÌM HIỂU NGAY'. At the bottom, there is a dark grey section with the text 'Stackoverflow CV TEMPLAT cực chất cho DEV' and a blue button with white text 'THỬ NGAY' and a right arrow icon. The URL 'dngcv.vn' is visible in the bottom left corner.

clearTVKey

KHÔNG PHẢI MUA GÓI CƯỚC
TRUYỀN HÌNH VẪN XEM ĐƯỢC
HÀNG TRĂM KÊNH HD
CHẤT LƯỢNG CAO

1080p
FULL HD

**ĂNG-TEN TIVI XEM CẢ
TRĂM KÊNH HD MIỄN PHÍ**

TÌM HIỂU NGAY

Stackoverflow CV TEMPLAT
cực chất cho DEV

THỬ NGAY ➔

dngcv.vn

Bài viết liên quan

- [ios] dark and light mode with rxtheme
- lazy var trong ios swift
- swiftui - công cụ mới để xây dựng ui
- core audio là gì?
- ios: tạo licenses / acknowledgements tự động trong ios settings app

Chia sẻ

 Facebook

 Twitter

 Google+