

I. Authentication (xác thực)

1. Get Started : Bắt đầu với Firebase Authentication trên IOS

Bạn có thể sử dụng Firebase Authentication cho phép người dùng đăng nhập vào ứng dụng của bạn bằng một hay nhiều cách thức đăng nhập khác nhau .Bao gồm địa chỉ email và mật khẩu đăng nhập.

Hướng dẫn sau đây giúp bạn bắt đầu với Firebase Authentication bằng cách hiển thị cho bạn làm thế nào để thêm địa chỉ email và mật khẩu đăng nhập vào ứng dụng của bạn.

1.1 Kết nối ứng dụng của bạn với Firebase Authentication

- Cài đặt Firebase SDK
- Trong giao diện điều khiển Firebase ,thêm ứng dụng của bạn vào dự án Firebase.

1.2 Thêm Firebase Authentication vào dự án Xcode của bạn

- Đầu tiên, đảm bảo sự phụ thuộc sau đây là trong Podfile dự án của bạn .

```
pod 'Firebase/Auth'
```

- Sau đó chạy `pod install` mở file khởi tạo `.xcworkspace`


1.3 Khởi tạo Firebase SDK

- Trong đại diện ứng dụng của bạn, đầu tiên nhập mật khẩu cho Firebase

NHANH

OBJECTIVE-C

```
import Firebase
```


[AppDelegate.swift](#) 

- Sau đó ,trong phương thức `application:didFinishLaunchingWithOptions:` khởi tạo các đối tượng `FIRApp` :

NHANH

OBJECTIVE-C

```
// Use Firebase library to configure APIs
FIRApp.configure()
```

[AppDelegate.swift](#) 

- Nghe để xác thực trạng thái (Listen for authentication state)

Đối với mỗi app's views của bạn cần thông tin người dùng đăng nhập đính kèm trình lắng nghe tới đối tượng `FIRAuth`.


- Trình lắng nghe được gọi đến bất cứ khi nào khi có sự thay đổi của trạng thái đăng nhập của người dùng.

- Đính kèm trình lắng nghe trong bộ điều khiển của phương thức `viewWillAppear` :

NHANH

OBJECTIVE-C

```
self.handle = [[FIRAuth auth]
    addAuthStateDidChangeListener:^(FIRAuth *_Nonnull auth, FIRUser *
        // ...
    )];
```

[MainViewController.m](#) 

- Và lấy ra trình lắng nghe của phương thức `viewWillAppear`

```
self.handle = [[FIRAuth auth]
    addAuthStateDidChangeListener:^(FIRAuth *_Nonnull auth, FIRUser
*_Nullable user) {
    // ...
}];
```

[MainViewController.m](#)

1.4 Đăng ký tài khoản người dùng mới

Tạo một form cho phép ng dùng mới đăng ký trên app của bạn. Sử dụng địa chỉ email của họ và password. Khi hoàn thành form, validate địa chỉ email và pass được cung cấp từ người dùng. Sau đó, chuyển đến phương thức `createUser` :

```
[[FIRAuth auth]
    createUserWithEmail:email
    password:password
    completion:^(FIRUser *_Nullable user,
        NSError *_Nullable error) {
        // ...
    }];
```

[EmailViewController.m](#)

1.5 Đăng nhập người dùng hiện có:

Tạo một form cho phép người dùng hiện có đăng nhập sử dụng địa chỉ email và password của họ. Khi người dùng hoàn thành form. Gọi đến phương thức `signIn` :

```
[[FIRAuth auth] signInWithEmail:_emailField.text
                    password:_passwordField.text
                    completion:^(FIRUser *user, NSError *error) {
                        // ...
                    }];
```

[EmailViewController.m](#)

1.6 Lấy thông tin người dùng

Sau khi người dùng đăng nhập thành công, bạn có thể lấy thông tin về người dùng. Ví dụ, Trong trình lắng nghe trạng thái xác thực của bạn:

```
if (user) {
    NSString *uid = user.uid; // Unique ID, which you can use to identify
    the user on the client side
    NSString *email = user.email;
    NSString *photoURL = user.photoURL;
    [user getTokenWithCompletion:^(NSString *token, NSError *error) {
        // token, if not nil, is an ID token, which you can safely send to a
        backend
    }];
}
```

1.7 Các bước tiếp theo

Tìm hiểu các đề hỗ trợ thêm cho nhà cung cấp nhận dạng khác và tài khoản khách vô danh

2. Quản lý người dùng trong Firebase

2.1 Khởi tạo người dùng

Bạn tạo một người dùng mới trong dự án Firebase của bạn bằng cách gọi đến phương thức : `createUserWithEmail:password:completion:` hoặc đăng nhập người dùng lần đầu tiên sử dụng nhà cung cấp dạng liên kết : ví dụ Google sign in , Facebook Login. Bạn cũng có thể tạo một password – form xác thực người dùng phần xác thực của bảng điều khiển Firebase, trên trang người dùng

2.2 Nhận người dùng đã đăng nhập hiện tại

Khuyến khích sử dụng trình nghe trên đối tượng `Auth` để lấy được người dùng hiện tại :

```
self.handle = [[FIRAuth auth]
    addAuthStateDidChangeListener:^(FIRAuth *_Nonnull auth, FIRUser
*_Nullable user) {
    // ...
}]];
```

Bằng việc sử dụng trình lắng nghe , bạn đảm bảo được rằng đối tượng Auth không ở trong trạng thái trung gian- chẳng hạn như khởi tạo-khi bạn nhận được người dùng hiện tại .

Bạn cũng có thể lấy được người dùng hiện tại bằng các sử dụng thuộc tính `currentUser` Nếu người dùng chưa đăng nhập, `currentUser` là nil :

```
if ([FIRAuth auth].currentUser) {
    // User is signed in.
    // ...
} else {
    // No user is signed in.
    // ...
}
```

Ghi chú : CurrentUser cũng phải được nil bởi vì đối tượng Auth không phải kết thúc khi làm việc. Nếu bạn sử dụng một trình lắng nghe để theo dõi trạng thái đăng nhập của người dùng bạn không cần xử lý trường hợp này.

2.3 Lấy hồ sơ của người dùng

Để lấy được thông tin hồ sơ của người dùng , sử dụng thuộc tính của thể hiện `FIRUser` ví dụ :

```
FIRUser *user = [FIRAuth auth].currentUser;
NSString *email = user.email;
// The user's ID, unique to the Firebase project.
// Do NOT use this value to authenticate with your backend server,
// if you have one. Use getTokenWithCompletion:completion: instead.
NSString *uid = user.uid;
NSURL *photoURL = user.photoURL;
```

2.4 Lấy thông tin hồ sơ của nhà cung cấp cấp thể của người dùng

Để lấy thông tin hồ sơ được tải từ nhà cung cấp đăng nhập được liên kết với người dùng, sử dụng thuộc tính `providerData` ví dụ :

```
id<FIRUserInfo> userInfo = [FIRAuth
auth].currentUser.providerData[indexPath.row];
```

```
cell.textLabel.text = [userInfo providerID];  
// Provider-specific UID  
cell.detailTextLabel.text = [userInfo uid];
```

2.5 Cập nhập hồ sơ của người dùng

Bạn có thể cập nhập thông tin hồ sơ của người dùng – Tên hiển thị và ULR ảnh hồ sơ của người dùng với lớp `FIRUserProfileChangeRequest` ví dụ :

```
FIRUserProfileChangeRequest *changeRequest =  
    [[FIRAuth auth].currentUser profileChangeRequest];  
changeRequest.displayName = userInput;  
[changeRequest commitChangesWithCompletion:^(NSError *_Nullable error) {  
    // ...  
}];
```

2.6 Đặt địa chỉ email của người dùng

Bạn có thể đặt địa chỉ email của người dùng với phương thức :

`updateEmail:completion` ví dụ :

```
[[FIRAuth auth]  
    .currentUser  
    updateEmail:userInput  
    completion:^(NSError *_Nullable error) {  
        // ...  
    }];
```

*Quan trọng : để đặt địa chỉ email ,người dùng phải đăng nhập trong thời gian gần nhất.
Xem xác thực lại người dùng.*

2.7 Gửi email xác minh cho người dùng

Bạn có thể gửi email xác minh đến người dùng với phương thức

`sendEmailVerificationWithCompletion:` ví dụ :

```
[[FIRAuth auth]  
    .currentUser sendEmailVerificationWithCompletion:^(NSError  
*_Nullable error) {  
    // ...  
}];
```

Bạn có thể tùy chỉnh mẫu email được sử dụng trong phần xác thực của bảng điều khiển Firebase trên trang Email Templates .Xem mẫu trong trung tâm trợ giúp của Firebase.

2.8 Đặt mật khẩu của người dùng

Bạn có thể đặt mật khẩu với phương thức `updatePassword:completion:` ví dụ :

```
[[FIRAuth auth]
    .currentUser
    updatePassword:userInput
    completion:^(NSError *_Nullable error) {
        // ...
    }];
```

Quan trọng : để đặt mật khẩu của người dùng, người dùng phải đăng nhập trong thời gian gần đây. Xem lại xác thực người dùng.

2.9 Gửi email đặt lại mật khẩu

Bạn có thể gửi email đặt lại mật khẩu đến người dùng với phương thức

`sendPasswordResetWithEmail:completion:` ví dụ :

```
[[FIRAuth auth]
    sendPasswordResetWithEmail:userInput
    completion:^(NSError *_Nullable error) {
        // ...
    }];
```

Bạn có thể tùy chỉnh mẫu email được sử dụng trong phần Xác thực của bảng điều khiển Firebase, trên trang Email Templates. Xem Mẫu email trong Trung tâm trợ giúp Firebase.

Bạn cũng có thể gửi email đặt lại mật khẩu từ bảng điều khiển Firebase.

2.10 Xóa một người dùng

Bạn có thể xóa một tài khoản người dùng với phương thức :

```
FIRUser *user = [FIRAuth auth].currentUser;

[user deleteWithCompletion:^(NSError *_Nullable error) {
    if (error) {
        // An error happened.
    } else {
        // Account deleted.
    }
}];
```

Bạn có thể xóa nhiều người dùng khỏi phần xác thực của bảng điều khiển Firebase trên trang người dùng.

Quan trọng : để xóa một người dùng, người dùng phải đăng nhập gần đây. Xem xác thực lại người dùng

2.11 Xác nhận lại người dùng

Một vài hoạt động bảo mật nhạy cảm- ví dụ xóa một tài khoản, thiết lập địa chỉ email chính và thay đổi mật khẩu – Yêu cầu người dùng phải đăng nhập gần đây. Nếu bạn thực hiện một trong các hành động này, và người dùng đã đăng nhập quá lâu, hoạt động sẽ bị thất bại với thông báo lỗi : `FIRAuthErrorCodeCredentialTooOld` . , Khi điều này xảy ra, hãy xác thực lại người dùng bằng cách nhận thông tin đăng nhập mới từ người dùng và chuyển các thông tin đăng nhập tới `reauthenticate` :

```
FIRUser *user = [FIRAuth auth].currentUser;
FIRAuthCredential *credential;

// Prompt the user to re-provide their sign-in credentials

[user reauthenticateWithCredential:credential completion:^(NSError
*_Nullable error) {
    if (error) {
        // An error happened.
    } else {
        // User re-authenticated.
    }
}
}];
```

2.12 Import tài khoản người dùng

Bạn có thể import người dùng từ file trong dự án Firebase của bạn bằng các sử dụng lệnh `auth:Import` của Firebase CLI :

```
firebase auth:import users.json --hash-algo=scrypt --rounds=8 --mem-
cost=14
```

3. Xác thực mật khẩu – xác thực với Firebase sử dụng mật khẩu dựa trên tài khoản trên IOS.

Bạn có thể sử dụng Xác thực Firebase để cho phép người dùng xác thực với Firebase bằng địa chỉ email và mật khẩu của họ và để quản lý tài khoản dựa trên mật khẩu của ứng dụng.

3.1 Trước khi bắt đầu

Thêm Firebase vào dự án IOS của bạn. Bao gồm các nhóm sau trong Podfile của bạn :

```
pod 'Firebase/Auth'
```

Nếu bạn chưa kết nối ứng dụng của mình đến dự án Firebase của bạn, hãy làm như vậy từ bảng điều khiển Firebase

Kích hoạt Email/mật khẩu đăng nhập :

- Trong bảng điều khiển Firebase, mở phần Auth
- Trên tab phương thức đăng nhập, kích hoạt phương thức email/mật khẩu và lưu .

3.2 Tạo một tài khoản dựa trên mật khẩu

Để tạo một tài khoản người dùng mới với một mật khẩu, hoàn thành các bước sau trong hoạt động đăng nhập của ứng dụng :

Bước 1 : Import chức năng Firebase vào `UIApplicationDelegate` subclass :

```
import Firebase
```

Bước 2: Cấu hình a FIRApp shared instance (một ví dụ chia sẻ FIRApp), thường ở trong ứng dụng của bạn phương thức :

```
application:didFinishLaunchingWithOptions:
```

```
// Use Firebase library to configure APIs
FIRApp.configure()
```

Bước 3: Khi người dùng mới đăng ký sử dụng biểu mẫu đăng ký của ứng dụng, hãy hoàn tất bất kỳ bước xác nhận tài khoản mới nào ứng dụng của bạn yêu cầu. Ví dụ như xác minh mật khẩu của tài khoản mới được gõ đúng và đáp ứng các yêu cầu phức tạp của bạn

Bước 4: Tạo mới một tài khoản bằng cách chuyển địa chỉ email và mật khẩu của người dùng mới đến : `createUserWithEmail:email:password:completion`

```
FIRAuth.auth()?.createUser(withEmail: email, password: password) { (user,
error) in
    // ...
}
```

Nếu tài khoản mới được tạo thành công, người dùng được đăng nhập, và bạn có thể lấy dữ liệu tài khoản của người dùng từ đối tượng người dùng được chuyển đến phương thức gọi lại. (callback method.)

3.3 Đăng nhập người dùng có địa email và mật khẩu.

Các bước đăng nhập người dùng có mật khẩu tương tự như các bước để tạo một tài khoản mới, trong hoạt động đăng nhập của ứng dụng, hãy thực hiện theo các bước sau:

Bước 1: Import chức năng Firebase trong `UIApplicationDelegate` subclass:

```
import Firebase
```

Bước 2: Cấu hình a `FIRApp` shared instance , thường ở trong ứng dụng của bạn phương thức : `application:didFinishLaunchingWithOptions:`

```
// Use Firebase library to configure APIs
FIRApp.configure()
```

Bước 3 Khi người dùng đăng nhập vào app của bạn, thông qua địa chỉ email và password để : `signInWithEmail:email:password:completion:`

```
FIRAuth.auth()?.signIn(withEmail: email, password: password) { (user,
error) in
    // ...
}
```

Nếu người dùng đăng nhập thành công, bạn có thể lấy dữ liệu tài khoản người dùng từ form đối tượng người dùng được chuyển đến phương thức gọi lại. (callback method.)

3.4 Các bước tiếp theo

Sau khi người dùng đăng nhập lần đầu tiên, một tài khoản người dùng mới được tạo và liên kết với các thông tin đăng nhập – Đó là tên người dùng và mật khẩu hoặc thông tin nhà cung cấp auth – người dùng đã đăng nhập. Tài khoản mới này được lưu trữ như một phần của dự án Firebase của bạn và có thể được sử dụng để xác định người dùng trên mọi ứng dụng trong dự án của bạn, bất kể người dùng đăng nhập như thế nào.

- Trong ứng dụng của bạn, bạn có thể lấy thông tin tiểu sử cơ bản của người dùng từ đối tượng của `FIRUser`. Xem quản lý người dùng.
- Trong Cơ sở dữ liệu thời gian thực Firebase của bạn và Quy tắc bảo mật đám mây (Cloud Storage [Security Rules](#)), bạn có thể nhận ID người dùng duy nhất đã đăng nhập từ biến `auth` và sử dụng nó để kiểm soát dữ liệu người dùng có thể truy cập.

Bạn có thể cho phép người dùng đăng nhập vào ứng dụng của bạn bằng cách sử dụng nhiều nhà cung cấp xác thực bằng cách liên kết chúng chỉ ủy nhiệm của *auth* với tài khoản người dùng hiện có.

Để đăng xuất người dùng, gọi **signOut** :

```
let firebaseAuth = FirebaseAuth.auth()
do {
    try firebaseAuth?.signOut()
} catch let signOutError as NSError {
    print ("Error signing out: %@", signOutError)
}
```

Bạn cũng có thể muốn thêm mã xử lý lỗi cho toàn bộ các lỗi xác thực. Xem [Handle Errors](#).

4. Xác thực sử dụng đăng nhập Google trên IOS

Bạn có thể cho phép người dùng xác thực với Firebase bằng tài khoản Google của họ bằng cách tích hợp Đăng nhập Google vào ứng dụng của bạn.

4.1 Trước khi bắt đầu

Bước 1: Thêm Firebase vào dự án IOS của bạn. Bao gồm các nhóm sau trong tệp Podfile của bạn:

```
pod 'Firebase/Auth'
pod 'GoogleSignIn'
```

Bước 2: Nếu bạn không thể kết nối ứng dụng của bạn đến dự án Firebase, Làm như vậy từ bảng điều khiển Firebase.

Bước 3: Kích hoạt đăng nhập Google trong bảng điều khiển Firebase:

- Trong bảng điều khiển Firebase, mở phần Auth
- Trên tab Phương thức đăng nhập, bật phương thức đăng nhập Google và nhấp vào Lưu.

4.2 Nhập các tệp tiêu đề bắt buộc

Đầu tiên, bạn phải nhập tệp tin tiêu đề Firebase SDK và tệp đăng nhập Google SDK vào ứng dụng.

```
@import Firebase;
@import GoogleSignIn;
```

Trong trình điều khiển chế độ xem của chế độ đăng nhập , hãy nhập tệp tin tiêu đề sau:

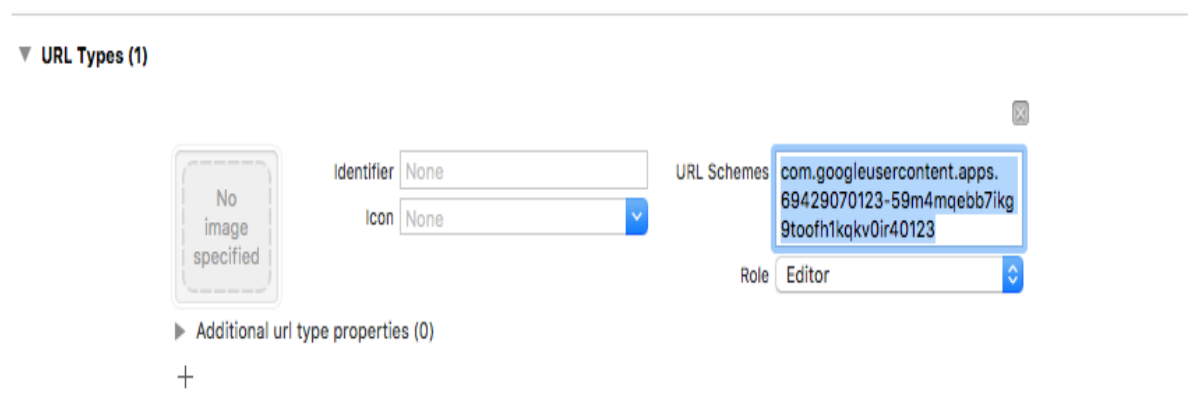
```
@import Firebase;
@import GoogleSignIn;
```

4.3 Thực hiện đăng nhập Google

Thực hiện đăng nhập Google bằng cách làm theo các bước sau. Xem tài liệu dành cho nhà phát triển đăng nhập Google để biết chi tiết về cách sử dụng Đăng nhập Google với iOS.

Bước 1: Thêm lược đồ URL tùy chỉnh vào dự án Xcode của bạn

- Mở cấu hình dự án của bạn: nhấp đúp vào tên dự án trong khung nhìn bên trái. Chọn ứng dụng của bạn từ phần **TARGETS**, sau đó chọn tab **Info** và mở rộng phần **URL Types**.
- Nhấp vào nút + và thêm lược đồ URL cho ID khách hàng đã đảo ngược của bạn. Để tìm giá trị này, hãy mở tệp cấu hình GoogleService-Info.plist và tìm khoá REVERSED_CLIENT_ID. Sao chép giá trị của khóa đó và dán vào hộp URL Schemes trên trang cấu hình. Để trống các trường khác.
- Khi hoàn tất, cấu hình của bạn sẽ giống như sau (nhưng với các giá trị ứng dụng cụ thể của bạn):



Bước 2: Khai báo rằng đại diện ứng dụng thực hiện các giao thức `GIDSignInDelegate` :

```
@interface AppDelegate : UIResponder<UIApplicationDelegate,
GIDSignInDelegate>
```

Bước 3: Trong đại diện (delegate) của ứng dụng

Phương thức : `application:didFinishLaunchingWithOptions:`

Cấu hình đối tượng `FIRApp` và thiết lập đại diện đăng nhập :

```
// Use Firebase library to configure APIs
[FIRApp configure];
```

```
[GIDSignIn sharedInstance].clientID = [FIRApp
defaultApp].options.clientID;
[GIDSignIn sharedInstance].delegate = self;
```

Bước 4: thực hiện phương thức `application:openURL:options:` của đại diện của ứng dụng. Phương thức nên gọi phương thức `handleURL` của cá thể `GIDSignIn`, nó sẽ xử lý đúng URL mà ứng dụng của bạn nhận được khi kết thúc quá trình xác thực.

```
- (BOOL)application:(nonnull UIApplication *)application
    openURL:(nonnull NSURL *)url
    options:(nonnull NSDictionary<NSString *, id> *)options {
    return [[GIDSignIn sharedInstance] handleURL:url
        sourceApplication:options[UIApplicationOpenURLOptionsSourceApplic
ationKey]
        annotation:options[UIApplicationOpenURLOptionsAnnotationKe
y]];
}
```

Để ứng dụng của bạn chạy trên iOS 8 trở lên, cũng thực hiện việc không dùng phương thức `application:openURL:sourceApplication:annotation:`

```
- (BOOL)application:(UIApplication *)application
    openURL:(NSURL *)url
    sourceApplication:(NSString *)sourceApplication
    annotation:(id)annotation {
    return [[GIDSignIn sharedInstance] handleURL:url
        sourceApplication:sourceApplication
        annotation:annotation];
}
```

Bước 5: Trong đại diện ứng dụng, thực hiện gia thức `GIDSignInDelegate` để xử lý quá trình đăng nhập bằng cách xác định các phương thức sau:

```
- (void)signIn:(GIDSignIn *)signIn
didSignInForUser:(GIDGoogleUser *)user
    withError:(NSError *)error {
    // ...
    if (error == nil) {
        GIDAuthentication *authentication = user.authentication;
        FIRAuthCredential *credential =
            [FIRGoogleAuthProvider credentialWithIDToken:authentication.idToken
                accessToken:authentication.accessToken];
    }
}
```

```

    // ...
} else {
    // ...
}
}

- (void)signIn:(GIDSignIn *)signIn
didDisconnectWithUser:(GIDGoogleUser *)user
    withError:(NSError *)error {
    // Perform any operations when the user disconnects from app here.
    // ...
}

```

Bước 6: Khai báo điều khiển của trình đăng nhập của bạn thực hiện giao thức

GIDSignInUIDelegate:

```

@interface MainViewController : UITableViewController<GIDSignInUIDelegate>

```

Bước 7: Trong chế độ xem của bộ điều khiển, hãy ghi đè phương thức `viewDidLoad` để thiết lập giao diện người dùng của đối tượng `GIDSignIn`, và (tùy ý) để đăng nhập bằng có thể đăng nhập bằng âm thanh.

```

- (void)viewDidLoad {
    [super viewDidLoad];

    [GIDSignIn sharedInstance].uiDelegate = self;
    [[GIDSignIn sharedInstance] signIn];

    // TODO(developer) Configure the sign-in button look/feel
    // ...
}

```

Bước 8: Thêm `GIDSignInButton` vào kịch bản phân cảnh, tệp XIB của bạn hoặc nhanh chóng lập trình theo chương trình. Để thêm nút vào tệp tin kịch bản hoặc tệp XIB, hãy thêm Chế độ xem và đặt lớp tùy chỉnh của nó thành `GIDSignInButton`.

Khi bạn thêm một chế độ xem `GIDSignInButton` vào kịch bản phân cảnh, nút đăng nhập không hiển thị trong trình tạo giao diện. Chạy ứng dụng để xem nút đăng nhập.

Bước 9: Nếu bạn muốn tùy chỉnh nút, hãy thực hiện theo các bước sau:

- Trong tệp tin tiêu đề của trình điều khiển chế độ xem của bạn, hãy khai báo nút đăng nhập làm thuộc tính (property)

```

@property(weak, nonatomic) IBOutlet GIDSignInButton *signInButton;

```

- Kết nối nút vào thuộc tính `signInButton` mà bạn vừa khai báo.
- Tùy chỉnh nút bằng cách thiết lập các thuộc tính của đối tượng

`GIDSignInButton`

4.4 Xác thực với Firebase

Trong phương thức `signIn:didSignInForUser:withError:`, nhận ID thông báo Google và mã truy cập của Google từ đối tượng `GIDAuthentication` và trao đổi thông tin cho chứng chỉ Firebase:

```
- (void)signIn:(GIDSignIn *)signIn
didSignInForUser:(GIDGoogleUser *)user
withError:(NSError *)error {
    // ...
    if (error == nil) {
        GIDAuthentication *authentication = user.authentication;
        FIRAuthCredential *credential =
            [FIRGoogleAuthProvider credentialWithIDToken:authentication.idToken
                                              accessToken:authentication.accessToken];
    } else {
        // ...
    }
}
```

Cuối cùng, xác thực với Firebase sử dụng chứng chỉ :

```
[[FIRAuth auth] signInWithCredential:credential
completion:^(FIRUser *user, NSError *error) {
    // ...
    if (error) {
        // ...
        return;
    }
}
```

4.5 Các bước tiếp theo

Sau khi người dùng đăng nhập lần đầu tiên, tài khoản người dùng mới được tạo và liên kết với thông tin xác thực-đó là tên người dùng và mật khẩu hoặc thông tin nhà cung cấp auth-người dùng đã đăng nhập. Tài khoản mới này được lưu trữ như một phần của dự án Firebase của bạn và có thể được sử dụng để xác định người dùng trên mọi ứng dụng trong dự án của bạn, bất kể người dùng đăng nhập như thế nào.

- Trong các ứng dụng của bạn, bạn có thể nhận được thông tin hồ sơ cơ bản của người dùng từ các đối tượng `FIRUser`. Xem [Manage Users](#).

- Trong Cơ sở dữ liệu thời gian thực Firebase của bạn và Quy tắc bảo mật đám mây, bạn có thể nhận ID người dùng duy nhất đã đăng nhập từ biến *auth* và sử dụng nó để kiểm soát dữ liệu người dùng có thể truy cập.

Bạn có thể cho phép người dùng đăng nhập vào ứng dụng của bạn bằng cách sử dụng nhiều nhà cung cấp xác thực bằng cách liên kết chúng chỉ ủy nhiệm của *auth* với tài khoản người dùng hiện có.

Để đăng xuất người dùng, gọi **signOut** :

```
NSError *signOutError;
BOOL status = [[FIRAuth auth] signOut:&signOutError];
if (!status) {
    NSLog(@"Error signing out: %@", signOutError);
    return;
}
```

Bạn cũng có thể muốn thêm mã xử lý lỗi cho toàn bộ các lỗi xác thực. Xem **Handle Errors**.

5. Xác thực sử dụng đăng nhập Facebook trên IOS

Bạn có thể cho phép người dùng của bạn xác thực với Firebase bằng cách sử dụng tài khoản Facebook của họ bằng cách tích hợp Đăng nhập Facebook vào ứng dụng của bạn.

5.1 Trước khi bắt đầu

Bước 1 : Thêm Firebase vào dự án IOS của bạn. Bao gồm các nhóm sau trong tệp Podfile của bạn:

```
pod 'Firebase/Auth'
```

Bước 2: Nếu bạn không thể kết nối ứng dụng của bạn đến dự án Firebase, Làm như vậy từ bảng điều khiển Firebase.

Bước 3: Trên trang [Facebook for Developers](#), tải **App ID** và **App Secret** cho ứng dụng .

Bước 4: Kích hoạt đăng nhập facebook

- Trong bảng điều khiển Firebase, mở phần Auth
- Trên tab **Sign in method**, m kích hoạt phương thức đăng nhập Facebook, và xác định **App ID** và **App Secret** mà bạn nhận được từ Facebook
- Then, make sure your **OAuth redirect URI** (e.g. `my-app-12345.firebaseio.com/__/auth/handler`) is listed as one of your **OAuth redirect URIs** in your Facebook app's settings page on the [Facebook for Developers](#) site in the **Product Settings > Facebook Login** config. (em không dịch được đoạn này ạ ☹)

5.2 Xác thực với Firebase

Bước 1: Tích hợp Đăng nhập Facebook vào ứng dụng của bạn bằng cách làm theo các tài liệu của nhà phát triển. Khi bạn khởi tạo các đối tượng `FBSDKLoginButton`, thiết lập một delegate để nhận các sự kiện đăng nhập và đăng xuất. Ví dụ:

```
let loginButton = FBSDKLoginButton()
loginButton.delegate = self
```

Trong delegate, thực hiện : `didCompleteWithResult:error:`

```
func loginButton(loginButton: FBSDKLoginButton!, didCompleteWithResult
result: FBSDKLoginManagerLoginResult!, error: NSError?) {
    if let error = error {
        print(error.localizedDescription)
        return
    }
    // ...
}
```

Bước 2: Nhập chắc nắn Firebase vào `UIApplicationDelegate` subclass :

```
import Firebase
```

Bước 3: Cấu hình `FIRApp` *shared instance* ,Thường trong ứng dụng phương thức : `application:didFinishLaunchingWithOptions:`

```
// Use Firebase library to configure APIs
FIRApp.configure()
```

Bước 4: Sau khi người dùng đăng nhập thành công, trong quá trình triển khai `didCompleteWithResult:error:` ,nhận mã truy cập cho người dùng đăng nhập và trao đổi nó cho một chứng chỉ Firebase:

```
let credential = FIRFacebookAuthProvider.credential(withAccessToken:
FBSDKAccessToken.current().tokenString)
```

Bước 5: Cuối cùng, Cuối cùng, xác nhận với Firebase sử dụng chứng chỉ Firebase:

```
FIRAuth.auth()?.signIn(with: credential) { (user, error) in
    // ...
    if let error = error {
```



```
// ...  
return  
}
```

5.3 Bước tiếp theo

Sau khi người dùng đăng nhập lần đầu tiên, tài khoản người dùng mới được tạo và liên kết với thông tin xác thực-đó là tên người dùng và mật khẩu hoặc thông tin nhà cung cấp auth-người dùng đã đăng nhập. Tài khoản mới này được lưu trữ như một phần của dự án Firebase của bạn và có thể được sử dụng để xác định người dùng trên mọi ứng dụng trong dự án của bạn, bất kể người dùng đăng nhập như thế nào.

- Trong các ứng dụng của bạn, bạn có thể nhận được thông tin hồ sơ cơ bản của người dùng từ các đối tượng FIRUser. Xem Manage Users.
- Trong Cơ sở dữ liệu thời gian thực Firebase của bạn và Quy tắc bảo mật đám mây, bạn có thể nhận ID người dùng duy nhất đã đăng nhập từ biến auth và sử dụng nó để kiểm soát dữ liệu người dùng có thể truy cập.

Bạn có thể cho phép người dùng đăng nhập vào ứng dụng của bạn bằng cách sử dụng nhiều nhà cung cấp xác thực bằng cách liên kết chúng chỉ ủy nhiệm của auth với tài khoản người dùng hiện có.

5.4. Đăng xuất người dùng

Gọi `signOut` :

```
let firebaseAuth = FirebaseAuth.auth()  
do {  
  try firebaseAuth?.signOut()  
} catch let signOutError as NSError {  
  print ("Error signing out: %@", signOutError)  
}
```

Bạn cũng có thể muốn thêm mã xử lý lỗi cho toàn bộ các lỗi xác thực. Xem Handle Error.

II. *Realtime Database (Cơ sở dữ liệu thời gian thực)*

1. Get Started - Cài đặt và thiết lập trên iOS

Cơ sở dữ liệu thời gian thực Firebase là một cơ sở dữ liệu được lưu trữ dựa trên đám mây. Dữ liệu được lưu trữ dưới dạng JSON và được đồng bộ hoá trong thời gian thực với mọi máy khách kết nối. Khi bạn xây dựng các ứng dụng nền tảng với SDK Android, iOS và JavaScript của chúng tôi, tất cả khách hàng của bạn sẽ chia sẻ một phiên bản Cơ sở dữ liệu thời gian thực và tự động cập nhật những dữ liệu mới nhất.

1.1 Điều kiện tiên quyết

- Cài đặt Firebase SDK
- Thêm ứng dụng và dự án Firebase vào bảng điều khiển Firebase

⇒ Note :

1.2 Thêm cơ sở dữ liệu thời gian thực Firebase vào ứng dụng

Đảm bảo sự phụ thuộc sau đây là trong Podfile của dự án của bạn:

```
pod 'Firebase/Database'
```

Chạy `pod install` và mở tệp tin `.xcworkspace` file đã tạo.

1.3 Cấu hình cơ sở dữ liệu thời gian thực Firebase

Cơ sở dữ liệu thời gian thực cung cấp một quy tắc ngôn ngữ khai báo cho phép bạn xác định cách cấu trúc dữ liệu của mình, cách thức lập chỉ mục và khi dữ liệu của bạn có thể được đọc và ghi vào. Theo mặc định, khả năng đọc và ghi vào cơ sở dữ liệu của bạn bị hạn chế do đó chỉ những người dùng được xác thực mới có thể đọc hoặc ghi dữ liệu. Để bắt đầu mà không cần thiết lập xác thực, bạn có thể định cấu hình các quy tắc của mình để truy cập công cộng. Điều này sẽ làm cho cơ sở dữ liệu của bạn mở cho mọi người, ngay cả những người không sử dụng ứng dụng của bạn, vì vậy hãy chắc chắn hạn chế cơ sở dữ liệu của bạn một lần nữa khi thiết lập xác thực.

1.4 Cài đặt cơ sở dữ liệu thời gian thực Firebase

Bạn phải khởi tạo Firebase trước khi bất kỳ tham chiếu ứng dụng Firebase nào được tạo ra hoặc sử dụng. Nếu bạn đã làm điều này cho một tính năng Firebase khác, bạn có thể bỏ qua bước này.

- Import chức năng Firebase vào `UIApplicationDelegate` subclass:

```
import Firebase
```

- Định cấu hình `FIRApp` *shared instance* , Thường trong ứng dụng phương thức : `application:didFinishLaunchingWithOptions:`

```
// Use Firebase library to configure APIs
FIRApp.configure()
```

- Một khi bạn đã khởi tạo *Firebase Realtime Database*, xác định và tạo một tham chiếu đến cơ sở dữ liệu của bạn như sau:

```
@property (strong, nonatomic) FIRDatabaseReference *ref;

self.ref = [[FIRDatabase database] reference];
```

1.5 Chuẩn bị cho khởi chạy (Launch)

- Trước khi khởi chạy ứng dụng của bạn, chúng tôi khuyên bạn nên xem qua danh sách kiểm tra khởi chạy của chúng tôi để đảm bảo ứng dụng của bạn đã sẵn sàng để khởi chạy!

1.6 Các bước tiếp theo

- Tìm hiểu cách cấu trúc dữ liệu cho Cơ sở dữ liệu thời gian thực.
- Đọc và ghi dữ liệu.
- Xem cơ sở dữ liệu của bạn trong bảng điều khiển Firebase.

2. Structure Data (Cấu trúc dữ liệu)

Hướng dẫn này bao gồm một số khái niệm chính trong kiến trúc dữ liệu và thực hành tốt nhất cho cấu trúc dữ liệu JSON trong Firebase Realtime Database của bạn.

Xây dựng một cơ sở dữ liệu có cấu trúc đúng yêu cầu cần suy tính kỹ trước khá nhiều. Quan trọng nhất, bạn cần phải lập kế hoạch cho dữ liệu sẽ được lưu lại và sau đó được lấy ra để làm sao cho quy trình đó trở nên dễ dàng nhất có thể.

2.1 Cách dữ liệu được cấu trúc: JSON tree

Tất cả dữ liệu Firebase Realtime Database được lưu trữ dưới dạng các đối tượng JSON. Bạn có thể xem cơ sở dữ liệu dưới dạng cây JSON được lưu trữ qua đám mây. Không giống như một cơ sở dữ liệu SQL, không có bảng hoặc bản ghi. Khi bạn thêm dữ liệu vào cây JSON, nó sẽ trở thành một nút trong cấu trúc JSON hiện có với khóa liên quan. Bạn có thể cung cấp các chìa khóa của riêng bạn, chẳng hạn như ID người dùng hoặc tên người dùng, hoặc chúng có thể được cung cấp cho bạn bằng cách sử dụng `childByAutoId`.

⇒ Nếu bạn tạo các khóa của riêng mình, chúng phải được mã hoá UTF-8, có thể là tối đa là 768 byte và không thể chứa ký tự kiểm soát 0-31 hoặc 127 của., \$, #, [,], / Hoặc ASCII.

Ví dụ: hãy xem xét ứng dụng trò chuyện cho phép người dùng lưu trữ hồ sơ cơ bản và danh sách liên hệ. Một hồ sơ người dùng điển hình được đặt tại một đường dẫn, chẳng hạn như `/users/$uid`. Người dùng `alovelace` có thể có mục nhập cơ sở dữ liệu trông giống như sau:

```
{
  "users": {
    "alovelace": {
```

```

    "name": "Ada Lovelace",
    "contacts": { "ghopper": true },
  },
  "ghopper": { ... },
  "eclarke": { ... }
}

```

Mặc dù cơ sở dữ liệu sử dụng cây JSON, dữ liệu được lưu trữ trong cơ sở dữ liệu có thể được đại diện dưới dạng một số loại cây nhất định tương ứng với các loại JSON có sẵn để giúp bạn viết mã có thể duy trì nhiều hơn.

2.2 Thực hành tốt nhất cho các cấu trúc dữ liệu

– Tránh lồng dữ liệu

Bởi vì Firebase Realtime Database cho phép lồng dữ liệu lên đến 32 độ sâu, bạn có thể bị nghĩ sai hướng rằng đây nên là cấu trúc mặc định. Tuy nhiên, khi bạn lấy dữ liệu tại một vị trí trong cơ sở dữ liệu của bạn, bạn cũng lấy lại tất cả các nút con của nó. Ngoài ra, khi bạn cấp cho ai đó đọc hoặc ghi truy cập tại một nút trong cơ sở dữ liệu của bạn, bạn cũng cho phép họ truy cập vào tất cả các dữ liệu trong nút đó. Do đó, trên thực tế, tốt nhất nên giữ cấu trúc dữ liệu bằng phẳng nhất có thể.

Ví dụ về lý do tại sao dữ liệu lồng nhau là xấu, hãy xem xét cấu trúc lồng nhau sau đây:

```

{
  // This is a poorly nested data architecture, because iterating the
  children
  // of the "chats" node to get a list of conversation titles requires
  // potentially downloading hundreds of megabytes of messages
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": { "sender": "ghopper", "message": "Relay malfunction found.
Cause: moth." },
        "m2": { ... },
        // a very long list of messages
      }
    },
    "two": { ... }
  }
}

```

Với thiết kế lồng nhau này, lặp qua các dữ liệu trở nên có vấn đề. Ví dụ: liệt kê các tiêu đề cuộc trò chuyện trò chuyện yêu cầu phải tải toàn bộ `chats` tree, bao gồm tất cả các thành viên và tin nhắn tới máy khách.

2.3 Làm phẳng cấu trúc dữ liệu

Nếu dữ liệu được chia thành các đường dẫn riêng biệt, còn gọi là denormalization (kỹ thuật tối ưu thiết kế csdl), nó có thể được tải xuống hiệu quả trong các cuộc gọi riêng biệt, vì nó là cần thiết. Xem xét cấu trúc phẳng này:

```
{
  // Chats contains only meta info about each conversation
  // stored under the chats's unique ID
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "lastMessage": "ghopper: Relay malfunction found. Cause: moth.",
      "timestamp": 1459361875666
    },
    "two": { ... },
    "three": { ... }
  },

  // Conversation members are easily accessible
  // and stored by chat conversation ID
  "members": {
    // we'll talk about indices like this below
    "one": {
      "ghopper": true,
      "alovelace": true,
      "eclarke": true
    },
    "two": { ... },
    "three": { ... }
  },

  // Messages are separate from data we may want to iterate quickly
  // but still easily paginated and queried, and organized by chat
  // conversation ID
  "messages": {
    "one": {
      "m1": {
        "name": "eclarke",
        "message": "The relay seems to be malfunctioning.",
        "timestamp": 1459361875337
      },
      "m2": { ... },
```

```

    "m3": { ... }
  },
  "two": { ... },
  "three": { ... }
}
}

```

It's now possible to iterate through the list of rooms by downloading only a few bytes per conversation, quickly fetching metadata for listing or displaying rooms in a UI. Messages can be fetched separately and displayed as they arrive, allowing the UI to stay responsive and fast (Giờ đây, có thể duyệt qua danh sách các phòng bằng cách tải xuống chỉ vài byte cho mỗi cuộc hội thoại, nhanh chóng tìm nạp siêu dữ liệu để liệt kê hoặc hiển thị các phòng trong giao diện người dùng. Các thư có thể được tìm nạp riêng lẻ và hiển thị khi chúng đến nơi, cho phép UI đáp ứng nhanh và nhanh chóng)

2.4 Tạo dữ liệu có quy mô

Khi xây dựng ứng dụng, tốt hơn là nên tải xuống một tập hợp con của một danh sách. Điều này đặc biệt phổ biến nếu danh sách chứa hàng ngàn hồ sơ. Khi mối quan hệ này là tĩnh và một hướng, bạn chỉ có thể lòng các đối tượng con dưới cha mẹ.

Đôi khi, mối quan hệ này là năng động hơn, hoặc nó có thể là cần thiết để *denormalize* dữ liệu này. Nhiều lần bạn có thể *denormalize* dữ liệu bằng cách sử dụng một truy vấn để lấy ra một tập con của dữ liệu, như đã thảo luận trong Retrieve Data.

Nhưng ngay cả điều này có thể không đủ. Hãy xem xét, ví dụ, một mối quan hệ hai chiều giữa người dùng và các nhóm. Người dùng có thể thuộc về một nhóm, và các nhóm bao gồm một danh sách người dùng. Khi nói đến thời gian để quyết định nhóm người sử dụng thuộc về, mọi thứ trở nên phức tạp.

Điều cần thiết là cách tạo nhả để liệt kê các nhóm người dùng thuộc về và chỉ lấy dữ liệu cho các nhóm đó. Một chỉ mục của các nhóm có thể giúp ích rất nhiều ở đây:

```

// An index to track Ada's memberships
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      // Index Ada's groups in her profile
      "groups": {

```

```

        // the value here doesn't matter, just that the key exists
        "techpioneers": true,
        "womentechmakers": true
    }
},
...
},
"groups": {
    "techpioneers": {
        "name": "Historical Tech Pioneers",
        "members": {
            "alovelace": true,
            "ghopper": true,
            "eclarke": true
        }
    },
    ...
}
}

```

Bạn có thể nhận thấy rằng điều này trùng lặp một số dữ liệu bằng cách lưu trữ các mối quan hệ dưới cả hai bản ghi của Ada và dưới nhóm. Bây giờ ADA LOVELACE được lập chỉ mục dưới một nhóm, và các nhà nghiên cứu công nghệ được liệt kê trong hồ sơ của Ada. Vì vậy, để xóa Ada khỏi nhóm, nó phải được cập nhật ở hai nơi.

Đây là sự dư thừa cần thiết cho các mối quan hệ hai chiều. Nó cho phép bạn nhanh chóng tìm nạp thành viên của Ada, ngay cả khi danh sách người dùng hoặc các nhóm có quy mô lên hàng triệu hoặc khi các quy tắc bảo mật cơ sở dữ liệu thời gian thực ngăn truy cập vào một số hồ sơ.

Cách tiếp cận này, đảo ngược dữ liệu bằng cách liệt kê các ID là các khóa và thiết lập giá trị là true, làm cho việc kiểm tra một khóa đơn giản như đọc `/users/$uid/groups/$group_id group_id` và kiểm tra nếu nó là null. Chỉ mục nhanh hơn và hiệu quả hơn so với truy vấn hoặc quét dữ liệu.

2.5 Các bước tiếp theo

- Lưu dữ liệu vào Realtime Database
- Lấy lại dữ liệu từ Realtime Database

3. Đọc và ghi dữ liệu trên IOS

3.1 Lấy một FIRDatabaseReference

Để đọc hoặc ghi dữ liệu từ csdl , bạn cần một ví dụ của **FIRDatabaseReference** :

```
@property (strong, nonatomic) FIRDatabaseReference *ref;

self.ref = [[FIRDatabase database] reference];
```

3.2 Đọc và ghi dữ liệu

Tài liệu này bao gồm các vấn đề cơ bản về đọc và viết dữ liệu của Firebase.

Dữ liệu của Firebase được ghi vào một tài liệu tham khảo FIRDatabase và được lấy ra bằng cách gắn một trình nghe không đồng bộ vào tham chiếu. Trình nghe được kích hoạt một lần cho trạng thái ban đầu của dữ liệu và bất cứ khi nào dữ liệu thay đổi.

Note : Theo mặc định, khả năng đọc và ghi vào cơ sở dữ liệu của bạn bị hạn chế do đó chỉ những người dùng được xác thực mới có thể đọc hoặc ghi dữ liệu. Để bắt đầu mà không cần thiết lập xác thực, bạn có thể định cấu hình các quy tắc của mình để truy cập công cộng. Điều này sẽ làm cho cơ sở dữ liệu của bạn mở cho mọi người, ngay cả những người không sử dụng ứng dụng của bạn, vì vậy hãy chắc chắn hạn chế cơ sở dữ liệu của bạn một lần nữa khi thiết lập xác thực

3.3 Thao tác ghi cơ bản

Đối với các thao tác ghi cơ bản, bạn có thể sử dụng `setValue` để lưu dữ liệu vào một tham chiếu được chỉ định, thay thế bất kỳ dữ liệu hiện có nào ở đường dẫn đó. Bạn có thể sử dụng phương pháp này để chuyển các loại tương ứng với các loại JSON có sẵn như sau :

- NSString
- NSNumber
- NSDictionary
- NSArray

Ví dụ, bạn có thể thêm một người dùng với `setValue` như sau:

```
[[[_ref child:@"users"] child:user.uid]
  setValue:@{@"username": username}];
```

Sử dụng `setValue` theo cách này ghi đè dữ liệu ở vị trí đã chỉ định, bao gồm bất kỳ nút con nào. Tuy nhiên, bạn vẫn có thể cập nhật một nút con mà không cần viết lại toàn bộ đối tượng. Nếu bạn muốn cho phép người dùng cập nhật hồ sơ của họ, bạn có thể cập nhật tên người dùng như sau:

```
[[[_ref child:@"users"] child:user.uid] child:@"username"]
  setValue:username];
```

3.4 Nghe các sự kiện giá trị (Listen for value events)

Để đọc dữ liệu tại một đường dẫn và lắng nghe những thay đổi, hãy sử dụng phương thức `observeEventType:withBlockOrobserveSingleEventOfType:` của **FIRDatabaseReference** để quan sát sự kiện `FIRDataEventTypeValue`.

Event type	Typical usage
<code>FIRDataEventTypeValue</code>	Read and listen for changes to the entire contents of a path.

Bạn có thể sử dụng sự kiện `FIRDataEventTypeValue` để đọc dữ liệu ở một đường dẫn nhất định vì nó tồn tại vào thời điểm xảy ra sự kiện. Phương pháp này được kích hoạt một lần khi trình nghe được đánh kèm và một lần nữa mỗi khi dữ liệu bao gồm dữ liệu con có sự thay đổi. Gọi lại sự kiện được gửi qua `snapshot` chứa tất cả dữ liệu tại vị trí đó, bao gồm dữ liệu con. Nếu không có dữ liệu, giá trị của `snapshot` là nil.

Important: Sự kiện `FIRDataEventTypeValue` được kích hoạt mỗi khi dữ liệu được thay đổi tại tham chiếu cơ sở dữ liệu được chỉ định, bao gồm cả thay đổi đối với dữ liệu con. Để giới hạn kích thước `snapshot` của bạn, hãy chỉ đánh kèm ở mức cao nhất cần để xem các thay đổi. Ví dụ, gắn một trình nghe vào gốc của cơ sở dữ liệu của bạn không được khuyến khích.

Ví dụ sau minh họa một ứng dụng viết blog xã hội lấy ra các chi tiết của một bài đăng từ cơ sở dữ liệu:

```
_refHandle = [_postRef observeEventType:FIRDataEventTypeValue
withBlock:^(FIRDataSnapshot * _Nonnull snapshot) {
    NSDictionary *postDict = snapshot.value;
    // ...
}];
```

Trình nghe nhận được một `FIRDataSnapshot` có chứa dữ liệu tại vị trí xác định trong cơ sở dữ liệu tại thời điểm sự kiện trong thuộc tính giá trị của nó. Bạn có thể gán các giá trị cho các loại bản ghi thích hợp, chẳng hạn như `NSDictionary`. Nếu không có dữ liệu ở vị trí, giá trị là nil.

3.5 Đọc dữ liệu một lần

Trong một số trường hợp, bạn có thể muốn một cuộc gọi lại được gọi là một lần và sau đó được gỡ bỏ ngay lập tức, chẳng hạn như khi khởi tạo một thành phần UI mà bạn không mong đợi thay đổi. Bạn có thể sử dụng phương pháp `observeSingleEventOfType` để đơn giản hóa kịch bản này: sự kiện gọi lại được thêm vào kích hoạt một lần và sau đó không kích hoạt lại.

Điều này rất hữu ích cho dữ liệu mà chỉ cần được nạp một lần và không phải là dữ kiện sẽ thay đổi thường xuyên hoặc yêu cầu nghe chủ động. Ví dụ: ứng dụng viết blog trong các ví dụ trước sử dụng phương pháp này để tải hồ sơ của người dùng khi họ bắt đầu đăng bài đăng mới:

```
NSString *userID = [FIRAuth auth].currentUser.uid;
[[[_ref child:@"users"] child:userID]
observeSingleEventOfType:FIRDataEventTypeValue withBlock:^(FIRDataSnapshot
* _Nonnull snapshot) {
    // Get user value
    User *user = [[User alloc]
initWithUsername:snapshot.value[@"username"]];

    // ...
} withCancelBlock:^(NSError * _Nonnull error) {
    NSLog(@"%@", error.localizedDescription);
}];
```

3.6 Đăng cập nhật hoặc xóa dữ liệu

– Cập nhật các trường cụ thể

Để đồng thời viết cho các dữ liệu con cụ thể của nút mà không ghi đè lên các nút con khác, sử dụng phương thức `updateChildValues`.

Khi gọi hàm `updateChildValues`, bạn có thể cập nhật các giá trị cấp thấp hơn bằng cách xác định đường dẫn cho khoá. Nếu dữ liệu được lưu trữ ở nhiều vị trí để tăng cường quy mô, bạn có thể cập nhật tất cả các phiên bản của dữ liệu đó bằng cách sử dụng **data fan-out**. Ví dụ: ứng dụng viết trên mạng xã hội có thể muốn tạo một bài đăng và đồng thời cập nhật nó lên nguồn cấp dữ liệu hoạt động gần đây và nguồn cấp dữ liệu hoạt động của người đăng. Để thực hiện việc này, ứng dụng viết blog sử dụng mã như sau:

```
NSString *key = [[_ref child:@"posts"] childByAutoId].key;
NSDictionary *post = @{@"uid": userID,
                        @"author": username,
                        @"title": title,
                        @"body": body};
NSDictionary *childUpdates = @{@"/posts/" stringByAppendingString:key]:
post,
                              [NSString stringWithFormat:@"/user-
posts/%@/%@", userID, key]: post};
[_ref updateChildValues:childUpdates];
```

Ví dụ này sử dụng `childByAutoId` để tạo một bài đăng trong nút chứa các bài viết cho tất cả người dùng tại `/posts/$postid` và đồng thời lấy khoá với `getKey()`. Khóa sau đó có thể được sử dụng để tạo mục nhập thứ hai trong bài đăng của người dùng tại `/user-posts/$userid/$postid`.

Sử dụng các đường dẫn này, bạn có thể thực hiện cập nhật đồng thời nhiều vị trí trong cây JSON với một lời gọi đến `updateChildValues`, chẳng hạn như cách ví dụ này tạo ra bài đăng mới ở cả hai vị trí. Cập nhật đồng thời bằng cách này là nguyên tử (atomic): hoặc tất cả các bản cập nhật thành công hoặc tất cả các bản cập nhật thất bại.

- Xóa dữ liệu

Cách đơn giản nhất để xóa dữ liệu là gọi `removeValue` dựa trên vị trí của dữ liệu đó.

Bạn cũng có thể xóa bằng cách xác định nil như là giá trị cho một hoạt động ghi khác như `setValue` hoặc `updateChildValues`. Bạn có thể sử dụng kỹ thuật này với `updateChildValues` để xóa nhiều dữ liệu con trong một lời gọi API duy nhất.

3.7 Gỡ bỏ trình nghe

Trình quan sát không tự động ngừng đồng bộ hóa dữ liệu khi bạn rời khỏi `ViewController`. Nếu một trình quan sát không được gỡ bỏ đúng cách, nó sẽ tiếp tục đồng bộ dữ liệu với bộ nhớ cục bộ. Khi trình quan sát không còn cần thiết nữa, hãy loại bỏ nó bằng cách truyền đi `FIRDatabaseHandle` tới phương thức `removeObserverWithHandle`.

Khi bạn thêm một khối gọi lại cho một tham chiếu, một `FIRDatabaseHandle` sẽ được trả lại. Các xử lý này có thể được sử dụng để loại bỏ khối gọi lại.

Nếu nhiều trình nghe đã được thêm vào một tài liệu tham khảo cơ sở dữ liệu, mỗi trình nghe được gọi là khi một sự kiện được nâng lên. Để dừng đồng bộ hóa dữ liệu ở vị trí đó, bạn phải xóa tất cả các trình quan sát tại một vị trí bằng cách gọi phương thức `removeAllObservers`.

Gọi `removeObserverWithHandle` hoặc `removeAllObservers` trên trình lắng nghe không tự động xóa trình nghe được đăng ký trên các nút con của nó; Bạn cũng phải theo dõi những tài liệu tham khảo hoặc xử lý để loại bỏ chúng.

3.8 Lưu dữ liệu như các tiến trình

Khi làm việc với dữ liệu có thể bị hỏng do sửa đổi đồng thời, chẳng hạn như các bộ đếm gia tăng, bạn có thể sử dụng một hoạt động tiến trình. Bạn đưa ra thao tác này hai đối số: một chức năng cập nhật và một callback hoàn thành tùy chọn. Chức năng cập nhật lấy trạng thái hiện tại của dữ liệu như một đối số và trả về trạng thái mong muốn mới mà bạn muốn ghi.

Ví dụ: trong ứng dụng viết blog xã hội ví dụ, bạn có thể cho phép người dùng gắn dấu sao và bỏ các bài đăng và theo dõi có bao nhiêu ngôi sao mà một bài đăng đã nhận được như sau:

```
[ref runTransactionBlock:^(FIRTransactionResult * _Nonnull(FIRMutableData *
_Nonnull currentData) {
    NSMutableDictionary *post = currentData.value;
    if (!post || [post isEqual:[NSNull null]]) {
        return [FIRTransactionResult successWithValue:currentData];
    }

    NSMutableDictionary *stars = post[@"stars"];
    if (!stars) {
        stars = [[NSMutableDictionary alloc] initWithCapacity:1];
    }
    NSString *uid = [FIRAuth auth].currentUser.uid;
    int starCount = [post[@"starCount"] intValue];
    if (stars[uid]) {
        // Unstar the post and remove self from stars
        starCount--;
        [stars removeObjectForKey:uid];
    } else {
        // Star the post and add self to stars
        starCount++;
        stars[uid] = @YES;
    }
    post[@"stars"] = stars;
    post[@"starCount"] = @(starCount);

    // Set value and report transaction success
    currentData.value = post;
    return [FIRTransactionResult successWithValue:currentData];
} andCompletionBlock:^(NSError * _Nullable error,
                        BOOL committed,
                        FIRDataSnapshot * _Nullable snapshot) {
    // Transaction completed
    if (error) {
        NSLog(@"%@", error.localizedDescription);
    }
}];
```

Việc sử dụng một tiến trình ngăn không cho số liệu sao không chính xác nếu nhiều người dùng cùng một bài viết cùng một lúc hoặc khách hàng có dữ liệu cũ. Giá trị chứa trong lớp `FIRMutableData` ban đầu là giá trị cuối cùng được biết của khách

hàng đối với đường dẫn, hoặc là không có nếu không có. Máy chủ so sánh giá trị ban đầu với giá trị hiện tại và chấp nhận tiến trình nếu các giá trị khớp, hoặc từ chối nó. Nếu tiến trình bị từ chối, máy chủ trả về giá trị hiện tại cho khách hàng, điều này sẽ chạy lại tiến trình với giá trị được cập nhật. Việc này lặp lại cho đến khi tiến trình được chấp nhận hoặc quá nhiều lần đã được thực hiện.

Lưu ý: Bởi vì `runTransactionBlock:andCompletionBlock:` được gọi là nhiều lần, nó phải có khả năng xử lý dữ liệu không. Ngay cả khi có dữ liệu trong cơ sở dữ liệu từ xa của bạn, nó có thể không được lưu trữ cục bộ khi chức năng giao dịch được chạy, kết quả là không có giá trị ban đầu.

3.9 Ghi dữ liệu ngoại tuyến

Nếu một client mất kết nối mạng, ứng dụng của bạn sẽ tiếp tục hoạt động chính xác.

Mỗi client kết nối với một cơ sở dữ liệu Firebase duy trì phiên bản nội bộ của bất kỳ dữ liệu đang hoạt động. Khi dữ liệu được viết, nó sẽ được ghi vào phiên bản nội bộ đầu tiên trước. Các client Firebase sau đó đồng bộ dữ liệu đó với các máy chủ cơ sở dữ liệu từ xa và với các client khác trên cơ sở "nỗ lực tốt nhất".

Kết quả là, tất cả các bản ghi vào cơ sở dữ liệu kích hoạt các sự kiện cục bộ ngay lập tức, trước khi bất kỳ dữ liệu nào được ghi vào máy chủ. Điều này có nghĩa là ứng dụng của bạn vẫn đáp ứng bất kể độ trễ hoặc kết nối mạng.

Khi kết nối được thiết lập lại, ứng dụng của bạn nhận được tập hợp các sự kiện thích hợp để client đồng bộ với trạng thái máy chủ hiện tại, mà không phải ghi bất kỳ mã tùy chỉnh nào.

3.10 Các bước tiếp theo

- Làm việc với danh sách của dữ liệu
- Tìm hiểu các cấu trúc dữ liệu

III. Storage

Get started.

- Cloud Storage for Firebase cho phép bạn tải lên và chia sẻ nội dung do người dùng tạo ra, ví dụ như hình ảnh và video, xây dựng nội dung đa phương tiện cho ứng dụng của mình. Dữ liệu sẽ được lưu trữ trong bộ nhớ Google Cloud Storage, có thể lưu trữ dữ liệu có dung lượng lớn, dễ dàng truy cập ở bất kỳ đâu. Đồng thời cho phép bạn tải lên các tệp này một cách an toàn trực tiếp từ các thiết bị di động và các trình duyệt web.
- Để sử dụng tiện ích này thì:

1.Cài đặt Firebase SDK.

2.Thêm ứng dụng vào trong Firebase.

- Thiết lập truy cập công khai cho người dùng.

Cloud Storage for Firebase cung cấp một quy tắc ngôn ngữ khai báo cho phép bạn xác định cách cấu trúc dữ liệu của bạn, cách lập chỉ mục và dữ liệu của bạn có thể được đọc và ghi vào. Theo mặc định, đọc và ghi truy cập vào Storage bị hạn chế vì vậy chỉ những người dùng đã đăng ký mới có thể đọc hoặc ghi dữ liệu. Để bắt đầu mà không cần đăng ký, bạn có thể định cấu hình các quy tắc của mình để truy cập công khai, tuy nhiên điều này sẽ làm cho tất cả mọi người kể cả người không sử dụng ứng dụng của bạn cũng có thể truy cập đến vì vậy hãy thiết lập các xác thực cần thiết.

- Thiết lập Cloud Storage

Để sử dụng được Firebase SDK trong project thì phải tạo thêm ứng dụng trong Firebase Console.

Bước1: import modul Firebase vào UIApplicationDelegate

SWIFT

OBJECTIVE-C

```
import Firebase
```

Bước 2: Configure một đối tượng FIRApp, trong ứng dụng của bạn

SWIFT

OBJECTIVE-C

```
// Use Firebase library to configure APIs
FIRApp.configure()
```

Bước 3: Tham chiếu đến nơi lưu trữ, sử dụng ứng dụng Firebase mặc định:

OBJECTIVE-C

SWIFT

```
FIRStorage *storage = [FIRStorage storage];
```

Thiết lập nâng cao

Một số trường hợp cần có thêm một số thiết lập:

- Sử dụng kho lưu trữ ở nhiều vùng địa lý: là cần thiết vì nếu ứng dụng của bạn được sử dụng bởi rất nhiều người trên toàn thế giới(giúp giảm độ trễ)
- Sử dụng nhóm lưu trữ trong các lớp lưu trữ khác nhau: nếu có các dữ liệu khác nhau thì các lớp lưu trữ khác nhau giúp thiết lập các nhóm lưu trữ,tiện cho việc truy cập, tạo bản sao lưu.
- Sử dụng nhóm lưu trữ với nhiều người dùng đã được chứng thực trong cùng một ứng dụng: giúp bạn quản lý các tài khoản đã đăng ký khi xây dựng ứng dụng cho phép nhiều tài khoản đăng nhập.

Sử dụng nhiều bộ nhớ lưu trữ: Nếu bạn muốn sử dụng nhóm lưu trữ khác với cài đặt mặc định được cung cấp ở trên hoặc sử dụng nhiều nhóm lưu trữ trong một ứng dụng, bạn có thể tạo một ứng dụng của FIRStorage để tham chiếu nhóm dữ liệu của bạn:

OBJECTIVE-C

SWIFT

```
// Get a non-default Storage bucket
FIRStorage storage = [FIRStorage storageWithURL:@"gs://my-custom-bucket"];
```

Nhập dữ liệu: Khi nhập một nhóm dữ liệu cần lưu trữ vào Firebase, bạn sẽ phải cấp Firebase khả năng truy cập các tệp này bằng công cụ gsutil trong Google Cloud SDK:

```
gsutil -m acl ch -r -u firebase-storage@system.gserviceaccount.com:0 gs://<your-cloud-storage-b
```

Sử dụng ứng dụng Firebase tùy chỉnh: Nếu bạn đang xây dựng ứng dụng phức tạp hơn bằng cách sử dụng FIRApp tùy chỉnh, bạn có thể tạo một thể hiện của FIRStorage được khởi tạo với ứng dụng đó:

OBJECTIVE-C

SWIFT

```
// Get the default bucket from a custom FIRApp
FIRStorage storage = [FIRStorage storageForApp:customApp];

// Get a non-default bucket from a custom FIRApp
FIRStorage storage = [FIRStorage storageForApp:customApp withURL:@"gs://my-custom-bucket"];
```

Create a Reference:

Tệp của bạn được lưu trữ trong bộ nhớ Google Cloud Storage. Các tệp trong nhóm này được trình bày theo cấu trúc phân cấp. Bằng cách tạo tham chiếu đến tệp, ứng dụng của bạn sẽ có quyền truy cập vào tệp đó. Các tệp này sau đó có thể được sử dụng để tải lên, cập nhật dữ liệu hoặc xóa, khi tham chiếu dữ liệu có thể trở đến bất kỳ đâu.

Một số hạn chế:

1. Tổng chiều dài tham chiếu.fullPath phải từ 1 đến 1024 byte khi mã hoá UTF-8.
2. Không có ký tự trả lại hàng hoặc ký tự dòng.
3. Tránh sử dụng các ký tự: #, [,], *, ?, vì chúng khó sử dụng với công cụ khác như Firebase Realtime hoặc gsutil.

Upload file:

Trước tiên hãy tạo một tham chiếu Cloud Storage tới vị trí trong Cloud Storage bạn muốn tải lên tệp tin

Có 2 cách để tải dữ liệu đó là:

- Tải lên từ dữ liệu trong bộ nhớ
- Tải lên từ một URL đại diện cho một tệp trên thiết bị

Bạn có thể upload từ Memory, Stream hoặc Local File.

Download file

Để tải xuống tệp, trước tiên phải tạo một tham chiếu Cloud Storage tới tệp muốn tải xuống. Bạn có thể tạo tham chiếu bằng cách nối các đường dẫn con vào gốc lưu trữ hoặc bạn có thể tạo một tham chiếu từ một gs: // hoặc https: // URL hiện có đang tham chiếu một đối tượng trong Cloud Storage, theo 3 cách:

- Tải xuống NSData trong bộ nhớ
- Tải xuống NSURL cho một tệp trên thiết bị
- Tạo một NSURL cho tệp trực tuyến.

IV. AdMob

- Là một phần của nền tảng dịch vụ di động Firebase, sử dụng tệp tin có tên GoogleService-Info.plist để lưu trữ thông tin cấu hình về ứng dụng của bạn
- Sử dụng SDK quảng cáo trên điện thoại di động của Google. SDK quảng cáo trên điện thoại di động của Google giúp các nhà phát triển ứng dụng thu thập thông tin chi tiết về người dùng của họ, tăng lượng mua hàng trong ứng dụng và tối đa hoá doanh thu quảng cáo. Để thực hiện việc này, tích hợp mặc định của SDK quảng cáo trên điện thoại di động thu thập thông tin như thông tin thiết bị, thông tin vị trí do nhà xuất bản cung cấp và thông tin mua hàng chung trong ứng dụng (chẳng hạn như giá mua hàng và đơn vị tiền tệ).

Get Started.

1. Thêm Bộ công cụ phát triển Firebase và Mobile Ads

Cách tốt nhất để lấy tệp này là đăng nhập vào Firebase Console và đăng ký một ứng dụng. Khi được yêu cầu nhận dạng, hãy nhập ID nhóm từ dự án bạn muốn sử dụng. Cách tốt nhất để thêm Firebase và SDK Quảng cáo trên Điện thoại di động là sử dụng CocoaPods, được mô tả dưới đây:

- Đầu tiên tạo 1 tệp
- Chạy cập nhật cho tệp
- Xây dựng chương trình

a.Yêu cầu đầu tiên để xây dựng:

+ Thêm GADBannerView: vào Main.storyboard chọn UIView chọn GADBannerView

+ Thêm các ràng buộc trên GADBannerView:

- +Xác định vị trí ở cuối màn hình
- + Đặt kích thước hiển thị rộng 320 và cao 50
- + Trung tâm

Click vào Align, thêm một ràng buộc cho Horizontal trong Container với một giá trị là 0(giúp trang quảng cáo hiển thị theo chiều ngang)

- Thêm tham chiếu vào mã GADBannerView:

GADBannerView cần một tham chiếu trong mã để tải quảng cáo vào nó: Mở Editor chọn View> Assistant Editor> Show Assistant Editor. Tiếp theo, giữ phím điều khiển, nhấp vào GADBannerView (ở ô giữa) và kéo con trỏ của bạn lên tới ViewController. Đối với dự án Swift, làm theo các bước trên nhưng thêm một tham chiếu đến GADBannerView trong tệp ViewController.swift. Xcode tạo ra và kết nối một tài sản cho bạn. Đặt tên cho nó là "bannerView", và chọn Connect.

Để giải quyết lỗi biên dịch, thêm `@import GoogleMobileAds` vào `ViewController.h` hoặc nhập `GoogleMobileAds` vào `ViewController.swift` để trình biên dịch biết rằng GADBannerView là một lớp hợp lệ.

- Khởi tạo SDK quảng cáo trên điện thoại di động của Google: bằng cách gọi `configureWithApplicationID`: trong ứng dụng: `didFinishLaunchingWithOptions`: phương thức `AppDelegate.m` hoặc `AppDelegate.swift`.
- Tải quảng cáo vào GADBannerView
- Cuối cùng, thêm mã vào `ViewController.m` hoặc `ViewController.swift` nạp một quảng cáo vào chế độ xem banner

App Transport Security

- Là một tính năng bảo mật được giới thiệu trong iOS 9. Nó được kích hoạt mặc định cho các ứng dụng mới và thực thi kết nối an toàn.
- Tất cả thiết bị iOS 9 và iOS 10 đang chạy các ứng dụng được xây dựng với Xcode 7 trở lên mà không tắt ATS sẽ bị ảnh hưởng bởi thay đổi này, như: ảnh hưởng tích hợp ứng dụng của bạn với SDK quảng cáo trên điện thoại di động của Google.
- Thông báo nhật ký sau xuất hiện khi một ứng dụng tuân thủ ATS không cố gắng phân phối quảng cáo thông qua HTTP trên iOS 9 hoặc iOS 10: An ninh Vận tải ứng dụng đã chặn tải tài nguyên HTTP vì nó không an toàn.

Cách xử lý:

Để đảm bảo quảng cáo của bạn không bị ảnh hưởng bởi ATS, hãy thực hiện theo các bước sau:

- Cập nhật lên phiên bản 7.15.0 hoặc mới hơn của SDK quảng cáo trên điện thoại di động.
- Thêm các `NSAllowsArbitraryLoads`, `NSAllowsArbitraryLoadsForMedia` và `NSAllowsArbitraryLoadsInWebContent` ngoại lệ vào tệp tin `Info.plist` của ứng dụng để vô hiệu hóa các hạn chế ATS

Banner Ads

- **Banner Size:** Hỗ trợ kích thước quảng cáo trên điện thoại di động.
- **Smart Banners:** biểu ngữ thông minh có khả năng hiển thị quảng cáo theo kích thước màn hình điện thoại, chiều quảng cáo màn hình được chia làm 3 cấp:
 - o độ cao màn hình thiết bị ≤ 400

- 400 <độ cao màn hình thiết bị <= 720
 - độ cao màn hình thiết bị > 720
- nếu quảng cáo hình ảnh không đủ lớn để chiếm không gian phân bố thì hình ảnh được điền vào giữa.
- Sau khi chạy quảng cáo hết một vòng thì nó được lặp lại quảng cáo trước đó.
 - Thêm người gửi thông tin quảng cáo.
 - AdView: didFailToReceiveAdWithError: Gọi lại này được gửi khi loadRequest: đã thất bại, thường là do mạng thất bại, lỗi cấu hình ứng dụng hoặc thiếu khoảng không quảng cáo.
 - AdViewDidDismissScreen: Đã gửi khi người dùng đã thoát khỏi giao diện người dùng toàn màn hình của người gửi.
 - AdViewWillDismissScreen: Gửi ngay trước khi giao diện người dùng toàn màn hình của người gửi không nhận, khôi phục ứng dụng của bạn và trình điều khiển chế độ xem gốc.
 - AdViewWillLeaveApplication: gửi trước khi ứng dụng được tiếp cận hoặc chấm dứt.