Firebase Authentication

What is Firebase Authentication?

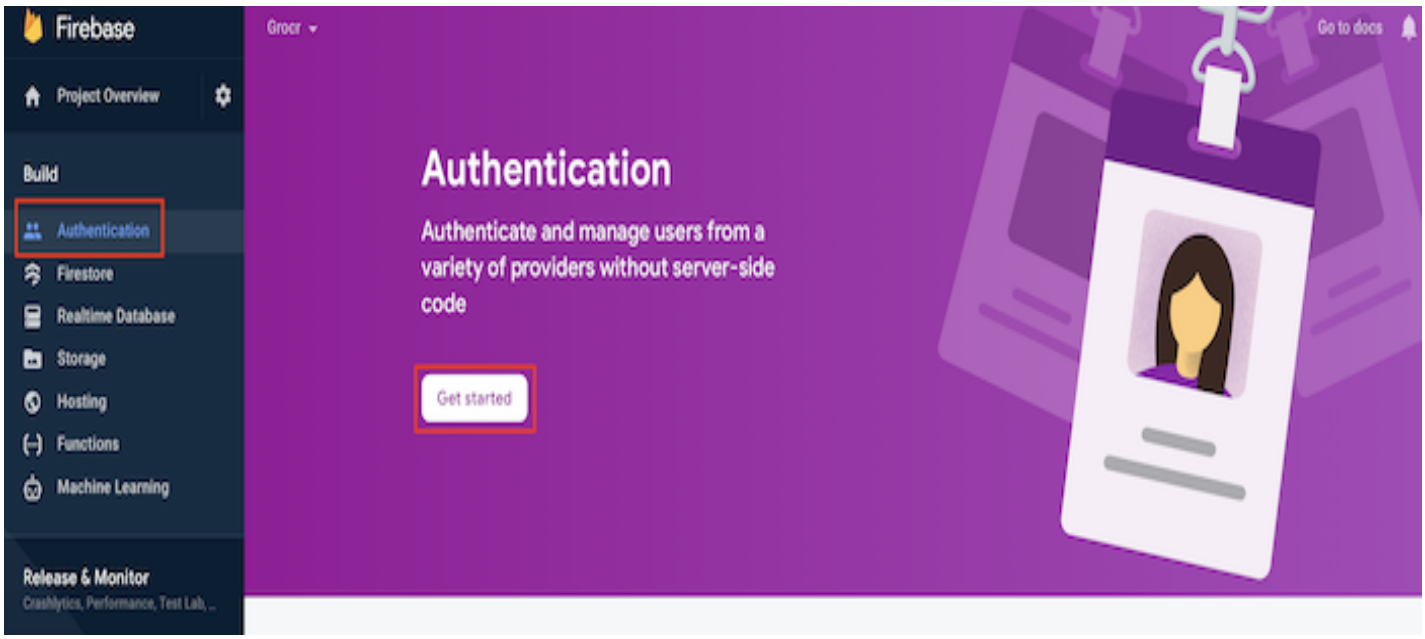
Firebase Authentication is a backend service provided by Google that makes it easy to login users to your app.
It supports different types of authentication such as email and password, anonymous login and phone numbers login.
It can also authenticate with different providers such as Google, Facebook, and Twitter.

When should we use Firebase Authentication?

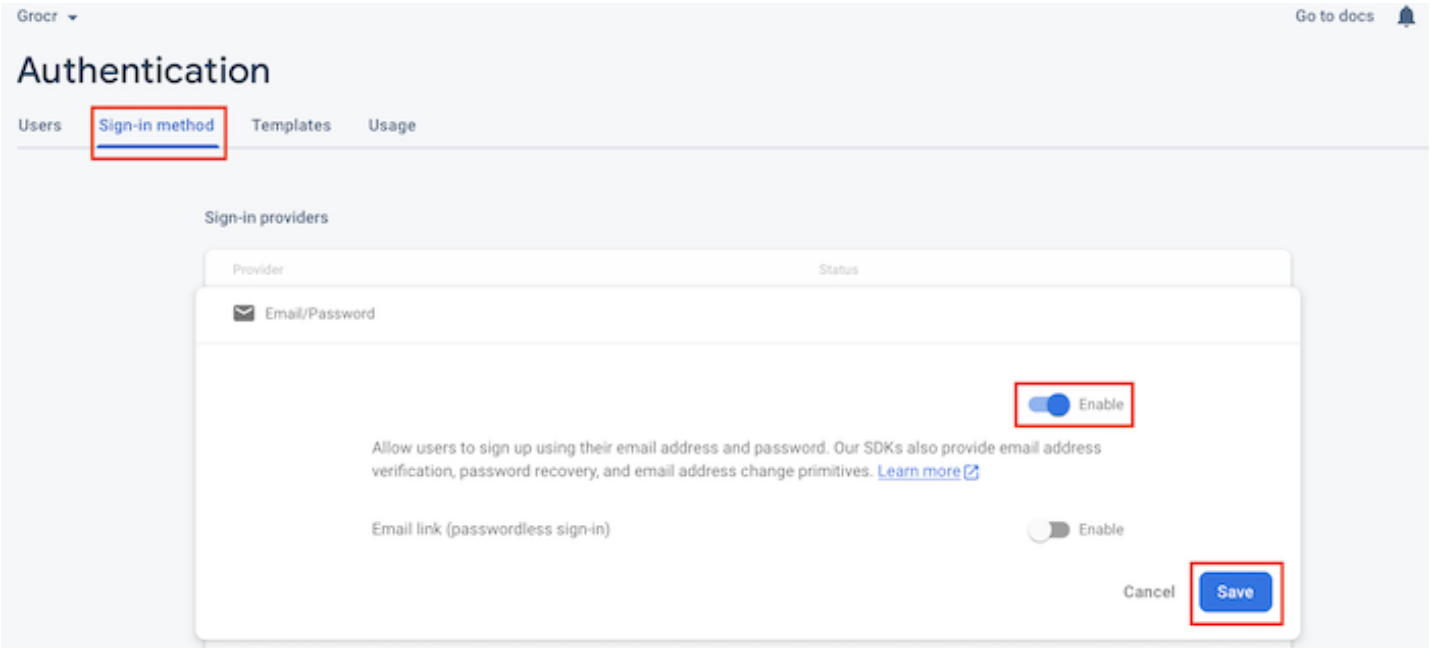
We should use Firebase Authentication anytime the user needs to login or sign up for a mobile application.

Setting up email and password authentication

To enable email and password authentication, go to Firebase Dashboard. Click "*Authentication"* and then click *"Get Started."*



Select “*Sign-in-method.”* In Sign-in-providers, select the *Email/Password* row.
Toggle *Enable* and click *Save.*



Now we have authentication set up in our Firebase Dashboard.

Add Firebase Dependencies to XCode project

Lets go to our XCode project and add the dependencies for Firebase Authentication using CocoaPods.

First let's open our Podfile.

- To create a Podfile.
  - Open terminal
  - Navigate to your project location using *“cd”*
    For example if your project is in a folder called “Projects” in Desktop.
    Then to navigate to my projects I will type
    cd Desktop/Projects
  - Type *pod init*
  - Type *open podfile*
  - In your Podfile type *pod 'Firebase/Auth'*
  - Click File -> Save
  - Go back to terminal and type *pod install*

After typing ***pod install.*** You should see all the dependencies being added to your project. From now on we are going to open our .**xcworkspace.** We will no longer use .xcodeproj.

```
MacBook-Air:~ Danny$ cd Desktop/CodingDojo/sample/
MacBook-Air:sample Danny$ pod init
MacBook-Air:sample Danny$ open podfile
MacBook-Air:sample Danny$ pod install
Analyzing dependencies
Downloading dependencies
Installing Firebase (7.0.0)
Installing FirebaseAuth (7.0.0)
Installing FirebaseCore (7.0.0)
Installing FirebaseCoreDiagnostics (7.0.0)
Installing GTMSessionFetcher (1.5.0)
Installing GoogleDataTransport (8.0.0)
Installing GoogleUtilities (7.0.0)
Installing PromisesObjC (1.2.11)
Installing nanopb (2.30906.0)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `sample.xcworkspace` for this project from
now on.
Pod installation complete! There is 1 dependency from the Podfile and 9 total pods installed.

[!] Automatically assigning platform `iOS` with version `14.4` on target `sample` because no pl
atform was specified. Please specify a platform for this target in your Podfile. See `https://g
uides.cocoapods.org/syntax/podfile.html#platform`.

[!] Your project does not explicitly specify the CocoaPods master specs repo. Since CDN is now
used as the default, you may safely remove it from your repos directory via `pod repo remove ma
ster`. To suppress this warning please add `warn_for_unused_master_specs_repo => false` to your
 Podfile.
MacBook-Air:sample Danny$ _
```

The Podfile should look something like this

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'sample' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for sample
  pod 'Firebase/Auth'

  target 'sampleTests' do
    inherit! :search_paths
    # Pods for testing
  end

  target 'sampleUITests' do
    # Pods for testing
  end

end
```

In the image above, we are just adding **pod 'Firebase/Auth'**

Signing up a user

Now we have set up the dependencies using CocoaPods and enabled email and password authentication in our Firebase Dashboard.

We will now go over the code for signing up a user to Firebase

```
Auth.auth().createUser(withEmail: String, password: String) {  (AuthDataResult?, Error?) in
        // code ..
}
```

What is Auth?

- Auth manages Authentication for Firebase apps.

What is auth() ?

- auth() gets the auth object for the default Firebase apps.

What is createUser() ?

- createUser() creates and, on success, signs in a user with the given email address and password.
- Keep in mind that createUser() takes in 2 parameters: email and password. Both of these data type is a String.

What is AuthDataResult?

- AuthDataResult is a helper object that contains the result of a successful sign-in, link and reauthenticate action.

We will now add code in our *ViewController.swift* file to create an account.

Below I created a function that has the logic to create an account.

```swift
func createAccount() {
    Auth.auth().createUser(withEmail: emailTextField.text!, password: passwordTextField.text!) {
        (authResult: AuthDataResult?, error: Error?) in
        if let error = error {
            print("Something went wrong. Cannot create an account. Error description: \(error.localizedDescription)")
        } else {
            print("User \(self.emailTextField.text!) successfully created an account")
        }
    }
}
```

Log in a user

We will now go over the code for login in a user to Firebase.

```
Auth.auth().signIn(withEmail: String, password: String) {  (AuthDataResult?, Error?) in
        // code ..
}
```

As you can see the code is very similar to the previous. The only difference is that it has a function called signIn()

What is signIn() ?

- In this example signIn() sign in a user with an email and password

What is AuthDataResult?

- AuthDataResult is a helper object that contains the result of a successful sign-in, link and reauthenticate action.

We will now add code in our *ViewController.swift* file to login a user.

Below I created a function that has the logic to login a user.

```
func loginUser() {
    Auth.auth().signIn(withEmail: emailTextField.text!, password: passwordTextField.text!) {
        (authResult: AuthDataResult?, error: Error?) in
        if let error = error, authResult == nil {
            print("Unable to login user \(self.emailTextField.text!) Error description: \(error.localizedDescription)")
        } else {
            print("User \(self.emailTextField.text!) successfully log in")
        }
    }
}
```

Authentication State

What is Authentication state?

- Authentication state is an observer where Firebase checks the current user login state
- Authentication state can either have 2 status
  - Current user is logged in.
  - Current user is not logged in.

Why is Authentication state important?

- During the development cycle it is best practice to make the user have a great experience using your mobile application.
- In this case if the user closes your app and tries to re open your app. We should check if the current user is still logged in. If the current user is still logged in we should skip the login page and proceed to the home page.

Lets look over the code for the authentication state

```
Auth.auth().addStateDidChangeListener() {  (Auth, User?) in
        // code ..
}
```

What is addStateDidChangeListener() ?

- addStateDidChangeListener() is a block that listens to any changes to the authentication state of a user.

```swift
func checkAuthenticationState() {
    Auth.auth().addStateDidChangeListener {
        (auth: Auth, user: User?) in
        if user == nil {  // current user does not exist
            self.navigationController?.popToRootViewController(animated: true)
        } else {
            self.performSegue(withIdentifier: "HomeSegue", sender: nil)
        }
    }
}
```

You can also get the current user by using the currentUser property.

```
if Auth.auth().currentUser != nil {
    print("User: \(Auth.auth().currentUser?.email) is currently signed in")
} else {
    print("No user is signed in")
}
```