

Dependency Management



Dependency Managers

A *dependency manager* is a tool that makes it easy to add, remove, update, and manage third-party dependencies used by your app.

- For example, instead of reinventing your own networking library, you could simply pull in AlamoFire using a dependency manager.
 - You can specify the exact version you want to use, or even a range of acceptable versions.
 - Even if AlamoFire gets updated, your app can continue to use the older version until / unless you're ready to update it.

CocoaPods

CocoaPods is a third-party dependency manager designed specifically for Swift and Objective-C projects.

- contains over 30,000 libraries
- used in over 2,000,000 apps
- built on top of Ruby, which ships with all recent versions of Mac OS X
- *pods* are libraries or frameworks added to your project via CocoaPods

CocoaPods is still a very useful tool for adding and managing dependencies in a project, but there is another option that is starting to replace it.

Swift Package Manager

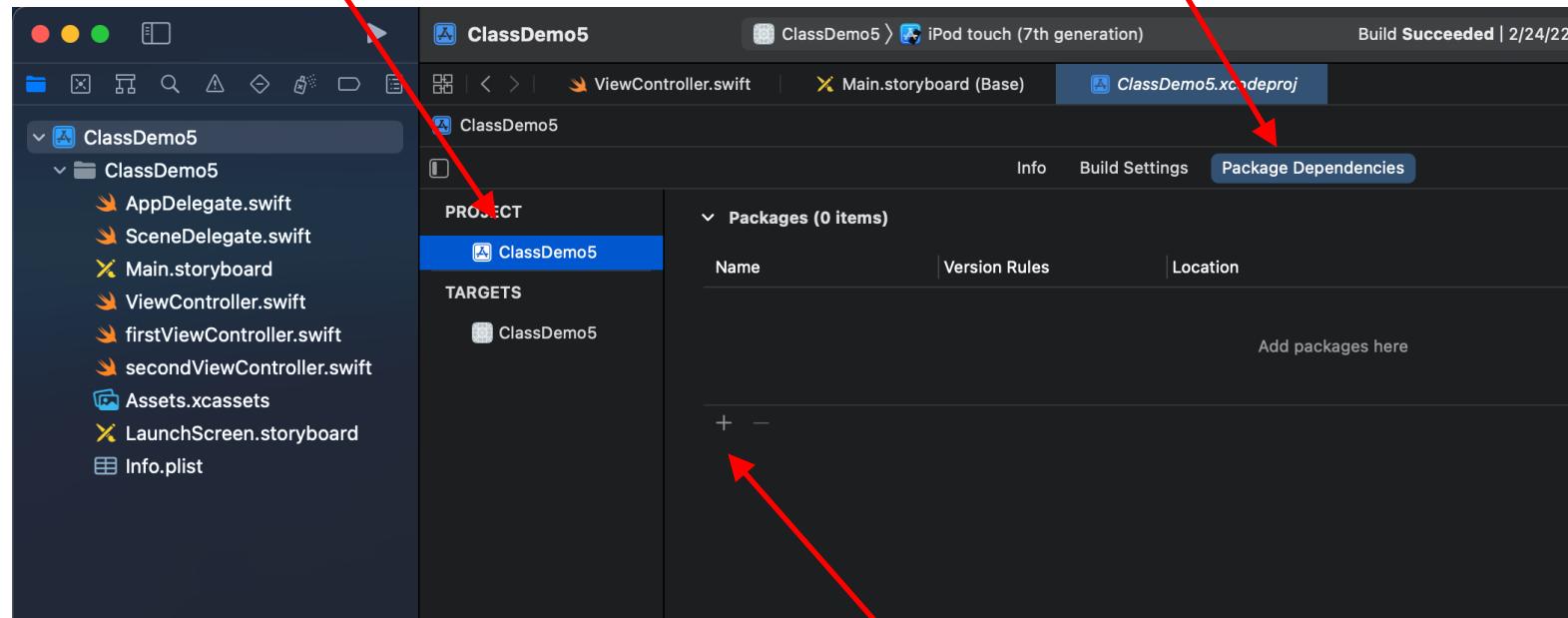
The *Swift Package Manager* (SwiftPM) has become the easier option for iOS dependency management.

- With it, you no longer have to rely on a third party dependency manager such as CocoaPods.
- It's been around since Swift 3.0, but only available for a limited collection of packages
- Apple significantly improved support for it in Swift 5 / Xcode 11
- A *package* is simply a collection of Swift source code files, plus a metadata ("manifest") file called `Package.swift` that defines various properties about the package, such as the name, code dependencies, etc.

Adding a package using SwiftPM

To add a package to your project:

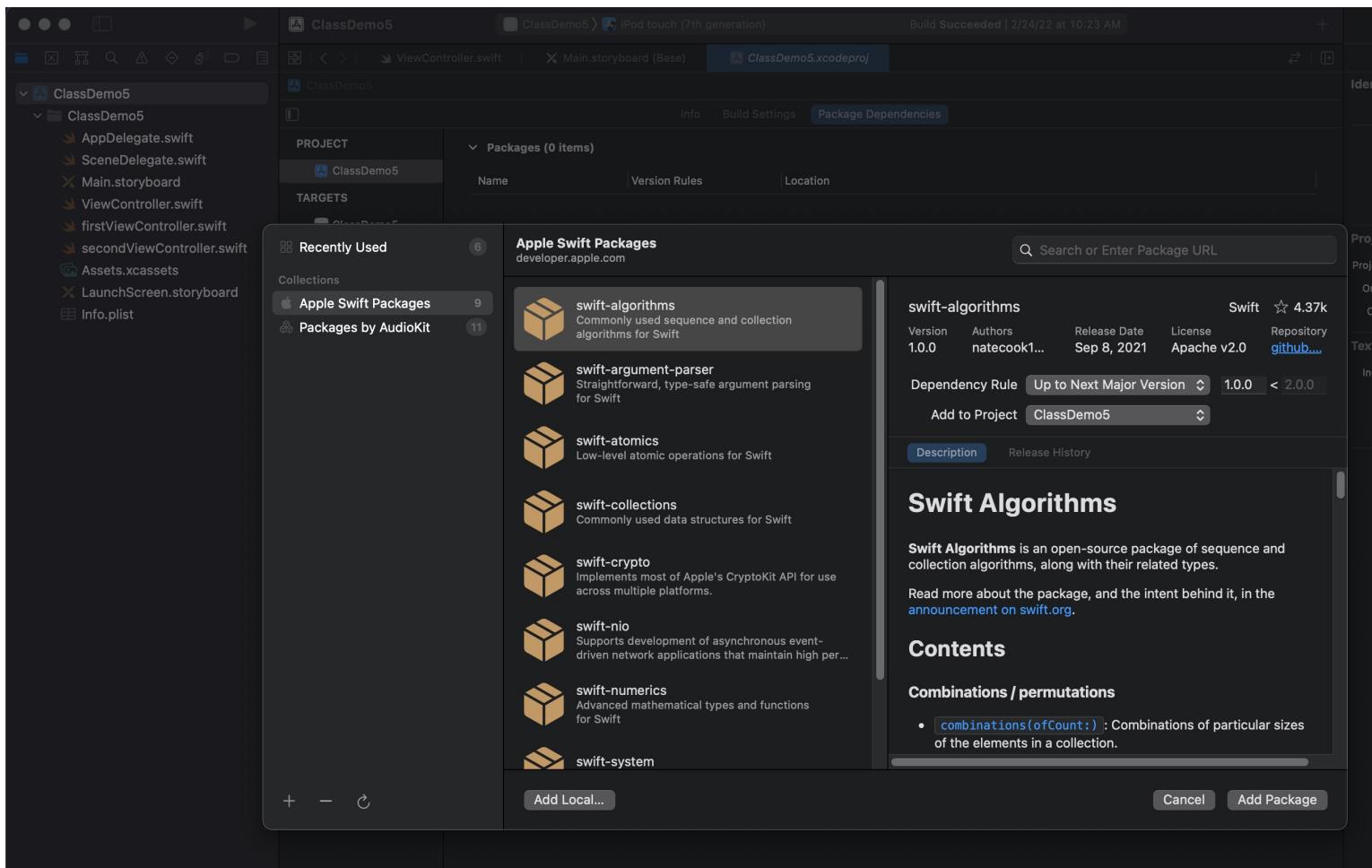
1. Select your project



3. Click the "+" button

Adding a package using SwiftPM (cont.)

This pops up a window where you can either use a recently used package, or enter a URL where the package can be found.



Firebase



What is Firebase?

Firebase is a “Backend-as-a-Service” (BaaS). It is a mobile and web application development platform that manages servers for you so you can focus on your app.

- Real-time database: you can create a database on a server and access it though a Web socket, which is much faster than HTTP.
- File storage: you can save binary files (especially images) securely on Google Cloud Storage
- Authentication: Firebase auth has a built-in email/password authentication system you can use for your app.

What is Firebase? (cont.)

Lots of other stuff, too. . .

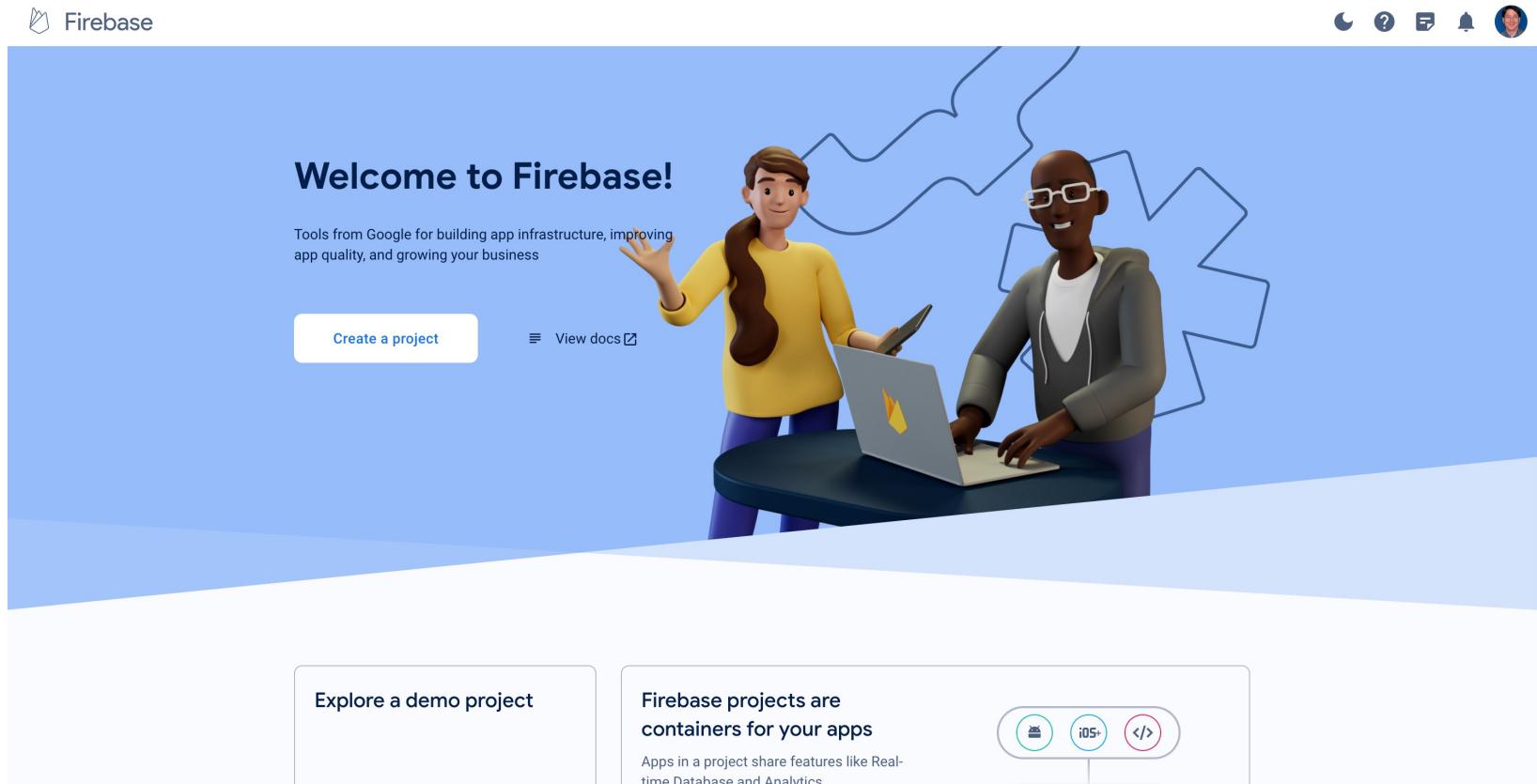
- Hosting
- Analytics
- Cloud Messaging
- A collection of other Google products, including
 - Remote config
 - Test Lab
 - Crash
 - Notifications
 - Dynamic Links
 - AdMob

Setting up Firebase

1. Create a Google account, if you don't already have one. (You can use the same one you have for Gmail, Google Drive, etc.)

Setting up Firebase (cont.)

2. Go to <https://console.firebaseio.google.com>



3. Add Firebase to your project by clicking on “Create project”

Setting up Firebase (cont.)

3. Enter your project name.

X Create a project (Step 1 of 3)

Let's start with a name for
your project ?

Project name

Edit cd-firebase-311eb

Continue

Setting up Firebase (cont.)

4. It will offer to set up Google Analytics. Either accept (and complete an additional screen) or decline and hit “Create Project”. Wait.

×

Create a project (Step 2 of 2)

Google Analytics for your Firebase project

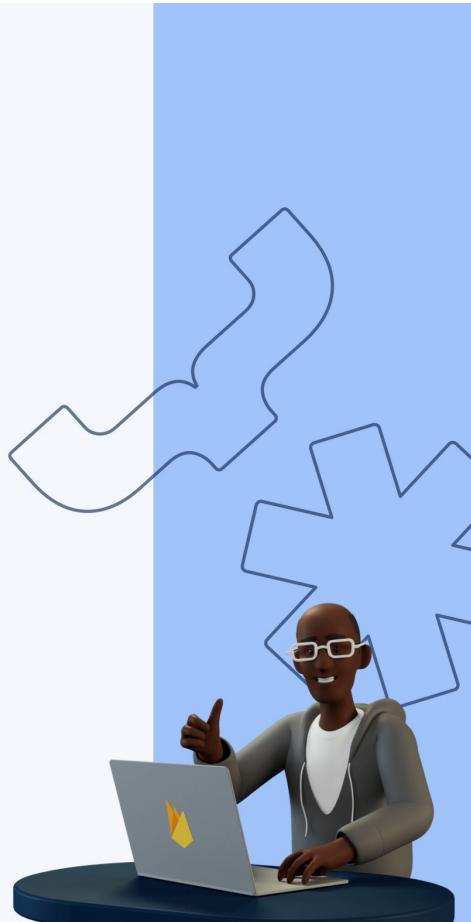
Google Analytics is a free and unlimited analytics solution that enables targeting, reporting, and more in Firebase Crashlytics, Cloud Messaging, In-App Messaging, Remote Config, A/B Testing, and Cloud Functions.

Google Analytics enables:

- ×
- A/B-testing ?
- ×
- Breadcrumb logs in Crashlytics ?
- ×
- User segmentation & targeting across ? Firebase products
- ×
- Event-based Cloud Functions triggers ?
- ×
- Free unlimited reporting ?

Enable Google Analytics for this project
Recommended

[Previous](#) [Create project](#)



Setting up Firebase (cont.)

This takes you to the “Get started” screen. Click on the "iOS" button.

The screenshot shows the Firebase Project Overview screen for a project named 'CD-Firebase'. The left sidebar contains navigation links for 'Project Overview', 'What's new', 'Extensions (NEW)', 'Release Monitoring (NEW)', 'Product categories', 'Build', 'Release & Monitor', 'Analytics', 'Engage', and 'All products'. A 'Customize your nav!' box provides instructions for navigating the console. The main content area features a 'Get started by adding Firebase to your app' message with a 'Spark plan' button. It includes a 'CD-Firebase' logo and two 3D characters, a man and a woman, interacting with a yellow Firebase logo. Below this, there are two cards: 'Authentication' (with an icon of a user ID card and a lock) and 'Cloud Firestore' (with an icon of a stack of servers and a magnifying glass). A 'Store and sync app data in milliseconds' header is present above these cards. The top right of the screen shows a user profile and notification icons.

Setting up Firebase (cont.)

Enter the Bundle ID for your app. (Remember how to find this?) Click on “Register app”.

×

Add Firebase to your Apple app

1 Register app

Apple bundle ID ⑦

App nickname (optional) ⑦

App Store ID (optional) ⑦

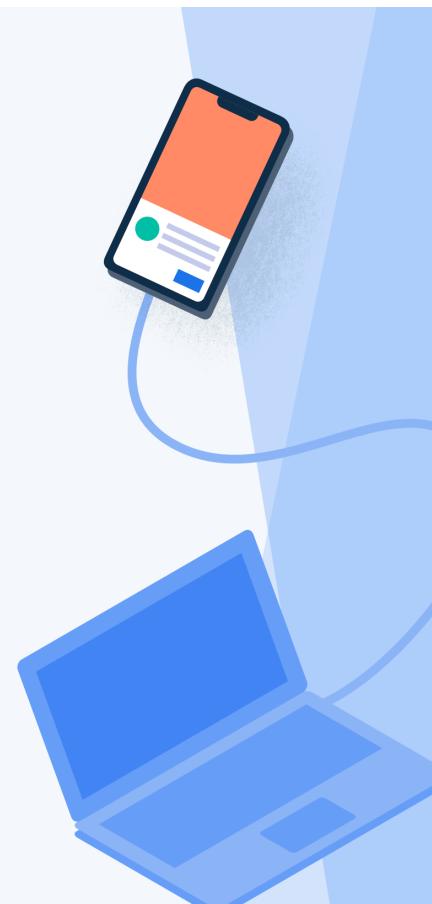
Register app

2 Download config file

3 Add Firebase SDK

4 Add initialization code

5 Next steps



Setting up Firebase (cont.)

This will create a `GoogleService-Info.plist` file. Follow the instructions, and move it to your project in Xcode. (Be sure to check “Copy Items if needed” in Xcode.)

× Add Firebase to your Apple app

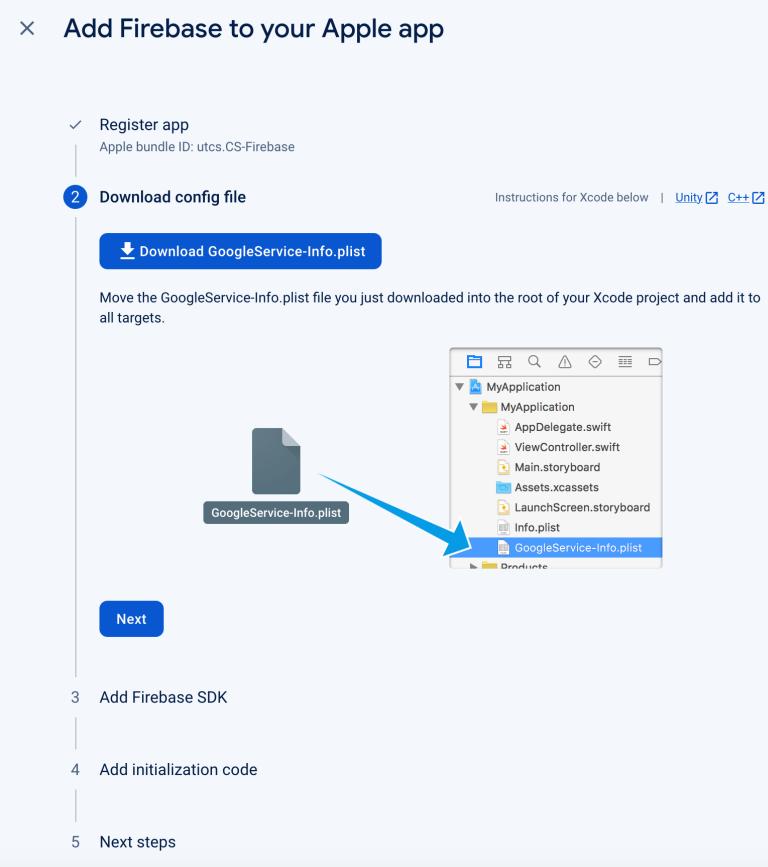
✓ Register app
Apple bundle ID: `utcs.CS-Firebase`

2 Download config file

[Download GoogleService-Info.plist](#)

Instructions for Xcode below | [Unity](#) [C++](#)

Move the `GoogleService-Info.plist` file you just downloaded into the root of your Xcode project and add it to all targets.



Next

3 Add Firebase SDK

4 Add initialization code

5 Next steps



Setting up Firebase (cont.)

Clicking “Next” will give you the instructions on how to add the Firebase SDK to your project. By default, it will show you the instructions for Swift Package Manager, but you can switch it to CocoaPods if you need to.

× Add Firebase to your Apple app

✓ Register app
Apple bundle ID: utcs.CS-Firebase

✗ Download config file

3 Add Firebase SDK [CocoaPods](#) | [Download ZIP](#) | [Unity](#) | [C++](#)

Use [Swift Package Manager](#) to install and manage Firebase dependencies.

1. In Xcode, with your app project open, navigate to **File > Add Packages**
2. When prompted, enter the Firebase iOS SDK repository URL:
`https://github.com/firebase/firebase-ios-sdk`
3. Select the SDK version that you want to use.
We recommend using the default (latest) SDK version, but you can use an older version, if needed.
4. Choose the Firebase libraries that you want to use.

After you click **Finish**, Xcode will automatically begin resolving and downloading your dependencies in the background.

[Previous](#) [Next](#)

4 Add initialization code

5 Next steps



Setting up Firebase (cont.)

Select Swift (or SwiftUI if you're using that). Click “Next”. On the next screen, click “Continue to console”, and you're done! Build and run your app.

[X](#) Add Firebase to your Apple app

✓ Register app
Apple bundle ID: utcs.CS-Firebase

✍ Download config file

✍ Add Firebase SDK

4 Add initialization code

To connect Firebase when your app starts up, add the initialization code below to your app's main entry point.

SwiftUI Swift Objective-C

```
import UIKit
import FirebaseCore

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

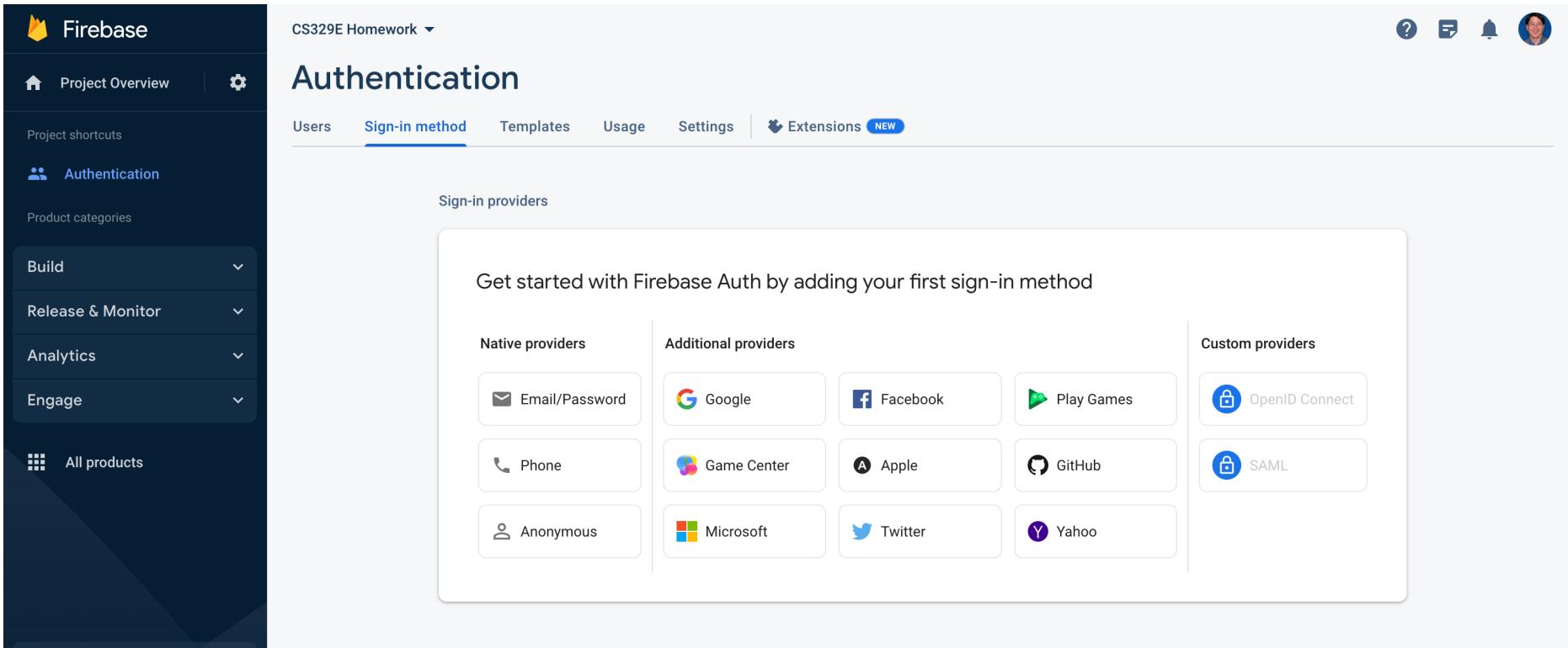
    func application(_ application: UIApplication,
                    didFinishLaunchingWithOptions launchOptions:
                        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        FirebaseApp.configure()
        return true
    }
}
```

Previous [Next](#)



Authentication by email address

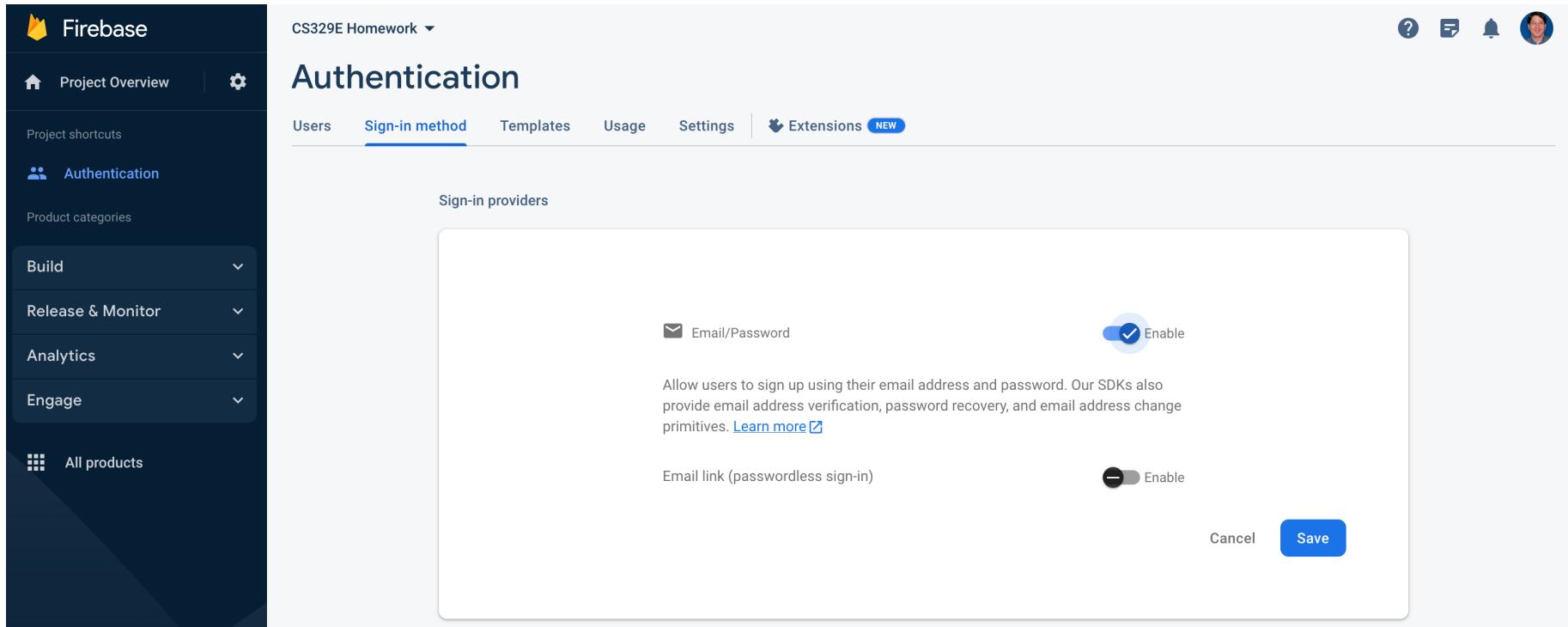
Find “Authentication” and either click on “Set up sign-in method” or click on the “Sign-in method” tab. Select "Email/password".



The screenshot shows the Firebase console interface. On the left is a dark sidebar with the Firebase logo, a 'Project Overview' button, a gear icon for settings, 'Project shortcuts', 'Authentication' (which is highlighted in blue), and 'Product categories'. Below these are sections for 'Build', 'Release & Monitor', 'Analytics', 'Engage', and 'All products'. The main content area has a header 'CS329E Homework' with a dropdown arrow, and a navigation bar with 'Users', 'Sign-in method' (which is underlined in blue), 'Templates', 'Usage', 'Settings', and 'Extensions (NEW)'. The 'Sign-in method' tab is currently active. The main content is titled 'Authentication' and features a section titled 'Sign-in providers' with a sub-section 'Get started with Firebase Auth by adding your first sign-in method'. It shows three categories: 'Native providers' (Email/Password, Phone, Anonymous), 'Additional providers' (Google, Facebook, Play Games, Game Center, Apple, GitHub, Microsoft, Twitter, Yahoo), and 'Custom providers' (OpenID Connect, SAML). The 'Sign-in method' tab is highlighted in blue, indicating it is the active tab.

Authentication by email address (cont.)

Select “Enable” and then “Save”.



The screenshot shows the Firebase console's Authentication page for the project 'CS329E Homework'. The 'Sign-in method' tab is selected. Under 'Sign-in providers', the 'Email/Password' provider is listed with its 'Enable' switch turned on. A description below explains that it allows users to sign up using their email address and password, and provides primitives for email address verification, password recovery, and email address change. The 'Email link (passwordless sign-in)' provider is listed with its 'Enable' switch turned off. At the bottom right are 'Cancel' and 'Save' buttons.

CS329E Homework ▾

Authentication

Users Sign-in method Templates Usage Settings Extensions NEW

Sign-in providers

✉ Email/Password Enable

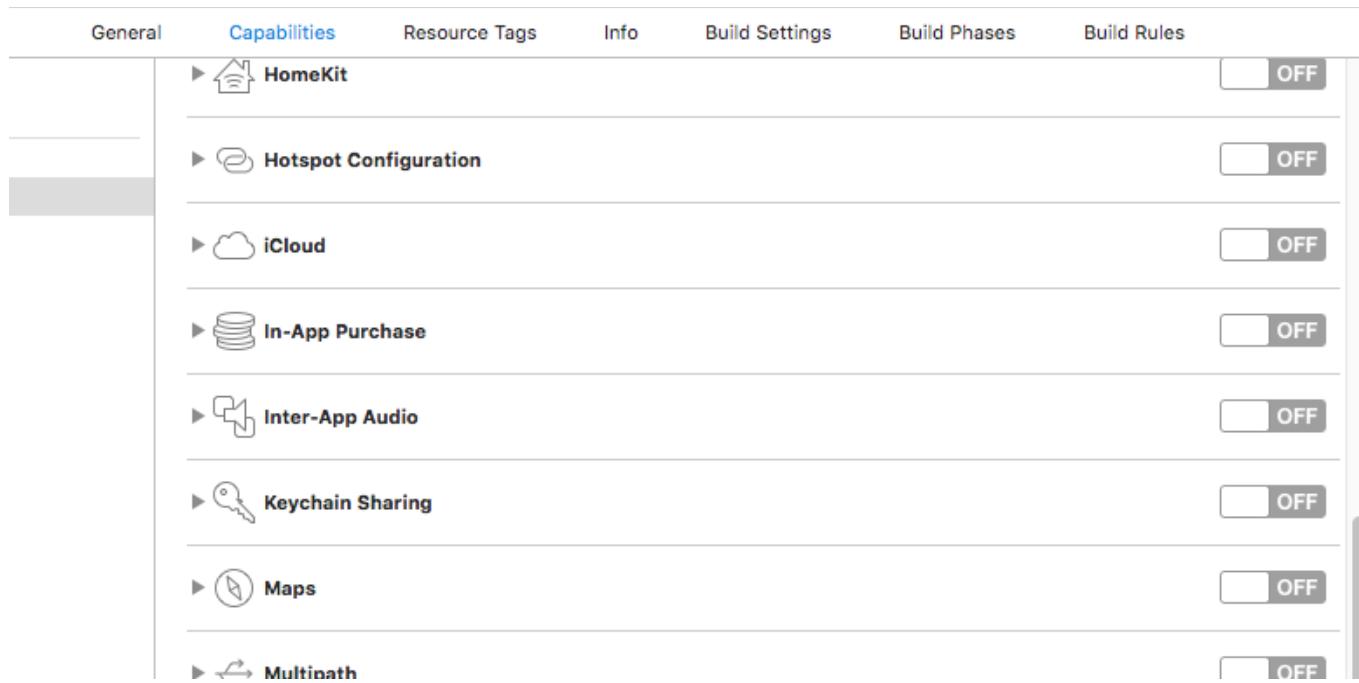
Allow users to sign up using their email address and password. Our SDKs also provide email address verification, password recovery, and email address change primitives. [Learn more](#)

Email link (passwordless sign-in) Enable

Cancel Save

Authentication by email address (cont.)

Since Firebase stores credentials in the keychain, in Xcode, go to your project properties, and under the Capabilities tab, toggle “Keychain Sharing” on.



Now you can authenticate users for your app using their email and password.

Authenticating by email address (cont.)

Use `Auth.auth().createUser` to register a new user into the authentication database. It takes three arguments:

- `withEmail`: a text string, ideally the `text` parameter from a `TextField` object.
- `password`: a similar text string.
- a closure specifying what to do once the user is created.

Authenticating by email address (cont.)

If there are no errors with the email address or password, use `Auth.auth().signIn` to authenticate an email address / password pair. It takes two arguments:

- `withEmail`: a text string, ideally the `text` parameter from a `TextField` object.
- `password`: a similar text string.

Note: Firebase expects both the userid and password and be at least 7 characters long!

Authenticating by email address (cont.)

If we want to immediately log the user in once the account is created, we put the call to `signIn()` in the closure of `createUser()`:

```
// Assume uidField and pwdField are outlets of text fields

Auth.auth().createUser(
    withEmail: uidField.text!,
    password: pwdField.text!) { user, error in

    if error == nil {
        Auth.auth().signIn(
            withEmail: uidField.text!,
            password: pwdField.text!)
    }
}
```

Firestore



Using Firestore to store data

- Set up the Firebase server as shown earlier.
- When you add packages, include Firestore.
- In your project:

```
import FirebaseCore
import FirebaseFirestore
```

- On the Firebase console, go to "Cloud Firestore" to set up a database. Define:
 - at least one *collection*: kind of like defining a class
 - at least one *document* within that collection: kind of like defining an object within a class
 - at least one *field* in each document: kind of like defining a property that all objects have

Firestore methods

`addDocument()` inserts a new item into a specified collection. It takes two arguments:

- `data`: a dictionary containing field names and values
- a *closure* that takes one argument, a Boolean that tells you whether an error occurred.

```
let db = Firestore.firestore()
var ref: DocumentReference? = nil

ref = db.collection("colName").addDocument(data:
    ["fieldName": valueToAssign]) { err in
    if let err = err {
        < code to run if an error occurs >
    } else {
        < code to run if no error occurs >
    }
}
```

Firestore methods

`getDocuments()` retrieves items from a specified collection. It takes one argument:

- a *closure* that takes two arguments, a *querySnapshot* that contains the items, and a Boolean that tells you whether an error occurred.

```
let db = Firestore.firestore()
var ref: DocumentReference? = nil

ref = db.collection("colName").getDocuments() {
    (querySnapshot, err) in
    if let err = err {
        < code to run if an error occurs >
    } else {
        < code to run if no error occurs >
    }
}
```