THE UNIVERSITY OF CHICAGO

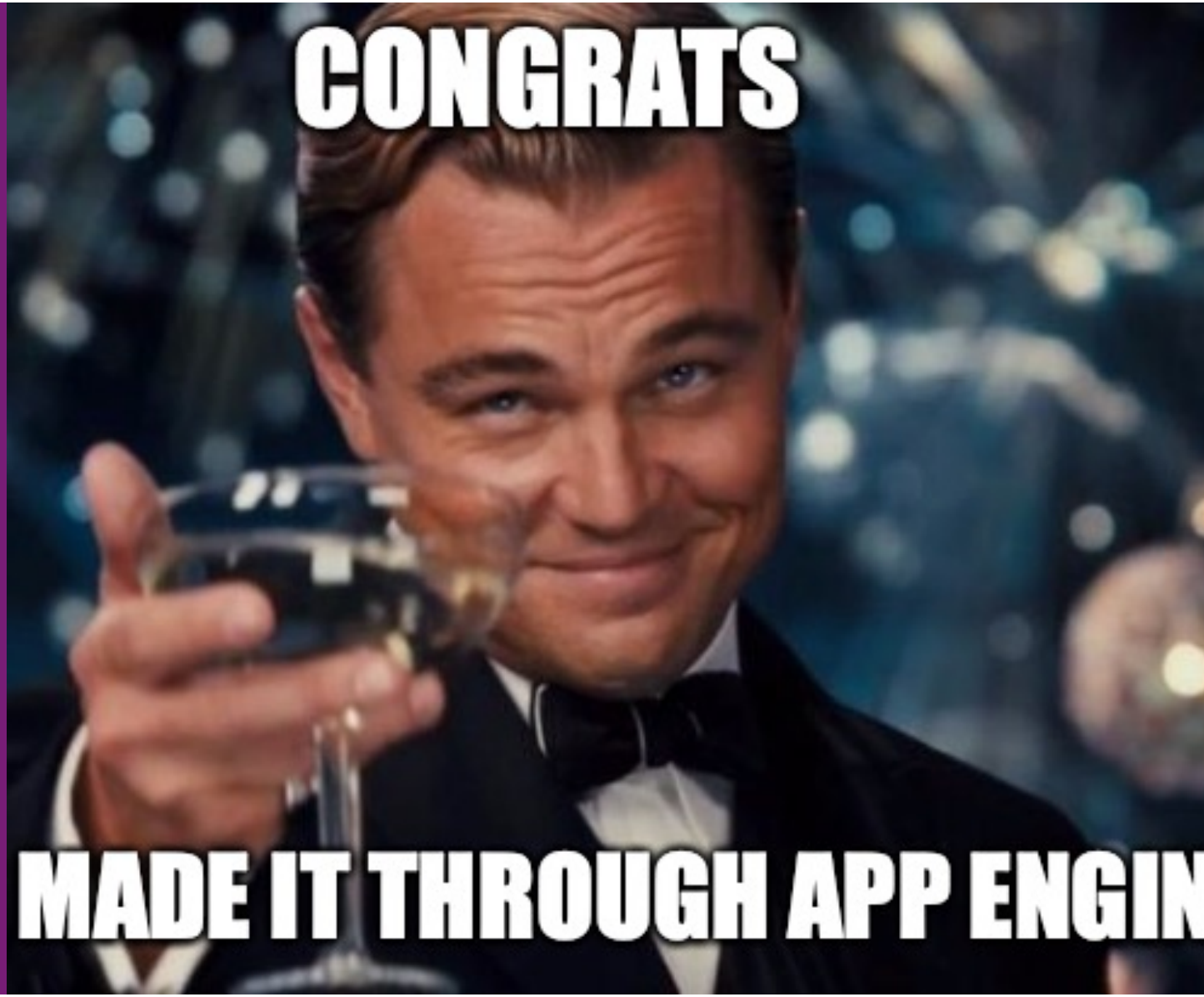
MPCS 51033 • AUTUMN 2019 • SESSION 4

BACKENDS FOR MOBILE APPLICATIONS

CLASS NEWS

CLASS NEWS

- Homework

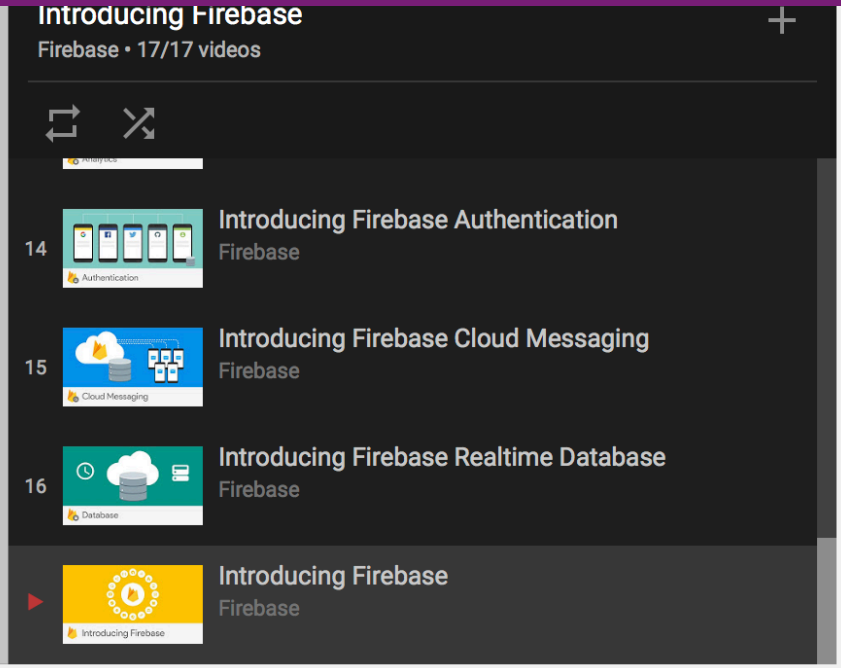


FIREBASE🔥

FIREBASE

- https://youtu.be/O17OWyx08Cg?list=PLI-K7zZEsYLmOF_07IayrTntevxtbUxDL

FIREBASE

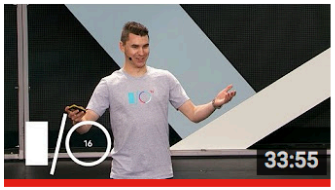


Introducing Firebase

Firebase

Subscribed | 56,536

287,389 views



Zero to App: Develop with Firebase - Google I/O 2016

Firebase

119,707 views

FIREBASE

- Firebase pricing structure

Products	Spark Plan Generous limits for hobbyists **Free**	Flame Plan Fixed pricing for growing apps **$25/month**	Blaze Plan Calculate pricing for apps at scale **Pay as you go** ✓ Free usage from Spark plan included*
A/B Testing	Free		
Analytics	Free		
App Indexing	Free		
Authentication			
Phone Auth - US, Canada, and India	10k/month	10k/month	$0.01/verification
Phone Auth - All other countries	10k/month	10k/month	$0.06/verification
Other Authentication services	✓	✓	✓
Cloud Firestore			
Stored data	1 GiB total	2.5 GiB total	$0.18/GiB
Network egress	10GiB/month	20GiB/month	Google Cloud pricing
Document writes	20K/day	100K/day	$0.18/100K
Document reads	50K/day	250K/day	$0.06/100K
Document deletes	20K/day	100K/day	$0.02/100K

FIREBASE

REDWHALE STARTUPS VENTURE CAPITAL TECHNOLOGY GROWTH ABOUT & EDITORS DESK | SUB

Firebase

Don't Get Service Trapped!

Firebase Costs Increased by 7,000%!

HomeAutomation Follow
May 17, 2017 · 10 min read

Update (05/17/2017 8:50 AM):

> *I am happy to say that I received a phone call promptly this morning from the executive team at Firebase. They professed their sincere apology for the situation and explained what happened specifically with the account.*
>
> *While we haven't reached a final conclusion on everything, it would appear the specifics of the article have been heard. Many of the complaints (namely the metrics, among other things) were outlined as things they would put a focus on to improve upon quickly.*
>
> *I will make sure to keep it updated as any relevant information is provided.*

Update (05/17/2017 1:00 PM):

> *Upon further discussions, credits have been taken care of and a plan has been set to address the concerns aggressively. Andrew (Founder of Firebase) and his*

Image by Aphinya Dechalert. Titled: Read/Write Monster Feasting on Firebase Bill

How NOT to get a $30k bill from Firebase

the secret is in the architecture

Aphinya Dechalert Follow
May 27 · 5 min read ★

July last year, a crowd funding campaign went viral in Colombia. It was all good and dandy in the first 48 hours. They managed to reach over 2 million sessions and over 20 million page views—with a website that stayed fully functional without a hitch.

Until they saw the bill.

HACKERNOON

AI Crypto Coding Futurism Startups About Community Sponsors

Be seen in a new tech job.

Nicolas Contreras V
@mi500

How we spent 30k USD in Firebase in less than 72 hours

October 14th 2019

TWEET THIS

aVacaPorDeLaCalle became the largest crowdfunding campaign in Colom
collecting 3 times more than the previous record so far in only two days! I
o became one of the biggest political crowdfunding campaign in history.

crowdfunding, alternativa hasta para la financiación lectoral

colombia existen 13 plataformas de financiación colectiva online
ancisco Rincón - frincon@larepublica.com.co...
w.larepublica.co

IG SUCCESS FOR VAKI

48 hours after the campaign was released, we had reached many record
e campaign collected 3 times more than the previous record in Colombia
at time. We had reached more than 2 million sessions, more than 20 milli
ages visited and received more than 15 thousand supports. This averages
thousand users active on the site in average and collecting more than 20 s
orts per minute.

Lessons learnt (the hard way) using Firebase RealTime Database

Pablo A. Martínez Follow
Feb 24, 2018 · 8 min read ★

Firebase
Realtime Database

TL;DR

Last year at GreenLionSoft we decided to progressively ditch various of the goodies provided by AWS and go "all in" into the Google's Firebase suite.

In this article I want you to learn some critical tips to avoid the errors we committed while integrating Firebase Realtime Database that resulted in a unexpected monthly bill of more than **1.000€**.

. . .

FIREBASE IN ACTION

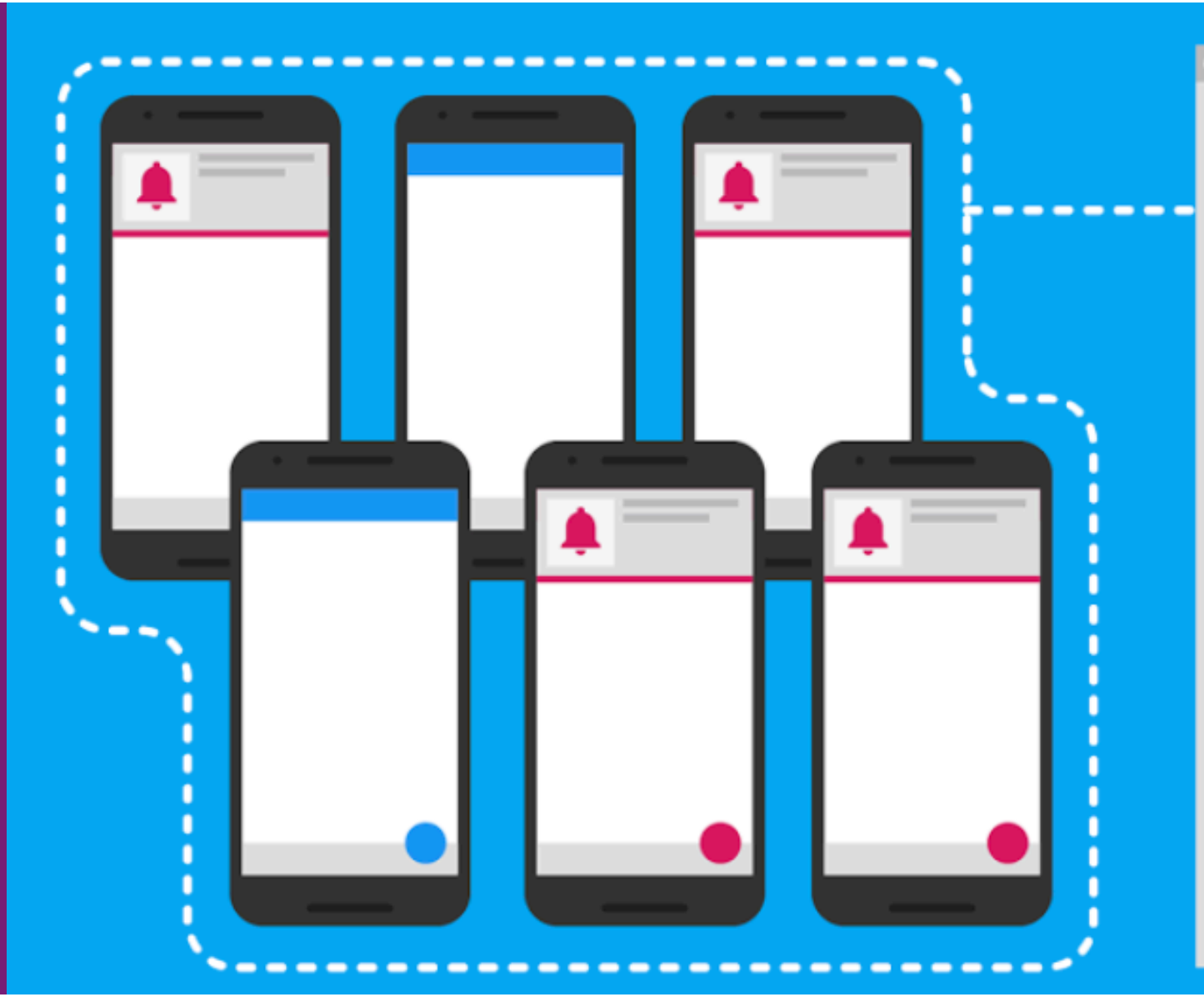
PREREQUISTES FOR FIREBASE

- Before you begin, you need a few things set up in your environment:
  - Xcode 7.0 or later
  - An Xcode project targeting iOS 7 or above
  - The bundle identifier of your app
  - CocoaPods 1.0.0 or later

FIREBASE IN ACTION

PREREQUISTES FOR FIREBASE

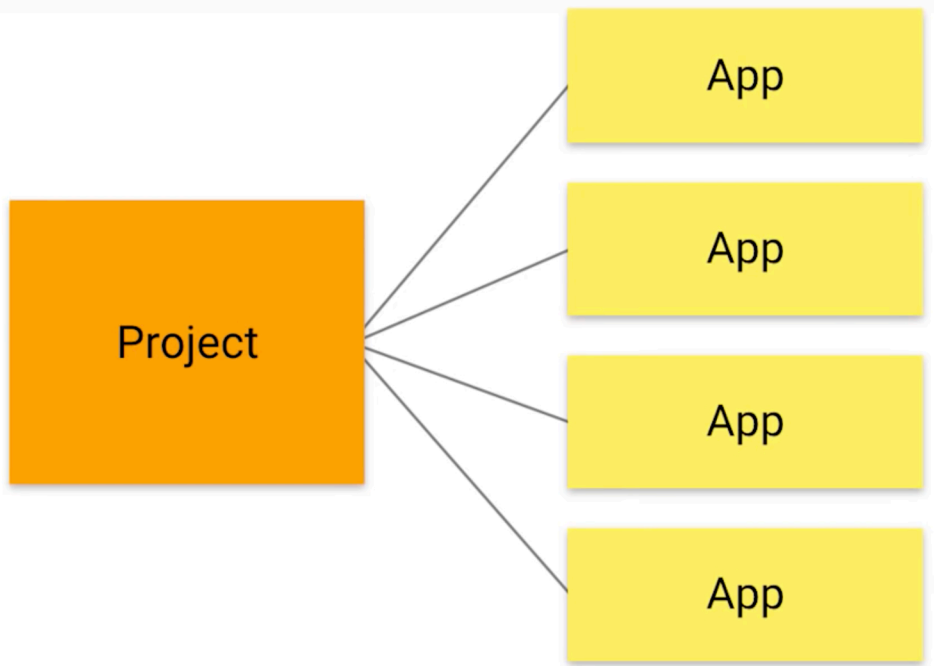
- For Cloud Messaging
  - A physical iOS device
  - APNs certificate with Push Notifications enabled
  - In Xcode, enable Push Notifications in App > Capabilities



FIREBASE IN ACTION

PROJECTS VS. APPS

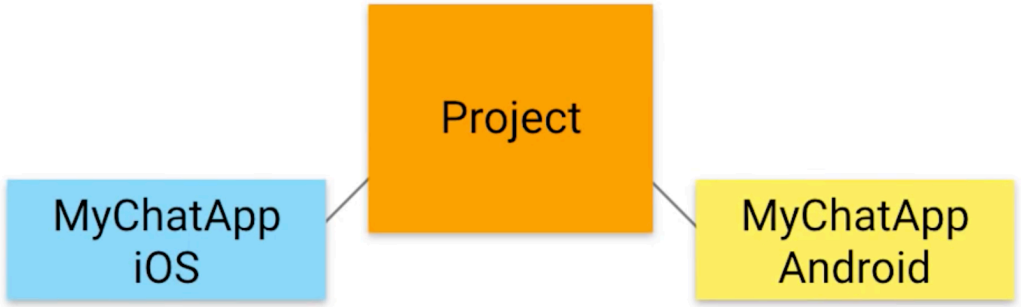
- Project consists of 1+ apps
- Apps all share the same backend



FIREBASE IN ACTION

PROJECTS VS. APPS

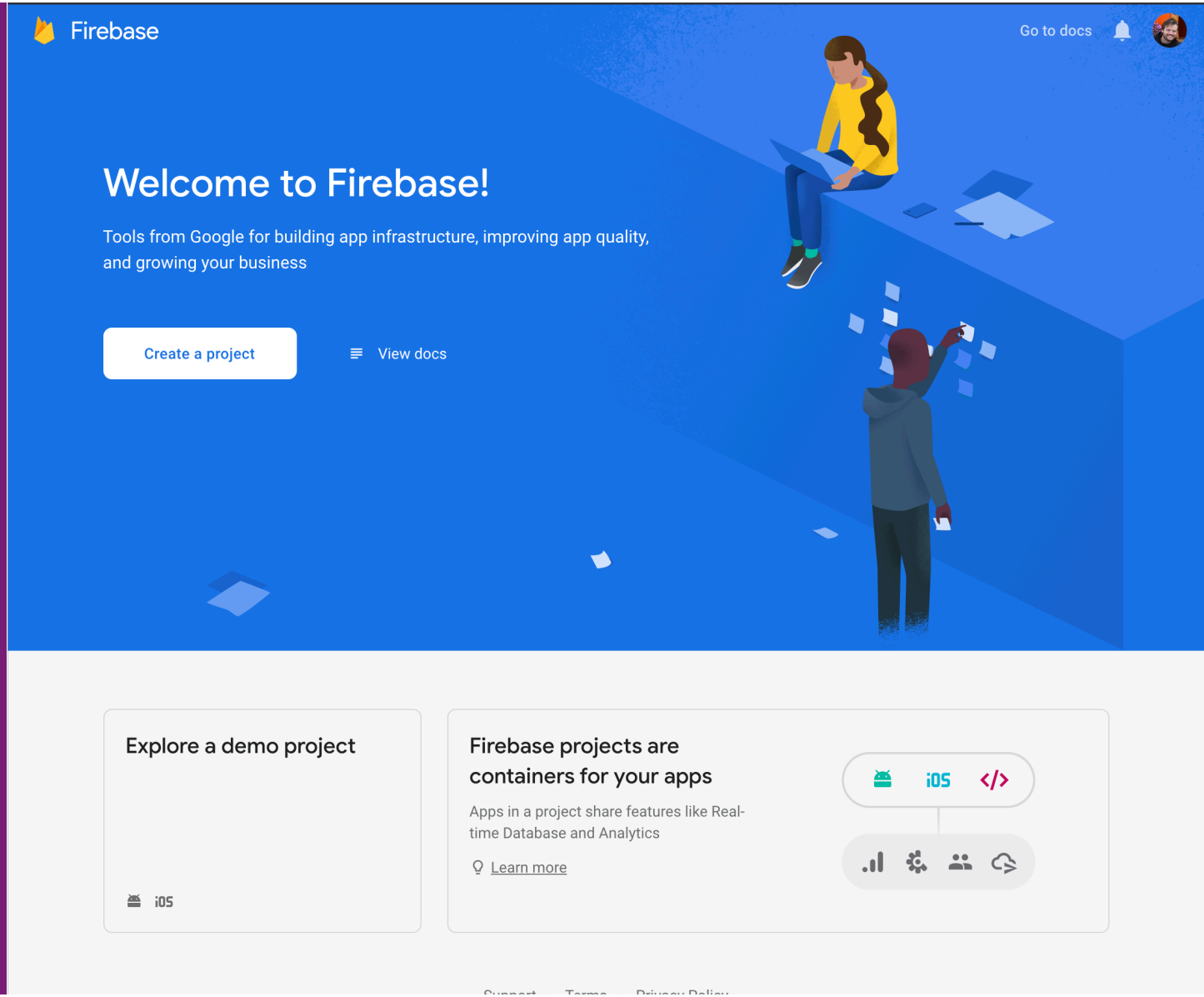
- Cross platform app
  - iOS and Android would be different apps in the same project
  - Access same data from different platforms
  - Send messages to all apps



FIREBASE IN ACTION

PROJECTS VS. APPS

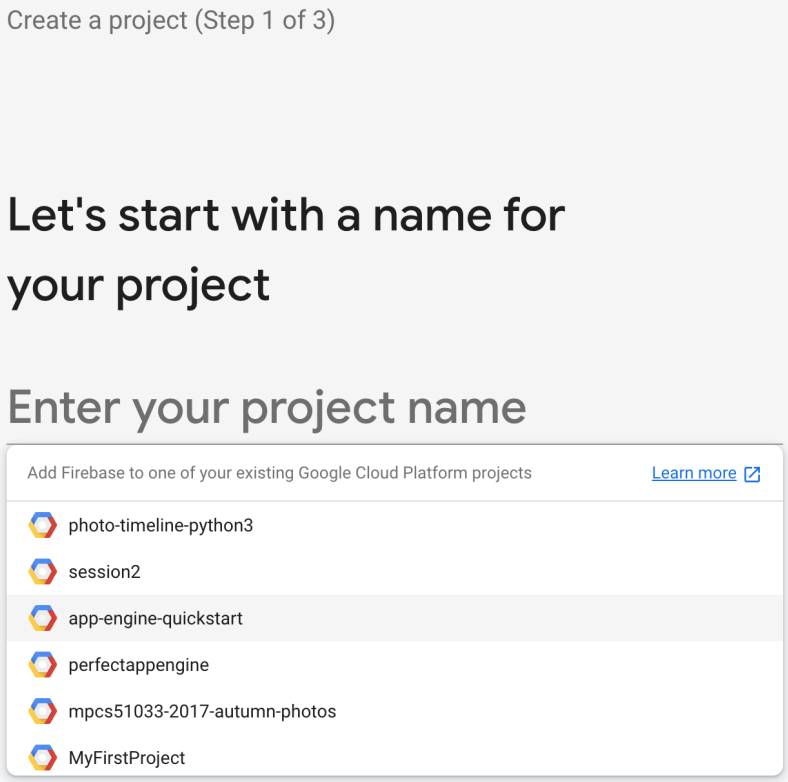
- Create a Firebase project



FIREBASE IN ACTION

PROJECTS VS. APPS

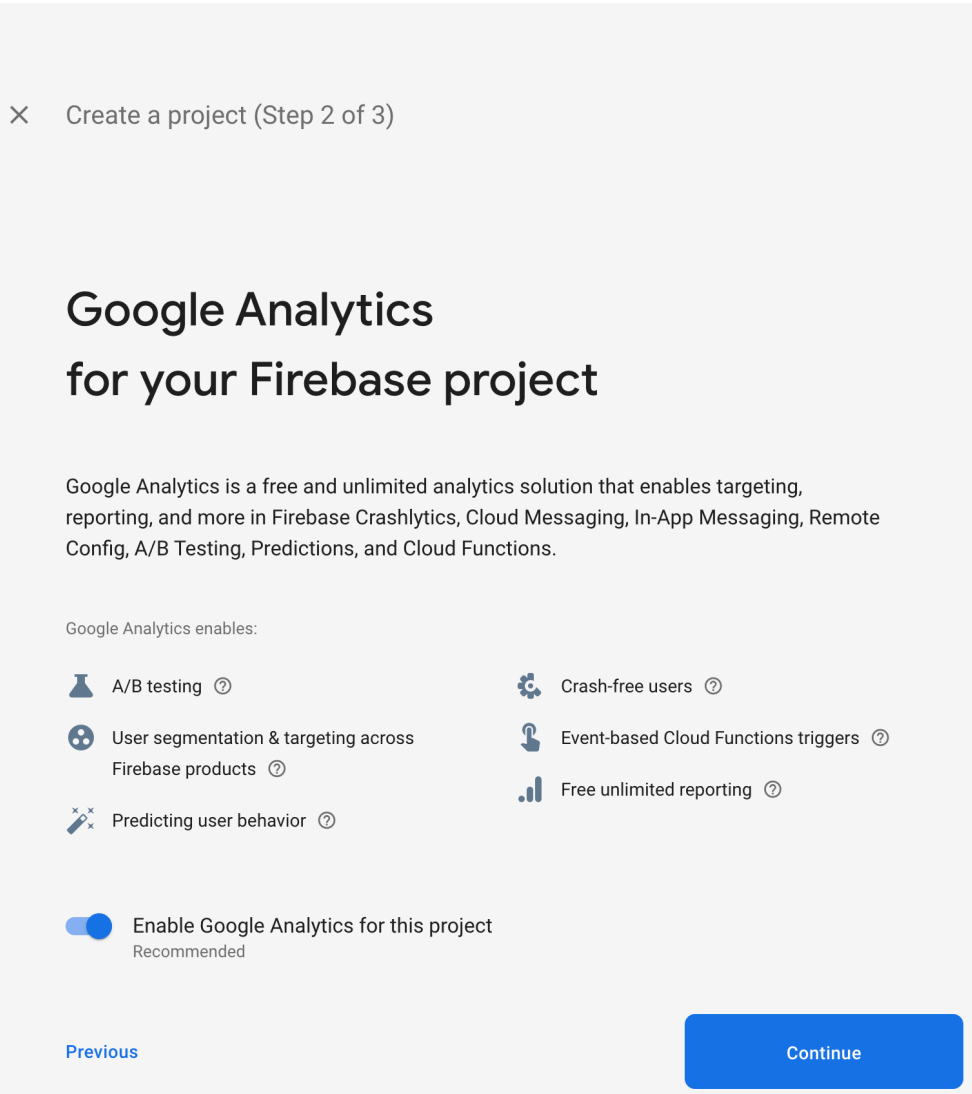
- Create a Firebase project
  - Link to a GCP project



FIREBASE IN ACTION

PROJECTS VS. APPS

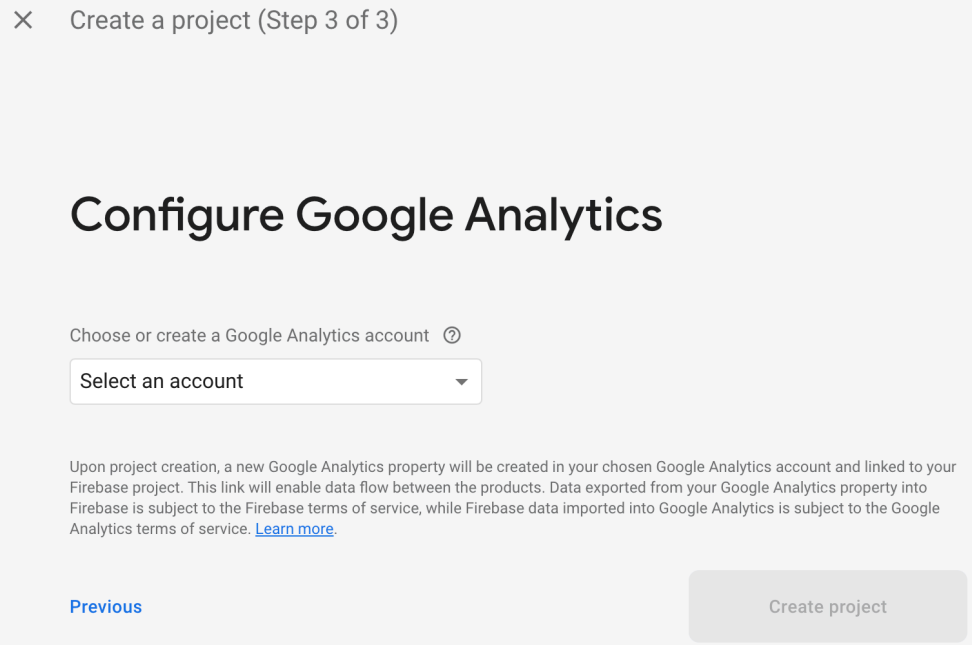
- Analytics



FIREBASE IN ACTION

PROJECTS VS. APPS

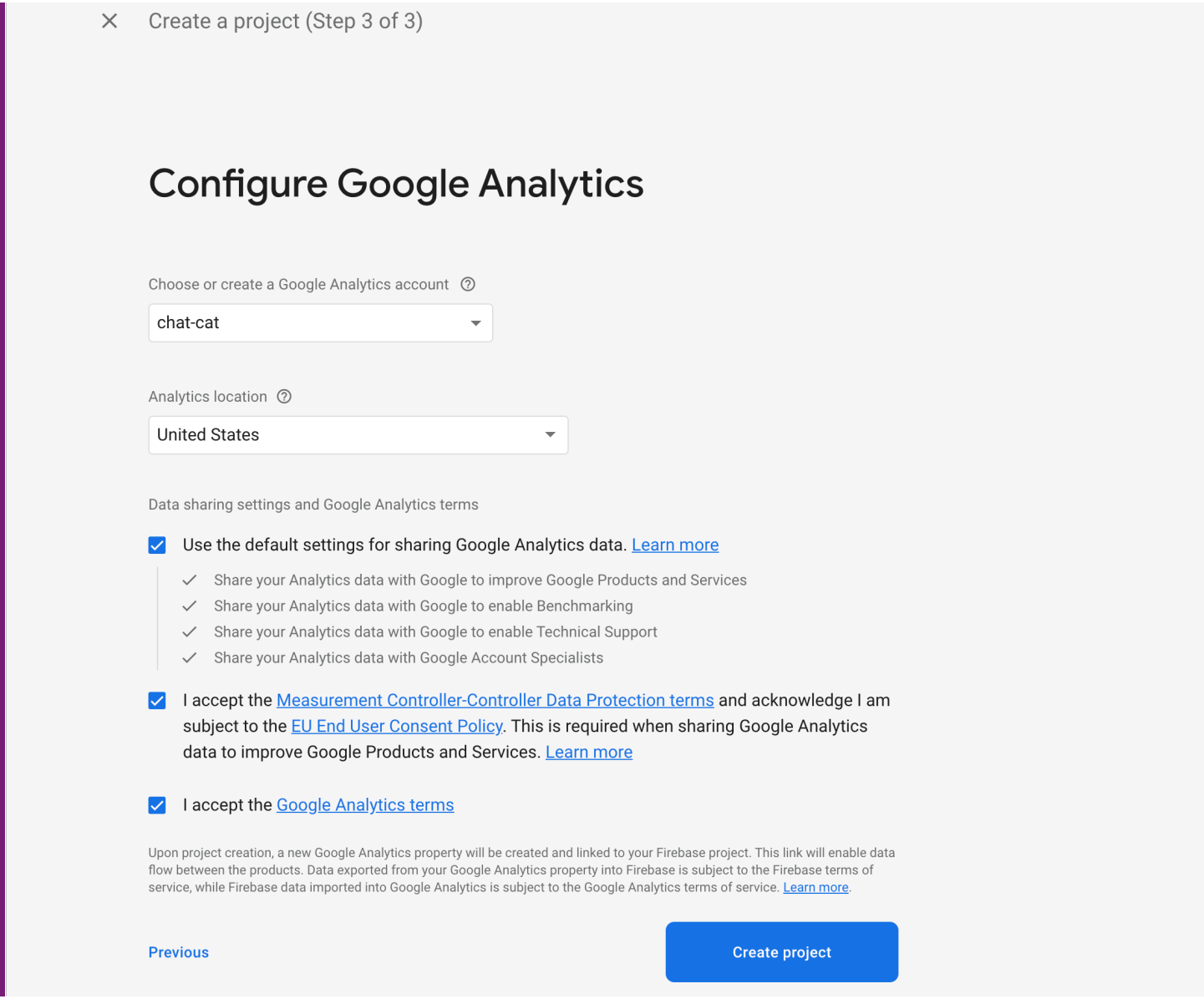
- Analytics



FIREBASE IN ACTION

PROJECTS VS. APPS

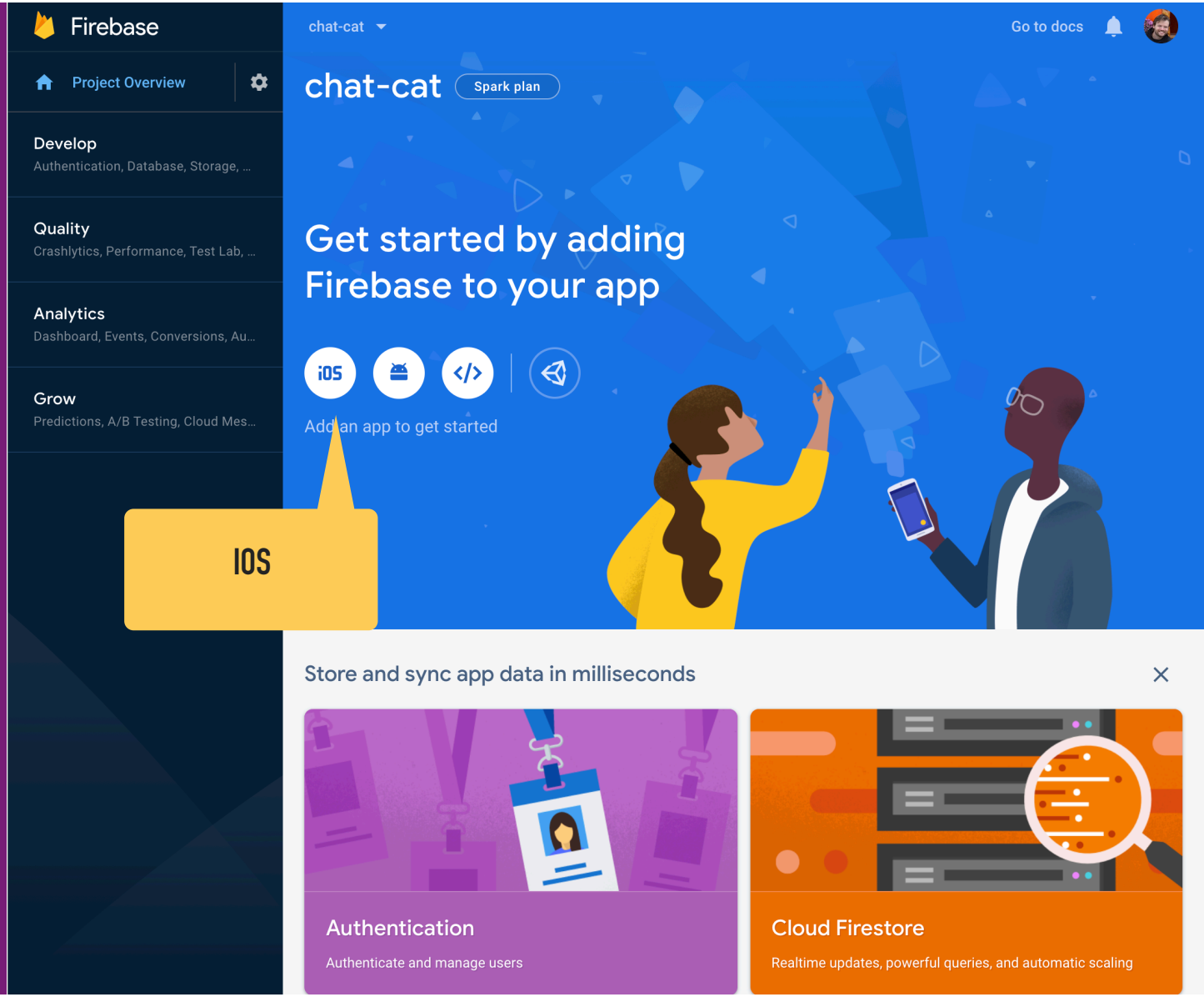
- Analytics



FIREBASE IN ACTION

PROJECTS VS. APPS

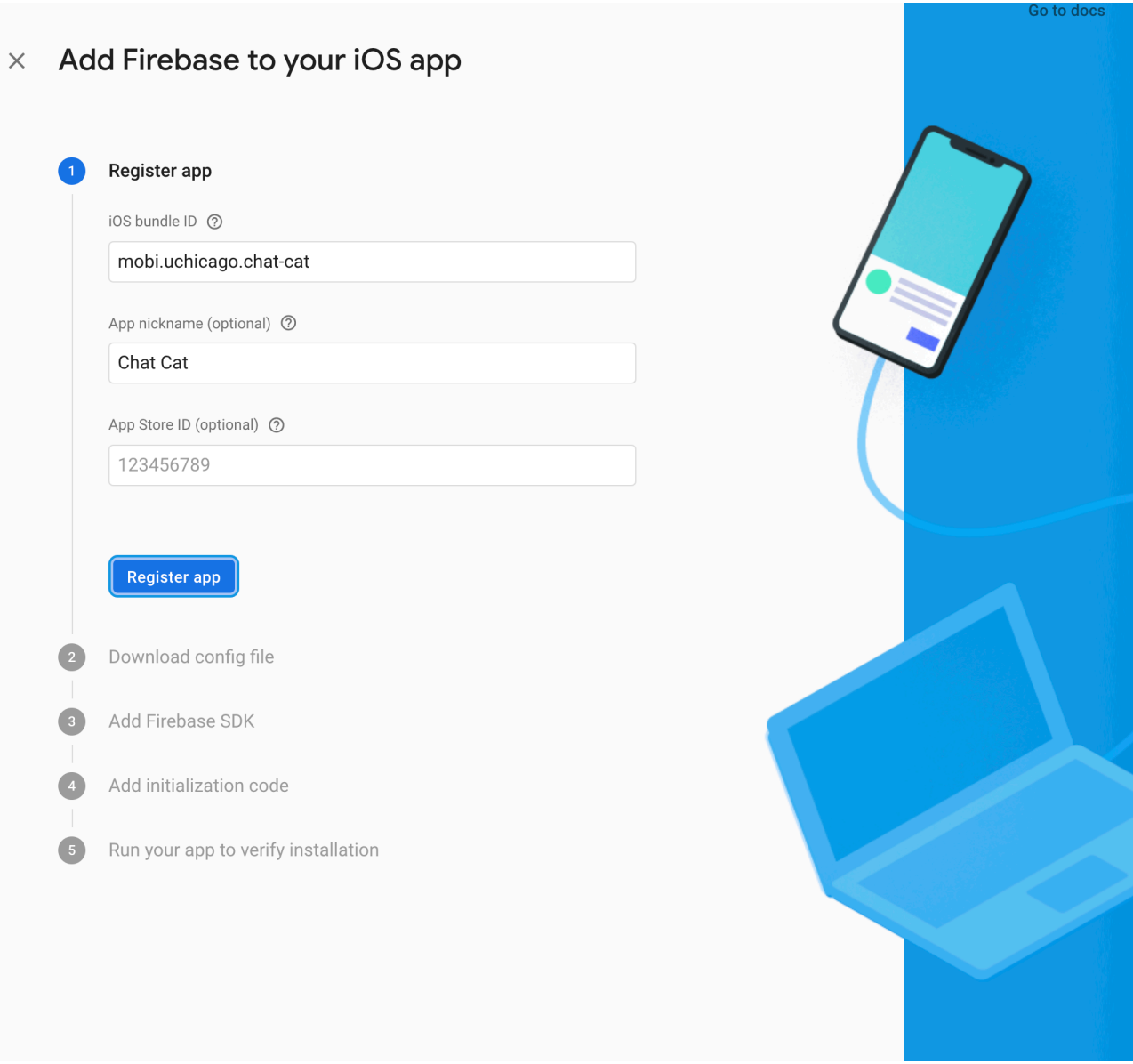
- Console



FIREBASE IN ACTION

REGISTER APP

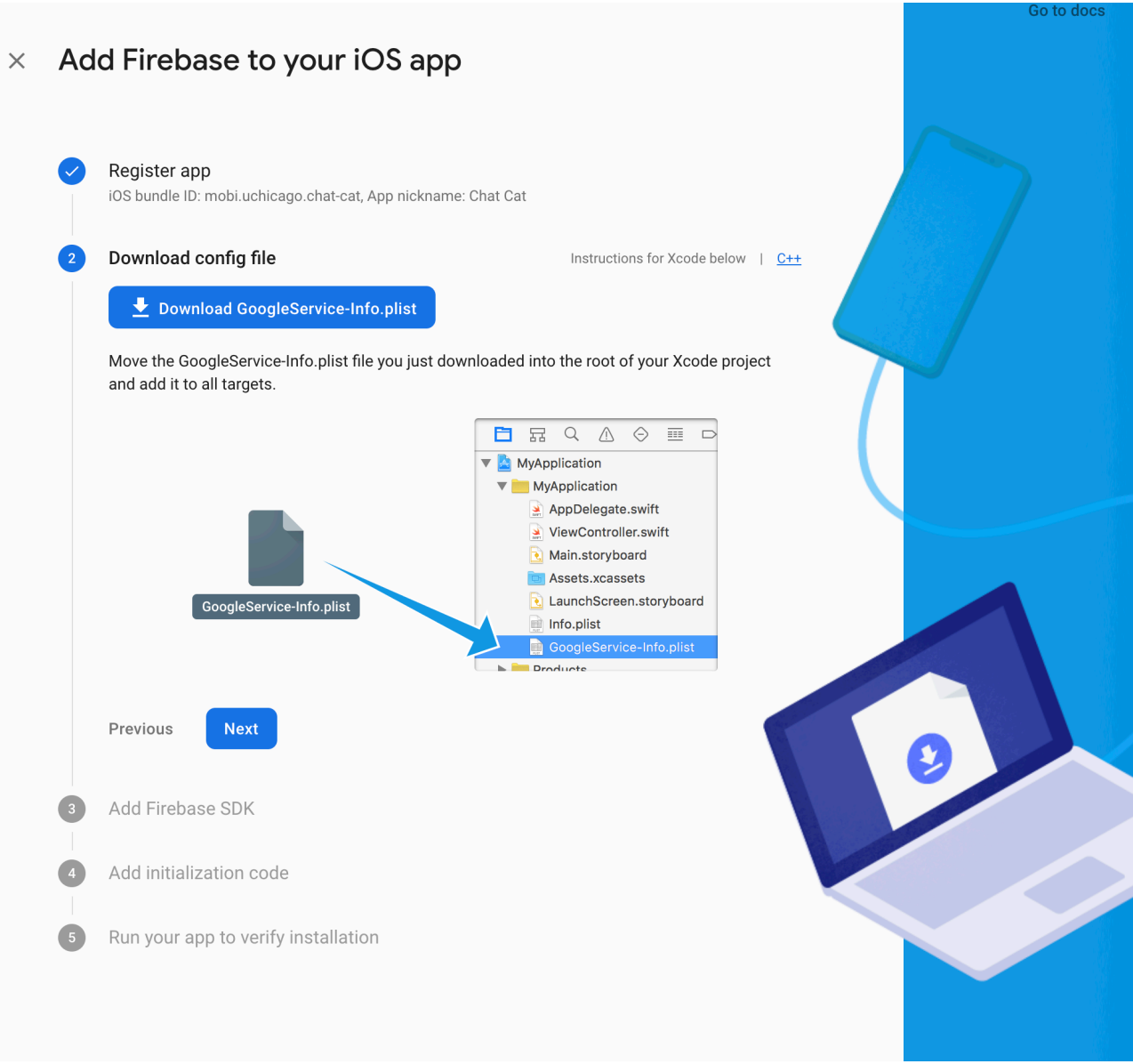
- Add Firebase to your app
- Bundle id needs to match you app bundle id
- App Store ID will be generated when you submit
  - Invites, Dynamic Links



FIREBASE IN ACTION

CONFIG FILE

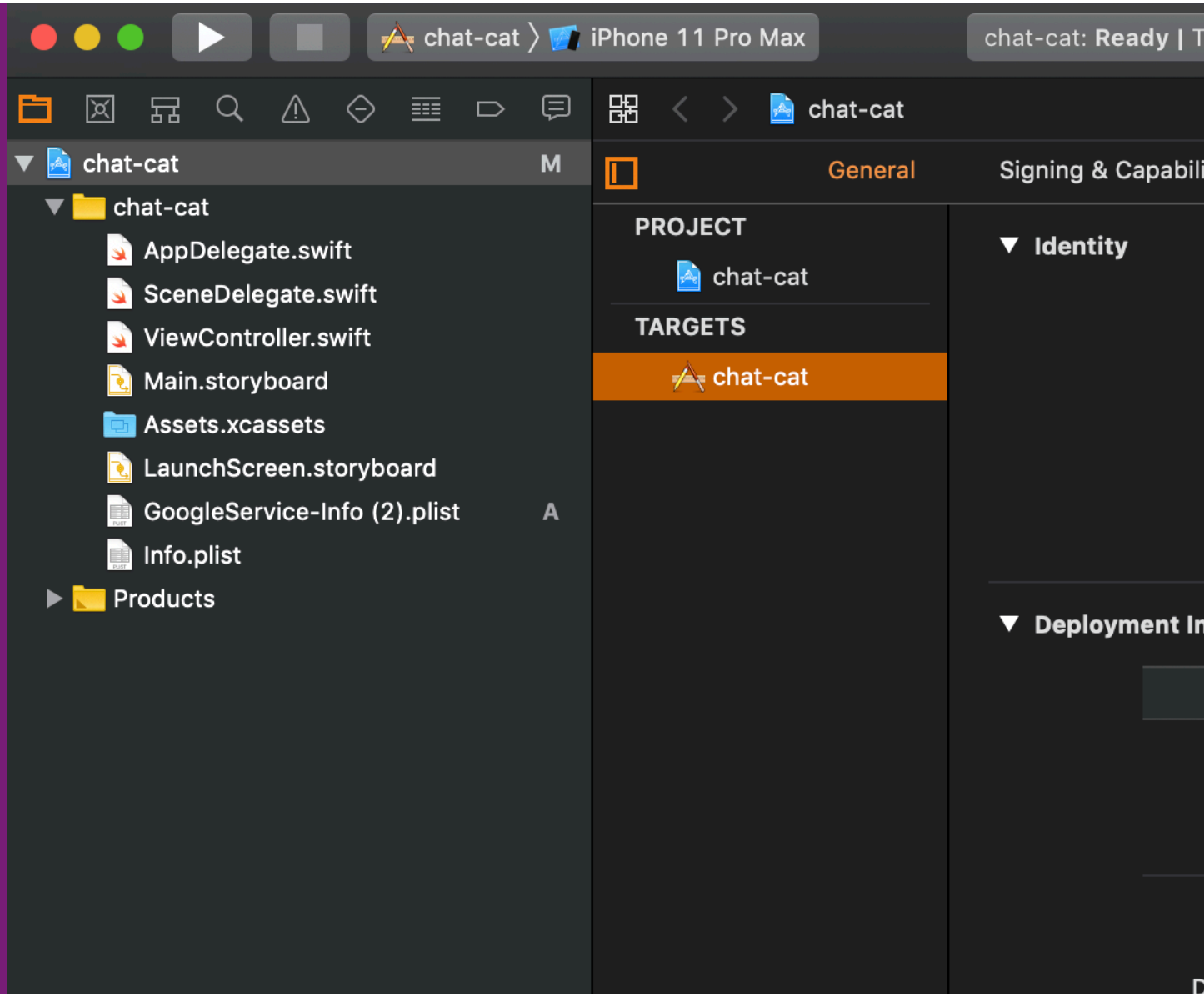
- Generates a .plist to include in your project



FIREBASE IN ACTION

CONFIG FILE

- Put in your project



FIREBASE IN ACTION

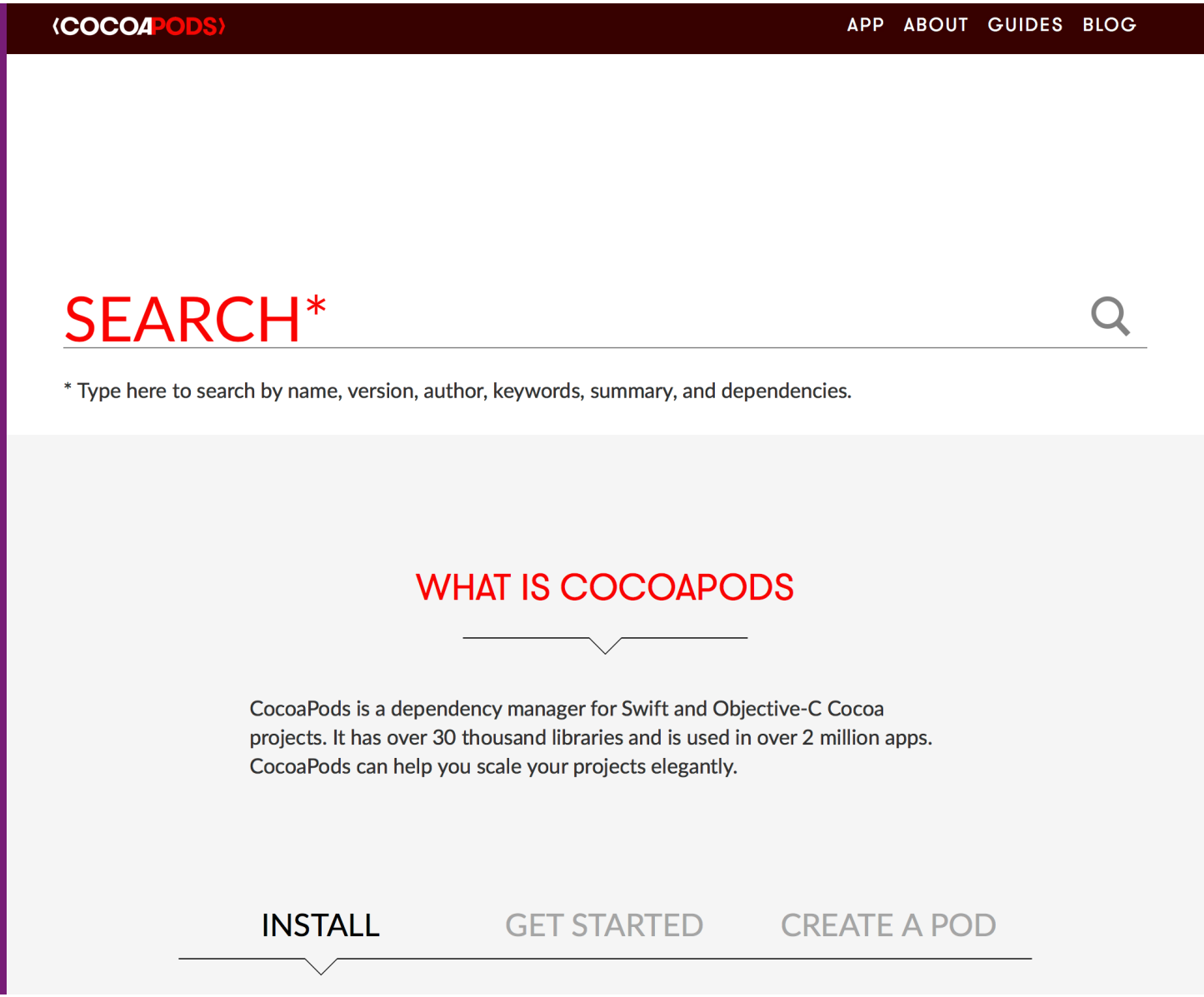
CONFIG FILE

Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
AD_UNIT_ID_FOR_BANNER_TEST	String	ca-app-pub-3940256099942544/2934735716
AD_UNIT_ID_FOR_INTERSTITIAL_T...	String	ca-app-pub-3940256099942544/4411468910
CLIENT_ID	String	487781121319-ms6r6t0ninob0i8ri4mord3kd5roto9v.apps.googleusercontent.com
REVERSED_CLIENT_ID	String	com.googleusercontent.apps.487781121319-ms6r6t0ninob0i8ri4mord3kd5roto9v
API_KEY	String	AIzaSyDjXHxBu32Fwmh9urpLpB-B33av8hUPoNA
GCM_SENDER_ID	String	487781121319
PLIST_VERSION	String	1
BUNDLE_ID	String	mobi.uchicago.firechat
PROJECT_ID	String	firechat-66f87
STORAGE_BUCKET	String	firechat-66f87.appspot.com
IS_ADS_ENABLED	Boolean	YES
IS_ANALYTICS_ENABLED	Boolean	NO
IS_APPINVITE_ENABLED	Boolean	YES
IS_GCM_ENABLED	Boolean	YES
IS_SIGNIN_ENABLED	Boolean	YES
GOOGLE_APP_ID	String	1:487781121319:ios:78425b1380795a79
DATABASE_URL	String	https://firechat-66f87.firebaseio.com

FIREBASE IN ACTION

ADD SDK

- Firebase uses cocoapods for SDK distribution
- Changes xcodeproject to xcworkspace



FIREBASE IN ACTION

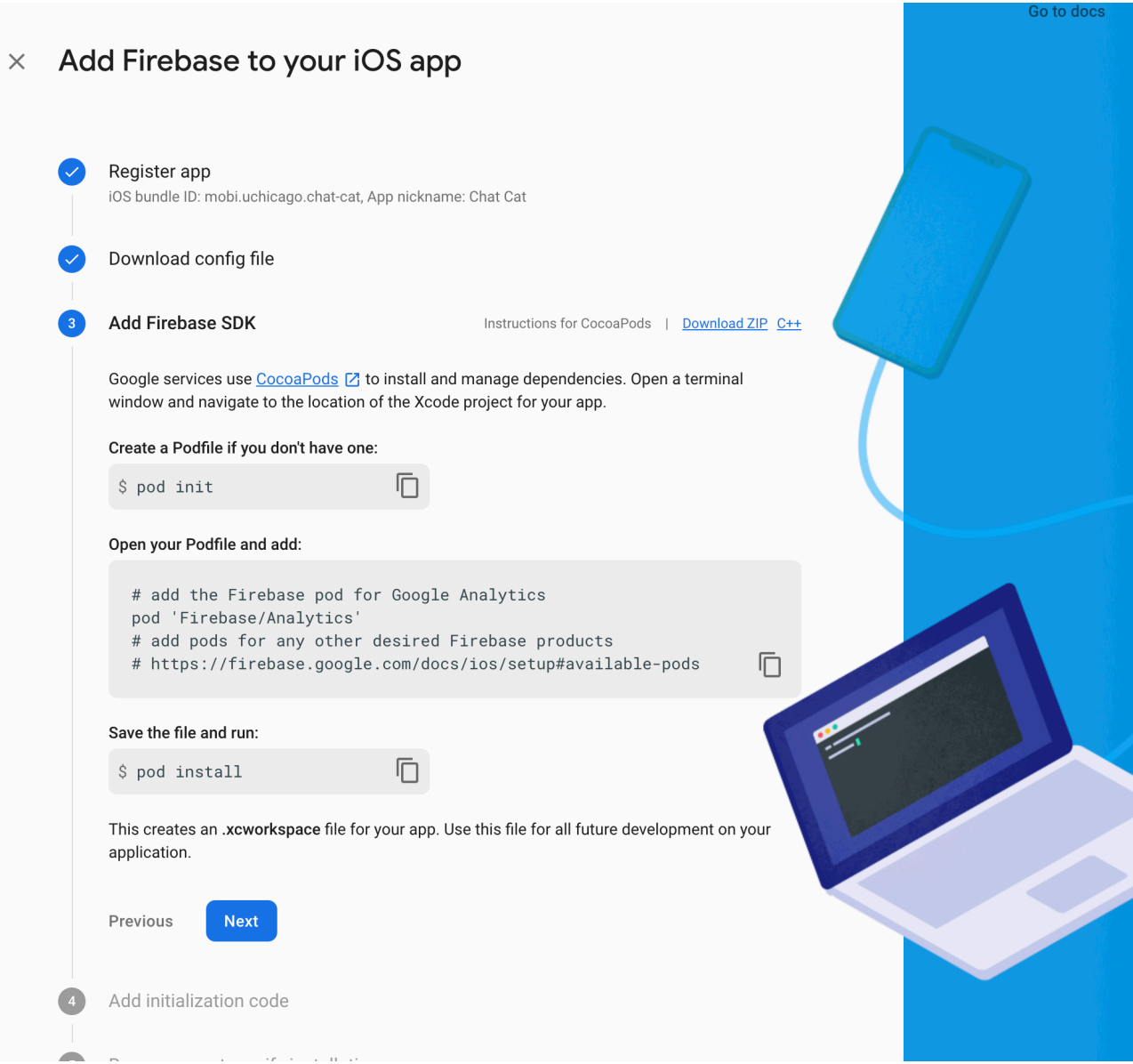
ADD SDK

HTTPS://WWW.YOUTUBE.COM/WATCH?V=IEAJVNRDZA0

FIREBASE IN ACTION

ADD SDK

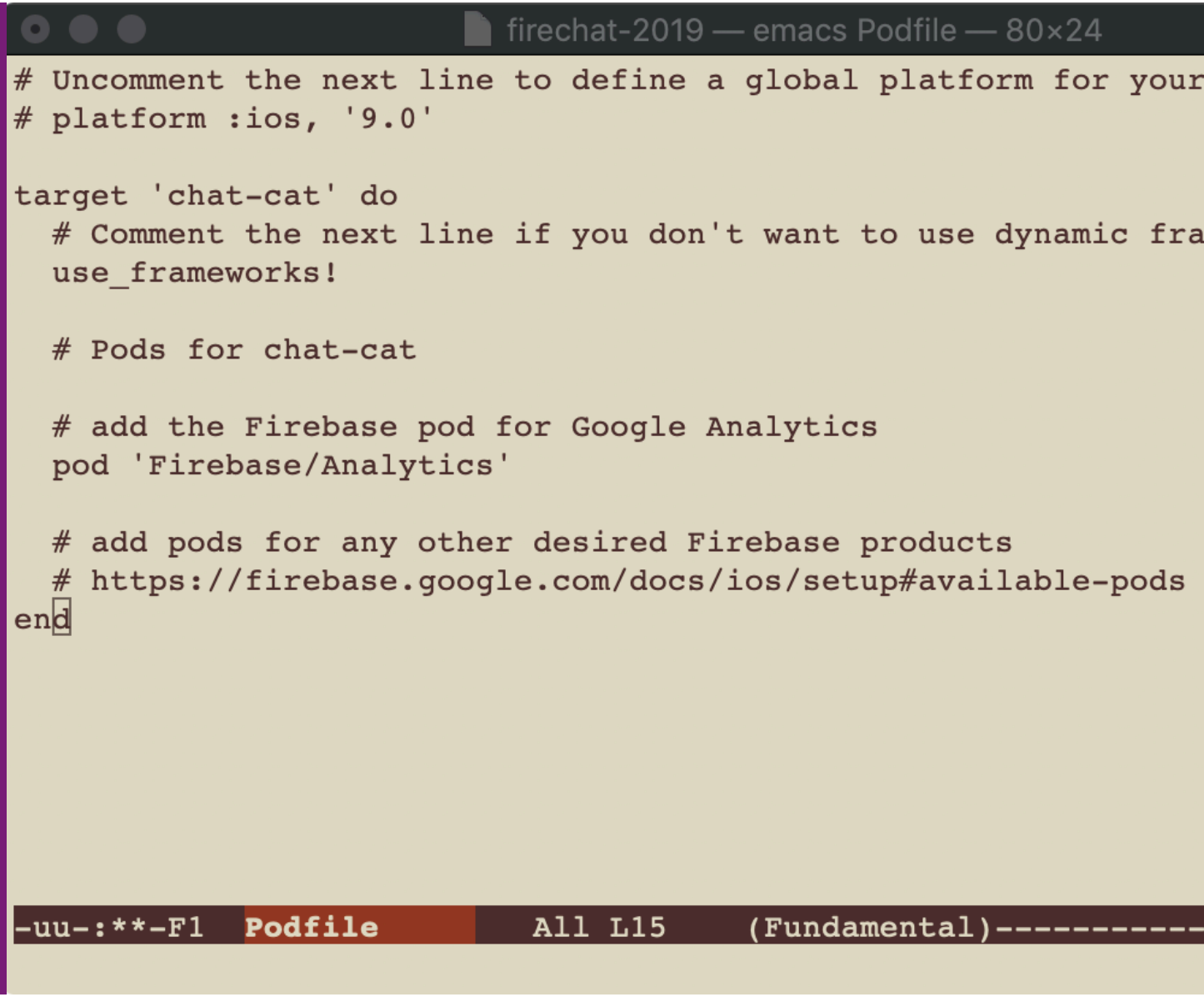
- Pod install Firebase SDK



FIREBASE IN ACTION

ADD SDK

- Pod install Firebase SDK



FIREBASE IN ACTION

ADD SDK

- Firebase pods
- There are multiple Firebase pods
  - Only use what you need

Available pods

The following pods are available for the various Firebase products.

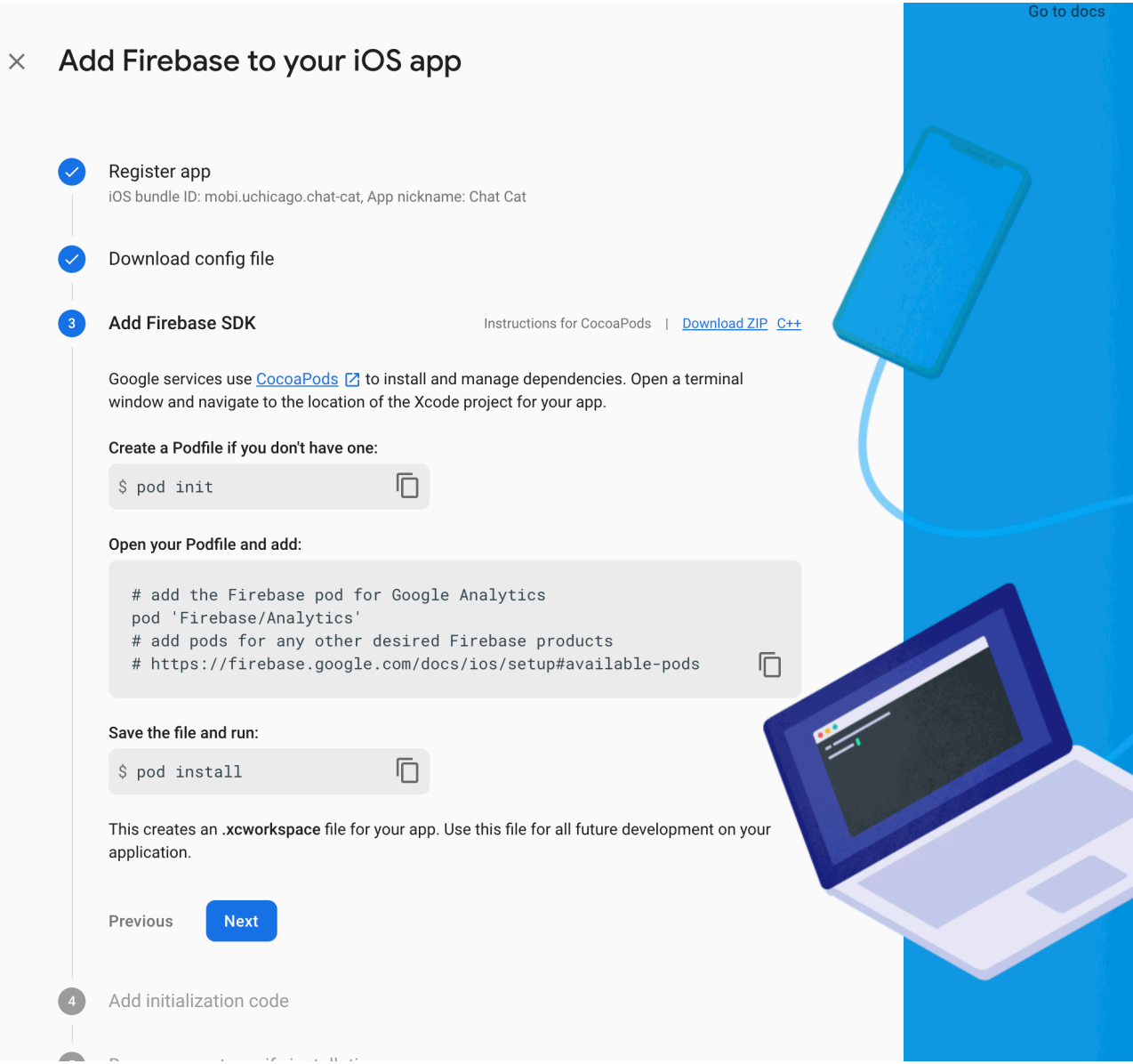
★ You no longer need to add the iOS pod **`Firebase/Core`**. This SDK included the Firebase SDK for G Analytics (or any of the Firebase products that require or recommend the use of Analytics), you n pod: **`Firebase/Analytics`**.

Firebase product	Pods
AdMob	`pod 'Firebase/AdMob'` *(required)* `pod 'Firebase/Analytics'`
Analytics	`pod 'Firebase/Analytics'`
Authentication	`pod 'Firebase/Auth'`
Cloud Firestore	`pod 'Firebase/Firestore'`
Cloud Functions for Firebase Client SDK	`pod 'Firebase/Functions'`
Cloud Messaging	`pod 'Firebase/Messaging'` *(recommended)* `pod 'Firebase/Analytics'`
Cloud Storage	`pod 'Firebase/Storage'`

FIREBASE IN ACTION

ADD SDK

- Install the Firebase pods in your project



FIREBASE IN ACTION

ADD SDK

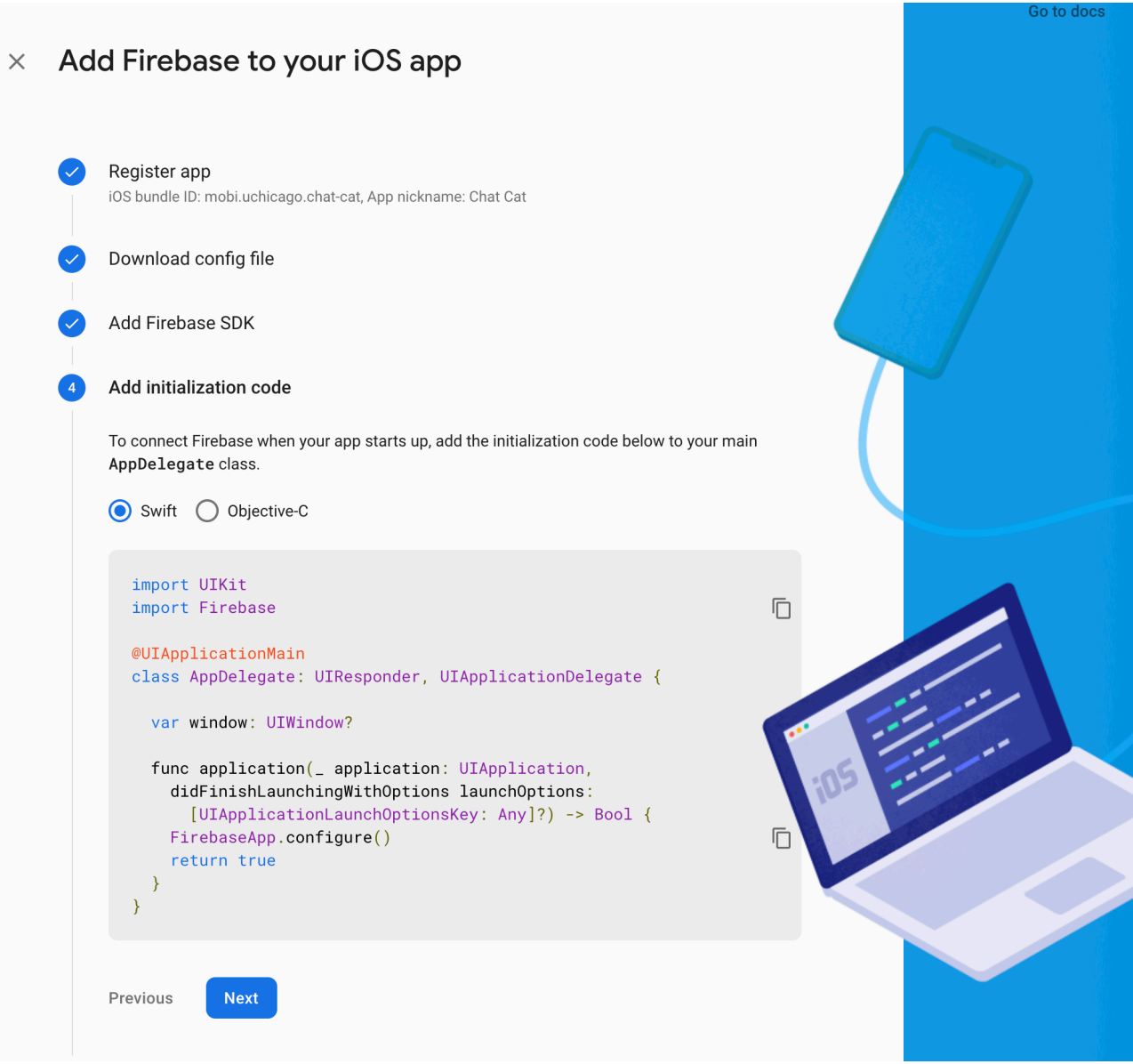
```
Installing FirebaseAuthInterop (1.0.0)
Installing FirebaseCore (6.3.0)
Installing FirebaseCoreDiagnostics (1.1.0)
Installing FirebaseCoreDiagnosticsInterop (1.0.0)
Installing FirebaseDatabase (6.1.0)
Installing FirebaseInstanceID (4.2.5)
Installing GoogleAppMeasurement (6.1.2)
Installing GoogleDataTransport (2.0.0)
Installing GoogleDataTransportCCTSupport (1.1.0)
Installing GoogleUtilities (6.3.1)
Installing leveldb-library (1.22)
Installing nanopb (0.3.901)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `chat-cat.xcworkspace` for this project from now on.
Pod installation complete! There are 2 dependencies from the Podfile and 14 total pods installed.

[!] Automatically assigning platform `iOS` with version `13.1` on target `chat-cat` because no platform was
specified. Please specify a platform for this target in your Podfile. See `https://guides.cocoapods.org/synt
ax/podfile.html#platform`.
tabinkowski:firebase/chat-cat (master)
530 %
```

WORK IN WORKSPACE

FIREBASE IN ACTION

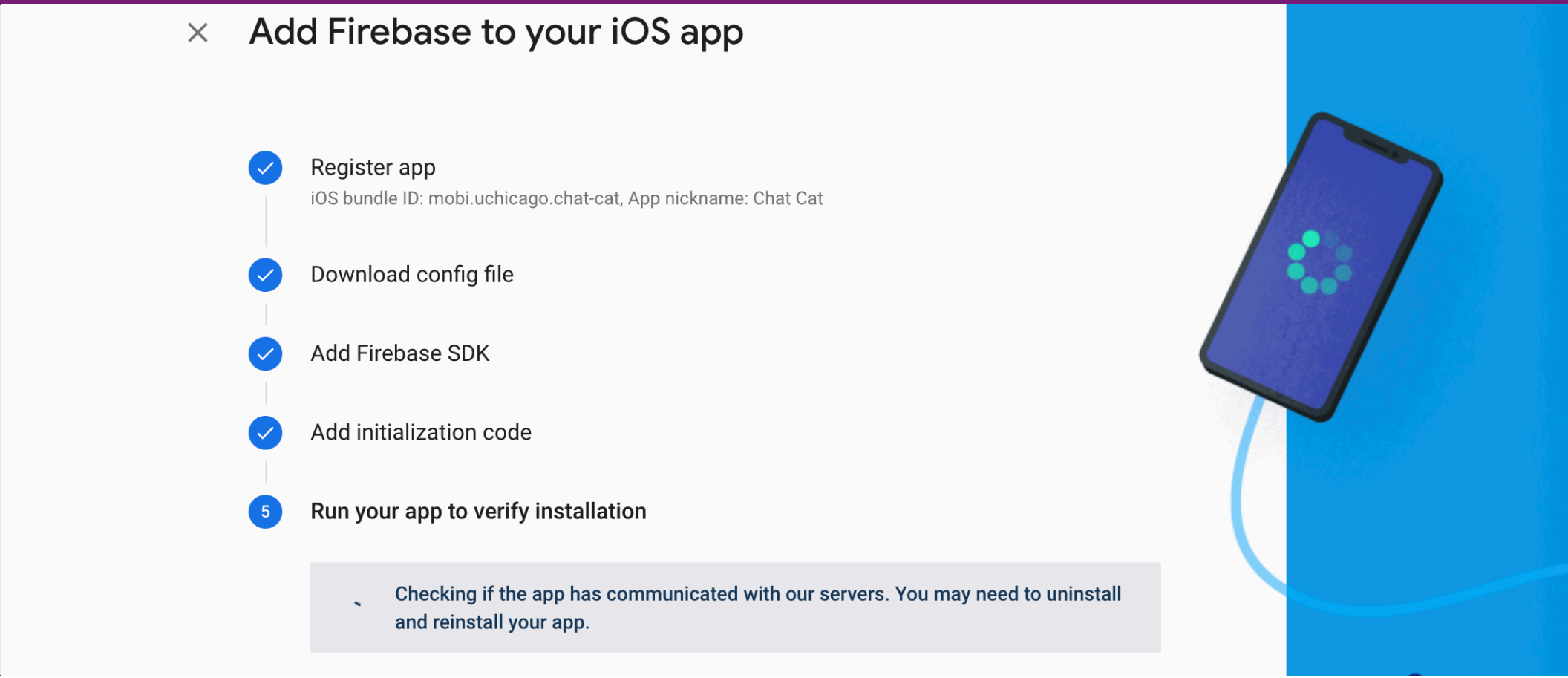
ADD SDK

- Add initialization code to AppDelegate
  - Initializes services
  - Reads the plist and sets global variables



FIREBASE IN ACTION

CONSOLE

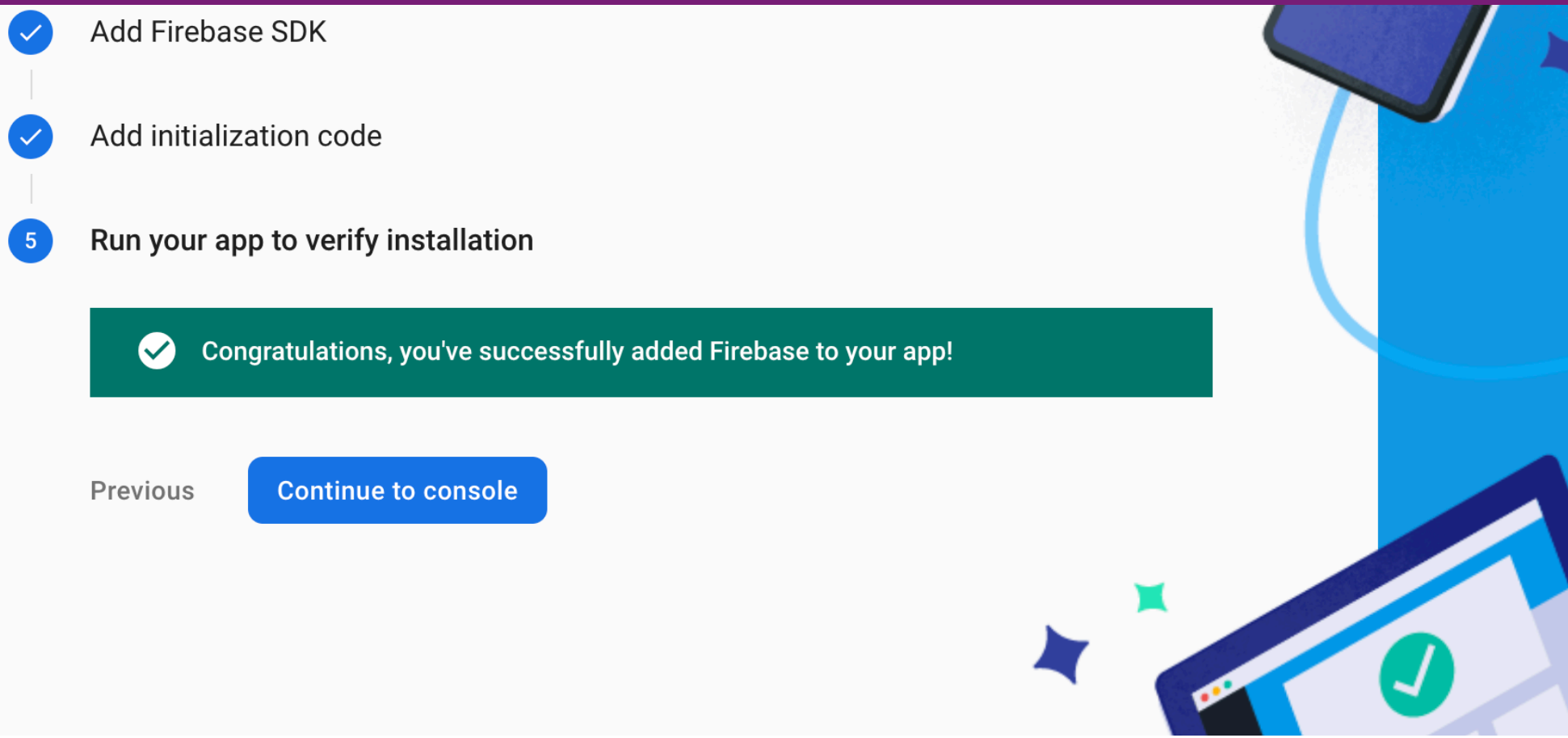


FIREBASE IN ACTION

BUILD THE APP

FIREBASE IN ACTION

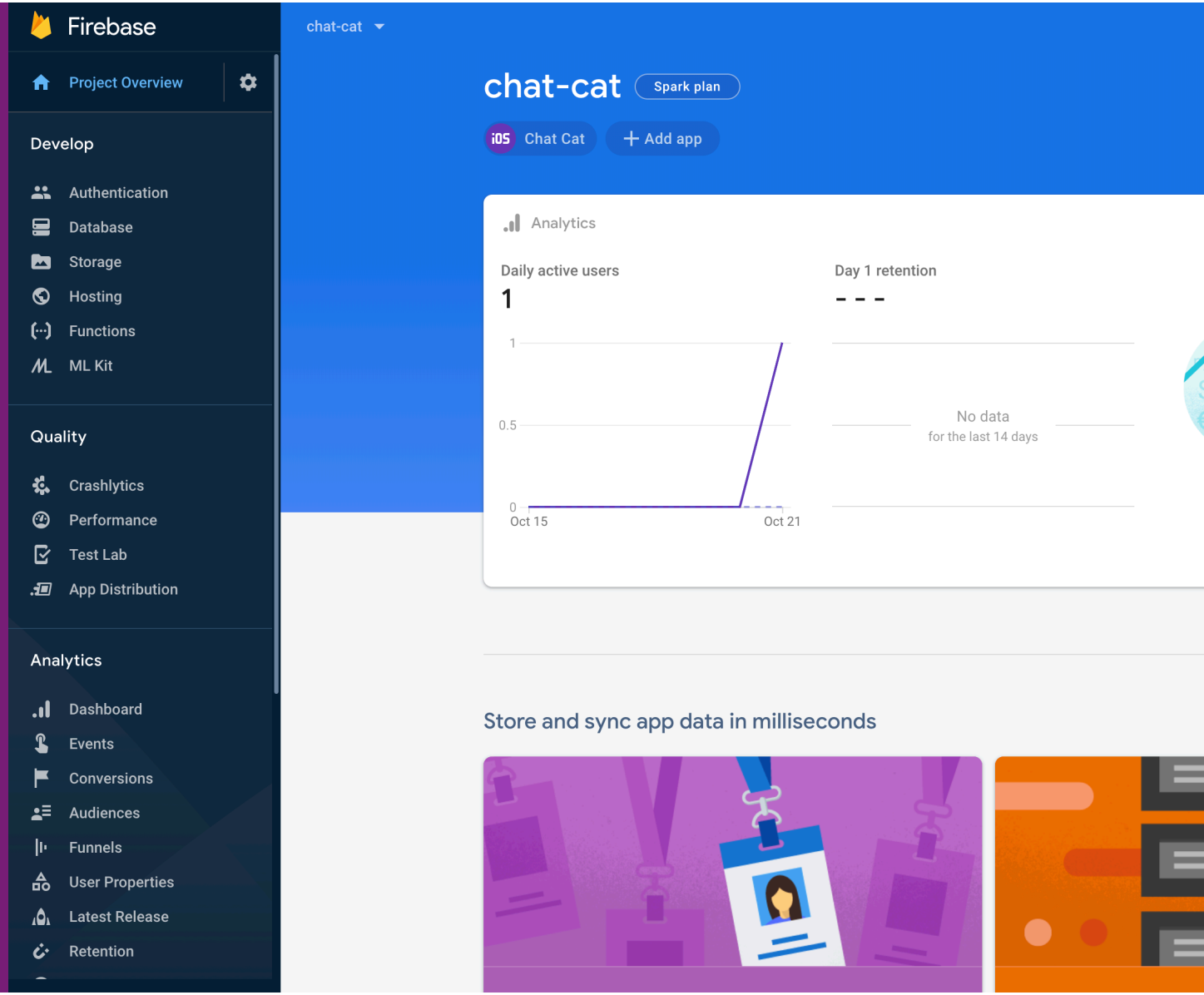
CONSOLE



FIREBASE CONSOLE

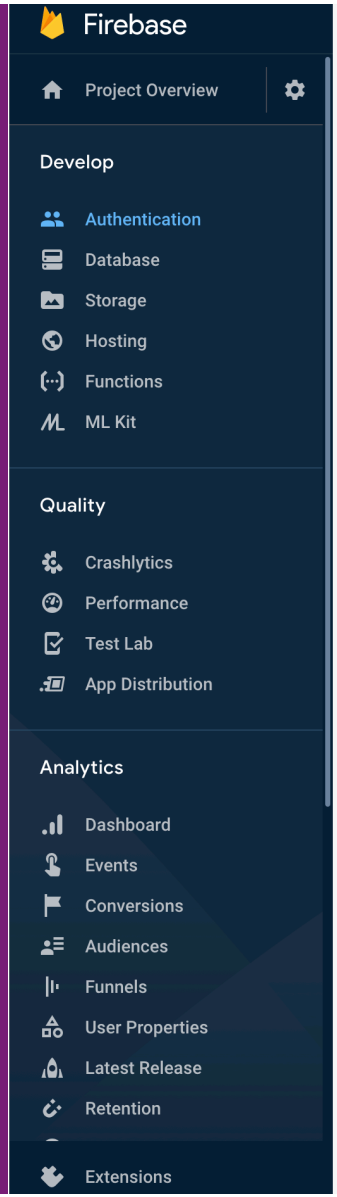
FIREBASE CONSOLE

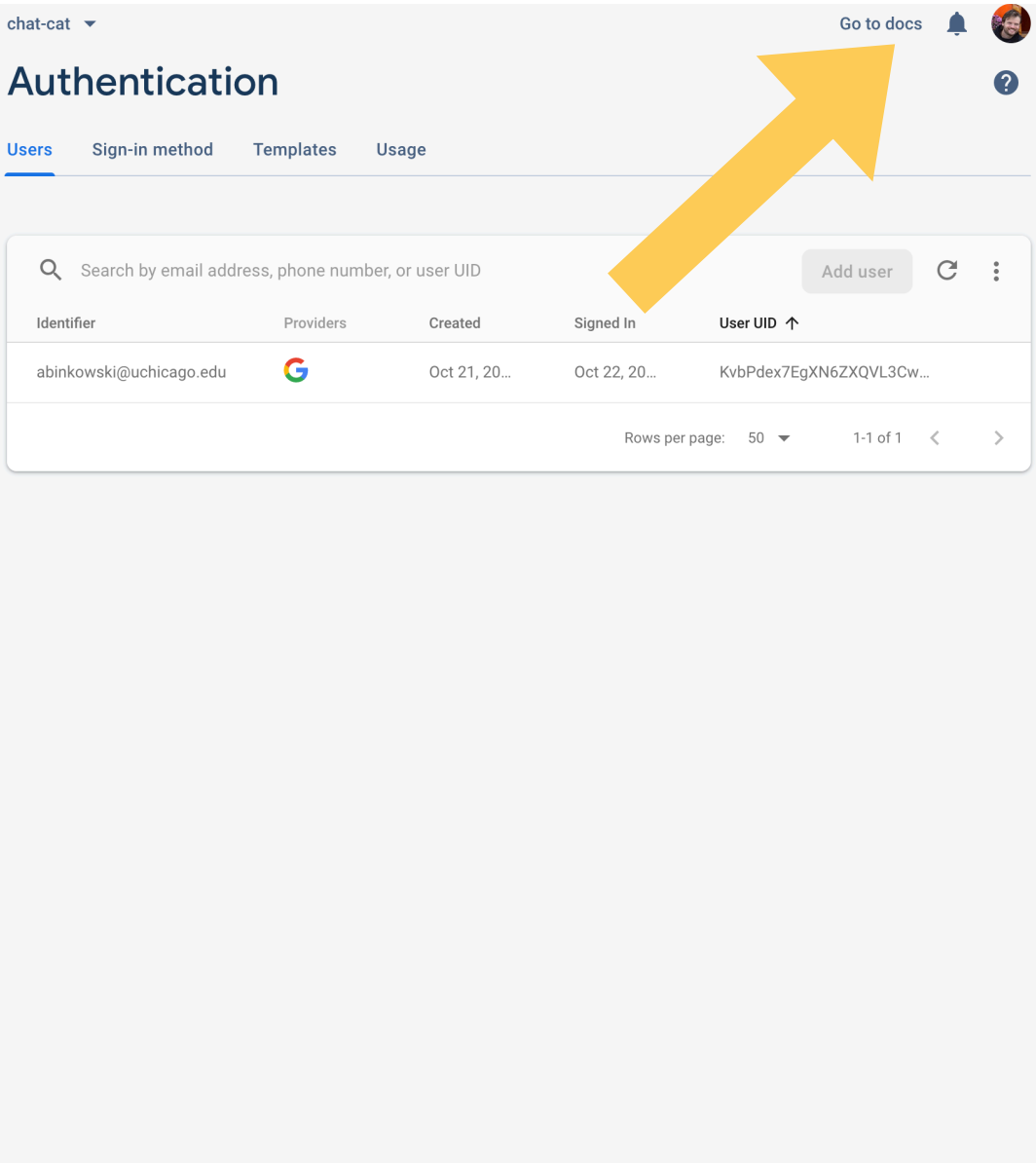
- Enable services
- Data
- Analytics
- Configure



FIREBASE CONSOLE

- Everything you wanted to know about your app
- Quick access to documentation

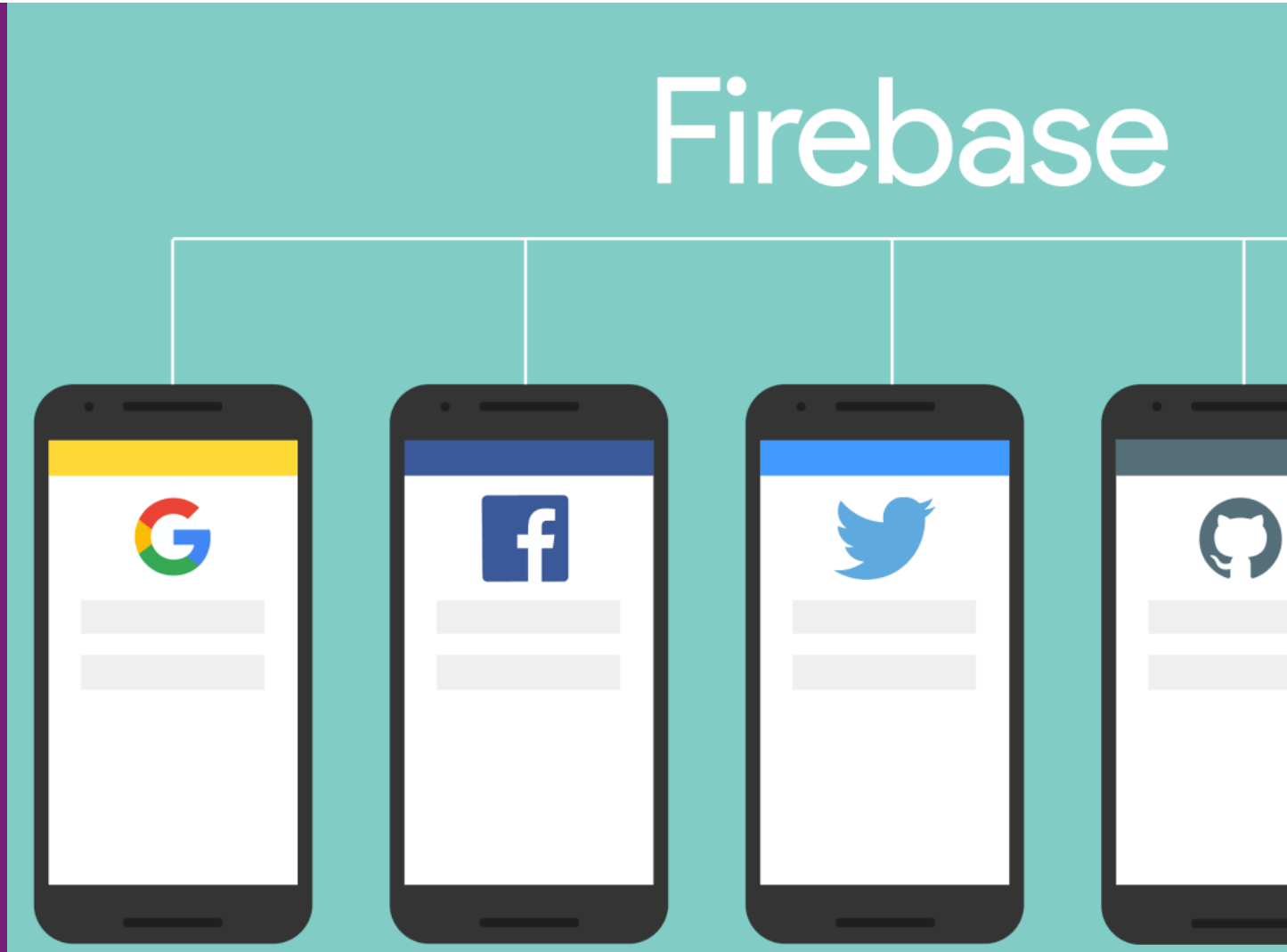




FIREBASE AUTHENTICATION

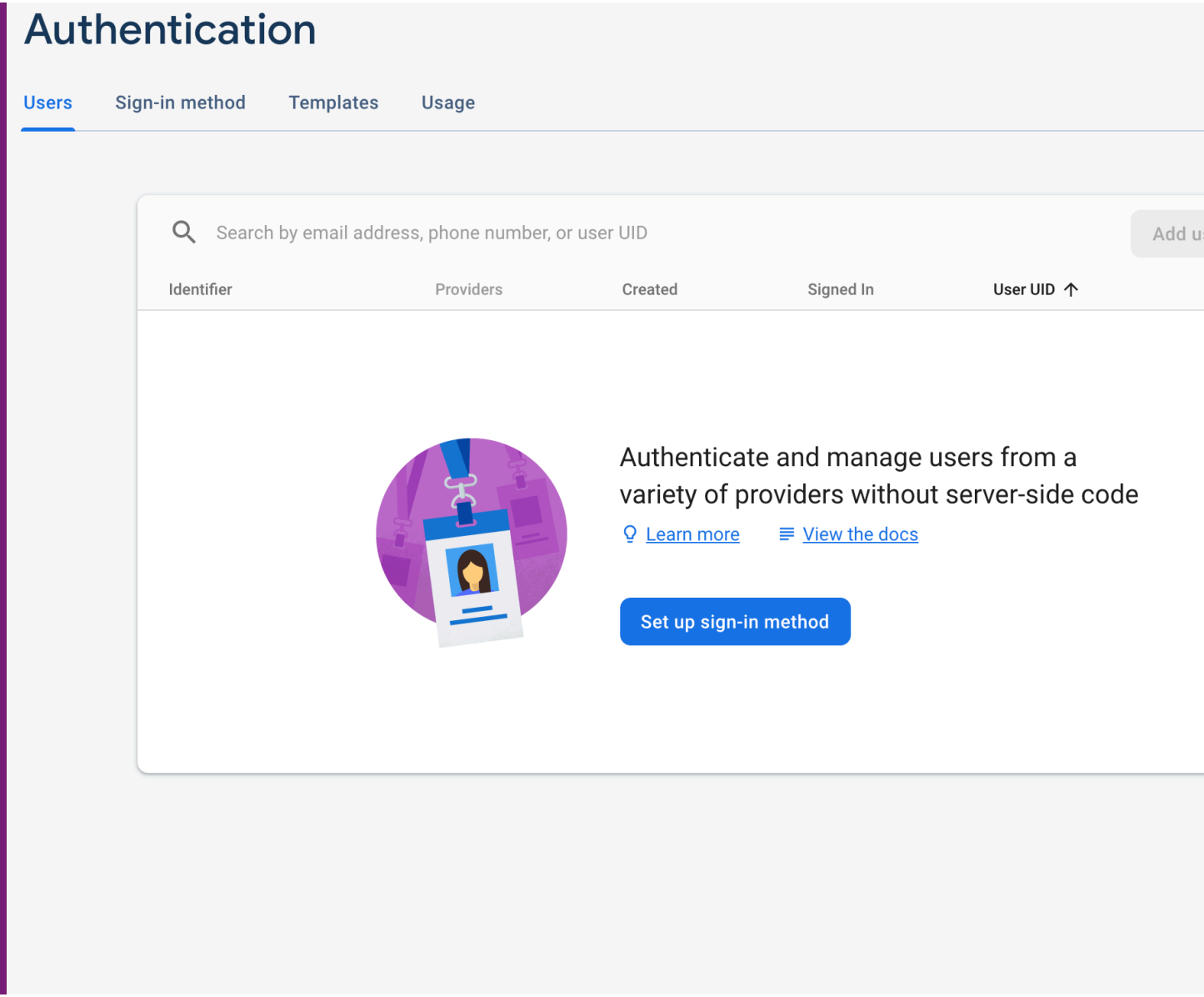
FIREBASE AUTHENTICATION

- https://www.youtube.com/watch?v=8sGY55yxicA&index=14&list=PLI-K7zZEsYLmOF_07IayrTntevxtbUxDL



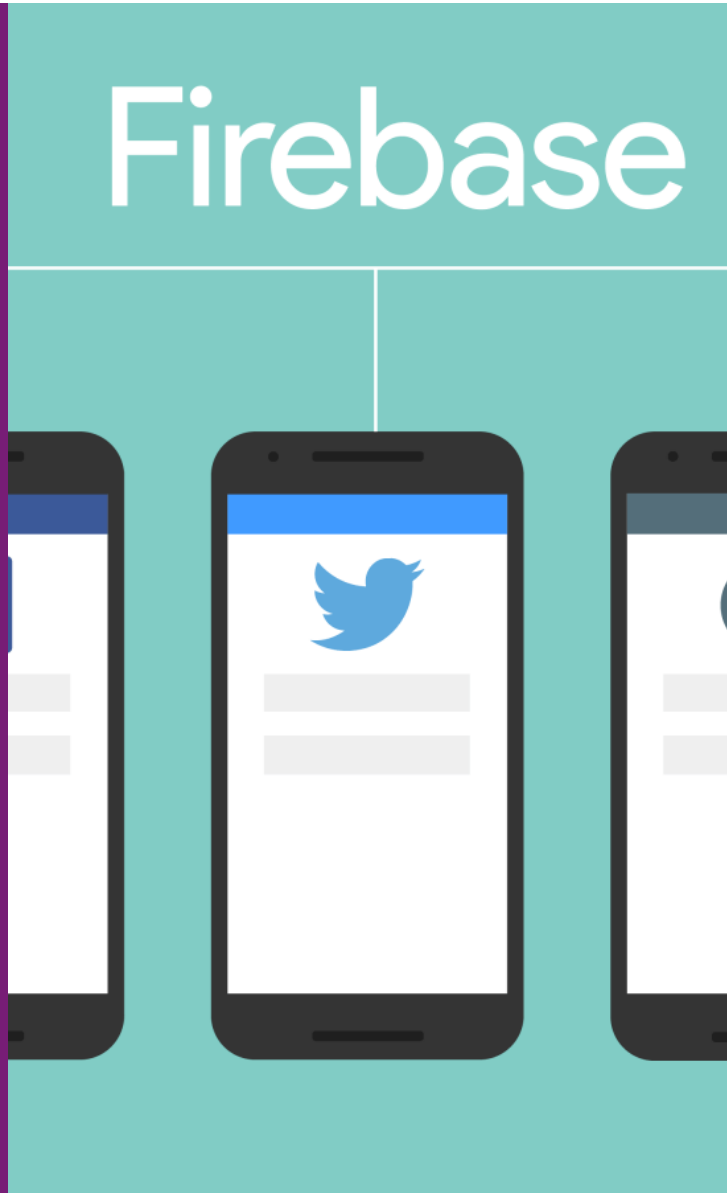
FIREBASE AUTHENTICATION

- Provide a secure and customized experience for users
- Many options, including your own custom systems



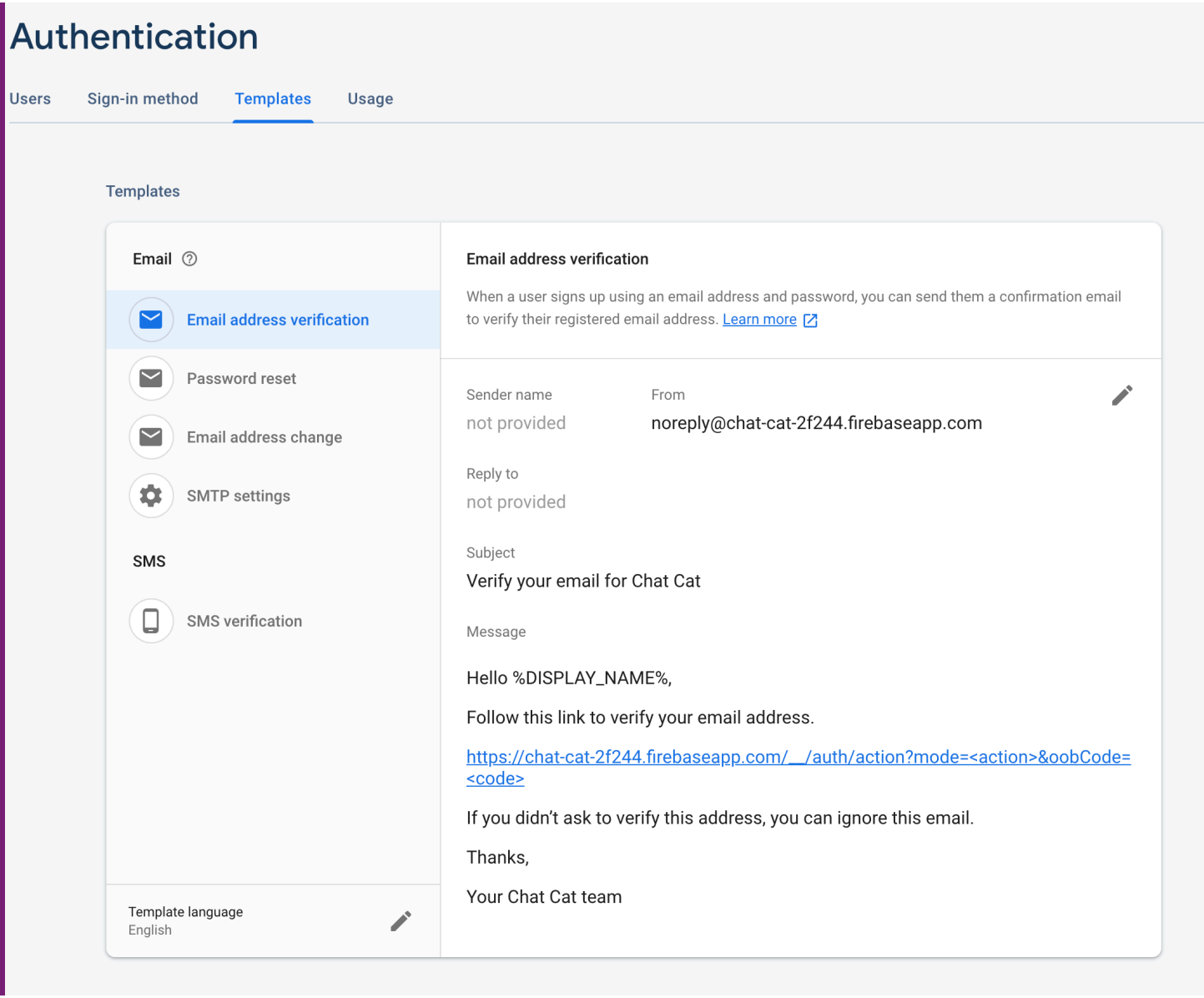
FIREBASE AUTHENTICATION

- Authentication flow
  - User logins to service through interface
  - Credentials passed to Firebase Authentication SDK
  - Backend does authorization and passes back response from the client



FIREBASE AUTHENTICATION

- Templates for interaction



FIREBASE AUTHENTICATION

- Using FirebaseUI Auth

Using FirebaseUI Auth

1	Set up sign-in methods	For email address and password sign-in and any federated identity providers you want to support, enable them in the Firebase console and complete any configuration required by the identity provider, such as setting your OAuth redirect URL.
2	Customize the sign-in UI	You can customize the sign-in UI by setting FirebaseUI options, or fork the code on GitHub to customize the sign-in experience further.
3	Use FirebaseUI to perform the sign-in flow	Import the FirebaseUI library, specify the sign-in methods you want to support, and initiate the FirebaseUI sign-in flow.

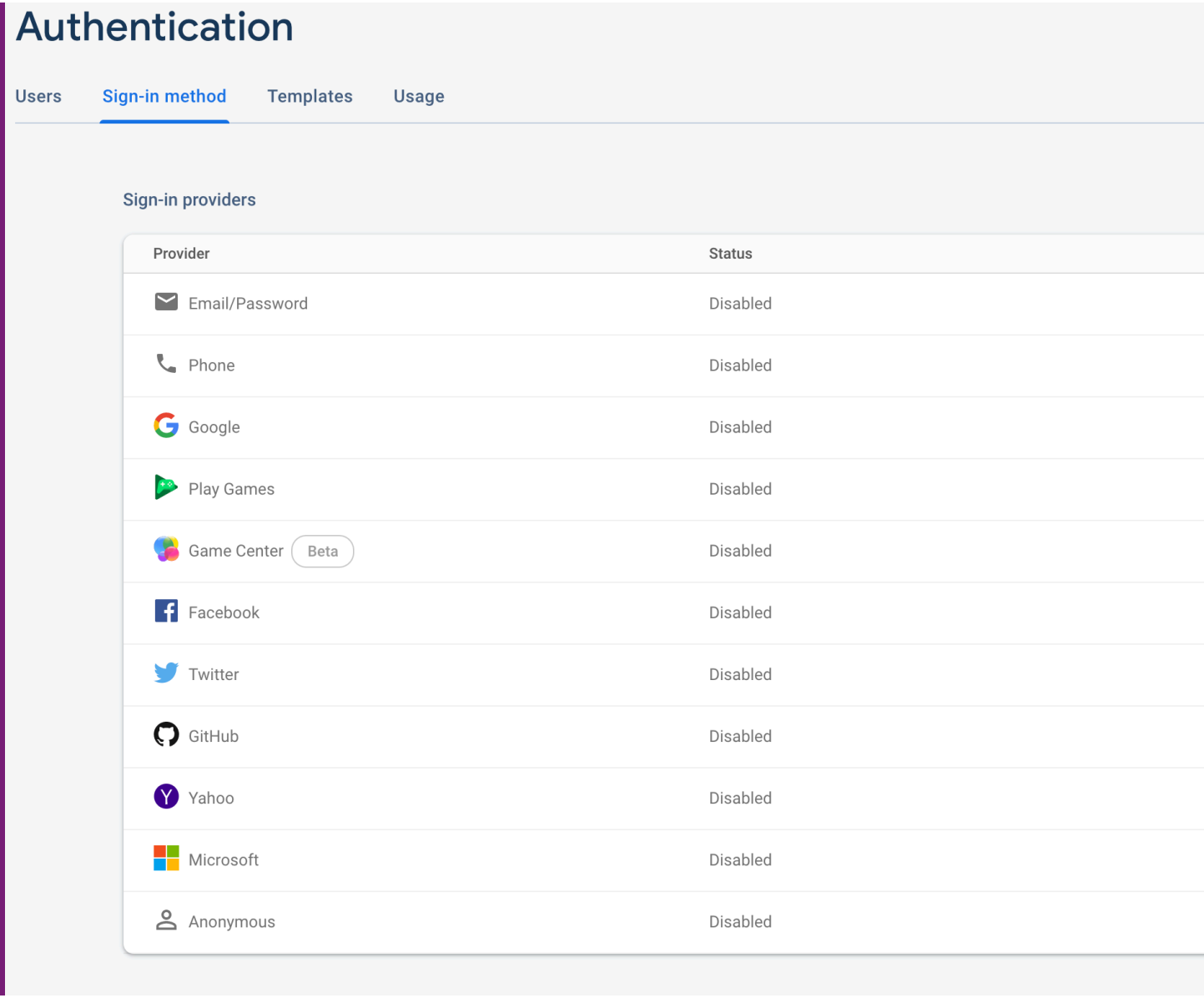
FIREBASE AUTHENTICATION

- Firebase Authentication SDK
- You create the UI

Using the Firebase Authentication SDK		
1	Set up sign-in methods	For email address and password sign-in and any federated identity providers you want to support, enable them in the Firebase console and complete any configuration required by the identity provider, such as setting your OAuth redirect URL.
2	Implement UI flows for your sign-in methods	For email address and password sign-in, implement a flow that prompts users to type their email addresses and passwords. For federated sign-in, implement the flow required by each provider.
3	Pass the user's credentials to the Firebase Authentication SDK	Pass the user's email address and password or the OAuth token that was acquired from the federated identity provider to the Firebase Authentication SDK.

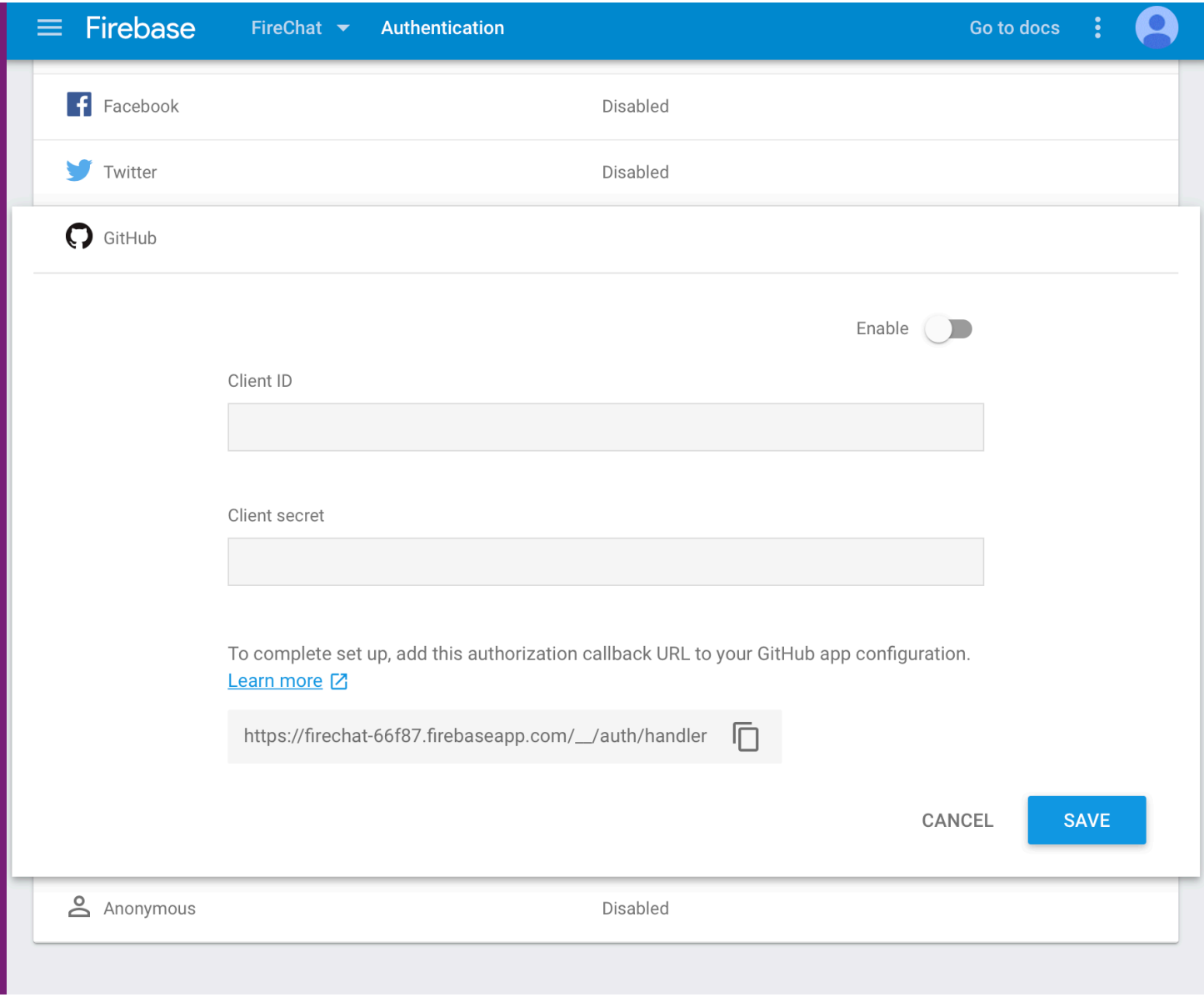
FIREBASE AUTHENTICATION

- Select method



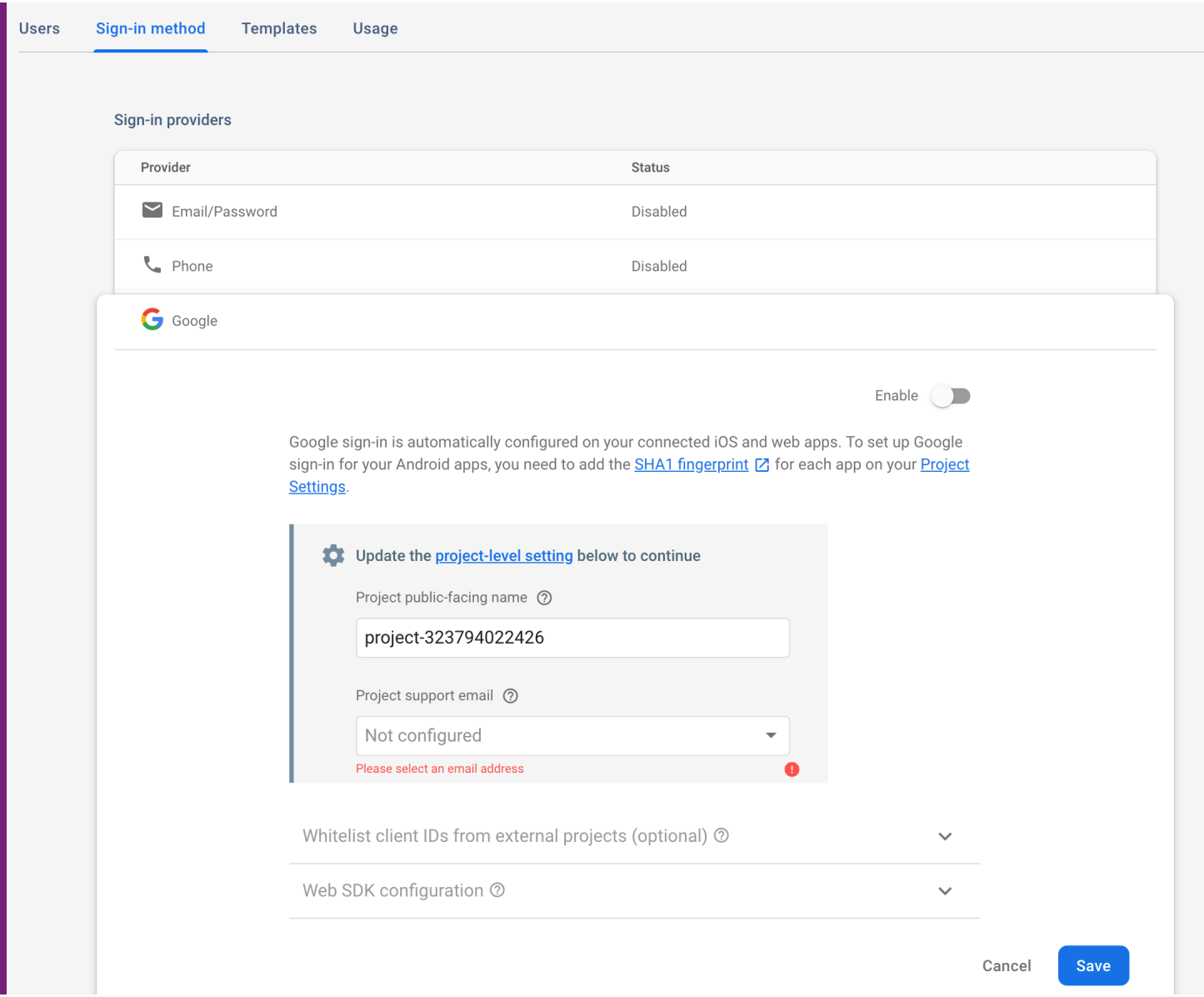
FIREBASE AUTHENTICATION

- Third-party authentication requires granting access
- Set up on each service



FIREBASE AUTHENTICATION

- Get Google for 'free'

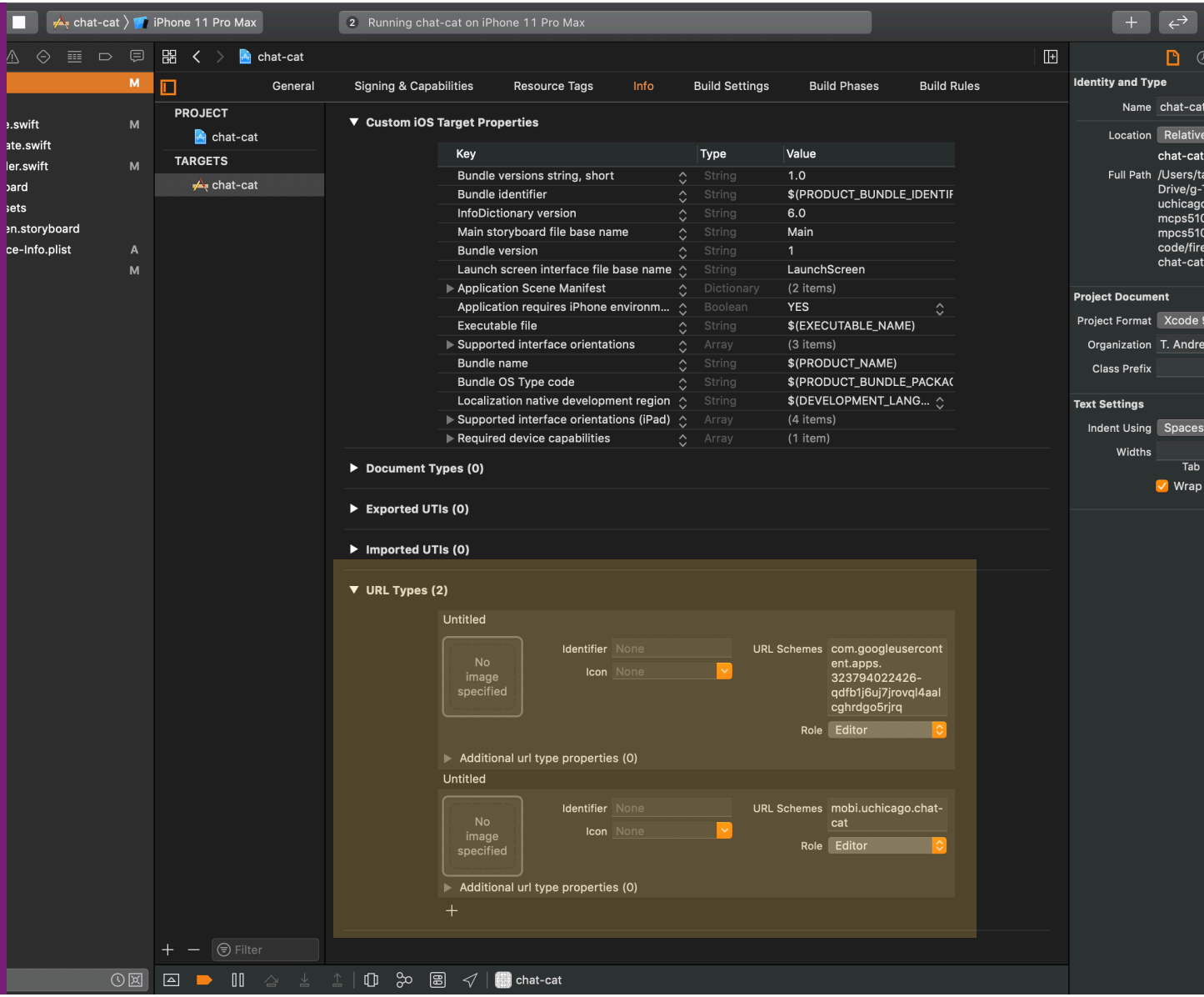


FIREBASE AUTHENTICATION

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

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

  # Pods for chat-cat

  # add the Firebase pod for Google Analytics
  pod 'Firebase/Analytics'

  # add pods for any other desired Firebase products
  pod 'Firebase/Database'
  pod 'Firebase/Auth'
  pod 'GoogleSignIn'
  # https://firebase.google.com/docs/ios/setup#available-pods
end
```

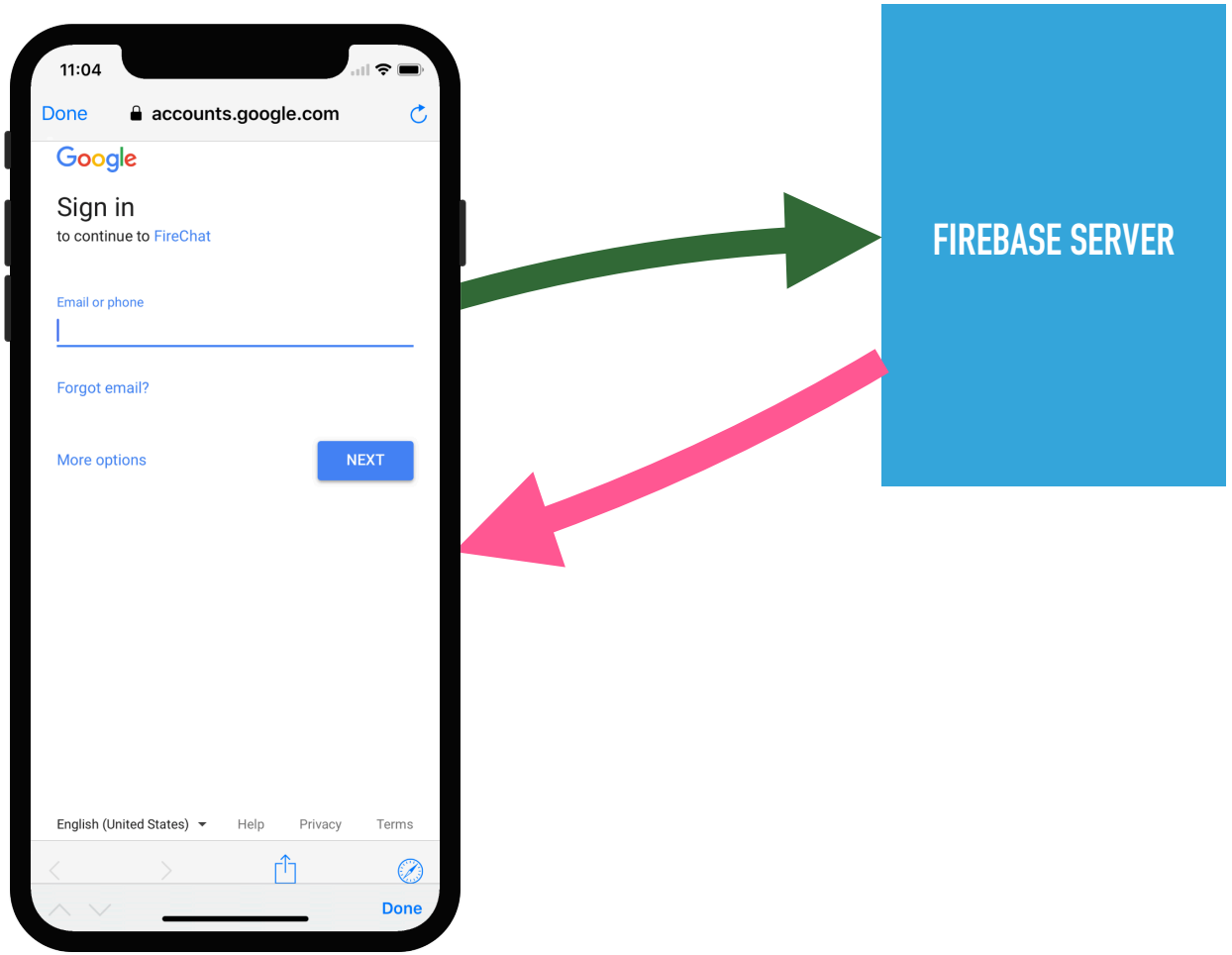
FIREBASE AUTHENTICATION

- Add URL types to the target info
- Provides way to launch app from login callback
- Info in Google-info.plist



FIREBASE AUTHENTICATION

- Flow
  - App launches browser
  - Browser logs in
  - Successful login redirects to URL
  - URL launches app



FIREBASE AUTHENTICATION

- Update the app delegate to handle authentication

```swift
// https://medium.com/@ibjects/google-sign-in-using-firebase-for-ios-straight-forward-step-by-step-guide-b2d2984fbf8e

import UIKit
import Firebase
import GoogleSignIn

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, GIDSignInDelegate {

  func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
      [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    // Use Firebase library to configure APIs
    FirebaseApp.configure()

    GIDSignIn.sharedInstance().clientID = FirebaseApp.app()?.options.clientID
    return true
  }

  /// Open the app from a callback
  @available(iOS 9.0, *)
  func application(_ app: UIApplication, open url: URL, options: [UIApplication.OpenURLOptionsKey : Any] = [:]) ->
      Bool {
    let handled = GIDSignIn.sharedInstance().handle(url)
    return handled
  }

  func application(_ application: UIApplication, open url: URL, sourceApplication: String?, annotation: Any) -> Bool {
    return GIDSignIn.sharedInstance().handle(url)
  }

  /// Google Sign-in Sign-out Delegate Methods
  func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!, withError error: Error?) {•••}

  func sign(_ signIn: GIDSignIn!, didDisconnectWith user: GIDGoogleUser!, withError error: Error!) {
    // Perform any operations when the user disconnects from app here.•••
  }
```

FIREBASE AUTHENTICATION

```
// Open the app from URL callback during Google authentication
@available(iOS 9.0, *)
func application(_ application: UIApplication,
                 open url: URL,
                 options: [UIApplicationOpenURLOptionsKey : Any]) -> Bool {
  return GIDSignIn.sharedInstance().handle(url,
                                           sourceApplication:options[UIApplicationOpenURLOptionsKey.sourceApplication] as? String,
                                           annotation: [:])
}



//
// MARK: - Google Sign In/Out Delegate Methods
//
func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!, withError error: Error?) {•••}
```

- Allows the app to open from a callback
- Standard implementation (not custom to Firebase)

FIREBASE AUTHENTICATION

```swift
func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!, withError error: Error!) {
  //Sign in functionality will be handled here
  if let error = error {
    print(error.localizedDescription)
    return
  }
  guard let auth = user.authentication else { return }
  let credentials = GoogleAuthProvider.credential(withIDToken: auth.idToken, accessToken: auth.accessToken)

  // Pass to Firebase
  Auth.auth().signIn(with: credentials) { (authResult, error) in
    if let error = error {
      print(error.localizedDescription)
    } else {
      print("Login Successful.")

      //This is where you should add the functionality of successful login
      //i.e. dismissing this view or push the home view controller etc
      let user: GIDGoogleUser = GIDSignIn.sharedInstance()!.currentUser
      let fullName = user.profile.name
      let email = user.profile.email
      print(fullName!)
      print(email!)
    }
  }
}
```

AUTHENTICATE FROM GOOGLE

SUCCESS FROM GOOGLE NOW PASS TO FIREBASE

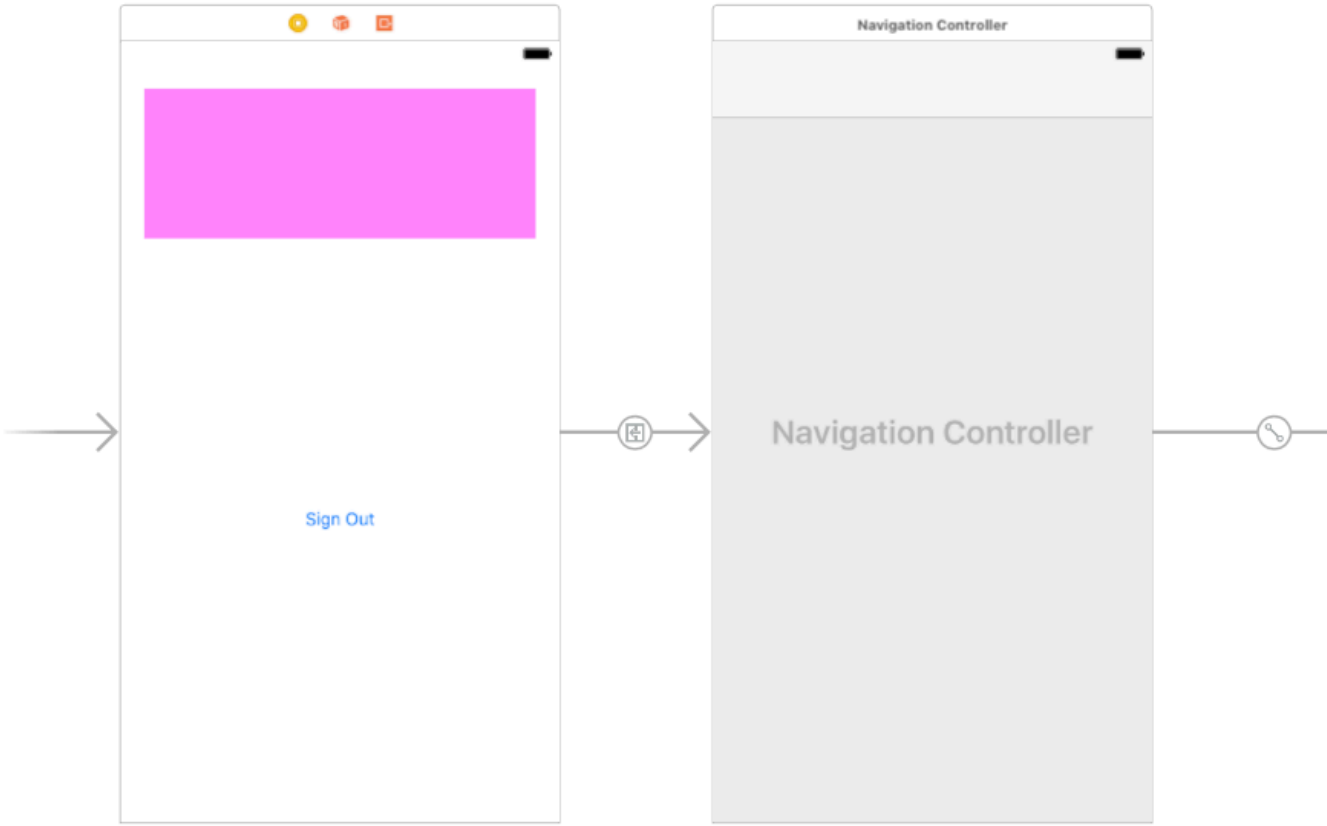
FIREBASE AUTHENTICATION

```
//
// MARK: - Google Sign In/Out Delegate Methods
//
func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!, withError error: Error?) {•••}


func sign(_ signIn: GIDSignIn!, didDisconnectWith user: GIDGoogleUser!, withError error: Error!) {
  // Perform any operations when the user disconnects from app here.
  print("Signed out...")
}
```

DO SOMETHING WHEN WE LOG OUT

FIREBASE AUTHENTICATION

- Login view controller
  - Google button
  - Logout button
  - Authenticate
  - Segue when successful



FIREBASE AUTHENTICATION

- Login view controller
  - Google button
  - Logout button
  - Authenticate
  - Segue when successful

```
import UIKit
import Firebase
import FirebaseAuth
import GoogleSignIn

class ViewController: UIViewController, GIDSignInDelegate {

  // Google button
  @IBOutlet weak var LoginButton: GIDSignInButton!

  @IBAction func googleSignInPressed(_ sender: Any) {
    GIDSignIn.sharedInstance().signIn()
  }

  @IBAction func tapLogout(_ sender: Any) {
    //GIDSignIn.sharedInstance().signOut()
    let firebaseAuth = Auth.auth()
    do {
      try firebaseAuth.signOut()
    } catch let signOutError as NSError {
      print ("Error signing out: %@", signOutError)
    }
    print("signed out")
  }

  override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view.
    GIDSignIn.sharedInstance()?.presentingViewController = self
    GIDSignIn.sharedInstance().delegate = self
  }

  override func viewDidAppear(_ animated: Bool) {

    let user = Auth.auth().currentUser
    if user?.uid == nil {
      print("User not logged in")
    } else {
      print("User logged in:",Auth.auth().currentUser?.email ?? "not logged i")
    }

    // Setup listener
    Auth.auth().addStateDidChangeListener() { auth, user in

        // Automatically segue after a successfull login; users seem to
        // hang around while developing, you may need to clean and remove
        // app from simulator
        if user != nil {
          self.performSegue(withIdentifier: "segueToMessages", sender: nil)
        }
      }
  }
```

SEGUE IF AUTHENTICATED USER

FIREBASE AUTHENTICATION

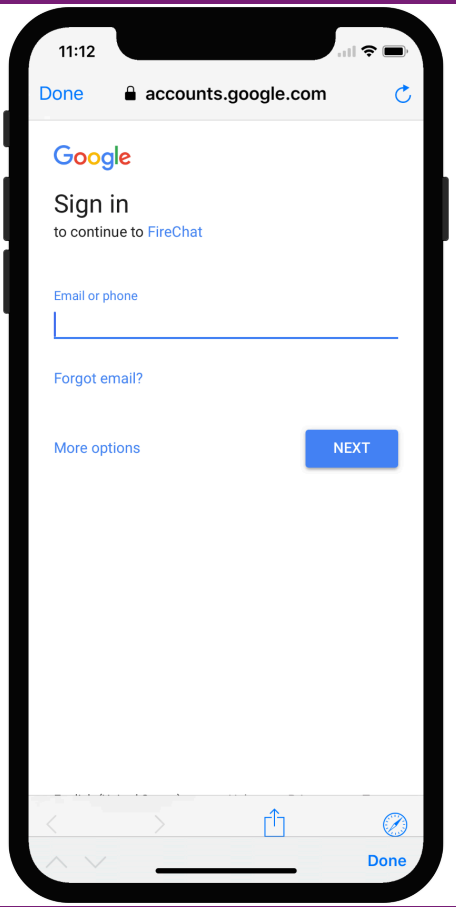
```
override func viewDidAppear(_ animated: Bool) {

  let user = Auth.auth().currentUser
  if user?.uid == nil {
    print("User not logged in")
  } else {
    print("User logged in:",Auth.auth().currentUser?.email ?? "not logged in")
  }

  // Setup listener
  Auth.auth().addStateDidChangeListener() { auth, user in

      // Automatically segue after a successfull login; users seem to
      // hang around while developing, you may need to clean and remove
      // app from simulator
      if user != nil {
        self.performSegue(withIdentifier: "segueToMessages", sender: nil)
      }
    }
}
```

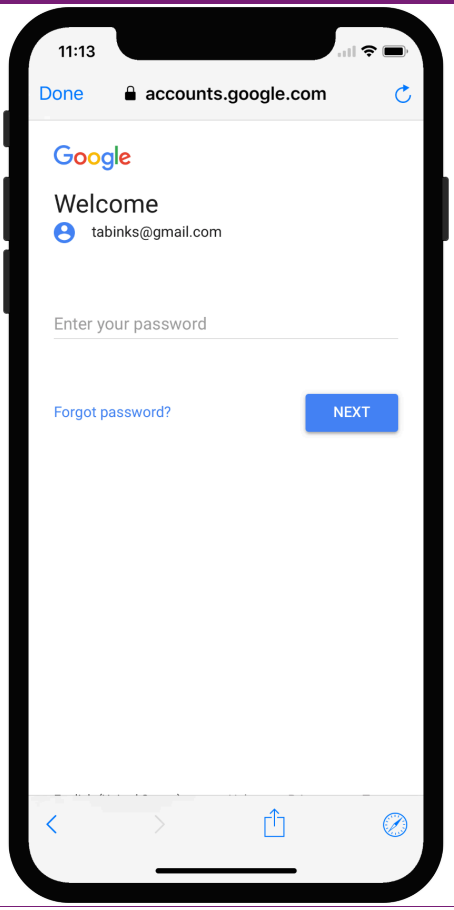
ADD OBSERVER IN CASE USER LOGS OUT

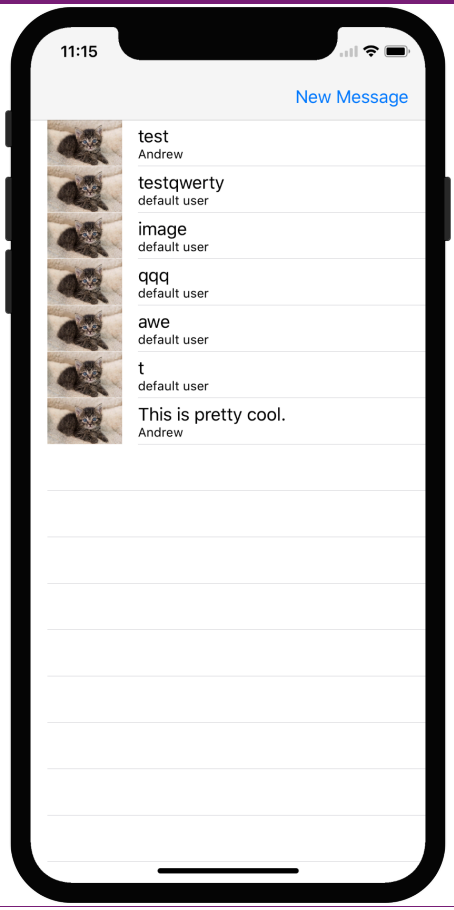
FIREBASE AUTHENTICATION

```
@IBAction func tapLogout(_ sender: Any) {
  //GIDSignIn.sharedInstance().signOut()
  let firebaseAuth = Auth.auth()
  do {
    try firebaseAuth.signOut()
  } catch let signOutError as NSError {
    print ("Error signing out: %@", signOutError)
  }
  print("signed out")
}
```

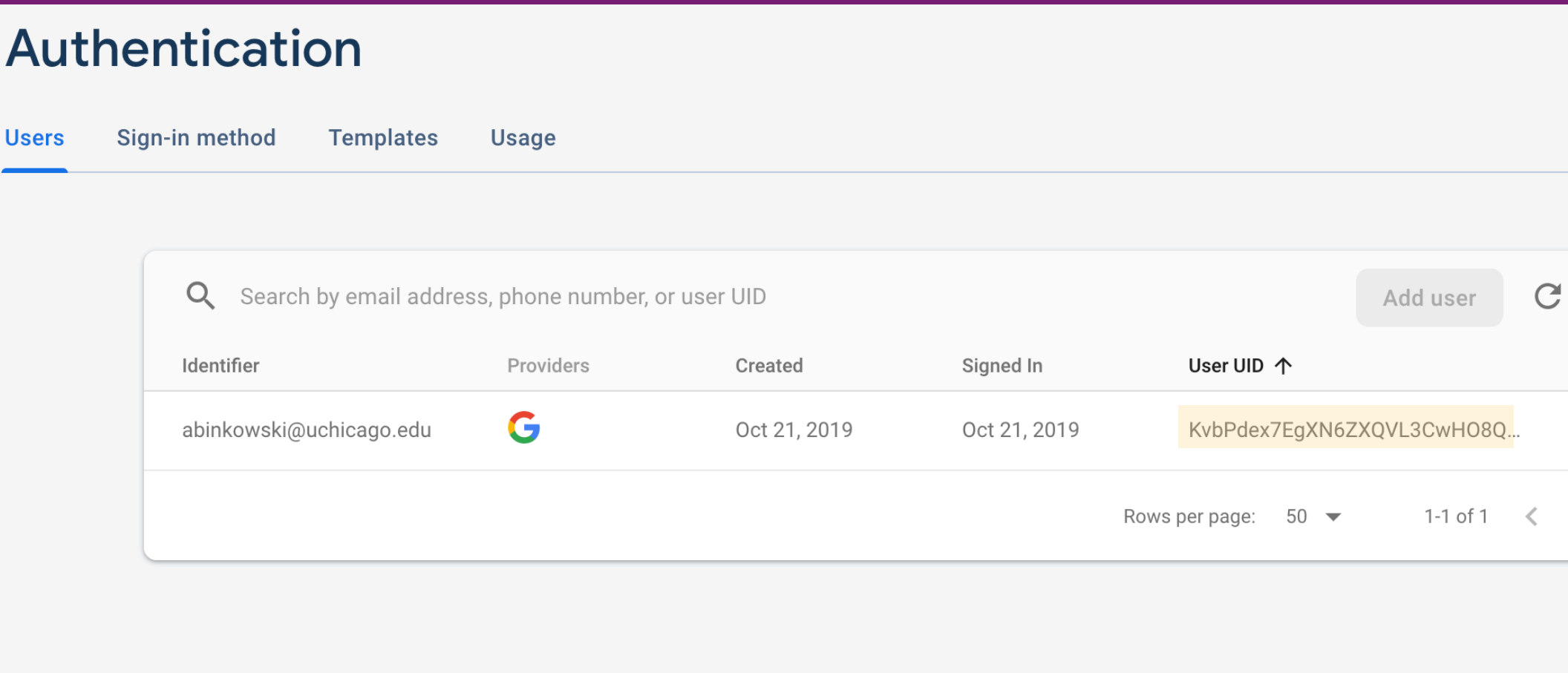
FIREBASE AUTHENTICATION







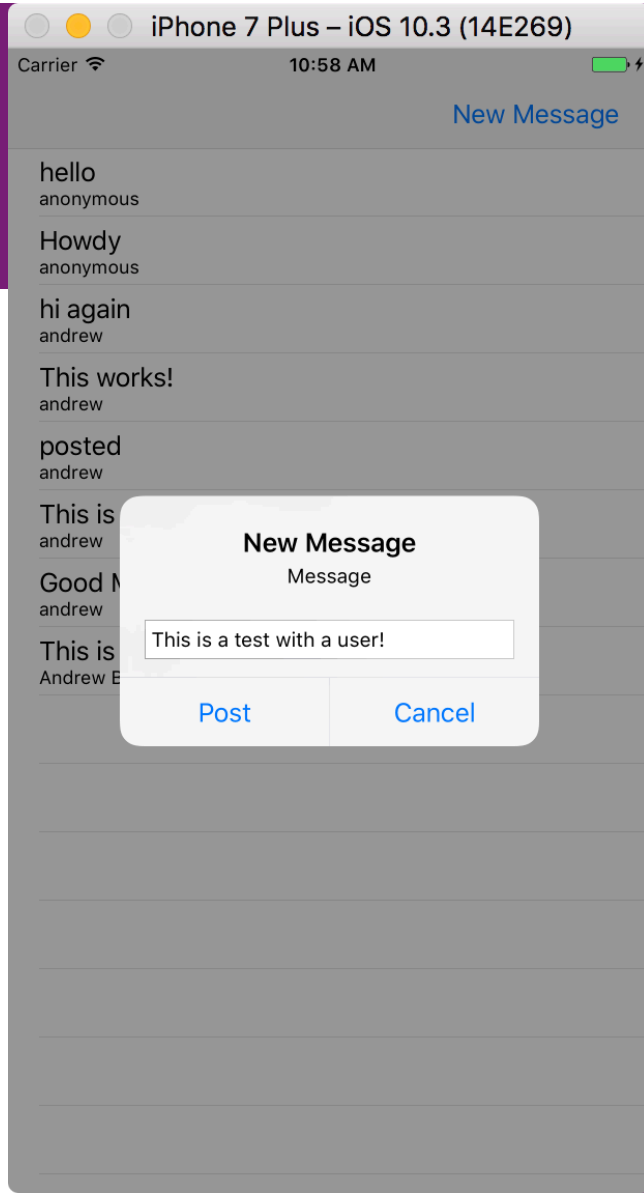
FIREBASE AUTHENTICATION



FIREBASE AUTHENTICATION

firechat-66f87

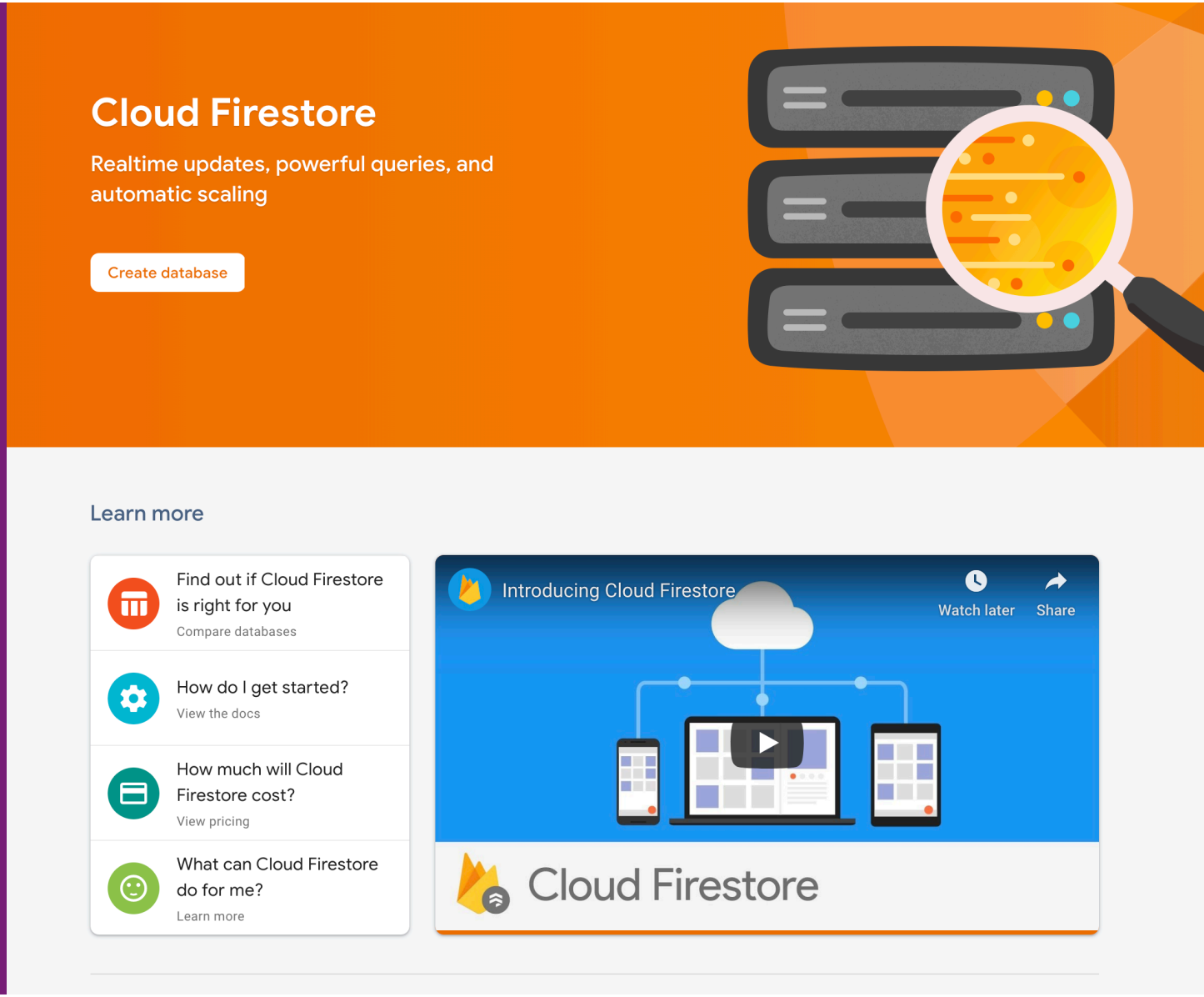
- **messages**
  - **1**
  - **2**
  - **3**
  - **4**
  - **-Ki4SXKxO7AnysU9doF8**
  - **-Ki4Sgutqv5wPpXu94N4**
  - **-Ki5PEI6V57IwidD-a_N**
  - **-Ki5vlIWFwAHsVOWydzz**
    - **name:** "Andrew Binkowski"
    - **text:** "This is a test with a user!"



FIREBASE DATABASES

FIREBASE

- Options for storing data
  - Realtime Database (classic)
  - Cloud Firestore



DATABASE

Choose a Database: Cloud Firestore or Realtime Database

Firebase offers two cloud-based, client-accessible database solutions that support realtime data syncing:

- **Cloud Firestore** is Firebase's newest database for mobile app development. It builds on the successes of the Realtime Database with a new, more intuitive data model. Cloud Firestore also features richer, faster queries and scales further than the Realtime Database.
- **Realtime Database** is Firebase's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in realtime.

Which database is right for your project?

HTTPS://FIREBASE.GOOGLE.COM/DOCS/DATABASE/RTDB-VS-FIRESTORE?AUTHUSER=0

We recommend Cloud Firestore for most developers starting a new project. Cloud Firestore offers additional functionality, performance, and scalability on an infrastructure designed to support more powerful features in future

DATABASE

PROS

- Realtime
  - Giant JSON tree
  - Simple queries
  - Automatic scaling to a point
  - Old way, but useful for more 'realtime' apps
- Firestore
  - Document based
  - More complex queries
  - Automatic scaling
  - New way, more robust

DATABASE

PROS

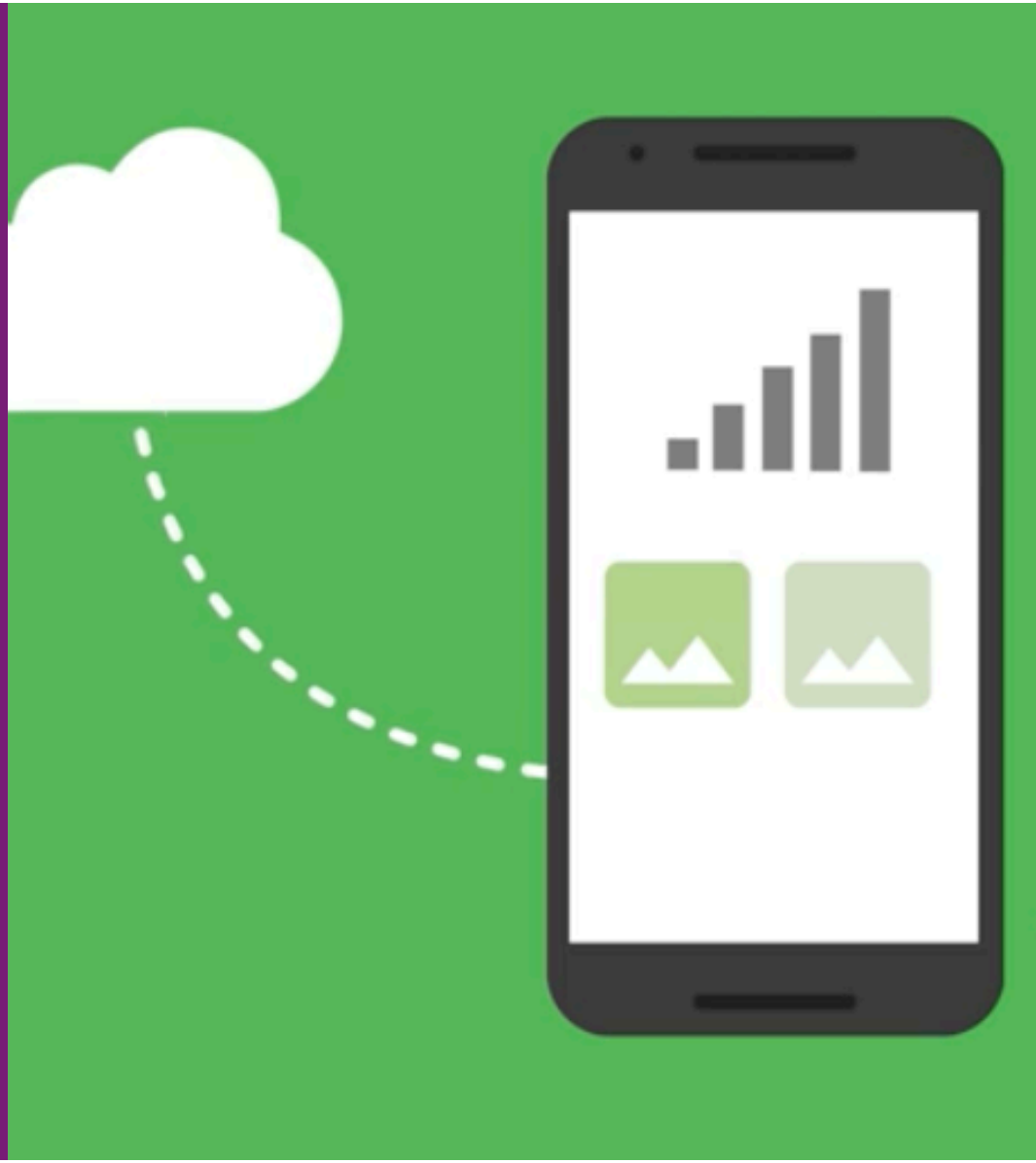
- Realtime
  - Giant JSON tree
  - Simple queries
  - Automatic scaling t
  - Old way, but useful for more 'realtime' apps
- Firestore
  - ment based
  - complex queries
  - natic scaling
  - New way, more robust

CAN USE BOTH IN THE SAME PROJECT (APP)

REALTIME DATABASE

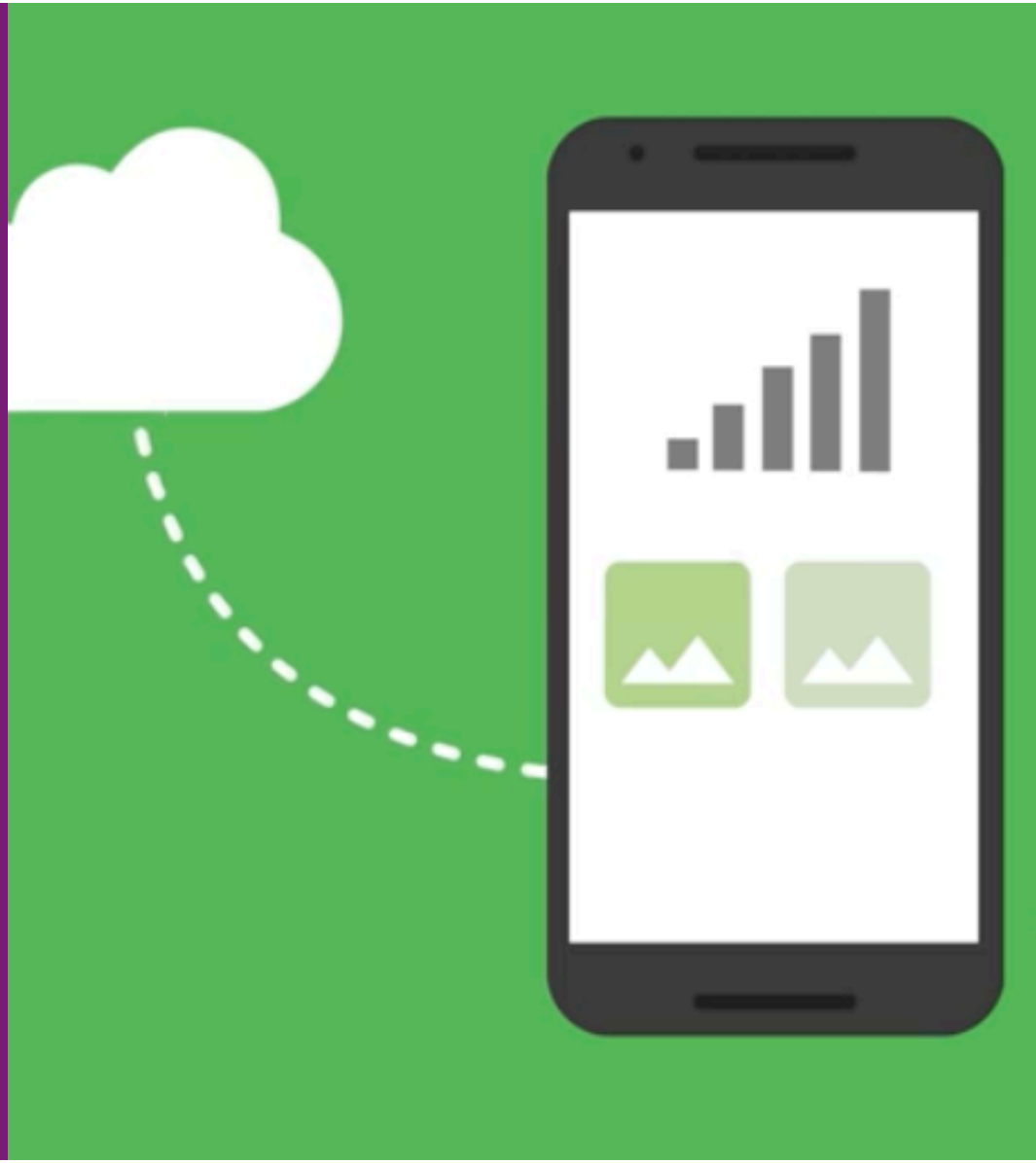
REALTIME DATABASE

- Store and sync data with NoSQL cloud database
- Data is synced across all clients in realtime, and remains available when your app goes offline
  - https://youtu.be/O17OWyx08Cg?list=PLl-K7zZEsYLmOF_07IayrTntevxtbUxDL

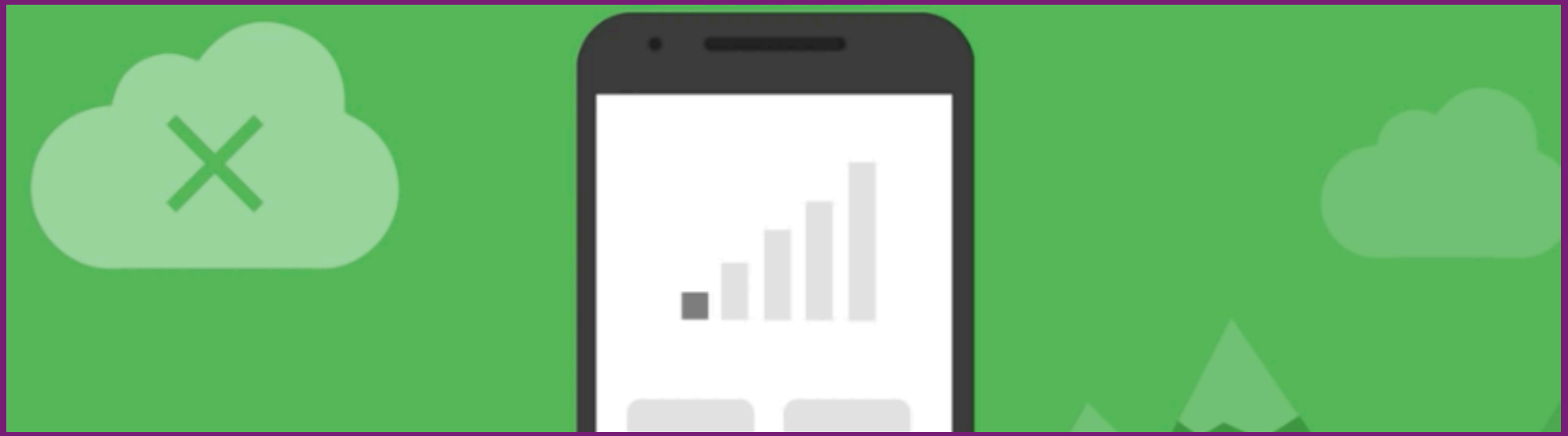


REALTIME DATABASE

- Data is stored as JSON
- Synchronized in realtime to all clients
  - Milliseconds
- All networking code is included as part of the SDK



REALTIME DATABASE



- Data is stored in a local cache while the user in offline
- Changes are automatically synced when the device comes back on

REALTIME DATABASE

- Implementation steps

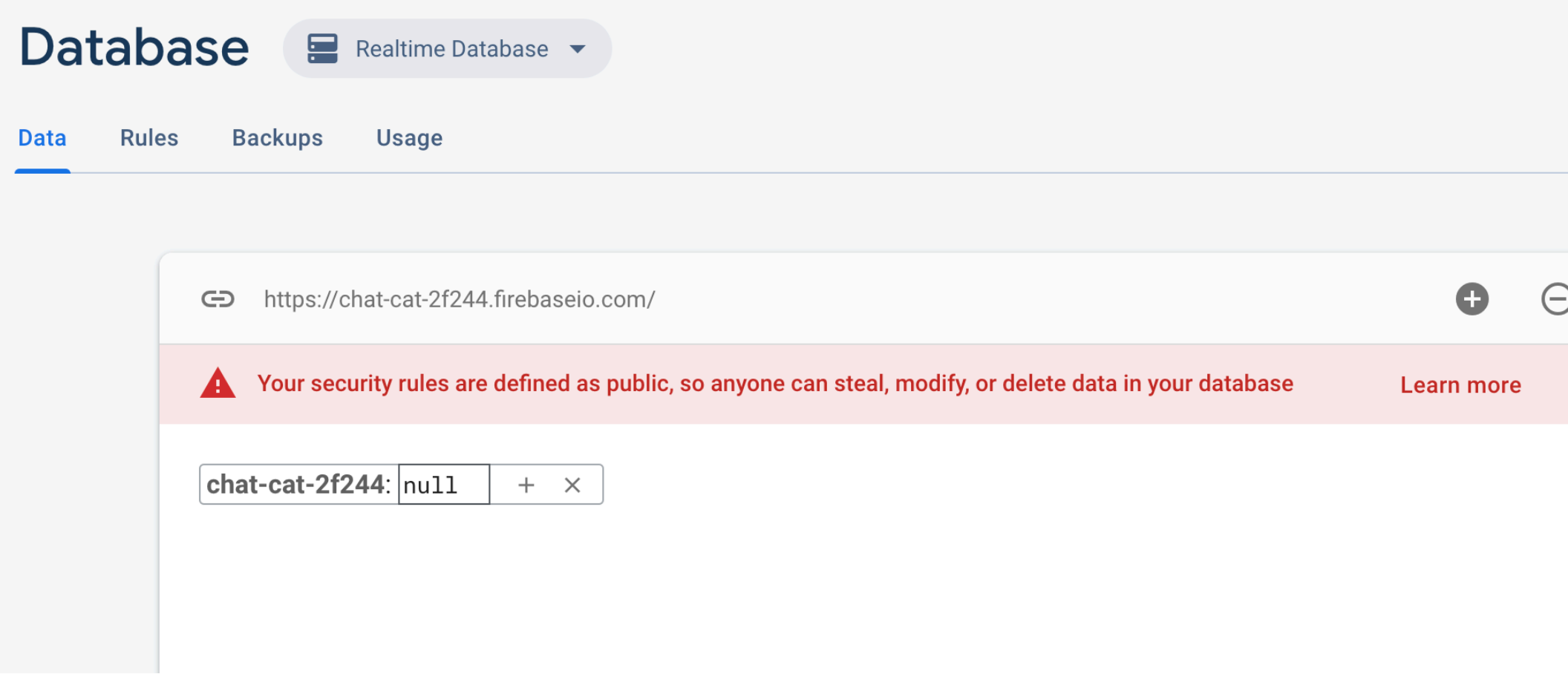
1	Integrate the Firebase Realtime Database SDKs	Quickly include clients via Gradle, CocoaPods, or a script include.
2	Create Realtime Database References	Reference your JSON data, such as "users/user:1234/phone_number" to set data or subscribe to data changes.
3	Set Data and Listen for Changes	Use these references to write data or subscribe to changes.
4	Enable Offline Persistence	Allow data to be written to the device's local disk so it can be available while offline.
5	Secure your data	Use Firebase Realtime Database Security Rules to secure your data.

REALTIME DATABASE

```
target 'chat-cat' do
  # Comment the next line if you d                    se dynamic fr
  use_frameworks!

  # Pods for chat-cat

  # add the Firebase pod for Googl
  pod 'Firebase/Analytics'

  # add pods for any other desired Firebase products
  pod 'Firebase/Database'
  # https://firebase.google.com/docs/ios/setup#available-pods
end
```

NEED TO POD INSTALL AGAIN

REALTIME DATABASE



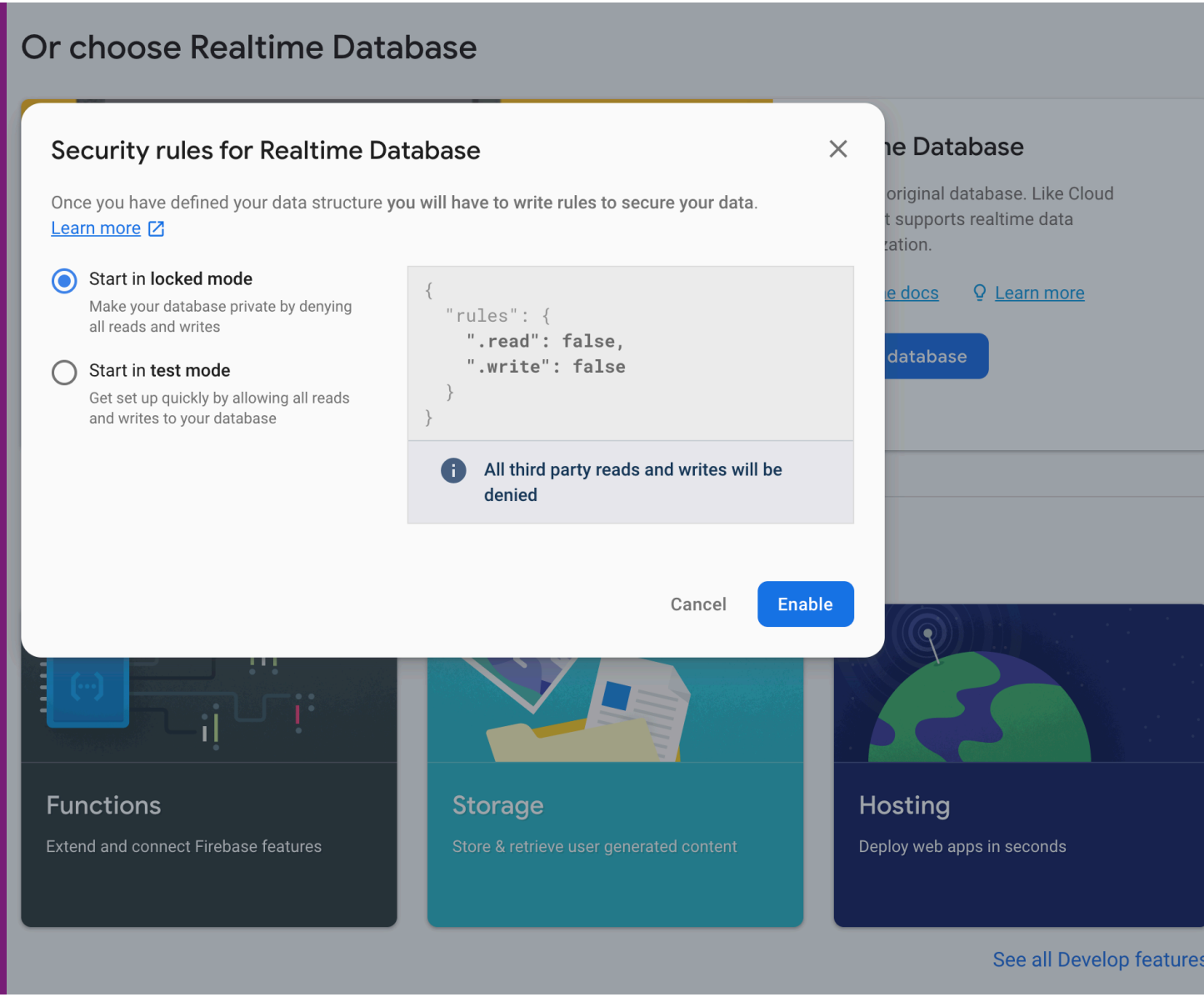
DATABASE SECURITY

REALTIME DATABASE

- Security is handled through "Firebase Realtime Database Security" rules
  - You set permissions on the database
  - Authentication is handled by "Firebase Authentication"

REALTIME DATABASE

- Rules are set through the console
- Command line



REALTIME DATABASE

```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

- Default
  - Requires authentication

REALTIME DATABASE

```
// These rules give anyone, even people who are not users of your app,
// read and write access to your database
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

- Public
  - Anyone

REALTIME DATABASE

```
// These rules grant access to a node matching the authenticated
// user's ID from the Firebase auth token
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

- User
  - Each user has their own database

REALTIME DATABASE

```
// These rules don't allow anyone read or write access to your database
{
  "rules": {
    ".read": false,
    ".write": false
  }
}
```

- Private
  - No one can access outside of console

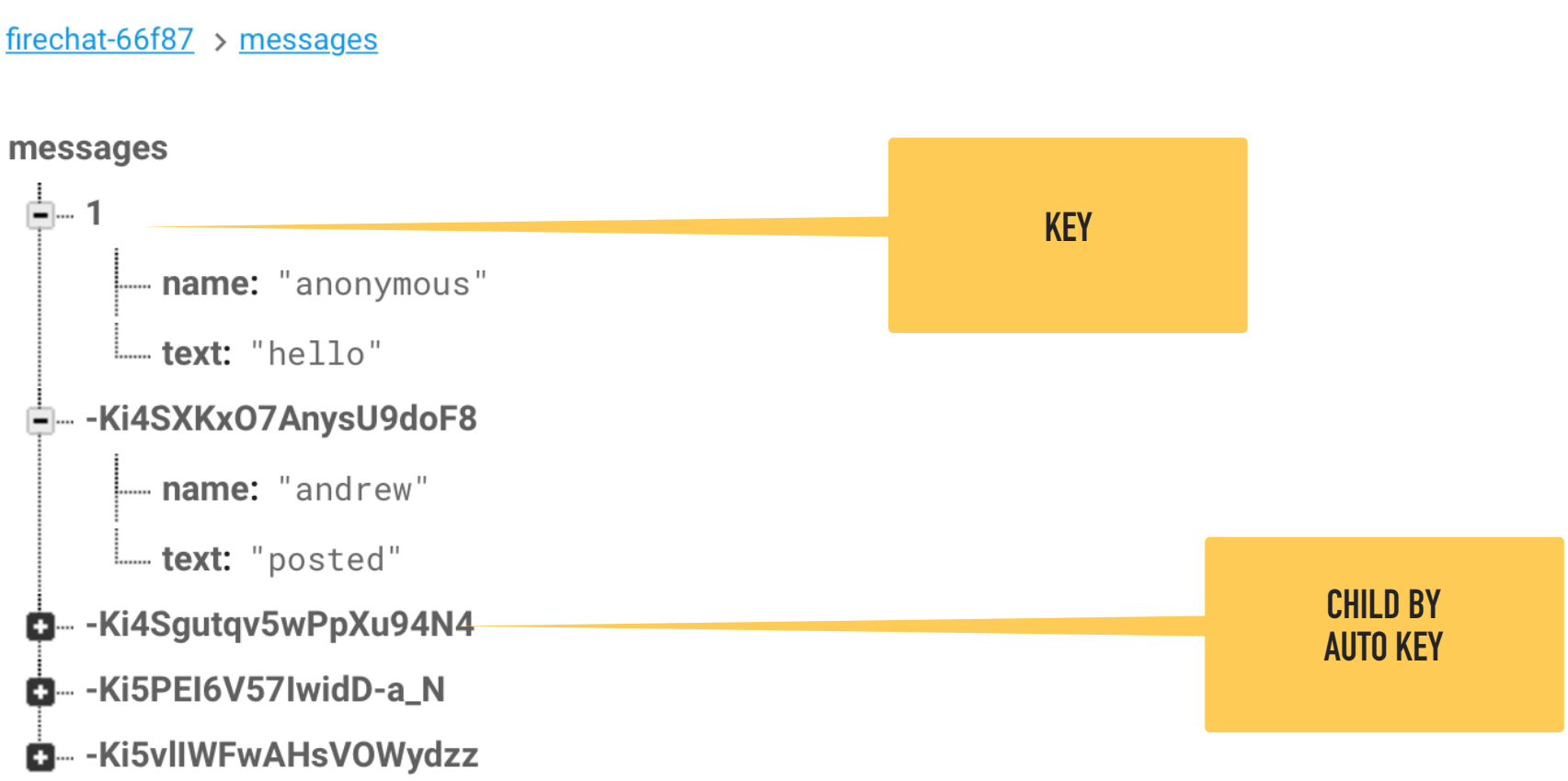
DATA STRUCTURE

DATA STRUCTURE

- JSON trees stored online
- Use multiple trees (not linked)
- Data is represented as a node accessible by a key
  - Use your own key (string)
  - `childByAutoId`

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

DATA STRUCTURE



DATA STRUCTURE

BEST PRACTICES FOR DATA

- Avoid nesting data
  - Fetch returns all data
  - Access control
  - Parsing in Swift isn't fun

```
{
  // This is a poorly nested data architecture,
  // of the "chats" node to get a list of conve
  // potentially downloading hundreds of megaby
  "chats": {
    "one": {
      "title": "Historical Tech Pioneers",
      "messages": {
        "m1": { "sender": "ghopper", "message":
        "m2": { ... },
        // a very long list of messages
      }
    },
    "two": { ... }
  }
}
```

DATA STRUCTURE

BEST PRACTICES FOR DATAA

- Flatten data
  - Isolate into different calls
  - Use metadata for faster fetching
  - Maintain unique references

```
{
  // 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.",
```

DATA STRUCTURE

BEST PRACTICES FOR DATAA

- Flatten data
  - Isolate into different calls
  - Use metadata for faster fetching
  - Maintain unique references

```
{
  // 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.",
```

THE KEY "ONE" IS USED ACROSS DIFFERENT DICTIONARIES

ALL REFER TO THE SAME INSTANCE OF THE CHAT "ONE"

DATA STRUCTURE

BEST PRACTICES FOR DATA

- Don't be afraid of redundant information
- Think "ease of retrieval" instead of efficient storage

```
// 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
      }
    },
    ...
  }
}
```

/USERS/$UID/GROUPS/$GROUP_ID

DATA STRUCTURE

BEST PRACTICES FOR DATA

- Don't be afraid of redundant information
- Think "ease of retrieval" instead of efficient storage

```
// 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
      }
    },
    ...
  }
}
```

/USERS/$UID/GROUPS/$GROUP_ID

SELECT ALL FROM GROUPS WHERE MEMBERS CONTAINS GHOPPER

DATA STRUCTURE

- There is a CLI tool that can profile your database
  - Speed
  - Bandwidth
  - Query Indices

```
{
  "scores": {
    "bruhathkayosaurus" : 55,
    "lambeosaurus" : 21,
    "linhenykus" : 80,
    "pterodactyl" : 93,
    "stegosaurus" : 5,
    "triceratops" : 22
  }
}
```

DATA STRUCTURE

- You can specify which keys to index on to improve performance

```
{
  "scores": {
    "bruhathkayosaurus" : 55,
    "lambeosaurus" : 21,
    "linhenykus" : 80,
    "pterodactyl" : 93,
    "stegosaurus" : 5,
    "triceratops" : 22
  }
}
```

```
{
  "rules": {
    "scores": {
      ".indexOn": ".value"
    }
  }
}
```

READ DATA

READ AND WRITE DATA

- Get a reference to the database

```
/// Database reference
let reference = Database.database().reference()
```

- Access nodes by key
- Set values by key

```
self.ref.child("users/(user.uid)/username").setValue(username)
```

```
self.ref.child("users").child(user.uid).setValue(["username": username])
```

READ AND WRITE DATA

- Observe changes to the database
- Take a snapshot when you need it

```
override func viewDidAppear(_ animated: Bool) {
  super.viewDidAppear(animated)

  // Set up observer for the messages data
  reference.observe(.value, with: { snapshot in
    print(snapshot.value!)

    // Create an array of the incoming data; convert it; then update the tab
    var freshData = [MessageItem]()
    for item in snapshot.children {
      let messageItem = MessageItem(snapshot: item as! DataSnapshot)
      freshData.append(messageItem)
    }

    self.messages = freshData
    print(self.messages)
    self.tableView.reloadData()

  })

  // Keep track of the user
  Auth.auth().addStateDidChangeListener { auth, user in
    guard let user = user else { return }
    self.user = user
  }

}
```

OBSERVE CHANGERS TO DATA

DATA RETURNED AS SNAPSHOT

READ AND WRITE DATA

- The values are returned as native Swift types not JSON
  - Dictionary, Array, String, Int, Double

READ AND WRITE DATA

- Best practice to convert to custom struct

```
struct MessageItem {

  let key: String
  let name: String
  let text: String
  let ref: DatabaseReference?

  init(name: String, text: String, key: String = "") {
    self.key = key
    self.name = name
    self.text = text
    self.ref = nil
  }

  init(snapshot: DataSnapshot) {
    key = snapshot.key
    let snapshotValue = snapshot.value as! [String: AnyObject]
    name = snapshotValue["name"] as! String
    text = snapshotValue["text"] as! String
    ref = snapshot.ref
  }

}
```

READ AND WRITE DATA

- Best practice to convert to custom struct

```
for item in snapshot.children {
  let messageItem = MessageItem(snapshot: item as! DataSnapshot)
  freshData.append(messageItem)
}
```

```
struct MessageItem {

  let key: String
  let name: String
  let text: String
  let ref: DatabaseReference?

  init(name: String, text: String, key: String =
    self.key = key
    self.name = name
    self.text = text
    self.ref = nil
  }

  init(snapshot: DataSnapshot) {
    key = snapshot.key
    let snapshotValue = snapshot.value as! [Strir
    name = snapshotValue["name"] as! String
    text = snapshotValue["text"] as! String
    ref = snapshot.ref
  }

}
```

READ AND WRITE DATA

- Best practice to convert to custom struct

```
struct MessageItem {

  let key: String
  let name: String
  let text: String
  let ref: DatabaseReference?

  init(name: String, text: String, key: String = "") {
    self.key = key
    self.name = name
    self.text = text
    self.ref = nil
  }

  init(snapshot: DataSnapshot) {
    key = snapshot.key
    let snapshotValue = snapshot.value as! [String: AnyObject]
    name = snapshotValue["name"] as! String
    text = snapshotValue["text"] as! String
    ref = snapshot.ref
  }

}
```

STRUCT DOESN'T KNOW ABOUT FIREBASE...KEEP A DIRECT REFERENCE TO IT

READ AND WRITE DATA

```
let userID =    Auth.auth()?.currentUser?.uid
ref.child("users").child(userID!).observeSingleEvent(of: .value, with: { (snapshot) in
  // Get user value
  let value = snapshot.value as? NSDictionary
  let username = value?["username"] as? String ?? ""
  let user = User.init(username: username)

  // ...
  }) { (error) in
    print(error.localizedDescription)
}
```

- Single callback for data not expected to change (eg. game levels, locations, members, etc.)

WRITE DATA

WRITE DATA

```
let reference =    Database.database().reference().child("messages")
var messages =  [MessageItem]()
var user:    User?


@IBAction func tapNewMessage(_ sender: UIBarButtonItem) {
  let alert = UIAlertController(title: "New Message",
                                message: "Message",
                                preferredStyle: .alert)

  let saveAction = UIAlertAction(title: "Post",
                                 style: .default) { _ in

                                   guard let textField = alert.textFields?.first,
                                     let text = textField.text else {
                                       return
                                   }

                                   let data = ["name":self.user?.displayName,"text": text]
                                   self.reference.childByAutoId().setValue(data)

  }
```

ADD A NEW MESSAGE

WRITE DATA

```
let reference =    Database.database().reference().child("messages")
var messages = [MessageItem]()
var user:    User?


@IBAction func tapNewMessage(_ sender: UIBarButtonItem) {
  let alert = UIAlertController(title: "New Message",
                                message: "Message",
                                preferredStyle: .alert)

  let saveAction = UIAlertAction(title: "Post",
                                 style: .default) { _ in

                                  guard let textField = alert.textFields?.first,
                                    let text = textField.text else {
                                      return
                                  }

                                  let data = ["name":self.user?.displayName,"text": text]
                                  self.reference.childByAutoId().setValue(data)

  }
```

CAN LEAVE FIELDS EMPTY

WRITE DATA

- Data doesn't have to be homogenous

WRITE DATA

- Overwrite the entire node and children

```
self.ref.child("users").child(user.uid).setValue(["username": username])
```

- Updates a value

```
self.ref.child("users/(user.uid)/username").setValue(username)
```

WRITE DATA

```
let key = ref.child("posts").childByAutoId().key
let post = ["uid": userID,
            "author": username,
            "title": title,
            "body": body]
let childUpdates = ["/posts/\(key)": post,
                    "/user-posts/\(userID)/\(key)/": post]
ref.updateChildValues(childUpdates)
```

- Use `updateChildValues` to make changes to multiple locations

WRITE DATA

```
let key = ref.child("posts").childByAutoId().key
let post = ["uid": userID,
            "author": username,
            "title": title,
            "body": body]
let childUpdates = ["/posts/\(key)": post,
                    "/user-posts/\(userID)/\(key)/": post]
ref.updateChildValues(childUpdates)
```

- Delete data by call `deleteValue` on a node, set to `nil`
- Multiple children delete using `updateChildValues` to nil

WORKING WITH LISTS OF DATA

WORKING WITH LISTS OF DATA

- Using `childByAutoId` keeps data sorted by timestamp and unique
- Call `getKey` to find the unique key of a node
- Use the keys for flattening your data structure

```
let reference = FIRDatabase.database().reference().child("messages")
var messages = [MessageItem]()
var user: FIRUser?


@IBAction func tapNewMessage(_ sender: UIBarButtonItem) {
  let alert = UIAlertController(title: "New Message",
                                message: "Message",
                                preferredStyle: .alert)

  let saveAction = UIAlertAction(title: "Post",
                                 style: .default) { _ in

                                  guard let textField = alert.textFields?.first,
                                    let text = textField.text else {
                                      return
                                  }

                                  let data = ["name":self.user?.displayName,"text": text]
                                  self.reference.childByAutoId().setValue(data)
  }
```

WORKING WITH LISTS OF DATA

```
reference.observe(.value, with: { snapshot in
  print(snapshot.value!)

  var freshData = [MessageItem]()
  for item in snapshot.children {
```

- Returns everything
- Returns an array

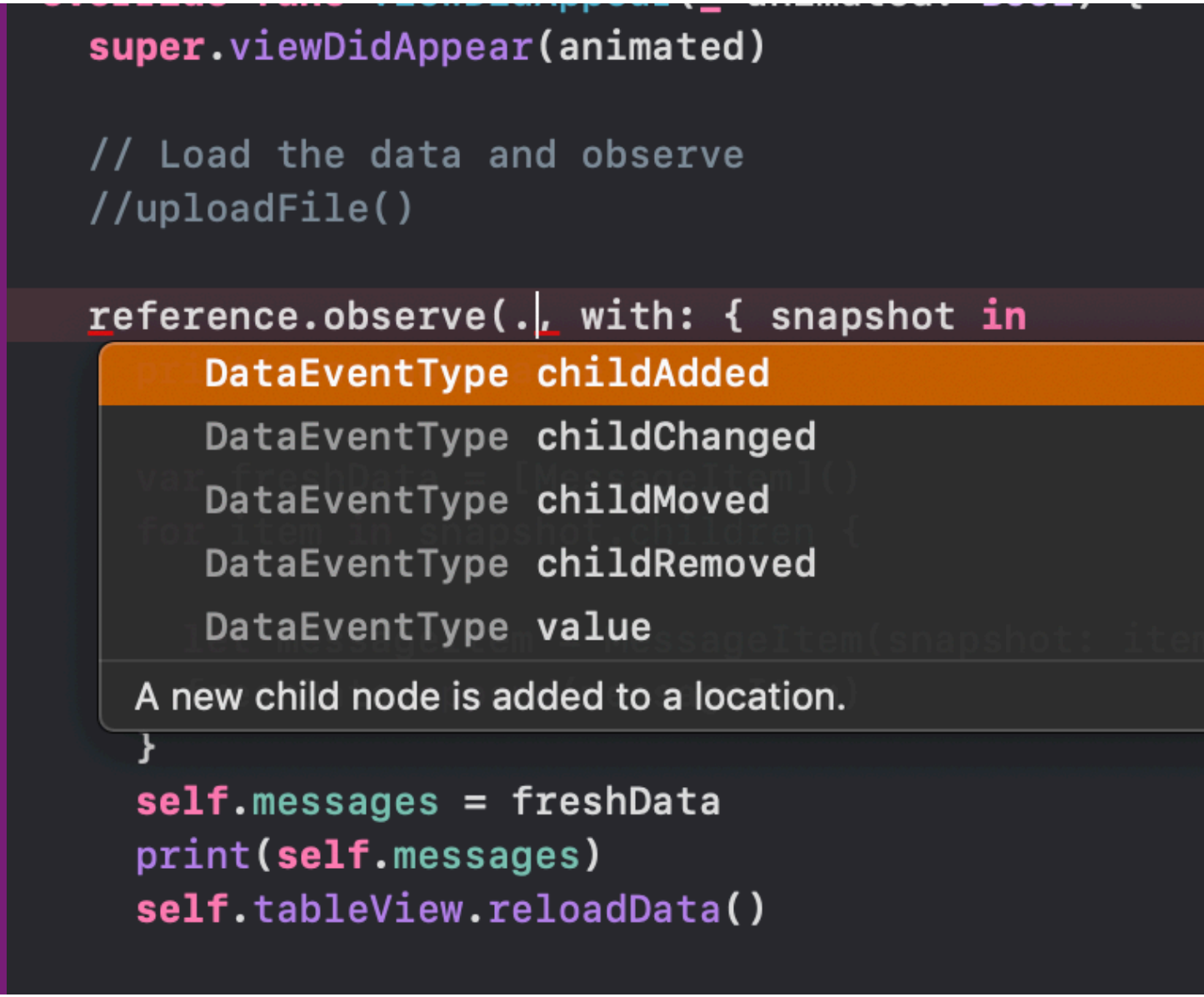
WORKING WITH LISTS OF DATA

Event type	Typical usage
`DataEventTypeChildAdded`	Retrieve lists of items or listen for additions to a list of items. This event is triggered once for each existing child and then again every time a new child is added to the specified path. The listener is passed a snapshot containing the new child's data.
`DataEventTypeChildChanged`	Listen for changes to the items in a list. This event is triggered any time a child node is modified. This includes any modifications to descendants of the child node. The snapshot passed to the event listener contains the updated data for the child.
`DataEventTypeChildRemoved`	Listen for items being removed from a list. This event is triggered when an immediate child is removed.The snapshot passed to the callback block contains the data for the removed child.
`DataEventTypeChildMoved`	Listen for changes to the order of items in an ordered list. This event is triggered whenever an update causes reordering of the child. It is used with data that is ordered by **`queryOrderedByChild`** or **`queryOrderedByValue`**.

- Monitor for child events

WORKING WITH LISTS OF DATA

- Monitor for child events
- Think about how your data is structured



WORKING WITH LISTS OF DATA

```
// Listen for new comments in the Firebase database
commentsRef.observe(.childAdded, with: { (snapshot) -> Void in
  self.comments.append(snapshot)
  self.tableView.insertRows(at: [IndexPath(row: self.comments.count-1, section: self.kSectionComment
})
// Listen for deleted comments in the Firebase database
commentsRef.observe(.childRemoved, with: { (snapshot) -> Void in
  let index = self.indexOfMessage(snapshot)
  self.comments.remove(at: index)
  self.tableView.deleteRows(at: [IndexPath(row: index, section: self.kSectionComments)], with: UITab
})
```

RETURNS THE CHANGED NODE ONLY IN THE SNAPSHOT

WORKING WITH LISTS OF DATA

Method	Usage
queryOrderedByKey	Order results by child keys.
queryOrderedByValue	Order results by child values.
queryOrderedByChild	Order results by the value of a specified child key.

KEYS HAVE INHERENT TIME STAMP

- Query for sorted data

```
let myTopPostsQuery = (ref.child("user-
posts").child(getUid())).queryOrdered(byChild: "starCount")
```

WORKING WITH LISTS OF DATA

- Filtering returned data

Method	Usage
queryLimitedToFirst	Sets the maximum number of items to return from the beginning of the ordered list of results.
queryLimitedToLast	Sets the maximum number of items to return from the end of the ordered list of results.
queryStartingAtValue	Return items greater than or equal to the specified key or value, depending on the order-by method chosen.
queryEndingAtValue	Return items less than or equal to the specified key or value, depending on the order-by method chosen.
queryEqualToValue	Return items equal to the specified key or value, depending on the order-by method chosen.

```
// Last 100 posts, these are automatically the 100 most recent
// due to sorting by push() keys
let recentPostsQuery = (ref?.child("posts").queryLimited(toFirst: 100))!
```

STORE DATA OFFLINE

STORE DATA OFFLINE

- Firebase keeps a local cache that it uses when there is no network connectivity
- The app functions the same
  - Callbacks are made
  - Data can be added, deleted, edited
- Changes are stored in queue
- Once back online, all changes are applied to the datastore
- Stored for active listeners by default

STORE DATA OFFLINE

```
let scoresRef =    Database.database().referenceWithPath("scores")
scoresRef.keepSynced(true)
```

- Stored for active listeners by default
- Request other paths be stored

STORE DATA OFFLINE

```
let scoresRef =     Database.database().referenceWithPath("scores")
scoresRef.queryOrderedByValue().queryLimitedToLast(4).observeEventType(.ChildAdded, withBlock: { snapshot in
  print("The \(snapshot.key) dinosaur's score is \(snapshot.value)")
})
```

- Data queries use the cached version (if available)

STORE DATA OFFLINE

```
// Use Firebase library to configure APIs
FirebaseApp.configure()
Database.database().isPersistenceEnabled = true
```

- On by default 😀

STORE DATA OFFLINE

```
let presenceRef =    Database.database().referenceWithPath("message");
// Write a string when this client loses connection
presenceRef.onDisconnectSetValue("I disconnected!")
```

```
let userLastOnlineRef =    Database.database().referenceWithPath("users/joe/lastOnline")
userLastOnlineRef.onDisconnectSetValue(FIRServerValue.timestamp())
```

- Manage presence (ie who is connected)

MANAGE PRESENCE

MANAGE PRESENCE

- Identify who is online
  - Add a child to user
  - Update when they are online
  - Remove the child when they are off

```
// since I can connect from multiple devices, we store each co
// any time that connectionsRef's value is null (i.e. has no c
let myConnectionsRef = FIRDatabase.database().referenceWithPat

// stores the timestamp of my last disconnect (the last time I
let lastOnlineRef = FIRDatabase.database().referenceWithPath("

let connectedRef = FIRDatabase.database().referenceWithPath(".

connectedRef.observeEventType(.Value, withBlock: { snapshot in
  // only handle connection established (or I've reconnected a
  guard let connected = snapshot.value as? Bool where connecte
    return
  }
  // add this device to my connections list
  // this value could contain info about the device or a times
  let con = myConnectionsRef.childByAutoId()
  con.setValue("YES")

  // when this device disconnects, remove it
  con.onDisconnectRemoveValue()

  // when I disconnect, update the last time I was seen online
  lastOnlineRef.onDisconnectSetValue(FIRServerValue.timestamp(
})
```

MANAGE PRESENCE

```
// since I can connect from multiple devices, we store each connection instance separately
// any time that connectionsRef's value is null (i.e. has no children) I am offline
let myConnectionsRef = FIRDatabase.database().referenceWithPath("users/joe/connections")

// stores the timestamp of my last disconnect (the last time I was seen online)
let lastOnlineRef = FIRDatabase.database().referenceWithPath("users/joe/lastOnline")

let connectedRef = FIRDatabase.database().referenceWithPath(".info/connected")

connectedRef.observeEventType(.Value, withBlock: { snapshot in
  // only handle connection established (or I've reconnected after a loss of connection)
  guard let connected = snapshot.value as? Bool where connected else {
    return
  }
  // add this device to my connections list
  // this value could contain info about the device or a timestamp instead of just true
  let con = myConnectionsRef.childByAutoId()
  con.setValue("YES")

  // when this device disconnects, remove it
  con.onDisconnectRemoveValue()

  // when I disconnect, update the last time I was seen online
  lastOnlineRef.onDisconnectSetValue(FIRServerValue.timestamp())
})
```

MANAGING USERS ON/OFFLINE

3 DICTIONARIES

MANAGE PRESENCE

```
// since I can connect from multiple devices, we store each connection instance separately
// any time that connectionsRef's value is null (i.e. has no children) I am offline
let myConnectionsRef = FIRDatabase.database().referenceWithPath("users/joe/connections")

// stores the timestamp of my last disconnect (the last time I was seen online)
let lastOnlineRef = FIRDatabase.database().referenceWithPath("users/joe/lastOnline")

let connectedRef = FIRDatabase.database().referenceWithPath(".info/connected")

connectedRef.observeEventType(.Value, withBlock: { snapshot in
  // only handle connection established (or I've reconnected after a loss of connection)
  guard let connected = snapshot.value as? Bool where connected else {
    return
  }
  // add this device to my connections list
  // this value could contain info about the device or a timestamp instead of just true
  let con = myConnectionsRef.childByAutoId()
  con.setValue("YES")

  // when this device disconnects, remove it
  con.onDisconnectRemoveValue()

  // when I disconnect, update the last time I was seen online
  lastOnlineRef.onDisconnectSetValue(FIRServerValue.timestamp())
})
```

MANAGE PRESENCE

```
// since I can connect from multiple devices, we store each connection instance separately
// any time that connectionsRef's value is null (i.e. has no children) I am offline
let myConnectionsRef = FIRDatabase.database().referenceWithPath("users/joe/connections")

// stores the timestamp of my last disconnect (the last time I was seen online)
let lastOnlineRef = FIRDatabase.database().referenceWithPath("users/joe/lastOnline")

let connectedRef = FIRDatabase.database().referenceWithPath(".info/connected")

connectedRef.observeEventType(.Value, withBlock: { snapshot in
  // only handle connection established (or I've reconnected after a loss of connection)
  guard let connected = snapshot.value as? Bool where connected else {
    return
  }
  // add this device to my connections list
  // this value could contain info about the device or a timestamp instead of just true
  let con = myConnectionsRef.childByAutoId()
  con.setValue("YES")

  // when this device disconnects, remove it
  con.onDisconnectRemoveValue()

  // when I disconnect, update the last time I was seen online
  lastOnlineRef.onDisconnectSetValue(FIRServerValue.timestamp())
})
```

MANAGE PRESENCE

```
// since I can connect from multiple devices, we store each connection instance separately
// any time that connectionsRef's value is null (i.e. has no children) I am offline
let myConnectionsRef = FIRDatabase.database().referenceWithPath("users/joe/connections")

// stores the timestamp of my last disconnect (the last time I was seen online)
let lastOnlineRef = FIRDatabase.database().referenceWithPath("users/joe/lastOnline")

let connectedRef = FIRDatabase.database().referenceWithPath(".info/connected")

connectedRef.observeEventType(.Value, withBlock: { snapshot in
  // only handle connection established (or I've reconnected after a loss of connection)
  guard let connected = snapshot.value as? Bool where connected else {
    return
  }
  // add this device to my connections list
  // this value could contain info about the device or a timestamp instead of just true
  let con = myConnectionsRef.childByAutoId()
  con.setValue("YES")

  // when this device disconnects, remove it
  con.onDisconnectRemoveValue()

  // when I disconnect, update the last time I was seen online
  lastOnlineRef.onDisconnectSetValue(FIRServerValue.timestamp())
})
```

MANAGE PRESENCE

```
if let name = user?.displayName {
  let lastMessage = Database.database().reference(withPath: "/users/\(name)/lastMessage/")
  lastMessage.setValue(ServerValue.timestamp())
}
```

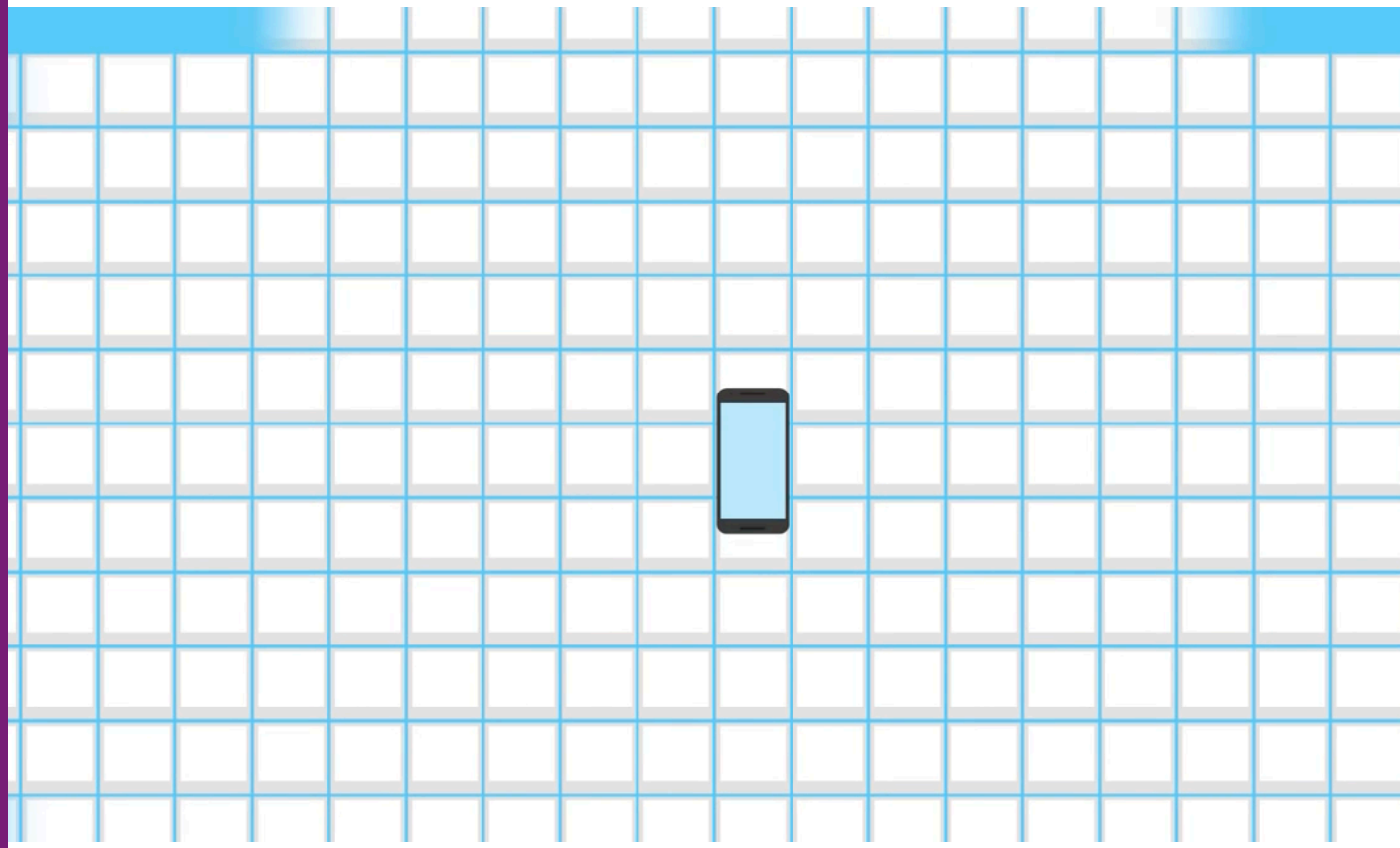
- Simple way to do it

BREAK TIME 

FIREBASE STORAGE

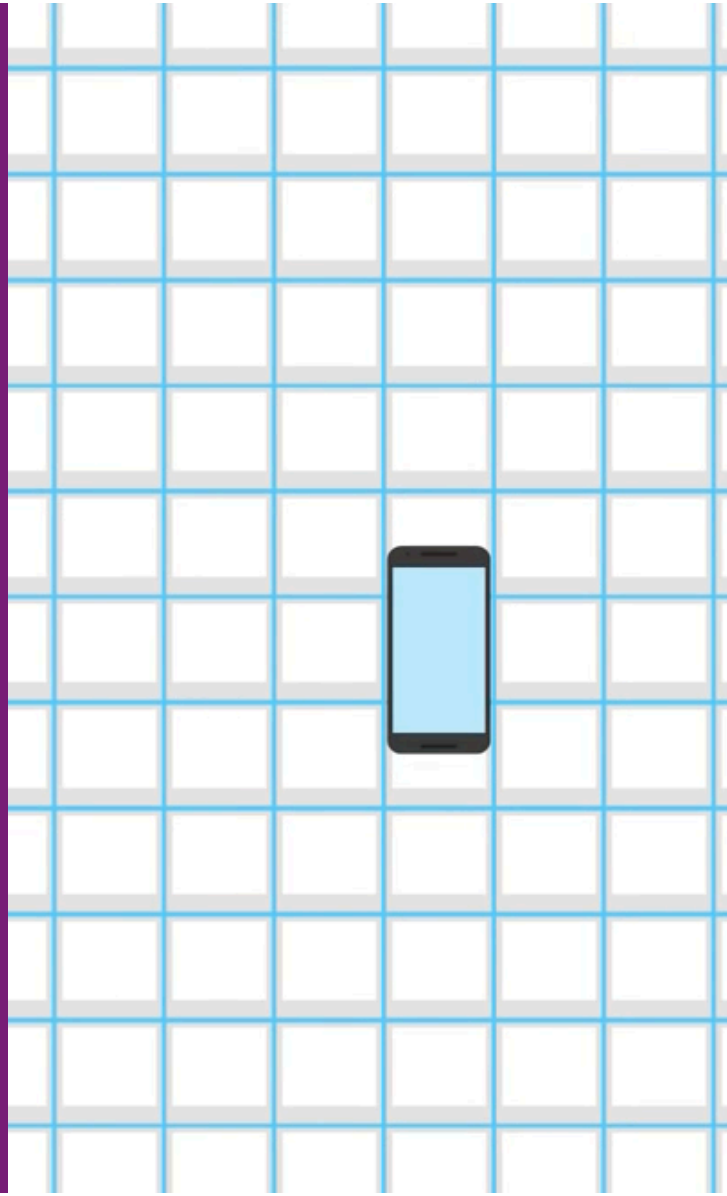
FIREBASE STORAGE

- https://www.youtube.com/watch?list=PLI-K7zZEsYLmOF_07IayrTntevxtbUxDL&v=_tyjqozrEPY



FIREBASE STORAGE

- Cloud Storage for Firebase stores files (blobs)
  - Images
  - Video
  - ...
- Storage is also accessible through Google Cloud Storage



FIREBASE STORAGE

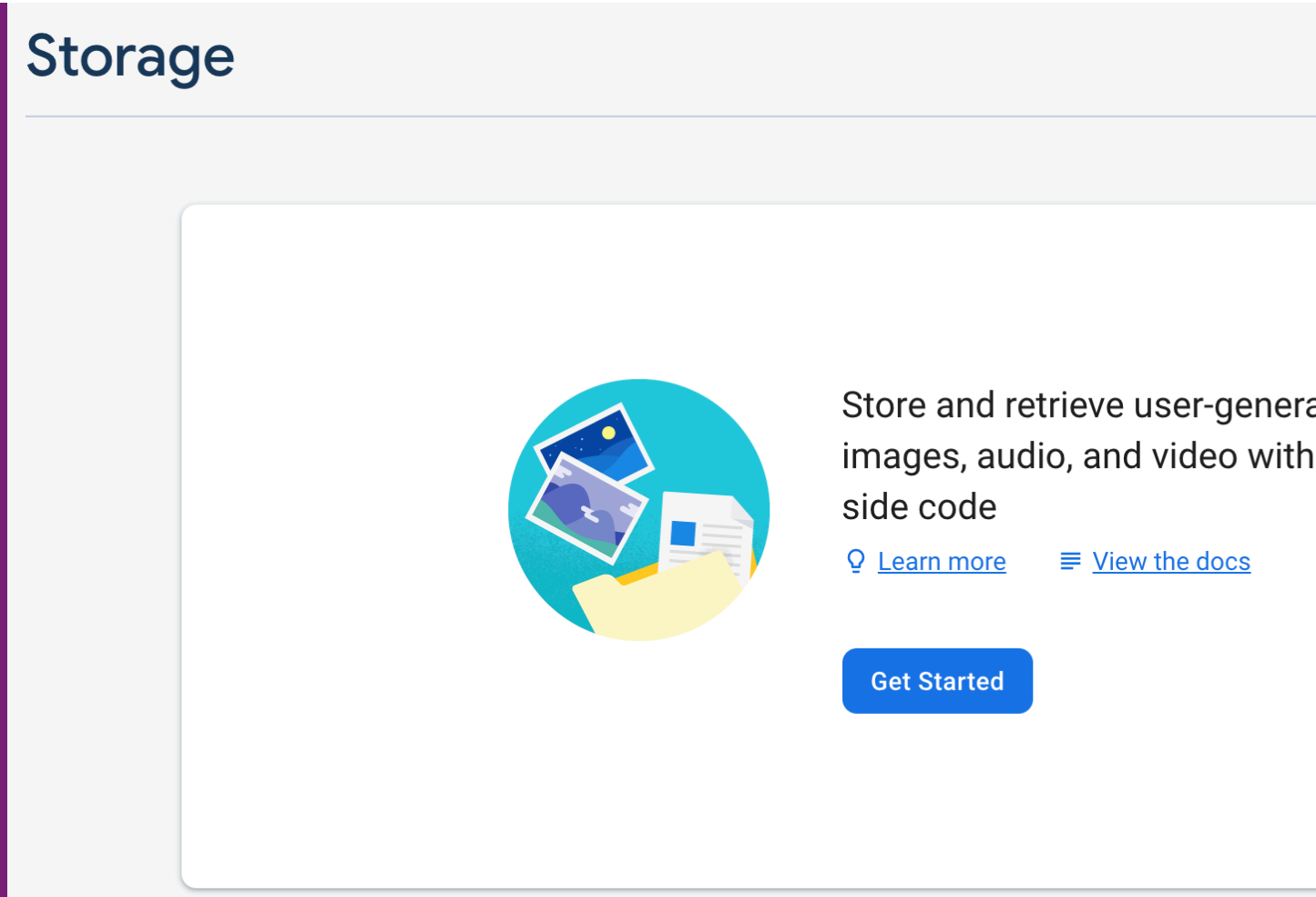
1	Integrate the Firebase SDKs for Cloud Storage.	Quickly include clients via Gradle, CocoaPods, or a script include.
2	Create a Reference	Reference the path to a file, such as "images/mountains.png", to upload, download, or delete it.
3	Upload or Download	Upload or download to native types in memory or on disk.
4	Secure your Files	Use Firebase Security Rules for Cloud Storage to secure your files.

FIREBASE STORAGE

```
target 'chat-cat' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for chat-cat

  # add the Firebase pod for Google Analytics
  pod 'Firebase/Analytics'

  # add pods for any other des    d Fi          ucts
  pod 'Firebase/Database'
  pod 'Firebase/Auth'
  pod 'GoogleSignIn'
  pod 'FirebaseStorage'

  # https://firebase.google.com/docs/ios/setup#available-pods
```

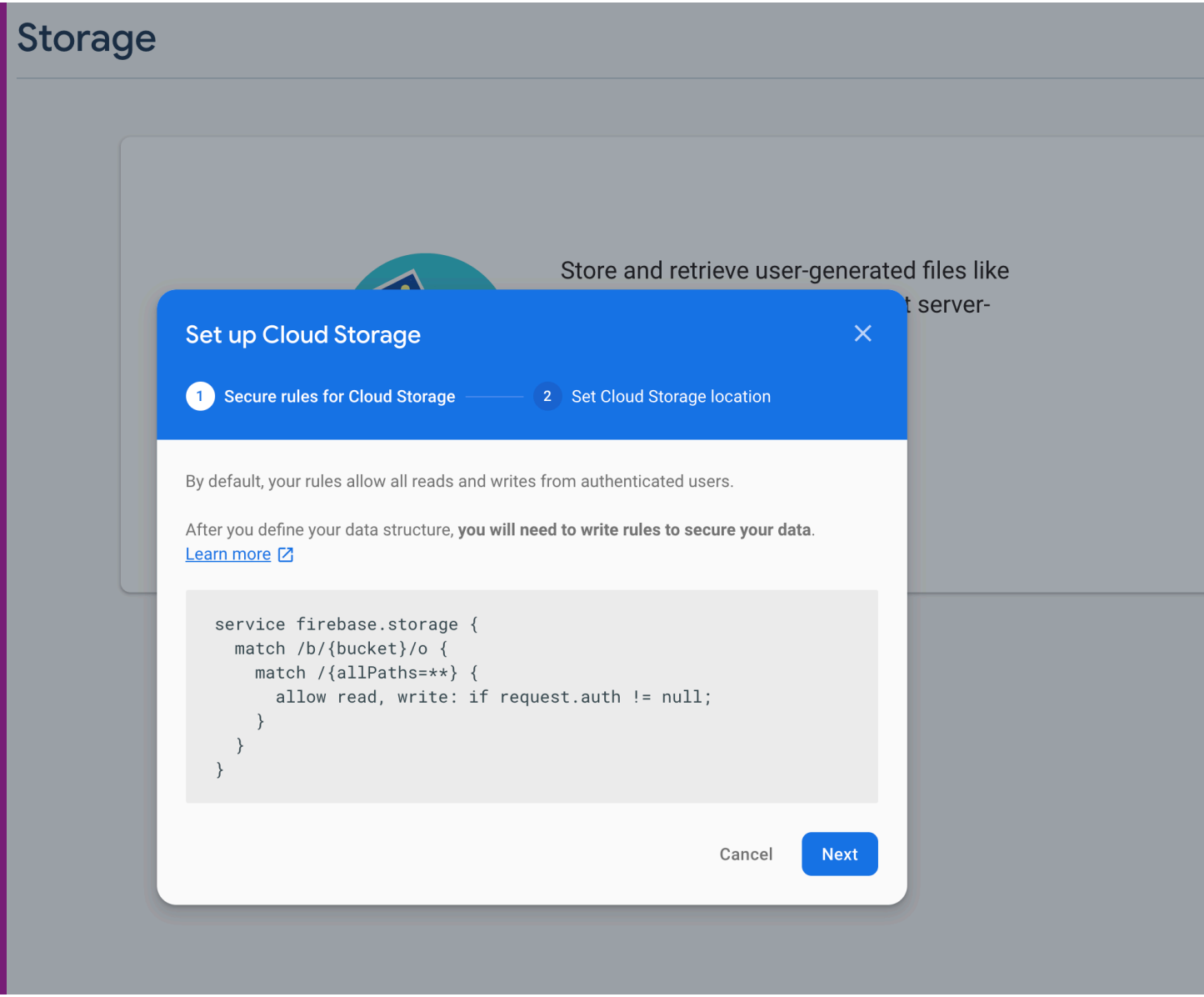
FIREBASE STORAGE

- Online console for storage



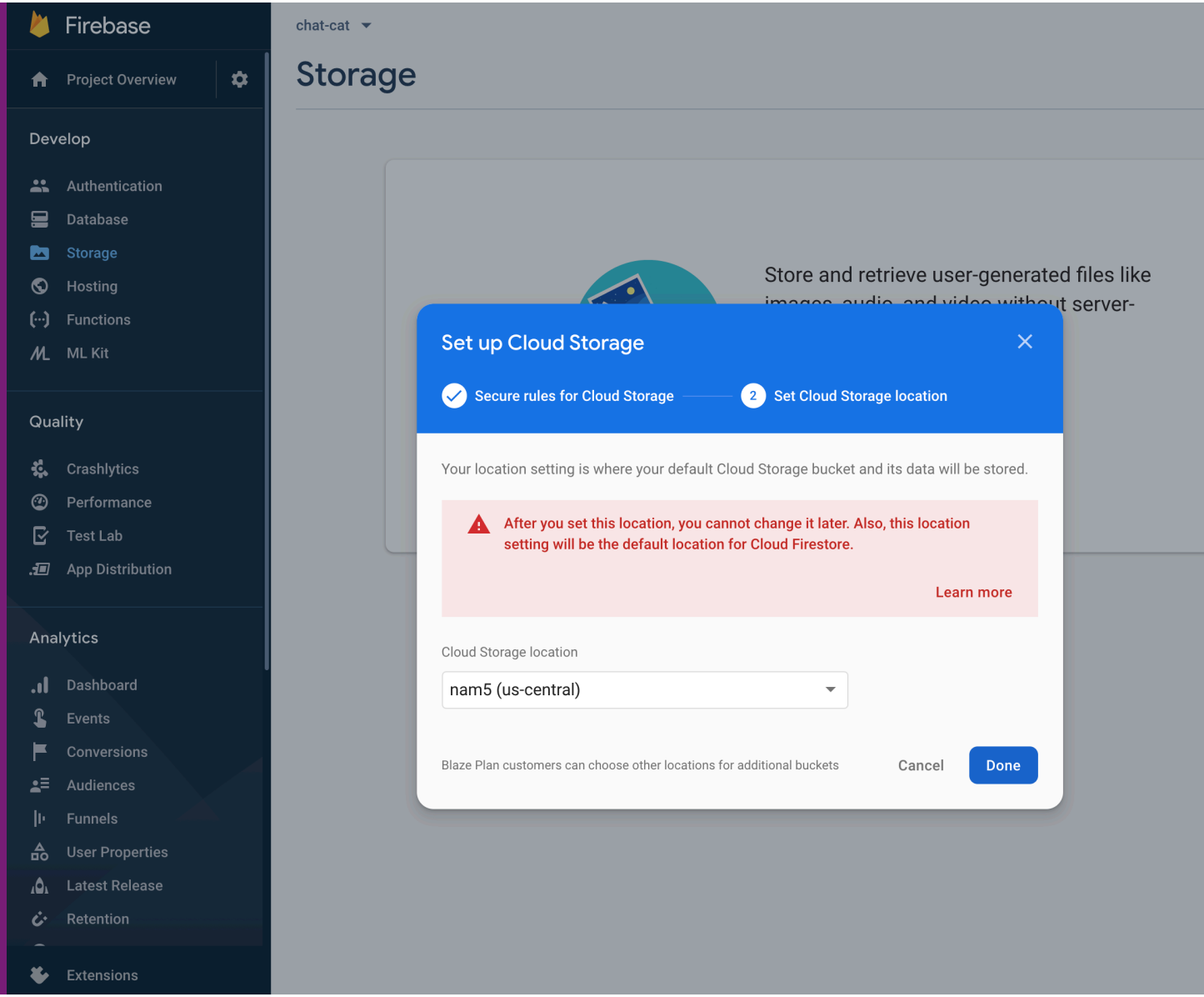
FIREBASE STORAGE

- Set permission rules



FIREBASE STORAGE

- Set location



FIREBASE STORAGE

```
// Only authenticated users can read or write to the bucket
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

- Default - users only

FIREBASE STORAGE

```
// Anyone can read or write to the bucket, even non-users of your app.
// Because it is shared with Google App Engine, this will also make
// files uploaded via GAE public.
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write;
    }
  }
```

- Public

FIREBASE STORAGE

```
// Grants a user access to a node matching their user ID
service firebase.storage {
  match /b/{bucket}/o {
    // Files look like: "user/<UID>/path/to/file.txt"
    match /user/{userId}/{allPaths=**} {
      allow read, write: if request.auth.uid == userId;
    }
  }
```

- User based storage

FIREBASE STORAGE

```
// Access to files through Firebase Storage is completely disallowed.
// Files may still be accessible through Google App Engine or GCS APIs.
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if false;
    }
  }
}
```

- Private
- Static images uploaded by developer

FIREBASE STORAGE

- References to our storage
- Data accesible by path

```
// Points to the root reference
let storageRef =    Storage.storage().reference()

// Points to "images"
let imagesRef = storageRef.child("images")

// Points to "images/space.jpg"
// Note that you can use variables to create child values
let fileName = "space.jpg"
let spaceRef = imagesRef.child(fileName)

// File path is "images/space.jpg"
let path = spaceRef.fullPath;

// File name is "space.jpg"
let name = spaceRef.name;

// Points to "images"
let images = spaceRef.parent()
```

FIREBASE STORAGE

```
// Create a root reference
let storageRef = storage.reference()

// Create a reference to "mountains.jpg"
let mountainsRef = storageRef.child("mountains.jpg")

// Create a reference to 'images/mountains.jpg'
let mountainImagesRef = storageRef.child("images/mountains.jpg")

// While the file names are the same, the references point to different files
mountainsRef.name == mountainImagesRef.name;            // true
mountainsRef.fullPath == mountainImagesRef.fullPath;    // false
```

- Create a reference node

UPLOAD

FIREBASE STORAGE

- Upload from image in memory

```
// Data in memory
let data = Data()

// Create a reference to the file you want to upload
let riversRef = storageRef.child("images/rivers.jpg")

// Upload the file to the path "images/rivers.jpg"
let uploadTask = riversRef.putData(data, metadata: nil) { (metadata, error) in
  guard let metadata = metadata else {
    // Uh-oh, an error occurred!
    return
  }
  // Metadata contains file metadata such as size, content-type.
  let size = metadata.size
  // You can also access to download URL after upload.
  riversRef.downloadURL { (url, error) in
    guard let downloadURL = url else {
      // Uh-oh, an error occurred!
      return
    }
  }
}
```

FIREBASE STORAGE

- Upload from image on disk

```
// File located on disk
let localFile = URL(string: "path/to/image")!

// Create a reference to the file you want to upload
let riversRef = storageRef.child("images/rivers.jpg")

// Upload the file to the path "images/rivers.jpg"
let uploadTask = riversRef.putFile(from: localFile, metadata: nil) { metadata, error in
  guard let metadata = metadata else {
    // Uh-oh, an error occurred!
    return
  }
  // Metadata contains file metadata such as size, content-type.
  let size = metadata.size
  // You can also access to download URL after upload.
  riversRef.downloadURL { (url, error) in
    guard let downloadURL = url else {
      // Uh-oh, an error occurred!
      return
    }
  }
}
```

FIREBASE STORAGE

```
func uploadFile() {
  // Local file you want to upload
  let kitten = UIImage(named:"Kitten")!
  let data = kitten.pngData()!

  let kittenRef = storageRef.child("images/kitten.jpg")

  // Upload the file to the path "images/rivers.jpg"
  let _ = kittenRef.putData(data, metadata: nil) { (metadata, error) in
    guard metadata != nil else {
      // Uh-oh, an error occurred!
      return
    }

  }
}
```

FIREBASE STORAGE

- Default metadata is taken from the image
- You can override it

```
// Create storage reference
let mountainsRef = storageRef.child("images/mountains.jpg")

// Create file metadata including the content type
let metadata = StorageMetadata()
metadata.contentType = "image/jpeg"

// Upload data and metadata
mountainsRef.putData(data, metadata: metadata)

// Upload file and metadata
mountainsRef.putFile(from: localFile, metadata: metadata)
```

FIREBASE STORAGE

```
// Add a progress observer to an upload task
let observer = uploadTask.observe(.progress) { snapshot in
  // A progress event occured
}
```

- Observe progress on a file upload

FIREBASE STORAGE

```
// Start uploading a file
let uploadTask = storageRef.putFile(from: localFile)

// Pause the upload
uploadTask.pause()

// Resume the upload
uploadTask.resume()

// Cancel the upload
uploadTask.cancel()
```

- Manage uploads

FIREBASE STORAGE

```swift
// Local file you want to upload
let localFile = URL(string: "path/to/image")!

// Create the file metadata
let metadata = FIRStorageMetadata()
metadata.contentType = "image/jpeg"

// Upload file and metadata to the object 'images/mountains.jpg'
let uploadTask = storageRef.putFile(localFile, metadata: metadata)

// Listen for state changes, errors, and completion of the upload.
uploadTask.observe(.resume) { snapshot in
  // Upload resumed, also fires when the upload starts
}

uploadTask.observe(.pause) { snapshot in
  // Upload paused
}

uploadTask.observe(.progress) { snapshot in
  // Upload reported progress
  let percentComplete = 100.0 * Double(snapshot.progress!.completedUnitCount)
    / Double(snapshot.progress!.totalUnitCount)
}
```

UPLOAD AND OBSERVE

FIREBASE STORAGE

- Full example, with all the trimmings

```
// Create the file metadata
let metadata = StorageMetadata()
metadata.contentType = "image/jpeg"

// Upload file and metadata to the object 'images/mountains.jpg'
let uploadTask = storageRef.putFile(from: localFile, metadata: metadata)

// Listen for state changes, errors, and completion of the upload.
uploadTask.observe(.resume) { snapshot in
  // Upload resumed, also fires when the upload starts
}

uploadTask.observe(.pause) { snapshot in
  // Upload paused
}

uploadTask.observe(.progress) { snapshot in
  // Upload reported progress
  let percentComplete = 100.0 * Double(snapshot.progress!.completedUnitCount)
    / Double(snapshot.progress!.totalUnitCount)
}

uploadTask.observe(.success) { snapshot in
  // Upload completed successfully
}

uploadTask.observe(.failure) { snapshot in
  if let error = snapshot.error as? NSError {
    switch (StorageErrorCode(rawValue: error.code)!) {
    case .objectNotFound:
      // File doesn't exist
      break
    case .unauthorized:
      // User doesn't have permission to access file
      break
    case .cancelled:
      // User canceled the upload
      break

    /* ... */

    case .unknown:
      // Unknown error occurred, inspect the server response
      break
    default:
      // A separate error occurred. This is a good place to retry the upload.
      break
    }
  }
}
```

DOWNLOAD

FIREBASE STORAGE

- Once you have a reference, you can download files from Cloud Storage in three ways:
  - Download to Data in memory
  - Download to an URL representing a file on device
  - Generate an URL representing the file online

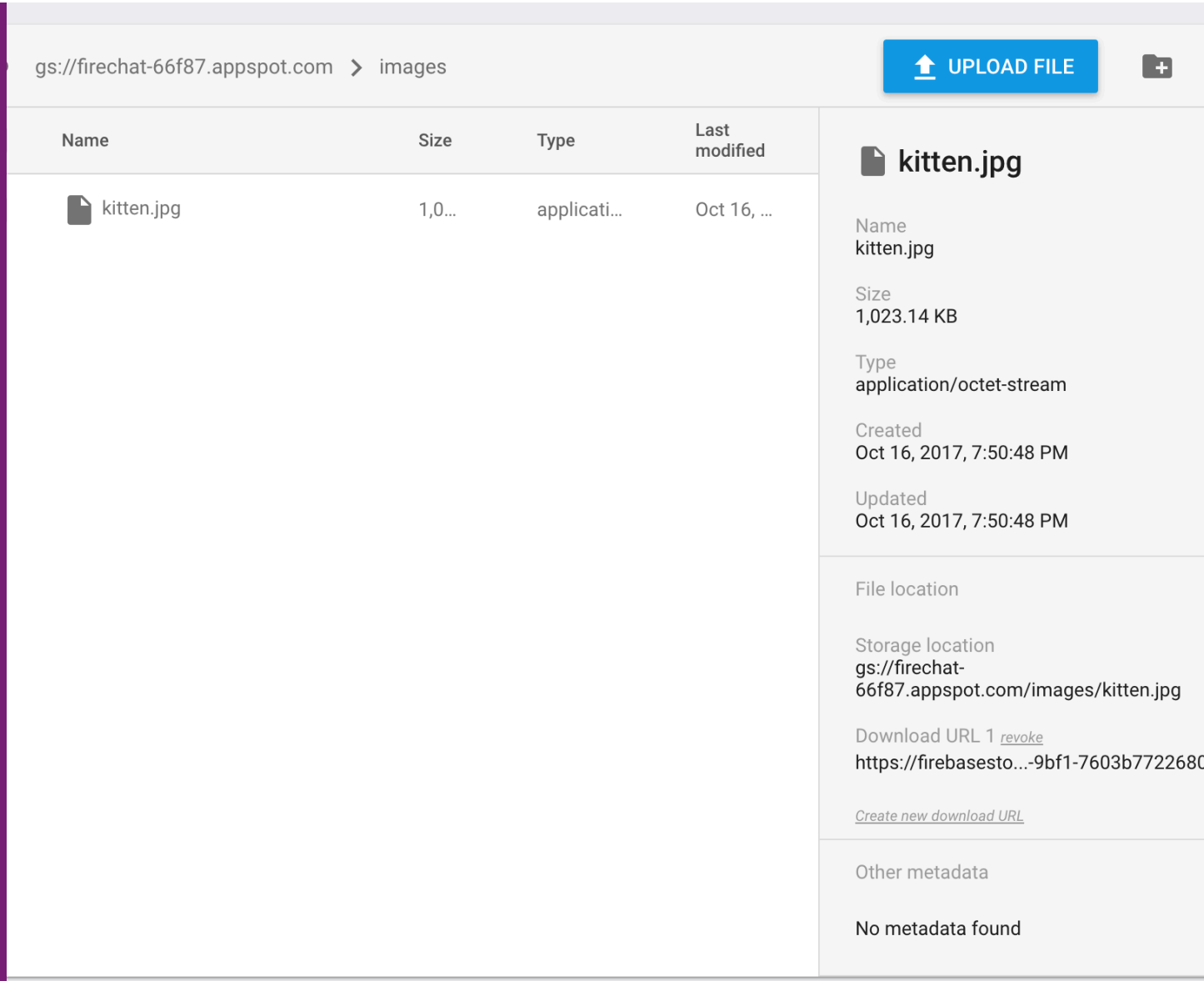
FIREBASE STORAGE

```
// Create a reference with an initial file path and name
let pathReference = storage.reference(withPath: "images/stars.jpg")

// Create a reference from a Google Cloud Storage URI
let gsReference = storage.reference(forURL: "gs://<your-firebase-storage-bucket>/images/sta

// Create a reference from an HTTPS URL
// Note that in the URL, characters are URL escaped!
let httpsReference = storage.reference(forURL: "https://firebasestorage.googleapis.com/b/bu
```

- Files are accessible through different APIs

FIREBASE STORAGE

- Files are accessible through different APIs and locations



FIREBASE STORAGE

- Files are accessible through different APIs
- Download to memory

```
// Create a reference to the file you want to download
let islandRef = storageRef.child("images/island.jpg")

// Download in memory with a maximum allowed size of 1MB (1 * 1024 * 1024 bytes)
islandRef.getData(maxSize: 1 * 1024 * 1024) { data, error in
  if let error = error {
    // Uh-oh, an error occurred!
  } else {
    // Data for "images/island.jpg" is returned
    let image = UIImage(data: data!)
  }
}
```

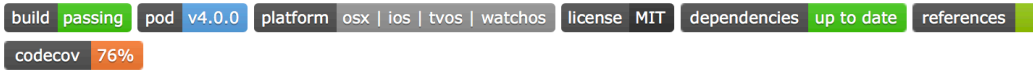
FIREBASE STORAGE

- Download to file on disk

```
// Create a reference to the file you want to download
let islandRef = storageRef.child("images/island.jpg")

// Create local filesystem URL
let localURL = URL(string: "path/to/image")!

// Download to the local filesystem
let downloadTask = islandRef.write(toFile: localURL) { url, error in
  if let error = error {
    // Uh-oh, an error occurred!
  } else {
    // Local file URL for "images/island.jpg" is returned
  }
}
```

FIREBASE STORAGE

- Why work so hard?
- FirebaseUI (part of Storage) will download and cache
  - Based on SDWedImage



This library provides an async image downloader with cache support. For convenience, we added c
elements like `UIImageView`, `UIButton`, `MKAnnotationView`.

Features

- [x] Categories for `UIImageView`, `UIButton`, `MKAnnotationView` adding web image and cache m
- [x] An asynchronous image downloader
- [x] An asynchronous memory + disk image caching with automatic cache expiration handling
- [x] A background image decompression
- [x] A guarantee that the same URL won't be downloaded several times
- [x] A guarantee that bogus URLs won't be retried again and again
- [x] A guarantee that main thread will never be blocked
- [x] Performances!
- [x] Use GCD and ARC

Supported Image Formats

- Image formats supported by UIImage (JPEG, PNG, ...), including GIF
- WebP format, including animated WebP (use the `WebP` subspec)

Requirements

FIREBASE STORAGE

```
  # add the Firebase pod for Google Analytics
  pod 'Firebase/Analytics'

  # add pods for any other desired Firebase products
  pod 'Firebase/Database'
  pod 'Firebase/Auth'
  pod 'GoogleSignIn'
  pod 'FirebaseStorage'

  pod 'FirebaseUI/Storage'
  pod 'SDWebImage'

end
```

FIREBASE STORAGE

```
import UIKit
import Firebase
import FirebaseStorage
import FirebaseUI
```

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

  // Configure the cell...
  let item = messages[indexPath.row]
  cell.textLabel?.text = item.text
  cell.detailTextLabel?.text = item.name

  let reference = storageRef.child("images/kitten.jpg")

  // Placeholder image
  let placeholderImage = UIImage(named: "Placeholder")

  // Load the image using SDWebImage
  cell.imageView?.sd_setImage(with: reference, placeholderImage: placeholderImage)

  return cell
```

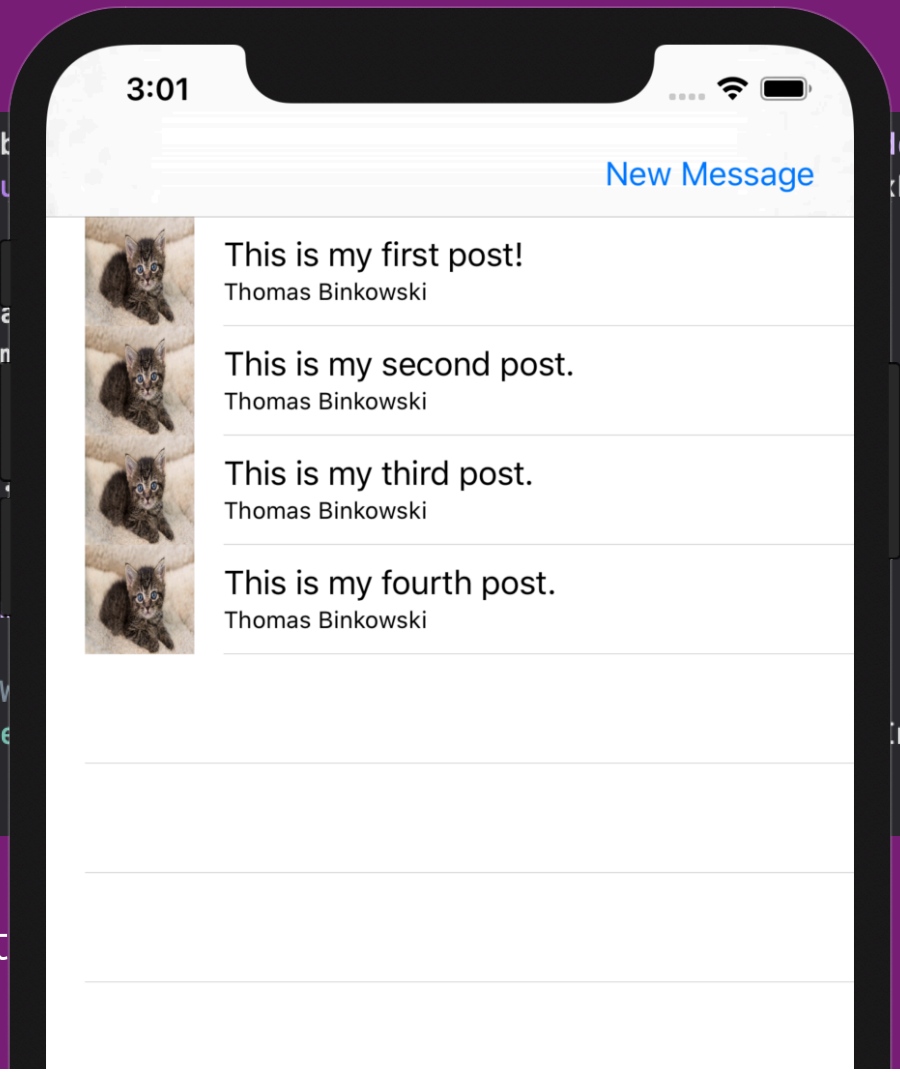
- Download directly to UIImage; swap placeholder image out

FIREBASE STORAGE

```
import UIKit
import Firebase
import FirebaseStorage
import FirebaseUI
```

```
override func tableView(_ tab
  let cell = tableView.dequeu

  // Configure the cell...
  let item = messages[indexPa
  cell.textLabel?.text = item
  cell.detailTextLabel?.text

  let reference = storageRef.

  // Placeholder image
  let placeholderImage = UII

  // Load the image using SDW
  cell.imageView?.sd_setImage

  return cell
```

```
lexPath) -> UITableViewCell {
tPath)

Image)
```



- Download directly t out

FIREBASE ANALYTICS

FIREBASE ANALYTICS

- https://youtu.be/iT6EaIwtonY?list=PLI-K7zZEsYLmOF_07IayrTntevxtbUxDL

FIREBASE ANALYTICS

- Firebase Analytics is a free app measurement solution that provides insight on app usage and user engagement
- Installed in Core

FIREBASE ANALYTICS

- Integrate with other services

BigQuery	Link your Firebase Analytics app to BigQuery where you can perform custom analysis on your entire Analytics dataset and import other data sources.
Firebase Crash Reporting	Firebase Analytics logs events for each crash so you can get a sense of the rate of crashes for different versions or regions, allowing you to gain insight into which users are impacted. You can also create audiences for users who have experienced multiple crashes and respond with Firebase Notifications directed at that audience.
Firebase Notifications	Firebase Analytics automatically logs events that correspond to your Firebase Notifications and supports reporting on the impact of each campaign.
Firebase Remote Config	Use Firebase Analytics audience definitions to change the behavior and appearance of your app for different audiences without distributing multiple versions of your app.
Google Tag Manager	Integrating Google Tag Manager alongside Firebase Analytics enables you to manage your Firebase Analytics implementation remotely from a web interface after your app has been distributed.

FIREBASE ANALYTICS

- Key data is pulled from authentication
- Consider what you what to know

1	Connect your app to Firebase	Getting started with Analytics is easy. Just add the Firebase SDK to your new or existing app, and data collection begins automatically. You can view analytics data in the Firebase console within hours.
2	Log custom data	You can use Analytics to log custom events that make sense for your app, like E-Commerce purchases or achievements.
3	Create audiences	You can define the audiences that matter to you in the Firebase console.
4	Target audiences	Use your custom audiences to target messages, promotions, or new app features using other Firebase features, such as Notifications, and Remote Config.

FIREBASE ANALYTICS

- Automatically tracked events

Event name	Triggered...
first_open	the first time a user launches an app after installing or re-installing it. This event is not triggered when a user downloads the app onto a device, but instead when he or she first uses it. To see raw download numbers, look in Google Play Developer Console or in iTunesConnect.
in_app_purchase	when a user completes an in-app purchase that is processed by the App Store on iTunes or Google Play. The product ID, product name, currency, and quantity are passed as parameters. This event is triggered only by versions of your app that include the Firebase SDK. Also, **subscription revenue, paid app-purchase revenue, and refunds are not automatically tracked**, and so your reported revenue may differ from the values you see in the Google Play Developer Console. Events which are flagged as being invalid or sandbox (test) are ignored.
user_engagement	periodically, while the app is in the foreground.
session_start	when a user engages the app for more than the minimum session duration after a period of inactivity that exceeds the session timeout duration.

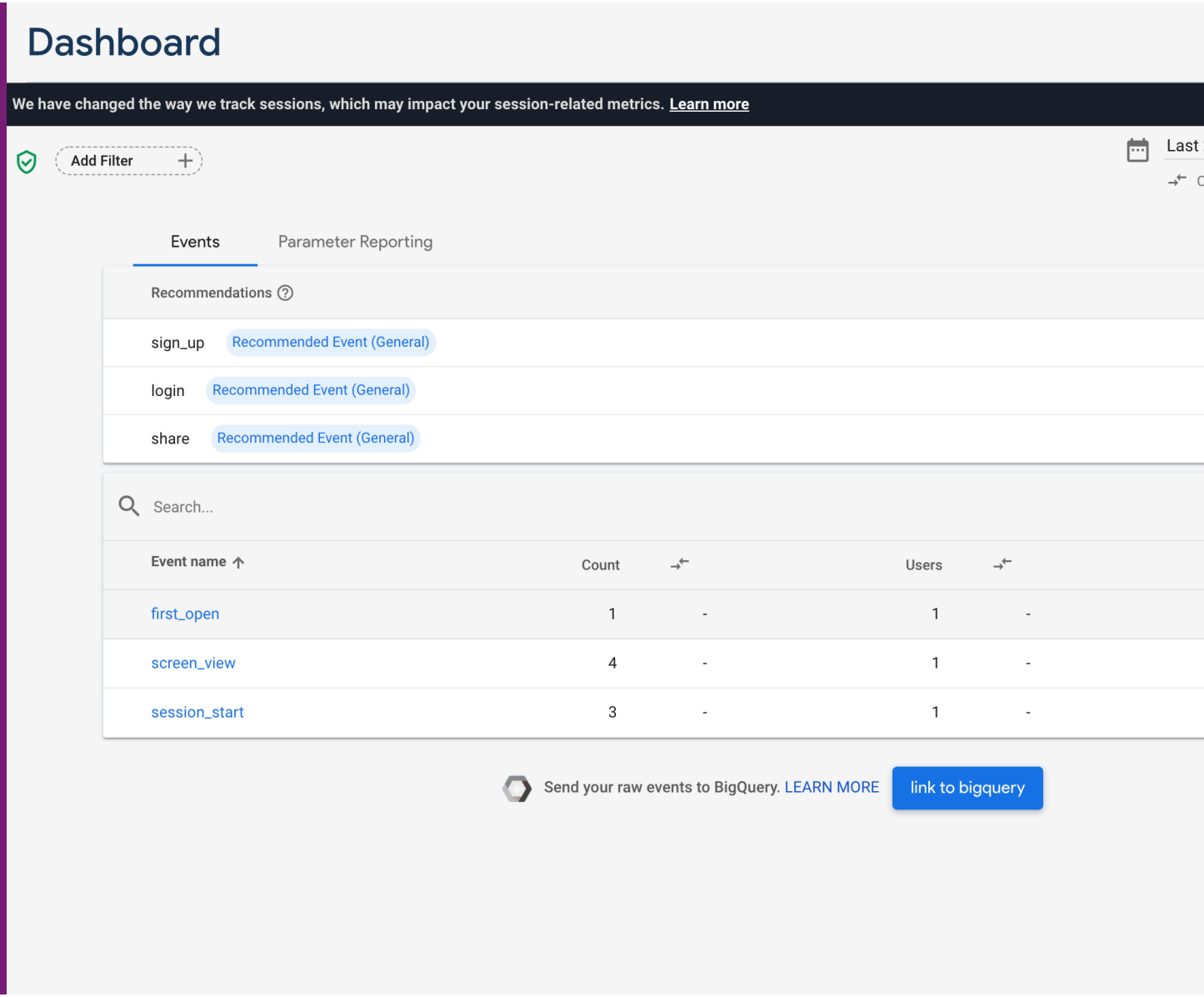
FIREBASE ANALYTICS

- New Pod

```
target 'chat-cat' do
  # Comment the next line if you don't want to use dynamic fr
  use_frameworks!

  # Pods for chat-cat

  # add the Firebase pod for Google Analytics
  pod 'Firebase/Analytics'

  # add pods for any other desired Firebase products
  pod 'Firebase/Database'
  pod 'Firebase/Auth'
  pod 'GoogleSignIn'
  pod 'FirebaseStorage'

  pod 'FirebaseUI/Storage'
  pod 'SDWebImage'

end
```

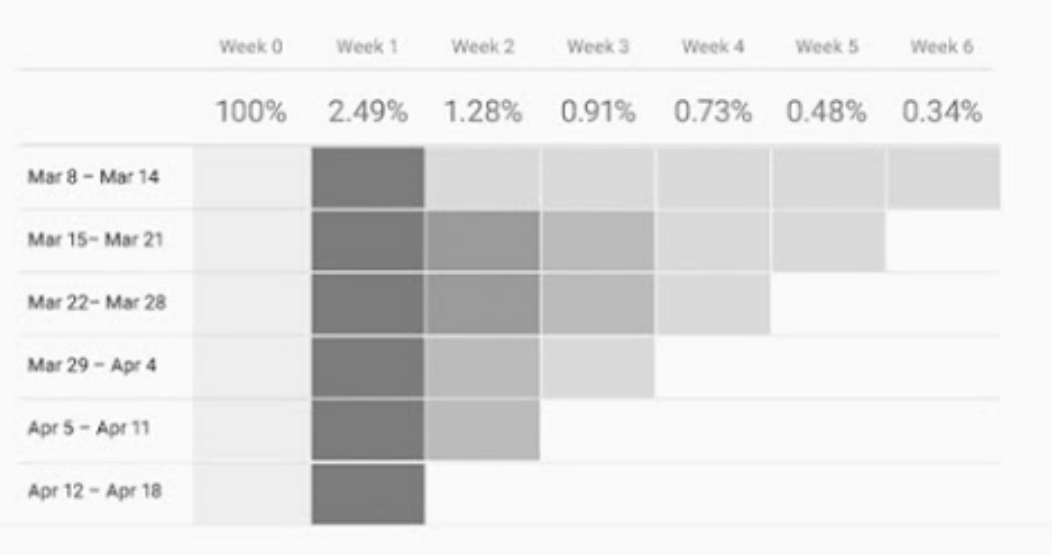
FIREBASE ANALYTICS

- Predefined events



FIREBASE ANALYTICS

- Active users
- Average revenue, IAP
- first_open, first_open attribution
- Retention cohort
- User engagement (daily, per user, sessions, avg. session duration)
- Device and App info
- Location
- Demographics
- Screen views
- ...much much more



FIREBASE ANALYTICS

```
Analytics.logEvent(AnalyticsEventSelectContent, parameters: [
  AnalyticsParameterItemID: "id-\(title!)",
  AnalyticsParameterItemName: title!,
  AnalyticsParameterContentType: "cont"
  ])
```

- Custom events

FIREBASE ANALYTICS

```
analytics.logEvent('select_content', {
  content_type: 'image',
  content_id: 'P12453',
  items: [{ name: 'Kittens' }]
});
```

- Custom events

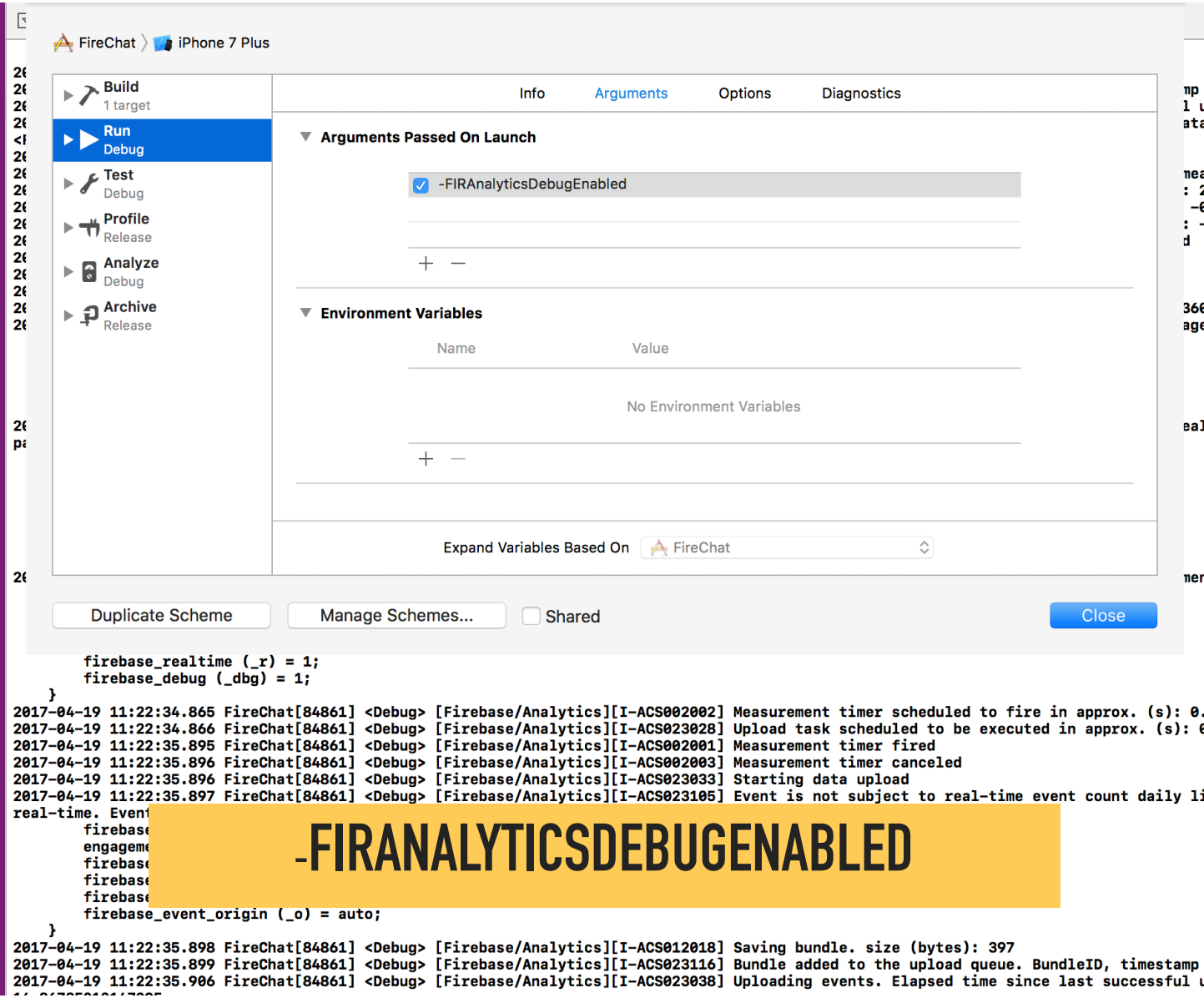
FIREBASE ANALYTICS

```
                                    guard let textField = alert.textFields?.first,
                                      let text = textField.text else {
                                        return
                                    }

                                    let data = ["name":self.user?.displayName,"text": text]
                                    self.referenc                    .setValue(data)

}


let cancelAction = UIAlertAction(title: "Can
                                  style: .d      ault)
_ = UIAlertAction(title: "Cancel", style: .default) { (_) in
  Analytics.logEvent("message-cancelled", parameters: nil)
}
alert.addTextField()
```

TRACK CANCELLED MESSAGES

FIREBASE ANALYTICS

- View login in the console
  - Products > Scheme
  - Passed on launch



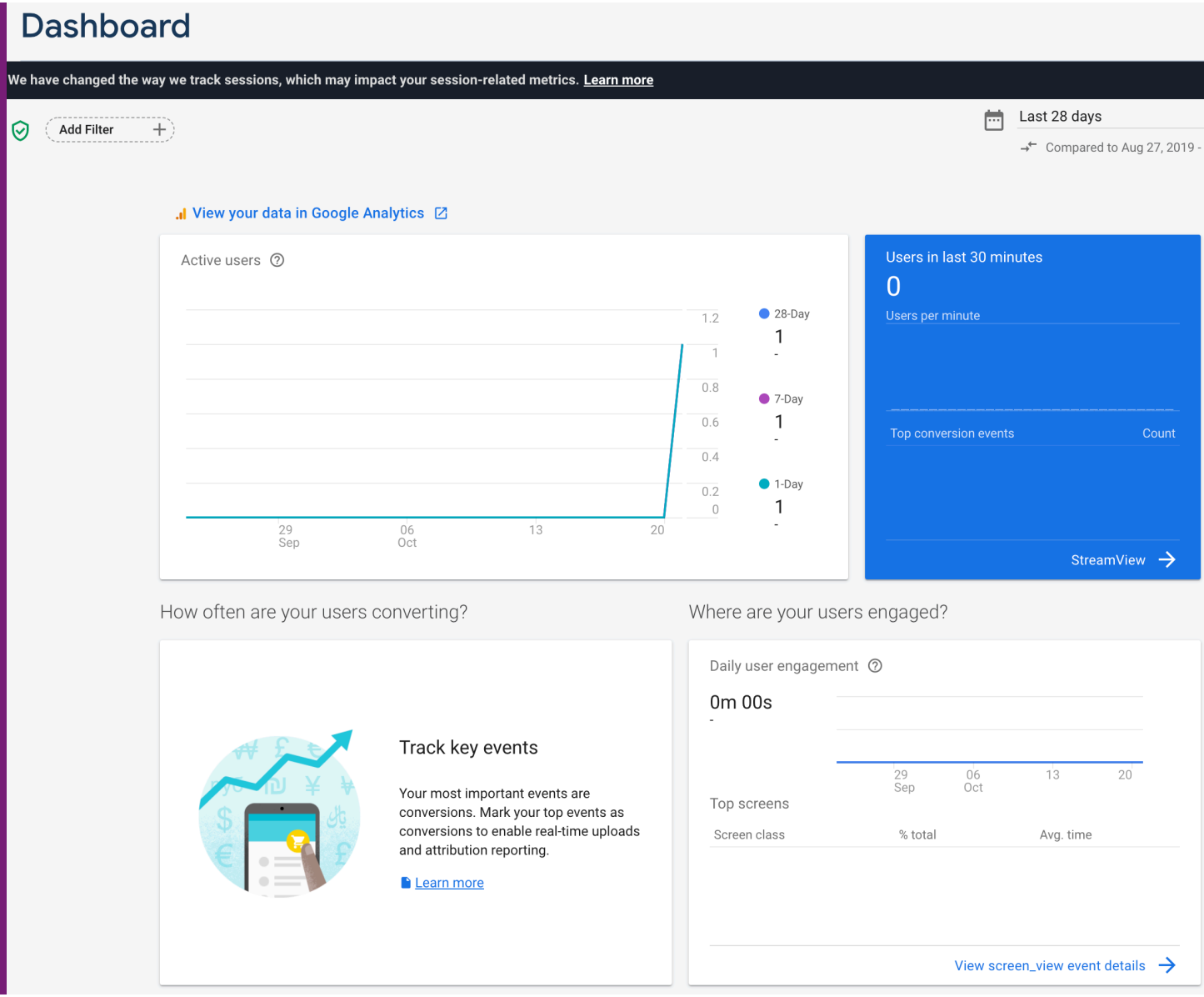
FIREBASE ANALYTICS

```
2017-04-19 11:22:20.771 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS012018] Saving bundle. size (bytes): 392
2017-04-19 11:22:20.771 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023116] Bundle added to the upload queue. BundleID, timestamp (ms): 4, 1492618939686
2017-04-19 11:22:20.779 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023038] Uploading events. Elapsed time since last successful upload (s): 1.4290108680725
2017-04-19 11:22:20.780 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023039] Measurement data sent to network. Timestamp (ms), data: 1492618940779,
<FIRAPBMeasurementBatch: 0x60000000a2e0>
2017-04-19 11:22:20.782 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS900000] Uploading data. Host: https://app-measurement.com/a
2017-04-19 11:22:20.908 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS901006] Received SSL challenge for host. Host: https://app-measurement.com/a
2017-04-19 11:22:21.039 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023044] Successful upload. Got network response. Code, size: 204, -1
2017-04-19 11:22:21.041 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS002002] Measurement timer scheduled to fire in approx. (s): -0.3548879027366638
2017-04-19 11:22:21.041 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023028] Upload task scheduled to be executed in approx. (s): -0.3548879027366638
2017-04-19 11:22:21.048 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023024] No data to upload. Upload task will not be scheduled
2017-04-19 11:22:21.048 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS002003] Measurement timer canceled
2017-04-19 11:22:34.853 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS033003] Scheduling user engagement timer
2017-04-19 11:22:34.853 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS002003] Engagement timer canceled
2017-04-19 11:22:34.854 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS002002] Engagement timer scheduled to fire in approx. (s): 3600
2017-04-19 11:22:34.854 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023051] Logging event: origin, name, params: auto, user_engagement (_e), {
        firebase_screen_id (_si) = 7128337720279460482;
        engagement_time_msec (_et) = 15878;
        firebase_screen_class (_sc) = FireChat.MessagesTableViewController;
        firebase_event_origin (_o) = auto;
    }
2017-04-19 11:22:34.854 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023073] Debug mode is enabled. Marking event as debug and real-time. Event name,
parameters: user_engagement (_e), {
        firebase_screen_id (_si) = 7128337720279460482;
        engagement_time_msec (_et) = 15878;
        firebase_screen_class (_sc) = FireChat.MessagesTableViewController;
        firebase_event_origin (_o) = auto;
        firebase_realtime (_r) = 1;
        firebase_debug (_dbg) = 1;
    }
2017-04-19 11:22:34.864 FireChat[84861] <Debug> [Firebase/Analytics][I-ACS023072] Event logged. Event name, event params: user_en
        firebase_screen_id (_si) = 7128337720279460482;
        engagement_time_msec (_et) = 15878;
        firebase_screen_class (_sc) = FireChat.MessagesTableViewController;
        firebase_event_origin (_o) = auto;
        firebase_realtime (_r) = 1;
```

IT'S A LITTLE CREEPY

FIREBASE ANALYTICS

- Data is accessed in the online console
- Time hour delay



FIREBASE ANALYTICS

```
firebase.analytics().setUserProperties({favorite_food: 'apples'});
```

- Add custom properties to the `user` for analytics analysis later

ASSIGNMENT 4

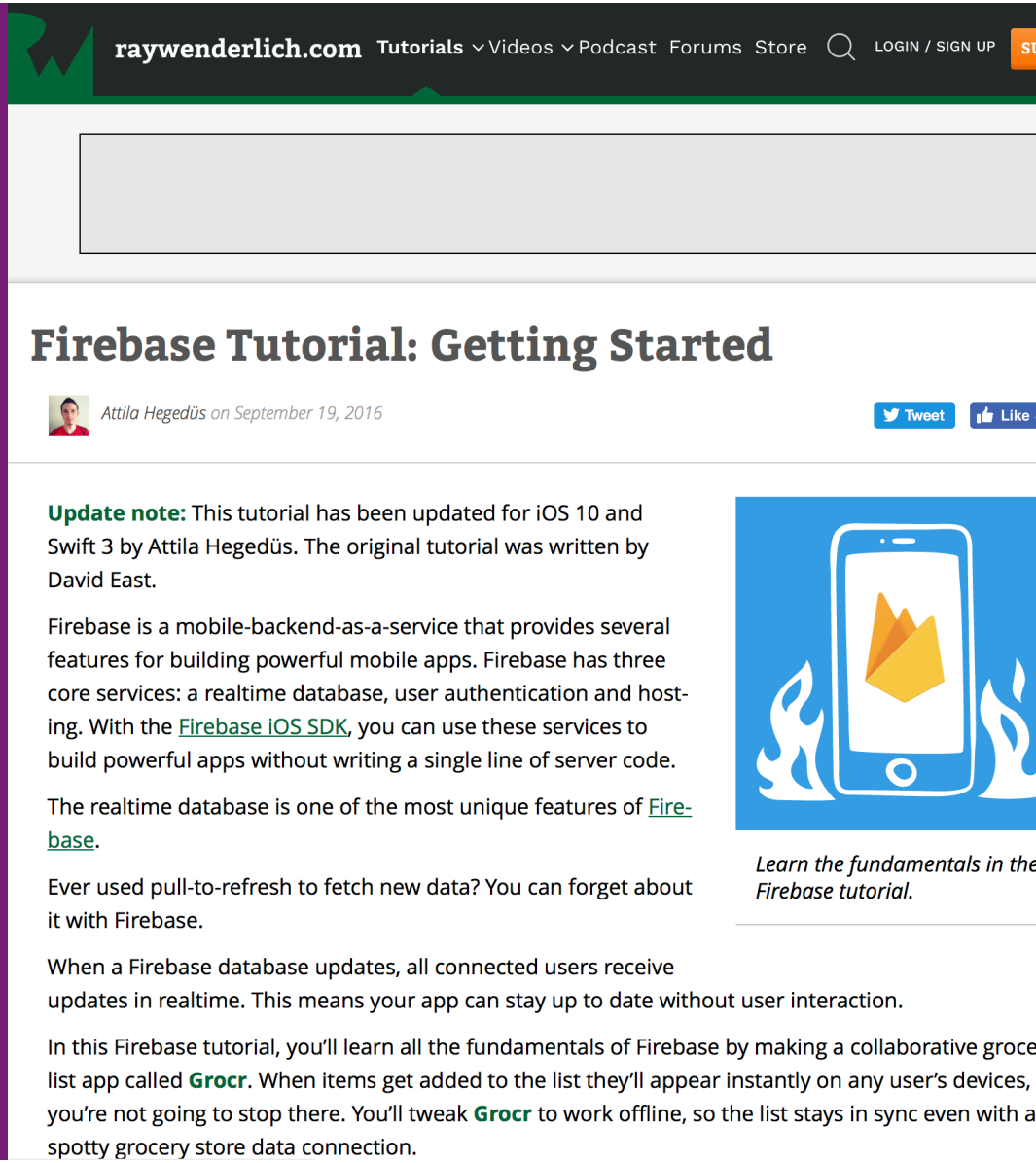
ASSIGNMENT 3

- Part 1
  - Firebase tutorials
- Part 2
  - Develop a group messaging application using Firebase

PART 1

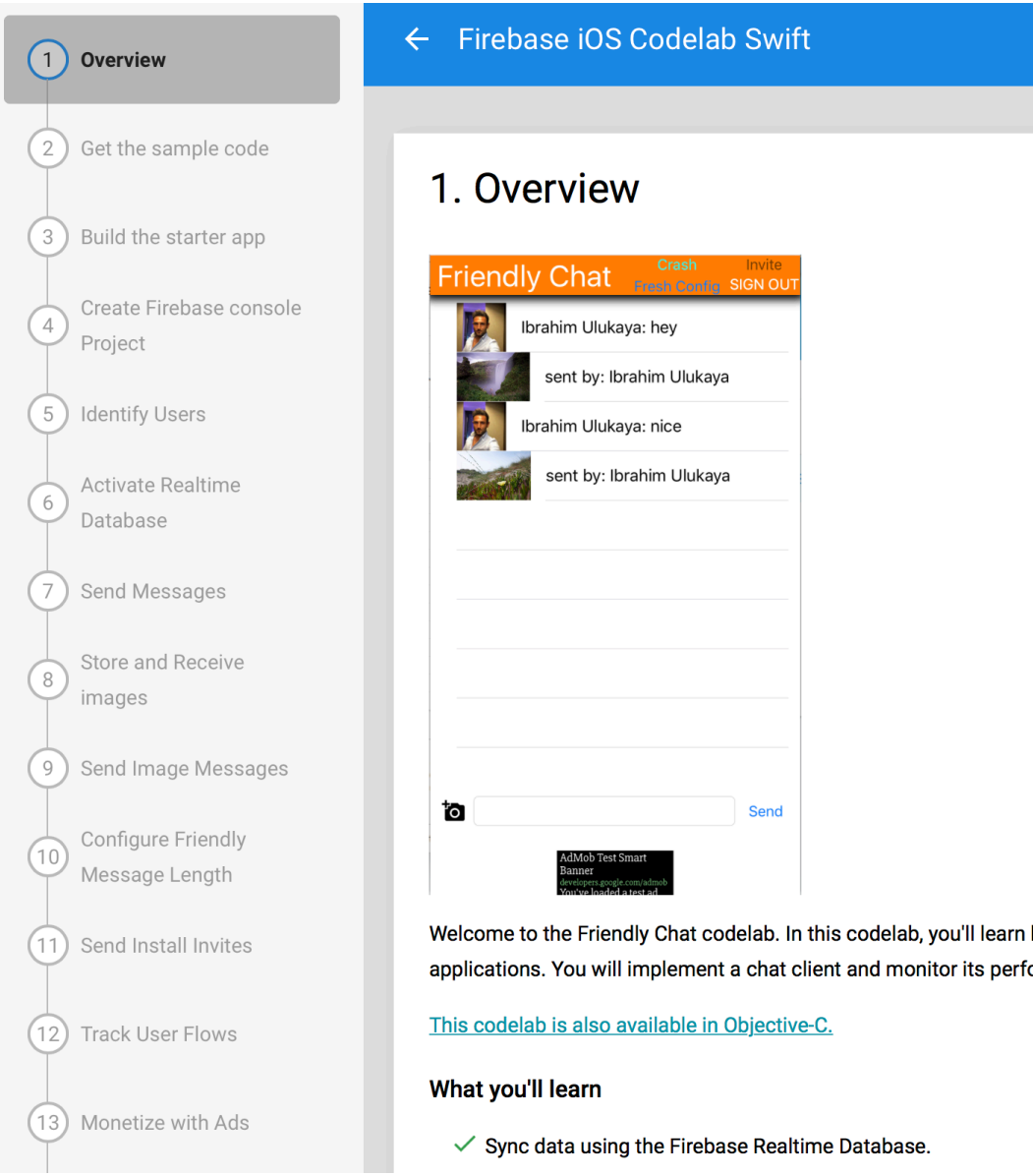
ASSIGNMENT 3 - PART 1

- Ray Weinerlich Tutorial to build a grocery app
- https://www.raywenderlich.com/139322/firebase-tutorial-getting-started-2



ASSIGNMENT 3 - PART 1

- Firebase codelab tutorial to build a (very sparse) chat application
- https://codelabs.developers.google.com/codelabs/firebase-ios-swift/#0



PART 2

NEXT WEEK TOPICS

- Cloud Firestore
- Cloud functions
- Firebase Cloud Messaging
- Remote config
- App Indexing
- Dynamic links
- Admob
- Invites
- Ad words
- Notification

IF YOU WANT TO GET A PAID DEVELOPER ACCOUNT, THIS IS THE WEEK. THIS WILL ALLOW YOU TO SEND PUSH NOTIFICATIONS.

THIS IS OPTIONAL.

ASSIGNMENT 3 - PART 2

- Build a group messaging application using Firebase (yes the world needs another one)

ASSIGNMENT 3 - PART 2

- Authenticated users will be able to send messages to other authenticated users
- The messages can be text or images
- Users will invite other users to a chat using Firebase Invites (next week) and utilizing Dynamic links
- Notifications when someone joins a group

ASSIGNMENT 3 - PART 2

- The application interface can be "sparse" as long as it supports the functionality

THE UNIVERSITY OF CHICAGO

MPCS 51033 • AUTUMN 2019 • SESSION 4

BACKENDS FOR MOBILE APPLICATIONS