



MPCS 51033 • AUTUMN 2017 • SESSION 4

BACKENDS FOR MOBILE APPLICATIONS

CLASS NEWS

CLASS NEWS

- Assignment 1&2 graded. Look for grade.txt file in repositories.
- Office hours Thursday 10AM in Young 308

MPCS51033-20... Andrew

All Threads

Channels

assignment1

general

random

Direct Messages

slackbot

Andrew (you)

acham1

Xuefeng

Yi Wang

+ Invite People

Apps

UC

EC

M

⌘1

⌘2

⌘3

⌘4

⌘5

+

#general

Andrew 10:37 AM added an integration to this channel: [twitter](#)

Andrew 9:09 PM Hi. I'm not going to have any formal office hours tomorrow but let me know if you have any questions. I plan on having regular office hours Thursday around 10 starting next week.

Thursday, September 28th

Andrew 10:58 AM <https://cloudplatform.googleblog.com/2016/10/introducing-Google-Cloud-Shells-new-code-editor.html?m=1>

Google Cloud Platform Blog
[Introducing Google Cloud Shell's new code editor](#)
Posted by Sachin Kotwani, Product Manager We've heard from a lot of Google Cloud Platform (GCP) users that they like to edit code and c... (9kB) ▾

Here's an example of how you can use the Cloud Shell code editor to create a sample app, push your changes to Google Cloud Source Repository, deploy the app to Google App Engine Standard, and use Stackdriver Debugger.

Andrew 11:15 AM

Thread

Yi Wang and you

Yi Wang Today at 10:56 AM Direct message

I received the coupon email in my uchicago email box, but when I clicked the url to redeem, it redirected to a page that is logged in with my gmail. I didn't notice that and finished the whole process.

2 replies

Andrew 6 hours ago If it worked with your gmail, then we can go ahead and try it. I'll take a look in class before we actually start using it to make sure it looks ok. I will send an email to my contact at google just to double check.

Yi Wang 5 hours ago OK. Thank you!

Reply... ☺

Message #general @ ☺

CLASS NEWS

- Assignment 1 & 2 graded
- This assignment will be 2 week, but plan accordingly
- Halloween makeup class options?

MPCS51033-20... Andrew

All Threads

Channels

assignment1

general

random

Direct Messages

slackbot

Andrew (you)

acham1

Xuefeng

Yi Wang

+ Invite People

Apps

UC

EC

M

⌘1

⌘2

⌘3

⌘4

⌘5

+

#general

Andrew 10:37 AM added an integration to this channel: [twitter](#)

Andrew 9:09 PM Hi. I'm not going to have any formal office hours tomorrow but let me know if you have any questions. I plan on having regular office hours Thursday around 10 starting next week.

Thursday, September 28th

Andrew 10:58 AM <https://cloudplatform.googleblog.com/2016/10/introducing-Google-Cloud-Shells-new-code-editor.html?m=1>

Google Cloud Platform Blog
[Introducing Google Cloud Shell's new code editor](#)
Posted by Sachin Kotwani, Product Manager We've heard from a lot of Google Cloud Platform (GCP) users that they like to edit code and c... (9kB) ▾

Thread

Yi Wang and you

Yi Wang Today at 10:56 AM Direct message

I received the coupon email in my uchicago email box, but when I clicked the url to redeem, it redirected to a page that is logged in with my gmail. I didn't notice that and finished the whole process.

2 replies

Andrew 6 hours ago If it worked with your gmail, then we can go ahead and try it. I'll take a look in class before we actually start using it to make sure it looks ok. I will send an email to my contact at google just to double check.

Yi Wang 5 hours ago OK. Thank you!

Reply... ☺

Here's an example of how you can use the Cloud Shell code editor to create a sample app, push your changes to Google Cloud Source Repository, deploy the app to Google App Engine Standard, and use Stackdriver Debugger.

Andrew 11:15 AM

Message #general @ ☺

FIREBASE

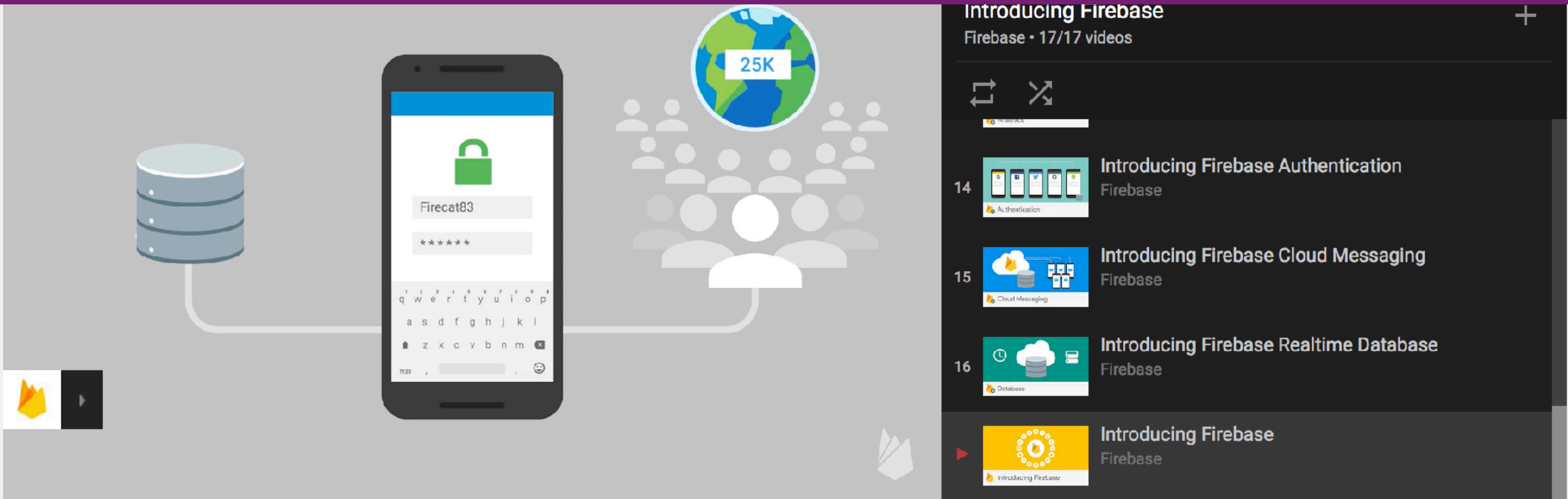


FIREBASE

- https://youtu.be/O17OWyx08Cg?list=PLI-K7zZEesYLmOF_07layrTntevxtbUxDL



FIREBASE



Introducing Firebase



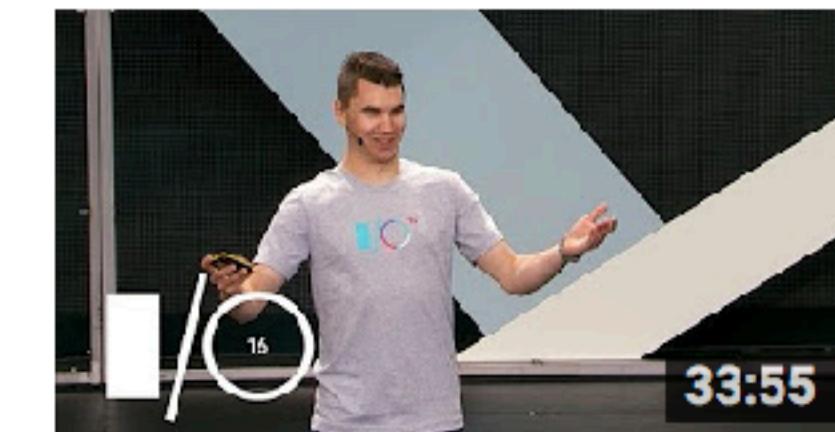
Firebase

Subscribed



56,536

287,389 views



Zero to App: Develop with
Firebase - Google I/O 2016
Firebase
119,707 views

33:55

FIREBASE IN ACTION

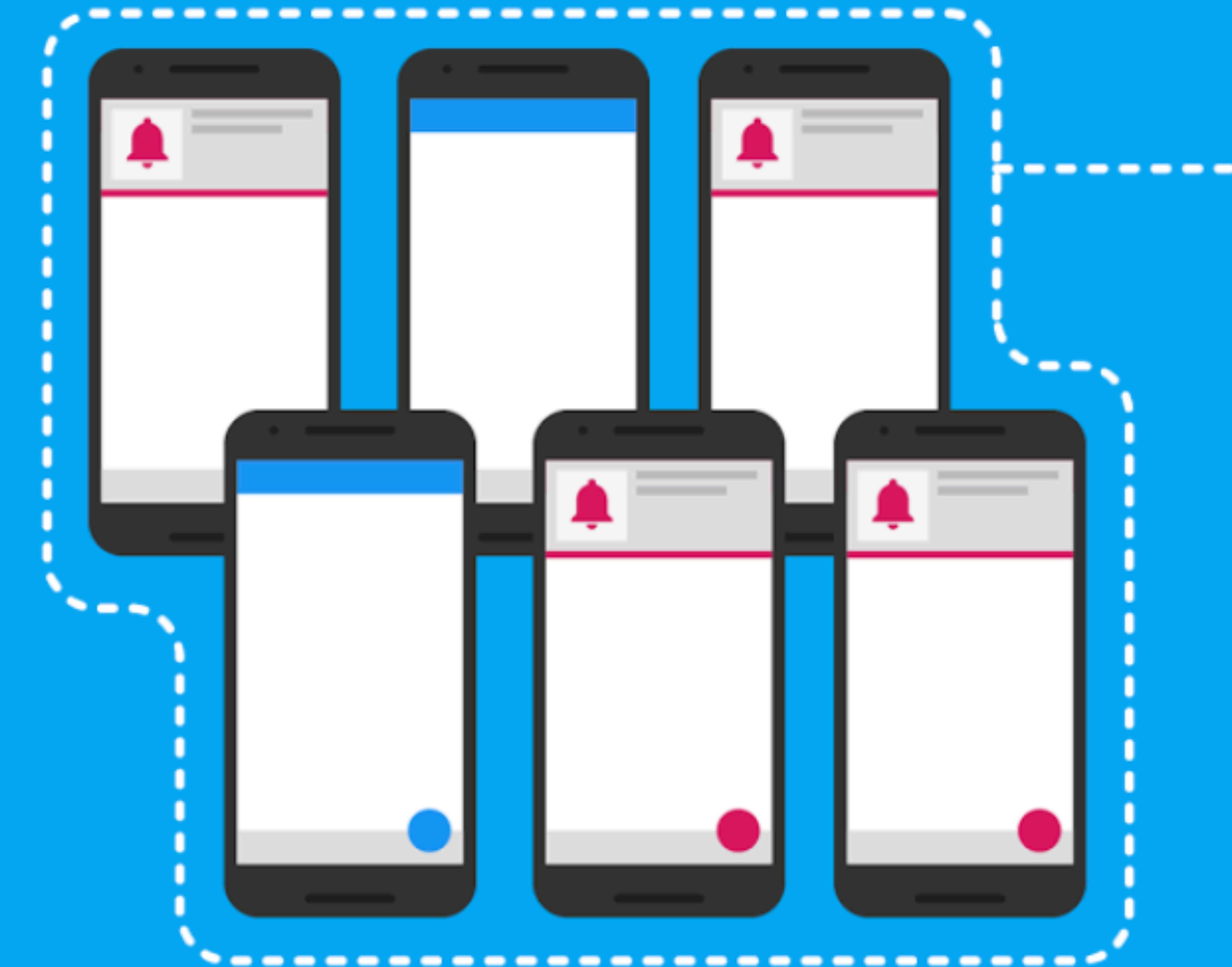
PREREQUISITES FOR FIREBASE

- 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

PREREQUISITES FOR FIREBASE

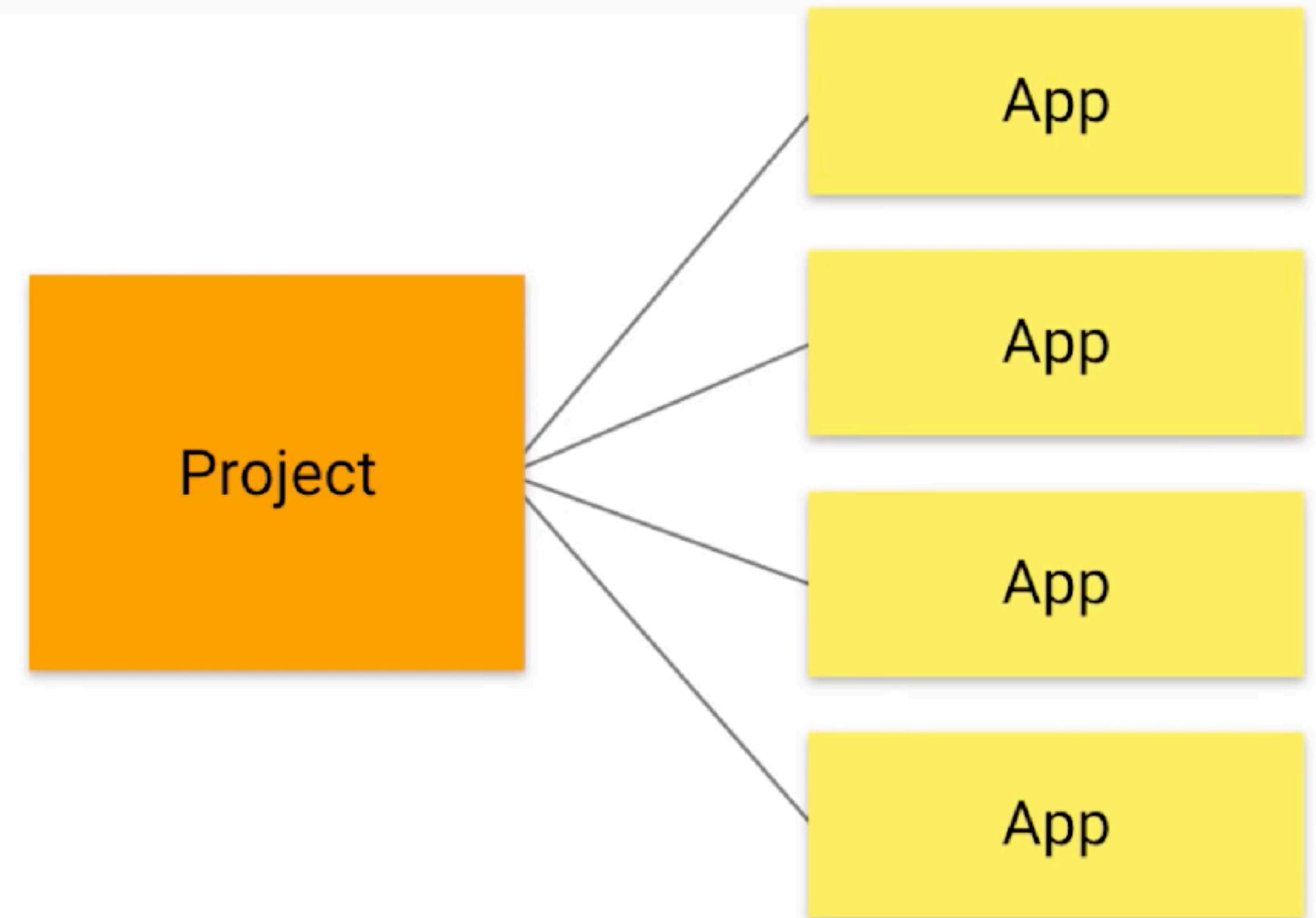
- For Cloud Messaging
 - A physical iOS device
 - APNs certificate with Push Notifications enabled
 - In Xcode, enable Push Notifications in App > Capabilities



FIREBASE IN ACTION

PROJECTS VS. APPS

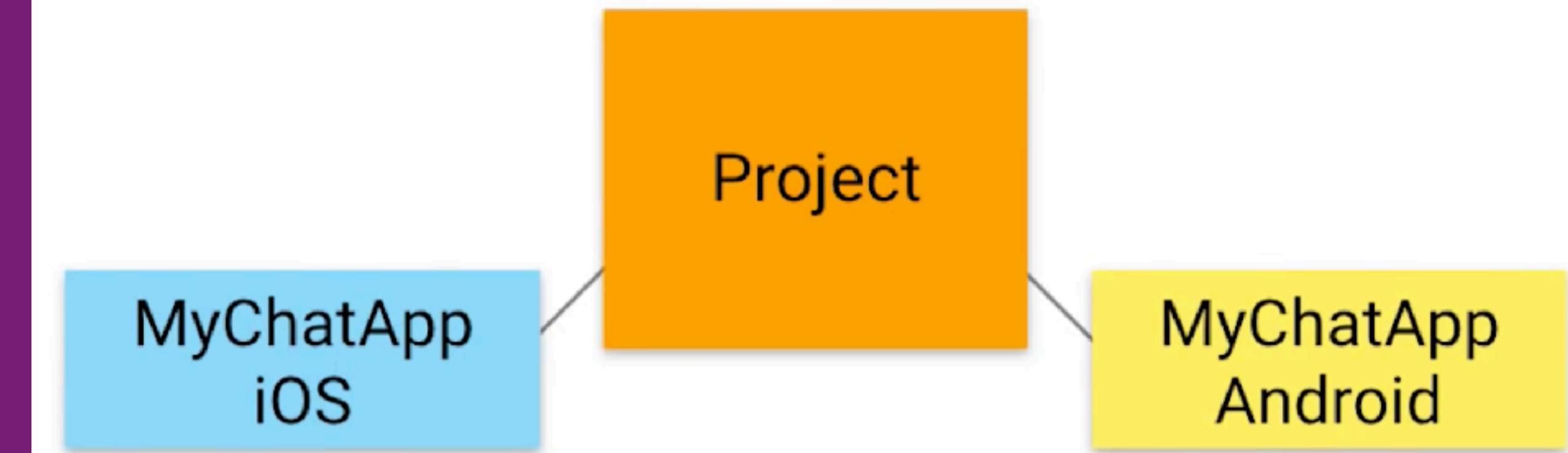
- Project consists of 1+ apps
- Apps all share the same backend



FIREBASE IN ACTION

PROJECTS VS. APPS

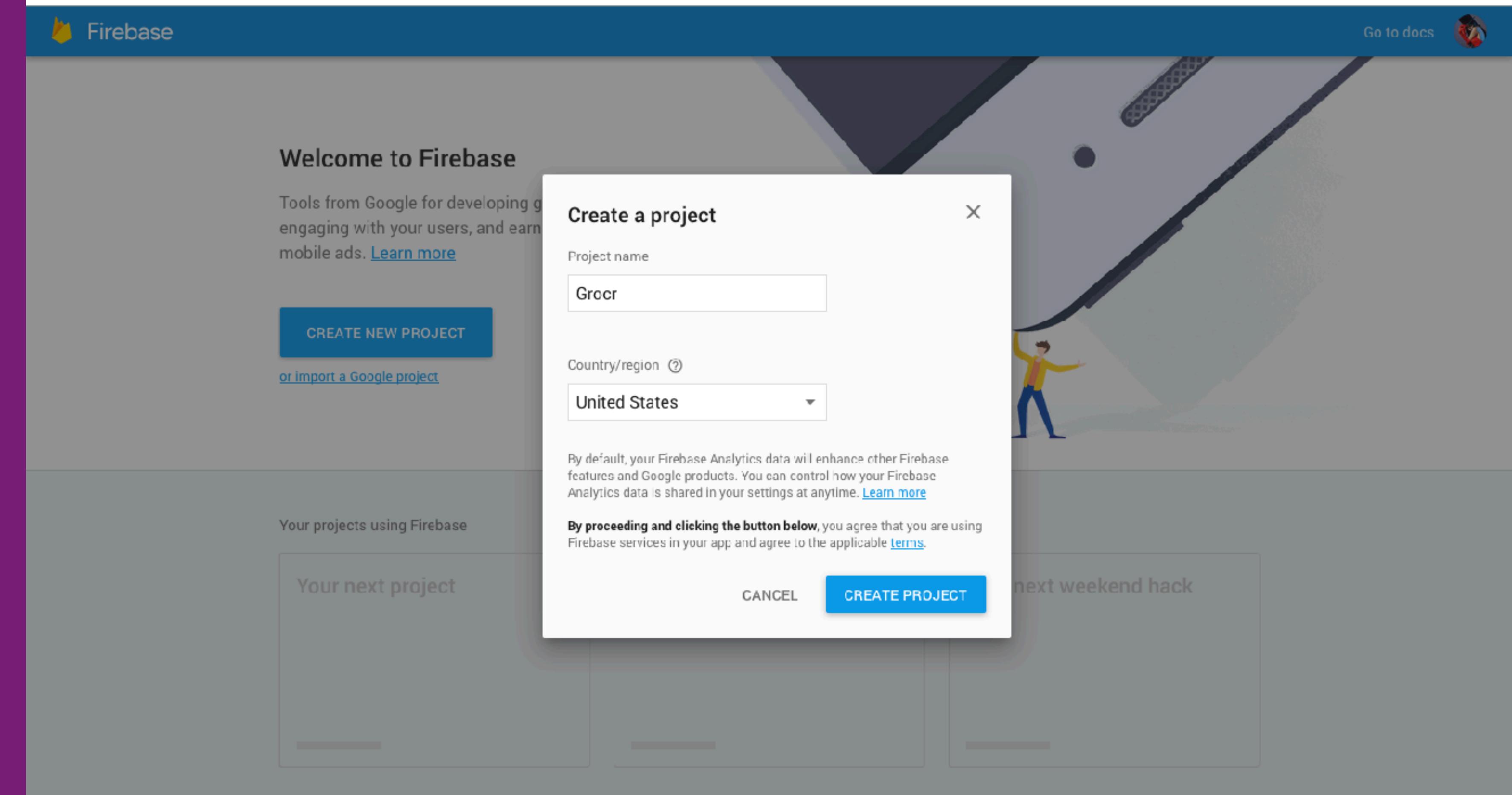
- Cross platform app
 - iOS and Android would be different apps in the same project
 - Access same data from different platforms
 - Send messages to all apps



FIREBASE IN ACTION

PROJECTS VS. APPS

- Create a Firebase project



FIREBASE IN ACTION

REGISTER APP

- Add Firebase to your app

The screenshot shows the Firebase console's Overview page for a project named "Grocr". The left sidebar contains navigation links for Analytics, Auth, Database, Storage, Hosting, Remote Config, Test Lab, and Crash. Below these are sections for Notifications, Dynamic Links, AdMob, and a "Spark Free" plan with an "UPGRADE" button. The main content area features a welcome message "Welcome to Firebase! Get started here." and three large circular buttons for "Add Firebase to your Android app" (Android icon), "Add Firebase to your iOS app" (iOS icon), and "Add Firebase to your web app" (HTML/CSS icon). Below this, there are three cards: "Analytics" (illustration of a person at a laptop with a growth chart), "Auth" (illustration of user badges), and "Database" (illustration of a database structure). Each card has a brief description: "Get detailed analytics to measure and analyze how users engage with your app", "Authenticate and manage users from a variety of providers without server-side code", and "Store and sync data in realtime across all connected clients".

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

The screenshot shows the Firebase console with the 'FireChat' project selected. The left sidebar includes 'Overview', 'Analytics', 'Authentication', 'Database', 'Storage', 'Hosting', 'Functions', 'Test Lab', and 'Crash Reporting'. The main area is titled 'Add Firebase to your iOS app' and is step 1 of 4. It contains fields for 'iOS bundle ID' (mobi.uchicago.firechat), 'App nickname (optional)' (FireChat), and 'App Store ID (optional)' (123456789). A 'REGISTER APP' button is at the bottom right, with a note 'in project FireChat' below it.

FIREBASE IN ACTION

CONFIG FILE

- Generates a .plist to include in your project

Add Firebase to your iOS app

1 Register app 2 Download config file 3 Add Firebase SDK 4 Add initialization code

Xcode instructions

Alternatives: [Unity](#) [C++](#)

1. [Download GoogleService-Info.plist](#)
2. Move the **GoogleService-Info.plist** file you just downloaded into the root of your Xcode project and add it to all targets.

The diagram shows a blue arrow pointing from a small icon of a document labeled "GoogleService-Info.plist" to a screenshot of the Xcode project navigator. The project is named "MyApplication". Inside the project, there is a folder also named "MyApplication" containing several files: AppDelegate.swift, ViewController.swift, Main.storyboard, Assets.xcassets, LaunchScreen.storyboard, Info.plist, and GoogleService-Info.plist. The "GoogleService-Info.plist" file is highlighted with a blue selection bar at the bottom of the list.

Already added the pod and initialization code?
[Skip to the console](#)

CONTINUE

UPGRADE

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	AlzaSyDjXHxBu32Fwmh9urpLpB-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 xcdeproject to xcworkspace

SEARCH*

* Type here to search by name, version, author, keywords, summary, and dependencies.



WHAT IS COCOAPODS

CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It has over 30 thousand libraries and is used in over 2 million apps. CocoaPods can help you scale your projects elegantly.

[INSTALL](#)[GET STARTED](#)[CREATE A POD](#)

FIREBASE IN ACTION

ADD SDK



FIREBASE IN ACTION

ADD SDK

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

Pod	Service
pod 'Firebase/Core'	Prerequisite libraries and Analytics
pod 'Firebase/AdMob'	AdMob
pod 'Firebase/Messaging'	Cloud Messaging / Notifications
pod 'Firebase/Database'	Realtime Database
pod 'Firebase/Invites'	Invites
pod 'Firebase/DynamicLinks'	Dynamic Links
pod 'Firebase/Crash'	Crash Reporting
pod 'Firebase/RemoteConfig'	Remote Config
pod 'Firebase/Auth'	Authentication
pod 'Firebase/Storage'	Storage

FIREBASE IN ACTION

ADD SDK

- Install the Firebase pods in your project

The screenshot shows the Firebase console with the 'Overview' tab selected. A modal window titled 'Add Firebase to your iOS app' is open, divided into four steps: 1. Register app, 2. Download config file, 3. Add Firebase SDK (which is currently selected), and 4. Add initialization code. Step 3 is highlighted with a blue background. The modal contains instructions for using CocoaPods to add the Firebase Core pod to an Xcode project. It provides three command-line steps: 1. Create a Podfile if you don't have one: `$ pod init`, 2. Open your Podfile and add: `pod 'Firebase/Core'` (with a note that it includes Firebase Analytics by default), and 3. Save the file and run: `$ pod install`. Below these steps, a note states: 'This creates an .xcworkspace file for your app. Use this file for all future development on your application.' At the bottom left of the modal, there's a link 'Already added the pod and initialization code? Skip to the console'. On the right side of the modal, there's a blue 'CONTINUE' button. The background of the modal shows a blurred view of the Firebase console interface.

Add Firebase to your iOS app

1 2 3 4

Register app Download config file Add Firebase SDK Add initialization code

CocoaPods instructions

Alternatives: [Download ZIP](#) [Unity](#) [C++](#)

Google services use [CocoaPods](#) to install and manage dependencies. Open a terminal window and navigate to the location of the Xcode project for your app.

1. Create a Podfile if you don't have one: `$ pod init`
2. Open your Podfile and add: `pod 'Firebase/Core'`
includes Firebase Analytics by default
3. Save the file and run: `$ pod install`

This creates an .xcworkspace file for your app. Use this file for all future development on your application.

Already added the pod and initialization code?
[Skip to the console](#)

CONTINUE

Spark Free \$0/month UPGRADE

FIREBASE IN ACTION

ADD SDK

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

The screenshot shows the Firebase console with the 'Overview' tab selected. A modal window titled 'Add Firebase to your iOS app' is open, showing a four-step process. Step 4, 'Add initialization code', is currently active. It contains instructions to add initialization code to the main AppDelegate class, with options for Swift or Objective-C. The Swift code provided is:

```
import UIKit
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(application: UIApplication,
                    didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        FIRApp.configure()
        return true
    }
}
```

A blue 'FINISH' button is at the bottom right of the modal.

FIREBASE IN ACTION

CONSOLE

 **Firebase** FireChat ▾ Go to docs ⋮ 

 [Overview](#)  Overview 

 [Analytics](#)

DEVELOP

 [Authentication](#)

 [Database](#)

 [Storage](#)

 [Hosting](#)

 [Functions](#)

 [Test Lab](#)

 [Crash Reporting](#)

FireChat mobile apps

 **mobi.uchicago.firechat** 

[Explore Firebase Analytics →](#)

0 Monthly active users	\$0 In-app purchases
Crashes (30 days)	
0 Users impacted	0 Instances

 [Add another app](#)

FIREBASE IN ACTION

BUILD THE APP

```
2017-04-19 01:16:22.866 FireChat[63153] <Notice> [Firebase/Messaging][I-IID001001] FIRInstanceID AppDelegate proxy enabled,  
will swizzle app delegate remote notification handlers. To disable add "FirebaseAppDelegateProxyEnabled" to your Info.plist and  
set it to NO  
2017-04-19 01:16:22.883 FireChat[63153] <Notice> [Firebase/Analytics][I-ACS023007] Firebase Analytics v.3800000 started  
2017-04-19 01:16:22.884 FireChat[63153] <Notice> [Firebase/Analytics][I-ACS023008] To enable debug logging set the following  
application argument: -FIRAnalyticsDebugEnabled (see http://goo.gl/RfcP7r)  
2017-04-19 01:16:22.905 FireChat[63153] <Notice> [Firebase/Analytics][I-ACS003007] Successfully created Firebase Analytics App  
Delegate Proxy automatically. To disable the proxy, set the flag FirebaseAppDelegateProxyEnabled to NO in the Info.plist  
2017-04-19 01:16:23.209 FireChat[63153] <Warning> [Firebase/Analytics][I-ACS005000] The AdSupport Framework is not currently  
linked. Some features will not function properly. Learn more at http://goo.gl/9vSsPb  
2017-04-19 01:16:23.520 FireChat[63153] <Notice> [Firebase/Analytics][I-ACS023012] Firebase Analytics enabled
```

CONSOLE GETS PRETTY MESSY FROM FIREBASE LOGGING

REALTIME DATABASE

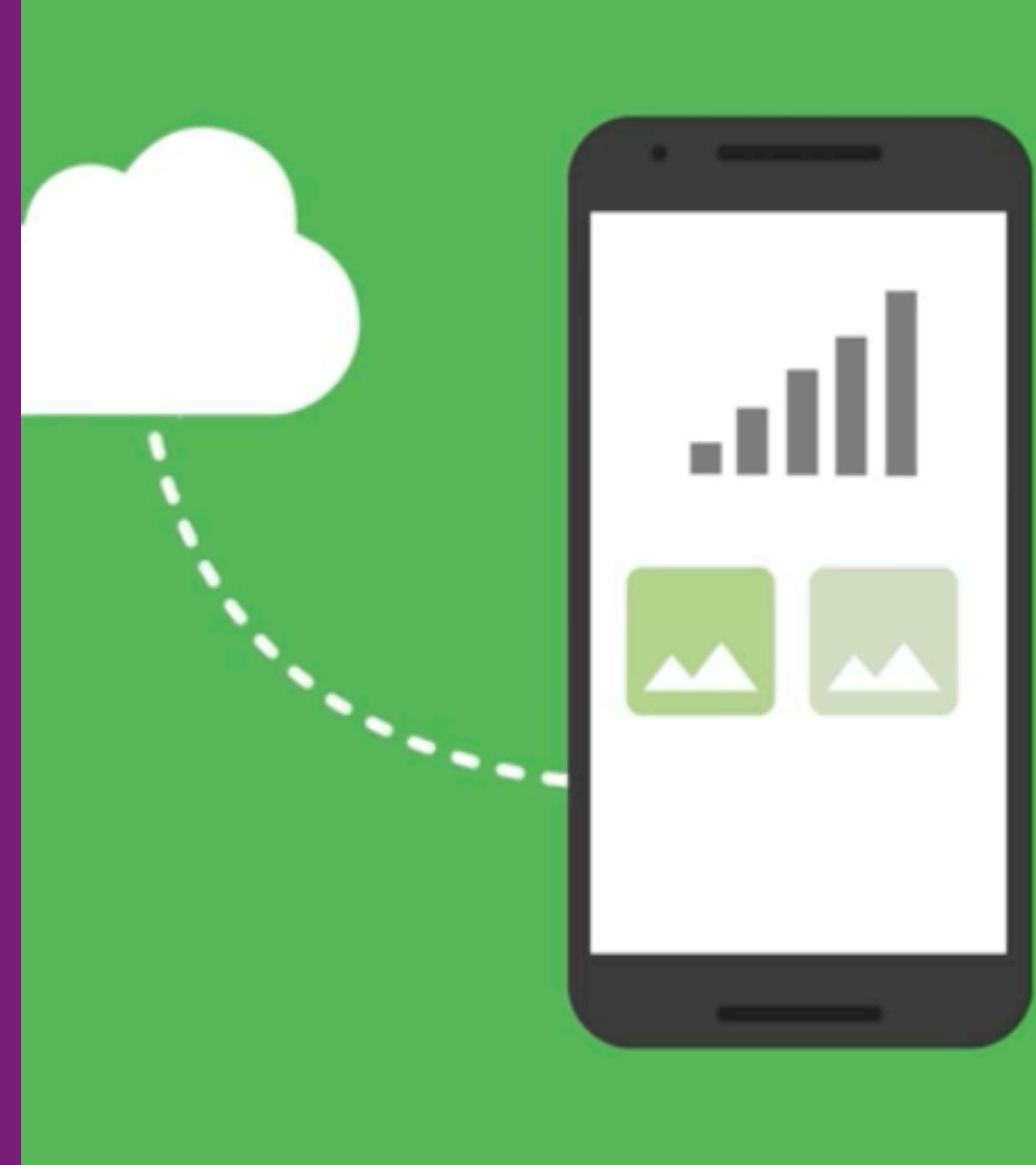
REALTIME DATABASE

- 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=PLI-K7zZEesYLmOF_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 is offline
- Changes are automatically synced when the device comes back online

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.

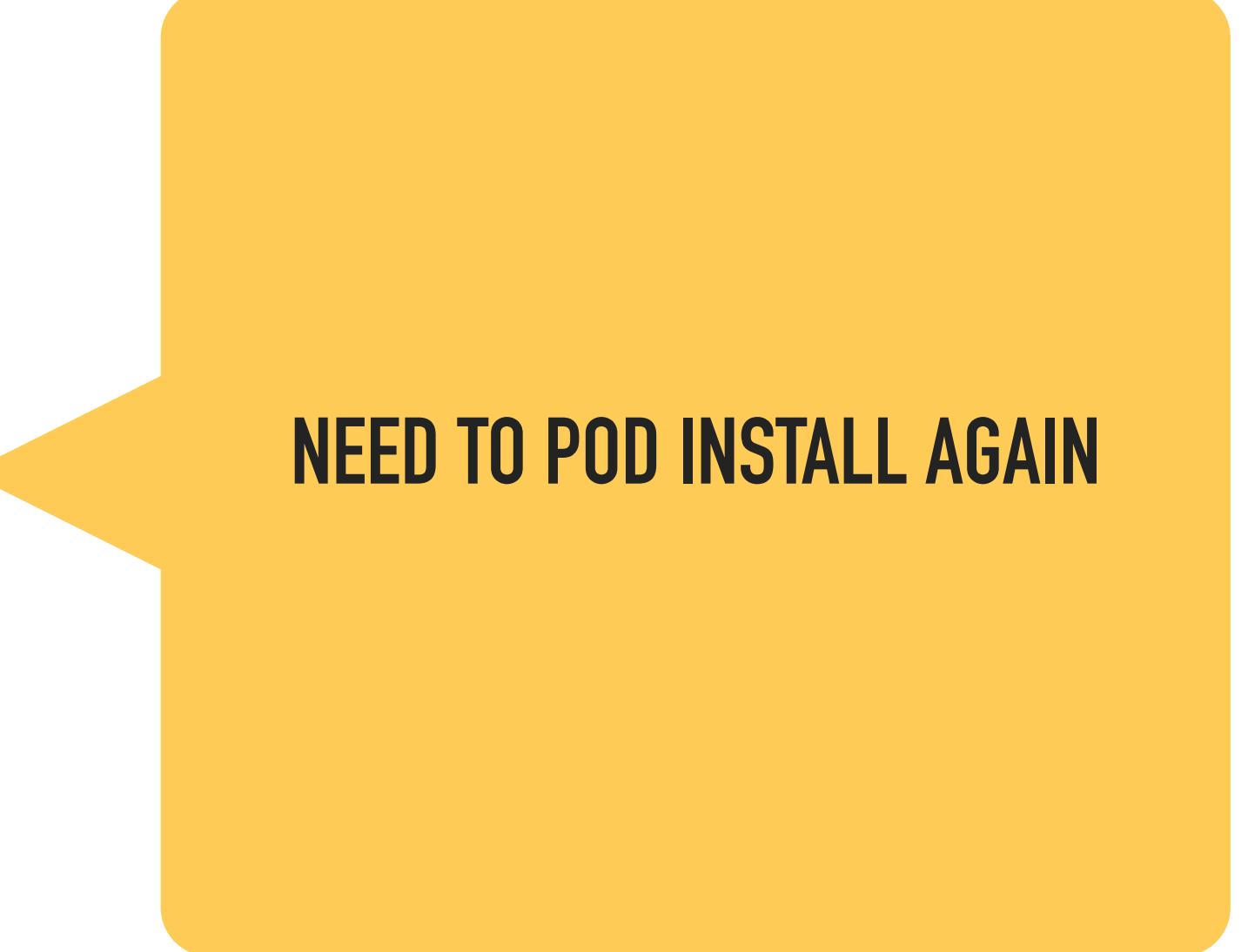
REALTIME DATABASE

```
platform :ios, '9.0'

target 'FireChat' do
  # Comment the next line if you're not using Swift and don't want to use dynamic
  # frameworks
  use_frameworks!

  # Pods for FireChat
  pod 'Firebase/Core'
  pod 'Firebase/Database'

end
```



NEED TO POD INSTALL AGAIN

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

The screenshot shows the Firebase Realtime Database Rules tab. The rules are defined as follows:

```
1 {  
2   "rules": {  
3     ".read": true,  
4     ".write": true  
5   }  
6 }
```

The Simulator panel is open, showing the following configuration:

- Simulation type: Read (radio button selected)
- Location: <https://firechat-66f87.firebaseio.com>
- Authenticated: Off (switch is off)

A large blue "RUN" button is at the bottom of the simulator panel.

REALTIME DATABASE

```
// These rules require authentication
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

- 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

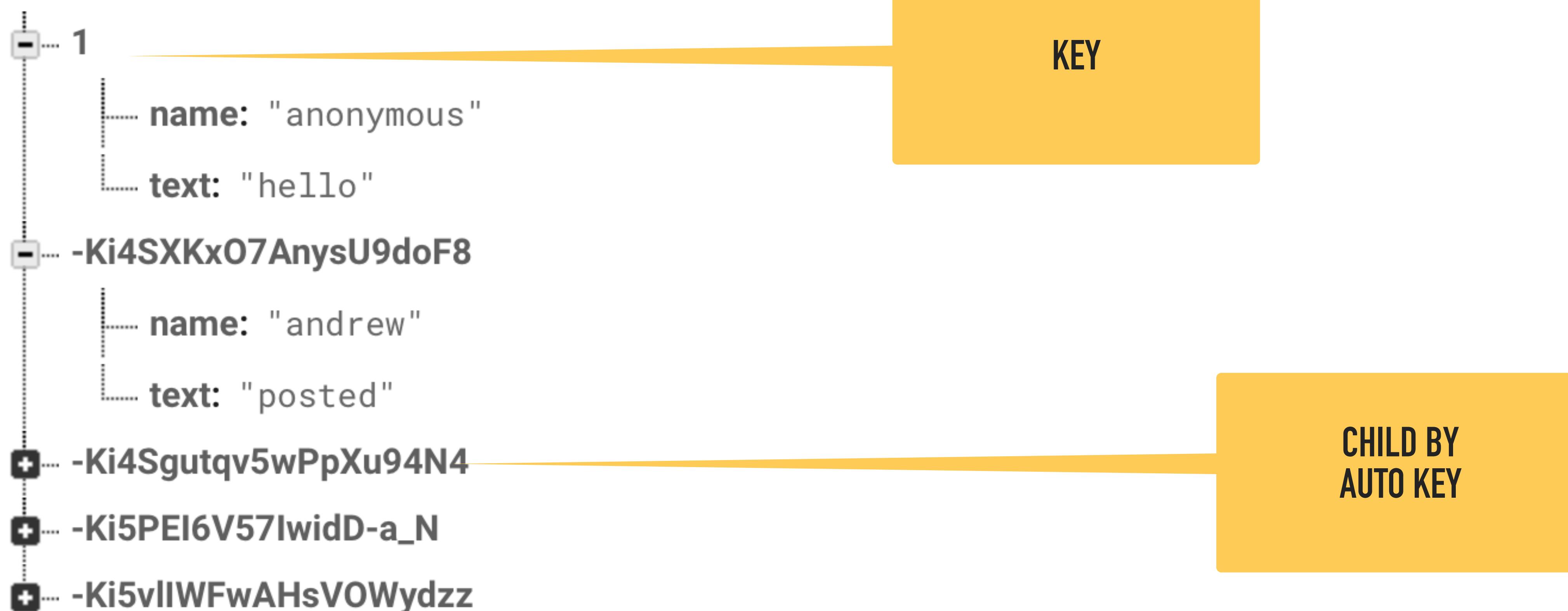
- JSON trees stored online
- Use multiple trees (not linked)
- Data is represented as a node accessible by a key
 - Use your own key (string)
 - `childByAutold`

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

DATA STRUCTURE

[firechat-66f87](#) > [messages](#)

messages



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 conversations  
    // potentially downloading hundreds of megabytes  
    "chats": {  
        "one": {  
            "title": "Historical Tech Pioneers",  
            "messages": {  
                "m1": { "sender": "ghopper", "message": "Hello, world!" },  
                "m2": { ... },  
                // a very long list of messages  
            }  
        },  
        "two": { ... }  
    }  
}
```

DATA STRUCTURE

BEST PRACTICES FOR DATA

- 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.",  
                "timestamp": 1459361875667  
            }  
        }  
    }  
}
```

DATA STRUCTURE

BEST PRACTICES FOR DATA

- 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.",  
                "timestamp": 1459361875667  
            }  
        }  
    }  
}
```

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 AND WRITE DATA

READ AND WRITE DATA

- Get a reference to the database

```
var ref: FIRDatabaseReference!  
  
ref = FIRDatabase.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

OBSERVE CHANGERS TO DATA

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

    var freshData = [MessageItem]()
    for item in snapshot.children {
        let messageItem = MessageItem(snapshot: item as! FIRDataSnapshot)
        freshData.append(messageItem)
    }
    self.messages = freshData
    print(self.messages)
    self.tableView.reloadData()
})
```

DATA RETURNED AS SNAPSHOT

READ AND WRITE DATA

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

```
struct MessageItem {  
  
    let key: String  
    let name: String  
    let text: String  
    let ref: FIRDatabaseReference?  
  
    init(name: String, text: String, key: String = "") {  
        self.key = key  
        self.name = name  
        self.text = text  
        self.ref = nil  
    }  
  
    init(snapshot: FIRDataSnapshot) {  
        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

```
struct MessageItem {  
  
    let key: String  
    let name: String  
    let text: String  
    let ref: FIRDatabaseReference?  
  
    init(name: String, text: String, key: String = "") {  
        self.key = key  
        self.name = name  
        self.text = text  
        self.ref = nil  
    }  
  
    init(snapshot: FIRDataSnapshot) {  
        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

```
let userID = FIRAuth.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.)

READ AND 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)
```

READ AND 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

READ AND WRITE DATA

```
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)
    }
}
```

ADD A NEW MESSAGE

READ AND 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

```
base().reference().child("messages")  
  
ender: UIBarButtonItem) {  
    title: "New Message",  
    message: "Message",  
    preferredStyle: .alert)  
  
(title: "Post",  
    style: .default) { _ in  
  
    guard let textField = alert.textField  
        let text = textField.text else {  
            return  
    }  
  
    let data = ["name":self.user?.display  
    self.reference.childByAutoId().setVal
```

WORKING WITH LISTS OF DATA

Event type	Typical usage
FIRDataEventTypeChildAdded	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.
FIRDataEventTypeChildChanged	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.
FIRDataEventTypeChildRemoved	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.
FIRDataEventTypeChildMoved	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 <code>queryOrderedByChild</code> or <code>queryOrderedByValue</code> .

- Monitor for child events

WORKING WITH LISTS OF DATA

controller.swift
oard
list

```
64    override func viewWillAppears(animated: Bool) {  
65        super.viewWillAppears(animated)  
66  
67        // Load the data and observe  
68        // uploadFile()  
69  
70        reference.observe(. , with: { snapshot in  
71            FIRDataEventType childAdded  
72            FIRDataEventType childChanged  
73            FIRDataEventType childMoved  
74            FIRDataEventType childRemoved  
75            FIRDataEventType value  
76            FIRDataSnapshot item as! FIRDataSnapshot  
77            A new child node is added to a location.  
78            self.messages = freshData  
79            //print(self.messages)  
80            self.tableView.reloadData()  
81        })
```

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.kSectionComments)])
})
// 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: UITableViewRowAnimation.fade)
})
```

RETURNS THE CHANGED NODE ONLY

WORKING WITH LISTS OF DATA

```
_commentsRef.observeEventType(.Value, withBlock: { snapshot in
    for child in snapshot.children {
        ...
    }
})
```

- Returns everything
- Returns an array

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.

- 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.
queryEqualToString	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 = FIRDatabase.database().referenceWithPath("scores")
scoresRef.keepSynced(true)
```

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

STORE DATA OFFLINE

```
let scoresRef = FIRDatabase.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

```
[FIRDatabase database].persistenceEnabled = YES;
```

- That's it 😊

STORE DATA OFFLINE

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

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

- Manage presence (ie who is connected)

STORE DATA OFFLINE

```
// 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

STORE DATA OFFLINE

```
// 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())
})|
```

STORE DATA OFFLINE

```
// 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())
})|
```

STORE DATA OFFLINE

```
// 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())
})|
```

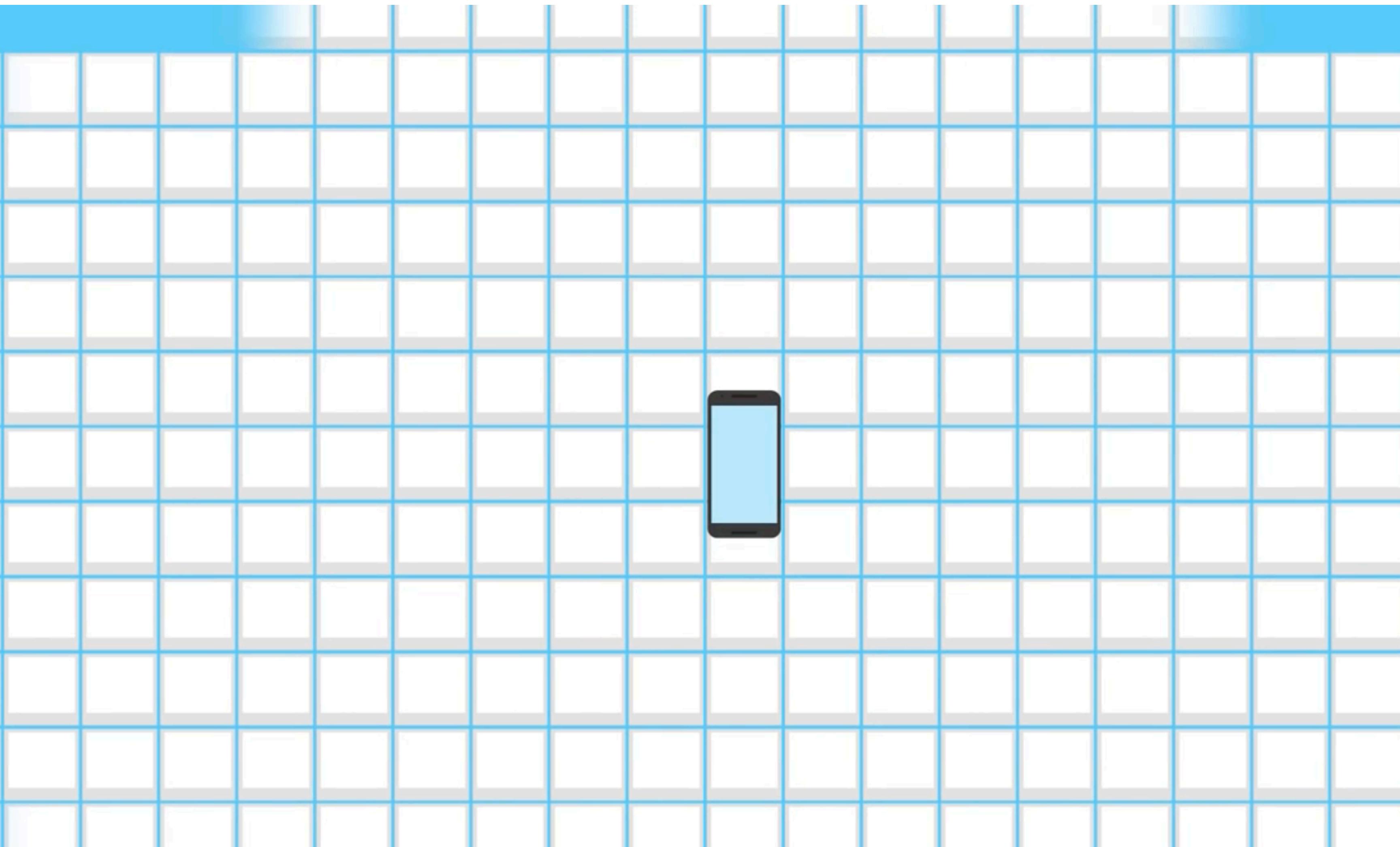
BREAK TIME



FIREBASE STORAGE

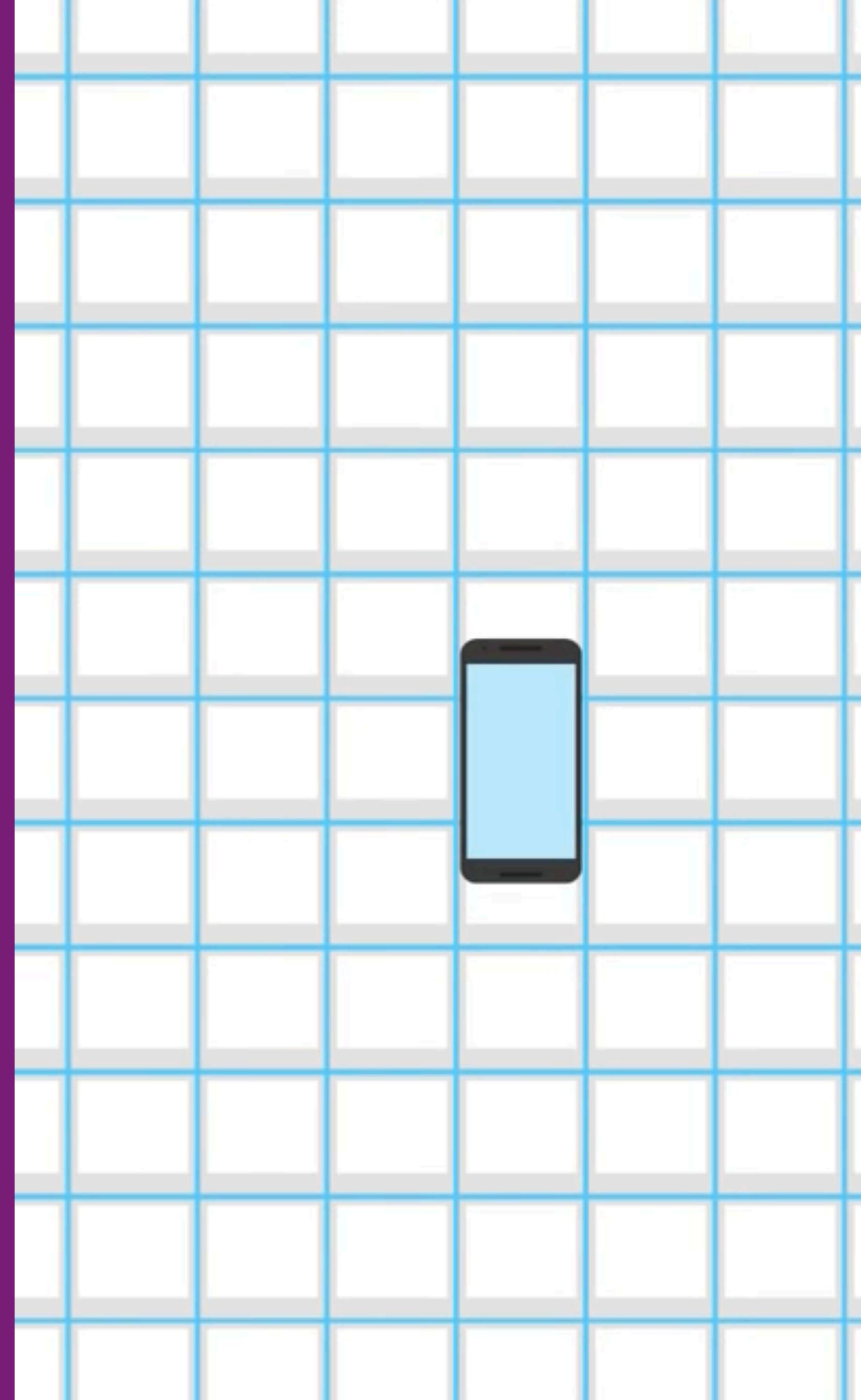
FIREBASE STORAGE

- [https://
www.youtube.co
m/watch?list=PLI-
K7zZEsvxtbUxDL
ayrTntevxtbUxDL
&v=_tyjqozrEPY](https://www.youtube.com/watch?list=PLIK7zZEsvxtbUxDLayrTntevxtbUxDL&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

Storage

FILES RULES

<input type="checkbox"/>	Name	Size	Type	Last modified
	gs://firechat-66f87.appspot.com			

★ Default security rules require users to be authenticated

[LEARN MORE](#) [DISMISS](#)

 Store and retrieve user-generated files like images, audio, and video without server-side code

[Learn more](#)

 UPLOAD FILE

FIREBASE STORAGE

- Online console for storage
- Set permission rules

The screenshot shows the Firebase Storage Rules page. At the top, there are navigation links for 'FILES' and 'RULES'. The 'RULES' tab is selected, indicated by a blue underline. Below the tabs, a star icon and the text 'Default security rules require users to be authenticated' are displayed. A code block shows the default security rules:

```
1 service firebase.storage {  
2     match /b/{bucket}/o {  
3         match /{allPaths=**} {  
4             allow read, write: if request.auth != null;  
5         }  
6     }  
7 }  
8 }
```

FIREBASE STORAGE

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

target 'FireChat' do
  # Comment the next line if you're not using Swift and don't want to use dynamic frameworks!
  use_frameworks!

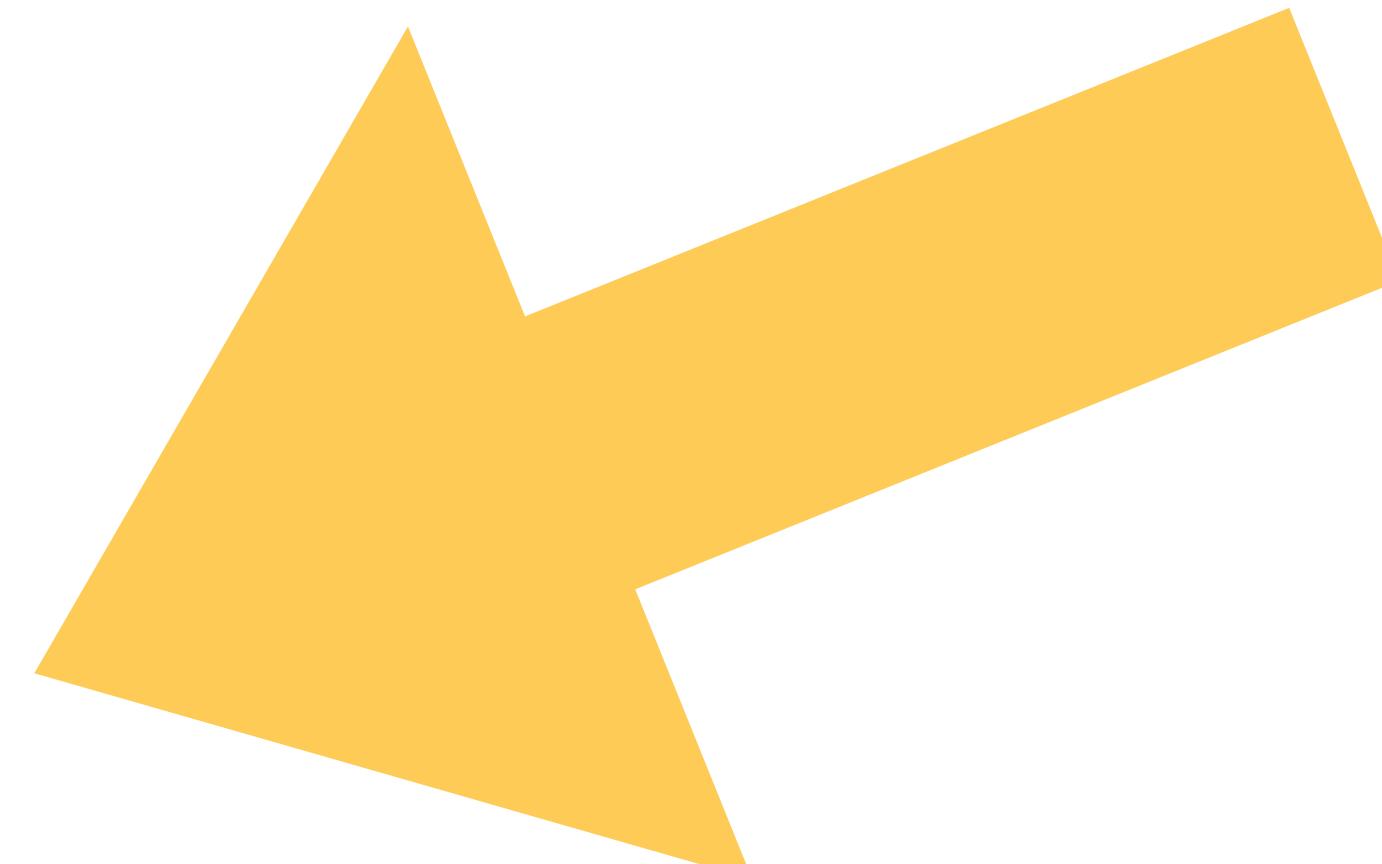
  # Pods for FireChat
  pod 'Firebase/Core'
  pod 'Firebase/Database'

  pod 'Firebase/Auth'
  pod 'GoogleSignIn'

  pod 'Firebase/Crash'

  pod 'Firebase/Storage'

end
```



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 accessible by path

```
// Points to the root reference
let storageRef = FIRStorage.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

```
// 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.put(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, and download URL.
    let downloadURL = metadata.downloadURL
}
```

- Upload from image in memory

FIREBASE STORAGE

```
// 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(localFile, metadata: nil) { metadata, error in
    if let error = error {
        // Uh-oh, an error occurred!
    } else {
        // Metadata contains file metadata such as size, content-type, and download URL.
        let downloadURL = metadata!.downloadURL()
    }
}
```

- Upload from image on disk

FIREBASE STORAGE

```
func uploadFile() {
    // Local file you want to upload
    let data = UIImagePNGRepresentation(UIImage(named:"Kitten")!)
    let kittenRef = storageRef.child("images/kitten.jpg")

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

    // Upload the file to the path "images/kitten.jpg"
    let _ = kittenRef.put(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, and download URL.
        let downloadURL = metadata.downloadURL()
        print("📸 URL: \(downloadURL!)")
    }
}
```

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 = FIRStorageMetadata()
metadata.contentType = "image/jpeg"

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

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

FIREBASE STORAGE

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

- Observe progress on a file upload

FIREBASE STORAGE

```
// 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

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/stars.jpg")

// 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/bucket/o/im
```

- Files are accessible through different APIs

FIREBASE STORAGE

- Files are accessible through different APIs and locations

gs://firechat-66f87.appspot.com > images

UPLOAD FILE **+**

Name	Size	Type	Last modified
kitten.jpg	1,0...	application/	Oct 16, ...

kitten.jpg

Name
kitten.jpg

Size
1,023.14 KB

Type
application/octet-stream

Created
Oct 16, 2017, 7:50:48 PM

Updated
Oct 16, 2017, 7:50:48 PM

File location

Storage location
gs://firechat-66f87.appspot.com/images/kitten.jpg

Download URL 1 [revoke](#)
<https://fbx.fbsr...-9bf1-7603b7722680>

[Create new download URL](#)

Other metadata

No metadata found

FIREBASE STORAGE

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

// Fetch the download URL
starsRef.downloadURL { url, error in
  if let error = error {
    // Handle any errors
  } else {
    // Get the download URL for 'images/stars.jpg'
  }
}
```

- Files are accessible through different APIs

FIREBASE STORAGE

```
// 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
    }
}
```

- Download to file on disk

FIREBASE STORAGE

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



build passing pod v4.0.0 platform osx | ios | tvos | watchos license MIT dependencies up to date references
codecov 76%

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

Features

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

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

target 'FireChat' do
  # Comment the next line if you're not using Swift and don't want to use dynamic frameworks
  use_frameworks!

  # Pods for FireChat
  pod 'Firebase/Core'
  pod 'Firebase/Database'
  pod 'Firebase/Auth'
  pod 'GoogleSignIn'
  pod 'Firebase/Crash'
  pod 'Firebase/Storage'

  pod 'FirebaseUI/Storage'
  pod 'SDWebImage'

end
```

FIREBASE STORAGE

```
import UIKit
import Firebase
import SDWebImage
import FirebaseStorageUI

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

    // Load the image using SDWebImage
    let reference = storageRef.child("images/kitten.jpg")
    let placeholderImage = UIImage(named: "Placeholder")
    // Load the image using SDWebImage
    cell.imageView?.sd_setImage(with: reference, placeholderImage: placeholderImage)
    return cell
}
```

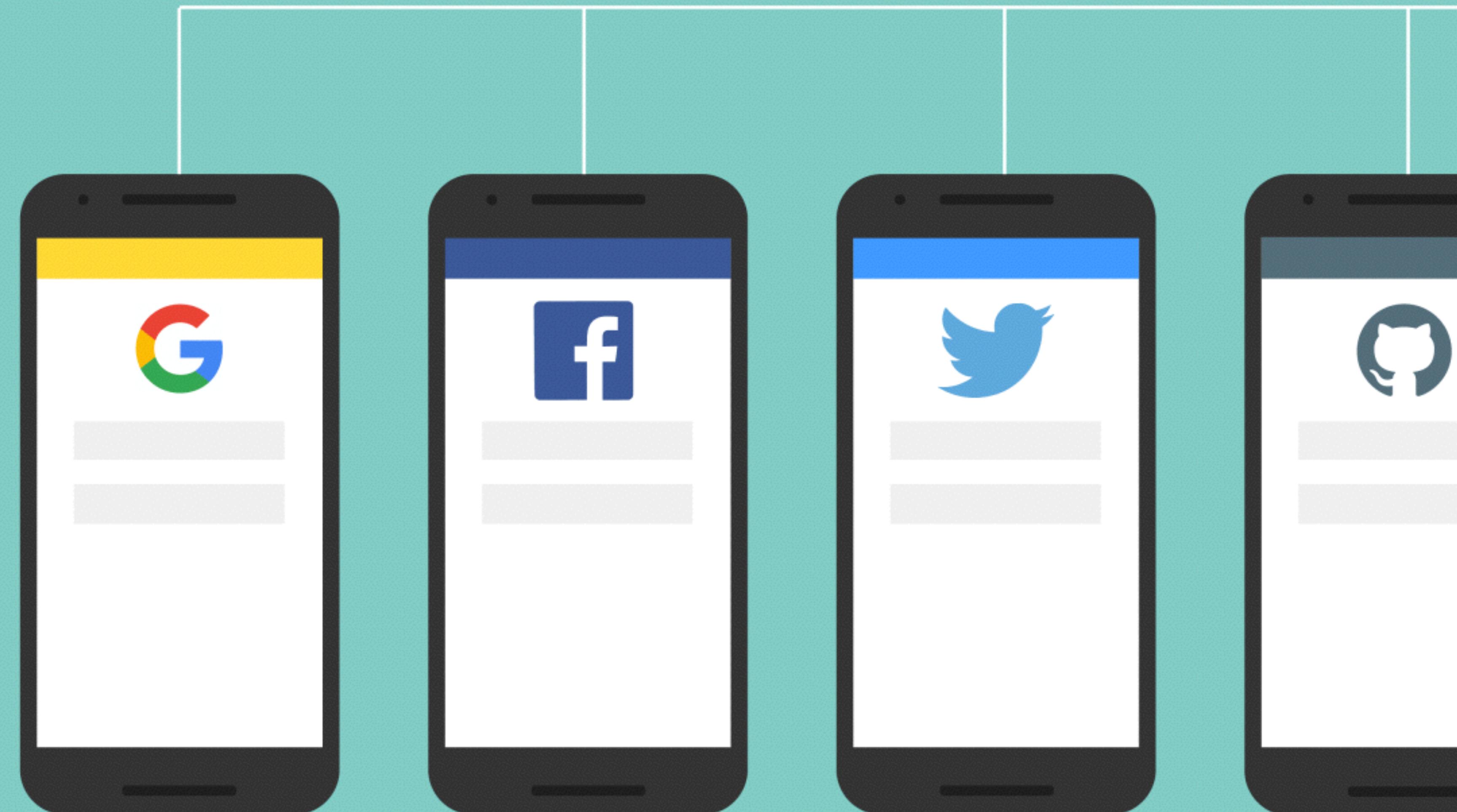
- Download directly to UIImage; swap placeholder image out

FIREBASE AUTHENTICATION

FIREBASE AUTHENTICATION

- [https://
www.youtube.co
m/watch?
v=8sGY55yxicA&i
ndex=14&list=PLI
=_
K7zZEyLmOF_07
layrTntevxtbUxDL](https://www.youtube.com/watch?v=8sGY55yxicA&index=14&list=PLI_=_K7zZEyLmOF_07layrTntevxtbUxDL)

Firebase



FIREBASE AUTHENTICATION

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

Authentication

WEB SETUP ?

USERS SIGN-IN METHOD EMAIL TEMPLATES

Search by email address or user UID ADD USER C

Email	Providers	Created ↓	Signed In	User UID ↑
-------	-----------	-----------	-----------	------------



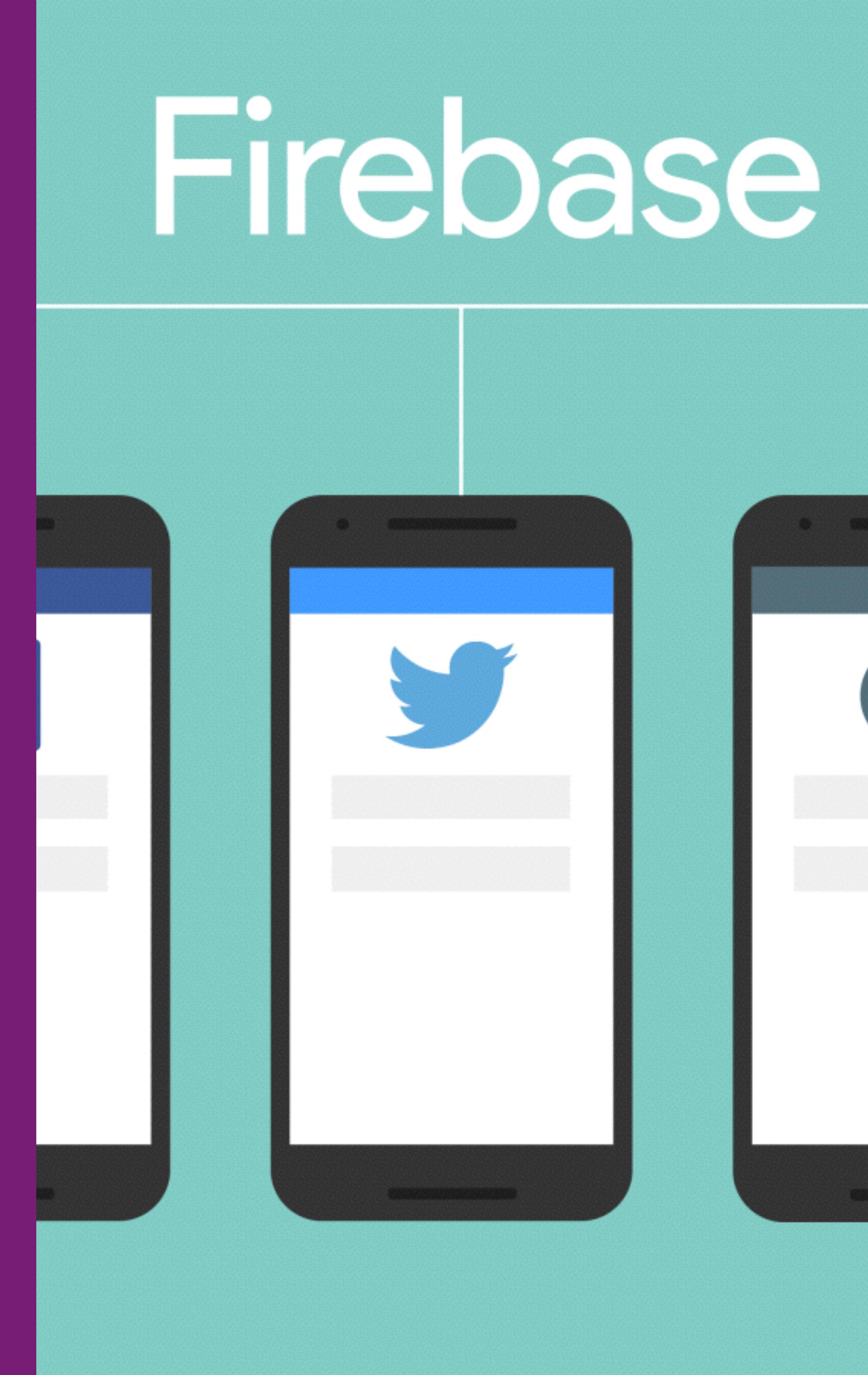
Authenticate and manage users from a variety of providers without server-side code

[Learn more](#)

SET UP SIGN-IN METHOD

FIREBASE AUTHENTICATION

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



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 FireChat ▾

Authentication

USERS SIGN-IN METHOD EMAIL TEMPLATES

Sign-in providers

Provider	Status
Email/Password	Disabled
Google	Disabled
Facebook	Disabled
Twitter	Disabled
Github	Disabled
Anonymous	Disabled

FIREBASE AUTHENTICATION

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

Firebase FireChat Authentication Go to docs :

Facebook Disabled

Twitter Disabled

Github

Enable

Client ID

Client secret

To complete set up, add this authorization callback URL to your GitHub app configuration.
[Learn more](#) 

`https://firechat-66f87.firebaseio.com/_/auth/handler` 

CANCEL SAVE

Anonymous Disabled

FIREBASE AUTHENTICATION

```
platform :ios, '9.0'

target 'FireChat' do
  # Comment the next line if you're not using Swift and don't want
  # to use dynamic frameworks
  use_frameworks!

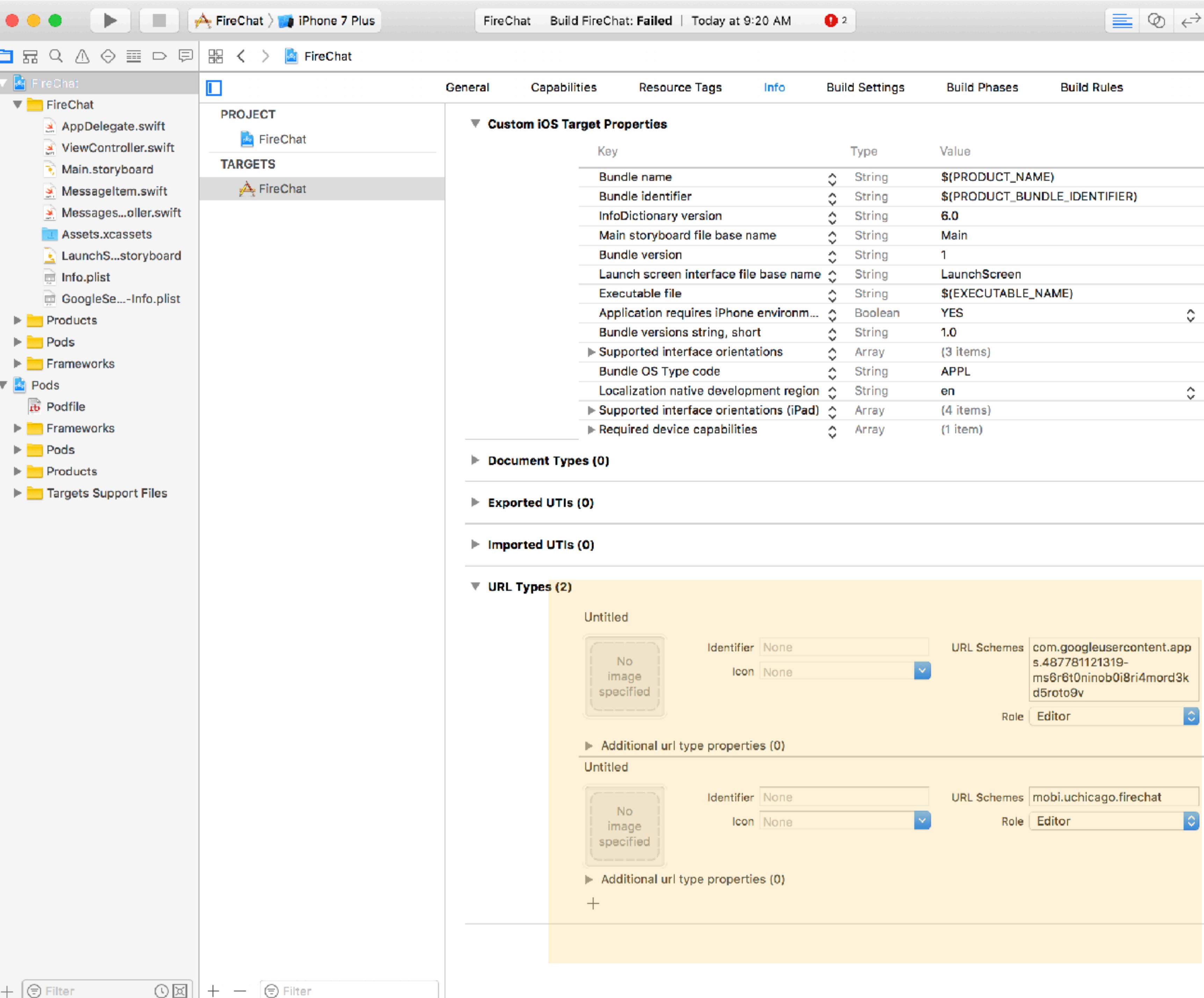
  # Pods for FireChat
  pod 'Firebase/Core'
  pod 'Firebase/Database'

  pod 'Firebase/Auth'
  pod 'GoogleSignIn'

end
```

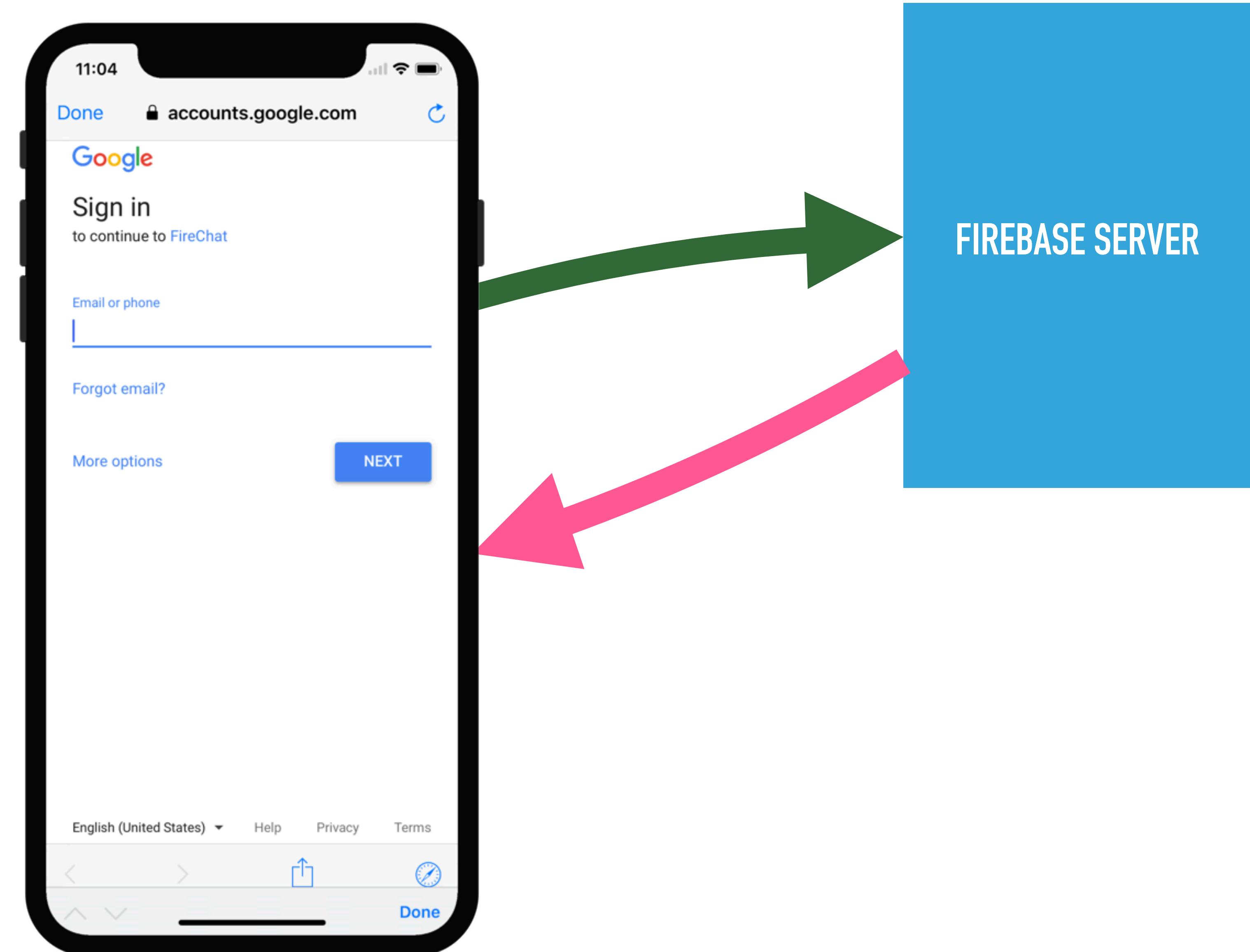
FIREBASE AUTHENTICATION

- Add URL types to the target info
- Provides way to launch app from login callback



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

```
// AppDelegate.swift
// FireChat
//
// Created by T. Andrew Binkowski on 4/19/17.
// Copyright © 2017 T. Andrew Binkowski. All rights reserved.

import UIKit
import Firebase
import GoogleSignIn

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, GIDSignInDelegate {

    var window: UIWindow?

    override init() { ... }

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

        func sign(_ signIn: GIDSignIn!, didDisconnectWith user: GIDGoogleUser!, withError error: Error!) {
            ...
        }
    }

    func applicationWillResignActive(_ application: UIApplication) { ... }

    func applicationDidEnterBackground(_ application: UIApplication) { ... }

    func applicationWillEnterForeground(_ application: UIApplication) { ... }

    func applicationDidBecomeActive(_ application: UIApplication) { ... }

    func applicationWillTerminate(_ application: UIApplication) { ... }
}
```

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

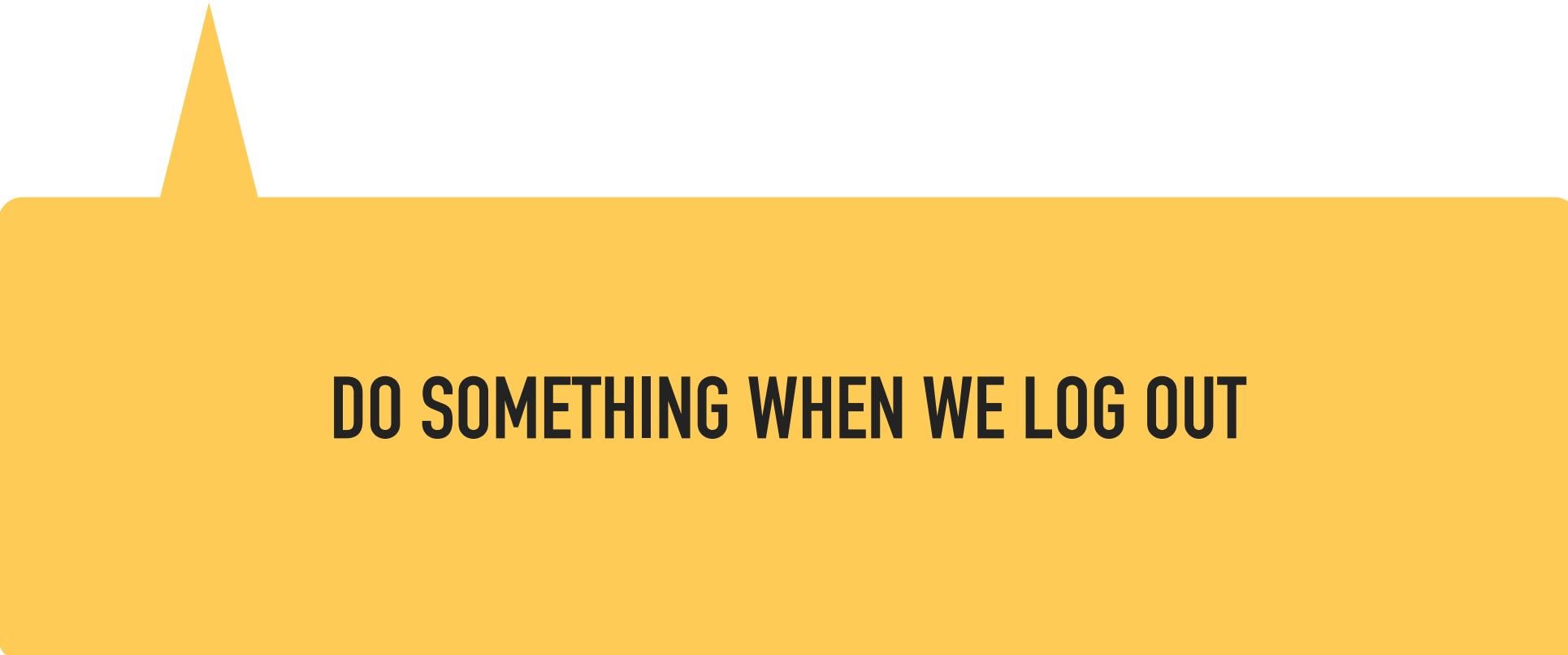
```
//  
// MARK: - Google Sign In/Out Delegate Methods  
  
func sign(_ signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!,WithError error: Error?) {  
    if let error = error {  
        print(error.localizedDescription)  
        return  
    }  
  
    // Use Google authentication to get credentials  
    guard let authentication = user.authentication else { return }  
    let credential = FIRGoogleAuthProvider.credential(withIDToken: authentication.idToken,  
                                                    accessToken: authentication.accessToken)  
    FIRAuth.auth()?.signIn(with: credential) { (user, error) in  
        if let error = error {  
            print(error.localizedDescription)  
            return  
        }  
    }  
}  
  
func sign(_ signIn: GIDSignIn!, didDisconnectWith user: GIDGoogleUser!,WithError error: Error!) { ... }
```

AUTHENTICATE
FROM GOOGLE

SUCCESS FROM
GOOGLE NOW PASS
TO FIREBASE

FIREBASE AUTHENTICATION

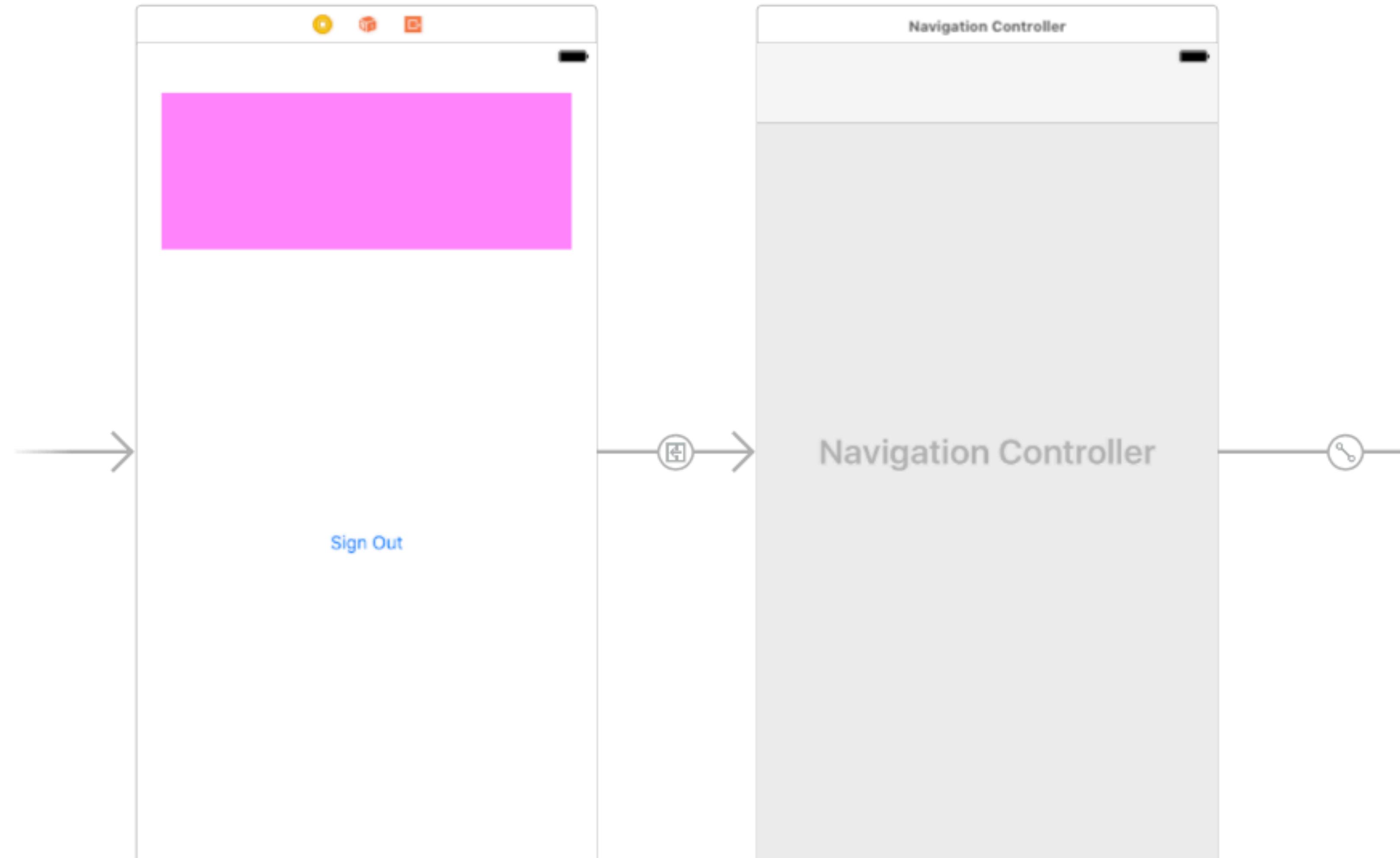
```
//  
// MARK: - Google Sign In/Out Delegate Methods  
  
func signIn(signIn: GIDSignIn!, didSignInFor user: GIDGoogleUser!,WithError error: Error?) { ... }  
  
func signIn(signIn: GIDSignIn!, didDisconnectWith user: GIDGoogleUser!,WithError error: Error!) {  
    // Perform any operations when the 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 GoogleSignIn

class ViewController: UIViewController, GIDSignInUIDelegate {

    /**
     @IBOutlet weak var signInButton: GIDSignInButton!

     /**
     @IBAction func tapSignOut(_ sender: UIButton) {
        print("Sign out")
        let firebaseAuth = FIRAuth.auth()
        do {
            try firebaseAuth?.signOut()
        } catch let signOutError as NSError {
            print ("Error signing out: %@", signOutError)
        }
    }

    /**
    // MARK: -
    //

    override func viewDidLoad() {
        super.viewDidLoad()

        // Login automatically or call dialogue
        GIDSignIn.sharedInstance().uiDelegate = self
        GIDSignIn.sharedInstance().signIn()

        // Setup listener
        FIRAuth.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)
            }
        }
    }

    deinit {
    }
}
```

SEGUE IF
AUTENTICATED
USER

FIREBASE AUTHENTICATION

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Login automatically or call dialogue
    GIDSignIn.sharedInstance().uiDelegate = self
    GIDSignIn.sharedInstance().signIn()

    // Setup listener
    FIRAuth.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

```
import UIKit
import Firebase
import GoogleSignIn

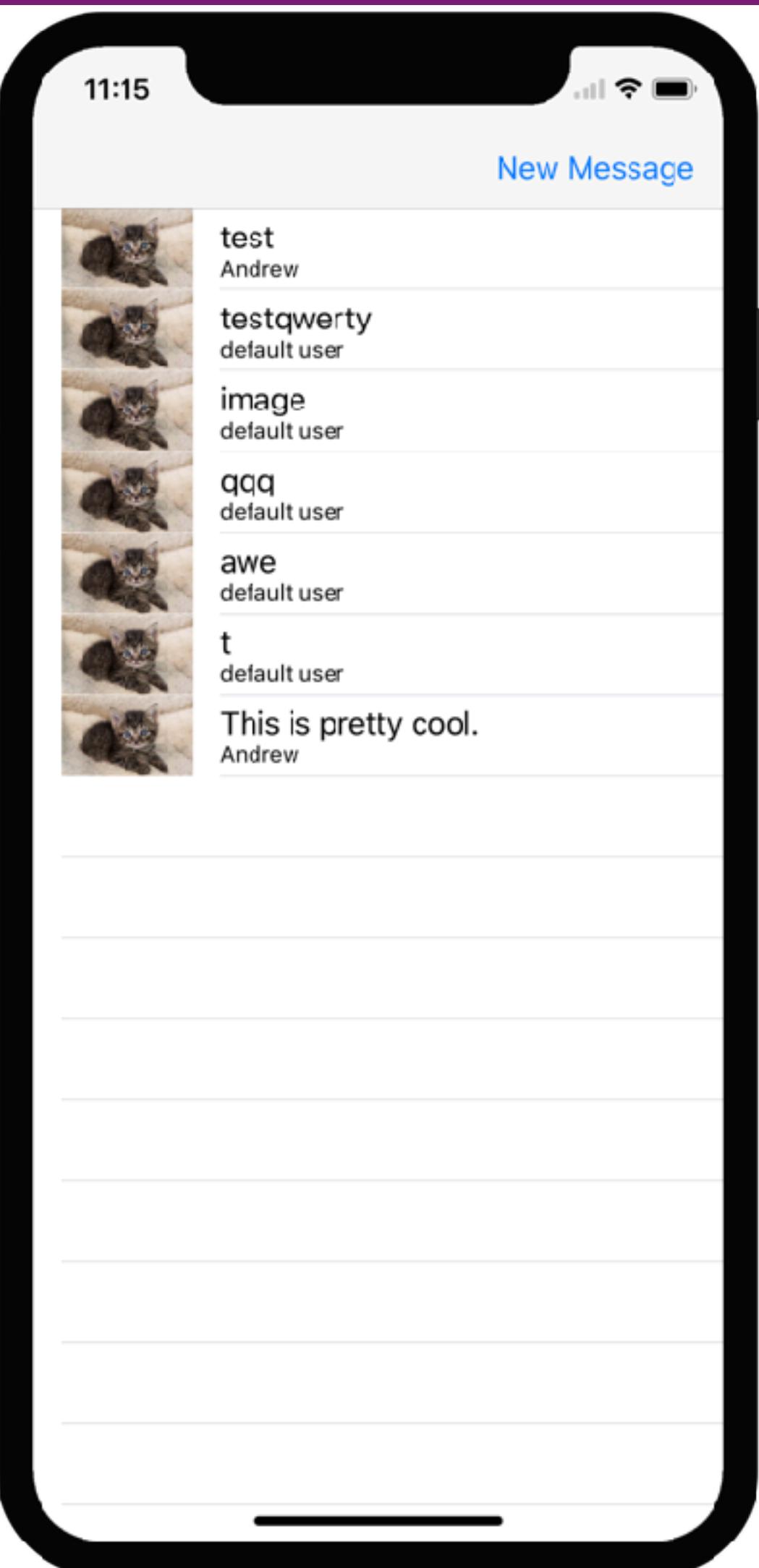
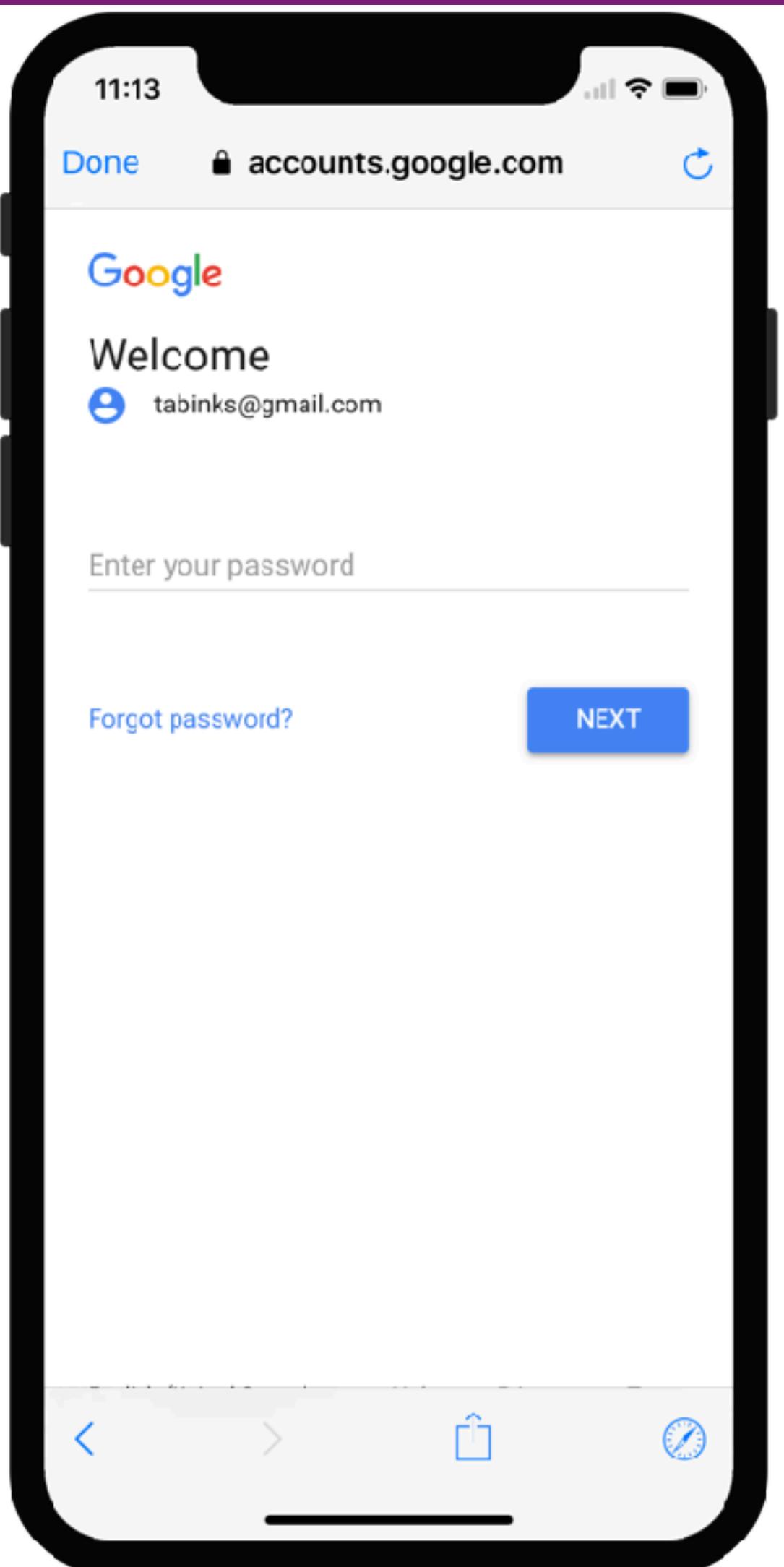
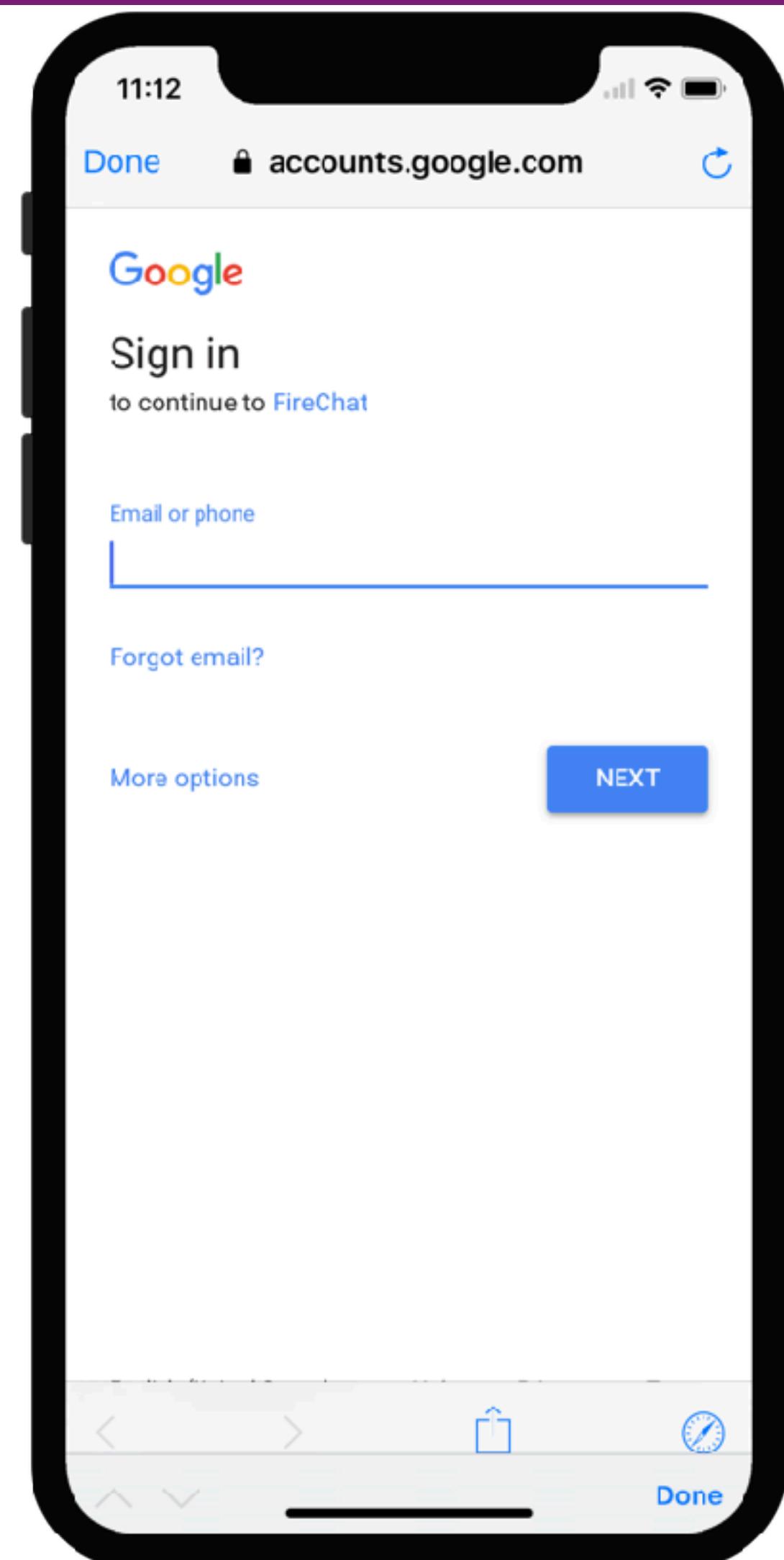
class ViewController: UIViewController, GIDSignInUIDelegate {

    /**
     @IBOutlet weak var signInButton: GIDSignInButton!

    /**
     @IBAction func tapSignOut(_ sender: UIButton) {
        print("Sign out")
        let firebaseAuth = FIRAuth.auth()
        do {
            try firebaseAuth?.signOut()
        } catch let signOutError as NSError {
            print ("Error signing out: %@", signOutError)
        }
    }

    /**
     MARK:
```

FIREBASE AUTHENTICATION



FIREBASE AUTHENTICATION

Screenshot of the Firebase Authentication console.

The left sidebar shows the following navigation:

- Overview
- Analytics
- Authentication (selected)
- Database
- Storage
- Hosting
- Functions
- Test Lab
- Crash Reporting

The main content area is titled "Authentication". It has three tabs: "USERS" (selected), "SIGN-IN METHOD", and "EMAIL TEMPLATES".

The "USERS" tab displays a table of users:

Email	Providers	Created	Signed In	User UID ↑
tabinks@gmail.com	G	Apr 19, 2...	Apr 19, 2...	TviH06LV2HQ5HbCnXTEv...

Below the table are buttons for "ADD USER" and a clipboard icon. At the bottom, there are pagination controls: "Rows per page: 50", "1-1 of 1", and navigation arrows.

FIREBASE AUTHENTICATION

firechat-66f87

messages

1

2

3

4

-Ki4SXKxO7AnysU9doF8

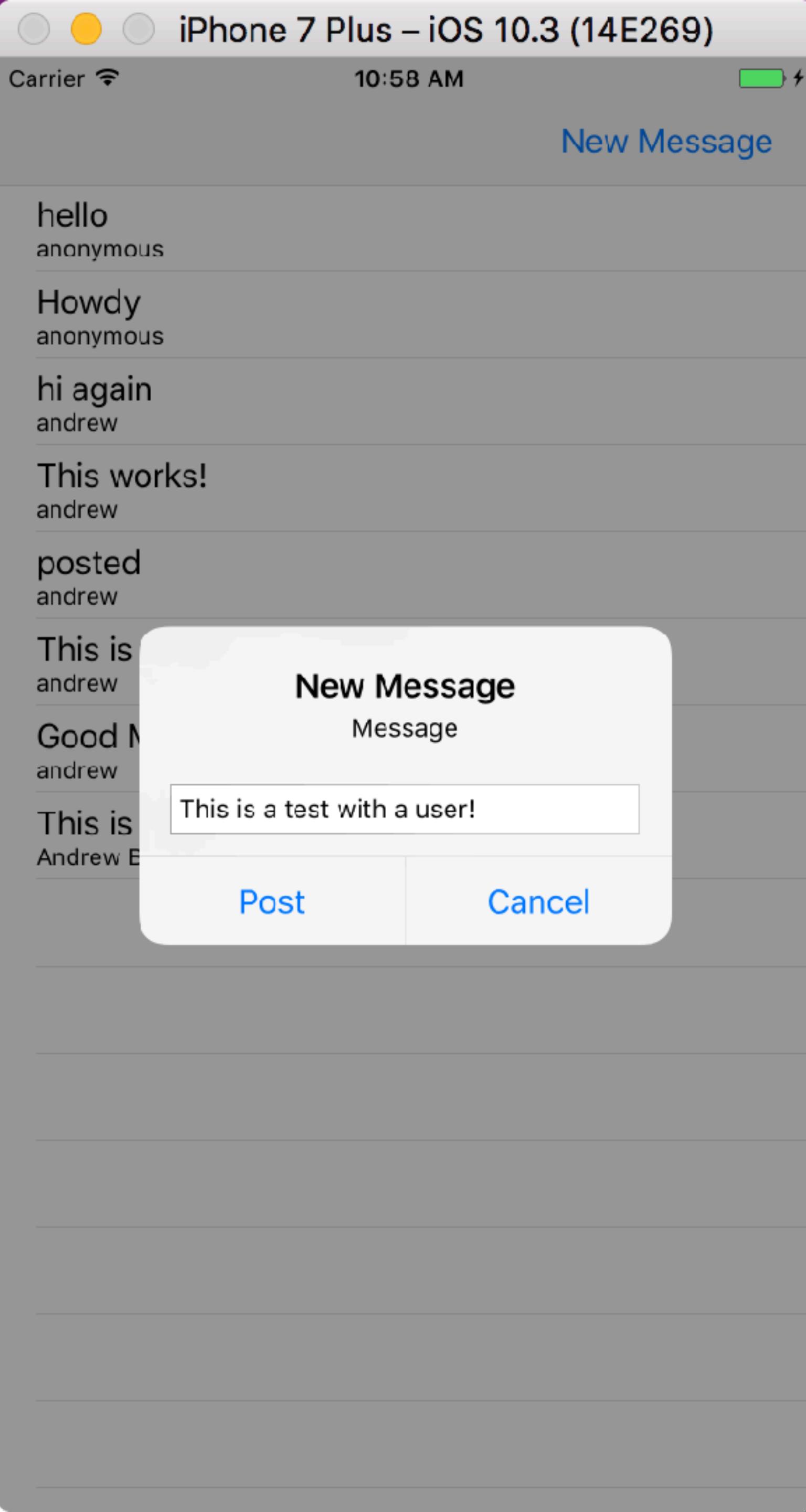
-Ki4Sgutqv5wPpXu94N4

-Ki5PEI6V57IwidD-a_N

-Ki5vIIWFwAHsVOWydzz

name: "Andrew Binkowski"

text: "This is a test with a user!"



FIREBASE ANALYTICS

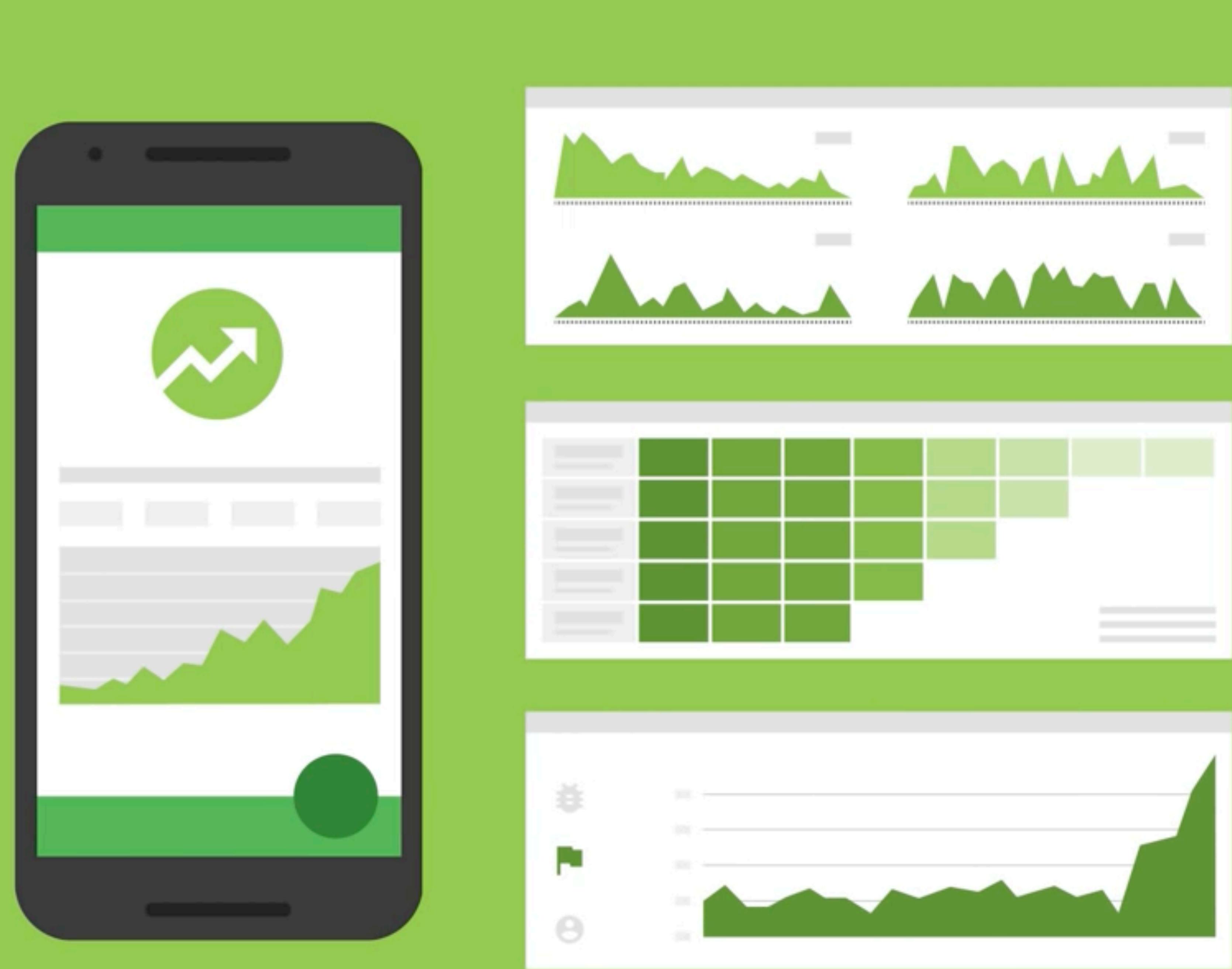
FIREBASE ANALYTICS

- [https://youtu.be/
iT6EalwtonY?
list=PLI-
K7zZEsvYLmOF_07
layrTntevxtbUxDL](https://youtu.be/iT6EalwtonY?list=PLI-K7zZEsvYLmOF_07layrTntevxtbUxDL)



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 want 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	<p>the first time a user launches an app after installing or re-installing it.</p> <p>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.</p>
in_app_purchase	<p>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.</p> <p>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.</p>
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

- Predefined events

```
FIRAnalytics.logEvent(withName: kFIREventLogin, parameters: nil)
```

- Custom events

```
FIRAnalytics.logEvent(withName: "message-cancelled", parameters: nil)
```

FIREBASE ANALYTICS

```
@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]
        self.reference.childByAutoId().setValue(data)
    }

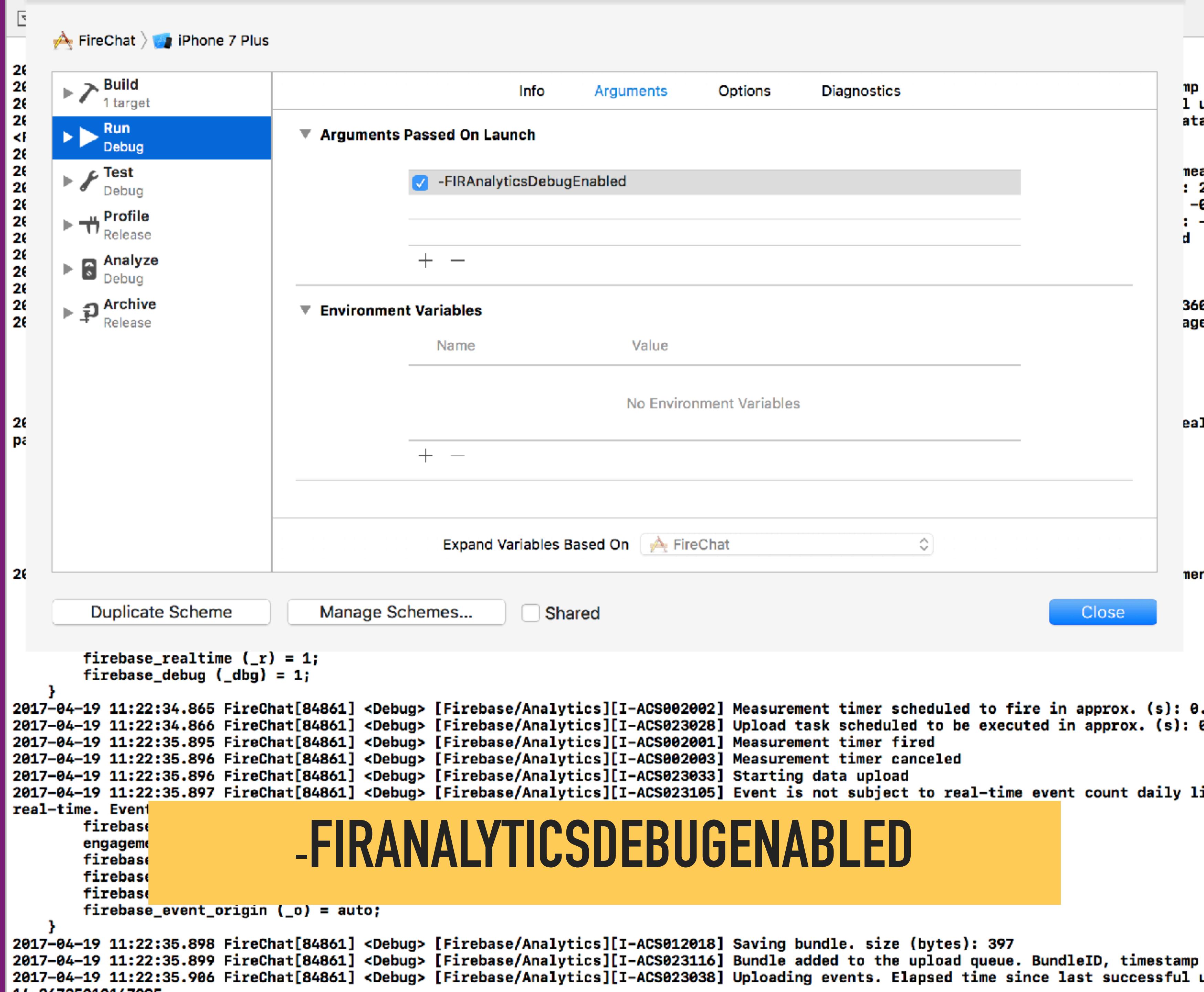
    let cancelAction = UIAlertAction(title: "Cancel",
                                   style: .default)
    _ = UIAlertAction(title: "Cancel", style: .default) { (_) in
        FIRAnalytics.logEvent(withName: "message-cancelled", parameters: nil)
    }

    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
```

IT'S A LITTLE CREEPY

FIREBASE ANALYTICS

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

Firebase FireChat Go to docs : 

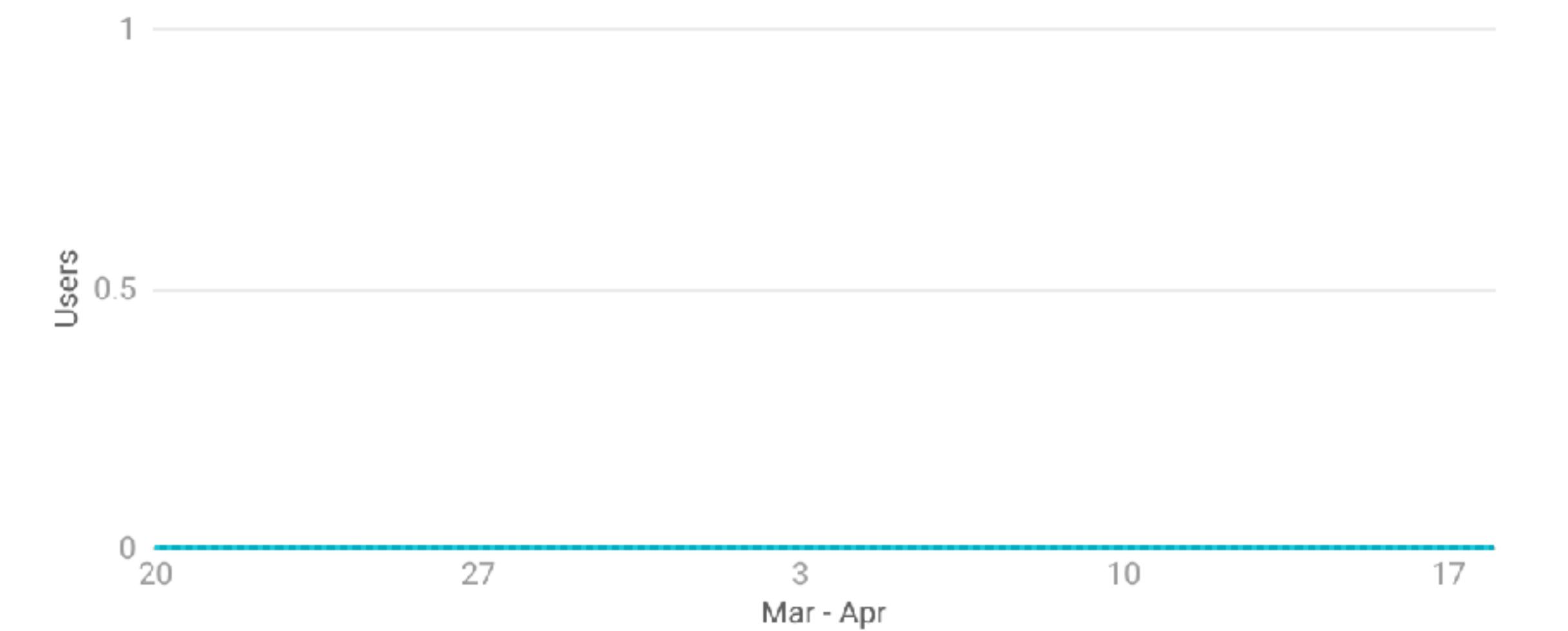
Analytics iOS mobi.uchicago.firechat

DASHBOARD EVENTS AUDIENCES ATTRIBUTION FUNNELS COHORTS STREAM ?

Add Filter +

Last 30 days Compared to Feb 18, 2017 - Mar 19, 2017

Active users



Period	Value
Monthly	0
Weekly	0
Daily	0

Mar - Apr

Average revenue

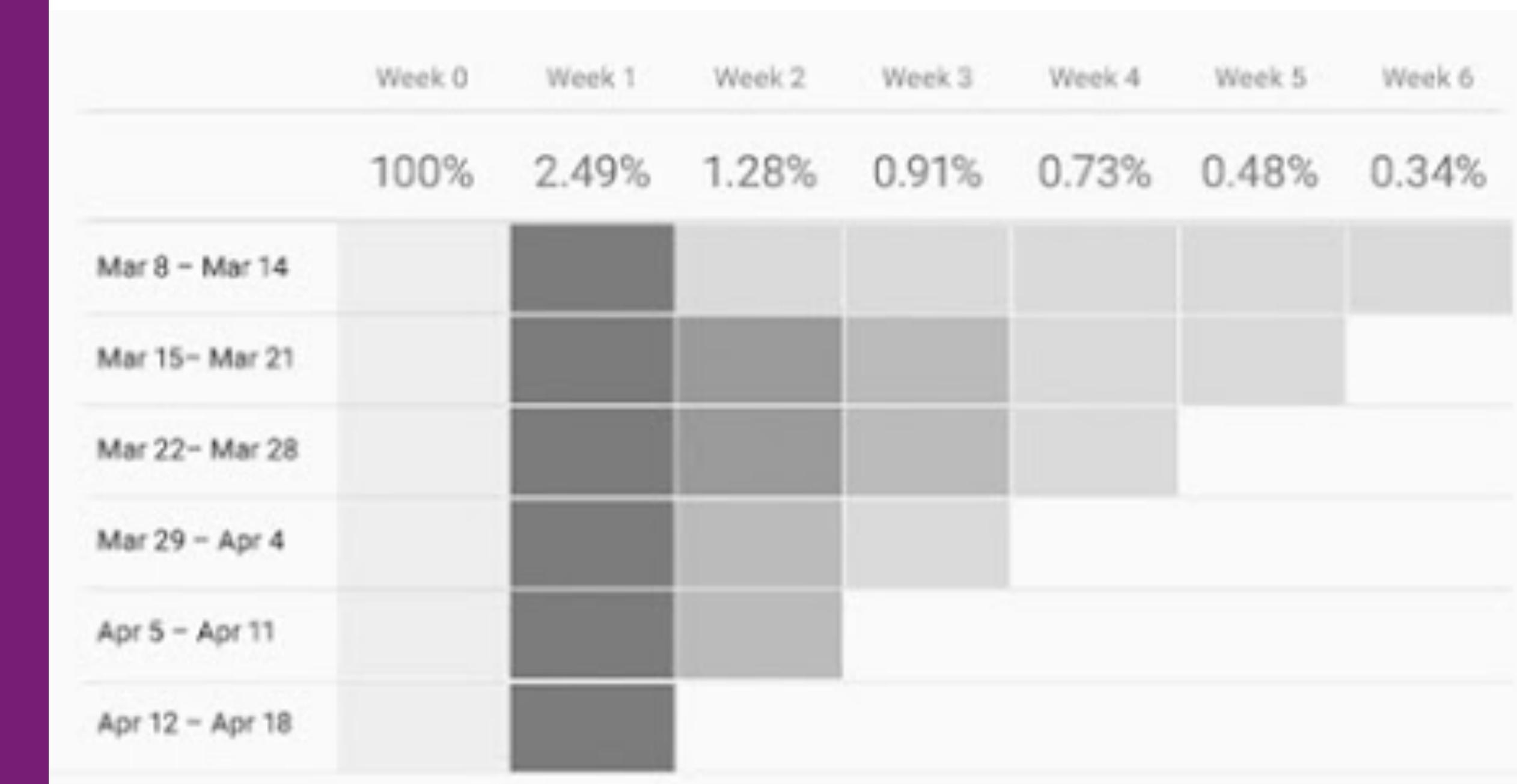
ARPU	ARPPU
Monthly	Monthly
\$0.00	\$0.00

first_open attribution

Source	First Open	Lifetime Value (LTV)
120 days ending Apr 18		

FIREBASE ANALYTICS

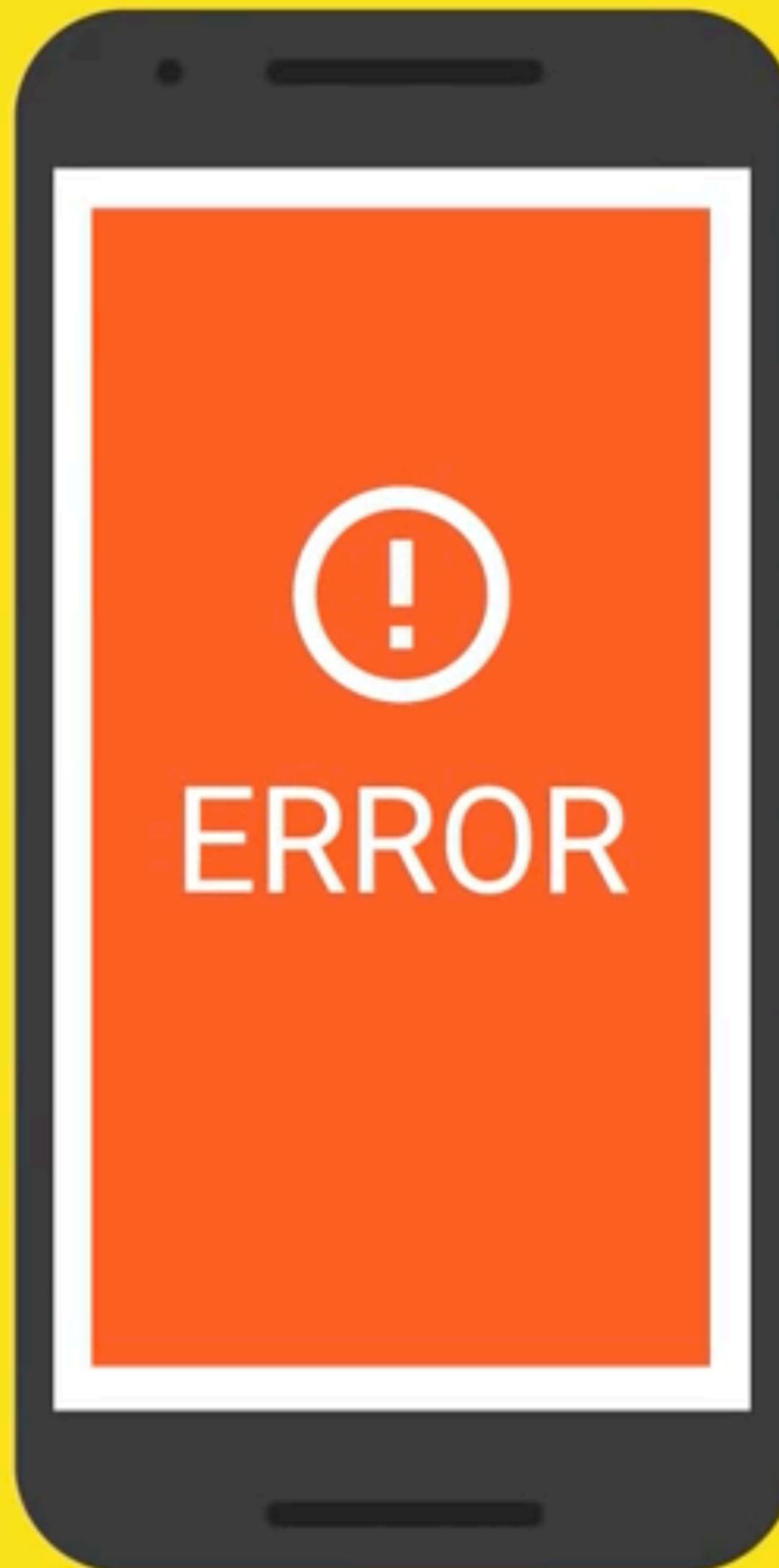
- Active users
- Average revenue, IAP
- first_open, first_open attribution
- Retention cohort
- User engagement (daily, per user, sessions, avg. session duration)
- Device and App info
- Location
- Demographics
- ...



FIREBASE CRASH REPORTING

FIREBASE CRASH REPORTING

- [#](https://www.youtube.com/watch?list=PLIK7zZEsvYLmOF_07layrTntevxtbUxDL&v=B7mlLVAkcfU)



FIREBASE CRASH REPORTING

- Part of Firebase Core
- Free

Monitor fatal and non-fatal errors

Monitor fatal errors in iOS and fatal and non-fatal errors in Android. Reports are triaged by the severity of impact on users.

Collect the data you need to diagnose problems

Each report contains a full stack trace as well as device characteristics, performance data, and user circumstances when the error took place. Similar reports are automatically grouped into issues to make it easier to identify related bugs.

Email alerts

Enable email alerts to receive frequent updates when new crashes are uncovered or regressions are detected.

Integrate with Analytics

Errors captured are set as **app_exception** events in Analytics, allowing you to filter audiences based on who sees errors.

In addition to grouping errors into similar stack traces, Crash Reporting also integrates with Analytics to provide you with the list of events that preceded a crash. This information helps to simplify your debugging process.

FIREBASE CRASH REPORTING

```
platform :ios, '9.0'

target 'FireChat' do
  # Comment the next line if you're not using Swift and don't want to use dynamic frameworks
  use_frameworks!

  # Pods for FireChat
  pod 'Firebase/Core'
  pod 'Firebase/Database'

  pod 'Firebase/Auth'
  pod 'GoogleSignIn'

  pod 'Firebase/Crash'

end
```

FIREBASE CRASH REPORTING

```
@IBAction func didPressCrash(_ sender: AnyObject) {  
    FIRCrashMessage("Forced Crash")  
    fatalError()  
}
```

- Add additional information to crash messages
- Should be anonymous

FIREBASE CRASH REPORTING

- Logs crash data and crash-free data

Firebase FireChat ▾ Go to docs ⋮ ?

Crash Reporting iOS mobi.uchicago.firechat ▾



No sign of any bugs
If you haven't already, please install the SDK
[Learn more](#)

INSTALL SDK

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>

Firebase Tutorial: Getting Started



Attila Hegedüs on September 19, 2016

 Tweet

 Like

Update note: This tutorial has been updated for iOS 10 and Swift 3 by Attila Hegedüs. The original tutorial was written by David East.

Firebase is a mobile-backend-as-a-service that provides several features for building powerful mobile apps. Firebase has three core services: a realtime database, user authentication and hosting. With the [Firebase iOS SDK](#), you can use these services to build powerful apps without writing a single line of server code.

The realtime database is one of the most unique features of [Firebase](#).

Ever used pull-to-refresh to fetch new data? You can forget about it with Firebase.

When a Firebase database updates, all connected users receive updates in realtime. This means your app can stay up to date without user interaction.

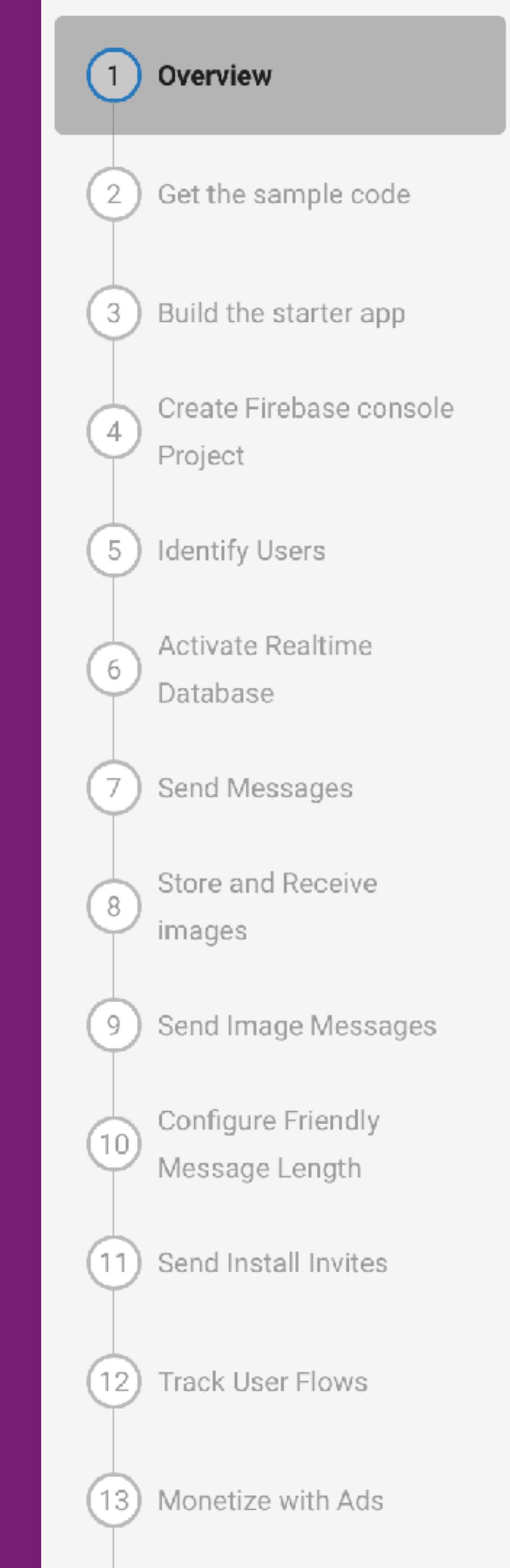
In this Firebase tutorial, you'll learn all the fundamentals of Firebase by making a collaborative grocery list app called [Grocr](#). When items get added to the list they'll appear instantly on any user's devices, you're not going to stop there. You'll tweak [Grocr](#) to work offline, so the list stays in sync even with a spotty grocery store data connection.



Learn the fundamentals in the Firebase tutorial.

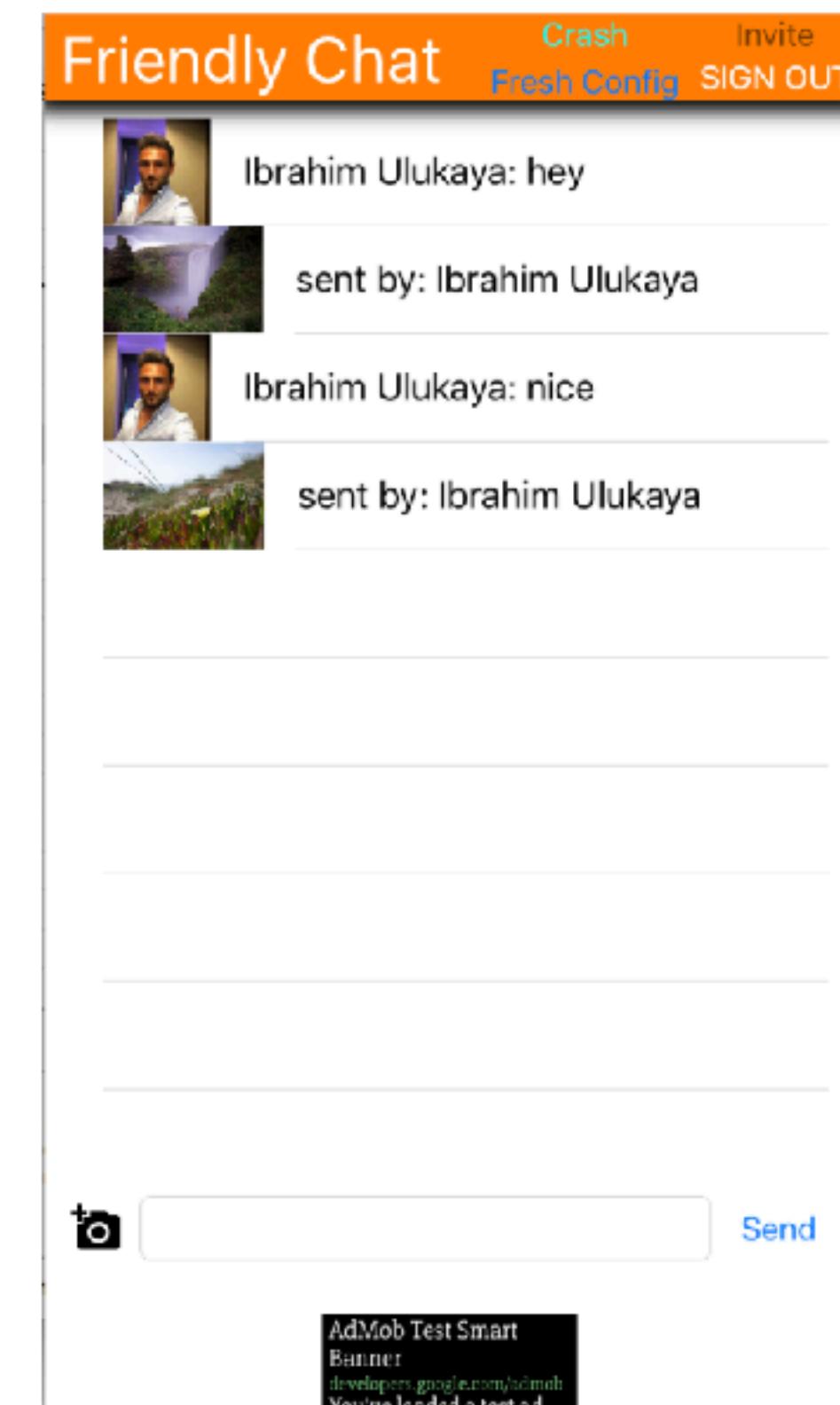
ASSIGNMENT 3 - PART 1

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



← Firebase iOS Codelab Swift

1. Overview



Welcome to the Friendly Chat codelab. In this codelab, you'll learn how to build a real-world mobile applications. You will implement a chat client and monitor its performance.

[This codelab is also available in Objective-C.](#)

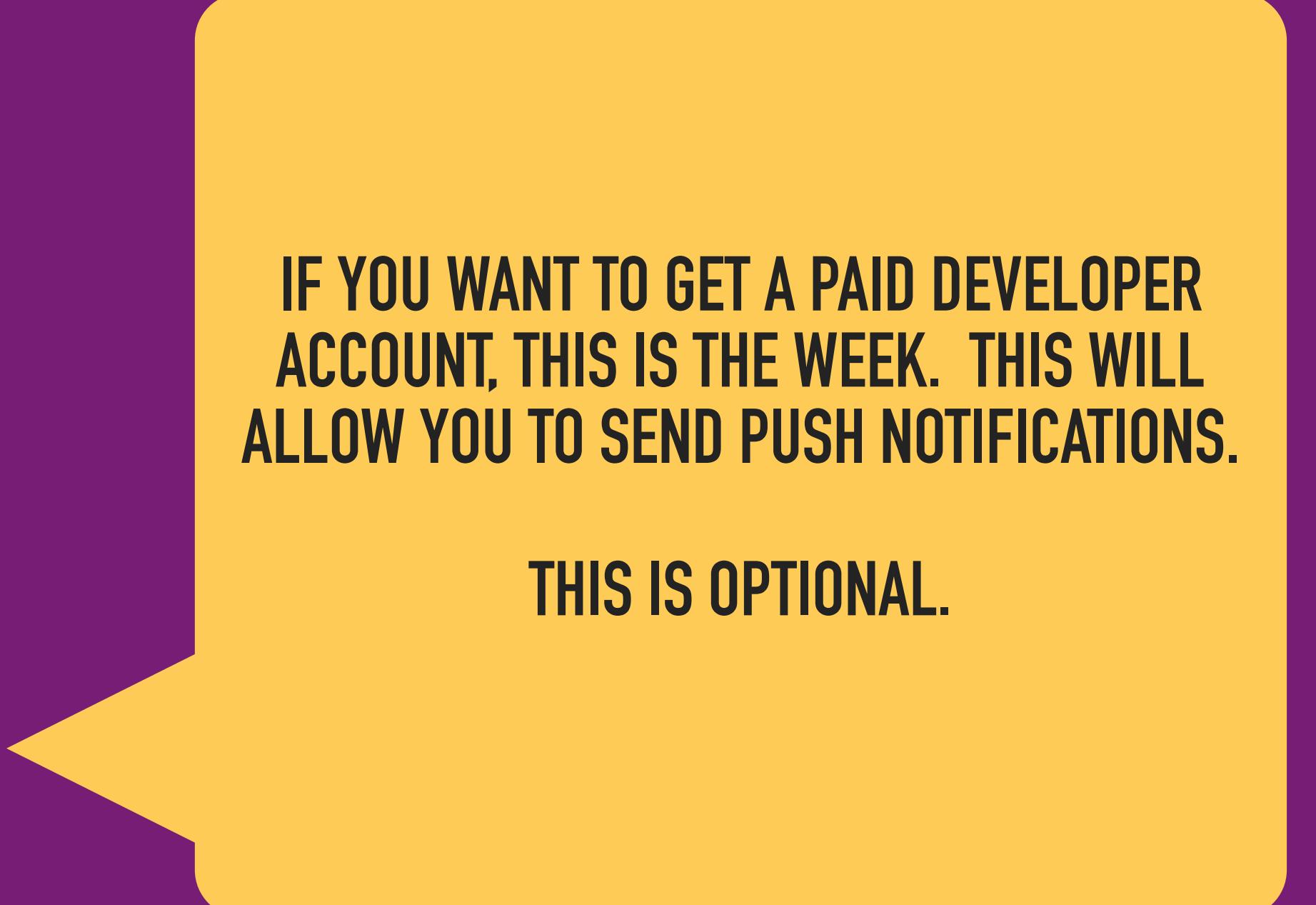
What you'll learn

- ✓ Sync data using the Firebase Realtime Database.

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 2017 • SESSION 4

BACKENDS FOR MOBILE APPLICATIONS