Kevin Chen

3/11/17

Firebase Notes

Set up the development environment

* In Android Studio SDK manager, install some recent SDKs in the SDK Platforms tab, and install Google Play services in the tools tab.
* After creating a google account, go to console.firebase.google.com

**Overview of Firebase**

What is firebase?

* Encompasses many developer products that provide a platform for developing, marketing, and monetizing mobile applications.
* 3 separate categories of products: develop, grow, and earn. Firebase Analytics spans across all of these categories.
* Supports many important common tasks
  + Measure app usage and common demographic information
  + Signup and login via several identity providers (e.g. Facebook, Google)
  + Detect and solve app crashes
  + Customize app behavior for specific user audiences
  + Communicate with users via notification campaigns
  + Easily share content with other people who may not even have the app
  + Earn money via app advertising

Firebase features: develop

* Helps you build your application by saving you the time to have to build certain features
* Firebase Cloud Messaging provides a way to deliver notifications and to send and receive messages to and from users.
* Firebase Authentication: easy way to let users sign up and log into the app.
* Realtime Database: stores app’s data in the Cloud and synchronizes with client apps. Works even if the app doesn’t have network connectivity.
* Storage: storing content with features (e.g. retry uploads/downloads, security levels)
* Hosting in the form of CDN (Content Delivery Network).
* Remote Config: deliver custom experiences to users w/o requiring downloading or installing an update.
* Test Lab: find problems before they make it out to the users (e.g. running tests)
* Crash Report: measure the stability of your app.

Firebase features: Grow and earn

* Notifications: schedule and send messages to certain users. Specify certain groups of users by targeting specific audiences you’ve created with Firebase analytics.
* App Indexing: surface your app’s content right when the user is most ready to need it.
* Dynamic links: users can share content with other people (can participate even if they don’t have the app installed).
* Invites: users can invite other people to use your app by sharing content.
* AdWords: get insight into how your advertising investments are driving installations and in-app actions.
* AdMob: integrate advertising into your app.

**Using Firebase Analytics**

Overview of the Sample Project

* In this course, we will be using Google analytics, remote config, and authentication.

Create the Firebase Project

* Go to console.firebase.google.com and click on “Create a New Project”. Click on “Create a New Project”. Then click on “Add Firebase to Your Android App”
* Enter in the package name (found in AndroidManifest.xml and then generate a “Debug signing certificate SHA-1” by clicking on the help link.
* This will then prompt you to download a file named googleservices.json. Save this file into the app directory of the project.
* Now you need to modify the project’s build.gradle file to include the Services plugin. Add the following line of code inside the dependencies of the project’s dependencies section: classpath 'com.google.gms:google-services:3.0.0'.
* Add the following line of code at the bottom of the app module’s build.gradle file: apply plugin: 'com.google.gms.google-services'
* Add references to the app module’s build.gradle file inside the dependencies section:

compile 'com.google.firebase:firebase-core:9.0.2'

compile 'com.google.firebase:firebase-analytics:9.0.2'

compile 'com.google.firebase:firebase-config:9.0.2'

compile 'com.google.firebase:firebase-auth:9.0.2'

* Rebuild the project.

Intro to Firebase Analytics

* Provides free and unlimited support for up to 500 distinct events within the app, each of which can have 25 properties
* Provides a variety of predefined event types that are common
* Seamlessly integrates with other Firebase features automatically.
* Records common events and demographic data automatically
* Key API functions:
  + Retrieve an instance of Firebase Analytics: FirebaseAnalytics.getInstance()
  + Log events: logEvent(String evtStr, Bundle params). evtStr is the string used to specify the event name. params are a Bundle. Add params inside the Bundle by calling the instance method putInt()/putString() (depending on the type of the value), passing in a key and value. Some events have predefined parameter values, but you are free to make up your own events.

Add the Firebase Analytics API

* (If you get rendering problems in the Design view of an xml file, then try a lower version of the API by selecting the menu with the Android icon.)
* Create a private field of type FirebaseAnalytics in the activity to hold the instance of Firebase Analytics.
* In the onCreate function, retrieve the instance of Firebase analytics by requesting the instance from the package: Do this by calling the method FirebaseAnalytics.getInstance(), passing in the context (this), and assigning it to the FirebaseAnalytics object created in the previous step.
* Set the minimum session duration by calling the instance method setMinimumSessionDuration of the FirebaseAnalytics object, passing in the minimum time session in milliseconds (5000 is good). This will tell the analytics package to wait 5 seconds before counting this as a session in case the user accidentally launched the app.
* (Get the ID of a view by calling the instance method getID on the view.)
* Log an event by calling the instance function logEvent on the FirebaseAnalytics object. For example, if you want to log a certain button being clicked, pass in a button name and a Bundle with a key-value pair of “ButtonID” and the ID of the view. (Use the instance method, putInt(), of Bundle to put the key-value pair, passing in the key and value as parameters inside the function.)
* Firebase Analytics comes with some predefined Param types, which can be found in FirebaseAnalytics.Param class. Use this as the first parameter of putInt()/putString()/etc. for the Bundle class.
* Firebase Analytics comes with some predefined event types, which can be found in FirebaseAnalytics.Event class. Use this as the first parameter of the logEvent method.

Exercise the App (Run the App)

* Seeing the analytics data using Android logcat:
  + When testing the app, enable verbose logging on the app so you can see the analytics events as they’re being captured. This needs to be done because back end analytics service doesn’t display the events that you are recording right away.
  + To see the analytics events as they are being captured, send some commands to the debugger by opening the terminal (found in the bottom of Android Studio). Make sure the emulator is open when you do this.
  + These commands are some debugging properties for the ADB:

adb shell setprop log.tag.FA VERBOSE

adb shell setprop log.tag.FA-SVC VERBOSE

adb logcat –v time –s FA FA-SVC

* + (If the adb command is not found, add the Android SDK tools folder to your PATH.)
  + Open up the Android monitor, filter for the string “/FA”. Then select the “Debug app” icon.
* Seeing the data in the console online:
  + Open up the console, select the project, and select the Analytics page.
  + Active users panel, average revenue panel, retention data for sets of cohorts (see how well the app is doing in retaining users that all started using it within a certain period of time), user engagement data, # of users using each version of the app, OS/Models where the app is running on, location of where the app is being run, demographics data.
  + Much of the data mentioned above is captured automatically.
  + The Events tab shows the events that the code is deliberately recording (in addition to other events that are automatically being recorded). Can link data to BigQuery to do deeper analysis.
  + Audiences tab demonstrates the different types of audiences using the app. Some audiences are automatically recorded for you.

**Incorporating Remote Config**

Overview of Firebase Remote Config

* Remote Config gives users a customized experience (customized app behavior) without having the user download and install an update
  + Define-in app default values
  + Create server-side parameters and conditions
  + Fetch values from server to override in-app defaults
  + Use with Firebase Analytics to customize for specific audiences

Understand Parameters and Conditions

* Define the default values, then create the corresponding parameters on the sever along with any conditions we want to apply to those servers, and then implement the remote config logic to retrieve parameters from the server and update them within our app.
* Defining default values
  + Can be done by setting the values programmatically using Java Map object.
  + Another way is to define an xml resource that contains the default values. Create a defaultsMap tag. Within it, put entry tags. Within each entry tag, put a key tag with the key inside the tags and do the same with a value tag.
* Creating corresponding parameters on the server
  + In the Firebase console, select the Remote Config tab.
  + Then click “Add Parameter” and add the parameters for each parameter.
  + Make sure you use the same names as you did in the Android app.

Implement Firebase Remote Config

* Add the FirebaseRemoteConfig member variable inside the activity class.
* Then get the Remote Config instance using the static method FirebaseRemoteConfig.getInstance(), and assign it back to the FirebaseRemoteConfig reference.
* For more rapid testing, enable developer mode by disabling the Remote Config caching limit. Config fetches are normally limited to 5 per hour, but doing this enables many more requests to facilitate testing. Do this by using the code: new FirebaseRemoteConfigSettings configSettings = new FirebaseRemoteConfigSettings.Builder().setDeveloperModeEnabled(BuildConfig.DEBUG).build();
  + Setting the developer method to debug will enable the zero caching.
* Then set the config settings using the setConfigSettings instance method of the FirebaseRemoteConfig object, passing in the config settings object created in the previous step.
* Then retrieve the default settings that were created in the XML resource by calling the instance method setDefaults() of the FirebaseRemoteConfig object, passing in the xml resource (R.xml.*nameoffile*).
* Check whether developer mode is on by using the instance method: *FirebaseRemoteConfigObjectName*.getInfo().getConfigSettings().isDeveloperModeEnabled(). Set the cache duration to zero to always get new values.
* Fetching the values from the Remote Config service: *FirebaseRemoteConfigObjectName*.fetch(*CacheDuration*).addOnCompleteListener(), passing in an anonymous class that implements the OnCompleteListener interface, which has a single method onComplete().
  + The onComplete() function has a parameter of type Task. Call the instance method isSuccessful() of Task to see if you successfully retrieved it.
  + To take the default values from the server and override the ones in the app, use the instance method activateFetched() of the FirebaseRemoteConfig object.
* Use the instance functions the FirebaseRemoteConfig object getBoolean/getString/etc., passing in the key value, to get the updated values.

Exercise the app

* Change a parameter’s value in the Remote config, and publish those changes.
* You can set up parameter values based on various conditions in the console. To create a condition, go to the conditions tab and create a new condition.
  + Before you create your first condition, there will be no condition tab. Instead, click on a parameter, select “Add Value for Condition”, and select “Define New Condition”
  + After creating the conditions, select “Add Value for Condition” to add a value for a given condition.

**Using Firebase Authentication**

Overview of Firebase Authentication

* Easily let users sign up for and sign into your app.
* Supports: Google, Facebook, Twitter, GitHub, Email and password
* Uses OAuth and OpenID Connect for custom authentication solutions
* Steps:
  + Enable the authentication method in the console
  + Get an instance of the FirebaseAuth object and implement an AuthStateListener to detect changes in the authentication state
  + Create a user, validate credentials, and sign them in, and sign them out.
  + Handle authentication errors (e.g. wrong username, wrong password)

Implement Email and Password Authentication

* Key methods
  + Get an instance of the FirebaseAuth library: FirebaseAuth.getInstance()
  + Create AuthStateListener(), which is called when the user is signed in or out.
  + createUserWithEmailAndPassword(username, password) creates the user.
  + signInWithEmailAndPassword(username, password) signs the user in.
  + signOut() signs the user out.
* Go to the console and enable the email and password validation method
  + Click on the Authentication tab in the Firebase project.
  + Choose “Setup Sign-in Method”, click on Email/Password, and click enable.
* Add the following lines of code in your project
  + Create member variable references to a FirebaseAuth and FirebaseAuth.AuthStateListener objects.
  + Get a reference to the FirebaseAuth object by calling FirebaseAuth.getInstance() and then assign the reference to the FirebaseAuth object.
  + Create a new anonymous FirebaseAuth.AuthStateListener object that overrides the onAuthStateChanged() method, which takes in a NonNull FirebaseAuth object. Inside the onAuthStateChanged() method, get a reference to the current user by calling the instance method getCurrentUser() from the FirebaseAuth object, which returns an object of type FirebaseUser. If the user is not signed in, this method will return null.
  + Get the userID by calling the instance method user.getUiD() of the FirebaseUser object.
  + Attach the listener onto the auth object by calling the instance method addAuthStateListener() from the FirebaseAuth class, passing in the FirebaseAuth.AuthStateListener object. You can do this in the onStart() function.
  + Remove the AuthListener by calling the instance method removeAuthStateListener from the FirebaseAuth class, passing in the FirebaseAuth.AuthStateListener object. You can do this in the onStop() function. You should first check if the FirebaseAuth.AuthStateListener object is not null before doing this.
  + Get the email of a user by calling the instance method getEmail() from the FirebaseUser class.
  + To create a new user account, call *FirebaseAuthObjectName*.createUserWithEmailAndPassword(*email*, *password*).addOnCompleteListener(this, *Anonymous OnCompleteListener*). The anonymous OnCompleteListener class needs to override the onComplete function, which takes in a NonNull object of type task. Call the instance method isSuccessful() of Task to see if you successfully created a new account.
  + Signing in works the exact same way except you call signInWithEmailAndPassword instead of createUserWithEmailAndPassword.
  + To sign the user out, call the instance method signOut() from the FirebaseAuth object.

Add Firebase Auth Error Handling

* Add an OnFailureListener (using addOnFailureListner) to the Task that is returned by addOnCompleteListener. This listener is passed a generic exception argument. Check for which type of exception it is by using instanceof.
  + FirebaseAuthInvalidCredentialsException indicates that the password supplied to the sign-in method was incorrect.
  + FirebaseAuthInvalidUserException indicates that the email was correct but that there’s no such user
  + FirebaseAuthUserCollisionException tells us that we tried to create a user that already has that email address.
* Example: mAuth.signInWithEmailAndPassword(…).addOnCompleteListener(…).addOnFailureListner(…);
* If it’s not any of the above exceptions, you can get the localized message of the exception by using the instance method getLocalizedMessage() of the Exception.

View Users in the Firebase Console

* Go to the Auth tab in the Firebase Console. You can see the list of users there.
* Manually add a user by clicking on “Add User”
* Can Reset, disable, or delete accounts manually by clicking on the options icon (three horizontal dots) for the corresponding accounts.
  + A disabled user can’t log in.