Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Android & Firebase

Firebase Integration

Agenda

- Firebase history
- The all new Firebase
- Real-time database
- Firestore
- Authentication
- Storage
- Remote config
- Hosting
- Crash reporting
- Test lab
- Firebase cloud messaging
- Dynamic links
- App indexing
- Analytics
- CoffeeMate Highlights & Demos along the way…

Agenda

- ❑ Firebase history
- ❑ The all new Firebase
- ❑ Real-time database
- ❑ Firestore
- ❑ Authentication
- ❑ Storage
- ❑ Remote config
- ❑ Hosting
- ❑ Crash reporting
- ❑ Test lab
- ❑ Firebase cloud messaging
- ❑ Dynamic links
- ❑ App indexing
- ❑ Analytics
- ❑ CoffeeMate Highlights & Demos along the way…

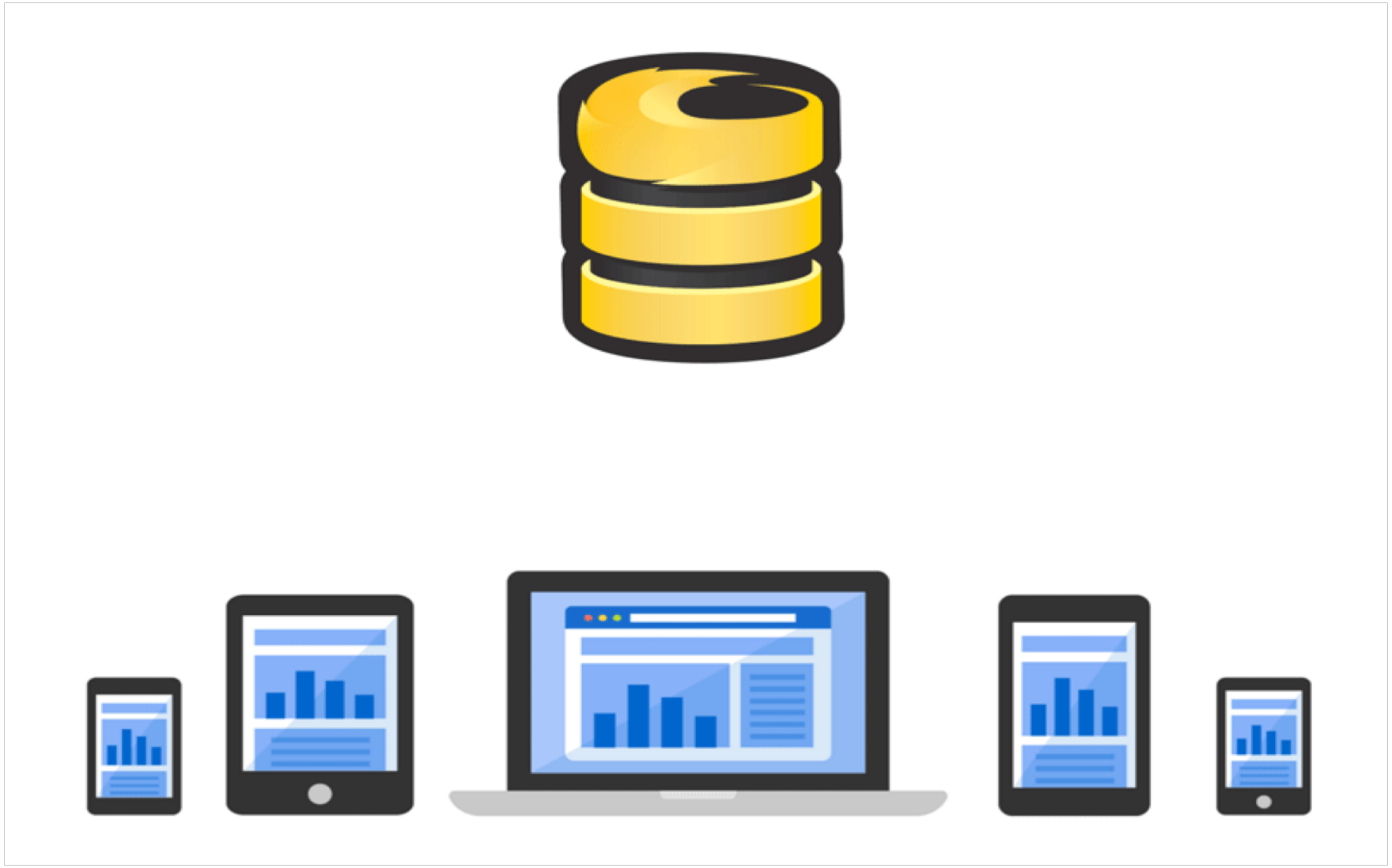
Goodbye RDBMS…

Database

Real Time Database

- Unlike RDBMS, data is stored as JSON. It is no-SQL JSON database.
- What makes it real time it its ability to notify listeners of any change in data.
- Whenever any change is made in the JSON database structure, the firebase SDK notifies the app.
- So you can forget about REST API calls, connectivity checks, 3rd party libraries and polling.

The Core Magic Of Firebase *



Saving Data In Firebase

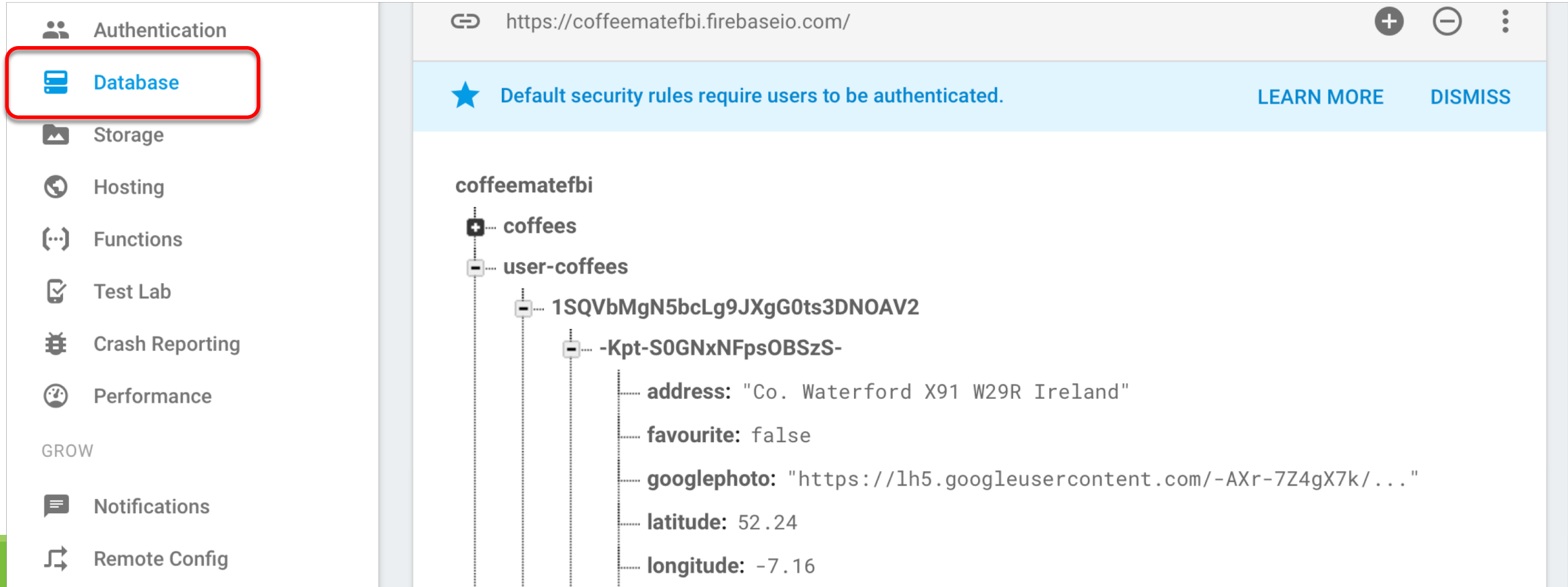
- Get database reference (your base node)
  - DatabaseReference mDBRef = FirebaseDatabase.*getInstance*().getReference();
- Point it to the right JSON node
  - mDBRef = mDBRef.child("mydb").child("table1");
- Set the value at the pointed node
  - mDBRef.child('row1').setValue(myPOJO);
- Or push the value to create a unique key and set the value
  - mDBRef.push().setValue(myPOJO);

Create a Key

Firebase takes care of the Key

View Your Saved Data

- Log on to https://firebase.google.com.
- Go to console and select your project.
- Hit database and select data tab



Retrieving Data

- Data is retrieved via callbacks listeners.
- There are mainly 2 types of listeners
  - ValueEventListener – get entire child structure.
  - ChildEventListener – get only the child that got changed / added or deleted.
- Attach these listeners to Database reference we saw earlier.
- Both of these listeners are called once, so app can get the data and prepare UI

Attaching Data Listeners

- ValueEventListeners can be added in 2 ways:
  - addListenerForSingleValueEvent (valEvListener); //next slide
    - *Get the entire data snapshot only once*
  - addValueEventListener(valEvListener);
    - *Get entire data snapshot whenever there is a change in any of the child nodes*
- ChildEventListener can be added in only one way:
  - addChildEventListener(childEvListener)
    - *Get updates as child nodes*

Retrieving Data As POJO

```
public ChildEventListener childEvListener = new ChildEventListener() {

    public void onChildAdded(DataSnapshot dataSnapshot, String s) {
        MyPOJO m = dataSnapshot.getValue(MyPOJO.class);
    }

    public void onChildChanged(DataSnapshot dataSnapshot, String s) {}

    public void onChildRemoved(DataSnapshot dataSnapshot) {}

    public void onChildMoved(DataSnapshot dataSnapshot, String s) {}

    public void onCancelled(DatabaseError databaseError) {}
};
```

Retrieving Data As POJO

```
public ValueEventListener valEvListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        MyPOJO m = dataSnapshot.getValue(MyPOJO.class);
    }

    @Override
    public void onCancelled(DatabaseError databaseError) {}
};
```

Filtering Data

- Set DatabaseReference to the right parent node.
  - Db = base.child('Institute').child('year');
- To get all students with major = 'Computing'.
  - Set orderByChild('major').
  - Set equalTo('Computing');
  - Add listeners

```
Db.orderByChild('major').equalTo('Computing')
.addChildEventListener(childEvListener);
```

Pagination

- Create a Query object or keep using DatabaseReference. Query is super class of DatabaseReference class.
- Set orderBy some child key (and filter if needed)
- Set limits (startAt, limit etc.)
- Attach data listeners
- And done. Appropriate callback will be called when operation is complete.

Pagination Code

```
// class level
final int limit = 50;
int start = 0;

// event level
Query userListQuery = userDBRef.orderByChild("email")
.limitToFirst(limit).startAt(start);
userListQuery.addValueEventListener(new ValueEventListener() {
    @Override
     public void onDataChange(DataSnapshot d) {
         // Do something
        start += (limit+1);
```

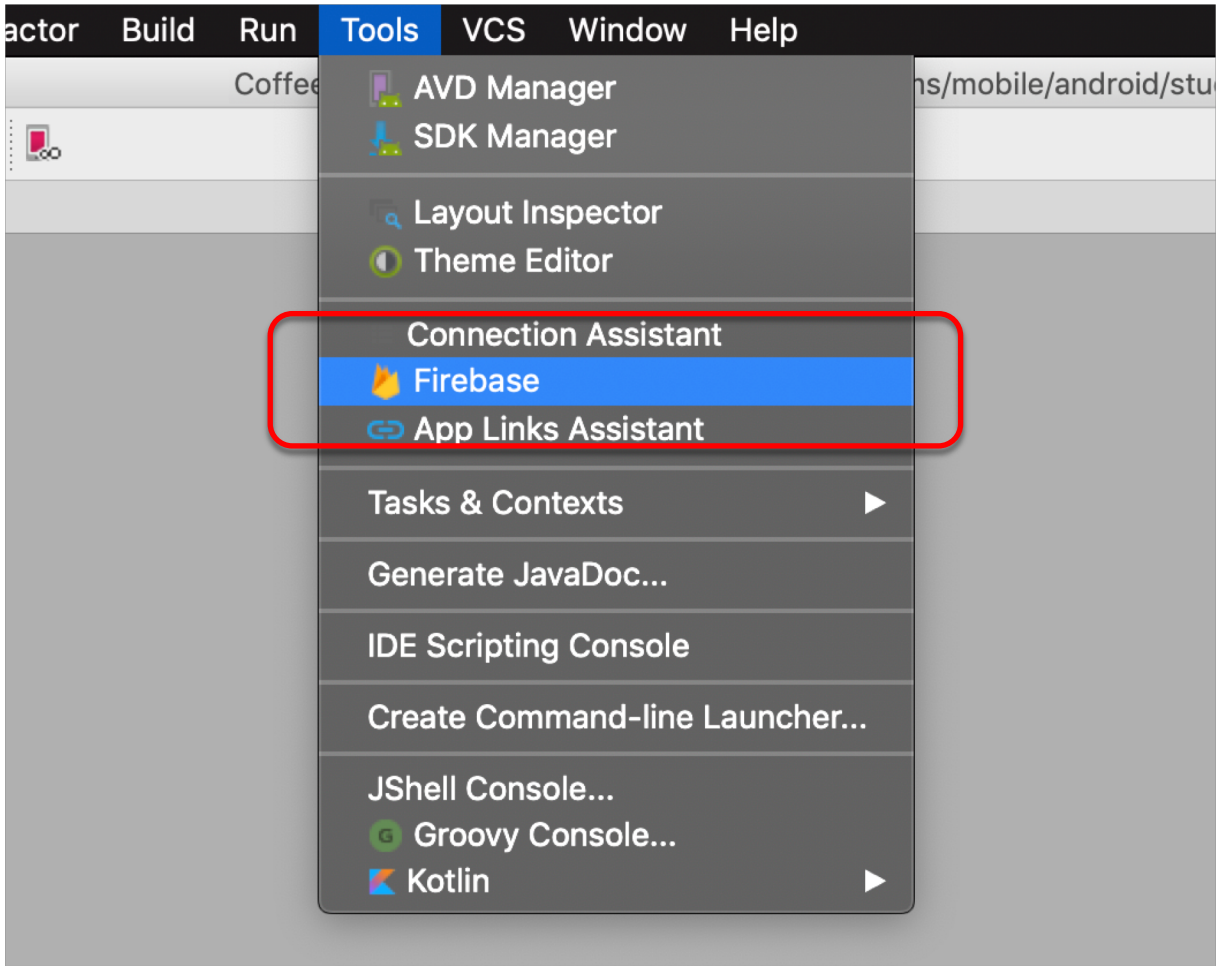
Firebase Realtime Database

Quick DEMO…

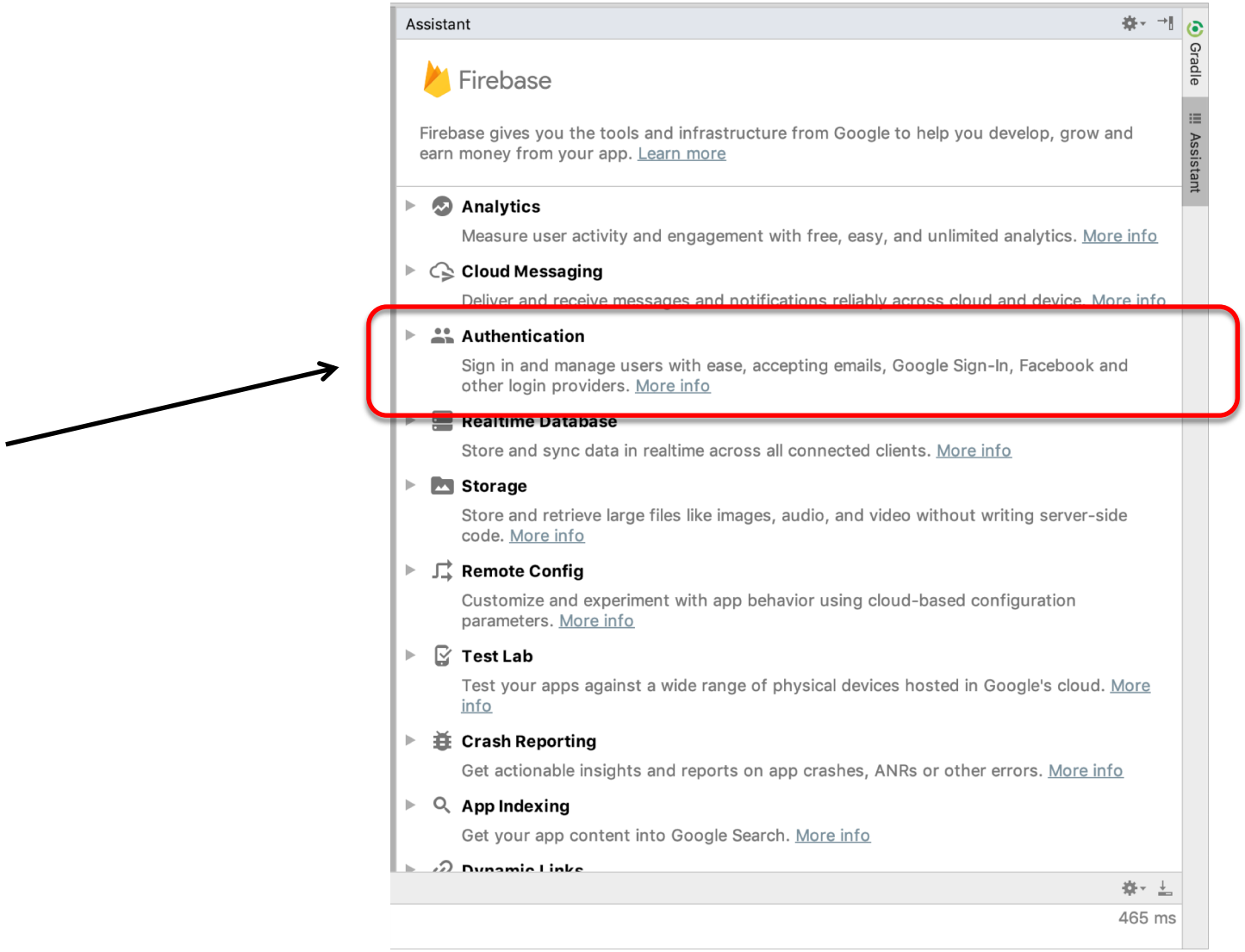
CoffeeMate.9.0

Setup
&
Code
Highlights

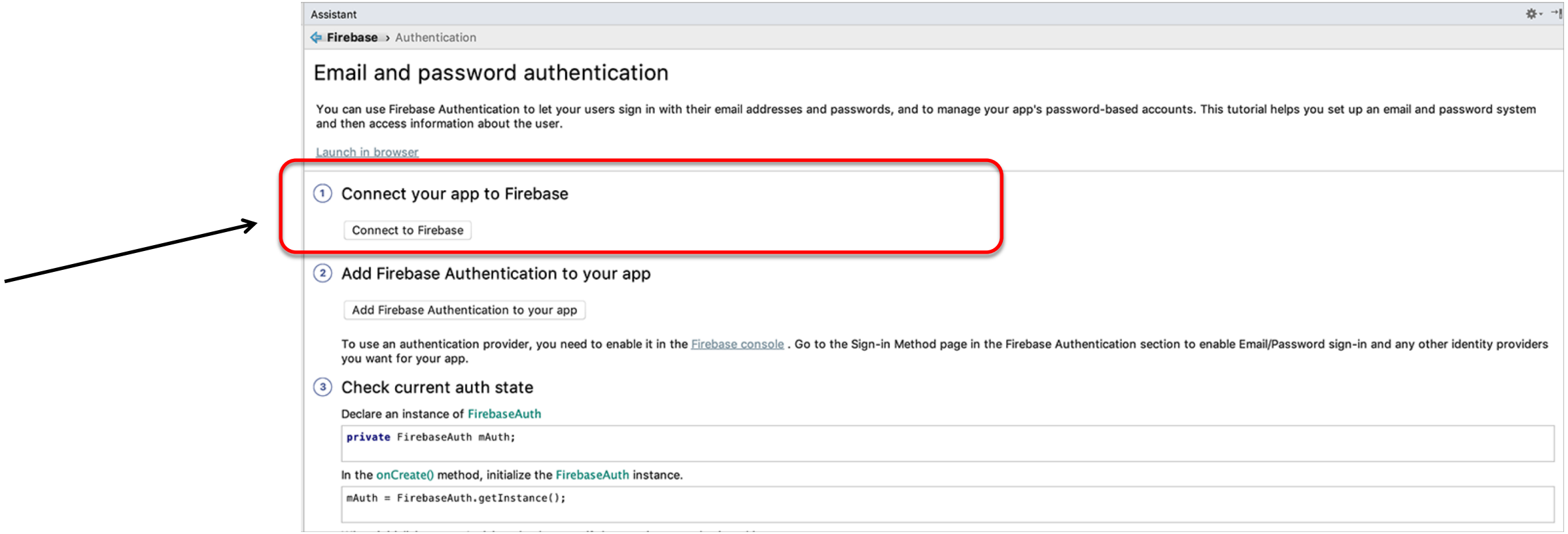
1. Connect your App to Firebase *



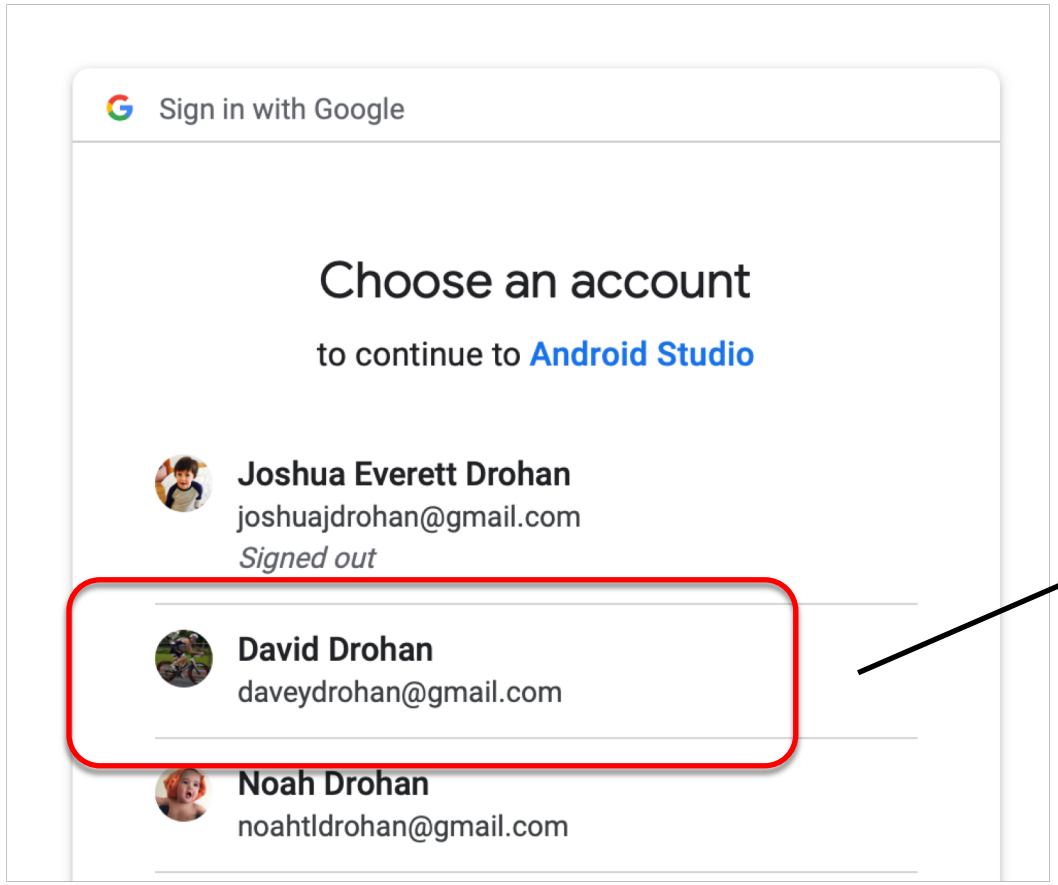
1. Connect your App to Firebase *

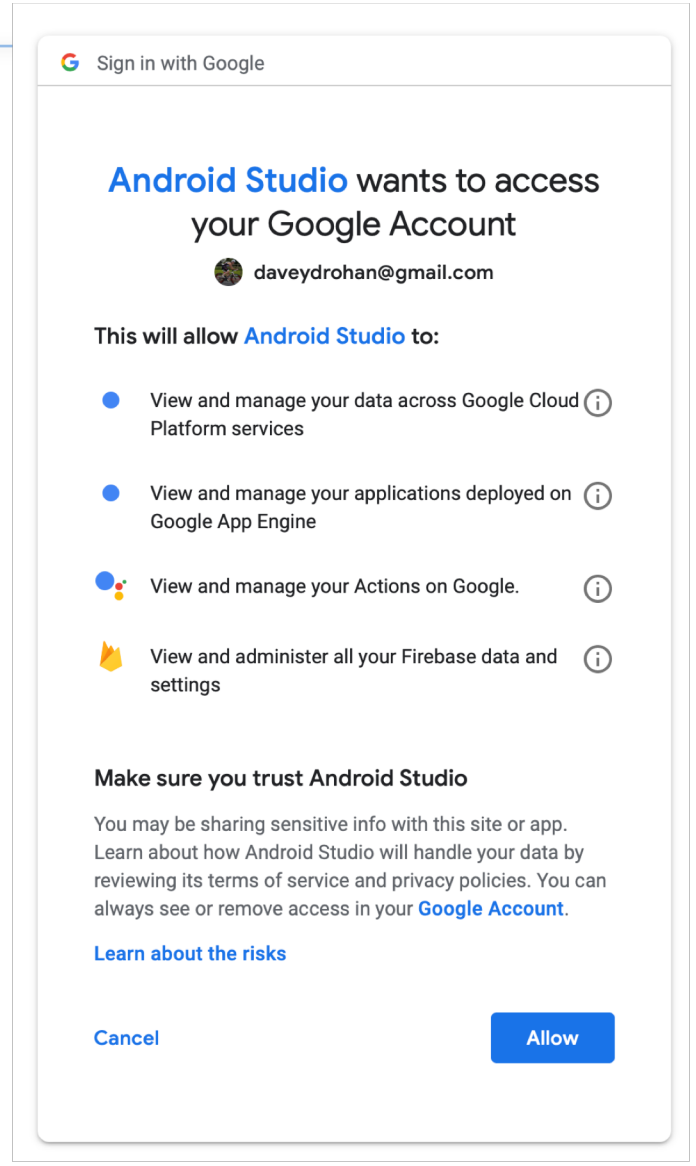


1. Connect your App to Firebase *

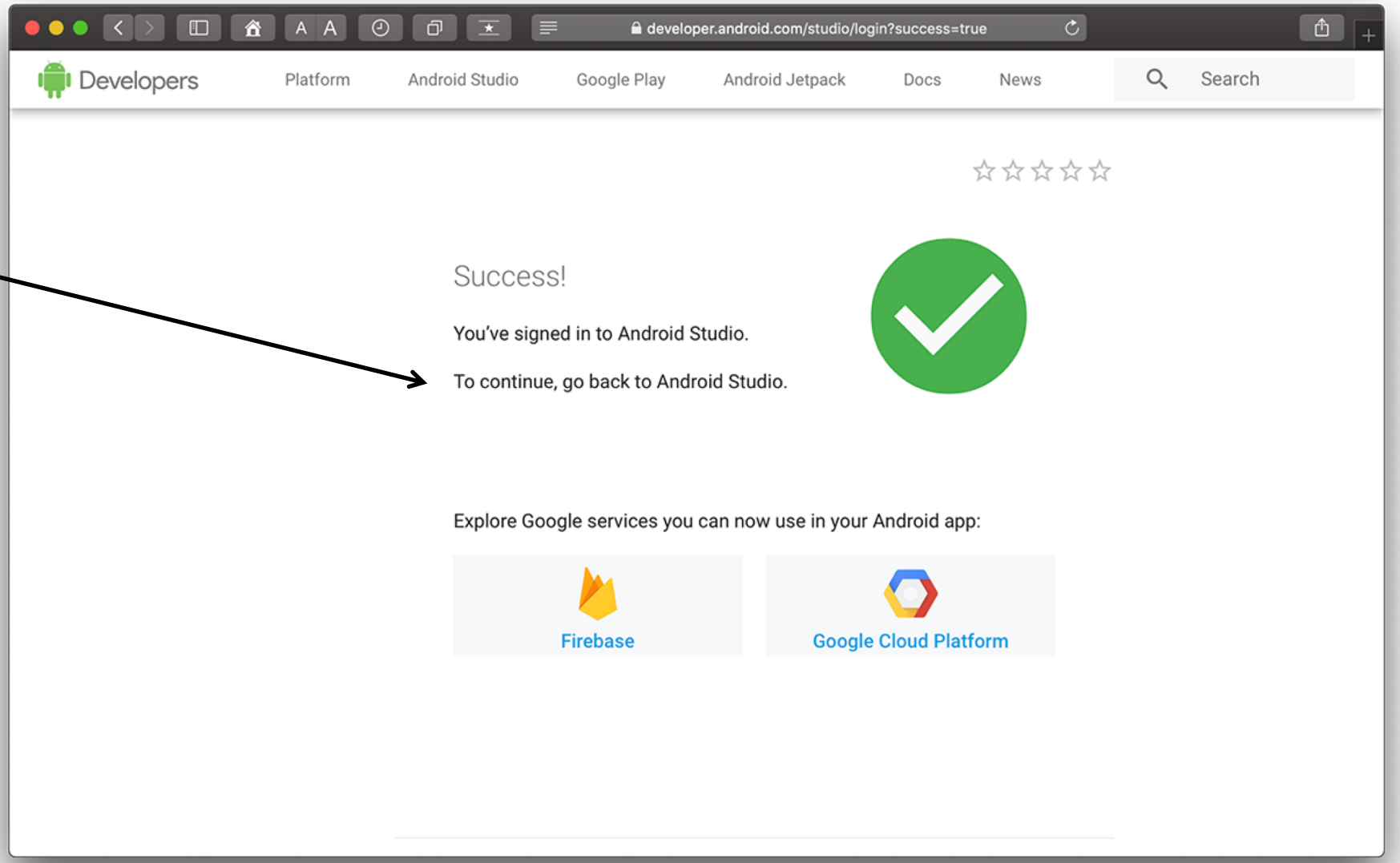


1. Connect your App to Firebase *

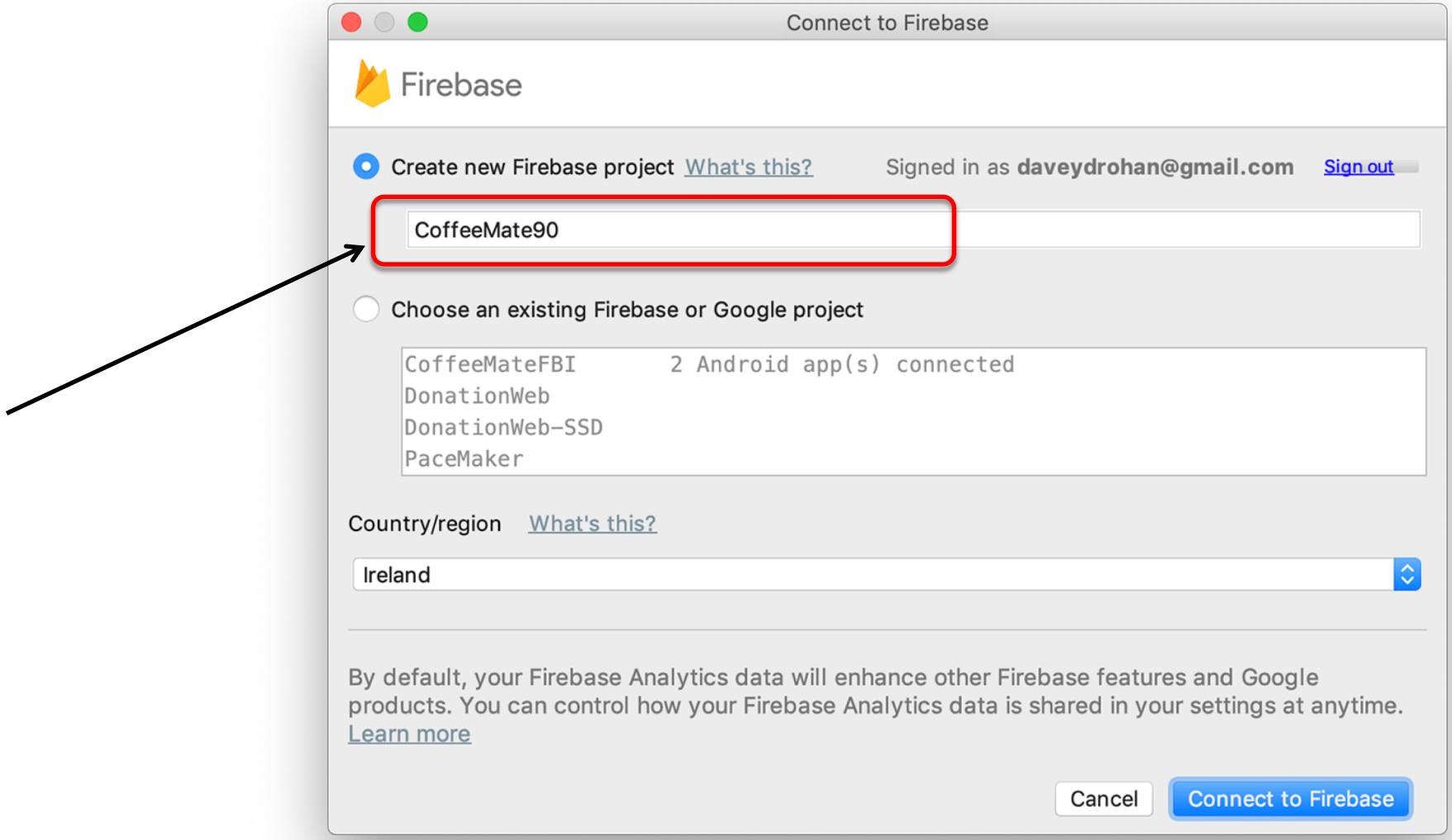




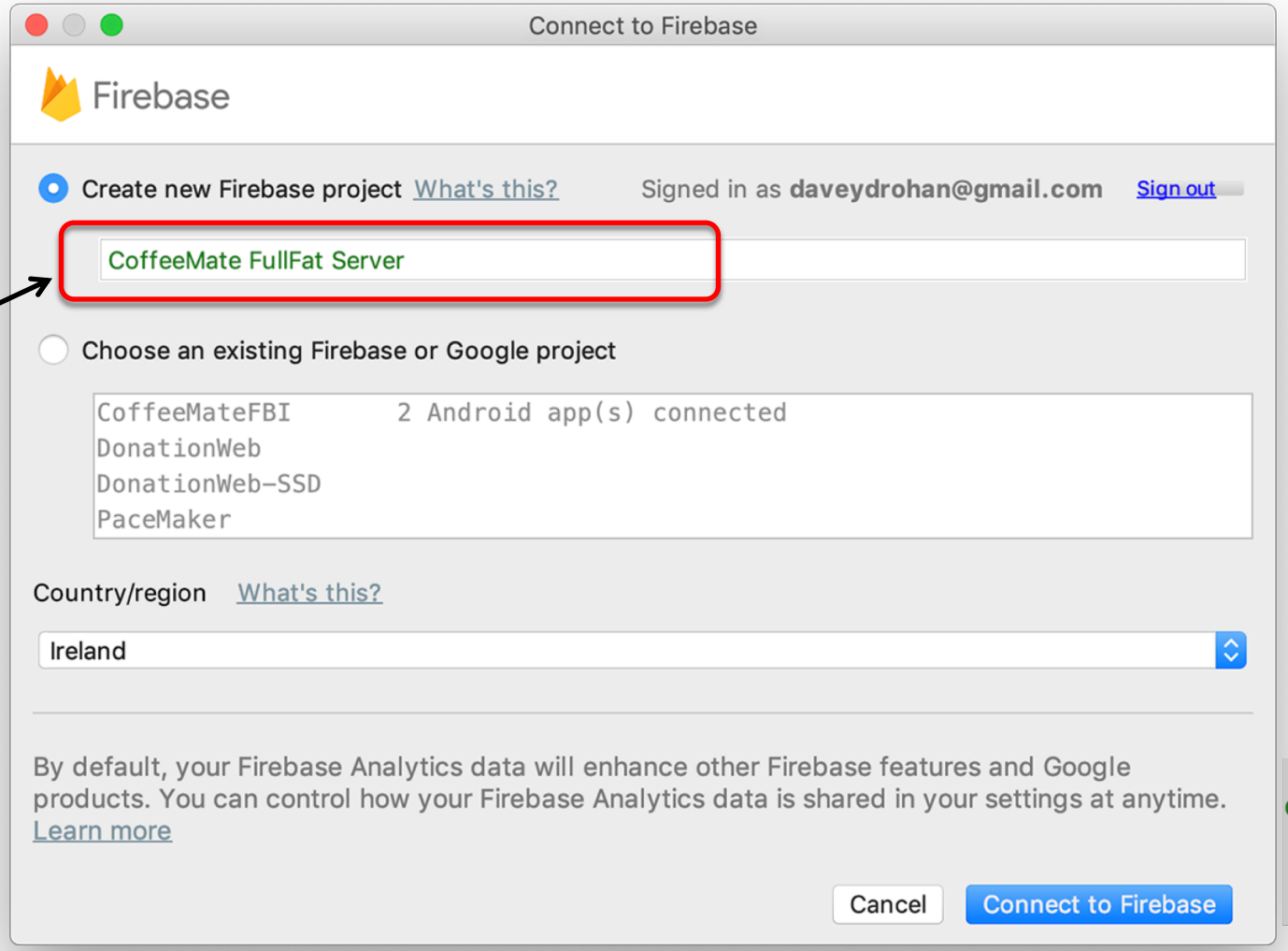
1. Connect your App to Firebase *

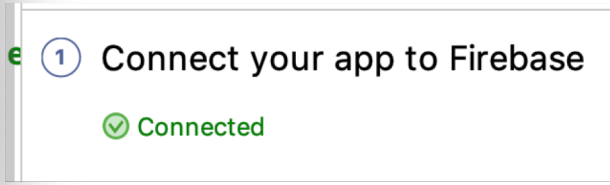


1. Connect your App to Firebase *

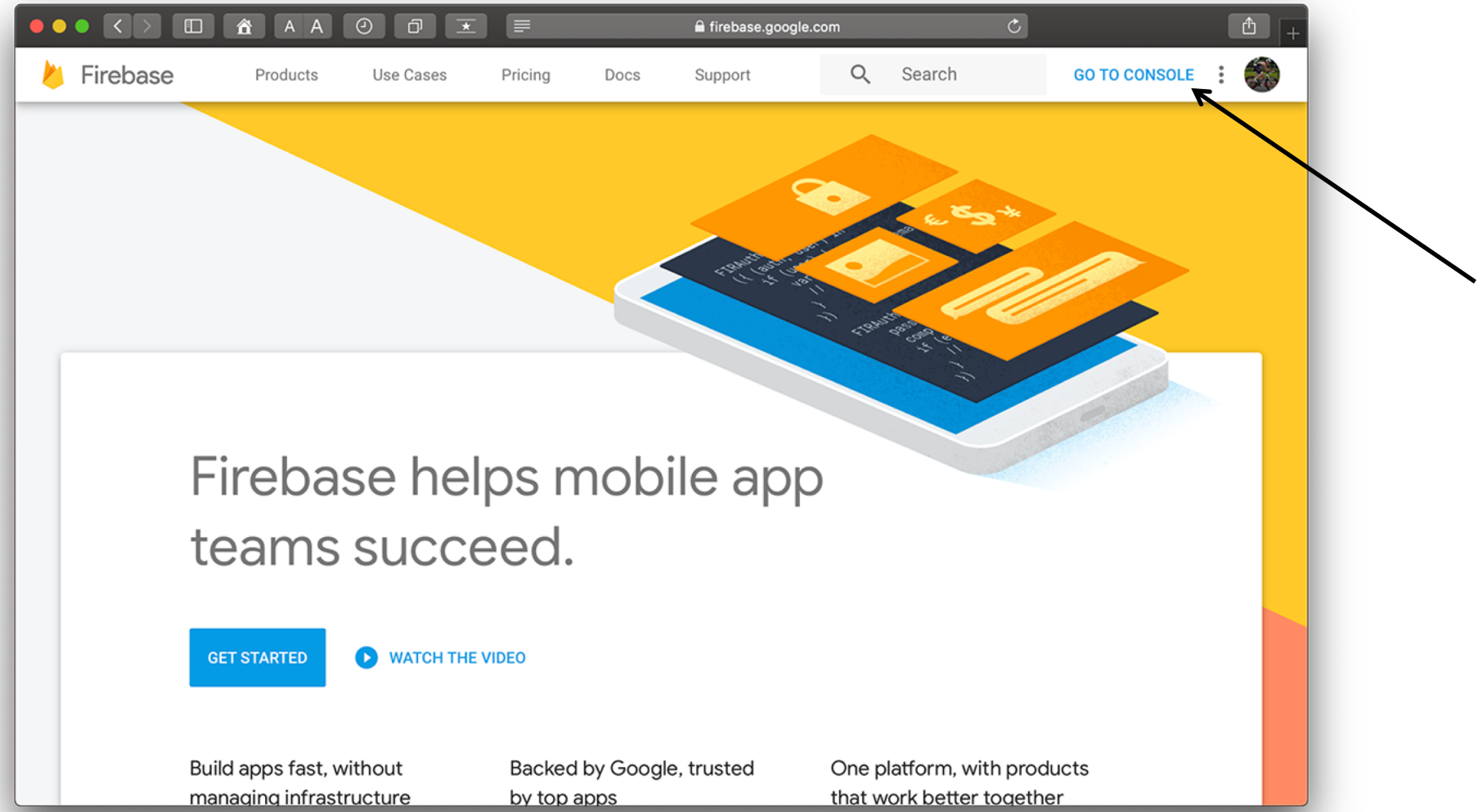


1. Connect your App to Firebase *

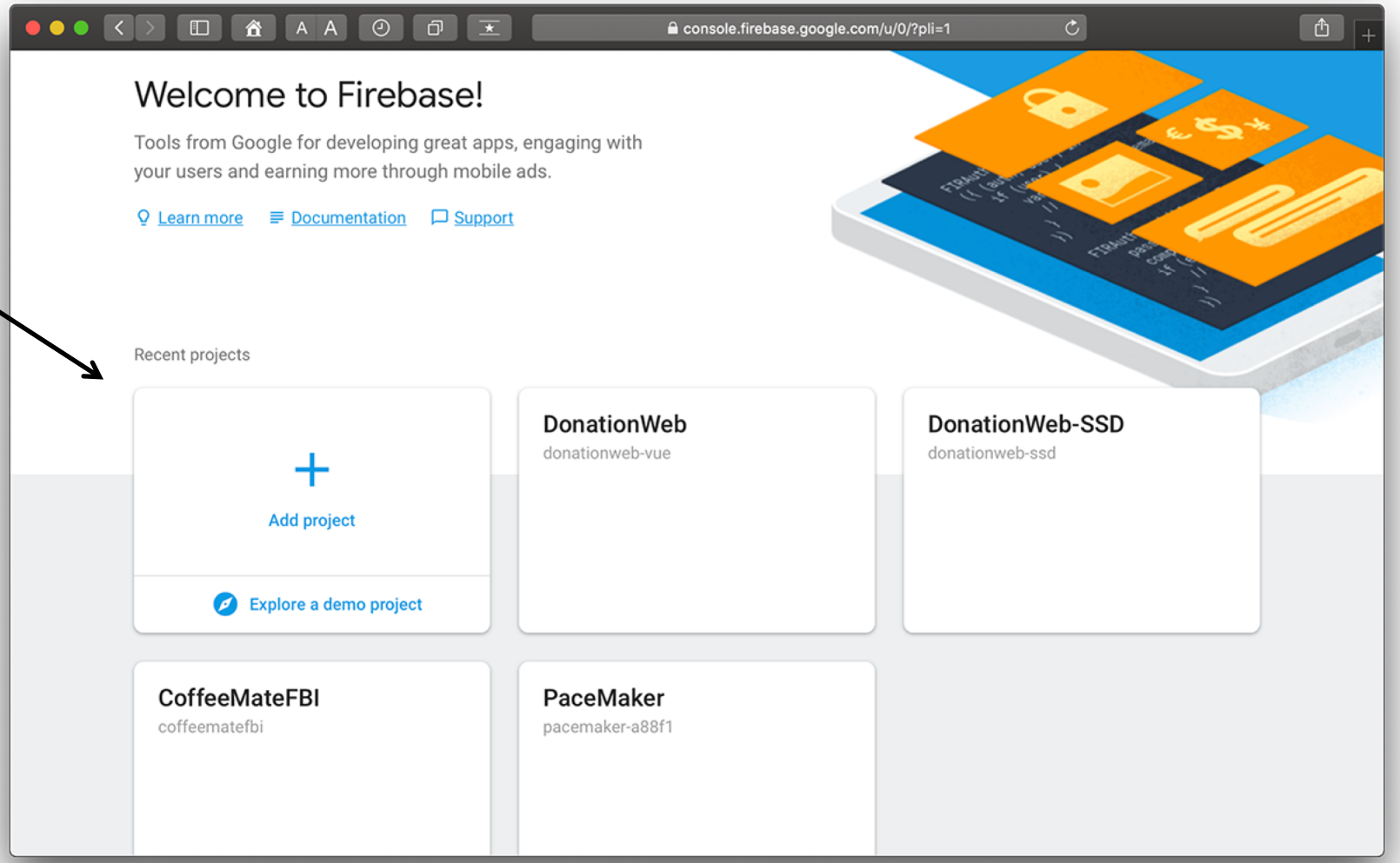




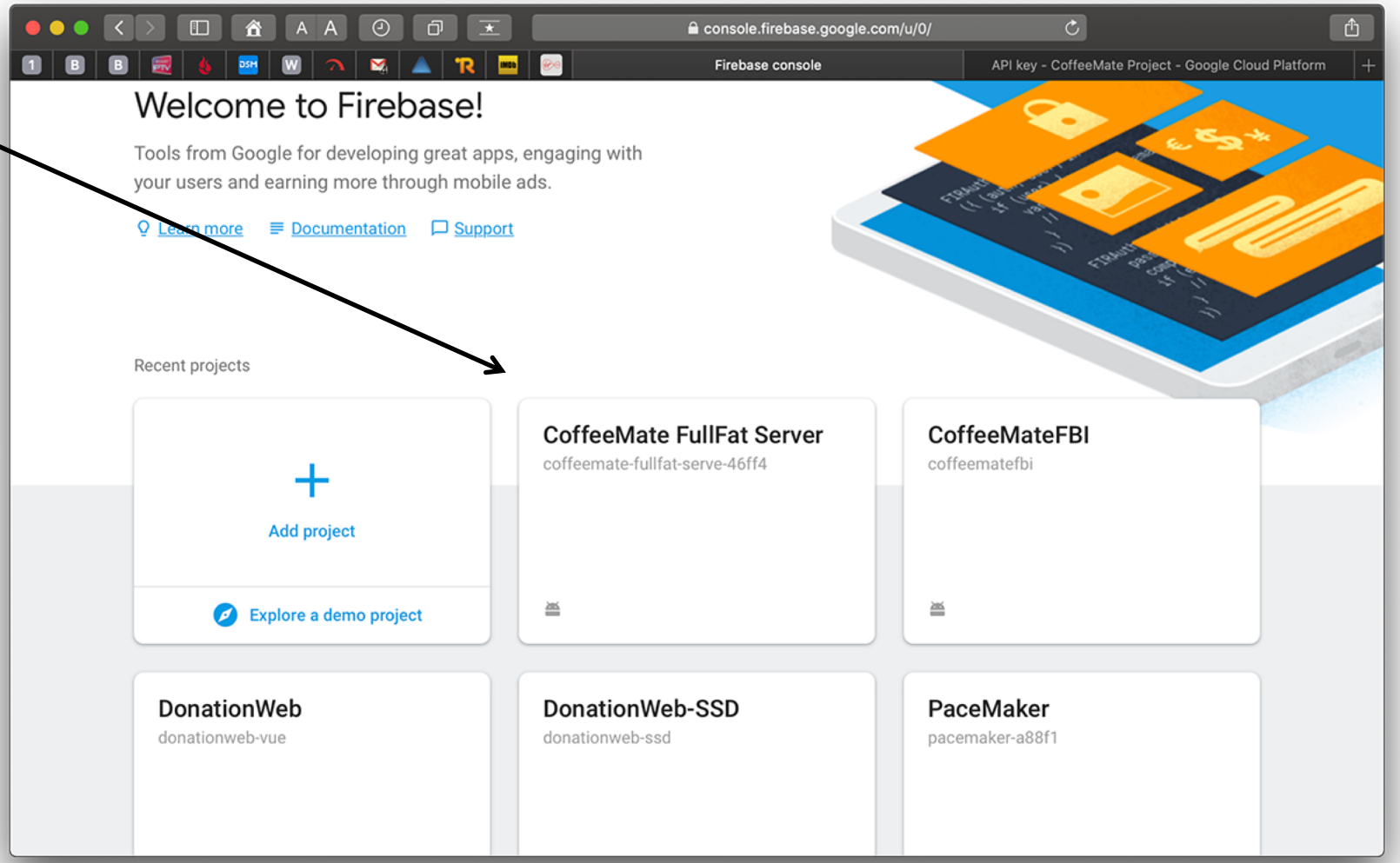
2. Add Firebase to your Android App *



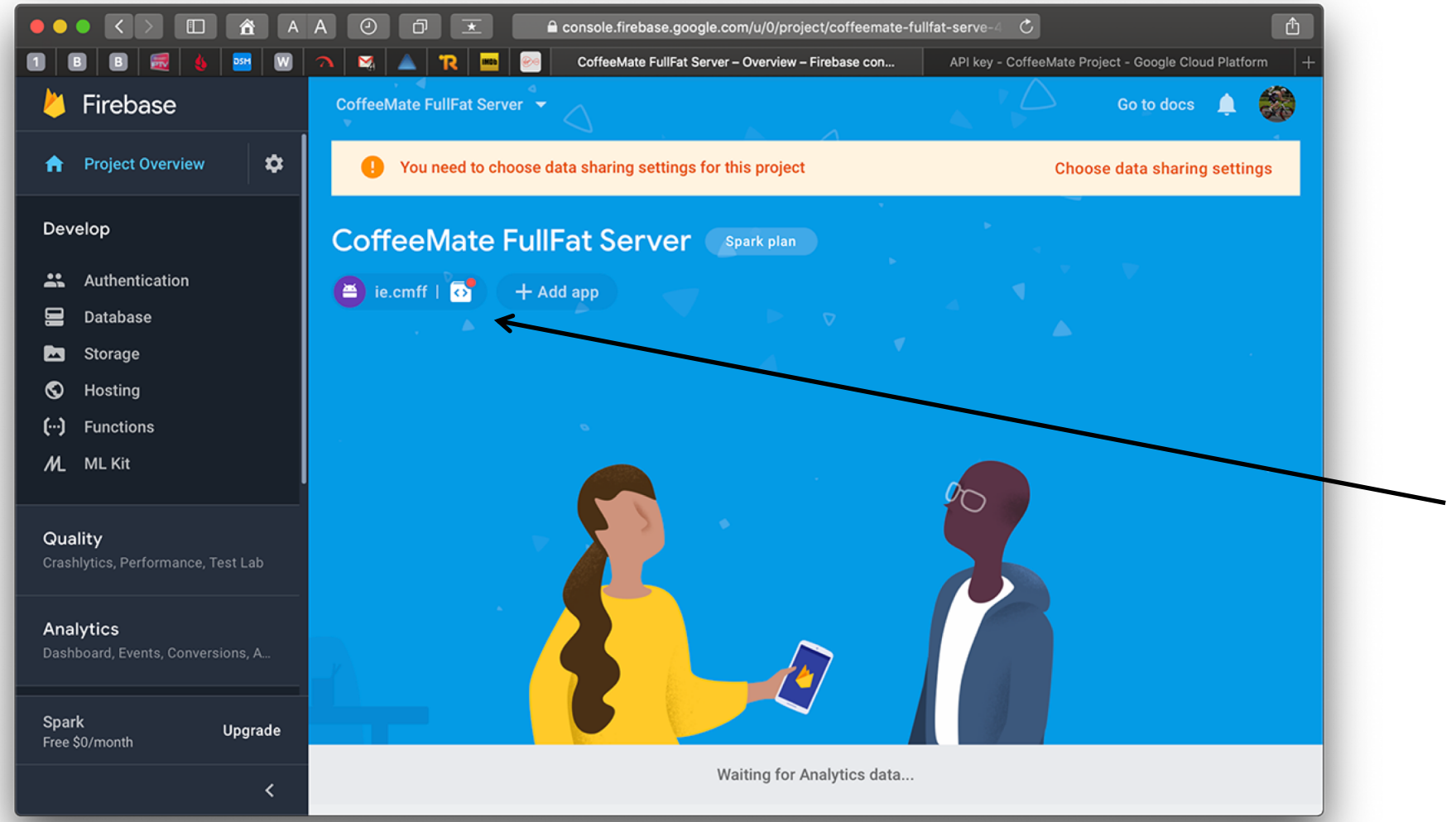
2. Add Firebase to your Android App *



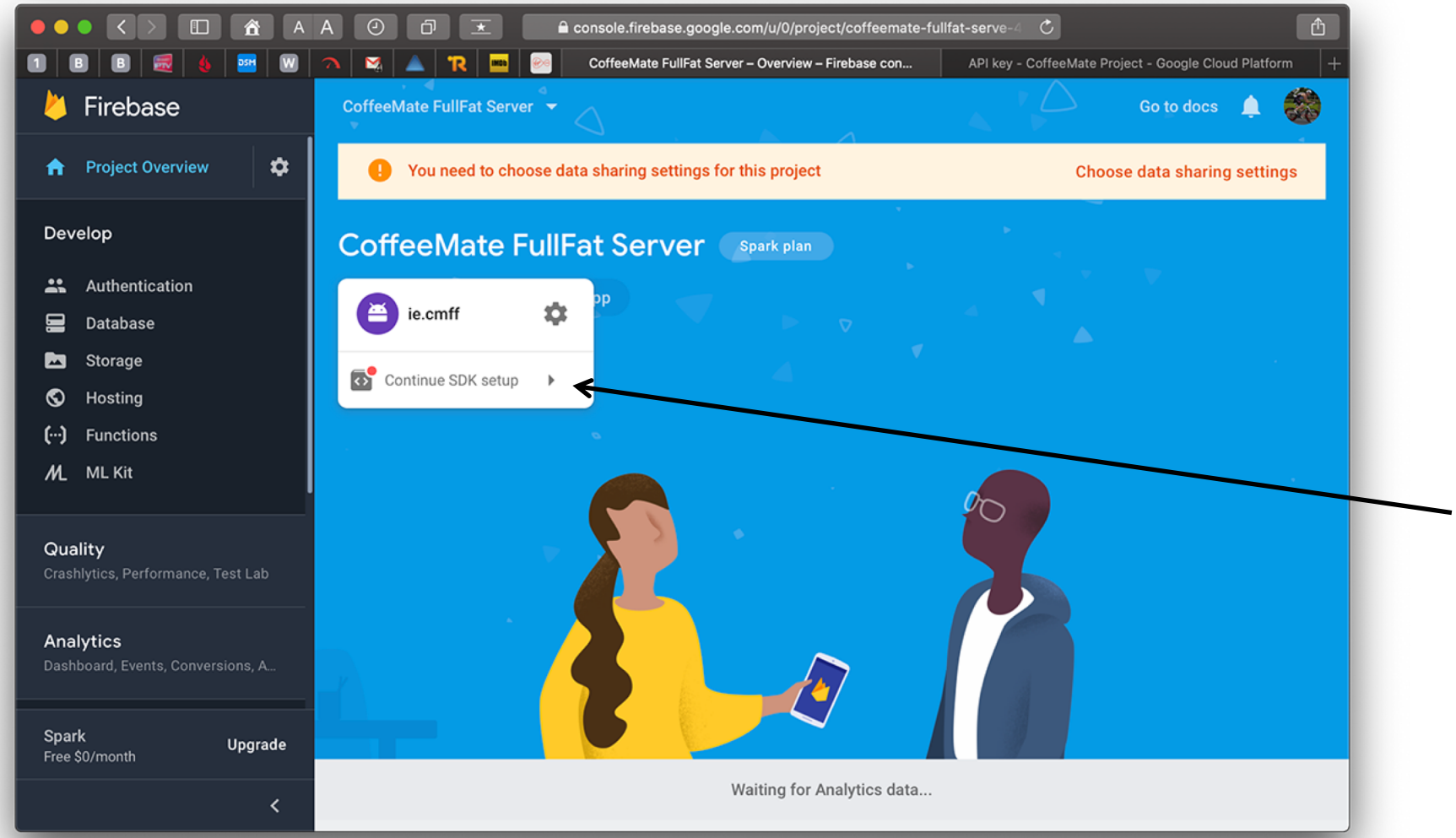
2. Add Firebase to your Android App *



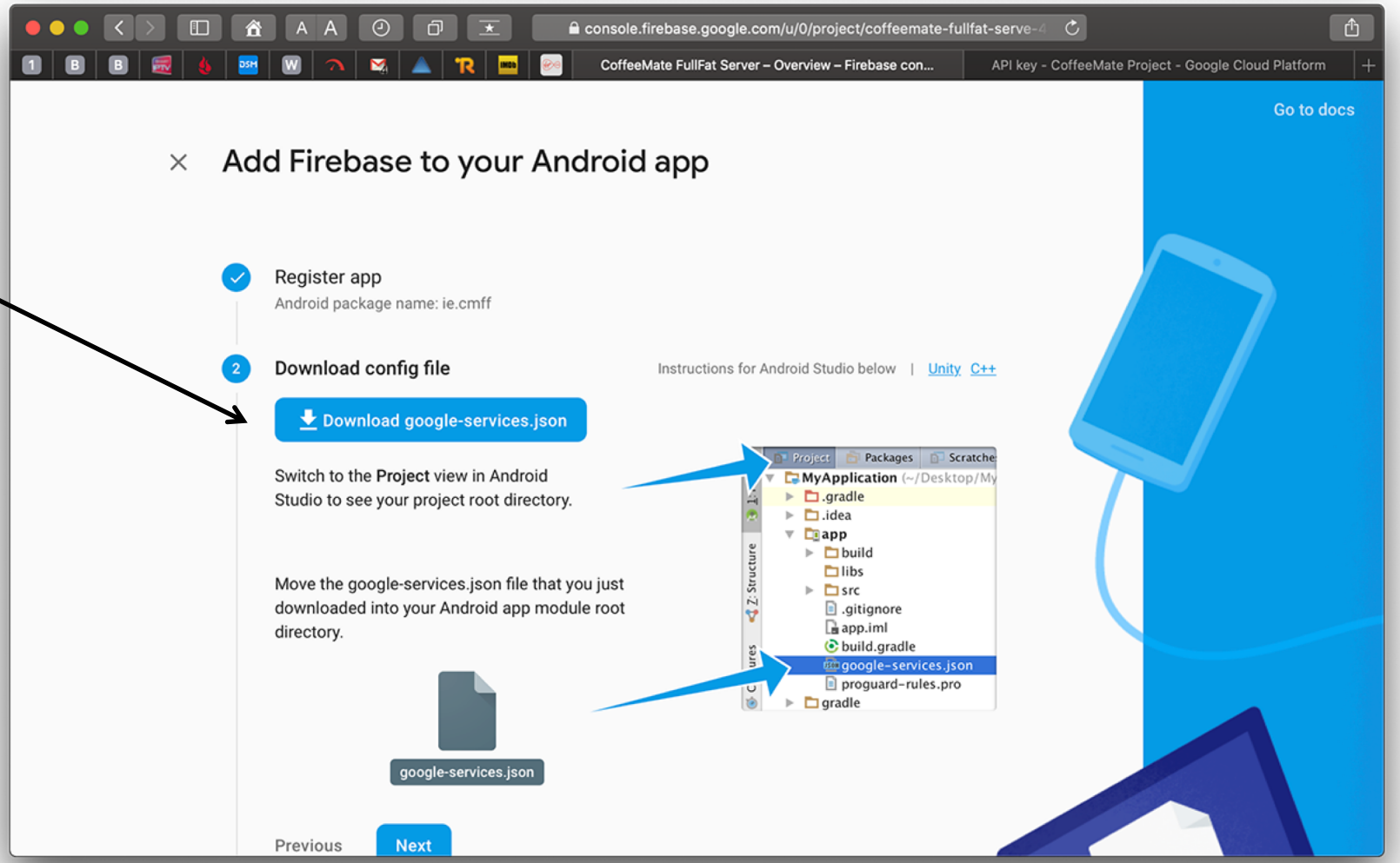
2. Add Firebase to your Android App *



2. Add Firebase to your Android App *

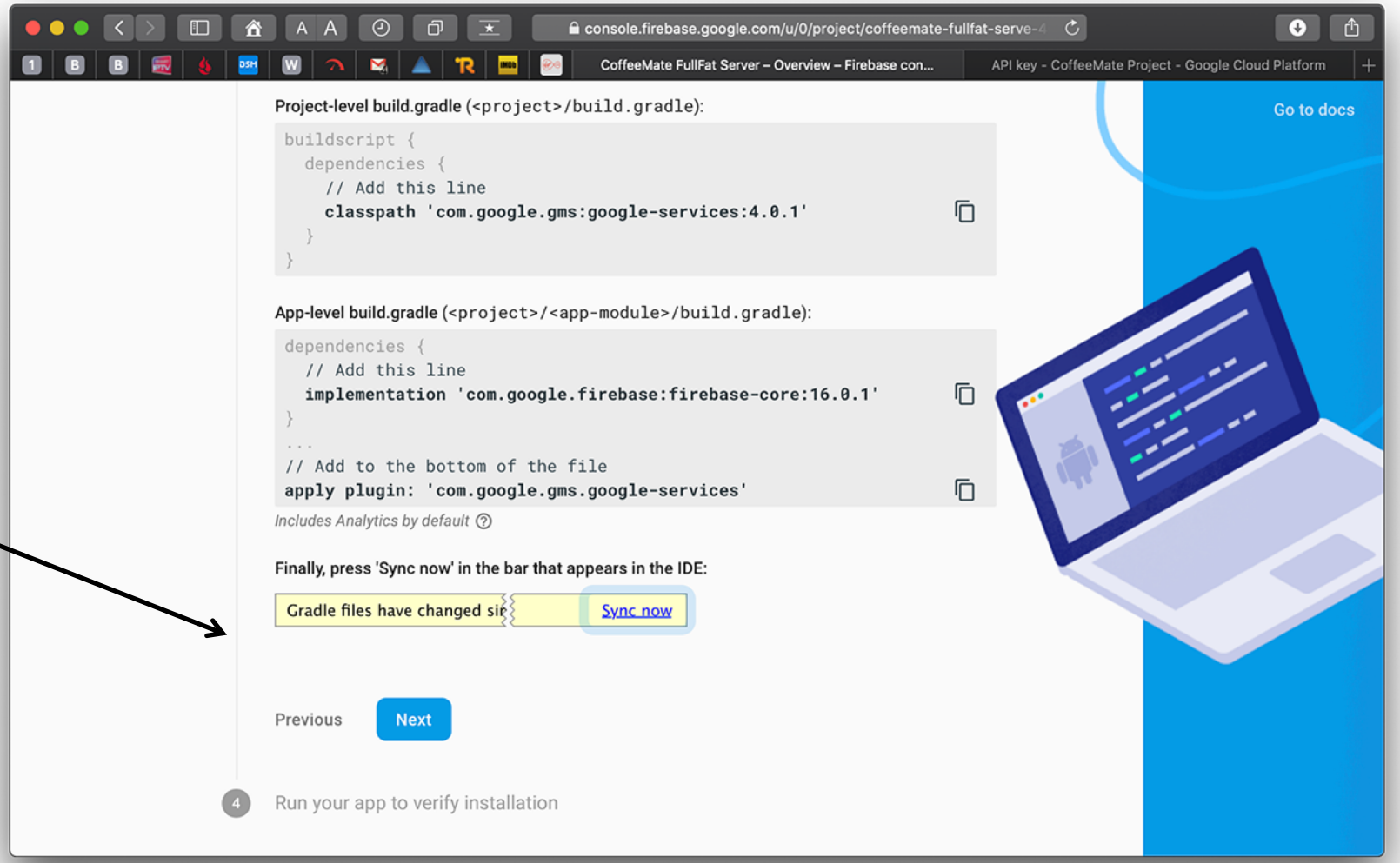


2. Add Firebase to your Android App *

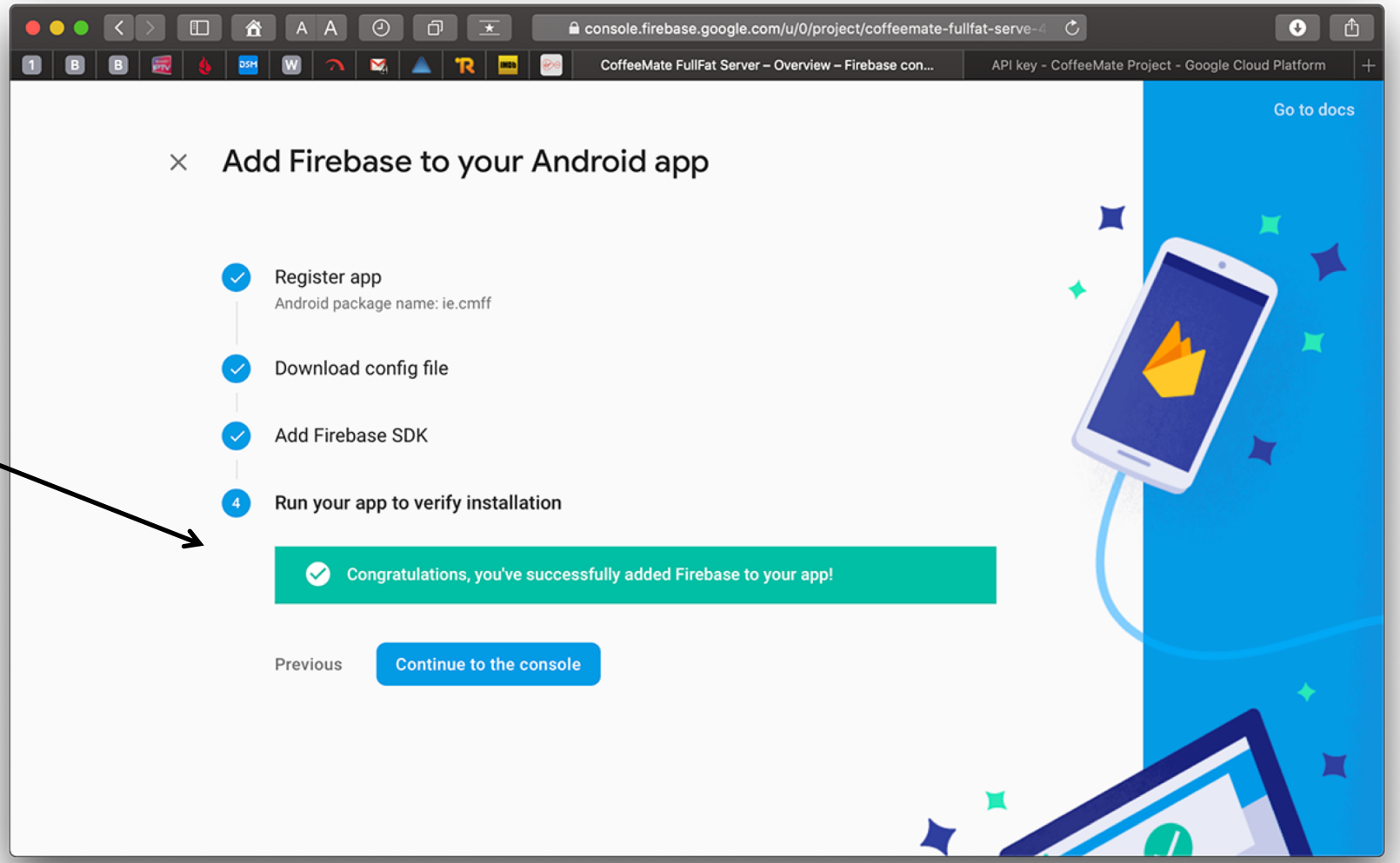


2. Add Firebase to your Android App *

2. Add Firebase to your Android App *

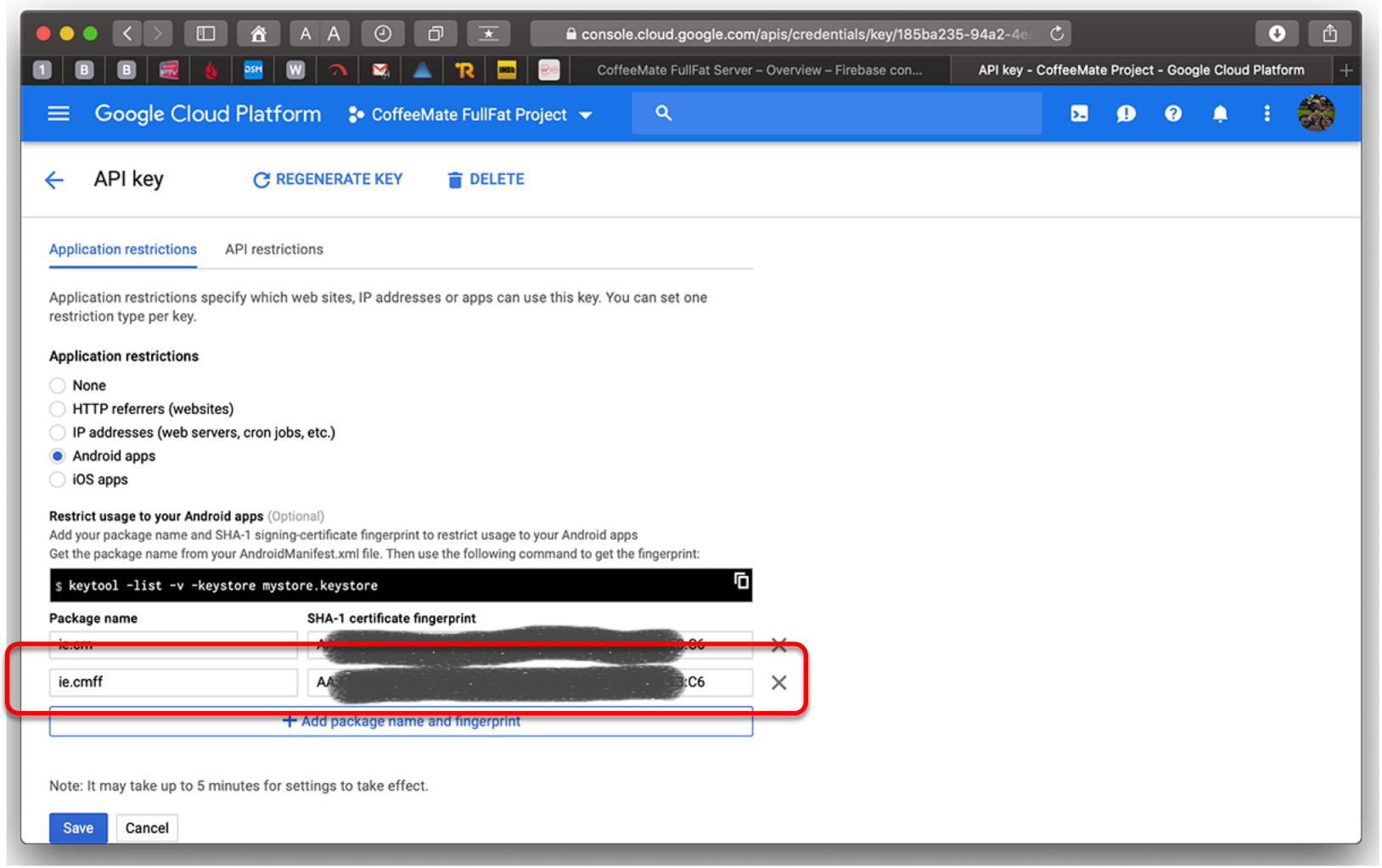


2. Add Firebase to your Android App *



2. Add Firebase to your Android App *

Remember to update/add to your google maps key if you change your package name (or else your map won't work!)



CoffeeMate.9.0 Dependencies (Dec/2018) *

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support:support-v4:28.0.0'
    implementation 'com.android.support:design:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    implementation 'com.makeramen:roundedimageview:2.2.1'
    implementation "com.android.support:recyclerview-v7:28.0.0"
    implementation "com.android.support:cardview-v7:28.0.0"

    implementation 'com.google.android.gms:play-services-auth:15.0.1'
    implementation 'com.google.android.gms:play-services-maps:15.0.1'
    implementation 'com.google.android.gms:play-services-location:15.0.1'

    implementation 'com.shobhitpuri.custombuttons:google-signin:1.0.0'

    implementation 'com.google.firebase:firebase-core:16.0.1'
    implementation 'com.google.firebase:firebase-auth:16.0.1'
    implementation 'com.google.firebase:firebase-database:16.0.1'

    implementation 'com.firebaseui:firebase-ui-database:4.0.0'

    implementation 'com.github.bumptech.glide:glide:4.8.0'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.8.0'
}
```

FirebaseManager – our Database utility class *

```
public class FirebaseManager {
    private static final String TAG = "coffeemate";
    public DatabaseReference    mFirebaseDatabase;
    public static   String      mFirebaseUserId;
    public FirebaseListener     mFirebaseListener;
```

- Our Firebase db reference.

- Binding to our db instance.

```
public void open() {
    //Set up local caching
    FirebaseDatabase.getInstance().setPersistenceEnabled(true);

    //Bind to remote Firebase Database
    mFirebaseDatabase = FirebaseDatabase.getInstance().getReference();
    Log.v(TAG, "Database Connected :" + mFirebaseDatabase);
}
```

FirebaseManager – usage *

```
public FirebaseAuth mFirebaseAuth;
public FirebaseUser mFirebaseUser;
public FirebaseManager mFirebaseManager;

@Override
public void onCreate() {
    super.onCreate();
    Log.v("coffeemate", "CoffeeMate App Started");
    mInstance = this;
    mFirebaseManager = new FirebaseManager();
    mFirebaseManager.open();
}
```

- Our utility class reference (inside our Application object class).
- Creating & 'Opening' a connection to our Firebase db.

FirebaseManager – Querying the database *

```
public Query getAllCoffees()
{
    return mFirebaseDatabase.child("user-coffees")
            .child(mFirebaseUserId)
            .orderByChild("rating");
}

public Query getFavouriteCoffees()
{
    return mFirebaseDatabase.child("user-coffees")
            .child(mFirebaseUserId)
            .orderByChild("favourite")
            .equalTo(true);
}
```

- ‘Query’ing the ‘*mFirebaseUserId*’ node of the ‘*user-coffees*’ node.
- As above, but only where the ‘*favourite*’ field is ‘*true*’.

FirebaseManager – usage (in CoffeeFragment) *

```
@Override
public void onResume() {
    super.onResume();

    if (!favourites)
        query = app.mFirebaseManager.getAllCoffees();
    else
        query = app.mFirebaseManager.getFavouriteCoffees();

    updateView(query);
}
```

- Returning a ‘query’ of all coffees, (or favourite coffees) which we pass to our custom FirebaseUI adapter via *updateView()*

FirebaseManager – our Database utility class *

- Adding an 'ValueEventListener' and triggering our callback

```
public void getACoffee(final String coffeeKey)
{
    mFirebaseDatabase.child("user-coffees").child(mFirebaseUserId)
            .child(coffeeKey).addValueEventListener(
            new ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot dataSnapshot) {
                    Log.v(TAG,"The read Succeeded: " + dataSnapshot.toString());
                    mFirebaseListener.onSuccess(dataSnapshot);
                }
                @Override
                public void onCancelled(DatabaseError firebaseError) {
                    Log.v(TAG,"The read failed: " + firebaseError.getMessage());
                    mFirebaseListener.onFailure();
                }
            });
}
```

FirebaseManager – usage (in EditFragment) *

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);

    if(getArguments() != null)
        app.mFirebaseManager.getACoffee(
                getArguments().getString("coffeekey")
        );
}
```

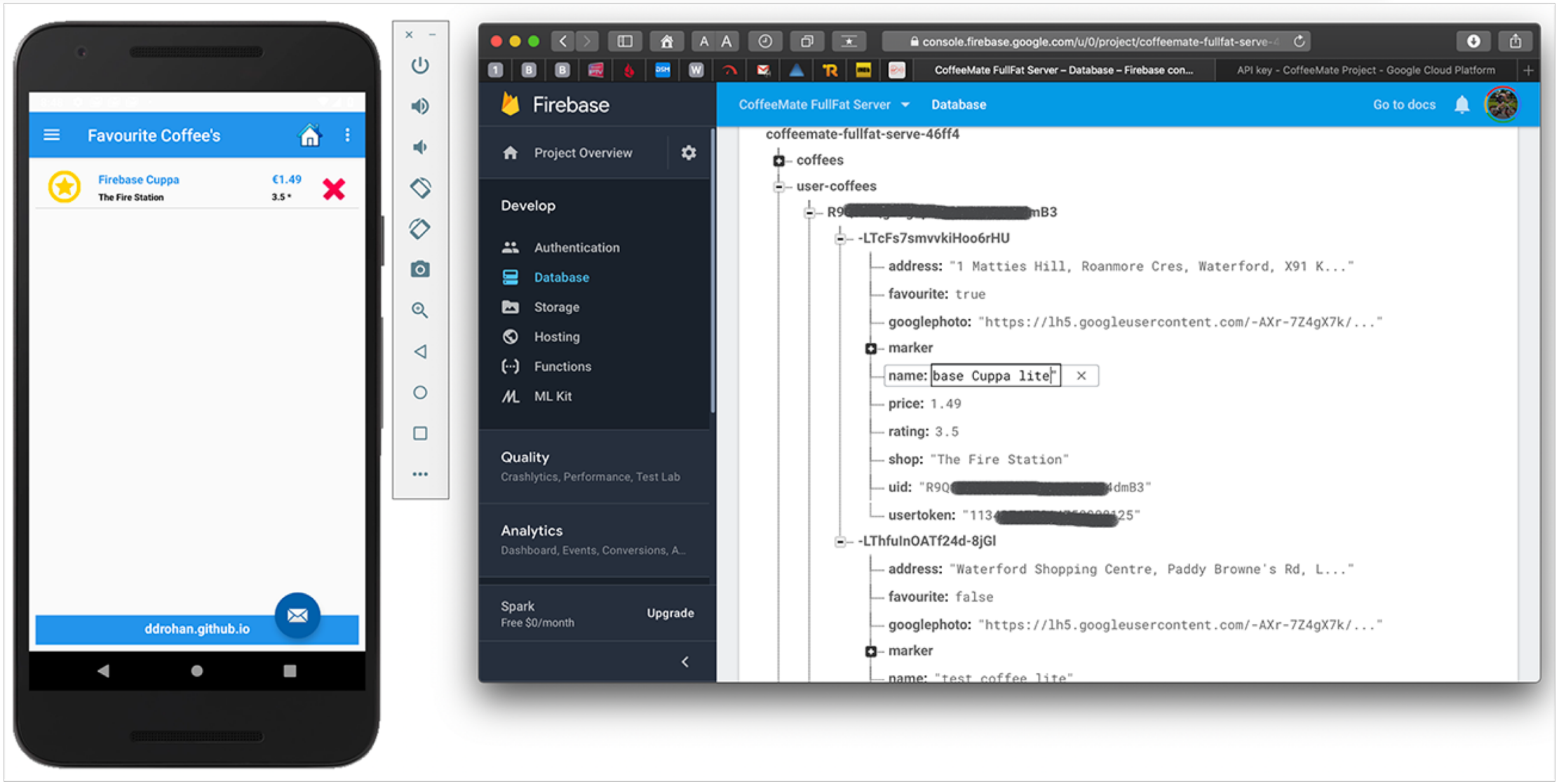
- Retrieving the '*coffeeKey*' from the bundle and calling '*getACoffee()*'- which in turn triggers the callback on the edit screen

FirebaseManager – usage (in EditFragment) *

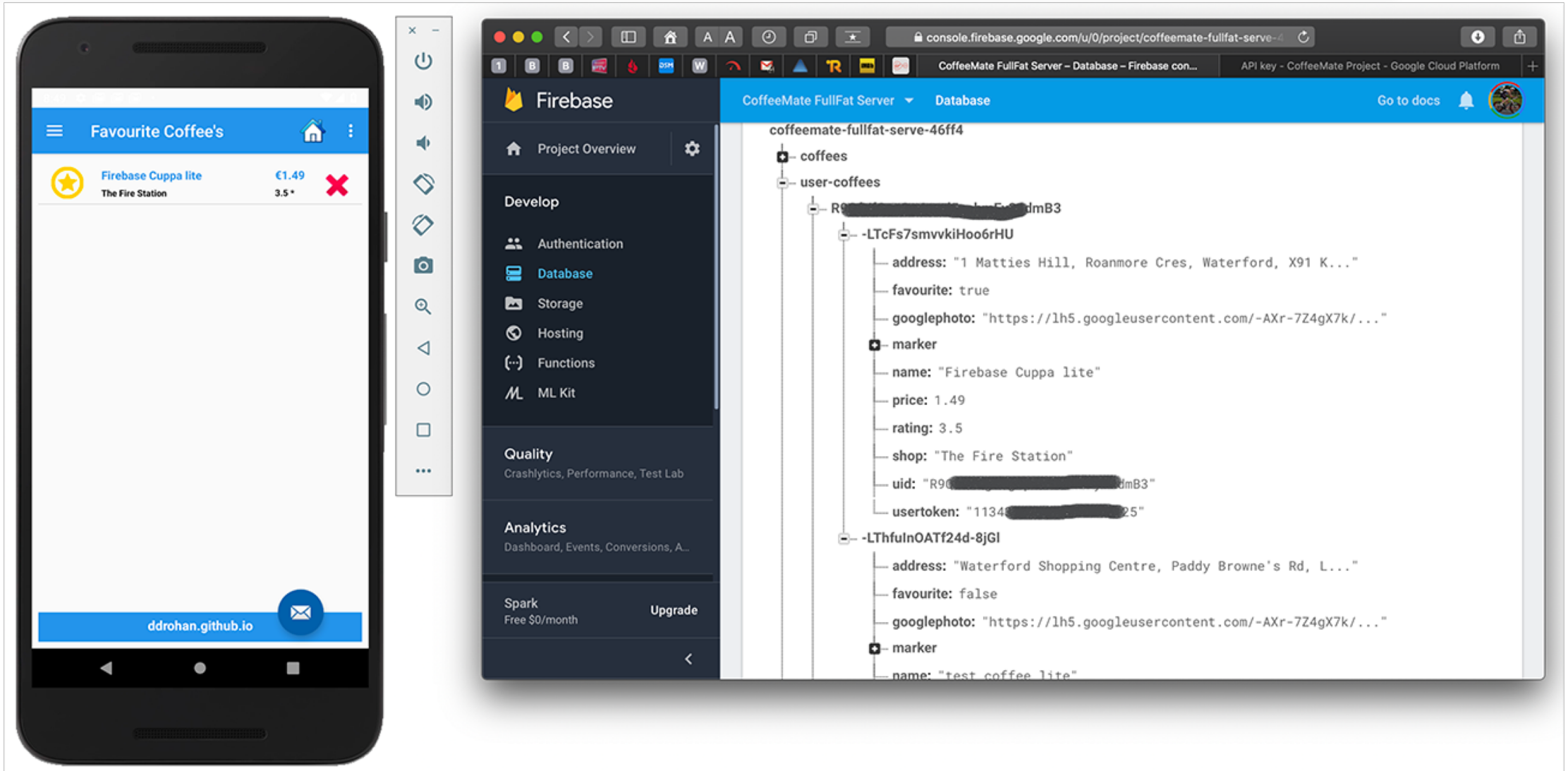
```
//@Override
public void updateUI() {...}

@Override
public void onSuccess(DataSnapshot dataSnapshot) {
    aCoffee = dataSnapshot.getValue(Coffee.class);
    Log.v("coffeemate","getACoffee() snapshot is " + aCoffee );
    if(aCoffee != null)
        updateUI();
}

@Override
public void onFailure() {
    Log.v("coffeemate", "Unable to Read/Update Coffee Data");
    Toast.makeText(getActivity(),"Unable to Read/Update Coffee Data",
            Toast.LENGTH_LONG).show();
}
```

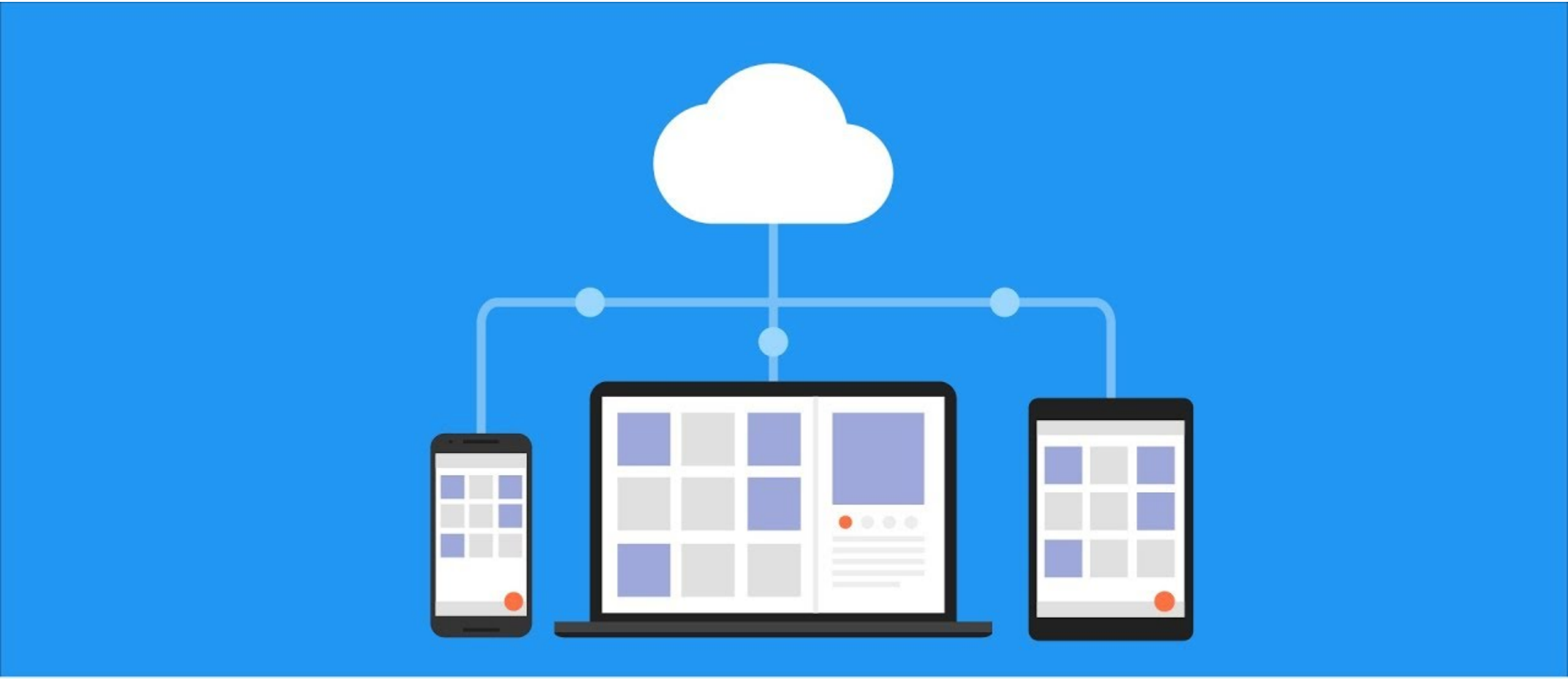
- Retrieving the '*coffeeKey*' from the bundle and calling '*getACoffee()*'- which in turn triggers the callback on the edit screen

Firebase Console (actual data)



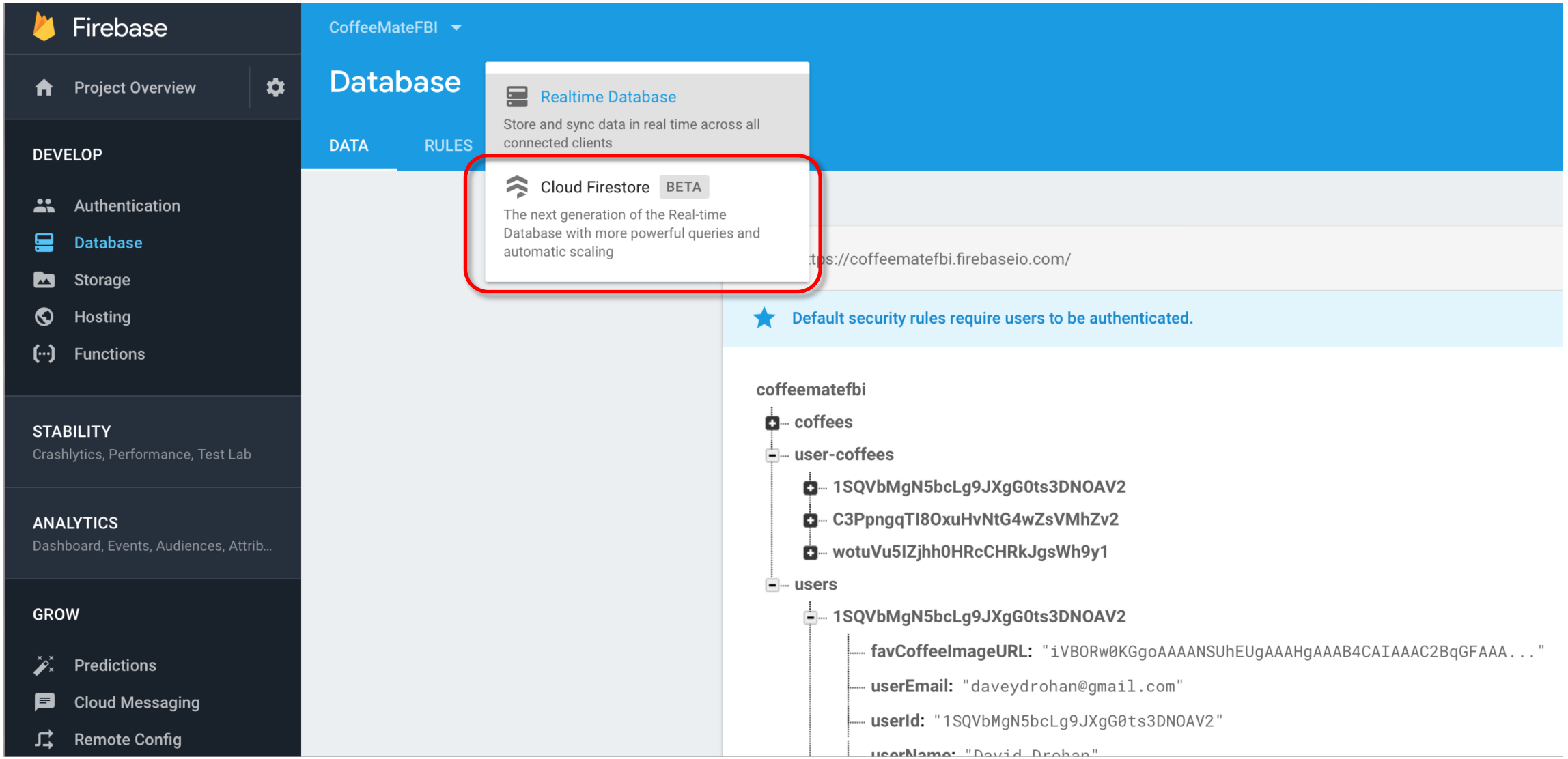
Firebase Console (actual data)





Cloud Firestore

Firebase Console – Cloud Firestore



What is it?

- Cloud Firestore is a NoSQL document database that lets you easily store, sync, and query data for your mobile and web apps—at a global scale.
- Though this may sound like something similar to the **Realtime Database**, Firestore brings many new things to the platform that makes it into something completely different from Realtime Database.

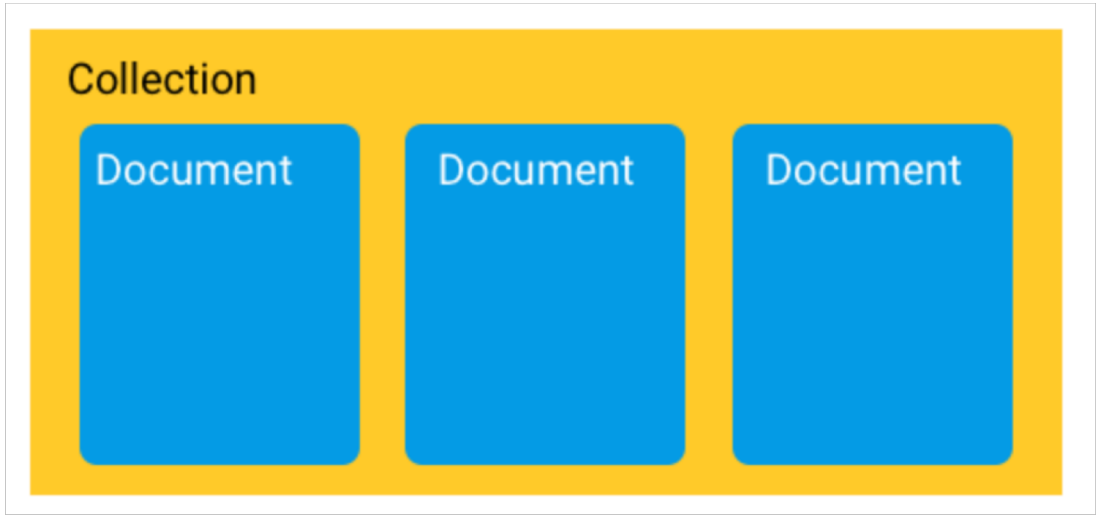
Improved Querying and Data Structure

- Where Realtime Database stores data in the form of a giant JSON tree, Cloud Firestore takes a much more structured approach.
- Firestore keeps its data inside objects called **documents**. These documents consist of key-value pairs and can contain any kind of data, from strings to binary data to even objects that resemble JSON trees (Firestore calls it maps).

(Sounds a bit like mongodb to me ☺)

Improved Querying and Data Structure

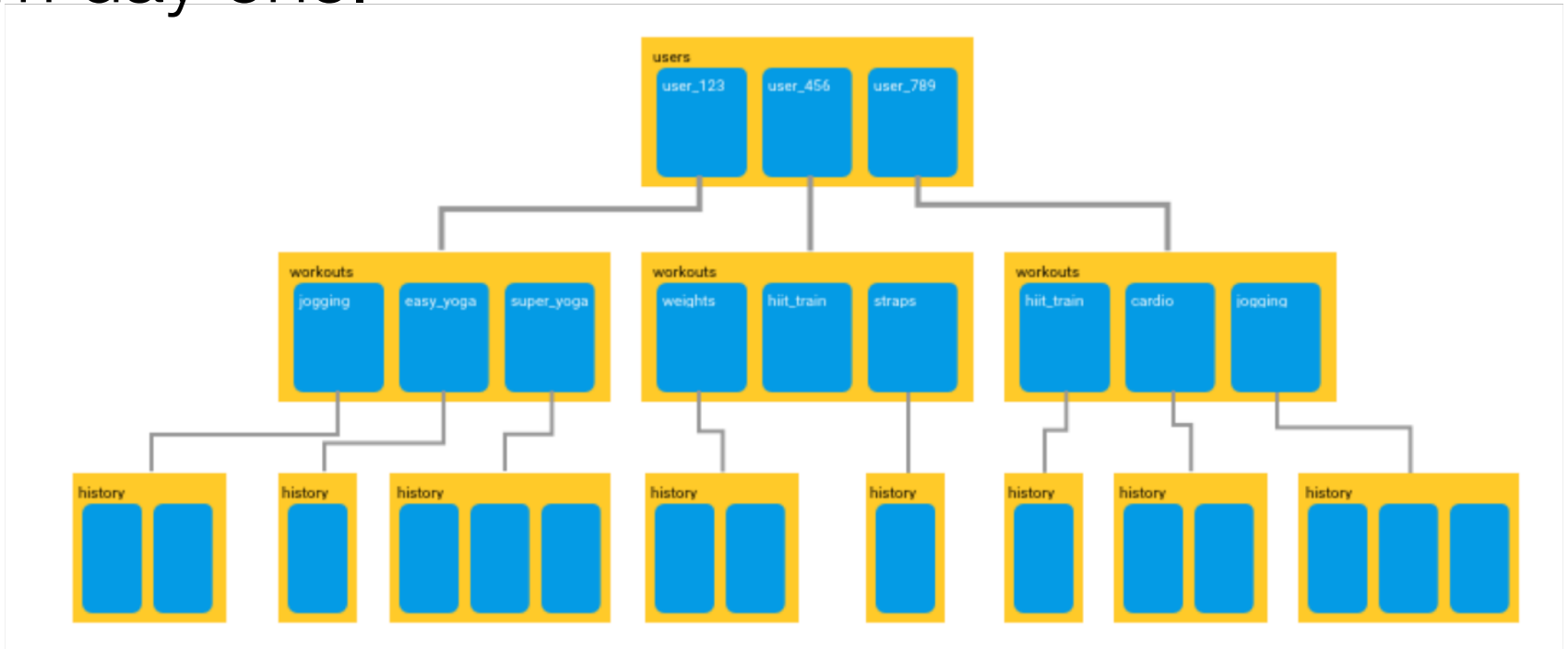
- The documents, in turn, are grouped into **collections**.



- Firestore database can consist of multiple collections that can contain documents pointing towards sub-collections.
- These sub-collections can again contain documents that point to other sub-collections, and so on.

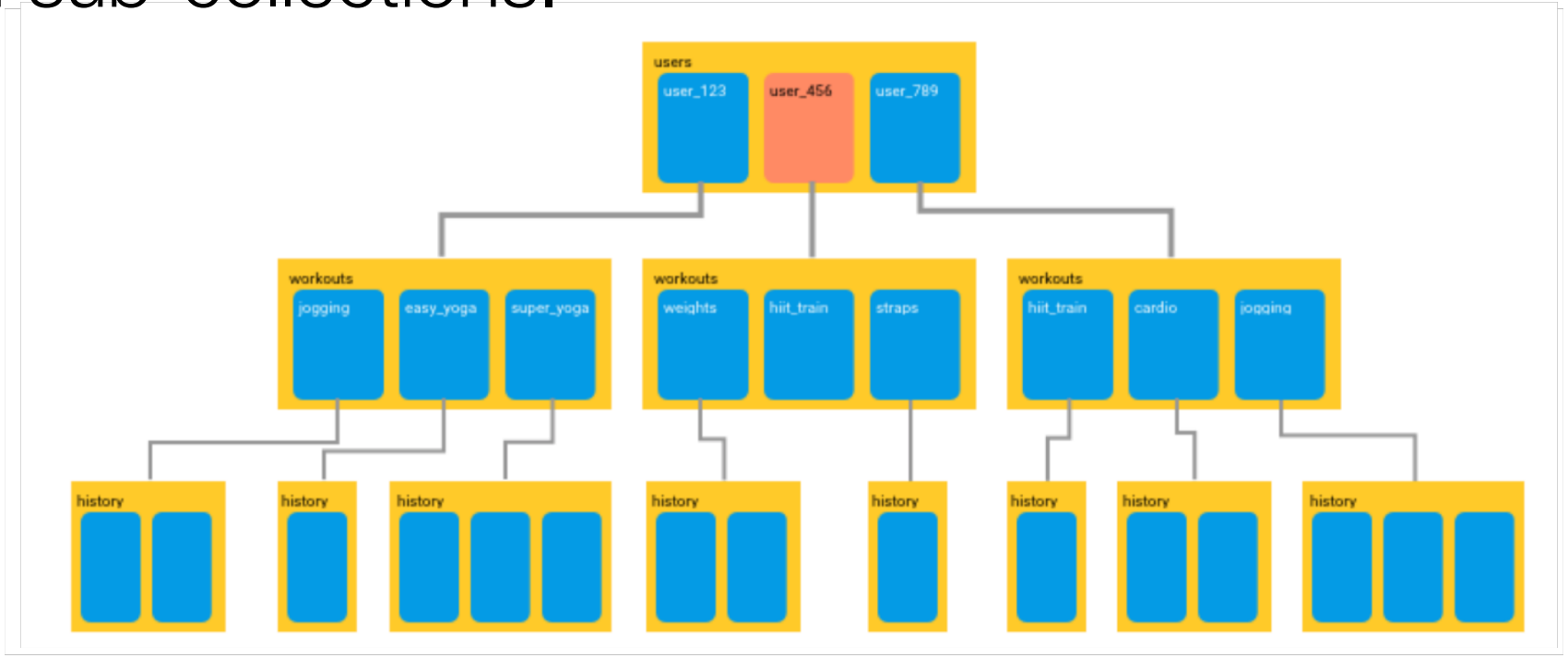
Improved Querying and Data Structure

- You can build hierarchies to store related data and easily retrieve any data that you need using queries. All queries can scale with the size of your result set, so your app is ready to scale from day one.



Improved Querying and Data Structure

- Firestore's queries are ***shallow.*** By this, I mean to say that in Firestore, you can simply fetch any document that you want without having to fetch all of the data that is contained in any of its linked sub-collections.



Better Scalability

- Though Firebase's Realtime Database is capable of scaling, things will start to get crazy when your app becomes really popular or if your database becomes really massive.
- Cloud Firestore is based on Googles Cloud infrastructure. This allows it to scale much more easily and to a greater capacity than the Realtime Database.

Multi-Region Database

- In Firestore, your data is automatically copied to various regions. So if one data center goes offline due to some unforeseen reason, you can be sure that your app's data is still safe somewhere else.
- Firestore's multi-region database also provides strong consistency. Any changes to your data will be mirrored across every copy of your database.

Different Pricing Model

- The Realtime Database charges its users based on the amount of data that you have stored in the database.
- Cloud Firestore also charges you for the same, but the cost is significantly lower than that of Realtime Database and instead of basing the cost on the amount of data stored, Firestore's pricing is driven by the number of reads/writes that you perform.

Pricing https://firebase.google.com/pricing/

Products	Spark Plan Generous limits for hobbyists Free	Flame Plan Fixed pricing for growing apps $25/month	Blaze Plan Calculate pricing for apps at scale Pay as you go ✓ Free usage from Spark plan included*
Free Products A/B Testing, Analytics, App Indexing, Authentication (except Phone Auth), Cloud Messaging (FCM), Crashlytics, Dynamic Links, Invites, Performance Monitoring, Predictions, and Remote Config.	✓ Included	✓ Included Free	✓ Included Free
Realtime Database			
Simultaneous connections	100	100k	100k/database
GB stored	1 GB	2.5 GB	$5/GB
GB downloaded	10 GB/month	20 GB/month	$1/GB
Multiple databases per project	✗	✗	✓

Pricing https://firebase.google.com/pricing/

Products	Spark Plan Generous limits for hobbyists Free	Flame Plan Fixed pricing for growing apps $25/month	Blaze Plan Calculate pricing for apps at scale Pay as you go ✓ Free usage from Spark plan included*
Free Products A/B Testing, Analytics, App Indexing, Authentication (except Phone Auth), Cloud Messaging (FCM), Crashlytics, Dynamic Links, Invites, Performance Monitoring, Predictions, and Remote Config.	✓ Included	✓ Included Free	✓ Included Free
Cloud Firestore			
Stored data	1 GB total	2.5 GB total	$0.18/GB
Bandwidth	10GB/month	20GB/month	Google Cloud Pricing
Document writes	20K/day	100K/day	$0.18/100K
Document reads	50K/day	250K/day	$0.06/100K
Document deletes	20K/day	100K/day	$0.02/100K

Realtime Database

Cloud Firestore

Some important points though…

- Do not think RDBMS, think JSON. How data should be structured is very important.
- Firebase has a recycler view, that integrates with real time database smoothly without any listeners. (FirebaseUI)
- Test lab which is available in paid plan (Blaze), is an amazing feature for testing your app on different real and virtual devices (next section)
- Set developer mode to true when testing Remote Config (next section).

References & Links

- Presentation by Kaushal Dhruw & Shakti Moyal 2016
- https://firebase.google.com
- https://hackernoon.com/introduction-to-firebase-218a23186cd7
- Demo app available at https://goo.gl/WBP5fR

Questions?


Thanks!