Basics of Firebase Realtime Database

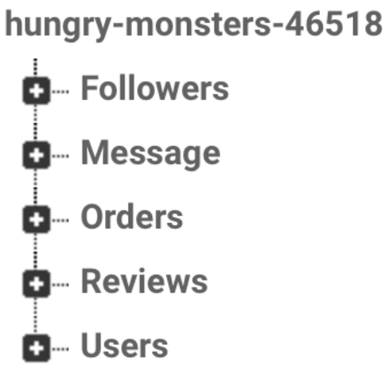
Getting a reference to your database

Once you have successfully set up your project on Firebase, you will need to get access to your database reference in order to read or write to it.

DatabaseReference databaseReference = FirebaseDatabase.getInstance().getReference();

Representation of data on Firebase

Data in Firebase Realtime Database is organised in a nested tree structure with each change either creating a new node or updating an existing one.
For example, a project may have this kind of high level structure -



```
hungry-monsters-46518
  + Followers
  + Message
  + Orders
  + Reviews
  + Users
```

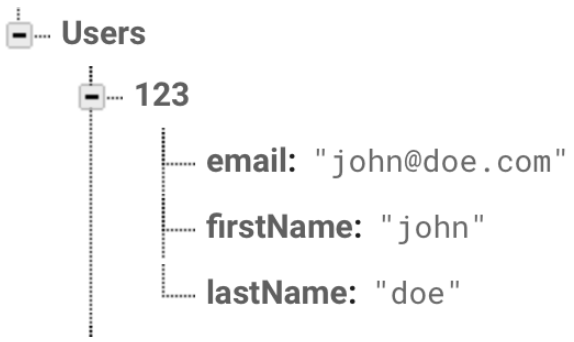
Each node here can contain sub-nodes representing individual “orders”, “reviews” etc

Adding nodes to your Firebase Data

Firebase automatically creates nodes that are missing by using the **child(Object param)** method. This also updates nodes if they already exist.
Consider the following code —>

databaseReference.child(“Users”).child(123).child("firstName").setValue("john");
databaseReference.child(“Users”).child(123).child("lastName").setValue("doe");
databaseReference.child(“Users”).child(123).child(“email”).setValue("john@doe.com");

Every call to child() checks whether a node with the string passed as parameter already exists in your database. If it doesn’t, one is created. The above code for example would create a node as below -



```
- Users
    - 123
        email: "john@doe.com"
        firstName: "john"
        lastName: "doe"
```

Listening for changes in data

In the above example, a possible use case would be for your app to perform certain actions when specific data entries in the database is updated.
Firebase provides a method called **addListenerForSingleValueEvent** to observe changes in specific nodes in your data tree.
In the above example, to observe a change in user email, we could do -

```
databaseReference.child("Users").child(123)
      .child("email").addListenerForSingleValueEvent(new ValueEventListener() {

@Override
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
   updatedEmail = dataSnapshot.getValue(String.class);
  }
}
```

Here, using the **child()** method, we specify exactly which node we want to observe. (The email field of user with ID 123)
Then we add a listener and implement **onDataChange.** The **dataSnapshot** here represents the state of the field that we are listening to when it is updated.
Lastly, notice that we need to specify the data type of the value we are expecting(String.class in this case).
This can be expanded further to observe changes in entire models or for multiple user IDs at once.

Firebase Auth

Firebase provides a simple API for authenticating users to your app using a username and password.

Signing up new users —>

```
FirebaseAuth auth = FirebaseAuth.getInstance();

auth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
   @Override
   public void onComplete(@NonNull Task<AuthResult> task) {
                  if (task.isSuccessful()) {
                  task.getResult().getUser() ——> gives the newly created user
                  }
         }
}
```

Logging in an existing user —>

```
auth.signInWithEmailAndPassword(email, password)
      .addOnCompleteListener(SignInActivity.this, new OnCompleteListener<AuthResult>() {
         @Override
         public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {
                  // successful login!
            }
    }
}
```

Please note however that Firebase Auth does not automatically update content in any “Users” node that you may have created in your realtime database. To achieve this, you would need to implement Firebase Auth and append to your database node on success