

Firestore benefits

Key/Value document (NoSQL)

Real time database

Hosted

URL based

Integrated Authentication

Firestore NoSQL data design principles:

- Don't treat firestore like a relational database
- Use Root branches as your primary container
- Avoid deep nesting
- Duplicating Data (More art than science)
- Design around data access
- Data design is driven by data access nodes
- Iterative Design

Understand Data

- Arrays
- Null locations
- Reference

The whole project is in this github [link](#)

Here's the breakdown how it works:

First need to add dependencies for an Android project in build.gradle file:

```
compile 'com.google.firebase:firebase-core:16.0.1'  
compile 'com.google.firebase:firebase-database:16.0.6'  
compile 'com.google.firebase:firebase-auth:16.1.0'  
compile 'com.firebaseui:firebase-ui-auth:4.3.0'
```

Declare Firestore variables

```
private FirebaseDatabase mFirebaseDatabase;  
private DatabaseReference mMessagesDatabaseReference;  
private ChildEventListener mChildEventListener;
```

```
private FirebaseAuth mAuth;  
private FirebaseAuth.AuthStateListener mAuthStateListener;
```

Initialize declared variables in onCreate()

```
    FirebaseDatabase.getInstance();  
    mAuth = FirebaseAuth.getInstance();  
    mMessagesDatabaseReference =  
mFirebaseDatabase.getReference().child("messages");
```

Send button will do the mapping of a POJO object with the database and push it there:

```
    // Send button sends a message and clears the EditText  
    mSendButton.setOnClickListener(new View.OnClickListener() {  
        @Override  
        public void onClick(View view) {  
            // TODO: Send messages on click  
  
            FriendlyMessage friendlyMessage = new  
FriendlyMessage(mMessageEditText.getText().toString(), mUsername,  
null);  
mMessagesDatabaseReference.push().setValue(friendlyMessage);  
  
            // Clear input box  
            mMessageEditText.setText("");  
        }  
    });
```

We then create the database event change listener method in MainActivity,

```
private void attachDatabaseReadListener() {  
  
    if(mChildEventListener == null) {  
        mChildEventListener = new ChildEventListener() {  
            @Override  
            public void onChildAdded(@NonNull DataSnapshot
```

```

dataSnapshot, @Nullable String s) {
    FriendlyMessage friendlyMessage =
dataSnapshot.getValue(FriendlyMessage.class);
    mMessageAdapter.add(friendlyMessage);
}

@Override
public void onChildChanged(@NonNull DataSnapshot
dataSnapshot, @Nullable String s) {

}

@Override
public void onChildRemoved(@NonNull DataSnapshot
dataSnapshot) {

}

@Override
public void onChildMoved(@NonNull DataSnapshot
dataSnapshot, @Nullable String s) {

}

@Override
public void onCancelled(@NonNull DatabaseError
databaseError) {

}
};

mMessagesDatabaseReference.addChildEventListener(mChildEventListener);
}
}

```

The flow goes like this from here:

onResume() : AddAuthStateChangeListener()

-> OnAuthStateChange() : if there's user it will call attachDatabaseReadListener()

otherwise, it will start the Firebase Auth SignIn procedure

Similarly in onPause() we will need to remove the Firebase Auth and Database listeners, as we no longer intend to listen to these when activity is not visible.

This is how authentication listeners works:

```
mFirebaseAuthStateListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth
firebaseAuth) {
        FirebaseUser user = firebaseAuth.getCurrentUser();

        if(user != null){
            Toast.makeText(MainActivity.this, "You have
succesfully signed in!", Toast.LENGTH_SHORT).show();
            onSignedInInitilize(user.getDisplayName());
        }else {
            onSignedOutCleanup();
            // Choose authentication providers
            List<AuthUI.IdpConfig> providers = Arrays.asList(
                new
AuthUI.IdpConfig.EmailBuilder().build(),
                new
AuthUI.IdpConfig.GoogleBuilder().build());

            // Create and launch sign-in intent
            startActivityForResult(
                AuthUI.getInstance()
                    .createSignInIntentBuilder()
                    .setIsSmartLockEnabled(false) //to
remember credentials
                    .setAvailableProviders(providers)
                    .build(),
                RC_SIGN_IN);
        }
    }
}
```

```
    }  
};
```

However, it returns from the auth activity after sign in or sign out, we check in this method:

```
@Override  
protected void onActivityResult(int requestCode, int resultCode,  
@Nullable Intent data) {  
    super.onActivityResult(requestCode, resultCode, data);  
  
    if(requestCode == RC_SIGN_IN){  
        Toast.makeText(this, "Signed In!",  
Toast.LENGTH_SHORT).show();  
    } else {  
        Toast.makeText(this, "Signed Out!",  
Toast.LENGTH_SHORT).show();  
        finish();  
    }  
}
```