Firebase Essentials

Android Edition

Firebase Essentials – Android Edition

First Edition

Rev 1.0

eBookFrenzy

Table of Contents

Chapter 1

1. Introduction

In 2014, Google completed the acquisition of a San Francisco-based company named Firebase, Inc. Firebase, Inc. provided a range of developer solutions designed to accelerate the integration of cloud-based features into mobile and web apps. After purchasing the company, Google combined the services provided by Firebase with a number of complementary features previously included as part of the Google Cloud Platform. The combined features from the two platforms are what is now known simply as *Firebase*.

In early 2017, Google acquired Fabric.io from Twitter, Inc. and is now in the process of integrating key Fabric features into Firebase.

This book covers the key features of Android app development using Firebase. Topics covered in this book include user authentication, realtime databases, cloud-based file storage, instant messaging, dynamic links, app indexing, cloud functions, analytics, performance monitoring and more.

The book is organized into chapter groups that focus on specific Firebase features, with each topic area consisting of a detailed overview followed by tutorial style examples that put theory into practice.

1.1 Downloading the Code Samples

The source code and Android Studio project files for the examples contained in this book are available for download at:

http://www.ebookfrenzy.com/direct/firebase_android

The steps to load a project from the code samples into Android Studio are as follows:

1. From the Welcome to Android Studio dialog, select the Open an existing Android Studio project option.

2. In the project selection dialog, navigate to and select the folder containing the project to be imported and click on OK.

3. Before running the app, follow the steps in the corresponding chapter to connect the project to your Firebase account.

1.2 Feedback

We want you to be satisfied with your purchase of this book. If you find any errors in the book, or have any comments, questions or concerns please contact us at feedback@ebookfrenzy.com.

1.3 Errata

While we make every effort to ensure the accuracy of the content of this book, it is inevitable that a book covering a subject area of this size and complexity may include some errors and oversights. Any known issues with the book will be outlined, together with solutions, at the following URL:

http://www.ebookfrenzy.com/errata/firebase-android.html

In the event that you find an error not listed in the errata, please let us know by emailing our technical support team at feedback@ebookfrenzy.com. They are there to help you and will work to resolve any problems you may encounter.

Chapter 2

2. Getting Started with Firebase

Working with Firebase requires a Google account and at least one Firebase project. This chapter will cover the steps that need to be taken in preparation for working with Firebase in the subsequent chapters of the book.

2.1 Signing into the Firebase Console

The first step in working with Firebase involves signing in to the Firebase console. Begin by opening a browser window and navigating to *https://firebase.google.com*, at which point a screen similar to the following will appear:

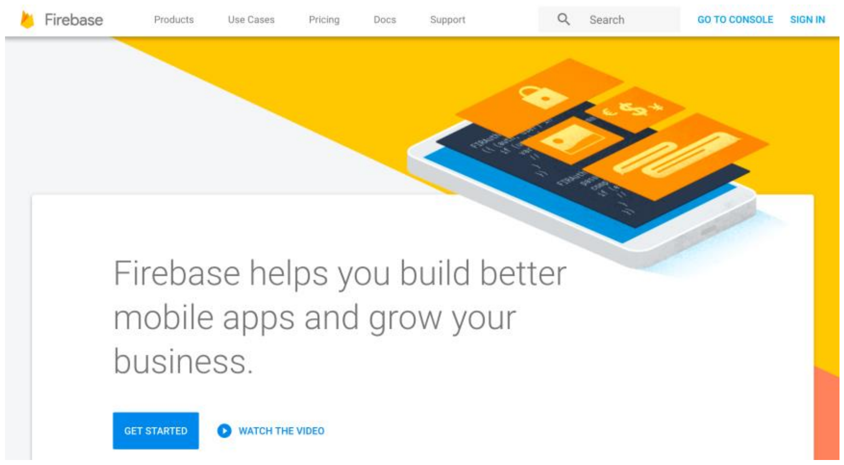


Figure 2-1

Click on either the *Sign In* link in the top right-hand corner, or the *Get Started* button and enter the credentials for your Google account. After signing in, the following screen will appear indicating that no Firebase projects yet exist for the account:

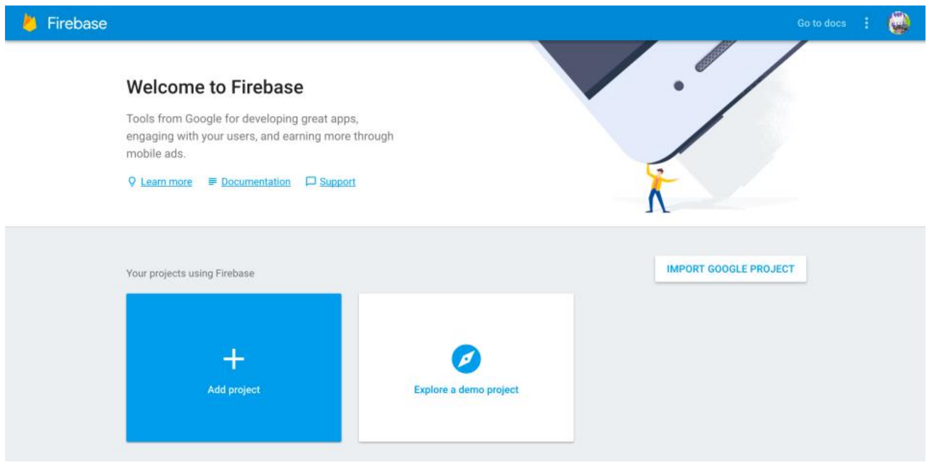
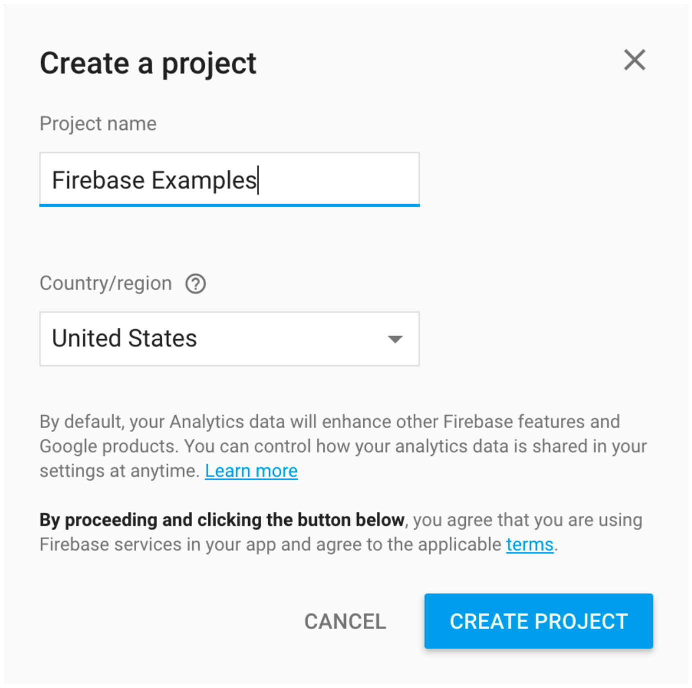

Firebase
Go to docs
Welcome to Firebase
Tools from Google for developing great apps, engaging with your users, and earning more through mobile ads.
Learn more
Documentation
Support
Your projects using Firebase
IMPORT GOOGLE PROJECT
Add project
Explore a demo project

Figure 2-2

2.2 **Creating a New Project**

To create a new project, simply click on the *Add project* square in the console, enter *Firebase Examples* into the Project name field and select your country from the drop down list:



Create a project

Project name: Firebase Examples

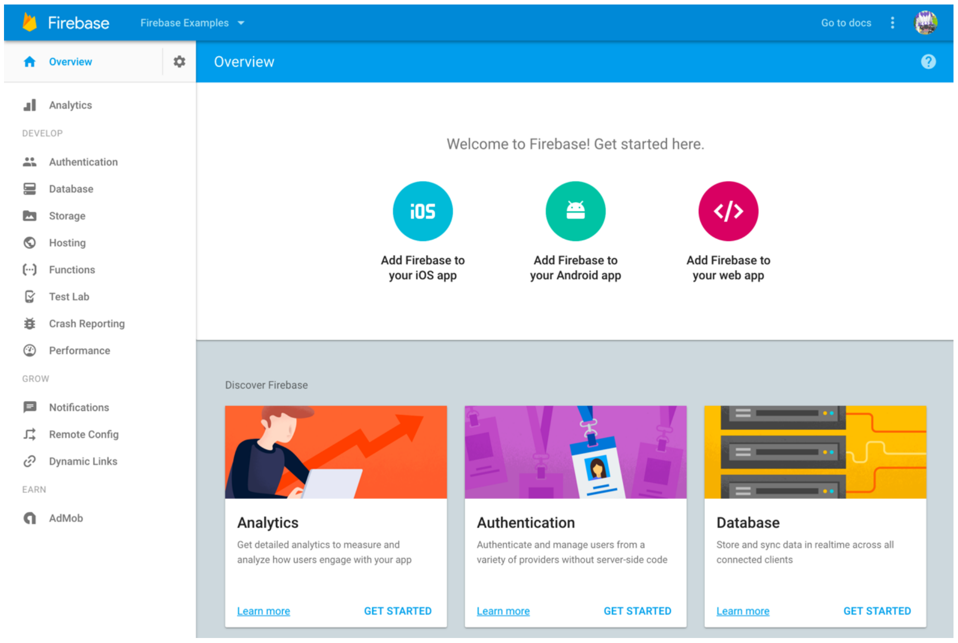
Country/region: United States

By default, your Analytics data will enhance other Firebase features and Google products. You can control how your analytics data is shared in your settings at anytime. Learn more

By proceeding and clicking the button below, you agree that you are using Firebase services in your app and agree to the applicable terms.

Figure 2-3

After completing the form, click on the *Create Project* button. There will be a short delay as the project is created after which the main Firebase console screen will appear as illustrated in Figure 2-4:



Welcome to Firebase! Get started here.

Analytics: Get detailed analytics to measure and analyze how users engage with your app

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

Database: Store and sync data in realtime across all connected clients

Figure 2-4

2.3 **Firebase Projects**

Before moving on to the next chapter, it is important to understand that a Firebase Project is different from an Android Studio project. While an Android Studio project contains a single app, a Firebase project typically contains multiple app projects. Google restricts the number of Firebase projects available to each account, so it is best to group multiple Android Studio app projects within a single Firebase project.

2.4 **Firebase Pricing Plans**

Google offers three pricing tiers for Firebase named Spark, Flame and Blaze, each of which provides increasing levels of database storage and usage quotas. All of the examples contained within this book have been designed to work with the free Spark plan level. Pricing details for the Flame and Blaze plans, which differ by region, can be found online at:

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

All newly created Firebase accounts default to the Spark plan level.

2.5 **Summary**

As will become evident in future chapters, working with Firebase involves a considerable amount of time spent within the Firebase console. This chapter has outlined the process of logging into the console and creating an initial Firebase project.

Before any Firebase functionality can be built into an Android Studio app project, that app project must be associated with a Firebase project. A single Firebase project can contain multiple Android Studio app projects. Since Google limits the number of Firebase projects available per account, a single Firebase project will typically be used to hold multiple app projects.

Firebase is available at three different pricing levels named Spark, Flame and Blaze. The free of charge Spark plan is used exclusively throughout this book.

Chapter 3

3. Firebase User Authentication

Many apps and web services need to provide some form of authentication system in order to identify users, control access to premium content and to protect user data. Without some way to identify one user from another it would also be impossible for the app to know which data and settings belong to which user.

Authentication options can range from requiring an email address and a password to allowing users to sign in using credentials from third-party platforms such as Facebook, Google and Twitter.

Regardless of the motivations for adding user authentication to an app, developers often find that implementation is much more complex than it seems on the surface. Not only must authentication be performed securely and reliably, it must also allow for users to change their account settings, provide support for forgotten passwords and integrate with a range of vastly different third-party authentication APIs. Databases have to be implemented and stored securely and an administration interface developed for manually adding, editing and deleting users.

Fortunately there is an easier option than building all of this infrastructure. All of these requirements can be met with minimal effort by using Firebase Authentication.

3.1 An Overview of Firebase Authentication

Firebase authentication provides a way to add user account creation and sign in capabilities to an app with a minimal amount of coding. Once a user has been authenticated with Firebase, the user is assigned a unique Firebase user ID which can be used when integrating other Firebase services such as data storage and cloud messaging.

Firebase uses the concept of *authentication providers* to facilitate the identification and registration of users. The list of supported Firebase authentication providers currently consists of Google, Facebook, Twitter, GitHub, phone number and email/password authentication. Firebase also provides support for users to sign in anonymously with a temporary account and then subsequently link that account to an authentication provider-based account.

In addition to integrating with the supported authentication providers, Firebase also supports integration with custom authentication systems.

Firebase supports all of the standard authentication features such as handling forgotten passwords and managing user accounts and profiles both programmatically and through the Firebase console.

Two forms of Firebase authentication are available, one involving the use of FirebaseUI Auth and the other a lower level approach using the Firebase SDK. In practice, these involve the use of the following collection of Firebase authentication classes.

3.1.1 FirebaseAuth Instance

Much of the Firebase SDK authentication process involves the use of the FirebaseAuth shared instance. Once a reference to this object has been obtained, it can be used to perform a range of tasks such as creating accounts, signing users in and out and accessing or updating information relating to the current user.

A key function of the FirebaseAuth instance is the authentication state listener (AuthStateListener). When added to the FirebaseAuth instance, it is via this listener that the app receives notification of any changes to the user's authentication status.

Though useful for obtaining user information when using FirebaseUI Auth, the FirebaseAuth instance is primarily used in conjunction with the Firebase SDK approach to authentication.

3.1.2 AuthUI Instance

The AuthUI instance is used extensively in the FirebaseUI Auth authentication process. The class contains a range of methods including a sign-in intent builder and a sign out method. The intent builder method is called to create and configure an Intent object that is then used to launch the FirebaseUI authentication activity. This activity is responsible for all aspects of the user account creation and sign-in process. Configuration options available when using the builder method include changes to the color theme of the sign-in user interface, a logo for branding purposes and a list of the authentication providers that are to be offered as sign-in options to the user.

3.1.3 FirebaseUser Class

The FirebaseUser class is used to encapsulate the profile information for the currently authenticated user. An object of this type is returned, for example, when a call is made to the *getCurrentUser()* method of the FirebaseAuth instance. The data stored in the object will vary depending on which authentication provider is currently being used, but typically includes information such as the user's display name, email address, a URL to a profile photo and the ID of the authentication provider used to sign into the app. Methods are also included for performing tasks such as updating the user's profile information, verifying the user's email address, accessing the user's Firebase user ID and deleting the user's account.

3.1.4 AuthCredential Classes

The AuthCredential class is used to encapsulate user account credentials in a way that is compatible with Firebase. This class is used when exchanging a token from a third-party authentication provider for the credentials of a Firebase account. When a user signs in using the authentication provider for a third-party platform such as Facebook or Twitter, for example, the app is provided with a user token for that platform. Once obtained, this token needs to be passed to Firebase where it is used to create a Firebase account for the user. Before this can take place, however, the third-party provider token must be converted to an AuthCredential object by making a call to the *getCredential()* method of the corresponding authentication provider class. For each authentication provider there is a corresponding AuthCredential subclass:

- EmailAuthCredential
- PhoneAuthCredential
- FacebookAuthCredential
- GithubAuthCredential
- GoogleAuthCredential
- TwitterAuthCredential

3.1.5 Authentication Provider Classes

Each of the authentication providers has its own class that is used during the authentication process (specifically to create an AuthCredential object as outlined above). Firebase currently includes the following authentication provider classes:

- EmailAuthProvider
- PhoneAuthProvider
- FacebookAuthProvider
- GithubAuthProvider
- GoogleAuthProvider
- TwitterAuthProvider

3.2 FirebaseUI Auth Authentication

Of the two Firebase authentication options (namely FirebaseUI Auth and Firebase SDK), FirebaseUI Auth requires by far the least time and programming effort to integrate. In fact, configuring an Android Studio project to support Firebase authentication typically takes longer than writing the actual code to implement FirebaseUI authentication.

FirebaseUI Auth provides everything necessary to implement user authentication including all of the user interface screens that take the user through the account creation and sign-in process.

User authentication can be integrated into an app using FirebaseUI Auth by following a few simple steps:

1. Enable the required authentication providers in the Firebase console.
2. Register the app with the third-party authentication providers for which support is required (Google, Facebook, Twitter and GitHub).
3. Add the FirebaseUI Auth libraries to the Android Studio project.
4. Obtain a reference to the shared FirebaseAuth instance.
5. Use the AuthUI class to configure and build the FirebaseUI authentication intent.
6. Use the intent to launch the FirebaseUI Auth activity.
7. Handle the results returned by the activity.

Each of these steps will be covered in later chapters, beginning with the next chapter entitled *Email/Password Authentication using FirebaseUI Auth*.

3.3 Firebase SDK Authentication

Although integrating authentication using the Firebase SDK is more time consuming, it does have the advantage of flexibility. Unlike FirebaseUI Auth, the Firebase SDK provides full control over the look, feel and behavior of the authentication process (with the exception of any authentication screens presented by third-party authentication providers). In basic terms, Firebase SDK authentication is implemented using the following steps:

1. Enable the required authentication providers in the Firebase console.
2. Register the app with the third-party authentication providers for which support is required (Google, Facebook, Twitter and GitHub).
3. Add the Firebase SDK libraries to the Android Studio project.
4. Obtain a reference to the shared FirebaseAuth instance.
5. Implement and add an AuthStateListener instance to the FirebaseAuth instance and write callback methods.
6. Design the user interface layout for the sign-in screen including options for forgotten passwords.
7. Implement code to handle the account creation, sign-in, sign-out and password reset operations, including adaptations for each of the authentication providers to be supported.
8. Exchange tokens from third-party authentication providers for equivalent Firebase credentials.
9. Handle authentication results within the callback of the AuthStateListener instance.

Beginning with the chapter entitled *Email/Password Authentication using the Firebase SDK*, details on how to perform the above tasks will be covered for the more widely used authentication providers.

3.4 Summary

User authentication is vital both for controlling access to app content and features and protecting user data. Firebase Authentication allows user authentication to be added to an Android app with a minimum amount of time and effort with support for account management and built-in support for a range of popular third-party authentication providers including Google, Twitter and Facebook.

Authentication can be implemented using either FirebaseUI Auth, or the Firebase SDK. FirebaseUI Auth provides a quick and easy way to integrate authentication with minimal effort, while the Firebase SDK approach requires more work but provides greater flexibility.

Chapter 4

4. Email/Password Authentication using FirebaseUI Auth

With the basics of Firebase authentication covered in the previous chapter, this chapter will demonstrate the implementation of email/password based authentication within an Android app using the FirebaseUI Auth API. Subsequent chapters will extend this example to include Firebase user authentication using phone number verification, Google, Facebook and Twitter.

4.1 Creating the Example Project

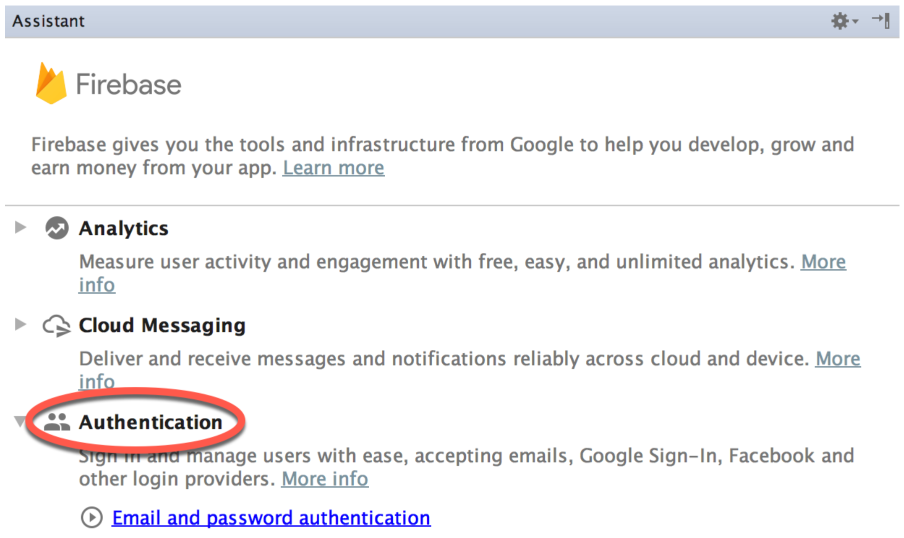
The first step in this exercise is to create the new project. Begin by launching Android Studio and, if necessary, closing any currently open projects using the *File -> Close Project* menu option so that the Welcome screen appears.

Select the *Start a new Android Studio project* quick start option from the welcome screen and, within the new project dialog, enter *FirebaseAuth* into the Application name field and your domain as the Company Domain setting (or make up a fictitious domain for testing purposes) before clicking on the *Next* button.

On the form factors screen, enable the *Phone and Tablet* option and set the minimum SDK to API 16: Android 4.1 (Jellybean). Continue through the project setup screens, requesting the creation of an Empty Activity named *FirebaseAuthActivity* and a corresponding layout named *activity_firebase_auth.*

4.2 Connecting the Project to Firebase

Before an Android Studio project can make use of the Firebase features, it must first be connected to a Firebase project using the Firebase assistant panel. Open the Firebase assistant by selecting the *Tools -> Firebase* menu option. When the Firebase assistant panel appears, locate and select the *Authentication* section as illustrated in Figure 4-1:



Firebase gives you the tools and infrastructure from Google to help you develop, grow and earn money from your app. Learn more

Measure user activity and engagement with free, easy, and unlimited analytics. More info

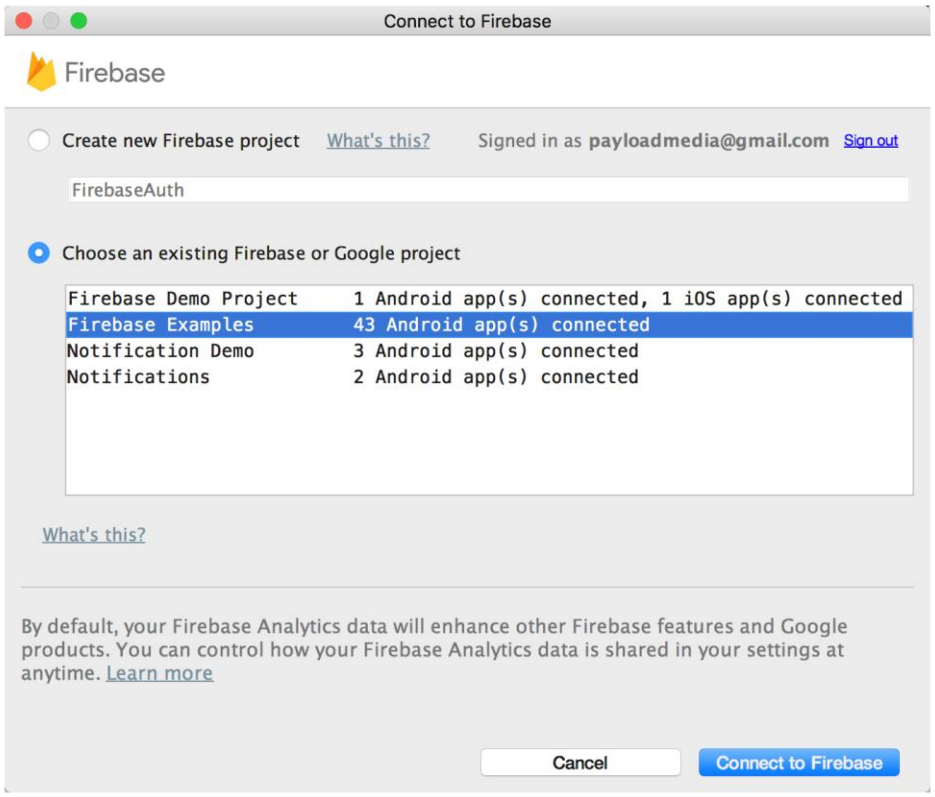
Deliver and receive messages and notifications reliably across cloud and device. More info

Sign in and manage users with ease, accepting emails, Google Sign-In, Facebook and other login providers. More info

Figure 4-1

Click on the *Email and password authentication* link and, in the resulting panel, click on the *Connect to Firebase* button to display the Firebase connection dialog as shown in Figure 4-2 below.

Choose the option to store the app in an existing Firebase project and select the *Firebase Examples* project created in the *Getting Started with Firebase* chapter before clicking on the *Connect to Firebase* button:

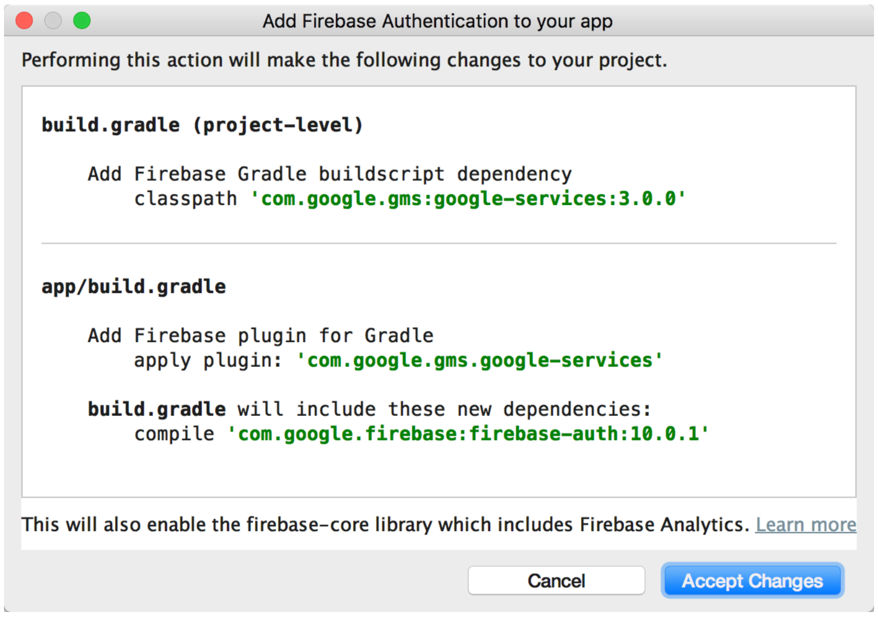

Connect to Firebase
Firebase
Create new Firebase project
What's this?
Signed in as payloadmedia@gmail.com
Sign out
FirebaseAuth
Choose an existing Firebase or Google project
What's this?
Cancel
Connect to Firebase

```
Firebase Demo Project     1 Android app(s) connected, 1 iOS app(s) connected
Firebase Examples         43 Android app(s) connected
Notification Demo         3 Android app(s) connected
Notifications             2 Android app(s) connected
```

By default, your Firebase Analytics data will enhance other Firebase features and Google products. You can control how your Firebase Analytics data is shared in your settings at anytime. Learn more

Figure 4-2

With the project's Firebase connection established, refer to the Firebase assistant panel once again, this time clicking on the *Add Firebase Authentication to your app* button. A dialog will appear (Figure 4-3) outlining the changes that will be made to the project build configuration to enable Firebase authentication:



Performing this action will make the following changes to your project.

```
build.gradle (project-level)

    Add Firebase Gradle buildscript dependency
        classpath 'com.google.gms:google-services:3.0.0'

app/build.gradle

    Add Firebase plugin for Gradle
        apply plugin: 'com.google.gms.google-services'

    build.gradle will include these new dependencies:
        compile 'com.google.firebase:firebase-auth:10.0.1'
```

This will also enable the firebase-core library which includes Firebase Analytics. Learn more

Figure 4-3

Click on the *Accept Changes* button to implement the changes to the project configuration.

After these steps are complete, the FirebaseAuth project will have been added to the *Firebase Examples* project. The core Firebase libraries necessary for adding authentication have also been added to the build configuration. An additional configuration file has also been downloaded and added to the project. This file is named *google-services.json* and is located under *FirebaseAuth -> app* within the project tree structure. To access this file, switch the Project tool window into *Project* mode and navigate to the file as shown in Figure 4-4 below:

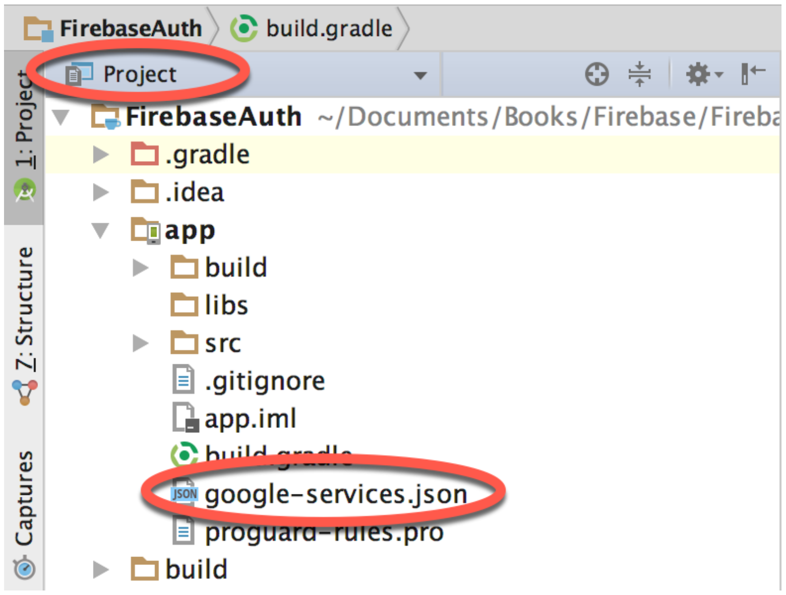


Figure 4-4

A review of the file content will reveal a wide range of Firebase client information that connects the app to the Firebase services including the project, app and client IDs, API keys and certificates. In general there is no need to manually edit this file. In the event that the file is deleted or corrupted, simply reconnect the app to Firebase using the Firebase assistant. Copies may also be downloaded from within the Firebase console.

Return the Project tool window to Android mode before proceeding.

4.3 Adding the FirebaseUI Project Dependencies

Since this project is going to make use of the FirebaseUI Auth API, two more dependencies need to be added to the module level *build.gradle* file for the project. Within the Project tool window, locate and double-click on the *build.gradle (Module: app)* file as shown in Figure 4-5 so that the file loads into the editor:

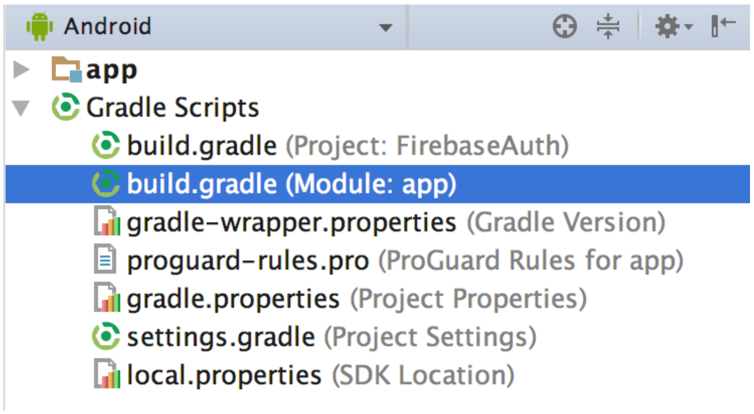

Android
app
Gradle Scripts
build.gradle (Project: FirebaseAuth)
build.gradle (Module: app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

Figure 4-5

Once the Gradle file is open, modify the dependencies section to include the *firebase-ui* and *firebase-ui-auth* libraries (note that more recent versions of these libraries may have been released since this book was published):

```
ependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.3.1'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.google.firebase:firebase-auth:10.0.1'
    compile 'com.firebaseui:firebase-ui:2.0.1'
    compile 'com.firebaseui:firebase-ui-auth:2.0.1'
    testCompile 'junit:junit:4.12'
}
```

In addition to the FirebaseUI dependencies, the build system will also need to download some additional libraries in order to be able to support Twitter-based authentication. These libraries are available via Google's Fabric.io platform and can be integrated into the Gradle build process for the project by adding a repository entry to the module level *build.gradle* file as follows:

```
.
.
.
    compile 'com.firebaseui:firebase-ui:2.0.1'
    compile 'com.firebaseui:firebase-ui-auth:2.0.1'
    testCompile 'junit:junit:4.12'
}

repositories {
    maven { url 'https://maven.fabric.io/public' }
}
.
.
```

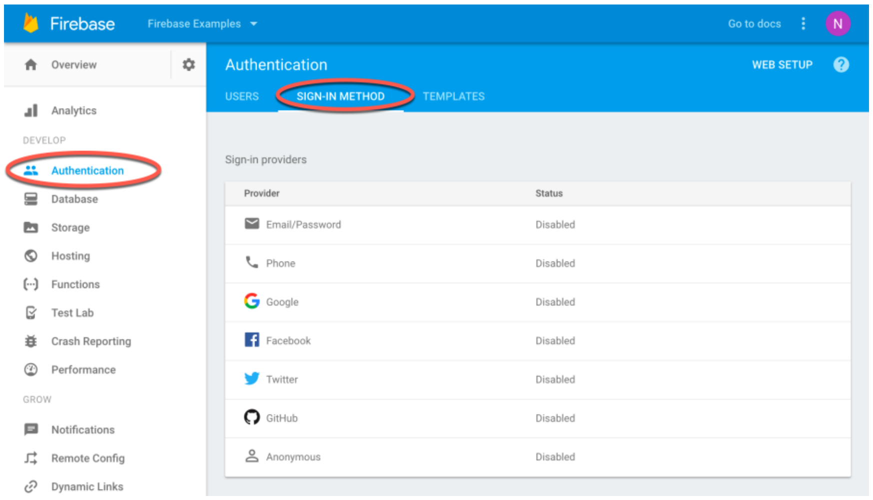
4.4 Enabling Firebase Email/Password Authentication

Now that the Android Studio project has been configured to support Firebase authentication, some authentication settings need to be setup within the Firebase console. Open a browser window and navigate to your Firebase console at:

https://console.firebase.google.com/

Once signed into the Firebase console, select the *Firebase Examples* project from the list of available projects and note that the FirebaseAuth app has been added to the project.

In the left-hand navigation panel, select the *Authentication* option and, in the resulting panel, select the *Sign-In Methods* tab as highlighted in Figure 4-6:

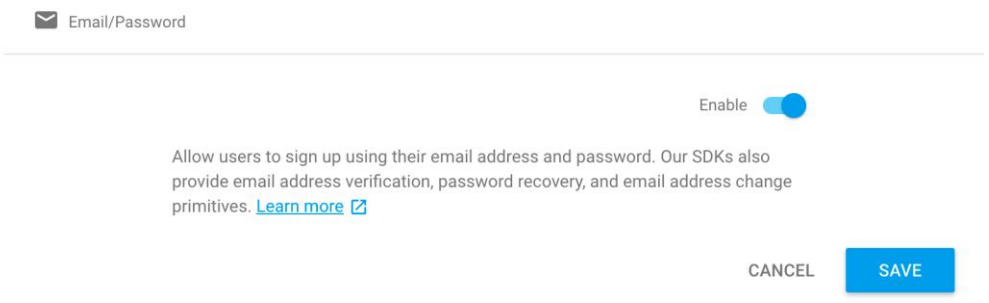


Sign-in providers

Provider	Status
Email/Password	Disabled
Phone	Disabled
Google	Disabled
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Anonymous	Disabled

Figure 4-6

Note that by default none of the sign-in providers are enabled for apps residing within the *Firebase Examples* project. For the purposes of this example, only the Email/Password provider needs to be enabled. Click on the *Email/Password* entry in the providers list and turn on the *Enabled* switch when the settings dialog appears (Figure 4-7). Once the provider is enabled, click on the *Save* button to commit the change.



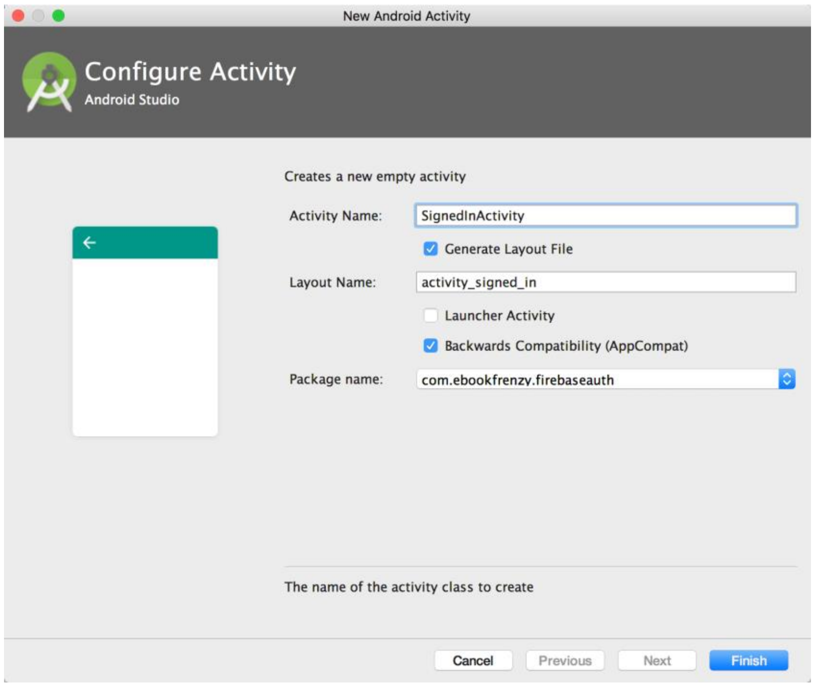
Allow users to sign up using their email address and password. Our SDKs also provide email address verification, password recovery, and email address change primitives. Learn more

Figure 4-7

4.5 **Adding the Signed-In Activity**

The purpose of adding authentication to an app is to withhold access to functionality or content until a user's identity has been established and verified. This means that areas of the app need to be inaccessible to the user until the authentication process has successfully completed. In this example project, the restricted area of the app will be represented by a second activity which is only presented after the user has signed in to the app.

Add this second activity to the project now by locating the *app -> java -> <yourdomain>.firebaseauth* entry in the Project tool window and right-clicking on it. When the menu appears, select the *New -> Activity -> Empty Activity* menu option. In the *New Android Activity* dialog (Figure 4-8), name the activity *SignedInActivity*:



Creates a new empty activity

Activity Name: SignedInActivity

[✓] Generate Layout File

Layout Name: activity_signed_in

[ ] Launcher Activity

[✓] Backwards Compatibility (AppCompat)

Package name: com.ebookfrenzy.firebaseauth

The name of the activity class to create

Figure 4-8

Before clicking on the *Finished* button, make sure that the *Generate Layout File* option is enabled and the *Launcher Activity* option disabled.

4.6 **Designing the SignedInActivity Layout**

The user interface layout for the SignedInActivity is going to need to be able to display the user's profile photo, email address and display name and also provide buttons for signing out of and deleting the account. Begin the design process by loading the *activity_signed_in.xml* file into the layout editor, turning off Autoconnect mode and adding and configuring widgets so that the layout matches Figure 4-9. When Android Studio requests an image to display on the ImageView, select the *common_google_signin_btn_icon_dark* icon from the list.

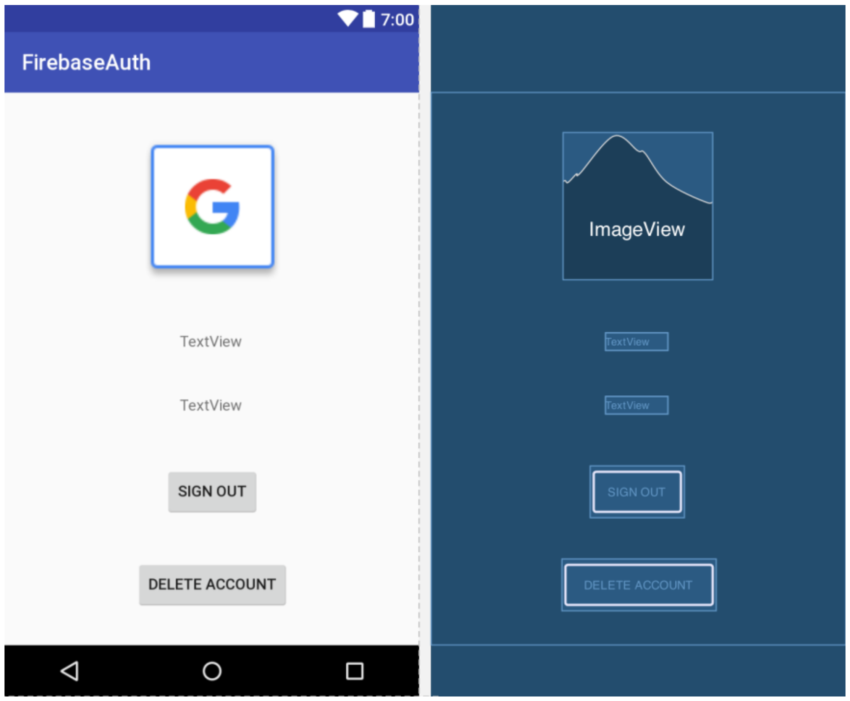


Figure 4-9

Select all five widgets, right-click on the ImageView and select *Center Horizontally* from the popup menu. Right-click on the ImageView again, this time selecting the *Center Vertically* menu option. The widgets should now be centered horizontally and constrained using a vertical chain configuration.

Using the Property tool window, change the IDs for the ImageView and two TextView widgets to *imageView*, *email* and *displayname* respectively. Also configure the two buttons to call onClick methods named *signOut* and *deleteAccoun*t.

4.7 **Firebase Initialization Steps**

Each time the app is launched, it will need to obtain an instance of the FirebaseAuth object and then check to verify whether the current user is already signed in to the app. If the user is already signed in, the app simply needs to launch SignedInActivity so that the user can interact with the app. In the event that the user has yet to sign in, the app will need to initiate the sign-in process.

Locate the FirebaseAuthActivty class within the Project tool window (*app -> java -> <yourdomain>.firebaseauth -> FirebaseAuthActivity*) and double click on it to load the Java file into the editor. Once loaded, modify the code so that it reads as follows:

```
package com.ebookfrenzy.firebaseauth;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

import com.google.firebase.auth.FirebaseAuth;

public class FirebaseAuthActivity extends AppCompatActivity {
    private FirebaseAuth auth;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_firebase_auth);

        auth = FirebaseAuth.getInstance();

        if (auth.getCurrentUser() != null) {
            startActivity(new Intent(this, SignedInActivity.class));
            finish();
        } else {
            authenticateUser();
        }
    }
}
```

The code has been implemented such that a method named *authenticateUser()* is called if the user is not currently signed in. Clearly the next step is to implement this method.

4.8 Initiating the Sign-in Process

It is the responsibility of the *authenticateUser()* method to present the user with options to sign into the app using existing email and password credentials, or to facilitate the creation of a new account if the user does not already have one. The authentication process also needs to take steps to verify that valid information has been entered and to allow the user to retrieve and reset a lost or forgotten password. In this example, all of these tasks will be performed for us by FirebaseUI Auth.

The code within the *authenticateUser()* method is going to use the *createSignInIntentBuilder()* method of the Firebase AuthUI instance to create a new intent object configured to meet the authentication requirements of our app. This intent will then be started using the *startActivityForResult()* method together with a request code that will be referenced when handling the activity result.

Remaining within the *FirebaseAuthActivity.java* file, implement the *authenticateUser()* method as follows:

```
package com.ebookfrenzy.firebaseauth;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

import com.firebase.ui.auth.AuthUI;

import com.firebase.ui.auth.ErrorCodes;
import com.firebase.ui.auth.IdpResponse;
import com.firebase.ui.auth.ResultCodes;
import com.google.firebase.auth.FirebaseAuth;

import java.util.ArrayList;
import java.util.List;
```

```
public class FirebaseAuthActivity extends AppCompatActivity {

    private FirebaseAuth auth;
    private static final int REQUEST_CODE = 101;

    private void authenticateUser() {
        startActivityForResult(
                AuthUI.getInstance().createSignInIntentBuilder()
                        .setAvailableProviders(getProviderList())
                        .setIsSmartLockEnabled(false)
                        .build(),
                REQUEST_CODE);
    }
.
.
}
```

In the above code, the list of sign-in providers to be supported by the app is obtained via a call to a method named *getProviderList()*. This method now needs to be implemented within the FirebaseAuthActivity class:

```
private List<AuthUI.IdpConfig> getProviderList() {

    List<AuthUI.IdpConfig> providers = new ArrayList<>();

    providers.add(
       new AuthUI.IdpConfig.Builder(AuthUI.EMAIL_PROVIDER).build());

    return providers;
}
```

For this chapter, only the email/password provider is needed, though additional providers such as Google, Facebook and Twitter will be added to the list in subsequent chapters.

4.9 Handling the Sign In Activity Result

The next task is to add some code to handle the result of the sign-in process after control is returned to the app from the AuthUI activity. When an activity is launched using the *startActivityForResult()* method, the result is passed to the originating activity via a call to the *onActivityResult()* method if it has been implemented. This method is passed the request code originally included in the start request (represented by the value assigned to the REQUEST_CODE constant in this example) together with a result code indicating whether the activity was successful.

With the *FirebaseAuthActivity.java* file still in the editor, add the *onActivityResult()* method to the class so that it reads as follows:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    IdpResponse response = IdpResponse.fromResultIntent(data);

    if (requestCode == REQUEST_CODE) {
        if (resultCode == ResultCodes.OK) {
            startActivity(new Intent(this, SignedInActivity.class));
            return;
        }
    } else {
        if (response == null) {
            // User cancelled Sign-in
            return;
        }

        if (response.getErrorCode() == ErrorCodes.NO_NETWORK) {
            // Device has no network connection
            return;
        }

        if (response.getErrorCode() == ErrorCodes.UNKNOWN_ERROR) {
            // Unknown error occurred
            return;
        }
    }
}
```

The method begins by calling the superclass, then checks that the request code matches that referenced when the activity was started. This step is recommended to ensure that the handler is not being called in response to any other activities that may have been launched by the app. The response code is then extracted from the data, evaluated and, in the event that the sign-in process was successful, the SignedInActivity is launched giving the user access to the previously restricted area of the app. Tests have also been added to detect when the user cancelled the sign-in process or when the device does not have an active network connection

4.10 **Handling the Sign-in**

When the user has successfully signed in, the SignedInActivity activity is launched. For the purposes of this tutorial, this activity will simply display a subset of the information known about the user and the sign-in provider. The SignedInActivity layout is also designed to display the profile picture associated with the user's account if one is available, though this will only be used in later chapters when Google, Facebook and Twitter sign-in support is added to the project. The layout also contains a button that allows the user to sign out of the app.

When the SignedInActivity activity starts, the FirebaseAuth object can be used to obtain a reference to the FirebaseUser object for the currently signed in user. It is from this object that details about the current user and sign-in provider can be obtained.

Locate the *SignedInActivity.java* class file within the project tool window and double click on it to load it into the code editor. Within the editor, modify the existing *onCreate()* method as follows:

```
package com.ebookfrenzy.firebaseauth;

import android.content.Intent;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

import com.firebase.ui.auth.AuthUI;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;

public class SignedInActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_signed_in);

        FirebaseUser currentUser =
                FirebaseAuth.getInstance().getCurrentUser();

        if (currentUser == null) {
            startActivity(new Intent(this, FirebaseAuthActivity.class));
            finish();
            return;
        }

        TextView email = (TextView) findViewById(R.id.email);
        TextView displayname = (TextView) findViewById(R.id.displayname);

        email.setText(currentUser.getEmail());
        displayname.setText(currentUser.getDisplayName());
    }
.
.
}
```

The code added to the *onCreate()* method performs a check to verify that the user is actually signed in and, if not, returns control to the original activity where the sign-in process will repeat. If, on the other hand, the user is signed in, a reference to the FirebaseUser instance for that user is obtained and used to access the email and display name. These values are then displayed on the TextView widgets in the user interface layout.

4.11 **Signing Out**

When the user interface layout for the SignedInActivity activity was designed earlier in this chapter, it included a button intended to allow the user to sign out of the app. This button was configured to call a method named *signOut()* when tapped by the user. Remaining in the *SignedInActivity.java* file, implement this method now so that it reads as follows:

```
public void signOut(View view) {
    AuthUI.getInstance()
            .signOut(this)
            .addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()) {
                        startActivity(new Intent(
                                SignedInActivity.this,
                                FirebaseAuthActivity.class));
                        finish();
                    } else {
                        // Report error to user
                    }
                }
            });
}
```

The method begins by calling the *signOut()* method of the AuthUI instance. The sign out process is handled asynchronously, so a completion handler is provided which will be called once the sign out is completed. In the case of a successful sign out, the user is returned to the main sign-in activity.

4.12 **User Account Removal**

The final task to be performed before testing the app is to implement the code for the Delete User button in the SignedInActivity layout. When added, this button was configured to call a method named *deleteAccount()* when tapped. The code for this method reads as follows and should now be added to the *SignedInActivity.java* file:

```
public void deleteAccount(View view) {
    AuthUI.getInstance()
            .delete(this)
            .addOnCompleteListener(new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) {
                    if (task.isSuccessful()) {
                        startActivity(new Intent(SignedInActivity.this,
                                 FirebaseAuthActivity.class));
                        finish();
                    } else {
                        // Notify user of error
                    }
                }
            });
}
```

As with the signing out process, the method begins by obtaining a reference to the AuthUI instance, this time making a call to the *delete()* method. The deletion process is, once again, an asynchronous task, requiring the provision of a completion handler to be called when the task completes. On a successful account deletion, the user is returned to the main activity.

With this task complete the example app is ready to be tested, an area that is covered in the next chapter entitled *Testing and Customizing FirebaseUI Auth Authentication*.

4.13 **Summary**

This chapter has provided a detailed tutorial to implementing email/password based authentication within an Android app using FirebaseUI Auth. FirebaseUI Auth provides a comprehensive, pre-built user authentication system including user interface and backend functionality that reduces the amount of code that needs to be written by the developer.

This authentication solution requires that a basic sequence of steps be performed consisting of connecting the Android Studio project to Firebase, adding Firebase build dependencies and enabling authentication for the project within the Firebase console. Once the configuration tasks are completed, the authentication process is initiated within the code of the Android app and listeners implemented to respond to results of the authentication process.

The next chapter, entitled *Testing and Customizing FirebaseUI Auth Authentication*, will focus on testing the app developed in this chapter and introduce some of the options available for customizing the user sign-in experience when using FirebaseUI Auth.

Chapter 5

5. Testing and Customizing FirebaseUI Auth Authentication

The previous chapter worked through the development of an Android app that uses the FirebaseUI Auth API to implement email/password based user authentication. At the end of that chapter, the app was largely complete and ready to be tested. This chapter will work through the testing of the app and, in doing so, demonstrate both the account creation and sign-in features of FirebaseUI Auth and the user account management options within the Firebase console.

Once testing of the app is complete, this chapter will also introduce some of the options that are available to customize the appearance and behavior of the FirebaseUI Auth interface.

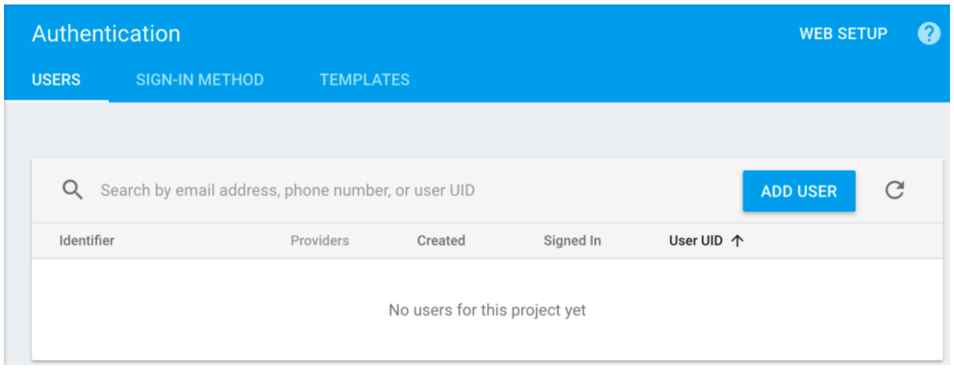
5.1 Authentication Testing Environment

The Firebase authentication features can be tested on either a physical device or a suitably configured emulator. In particular, the emulator must be running a version of Android that includes the Google APIs.

5.2 Firebase Console User Management

Before testing the FirebaseAuth app it is worth gaining some familiarity with the user management features located in the Authentication section of the Firebase console.

Log into the Firebase console and select the *Firebase Examples* project followed by the *Authentication* option located in the left-hand navigation panel. Within the Authentication screen, select the Users tab as illustrated in Figure 5-1:



No users for this project yet

Figure 5-1

Note that there are currently no users associated with this project. As users are added from within the app, they will be listed within this panel. A button is also provided for the manual addition of users. The list may be refreshed at any time by clicking on the reload button located to the right of the *Add User* button.

5.3 Testing the FirebaseAuth App

Compile and run the FirebaseAuth app, either on a physical device or a simulator session. Once the app has loaded, it will immediately prompt for the user to enter an email address as shown in Figure 5-2:

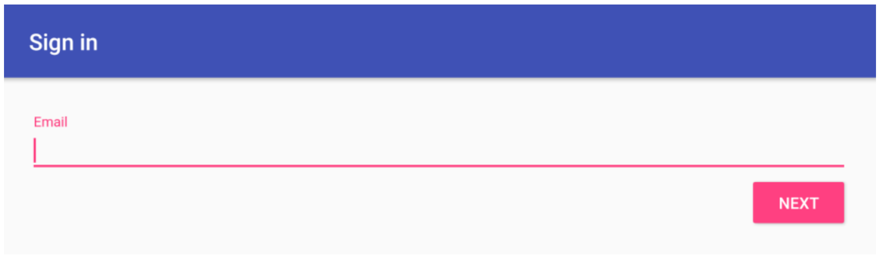


Figure 5-2

Start by entering an invalid email address (for example one containing spaces) and selecting the *Next* button. Note that the Firebase authentication system reports that the email address is not correctly formatted.

Enter your email address into the Email field and tap the *Next* button. When a valid email address is entered, the authentication process contacts Firebase to find out if the email address is already stored in the list of users for the *Firebase Examples* project. Since the email address is not yet associated with any existing users, the account creation screen (Figure 5-3) is displayed:

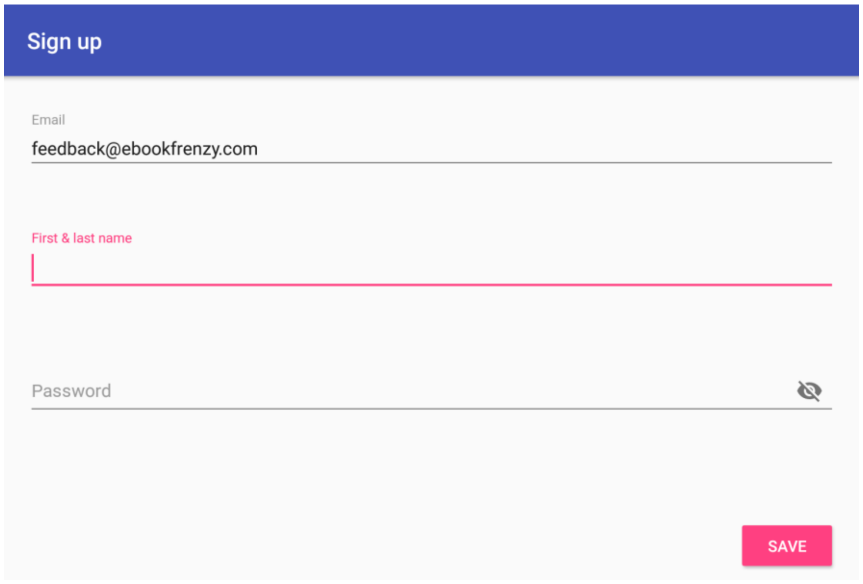


Figure 5-3

Enter your name and specify a password to be associated with the account before tapping the *Save* button. The account creation screen will insist on a password of at least 6 characters in length and provides the option to display the password by tapping the eye icon in the far right of the password field.

Once the name and password information have been saved, the account will be created and the user signed in, causing the SignedInActivity screen to appear displaying the email address and name associated with the newly created account.

Return to the browser window displaying the list of authenticated users and click on the refresh button. Once the list has updated the newly added account should be listed. This entry in the list will include the email address, an envelope icon indicating that this is an email address/password based account, the creation and last sign-in date and the user's unique identifier (UID). Moving the mouse pointer over the list entry will cause additional options to appear. The button containing three vertically arranged dots will, when clicked, present a menu of account management options as illustrated in Figure 5-4:

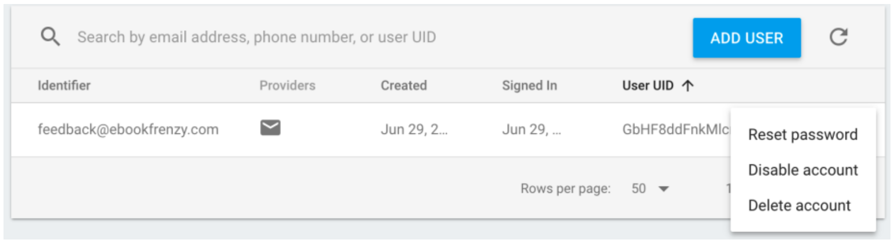
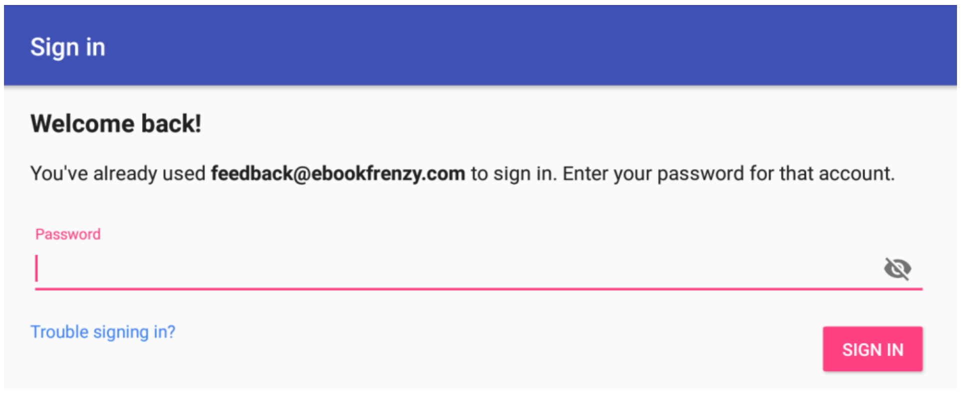


Figure 5-4

A deleted account cannot be subsequently restored, though a disabled account can be re-enabled at any time using the same menu. Requesting a password reset will result in a password reset email containing a link allowing the user to reset the password being sent to the email address associated with the account.

Within the running FirebaseAuth app, tap the *Sign Out* button to return to the original sign-in screen and enter the same email address before tapping the *Next* button. This time, Firebase will discover that an account associated with the email address exists and request the corresponding password:


Sign in

Welcome back!

You've already used **feedback@ebookfrenzy.com** to sign in. Enter your password for that account.

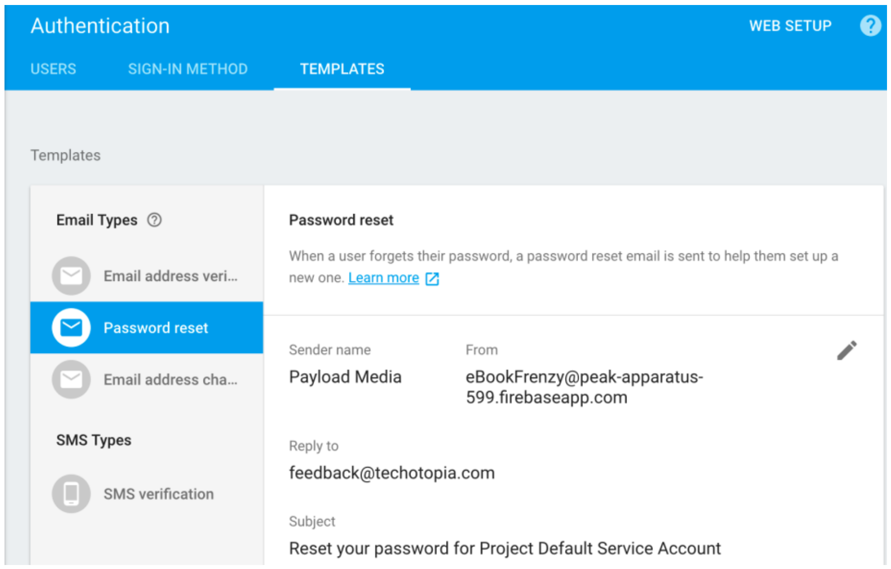
Figure 5-5

As can be seen in Figure 5-5 above, the password screen includes a *Trouble signing in?* link which, when selected will provide the user with the option to receive the same password reset email as that sent from within the Firebase console.

Enter the password for your account and wait to be logged into the SignedInActivity screen.

5.4 Customizing the Password Reset Email Message

Firebase provides three email templates for messages that will be sent to users for email address verification, password reset and email address change. These templates are customizable and accessible from the *Templates* tab within the Authentication screen of the Firebase console. Figure 5-6, for example, shows the standard message template that is sent when a user requests a password reset email:



Password reset

When a user forgets their password, a password reset email is sent to help them set up a new one. Learn more

Sender name: Payload Media

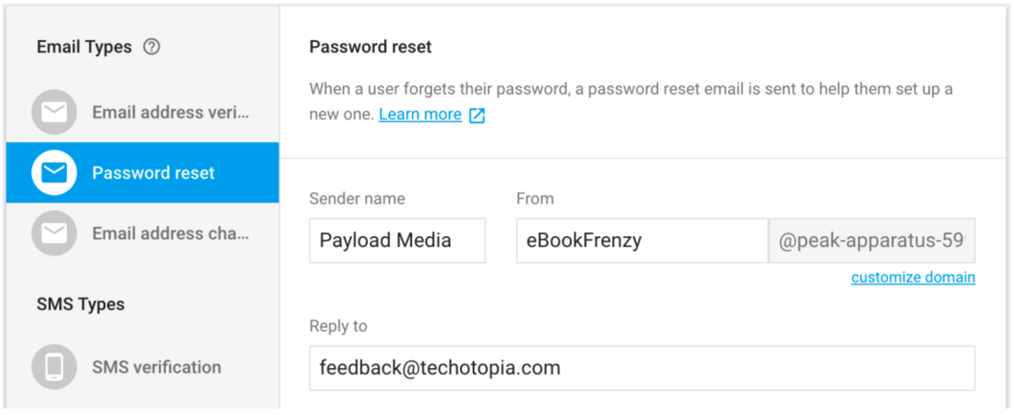
From: eBookFrenzy@peak-apparatus-599.firebaseapp.com

Reply to: feedback@techotopia.com

Subject: Reset your password for Project Default Service Account

Figure 5-6

Click on the pencil icon to edit the settings for the message. By default, the email message will use the specified subject and title text. There will be no reply-to email address, no sender name and the message will appear to have been sent by *noreply@<app-identifier>.firebaseapp.com*. The email address settings can be changed within the top section of the template editing screen shown in Figure 5-7:



Password reset

When a user forgets their password, a password reset email is sent to help them set up a new one. Learn more

Sender name: Payload Media

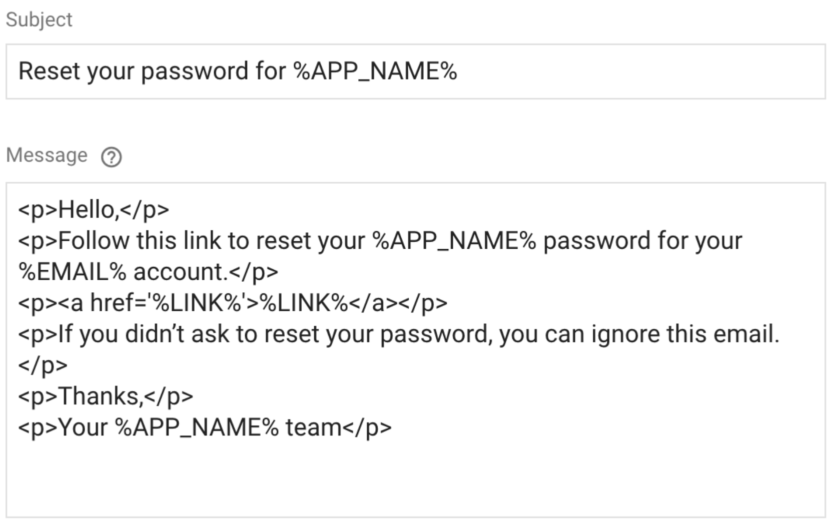
From: eBookFrenzy @peak-apparatus-59

Reply to: feedback@techotopia.com

Figure 5-7

To change the domain containing the from email address it will be necessary to prove ownership of the domain. Click on the *customize domain* link, enter the URL of your web site domain, click on the *Continue* and edit the settings for the domain's DNS records as instructed. After the changes have been made click on the *Verify* button. It may take up to 48 hours before the DNS settings take effect and the domain is successfully verified.

The middle section of the screen allows the subject and message content to be customized:



Subject

Reset your password for %APP_NAME%

Message

```
<p>Hello,</p>
<p>Follow this link to reset your %APP_NAME% password for your
%EMAIL% account.</p>
<p><a href='%LINK%'>%LINK%</a></p>
<p>If you didn't ask to reset your password, you can ignore this email.
</p>
<p>Thanks,</p>
<p>Your %APP_NAME% team</p>
```

Figure 5-8

Both the subject and message content allow the use of the following placeholder variables which are automatically replaced by the corresponding values when the message is sent to the user:

- **%DISPLAY_NAME%** - The user's first name.
- **%APP_NAME%** - The name of the app for which the user is requesting a password reset.
- **%LINK%** - The link on which the user clicks to initiate the password change.
- **%EMAIL%** - The user's email address.
- **%NEW_EMAIL%** - Used only when the user is performing an email address change, this is replaced by the new email address.

The name used to replace the %APP_NAME% placeholder is set by default to "Project Default Service Account". To change this, click on the cog icon to the right of the *Firebase Examples* project name in the upper left-hand corner of the Firebase console and select the *Project Settings* option (highlighted in Figure 5-9):

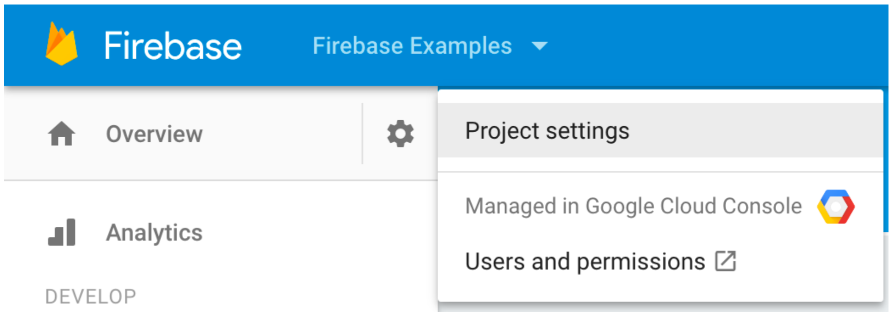
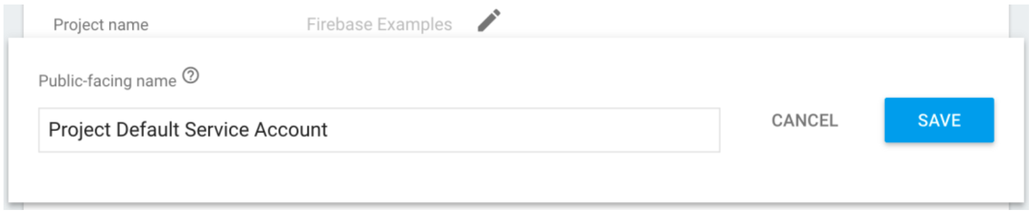


Figure 5-9

On the General screen of the Settings panel, click on the pencil icon to the right of the *Public-facing name* value, enter the new name into the resulting dialog and click *Save* to commit the change:

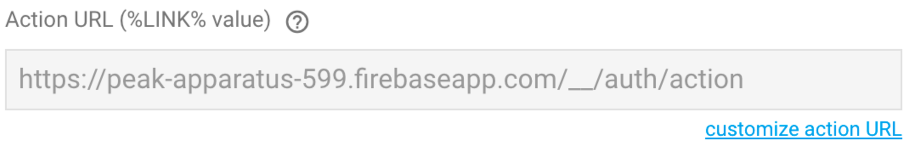

Project name Firebase Examples

Public-facing name

Project Default Service Account

Figure 5-10

The final customization option (Figure 5-11) allows the password URL to be changed (also referred to as the action URL) to reference your own server instead of letting Firebase handle the password change. When a custom URL has been specified, this URL will be substituted for the %LINK% placeholder within the email message, with two additional parameters (mode and oobCode) appended to the end:


Action URL (%LINK% value)

https://peak-apparatus-599.firebaseapp.com/__/auth/action

Figure 5-11

5.5 Testing the Password Reset

Take some time to customize the password reset email template, then return to the running FirebaseAuth app. Tap the *Sign Out* button to return to the sign-in screen, re-enter your email address and click *Next*. On the password screen, select the *Trouble signing in?* link and follow the steps to send a password reset email. When the email arrives in your inbox, review it to make sure it matches the changes before clicking on the provided link to change the password. The reset link may only be clicked once and expires shortly after the email has been sent to the user.

5.6 Enabling Smart Lock for Passwords

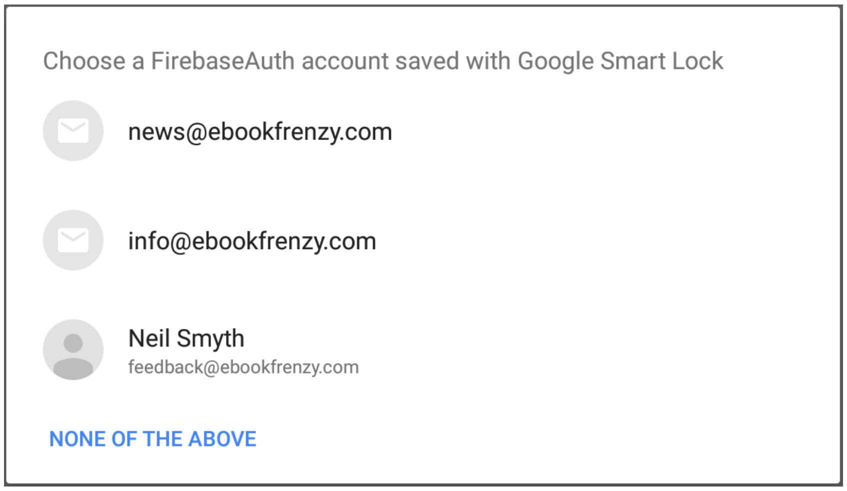
So far, the example app has not made use of Smart Lock for Passwords. Smart Lock is the technology behind the Google Chrome browser feature that remembers login and password information for the web sites visited by a user. This feature, when enabled, is also available to Android apps using Firebase authentication.

Smart Lock for Passwords allows the authentication system to present a list of credentials previously stored on the device by the user using Smart Lock. The user simply selects a previously stored identity to sign into or sign up for access to the app.

Smart Lock is enabled when calling the *createSignInIntentBuilder()* method on the AuthUI instance. To enable Smart Lock for the FirebaseAuth project, edit the *FirebaseAuthActivity.java* file, locate the *authenticateUser()* method and modify the *setIsSmartLockEnabled()* method to pass through a true value:

```
private void authenticateUser() {
    startActivityForResult(
            AuthUI.getInstance().createSignInIntentBuilder()
                    .setAvailableProviders(getProviderList())
                    .setIsSmartLockEnabled(true)
                    .build(),
            REQUEST_CODE);
}
```

When the app is run on a device, the user will be presented with a list of identities that have been previously saved to the user's Google account:


Choose a FirebaseAuth account saved with Google Smart Lock

news@ebookfrenzy.com

info@ebookfrenzy.com

Neil Smyth
feedback@ebookfrenzy.com

NONE OF THE ABOVE

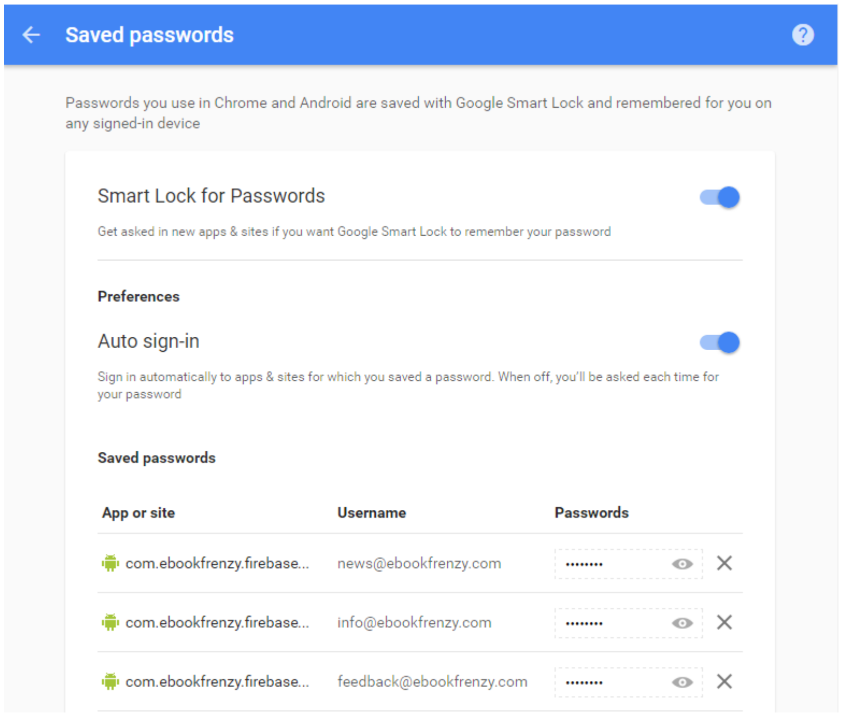
Figure 5-12

If an identity is selected from the list for which an account has already been created for the Firebase project, and for which Smart Lock has been used to save the password, the user will be directly logged into the device. If no account yet exists, the user will be required to create an account before proceeding.

The identities associated with the user's Google account may be reviewed and edited at any time by visiting the following URL:

https://passwords.google.com

Figure 5-13, for example, shows the Google Smart Lock entries for the FirebaseAuth app as stored by Google Smart Lock for passwords:


Saved passwords

Passwords you use in Chrome and Android are saved with Google Smart Lock and remembered for you on any signed-in device

Smart Lock for Passwords

Get asked in new apps & sites if you want Google Smart Lock to remember your password

Preferences

Auto sign-in

Sign in automatically to apps & sites for which you saved a password. When off, you'll be asked each time for your password

Saved passwords

App or site	Username	Passwords
com.ebookfrenzy.firebase...	news@ebookfrenzy.com	••••••••
com.ebookfrenzy.firebase...	info@ebookfrenzy.com	••••••••
com.ebookfrenzy.firebase...	feedback@ebookfrenzy.com	••••••••

Figure 5-13

5.7 Color Customization

A different color theme may be specified for the FirebaseUI Auth account creation and sign-in screens. This is particularly useful when the screens are required to match the overall branding or color theme of the app. Where possible, it is recommended that the color selections chosen conform to the Material design guidelines.

Material design was created by the Android team at Google and dictates that the elements that make up the user interface of Android and the apps that run on it appear and behave in a certain way in terms of behavior, shadowing, animation and style. Details on Material color styles, color palettes and schemes can be found online at the following URL:

https://material.google.com/style/color.html

The referenced web page also includes a palette of colors that have been chosen to work together within the theme of an Android app.

Declaring a custom theme for the Firebase authentication UI is a two-step process which begins with adding a new style to the project's *styles.xml* file located under *app -> res -> values* within the project tool window:

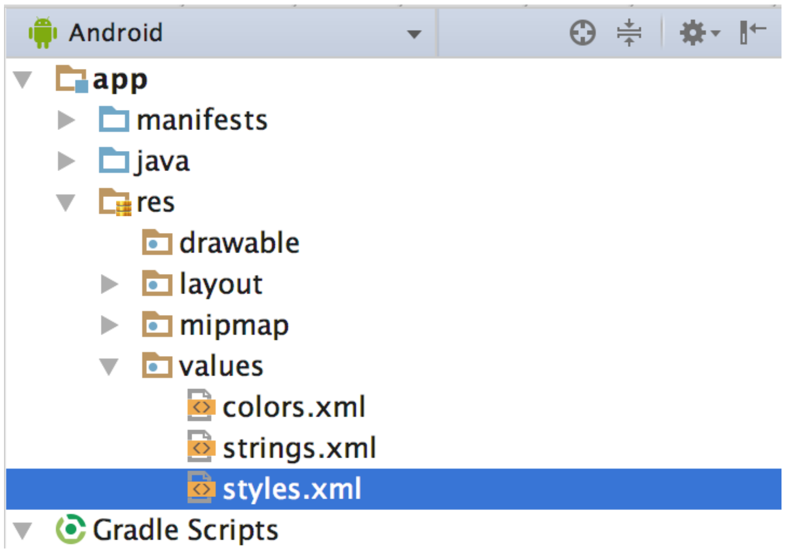


Figure 5-14

Within the FirebaseAuth project, edit this *styles.xml* file and add the following custom theme entry:

```
<style name="CustomTheme" parent="FirebaseUI">
    <item name="colorPrimary">@color/material_green_600</item>
    <item name="colorPrimaryDark">@color/material_green_700</item>
    <item name="colorAccent">@color/material_purple_a700</item>
    <item name="colorControlNormal">@color/material_green_500</item>
    <item name="colorControlActivated">@color/material_lime_a800</item>
    <item name="colorControlHighlight">@color/material_green_a200</item>
    <item name="android:windowBackground">@color/material_lime_50</item>
    <item name="colorButtonNormal">@color/material_purple_500</item>
</style>
```

The color values referenced above were all taken from the Material color palette. The next step is to declare the colors referenced in the above style. These need to be declared in *colors.xml* file located under *app -> res -> values*. Edit this file and add the following color entries:

```
<color name="material_green_600">#43A047</color>
<color name="material_green_500">#4CAF50</color>
<color name="material_green_700">#388E3C</color>
<color name="material_green_a200">#69F0AE</color>
<color name="material_lime_a800">#AFB42B</color>
<color name="material_lime_50">#F9FBE7</color>
<color name="material_purple_a700">#AA00FF</color>
<color name="material_purple_500">#9C27B0</color>
```

Once the style and colors have been declared, the new theme needs to be referenced during the building of the sign-in intend via a call to the *setTheme()* method. Edit the *FirebaseAuthActivity.java* file, locate the *authenticateUser()* method and modify it to change the theme:

```
private void authenticateUser() {
    startActivityForResult(
            AuthUI.getInstance().createSignInIntentBuilder()
                    .setTheme(R.style.CustomTheme)
                    .setAvailableProviders(getProviderList())
                    .setIsSmartLockEnabled(true)
                    .build(),
            REQUEST_CODE);
}
```

Compile and run the app and verify that the sign-in and account creation screens have adopted the specified theme. To gain a familiarity with the different color settings, experiment with different colors within the theme using the Material color palette as a guideline.

5.8 **Specifying a Logo**

In addition to changing the color theme, FirebaseUI Auth also allows a logo to be specified for presentation during the sign-in process. For this example, the *firelogo.png* image file contained within the *project_images* folder of the sample code download will be used. If you have not already downloaded the code samples, the archive can be downloaded using the following link:

http://www.ebookfrenzy.com/direct/firebase_android

Locate the *firelogo.png* image in the file system navigator for your operating system and copy the image file. Right-click on the *app -> res -> drawable* entry in the Project tool window and select Paste from the menu to add the file to the folder. When the copy dialog appears, click on OK to accept the default settings:

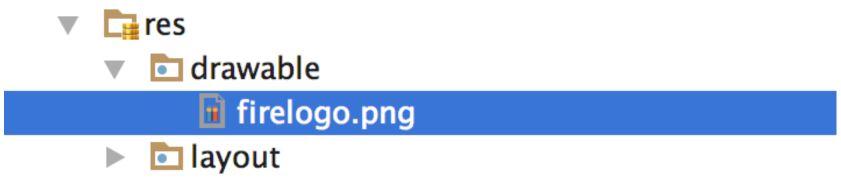


Figure 5-15

With the logo added to the project, all that remains is to include it when the sign-in intent is constructed. In this case, the logo is added via a call to the *setLogo()* method as outlined in the following listing:

```
private void authenticateUser() {
    startActivityForResult(
            AuthUI.getInstance().createSignInIntentBuilder()
```

```
                    .setTheme(R.style.CustomTheme)
                    .setLogo(R.drawable.firelogo)
                    .setAvailableProviders(getProviderList())
                    .setIsSmartLockEnabled(true)
                    .build(),
            REQUEST_CODE);
}
```

Although the logo has been included in the activity, it would not be visible were the app to be run now. This is because the logo only appears when the authentication provider selection screen is displayed. Since the app is currently configured to use only one provider in the form of email and password authentication, this screen is not yet used. The logo will, however, become visible during the next chapter (entitled *Google Sign-In Authentication using FirebaseUI Auth*) when another authentication provider is added to the project.

5.9 **Deleting the Test Account**

The final area to test is the deletion of the user's account. The SignedInActivity screen contains a button configured to delete the current user from the list of registered users. Verify that the account deletion code works by launching the FirebaseAuth app and signing in using the previously registered account credentials. Once the SignedInActivity screen appears, tap the Delete Account button. The user will be signed out of the app and the account deleted. Open the Firebase console in a browser window, navigate to the Authentication page for the *Firebase Examples* project and verify that the account is no longer listed on the Users screen.

5.10 **Summary**

In the previous chapter, an example project was created that used the FirebaseUI Auth system to implement email and password based authentication. This chapter has performed a sequence of tests intended not only to verify that the code works but also to outline many of the features that are provided by default when using Firebase Authentication. This chapter has also explored the various ways in which the Firebase authentication process can be configured and customized, including the use of Smart Lock, customizing the password reset email message, changing color themes and adding a logo.

Chapter 6

6. Google Sign-In Authentication using FirebaseUI Auth

The next area of Firebase authentication to cover involves the use of the Google Sign-in provider. As with the email and password provider, Google Sign-in uses an email address and a password to authenticate a user. In this case, however, the user’s existing Google account credentials are used to create the user’s account.

In this chapter, the FirebaseAuth project will be extended to include Google Sign-in authentication.

6.1 Preparing the Project

Most of the steps required to configure the project to work with Google Sign-in as an authentication provider have already been performed in the chapter entitled *Email/Password Authentication using FirebaseUI Auth*. These steps include connecting the project to Firebase from within Android Studio and adding the build dependencies to the Gradle build files.

Two additional steps are, however, required in preparation for adding Google Sign-in support to the project. These consist of adding the SHA-1 fingerprint for the app to the Firebase project and enabling Google Sign-in as a supported sign-in method.

6.2 Obtaining the SHA-1 Fingerprint

As an Android developer you have two unique certificates used to sign the apps you develop using Android Studio. The *debug certificate* is used to sign apps that are being developed and tested. The *release certificate* on the other hand, is used to sign the apps when they are ready to be uploaded to the Google Play store.

These certificates take the form of SHA-1 fingerprints. By default the debug key is stored in a file named *debug.keystore* and located in the *.android* folder of your home directory. The release keystore file is created when you first generate the release APK file for your app and will be placed in a location of your choosing.

Ultimately, both the release and debug fingerprints will need to be added to the Firebase project with which the Android app is associated in order to use the Google Sign-in provider. At this stage, however, only the debug fingerprint needs to be added.

There are a number of options for obtaining the SHA-1 fingerprint, the most common of which is to use the keytool command-line tool included with the Android SDK. On Windows, open a command prompt window and execute the following command to obtain the debug key:

```
keytool -exportcert -list -v -alias androiddebugkey -keystore
              %USERPROFILE%\.android\debug.keystore
```

The command will prompt for the password which, by default, set to *android*.

Similarly, the following command can be executed within a terminal window on macOS or Linux to obtain the debug key:

```
keytool -exportcert -list –v -alias androiddebugkey -keystore
```