

# **Firestore Essentials**

## **Android Edition**

---



Firestore Essentials – Android Edition

First Edition

© 2017 Neil Smyth / Payload Media, Inc. All Rights Reserved.

This book is provided for personal use only. Unauthorized use, reproduction and/or distribution strictly prohibited. All rights reserved.

The content of this book is provided for informational purposes only. Neither the publisher nor the author offers any warranties or representation, express or implied, with regard to the accuracy of information contained in this book, nor do they accept any liability for any loss or damage arising from any errors or omissions.

This book contains trademarked terms that are used solely for editorial purposes and to the benefit of the respective trademark owner. The terms used within this book are not intended as infringement of any trademarks.

Rev 1.0





# Table of Contents

<b>1. Introduction</b>	<b>1</b>
1.1 Downloading the Code Samples	1
1.2 Feedback	1
1.3 Errata	1
<b>2. Getting Started with Firebase</b>	<b>3</b>
2.1 Signing into the Firebase Console	3
2.2 Creating a New Project	4
2.3 Firebase Projects	5
2.4 Firebase Pricing Plans	5
2.5 Summary	5
<b>3. Firebase User Authentication</b>	<b>7</b>
3.1 An Overview of Firebase Authentication	7
3.1.1 <i>FirebaseAuth Instance</i>	7
3.1.2 <i>AuthUI Instance</i>	8
3.1.3 <i>FirebaseUser Class</i>	8
3.1.4 <i>AuthCredential Classes</i>	8
3.1.5 <i>Authentication Provider Classes</i>	8
3.2 FirebaseUI Auth Authentication	9
3.3 Firebase SDK Authentication	9
3.4 Summary	10
<b>4. Email/Password Authentication using FirebaseUI Auth</b>	<b>11</b>
4.1 Creating the Example Project	11
4.2 Connecting the Project to Firebase	11
4.3 Adding the FirebaseUI Project Dependencies	14
4.4 Enabling Firebase Email/Password Authentication	15
4.5 Adding the Signed-In Activity	16
4.6 Designing the SignedInActivity Layout	16
4.7 Firebase Initialization Steps	17
4.8 Initiating the Sign-in Process	18
4.9 Handling the Sign In Activity Result	19
4.10 Handling the Sign-in	20
4.11 Signing Out	21
4.12 User Account Removal	22
4.13 Summary	23
<b>5. Testing and Customizing FirebaseUI Auth Authentication</b>	<b>25</b>
5.1 Authentication Testing Environment	25
5.2 Firebase Console User Management	25
5.3 Testing the FirebaseAuth App	26
5.4 Customizing the Password Reset Email Message	27
5.5 Testing the Password Reset	30
5.6 Enabling Smart Lock for Passwords	30
5.7 Color Customization	32
5.8 Specifying a Logo	33

5.9 Deleting the Test Account .....	34
5.10 Summary.....	34
<b>6. Google Sign-In Authentication using FirebaseUI Auth.....</b>	<b>35</b>
6.1 Preparing the Project.....	35
6.2 Obtaining the SHA-1 Fingerprint .....	35
6.3 Adding the SHA-1 Key to the Firebase Project .....	36
6.4 Enabling the Google Sign-In Method .....	37
6.5 Adding the Google Sign-In Provider within the App.....	38
6.6 Testing Google Sign-In .....	39
6.7 Displaying the User’s Profile Photo .....	41
6.8 Summary.....	44
<b>7. Facebook Login Authentication using FirebaseUI Auth .....</b>	<b>47</b>
7.1 Preparing the Project.....	47
7.2 Obtaining the Facebook App ID and App Secret .....	47
7.3 Adding the App ID and Secret to the Firebase Project .....	49
7.4 Adding the Facebook Login Product to the App .....	49
7.5 Setting the OAuth Redirect URI .....	50
7.6 Adding the Debug and Release Key Hash .....	51
7.7 Modifying the App.....	52
7.8 Testing Facebook Login Authentication .....	54
7.9 Summary.....	56
<b>8. Twitter Sign-In Authentication using FirebaseUI Auth .....</b>	<b>57</b>
8.1 Preparing the Project.....	57
8.2 Enabling Twitter Authentication.....	57
8.3 Creating a Twitter App.....	57
8.4 Obtaining the Twitter API Keys.....	58
8.5 Generating the Access Token .....	59
8.6 Adding the API Keys to the Project.....	59
8.7 Adding Twitter to the Provider List.....	60
8.8 Adding the Twitter Library Dependency.....	60
8.9 Testing Twitter Authentication.....	60
8.10 Summary.....	62
<b>9. Phone Number Sign-in Authentication using FirebaseUI Auth .....</b>	<b>63</b>
9.1 Preparing the Project.....	63
9.2 Enabling Phone Number Authentication .....	63
9.3 Adding Phone Verification to the Provider List .....	63
9.4 Testing Phone Number Authentication .....	64
9.5 Summary.....	66
<b>10. Email/Password Authentication using the Firebase SDK.....</b>	<b>67</b>
10.1 Creating the PasswordAuth Project.....	67
10.2 Enabling Authentication Providers .....	67
10.3 Connecting the Project to Firebase .....	68
10.4 Designing the User Interface Layout .....	68
10.5 Getting View Object References.....	71
10.6 Adding a User Notification Method.....	72
10.7 Accessing the FirebaseAuth Instance .....	72
10.8 Adding the Authentication State Listener .....	73

10.9 Adding and Removing the Listener .....	74
10.10 Creating the User Account .....	75
10.11 Implementing the Sign-In Method.....	76
10.12 Signing Out.....	77
10.13 Implementing the Password Reset Option .....	77
10.14 Testing the Project .....	77
10.15 Summary.....	78
<b>11. Managing User Accounts using the Firebase SDK .....</b>	<b>79</b>
11.1 Enhancing the User’s Profile Information .....	79
11.2 Changing a User’s Email Address .....	80
11.3 Changing a User’s Password .....	81
11.4 Deleting a User Account .....	82
11.5 Verifying the User’s Email Address .....	82
11.6 Understanding Security Sensitive Actions .....	83
11.7 Summary.....	83
<b>12. Handling Firebase Authentication Errors and Failures .....</b>	<b>85</b>
12.1 Completion Listeners and Basic Failure Detection .....	85
12.2 The Failure Listener.....	86
12.3 FirebaseAuth Exception Types .....	87
12.4 FirebaseAuth Error Codes .....	88
12.5 Handling Secure Action Exceptions .....	89
12.6 Summary.....	92
<b>13. Google Authentication using the Firebase SDK.....</b>	<b>93</b>
13.1 Firebase Authentication with the Google Sign-In API.....	93
13.2 Creating the FirebaseAuth Project.....	93
13.3 Connecting the Project to Firebase .....	94
13.4 Configuring the SHA-1 Fingerprint Key .....	94
13.5 Adding the Google Play Authentication Library.....	94
13.6 Designing the User Interface Layout.....	95
13.7 Adding a User Notification Method .....	97
13.8 Accessing the FirebaseAuth Instance .....	97
13.9 Adding the Authentication State Listener.....	98
13.10 Adding and Removing the Listener .....	100
13.11 Initializing the Google API Client.....	101
13.12 Implementing the Sign In Method .....	102
13.13 Registering the User with Firebase .....	103
13.14 Signing Out.....	104
13.15 Testing the Project .....	105
13.16 Summary.....	105
<b>14. Facebook Authentication using the Firebase SDK.....</b>	<b>107</b>
14.1 Authentication with the Firebase and Facebook SDKs .....	107
14.2 Creating the FirebaseAuth Project.....	107
14.3 Connecting the Project to Firebase .....	107
14.4 Firebase and Facebook Console Configuration.....	108
14.5 Configuring the Android Studio Project.....	108
14.6 Adding the Facebook SDK Library .....	109
14.7 Designing the User Interface Layout.....	110
14.8 Adding the Facebook LoginButton.....	112

14.9 Facebook SDK Initialization Steps .....	113
14.10 Handling Authentication Results and Callbacks.....	114
14.11 Testing Facebook Authentication .....	115
14.12 Initializing Firebase .....	116
14.13 Registering the User in Firebase .....	120
14.14 Testing the Project.....	121
14.15 Detecting the Facebook Log Out .....	121
14.16 Summary.....	123
<b>15. Twitter Log In Authentication using the Firebase SDK .....</b>	<b>125</b>
15.1 Authentication using Firebase and the Twitter SDK .....	125
15.2 Creating the TwitterAuth Project .....	125
15.3 Connecting the Project to Firebase .....	125
15.4 Enabling Twitter Authentication.....	126
15.5 Adding the Twitter Library Dependency.....	126
15.6 Designing the User Interface Layout .....	126
15.7 Adding the Twitter Login Button .....	128
15.8 Performing Initialization Tasks .....	129
15.9 Handling Callbacks and Activity Results.....	130
15.10 Testing the Twitter Authentication.....	131
15.11 Initializing Firebase .....	133
15.12 Registering the User in Firebase .....	135
15.13 Signing Out.....	137
15.14 Testing the Project.....	137
15.15 Configuring Privacy and Terms of Service URLs.....	138
15.16 Enabling Email Address Permission .....	138
15.17 Requesting the User's Email Address .....	138
15.18 Testing Email Permission .....	139
15.19 Summary.....	140
<b>16. Phone Number Authentication using the Firebase SDK.....</b>	<b>141</b>
16.1 Phone Number Authentication.....	141
16.2 Creating the Example Project .....	141
16.3 Connecting the Project to Firebase .....	142
16.4 Updating the firebase-auth Package .....	142
16.5 Designing the User Interface .....	142
16.6 Performing Initialization Tasks .....	143
16.7 Sending the Verification Code .....	144
16.8 Adding the Verification Callbacks .....	145
16.9 Verifying the Code .....	146
16.10 Signing In with the Credential.....	147
16.11 Resending the Verification Code .....	147
16.12 Signing Out.....	148
16.13 Testing the App.....	148
16.14 Summary.....	148
<b>17. Anonymous Authentication using the Firebase SDK .....</b>	<b>149</b>
17.1 Anonymous Firebase Authentication .....	149
17.2 Creating the AnonAuth Project.....	150
17.3 Connecting the Project to Firebase .....	150
17.4 Enabling Anonymous Authentication .....	150
17.5 Designing the User Interface Layout .....	150



17.6 Accessing the FirebaseAuth Instance .....	152
17.7 Adding the Authentication State Listener.....	153
17.8 Performing the Anonymous Authentication.....	155
17.9 Signing Out.....	156
17.10 Testing the Project.....	156
17.11 Summary.....	156
<b>18. Linking and Unlinking Firebase Authentication Providers.....</b>	<b>157</b>
18.1 Firebase Authentication Account Linking .....	157
18.2 Limitations of Firebase Account Linking .....	157
18.3 Implementing Email/Password Authentication .....	158
18.4 Testing the Project.....	159
18.5 Linking Other Account Types .....	159
18.6 Unlinking an Authentication Provider.....	160
18.7 Summary.....	161
<b>19. Firebase Realtime Database .....</b>	<b>163</b>
19.1 An Overview of the Firebase Realtime Database .....	163
19.2 How is the Data Stored? .....	163
19.3 Nesting Data .....	163
19.4 Separating User Data .....	164
19.5 Avoiding Deep Nesting.....	165
19.6 Supported Data Types.....	167
19.7 Adding Realtime Database Support to Projects.....	168
19.8 Offline Data Handling.....	169
19.9 Summary.....	169
<b>20. Writing Firebase Realtime Database Data .....</b>	<b>171</b>
20.1 Obtaining the Database Reference .....	171
20.2 Performing a Basic Write Operation .....	172
20.3 Accessing Child Nodes .....	172
20.4 Writing to Multiple Nodes .....	173
20.5 Storing a Java Object.....	174
20.6 Deleting Data .....	174
20.7 Handling Write Errors .....	175
20.8 Summary.....	175
<b>21. Reading Firebase Realtime Database Data .....</b>	<b>177</b>
21.1 Detecting Value Changes .....	177
21.2 Working with a DataSnapshot Object.....	178
21.3 Extracting Data from a DataSnapshot Object .....	180
21.4 Reading Data into a Java Object .....	180
21.5 Summary.....	181
<b>22. Firebase Realtime Database Rules.....</b>	<b>183</b>
22.1 Accessing and Changing the Database Rules .....	183
22.2 Understanding Database Rule Types .....	184
22.2.1 Read and Writing Rules .....	184
22.2.2 Data Validation Rules .....	185
22.2.3 Indexing Rules.....	185
22.3 Structuring Database Rules.....	185
22.4 Securing User Data.....	188

22.5 \$ Variables .....	188
22.6 Predefined Variables.....	189
22.7 RuleDataSnapShot Methods.....	189
22.8 Data Validation Rules.....	191
22.9 Indexing Rules.....	191
22.10 Summary.....	193
<b>23. Working with Firebase Realtime Database Lists .....</b>	<b>195</b>
23.1 The push() Method .....	195
23.2 Listening for Events .....	196
23.3 Performing List Queries .....	197
23.4 Querying and Indexes.....	200
23.5 Summary.....	200
<b>24. A Basic Firebase Realtime Database Tutorial .....</b>	<b>201</b>
24.1 About the Realtime Database Project .....	201
24.2 Creating the Realtime Database Project.....	201
24.3 Adding User Authentication .....	201
24.4 Adding the Second Activity.....	202
24.5 Designing the SignedInActivity User Interface .....	203
24.6 Configuring the Project for Realtime Database Access .....	204
24.7 Performing the Authentication.....	205
24.8 Accessing the Database .....	207
24.9 Writing to the Database .....	208
24.10 Signing Out.....	209
24.11 Configuring the Database Rules.....	210
24.12 Testing the App.....	210
24.13 Adding a Validation Rule.....	211
24.14 Summary.....	212
<b>25. A Firebase Realtime Database List Data Tutorial .....</b>	<b>213</b>
25.1 About the Data List App Project .....	213
25.2 Creating the Data List Project.....	213
25.3 Configuring the Project for Realtime Database Access .....	213
25.4 Designing the User Interface .....	213
25.5 Performing Initialization Tasks .....	216
25.6 Implementing the ListView.....	217
25.7 Implementing Child Event Listener.....	218
25.8 Adding Items to the List.....	220
25.9 Deleting List Items .....	220
25.10 Querying the List.....	221
25.11 Changing the Database Rules .....	222
25.12 Testing the App.....	222
25.13 Summary.....	224
<b>26. Firebase Cloud Messaging .....</b>	<b>225</b>
26.1 An Overview of Firebase Cloud Messaging.....	225
26.2 Creating the Firebase Messaging Project .....	225
26.3 Adding Firebase Messaging Support to the Project .....	225
26.4 Designing the User Interface .....	226
26.5 Obtaining and Monitoring the Registration Token.....	226
26.6 Sending a Notification to the Device .....	228

26.7 Receiving the Notification.....	229
26.8 Including Custom Data within the Notification .....	230
26.9 Foreground App Notification Handling.....	231
26.10 Sending to Apps and User Segments .....	232
26.11 Conversion Events.....	234
26.12 Summary.....	234
<b>27. Sending Firebase Cloud Messages from a Node.js Server .....</b>	<b>235</b>
27.1 An Introduction to Node.js .....	235
27.2 Installing Node.js on macOS .....	235
27.3 Installing Node.js on Windows .....	236
27.4 Installing Node.js on Linux.....	236
27.5 Initializing and Configuring Node.js .....	236
27.6 Running a Test .....	237
27.7 Installing the Firebase Admin SDK .....	237
27.8 Generating the Service Account Credentials .....	237
27.9 Initializing the Admin SDK.....	238
27.10 Adding the Destination Registration Token .....	239
27.11 Understanding Message Payloads .....	239
27.12 Defining the Payload and Sending the Message.....	240
27.13 Testing the Code .....	241
27.14 Topics.....	241
27.15 Using Topic Conditions .....	243
27.16 Summary.....	244
<b>28. Managing Firebase Cloud Messaging Device Groups with Node.js .....</b>	<b>245</b>
28.1 Understanding Device Groups .....	245
28.2 Requirements for Creating a Device Group.....	245
28.3 Creating a Device Group using HTTP .....	246
28.4 Creating a Device Group using Node.js.....	246
28.5 Sending a Message to a Device Group.....	247
28.6 Managing Device Group Members in Node.js .....	248
28.7 Summary.....	249
<b>29. Managing Firebase Messaging Device Groups from an Android App .....</b>	<b>251</b>
29.1 Managing Device Groups from a Client App.....	251
29.2 Generating the Client ID .....	251
29.3 Creating the Client App Project .....	252
29.4 Adding the Group Button.....	253
29.5 Obtaining the ID Token .....	253
29.6 Declaring the Sender ID .....	254
29.7 Creating an Asynchronous Task.....	254
29.8 Creating the Device Group.....	255
29.9 Executing the AsyncTask.....	257
29.10 Testing the App .....	258
29.11 Removing Devices from a Device Group.....	258
29.12 Summary.....	258
<b>30. Firebase Cloud Upstream Messaging.....</b>	<b>259</b>
30.1 Firebase Cloud Upstream Messaging Architecture.....	259
30.2 Implementing the App Server.....	260
30.3 Sending an Upstream Message from the Client .....	260

30.4 Summary.....	261
<b>31. A Firebase Cloud Messaging Upstream XMPP App Server .....</b>	<b>263</b>
31.1 About the FCM App Server Project.....	263
31.2 Project Dependencies.....	263
31.3 CcsClient.java.....	264
31.4 PayloadProcessor.java.....	264
31.5 RegisterProcessor.java.....	264
31.6 AccountStore.java.....	264
31.7 MessageProcessor.java.....	265
31.8 Building and Running the Server .....	265
31.9 Summary.....	266
<b>32. An Example Firebase Upstream Cloud Messaging Client App .....</b>	<b>267</b>
32.1 Creating the Project.....	267
32.2 Adding Firebase Messaging Support to the Project .....	267
32.3 Designing the User Interface .....	267
32.4 Obtaining the Registration Token.....	269
32.5 Handling Incoming Messages .....	270
32.6 Registering the User .....	272
32.7 Sending the Upstream Message .....	273
32.8 Testing the Upstream Project.....	273
32.9 Summary.....	274
<b>33. Firebase Cloud Storage .....</b>	<b>275</b>
33.1 An Overview of Firebase Cloud Storage .....	275
33.2 Storage References.....	275
33.3 Storage Reference Properties.....	276
33.4 Uploading a File to Firebase Cloud Storage .....	276
33.5 Uploading a File as a Stream.....	277
33.6 Uploading a File From Memory Data.....	277
33.7 Monitoring and Managing the Upload .....	277
33.8 Accessing Upload Task Snapshot Properties .....	278
33.9 Reading Metadata from an Uploaded File.....	278
33.10 Customizing the File Metadata.....	281
33.11 Updating File Metadata .....	282
33.12 Deleting an Uploaded File.....	282
33.13 Resuming an Interrupted Upload .....	283
33.14 Managing Cloud Storage in the Firebase Console .....	284
33.15 Summary.....	284
<b>34. A Guide to Firebase Cloud Storage Security Rules.....</b>	<b>285</b>
34.1 Understanding Cloud Storage Security Rules .....	285
34.2 Security Rules Structure .....	286
34.3 The match Keyword.....	286
34.4 The allow Keyword .....	287
34.5 The if Statement .....	287
34.6 Wildcard Matching .....	289
34.7 Protecting User Files.....	290
34.8 Summary.....	290
<b>35. A Firebase Cloud Storage Tutorial.....</b>	<b>293</b>

35.1 About the Firebase Cloud Storage Example .....	293
35.2 Creating the Project .....	293
35.3 Adding Authentication to the Project .....	293
35.4 Adding the Signed In Activity .....	294
35.5 Designing the SignedInActivity User Interface .....	295
35.6 Performing the Authentication .....	297
35.7 Recording Video .....	299
35.8 Adding Firebase Cloud Storage Support .....	300
35.9 Uploading the Video File to the Cloud .....	300
35.10 Updating the Progress Bar .....	302
35.11 Downloading the Video File .....	303
35.12 Setting the Storage Security Rules .....	304
35.13 Testing the Project .....	304
35.14 Summary .....	304
<b>36. Firebase Remote Config .....</b>	<b>305</b>
36.1 An Overview of Firebase Remote Config .....	305
36.2 The FirebaseRemoteConfig Object .....	305
36.3 Declaring and Setting In-App Parameters .....	306
36.4 Accessing Remote Config Parameters .....	306
36.5 Setting Server Side Parameters .....	307
36.6 Fetching the Parameters .....	308
36.7 Activating Fetched Parameters .....	309
36.8 Working with Conditions .....	309
36.9 Parameter Fetching Strategies .....	312
36.10 Getting Status Information .....	313
36.11 Summary .....	313
<b>37. A Firebase Remote Config Tutorial .....</b>	<b>315</b>
37.1 Creating the Project .....	315
37.2 Adding Remote Config Support to the Project .....	315
37.3 Designing the User Interface .....	315
37.4 Declaring the Default In-App Parameters .....	317
37.5 Loading the Config Parameters .....	318
37.6 Adding the Server-side Parameters .....	319
37.7 Fetching and Activating the Remote Parameters .....	320
37.8 Handling Changes at Startup .....	321
37.9 Creating a Conditional Parameter Change .....	322
37.10 Summary .....	323
<b>38. Firebase Analytics .....</b>	<b>325</b>
38.1 An Overview of Firebase Analytics .....	325
38.2 The FirebaseAnalytics Object .....	325
38.3 Events .....	325
38.3.1 Predefined Events Types .....	326
38.3.2 Suggested Event Types .....	326
38.3.3 Custom Event Types .....	326
38.4 Viewing Events in the LogCat Window .....	326
38.5 User Properties .....	327
38.6 Audiences .....	327
38.7 Summary .....	327
<b>39. A Guided Tour of the Firebase Analytics Dashboard .....</b>	<b>329</b>

39.1 The Firebase Analytics Dashboard.....	329
39.2 Filtering the Data.....	329
39.2.1 Active users.....	330
39.2.2 Average revenue.....	330
39.2.3 first_open attribution.....	331
39.2.4 Retention cohorts.....	331
39.2.5 User engagement.....	332
39.2.6 In-app purchases.....	333
39.2.7 App version.....	333
39.2.8 Devices.....	334
39.3 Location.....	334
39.3.1 Demographics.....	335
39.3.2 Interests.....	335
39.4 Summary.....	336
<b>40. An Overview of the Firebase Analytics Screens.....</b>	<b>337</b>
40.1 The Events Screen.....	337
40.2 The Audiences Screen.....	338
40.3 The Attribution Screen.....	339
40.4 The Funnels Screen.....	340
40.5 The Cohorts Screen.....	340
40.6 The StreamView Screen.....	341
40.7 The DebugView Screen.....	342
40.8 User Properties Screen.....	345
40.9 Summary.....	346
<b>41. A Firebase Analytics Tutorial.....</b>	<b>347</b>
41.1 Creating the Firebase Analytics Project.....	347
41.2 Designing the User Interface.....	347
41.3 Adding Firebase Analytics Support to the Project.....	348
41.4 Performing the App Initialization.....	348
41.5 Adding the orderItem() Method.....	349
41.6 Enabling Debugging Modes.....	350
41.7 Adding the User Property in the Firebase Console.....	350
41.8 Testing the App.....	350
41.9 Creating an Audience.....	352
41.10 Summary.....	353
<b>42. Firebase Test Lab.....</b>	<b>355</b>
42.1 An Overview of Firebase Test Lab.....	355
42.2 Robo Testing.....	355
42.3 Instrumentation Testing.....	356
42.4 Game Loop Testing.....	356
42.5 Test Lab Pricing.....	356
42.6 Summary.....	356
<b>43. A Firebase Test Lab Robo Testing Example.....</b>	<b>357</b>
43.1 Getting the Test App.....	357
43.2 Running a Robo Test.....	357
43.3 Reviewing Robo Test Results.....	360
43.4 Introducing a Bug.....	362
43.5 Summary.....	363

<b>44. A Firebase Test Lab Instrumentation Testing Example .....</b>	<b>365</b>
44.1 Getting the Test App .....	365
44.2 An Overview of Espresso .....	365
44.3 Enabling Test Lab in the Project .....	366
44.4 Creating the Espresso Test Class .....	366
44.5 Recording the Test .....	366
44.6 Reviewing the Test Class .....	368
44.7 Running the Test Locally .....	369
44.8 Firebase Console Testing .....	370
44.9 Adding the Screenshotter Library to the Project .....	371
44.10 Adding the Screenshotter Code .....	373
44.11 Running the Test from Android Studio .....	374
44.12 Summary .....	376
<b>45. Firebase Dynamic Links .....</b>	<b>377</b>
45.1 An Overview of Firebase Dynamic Links .....	377
45.2 The Anatomy of a Dynamic Link .....	377
45.3 Creating Dynamic Links in the Firebase Console .....	378
45.4 Reviewing a Link in the Console .....	380
45.5 Creating a Dynamic Link in Code .....	382
45.6 Sharing a Dynamic Link .....	383
45.7 Receiving a Dynamic Link .....	384
45.8 Summary .....	386
<b>46. Creating, Receiving and Testing a Firebase Dynamic Link .....</b>	<b>387</b>
46.1 About the Dynamic Links Example App .....	387
46.2 Creating the Dynamic Links Project .....	387
46.3 Designing the User Interface .....	387
46.4 Connecting the Project to Firebase .....	388
46.5 Performing Basic Initialization .....	388
46.6 Adding the Link Generation Code .....	389
46.7 Sharing the Dynamic Link .....	390
46.8 Adding the Main Activity Intent Filter .....	391
46.9 Detecting a Dynamic Link Launch .....	391
46.10 Parsing the Deep Link URL .....	393
46.11 Generating the Dynamic Link .....	394
46.12 Testing a Dynamic Link .....	394
46.13 Adding a Second Activity .....	395
46.14 Configuring Automatic Activity Launching .....	396
46.15 Testing Automatic Launching .....	397
46.16 Dynamic Link Analytics .....	397
46.17 Summary .....	398
<b>47. Firebase Invites .....</b>	<b>399</b>
47.1 Creating a Firebase Invite Intent .....	399
47.2 Sending the Invitation .....	400
47.3 Handling the Activity Result .....	400
47.4 Testing Invitations .....	401
47.5 Summary .....	401
<b>48. Firebase App Indexing .....</b>	<b>403</b>
48.1 An Overview of Firebase App Indexing .....	403

48.2 Public Content Indexing .....	403
48.3 Personal Content Indexing .....	404
48.4 User Action Logging .....	404
48.5 Summary .....	404
<b>49. Implementing Firebase App Indexing .....</b>	<b>405</b>
49.1 Public Content Indexing .....	405
49.2 Personal Content Indexing .....	406
49.3 App Indexing Service .....	407
49.4 Logging User Actions .....	407
49.5 Removing Index Entries .....	408
49.6 Summary .....	408
<b>50. A Firebase App Indexing Tutorial .....</b>	<b>409</b>
50.1 About the Example App .....	409
50.2 The Database Schema .....	409
50.3 Loading and Running the Project .....	409
50.4 Adding Firebase App Indexing Support .....	411
50.5 Logging User Actions .....	411
50.6 Adding Content Indexing .....	413
50.7 Testing Content Indexing .....	414
50.8 Indexing User Added Content .....	417
50.9 Testing Index Updates .....	419
50.10 Deleting Index Entries .....	419
50.11 Summary .....	420
<b>51. Firebase Performance Monitoring .....</b>	<b>421</b>
51.1 An Overview of Firebase Performance Monitoring .....	421
51.2 Adding Performance Monitoring to a Project .....	421
51.3 Defining Custom Traces .....	423
51.4 Enabling Logcat Output .....	424
51.5 Reviewing Performance Data in the Firebase Console .....	424
51.6 Turning Off Performance Monitoring .....	427
51.7 Summary .....	428
<b>52. Firebase Cloud Functions .....</b>	<b>429</b>
52.1 A Cloud Functions Overview .....	429
52.2 Firebase Authentication Functions .....	429
52.3 Realtime Database Functions .....	430
52.4 Google Analytics Functions .....	431
52.5 HTTP Functions .....	431
52.6 Cloud Storage Functions .....	432
52.7 Performing External Operations .....	433
52.8 Modifying Cloud Function Settings .....	434
52.9 Summary .....	434
<b>53. Installing the Firebase CLI .....</b>	<b>435</b>
53.1 Installing Node.js .....	435
53.2 Installing the Firebase Tools Package .....	435
53.3 Logging into Firebase .....	435
53.4 Creating a Cloud Functions Project .....	436
53.5 Reviewing the Project .....	437



53.6 Deploying a Simple HTTP Cloud Function .....	437
53.7 Reviewing the Logs .....	438
53.8 Removing a Deployed Function .....	439
53.9 Summary.....	439
<b>54. A Firebase Cloud Functions Tutorial .....</b>	<b>441</b>
54.1 About the Example Project .....	441
54.2 Loading and Configuring the App .....	441
54.3 Trying out the App .....	442
54.4 Writing the Authentication Cloud Function.....	443
54.5 Writing the Realtime Database Function.....	444
54.6 Deploying the Cloud Functions .....	445
54.7 Testing the Cloud Functions.....	445
54.8 Summary.....	447
<b>55. A Cloud Functions and Firebase Cloud Storage Example .....</b>	<b>449</b>
55.1 The CloudStorage App .....	449
55.2 Setting Cloud Storage Rules.....	450
55.3 Installing Dependencies .....	451
55.4 Writing the Cloud Function.....	451
55.5 Testing the Cloud Function .....	453
55.6 Increasing Cloud Function Memory.....	454
55.7 Summary.....	456
<b>Index.....</b>	<b>457</b>



## 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](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](mailto: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>

## Introduction

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](mailto:feedback@ebookfrenzy.com). They are there to help you and will work to resolve any problems you may encounter.

## 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:

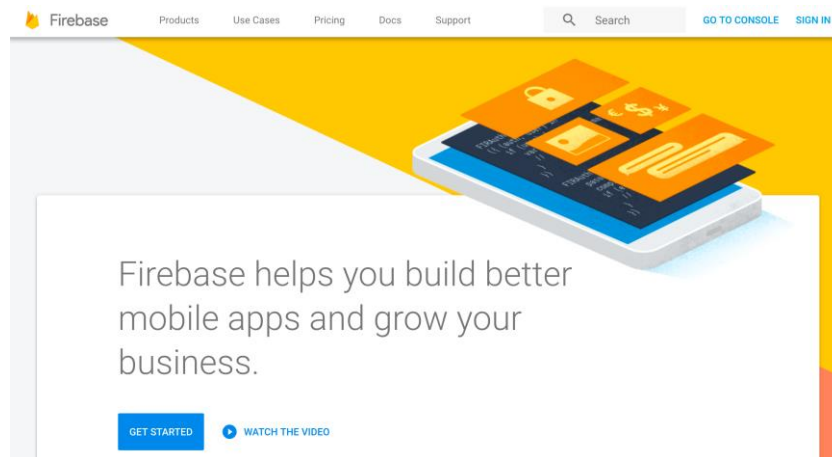


Figure 2-1

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:

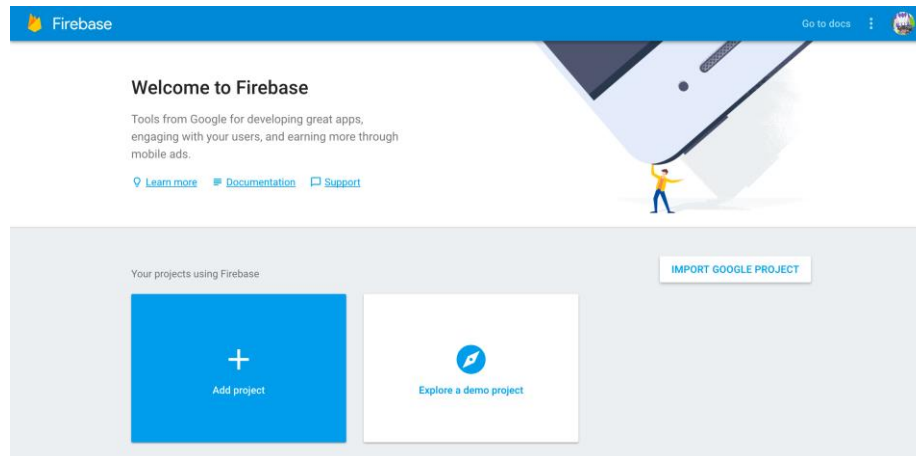


Figure 2-2

## 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

Country/region ?

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](#).

CANCEL CREATE PROJECT

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:

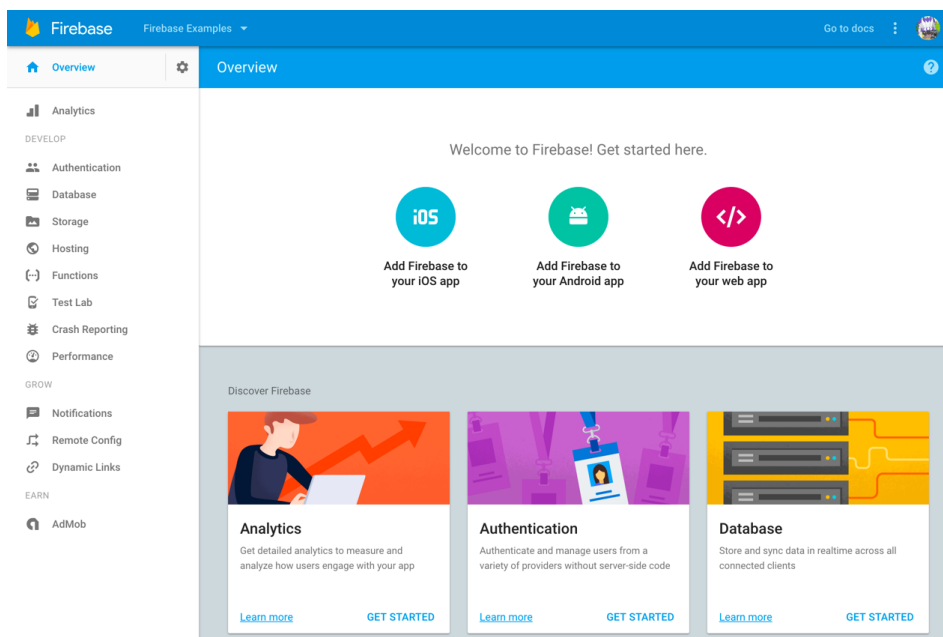


Figure 2-4

## 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.





## 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 FirebaseAuth 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.

## Firebase User Authentication

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. Firestore 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 FirestoreUI Auth, or the Firestore SDK. FirestoreUI Auth provides a quick and easy way to integrate authentication with minimal effort, while the Firestore SDK approach requires more work but provides greater flexibility.

## 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:

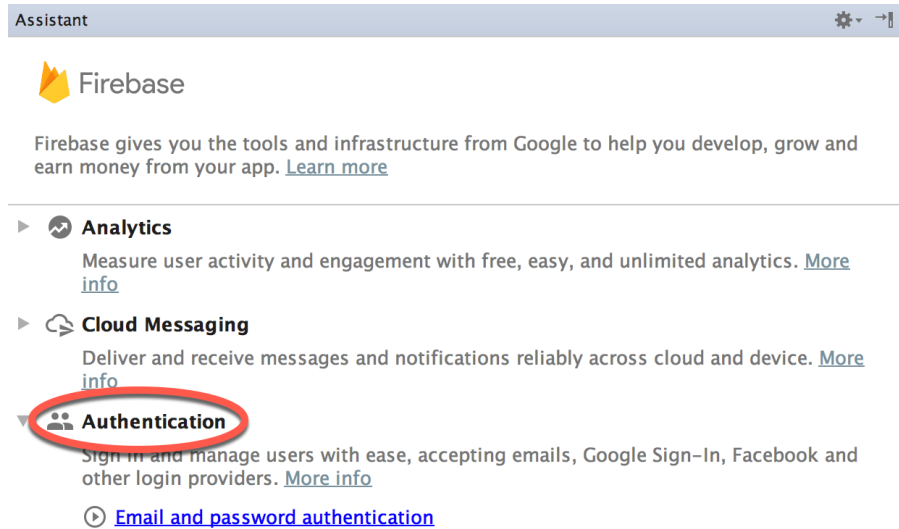


Figure 4-1

## Email/Password Authentication using FirebaseUI Auth

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:

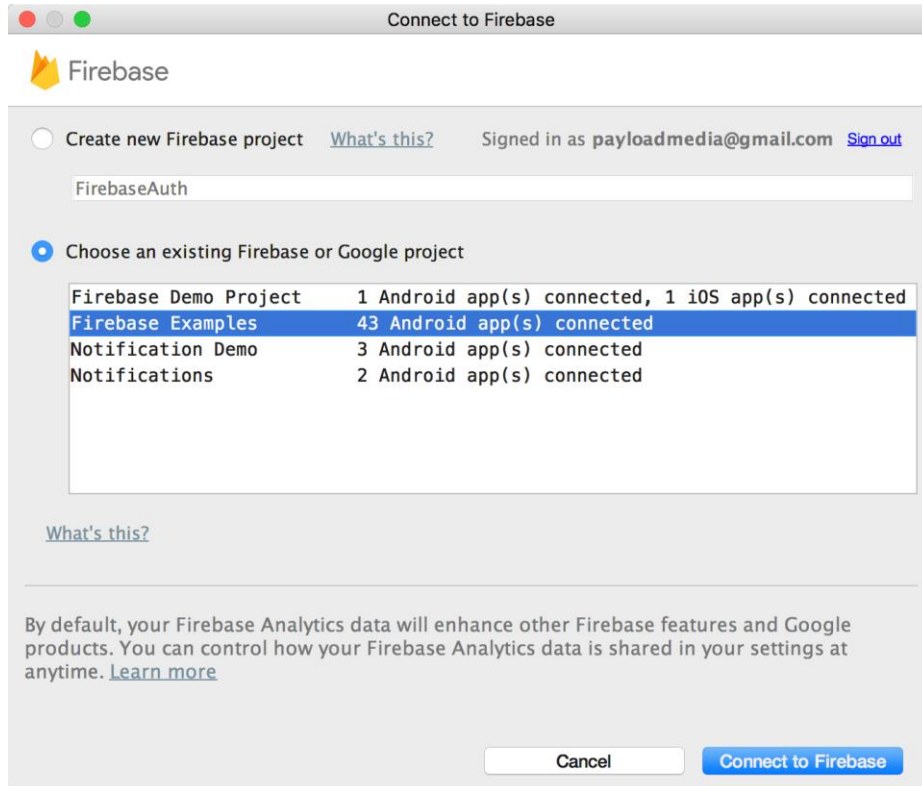


Figure 4-2

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:

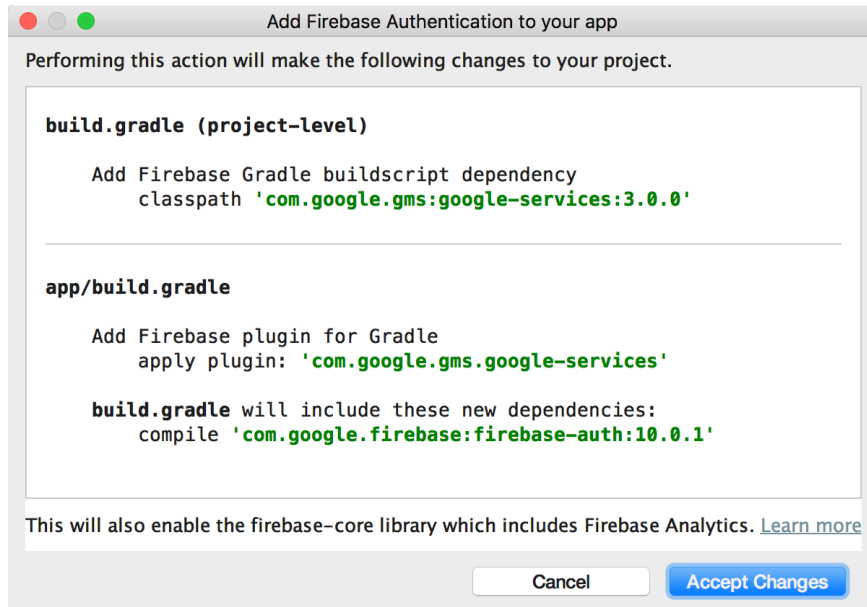


Figure 4-3

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:

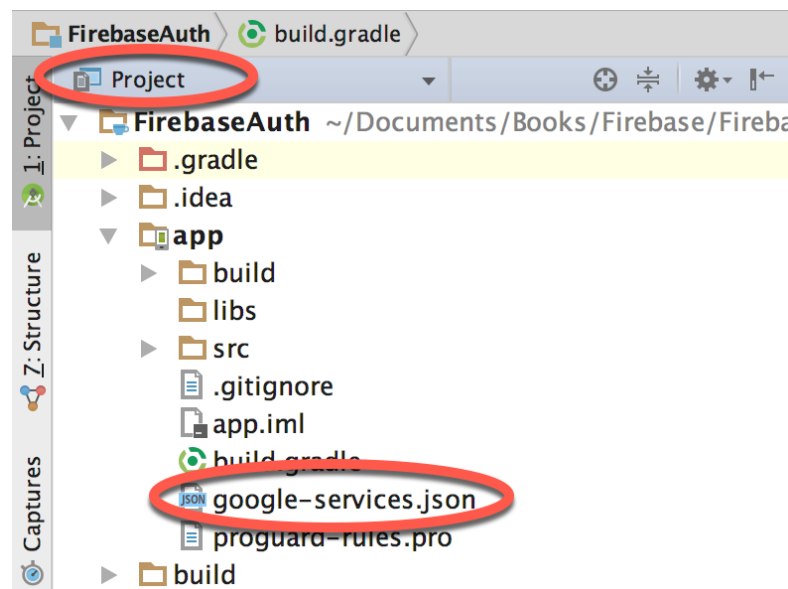


Figure 4-4

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.

## Email/Password Authentication using FirebaseUI Auth

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:

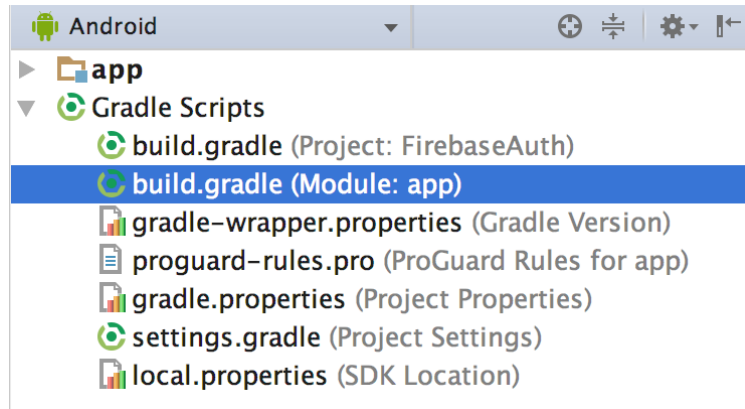


Figure 4-5

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):

```
dependencies {
    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:

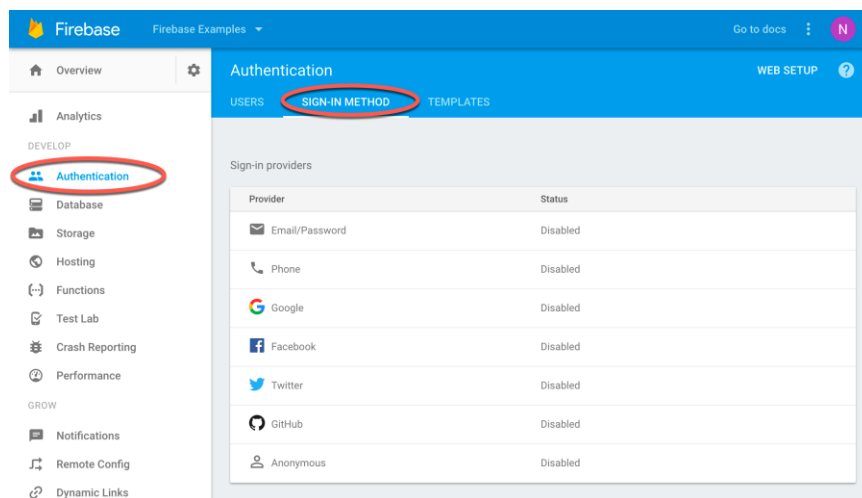


Figure 4-6

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.

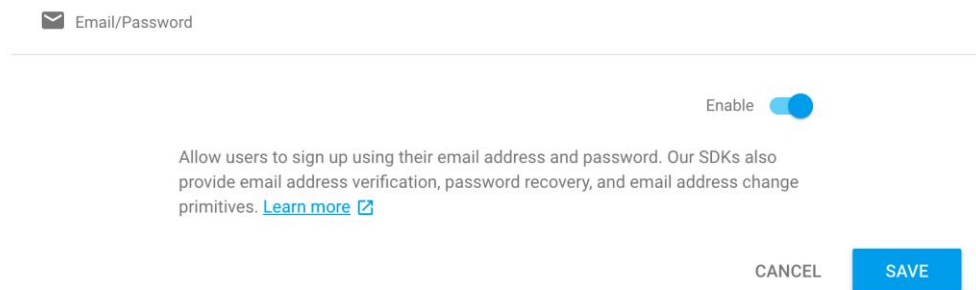


Figure 4-7

## 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`:

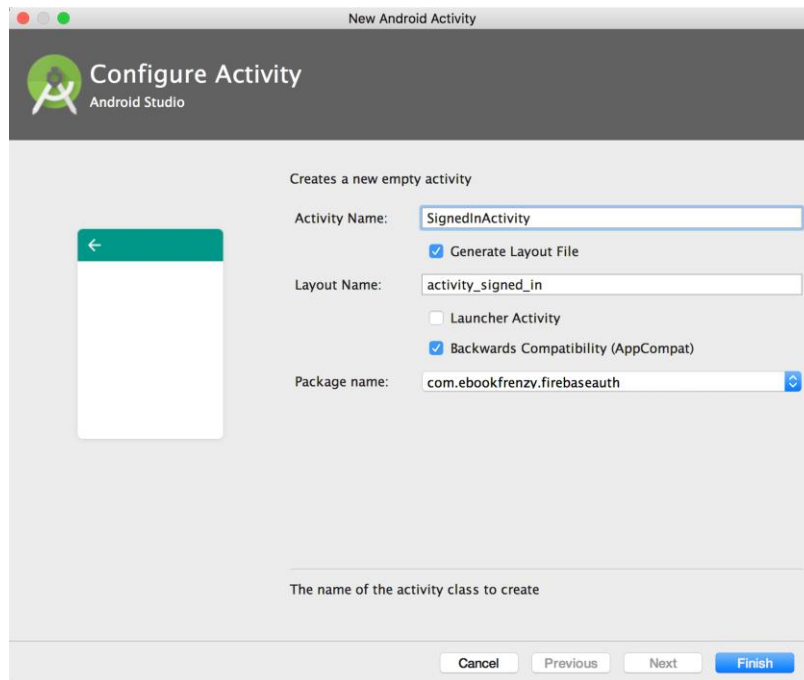


Figure 4-8

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.

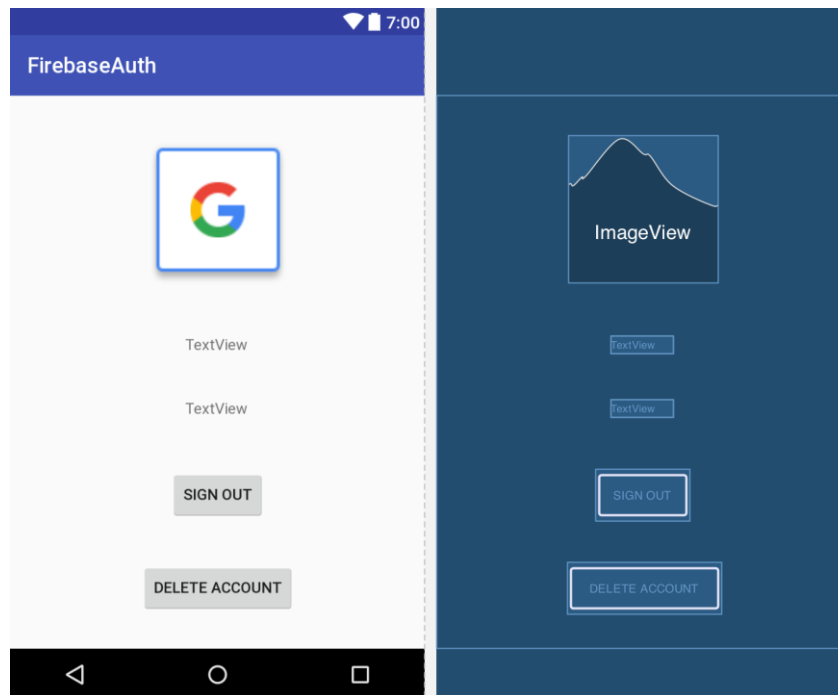


Figure 4-9

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 `deleteAccount`.

#### 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 `FirebaseAuthActivity` 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 {
```

## Email/Password Authentication using FirebaseUI Auth

```
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) {
```

## Email/Password Authentication using FirebaseUI Auth

```
        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)
}

```

## Email/Password Authentication using FirebaseUI Auth

```
.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.



## 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:

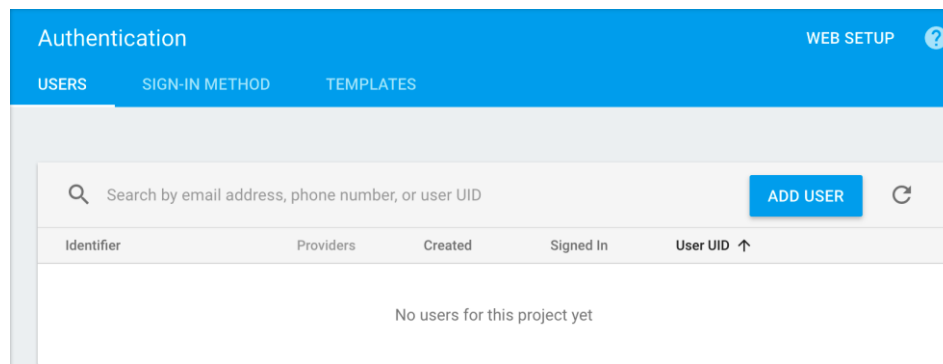


Figure 5-1

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:

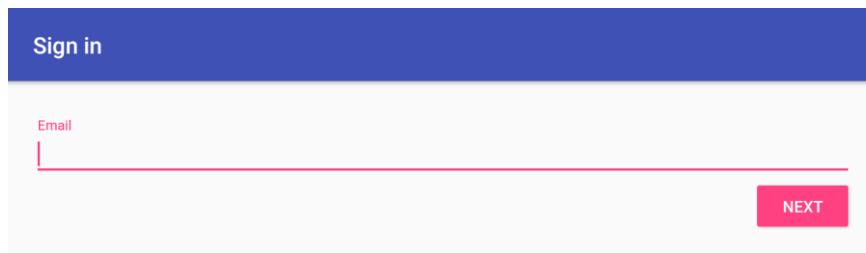


Figure 5-2

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:

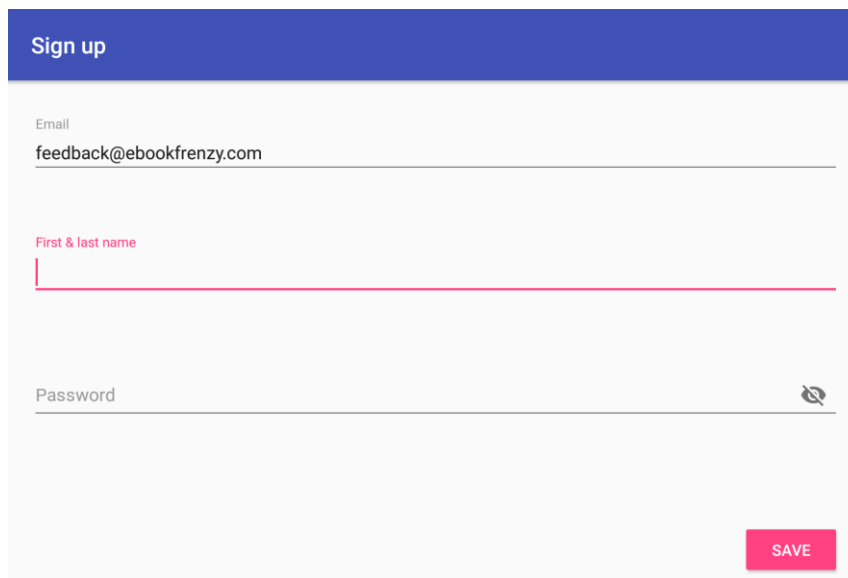


Figure 5-3

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:

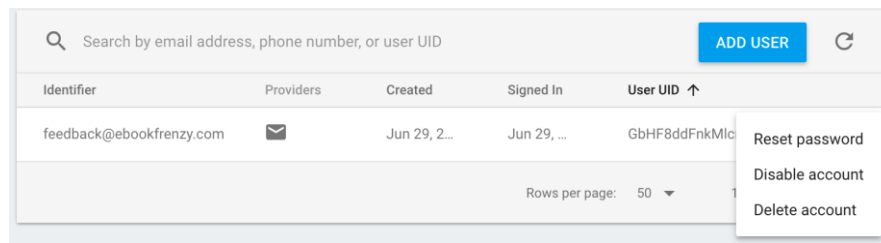


Figure 5-4

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:

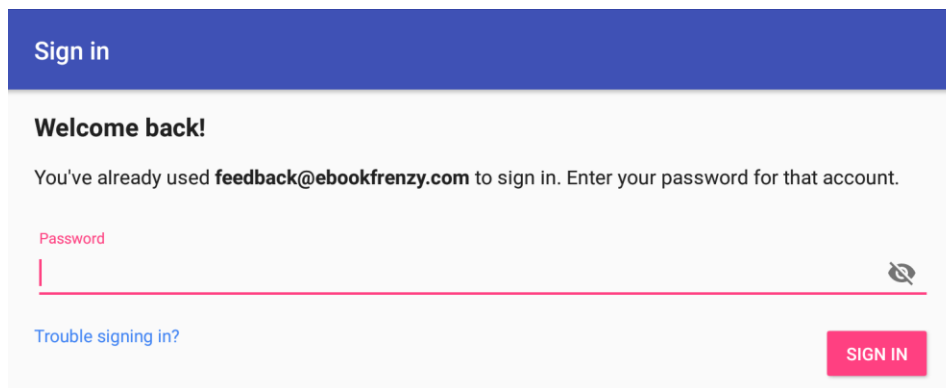


Figure 5-5

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:

## Testing and Customizing FirebaseUI Auth Authentication

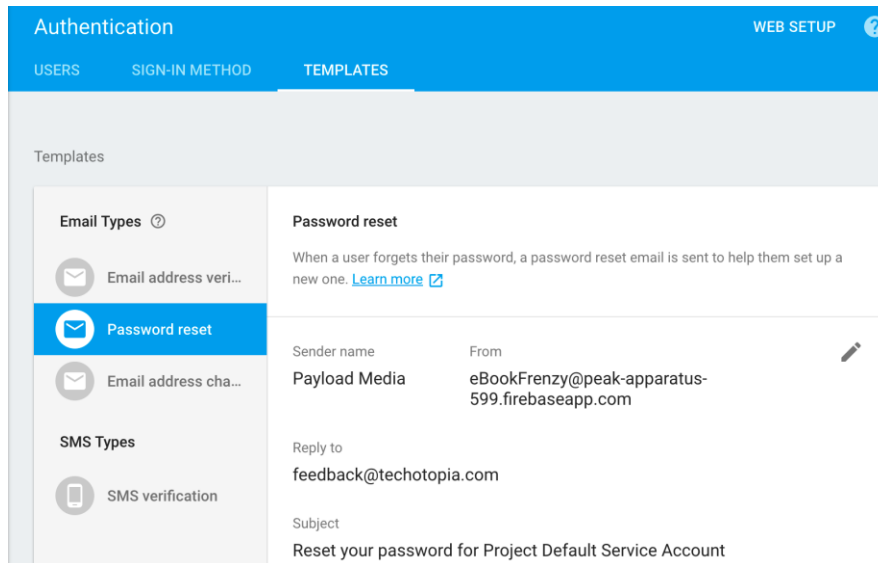


Figure 5-6

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:

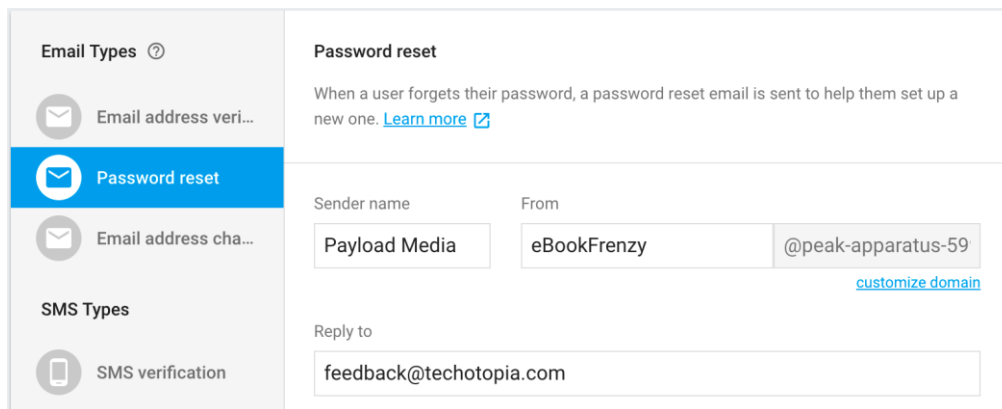


Figure 5-7

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:

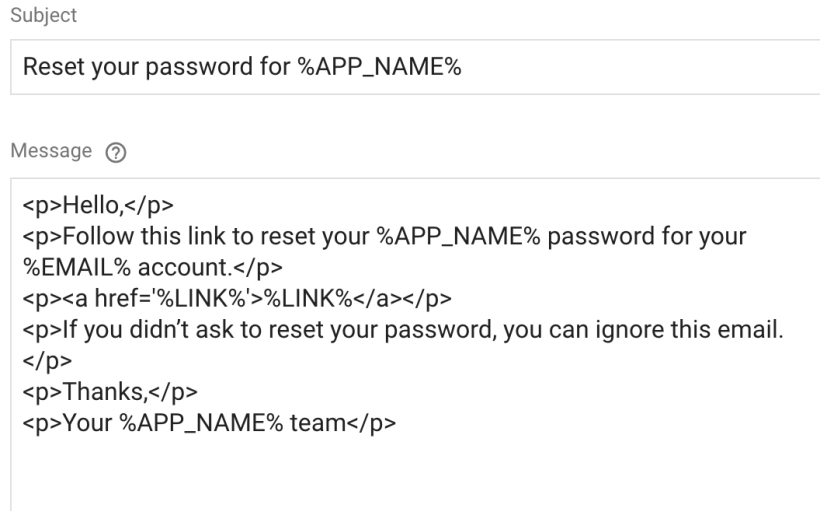


Figure 5-8

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):

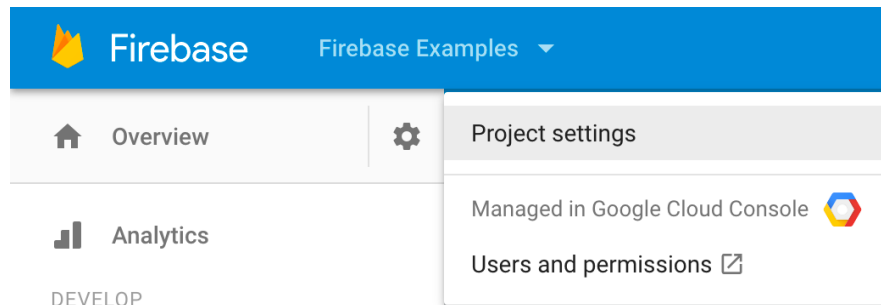


Figure 5-9

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:

## Testing and Customizing FirebaseUI Auth Authentication



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:

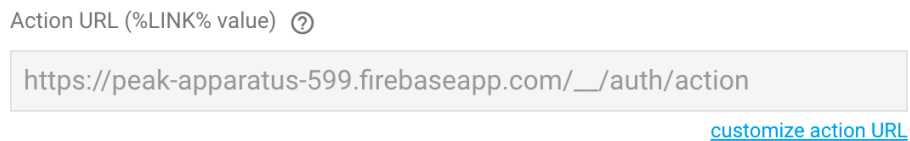


Figure 5-11

## 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:



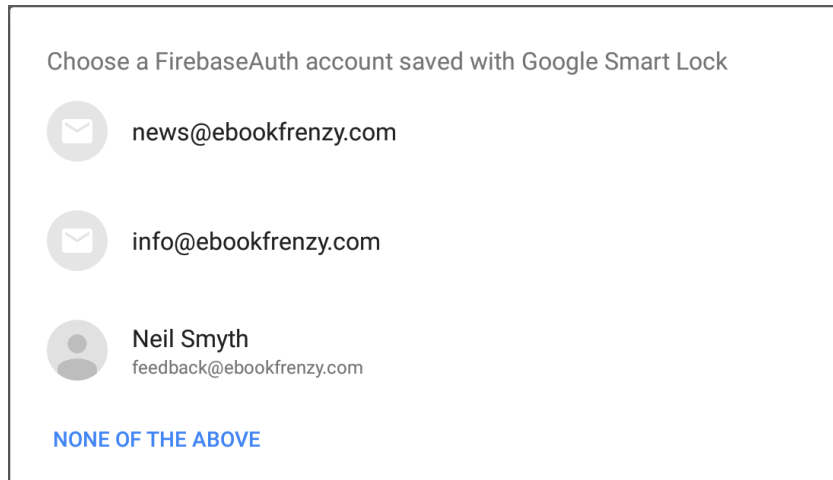


Figure 5-12

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:

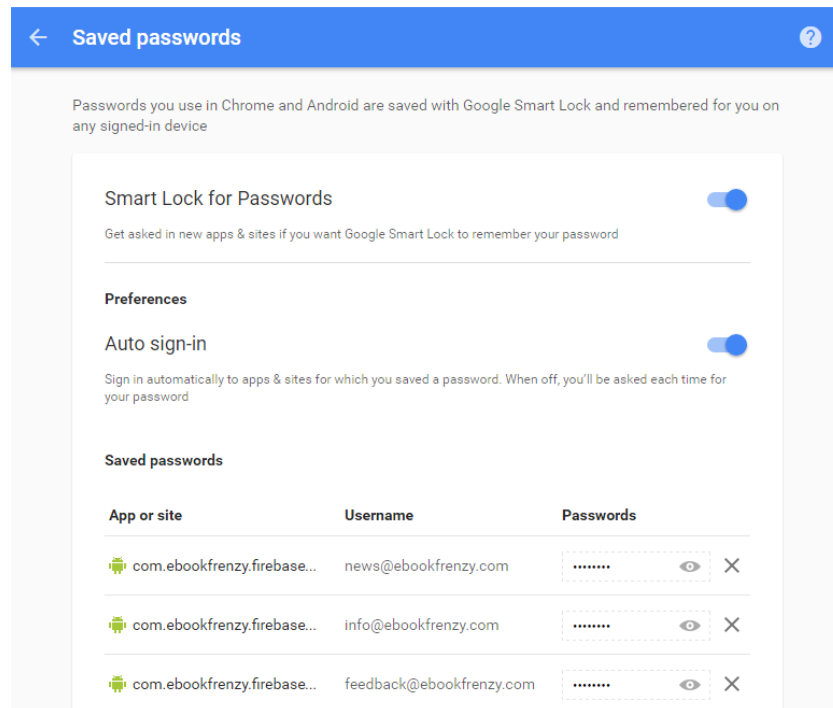


Figure 5-13

## 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:

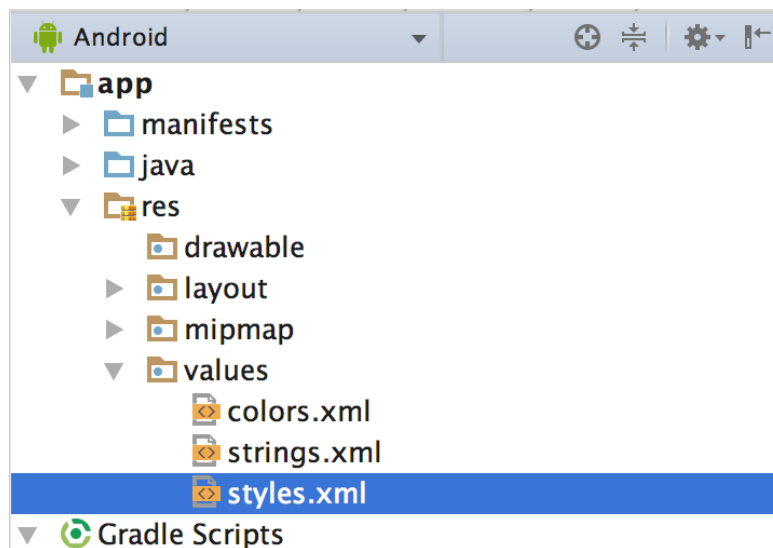


Figure 5-14

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 intent 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](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()

```

## Testing and Customizing FirebaseUI Auth Authentication

```
        .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.

## 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
```