Firebase Storage (Kotlin)

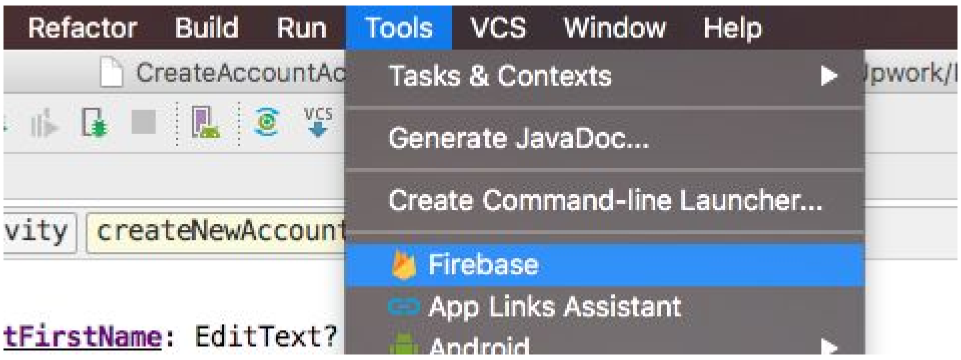
In this lesson on Firebase, we will learn how to use Firebase Storage API to upload files like images from an Android app and retrieve them.

To achieve the objective, we will use our last lesson on Firebase Authentication as a base and add a new feature to it. During a new user registration, we will allow user to upload his profile picture as well.

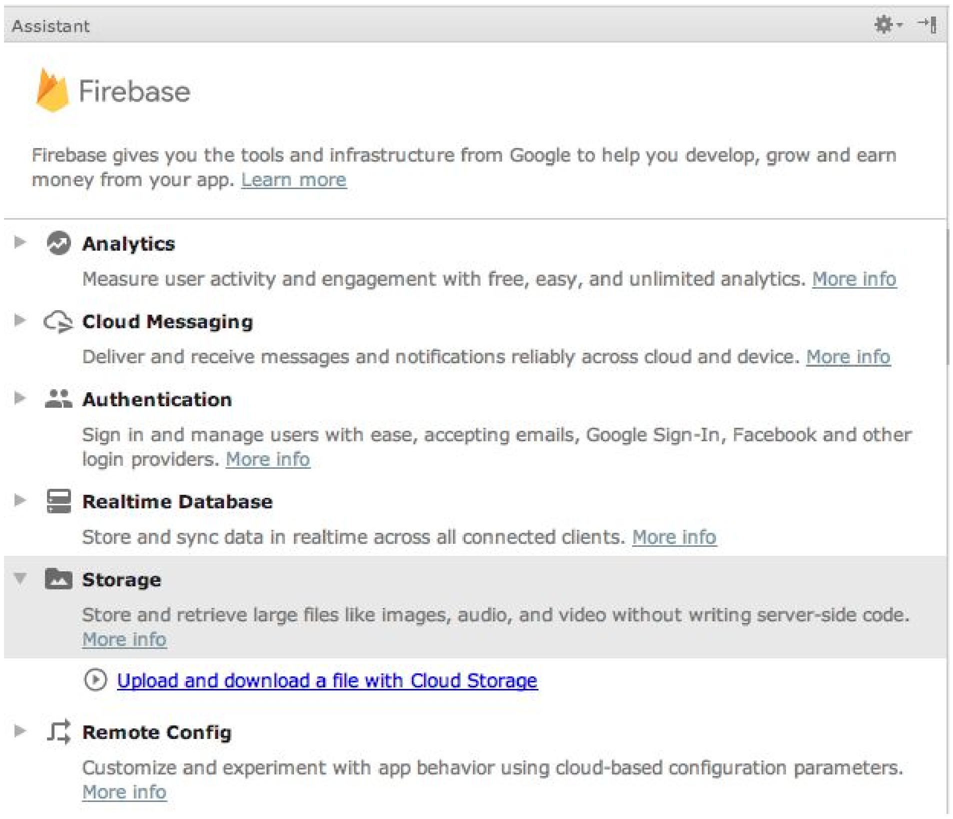
Although the lesson continues the previous learnings, all of the sample shown here can be independently used by you.

Adding Dependencies

To start, we will have to add new dependencies in our project so that it supports Firebase storage as well. Let’s do it now. Go to Tools > Firebase:



In the Firebase Assistant which opens, select ‘Firebase Storage’:


Assistant

Firebase

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

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

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

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

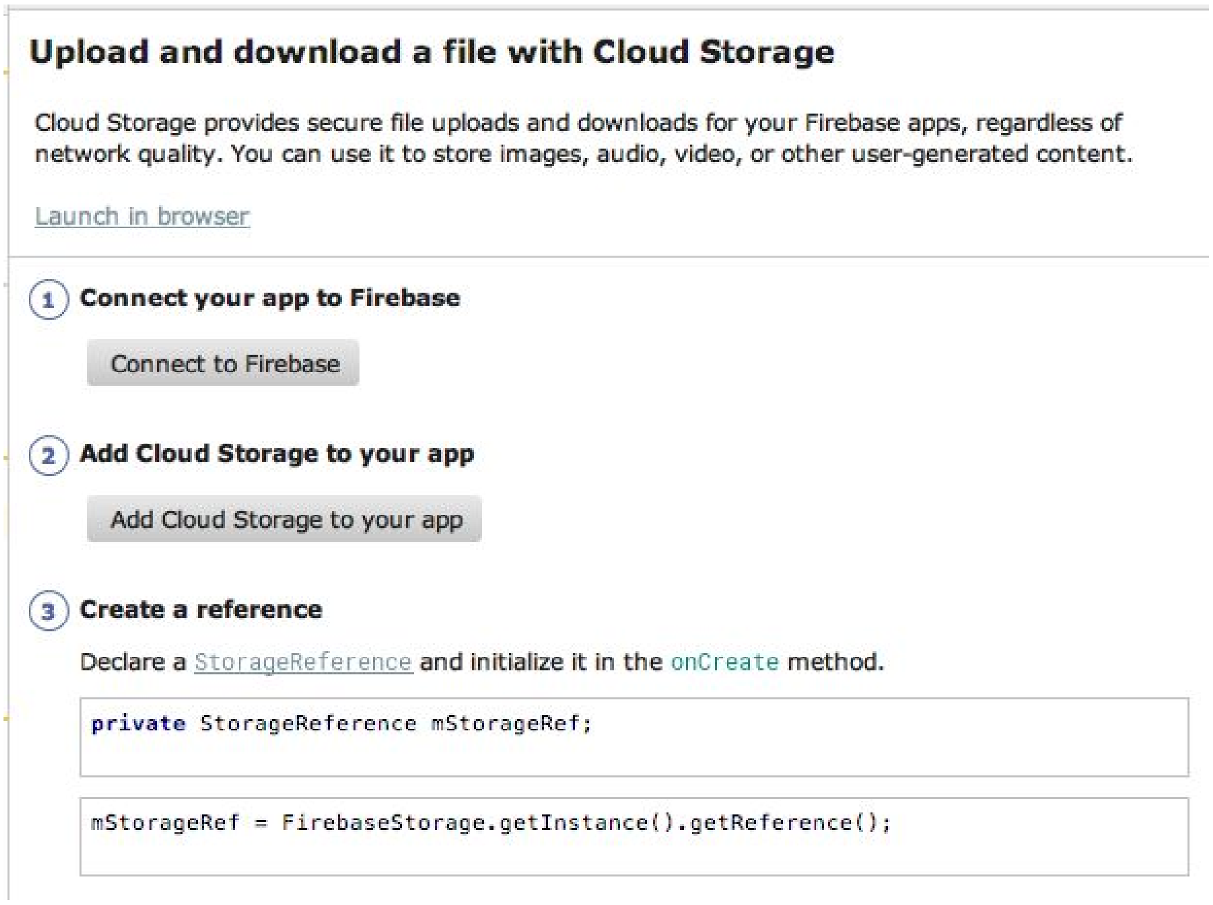
Realtime Database
Store and sync data in realtime across all connected clients. More info

Storage
Store and retrieve large files like images, audio, and video without writing server-side code. More info

Upload and download a file with Cloud Storage

Remote Config
Customize and experiment with app behavior using cloud-based configuration parameters. More info

Now, the process is much automated as shown:


Upload and download a file with Cloud Storage

Cloud Storage provides secure file uploads and downloads for your Firebase apps, regardless of network quality. You can use it to store images, audio, video, or other user-generated content.

Launch in browser

1 Connect your app to Firebase

Connect to Firebase

2 Add Cloud Storage to your app

Add Cloud Storage to your app

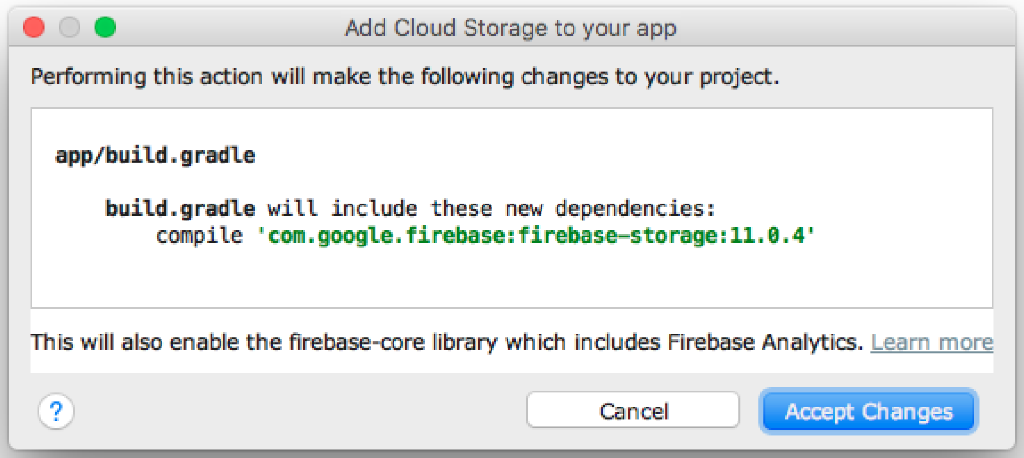
3 Create a reference

Declare a `StorageReference` and initialize it in the `onCreate` method.

```
private StorageReference mStorageRef;
```

```
mStorageRef = FirebaseStorage.getInstance().getReference();
```

Complete first 2 steps. When you do this, following dependencies will be asked to be added to the project:


Add Cloud Storage to your app

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

```
app/build.gradle

    build.gradle will include these new dependencies:
        compile 'com.google.firebase:firebase-storage:11.0.4'
```

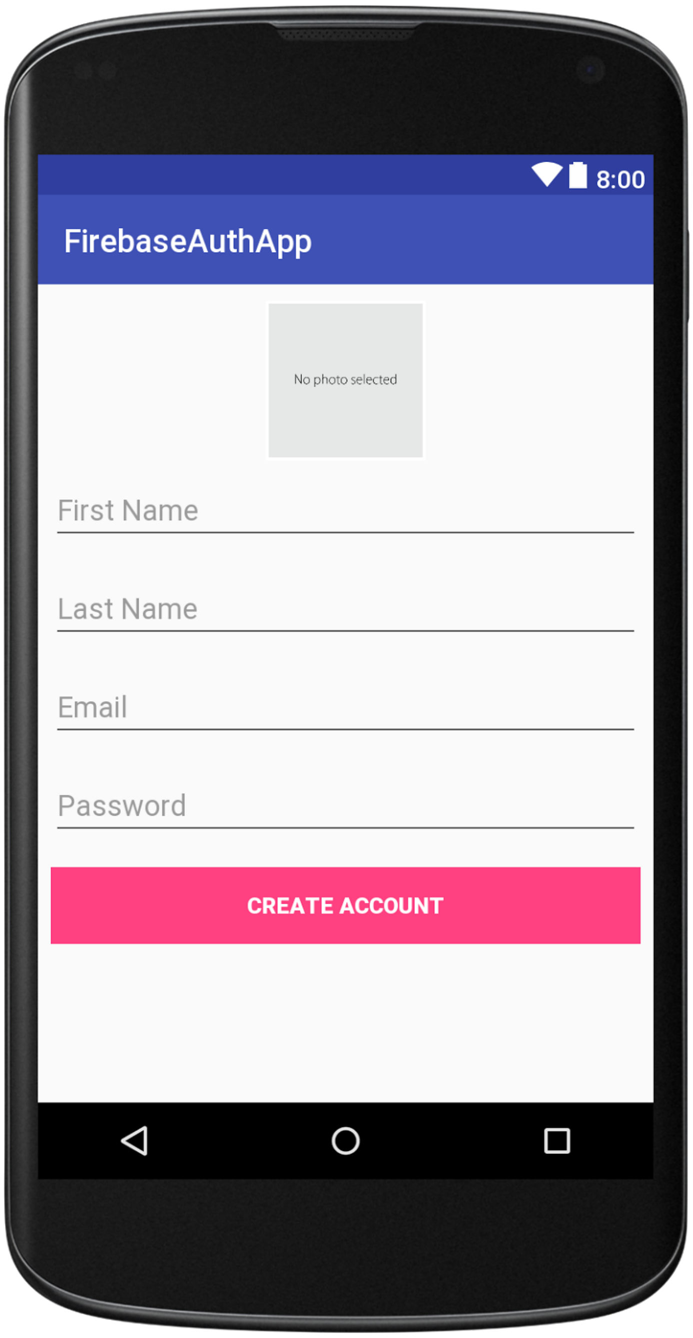
This will also enable the firebase-core library which includes Firebase Analytics. Learn more

Let the Gradle sync complete.

Design modification

We will now modify our UI to include an ImageView which will allow user to include an image from Camera or Gallery. This image will be then uploaded to Firebase Storage.

Our new UI will look like:



We just added an ImageButton where we can show our selected image.

First of all, we will add relevant dependencies to allow our user to crop an image we pick from gallery:

```
implementation 'com.squareup.picasso:picasso:2.5.2'
implementation
'com.theartofdev.edmodo:android-image-cropper:2.5.1'
```

Note that picasso was added for showing image later when we need to show user-specific image in our app.

Now, let us start by adding Storage specific information and allow user to pick an image from Gallery.

```
//profile pic references
private var profilePic: ImageButton? = null
private var resultUri: Uri? = null
private val GALLERY_CODE = 1
private val IMAGE_KEY = "user_profile_picture"
```

Now, we will initialise them:

```
profilePic = findViewById<View>(R.id.ib_profile_pic) as ImageButton

mFirebaseStorage = FirebaseStorage.getInstance().reference.child(IMAGE_KEY)
```

Finally, we set a listener on ImageButton:

```
profilePic!!.setOnClickListener { pickImageFromGallery() }
```

In above call, we can pick an image from Gallery with following function:

```
private fun pickImageFromGallery() {
  val galleryIntent = Intent()
  galleryIntent.action = Intent.ACTION_GET_CONTENT
  galleryIntent.type = "image/*"
  startActivityForResult(galleryIntent, GALLERY_CODE)
}
```

Once we have the image from gallery, the onActivityResult will be called where we crop the image.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent) {
  super.onActivityResult(requestCode, resultCode, data)

  if (requestCode == GALLERY_CODE && resultCode == Activity.RESULT_OK) {
    val mImageUri = data.data

    CropImage.activity(mImageUri)
      .setAspectRatio(1, 1)
      .setGuidelines(CropImageView.Guidelines.ON)
      .start(this)
  }

  if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE) {
    val result = CropImage.getActivityResult(data)
    if (resultCode == Activity.RESULT_OK) {
      resultUri = result.getUri()
      profilePic!!.setImageURI(resultUri)
    } else if (resultCode == CropImage.CROP_IMAGE_ACTIVITY_RESULT_ERROR_CODE) {
      val error = result.getError()
    }
  }
}
```

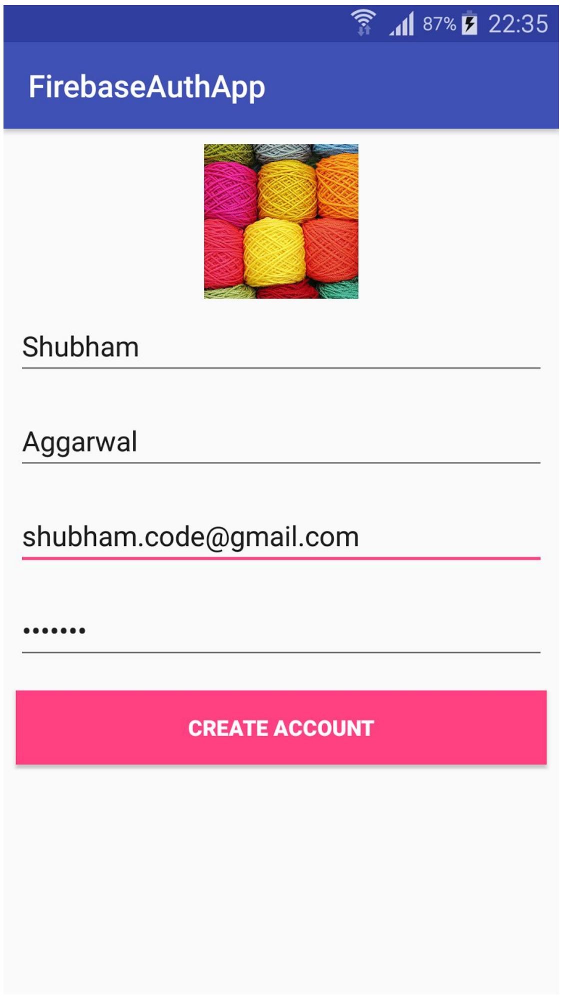
When we do this, we’re starting a new activity not present in our app. We must this to our Manifest file:

```
<activity
android:name="com.theartofdev.edmodo.cropper.CropImageActivity"/>
```

We can upload this image when user is registering an account. We will try to keep the code independent of user registration:

```
private fun uploadImage() {

   val imagePath =
mFirebaseStorage!!.child(IMAGE_KEY).child(resultUri!!.lastPathSegm
ent)

   imagePath.putFile(resultUri!!).addOnSuccessListener {
       mDatabaseReference!!
               .child(mAuth!!.currentUser!!.uid)
               .child("image")
               .setValue(resultUri.toString())
   }
}
```

Once I do this, I can pick an Image from Gallery and it will be shown in my ImageButton view:

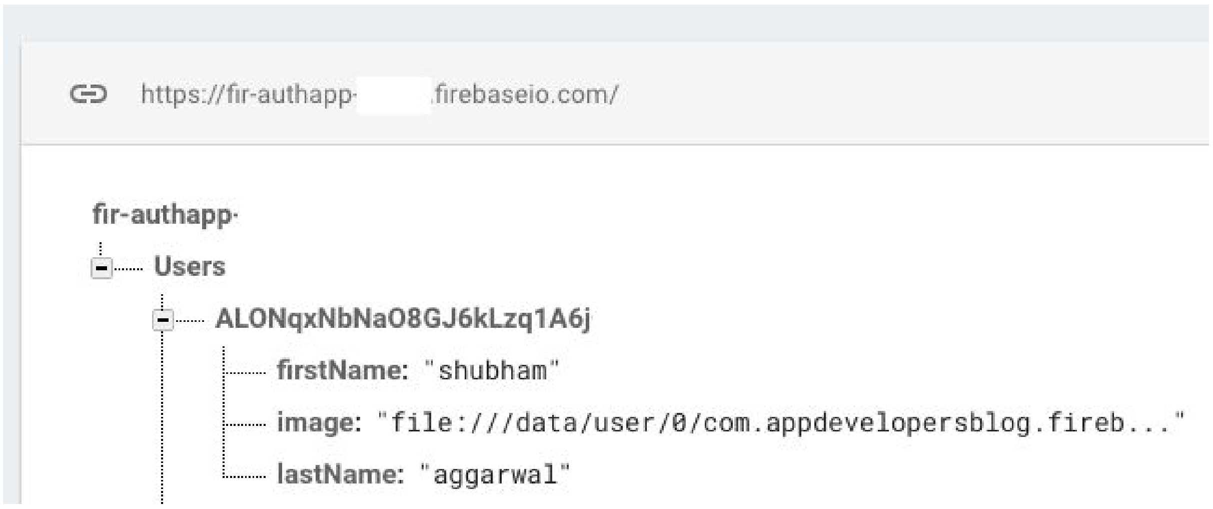


This method is called when a user registers an account. Let’s see the complete method from last lesson here, with additions:

```
private fun createNewAccount() {

   firstName = etFirstName?.text.toString()
   lastName = etLastName?.text.toString()
   email = etEmail?.text.toString()
   password = etPassword?.text.toString()

   if (!TextUtils.isEmpty(firstName) && !TextUtils.isEmpty(lastName)
           && !TextUtils.isEmpty(email) && !TextUtils.isEmpty(password)) {

      mProgressBar!!.setMessage("Registering User...")
      mProgressBar!!.show()

      mAuth!!.createUserWithEmailAndPassword(email!!, password!!)
        .addOnCompleteListener(this) { task ->

            if (task.isSuccessful) {
            // Sign in success, update UI with the signed-in user's information
            Log.d(TAG, "createUserWithEmail:success")

            val userId = mAuth!!.currentUser!!.uid

            verifyEmail()

            //update user profile information
            val currentUserDb = mDatabaseReference!!.child(userId)
            currentUserDb.child("firstName").setValue(firstName)
            currentUserDb.child("lastName").setValue(lastName)

            uploadImage()
            mProgressBar!!.hide()

            updateUserInfoAndUI()
          } else {
            mProgressBar!!.hide()
            // If sign in fails, display a message to the user.
            Log.w(TAG, "createUserWithEmail:failure", task.exception)
            Toast.makeText(this@CreateAccountActivity, "Authentication failed.",
                  Toast.LENGTH_SHORT).show()
            }
        }
    } else {
        Toast.makeText(this, "Enter all details", Toast.LENGTH_SHORT).show()
    }
}
```

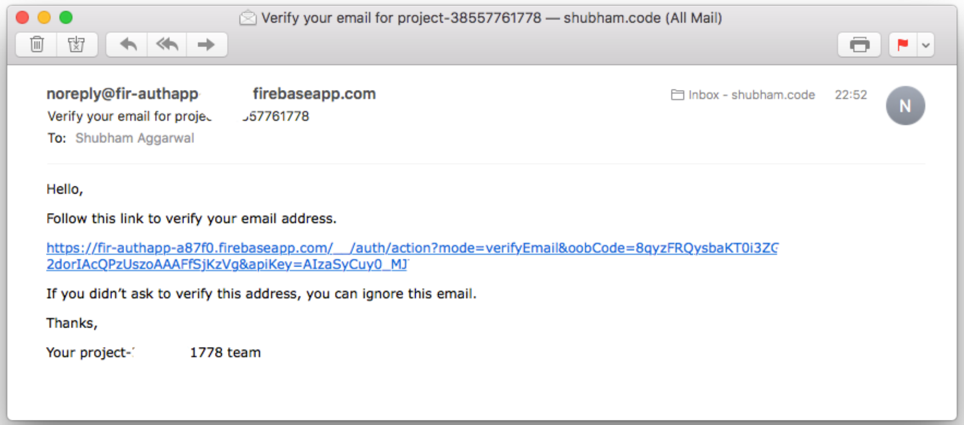
Now, you can see that image is uploaded once user has created an account.

We can cross check this in our Database. After uploading image, it has an extra field as shown:



```
https://fir-authapp-        .firebaseio.com/

fir-authapp-
  Users
    ALONqxNbNaO8GJ6kLzq1A6j
      firstName: "shubham"
      image: "file:///data/user/0/com.appdevelopersblog.fireb..."
      lastName: "aggarwal"
```

Not related to this lesson but we also see a verification email in our Inbox:


Verify your email for project-38557761778 — shubham.code (All Mail)

noreply@fir-authapp firebaseapp.com
Verify your email for proje 57761778
To: Shubham Aggarwal

Inbox - shubham.code 22:52

Hello,

Follow this link to verify your email address.

https://fir-authapp-a87f0.firebaseapp.com/__/auth/action?mode=verifyEmail&oobCode=8qyzFRQysbaKT0i3ZC
2dorIAcQPzUszoAAAFfSjKzVg&apiKey=AIzaSyCuy0_MJ

If you didn’t ask to verify this address, you can ignore this email.

Thanks,

Your project- 1778 team

Clearly, everything is working !

Retrieving image from Firebase

In our MainActivity where we show user specific information, we will now fetch user Image from Firebase. For this, we start by creating references:

```
private var profilePic: ImageButton? = null
```

Initialise them:

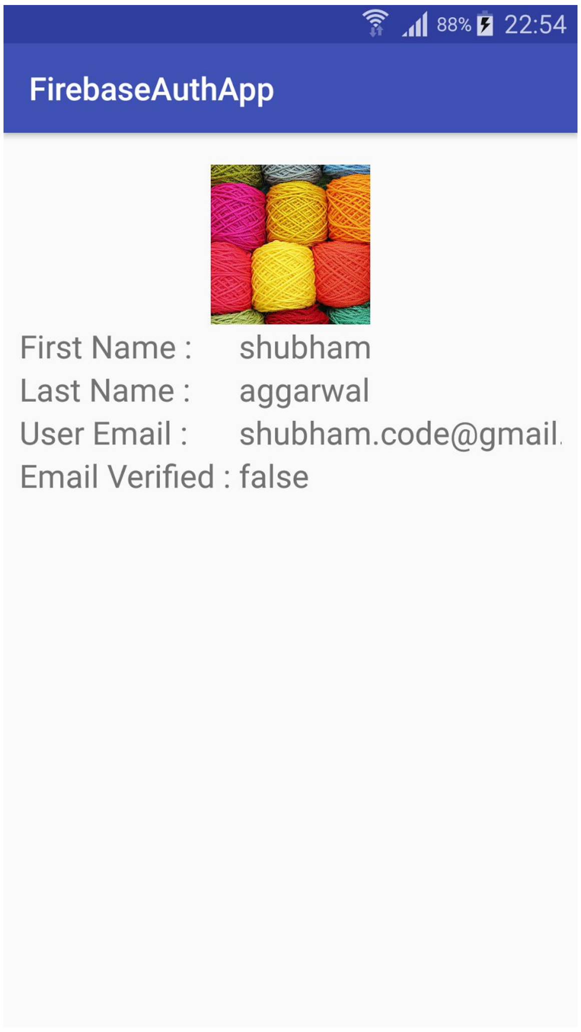
```
profilePic = findViewById<View>(R.id.ib_profile_pic) as
ImageButton
```

Finally when we fetch information:

```
mUserReference.addValueEventListener(object : ValueEventListener {
   override fun onDataChange(snapshot: DataSnapshot) {
       tvFirstName!!.text = snapshot.child("firstName").value as String
       tvLastName!!.text = snapshot.child("lastName").value as String

       Picasso.with(this@MainActivity)
               .load(snapshot.child("image").value as Uri?)
               .into(profilePic)
   }

   override fun onCancelled(databaseError: DatabaseError) {}
})
```

With picasso, we load the image from URI obtained from Firebase into our ImageView.

When we’re logged in, image is retreieved from firebase:



First Name : shubham
Last Name : aggarwal
User Email : shubham.code@gmail.
Email Verified : false

Isn’t that awesome!

Conclusion

In this lesson on Firebase Storage, we learned how to take leverage from the power of Firebase Storage to upload user files like images and personalise our app.

This app can be extended to a full feature app of user specific details and data. Go on and play with the code.