

## MADLAB Project with Firebase:

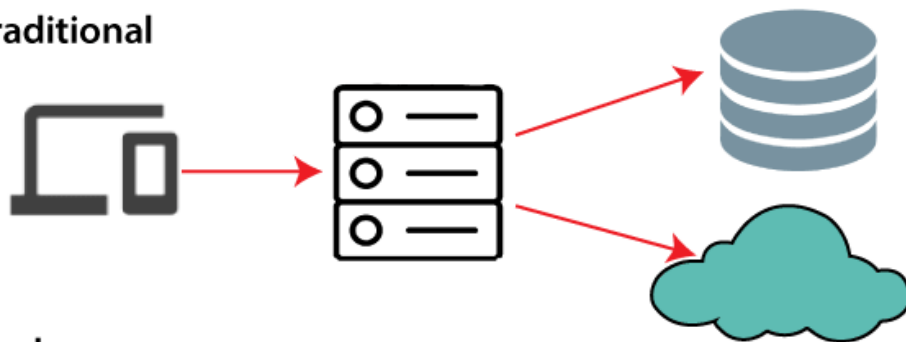
### Firebase:

## Introduction

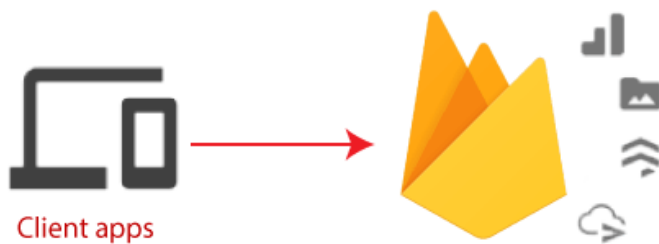
In the era of rapid prototyping, we can get bright ideas, but sometimes they are not applicable if they take too much work. Often, the back-end is the limiting factor - many considerations never apply to server-side coding due to lack of knowledge or time.

Firebase is a Backend-as-a-Service(BaaS) which started as a YC11 startup. It grew up into a next-generation app-development platform on Google Cloud Platform. Firebase (a NoSQL JSON database) is a real-time database that allows storing a list of objects in the form of a tree. We can synchronize data between different devices.

### Traditional



### Firebase



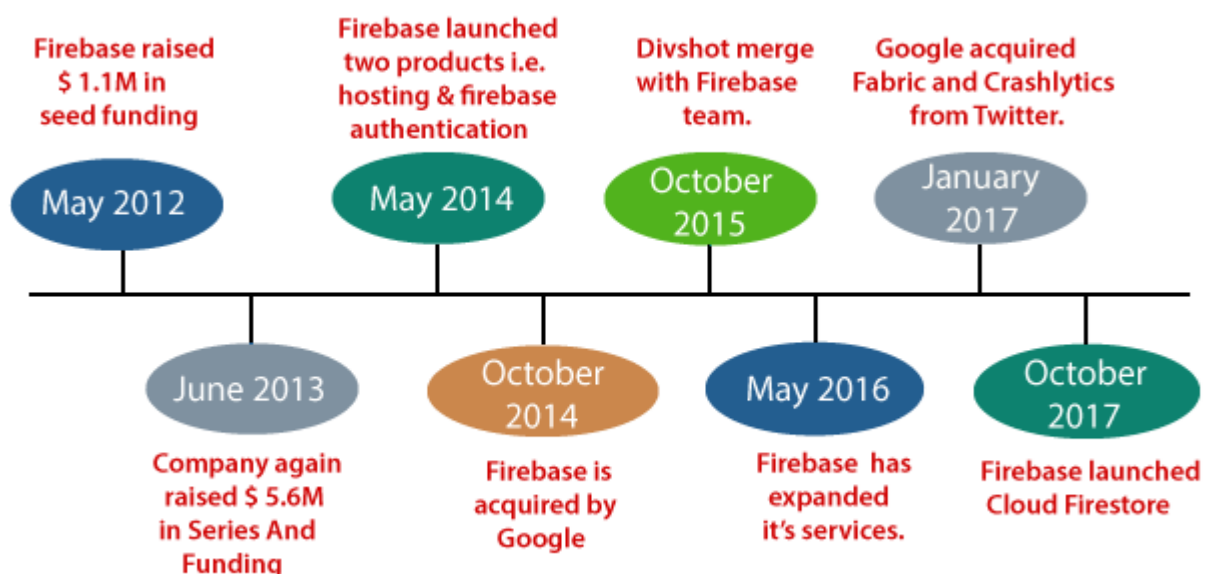
Google Firebase is Google-backed application development software which allows developers to develop **Android**, **iOS**, and **Web apps**. For reporting and fixing app crashes, tracking analytics, creating marketing and product experiments, firebase provides several tools. Firebase has three main services, i.e., a real-time database, user authentication, and hosting.

## History of Firebase

**Firebase** evolved from **Envolve**. Envolve is a prior startup founded by **James Tamplin** and **Andrew Lee** in 2011. Envolve provided developers an API which allowed

the integration of online chat functionality into their websites. After releasing the chat service, it found that the envlove was being used to pass application data, which were not chat messages. Developers used Envolve to sync application to separate the real-time architecture and the chat system which powered it. In September 2011, Tamplin and Lee founded firebase as a separate company. It was lastly launched to the public in April 2012.

Firebase Real-time Database was the first product of firebase. It is an API which syncs application data across Android, iOS, and Web devices. It gets stored on Firebase's cloud. Then the firebase real-time database helps the developers to build real-time, collaborative applications.



## Why use Firebase?

- Firebase manages real-time data in the database. So, it easily and quickly exchanges the data to and from the database. Hence, for developing mobile apps such as live streaming, chat messaging, etc., we can use Firebase.
- Firebase allows syncing real-time data across all devices - iOS, Android, and Web - without refreshing the screen.
- Firebase provides integration to Google Advertising, AdMob, Data Studio, BigQuery DoubleClick, Play Store, and Slack to develop our apps with efficient and accurate management and maintenance.
- Everything from databases, analytics to crash reports are included in Firebase. So, the app development team can stay focused on improving the user experience.
- Firebase applications can be deployed over a secured connection to the firebase server.

- Firebase offers a simple control dashboard.
- It offers a number of useful services to choose from.

## Pros and Cons of Firebase

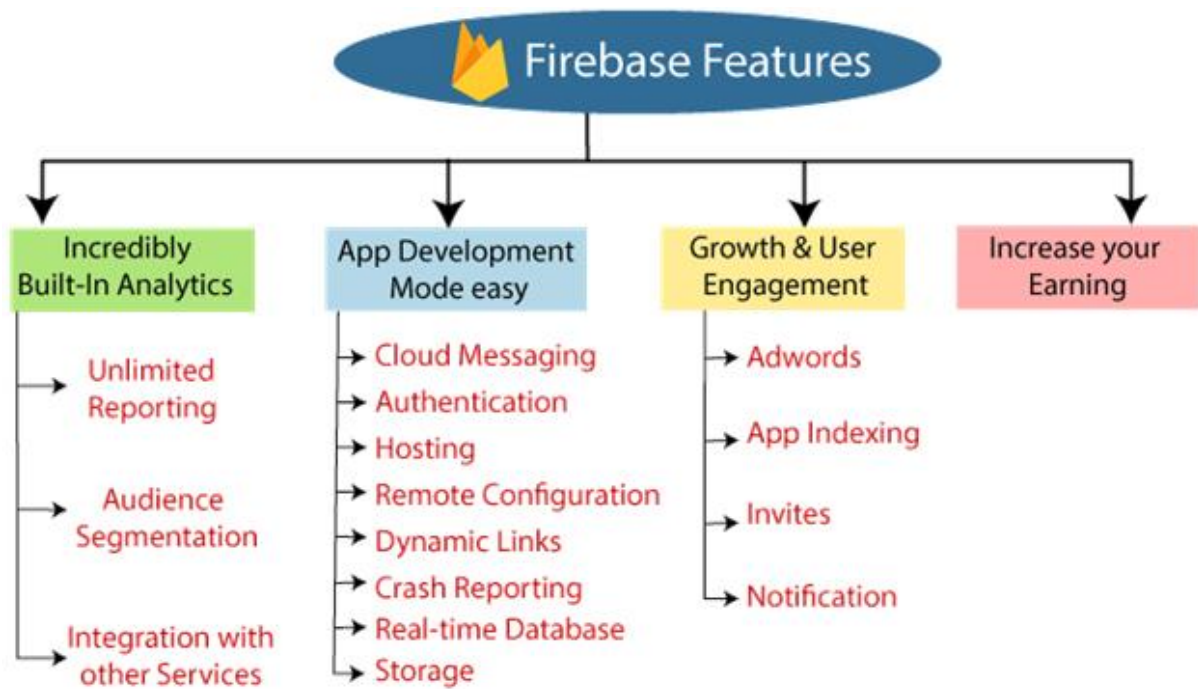
Firebase has a lot of pros or advantages. Apart from the advantages, it has disadvantages too. Let's take a look at these advantages and disadvantages:

### Pros

- Firebase is a real-time database.
- It has massive storage size potential.
- Firebase is serverless.
- It is highly secure.
- It is the most advanced hosted BaaS solution.
- It has minimal setup.
- It provides simple serialization of app state.
- We can easily access data, files, auth, and more.
- There is no server infrastructure required to power apps with data.
- It has JSON storage, which means no barrier between data and objects.

### Cons

- Firebase is not widely used, or battle-tested for enterprises.
- It has very limited querying and indexing.
- It provides no aggregation.
- It has no map-reduce functionality.
- It cannot query or list users or stored files.



## Firestore



We have two options with Firebase, i.e., Firebase Real-time Database and Cloud Firestore. Cloud Firestore is newer, but it is not replacing the Firebase Real-time Database. Cloud Firestore is a flexible as well as scalable NoSQL cloud database. It is used to store and sync data for client and server-side development. It is used for mobile, web, and server development from Google Cloud Platform and Firebase. Like the Firebase Real-time Database, it keeps syncing our data via real-time listeners to the client app. It provides offline support for mobile and web so we can create responsive apps that work regardless of network latency or Internet connectivity.

Cloud Firestore also provides seamless integration with Google Cloud Platform products and other Firebase, including cloud functions.

# Key capabilities

## Flexibility

The Firestore data model supports a flexible, hierarchical data structure. It stores our data in the document, which is organized into a collection. In Firestore, the documents can contain complex nested objects rather than sub-collections.

## Expressive querying

In Firestore, we can use queries for retrieving specific, individual documents or for retrieving all the documents in a collection that match our query parameters. Our queries combine filtering and sorting and can include multiple, chained filters. The query performance is proportional to the size of our result set because queries are indexed by default.

## Real-time updates

Firestore is quite similar to Firebase Realtime Database. Firestore also uses data synchronization for updating data on any connected device. It is designed for making simple one time fetch queries efficiently.

## Offline Supports

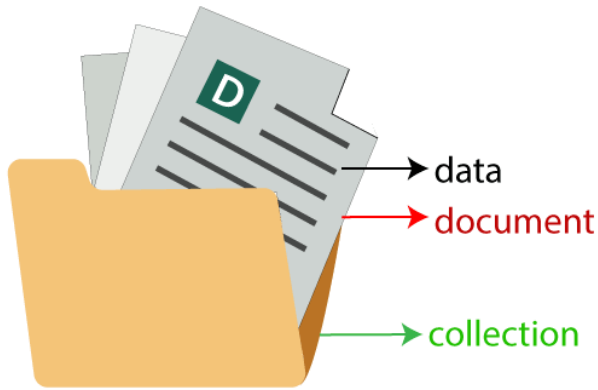
Cloud Firestore enables us to make a cache of the app data, which it is actively using. This makes the app to read, write, query, and listen to the data even when the device is offline. When the device comes in an online mode, Cloud Firestore synchronizes the local changes back to it.

## Designed to scale

Cloud Firestore provides us the best infrastructure of the Google Cloud Platform: automated multi-region data replication, atomic batch operations, strong consistency guarantees, and real transaction support. We designed Cloud Firestore for handling the toughest database workloads from the world's largest apps.

# How does it work?

Cloud Firestore, a cloud-hosted, NoSQL database, is accessed directly through the native SDK by our iOS, Android, and web apps. In addition to REST and RPC APIs, Cloud Firestore is also available in native Node.js, Java, Python, and Go SDKs.



After Cloud Firestore's NoSQL data model, we can store data in documents that have field mappings for values. The documents are stored in a container called collections. These containers are used to organize our data and create queries. There are different data types, from simple string and numbers to complex nested objects, supported by documents. We can also create sub-collection within a document and create a hierarchical data structure that scales to the growth of our database. The Firestore data model supports whatever data structure works best for our app.

Additionally, the query in Cloud Firestore is expressive, efficient, and flexible. The shallow queries are created to retrieve data at the document level without the need to retrieve the entire collection or any nested subdivision. Add sorting, filtering, and limits for our queries or cursors to index the result. Add a real-time listener to our app for keeping the data running. Every time it is updated without recovering our entire database.

Adding real-time listeners to our app informs us with a data snapshot whenever our customer apps are changing data, only getting new changes.

For protecting our data access in Cloud Firestore, Firebase authentication, and Cloud Firestore security rules are used for Identity and Access Management (IAM).

## Firestore vs. Realtime Database

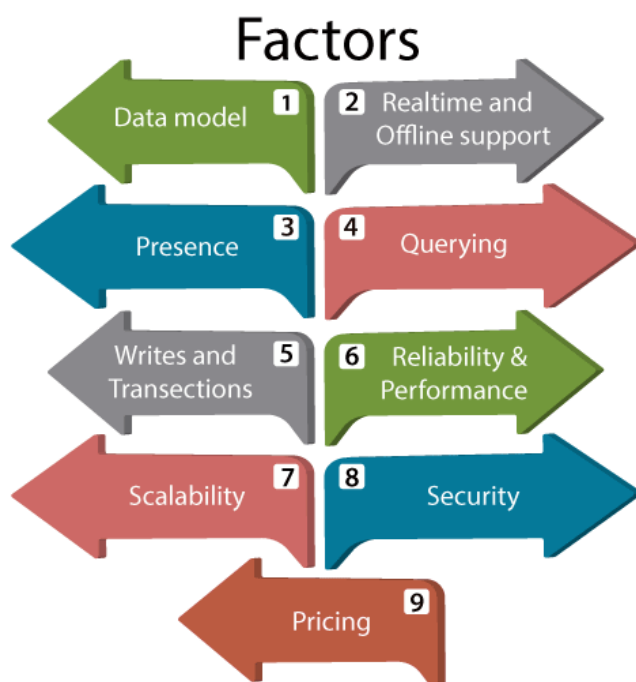
For the development of any application such as desktop, mobile, and web, a database is our prerequisite to store data. Storage is required where we can store and manipulate the data in such a way that every application can access the same data. Firebase provides Firestore and Firebase Realtime Database. These databases are cloud-based, client accessible database solutions which support real-time data syncing.

## Cloud Firestore

Firestore is the newest database used for mobile app development. Cloud Firestore has richer features, faster queries, and scale further than the real-time database. Cloud Firestore is a flexible as well as scalable NoSQL cloud database. It is used to store and sync data for client and server-side development. It is used for mobile, web, and server development from Google Cloud Platform and Firebase.

## Realtime Database

It is the original database of Firebase. It is a low latency solution and efficient for mobile applications which require synced states across the client's real-time. The Firebase Realtime Database is a cloud-hosted database in which data is stored as JSON. The data is synchronized in real-time to every connected client.



Realtime Database	Cloud Firestore
In the Realtime database, data is stored as one large JSON tree.	In Cloud Firestore, data is stored as a collection of documents.
The data is simple, so it is very easy to store.	In documents, simple data is very easy to store. Documents are very similar to JSON.
The complex, hierarchical data is difficult to organize on a large scale.	By using sub-collections, the complex and hierarchical data is easy to organize on a large scale. It requires very less de-normalization and data flattening.

Realtime Database	Cloud Firestore
Offline support for Android and iOS clients.	Offline support for Android and iOS clients.

Realtime Database	Cloud Firestore
Realtime database supports presence.	The Firestore doesn't support presence natively. We can take advantage of the help of real-time databases by syncing Cloud Firestore and Firebase Realtime database using cloud functions.

Realtime Database	Cloud Firestore
It allows us to perform deep queries with limited sorting and filtering functionality.	It allows us to perform indexed queries with compound sorting and filtering.



The queries can sort or filter on a property, but not for both.	In Cloud Firestore, we can chain filter and combine filtering and sorting on a property in a single query.
It has deep queries by default, so it always returns the entire subtree.	It has shallow queries, so these queries return documents in a particular collection or collection group. These queries don't return sub collection data.
Realtime database queries can access data at any point, down to individual leaf-node values in the JSON tree.	Cloud Firestore queries always return whole documents.
These queries don't require an index. But, the performance of certain queries degrades as our dataset grows.	Cloud Firestore queries are indexed by default. The performance of the query is proportional to the size of our result set rather than a dataset.

## Scalability

Realtime Database	Cloud Firestore
Sharding is required for scaling.	Here, scaling is automatic.
It scales approximately 200000 concurrent connections, and 1000 writes per second in a single database. Beyond that, it is required to divide the data into multiple databases.	The scaling is done automatically. At present, it can scale up to 1 million concurrent connections, and 10000 writes per second. The limits can be increased in the future.
For an individual piece of data, there is no local limit on write rates.	It has limits for individual documents or indexes.

## Security

Realtime Database	Cloud Firestore
-------------------	-----------------

It uses cascading rules-language, which separates authorization and validation.	It uses no-cascading rules which combine authorization and validation.
By Realtime database rules, reads and writes from mobile SDKs secured.	With the help of Cloud Firestore security rules, reads and writes from server SDKs secured.
The read and write rules are a cascade.	The rules are not cascaded unless we use a wildcard.
The data is validated separately using the validation rule.	The rules can constrain queries. If the result of the query might contain data, the user doesn't have to access it, then the entire query fails.

## Pricing

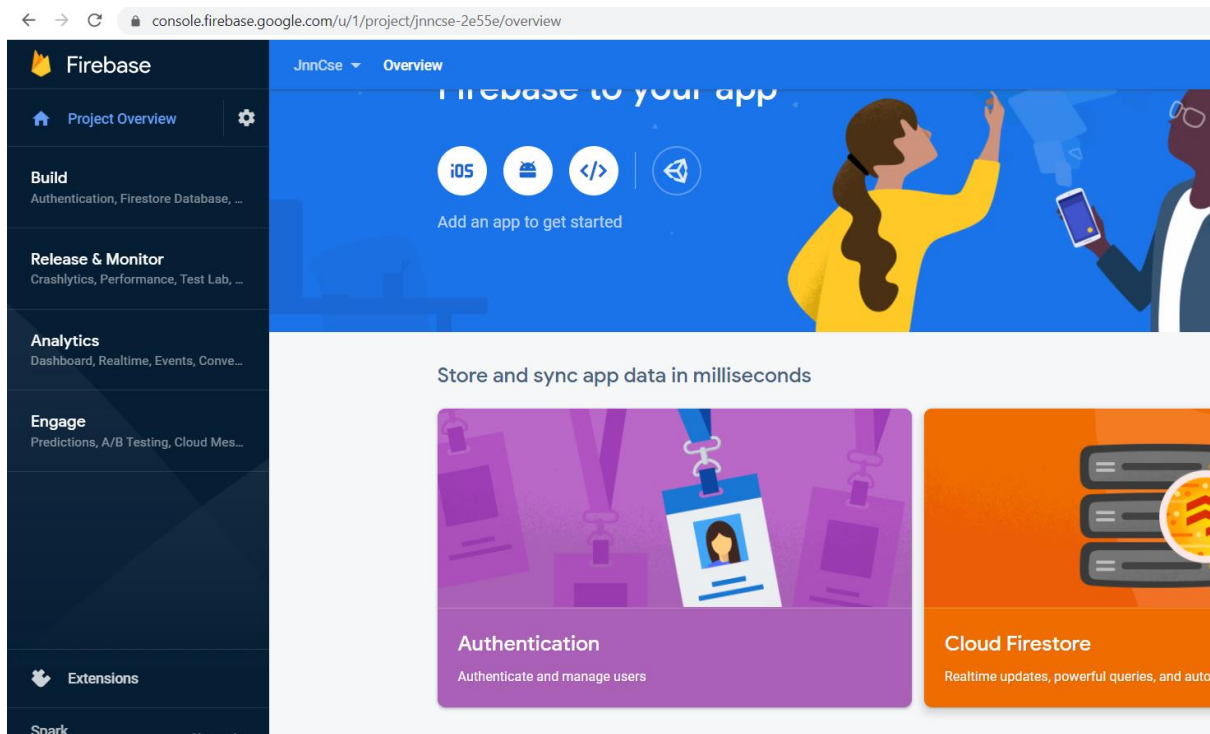
Realtime Database	Cloud Firestore
Charges for bandwidth and storage only, but at a higher rate.	Mainly the operations performed in our database and are charged at a low rate, bandwidth, and storage.

### Creating a firebase account:

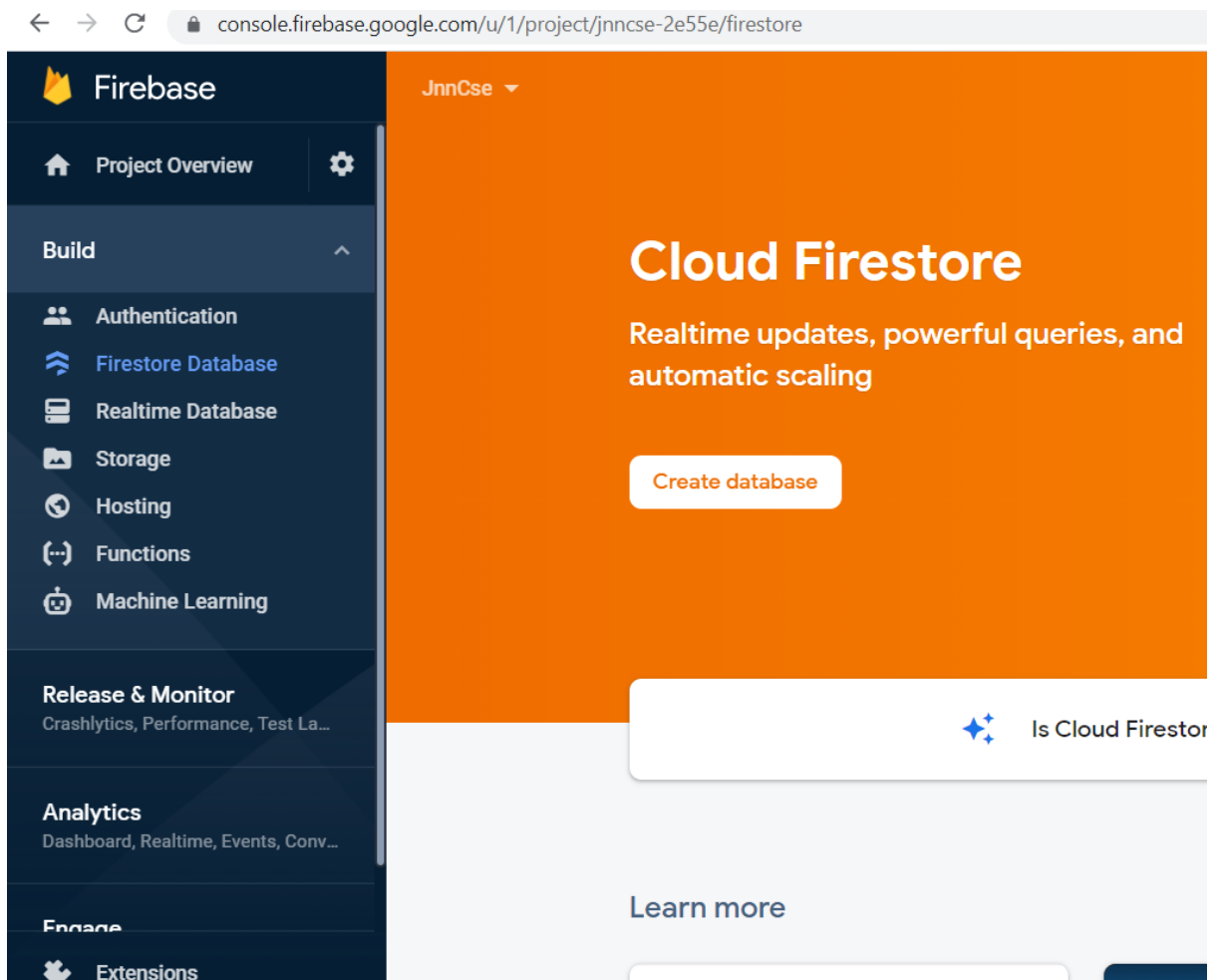
One need to have a gmail account to start using firebase.

URL: <https://console.firebase.google.com/>

Create a project inside it.






## Goto Cloud Firestore




## Create database option


← → ↺ console.firebase.google.com/u/1/project/jnncse-2e55e/firestore/data~2F


 **Firestore**


 **Project Overview** 


**Build**


 **Authentication**


 **Firestore Database**

 **Realtime Database**

 **Storage**

 **Hosting**

 **Functions**

 **Machine Learning**

**Release & Monitor**  
Crashlytics, Performance, Test La...


**Analytics**  
Dashboard, Realtime, Events, Conv...


JnnCse ▾

# Cloud Firestore

[Data](#) [Rules](#) [Indexes](#) [Usage](#)

Prototype and test end-to-end with the L




 jnncse-2e55e

[+ Start collection](#)


## Change rules to make it accessible

### Cloud Firestore

[Data](#) [Rules](#) [Indexes](#) [Usage](#)

 Published changes can take up

[Edit rules](#) [Monitor rules](#)



Guard your data with rules that define who has access to it and how it is structured

[View the docs](#)

```
1 rules_version = '2';
2 service cloud.firestore {
3   match /databases/{database}/documents {
4     match /{document=**} {
5       allow read, write;
6     }
7   }
8 }
```

## Android:

Create a project with empty activity

Add firebase dependences:

(i) **build.gradle of Project**

Add this in dependencies block:

```
classpath 'com.google.gms:google-services:4.3.8'
```

(ii) **build.gradle of App**

Add following:

Before android section:

```
apply plugin: 'com.google.gms.google-services'
```

Inside default section

```
multiDexEnabled true
```

**Inside dependencies section:**

```
implementation 'androidx.multidex:multidex:2.0.1'
```

```
implementation platform('com.google.firebase:firebase-bom:28.1.0')
```

```
implementation 'com.google.firebase:firebase-analytics-ktx'
```

```
implementation 'com.firebaseui:firebase-ui-firestore:6.2.1'
```

```
implementation 'com.github.bumptech.glide:glide:4.11.0'
```

```
annotationProcessor
```

```
'com.github.bumptech.glide:compiler:4.11.0'
```

**Click SyncNow for Gradle**

**Note: Syncing requires internet, so if internet is disconnected, results are indeterminate**

activity\_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <EditText
            android:id="@+id/pid"
            android:layout_width="100sp"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Product ID"
            android:inputType="textPersonName" />

        <EditText
            android:id="@+id/pname"
            android:layout_width="200sp"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Product Name"
            android:inputType="textPersonName" />

        <EditText
            android:id="@+id/cost"
            android:layout_width="80sp"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="Cost"
            android:inputType="numberDecimal" />

        <Spinner
            android:id="@+id/ptyp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:entries="@array/product_cat" />

        <Button
            android:id="@+id/button"
            android:layout_width="200sp"
```

```

        android:layout_height="wrap_content"
        android:onClick="add"
        android:text="Add/Update" />

        <androidx.recyclerview.widget.RecyclerView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:id="@+id/rv"/>

    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>

```

MainActivity.kt:

```

class MainActivity : AppCompatActivity() {
    private var mAuth: FirebaseAuth?=null
    private var mFirebaseDatabaseInstances:FirebaseFirestore?=null
    lateinit var cv:RecyclerView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //Get Firebase Instances
        val policy =
StrictMode.ThreadPolicy.Builder().permitAll().build()
        StrictMode.setThreadPolicy(policy)
        mAuth=FirebaseAuth.getInstance()
        mFirebaseDatabaseInstances= FirebaseFirestore.getInstance()
        cv=findViewById<View>(R.id.rv) as RecyclerView
        loadData()

    }
    fun add(v: View?)
    {
        var pidA=findViewById<View>(R.id.pid) as EditText
        var pnameA=findViewById<View>(R.id.pname) as EditText
        var pcostA=findViewById<View>(R.id.cost) as EditText
        var ptypA=findViewById<View>(R.id.ptyp) as Spinner
        var pid=pidA.text.toString()
        var pname=pnameA.text.toString()
        var pcost=pcostA.text.toString().toDouble()
        var ptyp=ptypA.selectedItem.toString()
        val
docRef=mFirebaseDatabaseInstances?.collection("products")?.document(
pid!!)
        var url=""
docRef?.get()?.addOnSuccessListener { documentSnapshot ->
    val prod=documentSnapshot.toObject(Product::class.java) as
Product

```

```

if(prod!=null)
    url=prod!!.url

    //Toast.makeText(this,url,Toast.LENGTH_LONG).show()
    var p=Product(pid,pname,pcost,url,ptyp)

    FirebaseDatabaseInstances?.collection("products)?.document(p
    id!)??.set(p)
    Toast.makeText(this,"Product added
    successfully",Toast.LENGTH_LONG).show()

}

}

fun loadData()
{
    val query: Query = FirebaseFirestore.getInstance()
        .collection("products")
    val options: FirestoreRecyclerOptions<Product?> =
    FirestoreRecyclerOptions.Builder<Product>()
        .setQuery(query, Product::class.java)
        .build()
    val adapter: FirestoreRecyclerAdapter<*, *> = object :
    FirestoreRecyclerAdapter<Product?,
    RecyclerView.ViewHolder?>(options) {

        override fun onCreateViewHolder(group: ViewGroup, i:
        Int): ProductHolder {
            // Using a custom layout called R.layout.message for
            each item, we create a new instance of the viewholder
            val view: View = LayoutInflater.from(group.context)
                .inflate(R.layout.list_data, group, false)
            return ProductHolder(view)
        }

        override fun onBindViewHolder(holder:
        RecyclerView.ViewHolder, position: Int, model: Product) {

            (holder.itemView.findViewById<View>(R.id.tpid) as
            TextView).setText("Product ID: "+model.pid)
            (holder.itemView.findViewById<View>(R.id.tpname) as
            TextView).setText("Product Name:" + model.pname)
            (holder.itemView.findViewById<View>(R.id.tpcost) as
            TextView).setText("Product Cost:" + model.cost.toString())
            (holder.itemView.findViewById<View>(R.id.tptyp) as
            TextView).setText("Product Category:" + model.typ)
        }
    }
}

```



```

        if(model.url!="") {
            /*Thread(
                Runnable {

                    var url = URL(model.url)
                    runOnUiThread {
                        var bm =
BitmapFactory.decodeStream(url.openConnection().getInputStream())

(holder.itemView.findViewById<View>(R.id.imageView) as
ImageView).setImageBitmap(bm)

                    }
                }

            ).start();*/
            var url = URL(model.url)
            var
imageView=(holder.itemView.findViewById<View>(R.id.imageView) as
ImageView)

Glide.with(this@MainActivity).load(model.url).placeholder(R.drawable
.prod).error(R.drawable.prod).override(600,600).into(imageView);
        }

        (holder.itemView.findViewById<View>(R.id.delBtn)
as ImageButton).setOnClickListener {

mFirebaseDatabaseInstances!!.collection("products").document(model.p
id!!).delete()

                .addOnSuccessListener {
Toast.makeText(applicationContext, "Successfully deleted ",
Toast.LENGTH_SHORT).show() }
                .addOnFailureListener {
Toast.makeText(applicationContext, "Unable to delete",
Toast.LENGTH_SHORT).show() }
        }

        (holder.itemView.findViewById<View>(R.id.uploadBtn)
as ImageButton).setOnClickListener {
            var i=
Intent(applicationContext,UploadActivity::class.java)
            i.putExtra("pid",model.pid)
            i.putExtra("pname",model.pname)
            i.putExtra("cost",model.cost.toString())
            i.putExtra("typ",model.typ)
            startActivity(i)
        }
    }
}

```

```

    }
    //Final step, where "mRecyclerView" is defined in your xml layout as
    //the recyclerview
    //Final step, where "mRecyclerView" is defined in your xml layout as
    //the recyclerview

```

```

        cv.LayoutManager = LinearLayoutManager(this)
        cv.adapter=adapter
        adapter.startListening()
    }

```

```

}
class Product
{

```

```

    var pid=""
    var pname=""
    var cost=0.0
    var typ=""
    var url=""

```

```

constructor(pid:String,pname:String,cost:Double,url:String,typ:Strin
g)

```

```

    {
        this.pid=pid
        this.pname=pname
        this.cost=cost
        this.url=url
        this.typ=typ
    }

```

```

    constructor()

```

```

}
class ProductHolder(itemView:
View):RecyclerView.ViewHolder(itemView) {

```

```

    val tpid: TextView
    val tpname: TextView
    val tpcost: TextView
    val tptyp:TextView
    init {
        tpid=itemView.findViewById(R.id.tpid)

        tpname=itemView.findViewById(R.id.tpname)
        tpcost=itemView.findViewById(R.id.tpcost)
        tptyp=itemView.findViewById(R.id.tptyp)
    }

```

```
}
```

For all Firebase related APIs in code, click add dependencies and dependencies are automatically added

Create a new layout file (inside res/layout folder)  
layout\_data.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:layout_marginBottom="30dp"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"
        android:layout_marginBottom="25dp"
        android:orientation="horizontal">

        <ImageView
            android:id="@+id/imageView"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_marginEnd="20sp"
            android:layout_marginRight="20sp"
            android:src="@drawable/prod" />

        <LinearLayout
            android:layout_width="130dp"
            android:layout_height="wrap_content"
            android:layout_marginEnd="20sp"
            android:layout_marginRight="20sp"
            android:orientation="vertical">

            <TextView
                android:id="@+id/tpid"
                android:layout_width="wrap_content"

                android:layout_height="wrap_content"
```

```
        android:layout_gravity="start"
        android:gravity="left"
        android:text="ProductID"
        android:textColor="#E34B4B"
        android:textSize="14dp" />
```

```
<TextView
    android:id="@+id/tpname"
    android:layout_width="wrap_content"

    android:layout_height="wrap_content"
    android:layout_gravity="start"
    android:layout_marginTop="5sp"
    android:text="Product Name"
    android:textColor="#E36969"
    android:textSize="14dp" />
```

```
<TextView
    android:id="@+id/tpcost"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="start"
    android:layout_marginTop="5sp"
    android:text="Product Cost"
    android:textColor="#C65151"
    android:textSize="14dp" />
```

```
<TextView
    android:id="@+id/tptyp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Product Category"
    android:textColor="#A32626" />
```

```
</LinearLayout>
```

```
<LinearLayout
    android:layout_width="83dp"
    android:layout_height="46dp"
    android:orientation="horizontal">
```

```
    <ImageButton
        android:id="@+id/delBtn"
        android:layout_width="20dp"
        android:layout_height="60dp"
        android:layout_marginTop="10dp"
        android:background="#FFFFFF"
        android:src="@android:drawable/ic_menu_delete"
        android:tint="#F40F0F" />
```

```

<ImageButton
    android:id="@+id/uploadBtn"
    android:layout_width="20dp"
    android:layout_height="60dp"
    android:layout_marginStart="10sp"
    android:layout_marginLeft="10sp"
    android:layout_marginTop="10dp"
    android:background="#FFFFFF"
    android:src="@android:drawable/ic_menu_upload"
    android:tint="#045C2F" />

    </LinearLayout>

</LinearLayout>

</androidx.cardview.widget.CardView>

```

**Also put a prod image**

Further create array of product categories inside strings.xml of res/values folder:

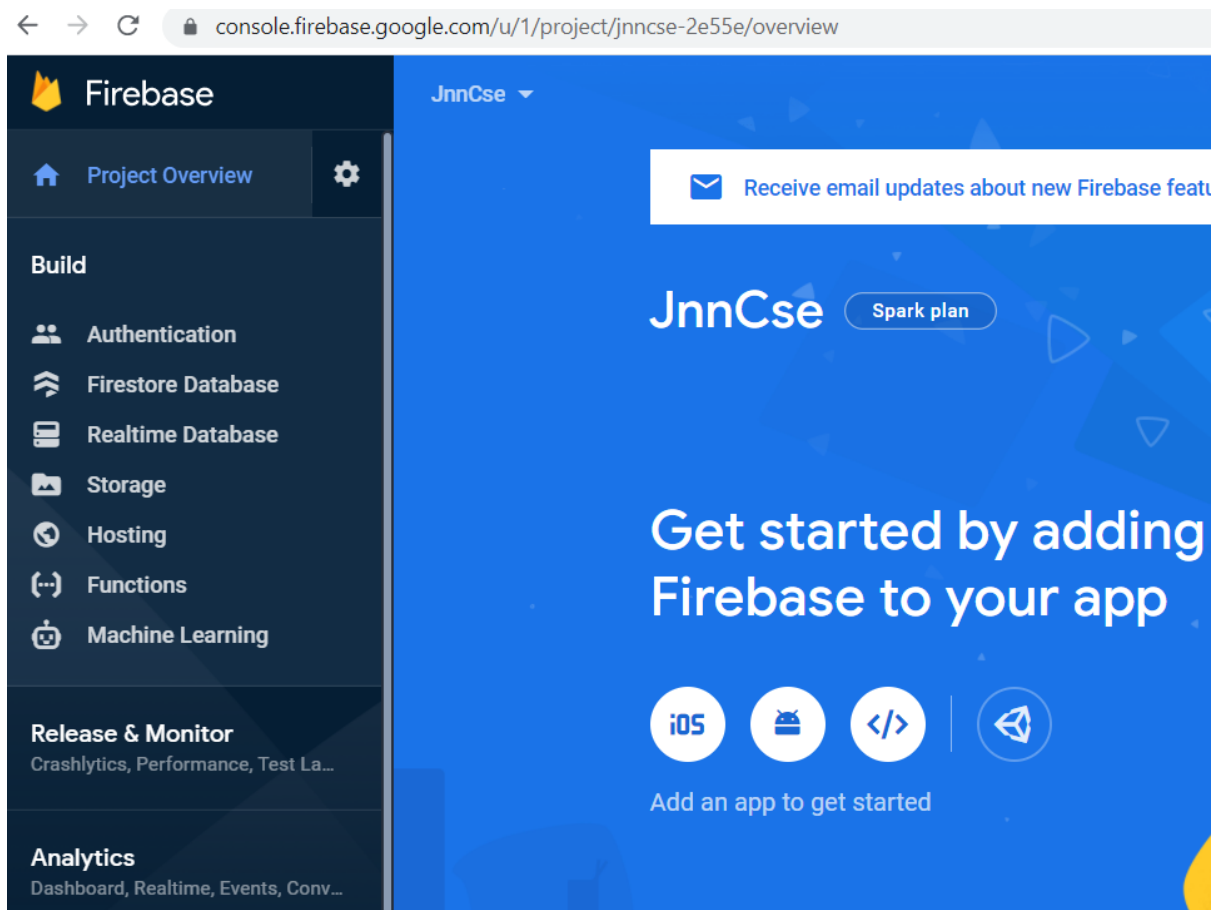
```

<resources>
    <string name="app_name">PFB</string>
    <string-array name="product_cat">
        <item>Household</item>
        <item>Electronic</item>
        <item>Eatable</item>
        <item>Others</item>
    </string-array>
</resources>

```

**Create next Activity-Upload**

**In Firebase link this app**



Use Add an app

- × Add Firebase to your Android app

### 1 Register app

Android package name 

```
|com.company.appname
```

App nickname (optional) 

My Android App

Debug signing certificate SHA-1 (optional) 

00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:(

Required for Dynamic Links, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.

Register app

**Package name should be same as android app package name**

← → ↻ console.firebase.google.com/u/1/project/jnncse-2e55e/overview

## × Add Firebase to your Android app

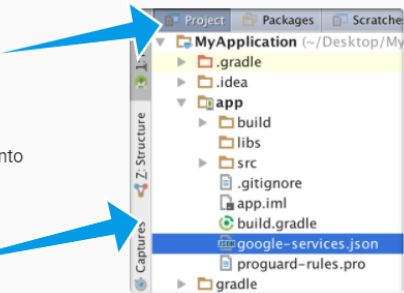
✓ Register app  
Android package name: com.example.pfb, App nickname: PFB

2 Download config file Instructions for Android Studio below | [Unity](#) [C++](#)

[Download google-services.json](#)

Switch to the Project view in Android Studio to see your project root directory.

Move the google-services.json file you just downloaded into your Android app module root directory.



google-services.json

**Download google-services.json and put inside app folder**

**Goto Storage option in Firebase project and create a folder uploads:**

← → ↻ console.firebase.google.com/u/1/project/jnncse-2e55e/storage/jnncse-2e55e.appspot.com/files

Go to docs

Storage

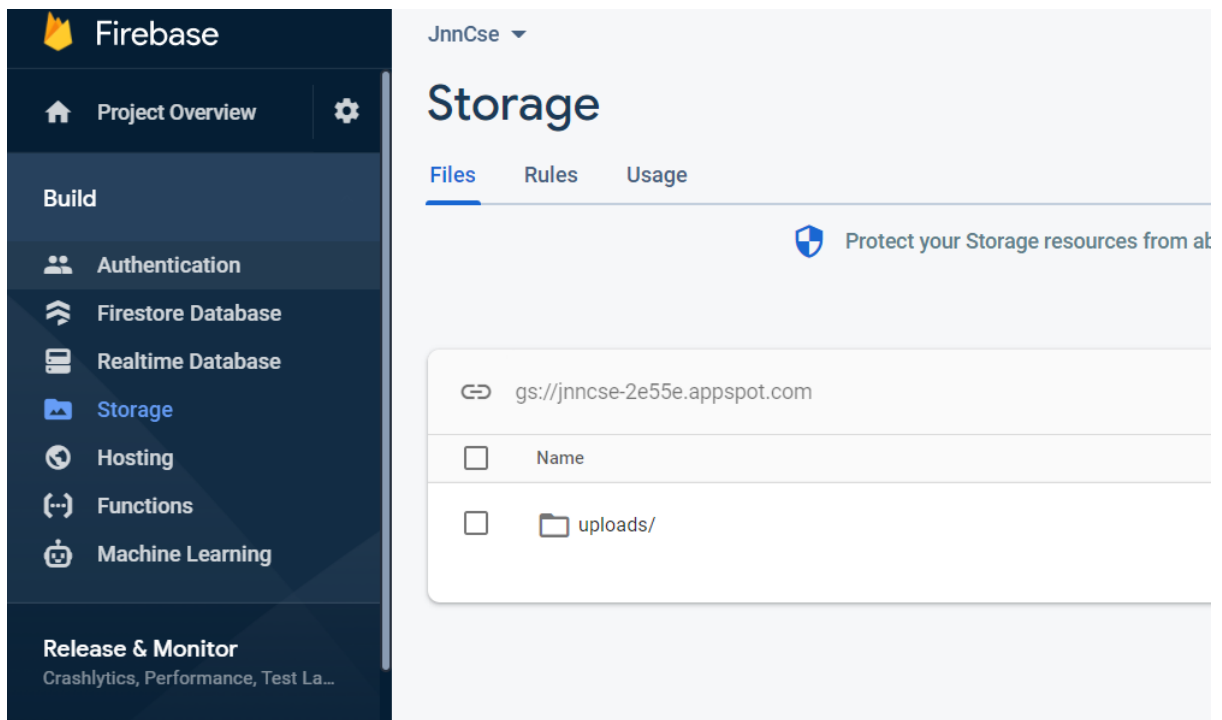
Files Rules Usage

Protect your Storage resources from abuse, such as billing fraud or phishing [Configure App Check](#) ×

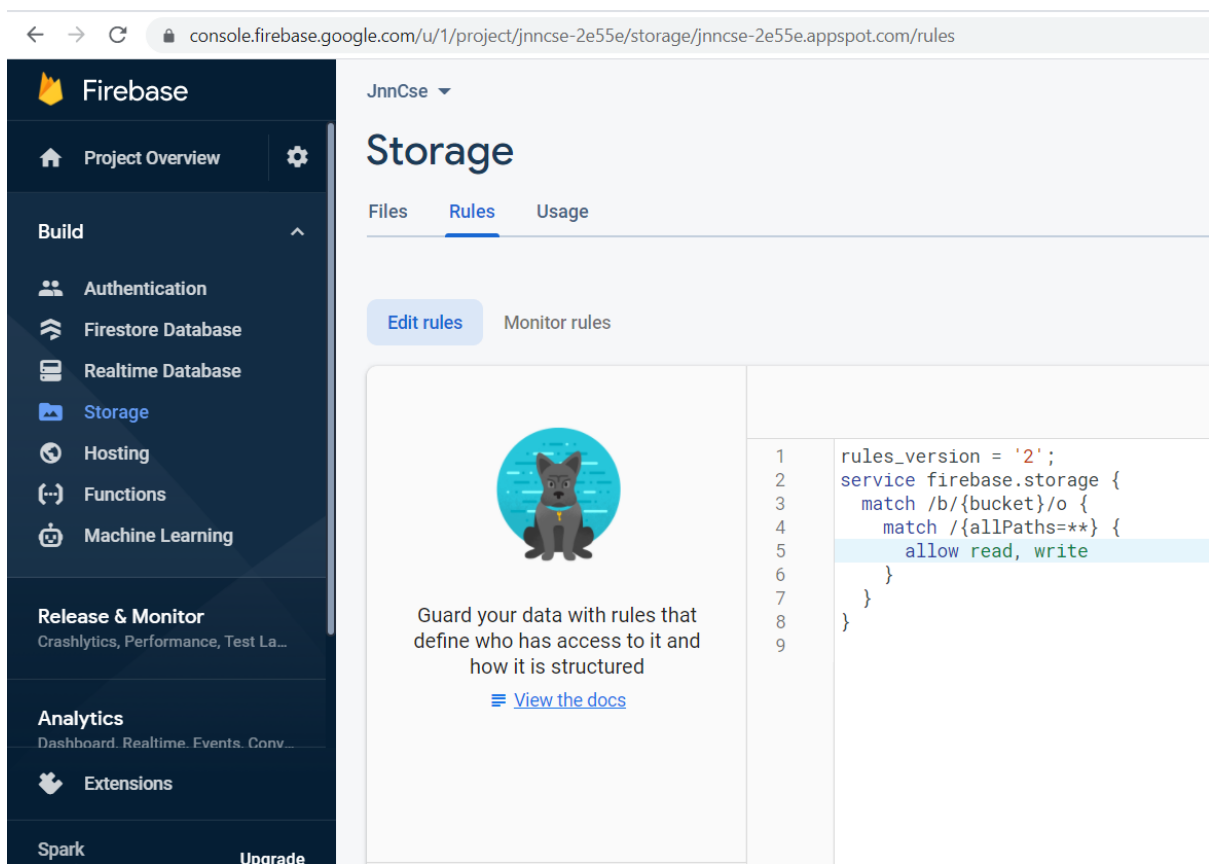
gs://jnncse-2e55e.appspot.com [Upload file](#)

<input type="checkbox"/>	Name	Size	Type	Last modified
There are no files here yet				





Edit rules section to make it accessible:



## Upload Activity details:

### activity\_upload.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".UploadActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"

        tools:context=".GalleryActivity"
        android:padding="30dp"
        android:orientation="vertical"
        >

        <ImageView
            android:id="@+id/uI"
            android:layout_width="wrap_content"
            android:layout_height="391dp"
            android:src="@drawable/prod" />

        <Button
            android:id="@+id/btn_choose_image"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:onClick="launchGallery"
            android:text="Choose image" />

        <Button
            android:id="@+id/btn_upload_image"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:onClick="uploadImage"
            android:text="Upload image" />

    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

### UploadActivity.kt:

```

class UploadActivity : AppCompatActivity() {
    private val PICK_IMAGE_REQUEST = 71
    private var filePath: Uri? = null
    private var firebaseStore: FirebaseStorage? = null
    private var storageReference: StorageReference? = null
    private var mFirebaseDatabaseInstances: FirebaseFirestore?=null
    var pid=""
    var pname=""
    var cost=0.0
    var url=""
    var typ=""
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_upload)
        firebaseStore = FirebaseStorage.getInstance()
        storageReference = FirebaseStorage.getInstance().reference
        var i=intent
        pid=i.getStringExtra("pid").toString()
        pname=i.getStringExtra("pname").toString()
        cost=i.getStringExtra("cost")!!.toDouble()
        typ=i.getStringExtra("typ").toString()

    }
    fun launchGallery(v:View?) {
        val intent = Intent()
        intent.type = "image/*"
        intent.action = Intent.ACTION_GET_CONTENT
        startActivityResultForResult(Intent.createChooser(intent, "Select
Picture"), PICK_IMAGE_REQUEST)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int,
data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == PICK_IMAGE_REQUEST && resultCode ==
Activity.RESULT_OK) {
            if(data == null || data.data == null){
                return
            }

            filePath = data.data
            try {
                val bitmap =
MediaStore.Images.Media.getBitmap(contentResolver, filePath)
                var ui=findViewById<View>(R.id.ui) as ImageView
                ui.setImageBitmap(bitmap)
            }
        }
    }
}

```

```

        } catch (e: IOException) {
            e.printStackTrace()
        }
    }
}

fun addUploadRecordToDb(uri: String){
    //val db = FirebaseFirestore.getInstance()
    // Toast.makeText(this, "inside..", Toast.LENGTH_LONG).show()
    mFirebaseDatabaseInstances= FirebaseFirestore.getInstance()
    val data = HashMap<String, Any>()

    url = uri

    //Toast.makeText(this, "{$pid,$pname,$cost,$url}", Toast.LENGTH_LONG).
    show()
    var p=Product(pid,pname,cost,url,typ)

    mFirebaseDatabaseInstances?.collection("products")?.document(pid!!)?
    .set(p)
    finish()
}

fun uploadImage(v:View?){
    if(filePath != null){
        val ref = storageReference?.child("uploads/" +
        UUID.randomUUID().toString())
        val uploadTask = ref?.putFile(filePath!!)

        val urlTask =
        uploadTask?.continueWithTask(Continuation<UploadTask.TaskSnapshot,
        Task<Uri>> { task ->
            if (!task.isSuccessful) {
                task.exception?.let {
                    throw it
                }
            }
            ref.downloadUrl
        })?.addOnCompleteListener { task->
            if (task.isSuccessful) {
                val downloadUri = task.result
                //
                Toast.makeText(this,downloadUri.toString(),Toast.LENGTH_LONG).show()
                addUploadRecordToDb(downloadUri.toString())
            } else {
                // Handle failures
            }
        }
    }
}

```

```

        }
    }?.addOnFailureListener{
    }
}else{
    Toast.makeText(this, "Please Upload an Image",
Toast.LENGTH_SHORT).show()
}
}
}

```

## Create Login Activity

activity\_login.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".LoginActivity">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="40sp"
            android:gravity="center"
            android:text="ADMIN LOGIN"
            android:textSize="24sp"
            android:textStyle="bold" />

        <RelativeLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <TextView
                android:id="@+id/textView2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="30sp"
                android:layout_marginLeft="30sp"

```

```
        android:layout_marginTop="10sp"
        android:layout_weight="1"
        android:text="UserName:" />
```

```
<EditText
    android:id="@+id/userName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/textView2"
    android:layout_weight="1"
    android:ems="10"
    android:hint="Username"
    android:inputType="textPersonName" />
```

```
<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/textView2"
    android:layout_marginStart="30sp"
    android:layout_marginLeft="30sp"
    android:layout_marginTop="30sp"
    android:layout_weight="1"
    android:text="Password" />
```

```
<EditText
    android:id="@+id/passWord"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/userName"
    android:layout_toRightOf="@id/textView3"
    android:ems="10"
    android:inputType="textPassword" />
```

```
</RelativeLayout>
```

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_marginTop="40sp"
    android:onClick="login"
    android:text="Login" />
```

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```

        android:layout_gravity="center"
        android:gravity="center"
        android:onClick="guest"
        android:text="Guest " />
</LinearLayout>

```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

### LoginActivity.kt:

```

class LoginActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
    }
    fun login(v: View?)
    {
        var u=findViewById<View>(R.id.userName) as EditText
        var p=findViewById<View>(R.id.password) as EditText
        if(u.text.toString()=="admin" && p.text.toString()=="jnncce")
        {
            var i=
Intent(applicationContext,MainActivity::class.java)
            startActivity(i)
        }
        else
        {
            Toast.makeText(this,"Invalid Admin
Credentials",Toast.LENGTH_LONG).show()
        }
    }
    fun guest(v:View?)
    {
        var i= Intent(applicationContext,GuestActivity::class.java)
        startActivity(i)
    }
}

```

### GuestActivity:

#### activity\_guest.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"

```

```

xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".GuestActivity">
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <androidx.recyclerview.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/rv"/>

</LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>

```

GuestActivity.kt:

```

class GuestActivity : AppCompatActivity() {
    private var mAuth: FirebaseAuth?=null
    private var mFirebaseDatabaseInstances: FirebaseFirestore?=null
    lateinit var cv: RecyclerView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_guest)
        val policy =
StrictMode.ThreadPolicy.Builder().permitAll().build()
        StrictMode.setThreadPolicy(policy)
        mAuth=FirebaseAuth.getInstance()
        mFirebaseDatabaseInstances= FirebaseFirestore.getInstance()
        cv=findViewById<View>(R.id.rv) as RecyclerView
        loadData()

    }
    fun loadData()
    {
        val query: Query = FirebaseFirestore.getInstance()
            .collection("products")
        val options: FirestoreRecyclerOptions<Product?> =
FirestoreRecyclerOptions.Builder<Product>()
            .setQuery(query, Product::class.java)
            .build()
        val adapter: FirestoreRecyclerAdapter<*, *> = object :
FirestoreRecyclerAdapter<Product?,
RecyclerView.ViewHolder?>(options) {

```



```

        override fun onCreateViewHolder(group: ViewGroup, i:
Int): ProductHolder {
            // Using a custom layout called R.layout.message for
each item, we create a new instance of the viewholder
            val view: View = LayoutInflater.from(group.context)
                .inflate(R.layout.list_guest, group, false)
            return ProductHolder(view)
        }

```

```

        override fun onBindViewHolder(holder:
RecyclerView.ViewHolder, position: Int, model: Product) {

            (holder.itemView.findViewById<View>(R.id.tpid) as
TextView).setText("Product ID: "+model.pid)
            (holder.itemView.findViewById<View>(R.id.tpname) as
TextView).setText("Product Name:" + model.pname)
            (holder.itemView.findViewById<View>(R.id.tpcost) as
TextView).setText("Product Cost:" + model.cost.toString())
            (holder.itemView.findViewById<View>(R.id.tptyp) as
TextView).setText("Product Category:" + model.typ)
            if(model.url!="") {
                /*Thread(
                    Runnable {

                        var url = URL(model.url)
                        runOnUiThread {
                            var bm =
BitmapFactory.decodeStream(url.openConnection().getInputStream())

                            (holder.itemView.findViewById<View>(R.id.imageView) as
ImageView).setImageBitmap(bm)
                        }
                    }

                ).start();*/
                var url = URL(model.url)
                var
imageView=(holder.itemView.findViewById<View>(R.id.imageView) as
ImageView)

                Glide.with(this@GuestActivity).load(model.url).placeholder(R.drawable
e.prod).error(R.drawable.prod).override(600,600).into(imageView);
            }

        }

```

```

    }
    //Final step, where "mRecyclerView" is defined in your xml layout as
    //the recyclerview
    //Final step, where "mRecyclerView" is defined in your xml layout as
    //the recyclerview

    cv.setLayoutManager = LinearLayoutManager(this)
    cv.adapter=adapter
    adapter.startListening()
}

}

```

list\_guest.xml inside res/layout folder:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="5dp"
    android:layout_marginBottom="30dp"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="wrap_content"

        android:layout_height="wrap_content"
        android:layout_marginBottom="25dp"
        android:orientation="horizontal">

        <ImageView
            android:id="@+id/imageView"
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:layout_marginEnd="20sp"
            android:layout_marginRight="20sp"
            android:src="@drawable/prod" />

        <LinearLayout
            android:layout_width="250dp"
            android:layout_height="wrap_content"
            android:layout_marginEnd="20sp"
            android:layout_marginRight="20sp"
            android:orientation="vertical">

```

```
<TextView
    android:id="@+id/tpid"
    android:layout_width="wrap_content"

    android:layout_height="wrap_content"
    android:layout_gravity="start"
    android:gravity="left"
    android:text="ProductID"
    android:textColor="#E34B4B"
    android:textSize="14dp" />
```

```
<TextView
    android:id="@+id/tpname"
    android:layout_width="wrap_content"

    android:layout_height="wrap_content"
    android:layout_gravity="start"
    android:layout_marginTop="5sp"
    android:text="Product Name"
    android:textColor="#E36969"
    android:textSize="14dp" />
```

```
<TextView
    android:id="@+id/tpcost"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="start"
    android:layout_marginTop="5sp"
    android:text="Product Cost"
    android:textColor="#C65151"
    android:textSize="14dp" />
```

```
<TextView
    android:id="@+id/tptyp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Product Category"
    android:textColor="#A32626" />
```

```
</LinearLayout>
```

```
</LinearLayout>
```

```
</androidx.cardview.widget.CardView>
```

Change in manifest to start LoginActivity on app start:

Updated manifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.pfb">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/Theme.PFB">
        <activity android:name=".GuestActivity"></activity>
        <activity android:name=".LoginActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".UploadActivity" />
        <activity android:name=".MainActivity">

        </activity>
    </application>

</manifest>
```

Create a splash activity before login

activity\_splash.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SplashActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
```

```

        <ImageView
            android:id="@+id/imageView2"
            android:layout_width="wrap_content"
            android:layout_height="400dp"
            app:srcCompat="@drawable/slimwall" />

        <Button
            android:id="@+id/button3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:onClick="proceed"
            android:text="CLICK TO PROCEED" />
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>

```

SplashActivity.xml:

```

class SplashActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_splash)
    }
    fun proceed(v: View?)
    {
        var i= Intent(applicationContext, LoginActivity::class.java)
        startActivity(i)
    }
}

```

Change manifest.xml to start Splash Activity at beginning:

Updated AndroidManifest.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.pfb">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.PFB">
        <activity android:name=".SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
    </application>
</manifest>

```

```
        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".GuestActivity" />
<activity android:name=".LoginActivity">

    </activity>
    <activity android:name=".UploadActivity" />
    <activity android:name=".MainActivity"></activity>
</application>

</manifest>
```