Unit-1 – Introduction to Firebase and services

1.1 Inroduction to Firebase and services

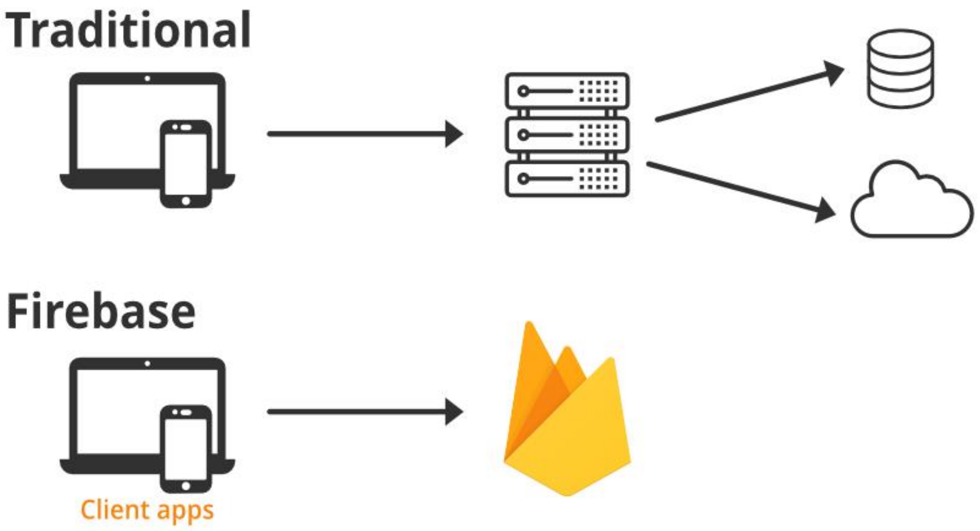
Firebase, a product of Google, is a powerful platform that enables developers to build, manage, and scale applications with ease.

It simplifies app development by offering a secure and efficient backend, eliminating the need for server-side programming.

Firebase supports multiple platforms, including Android, iOS, web, and Unity, making it a versatile solution for developers.

With features like real-time cloud storage, authentication, and NoSQL database support, Firebase ensures seamless data management and synchronization.
Its cloud-based infrastructure allows for faster development, enhanced security, and effortless scalability, empowering developers to focus on creating great user experiences.

Unit-1 – Introduction to Firebase and services



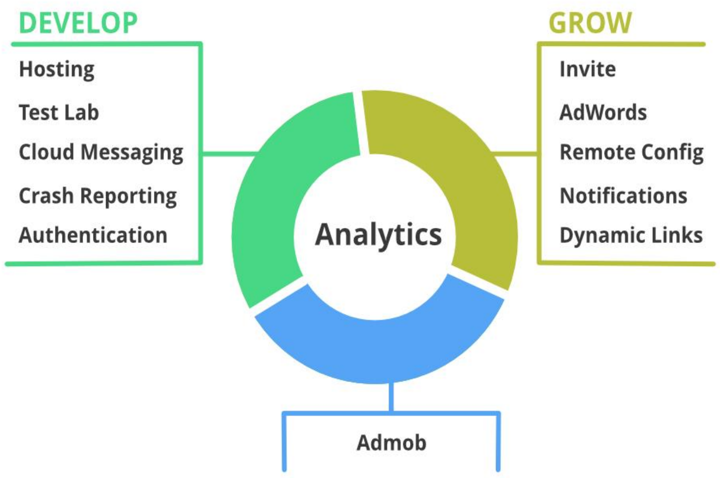
A brief history:
Firebase was originally launched as an online chat service provider under the name Envolve, offering real-time communication through an API for websites.

However, developers began utilizing it beyond chat functionality, leveraging its capabilities to synchronize application data, such as game states, in real-time across users.

Recognizing this broader potential, James Tamplin and Andrew Lee, the founders of Envolve, separated the underlying architecture from its chat system.

They refined and expanded its functionality, leading to the creation of Firebase in 2012—a platform designed to provide real-time database and backend services for modern app development.

Features:
Mainly there are 3 categories in which firebase provides its services.

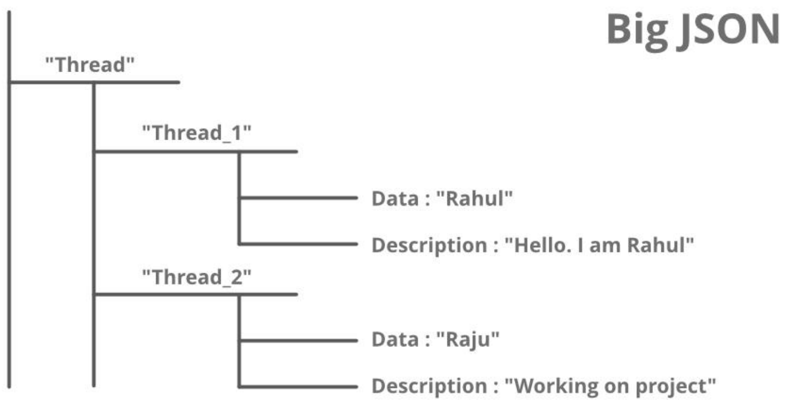


Unit-1 – Introduction to Firebase and services

Build better applications
This feature mainly includes backend services that help developers to build and manage their applications in a better way.

Services included under this feature are :

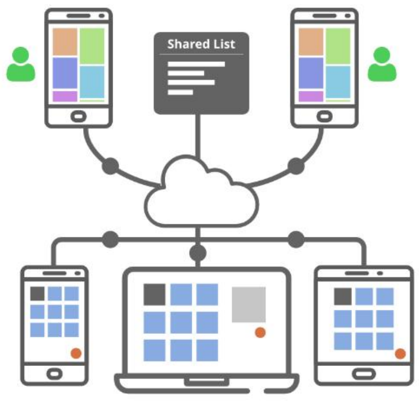
- **Realtime Database:** The Firebase Realtime Database is a cloud-based NoSQL database that manages your data at the blazing speed of milliseconds. In simplest term, it can be considered as a big JSON file.


Big JSON
"Thread"
"Thread_1"
Data : "Rahul"
Description : "Hello. I am Rahul"
"Thread_2"
Data : "Raju"
Description : "Working on project"

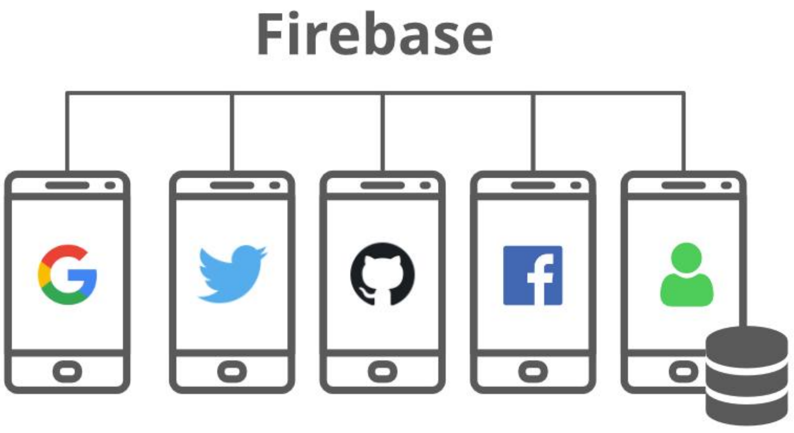
- **Cloud Firestore:**
  The cloud Firestore is a NoSQL document database that provides services like store, sync, and query through the application on a global scale.

  It stores data in the form of objects also known as Documents.

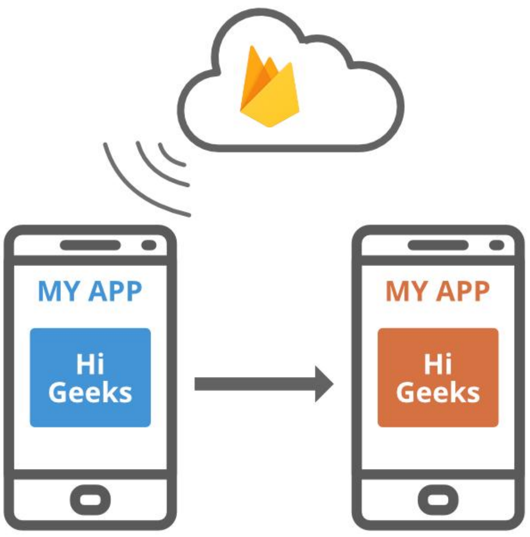
  It has a key-value pair and can store all kinds of data like, strings, binary data, and even JSON trees.



- **Authentication:**
  Firebase Authentication service provides easy to use UI libraries and SDKs to authenticate users to your app.
  It reduces the manpower and effort required to develop and maintain the user authentication service.
  It even handles tasks like merging accounts, which if done manually can be hectic.



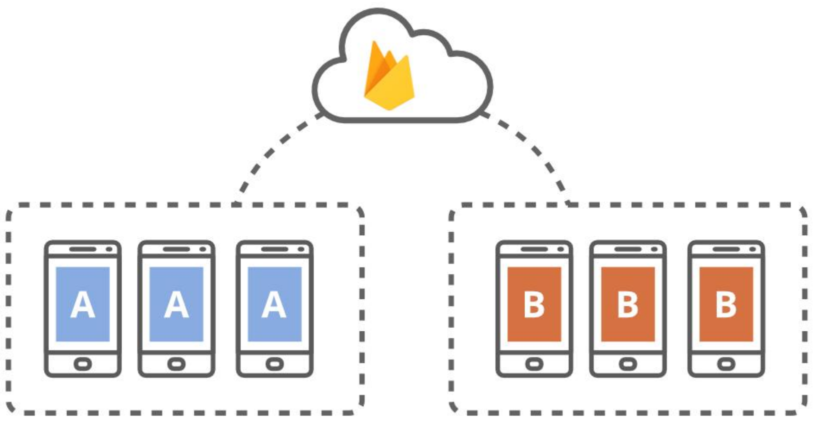
- **Remote Config:**
  The remote configuration service helps in publishing updates to the user immediately. The changes can range from changing components of the UI to changing the behavior of the applications.
  These are often used while publishing seasonal offers and contents to the application that has a limited life.



- **Hosting:**
  Firebase provides hosting of applications with speed and security.
  It can be used to host Static or Dynamic websites and micro services. It has the capability of hosting an application with a single command.

- **Firebase Cloud Messaging(FCM):**
  The FCM service provides a connection between the server and the application end users, which can be used to receive and send messages and notifications.
  These connections are reliable and battery-efficient.

Unit-1 – Introduction to Firebase and services



Improve app quality:
Here majorly all the application performance and testing features are provided.
All the features required to check and manage before launching your application officially are provided in this section.
Services included are:

- **Crashlytics:**
  It is used to get real-time crash reports.
  These reports can further be used to improve the quality of the application.
  The most interesting part of this service is that it gives a detailed description of the crash which is easier to analyze for the developers.
- **Performance monitoring:** This service gives an insight to the performance characteristics of the applications. The performance monitoring SDK can be used to receive performance data from the application, review them, and make changes to the application accordingly through the Firebase console.
- **Test lab:** This service helps to test your applications on real and virtual devices provided by Google which are hosted on the Google Datacenters. It is a cloud-based app-testing infrastructure which supports testing the application on a wide variety of devices and device configurations
- **App Distribution:**This service is used to pre-release applications that can be tested by trusted testers. It comes in handy as decreases the time required to receive feedback from the testers.

Grow your app:
This feature provides your application analytics and features that can help you to interact with your user and make predictions that help you to grow your app. Services provided are:

- **Google analytics:** It is a Free app measurement service provided by Google that provides insight on app usage and user engagement. It serves unlimited reporting for up to 500 distinct automatic or user-defined events using the Firebase SDK.
- **Predictions:** Firebase Predictions uses machine learning to the application's analytics data, further creating dynamic user segments that are based on your user's behavior. These are automatically available to use for the application through Firebase Remote Config, the Notifications composer, Firebase In-App Messaging, and A/B Testing.

- **Dynamic Links:** Deeps Links are links that directly redirects user to specific content. Firebase provides a Dynamic linking service that converts deep links into dynamic links which can directly take the user to a specified content inside the application. Dynamic links are used for converting web users to Native app users. It also increases the conversion of user-to-user sharing. In addition, it can also be used to integrate social media networks, emails, and SMS to increase user engagement inside the application.
- **A/B Testing:** It is used to optimize the application's experience by making it run smoothly, scaling the product, and performing marketing experiments.

Pros and Cons of Using Firebase:

Below listed are the advantages and disadvantages of using a Firebase backend:

Pros:

- Firebase provides a simple and developer-friendly interface, making it **easy to integrate into apps** without backend expertise.
- Firebase offers **real-time data synchronization**, making it ideal for chat apps, multiplayer games, and collaborative tools.
- Works seamlessly on Android, iOS, web, and Unity, enabling **multi-platform app development.**
- Provides **secure authentication** via Google, Facebook, Twitter, email/password, and phone authentication without extra coding.
- Google Cloud infrastructure ensures Firebase **scales** automatically based on user demand.
- **Serverless backend execution** allows running custom logic without managing servers.
- The Spark Plan **(free tier)** offers generous limits for small projects.
- Firebase provides **secure cloud storage** and **fast web hosting** with built-in SSL.
- Offers **real-time crash reporting** and detailed **user analytics** for better app performance and insights.

Cons:

- Firebase uses Firestore (NoSQL), which **lacks complex query support like SQL** databases, making it challenging for relational data handling.
- The free tier is great for small apps, but **costs can escalate** quickly as data storage, reads/writes, and users increase.
- Firebase is a Google product, so **migrating to another backend can be complex** if you need to switch later.
- While Cloud Functions offer flexibility, Firebase **doesn't provide full control over server-side logic** compared to traditional backends.
- Firebase security rules require careful management, and **improper setup can lead to data leaks** or unauthorized access.
- Although Firestore supports **offline mode**, its functionality **may not always be as reliable** as traditional databases.
- **Queries in Firestore are limited** (e.g., no full-text search or advanced filtering like SQL).
- Since Firebase is deeply **integrated with Google Cloud**, any service downtime can impact your app.

Companies using Firebase

Below are some reputable organizations that rely on a firebase backend for its functioning: The New York Times, Alibaba.com, Gameloft, Duolingo, Trivago, Venmo, Lyft.

Unit-1 – Introduction to Firebase and services

Pricing:
There are 2 plans available. Spark plan is initially free but as your user base grows you might need to upgrade to blaze plan. Firebase asks you to pay as you go. For most developers who are just getting started and are on a learning path, they are covered by spark plan.

1.2 Setting Up Firebase Project

Step 1: Create a Firebase Account

1. Go to **Firebase Console**.
2. Sign in using a Google account.

Step 2: Create a New Project

1. Click **"Add project"**.
2. Enter a **project name** (e.g., *MyAppProject*).
3. Choose whether to enable **Google Analytics** (optional but recommended).
4. Click **Create project** and wait for setup to finish.

Step 3: Register Your App

Depending on what you are building, select:

- **Web (</>)**
- **Android**
- **iOS**

For a **Web app**:

1. Click the **Web icon (</>)**.
2. Give your app a nickname.
3. Click **Register app**.
4. Copy the **Firebase configuration code** (API key, project ID, etc.).

Step 4: Add Firebase SDK

- Paste the configuration code into your project's main file (for web: usually index.html or main.js).
- Install Firebase using:
    - CDN (simple projects), or
    - npm (npm install firebase) for larger apps.

Step 5: Enable Firebase Services

From the Firebase Console, you can now enable:

- **Authentication** (login/signup)
- **Firestore Database** or **Realtime Database**
- **Storage** (for images/files)
- **Hosting** (for deploying your website)

Example:

- Go to **Build → Authentication → Get Started**
- Enable **Email/Password** or other sign-in methods.

Step 6: Test the Connection

- Run your app.
- Check the **Firebase Console → Project Overview** to confirm activity (users, database reads, etc.).

1.3 Siging into Firebase Console

Step 1: Create a Firebase Account

1. Go to **Firebase Console**.
2. Sign in using a Google account.

Step 2: Create a New Project

1. Click **"Add project"**.
2. Enter a **project name** (e.g., *MyAppProject*).
3. Choose whether to enable **Google Analytics** (optional but recommended).
4. Click **Create project** and wait for setup to finish.

Step 3: Register Your App

Depending on what you are building, select:

- **Web (</>)**
- **Android**
- **iOS**

For a **Web app**:

1. Click the **Web icon (</>)**.
2. Give your app a nickname.
3. Click **Register app**.
4. Copy the **Firebase configuration code** (API key, project ID, etc.).

Step 4: Add Firebase SDK

- Paste the configuration code into your project's main file (for web: usually index.html or main.js).
- Install Firebase using:
  - CDN (simple projects), or
  - npm (npm install firebase) for larger apps.

Step 5: Enable Firebase Services

From the Firebase Console, you can now enable:

- **Authentication** (login/signup)
- **Firestore Database** or **Realtime Database**
- **Storage** (for images/files)
- **Hosting** (for deploying your website)

Example:

- Go to **Build → Authentication → Get Started**
- Enable **Email/Password** or other sign-in methods.

Step 6: Test the Connection

- Run your app.
- Check the **Firebase Console → Project Overview** to confirm activity (users, database reads, etc.).

1.3 Signing into Firebase Console Using Android Studio (Step-by-Step)

Step 1: Open Android Studio

1. Launch **Android Studio** on your computer.
2. Open an **existing Android project** or create a **new project**:
   - Click **New Project**
   - Select a template (e.g., *Empty Activity*)
   - Click **Next**, enter app details, then **Finish**

Step 2: Open Firebase Assistant

1. In Android Studio, go to the **top menu bar**.

2. Click **Tools → Firebase**.
3. The **Firebase Assistant panel** will open on the right side of Android Studio.

Step 3: Choose a Firebase Service

1. In the Firebase Assistant panel, select a service such as:
    - **Authentication**
    - **Cloud Firestore**
    - **Realtime Database**
    - **Analytics**
2. Click on the required service (for example: **Authentication**).
3. Select an option like **"Authenticate using Email and Password"**.

Step 4: Connect Android Studio to Firebase

1. Click **"Connect to Firebase"**.
2. A browser window will open automatically.
3. You will be redirected to the **Firebase Console sign-in page**.

Step 5: Sign In to Firebase Console

1. Sign in using your **Google account**.
2. Grant necessary permissions to allow Android Studio to access Firebase.
3. Select:
    - An **existing Firebase project**, or
    - Click **Create New Firebase Project**

Step 6: Add Firebase to Android App

1. After successful sign-in, return to Android Studio.
2. Click **"Add Firebase to your Android app"**.
3. Android Studio will:
    - Register your app with Firebase
    - Download the google-services.json file
    - Add required dependencies automatically

Step 7: Sync Project

1. Android Studio will prompt you to **Sync Now**.
2. Click **Sync Now** to apply Firebase configuration.
3. Wait until the Gradle sync finishes successfully.

Step 8: Verify Firebase Connection

1. Go to **Firebase Console → Project Overview**.
2. Check if your **Android app appears** under registered apps.
3. Run the app in Android Studio to confirm proper integration.

Unit-1 – Introduction to Firebase and services

Step 1: Create or Open an Android Project

1. Open **Android Studio**.
2. Click **New Project** or open an existing project.
3. Choose **Empty Activity**.
4. Enter:
   - **App Name**
   - **Package Name** (e.g., com.example.myapp)
5. Select **Language** (Java or Kotlin).
6. Click **Finish**.

Step 2: Open Firebase Assistant

1. In Android Studio, go to the **top menu**.
2. Click **Tools → Firebase**.
3. The **Firebase Assistant** panel will appear on the right side.

Step 3: Choose a Firebase Service

1. In the Firebase Assistant, select a service such as:
   - Authentication
   - Cloud Firestore
   - Realtime Database
   - Analytics
2. Click the required service.
3. Select a feature (example: **Authentication using Email and Password**).

Step 4: Connect to Firebase

1. Click **Connect to Firebase**.
2. A browser window opens automatically.
3. Sign in with your **Google account**.
4. Select:
   - An existing Firebase project, or
   - **Create a new Firebase project**
5. Allow permissions to connect Android Studio with Firebase.

Step 5: Add Firebase to Android App

1. After connecting, click **Add Firebase to your Android App**.
2. Android Studio will automatically:

- Register the app in Firebase
- Download the **google-services.json** file
- Add Firebase dependencies to Gradle files

Step 6: Sync Gradle Files

1. Click **Sync Now** when prompted.
2. Wait until **Gradle Sync completes successfully**.
3. Resolve any errors if they appear.

Step 7: Enable Firebase Services in Console

1. Open **Firebase Console**.
2. Select your project.
3. Enable required services:
   - **Authentication → Get Started → Enable Email/Password**
   - **Firestore Database → Create Database**
   - **Realtime Database → Create Database**

Step 8: Test Firebase Integration

1. Run the app in Android Studio.
2. Check **Logcat** for Firebase initialization messages.
3. Visit **Firebase Console → Project Overview** to confirm app activity.

1.4 Integrating Firebase with Android Studio

In Firebase Console, add an Android app with your package name (e.g., com.example.myapp). Download google-services.json to app/. In project-level build.gradle, add `classpath 'com.google.gms:google-services:4.4.2'`. In app-level build.gradle, add `apply plugin: 'com.google.gms.google-services'` and dependencies like `implementation 'com.google.firebase:firebase-auth:23.0.0'` then sync.

1.5 Overview of Firebase Authentication

Firebase Authentication supports email/password, Google, Facebook, phone, and anonymous sign-ins with secure token management. Enable providers in Console > Authentication > Sign-in method.

1.5.1 FirebaseAuth Instance

Get the singleton via `FirebaseAuth mAuth = FirebaseAuth.getInstance();`. Use for sign-in/out and state listeners: `mAuth.addAuthStateListener(this::onAuthStateChanged)`.

Example:

```java
FirebaseAuth mAuth = FirebaseAuth.getInstance();
mAuth.signOut(); // Logs out user
```

1.5.2 AuthUI Instance

FirebaseUI simplifies UI with pre-built screens. Add dependency `implementation 'com.firebaseui:firebase-ui-auth:8.0.2'`.
Launch: `startActivity(AuthUI.getInstance().createSignInIntentBuilder().setAvailableProviders(providers).build());`.

Example:

```java
List<AuthUI.IdpConfig> providers = Arrays.asList(new
AuthUI.IdpConfig.EmailBuilder().build());
Intent signInIntent =
AuthUI.getInstance().createSignInIntentBuilder().setAvailableProviders(providers).build();
startActivityForResult(signInIntent, RC_SIGN_IN);
```

1.5.3 FirebaseUser Class

Represents the current user: `FirebaseUser user = mAuth.getCurrentUser();`. Access uid, email, displayName, photoUrl, and check `user.isEmailVerified()`.

Example:

```java
if (user != null) {
    String email = user.getEmail();
    String uid = user.getUid();
}
```

1.5.4 Authentication Provider Class

Handles provider data via IdpResponse from sign-in results.
Use IdpResponse.fromResult(intent) to get provider info like email from Google sign-in.

Example:

```java
IdpResponse response = IdpResponse.fromResult(intent);
if (response != null) {
   String email = response.getEmail();
}
```

1.6 Introduction to FirebaseUI Auth

FirebaseUI Auth provides customizable sign-in screens for multiple providers, handling flows like password reset and account linking automatically. Configure providers and themes for branded UI.

Example Setup:
Add to build.gradle: implementation 'com.firebaseui:firebase-ui-auth:8.0.2'.
Use createSignInIntentBuilder() for email/password or Google.

1.7 Firebase SDK Authentication

Core SDK offers direct APIs for custom auth flows. Example for email sign-up: mAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(task -> { if (task.isSuccessful()) { /* Navigate to home */ } });. Check current user in onStart(): if (mAuth.getCurrentUser() == null) startActivity(new Intent(this, LoginActivity.class));.

A. Multiple Choice Questions (MCQs)

Firebase is developed by:
a) Microsoft
b) Google
c) Amazon
d) Apple

Answer: b

Firebase provides which type of service?
a) SaaS
b) PaaS

c) BaaS
d) IaaS

Answer: c

Which of the following is a Firebase service?
a) SQLite
b) Firebase Authentication
c) MySQL
d) Apache

Answer: b

Firebase Console is used to:
a) Write code
b) Manage Firebase projects
c) Design UI
d) Run emulator

Answer: b

Which file is required to connect Firebase with Android Studio?
a) android.json
b) firebase.xml
c) google-services.json
d) config.json

Answer: c

Which plugin is added in Gradle for Firebase integration?
a) com.android.application
b) com.google.firebase
c) com.google.gms.google-services
d) firebase.sdk

Answer: c

Firebase Authentication is used for:
a) Database management
b) User authentication
c) File storage
d) App analytics

Answer: b

Which class provides access to authentication methods?
a) FirebaseUser

b) AuthUI
c) FirebaseAuth
d) AuthProvider

Answer: c

Which class represents the logged-in user?
a) AuthUser
b) FirebaseUser
c) UserAuth
d) LoginUser

Answer: b

AuthUI is part of which library?
a) Firebase Core
b) Firebase SDK
c) FirebaseUI
d) Firebase Auth

Answer: c

Which authentication method is supported by Firebase?
a) Email/Password
b) Google Sign-In
c) Facebook Sign-In
d) All of the above

Answer: d

Which provider class is used for Google authentication?
a) EmailAuthProvider
b) PhoneAuthProvider
c) GoogleAuthProvider
d) FacebookAuth

Answer: c

One-Mark Short Answer Questions

1.

What is Firebase?

Expand BaaS.

Name any one Firebase service.

Unit-1 – Introduction to Firebase and services

What is Firebase Console?

What is the use of google-services.json file?

What is Firebase Authentication?

What is FirebaseAuth instance?

What is AuthUI?

What is FirebaseUser class?

Name any one authentication provider.

What is Firebase SDK?

Which platform manages Firebase projects?

What is GoogleAuthProvider?

What is FirebaseUI Auth?

Why is Firebase used in Android apps?

Long Answer Questions (4 Marks Each)

1. Explain Firebase and its services.

2. Explain the steps to set up a Firebase project.

3. Explain signing into Firebase Console.

4. Explain how to integrate Firebase with Android Studio.

5. Explain Firebase Authentication.

6. Explain FirebaseAuth instance.

7. Explain AuthUI instance.

8. Explain FirebaseUser class.

9. Explain Authentication Provider classes.

Unit-1 – Introduction to Firebase and services

Unit-2: FirebaseUI Auth authentication

2.1 FirebaseUI Authentication

FirebaseUI is a pre-built UI library that integrates various authentication providers like Email/Password, Google, Facebook, etc., into your app.

It simplifies the process of authentication by handling most of the UI/UX and backend logic for you.

2.1.1 User Registration (Email and Password Authentication)

Firebase provides a method to register a new user with an email address and password.

Example: User Registration

```
import firebase from 'firebase/app';

import 'firebase/auth';

const registerUser = (email, password) => {

  firebase.auth().createUserWithEmailAndPassword(email, password)

    .then((userCredential) => {

      const user = userCredential.user;

      console.log('User Registered:', user);
    })
    .catch((error) => {
      console.error('Error registering user:', error.message);
    });
};
```

Explanation:

- `createUserWithEmailAndPassword(email, password)` is the method used to register a new user with the provided email and password.
- If successful, it returns a `userCredential` object that contains user info.

2.1.2 Manage Login/Logout (Email and Password Authentication)

Once users are registered, Firebase allows them to log in using their email and password. You can also log users out.

Example: User Login

```
const loginUser = (email, password) => {
 firebase.auth().signInWithEmailAndPassword(email, password)
  .then((userCredential) => {
    const user = userCredential.user;
    console.log('User Logged In:', user);
  })
  .catch((error) => {
    console.error('Login Error:', error.message);
  });
};
```

Example: User Logout

```
const logoutUser = () => {

  firebase.auth().signOut()

    .then(() => {

      console.log('User Logged Out');

    })

    .catch((error) => {

      console.log('Logout Error:', error.message);

    });

};
```

Explanation:

- `signInWithEmailAndPassword(email, password)` logs the user in with the given credentials.
- `signOut()` logs the current user out of the app

2.2.3 Password Reset and Email Verification

Firebase allows you to handle password resets and email verifications. This is useful for managing user accounts and ensuring email legitimacy.

Example: Password Reset

```
const resetPassword = (email) => {

  firebase.auth().sendPasswordResetEmail(email)

    .then(() => {

      console.log('Password Reset Email Sent');

    })

    .catch((error) => {
      console.error('Error sending password reset email:', error.message);
    });
};
```

Example: Send Email Verification

```
const sendEmailVerification = () => {
  const user = firebase.auth().currentUser;
  user.sendEmailVerification()
    .then(() => {
      console.log('Email Verification Sent');
    })
    .catch((error) => {
      console.error('Error sending verification email:', error.message);
    });
};
```

Explanation:

- `sendPasswordResetEmail(email)` sends a password reset link to the provided email.
- `sendEmailVerification()` sends an email to the current user for verification.

2.2 Google Sign-In Authentication

Firebase provides integration with Google Sign-In, making it easy for users to log in with their Google accounts.

Example: Google Sign-In

```
import firebase from 'firebase/app';

import 'firebase/auth';
```

```
const googleProvider = new firebase.auth.GoogleAuthProvider();


const googleSignIn = () => {
  firebase.auth().signInWithPopup(googleProvider)
    .then((result) => {
      const user = result.user;
      console.log('User signed in with Google:', user);
    })
    .catch((error) => {
      console.error('Google Sign-In Error:', error.message);
    });
};
```

Explanation:

- `signInWithPopup(googleProvider)` initiates the Google login process using a popup.
- Upon success, the user information is returned.

2.3 Facebook Sign-In Authentication

Similarly, Firebase supports Facebook Sign-In for authenticating users with their Facebook accounts.

Example: Facebook Sign-In

```
import firebase from 'firebase/app';

import 'firebase/auth';
```

```
const facebookProvider = new firebase.auth.FacebookAuthProvider();


const facebookSignIn = () => {
 firebase.auth().signInWithPopup(facebookProvider)
  .then((result) => {
   const user = result.user;
   console.log('User signed in with Facebook:', user);
  })
  .catch((error) => {
   console.error('Facebook Sign-In Error:', error.message);
  });
};
```

Explanation:

- `signInWithPopup(facebookProvider)` triggers Facebook login via a popup.
- It then returns user data if successful.

2.4 Handling Firebase Authentication Errors

Firebase provides error codes and messages that you can handle appropriately to improve the user experience.

Example: Handling Authentication Errors

```
const handleError = (error) => {
  switch (error.code) {
    case 'auth/user-not-found':
      console.error('No user found with that email.');
      break;
    case 'auth/wrong-password':
      console.error('Incorrect password.');
      break;
    case 'auth/invalid-email':
      console.error('Invalid email address.');
      break;
    default:
      console.error('An unknown error occurred:', 
error.message);
  }
};
```

Explanation:

- This function checks the error code returned by Firebase Authentication and handles specific error cases such as incorrect password or user not found.

2.5 Firebase Realtime Database vs. Firestore

Firebase provides two databases: **Realtime Database** and **Cloud Firestore**. Both serve the purpose of storing and syncing data, but they have different structures, use cases, and features.

Firebase Realtime Database

- **Structure**: Data is stored as a JSON tree.
- **Real-Time Sync**: Changes are reflected instantly across all connected clients.
- **Offline Support**: Fully supported on mobile clients.
- **Querying**: Limited querying capabilities (only shallow queries).
- **Scalability**: Less scalable for large applications.

Example: Using Firebase Realtime Database

```
import firebase from 'firebase/app';
import 'firebase/database';

const database = firebase.database();

// Write data
database.ref('users/1').set({
  username: 'john_doe',
  email: 'john@example.com'
});

// Read data
database.ref('users/1').once('value')
  .then((snapshot) => {
    console.log(snapshot.val());
  });
```

Cloud Firestore

- **Structure**: Uses collections and documents, similar to a NoSQL database.
- **Real-Time Sync**: Also supports real-time syncing, but more efficient and flexible.
- **Offline Support**: Fully supported on both mobile and web clients.
- **Querying**: More powerful querying, including compound queries, pagination, and sorting.
- **Scalability**: Highly scalable and suitable for large apps.

Example: Using Firestore

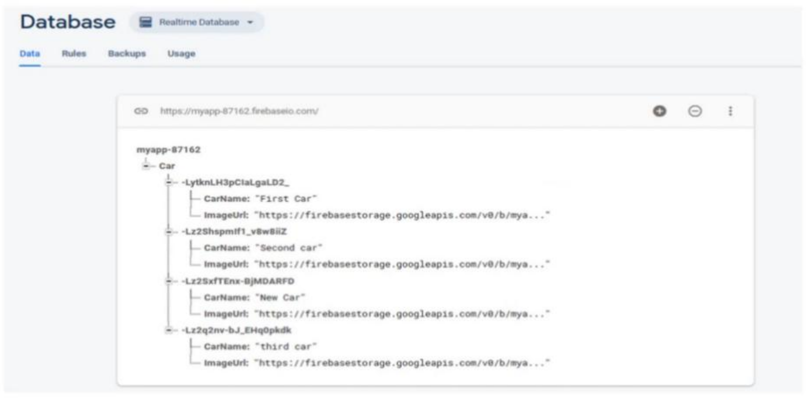
```
import firebase from 'firebase/app';
import 'firebase/firestore';

const firestore = firebase.firestore();

// Write data
firestore.collection('users').doc('1').set({
  username: 'john_doe',
  email: 'john@example.com'
});

// Read data
firestore.collection('users').doc('1').get()
  .then((doc) => {
    if (doc.exists) {
      console.log(doc.data());
    } else {
      console.log('No such document!');
    }
  });
```

Difference between Realtime Database vs. Firestore

Feature	Realtime Database	Firestore
Structure	JSON Tree	Collections and Documents

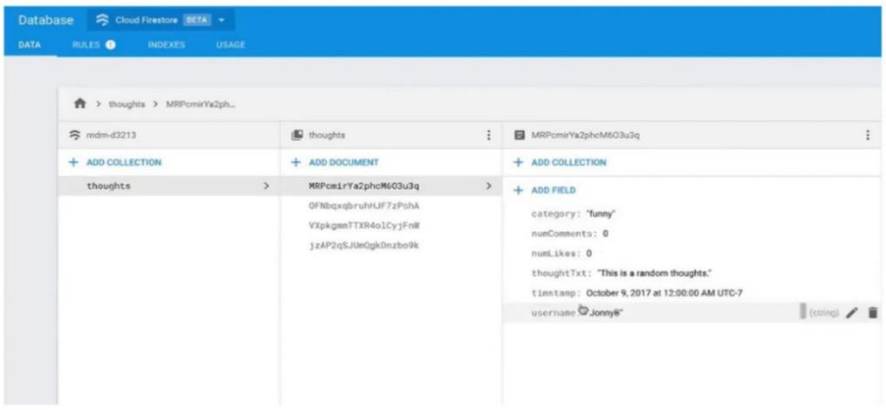
Feature	Realtime Database	Firestore
Real-Time Sync	Yes	Yes
Querying	Limited (shallow queries)	Complex queries supported
Scalability	Less scalable	Highly scalable
Offline Support	Yes (mobile)	Yes (mobile & web)
Pricing	Based on data downloaded/stored	Based on reads/writes/storage

Realtime Database:



```
myapp-87162
 └─ Car
     ├─ -LytknLH3pClaLgaLD2_
     │   ├─ CarName: "First Car"
     │   └─ ImageUrl: "https://firebasestorage.googleapis.com/v0/b/mya..."
     ├─ -Lz2Shspmlf1_v8w8iiZ
     │   ├─ CarName: "Second car"
     │   └─ ImageUrl: "https://firebasestorage.googleapis.com/v0/b/mya..."
     ├─ -Lz2SxfTEnx-BjMDARFD
     │   ├─ CarName: "New Car"
     │   └─ ImageUrl: "https://firebasestorage.googleapis.com/v0/b/mya..."
     └─ -Lz2q2nv-bJ_EHq0pkdk
         ├─ CarName: "third car"
         └─ ImageUrl: "https://firebasestorage.googleapis.com/v0/b/mya..."
```

Firestore Database:



```
thoughts
MRPcmirYa2phcM6O3u3q
OFNbqxqbruhHJF7zPshA
VXpkgmmTTXR4olCyjFnW
jzAP2qSJUmQgkDnzbo9k
```

```
category: "funny"
numComments: 0
numLikes: 0
thoughtTxt: "This is a random thoughts."
timestamp: October 9, 2017 at 12:00:00 AM UTC-7
username "Jonny6"   (string)
```

A. Multiple Choice Questions (MCQs)

Firebase Authentication is used for:
a) Data storage
b) User authentication
c) Image processing
d) App testing

Answer: b

Which method is used for user registration using email and password?
a) signInWithEmailAndPassword()
b) createUserWithEmailAndPassword()
c) registerUser()
d) addUser()

Answer: b

Which method is used to login a registered user?
a) loginUser()
b) authenticateUser()
c) signInWithEmailAndPassword()
d) checkUser()

Answer: c

Which method is used to logout a user?
a) logout()
b) exitUser()
c) signOut()
d) closeApp()

Answer: c

Which method sends a password reset email?
a) resetPassword()
b) sendPasswordResetEmail()
c) changePassword()
d) verifyPassword()

Answer: b

Which method is used for email verification?
a) verifyUser()
b) sendEmailVerification()
c) confirmEmail()
d) checkEmail()

Answer: b

Google Sign-In authentication is based on:
a) HTTP
b) JSON
c) OAuth 2.0
d) REST

Answer: c

Which social login is supported by Firebase?
a) Twitter
b) Facebook
c) GitHub
d) All of the above

Answer: d

Which class represents the currently logged-in user?
a) AuthUser
b) FirebaseAccount
c) FirebaseUser
d) UserAccount

Answer: c

FirebaseUI Auth is mainly used to:
a) Store data

b) Simplify authentication UI
c) Write backend code
d) Create database

Answer: b

Which method checks whether a user is logged in?
a) isUserActive()
b) getCurrentUser()
c) checkLogin()
d) userStatus()

Answer: b

Firebase Authentication errors are accessed using:
a) Logs
b) Exception
c) Error object
d) Database

Answer: b

Which Firebase database uses documents and collections?
a) SQLite
b) Realtime Database
c) Firestore
d) Shared Preferences

Answer: c

Which database supports complex queries and indexing?
a) Realtime Database
b) Firestore
c) SQLite
d) JSON

Answer: b

Which authentication does NOT require a password?
a) Email/Password
b) Google Sign-In
c) Custom Auth
d) Anonymous Auth

Answer: b

One-Mark Short Answer Questions

What is Firebase Authentication?

Name one Firebase authentication method.

Which method is used for email-password registration?

Which method is used for login?

What is the use of `signOut()` method?

What is password reset?

Which method sends a password reset email?

What is email verification?

Which method sends verification email?

What is Google Sign-In?

What is Facebook authentication?

What is FirebaseUI Auth?

What is OAuth?

Name the class representing authenticated user.

Which database uses JSON tree structure?

Long Answer Questions (4 Marks Each)

1. Explain Email and Password Authentication in Firebase.

2. Explain user registration and login/logout process using Firebase.

3. Explain password reset and email verification in Firebase.

4. Explain Google Sign-In authentication in Firebase.

5. Explain Facebook Sign-In authentication in Firebase.

6. Explain handling Firebase Authentication errors.

7. Differentiate Firebase Realtime Database and Firestore.

UNIT_3_Firebase Data Storage

Unit-3: Firebase Data storage

3.1 Introduction to NoSQL database and Firebase real-time database

Firebase Realtime Database is a powerful **NoSQL** cloud database that enables **real-time** data storage and synchronization across all clients.

It's particularly suited for applications requiring live updates, such as **chat apps** and **collaborative tools.**

By following the setup instructions and using the provided examples we can take the help of Firebase Realtime Database to build efficient and interactive web applications that meet the demands of modern real-time data needs.

What is Firebase Realtime Database?

Firebase Realtime Database is a NoSQL cloud database that allows developers to store and sync data in real time across all clients. This makes it perfect for applications that require live updates, such as chat apps, collaborative tools, and real-time analytics. Firebase ensures low-latency performance by synchronizing data across all connected clients in milliseconds.

Key Features of Firebase Realtime Database:

- **Real-time Synchronization:** Instantly updates data across all connected clients.
- **Offline Capabilities:** Data is available locally even when the device is offline, and updates are synchronized when the device reconnects.
- **Scalability:** Designed to handle large-scale applications.
- **Security:** Provides robust authentication and database security rules to ensure data protection.

How to Set Up Firebase Realtime Database

Step 1: Create a Firebase Project

UNIT_3_Firebase Data Storage

Before using Firebase Realtime Database, you need to set up a Firebase project:

- **Create a Firebase Project**: Go to the Firebase Console, click on "**Add project**," and follow the setup instructions.
- **Add Firebase to Your App:** Firebase provides detailed guides for adding Firebase to various platforms (**Web**, **iOS**, **Android**). Follow the instructions specific to your platform to include Firebase in our project.

Step 2: Set Up Realtime Database

Enable Realtime Database:

- Go to the Firebase Console.
- Select your project.
- Simply click on "**Realtime Database**" in the menu on the left-hand side.
- Click on "**Create Database**."
- Select your database location and security rules (Start in test mode for development purposes).

Step 3: Add Firebase SDK to Your Project

For a web application, include the Firebase **SDK** in your HTML file:

```
<!-- Firebase App (the core Firebase SDK) -->
<script src="https://www.gstatic.com/firebasejs/9.6.4/firebase-app.js"></script>

<!-- Firebase Realtime Database -->
<script src="https://www.gstatic.com/firebasejs/9.6.4/firebase-database.js"></script>
```

Step 4: Initialize Firebase

Initialize Firebase in your JavaScript file using the configuration details from your Firebase project settings:

```
// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyDmaZAcK7xwQTAsQJxaGnG56oB8RIJDMnc",
  authDomain: "samplep-d6b68.firebaseapp.com",
  projectId: "samplep-d6b68",
  storageBucket: "samplep-d6b68.appspot.com",
  messagingSenderId: "398502093281",
  appId: "1:398502093281:web:5d685b0733d4d6d66472a0",
  measurementId: "G-9E6K9YD8D1"
};

// Initialize Firebase
firebase.initializeApp(firebaseConfig);

// Initialize Realtime Database and get a reference to the service
const database = firebase.database();
```

Explanation: The provided code snippet configures and initializes a Firebase app using specific project settings. The firebaseConfig object contains the necessary credentials and identifiers for connecting to your Firebase project, and the **firebase.initializeApp(firebaseConfig)** line initializes the Firebase app. **The firebase.database()** call sets up and references the Firebase Realtime Database service**.**

Working with Firebase Realtime Database

Step 1: Writing Data

To write data to Firebase Realtime Database, use the set method. This example demonstrates how to write user data to the database:

```
function writeUserData(userId, name, email) {
 firebase.database().ref('users/' + userId).set({
  username: name,
  email: email
 });
}
// Example usage
writeUserData('1', 'John Doe', 'john.doe@example.com');
```

Explanation: The **writeUserData** function writes user data to the Firebase Realtime Database. It takes three parameters: **userId, name,** and **email**. It creates or updates a user entry in the users node of the database with the provided **userId**, setting the username and email fields. The example usage demonstrates calling the function to store data for a user with **ID** 1, **name John Doe**, and **email john.doe@example.com**

Step 2: Reading Data

To read data from Firebase Realtime Database, use the once method to read data once or the on method to listen for changes in the data.

- **Reading Data Once:**

```
function readUserData(userId) {
  const userRef = firebase.database().ref('users/' + userId);
  userRef.once('value').then((snapshot) => {
    const data = snapshot.val();
    console.log(data);
  });
}

// Example usage
readUserData('1');
```

Explanation: The **readUserData** function reads the data for the user with the specified userId from the users node in the Firebase Realtime Database. It uses the once method to retrieve the data once and logs the retrieved data to the console. The example usage retrieves and logs data for the user with ID 1.

- **Listening for Data Changes:**

```
function listenForUserData(userId) {
  const userRef = firebase.database().ref('users/' + userId);
  userRef.on('value', (snapshot) => {
    const data = snapshot.val();
    console.log(data);
  });
}

// Example usage
listenForUserData('1');
```

Explanation: The **listenForUserData** function sets up a real-time listener on the users node in the Firebase Realtime Database for the specified userId. It uses the on method to listen for any changes to the data. When the data changes, the new data is retrieved and logged to the console. The example usage sets up a listener for changes to the user data for the user with ID 1.

Step 3: Updating Data

To update specific fields in your data without overwriting the entire node, use the '**update**' method:

```
function updateUserData(userId, email) {
  const updates = {};
  updates['/users/' + userId + '/email'] = email;

  firebase.database().ref().update(updates);
}

// Example usage
updateUserData('1', 'new.email@example.com');
```

Explanation: The **updateUserData** function creates an updates object with the path to the user's email **(/users/{userId}/email)** as the key and the new email address as the value. It then uses the update method of the Firebase Realtime Database to update the specified user's email address. The example usage updates the email address for the user with ID 1 to **new.email@example.com**

Step 4: Deleting Data

To delete data from Firebase Realtime Database, use the remove method:

```
function deleteUser(userId) {
  firebase.database().ref('users/' + userId).remove()
    .then(() => {
      console.log('User removed successfully.');
    })
    .catch((error) => {
      console.error('Error removing user:', error);
    });
}

// Example usage
deleteUser('1');
```

Explanation: The **deleteUser** function removes the user with the specified userId from the users node in the Firebase Realtime Database. It uses the remove method to delete the user and handles the success and error cases using then and catch methods respectively. The example usage deletes the user with **ID 1.**

Firebase Realtime Database Example: Real-Time Chat Application
Let's build a simple real-time chat application to demonstrate Firebase Realtime Database in action.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Real-Time Chat Application</title>
<style>
body {
   font-family: Arial, sans-serif;
   margin: 0;
   padding: 0;
   background-color: #f4f4f4;
}

.chat-container {
   max-width: 600px;
   margin: 20px auto;
   padding: 20px;
   background-color: #fff;
   border-radius: 5px;
   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

#chat-window {
   height: 300px;
   overflow-y: scroll;
   border: 1px solid #ccc;
   padding: 10px;
   margin-bottom: 10px;
}
```

```
#message-input {
   width: calc(100% - 70px);
   padding: 5px 10px;
   margin-right: 10px;
}

#send-button {
   padding: 5px 10px;
   background-color: #007bff;
   color: #fff;
   border: none;
   border-radius: 3px;
   cursor: pointer;
}

#send-button:hover {
   background-color: #0056b3;
}
</style>
</head>
<body>
<div class="chat-container">
   <div id="chat-window"></div>
   <input type="text" id="message-input" placeholder="Type your
message...">
   <button id="send-button">Send</button>
</div>

<script src="https://www.gstatic.com/firebasejs/9.6.4/firebase-
app.js"></script>
<script src="https://www.gstatic.com/firebasejs/9.6.4/firebase-
database.js"></script>
<script>
const firebaseConfig = {
         apiKey: "AIzaSyDmaZAcK7xwQTAsQJxaGnG56oB8RIJDMnc",
         authDomain: "samplep-d6b68.firebaseapp.com",
         projectId: "samplep-d6b68",
         storageBucket: "samplep-d6b68.appspot.com",
         messagingSenderId: "398502093281",
         appId: "1:398502093281:web:5d685b0733d4d6d66472a0",
      measurementId: "G-9E6K9YD8D1"
    };

// Initialize Firebase
firebase.initializeApp(firebaseConfig);

const database = firebase.database();
const chatWindow = document.getElementById('chat-window');
const messageInput = document.getElementById('message-input');
const sendButton = document.getElementById('send-button');

sendButton.addEventListener('click', () => {
  const message = messageInput.value.trim();
  if (message !== '') {
    database.ref('messages').push({
      message: message,
      timestamp: firebase.database.ServerValue.TIMESTAMP
    });
    messageInput.value = '';
  }
});

database.ref('messages').on('child_added', (snapshot) => {
  const messageData = snapshot.val();
  const messageElement = document.createElement('div');
  messageElement.textContent = messageData.message;
  chatWindow.appendChild(messageElement);
  chatWindow.scrollTop = chatWindow.scrollHeight;
});
</script>
</body>
</html>
```

Output:

Final Output

Conclusion

Overall, Firebase Realtime Database offers developers a robust solution for building real-time applications that require synchronized data across clients. By setting up a Firebase project, adding Firebase to your app, and initializing the Realtime Database, you can quickly get started with storing and syncing data in real-time. You can also set up and integrate Firebase Realtime Database into your project, perform essential CRUD operations, and build interactive applications like real-time chat. The provided examples demonstrate how to write, read, update, and delete data, as well as how to build a real-time chat application.

A. MCQs (Multiple Choice Questions)

Firebase is developed by:
a) Microsoft
b) Google
c) Amazon
d) Meta

Answer: b

Firebase Realtime Database is a:
a) SQL database
b) NoSQL database
c) Graph database
d) Object database

Answer: b

Which data format is used by Firebase Realtime Database?
a) XML
b) CSV
c) JSON
d) HTML

Answer: c

Which method is used to write data into Firebase?
a) setValue()
b) getValue()
c) putValue()
d) addValue()

Answer: a

Which listener is used to read data continuously from Firebase?
a) OnClickListener

b) ValueEventListener
c) DataListener
d) FirebaseListener

Answer: b

Which operation removes data from Firebase?
a) delete()
b) clear()
c) removeValue()
d) erase()

Answer: c

Firebase stores data in the form of:
a) Tables and rows
b) Files and folders
c) Nodes and key-value pairs
d) Columns

Answer: c

Which Firebase service provides real-time data synchronization?
a) Firebase Storage
b) Firebase Auth
c) Firebase Realtime Database
d) Firebase Analytics

Answer: c

Firebase Realtime Database stores data as:
a) Tables
b) Rows
c) JSON tree
d) CSV file

Answer: c

Which method is used to update an existing value in Firebase?
a) updateValue()
b) setValue()
c) modifyValue()
d) changeValue()

Answer: b

UNIT_3_Firebase Data Storage

Which Firebase component connects Android app to database?
a) FirebaseAuth
b) FirebaseConsole
c) DatabaseReference
d) FirebaseStorage

Answer: c

Which listener triggers only once when reading data from Firebase?
a) ValueEventListener
b) SingleValueEventListener
c) addListenerForSingleValueEvent
d) OnDataChangeListener

Answer: c

Which method deletes a specific child node in Firebase?
a) clearValue()
b) remove()
c) removeValue()
d) deleteNode()

Answer: c

Firebase Realtime Database supports which type of synchronization?
a) Manual
b) Batch
c) Real-time
d) Offline only

Answer: c

Which method is used to generate a unique key in Firebase?
a) getKey()
b) push()
c) createKey()
d) unique()

Answer: b

Which Firebase feature allows data access without internet?
a) Analytics
b) Offline persistence
c) Cloud Functions
d) Crashlytics

Answer: b

UNIT_3_Firebase Data Storage

Which file is used to connect Android app with Firebase?
a) firebase.json
b) google-services.json
c) config.json
d) android-firebase.json

Answer: b

Firebase database rules are mainly used for:
a) UI design
b) Authentication
c) Data security
d) Data formatting

Answer: c

Which method retrieves data from DataSnapshot?
a) getData()
b) getValue()
c) fetchValue()
d) readValue()

Answer: b

B. One-Mark Short Answer Questions

What is NoSQL database?

What is Firebase?

Name the database provided by Firebase for real-time data storage.

What data format is used in Firebase Realtime Database?

What is a node in Firebase?

Which method is used to insert data into Firebase?

Name the listener used to read data from Firebase.

What does CRUD stand for?

Which method is used to update data in Firebase?

UNIT_3_Firebase Data Storage

Which method is used to delete data in Firebase?

What is real-time data synchronization?

Name any one Firebase service.

Long Answer Questions (4 Marks Each)

1.Explain NoSQL database and Firebase Realtime Database.

2. Explain how to write data into Firebase from an Android app.

3. Explain reading and displaying data from Firebase.

4. Explain CRUD operations on Firebase dataset.

5. Explain advantages of Firebase Realtime Database.

6. Explain Firebase data structure with example.

7. Explain the CRUD operations in Firebase Realtime Database.

Unit_4_Working with Data and API in android

Unit-4: Working with Data and API in Android
4.1 Introduction to API and web services
4.2 Introduction to RESTful architecture
4.3 Parsing JSON data in Android
4.4 Testing API
4.5 Introduction to Volley libraries
4.6 Creating GET and POST request in android
4.7 API response handling

What is API?

API stands for **A**pplication **P**rogramming **I**nterface.

A Web API is an application programming interface for the Web.

A Browser API can extend the functionality of a web browser.

A Server API can extend the functionality of a web server.

Services in Android with Example

Services in Android are a special component that facilitates an application to run in the background in order to perform long-running operation tasks.

The prime aim of a service is to ensure that the application remains active in the background so that the user can operate multiple applications at the same time.

A user-interface is not desirable for android services as it is designed to operate long-running processes without any user intervention.

A service can run continuously in the background even if the application is closed or the user switches to another application.

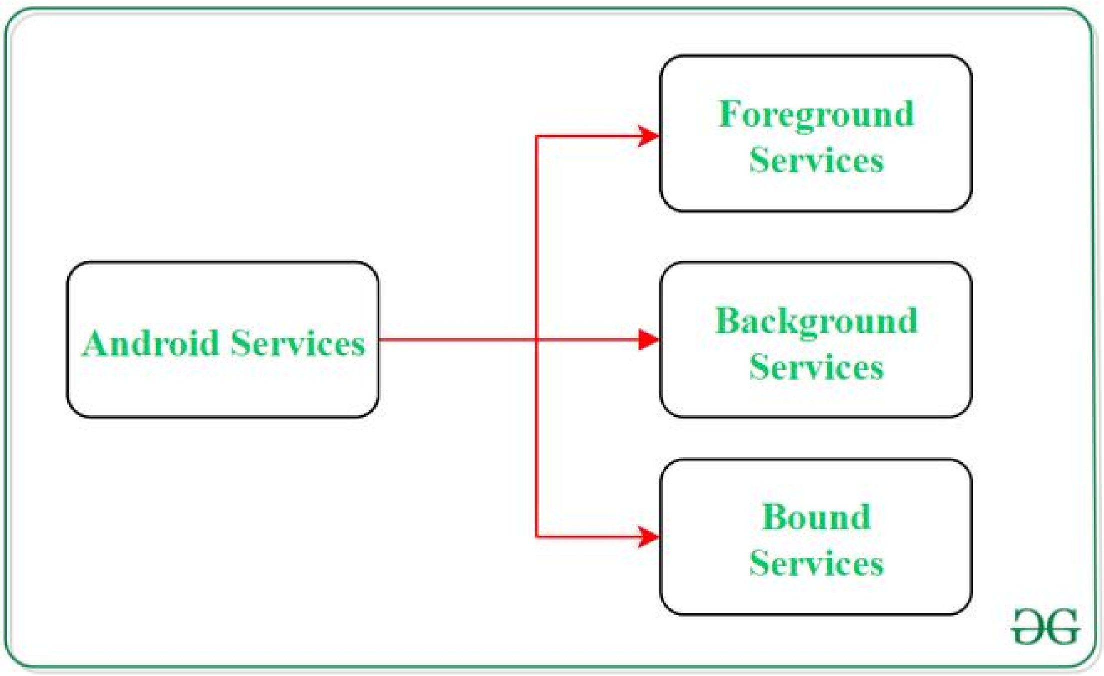
Further, application components can bind itself to service to carry out inter-process communication(IPC) .

There is a major difference between android services and threads, one must not be confused between the two.

Thread is a feature provided by the Operating system to allow the user to perform operations in the background.

While service is an android component that performs a long-running operation about which the user might not be aware of as it does not have UI.

Types of Android Services



1. Foreground Services:

Services that notify the user about its ongoing operations are termed as Foreground Services. Users can interact with the service by the notifications provided about the ongoing task. Such as in downloading a file, the user can keep track of the progress in downloading and can also pause and resume the process.

2. Background Services:

Background services do not require any user intervention. These services do not notify the user about ongoing background tasks and users also cannot access them. The process like schedule syncing of data or storing of data fall under this service.

3. Bound Services:

This type of android service allows the components of the application like activity to bound themselves with it. Bound services perform their task as long as any application component is bound to it. More than one component is allowed to bind themselves with a service at a time. In order to bind an application component with a service **bindService()** method is used.

The Life Cycle of Android Services

In android, services have 2 possible paths to complete its life cycle namely **Started and Bounded** .

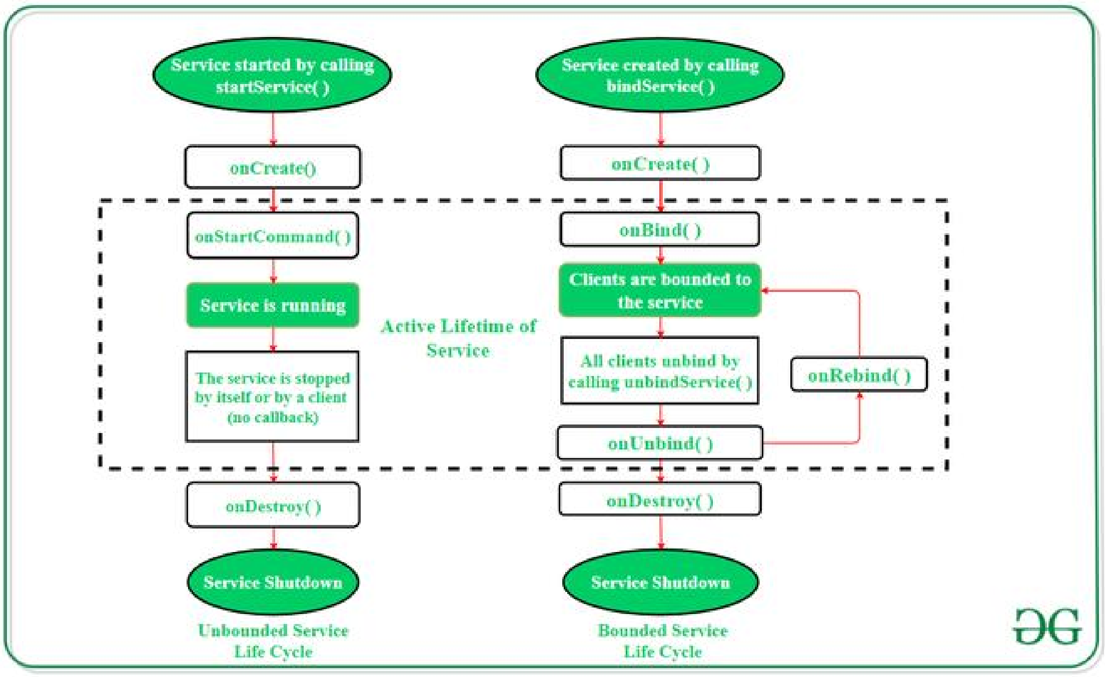
1. Started Service (Unbounded Service):

By following this path, a service will initiate when an application component calls the **startService()** method. Once initiated, the service can run continuously in the background even if the component is destroyed which was responsible for the start of the service. Two option are available to stop the execution of service:

- By calling **stopService()** method,
- The service can stop itself by using **stopSelf()** method.

2. Bounded Service:

It can be treated as a server in a client-server interface. By following this path, android application components can send requests to the service and can fetch results. A service is termed as bounded when an application component binds itself with a service by calling **bindService()** method. To stop the execution of this service, all the components must unbind themselves from the service by using **unbindService()** method.


Service started by calling startService( )
onCreate()
onStartCommand( )
Service is running
The service is stopped by itself or by a client (no callback)
onDestroy( )
Service Shutdown
Unbounded Service Life Cycle
Active Lifetime of Service
Service created by calling bindService( )
onCreate( )
onBind( )
Clients are bounded to the service
All clients unbind by calling unbindService( )
onRebind( )
onUnbind( )
onDestroy( )
Service Shutdown
Bounded Service Life Cycle

*To carry out a downloading task in the background, the **startService()** method will be called. Whereas to get information regarding the download progress and to pause or resume the process while the application is still in the background, **the service must be bounded with a component** which can perform these tasks.*

Fundamentals of Android Services

A user-defined service can be created through a normal class which is extending the **class Service** . Further, to carry out the operations of service on applications, there are certain callback methods which are needed to be **overridden** . The following are some of the important methods of Android Services:

Methods	Description
onStartCommand()	The Android service calls this method when a component(eg: activity) requests to start a service using startService(). Once the service is started, it can be stopped explicitly using stopService() or stopSelf() methods.
onBind()	This method is mandatory to implement in android service and is invoked whenever an application component calls the bindService() method in order to bind itself with a service. User-interface is also provided to communicate with the service effectively by returning an IBinder object. If the binding of service is not required then the method must return null.
onUnbind()	The Android system invokes this method when all the clients get disconnected from a particular service interface.
onRebind()	Once all clients are disconnected from the particular interface of service and there is a need to connect the service with new clients, the system calls this method.
onCreate()	Whenever a service is created either using onStartCommand() or onBind(), the android system calls this method. This method is necessary to perform a one-time-set-up.
onDestroy()	When a service is no longer in use, the system invokes this method just before the service destroys as a final clean up call. Services must implement this method in order to clean up resources like

Methods	Description
	registered listeners, threads, receivers, etc.

Example of Android Services

Playing music in the background is a very common example of services in android . From the time when a user starts the service, music play continuously in the background even if the user switches to another application. The user has to stop the service explicitly in order to pause the music. Below is the complete step-by-step implementation of this android service using a few callback methods.

Note *: Following steps are performed on Android Studio version 4.0*

Step 1: Create a new project

1. Click on File, then New => New Project.
2. Choose Empty Views Activity
3. Select language as Java/Kotlin
4. Select the minimum SDK as per your need.

Step 2: Modify strings.xml file

All the strings which are used in the activity are listed in this file.

```
<resources>    <string name="app_name">Services_In_Android</string>    <string
name="heading">Services In Android</string>    <string
name="startButtonText">Start the Service</string>    <string
name="stopButtonText">Stop the Service</string></resources>
```

Step 3: Working with the activity_main.xml file

Open the **activity_main.xml** file and add 2 Buttons in it which will start and stop the service. Below is the code for designing a proper activity layout.

```
<?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"
android:background="#168BC34A"    tools:context=".MainActivity">
   <LinearLayout       android:id="@+id/linearLayout"
android:layout_width="match_parent"       android:layout_height="wrap_content"
android:layout_centerVertical="true"       android:orientation="vertical"
app:layout_constraintBottom_toBottomOf="parent"
```

```
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="1.0"       tools:ignore="MissingConstraints">
      <TextView          android:id="@+id/textView1"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginBottom="170dp"          android:fontFamily="@font/roboto"
android:text="@string/heading"          android:textAlignment="center"
android:textAppearance="@style/TextAppearance.AppCompat.Large"
android:textColor="@android:color/holo_green_dark"
android:textSize="36sp"          android:textStyle="bold" />
      <Button          android:id="@+id/startButton"
android:layout_width="match_parent"
android:layout_height="match_parent"          android:layout_marginStart="20dp"
android:layout_marginTop="10dp"          android:layout_marginEnd="20dp"
android:layout_marginBottom="20dp"          android:background="#4CAF50"
android:fontFamily="@font/roboto"          android:text="@string/startButtonText"
android:textAlignment="center"
android:textAppearance="@style/TextAppearance.AppCompat.Display1"
android:textColor="#FFFFFF"          android:textStyle="bold" />
      <Button          android:id="@+id/stopButton"
android:layout_width="match_parent"
android:layout_height="match_parent"          android:layout_marginStart="20dp"
android:layout_marginTop="10dp"          android:layout_marginEnd="20dp"
android:layout_marginBottom="20dp"          android:background="#4CAF50"
android:fontFamily="@font/roboto"          android:text="@string/stopButtonText"
android:textAlignment="center"
android:textAppearance="@style/TextAppearance.AppCompat.Display1"
android:textColor="#FFFFFF"          android:textStyle="bold" />
      <ImageView          android:id="@+id/imageView"
android:layout_width="match_parent"
android:layout_height="wrap_content"          android:layout_marginTop="80dp"
app:srcCompat="@drawable/banner" />    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 4: Creating the custom service class

A custom service class will be created in the same directory where the **MainActivity** class resides and this class will extend the **Service class** . The callback methods are used to initiate and destroy the services. To play music, the **MediaPlayer object** is used. Below is the code to carry out this task.

```
import android.app.Service;import android.content.Intent;import
android.media.MediaPlayer;import android.os.IBinder;import
android.provider.Settings;import androidx.annotation.Nullable;
public class NewService extends Service {
  // declaring object of MediaPlayer    private MediaPlayer player;
```

```
   @Override
   // execution of service will start    // on calling this method    public int
onStartCommand(Intent intent, int flags, int startId) {
      // creating a media player which       // will play the audio of Default       //
ringtone in android device       player = MediaPlayer.create( this,
Settings.System.DEFAULT_RINGTONE_URI );
      // providing the boolean       // value as true to play       // the audio on loop
player.setLooping( true );
      // starting the process       player.start();
      // returns the status       // of the program       return START_STICKY;    }
   @Override
   // execution of the service will    // stop on calling this method    public void
onDestroy() {       super.onDestroy();
      // stopping the process       player.stop();    }
   @Nullable    @Override    public IBinder onBind(Intent intent) {       return
null;    }}
```

Step 5: Working with the MainActivity file

Now, the button objects will be declared and the process to be performed on clicking these buttons will be defined in the MainActivity class. Below is the code to implement this step.

```
import androidx.appcompat.app.AppCompatActivity;import
android.content.Intent;import android.os.Bundle;import
android.view.View;import android.widget.Button;
public class MainActivity extends AppCompatActivity implements
View.OnClickListener {
   // declaring objects of Button class    private Button start, stop;
   @Override    protected void onCreate(Bundle savedInstanceState)
{       super.onCreate( savedInstanceState );
setContentView( R.layout.activity_main );
      // assigning ID of startButton       // to the object start       start = (Button)
findViewById( R.id.startButton );
      // assigning ID of stopButton       // to the object stop       stop = (Button)
findViewById( R.id.stopButton );
      // declaring listeners for the       // buttons to make them respond       //
correctly according to the process       start.setOnClickListener( this );
stop.setOnClickListener( this );    }
   public void onClick(View view) {
      // process to be performed       // if start button is clicked       if(view == start){
         // starting the service          startService(new Intent( this,
NewService.class ) );       }
      // process to be performed       // if stop button is clicked       else if (view ==
stop){
         // stopping the service          stopService(new Intent( this,
NewService.class ) );
      }    }}
```

Step 6: Modify the AndroidManifest.xml file

To implement the services successfully on any android device, it is necessary to mention the created service in the **AndroidManifest.xml** file. It is not possible for a service to perform its task if it is not mentioned in this file. The service name is mentioned inside the **application tag** .

```
<?xml version="1.0" encoding="utf-8"?><manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.services_in_android">
  <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/AppTheme">      <activity
android:name=".MainActivity">         <intent-filter>            <action
android:name="android.intent.action.MAIN" />            <category
android:name="android.intent.category.LAUNCHER" />         </intent-filter>
</activity>      <meta-data         android:name="preloaded_fonts"
android:resource="@array/preloaded_fonts" />          <!-- Mention the service
name here -->      <service android:name=".NewService"/>       </application>
</manifest>
```

4.2 Introduction to RESTful architecture

REST (**Representational State Transfer**) is a widely used architectural style for designing **networked applications**, especially **web services** consumed by Android apps. In Android development, RESTful architecture enables apps to communicate efficiently with **remote servers** to send and retrieve data over the internet.

What is REST?

REST is an architectural approach that uses standard **HTTP protocols** to perform operations on resources. Each resource is identified by a **URL (Uniform Resource Locator)** and manipulated using HTTP methods.

Key Principles of RESTful Architecture

1)Client–Server Architecture

Android app acts as the **client**

Server manages **data and business logic**

Separation improves scalability and maintainability

2)Statelessness

Each request from the Android app contains all the information needed

Server does not store client session data

Improves performance and reliability

3)Resource-Based Structure

Data is treated as resources (e.g., users, products)

Each resource is accessed via a unique URL

Example:

https://api.example.com/users/1

Uniform Interface

Uses standard HTTP methods

GET – Retrieve data

POST – Create new data

PUT – Update existing data

DELETE – Remove data

Representation of Resources

Resources are represented using formats like **JSON** or **XML**

JSON is most commonly used in Android apps

RESTful Web Services in Android

Android applications consume REST APIs to:

Fetch data from servers

Send user input to backend systems

Synchronize data between devices

Unit_4_Working with Data and API in android

Typical communication flow:

Android app sends an HTTP request

Server processes the request

server sends a response (usually JSON)

Android app parses and displays the data

Example of REST API Request

GET https://api.example.com/products

JSON Response:

```
[
 {
   "id": 1,
   "name": "Mobile Phone",
   "price": 15000
 }]
```

Benefits of RESTful Architecture in Android

Lightweight and fast communication

Platform and language independent

Easy integration with web services

Scalable and flexible

Uses standard HTTP protocols

Common Android Libraries for REST

1)Retrofit

2)Volley

3)OkHttp

4.3 Parsing JSON data in Android

Introduction

JSON (**JavaScript Object Notation**) is a lightweight, text-based data format used to exchange data between **Android applications** and **web servers**. Parsing JSON in Android means **converting JSON data into usable Java objects** so that the app can display or process the data.

Why JSON is Used in Android

Lightweight and easy to read

Faster than XML

Platform independent

Supported by most RESTful web services

Types of JSON Data

1. JSON Object

```
{
  "id": 1,
  "name": "Laptop",
  "price": 55000}
```

2. JSON Array

```
[
  {
    "id": 1,
    "name": "Laptop"
  },
  {
    "id": 2,
    "name": "Mobile"
  }]
```

JSON Parsing in Android

Unit_4_Working with Data and API in android

Android provides built-in classes in the **org.json** package to parse JSON data:

JSONObject

JSONArray

JSONTokener

Steps to Parse JSON Data

Receive JSON response from server

Convert response into JSONObject or JSONArray

Extract values using keys

Store or display the extracted data

Example: Parsing JSON Object

Sample JSON Response

```
{
  "name": "Tablet",
  "price": 20000,
  "brand": "Samsung"}
```

Android Code Example

```
try {
   JSONObject jsonObject = new JSONObject(response);
   String name = jsonObject.getString("name");
   int price = jsonObject.getInt("price");
   String brand = jsonObject.getString("brand");

} catch (JSONException e) {
   e.printStackTrace();
}
```

Example: Parsing JSON Array

Sample JSON Response

```
[
  {
    "id": 1,
    "name": "Laptop"
  },
  {
    "id": 2,
    "name": "Mobile"
  }]
```

Android Code Example

```
try {
    JSONArray jsonArray = new JSONArray(response);

    for (int i = 0; i < jsonArray.length(); i++) {
        JSONObject obj = jsonArray.getJSONObject(i);
        int id = obj.getInt("id");
        String name = obj.getString("name");
    }

} catch (JSONException e) {
    e.printStackTrace();
}
```

Parsing JSON Using Volley (Example)

```
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(
        Request.Method.GET,
        url,
        null,
        response -> {
            try {
                String name = response.getString("name");
                int price = response.getInt("price");
            } catch (JSONException e) {
                e.printStackTrace();
            }
        },
        error -> {
            // Handle error
        }
);
```

Advantages of JSON Parsing

Easy to implement

Reduces network bandwidth

Fast processing

Human-readable format

4.4 Testing API

API testing in Android ensures that the application correctly communicates with the **RESTful web service** and handles responses, errors, and performance efficiently.

Step 1: Understand the API

Read API documentation

Note:

Base URL

Endpoints

HTTP methods (GET, POST, PUT, DELETE)

Request parameters

Response format (JSON)

Example:

https://api.example.com/users

Step 2: Test API Using External Tools

Before integrating in Android, test APIs using tools like:

Postman

cURL

Swagger UI

✔ Verify response status codes (200, 201, 404, 500)
✔ Check JSON response structure

Step 3: Create Android Project

-

Open Android Studio

-
-

Create a new project

-
-

Select required SDK

-
-

Configure package name

-

Step 4: Add Internet Permission

Add permission in AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Step 5: Add Networking Library

Use **Volley** or **Retrofit**.

Example (Volley Dependency):

```
implementation 'com.android.volley:volley:1.2.1'
```

Step 6: Write API Request Code

Test the API inside the app.

Example: GET Request Using Volley

```
RequestQueue queue = Volley.newRequestQueue(this);String url =
"https://api.example.com/users";
JsonArrayRequest request = new JsonArrayRequest(
        Request.Method.GET,
        url,
        null,
        response -> {
           // API success
           Log.d("API_RESPONSE", response.toString());
        },
        error -> {
           Log.e("API_ERROR", error.toString());
        }
);

queue.add(request);
```

Step 7: Run Application and Check Output

Run app on emulator or real device

Use **Logcat** to check:

Response data

Errors

Network failure

Step 8: Validate API Response

Check:

Correct data received

JSON parsed properly

No crashes

Correct UI update

Step 9: Handle Error Scenarios

Test:

- No internet connection
- Wrong URL
- Invalid parameters
- Server error (500)

Example Error Handling:

```
error -> {
   Toast.makeText(this, "API Error", Toast.LENGTH_SHORT).show();
}
```

Step 10: Performance & Security Testing

- Test response time
- Use HTTPS
- Validate input data
- Test large data handling

API Testing Flow Diagram

```
User Action
    ↓
Android App
    ↓
API Request (HTTP)
    ↓
REST Server
    ↓JSON Response
    ↓
Parsing & Validation
```

↓
UI Update

4.5 Introduction to Volley libraries

Volley is an **HTTP library** that makes networking very easy and fast, for Android apps. It was developed by Google and introduced during Google I/O 2013. It was developed because there is an absence in Android SDK, of a networking class capable of working without interfering with the user experience. Although Volley is a part of the Android Open Source Project(AOSP), Google announced in January 2017 that Volley will move to a standalone library. It manages the processing and caching of network requests and it saves developers valuable time from writing the same network call/cache code again and again. Volley is **not suitable for large download or streaming operations** since Volley holds all responses in memory during parsing.

Features of Volley:

1. Request queuing and prioritization
2. Effective request cache and memory management
3. Extensibility and customization of the library to our needs
4. Cancelling the requests

Advantages of Using Volley

1. All the tasks that need to be done with Networking in Android, can be done with the help of Volley.
2. Automatic scheduling of network requests.
3. Catching
4. Multiple concurrent network connections.
5. Cancelling request API.
6. Request prioritization.
7. Volley provides debugging and tracing tools.

How to Import Volley and Add Permissions

Before getting started with Volley, one needs to import Volley and add permissions in the Android Project. The steps to do so are as follows:

Create a new project. Open **build.gradle(Module: app)** and add the following dependency:

```
dependencies{
   //...
   implementation 'com.android.volley:volley:1.2.1'
}
```

In **AndroidManifest.xml** add the internet permission:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Classes in Volley Library

Volley has two main classes:

1. **Request Queue:** It is the interest one uses for dispatching requests to the network. One can make a request queue on demand if required, but typically it is created early on, at startup time, and keep it around and use it as a Singleton.
2. **Request:** All the necessary information for making web API call is stored in it. It is the base for creating network requests(GET, POST).

Types of Request using Volley Library

There are many types of request using Volley Library mentioned below:

1. String Request

```
String url = "https:// string_url/"; StringRequest stringRequest     = new
StringRequest(       Request.Method.GET, url,        new Response.Listener()
{          @Override          public void onResponse(String response)
{                    }       },       new Response.ErrorListener() {          @Override
public void onErrorResponse(VolleyError error)          {                    }       });
requestQueue.add(stringRequest);
```

2. JSONObject Request

```
String url = "https:// json_url/"; JsonObjectRequest     jsonObjectRequest     = new
JsonObjectRequest(       Request.Method.GET, url, null,        new
Response.Listener() {          @Override          public void onResponse(JSONObject
response)          {          }       },       new Response.ErrorListener()
{          @Override          public void onErrorResponse(VolleyError error)
{          }       }); requestQueue.add(jsonObjectRequest);
```

3. JSONArray Request

```
JsonArrayRequest     jsonArrayRequest     = new
JsonArrayRequest(       Request.Method.GET,        url,        null,        new
Response.Listener() {          @Override          public void onResponse(JSONArray
response)          {          }       },       new Response.ErrorListener()
{          @Override          public void onErrorResponse(VolleyError error)
{          }       }); requestQueue.add(jsonArrayRequest);
```

4. Image Request

```
int max - width = ...; int max_height = ...;
String URL = "http:// image_url.png";
ImageRequest     imageRequest     = new ImageRequest(URL,                new
Response.Listener() {                    @Override                    public void
onResponse(Bitmap response)                    {                    // Assign the
response                    // to an ImageView                    ImageView
imageView                        = (ImageView)
findViewById(                        R.id.imageView);
                 imageView.setImageBitmap(response);               }             },
max_width, max_height, null);
requestQueue.add(imageRequest);
```

5. Adding Post Parameters

```
String tag_json_obj = "json_obj_req";
String url = "https:// api.xyz.info/volley/person_object.json";
ProgressDialog pDialog = new ProgressDialog(this);
pDialog.setMessage("Loading...PLease wait"); pDialog.show();
JsonObjectRequest    jsonObjReq    = new JsonObjectRequest(      Method.POST,
url,       null,       new Response.Listener() {
       @Override         public void onResponse(JSONObject response)
{           Log.d(TAG, response.toString());           pDialog.hide();        }      },
new Response.ErrorListener() {
       @Override         public void onErrorResponse(VolleyError error)
{           VolleyLog.d(TAG, "Error: "                       + error.getMessage());
pDialog.hide();          }       }) {
     @Override      protected Map getParams()      {         Map params = new
HashMap();         params.put("name", "Androidhive");          params.put("email",
"abc@androidhive.info");         params.put("password", "password123");
       return params;       }
  };
AppController.getInstance()    .addToRequestQueue(jsonObjReq, tag_json_obj);
```

6. Adding Request Headers

```
String tag_json_obj = "json_obj_req";
String    url    = "https:// api.androidhive.info/volley/person_object.json";
ProgressDialog pDialog = new ProgressDialog(this);
pDialog.setMessage("Loading..."); pDialog.show();
JsonObjectRequest    jsonObjReq    = new JsonObjectRequest(      Method.POST,
url,       null,       new Response.Listener() {
       @Override         public void onResponse(JSONObject response)
{           Log.d(TAG, response.toString());           pDialog.hide();        }      },
new Response.ErrorListener() {
       @Override         public void onErrorResponse(VolleyError error)
{           VolleyLog.d(TAG, "Error: "                       + error.getMessage());
pDialog.hide();          }       }) {
     @Override      public Map getHeaders() throws AuthFailureError
{        HashMap headers = new HashMap();         headers.put("Content-Type",
"application/json");         headers.put("apiKey", "xxxxxxxxxxxxxxx");         return
headers;      }
```

```
   };
AppController.getInstance()    .addToRequestQueue(jsonObjReq, tag_json_obj);
```

7. Handling the Volley Cache

```
// Loading request from cache Cache     cache     =
AppController.getInstance()       .getRequestQueue()       .getCache();

Entry entry = cache.get(url);
if (entry != null) {
try
 {       String           data           = new String(entry.data, "UTF-8");        // handle
data, like converting it        // to xml, json, bitmap etc.,     }     catch
(UnsupportedEncodingException e) {        e.printStackTrace();     } } } else{    // If
cached response doesn't exists }
// Invalidate cache
AppController.getInstance()    .getRequestQueue()    .getCache()    .invalidate(url,
true);
// Turning off cache // String request StringRequest     stringReq     = new
StringRequest(....);
// disable cache stringReq.setShouldCache(false);
// Deleting cache for particular cache</strong>
AppController.getInstance()    .getRequestQueue()    .getCache()    .remove(url);
// Deleting all the cache
AppController.getInstance()    .getRequestQueue()    .getCache()    .clear(url);
```

8. Cancelling Request

```
// Cancel single request String tag_json_arry = "json_req";
ApplicationController.getInstance()    .getRequestQueue()    .cancelAll("feed_requ
est");
// Cancel all request
ApplicationController.getInstance()    .getRequestQueue()    .cancelAll();
```

9. Request Prioritization

```
private Priority priority = Priority.HIGH;
StringRequest     strReq     = new StringRequest(       Method.GET,
Const.URL_STRING_REQ,       new Response           .Listener() {
               @Override               public void onResponse(String response) {
                  Log.d(TAG, response.toString());
msgResponse.setText(response.toString());                 hideProgressDialog();
               } },       new Response           .ErrorListener() {
```

```
            @Override               public void
onErrorResponse(VolleyError error) {
                VolleyLog.d(TAG,                    "Error: "
+ error.getMessage());          hideProgressDialog();           } }) {
    @Override       public Priority getPriority()     {       return priority;     }
  };
```

4.6 Creating GET and POST request in android.

Volley request with the help of **GET** request in Android. With the help of GET Request, we can display data from API in JSON format and use that data inside our application. In this article, we will take a look at posting our data to API using the **POST** request with the Volley library in Android.

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio.

*Note that select **Java** as the programming language.*

Step 2: Add the below dependency in your build.gradle file

Below is the dependency for Volley which we will be using to get the data from API. For adding this dependency navigate to the **app > Gradle Scripts > build.gradle(app)** and add the below dependency in the dependencies section.

```
implementation 'com.android.volley:volley:1.2.1'
```

After adding this dependency sync your project and now move towards the **AndroidManifest.xml** part.

Step 3: Adding permissions to the internet in the AndroidManifest.xml file

Navigate to the **app > AndroidManifest.xml** and add the below code to it.

```
<!--permissions for INTERNET--><uses-permission
android:name="android.permission.INTERNET"/>
```

Step 4: Working with the activity_main.xml file

Navigate to the **app > res > layout > activity_main.xml** and add the below code to that file. Below is the code for the **activity_main.xml** file.

```
<?xml version="1.0" encoding="utf-8"?><LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
```

```
android:layout_width="match_parent"    android:layout_height="match_parent"
android:orientation="vertical"    tools:context=".MainActivity">
   <!--edit text field for adding name-->    <EditText
android:id="@+id/idEdtName"       android:layout_width="match_parent"
android:layout_height="wrap_content"       android:layout_margin="10dp"
android:layout_marginTop="40dp"       android:hint="Enter your name" />
   <!--edit text for adding job-->    <EditText       android:id="@+id/idEdtJob"
android:layout_width="match_parent"       android:layout_height="wrap_content"
android:layout_margin="10dp"       android:hint="Enter your job" />
   <!--button for adding data-->    <Button       android:id="@+id/idBtnPost"
android:layout_width="match_parent"       android:layout_height="wrap_content"
android:layout_margin="20dp"       android:text="Send Data to API"
android:textAllCaps="false" />
   <!--text view for displaying our API response-->    <TextView
android:id="@+id/idTVResponse"       android:layout_width="match_parent"
android:layout_height="wrap_content"       android:layout_margin="10dp"
android:gravity="center_horizontal"       android:text="Response"
android:textAlignment="center"       android:textSize="15sp" />
   <!--progress bar for loading -->    <ProgressBar
android:id="@+id/idLoadingPB"       android:layout_width="wrap_content"
android:layout_height="wrap_content"       android:layout_gravity="center"
android:visibility="gone" />
</LinearLayout>
```

Step 5: Working with the MainActivity.java file

Go to the **MainActivity.java** file and refer to the following code. Below is the code for the **MainActivity.java** file. Comments are added inside the code to understand the code in more detail.

MainActivity.java:

```
import android.os.Bundle;import android.view.View;import
android.widget.Button;import android.widget.EditText;import
android.widget.ProgressBar;import android.widget.TextView;import
android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.android.volley.Request;import
com.android.volley.RequestQueue;import com.android.volley.VolleyError;import
com.android.volley.toolbox.StringRequest;import
com.android.volley.toolbox.Volley;
import org.json.JSONException;import org.json.JSONObject;
import java.util.HashMap;import java.util.Map;
public class MainActivity extends AppCompatActivity {
   // creating variables for our edittext,    // button, textview and progressbar.
private EditText nameEdt, jobEdt;    private Button postDataBtn;    private
TextView responseTV;    private ProgressBar loadingPB;
```

```
   @Override    protected void onCreate(Bundle savedInstanceState)
{       super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);              // initializing our views
nameEdt = findViewById(R.id.idEdtName);       jobEdt = findViewById(R.id.idEdtJob);
postDataBtn = findViewById(R.id.idBtnPost);       responseTV =
findViewById(R.id.idTVResponse);       loadingPB = findViewById(R.id.idLoadingPB);
// adding on click listener to our button.       postDataBtn.setOnClickListener(new
View.OnClickListener() {          @Override          public void onClick(View v)
{             // validating if the text field is empty or not.             if
(nameEdt.getText().toString().isEmpty() && jobEdt.getText().toString().isEmpty())
{                 Toast.makeText(MainActivity.this, "Please enter both the values",
Toast.LENGTH_SHORT).show();                 return;              }            // calling a
method to post the data and passing our name and job.
postDataUsingVolley(nameEdt.getText().toString(),
jobEdt.getText().toString());          }       });   }
   private void postDataUsingVolley(String name, String job) {       // url to post our
data       String url = "https://reqres.in/api/users";
loadingPB.setVisibility(View.VISIBLE);            // creating a new variable for our
request queue       RequestQueue queue =
Volley.newRequestQueue(MainActivity.this);             // on below line we are
calling a string        // request method to post the data to our API       // in this we
are calling a post method.       StringRequest request = new
StringRequest(Request.Method.POST, url, new
com.android.volley.Response.Listener<String>() {          @Override          public
void onResponse(String response) {             // inside on response method we are
// hiding our progress bar             // and setting data to edit text as empty
loadingPB.setVisibility(View.GONE);             nameEdt.setText("");
jobEdt.setText("");                       // on below line we are displaying a success
toast message.             Toast.makeText(MainActivity.this, "Data added to API",
Toast.LENGTH_SHORT).show();                try {                // on below line we are
parsing the response               // to json object to extract data from it.
JSONObject respObj = new JSONObject(response);                              // below
are the strings which we                // extract from our json object.
String name = respObj.getString("name");                String job =
respObj.getString("job");                          // on below line we are setting this
string s to our text view.             responseTV.setText("Name : " + name + "\n" +
"Job : " + job);              } catch (JSONException e)
{                e.printStackTrace();             }          }       }, new
com.android.volley.Response.ErrorListener() {          @Override          public void
onErrorResponse(VolleyError error) {             // method to handle errors.
Toast.makeText(MainActivity.this, "Fail to get response = " + error,
Toast.LENGTH_SHORT).show();          }       }) {          @Override          protected
Map<String, String> getParams() {             // below line we are creating a map for
// storing our values in key and value pair.             Map<String, String> params =
new HashMap<String, String>();                       // on below line we are passing
our key             // and value pair to our parameters.             params.put("name",
name);          params.put("job", job);                        // at last we are
// returning our params.            return params;          }       };       // below line is
to make       // a json object request.       queue.add(request);   }}
```

4.7 API response handling

Making API Calls using Volley Library in Android

Last Updated : 16 Aug, 2022

-
-
-

Volley is an HTTP library that's used for caching and making a network request in Android applications. It is an HTTP library that makes networking for Android apps easier and most importantly, faster. API stands for Application Programming Interface. It is a way for two or more computer programs to communicate with each other. By using its products or services communicate with other products and services without having to know how they're implemented.

***Note**: This Android article covered in both **Java** and **Kotlin** languages.*

Step By Step Implementation:

Step 1: Create a New Project in Android Studio

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. The code for that has been given in both Java and Kotlin Programming Language for Android.

Step 2: Add internet permission to your app

Go to **app > manifest.xml** file and add the internet permission. Below is the code for the manifest file.

```
<?xml version="1.0" encoding="utf-8"?><manifest
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
package="com.example.gfgvolleyapicall">
   <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.GFGvolleyApiCall">       <activity
android:name=".MainActivity"          android:exported="true"
tools:ignore="WrongManifestParent">          <intent-filter>             <action
android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>       </activity>    </application>    <!-- adding internet permission --
>    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

Step 3: Add the Volley dependency to build.gradle (Module : app) file

Go to **app > Gradle Scripts > build.gradle (Module : app) file** and add the dependency. Below is the code for the build.gradle file.

```
plugins {
  id 'com.android.application'
}

android {
  compileSdk 31

  defaultConfig {
     applicationId "com.example.gfgvolleyapicall"
     minSdk 21
     targetSdk 31
     versionCode 1
     versionName "1.0"

     testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
  }

  buildTypes {
     release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
     }
  }
  compileOptions {
     sourceCompatibility JavaVersion.VERSION_1_8
     targetCompatibility JavaVersion.VERSION_1_8
  }
}

dependencies {

  implementation 'androidx.appcompat:appcompat:1.4.2'
  implementation 'com.google.android.material:material:1.6.1'
  implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
  testImplementation 'junit:junit:4.13.2'
  androidTestImplementation 'androidx.test.ext:junit:1.1.3'
  androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

  implementation 'com.android.volley:volley:1.2.1'     // adding volley
dependency
}
```

Step 4: Working with activity_main.xml

Navigate to the **app > res > layout > activity_main.xml** and add the below code to that file. Below is the code for the **activity_main.xml** file.

```
<?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"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
android:backgroundTint="@color/white"
app:layout_constraintTop_toTopOf="parent">
      <LinearLayout          android:layout_width="match_parent"
android:layout_height="wrap_content"          android:orientation="horizontal">
         <TextView             android:id="@+id/textView3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"             android:layout_weight="1"
android:textColor="@color/purple_700"             android:textSize="20sp"
android:textAlignment="center"             android:text="welcome to geeks for
geeks" />       </LinearLayout>       </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 5: Working with the MainActivity File

Go to the MainActivity File and refer to the following code. Below is the code for the **MainActivity File**. Comments are added inside the code to understand the code in more detail.

```
package com.example.gfgvolleyapicall;
import static android.content.ContentValues.TAG;
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;import android.util.Log;import android.widget.Toast;
import com.android.volley.Request;import
com.android.volley.RequestQueue;import com.android.volley.Response;import
com.android.volley.VolleyError;import
com.android.volley.toolbox.StringRequest;import
com.android.volley.toolbox.Volley;
public class MainActivity extends AppCompatActivity {    private RequestQueue
mRequestQueue;    private StringRequest mStringRequest;    private String url =
"https://run.mocky.io/v3/85cf9aaf-aa4f-41bf-b10c-308f032f7ccc";        @Override
protected void onCreate(Bundle savedInstanceState)
{       super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
       getData();    }
   private void getData() {       // RequestQueue initialized       mRequestQueue =
Volley.newRequestQueue(this);              // String Request initialized
mStringRequest = new StringRequest(Request.Method.GET, url, new
Response.Listener<String>() {          @Override          public void
onResponse(String response) {
        Toast.makeText(getApplicationContext(), "Response :" +
response.toString(), Toast.LENGTH_LONG).show();//display the response on
screen        }      }, new Response.ErrorListener() {        @Override        public
void onErrorResponse(VolleyError error) {          Log.i(TAG, "Error :" +
error.toString());        }      });
    mRequestQueue.add(mStringRequest);   }}
```

Multiple Choice Questions (MCQs)

What does API stand for?
a) Application Program Interface
b) Applied Programming Interface
c) Application Programming Internet
d) Applied Program Internet

Answer: a

Which protocol is mainly used by RESTful web services?
a) FTP
b) SMTP
c) HTTP
d) TCP

Answer: c

Which data format is most commonly used in REST APIs?
a) XML
b) JSON
c) HTML
d) CSV

Answer: b

Which HTTP method is used to retrieve data from a server?
a) POST
b) PUT
c) DELETE
d) GET

Answer: d

Which library is used for network operations in Android?
a) Retrofit
b) Glide
c) Volley
d) Picasso

Answer: c

Which tool is commonly used for API testing?
a) Android Studio
b) Postman
c) Firebase
d) SQLite

Answer: b

Which class is used to handle JSON objects in Android?
a) JSONArray
b) JSONObject
c) JSONParser
d) JSONHandler

Answer: b

Which HTTP method is used to send data to the server?
a) GET
b) POST
c) FETCH
d) SEND

Answer: b

Short Answer Questions (One Mark Each)

1. Define API.

2. What is a web service?

3. Expand REST.

4. Name any one HTTP method.

5. What is JSON?

6. Name one API testing tool.

7. What is Volley?

8. What is the use of GET method?

9. What is the use of POST method?

10. What is JSONArray?

11. What does REST architecture use for communication?

12. What is API response?

Long Answer Questions (4 Marks Each)

Explain API and web services used in Android applications.

Describe RESTful architecture and its HTTP methods.

Explain JSON parsing in Android with an example.

Explain the Volley library and its advantages.

Write a program to create a GET request using Volley.

Write a program to create a POST request using Volley.

Explain API testing and its importance.

Explain API response handling in Android applications.