Module 6

Networking and APIs

Que-1) Explain the structure of a REST API. What is Retrofit in Android, and how does it simplify API calls?

Ans:

Structure of a REST API

A **REST API (Representational State Transfer API)** follows a client-server architecture and uses HTTP methods to perform operations on resources. The key elements of a REST API are:

1. **Base URL**
    - Example: https://api.example.com/
2. **HTTP Methods**
    - GET → Retrieve data
    - POST → Send new data
    - PUT → Update existing data
    - DELETE → Remove data
    - PATCH → Partially update data
3. **Endpoints (Resources)**
    - Example:
        - GET /users → Get all users
        - POST /users → Create a new user
        - GET /users/{id} → Get a specific user
        - PUT /users/{id} → Update a user
        - DELETE /users/{id} → Delete a user
4. **Request Headers**
    - Include metadata like authentication tokens, content type, etc.
5. **Request Body** (for POST, PUT, PATCH)
    - Data sent in JSON format

6. **Response Codes**
    - 200 OK → Success
    - 201 Created → Resource created
    - 400 Bad Request → Invalid request
    - 401 Unauthorized → Authentication required
    - 404 Not Found → Resource not found
    - 500 Internal Server Error → Server issue

What is Retrofit in Android?

Retrofit is a type-safe HTTP client for Android and Java that simplifies API calls by providing an abstraction over raw HTTP requests.

How Retrofit Simplifies API Calls?

1. **Easier API Interface Definition**
2. **Automatic JSON Parsing**
    - Uses **Gson** or **Moshi** to convert JSON responses into Java objects automatically.
3. **Handles Asynchronous Calls**
    - Supports **synchronous** and **asynchronous** API calls using Call.enqueue(), preventing blocking on the main thread.
4. **Built-in Error Handling**
    - Handles HTTP error codes and exceptions gracefully.
5. **Interceptor & Authentication Support**
    - Easily integrates with **OkHttp** for logging, authentication, and caching.

Example: Using Retrofit in Android

Step 1: Add Dependencies

Step 2: Define API Interface

Step 3: Create Retrofit Instance

Step 4: Make API Call

Que-2) What are the benefits of using Firebase in Android development? Explain Firebase Authentication and how it can be integrated with an Android app.

Ans:

Benefits of Using Firebase in Android Development

Firebase, developed by Google, is a **Backend-as-a-Service (BaaS)** platform that provides a suite of tools to help developers build feature-rich Android apps quickly and efficiently.

Key Benefits of Firebase in Android Development

1. **Real-time Database & Firestore**
   - **Cloud Firestore** and **Realtime Database** allow automatic data syncing across devices.
   - Works **offline** and syncs when online.
2. **Firebase Authentication**
   - Provides **secure** and **easy-to-implement** authentication methods:
     - **Email/Password Authentication**
     - **Google, Facebook, Twitter Sign-In**
     - **Phone Number Authentication**
     - **Anonymous Authentication**
3. **Firebase Cloud Messaging (FCM)**
   - Send **push notifications** and in-app messages for free.
4. **Firebase Crashlytics & Analytics**
   - **Crashlytics** detects real-time crashes and errors.
   - **Analytics** provides insights into user behaviour.
5. **Firebase Cloud Storage**
   - Securely store and retrieve **user-generated content** (images, videos, documents).
6. **Firebase Hosting & Cloud Functions**
   - **Hosting** provides fast and secure website hosting.
   - **Cloud Functions** allow server-side logic execution.
7. **Firebase ML Kit**
   - Pre-built **Machine Learning** models for **image recognition, text recognition, and translation**.

Firebase Authentication in Android

Firebase Authentication is a **secure and easy-to-use authentication system** that helps authenticate users.
It supports multiple **authentication providers** like:
Email & Password
Google Sign-In
Facebook, Twitter, and GitHub
Phone Authentication (OTP-based)
Anonymous Authentication

How to Integrate Firebase Authentication in an Android App

Step 1: Set Up Firebase in Android

1. Go to **Firebase Console**.
2. Click **"Create a Project"** → Add your **Android App**.
3. Register the package name (e.g., com.example.firebaseauth).
4. Download the google-services.json file and place it in the app/ folder.

Step 2: Add Firebase Authentication Dependencies

Step 3: Implement Email & Password Authentication

Step 4: Implement SignupActivity.kt

Step 5: Implement Logout in MainActivity.kt

Step 6: Enable Authentication in Firebase Console

1. Go to **Firebase Console** → Authentication → Sign-in methods.
2. Enable **Email/Password Authentication**.

Bonus: Google Sign-In Authentication

Que-3) Explain the concept of services in Android. What are the differences between foreground and background services, and when should each be used?

Ans:

Services in Android

A **Service** in Android is a component that runs in the background to perform long-running operations **without a user interface (UI)**. It is used for tasks like playing music, fetching data, or processing files.

Services can run even if the app is not in the foreground, making them useful for **background tasks that should continue independently of user interaction**.

Types of Services in Android

1 Foreground Service

- Runs **actively** and is **visible** to the user.
- Must display a **notification** in the status bar.
- Used for **long-running tasks** like **music playback, fitness tracking, or location updates**.

Example Use Cases:
Music players (Spotify, YouTube Music)
Ongoing call notifications (WhatsApp, Zoom)
Navigation apps (Google Maps)

2 Background Service

- Runs **silently in the background** without user interaction.
- Does **not** display a **notification**.
- Can be **restricted by Android** due to **battery optimizations (Doze Mode, App Standby)**.

Example Use Cases:
Fetching emails in the background
Syncing data with a server
Uploading files when the app is closed

3 Bound Service

- Allows **communication between components** (like an Activity and Service).
- Binds using **Binder** to send and receive data.
- Used when **multiple components** need to interact with a Service.

Example Use Cases:
Music app controlling playback from multiple activities
Fetching stock prices for multiple UI screens

Foreground vs Background Service: Key Differences

Feature	Foreground Service	Background Service
Visibility	Always visible (Notification required)	Runs invisibly
Execution Time	Long-running	May be killed by system if inactive
Use Cases	Music, calls, location tracking	Data sync, auto-backups
Android Restrictions	Less restrictive	Limited by battery optimizations
User Interaction	Direct (ongoing process)	Indirect (silent tasks)

Implementing Services in Android

1. Creating a Foreground Service

A **Foreground Service** needs a **persistent notification** to keep running.

Step 1: Create a Service Class

Step 2: Start the Foreground Service

Stopping the Foreground Service

2. Creating a Background Service

A **Background Service** does not require a persistent notification but may be restricted by **Android's Doze mode**.

Step 1: Create a Background Service

Step 2: Start the Background Service

Stopping the Background Service

When to Use Foreground vs. Background Services?

Use Case	Service Type
Playing music, fitness tracking, calls	**Foreground Service**
Location tracking, navigation	**Foreground Service**
Syncing data, fetching updates	**Background Service**
Uploading files, performing backups	**Background Service**

Newer Alternatives to Background Services

Due to **Android 8+ (Oreo) restrictions**, you should use alternatives like:
WorkManager → For deferrable background tasks (e.g., periodic sync).
JobIntentService → For short-lived tasks (e.g., background data processing).

Foreground Service → When tasks require a persistent process (e.g., music playback).

Que-4) Describe the principles of Material Design. What are the key elements, and how do they improve the user experience?

Ans:

Principles of Material Design

Material Design is a design system created by Google that provides guidelines for creating visually appealing, intuitive, and functional digital experiences. It is based on the concept of **"material as a metaphor,"** meaning it simulates real-world tactile surfaces while leveraging the benefits of digital technology.

Core Principles

1. **Material as a Metaphor**
    - Inspired by paper and ink, it introduces a sense of depth, layers, and motion.
    - Uses realistic lighting, shadows, and surfaces to create a more natural interaction experience.
2. **Bold, Graphic, and Intentional**
    - Uses strong typography, vibrant colors, and meaningful whitespace.
    - Emphasizes clarity and hierarchy to guide users through the interface.
3. **Motion Provides Meaning**
    - Animations and transitions help users understand changes in UI.
    - Motion enhances user feedback, making interactions feel more responsive.

Key Elements of Material Design

1. **Material Surfaces and Elevation**
    - UI components behave like physical objects with defined layers and shadows.
    - Depth is used to indicate hierarchy and importance.

2. **Typography and Readability**
    - Uses **Roboto** and other recommended fonts for consistency.
    - Establishes a clear typographic hierarchy to improve legibility.
3. **Color and Theming**
    - Uses a primary and secondary color palette.
    - Encourages dynamic theming with light and dark modes.
4. **Components and Layout**
    - Predefined UI components (e.g., buttons, cards, modals) ensure consistency.
    - Grid-based layouts help with responsive design.
5. **Motion and Interaction**
    - Provides smooth animations for state changes (e.g., button presses, loading).
    - Encourages intuitive gestures like swiping and dragging.
6. **Iconography and Imagery**
    - Uses simple, recognizable icons with a uniform style.
    - Integrates meaningful images to enhance communication.

How Material Design Improves User Experience

Consistency: Ensures a unified look and feel across platforms (Android, web, iOS).
Intuitiveness: Real-world metaphors make digital interactions more natural.
Accessibility: Focuses on readability, color contrast, and touch-friendly elements.
Efficiency: Predefined components reduce development time while maintaining quality.

Material Design is widely used in Google products and beyond, providing a refined balance between aesthetics and usability.