Module 6): Networking and APIs

• Theory Assignment:

Q. : 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 (Representational State Transfer) API is a standard architectural style for designing networked applications. The main components of a REST API structure are:

1. Base URL: The root address of the API that all endpoints build upon.
    - Example: https://api.example.com/
2. Endpoints: Specific paths appended to the base URL that represent resources.
    - Example: /users, /products, /orders
3. HTTP Methods: Define the type of operation to perform.
    - GET: Retrieve data. ○ POST: Create new data.
    - PUT or PATCH: Update existing data.
    - DELETE: Remove data.
4. Headers: Metadata sent with requests (e.g., authentication tokens, content type).

   Example: Authorization: Bearer <token>

   Content-Type: application/json

5. Request Body: The payload sent with POST or PUT methods, often in JSON format.
    - Example: {

      "name": "John Doe",

      "email": "johndoe@example.com"

      }

6. Response: The server's reply to a request, typically in JSON format.
- Example: {
  "status": "success",

```
  "data": { "id": 1, "name": "John Doe" }
}
```

Retrofit in Android :

Retrofit is a type-safe HTTP client for Android and Java that simplifies making API calls by abstracting the details of network operations.

Key Features of Retrofit:

- Simplifies API Requests: Converts API endpoints into simple Java/Kotlin interface methods.
- Support for Serialization: Automatically converts JSON responses to Java/Kotlin objects using libraries like Gson or Moshi.
- HTTP Method Annotation: Define HTTP methods directly in the interface (@GET, @POST, etc.).
- Customizable: Allows adding headers, interceptors, and authentication easily.
- Asynchronous Support: Works with Coroutines, RxJava, or traditional callbacks for handling API responses.

Retrofit Simplifies API Calls :

1. Define API Endpoints in an Interface: You declare API endpoints as Kotlin/Java methods with annotations like @GET or @POST.

```
interface ApiService {
@GET("users/{id}")
   suspend fun getUser(@Path("id") userId: Int): Response<User>

   @POST("users")
   suspend fun createUser(@Body newUser: User): Response<User>
}
```

2. Set Up Retrofit Instance: You configure the base URL and a JSON converter (e.g., Gson).

```
val retrofit = Retrofit.Builder()
.baseUrl("https://api.example.com/")
   .addConverterFactory(GsonConverterFactory.create())
   .build()

val apiService = retrofit.create(ApiService::class.java)
```

3. Make API Calls Easily:

Using Coroutines:

```
GlobalScope.launch {
   val response = apiService.getUser(1)
   if (response.isSuccessful) {
val user = response.body()
      println("User: $user")
   }
}
```

4. Automatic Serialization/Deserialization: Retrofit automatically converts JSON responses to objects and vice versa using libraries like Gson or Moshi.

Firebase Integration

• Theory Assignment:

Q :- 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 provides a set of backend services that simplify Android development by handling authentication, databases, cloud storage, push notifications, analytics, and more. Some key benefits include:

1. Easy Authentication – Supports various authentication methods like Email/Password, Google Sign-In, Facebook, Twitter, and phone authentication.
2. Real-time Database – Firebase Realtime Database and Firestore allow real-time data syncing across devices.
3. Cloud Storage – Helps store and retrieve media files efficiently.
4. Push Notifications – Firebase Cloud Messaging (FCM) enables real-time push notifications.
5. Crash Reporting – Firebase Crashlytics helps track and fix crashes quickly.
6. Hosting and Deployment – Firebase Hosting provides a fast and secure way to deploy web apps.

7. Performance Monitoring – Helps in analyzing app performance for optimization.
8. Machine Learning – Built-in ML Kit allows easy implementation of machine learning features.

Firebase Authentication:

Firebase Authentication is a service that allows users to sign in to an app securely using different authentication methods. It supports:

- Email and Password Authentication
- Google Sign-In
- Phone Number Authentication
- Social Media Login (Facebook, Twitter, etc.)
- Anonymous Authentication

Steps to Integrate Firebase Authentication in an Android App:

1. Add Firebase to Your Android Project

- Go to the Firebase Console.
- Create a new project and register your Android app.
- Download the google-services.json file and place it in the app directory.
- Add Firebase dependencies in build.gradle.

2. Enable Authentication Method

- In Firebase Console → Authentication → Sign-in Method → Enable "Email/Password" (or other methods you need).

3. Implement Firebase Email & Password Authentication

Sign Up

Advanced Android Concepts

• Theory Assignment:

Q :- 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 needing a user interface. Services are mainly used for tasks such as playing music, fetching data from a server, handling network transactions, and running background tasks.
Types of Services in Android
There are three main types of services in Android:

1. Foreground Services
2. Background Services

1. Foreground Services

A foreground service is a service that the user is actively aware of and is performing a noticeable operation. These services must display a persistent notification in the status bar.

When to Use Foreground Services?

- Playing music or podcasts
- Tracking location (e.g., GPS tracking apps)
- File downloads or uploads
- Running tasks that require user awareness

2. Background Services

A background service runs without direct user interaction. However, since Android 8.0 (Oreo), background execution is limited to improve battery performance.

When to Use Background Services?

- Syncing data in the background
- Periodic updates (e.g., checking for new messages)
- Executing scheduled tasks
-

Note: For background tasks, WorkManager, JobScheduler, or AlarmManager should be used instead of background services to avoid system restrictions.

Material Design and Animations

• Theory Assignment:

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

Ans. => Material Design Principles and Key Elements

Material Design is a design language developed by Google that focuses on creating a consistent, intuitive, and visually appealing user experience across platforms and devices. It is based on real-world material properties, motion, and depth to provide a seamless and engaging user interface.

1. Principles of Material Design

1.1 Material as a Metaphor
- Inspired by real-world materials (like paper and ink).
- Uses shadows, depth, and surfaces to create a realistic UI.
- Enhances user interaction by giving elements a physical feel.

1.2 Bold, Graphic, and Intentional
- Uses vibrant colors, typography, and imagery to grab attention.
- Focuses on clear hierarchy, large typography, and contrasting colors.
- Ensures important elements stand out.

1.3 Motion Provides Meaning
- Uses smooth animations to provide feedback and guide users.
- Transitions should be natural and seamless, avoiding sudden changes.
- Elements should react to user interactions with meaningful motion effects.

---

2. Key Elements of Material Design

2.1 Material Components
Google provides a set of pre-designed UI components that follow Material Design guidelines.
Examples:
- Buttons (Elevated, Text, Floating Action Button)
- Cards (To group related content)
- Navigation Drawer (For app navigation)
- App Bar & Toolbars (For branding and actions)
- Bottom Sheets (For additional options without leaving the screen)

2.2 Elevation & Shadows

- Uses z-index (depth) to create a sense of hierarchy.
- Higher elevation elements cast stronger shadows, making them look closer to the user.
- Example: Floating Action Button (FAB) appears raised above other elements.

2.3 Colors & Theming

Material Design defines a color system:

- Primary Color (Main brand color)
- Secondary Color (Accent or highlight)
- Surface Color (Background of elements)
- On Colors (Text/icons that contrast against backgrounds)

2.4 Typography

Uses a typography hierarchy for readability:

- Headline (Large titles)
- Subtitle (Smaller secondary text)
- Body Text (General content)
- Caption (Smallest text)

Recommended font: Roboto

2.5 Responsive Layout & Adaptive UI

- Supports different screen sizes (mobile, tablet, desktop).
- Uses Grid-Based Layouts for consistency.
- Supports Dark Mode for better accessibility.

---

3. How Material Design Improves User Experience

Consistency – Provides a uniform look across apps.

Accessibility – Ensures readability and usability for all users.

Engaging UX – Smooth animations and interactions enhance user experience.

Efficiency – Predefined guidelines speed up UI development.