**TEST PREPARATION QUESTION BANK for ICT ELECTIVES 2 - Kotlin**

You are required to focus on the slides and consult the codelabs when necessary, during your revision/preparations.

These questions will give you an idea of what to expect. The essay questions and some Multiple-choice questions require critical thinking to be able to produce the answers

1. Most Android apps connect to the internet to perform \_\_\_\_\_\_\_\_\_\_ operations, such as retrieving emails or messages.

Network operations

1. Open source and community-driven libraries are commonly used to build a data layer and retrieve data from a backend \_\_\_\_\_\_\_\_\_\_.

Backend server

1. Most web servers today use the \_\_\_\_\_\_\_\_\_\_ architecture for web services.

3-tier

1. REST stands for REpresentational State \_\_\_\_\_\_\_\_\_\_.

Transfer

1. A URL is a subset of a URI that specifies where a resource exists and the \_\_\_\_\_\_\_\_\_\_ for retrieving it.

Mechanism

1. Lazy initialization creates the \_\_\_\_\_\_\_\_\_\_ but not the object.

Reference

1. In Android development, understanding how to establish a \_\_\_\_\_\_\_\_\_\_ and communicate with a web service is vital for successful data retrieval.

Network connection

1. If you do not use "mutableStateOf()" to declare state in Compose, the state remains \_\_\_\_\_\_\_\_.

Immutable

1. A URI identifies a resource in the server by \_\_\_\_\_\_\_\_\_\_.

It’s path / name

1. Converters, such as ScalarsConverter, help Retrofit process the data it sends to and receives from the web service, treating it as a \_\_\_\_\_\_\_\_\_\_ or other primitive.

String

1. Most web servers today use the \_\_\_\_\_\_\_\_\_\_ architecture for web services.

3

1. Requests are made to RESTful web services in a standardized way, via Uniform Resource \_\_\_\_\_\_\_\_\_\_\_

Identifier

1. A URI identifies a resource in the server by \_\_\_\_\_\_\_\_\_\_.

~~Reference~~ it’s path

1. Converters, such as ScalarsConverter, help Retrofit process the data it sends to and receives from the web service, treating it as a \_\_\_\_\_\_\_\_\_\_ or other primitive.
2. Lazy initialization creates the \_\_\_\_\_\_\_\_\_\_ but not the object.
3. In Android development, understanding how to establish a \_\_\_\_\_\_\_\_\_\_ and communicate with a web service is vital for successful data retrieval.

Network

1. To preserve the new value of state across recompositions, you can use the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ composable inline function.

Remember

1. State hoisting in Compose is a pattern of moving state to a composable's caller to make the composable \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Stateless

1. A stateless composable doesn't own or modify \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

State

1. ViewModels in Compose provide the UI state and access to the business logic located in other layers of the app, ensuring a separation of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.+

Concerns?’

1. Why is it important for Android developers to learn what happens behind the scenes when apps send and receive data over the internet?

Performance optimization, error handling, security, resource management, compliance

1. Which type of libraries are commonly used to build a data layer and retrieve data from a backend server in Android development?

Retrofit, okHttp, Room, Volley, Kotlin coroutines, firebase realtime database/firebase

1. In the context of web services, what does REST stand for?

**Representational State Transfer**.

1. How do requests to RESTful web services typically occur?

Use http methods (get, post, put, delete)

Use URI

Use headers

1. What is the primary function of a Uniform Resource Locator (URL) in web services?

specify the location of a resource on the internet

1. Which term refers to software-based functionality offered over the internet that allows an app to make requests and retrieve data?

**Application Programming Interface (API)**.

1. What is a characteristic of RESTful web services?

A characteristic of RESTful web services is that they are **stateless**.

* + Client-server architecture
  + Uniform interface
  + Resource-based
  + Cacheable
  + Layered system

1. What is the primary purpose of the Retrofit library in Android development?

The primary purpose of the Retrofit library in Android development is to **simplify the process of making network requests to RESTful web services**

1. What is the role of converters, such as ScalarsConverter, in Retrofit?

The role of converters, such as ScalarsConverter, in Retrofit is to **transform the data sent to and received from web services** into a format that can be easily handled within the application.

1. What is the primary function of a Uniform Resource Locator (URL) in web services?
2. To enable an Android app to make connections to the internet, what permission must be added to the Android manifest?

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

1. What is the primary purpose of Room in Android app development?

The primary purpose of Room in Android app development is to provide a **persistent database library** that simplifies the process of managing local data storage using SQLite.

1. What is an Entity in the context of Room?

In the context of Room, an **Entity** represents a table within the database

1. What is LiveData used for in Android app development?

In Android app development, **LiveData** is used for **observing and reacting to data changes** in a lifecycle-aware manner. It is part of the Android Architecture Components and provides a way to handle data in a way that is responsive to the lifecycle of the app's components (such as activities and fragments).

1. What is the purpose of the Repository in the Android App Architecture?

The purpose of the **Repository** in Android App Architecture is to serve as a **centralized data management class** that abstracts the data sources and provides a clean API for data access to the rest of the application.

1. How do you define a table in Room?

~~GROOVY~~

~~dependencies {~~

~~def room\_version = "2.5.0" // Use the latest version available~~

~~implementation "androidx.room:room-runtime:$room\_version"~~

~~kapt "androidx.room:room-compiler:$room\_version" // For Kotlin, use kapt~~

~~implementation "androidx.room:room-ktx:$room\_version" // Optional for coroutines support~~

~~}~~

KOTLIN

**Defining user table**

import androidx.room.Entity

import androidx.room.PrimaryKey

import androidx.room.ColumnInfo

@Entity(tableName = "users")

data class User(

@PrimaryKey(autoGenerate = true) // Primary key with auto-generation

val id: Int = 0, // ID column

@ColumnInfo(name = "name") // Optional: Specify column name

val name: String, // Name column

@ColumnInfo(name = "email") // Optional: Specify column name

val email: String // Email column

)

**Create DAO**

import androidx.room.Dao

import androidx.room.Insert

import androidx.room.Query

@Dao

interface UserDao {

@Insert

suspend fun insert(user: User)

@Query("SELECT \* FROM users WHERE id = :userId")

suspend fun getUserById(userId: Int): User?

@Query("SELECT \* FROM users")

suspend fun getAllUsers(): List<User>

}

**Create database**

import androidx.room.Database

import androidx.room.Room

import androidx.room.RoomDatabase

import android.content.Context

@Database(entities = [User::class], version = 1)

abstract class AppDatabase : RoomDatabase() {

abstract fun userDao(): UserDao

companion object {

@Volatile

private var INSTANCE: AppDatabase? = null

fun getDatabase(context: Context): AppDatabase {

return INSTANCE ?: synchronized(this) {

val instance = Room.databaseBuilder(

context.applicationContext,

AppDatabase::class.java,

"app\_database"

).build()

INSTANCE = instance

instance

}

}

}

}

1. How do you define database queries in Room?

In Room, you define database queries using the **Data Access Object (DAO)** interface.

**Example DAO for a user entity**

import androidx.room.Dao

import androidx.room.Insert

import androidx.room.Update

import androidx.room.Delete

import androidx.room.Query

@Dao

interface UserDao {

// Insert a new user into the database

@Insert

suspend fun insert(user: User)

// Update an existing user in the database

@Update

suspend fun update(user: User)

// Delete a user from the database

@Delete

suspend fun delete(user: User)

// Retrieve a user by ID

@Query("SELECT \* FROM users WHERE id = :userId")

suspend fun getUserById(userId: Int): User?

// Retrieve all users from the database

@Query("SELECT \* FROM users")

suspend fun getAllUsers(): List<User>

// Retrieve users by name

@Query("SELECT \* FROM users WHERE name LIKE :name")

suspend fun findUsersByName(name: String): List<User>

}

1. Compare the roles of Entity and DAO in Room. What does each component handle in the database interaction process?

An **Entity** represents a table in the database. It defines the structure of the table, including its columns and their data types.

The primary purpose of an Entity is to model the data that will be stored in the database. It encapsulates the properties of the data and maps them to corresponding database columns.

**Example**

@Entity(tableName = "users")

data class User(

@PrimaryKey(autoGenerate = true) val id: Int = 0,

val name: String,

val email: String

)

A **DAO** is an interface (or abstract class) that provides methods for accessing the database. It defines the operations that can be performed on the Entity (like insert, update, delete, and query).

The primary purpose of a DAO is to abstract the database operations. It serves as an interface between the application logic and the database, allowing you to perform CRUD (Create, Read, Update, Delete) operations without needing to write raw SQL queries directly.

**Example**

@Dao

interface UserDao {

@Insert

suspend fun insert(user: User)

@Query("SELECT \* FROM users WHERE id = :userId")

suspend fun getUserById(userId: Int): User?

@Delete

suspend fun delete(user: User)

}

1. Analyze the significance of LiveData in Android app architecture. How does it improve the management of data and user interfaces?

In summary, LiveData significantly enhances data management and user interface management in Android applications by providing lifecycle awareness, enabling real-time data observation, ensuring automatic UI updates, and integrating seamlessly with ViewModels. By using LiveData, developers can create more responsive, maintainable, and reliable applications while adhering to modern architectural best practices. This ultimately leads to a better user experience and more efficient app development processes.

1. What is the primary role of a Repository in Android app architecture?
2. What is the primary benefit of having a single source of truth in data management?

The primary benefit of having a **single source of truth (SSOT)** in data management is **data consistency**. This concept refers to maintaining a unified and authoritative repository for data that serves as the definitive reference point for all applications and users.

1. In Android app development, which aspect of data management is typically abstracted by a Repository?

Here are the key aspects of data management that are abstracted by a Repository:

* + Data sources
  + Data operations
  + Data caching
  + Abstraction and implementation details

**Summary**

In summary, the Repository in Android app development abstracts the management of **data sources** and the logic for performing data operations. It provides a clean and unified API for accessing data, decouples the application from the specifics of data storage and retrieval, handles caching and synchronization, and simplifies testing. This abstraction contributes to a more maintainable, scalable, and testable codebase, following best practices in Android architecture, such as the MVVM (Model-View-ViewModel) pattern.

1. Which principle does the use of a Repository help to adhere to in software architecture?

The use of a **Repository** helps to adhere to the **Separation of Concerns** principle in software architecture. This principle emphasizes the need to separate a program into distinct sections, each responsible for a specific functionality or concern, allowing for more organized, maintainable, and scalable code.

1. How does the Repository pattern improve the testability of Android applications?

Decoupled architecture

Mocking data sources

Controlled testing environment

Focused tests

1. When implementing the Repository pattern, what benefits can be derived from separating data access and manipulation logic from other application logic?

Improved maintainability

Enhanced readability

Scalability

Testing flexibility

Flexibility in data sources

1. Imagine you are developing a shopping app. How would you design a Repository to handle both local data storage and remote data retrieval efficiently?

**Define repo interface**

interface ShoppingRepository {

suspend fun getProducts(): List<Product>

suspend fun getCartItems(): List<CartItem>

suspend fun addProductToCart(product: Product)

suspend fun syncCartWithRemote(): List<CartItem>

}

**Remote and local data sources**

interface LocalDataSource {

suspend fun getProducts(): List<Product>

suspend fun saveProducts(products: List<Product>)

suspend fun getCartItems(): List<CartItem>

suspend fun addProductToCart(product: Product)

}

interface RemoteDataSource {

suspend fun fetchProducts(): List<Product>

suspend fun syncCart(cartItems: List<CartItem>)

}

**Use viewmodel**

class ShoppingViewModel(private val repository: ShoppingRepository) : ViewModel() {

// Use LiveData or StateFlow to observe data in the UI

val products: LiveData<List<Product>> = liveData {

emit(repository.getProducts())

}

}

1. Analyze the importance of background thread handling by Retrofit during network requests. How does it impact the user experience in Android apps?
   * Prevents UI freezes
   * UX
     1. Smooth interactions
     2. Loeding indicators
   * Error handling
     1. Graceful error responses
   * Resource management
     1. Efficient resource utilization
   * Integration with coroutines
     1. Simplified asynchronous code

**Summary**

In summary, the handling of background threads by Retrofit during network requests is essential for maintaining responsiveness, improving user experience, enabling effective error handling, managing resources efficiently, and simplifying asynchronous programming with coroutines. By ensuring that network operations do not block the main thread, Retrofit enhances the overall quality and usability of Android applications, contributing to a more positive user experience.

1. Compare the roles of JSON objects and JSON arrays in representing structured data from web service responses. Which is typically used for grouping similar items?
2. In the context of creating a Retrofit service API, what is the purpose of defining a data model?
3. Why is it important to add Internet permissions to an Android app when using Retrofit?
4. Evaluate the advantages of using the Singleton pattern for managing Room Database instances in an Android app. Discuss its impact on performance and data integrity.
5. Imagine you are developing a social media app that fetches user posts from a server and stores them locally. Describe how you would implement a Repository to manage this data efficiently
6. What is the primary goal of using a Repository for data management?
7. Design a Room database schema for a task management app. Create an Entity class and a DAO interface to support task creation, retrieval, and deletion. Datababe table needs to have a minimum of four fields.
8. Explain the steps involved in creating a Room Database instance. Include the role of the abstract class and the database builder.
9. Compare and contrast the role of Room's Entity class and DAO interface in the context of database management Discuss how they interact with each other.
10. Evaluate the impact of using Room in Android app development compared to traditional SQLite database management. Discuss the benefits and potential challenges of adopting Room.
11. In the context of creating a Retrofit service API, why is it essential to add Internet permissions to an Android app? How does this impact app functionality?
12. Compare and contrast the roles of JSON objects and JSON arrays in representing structured data from web service responses.
13. Evaluate the significance of the kotlinx.serialization library in the context of Retrofit and Android app development. Discuss how it simplifies the handling of JSON data.
14. Design a Room database schema for a simple note-taking app. Define the Entity class and DAO interface to support note creation, retrieval, and deletion. The database needs to have a minimum of three data fields.
15. Compare and contrast the roles of Room's Entity class and DAO interface in the context of database management.
16. Suppose you are developing an e-commerce app with real-time inventory updates. Explain how a Repository can facilitate data synchronization between a server and the app's local database.
17. What are the core responsibilities of a Repository in Android development, and how does it achieve code separation?
18. Can a Repository manage data from multiple data sources simultaneously? If so, how?
19. Explain the significance of request and response serialization in Retrofit. How does this feature contribute to more efficient communication with web services?
20. Compare and contrast synchronous and asynchronous requests in Retrofit. When would you choose one over the other?
21. Evaluate the impact of using Room in Android app development compared to traditional SQLite database management. Discuss the benefits and potential challenges of adopting Room.
22. Evaluate the impact of a Repository on the architecture of large-scale Android applications. Discuss the potential challenges and benefits of using this pattern in such applications.
23. Describe the purpose of defining a data model in the context of creating a Retrofit service API.
24. In the context of creating a Retrofit service API, why is it essential to add Internet permissions to an Android app? How does this impact app functionality?
25. Evaluate the significance of the kotlinx.serialization library in the context of Retrofit and Android app development. Discuss how it simplifies the handling of JSON data.

Important

* + Database, database schema; room database
    1. Create DAO
    2. Set up database
  + States; state hoisting
    1. Create a preview