SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Mobile Application Development

Mobile Platforms and Application Development fundamentals

SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Data Handling in Android Applications
- Sensors and Media Handling in Android Applications
- Android Application Testing and security aspects

Persistance techniques in Android Applications

- Data persistence refers to the ability of an app to store data locally on a device even after the app is closed or the device is turned off.
- The choice of data persistence technique depends on the type and amount of data that needs to be stored, as well as the level of security and accessibility required for that data.
- There are several techniques available for data persistence in Android applications

Different techniques for data persistence in Android applications

- Shared Preferences
- Internal Storage
- External Storage
- SQLite Database
- Content Providers
- Cloud Storage

Shared Preference

- Shared Preferences are used to store primitive data types, such as Boolean, float, int, long, and string.
- This technique is suitable for small amounts of data and is used to store user preferences or settings.
- The key-value pairs are written to XML files that persist across user sessions, even if your app is killed. You can manually specify a name for the file or use per-activity files to save your data.

```
fun save(isLogged:Boolean, userName:String){
    val sharedPref = this
        .getSharedPreferences( name: "MyPrefs",Context.MODE_PRIVATE)
    val editor = sharedPref?.edit()
    editor?.putBoolean("isUserLoggedIn", isLogged)
    editor?.putString("username", userName)
    editor?.apply()
}
fun getUser(){
    val sharedPref = this
        .getSharedPreferences( name: "MyPrefs", Context.MODE_PRIVATE)
    val isUserLoggedIn = sharedPref?.getBoolean("isUserLoggedIn", false) ?: false
    val username = sharedPref?.getString("username", "")
}
```

Internal Storage

RAm, Flash Memory

- Internal storage is used to store private data in the device's memory.
- This technique is suitable for storing sensitive data that should not be accessible to other apps.
- The system provides a private directory on the file system for each app.
- When the user uninstalls your app, the files saved on the internal storage are removed.

```
fun writeToFile(){
    val filename = "myfile.txt"
    val fileContents = "Hello world!"

    this.openFileOutput(filename, Context.MODE_PRIVATE).use { it: FileOutputStream!
        it?.write(fileContents.toByteArray())
    }
}

fun readFromFile(){
    val filename = "myfile.txt"
    this.openFileInput(filename).use { it: FileInputStream!
        val fileContents = it?.bufferedReader().use { it: BufferedReader?
            it?.readText()
        }
        Log.d( tag: "MainActivity",  msg: "File contents: $fileContents") ^use
    }
}
```

External Storage

External hard drives

- External storage is used to store data in public directories that can be accessed by other apps or the user.
- This technique is suitable for storing large files or data that can be shared between different apps.

```
vate fun isExternalStorageWritable(): Boolear
 val state = Environment.getExternalStorageSt
 return Environment.MEDIA_MOUNTED == state
```

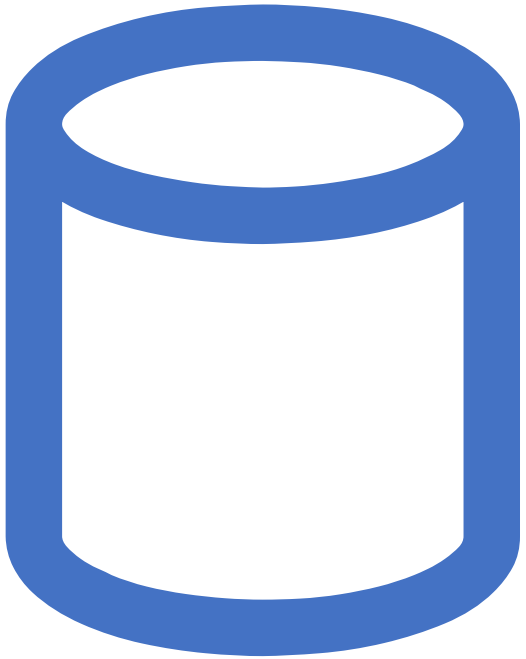
```
vate fun isExternalStorageReadable(): Boolear
 val state = Environment.getExternalStorageSt
 return Environment.MEDIA_MOUNTED == state ||
```

```
    if (isExternalStorageWritable()) {
        val file = File(getExternalFilesDir( type: null), filename)
        FileOutputStream(file).use { it: FileOutputStream
            it.write(fileContents.toByteArray())
        }
    }

}

fun readFromExternalFile(){
    val filename = "myfile.txt"

    if (isExternalStorageReadable()) {
        val file = File(getExternalFilesDir( type: null), filename)
        FileInputStream(file).use { it: FileInputStream
            val fileContents = it.bufferedReader().use { it: BufferedReader
                it.readText()
            }
```



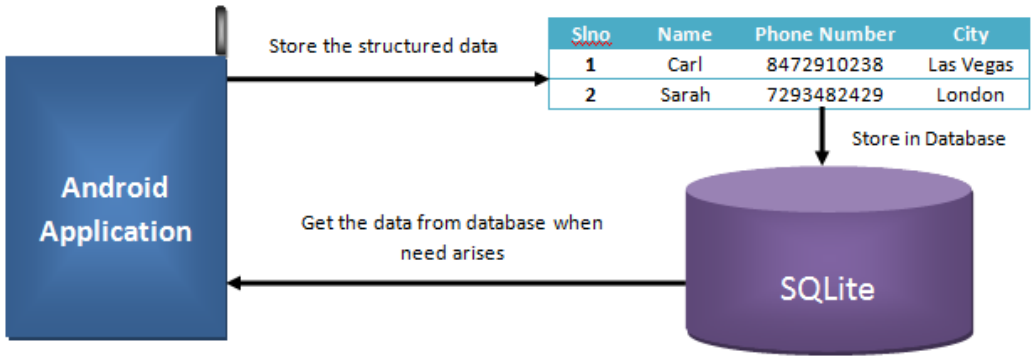
SQLite Databases

- SQLite is a lightweight database that can be used to store structured data.
- This technique is suitable for storing large amounts of structured data that can be queried and sorted.
- The Android SDK includes a sqlite3 shell tool that allows you to browse table contents, run SQL commands, and perform other useful functions on SQLite databases.

SQLite

- A SQL database engine which is developed using C that accesses its storage files directly.
- A software library that implements a self-contained, server less, zero-configuration, transactional SQL database engine.

Local database inside the android

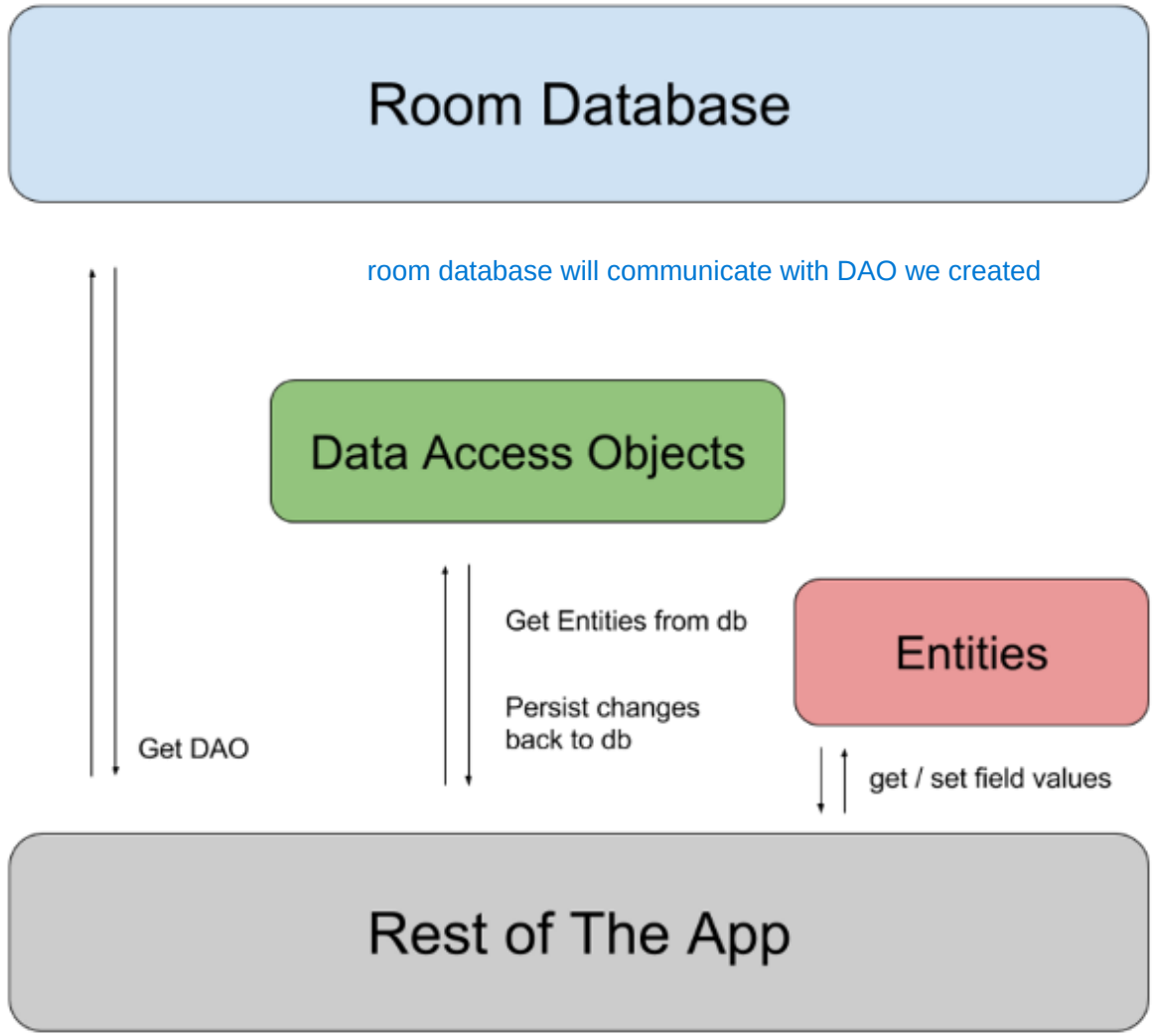


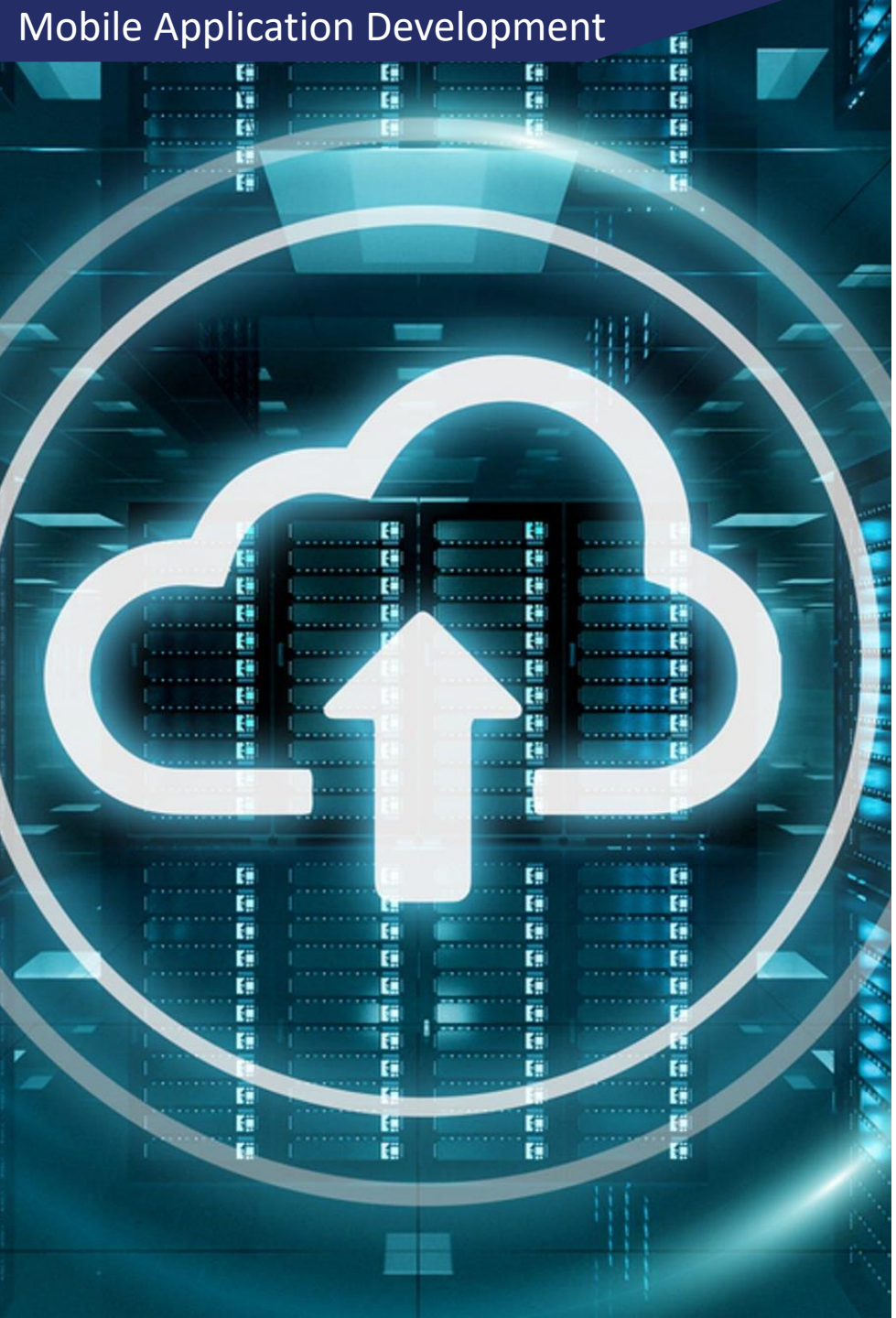
Save data in a local database using Room

- Apps that handle non-trivial amounts of structured data can benefit greatly from persisting that data locally.
  Structured data is typically organized in a format that makes it easy to access, manage, and analyze
- The most common use case is to cache relevant pieces of data so that when the device cannot access the network, the user can still browse that content while they are offline.
- The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite.
- Benefits of Room
  - Compile-time verification of SQL queries.
    when type a query it gives error message if there's an error.
  - Convenience annotations that minimize repetitive and error-prone boilerplate code.
  - Streamlined database migration paths.
    no of codes that we have to write is very less with this library
    content provider we have to write many lines of code

Primary components in Room

- The database class that holds the database and serves as the main access point for the underlying connection to your app's persisted data.
- Data entities that represent tables in your app's database.
- Data access objects (DAOs) that provide methods that your app can use to query, update, insert, and delete data in the database.

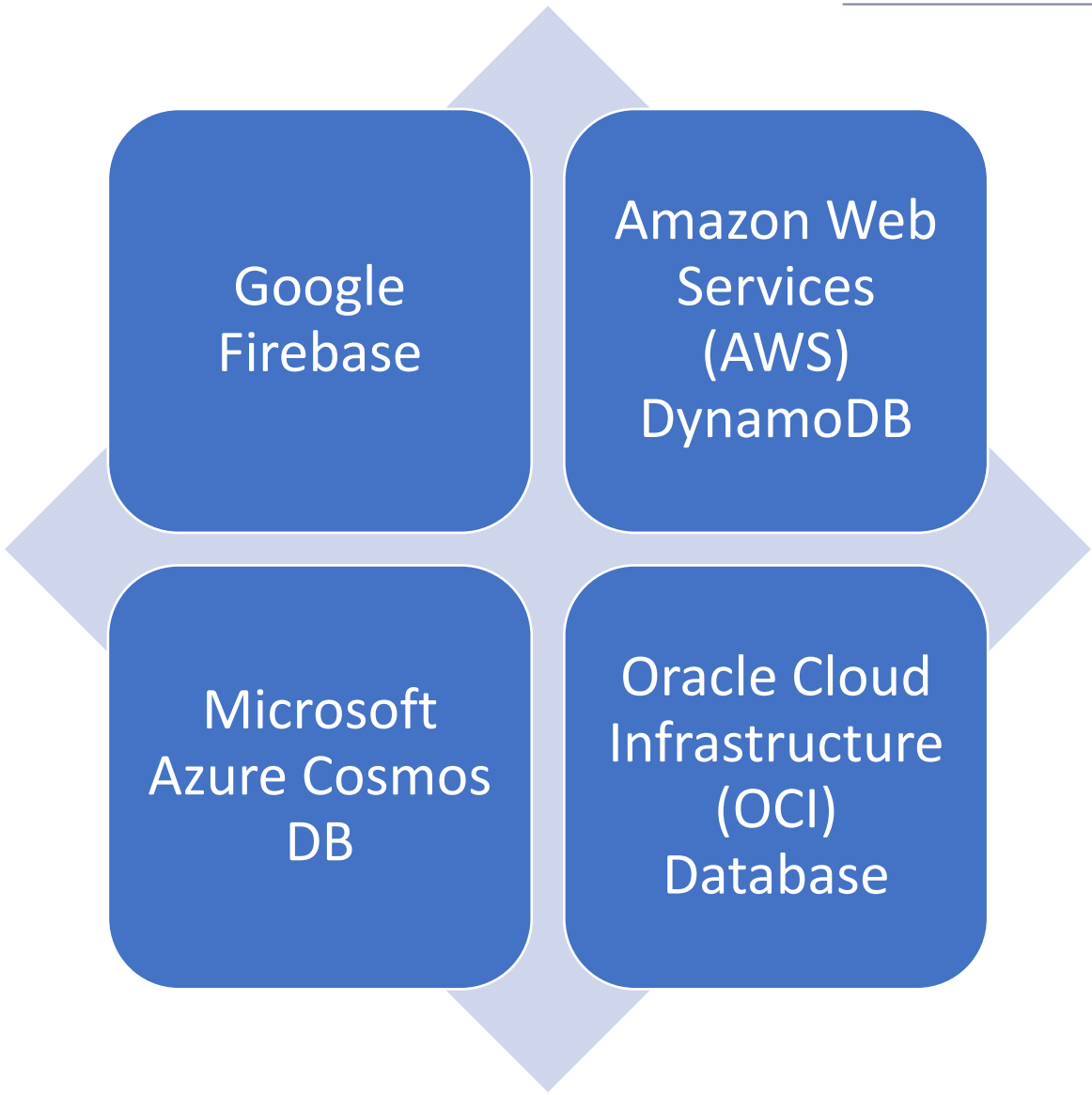




Cloud Storage

- Cloud storage refers to the storage of data on remote servers that can be accessed over the internet.
- There are several cloud storage providers available that offer APIs for Android developers to integrate cloud storage into their apps.
- Some popular cloud storage providers include Google Drive, Dropbox, Amazon S3, and Microsoft OneDrive.
- Cloud database refers to a database service that is hosted in the cloud and can be accessed over the internet.

Cloud Database Options



Firebase

authentication with the app

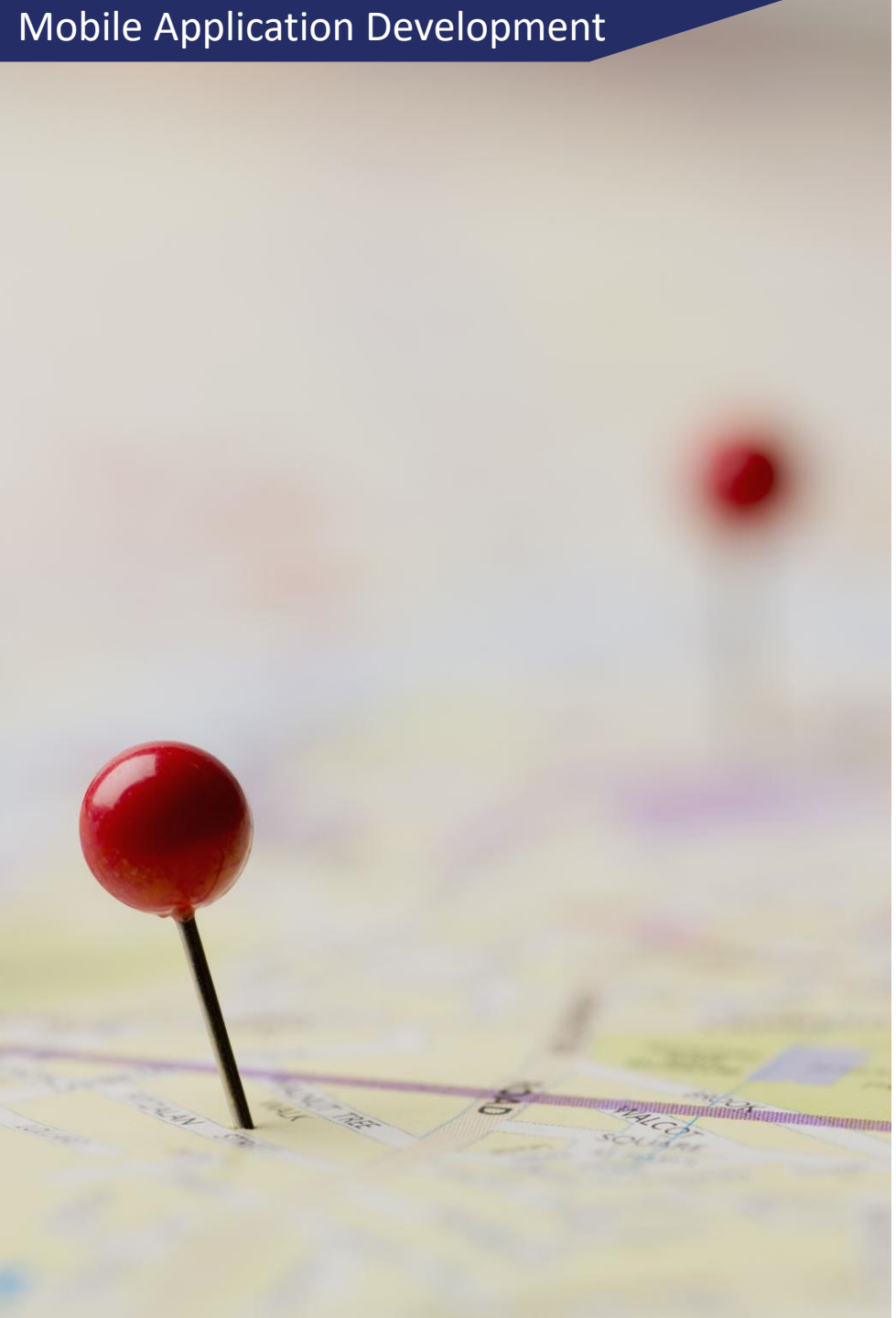
Send notifications connecting app and database

- Firebase is a comprehensive mobile development platform that provides a suite of tools and services for Android developers to build high-quality apps.
- Firebase offers a real-time cloud database, user authentication, cloud messaging, analytics, crash reporting, and more.
- Firebase Realtime Database is a NoSQL cloud database that stores data in JSON format and synchronizes data between clients in real-time. It provides offline data persistence and scales to handle millions of concurrent users.
- Firebase Authentication provides a simple way for users to authenticate with your app using email/password, social media accounts, or phone numbers. It also integrates with Google Sign-In and offers robust security features.
- Firebase Cloud Messaging (FCM) allows you to send push notifications to your app users, even when the app is not running in the foreground. It supports both Android and iOS devices and provides advanced targeting and customization options.

Firebase Continue.

get analytics about app

- Firebase Analytics helps you understand how users engage with your app and provides insights into user behavior, demographics, and app performance. It also integrates with Google Analytics and AdMob.
- Firebase Crash Reporting provides real-time reports on app crashes and exceptions, including detailed stack traces and device information. It also integrates with Google Play Console for easy app distribution and management.
- Firebase Performance Monitoring helps you optimize your app's performance by identifying slow network requests, rendering issues, and other performance bottlenecks. It provides detailed traces and metrics for each network request and UI interaction.
- Firebase provides a powerful and easy-to-use platform for Android developers to build and scale their apps. With Firebase, you can focus on building great user experiences and leave the infrastructure and backend management to Google.



SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Thank you!

References

- https://developer.android.com/training/data-storage/room
- https://developer.android.com/training/data-storage/shared-preferences
- https://firebase.google.com/docs