WHEN DEVELOPING AN ANDROID APPLICATION BASED ON A DATABASE SOFTWARE TOOLS USED.

Rayimov Khudaynazar
Master's student of Urganch State University
khudaynazar@urdu.uz

Abstract: *This article is a database based Android application instructions of the software tools used in the development and from them provides information about specific features of use. article,*

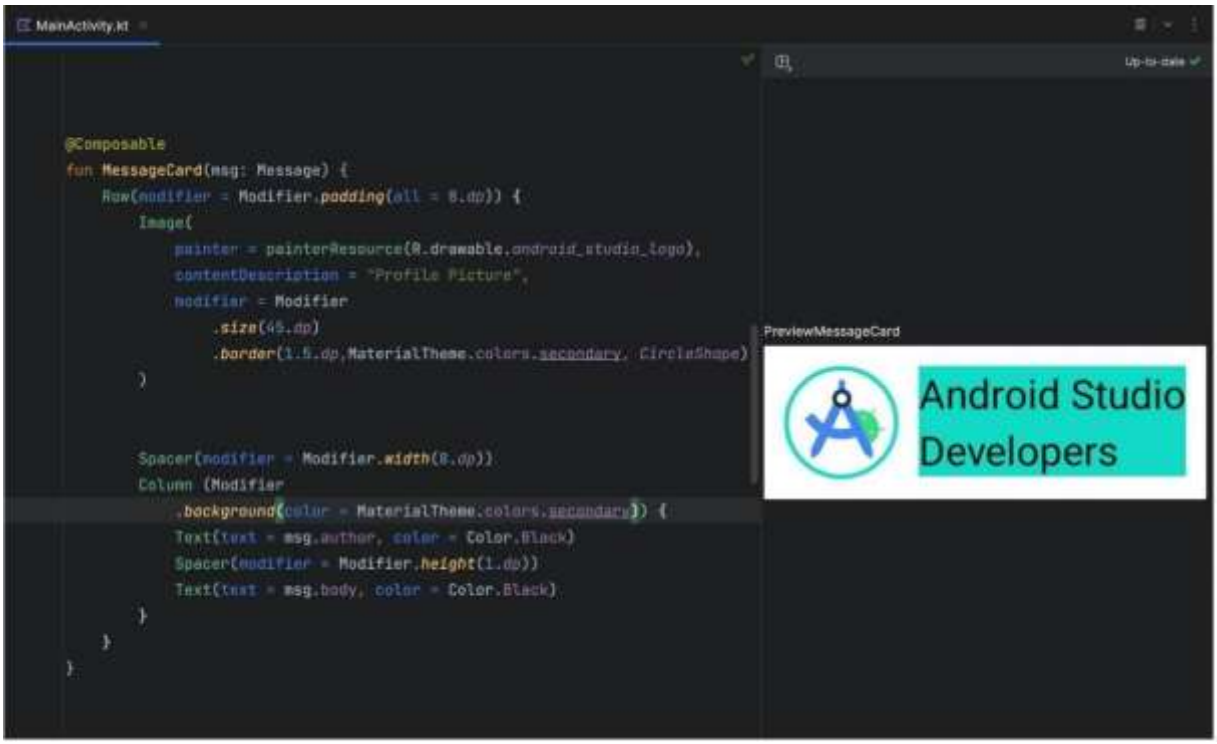
Describes software tools such as Android Studio, SQLite, and Retrofit, and how they are used in android apps, their benefits, provides information about its features and importance.

Keywords: *Android studio, Freebase, SQLte, Room Persistence Library, Realm, Retrofit, Software.*

Enter. Database system, data acquisition, storage, editing, and the structure that provides access to them. This system is programming is the main field and data processing, their input, read and modify, and make the modified data more accessible used for The information stored in the database is different can be, for example, text, numbers, images, files, dates, etc. How to access data through database queries, what data to get from them, what data to change, and you can specify what data to delete. Very much in the field of programming is important, and many popular databases (e.g.

MySQL, PostgreSQL, MongoDB, etc.) are widely used in the world. The Android application is also designed to store a database in itself.

Database driven for Android application development software tools are required. Backend and frontend divisions are usually these combined using tools. Database based Android the following basic programming tools are used to create applications:



Android Studio: Android applications provided by Google is a read-only database software tool for development.

This is the basic framework in Android application development, UI (user interface) components, creating projects, Android application providing important functions such as testing and commissioning etc is enough. Below we will see the code used in the android studio environment.

val databaseName = "MyDatabase" val version = 1

val context : Context = applicationContext

val database = Room.databaseBuilder(context, AppDatabase::class.java, databaseName)

.build()

// Using Room for example @Entity

data class User(@PrimaryKey val uid: Int,

@ColumnInfo(name = "first_name") val firstName: String?, @ColumnInfo(name

= "last_name") val lastName: String?

)

@Dao

*interface UserDao { @Query("SELECT * FROM user") fun getAll(): List<User>*

*@Query("SELECT * FROM user WHERE uid IN (:userIds)") fun loadAllByIds(userIds: IntArray): List<User>*

*@Query("SELECT * FROM user WHERE first_name LIKE :first AND " +*

"last_name LIKE :last LIMIT 1")

fun findByName(first: String, last: String): User @Insert fun insertAll(vararg users: User) @Delete

fun delete(user: User)

}

@Database(entities = [User::class], version = version) abstract class AppDatabase : RoomDatabase() { abstract fun userDao(): UserDao

}

This example uses the Room Persistence Library to create a database create, define, add, delete user tables and models and DAO (Data Access Object) to access data shows the interface. This code is a database based Android application Here's an example of a simple software tool used in the creation process shows.

Firebase: This is a platform offered by Google that user integration with social networks, authentication and unified offers many services such as database. Firebase is analytics server that can be used for making, storing and other purposes tool on the side.

Data logging, authentication, push on the Firebase platform send messages, save files and use other features the code for is written in the following form. This example is written in Kotlin:

import com.google.firebase.database.FirebaseDatabase import com.google.firebase.auth.FirebaseAuth

import com.google.firebase.messaging.FirebaseMessaging import com.google.firebase.storage.FirebaseStorage

val database = FirebaseDatabase.getInstance() val myRef = database.getReference("message") val auth = FirebaseAuth.getInstance()

val user = auth.currentUser

val messaging = FirebaseMessaging.getInstance() val storage = FirebaseStorage.getInstance()

val storageRef = storage.reference

val file = Uri.fromFile(File("path/ to/ file"))

val riversRef = storageRef.child("images/ ${file.lastPathSegment}") val uploadTask = riversRef.putFile(file) uploadTask.addOnSuccessListener {

}.addOnFailureListener {

}

In this code, writing and reading data through Firebase, perform user authentication, send and receive push messages To do this, it is possible to see views about file storage processes. Code to run Firebase configuration implementation and Firebase the SDK will need to be added.

SQLite: Android applications via embedded network or online services If not, it is recommended to use a local database. You local data using SQLite, the Android application management system you can create, manage and access databases.

Creating a table and writing data in a SQLite database and shows the reading.

import android.content.ContentValues import android.content.Context

import android.database.sqlite.SQLiteDatabase import android.database.sqlite.SQLiteOpenHelper class DatabaseHelper(context: Context) :

SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) { companion object {

private const val DATABASE_VERSION = 1

private const val DATABASE_NAME = "MyDatabase" private const val TABLE_NAME = "users"

private const val COLUMN_ID = "id"

private const val COLUMN_NAME = "name" private const val COLUMN_EMAIL = "email"

}

override fun onCreate(db: SQLiteDatabase) {

```
val createTableQuery = ("CREATE TABLE " " ("+ TABLE_NAME +
+ COLUMN_ID +" INTEGER PRIMARY KEY AUTOINCREMENT, " COLUMN_NAME +
+ COLUMN_EMAIL +
db.execSQL(createTableQuery)
}
" TEXT, "
" TEXT)")
override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) { db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
onCreate(db)
}
fun addUser(name: String, email: String): Long { val db = this.writableDatabase
val contentValues = ContentValues() contentValues.put(COLUMN_NAME, name) contentValues.put(COLUMN_EMAIL, email)
return db.insert(TABLE_NAME, null, contentValues)
}
}
fun main() {
val dbHelper = DatabaseHelper(context) dbHelper.onCreate(dbHelper.writableDatabase)
val userId = dbHelper.addUser("John Doe", "john@example.com") println("User ID: $userId")
}
```

This code creates a variable named `DatabaseHelper`, `DatabaseHelper` Inherits from `SQLiteOpenHelper`. Using `DatabaseHelper` opening a database, creating a table and adding data and reading operations can be performed. In the

`main` function A `DatabaseHelper` object is created and the table is created. In the next step to add new users via the `addUser` function is written to the database.

Room Persistence Library: This is a popular official with Android data storage and database manipulation library. Room easily with SQLite databases because it integrates with an object management system (ORM) can be used to connect to a database.

Realm: This is a platform for processing databases and Android used to store data for developing applications. Realm native databases as an

object management system (ORM) and facilitates simple placement of functions.

Retrofit: Retrofit addresses related APIs for server-connected applications helps to do. It is easy to send, receive and used to return and access RESTful services.

Database driven Android application development widely uses software tools. Each of them has its own characteristics have program code:

1. **Add the Retrofit library:**

implementation 'com.squareup.retrofit2:retrofit:2.9.0'

implementation 'com.squareup.retrofit2:converter-gson:2.9.0' // Gson converter for JSON parser

2. **Creating an interface for contacting a RESTful service:**

import retrofit2.Call import retrofit2.http.GET interface ApiService { @GET("posts")

fun getPosts(): Call<List<Post>>

}

3. **Data models:**

import com.google.gson.annotations.SerializedName data class Post(

val id: Int,

val title: String,

@SerializedName("body") val content: String

)

4. **Creating a Retrofit client and using the service:**

import retrofit2.Call import retrofit2.Callback import retrofit2.Response import retrofit2.Retrofit

import retrofit2.converter.gson.GsonConverterFactory fun main() {

val retrofit = Retrofit.Builder()

.baseUrl("https:// jsonplaceholder.typicode.com/")

.addConverterFactory(GsonConverterFactory.create())

.build()

val service = retrofit.create(ApiService::class.java) val call = service.getPosts()

```
call.enqueue(object : Callback<List<Post>> {
override fun onResponse(call: Call<List<Post>>, response: Response<List<Post>>) { if (response.isSuccessful) {
val posts = response.body() posts?.forEach { post ->
println("Title: ${post.title}, Content: ${post.content}")

}
}


}
override fun onFailure(call: Call<List<Post>>, t: Throwable) { println("Error: ${t.message}")
}
})
}
```

This code invokes a RESTful service using the Retrofit library and outputs the read data to the console. To run the code Android suspended on an emulator or host device in an Android app environment application should be launched.

REFERENCES.

I. Karajanova AA et al. ANDROID OPERATING SYSTEM

OPPORTUNITIES //World scientific research journal. - 2022. - T. 8. – no. 1. –S. 117-122.

II. Normatov R., Abduvakhabov A. ANDROID STUDIO

CREATE AN ANDROID APPLICATION WITH THE HELP //Evraziysky zurnal akademicheskikh issledovany. - 2022. - T. 2. – no. 12. - S. 1270-1276.

III. Darmayadi A., Izmazatnika YM Android Based Information

System Design //IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2020. – T. 879. - no. 1. – S. 012096.

IV. Khawas C., Shah P. Application of firebase in android app

development-a study //International Journal of Computer Applications. - 2018.

- T. 179. - no. 46. - S. 49-53.