Working with Databases

So far you are familiar with SharedPreferences, a key value store that we can use to persist simple pieces of data. For more complex structured data we will use a relational database. Modern android apps need to continue working even without an internet connection. Having a local database in our app helps us to persist data that enables our application to work without a connection. It also saves users' mobile data because it eliminates the need to fetch already persisted data every time we need to use it.

Every android device has inbuilt support for SQLite, a lightweight relational database with a very small memory footprint. We will use Room library as an ORM to provide abstraction over SQLite. In our existing MyContacts app we shall:

1. Add dependencies

Add the following dependencies in your app level build.gradle file

```
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.5.0'
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.0-
alpha03"
implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.0-alpha03"
implementation "androidx.activity:activity-ktx:1.3.1"
implementation "androidx.room:room-runtime:2.3.0"
kapt "androidx.room:room-compiler:2.3.0"
```

Also be sure to add the following to the same file under plugins. Afterwards sync your project.

```
id 'kotlin-kapt'
```

2. Create the Entity

Update your existing contact data class to look like this. This file should be in the `model` package

```
@Entity(tableName = "Contacts")
data class Contact(
    @PrimaryKey(autoGenerate = true) @NonNull var contactId: Int,
    var name: String,
    var phoneNumber: String,
    var email: String,
    var imageUrl: String?
)
```

The `@Entity` annotation creates a table called Contacts. The `@PrimaryKey(autoGenerate=true)` annotation generates a unique primary key for each inserted record.

3. Create a DAO.

A DAO is a database access object that allows us to easily access the db table. Each entity has its own Dao that looks like this. Create it inside the database package

```
@Dao
interface ContactDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertContact(contact: Contact)

    @Query("SELECT * FROM Contacts")
    fun getAllContacts(): LiveData<List<Contact>>

    @Query("SELECT * FROM Contacts WHERE contactId = :contactId")
    fun getContactById(contactId: Int): LiveData<Contact>
}
```

It is an interface that describes methods which we will invoke to access the data via different queries. It has both inbuilt functions and other custom queries which we define depending on the needs of our app.

4. Create a Database

In the database package add this class

```
@Database(entities = arrayOf(Contact::class), version = 1)
abstract class ContactsDatabase : RoomDatabase() {
    abstract fun contactDao(): ContactDao

    companion object {
        private var database: ContactsDatabase? = null

        fun getDatabase(context: Context): ContactsDatabase {
            if (database == null) {
                database = Room.databaseBuilder(context,
ContactsDatabase::class.java, "contactsDb")
                    .fallbackToDestructiveMigration().build()
            }
            return database as ContactsDatabase
        }
    }
}
```

The companion object here ensures that we only have one instance of the database at a time.
This class will require access to a global application context which we shall provide using the following class that we will create at the root of our main package.

```
class ContactsApp : Application() {
    companion object {
        lateinit var appContext: Context
    }

    override fun onCreate() {
        super.onCreate()
        appContext = applicationContext
    }
}
```

We will then add a reference to this class in our Manifest inside the Application tag like so:

```
<application
  ...
  android:name=".ContactsApp"
  .../>
```

5. Create a repository

In your repository package add this class

```
class ContactsRepository {
    val database = ContactsDatabase.getDatabase(ContactsApp.appContext)

    suspend fun saveContact(contact: Contact) {
        withContext(Dispatchers.IO) {
            database.contactDao().insertContact(contact)
        }
    }

    fun fetchContacts(): LiveData<List<Contact>> {
        return database.contactDao().getAllContacts()
    }

    fun getContactById(contactId: Int): LiveData<Contact> {
        return database.contactDao().getContactById(contactId)
    }
}
```

The repository has an instance of our database that invokes the methods on the dao to query and insert data to our database as required.

6. Create a viewmodel

Add a ContactsViewModel in your ViewModel package like so.

```
class ContactsViewModel : ViewModel() {
    var contactsRepository = ContactsRepository()
    lateinit var contactLiveData: LiveData<Contact>

    fun getContactById(contactId: Int) {
        contactLiveData = contactsRepository.getContactById(contactId)
    }

    fun saveContact(contact: Contact) {
        viewModelScope.launch {
            contactsRepository.saveContact(contact)
        }
    }
}
```

7. Trigger the query/insert from the UI

Finally from your activity you can trigger a query like so

```
class ViewContactActivity : AppCompatActivity() {
    val contactViewModel: ContactsViewModel by viewModels()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_view_contact)
        contactViewModel.getContactById(1)
    }

    override fun onResume() {
        super.onResume()
        contactViewModel.contactLiveData.observe(this, {contact->
            if (contact!=null){
                Toast.makeText(this, contact.name, Toast.LENGTH_LONG).
show()
            }
        })
    }
}
```

For an insert you could:

```kotlin
class AddContactActivity : AppCompatActivity() {
    val contactViewModel: ContactsViewModel by viewModels()
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_view_contact)
    }

    override fun onResume() {
        super.onResume()
        binding.btnSave.setOnClickListener{
          saveContact()
        }
    }

    fun saveContact(){
        var name = binding.etName.text.toString()
        var phone = binding.etName.text.toString()
        var email = binding.etName.text.toString()
        var contact = Contact(0, name, phone, email, "")
        contactViewModel.saveContact(contact)
    }
}
```