

android crud tutorial with example application

Published: September 19, 2026 | Index ID: #-

Android CRUD Tutorial with Example Application

In modern mobile development, the ability to create, read, update, and delete (CRUD) data reliably is essential. Whether you're building a simple note taking app or a complex inventory management system, a well structured CRUD implementation keeps your data consistent and your user experience smooth. This comprehensive Android CRUD tutorial walks you through every step of building a fully functional example application—from setting up the project to handling concurrency, testing, and best practices. By the end, you'll have a solid foundation for integrating CRUD operations into any Android project.

What is CRUD?

CRUD stands for **Create, Read, Update, Delete**, the four basic functions of persistent storage. In the context of Android, these operations typically interact with a local database (SQLite, Room) or a remote server via RESTful APIs. Understanding CRUD is the first step to building data centric applications that respond to user input and maintain state across sessions.

Create

Adding new records to your data store. In a note taking app, this would mean saving a new note.

Read

Retrieving existing records. This could involve displaying a list of notes in a RecyclerView.

Update

Modifying an existing record. For example, editing the title or content of a note.

Delete

Removing a record from the data store. This could be triggered by a swipe to delete gesture.

Why CRUD in Android Apps?

Android devices often need offline access to data. A local database ensures that users can create, read, update, and delete information even without a network connection. Additionally, a clean CRUD architecture separates concerns, making the codebase easier to maintain, test, and extend. By following a consistent pattern, you reduce bugs and improve performance.

Overview of Android Architecture Components

Android Jetpack provides a suite of libraries that simplify building robust applications. For CRUD, the most relevant components are:

- Room – An abstraction layer over SQLite that handles database access with compile time checks.
- ViewModel – Stores UI related data that survives configuration changes.
- LiveData – A lifecycle aware observable data holder that updates the UI automatically.
- Repository – Acts as a single source of truth, mediating between data sources and the rest of the app.

Using these components together results in a clean separation of concerns, testable code, and a responsive UI.

Setting Up the Project

Let's create a fresh Android Studio project that will serve as our example application. We'll use Kotlin for brevity and modern Android practices.

1. Open Android Studio. Choose New Project !' Empty Activity. Name it NoteApp and set the language to Kotlin.
2. Add dependencies. In app/build.gradle, include Room, Lifecycle, and Coroutines:

```
implementation "androidx.room:room-runtime:2.6.1"
```

```
kapt "androidx.room:room-compiler:2.6.1"
```

```
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.2"
```

```
implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.6.2"
```

```
implementation "org.jetbrains.kotlin:kotlinx-coroutines-android:1.7.3"
```

Sync the project after adding the dependencies.

Designing the Data Model

For this tutorial, we'll create a simple Note entity with an ID, title, and content. Room requires that each entity have a primary key.

1. Create a package data.model and add Note.kt:

```
import androidx.room.Entity
```

```
import androidx.room.PrimaryKey
```

```
@Entity(tableName = "notes")
```

```
data class Note(
```

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

```
    val title: String,
```

```
    val content: String
```

```
)
```

This simple model will be the foundation for all CRUD operations.

Creating the Room Database

Room needs a database class that extends RoomDatabase and provides abstract access to the DAO (Data Access Object).

1. Create a package data.database and add AppDatabase.kt:

```
import androidx.room.Database
```

```
import androidx.room.RoomDatabase
```

```

import androidx.room.Room

import android.content.Context

@Database(entities = [Note::class], version = 1, exportSchema = false)

abstract class AppDatabase : RoomDatabase() {

    abstract fun noteDao(): NoteDao

}

```

We'll also create a singleton instance so the database is only created once:

```

object DatabaseProvider {

    private var INSTANCE: AppDatabase? = null

    fun getDatabase(context: Context): AppDatabase {

        return INSTANCE ?: synchronized(this) {

            val instance = Room.databaseBuilder(

                context.applicationContext,

                AppDatabase::class.java,

                "note_database"

            ).build()

            INSTANCE = instance

            instance

        }

    }

}

```

Defining the DAO

A DAO contains methods annotated with SQL queries. These methods are the bridge between the database and the rest of the app.

1. Create a package data.dao and add NoteDao.kt:

```

import androidx.room.Dao

import androidx.room.Insert

import androidx.room.Update

import androidx.room.Delete

import androidx.room.Query

import kotlinx.coroutines.flow.Flow

```

@Dao

```
interface NoteDao {  
  
    @Insert  
    suspend fun insert(note: Note): Long  
  
    @Update  
    suspend fun update(note: Note)  
  
    @Delete  
    suspend fun delete(note: Note)  
  
    @Query("SELECT * FROM notes ORDER BY id DESC")  
    fun getAllNotes(): Flow<List<Note>>  
  
    @Query("SELECT * FROM notes WHERE id = :id LIMIT 1")  
    suspend fun getNoteById(id: Int): Note?  
}
```

Notice the use of Flow for the getAllNotes query. This allows the UI to observe changes automatically.

Implementing the Repository Pattern

The repository abstracts data operations, making it easier to swap data sources or add caching later.

1. Create a package data.repository and add NoteRepository.kt:

```
import kotlinx.coroutines.flow.Flow  
  
class NoteRepository(private val dao: NoteDao) {  
  
    fun getAllNotes(): Flow<List<Note>> = dao.getAllNotes()  
  
    suspend fun getNoteById(id: Int): Note? = dao.getNoteById(id)  
  
    suspend fun insert(note: Note): Long = dao.insert(note)  
  
    suspend fun update(note: Note) = dao.update(note)  
  
    suspend fun delete(note: Note) = dao.delete(note)  
}
```

Creating the ViewModel

The ViewModel holds UI related data and survives configuration changes. It also exposes LiveData that the UI observes.

1. Create a package ui.viewmodel and add NoteViewModel.kt:

```

import androidx.lifecycle.ViewModel

import androidx.lifecycle.viewModelScope

import androidx.lifecycle.LiveData

import androidx.lifecycle.asLiveData

import kotlinx.coroutines.launch

class NoteViewModel(private val repository: NoteRepository) : ViewModel() {

    val notes: LiveData< > = repository.getAllNotes().asLiveData()

    fun insert(note: Note) = viewModelScope.launch { repository.insert(note) }

    fun update(note: Note) = viewModelScope.launch { repository.update(note) }

    fun delete(note: Note) = viewModelScope.launch { repository.delete(note) }
}

```

To provide the ViewModel with a repository, use a ViewModelFactory:

1. Add NoteViewModelFactory.kt:

```

import androidx.lifecycle.ViewModel

import androidx.lifecycle.ViewModelProvider

class NoteViewModelFactory(private val repository: NoteRepository) : ViewModelProvider.Factory {

    override fun create(modelClass: Class): ViewModel {

        if (modelClass.isAssignableFrom(NoteViewModel::class.java)) {

            @Suppress("UNCHECKED_CAST")

```

Baca Juga (Artikel Terkait)

- [american government d and review answer key](http://adultvibex.com/american-government-d-and-review-answer-key.pdf)

URL: <http://adultvibex.com/american-government-d-and-review-answer-key.pdf>

- [american government chapter 5 section 2 answers](http://adultvibex.com/american-government-chapter-5-section-2-answers.pdf)

URL: <http://adultvibex.com/american-government-chapter-5-section-2-answers.pdf>

- [american government chapter 2 answers](http://adultvibex.com/american-government-chapter-2-answers.pdf)

URL: <http://adultvibex.com/american-government-chapter-2-answers.pdf>

- [american government chapter 11 section 1 the scope of congressional powers question](http://adultvibex.com/american-government-chapter-11-section-1-the-scope-of-congressional-powers-question.pdf)

URL: <http://adultvibex.com/american-government-chapter-11-section-1-the-scope-of-congressional-powers-question.pdf>

Tags: [#android](#) [#crud](#) [#tutorial](#) [#with](#) [#example](#) [#application](#)

