Project#1

Create an app that allows users to manage their daily tasks.

```
import android.os.Bundle

import androidx.activity.ComponentActivity

import androidx.activity.compose.setContent

import androidx.compose.foundation.layout.*

import androidx.compose.foundation.lazy.LazyColumn

import androidx.compose.foundation.lazy.items

import androidx.compose.material.*

import androidx.compose.runtime.*

import androidx.compose.ui.Modifier

import androidx.compose.ui.unit.dp

import androidx.lifecycle.LiveData

import androidx.lifecycle.ViewModel

import androidx.lifecycle.ViewModelProvider

import androidx.lifecycle.viewModelScope

import androidx.room.*

import kotlinx.coroutines.launch


@Entity(tableName = "tasks")

data class Task(

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

    val title: String,

    val desc: String,

    val due: String,
    val cat: String,

    val done: Boolean = false

)

@Dao

interface TaskDao {

    @Insert

 suspend fun insertTask(task: Task)

 @Update

 suspend fun updateTask(task: Task)

 @Delete

    suspend fun deleteTask(task: Task)


    @Query("SELECT * FROM tasks ORDER BY id ASC")

    fun getTasks(): LiveData<List<Task>>}


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

abstract class TaskDatabase : RoomDatabase() {

    abstract fun taskDao(): TaskDao

  companion object {

        @Volatile

        private var INSTANCE: TaskDatabase? = null


    fun getDatabase(context: android.content.Context): TaskDatabase {

      return INSTANCE ?: synchronized(this) {

         val instance = Room.databaseBuilder(
             context.applicationContext,
                TaskDatabase::class.java,
                "task_database"
              ).build()
              INSTANCE = instance
              instance
            } }}}

class TaskViewModel(application: android.app.Application) : ViewModel() {
   private val taskDao = TaskDatabase.getDatabase(application).taskDao()
   val tasks: LiveData<List<Task>> = taskDao.getTasks()

   fun addTask(task: Task) {
      viewModelScope.launch {
         taskDao.insertTask(task)
      }}
    fun updateTask(task: Task) {
      viewModelScope.launch {
         taskDao.updateTask(task)
      }  }

   fun deleteTask(task: Task) {
      viewModelScope.launch {
         taskDao.deleteTask(task)
      }}}
```

```
class MainActivity : ComponentActivity() {

   private val taskViewModel by lazy {

      ViewModelProvider(this, ViewModelProvider.AndroidViewModelFactory(application))

         .get(TaskViewModel::class.java)

   }

override fun onCreate(savedInstanceState: Bundle?) {

      super.onCreate(savedInstanceState)

      setContent {

         TaskApp(taskViewModel)

      }}}


@Composable

fun TaskApp(taskViewModel: TaskViewModel) {

   Scaffold(

      topBar = {

         TopAppBar(title = { Text("Tasks") })

      },

      content = {

         TaskScreen(taskViewModel)

      }

   )}

@Composable

fun TaskScreen(taskViewModel: TaskViewModel) {

   val tasks by taskViewModel.tasks.observeAsState(emptyList())
```

```
    var showDialog by remember { mutableStateOf(false) }


    if (showDialog) {
        TaskDialog(onDismiss = { showDialog = false }, onSave = { task ->
            taskViewModel.addTask(task)
            showDialog = false
        })
    }


    Column(modifier = Modifier.fillMaxSize().padding(16.dp)) {
        Button(onClick = { showDialog = true }) {
            Text("Add Task")
        }
        LazyColumn {
        items(tasks) { task ->
        TaskItem(task, taskViewModel)  }
        }  }}


@Composable
fun TaskItem(task: Task, taskViewModel: TaskViewModel) {
    var isDone by remember { mutableStateOf(task.done) }
    var showEditDialog by remember { mutableStateOf(false) }


    if (showEditDialog) {
        TaskDialog(
            onDismiss = { showEditDialog = false },
            onSave = { updatedTask ->
                taskViewModel.updateTask(updatedTask)
                showEditDialog = false
            },
            initialTask = task
        )
    }

    Card(
        modifier = Modifier.fillMaxWidth().padding(vertical = 4.dp),
        elevation = 4.dp,
        onClick = { showEditDialog = true }
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(task.title, style = MaterialTheme.typography.h6)
            Text("Due: ${task.due}")
            Text("Category: ${task.cat}")
            Checkbox(
                checked = isDone,
                onCheckedChange = {
                    isDone = it
                    taskViewModel.updateTask(task.copy(done = isDone))
                }
            )}}}
```

```
@Composable
fun TaskDialog(onDismiss: () -> Unit, onSave: (Task) -> Unit, initialTask: Task? = null) {
    var title by remember { mutableStateOf(initialTask?.title ?: "") }
    var desc by remember { mutableStateOf(initialTask?.desc ?: "") }
    var due by remember { mutableStateOf(initialTask?.due ?: "") }
    var cat by remember { mutableStateOf(initialTask?.cat ?: "") }

    AlertDialog(
        onDismissRequest = onDismiss,
        title = { Text("Task") },
        text = {
            Column {
                OutlinedTextField(
                    value = title,
                    onValueChange = { title = it },
                    label = { Text("Title") },
                    modifier = Modifier.fillMaxWidth().padding(8.dp)
                )
                OutlinedTextField(
                    value = desc,
                    onValueChange = { desc = it },
                    label = { Text("Description") },
                    modifier = Modifier.fillMaxWidth().padding(8.dp)
                )
        OutlinedTextField(
            value = due,
            onValueChange = { due = it },
            label = { Text("Due Date") },
            modifier = Modifier.fillMaxWidth().padding(8.dp)
        )
        OutlinedTextField(
            value = cat,
            onValueChange = { cat = it },
            label = { Text("Category") },
            modifier = Modifier.fillMaxWidth().padding(8.dp)
        )
    }
},
confirmButton = {
    Button(onClick = {
        val task = Task(
            id = initialTask?.id ?: 0,
            title = title,
            desc = desc,
            due = due,
            cat = cat,
            done = initialTask?.done ?: false
        )
        onSave(task)
```

```
            }) {
                Text("Save")
            }
        },
        dismissButton = {
            Button(onClick = onDismiss) {
                Text("Cancel")
    }}
    }}
```