Android Intern Task

1) Project structure (suggested)

```
/project-root
  /shared (KMM module)
    src/commonMain/kotlin
      models/Task.kt
      repository/TaskRepository.kt   // expect interface if implementing platform-backed storage
  /androidApp
    src/main/...
      AndroidManifest.xml
      MainActivity.kt
      ui/ (compose screens)
      util/ RecordingHelper.kt, StorageHelper.kt, CameraHelper.kt
    build.gradle.kts
  settings.gradle.kts
```

For speed you can keep everything inside the Android app and use plain Kotlin modules. If you want true KMM later, move models + repository interface into shared.

2) Shared data model (Kotlin)

Create Task.kt (shared):

```kotlin
import kotlinx.serialization.Serializable


@Serializable
data class SampleTask(
  val id: String,
  val task_type: String, // "text_reading" | "image_description" | "photo_capture"
  val text: String? = null,
  val image_url: String? = null,
  val image_path: String? = null,
  val audio_path: String? = null,
  val duration_sec: Int,
  val timestamp: String // ISO8601 string
```

)

We’ll persist an array of SampleTask objects to a JSON file (e.g., tasks.json) in app files dir.

3) Android: Gradle dependencies (app build.gradle.kts)

```
plugins {
  id("com.android.application")
  kotlin("android")
  kotlin("plugin.serialization") version "1.9.0" // or current
}

android {
  compileSdk = 34
  defaultConfig {
    applicationId = "com.example.sampletasks"
    minSdk = 24
    targetSdk = 34
  }
  buildFeatures {
    compose = true
  }
  composeOptions { kotlinCompilerExtensionVersion = "1.5.0" }
}

dependencies {
  implementation("androidx.core:core-ktx:1.10.1")
  implementation("androidx.activity:activity-compose:1.8.0")
  implementation("androidx.compose.ui:ui:1.5.0")
  implementation("androidx.compose.material:material:1.5.0")
  implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.6.1")
  implementation("org.jetbrains.kotlinx:kotlinx-serialization-json:1.6.0")
  implementation("io.coil-kt:coil-compose:2.4.0") // image loading
  implementation("androidx.navigation:navigation-compose:2.7.0")
}
```

Adjust versions to current in your environment.

4) AndroidManifest (permissions)

```
<manifest ...>
  <uses-permission android:name="android.permission.RECORD_AUDIO"/>
  <uses-permission android:name="android.permission.CAMERA"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
android:maxSdkVersion="28"/>
  <application
    android:requestLegacyExternalStorage="true" ...>
    <provider
      android:name="androidx.core.content.FileProvider"
      android:authorities="${applicationId}.fileprovider"
      android:exported="false"
      android:grantUriPermissions="true">
      <meta-data android:name="android.support.FILE_PROVIDER_PATHS"
                 android:resource="@xml/file_paths" />
    </provider>
  </application>
</manifest>
```

Add res/xml/file_paths.xml:

```
<paths>
  <external-files-path name="external_files" path="." />
  <files-path name="files" path="." />
</paths>
```

5) Navigation & MainActivity (Compose NavHost)

MainActivity.kt (abridged)

```
class MainActivity : ComponentActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContent {
      AppNavHost()
    }
  }
}
```

NavRoutes.kt

```
object Routes {
  const val START = "start"
  const val NOISE = "noise"
  const val SELECT = "select"
  const val TEXT_READING = "text_reading"
  const val IMAGE_DESC = "image_desc"
  const val PHOTO_CAPTURE = "photo_capture"
  const val HISTORY = "history"
}
```

AppNavHost():

```
@Composable
fun AppNavHost() {
  val nav = rememberNavController()
  NavHost(nav, startDestination = Routes.START) {
    composable(Routes.START) { StartScreen(onStart = { nav.navigate(Routes.NOISE) }) }
    composable(Routes.NOISE) { NoiseTestScreen(onPass = { nav.navigate(Routes.SELECT) }) }
    composable(Routes.SELECT) {
      TaskSelectionScreen(
        onText = { nav.navigate(Routes.TEXT_READING) },
        onImage = { nav.navigate(Routes.IMAGE_DESC) },
        onPhoto = { nav.navigate(Routes.PHOTO_CAPTURE) },
        onHistory = { nav.navigate(Routes.HISTORY) }
      )
    }
    composable(Routes.TEXT_READING) {
      TextReadingScreen(onSubmit = { /* save & popBack */ nav.popBackStack() })
    }
    composable(Routes.IMAGE_DESC) {
      ImageDescriptionScreen(onSubmit = { nav.popBackStack() })
    }
    composable(Routes.PHOTO_CAPTURE) {
      PhotoCaptureScreen(onSubmit = { nav.popBackStack() })
    }
    composable(Routes.HISTORY) {
      TaskHistoryScreen()
    }
  }
}
```

6) Screen implementations — key parts

Start Screen

```
@Composable
fun StartScreen(onStart: () -> Unit) {
  Column(Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.Center) {
    Text("Let’s start with a Sample Task for practice.", style = MaterialTheme.typography.h6)
    Spacer(Modifier.height(8.dp))
    Text("Pehele hum ek sample task karte hain.")
    Spacer(Modifier.height(24.dp))
    Button(onClick = onStart) { Text("Start Sample Task") }
  }
}
```

Noise Test Screen (simulate or use mic)

We'll simulate with a slider or small random generator; show pass/fail based on average dB.

```
@Composable
fun NoiseTestScreen(onPass: () -> Unit) {
  var db by remember { mutableStateOf(20f) }
  val status = when {
    db < 40f -> "Good to proceed"
    else -> "Please move to a quieter place"
```

```
 }
 Column(Modifier.fillMaxSize().padding(16.dp), verticalArrangement = Arrangement.Center) {
  Text("Noise test (0 - 60 dB)")
  Spacer(Modifier.height(12.dp))
  Slider(value = db, onValueChange = { db = it }, valueRange = 0f..60f)
  Text("${db.toInt()} dB")
  Spacer(Modifier.height(8.dp))
  Text(status)
  Spacer(Modifier.height(16.dp))
  Button(onClick = {
   if (db < 40f) onPass()
  }, enabled = db < 40f) {
   Text("Proceed")
  }
 }
}
```

Note: for real microphone sampling you must capture audio using AudioRecord and compute RMS -> dB. For prototype simulation is acceptable.

Task Selection Screen

```
@Composable
fun TaskSelectionScreen(onText: ()->Unit, onImage: ()->Unit, onPhoto: ()->Unit, onHistory: ()->Unit) {
 Column(Modifier.fillMaxSize().padding(16.dp)) {
  Text("Choose a task", style = MaterialTheme.typography.h6)
  Spacer(Modifier.height(12.dp))
  Button(onClick = onText) { Text("Text Reading Task") }
  Spacer(Modifier.height(8.dp))
  Button(onClick = onImage) { Text("Image Description Task") }
  Spacer(Modifier.height(8.dp))
  Button(onClick = onPhoto) { Text("Photo Capture Task") }
  Spacer(Modifier.height(20.dp))
  OutlinedButton(onClick = onHistory) { Text("Task History") }
 }
}
```

7) Recording UX: press & hold logic (Compose)

We’ll implement a composable HoldToRecordButton that uses pointer input to start/stop recording.

HoldToRecordButton.kt

```
@Composable
fun HoldToRecordButton(
  onStart: () -> Unit,
  onStop: (durationMs: Long) -> Unit,
  modifier: Modifier = Modifier
) {
  var isRecording by remember { mutableStateOf(false) }
  var startTime by remember { mutableStateOf(0L) }

  Box(modifier = modifier
    .size(80.dp)
    .pointerInput(Unit) {
      while (true) {
        val event = awaitPointerEventScope { awaitFirstDown() }
        // finger down
        isRecording = true
        startTime = System.currentTimeMillis()
        onStart()
        // wait for up
        awaitPointerEventScope {
          var up = false
          while (!up) {
            val ev = awaitPointerEvent()
            if (ev.changes.any { it.changedToUp() }) up = true
          }
        }
        val duration = System.currentTimeMillis() - startTime
        isRecording = false
        onStop(duration)
      }
    },
    contentAlignment = Alignment.Center
  ) {
    val label = if (isRecording) "Recording..." else "Hold to record"
    Surface(shape = CircleShape, elevation = 4.dp) {
      Box(Modifier.size(80.dp), contentAlignment = Alignment.Center) {
        Text(label, textAlign = TextAlign.Center, modifier = Modifier.padding(8.dp))
      }
    }
  }
}
```

This will call onStart() on press and onStop(durationMs) on release.

8) Recording helper (Android) — MediaRecorder approach

RecordingHelper.kt

```
class RecordingHelper(private val context: Context) {

  private var recorder: MediaRecorder? = null
  private var outputFile: File? = null

  fun startRecording(fileName: String = "audio_${System.currentTimeMillis()}.mp4") {
    val dir = context.getExternalFilesDir("recordings") ?: context.filesDir
    outputFile = File(dir, fileName)
    recorder = MediaRecorder().apply {
      setAudioSource(MediaRecorder.AudioSource.MIC)
      setOutputFormat(MediaRecorder.OutputFormat.MPEG_4)
      setAudioEncoder(MediaRecorder.AudioEncoder.AAC)
      setAudioSamplingRate(44100)
      setAudioEncodingBitRate(96000)
      setOutputFile(outputFile!!.absolutePath)
      prepare()
      start()
    }
  }

  fun stopRecording(): File? {
    return try {
      recorder?.apply {
        stop()
        reset()
        release()
      }
      recorder = null
      outputFile
    } catch (e: Exception) {
      recorder = null
      null
    }
  }
}
```

Important: Request runtime permission RECORD_AUDIO before starting. Wrap startRecording in if (permissionGranted).

9) Text Reading Screen (logic + validations)

Key behavior:

- Show description text (load from https://dummyjson.com/products or use retrofit; for prototype, embed sample text).
- Press & hold mic button to record; on release check duration (ms -> seconds).
- If duration < 10s or > 20s show inline error.
- After valid recording show playback (use ExoPlayer or MediaPlayer) and three checkboxes. Submit enabled only when all checked. On submit save a SampleTask.

TextReadingScreen.kt (simplified)

```
@Composable
fun TextReadingScreen(onSubmit: (SampleTask) -> Unit) {
  val context = LocalContext.current
  val recorder = remember { RecordingHelper(context) }
  var audioFile by remember { mutableStateOf<File?>(null) }
  var durationSec by remember { mutableStateOf(0) }
  var error by remember { mutableStateOf<String?>(null) }
  var checked1 by remember { mutableStateOf(false) }
  var checked2 by remember { mutableStateOf(false) }
  var checked3 by remember { mutableStateOf(false) }

  val sampleText = "Mega long lasting fragrance..."
  Column(modifier = Modifier.fillMaxSize().padding(16.dp)) {
    Text(sampleText)
    Spacer(Modifier.height(12.dp))
    HoldToRecordButton(
      onStart = {
        // runtime permission check required
        recorder.startRecording()
      },
      onStop = { durationMs ->
        val sec = (durationMs / 1000).toInt()
        durationSec = sec
        if (sec < 10) {
          error = "Recording too short (min 10 s)."
          // delete file if created
          audioFile = recorder.stopRecording()
          audioFile?.delete()
          audioFile = null
        } else if (sec > 20) {
          error = "Recording too long (max 20 s)."
          audioFile = recorder.stopRecording()
          audioFile?.delete()
          audioFile = null
        } else {
          error = null
          audioFile = recorder.stopRecording()
        }
      }
    )
    if (error != null) { Text(error!!, color = Color.Red) }
    Spacer(Modifier.height(12.dp))
    // playback: simple MediaPlayer
    if (audioFile != null) {
      Button(onClick = { /* play audioFile via MediaPlayer */ }) {
        Text("Play")
      }
    }
    Spacer(Modifier.height(12.dp))
    Row { Checkbox(checked1, onCheckedChange = { checked1 = it }); Text("No background noise")
}
    Row { Checkbox(checked2, onCheckedChange = { checked2 = it }); Text("No mistakes while
reading") }
    Row { Checkbox(checked3, onCheckedChange = { checked3 = it }); Text("Beech me koi galti nahi
hai") }
    Spacer(Modifier.height(12.dp))
    Row {
      OutlinedButton(onClick = { /* record again: reset state to allow re-record */ audioFile?.delete();
audioFile = null; durationSec = 0 }) { Text("Record again") }
      Spacer(Modifier.width(8.dp))
      Button(onClick = {
        val task = SampleTask(
          id = java.util.UUID.randomUUID().toString(),
          task_type = "text_reading",
          text = sampleText,
          audio_path = audioFile?.absolutePath,
          duration_sec = durationSec,
          timestamp = java.time.OffsetDateTime.now().toString()
        )
        onSubmit(task)
      }, enabled = checked1 && checked2 && checked3 && audioFile != null) {
        Text("Submit")
      }
    }
   }
}
```

Playback implementation: use MediaPlayer().apply { setDataSource(file.absolutePath); prepare(); start() }.

10) Image Description Screen

- Show an image (you can use Coil to load the URL given).
- Press & hold to record like text reading; same duration validation.
- After recording show playback and Submit.

Key difference: include image_url in the SampleTask.

11) Photo Capture Task

Use ActivityResultContracts.TakePicture() and a temporary file via FileProvider.

High-level:

1. Request camera permission.
2. Create file Uri, launch takePicture.
3. Preview the captured image using Image(Bitmap) or Coil with Uri.
4. Text field for typing description.
5. Optional press & hold mic for audio (duration validated).
6. Submit saves image_path + audio_path.

12) Storage helper — save tasks.json (simple)

StorageHelper.kt

```
class StorageHelper(private val context: Context) {
  private val json = Json { prettyPrint = true }
  private fun tasksFile(): File {
    return File(context.filesDir, "tasks.json")
  }

  fun loadTasks(): List<SampleTask> {
    val file = tasksFile()
    if (!file.exists()) return emptyList()
    return try {
      val text = file.readText()
      json.decodeFromString(ListSerializer(SampleTask.serializer()), text)
    } catch (e: Exception) {
      emptyList()
    }
  }

  fun saveTask(task: SampleTask) {
    val cur = loadTasks().toMutableList()
    cur.add(0, task)
    tasksFile().writeText(json.encodeToString(ListSerializer(SampleTask.serializer()), cur))
  }
}
```

When user submits a task call StorageHelper(context).saveTask(task).

13) Task History screen

Compute totals from loaded tasks, show LazyColumn items.

```
@Composable
fun TaskHistoryScreen() {
  val context = LocalContext.current
  val storage = remember { StorageHelper(context) }
  var tasks by remember { mutableStateOf(storage.loadTasks()) }
  val totalTasks = tasks.size
  val totalDuration = tasks.sumOf { it.duration_sec }


  Column(Modifier.fillMaxSize()) {
   Row(Modifier.fillMaxWidth().padding(16.dp), horizontalArrangement =
Arrangement.SpaceBetween) {
    Text("Total Tasks: $totalTasks")
    Text("Total Recording Duration: ${totalDuration}s")
   }
   LazyColumn {
    items(tasks) { t ->
     Card(Modifier.fillMaxWidth().padding(8.dp)) {
      Row(Modifier.padding(12.dp), verticalAlignment = Alignment.CenterVertically) {
       Column(Modifier.weight(1f)) {
        Text("ID: ${t.id.take(8)}")
        Text("Type: ${t.task_type}")
        Text("Duration: ${t.duration_sec}s • ${t.timestamp}")
        when (t.task_type) {
         "text_reading" -> Text(t.text ?: "")
         "image_description","photo_capture" -> {
          val thumb = t.image_url ?: t.image_path
          if (!thumb.isNullOrEmpty()) {
           Image(painter = rememberAsyncImagePainter(thumb), contentDescription = null,
modifier = Modifier.size(64.dp))
          }
         }
        }
       }
      }
     }
    }
   }
  }
```

}

14) Edge cases & implementation notes

- Permissions: Request RECORD_AUDIO and CAMERA at runtime. For simplicity, ask before pressing record / capture image.
- Threading: File I/O should be done off the UI thread (use LaunchedEffect + withContext(Dispatchers.IO) or coroutines).
- MediaPlayer: release after playback to avoid leaks.
- Duration validation: use integer seconds, as specified.
- File lifecycle: store audio & images under app-specific directories (context.getExternalFilesDir("recordings") or filesDir/photos) so you can produce the APK without external storage permissions (app-only storage works).
- KMM: If you want the models and storage interface in shared, define expect class PlatformStorage and implement actual in Android. For prototype, Android-only is quickest.

15) Build & publish (quick steps)

1. In Android Studio choose Build > Build Bundle(s) / APK(s) > Build APK(s).
2. Find generated APK at app/build/outputs/apk/debug/app-debug.apk.
3. Create a GitHub repo:
    - git init, add files, commit.
    - Push to GitHub (create remote).
4. Upload the APK to GitHub Releases or attach to the repo (or use GitHub Actions to build).

Include README with:

- How to run
- Permissions required
- Sample screenshots