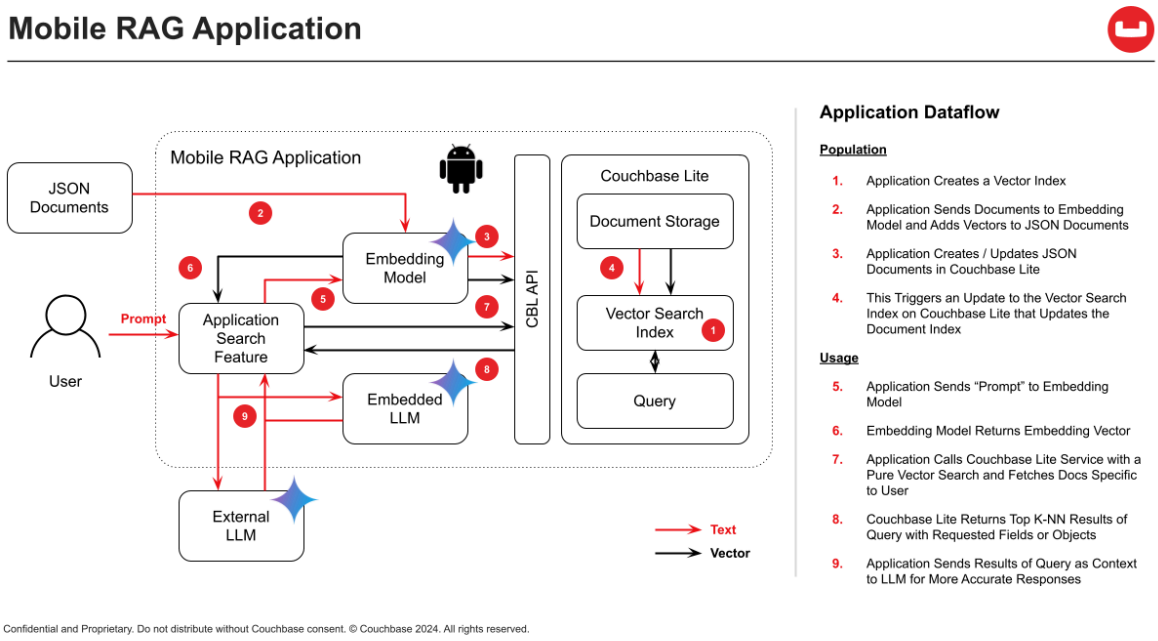
Mobile LLM Lab

Mobile LLM Lab

October 2025

1. Introduction



This guide will walk you through creating an Android mobile RAG application using Kotlin, Couchbase Lite, Gemini and LangChain components.

2. Install Android Studio

- Link to Android Studio - https://developer.android.com/studio
- Follow the installation instructions for your system.

Mobile LLM Lab

October 2025

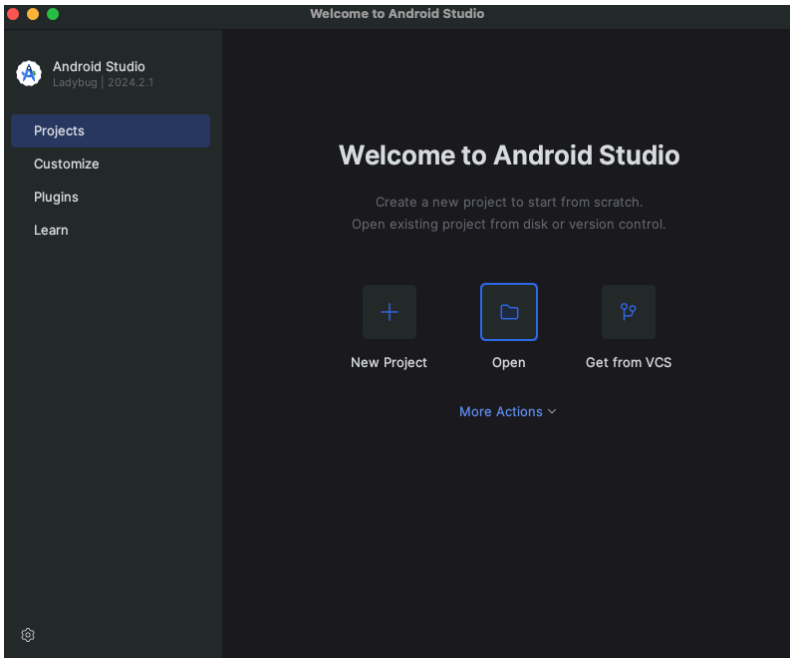
3. Clone the Existing Repo

Steps

- Open a command prompt and navigate to the directory where you will clone the existing repo.
- Use the following command to clone the existing Git repo.

```
git clone https://github.com/shivay-couchbase/couchbase-lite-workshop.git
```

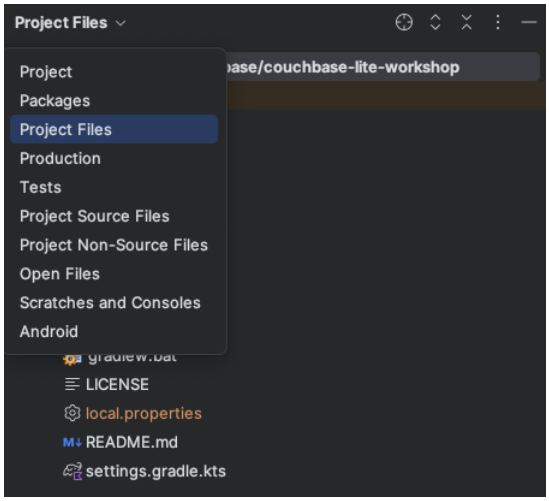
- Open Android Studio and select "Open" Navigate to the cloned repository and select the project folder.



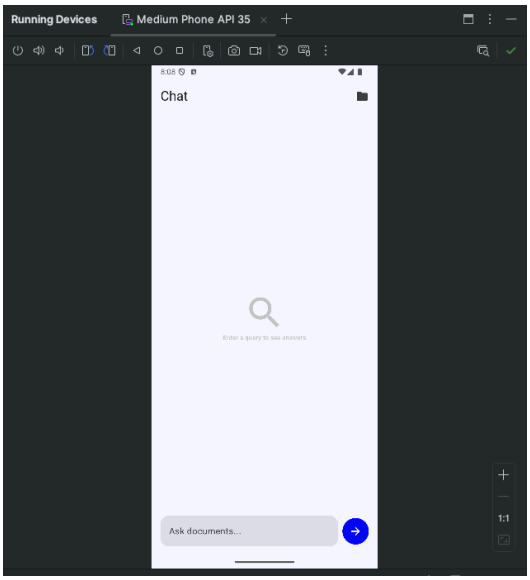
- Once the project is opened, Android Studio will automatically start syncing and building the project. Wait for this process to complete.
- Change the Directory View Panel from “Android” to “Project Files” from the top of the Directory View Panel dropdown.

Mobile LLM Lab

October 2025



- Click the green "Run" button in the top menu bar or use the keyboard shortcut (Shift + F10) to run the application. You should see your emulator panel open up and the application will load after a minute or so.



Mobile LLM Lab

October 2025

Component Mocking

The current implementation uses mock data and placeholder implementations for database operations. In the following sections, we'll replace these with actual Couchbase Lite operations and integrate the Gemini LLM for the RAG application.

Code Structure

The application follows a typical Android MVVM (Model-View-ViewModel) architecture. Here's an overview of the main components.

MainActivity (app/src/main/java/com/ml/couchbase/docqa/MainActivity.kt)

This is the entry point of the application. It sets up the navigation and initializes the database.

```
package com.ml.couchbase.docqa

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.ml.couchbase.docqa.ui.screens.ChatScreen
import com.ml.couchbase.docqa.ui.screens.DocsScreen
import dagger.hilt.android.AndroidEntryPoint

@AndroidEntryPoint
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
```

Mobile LLM Lab

October 2025

```
            val navHostController = rememberNavController()
            NavHost(
                navController = navHostController,
                startDestination = "chat",
                enterTransition = { fadeIn() },
                exitTransition = { fadeOut() }
            ) {
                composable("docs") { DocsScreen(onBackClick = {
navHostController.navigateUp() }) }
                composable("chat") {
                    ChatScreen(onOpenDocsClick = {
navHostController.navigate("docs") })
                }
            }
        }
    }
}
```

Data Models (app/src/main/java/com/ml/couchbase/docqa/data/DataModels.kt)

This file contains the data classes for the Couchbase Lite database used throughout the application.

```
package com.ml.couchbase.docqa.data

data class Chunk(
    var chunkId: Long = 0,
    var docId: String = 0.toString(),
    var docFileName: String = "",
    var chunkData: String = "",
    var chunkEmbedding: FloatArray = floatArrayOf()
)

data class Document(
    var docId: Long = 0,
```


Mobile LLM Lab

October 2025

```
    var docText: String = "",
    var docFileName: String = "",
    var docAddedTime: Long = 0,
)

data class RetrievedContext(val fileName: String, val context: String)

data class QueryResult(val response: String, val context:
List<RetrievedContext>)
```

DatabaseManager (app/src/main/java/com/ml/couchbase/docqa/data/DatabaseManager.kt)

This object manages the database initialization and provides access to the database instance.

Explanation

This file contains a mock implementation of the database manager. It includes,

- DatabaseManager object with placeholder initialization and mock database creation.
- MockDatabase class with placeholder implementations for database operations.
- Mock classes for Document, Array, Query, ResultSet, and Result.

```
package com.ml.couchbase.docqa.data

import android.content.Context
import android.util.Log

object DatabaseManager {
    private const val TAG = "DatabaseManager"

    fun init(context: Context, dbName: String = "myDatabase") {
        Log.i(TAG, "Placeholder: Database '$dbName' initialized
successfully")
```


Mobile LLM Lab

October 2025

```
    }

    fun getDatabase(): MockDatabase = MockDatabase()

    fun isVectorSearchEnabled(): Boolean = true
}

class MockDatabase {
    fun createIndex(name: String, config: Any) {
        // Placeholder implementation
    }

    fun save(document: MockDocument) {
        // Placeholder implementation
    }

    fun createQuery(sql: String): MockQuery = MockQuery()

    fun getDocument(id: String): MockDocument? = MockDocument()

    fun delete(document: MockDocument) {
        // Placeholder implementation
    }
}

class MockDocument {
    fun setString(key: String, value: String) {}
    fun setLong(key: String, value: Long) {}
    fun setArray(key: String, value: MockArray) {}
    val id: String = "mock_id"
}

class MockArray {
    fun addFloat(value: Float) {}
}

class MockQuery {
```

Mobile LLM Lab

October 2025

```
    fun execute(): MockResultSet = MockResultSet()
}

class MockResultSet : Iterable<MockResult> {
    override fun iterator(): Iterator<MockResult> =
listOf<MockResult>().iterator()
}

class MockResult {
    fun getString(key: String): String? = ""
    fun getArray(key: String): MockArray? = MockArray()
    fun getFloat(key: String): Float = 0f
    fun getInt(key: String): Int = 0
}
```

DocumentsDB (app/src/main/java/com/ml/couchbase/docqa/data/DocumentsDB.kt)

This class handles CRUD operations for documents in the database.

Explanation

This file contains a mock implementation of document operations, including,

- Placeholder methods for adding, removing, and retrieving documents.
- A mock implementation of getAllDocuments that returns dummy data.

```
package com.ml.couchbase.docqa.data

import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow

class DocumentsDB {
    private val database: MockDatabase = DatabaseManager.getDatabase()

    fun addDocument(document: Document): String {
```


Mobile LLM Lab

October 2025

```
        // Placeholder implementation
        return "mock_doc_id"
    }

    fun removeDocument(docId: String) {
        // Placeholder implementation
    }

    fun getAllDocuments(): Flow<List<Document>> = flow {
        // Placeholder implementation
        emit(List(5) { index -> Document(docId = index.toLong(),
docFileName = "mock_doc_$index.pdf") })
    }

    fun getDocsCount(): Long = 5L
}
```

ChunksDB (app/src/main/java/com/ml/couchbase/docqa/data/ChunksDB.kt)

This class manages operations related to document chunks, including vector search functionality.

Explanation

This file contains a mock implementation of chunk operations, including,

- Placeholder methods for adding, retrieving similar chunks, and removing chunks.

```
package com.ml.couchbase.docqa.data

class ChunksDB {
    private val database: MockDatabase = DatabaseManager.getDatabase()

    fun addChunk(chunk: Chunk) {
```


Mobile LLM Lab

October 2025

```
        // Placeholder implementation
    }

    fun getSimilarChunks(queryEmbedding: FloatArray, n: Int = 5):
List<Pair<Float, Chunk>> {
        // Placeholder implementation
        return List(n) { index ->
            Pair(0.5f, Chunk(chunkId = index.toLong(), docFileName =
"mock_doc.pdf", chunkData = "Mock chunk data"))
        }
    }

    fun removeChunks(docId: Long) {
        // Placeholder implementation
    }
}
```

UI Screens

These files contain the UI components for the chat interface and document management.

- ChatScreen (app/src/main/java/com/ml/couchbase/docqa/ui/screens/ChatScreen.kt)
- DocsScreen (app/src/main/java/com/ml/couchbase/docqa/ui/screens/DocsScreen.kt)

ViewModels

These ViewModels handle the business logic and data management for the UI components.

- ChatViewModel (app/src/main/java/com/ml/couchbase/docqa/ui/viewmodels/ChatViewModel.kt)
- DocsViewModel (app/src/main/java/com/ml/couchbase/docqa/ui/viewmodels/DocsViewModel.kt)

Mobile LLM Lab

October 2025

4. Initialize the Couchbase Lite Database

Steps

These steps will replace the mock implementation with the actual Couchbase Lite implementation for the DatabaseManager. The next steps would involve updating the ChunksDB and DocumentsDB classes to use the actual Couchbase Lite operations instead of the placeholder implementations.

- Highlight and replace the all content of DatabaseManager.kt with the following code.

```
package com.ml.couchbase.docqa.data

import android.content.Context
import android.util.Log
import com.couchbase.lite.*

object DatabaseManager {
    private lateinit var database: Database
    private const val TAG = "DatabaseManager"
    private var isVectorSearchEnabled = false

    fun init(context: Context, dbName: String = "myDatabase") {
        try {
            CouchbaseLite.init(context)
            Log.i(TAG, "CouchbaseLite initialized successfully")

            try {
                CouchbaseLite.enableVectorSearch()
                isVectorSearchEnabled = true
                Log.i(TAG, "Vector Search enabled successfully")
            } catch (e: CouchbaseLiteException) {
                Log.e(TAG, "Failed to enable Vector Search: ${e.message}")
                isVectorSearchEnabled = false
            }

            database = Database(dbName)
```

Mobile LLM Lab

October 2025

```
            Log.i(TAG, "Database '$dbName' opened successfully")
        } catch (e: CouchbaseLiteException) {
            Log.e(TAG, "Failed to initialize database: ${e.message}")
            throw RuntimeException("Database initialization failed", e)
        }
    }

    fun getDatabase(): Database = database

    fun isVectorSearchEnabled(): Boolean = isVectorSearchEnabled
}
```

- Remove any the mock classes (MockDatabase, MockDocument, MockArray, MockQuery, MockResultSet, and MockResult) from DatabaseManager.kt that may have missed.
- Update DocumentsDB.kt and ChunksDB.kt by replacing the command

```
private val database: MockDatabase = DatabaseManager.getDatabase()
```

With

```
private val database: Database = DatabaseManager.getDatabase()
```

- Make sure that both DocumentsDB.kt and ChunksDB.kt have the following import statement.

```
import com.couchbase.lite.*
```

- Sync and Run the application.

Mobile LLM Lab

October 2025

5. Add

Steps

These steps will add the actual implementation of the Gemini LLM class and test it in the MainActivity. When you run the application, you should see the response from the Gemini API in the logcat.

- Replace all of the contents of app/src/main/java/com/ml/shivay_couchbase/docqa/domain/llm/GeminiRemoteAPI.kt with the following code.

```
package com.ml.couchbase.docqa.domain.llm

import android.util.Log
import com.google.ai.client.generativeai.GenerativeModel
import com.google.ai.client.generativeai.type.GenerationConfig
import com.ml.couchbase.docqa.BuildConfig
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext

class GeminiRemoteAPI {

    private val apiKey = BuildConfig.geminiKey
    private val generativeModel: GenerativeModel

    init {
        // Here's a good reference on topK, topP and temperature
        // parameters, which are used to control the output of a LLM
        // See
        //
https://ivibudh.medium.com/a-guide-to-controlling-llm-model-output-explorin
g-top-k-top-p-and-temperature-parameters-ed6a31313910
        val configBuilder = GenerationConfig.Builder()
        configBuilder.topP = 0.4f
        configBuilder.temperature = 0.3f
        generativeModel =
```

Mobile LLM Lab

October 2025

```
            GenerativeModel(
                modelName = "gemini-1.5-flash",
                apiKey = apiKey,
                generationConfig = configBuilder.build()
            )
    }

    suspend fun getResponse(prompt: String): String? =
        withContext(Dispatchers.IO) {
            Log.e("APP", "Prompt given: $prompt")
            val response = generativeModel.generateContent(prompt)
            return@withContext response.text
        }
}
```

- Add the following to the **dependencies** section of your **app > build.gradle file**. Not the gradle > build.gradle.file.

```
implementation("com.google.ai.client.generativeai:generativeai:0.1.1")
```

- Generate Google Gemini API key by visiting the following link - https://ai.google.dev/gemini-api/docs/api-key. You will need a Google account.
- Add your Gemini API key to the local.properties file.

```
geminiKey="<YOUR API KEY HERE>"
```

- Replace all of the content of the MainActivity.kt file with the following.

```
package com.ml.couchbase.docqa

import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
```


Mobile LLM Lab

October 2025

```
import androidx.activity.compose.setContent
import androidx.activity.enableEdgeToEdge
import androidx.compose.animation.fadeIn
import androidx.compose.animation.fadeOut
import androidx.lifecycle.lifecycleScope
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import com.ml.couchbase.docqa.data.DatabaseManager
import com.ml.couchbase.docqa.domain.llm.GeminiRemoteAPI
import com.ml.couchbase.docqa.ui.screens.ChatScreen
import com.ml.couchbase.docqa.ui.screens.DocsScreen
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class MainActivity : ComponentActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        DatabaseManager.init(applicationContext)
        testGeminiAPI()
        setContent {
            val navHostController = rememberNavController()
            NavHost(
                navController = navHostController,
                startDestination = "chat",
                enterTransition = { fadeIn() },
                exitTransition = { fadeOut() }
            ) {
                composable("docs") { DocsScreen(onBackClick = {
navHostController.navigateUp() }) }
                composable("chat") {
                    ChatScreen(onOpenDocsClick = {
navHostController.navigate("docs") })
                }
```

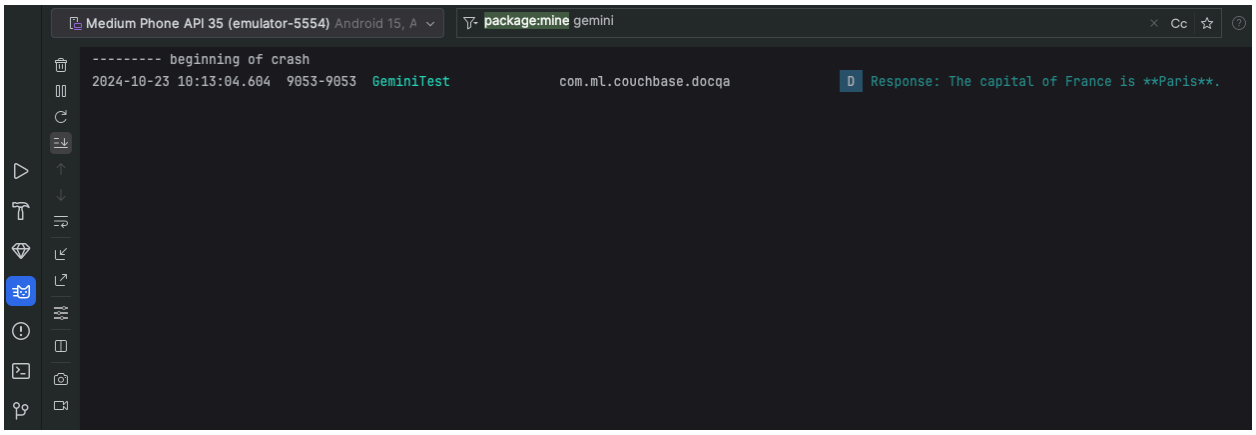
Mobile LLM Lab

October 2025

```
            }
        }
    }

    private fun testGeminiAPI() {
        val geminiAPI = GeminiRemoteAPI()
        lifecycleScope.launch {
            val response = geminiAPI.getResponse("What is the capital of
France?")
            Log.d("GeminiTest", "Response: $response")
        }
    }
}
```

- Sync Gradle and Run the application.
- Check to see if your application connected to the Gemini API by clicking on the cat icon (Logcat) on the left hand menu bar.
- In the search bar right after the package:mine, type in “gemini” to see if you got a response from the API. It should be “The capital of France is **Paris**.”.



6. Defining Database CRUD Operations

These implementations provide the basic functionality for adding, removing, and retrieving

documents and chunks using Couchbase Lite. The vector search functionality in ChunksDB and the corresponding method in ChunksUseCase have been left as placeholders to be implemented later.

Steps

- Replace all of the content in the **DocumentsDB.kt** file with the following.

```
package com.ml.couchbase.docqa.data

import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import kotlinx.coroutines.flow.flowOn
import com.couchbase.lite.Function
import com.couchbase.lite.*

class DocumentsDB {

    private val database: Database = DatabaseManager.getDatabase()

    fun addDocument(document: Document): String {
        val mutableDoc = MutableDocument()
        mutableDoc.setString("docFileName", document.docFileName)
        mutableDoc.setString("docText", document.docText)
        mutableDoc.setLong("docAddedTime", document.docAddedTime)
        database.save(mutableDoc)
        return mutableDoc.id
    }

    fun removeDocument(docId: String) {
        database.getDocument(docId)?.let {
            database.delete(it)
        }
    }
```


Mobile LLM Lab

October 2025

```
    @OptIn(ExperimentalCoroutinesApi::class)
    fun getAllDocuments(): Flow<List<Document>> = flow {
        val query = QueryBuilder
            .select(SelectResult.all())
            .from(DataSource.database(database))

        val result = query.execute()
        val documents = result.map { row ->
            val doc = row.getDictionary(database.name)
            Document(
                docId = doc?.getString("id")?.toLong() ?: 0,
                docFileName = doc?.getString("docFileName") ?: "",
                docText = doc?.getString("docText") ?: "",
                docAddedTime = doc?.getLong("docAddedTime") ?: 0
            )
        }
        emit(documents)
    }.flowOn(Dispatchers.IO)

    fun getDocsCount(): Long {
        val query = QueryBuilder

.select(SelectResult.expression(Function.count(Expression.string("*"))))
            .from(DataSource.database(database))

        val result = query.execute().firstOrNull()?.getInt(0)
        return result?.toLong() ?: 0L
    }

}
```

- Replace all of the content in the **ChunksDB.kt** file with the following.

```
package com.ml.couchbase.docqa.data
```

Mobile LLM Lab

October 2025

```
import com.couchbase.lite.*
import com.couchbase.lite.VectorEncoding
import com.couchbase.lite.VectorIndexConfiguration

class ChunksDB {

    private val database: Database = DatabaseManager.getDatabase()
    private val INDEX_NAME = "chunk_embedding_index"

    fun addChunk(chunk: Chunk) {
        val mutableDoc = MutableDocument()
        mutableDoc.setString("docFileName", chunk.docFileName)
        mutableDoc.setString("chunkData", chunk.chunkData)
        mutableDoc.setArray("chunkEmbedding", MutableArray().apply {
            chunk.chunkEmbedding.forEach { addFloat(it) }
        })
        database.save(mutableDoc)
    }



    fun removeChunks(docId: Long) {
        val query = QueryBuilder
            .select(SelectResult.expression(Meta.id))
            .from(DataSource.database(database))

.where(Expression.property("docId").equalTo(Expression.longValue(docId)))

        val result = query.execute()
        result.forEach { row ->
            val docId = row.getString("id") ?: return@forEach
            val doc = database.getDocument(docId)
            doc?.let { database.delete(it) }
        }
    }
}
```

Mobile LLM Lab

October 2025

- Replace all of the content in the **ChunksUseCase.kt** file with the following.

```
package com.ml.couchbase.docqa.domain

import android.util.Log
import com.ml.couchbase.docqa.data.Chunk
import com.ml.couchbase.docqa.data.ChunksDB
import com.ml.couchbase.docqa.domain.embeddings.SentenceEmbeddingProvider
import javax.inject.Inject
import javax.inject.Singleton

@Singleton
class ChunksUseCase
@Inject
constructor(private val chunksDB: ChunksDB, private val sentenceEncoder:
SentenceEmbeddingProvider) {

    fun addChunk(docId: String, docFileName: String, chunkText: String) {
        val embedding = sentenceEncoder.encodeText(chunkText)
        Log.e("APP", "Embedding dims ${embedding.size}")
        chunksDB.addChunk(
            Chunk(
                docId = docId,
                docFileName = docFileName,
                chunkData = chunkText,
                chunkEmbedding = embedding
            )
        )
    }

    fun removeChunks(docId: Long) {
        chunksDB.removeChunks(docId)
    }

    // fun getSimilarChunks(query: String, n: Int = 5): List<Pair<Float,
Chunk>> {
```



```
    //     val queryEmbedding = sentenceEncoder.encodeText(query)
    //     return chunksDB.getSimilarChunks(queryEmbedding, n)
    // }
}
```

- Sync and Run the App.

7. Implementing Vector Search

Steps

These implementations add vector search functionality to the ChunksDB, integrate it with the ChunksUseCase, and use it in the QAUseCase to retrieve relevant context for answering queries.

- Add the content before the **addChunk** function of the **ChunksDB.kt** file with the following.

```
    init {
        createVectorIndex()
    }

    private fun createVectorIndex() {
        val config = VectorIndexConfiguration("chunkEmbedding", 384, 3)
        config.encoding = VectorEncoding.none()
        database.createIndex(INDEX_NAME, config)
    }
```

- Add our Vector Search Query by adding the following content before the **removeChunks** function in the **ChunksDB.kt** file.

```
    fun getSimilarChunks(queryEmbedding: FloatArray, n: Int = 5):
List<Pair<Float, Chunk>> {
```


Mobile LLM Lab

October 2025

```
        val sql = """
        SELECT docFileName, chunkData, chunkEmbedding,
APPROX_VECTOR_DISTANCE(chunkEmbedding, ${'$'}embeddingArray ) as distance
        FROM ${database.name}
        ORDER BY distance
        LIMIT $n
        """
        val query = database.createQuery(sql)
        val embeddingArray = MutableArray().apply {
            queryEmbedding.forEach { addFloat(it) }
        }
        query.parameters = Parameters().apply {
            setArray("embeddingArray", embeddingArray)
        }

        val chunksWithScores = mutableListOf<Pair<Float, Chunk>>()
        query.execute().use { resultSet ->
            for (result in resultSet) {
                val docId = result.getString("docId") ?: ""
                val docFileName = result.getString("docFileName") ?: ""
                val chunkData = result.getString("chunkData") ?: ""
                val chunkEmbeddingArray = result.getArray("chunkEmbedding")
?: MutableArray()
                val distance = result.getFloat("distance")

                val chunkEmbedding =
FloatArray(chunkEmbeddingArray.count()) { i ->
                    chunkEmbeddingArray.getFloat(i)
                }

                val chunk = Chunk(
                    docId = docId,
                    docFileName = docFileName,
                    chunkData = chunkData,
                    chunkEmbedding = chunkEmbedding
                )
                chunksWithScores.add(Pair(distance, chunk))
```

Mobile LLM Lab

October 2025

```
            }
        }
        return chunksWithScores
    }
```

- Update the **getSimilarChunks** function in the **ChunksUseCase.kt** file with the following code.

```
fun getSimilarChunks(query: String, n: Int = 5): List<Pair<Float, Chunk>> {
        val queryEmbedding = sentenceEncoder.encodeText(query)
        return chunksDB.getSimilarChunks(queryEmbedding, n)
}
```

- Replace all of the content in the **QAUseCase.kt** file with the following.

```
package com.ml.couchbase.docqa.domain

import android.util.Log
import com.ml.couchbase.docqa.data.QueryResult
import com.ml.couchbase.docqa.data.RetrievedContext
import com.ml.couchbase.docqa.domain.llm.GeminiRemoteAPI
import javax.inject.Inject
import javax.inject.Singleton
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

@Singleton
class QAUseCase
@Inject
constructor(
    private val documentsUseCase: DocumentsUseCase,
    private val chunksUseCase: ChunksUseCase,
    private val geminiRemoteAPI: GeminiRemoteAPI
) {
```


Mobile LLM Lab

October 2025

```
    fun getAnswer(query: String, prompt: String, onResponse: ((QueryResult)
-> Unit)) {
        var jointContext = ""
        val retrievedContextList = ArrayList<RetrievedContext>()
        chunksUseCase.getSimilarChunks(query, n = 5).forEach {
            jointContext += " " + it.second.chunkData

retrievedContextList.add(RetrievedContext(it.second.docFileName,
it.second.chunkData))
        }
        Log.e("APP", "Context: $jointContext")
        val inputPrompt = prompt.replace("\$CONTEXT",
jointContext).replace("\$QUERY", query)
        CoroutineScope(Dispatchers.IO).launch {
            geminiRemoteAPI.getResponse(inputPrompt)?.let { llmResponse ->
                onResponse(QueryResult(llmResponse, retrievedContextList))
            }
        }
    }

    fun canGenerateAnswers(): Boolean {
        return documentsUseCase.getDocsCount() > 0
    }
}
```

8. Load Your Documents

Inside your project there's a folder called **sample_pdfs**. Within this folder you will find 2 sample pdfs, README.md and a video showing how to load the documents file. The README.md also has a few sample prompts you can use to test out the RAG functionality with the sample PDFs.

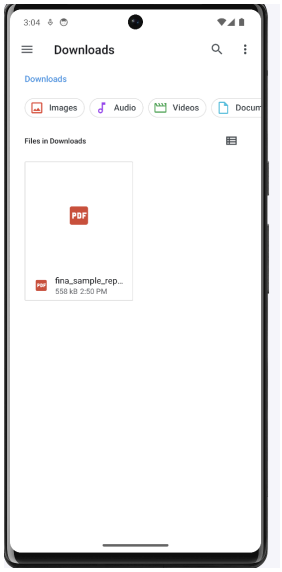
Steps

- Click on the folder icon at the top right hand corner of your application. The heading should say "Documents" with an arrow pointing back to the search.

Mobile LLM Lab

October 2025

- Click on the “PDF+” button at the bottom. This should take you to an empty folder option screen.
- Click on the hamburger icon on the upper left corner of the screen and select “Downloads”.
- Drag and drop a sample PDF file onto the emulator screen. You will get a message saying that it has been copied, but may not see the document there. To get the screen to refresh, try clicking on the white space or selecting another directory using the hamburger icon and then coming back. You should then see the document listed.



- Click on the document to begin the chunking process. Once that is complete, It will take you back to the documents folder where you see all of the available vector documents.
- Click the arrow at the top to return to the prompt screen.

9. Prompt Your Application

Steps

- Open the README.md file in **sample_pdfs** folder.
- Select a prompt for the document you just added to the application and paste it into the “Ask Documents…” field. Click the arrow and see the results.