**AndroidManifest.xml:**

*<?*xml version="1.0" encoding="utf-8"*?>*<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 <uses-permission android:name="android.permission.INTERNET" />  
 <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  
  
 <application  
 android:allowBackup="true"  
 android:dataExtractionRules="@xml/data\_extraction\_rules"  
 android:fullBackupContent="@xml/backup\_rules"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.Capitulo1\_practica1"  
 tools:targetApi="31">  
 <activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:label="@string/app\_name"  
 android:theme="@style/Theme.Capitulo1\_practica1">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>

**Paso 2:** Añadir las dependencias en **build.gradle.kts**(Module:app)

*implementation*("androidx.core:core-ktx:1.12.0")  
*implementation*("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")  
*implementation*("androidx.lifecycle:lifecycle-viewmodel-compose:2.7.0")  
*implementation*("androidx.activity:activity-compose:1.8.2")  
*implementation*("androidx.compose.ui:ui:1.5.4")  
*implementation*("androidx.compose.ui:ui-graphics:1.5.4")  
*implementation*("androidx.compose.material3:material3:1.1.2")  
*implementation*("com.google.ai.client.generativeai:generativeai:0.1.1")

**GeminiViewModel.kt:**

package com.example.capitulo1\_practica1  
  
import androidx.lifecycle.ViewModel  
import androidx.lifecycle.*viewModelScope*import com.google.ai.client.generativeai.GenerativeModel  
import kotlinx.coroutines.flow.MutableStateFlow  
import kotlinx.coroutines.flow.StateFlow  
import kotlinx.coroutines.flow.asStateFlow  
import kotlinx.coroutines.flow.update  
import kotlinx.coroutines.launch  
  
*// Esta clase extiende ViewModel, que es parte de la arquitectura de Android para manejar  
// y preservar datos relacionados con la UI a través de cambios de configuración*class GeminiViewModel : ViewModel() {  
 *// Inicialización del modelo generativo de Gemini  
 // Se configura con el modelo "gemini-pro" y requiere una API key* private val generativeModel = GenerativeModel(  
 modelName = "gemini-pro",  
 apiKey = "Aquí tu API KEY"  
 )  
  
 *// Estado mutable interno de la UI usando StateFlow  
 // MutableStateFlow permite modificar el estado y notificar a los observadores* private val \_uiState = *MutableStateFlow*(GeminiUiState())  
  
 *// Versión inmutable del estado expuesta a la UI  
 // Esto previene modificaciones no controladas desde fuera del ViewModel* val uiState: StateFlow<GeminiUiState> = \_uiState.*asStateFlow*()  
  
 *// Función para procesar preguntas del usuario* fun askQuestion(question: String) {  
 *// Lanza una corrutina en el alcance del ViewModel  
 viewModelScope*.*launch* **{** *// Actualiza el estado para mostrar el indicador de carga* \_uiState.*update* **{ it**.copy(isLoading = true) **}** try {  
 *// Intenta generar una respuesta usando el modelo de Gemini* val response = generativeModel.generateContent(question)  
 *// Actualiza el estado con la respuesta exitosa* \_uiState.*update* **{  
 it**.copy(  
 answer = response.text ?: "No response generated",  
 isLoading = false,  
 error = null  
 )  
 **}** } catch (e: Exception) {  
 *// En caso de error, actualiza el estado con el mensaje de error* \_uiState.*update* **{  
 it**.copy(  
 isLoading = false,  
 error = e.message  
 )  
 **}** }  
 **}** }  
}  
  
*// Clase de datos que representa el estado de la UI  
// Contiene toda la información necesaria para mostrar la interfaz de usuario*data class GeminiUiState(  
 val question: String = "", *// Pregunta actual* val answer: String = "", *// Respuesta del modelo* val isLoading: Boolean = false, *// Indicador de carga* val error: String? = null *// Mensaje de error si algo falla*)

GeminiApp.kt:

package com.example.capitulo1\_practica1  
  
import androidx.compose.foundation.layout.Column  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.foundation.layout.fillMaxWidth  
import androidx.compose.foundation.layout.padding  
import androidx.compose.material3.Button  
import androidx.compose.material3.CircularProgressIndicator  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.OutlinedTextField  
import androidx.compose.material3.Text  
import androidx.compose.runtime.Composable  
import androidx.compose.runtime.collectAsState  
import androidx.compose.runtime.getValue  
import androidx.compose.runtime.mutableStateOf  
import androidx.compose.runtime.remember  
import androidx.compose.runtime.setValue  
import androidx.compose.ui.Alignment  
import androidx.compose.ui.Modifier  
import androidx.compose.ui.unit.dp  
import androidx.lifecycle.viewmodel.compose.viewModel  
  
*// Esta es una función componible (Composable) que representa la interfaz principal de la aplicación*@Composable  
fun GeminiApp(  
 *// Parámetros de entrada: un modificador personalizable y una instancia del ViewModel* modifier: Modifier = Modifier,  
 viewModel: GeminiViewModel = viewModel()  
) {  
 *// Obtiene el estado actual de la UI desde el ViewModel y lo convierte en un estado observable* val uiState by viewModel.uiState.collectAsState()  
  
 *// Columna principal que contiene todos los elementos de la UI* Column(  
 modifier = modifier  
 .*fillMaxSize*() *// Ocupa todo el espacio disponible* .*padding*(16.*dp*) *// Añade un padding general* ) **{** *// Variable de estado para almacenar la pregunta del usuario* var question by remember **{** *mutableStateOf*("") **}** *// Campo de texto para que el usuario escriba su pregunta* OutlinedTextField(  
 value = question,*// Valor actual* onValueChange = **{** question = **it }**,*// Actualización del valor* label = **{** Text("Escribe tu pregunta") **}**,*// Etiqueta del campo* modifier = Modifier  
 .*fillMaxWidth*()*// Ancho completo* .*padding*(bottom = 16.*dp*)*// Espacio inferior* )  
  
 *// Botón para enviar la pregunta* Button(  
 onClick = **{** *// Solo envía la pregunta si no está vacía* if (question.*isNotBlank*()) {  
 viewModel.askQuestion(question)  
 }  
 **}**,  
 modifier = Modifier.*fillMaxWidth*()  
 ) **{** Text("Preguntar")  
 **}** *// Indicador de carga circular que se muestra durante el procesamiento* if (uiState.isLoading) {  
 CircularProgressIndicator(  
 modifier = Modifier  
 .*padding*(16.*dp*)  
 .*align*(Alignment.CenterHorizontally)  
 )  
 }  
  
 *// Muestra mensaje de error si existe* if (uiState.error != null) {  
 Text(  
 text = "Error: ${uiState.error}",  
 color = MaterialTheme.colorScheme.error,*// Color rojo para errores* modifier = Modifier.*padding*(top = 16.*dp*)  
 )  
 }  
  
 *// Muestra la respuesta si existe* if (uiState.answer.*isNotBlank*()) {  
 Text(  
 text = uiState.answer,  
 modifier = Modifier.*padding*(top = 16.*dp*)  
 )  
 }  
 **}**}

**MainActivity.kt:**

*// Declaración del paquete de la aplicación*package com.example.capitulo1\_practica1  
  
*// Importaciones necesarias para Activity y componentes de Jetpack Compose*import android.os.Bundle  
import androidx.activity.ComponentActivity  
import androidx.activity.compose.setContent  
import androidx.activity.enableEdgeToEdge  
import androidx.compose.foundation.layout.fillMaxSize  
import androidx.compose.material3.MaterialTheme  
import androidx.compose.material3.Surface  
import androidx.compose.ui.Modifier  
import com.example.capitulo1\_practica1.ui.theme.Capitulo1\_practica1Theme  
  
*// Clase principal que hereda de ComponentActivity, el punto de entrada de la aplicación*class MainActivity : ComponentActivity() {  
 *// Método onCreate que se llama cuando se crea la actividad* override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 *// Habilita la función edge-to-edge para aprovechar toda la pantalla  
 enableEdgeToEdge*()  
  
 *// setContent define el contenido de la interfaz usando Jetpack Compose  
 setContent* **{** *// Aplica el tema personalizado de la aplicación* Capitulo1\_practica1Theme **{** *// Surface es un contenedor que proporciona el fondo y otros atributos visuales* Surface(  
 *// Modifier.fillMaxSize() hace que la superficie ocupe toda la pantalla* modifier = Modifier.*fillMaxSize*(),  
 *// Usa el color de fondo definido en el tema Material* color = MaterialTheme.colorScheme.background  
 ) **{** *// GeminiApp es el componente principal de la interfaz de usuario* GeminiApp()  
 **}  
 }  
 }** }  
}