Guía paso a paso para recrear el proyecto gRPC-LLM (cliente gRPC → servidor intermedio → servidor final con LLM)
==========

1. Crear esqueleto del proyecto
```bash
mkdir grpc-llm && cd grpc-llm
cat > pom.xml <<'EOF'
# (en el siguiente paso pega el contenido completo del pom)
EOF
```

2. Sustituir el contenido de pom.xml (dependencias gRPC, protobuf, RxJava, JSON y plugin de compilación de .proto)
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      http://maven.apache.org/xsd/maven-4.0.0.xsd">

 <modelVersion>4.0.0</modelVersion>
 <groupId>com.grpc</groupId>
 <artifactId>ej3</artifactId>
 <version>1.0</version>

 <dependencies>
  <dependency>
   <groupId>com.google.protobuf</groupId>
   <artifactId>protobuf-java</artifactId>
   <version>3.22.2</version>
  </dependency>
  <dependency>
   <groupId>io.grpc</groupId>
   <artifactId>grpc-netty-shaded</artifactId>
   <version>1.54.0</version>
  </dependency>
  <dependency>
   <groupId>io.grpc</groupId>
   <artifactId>grpc-protobuf</artifactId>
   <version>1.54.0</version>
```

```
 </dependency>
 <dependency>
  <groupId>io.grpc</groupId>
  <artifactId>grpc-stub</artifactId>
  <version>1.54.0</version>
 </dependency>
 <dependency>
  <groupId>javax.annotation</groupId>
  <artifactId>javax.annotation-api</artifactId>
  <version>1.3.2</version>
 </dependency>
 <dependency>
  <groupId>io.reactivex.rxjava3</groupId>
  <artifactId>rxjava</artifactId>
  <version>3.1.8</version>
 </dependency>
 <dependency>
  <groupId>org.json</groupId>
  <artifactId>json</artifactId>
  <version>20230227</version>
 </dependency>
</dependencies>

<properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<build>
 <defaultGoal>clean generate-sources compile install</defaultGoal>
 <plugins>
  <plugin>
   <groupId>com.github.os72</groupId>
   <artifactId>protoc-jar-maven-plugin</artifactId>
   <version>3.6.0.1</version>
   <executions>
    <execution>
     <phase>generate-sources</phase>
     <goals>
      <goal>run</goal>
     </goals>
     <configuration>
```

```
        <includeMavenTypes>direct</includeMavenTypes>
        <inputDirectories>
         <include>src/main/resources</include>
        </inputDirectories>
        <!-- Descomentar SOLO SI TIENES macOS con arquitectura Intel (x86_64) --
>
        <!--<protocArtifact>com.google.protobuf:protoc:3.22.2:exe:osx-
x86_64</protocArtifact> -->
        <outputTargets>
         <outputTarget>
          <type>java</type>
          <outputDirectory>src/main/java</outputDirectory>
         </outputTarget>
         <outputTarget>
          <type>grpc-java</type>
          <pluginArtifact>io.grpc:protoc-gen-grpc-java:1.54.0</pluginArtifact>
          <outputDirectory>src/main/java</outputDirectory>
         </outputTarget>
        </outputTargets>
       </configuration>
      </execution>
     </executions>
    </plugin>
    <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-compiler-plugin</artifactId>
     <version>3.8.0</version>
     <configuration>
      <source>17</source>
      <target>17</target>
     </configuration>
    </plugin>
   </plugins>
  </build>
</project>
```

3. Crear la estructura de directorios
```bash
mkdir -p src/main/resources
mkdir -p src/main/java/client
```

```
mkdir -p src/main/java/server
mkdir -p src/main/java/service
```

4. Definir el contrato gRPC en src/main/resources/ask.proto
```proto
syntax = "proto3";

option java_multiple_files = true;
option java_package = "com.grpc.ej3";
option java_outer_classname = "AskProto";

// SERVICIO 1 (Middle)
service AskService {
 rpc preguntar (Pregunta) returns (Respuesta);
}

// SERVICIO 2 (Final)
service ResponseService {
 rpc responder (Pregunta) returns (Respuesta);
}

message Pregunta {
 string mensaje = 1;
}

message Respuesta {
 string mensaje = 1;
}
```

5. Generar código Java a partir del .proto
```bash
mvn clean compile
```
El plugin protoc-jar-maven-plugin genera las clases gRPC en src/main/java/com/grpc/ej3.

6. Implementar el servicio final que llama al LLM (Gemini u Ollama) en src/main/java/service/FinalService.java
```java
```

```
package service;

import com.grpc.ej3.*;
import io.grpc.stub.StreamObserver;

import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

import org.json.JSONArray;
import org.json.JSONObject;

public class FinalService extends
ResponseServiceGrpc.ResponseServiceImplBase {

   private static final String API_KEY =
"AIzaSyDFa8nSiWoqG0xeozY34Ax_lcrn53Hwo8w";
   private static final boolean MODO_LOCAL = false;
   private static final String OLLAMA_MODEL = "gemma3:4b";
   private static final double TEMPERATURE = 0.2;

   @Override
   public void responder(Pregunta req, StreamObserver<Respuesta> obs) {

      String pregunta = req.getMensaje();

      System.out.println("[FinalService] Recibida pregunta: " + pregunta);

      String respuestaLLM;

      if (MODO_LOCAL) {
         respuestaLLM = llamarOllama(pregunta);
      } else {
         respuestaLLM = llamarGemini(pregunta);
      }

      Respuesta resp = Respuesta.newBuilder()
            .setMensaje(respuestaLLM)
            .build();
```

```
        System.out.println("[FinalService] Enviando respuesta: " + respuestaLLM);

        obs.onNext(resp);
        obs.onCompleted();
    }

    private String llamarGemini(String preguntaUsuario) {

        String promptBase = "Responde de forma breve, clara y directa: ";
        String prompt = promptBase + preguntaUsuario;

        try {
            HttpClient client = HttpClient.newHttpClient();

            // Body JSON requerido por Gemini
            String body = """
            {
            "contents": [{
              "parts":[{"text":"%s"}]
            }],
            "generationConfig": {
              "temperature": %.2f
            }
            }
            """.formatted(prompt, TEMPERATURE);

            HttpRequest request = HttpRequest.newBuilder()
                    .uri(URI.create(
                      "https://generativelanguage.googleapis.com/v1beta/models/gemini-
2.5-flash:generateContent?key="
                        + API_KEY
                    ))
                    .header("Content-Type", "application/json")
                    .POST(HttpRequest.BodyPublishers.ofString(body))
                    .build();

            HttpResponse<String> response = client.send(
                    request, HttpResponse.BodyHandlers.ofString()

            );
```

```
        // Parsear JSON
        JSONObject json = new JSONObject(response.body());
        JSONArray candidates = json.getJSONArray("candidates");
        JSONObject content = candidates.getJSONObject(0)
                .getJSONObject("content");

        String texto = content
                .getJSONArray("parts")
                .getJSONObject(0)
                .getString("text");

        return texto;

    } catch (Exception e) {
        e.printStackTrace();
        return "Error al consultar Gemini: " + e.getMessage();
    }
}

private String llamarOllama(String preguntaUsuario) {

    String systemPrompt = "Responde de forma breve, clara y directa: ";

    try {
        HttpClient client = HttpClient.newHttpClient();

        // JSON para Ollama
        String body = """
        {
        "model": "%s",
        "system": "%s",
        "prompt": "%s",
        "temperature": %.2f
        }
        """.formatted(
                OLLAMA_MODEL,
                systemPrompt,
                preguntaUsuario,
                TEMPERATURE
        );
```

```
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create("http://localhost:11434/api/generate"))
            .header("Content-Type", "application/json")
            .POST(HttpRequest.BodyPublishers.ofString(body))
            .build();

        HttpResponse<String> response =
            client.send(request, HttpResponse.BodyHandlers.ofString());

        System.out.println("=== OLLAMA RESPONSE ===");
        System.out.println(response.body());
        System.out.println("==========================");

        StringBuilder respuestaFinal = new StringBuilder();

        String[] lineas = response.body().split("\n");

        for (String linea : lineas) {
          if (linea.isBlank()) continue;

          JSONObject json = new JSONObject(linea);

          if (json.has("response")) {
            respuestaFinal.append(json.getString("response"));
          }

          if (json.optBoolean("done", false)) {
            break;
          }
        }

        return respuestaFinal.toString().trim();

      } catch (Exception e) {
        return "Error al consultar modelo local Ollama: " + e.getMessage();
      }
    }
}
```

7. Implementar el servidor final en src/main/java/server/FinalServer.java

```java
package server;

import io.grpc.Server;
import io.grpc.ServerBuilder;
import service.FinalService;
import java.util.concurrent.TimeUnit;

public class FinalServer {

  public static void main(String[] args) throws Exception {

    Server server = ServerBuilder.forPort(50053)
        .addService(new FinalService())
        .build();

    System.out.println("FinalServer escuchando en puerto 50053...");
    server.start();
    server.awaitTermination(240, TimeUnit.SECONDS);
  }
}
```

8. Implementar el servicio intermedio (reenviador) en
src/main/java/service/MiddleService.java
```java
package service;

import com.grpc.ej3.*;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;

public class MiddleService extends AskServiceGrpc.AskServiceImplBase {

  private final ResponseServiceGrpc.ResponseServiceBlockingStub stubFinal;

  public MiddleService() {

    ManagedChannel channel = ManagedChannelBuilder
        .forAddress("localhost", 50053)
```

```
            .usePlaintext()
            .build();

        stubFinal = ResponseServiceGrpc.newBlockingStub(channel);
    }

    @Override
    public void preguntar(Pregunta req, StreamObserver<Respuesta> obs) {

        System.out.println("[MiddleService] Recibida pregunta de Cliente: " +
req.getMensaje());
        System.out.println("[MiddleService] Enviando pregunta a FinalServer...");

        Respuesta resp = stubFinal.responder(req);

        System.out.println("[MiddleService] Recibida respuesta a FinalServer: " +
resp.getMensaje());
        System.out.println("[MiddleService] Enviando respuesta a Cliente...");

        obs.onNext(resp);
        obs.onCompleted();
    }
}
```

9. Implementar el servidor intermedio en src/main/java/server/MiddleServer.java

```java
package server;

import io.grpc.Server;
import io.grpc.ServerBuilder;
import service.MiddleService;
import java.util.concurrent.TimeUnit;

public class MiddleServer {

    public static void main(String[] args) throws Exception {

        Server server = ServerBuilder.forPort(50052)
            .addService(new MiddleService())
            .build();
```

```
        System.out.println("MiddleServer escuchando en puerto 50052...");
        server.start();
        server.awaitTermination(240, TimeUnit.SECONDS);
    }
}
```

10. Implementar el cliente en src/main/java/client/Client.java

```java
package client;

import com.grpc.ej3.*;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import java.util.Scanner;

public class Client {

    private final AskServiceGrpc.AskServiceBlockingStub stub;
    private final ManagedChannel channel;

    public Client(String host, int port) {

        channel = ManagedChannelBuilder
                .forAddress(host, port)
                .usePlaintext()
                .build();

        stub = AskServiceGrpc.newBlockingStub(channel);
    }

    /**
     * Metodo reutilizable para enviar una pregunta
     */
    public void enviarPregunta(String mensaje) {

        System.out.println("[Client] Enviando pregunta -> " + mensaje);

        Pregunta p = Pregunta.newBuilder()
                .setMensaje(mensaje)
```

```
            .build();

        Respuesta r = stub.preguntar(p);

        System.out.println("\n=== RESPUESTA DEL SERVIDOR ===");
        System.out.println(r.getMensaje());
        System.out.println("==========================\n");
    }

    /**
     * Cierra el canal gRPC
     */
    public void shutdown() {
        System.out.println("[Client] Cerrando canal...");
        channel.shutdown();
    }

    public static void main(String[] args) {

        Client c = new Client("localhost", 50052);

        Scanner scanner = new Scanner(System.in);

        System.out.print("Escribe tu pregunta para el servidor gRPC: ");
        String pregunta = scanner.nextLine();

        c.enviarPregunta(pregunta);

        c.shutdown();
    }
}
```

11. Compilar y generar artefacto
```bash
mvn clean package
```

12. Lanzar los procesos (tres terminales)
```bash
# Terminal 1: servidor final (puerto 50053)
```

```
mvn exec:java -Dexec.mainClass=server.FinalServer

# Terminal 2: servidor intermedio (puerto 50052)
mvn exec:java -Dexec.mainClass=server.MiddleServer

# Terminal 3: cliente gRPC
mvn exec:java -Dexec.mainClass=client.Client
```

13. Probar el flujo
- El cliente pregunta en consola → MiddleServer reenvía a FinalServer → FinalServer llama al LLM (Gemini u Ollama) y devuelve la respuesta → MiddleServer la retorna al cliente.
- Para usar Gemini, reemplaza API_KEY por tu clave. Para usar Ollama local, pon MODO_LOCAL = true, asegura que el modelo gemma3:4b esté disponible y que Ollama corra en http://localhost:11434.

Con estos pasos tendrás recreado el proyecto completo gRPC-LLM.