Proyecto Aplicación Restaurante

Android - Express - JavaFX

Cristian Fernández

19 de noviembre de 2025

Índice

1. Descripción del proyecto

Este proyecto trata sobre la creación de una aplicación móvil, una aplicación de escritorio y una API-Rest todas comunicadas entre sí.

El objetivo es simular el funcionamiento interno de un restaurante con una comunicación lógica entre cliente<–>base de datos<–>trabajador:

- **Cliente:** El cliente del restaurante realizará su pedido a través de la aplicación móvil. Escojerá una mesa (de las 5 disponibles), se le mostrará toda la oferta de menús disponibles que pueda solicitar y se le permitirá hacer los pedidos que quiera. Por último podrá pagar su cuenta liberando así su mesa. Tanto la acción de pedir como la acción de pagar bloqueará la pantalla al usuario.
- **Trabajador:** El trabajador del restaurante gestionará la comanda a través de la aplicación de escritorio. Podrá seleccionar una mesa que esté ocupada, validar la comanda recibida y servir el pedido.
- **Base de datos:** Comunicará ambas aplicaciones. Debe estar escuchando constantemente peticiones para mantener actualizadas las comunicaciones.

La aplicación de móvil se desarrollara en **Android Studio** junto con la librería **Retrofit**, la aplicación de escritorio se desarrollara en **JavaFX** con la librería **Jackson** y la parte Backend se desarrollará con la base de datos **MongoDB** y una API REST creada con **Node.js/Express**.

2. Problemas encontrados

En líneas generales la mayor problemática de este proyecto ha girado en torno a la comunicación con la base de datos por medio de la API REST. Las librerías utilizadas en ambas aplicaciones (tanto Retrofit como la librería interna de Java HttpClient) son extremadamente estrictas en la nomenclatura de las variables empleadas en los modelos. Es decir, si la nominación de los atributos de una clase modelo no coinciden con los campos del documento de la base de datos se producirán una serie de errores muy molestos y en algunos casos difíciles de identificar. La solución más directa en este caso es tener una nomenclatura totalmente estricta de las propiedades de los modelos con respecto a los campos de los documentos de la base de datos (*un caso muy frecuente que da problemas es el ObjectId de mongo*).

Además, en situaciones en las que hay que realizar varias peticiones simultáneas, la API se "sobrecarga" liberando los recursos de la conexión y cerrándola de forma repentina. La solución que empleé en este caso fue aplicar un retardo entre ciertas peticiones, así no se pisan las unas con las otras y la conexión se mantiene estable.

Por último y no menos importante (pero sí muy subjetivo) es la gestión de la energía y el tiempo dedicados a completar las entregas. Es necesario establecer prioridades para dedicar la energía/tiempo a las tareas realmente importantes y así no quemarse en el proceso.

Un aprendizaje valioso que saco de esta práctica es que es muy recomendable documentar sobre la marcha ya que ahorra mucho tiempo en la recta final de la entrega.

3. Bibliografía

Enlaces a información consultada:

- **youtube**
  - *https://www.youtube.com/watch?v=MAw5Ku1OVFA*
- **stack overflow**
  - *https://stackoverflow.com/questions/45940861/android-8-cleartext-http-traffic-not-permitted/50834600*
- **geeksforgeeks**
  - *https://www.geeksforgeeks.org/android/android-recyclerview/*
  - *https://www.geeksforgeeks.org/kotlin/progressbar-in-android/*
- **mongodb.com**
  - *https://www.mongodb.com/resources/languages/express-mongodb-rest-api-tutorial*
- **IA de google - Gemini**