

# **UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA**

Ingeniería en Sistemas y Ciencias de la Computación

**Vibe Coding**

**Estudiante:**

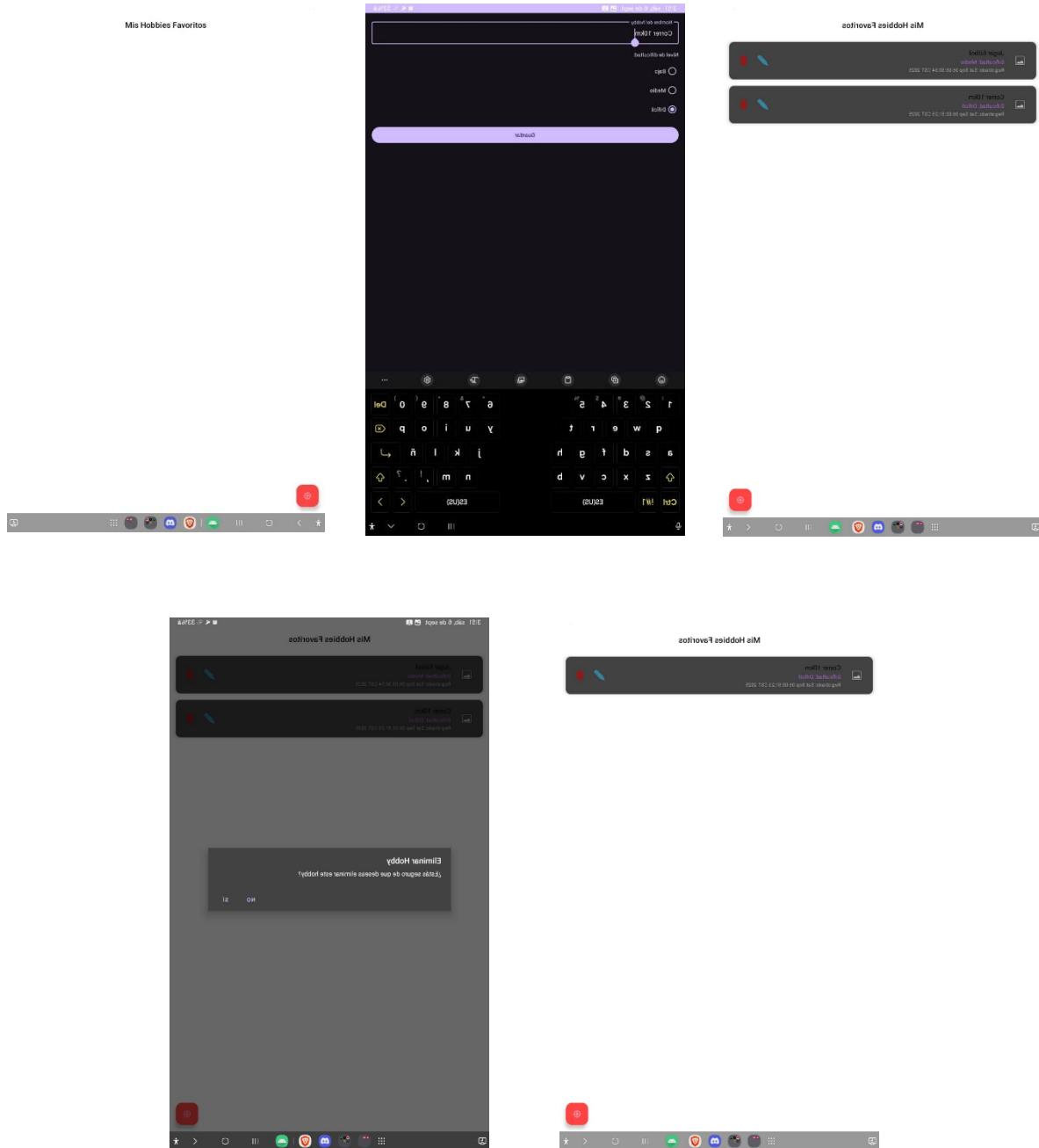
**Carlo René Hermógenes Rivera E. – 0905-24-7010**

Programación II

**Jutiapa, septiembre 2025**

# JobiCat

Capturas del programa:



## **Experiencia:**

Fue muy interesante utilizar el modo agente de copilot para realizar este proyecto, genuinamente me parece impactante lo precisa que puede llegar a ser con las indicaciones correctas. No me salió a la primera, en un principio no me gusto como quedo y al tratar de corregir errores, como uno donde no guardaba los hobbies, surgían otros, como que no me dejara eliminar los hobbies.

## **Prompt utilizado:**

# Instrucciones para el desarrollo de la aplicación JobiCat

### **## Descripción del proyecto**

Desarrollar una aplicación Android en Java llamada **\*\*JobiCat\*\*** que permita gestionar hobbies con diferentes niveles de dificultad.

### **## Objetivo de la aplicación**

- Llevar control de hobbies favoritos y su nivel de dificultad
- Niveles de dificultad: **\*\*bajo\*\***, **\*\*medio\*\*** y **\*\*difícil\*\***
- Almacenamiento en base de datos SQLite
- Funcionalidad CRUD completa: **\*\*Altas, Bajas y Cambios\*\***

### **## Requisitos técnicos**

#### **### Base de datos**

- [ ] Crear base de datos SQLite
- [ ] Diseñar tabla para almacenar hobbies con campos:
  - ID (clave primaria)
  - Nombre del hobby
  - Nivel de dificultad
  - Fecha de registro (opcional)

#### **### Estructura de la aplicación**

- [ ] Crear actividades necesarias:
  - MainActivity (lista de hobbies)
  - Activity para agregar nuevo hobby
  - Activity para editar hobby existente
- [ ] Implementar vistas correspondientes a cada actividad
- [ ] Crear adaptadores para mostrar lista de hobbies

#### **### Funcionalidades**

- [ ] **\*\*Alta de hobbies\*\*:**
  - Formulario con campos requeridos
  - Validación de campos no vacíos
  - Selección de nivel de dificultad
- [ ] **\*\*Baja de hobbies\*\*:**
  - Opción para eliminar hobbies individuales

- Confirmación antes de eliminar
- [ ] \*\*Cambio de hobbies\*\*:
  - Posibilidad de editar información existente
  - Validación de datos modificados

### ### Validaciones

- [ ] Verificar que ningún campo esté en blanco
- [ ] Validar formato de datos si es necesario
- [ ] Manejo de errores y mensajes al usuario

### ## Consideraciones importantes

- [ ] Mantener la estructura del proyecto ya existente
- [ ] Utilizar Java como lenguaje de programación
- [ ] Seguir buenas prácticas de desarrollo Android
- [ ] Implementar manejo adecuado de la base de datos

### ## Antes de comenzar

- ⚠ \*\*Preguntar al usuario antes de iniciar la codificación\*\* si:
- Se requiere algún campo adicional en la base de datos
  - Hay preferencias específicas para el diseño de la interfaz
  - Se necesita alguna funcionalidad adicional no mencionada
  - Hay dudas sobre la estructura del proyecto existente

### ## Entregable esperado

Aplicación Android funcional con todas las características mencionadas, integrada en el proyecto existente.

# AviService

Capturas del programa:

The image displays six screenshots of the AviService application, showing its various features:

- Screenshot 1: Main Menu (AviService)**
  - Top bar: Shows the date (6 de sept.) and battery level (35%).
  - Bottom bar: Shows system icons (calculator, file manager, browser, etc.).
  - Menu items: "Registrar Usuario", "Historial", and "Análisis de Patrones".
- Screenshot 2: Registro de Usuario (User Registration - Nicaragua)**
  - Form fields:
    - Nationality: Nicaragua
    - Name: Pepe Tofio
    - Age: 36
  - Buttons: "Registrar" (Register) and a numeric keypad.
- Screenshot 3: Registro de Usuario (User Registration - Chile)**
  - Form fields:
    - Nationality: Chile
    - Name: Luis
    - Age: 20
    - Gender: Masculino
  - Buttons: "Registrar" (Register).
- Screenshot 4: Historial de Usuarios (User History)**
  - Table header: "Lista".
  - Table rows:
    - Nationality: Chile  
Age: 20  
Sexo: Masculino  
Acceso: Permitido  
Tiempo: 4160 ms
    - Pepe Matias  
Nationalidad: Nicaragua  
Edad: 66  
Sexo: Masculino  
Acceso: Permitido  
Tiempo: 4251 ms
- Screenshot 5: Análisis de Patrones (Pattern Analysis)**
  - Text: "Promedio de tiempos por nacionalidad:  
Nicaragua: 4251.00 ms  
Chile: 4160.00 ms"
  - Text: "Correlación nombre-tiempo: 1.00"
  - Text: "Distribución por grupos de edad:  
61-100: 1  
36-60: 0  
1-19: 0  
18-38: 1"
  - Text: "Usuario con menor tiempo: Luis (4160 ms)"

## **Experiencia:**

Este segundo proyecto fue un poco mas sencillo gracias a la experiencia con el anterior, en esta ocasión decidí utilizar Claude sonnet en vez de GPT 4.1, que a diferencia de GPT 4.1, no me hizo consultas y trato de generar el código por completo. No se logro al primer intento, pero demoro mucho menos tiempo que antes corregir los errores que surgieron.

## **Prompt utilizado:**

Prompt de Desarrollo - Sistema AviService

### 1. Objetivo del Sistema

Desarrollar una aplicación móvil en Android (Java) llamada AviService, que simule el registro, procesamiento y análisis de datos de personas extranjeras en un aeropuerto. La aplicación debe permitir capturar datos personales, simular decisiones de acceso al país, calcular tiempos de procesamiento y realizar análisis de patrones con base en los registros almacenados.

### 2. Requerimientos Funcionales

#### 2.1. Captura de Datos del Usuario

Campos obligatorios a registrar:

Nacionalidad (String)

Nombre completo (String)

Edad (Integer)

Sexo (String): "Masculino", "Femenino", "Otro"

#### 2.2. Procesamiento y Análisis

Simulación de acceso al país: Algoritmo que determine aleatoriamente si el usuario puede ingresar.

Cálculo de tiempo de procesamiento: Medir el tiempo desde el inicio hasta la finalización del análisis.

Registro en base de datos: Guardar todos los datos del usuario junto con el tiempo de procesamiento y el resultado del acceso.

#### 2.3. Análisis de Patrones

La aplicación debe identificar y analizar patrones basados en:

Nacionalidad

Longitud del nombre y uso de caracteres especiales

Edad (rangos o grupos)

Sexo

Además, se debe identificar a los usuarios con los menores tiempos de procesamiento.

#### 2.4. Base de Datos

Tipo: SQLite

Tabla principal: Usuarios

Campos:

id (PK, AUTO\_INCREMENT)

nacionalidad (String)

nombre (String)  
edad (Integer)  
sexo (String)  
tiempoProcesamiento (Long - en milisegundos)  
accesoPermitido (Boolean)  
timestamp (Long - fecha de registro)  
Operaciones CRUD: Altas, Bajas, Cambios, Consultas  
3. Arquitectura Técnica Propuesta  
3.1. Modelo de Datos  
Clase Usuario:

Java

```
1
2
3
4
5
6
7
8
9
10
^
public class Usuario {
    private int id;
    private String nacionalidad;
    private String nombre;
    private int edad;
    private String sexo;
    private long tiempoProcesamiento;
    private boolean accesoPermitido;
    private long timestamp;
}
```

### 3.2. Componentes de la Aplicación

#### A. Activities

MainActivity – Pantalla principal con menú de navegación.

RegistroActivity – Formulario para captura de datos.

ResultadosActivity – Muestra el resultado del análisis (acceso permitido o no).

HistorialActivity – Lista de registros previos.

AnalisisActivity – Dashboard con estadísticas y patrones identificados.

#### B. Componentes de Base de Datos

DatabaseHelper – Extiende SQLiteOpenHelper.

UsuarioDAO – Implementa operaciones CRUD.

DatabaseManager – (Opcional) Gestiona la conexión y uso de la base de datos.

#### C. Componentes de Procesamiento

ProcesadorAcceso – Simula análisis y determina acceso.  
AnalizadorPatrones – Identifica tendencias y patrones.  
CalculadoraTiempos – Métodos para estadísticas de tiempos de procesamiento.  
D. Interfaz de Usuario  
Uso de RecyclerView para listas.  
Custom Views para visualización de datos estadísticos.  
Dialogs para confirmaciones y alertas.  
4. Flujo de Trabajo  
Captura de datos del usuario.  
Procesamiento simulado (acceso y tiempo).  
Guardado en base de datos.  
Análisis de patrones y generación de estadísticas.  
Validaciones Obligatorias  
Todos los campos son requeridos.  
Edad debe ser un número positivo.  
Nacionalidad y nombre deben ser no vacíos.  
5. Reglas de Negocio para Simulación  
Tiempo de procesamiento: Simulado entre 100 y 5000 ms.  
Acceso permitido: Determinado por un algoritmo aleatorio ponderado.  
Patrones a identificar:  
Promedio de tiempos por nacionalidad.  
Correlación entre longitud del nombre y tiempo de procesamiento.  
Distribución por grupos de edad.  
6. Instrucciones para el Desarrollo  
Antes de comenzar:  
Confirmar la arquitectura propuesta.  
Validar nombres de paquetes y clases.  
Verificar estructura de la base de datos.  
Aclarar cualquier ambigüedad en los requerimientos.  
Durante el desarrollo:  
Generar código completo y funcional.  
Incluir comentarios explicativos.  
Seguir buenas prácticas de programación en Java.  
Manejar excepciones adecuadamente.  
Validar entradas del usuario.  
7. Entregables Esperados  
Código fuente completo (archivos .java)  
Layouts en XML  
Archivo AndroidManifest.xml  
Documentación básica de uso  
8. Preguntas Obligatorias Antes de Iniciar  
¿Qué algoritmo específico se debe usar para simular si se permite el acceso?  
¿Qué patrones específicos deben priorizarse en el análisis?  
¿Qué estilo de interfaz se prefiere? (Material Design, clásico, etc.)  
¿Existen requisitos adicionales de validación de datos?  
¿Se requieren notificaciones o alertas especiales en algún proceso?

 Importante

No iniciar el desarrollo hasta recibir confirmación explícita de las especificaciones anteriores.  
La claridad y precisión son prioritarias sobre la velocidad de desarrollo.

**Repositorios:**

JobiCat: <https://github.com/CarloRH/JobiCat.git>

AviService: <https://github.com/CarloRH/AviService.git>