

Universidad del Quindío
Ingeniería de Sistemas y
Computación
Programación III

Proyecto final:
UrbanFleet

1. Definición del Proyecto final

1.1 Equipo de trabajo

Indicar en la tabla los participantes del proyecto.

1.2 Enunciado del proyecto

Se debe implementar un sistema multijugador que simula una flota de taxis en una ciudad virtual. Varios usuarios (clientes y conductores) se conectan en tiempo real desde la línea de comandos (CLI). Los clientes solicitan viajes y los conductores los aceptan; el sistema gestiona múltiples solicitudes en paralelo.

El objetivo es completar viajes para acumular puntos, ya sea por **eficiencia** (conductores) o por **cumplir destinos** (clientes).

1.3 Objetivos de Aprendizaje

- Aplicar concurrencia y procesos en Elixir.
- Implementar un sistema distribuido con **GenServers y Supervisores dinámicos**.
- Manejar persistencia en archivos de texto plano.
- Construir un sistema interactivo en tiempo real con **usuarios, solicitudes y ranking**.

1.4 Funcionalidades Requeridas

1.4.1. Gestión de Usuarios

- Registro y login de **clientes y conductores**.
- Usuarios almacenados en `users.dat` (nombre, rol, contraseña, puntaje acumulado).
- Consultar puntaje propio.
- Ranking global de mejores conductores y clientes.

1.4.2. Conexión

- Conectarse (`connect username password`).
- Si no existe → registro automático.
- Desconectarse (`disconnect`).

1.4.3. Solicitudes de Viaje

- Un cliente puede crear una solicitud de viaje:
`request_trip origen=Parque destino=Centro`
- Cada solicitud es un proceso independiente (GenServer) bajo un DynamicSupervisor.
- Un viaje admite un solo conductor.

1.4.4. Gestión de Viajes

- Los conductores listan viajes disponibles: `list_trips`.
- Un conductor acepta viaje: `accept_trip trip_id`.
- El viaje inicia con temporizador (ej: 20s) que simula duración.
- Al terminar:
 - El cliente recibe +10 puntos.
 - El conductor recibe +15 puntos.
 - Si el viaje expira sin conductor, el cliente pierde -5 puntos.

1.4.5. Resultados y Ranking

- Historial en `results.log` → fecha, cliente, conductor, origen, destino y estado.
- Actualización de puntajes en `users.dat`.

1.4.6. Persistencia

- `users.dat` → usuarios y puntajes.
- `results.log` → historial de viajes.
- `locations.dat` → lista de ubicaciones válidas de la ciudad.

1.4.7. Concurrencia y Tiempo Real

- Varios viajes en paralelo.
- Temporizador por cada viaje.
- El sistema debe manejar múltiples clientes y conductores simultáneamente.

1.5 Flujo Ejemplo

1. Ana (cliente) se conecta y crea un viaje:
`request_trip origen=Parque destino=Centro`
2. Luis (conductor) se conecta y lista viajes:
`list_trips` → aparece el viaje de Ana.
3. Luis acepta el viaje:
`accept_trip trip123`
4. El viaje dura 20 segundos (simulación).
5. Al terminar:
 - Ana obtiene +10 puntos.
 - Luis obtiene +15 puntos.
 - Se guarda en `results.log`:
`2025-09-25; cliente=ana; conductor=luis; origen=Parque; destino=Centro; status=Completado.`

1.6 Archivos Recomendados

proyecto_taxi/

| — lib/

| | — taxi/

| | | — server.ex # Servidor principal (gestión de usuarios y comandos)

| | | — trip.ex # GenServer de viaje

| | | — supervisor.ex # DynamicSupervisor de viajes

| | | — user_manager.ex # Gestión de usuarios

| | | — location.ex # Manejo de ubicaciones

| — data/

| | — users.dat

| | — results.log

| — locations.dat

| — test/

| — mix.exs

1.7 Posibles Extensiones

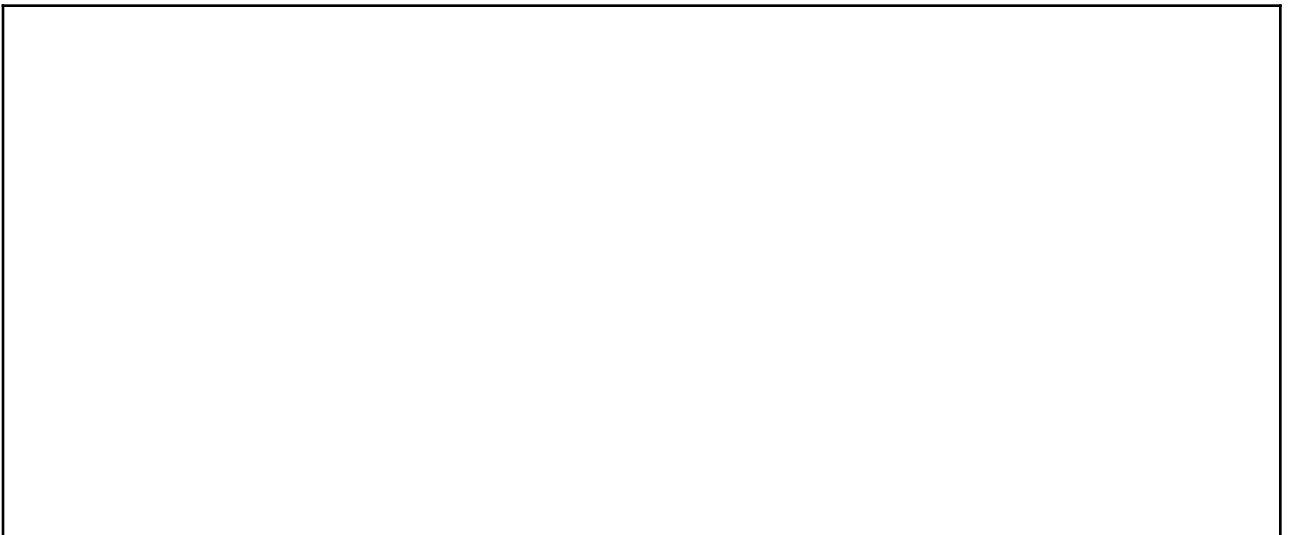
- Ranking separado para **clientes y conductores**.
- Penalizaciones por cancelación.
- “Bonos” por atender múltiples viajes seguidos.
- Persistencia en formato JSON para más claridad.
- Versión distribuida: que los viajes puedan correrse en nodos distintos de Elixir.

2. Desarrollo del Proyecto final

En las siguientes secciones se muestra el desarrollo del proyecto, complete la información solicitada.

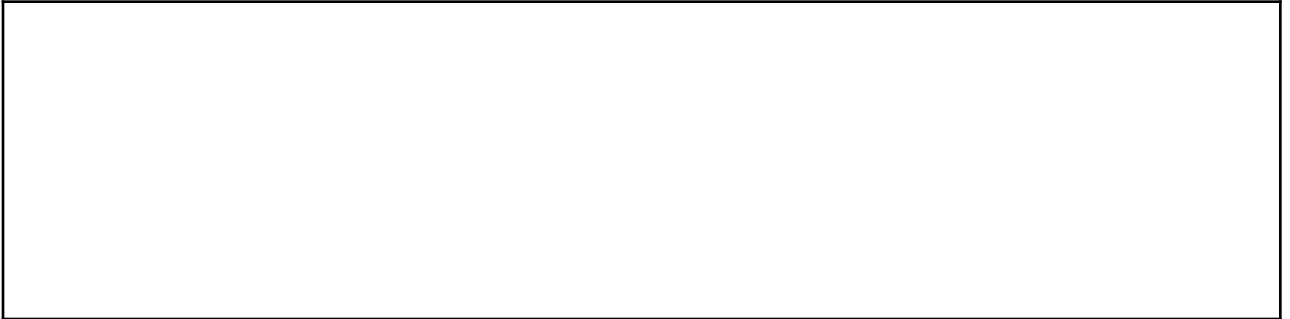
2.1. Diagrama de Arquitectura de Módulos

Colocar una imagen del modelo de datos que represente el enunciado.

A large, empty rectangular box with a thin black border, intended for the student to draw a data model diagram.

1.6 Modelo de datos

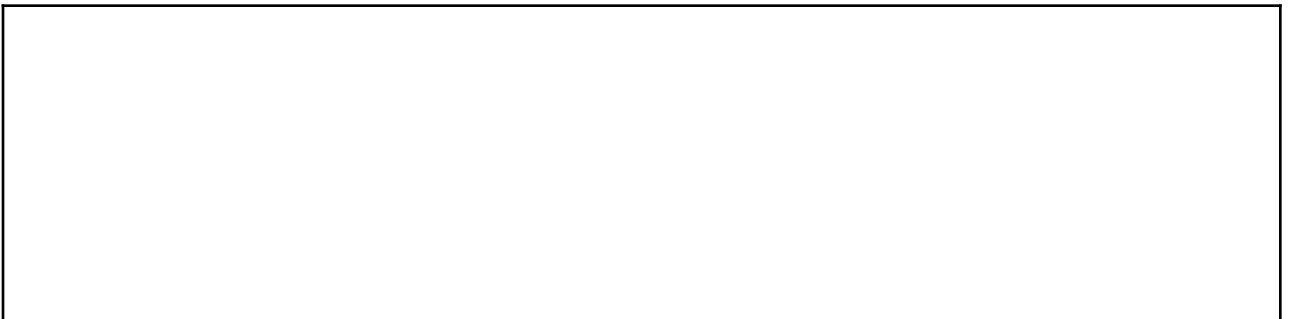
Colocar una imagen del modelo de datos que represente el enunciado.

A large, empty rectangular box with a black border, intended for a data model diagram.

1.7 Diagrama de Secuencia

A large, empty rectangular box with a black border, intended for a sequence diagram.

1.8 Diagrama de Procesos y Supervisión (OTP Tree)

A large, empty rectangular box with a black border, intended for a process and supervision diagram (OTP Tree).

3. Estructuración del proyecto

En las siguientes secciones cumplir con lo que se solicita y colocar una imagen de acuerdo a la sección.

2.1 Estructura del proyecto

Utilizar la estructura de trabajo.



4. Demostración

4.1 Video de demostración

Realizar un vídeo haciendo una demostración de la aplicación desarrollada cumpliendo con lo solicitado.



5. Repositorio

5.1 Link repositorio

Entregar el link del repositorio.

