Práctica 1: Sistemas Distribuidos. Sistema de Gestión de un Taller

Alexandre Lacoste Rodríguez

3 de noviembre de 2025

1. Introducción

Este documento describe el diseño y funcionamiento de un sistema de gestión de un taller de coches implementado en Go. El sistema opera en memoria y ofrece operaciones de alta, baja, modificación, consulta y asignación sobre clientes, vehículos, incidencias y mecánicos, así como la gestión de plazas de taller.

- **El código *taller.go* se encuentra en: Repositorio Práctica 1 de GitHub**
- **Enlace al vídeo explicativo: Video explicativo**

2. Alcance y objetivos

- Gestionar **clientes**, **vehículos**, **incidencias** y **mecánicos**.
- Asignar **vehículos a plazas** respetando la capacidad (2 plazas por mecánico activo).
- Consultas frecuentes: estado del taller, incidencias por vehículo y por mecánico, mecánicos disponibles, clientes con vehículos en taller.
- Operación **interactiva** por línea de comandos (menús y submenús).
- **Persistencia** no incluida (todo en memoria).

3. Arquitectura general

El programa sigue una arquitectura monolítica en memoria, centrada en la estructura raíz **Garage** que agrega colecciones (**slices**) de entidades y una lista de plazas (**Slots)** Cada entidad se referencia por su ID fuerte (**ClientID, MechanicID, IncidenceID, VehicleID**) para evitar ciclos de punteros y facilitar operaciones. A continuación mostraré el diagrama de casos de uso e iré explicando en profundidad sus usos y funciones.

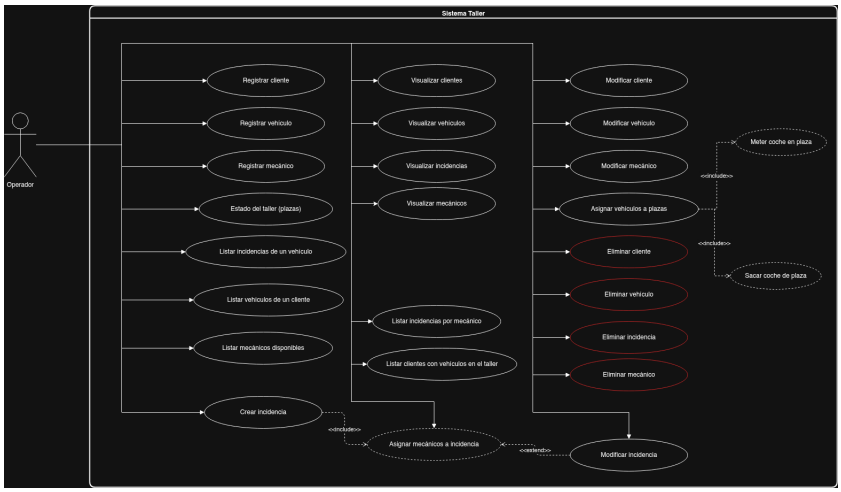

Figura 1: Diagrama de casos de Uso

4. Estructuras de datos

4.1. Tipos enumerados y aliases de ID

He usado tipos propios (en vez de int/string crudos) para **fortalecer el tipado** y evitar mezclar IDs por error.

- **Tipos de Incidencias**: IssueType { Mecánica, Eléctrica, Carrocería }
- **Prioridad**: Priority { Baja, Media, Alta }
- **Estado de Incidencia**: IssueStatus { Abierta, En proceso, Cerrada }
- **Especialidad mecánico**: SkillType { Mecánica, Eléctrica, Carrocería }
- **Aliases de ID**: ClientID (int64), MechanicID (int64), IncidenceID (int64), VehicleID (string).

4.2. Entidades

Relaciones lógicas:

- Cliente (1..N) Vehículo (por **OwnerID**). Un cliente tiene asociado uno o más vehículos. Un vehículo sólo es de un cliente.
- Vehículo (1..N) Incidencia (por **VehicleID**). Un vehículo sufre de 0 a N incidencias. Una incidencia es de un sólo vehículo.
- Incidencia (N..M) Mecánico (**Mechanics** []**MechanicID**). Un mecánico puede tener asociadas de 0 a N incidencias. Una incidencia puede tener asociados de 0 a M mecánicos.

- Slot (0..1) Vehículo (**VehicleID** opcional; ***nil*** si libre). Un vehículo puede estar o no asignado a una plaza. En una plaza puede haber o no un vehículo.

4.2.1. Client

Atributos:

- ID, Name, Phone, Email.
- Referencia a vehículos mediante **OwnerID** en Vehicle.
- Además se usa un slice []VehicleID en el cliente para acelerar listados de vehículos del propio cliente.

4.2.2. Mechanic

Atributos:

- ID, Name, Skill, Experience, Active.
- slice *issues* con IncidenceID para la relación inversa N..M.

4.2.3. Vehicle

Atributos:

- ID (matrícula), Brand, Model, OwnerID
- CheckInAt, ETA (fechas; cuando no están definidas se imprime "aún no definida" usando t.IsZero())
- slice *issues* con IncidenceID.

4.2.4. Incidence

Atributos:

- ID, VehicleID, kind (tipo), Prio, Description, Status.
- slice *mecanics* con MechanicID para la relación inversa M..N.

4.2.5. Slot

Atributos:

- Number
- VehicleID *VehicleID (puntero; *nil* si está libre).

4.3. Garage

Esta entidad no la represento dentro del diagrama relacional que viene a continuación debido a que se trata del garage en sí y se sobreentiende que corresponde a la base de datos (en nuestro caso es en memoria) donde se encontrarían almacenadas el resto de entidades.

Atributos:

- clients []*Client
- vehicles []*Vehicle
- mechanics []*Mechanic
- issues []*Incidence
- slots []*Slot.

4.4. Recopilación

Dadas las especificaciones, en caso de querer implementar persistencia de datos para el sistema en el futuro, veo conveniente plantear el Diagrama Relacional que seguiría para una posible futura base de datos. Asimismo, dicho diagrama ayudará a la comprensión de las relaciones entre las distintas estructuras de datos.

Figura 2: Diagrama Relacional de las entidades del sistema.

5. Invariantes y reglas de dominio

- Capacidad: 2 plazas por mecánico activo.
- IDs únicos por entidad; validación en altas de entidades.
- **Restricciones referenciales**:
  - Una incidencia requiere un Vehiculo válido.
  - Los mecánicos de una incidencia deben existir.
  - Al borrar hay que limpiar todas las referencias.

6. Interacción y flujo de la aplicación

La interacción se realiza por terminal en menús/submenús. El siguiente diagrama de flujo muestra la navegación de: Menú principal → Crear / Visualizar / Modificar / Eliminar / Salir y los submenús que derivan de estos.

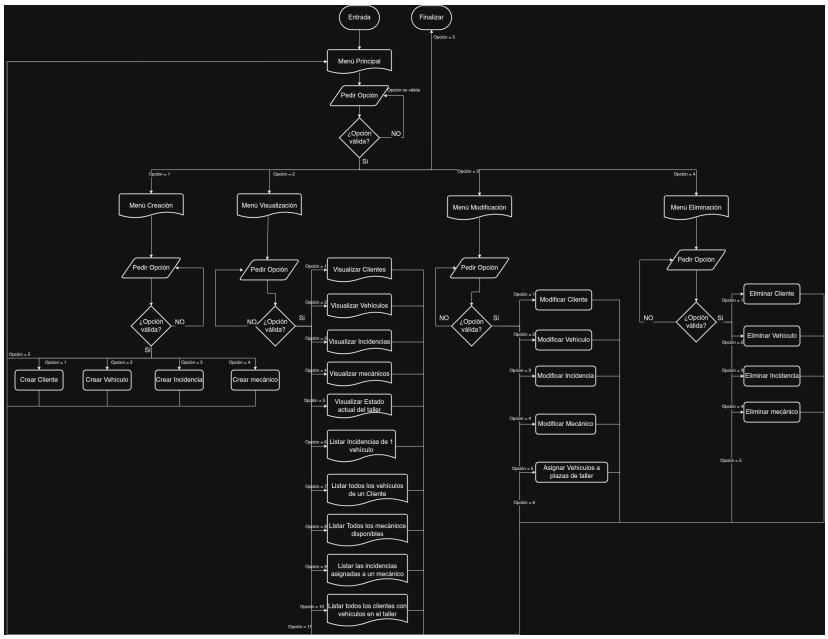

Figura 3: Diagrama de Flujo del sistema

7. Operaciones principales

7.1. Creación

- **Clientes, vehículos, Incidencias, Mecánicos.**
  - Cliente: *newClient()*
  - Vehículo: *newVehicle()*
  - Incidencia: *newissue()*
  - Mecánico: *newMech()*
- **Lecturas de IDs** con funciones de ayuda:
  - *askUniqueIntID(prompt, exists) :* Pide un ID único de tipo entero.
  - *askUniqueStrID(prompt, exists):* Pide un ID único de tipo string (para las matrículas)
- Funciones encargadas de **validaciones de unicidad** (usadas en *askUniqueIntID / askUniqueStrID*). Todas ellas devuelven un puntero a una entidad a partir de un ID que se le pase por argumento:
  - *ownerIDexists:* Devuelve un puntero a cliente.
  - *vidExists:* Devuelve un puntero a vehículo.

- *issueIDexists:* Devuelve un puntero a una incidencia.
- *mechIDexists:* Devuelve un puntero a un mecánico.

7.2. Visualización

- Listados por cada entidad y combinados.
  - Clientes: *displayClients()*
  - Vehículos: *displayVehicles()*
  - Incidencias: *displayIssues()*
  - Mecánicos: *displayMechs()*
  - Vehículos de un cliente: *listVFromClient()*
- **Estado del taller** (plazas ocupadas/libres).
  - *displaySlots()*
- **Clientes** con vehículos en el taller.
  - *listClientsVInGarage()*
- **Incidencias** por vehículo / por mecánico.
  - Por mecánico: *dispIssuesOfMech()*
  - Por vehículo: *listIncFromAvehicle()*
- **Mecánicos disponibles**: los no asignados a ninguna incidencia.
  - *listDispMech()*

7.3. Modificación

- Edición de campos básicos de las entidades principales y asignación de plazas en el taller.
  - **Cliente**: *modifyClient()*
  - **Vehículo**: *modifyVehicle()*
  - **Incidencia**: *modifyIssue()*
  - **Mecánico**: *modifyMech()*
  - **Asignación de plazas**: *modifySlots()*
- **Cambios de ID:** Este caso es más delicado pero puede confiar en que cuando se cambia el ID de alguna entidad, se cambia también todas sus referencias en el resto de entidades.

7.4. Eliminación

- Borrado de un **vehículo** (*delVehicle()*):
  - Elimina incidencias asociadas: *delIssueByID()*
  - Lo quita del dueño (quita el ID de su slice de *VehicleID*)
  - Libera el *slot* que estuviera ocupando.
  - Lo elimina del slice global.
- Borrado de **incidencia** (*delIssue()*):
  - Quita el ID de todos los vehículos y mecánicos que lo tengan.
  - La elimina de la lista global.
- Borrado de **cliente** (*delClient()*):
  - Itera por cada vehículo que tenga asociado y lo elimina, haciendo que a su vez de eliminen las incidencias asociadas a cada vehículo (eliminación en cascada).
  - Lo elimina del slice global.
- Borrado de **mecánico** (*delMech()*):
  - Elimina el *MechanicID* de todas las incidencias donde estuviere asignado.
  - Reduce la capacidad de *Slots* en 2 unidades. Los coches que hubiera en ellas sencillamente quedan sin estar en una plaza.
  - Se elimina del slice global.

8. Extensiones futuras

- **Persistencia** (SQLite/PostgreSQL) y repositorios por entidad.
- Índices en memoria / **mapas auxiliares** cuando se permitan.
- **Validaciones avanzadas** (emails, matrículas, etc.) y tests automáticos.
- **Interfaz TUI** o web separando dominio y presentación.

9. Conclusión

El sistema en memoria proporciona una base sólida para la gestión del taller, con tipos fuertes y relaciones claras. La arquitectura es adecuada para evolucionar hacia persistencia y mejores tiempos de consulta sin romper la API interna.