

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTADORES

CE-3101 BASES DE DATOS

---

# F1 Garage Manager

Proyecto de Curso

---

*Modalidad:* Verano Intensivo

*Profesor:* MSc. Andrés Vargas

*Estudiantes:* Artavia Leiton Anthony Jose

Chaves Mena Luis Felipe

Feng Wu Alexs Eduardo

Monge Navarro Bryan Alexander

*Periodo:* Diciembre 2025 – Enero 2026

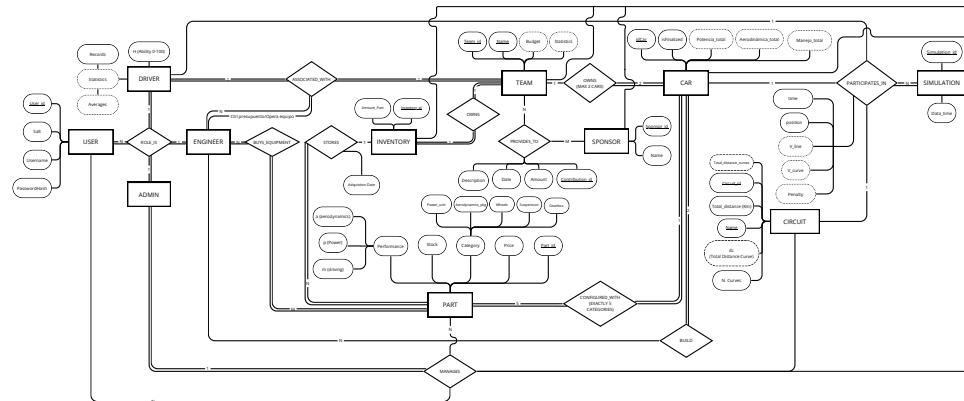
**Repositorio Oficial del Proyecto:**

<https://github.com/AnthonyArtavia20/F1GarageManagerBDVerano.git>

# Avance 1

## Modelo Conceptual ER

## 1. Diagrama ER



**Figura 1:** Diagrama Entidad-Relación para el proyecto *F1 Garage Manager*.

## 2. Entidades del Modelo y Reglas del Negocio

### 2.1. User

Entidad base para autenticación con atributos: `User_id`, `Username`, `Salt`, `PasswordHash`.

**Relaciones y Cardinalidades** Relación de especialización 1:1 con `Engineer`, `Driver` y `Admin`. Participación total del usuario.

#### Reglas de Negocio

- **RN-USER-01:** Usuario con exactamente un rol (`Engineer`, `Driver` o `Admin`).
- **RN-USER-02:** Contraseñas hash con salt único (`Argon2id/bcrypt`).
- **RN-USER-03:** Sesiones seguras con cookies `HttpOnly`, `Secure`, `SameSite`.
- **RN-USER-04:** Timeout de sesión y logout funcional.

### 2.2. Engineer

Rol para gestionar presupuesto, compras y armado de carros. Sin atributos propios.

#### Relaciones y Cardinalidades

- Relación N:M con `Part` (compras)
- Relación N:1 con `Team` (pertenencia)
- Relación con `Car` (armado, hasta 2 por equipo)

#### Reglas de Negocio

- **RN-ENG-01:** Único rol autorizado para comprar partes.
- **RN-ENG-02:** Gestiona presupuesto = suma de aportes.
- **RN-ENG-03:** Valida presupuesto y stock antes de comprar.
- **RN-ENG-04:** Máximo 2 carros por equipo.
- **RN-ENG-05:** Solo instala partes del inventario.
- **RN-ENG-06:** Inventario disminuye al instalar.
- **RN-ENG-07:** Parte anterior regresa al reemplazar.

## 2.3. Driver

Conductor con habilidad H (0-100). Atributos: Records y estadísticas.

### Relaciones y Cardinalidades

- Relación N:1 con Team (asociación)
- Relación 1:1 con Simulation (participación)
- Relación con Circuit vía simulaciones

### Reglas de Negocio

- **RN-DRV-01:** Habilidad H entre 0 y 100.
- **RN-DRV-02:** Asociado a equipo para simular.
- **RN-DRV-03:** Una simulación a la vez.
- **RN-DRV-04:** Acceso solo lectura a su perfil.
- **RN-DRV-05:** Estadísticas automáticas de historial.

## 2.4. Admin

Superusuario con acceso total al sistema.

### Relaciones y Cardinalidades

Acceso universal a todas las entidades.

### Reglas de Negocio

- **RN-ADM-01:** Acceso total a todas las operaciones.
- **RN-ADM-02:** Ejecuta simulaciones.
- **RN-ADM-03:** Acceso a Grafana global.
- **RN-ADM-04:** Gestión de usuarios.
- **RN-ADM-05:** Gestión de equipos, conductores, patrocinadores, partes, circuitos.

## 2.5. Part

Pieza para carros. Atributos: Part\_id, Stock, Price, Category (5 tipos), p,a,m (0-9).

### Relaciones y Cardinalidades

- Relación 1:5 con Car (5 partes/carro)
- Relación N:1 con Inventory (almacenamiento)

### **Reglas de Negocio**

- **RN-PRT-01:** Una de 5 categorías obligatorias.
- **RN-PRT-02:** p,a,m entre 0 y 9.
- **RN-PRT-03:** Stock  $\geq 0$ , validado.
- **RN-PRT-04:** Stock disminuye al comprar.
- **RN-PRT-05:** Solo comprable con stock.
- **RN-PRT-06:** Registro automático en inventario.

## **2.6. Sponsor**

Patrocinador con atributos: `Sponsor_id`, `Name`. Aportes: `Amount`, `Date`, `Description`.

**Relaciones y Cardinalidades** Relación M:N con Team (múltiples aportes a equipos).

### **Reglas de Negocio**

- **RN-SPN-01:** Múltiples aportes a múltiples equipos.
- **RN-SPN-02:** Registro de fecha, monto, descripción.
- **RN-SPN-03:** Presupuesto = suma de aportes.
- **RN-SPN-04:** Montos no negativos.
- **RN-SPN-05:** Registro permanente para auditoría.

## **2.7. Inventory**

Inventario del equipo. Atributos: `Inventory_id`, `Team_id`, `Part_id`, `Quantity`, `Adquisition_Date`.

### **Relaciones y Cardinalidades**

- Relación 1:1 con Team (un inventario/equipo)
- Relación N:1 con Part (almacenamiento)

### **Reglas de Negocio**

- **RN-INV-01:** Un inventario por equipo.
- **RN-INV-02:** Partes registradas automáticamente.
- **RN-INV-03:** Cantidad disminuye al instalar.
- **RN-INV-04:** Cantidad aumenta al reemplazar.
- **RN-INV-05:** Solo partes con cantidad  $> 0$ .
- **RN-INV-06:** Historial mantenido.

## 2.8. Car

Vehículo con atributos: `Car_id`, `P,A,M` (totales), `Is_Finalized`.

### Relaciones y Cardinalidades

- Relación con Engineer (armado)
- Relación N:1 con Team (máximo 2/equipo)
- Relación 1:5 con Part (5 partes)
- Relación 1:1 con Driver (conducción)
- Relación M:N con Simulation (participación)

### Reglas de Negocio

- **RN-CAR-01:** 5 partes (una por categoría).
- **RN-CAR-02:** Una parte por categoría.
- **RN-CAR-03:** Máximo 2 carros por equipo.
- **RN-CAR-04:** Solo partes del inventario.
- **RN-CAR-05:**  $P = \sum p_i$ ,  $A = \sum a_i$ ,  $M = \sum m_i$ .
- **RN-CAR-06:** Finalizado solo con 5 categorías.
- **RN-CAR-07:** Reemplazo devuelve parte.
- **RN-CAR-08:** Solo finalizados simulan.

## 2.9. Simulation

Ejecución de carrera. Atributos: `Simulation_id`, `Data_time`.

### Relaciones y Cardinalidades

- Relación N:1 con Circuit
- Relación M:N con Car
- Relación 1:1 con Driver

### Reglas de Negocio

- **RN-SIM-01:** Ejecutada en un circuito.
- **RN-SIM-02:** Solo carros finalizados.
- **RN-SIM-03:** Registra setup completo (5 partes con p,a,m).
- **RN-SIM-04:** Registra totales P,A,M y habilidad H.
- **RN-SIM-05:**  $V_{recta} = 200 + 3P + 0,2H - A$  (km/h)
- **RN-SIM-06:**  $V_{curva} = 90 + 2A + 2M + 0,2H$  (km/h)
- **RN-SIM-07:**  $Penalizacion = \frac{C \cdot 40}{1+H/100}$  (s)
- **RN-SIM-08:**  $T_{horas} = \frac{D_{rectas}}{V_{recta}} + \frac{D_{curvas}}{V_{curva}}$
- **RN-SIM-09:**  $T_{seg} = (T_{horas} \cdot 3600) + Penalizacion$
- **RN-SIM-10:** Ganador = menor  $T_{seg}$
- **RN-SIM-11:** Ranking por  $T_{seg}$
- **RN-SIM-12:** Persistencia completa.
- **RN-SIM-13:** Comparación de setups.

## 2.10. Team

Equipo con atributos: `Team_id`, `Name`, `Budget`, `Statistics`.

### Relaciones y Cardinalidades

- Relación 1:N con Engineer
- Relación 1:2 con Car
- Relación 1:1 con Inventory
- Relación 1:N con Driver
- Relación M:N con Sponsor

### Reglas de Negocio

- **RN-TEM-01:** Presupuesto = suma de aportes.
- **RN-TEM-02:** Presupuesto disminuye al comprar.
- **RN-TEM-03:** Valida saldo suficiente.
- **RN-TEM-04:** Máximo 2 carros.
- **RN-TEM-05:** Un inventario por equipo.
- **RN-TEM-06:** Registro de movimientos.
- **RN-TEM-07:** Al menos un ingeniero.

## 2.11. Circuit

Circuito con atributos: `Circuit_id`, `Total_distance` (D), `N_Curves` (C).

**Relaciones y Cardinalidades** Relación 1:N con Simulation.

### Reglas de Negocio

- **RN-CIR-01:**  $D > 0$  km.
- **RN-CIR-02:**  $C \geq 0$  curvas.
- **RN-CIR-03:** Parámetro global  $d_c$ .
- **RN-CIR-04:**  $D_{curvas} = C \cdot d_c$ .
- **RN-CIR-05:**  $D_{rectas} = D - D_{curvas}$ .
- **RN-CIR-06:**  $D_{rectas} \geq 0$ .

# Avance 2

*Crow's Foot + Esquema Inicial + Vistas Base*

## 1. Modelo Relacional y Schema Inicial

El segundo entregable transciona del modelo ER conceptual al modelo relacional, implementando el esquema inicial en SQL Server y desarrollando funcionalidades básicas. Incluye mapeo sistemático, normalización y creación de procedimientos almacenados preliminares.

### 1.1. Diagrama Crow's Foot

El diagrama Crow's Foot representa la notación lógica del modelo de datos, traduciendo relaciones conceptuales a una representación cercana a la implementación física. Especifica cardinalidades y direccionalidad para facilitar la creación de tablas.

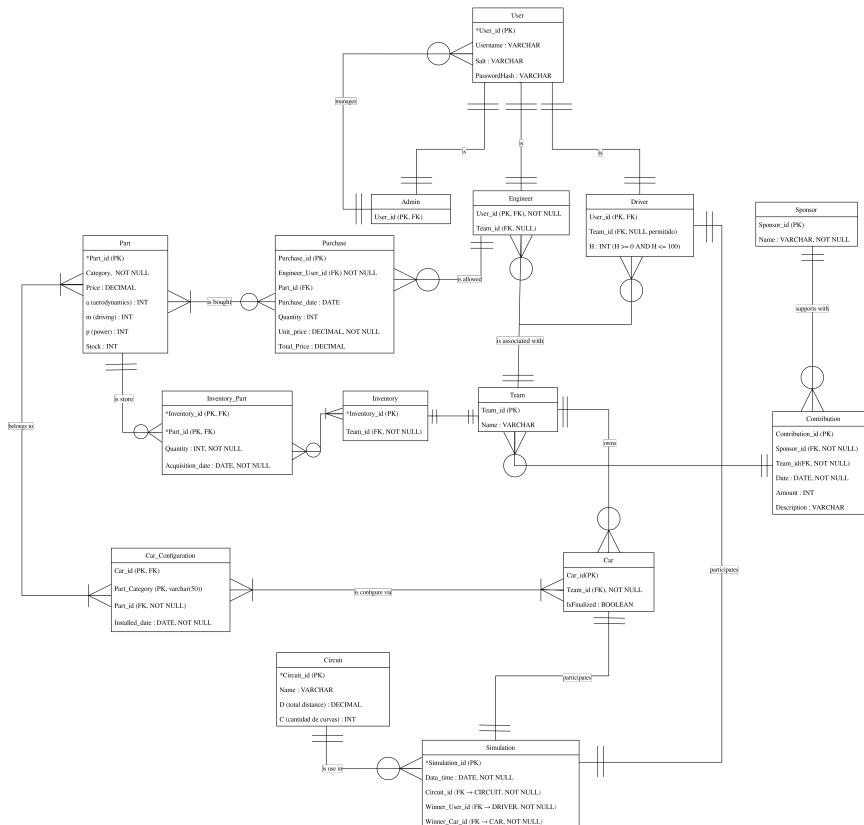
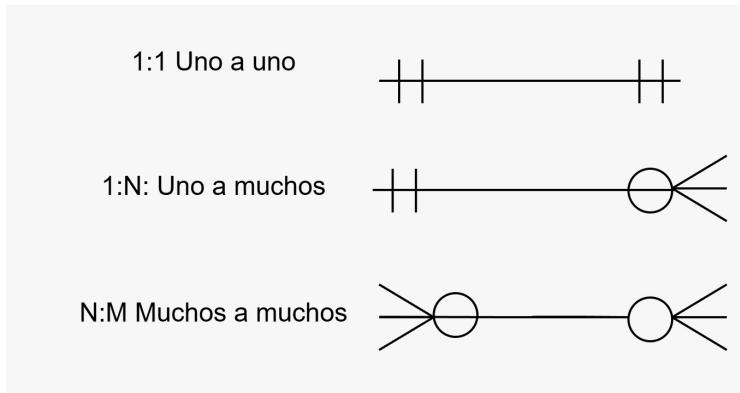


Figura 2: Diagrama Crow's Foot del modelo relacional.



**Figura 3:** Simbología del diagrama relacional.

## 1.2. Proceso de Mapeo del Modelo ER a Relacional

Para la adaptación del modelo ER a relacional se empleó el conocimiento adquirido durante las clases del curso, siguiendo un proceso algorítmico que garantiza la correcta traducción de las estructuras conceptuales a tablas relacionales. Este proceso se desarrolló en seis etapas principales, cada una enfocada en un tipo específico de estructura del modelo conceptual.

### 1.2.1. Mapeo de Entidades Fuertes

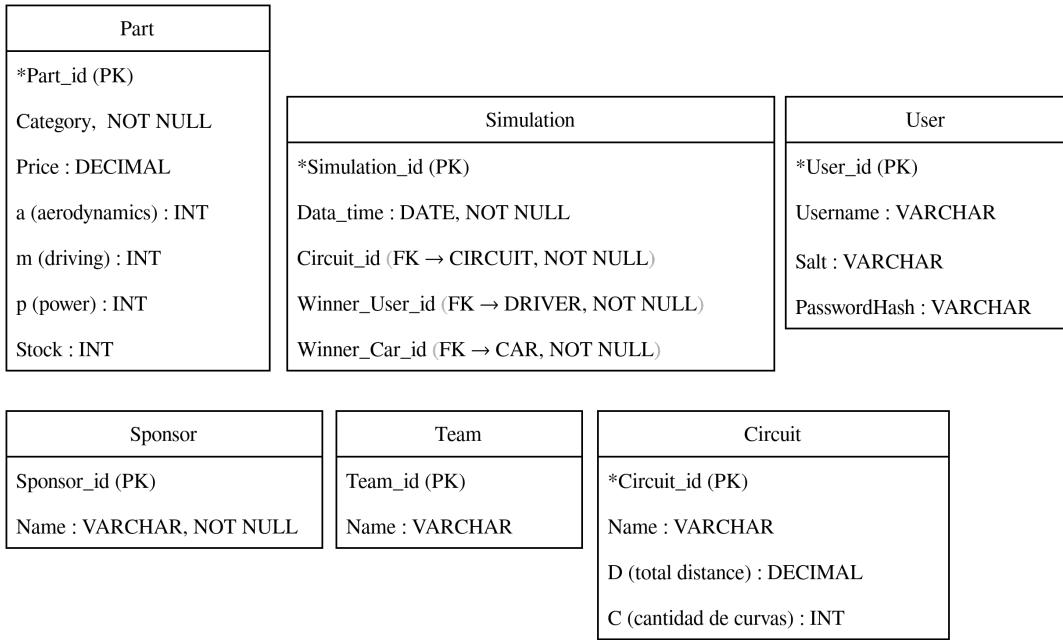
Se inició el proceso con el mapeo de las entidades fuertes, aquellas que poseen existencia independiente y no requieren de otras entidades para ser identificadas. Las entidades identificadas como fuertes fueron **USER**, **TEAM**, **SPONSOR**, **PART**, **CIRCUIT** y **SIMULATION**. Cada una de estas entidades se tradujo a una tabla relacional con su llave primaria simple y no compuesta, lo que facilita las operaciones de referencia y mantiene la independencia estructural que las caracteriza. Estas tablas constituyen la base del modelo, sobre las cuales se construyen las relaciones con el resto del sistema.

### 1.2.2. Mapeo de Entidades Débiles

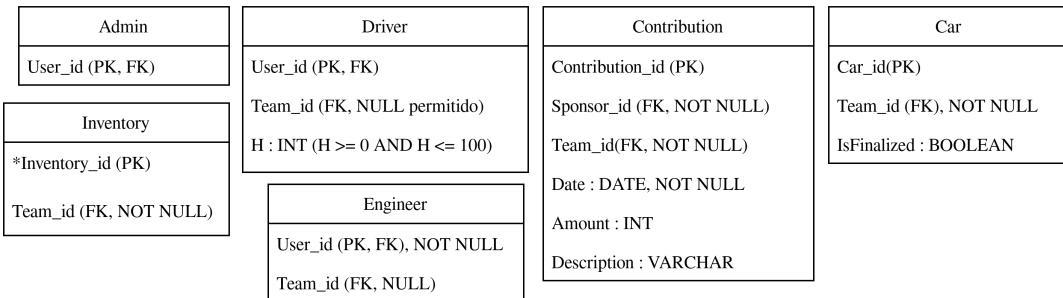
Posteriormente se procedió con el mapeo de las entidades débiles o dependientes, que requieren de la existencia de otras entidades para poder ser identificadas. Las entidades **ENGINEER**, **DRIVER**, **ADMIN**, **INVENTORY** y **CAR** fueron clasificadas como débiles debido a su dependencia existencial con sus respectivas entidades padre. En el caso de los roles de usuario (**ENGINEER**, **DRIVER**, **ADMIN**), estos heredan directamente la llave primaria de **USER**, estableciendo una relación de especialización. Por su parte, **INVENTORY** depende de **TEAM** mediante una relación uno a uno, mientras que **CAR** mantiene una dependencia directa con el equipo al que pertenece.

### 1.2.3. Mapeo de Relaciones Binarias

El proceso continuó con el mapeo de las relaciones binarias, clasificándolas según su cardinalidad. Para las relaciones uno a uno, como la existente entre **TEAM**

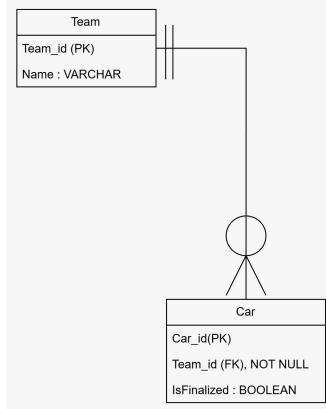


**Figura 4:** Entidades fuertes, resumen.



**Figura 5:** Entidades débiles, resumen.

e INVENTORY, se optó por incluir la llave foránea en la tabla con participación total, agregando además una restricción de unicidad para garantizar la cardinalidad. En el caso de las relaciones uno a muchos, se aplicó la técnica estándar de incluir la llave foránea en el lado muchos de la relación, como se observa en la relación entre TEAM y CAR, donde cada equipo puede poseer hasta dos carros. Las relaciones muchos a muchos requirieron la creación de tablas intermedias que almacenan las llaves de ambas entidades participantes, como es el caso de CONTRIBUTION para la relación entre SPONSOR y TEAM, PURCHASE para las compras de partes, INVENTORY\_PART para el almacenamiento de partes en inventarios, y CAR\_CONFIGURATION para la instalación de partes en los carros.



**Figura 6:** Ejemplo de relación entre TEAM y CAR.

### 1.3. Aplicación de Formas Normales

Durante todo el proceso de mapeo se aplicaron las formas normales necesarias para evitar redundancia de datos, anomalías de actualización y complejidad innecesaria en el modelo. Esta aplicación sistemática garantiza que el esquema resultante sea eficiente y mantenable a largo plazo.

#### 1.3.1. Primera Forma Normal (1NF)

La primera forma normal establece que todos los atributos deben ser atómicos, es decir, no pueden contener conjuntos de valores o estructuras complejas. Un ejemplo claro de la aplicación de esta forma normal se encuentra en la entidad PART, donde inicialmente se contempló almacenar el rendimiento como un atributo compuesto en formato **Performance** ( $p=5$ ,  $a=7$ ,  $m=3$ ). Sin embargo, esto violaría la atomicidad requerida por 1NF, por lo que se decidió descomponer este atributo en tres columnas independientes:  $p$ ,  $a$  y  $m$ , cada una almacenando un valor entero entre 0 y 9. De igual manera, los atributos compuestos del modelo conceptual fueron descompuestos en sus elementos simples durante el mapeo, asegurando que cada celda de la base de datos contenga únicamente valores atómicos.

#### 1.3.2. Segunda Forma Normal (2NF)

La segunda forma normal elimina las dependencias parciales, requiriendo que en tablas con llaves primarias compuestas, todos los atributos no clave dependan de la llave completa y no solo de parte de ella. Esta forma normal es particularmente relevante en las tablas intermedias que surgieron del mapeo de relaciones muchos a muchos. En la tabla INVENTORY\_PART, por ejemplo, la llave primaria está compuesta por **Inventory\_id** y **Part\_id**, y el atributo **Quantity** depende correctamente de ambas llaves, ya que representa cuántas unidades de una parte específica existen en un inventario específico. Una violación de 2NF habría ocurrido si se incluyera el atributo **Part\_Category** en esta tabla, pues este solo depende de **Part\_id** y no de la llave completa. Por esta razón, dicho atributo permanece correctamente ubicado en la tabla PART, donde pertenece.

### 1.3.3. Tercera Forma Normal (3NF)

La tercera forma normal elimina las dependencias transitivas, asegurando que ningún atributo no clave dependa de otro atributo no clave. La aplicación de esta forma normal se refleja principalmente en la decisión de no almacenar atributos derivados que pueden calcularse a partir de otros datos. En la tabla CAR, los totales de rendimiento P, A y M no se almacenan físicamente, ya que estos valores se derivan de la suma de los atributos correspondientes de las cinco partes instaladas en el carro. De igual manera, en la tabla TEAM, el atributo Budget no se almacena directamente, pues este se calcula como la suma de todos los aportes registrados en la tabla CONTRIBUTION. Un ejemplo de violación de 3NF habría sido incluir el atributo Team\_Name en la tabla CAR, ya que este depende de Team\_id (un atributo no clave desde la perspectiva de CAR) y no directamente de Car\_id. Esta información ya existe en la tabla TEAM, y duplicarla generaría redundancia y potenciales inconsistencias.

Como resultado de la aplicación sistemática de estas tres formas normales, se obtuvo un esquema de base de datos libre de redundancia significativa, con dependencias funcionales bien definidas y una estructura que facilita las operaciones de mantenimiento y actualización de datos. El modelo resultante es consistente, eficiente y preparado para soportar las operaciones del sistema sin anomalías de inserción, actualización o eliminación.

## 1.4. Schema Inicial en SQL Server

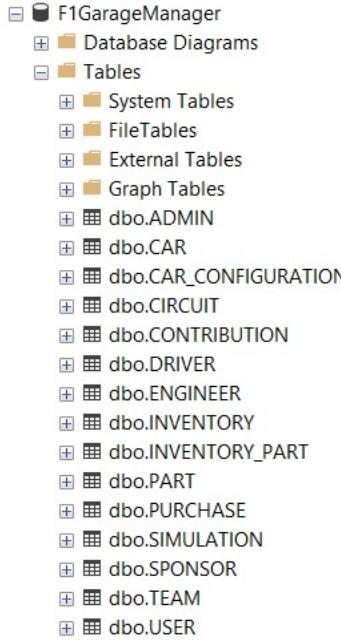
La implementación del esquema en SQL Server se realizó siguiendo una estrategia incremental que facilita la depuración y el mantenimiento del código. Se crearon scripts modulares organizados por funcionalidad, permitiendo que el equipo de desarrollo trabajara de forma paralela en distintos componentes del sistema.

El proceso de creación se dividió en cuatro etapas principales. Primero se estableció la base de datos F1GarageManager y se verificó la existencia del esquema dbo. Posteriormente se crearon las tablas de entidades fuertes sin restricciones de integridad referencial, estableciendo únicamente las llaves primarias y las restricciones de verificación sobre los atributos individuales. En una tercera etapa se procedió con la creación de las tablas de entidades débiles y las tablas intermedias para relaciones muchos a muchos, manteniendo nuevamente las restricciones de integridad referencial pendientes. Finalmente, mediante sentencias ALTER TABLE, se agregaron todas las llaves foráneas que establecen las relaciones entre las distintas tablas del sistema.

Esta aproximación de crear primero las tablas sin integridad referencial y agregar posteriormente las llaves foráneas mediante ALTER TABLE ofrece varias ventajas prácticas. Evita errores de dependencias circulares durante la creación, permite modificar fácilmente el orden de creación de las tablas sin preocuparse por restricciones, facilita la depuración al aislar problemas de estructura de problemas de relaciones, y mejora la legibilidad del código al separar claramente la definición de tablas de la definición de relaciones.

Para cada tabla se definieron explícitamente las llaves primarias utilizando la restricción PRIMARY KEY con nombres descriptivos mediante la cláusula

**CONSTRAINT.** Se empleó la funcionalidad `IDENTITY(1,1)` de SQL Server para la generación automática de valores de llaves primarias, garantizando unicidad sin intervención manual. Las restricciones de verificación se implementaron mediante cláusulas `CHECK` para garantizar la validez de los datos, como en el caso de los valores de rendimiento de las partes que deben estar entre 0 y 9, o las restricciones de categorías válidas en la tabla `PART`.



**Figura 7:** Tablas creadas en SQL Server Management Studio mostrando la estructura completa del esquema.

## 1.5. Stored Procedures Preliminares

Como parte de los requerimientos del entregable, se desarrollaron procedimientos almacenados preliminares que demuestran las operaciones fundamentales del sistema. Estos procedimientos, aunque básicos en su implementación inicial, establecen la estructura sobre la cual se construirán las funcionalidades completas en entregas posteriores.

### 1.5.1. Procedimiento para Cálculo de Presupuesto

El procedimiento `sp_GetTeamBudget` calcula el presupuesto disponible de un equipo mediante la suma de todos los aportes registrados por patrocinadores. Este procedimiento recibe como parámetro el identificador del equipo y retorna el total acumulado de contribuciones, representando el presupuesto disponible para compras de partes.

### 1.5.2. Procedimiento para Consulta de Inventario

El procedimiento `sp_GetTeamInventory` permite consultar las partes disponibles en el inventario de un equipo específico. Mediante operaciones de `JOIN` entre

las tablas INVENTORY, INVENTORY\_PART y PART, este procedimiento retorna el identificador de cada parte, su categoría y la cantidad disponible, proporcionando una vista consolidada del inventario del equipo.

#### 1.5.3. Procedimiento para Registro de Compras

El procedimiento `sp_RegisterPurchase` implementa la funcionalidad básica de registro de compras de partes. En esta versión preliminar, el procedimiento inserta un registro en la tabla PURCHASE con los datos del ingeniero que realiza la compra, la parte adquirida y la cantidad comprada. La versión actual utiliza valores simplificados para el cálculo de precios, los cuales serán reemplazados en entregas posteriores por la lógica completa que incluye validación de presupuesto, verificación de stock y actualización automática del inventario.

#### 1.5.4. Procedimiento para Consulta de Configuración

El procedimiento `sp_GetCarConfiguration` proporciona la configuración actual de un carro, mostrando qué parte está instalada en cada una de las cinco categorías obligatorias. Este procedimiento consulta directamente la tabla CAR\_CONFIGURATION y retorna las parejas de categoría y parte instalada, permitiendo visualizar el estado de armado del carro.

Estos procedimientos almacenados preliminares cumplen con el objetivo de demostrar la viabilidad de las operaciones centrales del sistema. Aunque su implementación actual es simplificada, establecen las interfaces y la estructura básica que será extendida en las siguientes entregas con validaciones completas, manejo de transacciones, control de errores y lógica de negocio completa según las reglas especificadas en el modelo conceptual.

## 2. Implementación Base del Frontend

Se desarrolló una versión preliminar del **Frontend** utilizando tecnologías como: **React**, **TypeScript**, **Vite** y **Tailwind CSS**. Esta implementación incluye las vistas base para las funcionalidades especificadas, con operaciones básicas funcionales.

**Nota.-** Se empleó el modelo de lenguaje GPT-4 como herramienta de asistencia técnica y creativa. Su contribución se centró principalmente en el diseño del sistema de diseño visual, abarcando la definición de la paleta de colores inspirada en la estética de la Fórmula 1, la configuración de los archivos: `tailwind.config.ts` para Tailwind CSS, `index.css` y algunos componentes de la interfaz gráfica.

Más información al respecto se puede encontrar en el código fuente.

## 2.1. Requerimientos y Tecnologías

Tecnología	Versión	Propósito
React	18.2.0	Biblioteca principal para construcción de interfaces de usuario
TypeScript	5.2.2	Tipado estático para mayor robustez y mantenibilidad del código
Vite	5.0.0	Herramienta de construcción de alta velocidad y servidor de desarrollo
Tailwind CSS	3.4.0	Framework de estilos utilitarios para diseño responsivo
React Router DOM	6.21.1	Sistema de enrutamiento y navegación entre páginas
React Hook Form	7.49.2	Manejo eficiente de formularios con validación integrada

Cuadro 1: Tecnologías principales utilizadas para la aplicación web.

## 2.2. Estructura del Proyecto

El código fuente sigue una arquitectura modular organizada en el directorio `src/`:

```
src/
  |- App.css
  |- App.tsx
  |- assets/
  |- components/
  |- hooks/
  |- index.css
  |- lib/
  |- main.tsx
  |- modules/
  |- vite-env.d.ts
```

`modules/` Contiene las páginas principales de la aplicación.

`components/` Aloja componentes reutilizables.

**Nota.-** Varios elementos individuales y *layouts* utilizados en el desarrollo del frontend se basan en plantillas y código de referencia obtenidos de los siguientes repositorios de componentes para Tailwind CSS:

<https://www.tailwind-kit.com/>

<https://tailgrids.com/>

`lib/` Incluye utilidades generales, constantes del sistema y funciones auxiliares.

`assets/` Contiene recursos estáticos como imágenes e iconos.

### 2.3. Módulos Implementados

Se desarrollaron los siguientes módulos funcionales que cubren algunas de las funcionalidades básicas del sistema:

Módulo	Descripción y Funcionalidad
<code>CarAssembly.tsx</code>	Armado de vehículos con cálculo de P, A, M mediante selección de 5 categorías de partes.
<code>Store.tsx</code>	Catálogo para visualización de las partes disponibles.
<code>Inventory.tsx</code>	Sección de inventario la cual presenta las partes disponibles en la tienda.
<code>Login.tsx</code>	Autenticación simulada para tres roles (Admin, Engineer, Driver).
<code>UserManagement.tsx</code>	Interfaz para administración de usuarios del sistema.
<code>Teams.tsx</code>	Sección para la gestión completa de equipos disponibles.
<code>Drivers.tsx</code>	Registro de conductores en el sistema, con su respectiva estadística H.
<code>Sponsors.tsx</code>	Vizualización de patrocinadores activos.
<code>DriverProfile.tsx</code>	Perfil de conductor en modo solo lectura, el cual muestra estadísticas personales.
<code>Analytics.tsx</code>	Estructura base preparada para integración futura con Grafana.
<code>NotFound.tsx</code>	Manejo de rutas inválidas con navegación de retorno a secciones principales del sistema.

**Cuadro 2:** Módulos del *frontend* implementados.

---

# Avance 3

React + Node.js + MSSQL-Server  
& Flujo Central Con Persistencia

---

## 1. Flujo de trabajo y distribución de responsabilidades

Para el desarrollo de **Progress 3**, el equipo adoptó una estrategia de dividir las tareas entre los integrantes, asignando a cada miembro un conjunto de responsabilidades. Esta organización permitió facilitar la integración progresiva entre el backend, la persistencia de datos y el frontend.

La distribución de tareas se estructuró en torno a cuatro personas principales, cada una dedicada a un módulo funcional específico del sistema, como se detalla a continuación.

### 1.1. Persona 1: Backend Base y Conexiones

La primera persona fue responsable de la construcción de la infraestructura base del servidor y la correcta comunicación con la base de datos.

Se realizó la configuración inicial del entorno de ejecución utilizando `Node.js` junto con el framework `Express`, estableciendo la estructura base del proyecto, el enrutamiento principal y los controladores iniciales. Asimismo, se configuró la conexión con `SQL Server` empleando el controlador oficial, asegurando una comunicación estable y eficiente entre el servidor y la base de datos.

Como parte de esta etapa, se definieron las variables de entorno necesarias para desacoplar la configuración sensible del código fuente, incluyendo credenciales de base de datos, puertos de ejecución y configuraciones de seguridad. Además, se configuraron aspectos fundamentales del servidor como el manejo de `CORS` y el puerto de escucha.

Finalmente, se implementaron pruebas iniciales de conexión a la base de datos para validar la correcta integración y garantizar la disponibilidad del sistema antes de avanzar con capas superiores de lógica de negocio.

### 1.2. Persona 2: Autenticación y Gestión de Sesiones

Esta segunda persona fue responsable del diseño e implementación de la autenticación del sistema, basada en sesiones persistentes mediante cookies.

Se desarrollaron puntos de acceso de inicio y cierre de sesión para validar las credenciales e iniciar y cerrar sesiones de forma segura. Se implementó un esquema de cifrado de contraseñas mediante `bcryptjs` para garantizar que estas nunca se almacenen en texto plano.

Además, se desarrolló un middleware de verificación de sesiones para validar la autenticidad de las solicitudes entrantes y restringir el acceso a los recursos protegidos. Este mecanismo garantiza que solo los usuarios autenticados puedan interactuar con los módulos sensibles del sistema.

Como parte del flujo de autenticación, se definieron vistas y rutas diferenciadas según el rol del usuario (Administrador, Ingeniero o Conductor), lo que permitió un panel dinámico que adapta la interfaz y las funcionalidades disponibles según los permisos asignados.

### **1.3. Persona 3: Módulo Financiero**

Esta tercera persona fue responsable del desarrollo del módulo financiero del sistema, que gestiona los patrocinadores, las contribuciones financieras y el cálculo del presupuesto disponible por equipo.

Se implementaron operaciones CRUD completas para la entidad Patrocinador, permitiendo a los usuarios registrar, modificar, ver y eliminar patrocinadores dentro del sistema. También se desarrolló la funcionalidad para registrar las contribuciones, asociando cada contribución con su respectivo patrocinador y equipo.

Una de las principales funciones de este módulo es calcular el presupuesto total del equipo, que se obtiene como la suma acumulada de todas las contribuciones registradas. Este cálculo se implementó tanto a nivel de consulta como mediante procedimientos almacenados, garantizando la consistencia y la reutilización lógica.

Finalmente, se desarrolló la página de la tienda, que permite a los usuarios ver las piezas disponibles para la compra, integrar la información financiera con el inventario y preparar el flujo de trabajo para futuras validaciones presupuestarias.

### **1.4. Persona 4: Módulo de Ensamblaje de Vehículos**

La cuarta persona se centró en el desarrollo del módulo de ensamblaje de vehículos, uno de los componentes más representativos del sistema desde una perspectiva funcional.

Se implementó la interfaz de ensamblaje de vehículos, que permite la selección de piezas para cada una de las cinco categorías obligatorias. Durante este proceso, se aplicaron validaciones estrictas para garantizar que solo se instalaran las piezas disponibles en el inventario y que no se duplicaran categorías.

Como parte del proceso de ensamblaje, se implementó el cálculo automático de los valores de rendimiento P, A y M. Estos valores se obtienen sumando los atributos individuales de las piezas instaladas. Este cálculo se actualiza dinámicamente a medida que se instalan o reemplazan piezas.

Además, se desarrolló la lógica necesaria para la instalación y el reemplazo de piezas, garantizando que las operaciones mantengan la consistencia del inventario, la correcta devolución de las piezas reemplazadas y la reducción de los niveles de stock al instalar nuevas piezas.

---

# Avance 4

Integración de Módulos, Correcciones Críticas y Preparación de Analítica

---

## 1. Organización del Trabajo y Estrategia de Desarrollo

Para el cuarto avance del proyecto se adoptó una estrategia de trabajo colaborativa mediante el uso del control de versiones mediante ramas independientes a partir de la rama `master`, la cual se mantuvo como versión estable del sistema.

El enfoque principal del Avance 4 se centró en consolidar la lógica del sistema, eliminar inconsistencias heredadas de entregables anteriores y preparar la arquitectura para la ejecución de simulaciones completas con persistencia y análisis de datos.

## 2. Correcciones Críticas del Sistema

### 2.1. Resolución del Problema de Autenticación (*Login Fantasma*)

Uno de los problemas más críticos detectados fue el denominado *login fantasma*, en el cual el sistema permitía autenticarse correctamente, pero la interfaz permanecía en un estado inconsistente hasta que el usuario recargaba manualmente la página. Este comportamiento generaba la percepción de que la autenticación había fallado, cuando en realidad la sesión se había creado correctamente.

Tras el análisis correspondiente, se determinó que el problema no se encontraba en la lógica del backend ni en la gestión de sesiones, sino en la falta de actualización del estado global de la aplicación frontend después del proceso de autenticación. La solución consistió en forzar la sincronización del estado de la aplicación tras el inicio de sesión, eliminando la necesidad de recargar manualmente la página. Con esta corrección, el flujo de autenticación quedó completamente funcional y consistente.

### 2.2. Corrección del Sistema de Encriptación de Contraseñas

Adicionalmente, se identificó un comportamiento inconsistente en el proceso de encriptación y validación de contraseñas entre distintos sistemas operativos. En entornos Windows, la validación se realizaba incorrectamente en texto plano, mientras que en sistemas Linux las comparaciones no se ejecutaban correctamente, impidiendo el inicio de sesión.

Este problema fue corregido unificando el proceso de hash y comparación de contraseñas mediante `bcrypt`, garantizando un comportamiento consistente e independiente del sistema operativo. Como resultado, el sistema de autenticación quedó completamente funcional en todos los entornos de desarrollo.

### 3. Verificación y Uso Correcto de Stored Procedures

Durante este avance se realizó una revisión exhaustiva del uso de procedimientos almacenados en las operaciones complejas del sistema, particularmente en el módulo de armado de vehículos. Este análisis reveló que parte de la lógica definida en SQL Server no estaba siendo utilizada correctamente, debido a la existencia de endpoints duplicados que coexistían desde etapas tempranas del desarrollo.

Se detectaron dos conjuntos de rutas distintas para operaciones relacionadas con el armado de carros, lo que provocaba que el frontend consumiera endpoints obsoletos que ejecutaban cálculos directos en lugar de invocar los procedimientos almacenados correspondientes. Esta duplicación fue eliminada, consolidando la arquitectura de rutas y asegurando que todas las operaciones críticas utilicen exclusivamente la lógica definida en la base de datos.

Como resultado, el módulo de armado ahora refleja fielmente los cálculos de rendimiento  $P, A, M$  definidos en los procedimientos almacenados, garantizando coherencia entre la capa de persistencia y la lógica de negocio.

Mercedes - Details				
Budget		Inventory		
Sponsors		Part	Category	Price
<b>\$14,710,000</b>	<b>\$10,000,000</b>	Low Drag Monza Spec	Aerodynamics_pkg	\$550,000
Sponsor 2	\$2,000,000	8-Speed Sequential Pro	Gearbox	\$420,000
Sponsor 4	\$5,000,000	Seamless Shift Gearbox	Gearbox	\$580,000
Sponsor 10		Mercedes-AMG M15 E Performance	Power_Unit	\$1,200,000
Drivers		Active Suspension Pro 2024	Suspension	\$450,000
hamilton	H: 95	Adaptive Dampers Pro	Suspension	\$380,000
rusell	H: 93	Pirelli P Zero Hard (C1)	Wheels	\$280,000
		Pirelli P Zero Soft (C3)	Wheels	\$220,000

**Figura 8:** Vista detallada de un equipo, mostrando presupuesto disponible, patrocinadores asociados, conductores registrados e inventario de partes con sus respectivas categorías, precios y stock.

## **4. Automatización de la Creación y Gestión de Usuarios**

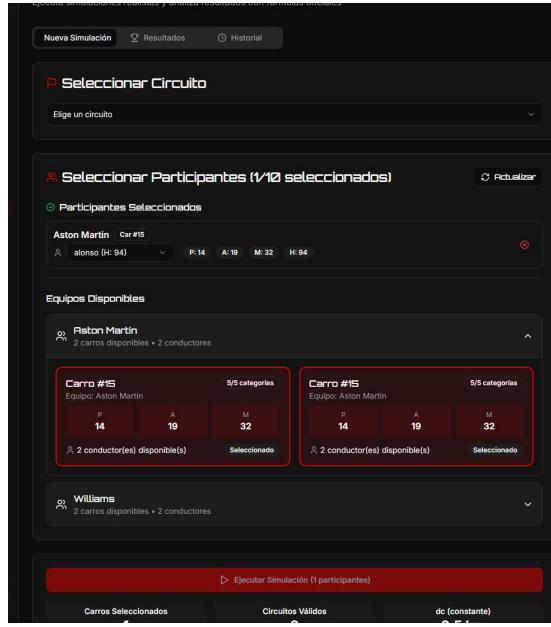
Se realizó una mejora significativa en el proceso de creación de usuarios, automatizando completamente la lógica mediante scripts y procedimientos almacenados. Este proceso permite ahora crear usuarios, asignar roles y establecer credenciales siguiendo una secuencia controlada y reproducible, reduciendo errores manuales y facilitando la administración del sistema.

Esta automatización sienta las bases para la futura implementación de interfaces administrativas que permitan la gestión completa de usuarios desde el frontend, incluyendo cambios de contraseña y actualización de información de perfil.

## **5. Desarrollo del Módulo de Circuitos y Simulaciones**

Como parte del avance funcional del sistema, se completó el módulo de circuitos, permitiendo registrar correctamente los parámetros necesarios para las simulaciones, tales como distancia total y número de curvas. Paralelamente, se inició el desarrollo del módulo de simulaciones, incorporando nuevas tablas y estructuras para soportar la persistencia de resultados, configuraciones de vehículos y rankings.

Aunque el módulo de simulaciones se encuentra en desarrollo activo, su arquitectura ya contempla la ejecución de las fórmulas definidas en el modelo conceptual, el cálculo de tiempos totales y la determinación del ganador, preparando el sistema para el cuarto entregable funcional.



**Figura 9:** Interfaz para la creación de una nueva simulación, donde el administrador selecciona el circuito, los carros participantes y los conductores asociados, validando configuraciones completas de cinco categorías antes de ejecutar la simulación.

## 5.1. Cálculo de distancias

$$D_{curvas} = C \cdot d_c$$

$$D_{rectas} = D - D_{curvas}$$

Debe cumplirse que:

$$D_{rectas} \geq 0$$

## 5.2. Velocidades (km/h)

$$V_{recta} = 200 + 3P + 0,2H - 1 \cdot A$$

$$V_{curva} = 90 + 2A + 2M + 0,2H$$

## 5.3. Penalización y tiempo total

Penalización (segundos):

$$Penalizacin = \frac{C \times 40}{1 + (H/100)}$$

**Tiempo (horas):**

$$Tiempo_{horas} = \left( \frac{D_{rectas}}{V_{recta}} \right) + \left( \frac{D_{curvas}}{V_{curva}} \right)$$

**Tiempo total (segundos):**

$$Tiempo_{segundos} = (Tiempo_{horas} \times 3600) + Penalizacin$$

## 5.4. Ganador

- El ganador es el carro con menor  $Tiempo_{segundos}$ .
- El orden final se determina ordenando tiempos de menor a mayor.
- La simulación debe persistirse para análisis posterior.

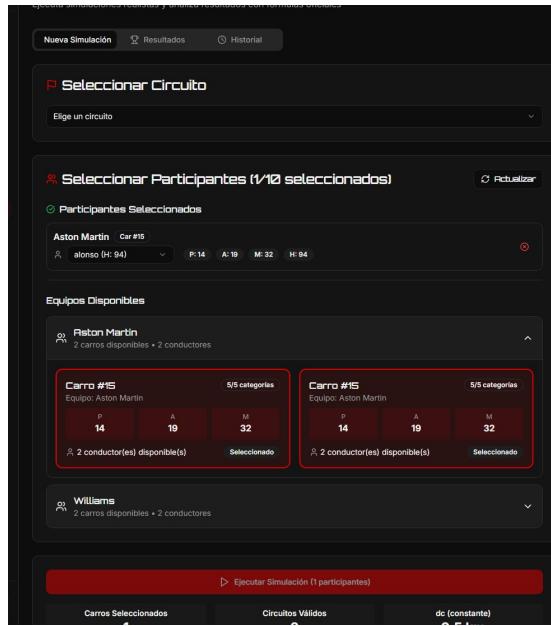
## 6. Preparación para Analítica y Visualización con Grafana

Finalmente, se inició una fase de investigación orientada a la integración de Grafana como herramienta de visualización y análisis de datos. Esta etapa se centró en comprender los mecanismos de conexión con SQL Server, el diseño de dashboards y la viabilidad de comparar configuraciones de vehículos y resultados de simulaciones.

Aunque esta funcionalidad aún no se encuentra implementada a nivel de código en la rama principal, la investigación realizada define claramente el camino para su integración futura, la cual se realizará en ramas separadas para no afectar la estabilidad del sistema.

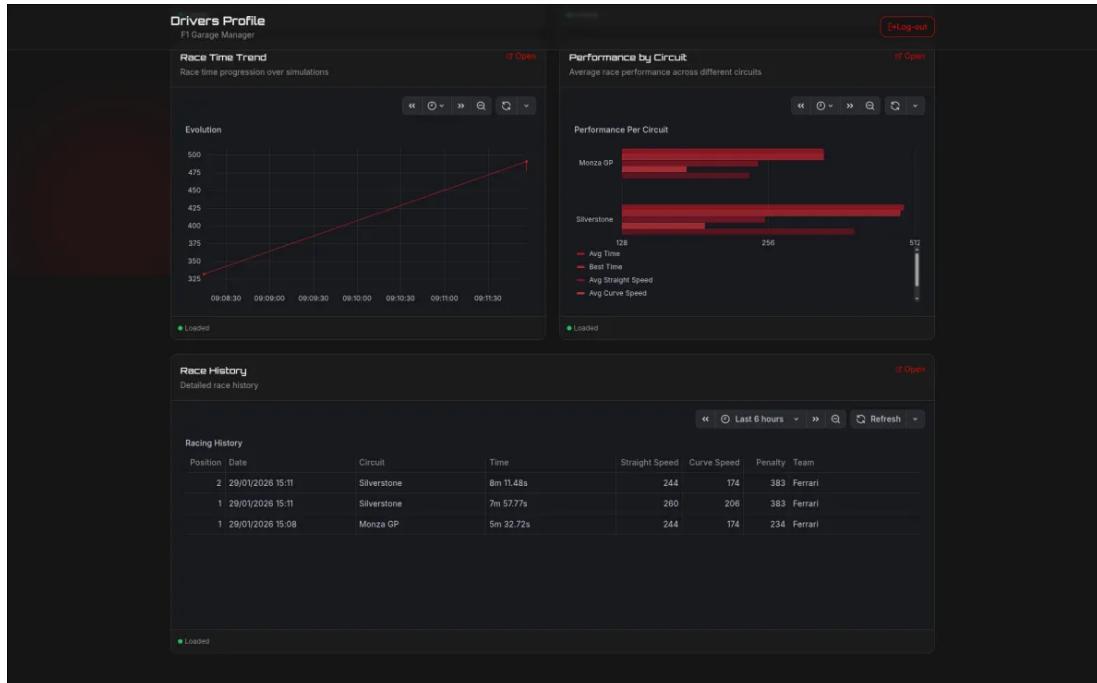
## 7. Estado Actual del Proyecto

Al cierre del Avance 4, el sistema presenta una base sólida y estable, con los errores críticos resueltos, una arquitectura de backend coherente, un uso correcto de procedimientos almacenados y una preparación clara para la implementación de simulaciones completas y analítica avanzada. La rama `master` contiene la versión estable del proyecto, mientras que los desarrollos en curso continúan de forma controlada en ramas independientes.

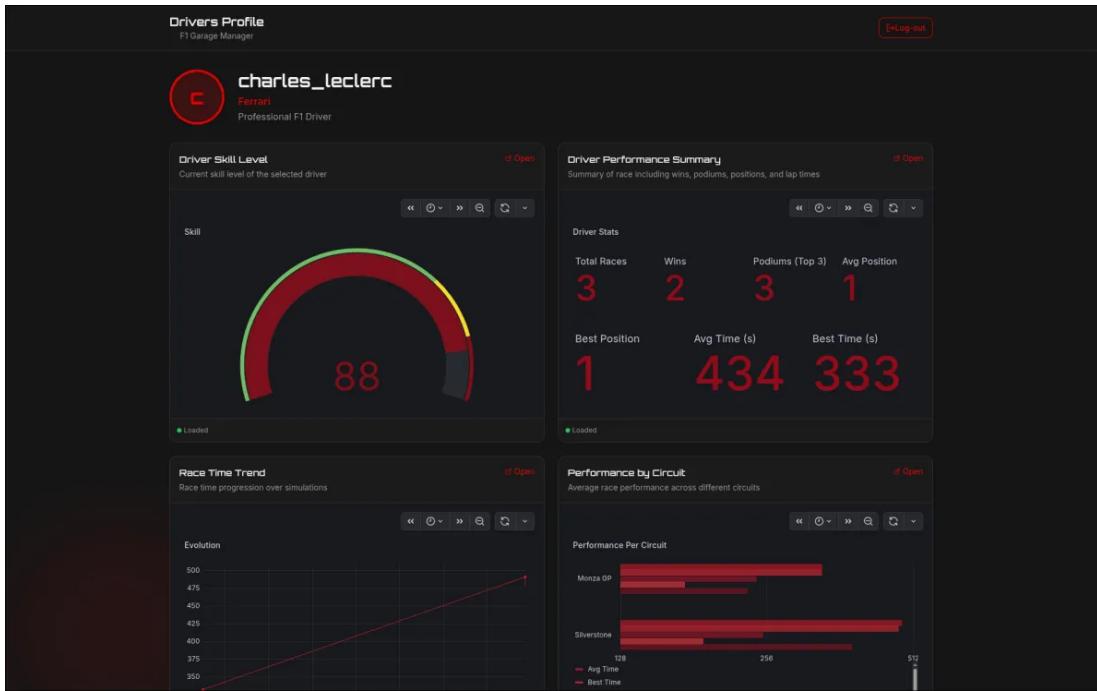


**Figura 10:** Interfaz para la creación de una nueva simulación, donde el administrador selecciona el circuito, los carros participantes y los conductores asociados, validando configuraciones completas de cinco categorías antes de ejecutar la simulación.

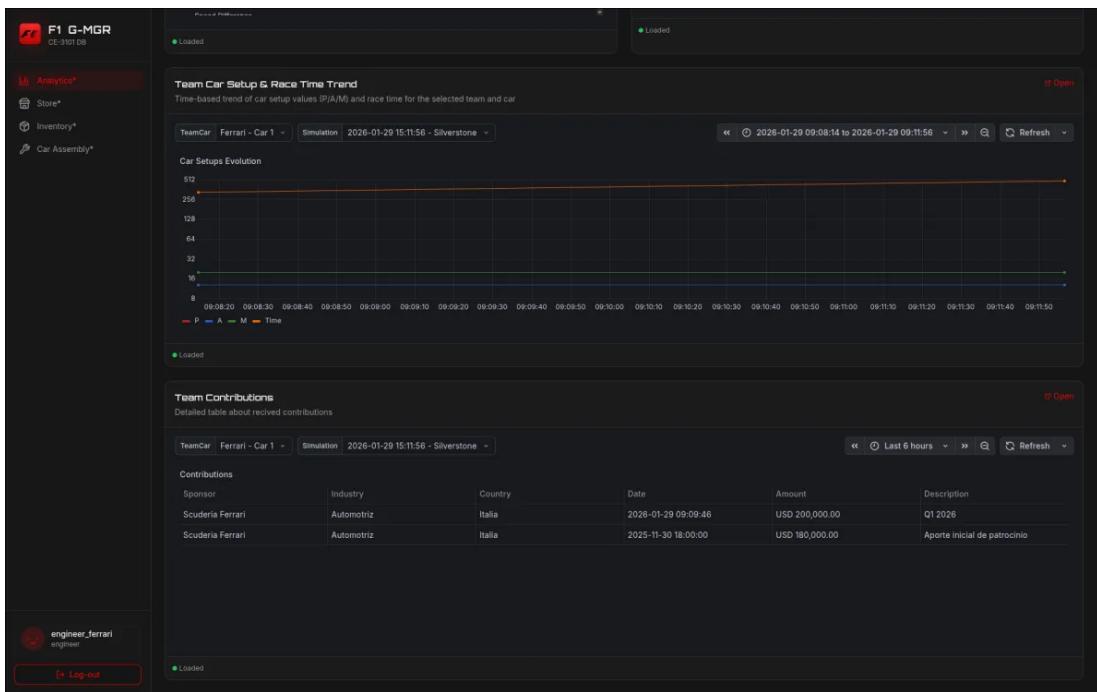
## 8. Análisis y Visualización de Datos (Dashboards)



**Figura 11:** Vista del perfil del conductor – Nivel de habilidad, estadísticas y rendimiento por circuito.



**Figura 12:** Vista de tendencias de tiempo de carrera y rendimiento por circuito.



**Figura 13:** Panel de analítica – Métricas generales del sistema y resumen de rendimiento por circuito.

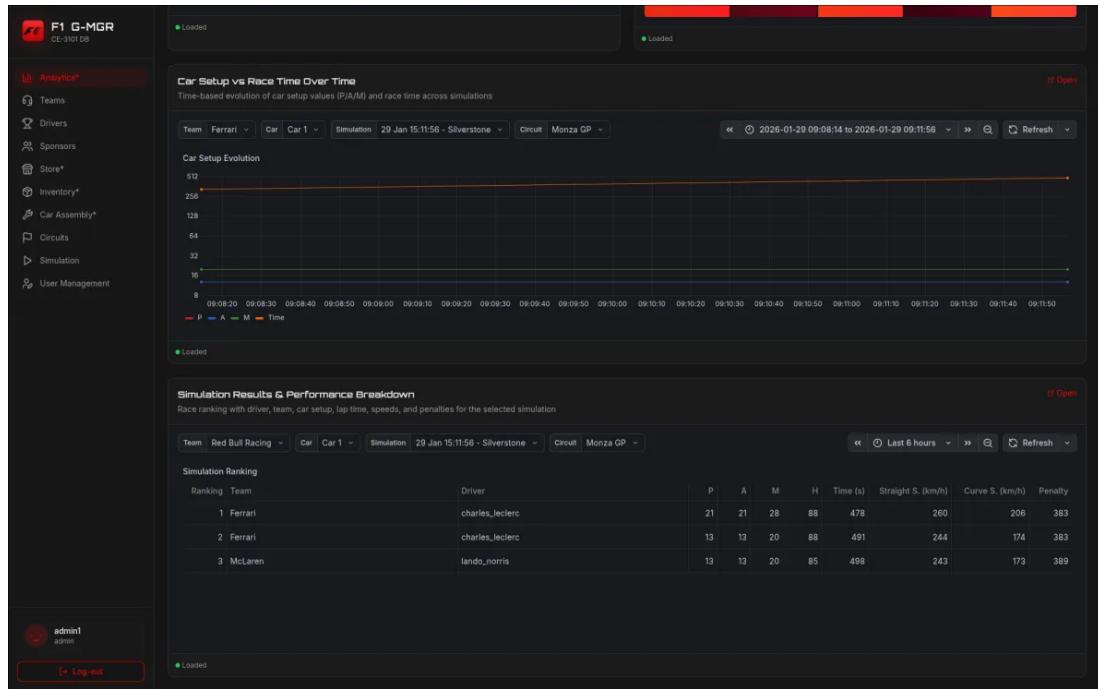


Figura 14: Evolución del setup del carro y comparación con tiempos de carrera.

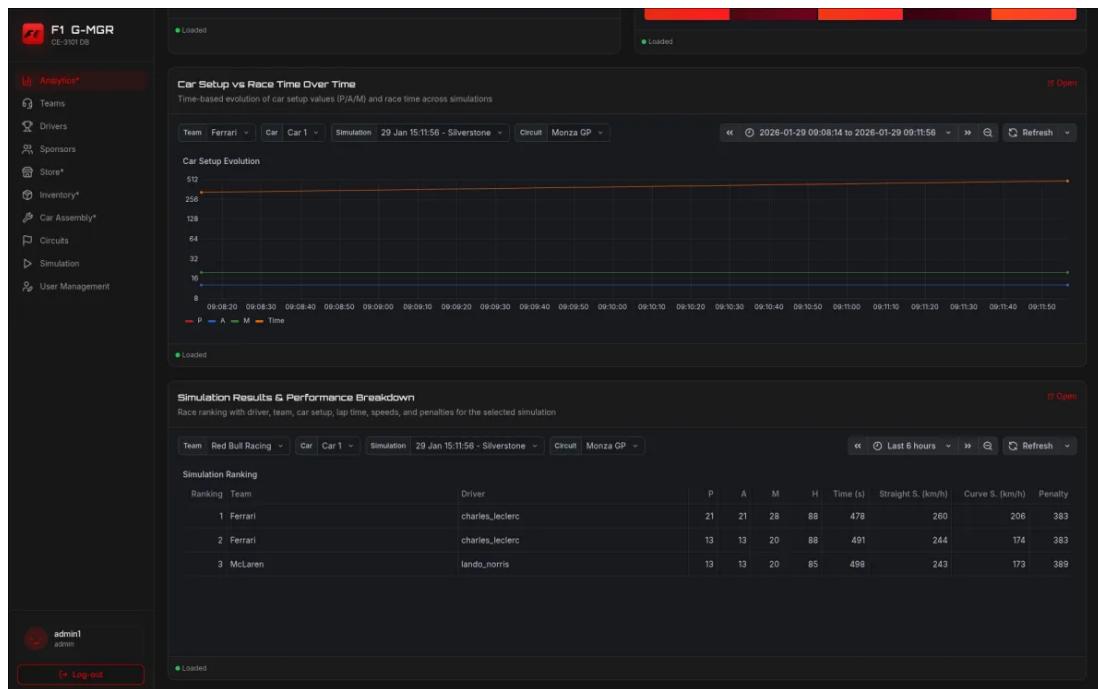
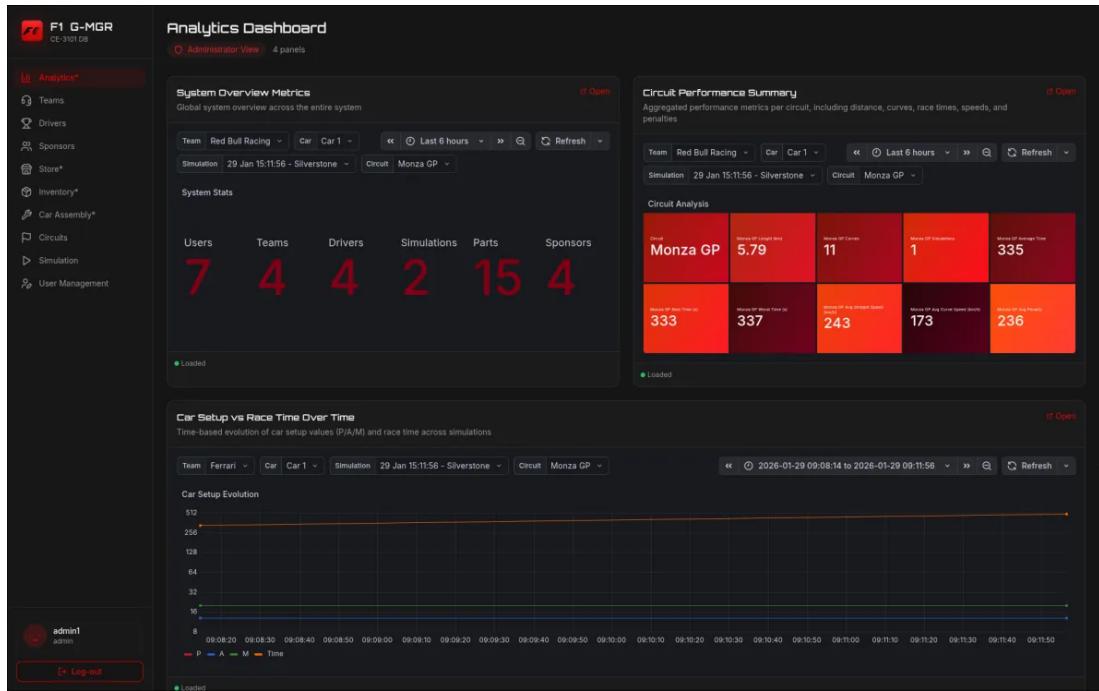


Figura 15: Resultados de simulación y desglose de performance por piloto y equipo.



**Figura 16:** Visualización de aportes de patrocinadores y evolución del setup del carro.