

Universidad Mariano Gálvez de Guatemala

Ingeniería en sistemas de información y ciencias de la computación

Ing. Mélvin Cali.



Erick Eduardo Camposeco Guerra 7690-14-14643

Angel Enrique Ibáñez Linares 7690-22-1911

Carlos Emanuel Del Cid Corado 7690-22-19541

FECHA: 09 /06/2025

Introducción del proyecto

En las diferentes ciudades modernas poseen un problema muy marcado que es la congestión vehicular y la contaminación ambiental que estos mismos provocan y se ha convertido en unos de los desafíos críticos a superar. La movilidad urbana en especial en las zonas céntricas y de alto tráfico, demanda soluciones que sean sostenibles, accesibles y eficientes. Una parte de la civilización no posee opciones rápidas y económicas para poder trasladarse a los diferentes puntos de la ciudad, lo que impacta negativamente su calidad de vida y su productividad diaria.

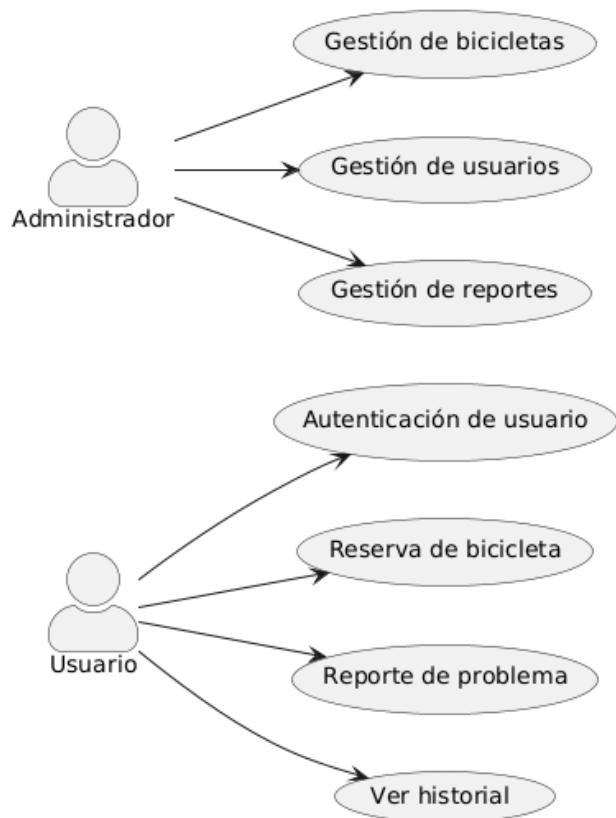
Objetivo general del sistema

Desarrollar una plataforma para la gestión de préstamo de bicicletas en terminales distribuidas en diferentes ubicaciones, garantizando una experiencia eficiente y segura para los usuarios.

EcoRide

Ecoride es una solución tecnológica cuyo principal propósito es poder mejorar la movilización urbana mediante un sistema de préstamos de bicicletas que sean accesibles a través de una plataforma web y que permita a los diferentes usuarios registrarse de una manera gratuita, reservar bicicletas desde terminales disponibles y devolverlas de forma eficiente. Además, el sistema cuenta con la capacidad de poder generar códigos QR para poder validar la reserva en las distintas terminales, incluirá reportes de fallas y una herramienta que garantiza la máxima administración del servicio para que sea confiable, seguro y sostenible.

Diagrama de casos de uso:



Casos De Uso

Caso de Uso: Autenticar usuario

Actor: Usuario

Descripción: Permite a un usuario registrarse o iniciar sesión en el sistema.

Precondición: El usuario debe tener una cuenta (si inicia sesión).

Postcondición: El usuario accede a su perfil.

Flujo principal:

- - Usuario ingresa correo y contraseña.
- - El sistema valida las credenciales.

Excepciones:

- - Datos inválidos → mensaje de error.

Caso de Uso: Reserva bicicleta

Actor: Usuario

Descripción: Permite al usuario seleccionar una bicicleta disponible desde una terminal.

Precondición: El usuario debe estar autenticado.

Postcondición: La bicicleta queda reservada con código QR/PIN.

Flujo principal:

- - Selecciona terminal origen.
- - Selecciona bicicleta y confirma.

Excepciones:

- - No hay bicicletas disponibles.

Caso de Uso: Reporte de problema

Actor: Usuario

Descripción: Permite enviar reportes de fallas o incidencias.

Precondición: Usuario autenticado.

Postcondición: Problema registrado.

Flujo principal:

- - Elige opción de reporte.
- - Llena formulario y lo envía.

Excepciones:

- - Fallo al enviar el formulario.

Caso de Uso: Consulta de historial

Actor: Usuario

Descripción: Permite al usuario ver su historial de préstamos.

Precondición: Usuario autenticado.

Postcondición: Historial mostrado.

Flujo principal:

- - Selecciona opción 'Ver historial'.
- - Sistema muestra lista con fechas y terminales.

Excepciones:

- - No hay registros.

Caso de Uso: Gestión usuarios

Actor: Administrador

Descripción: Permite activar, suspender o verificar cuentas.

Precondición: Administrador autenticado.

Postcondición: Usuario actualizado.

Flujo principal:

- - Selecciona opción de gestión.
- - Realiza acción sobre usuario.

Excepciones:

- - Usuario no existe.

Caso de Uso: Gestión bicicletas

Actor: Administrador

Descripción: Permite añadir, editar o eliminar bicicletas.

Precondición: Administrador autenticado.

Postcondición: Información actualizada.

Flujo principal:

- - Abre módulo de bicicletas.
- - Añade, edita o elimina registro.

Excepciones:

- - Bicicleta ya registrada o no existe.

Caso de Uso: Gestionar reportes

Actor: Técnico

Descripción: Revisa y resuelve reportes de fallas.

Precondición: Técnico autenticado.

Postcondición: Reporte resuelto.

Flujo principal:

- - Consulta lista de reportes.
- - Marca como resuelto.

Excepciones:

- - No hay reportes o sistema falla al guardar.

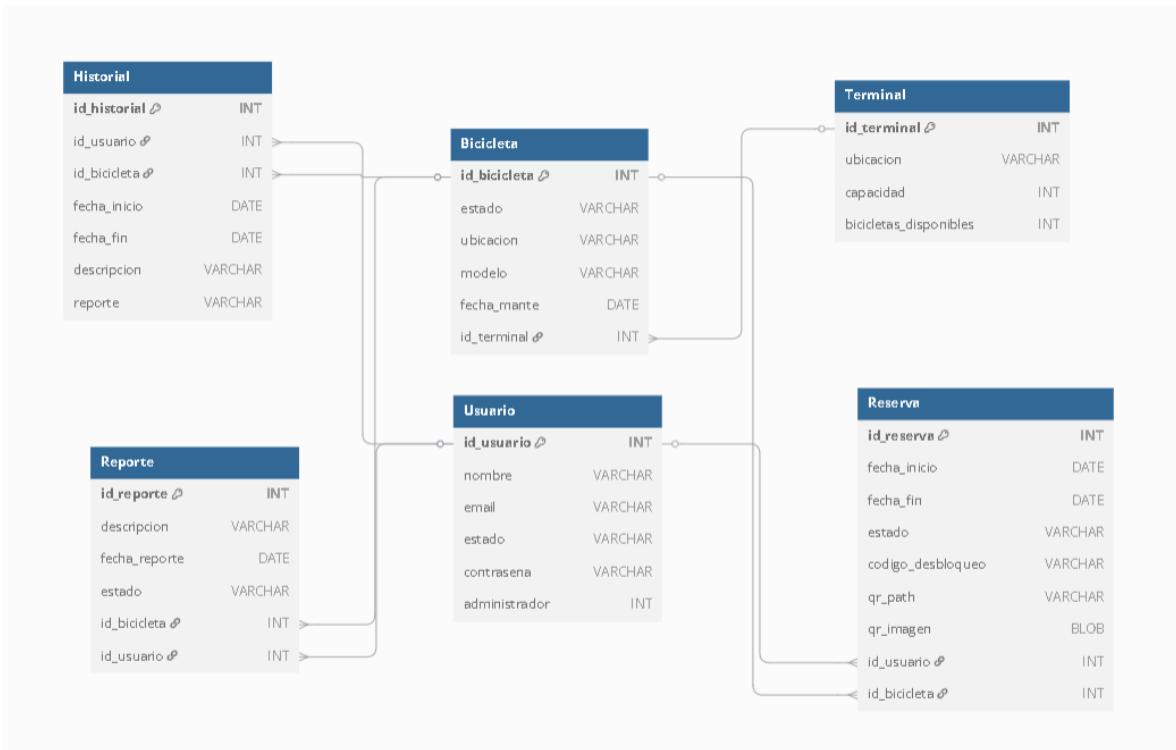
Requerimientos del sistema

ID	Descripción	Entradas	Salidas	Comportamiento Esperado	Excepciones y Errores
RF01	El sistema debe permitir a los usuarios registrarse e iniciar sesión	Nombre, correo, contraseña	Acceso al sistema, datos del perfil	El usuario se registra o inicia sesión correctamente según su rol	Usuario o contraseña incorrectos
RF02	El usuario podrá reservar una bicicleta desde una terminal de origen	Terminal origen, terminal destino, ID usuario	Confirmación de reserva, código QR	El sistema verifica disponibilidad, genera QR, actualiza estado de bicicleta y terminal	No hay bicicletas disponibles en la terminal seleccionada
RF03	El usuario podrá devolver una bicicleta en una terminal habilitada	ID de usuario (localStorage), QR/código reserva	Confirmación de devolución	Al devolver, se actualiza la reserva, la bicicleta vuelve a estado 'activo' y se suma la disponibilidad	No se encontró bicicleta activa o QR inválido
RF04	El usuario podrá reportar un problema tras devolver la bicicleta	ID bicicleta, ID usuario, descripción	Confirmación de reporte	El sistema almacena el reporte, verifica si la bicicleta debe quedar inactiva según acumulación de reportes	Error al enviar reporte, error de conexión
RF05	El sistema permitirá a los administradores visualizar y suspender usuarios	ID usuario, acción a realizar	Estado actualizado del usuario	El administrador podrá cambiar estado entre activo/suspendido	Usuario no encontrado o error al actualizar
RF06	El usuario podrá consultar la disponibilidad de bicicletas en terminales	Terminal seleccionada	Lista de bicicletas disponibles por terminal	El sistema muestra en tiempo real cuántas bicicletas están disponibles en la terminal seleccionada	Error de conexión con la base de datos o sin resultados

Requerimientos no funcionales

Categoría	Descripción
Rendimiento	El sistema debe responder en un tiempo máximo de 2 segundos para operaciones básicas (reserva, login y etc...). Debe ser capaz de atender al menos 100 usuarios concurrentes sin degradación perceptible en el rendimiento.
Seguridad	El acceso al sistema estará protegido mediante autenticación con Google a través de Google cloud. Los datos estarán cifrados y no se almacenarán credenciales en el cliente.
Escalabilidad	El sistema debe poder adaptarse al crecimiento de usuarios y terminales sin rediseño. Google cloud permite escalar automáticamente según la demanda.
Usabilidad	La interfaz será intuitiva, accesible y adaptativa. Funcionará correctamente en computadoras, tablets y smartphones.
Mantenibilidad	El código será modular y organizado, facilitando comprensión, depuración y mantenimiento. Se aplicarán buenas prácticas en HTML, CSS y JavaScript.
Compatibilidad	El sistema será compatible con navegadores modernos: Chrome, Firefox, Edge y Safari, y funcionará en Windows, Android e iOS.
Legalidad	El sistema respetará la privacidad del usuario. La información personal será mínima y usada sólo para autenticación.

Diagrama Entidad y relación



Historia de usuario

HU1 - Reserva de Bicicleta

Como: Usuario registrado

Quiero: Reservar una bicicleta en una terminal de origen

Para: Poder usarla y devolverla en una terminal de destino.

Criterios de Aceptación: Mostrar terminales con bicicletas disponibles.

- Permitir seleccionar origen/destino.
- Generar un código QR/PIN único por reserva.

HU2 - Historial de Préstamos

Como: Usuario registrado

Quiero: Ver mi historial de préstamos

Para: Saber cuánto he usado el servicio y mis viajes anteriores.

Criterios de Aceptación:

- Mostrar lista de préstamos con fechas, tiempos y terminales.
- Incluir detalles como penalizaciones (si aplican).

HU3 - Reporte de Problemas

Como: Usuario registrado

Quiero: Reportar problemas técnicos o físicos de una bicicleta

Para: Ayudar a mantener el servicio en buen estado.

Criterios de Aceptación:

- Permitir adjuntar fotos/descripción del problema.
- Notificar al técnico automáticamente.

HU4 - Monitoreo de Bicicletas (Admin)

Como: Administrador

Quiero: Ver el estado y ubicación en tiempo real de todas las bicicletas

Para: Gestionar redistribuciones y mantenimiento.

Criterios de Aceptación:

- Mostrar mapa con ubicaciones GPS actualizadas.
- Filtrar por estado (disponible/en uso/mantenimiento).

HU5 - Mantenimiento Correctivo (Técnico)

Como: Técnico

Quiero: Recibir y resolver reportes de fallas

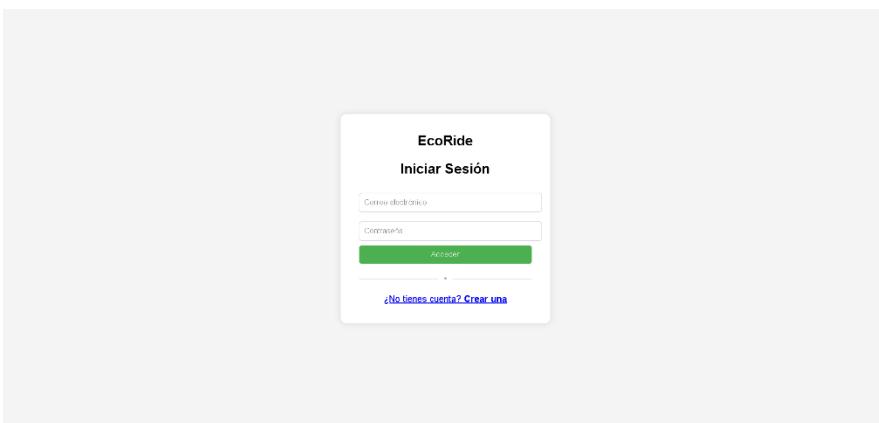
Para: Actualizar el estado de la bicicleta a "disponible" o "en reparación".

Criterios de Aceptación:

- Marcar reportes como "en progreso" o "resueltos".
- Notificar al usuario si reportó el problema.

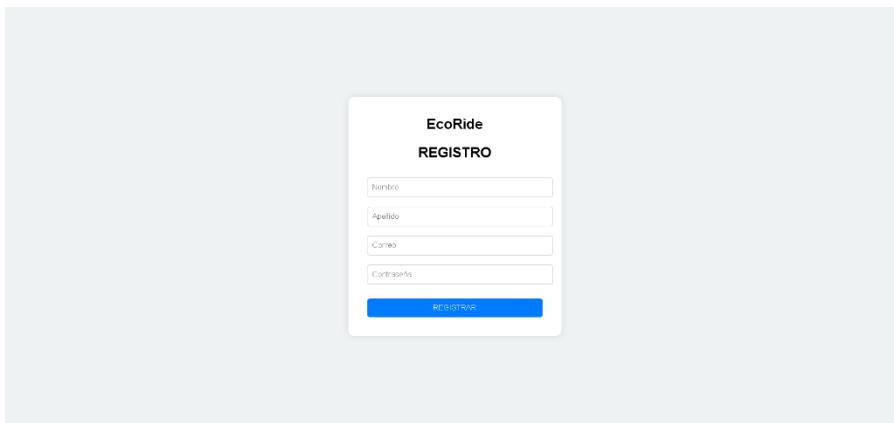
Prototipos

Login:



The login form is titled "EcoRide Iniciar Sesión". It contains fields for "Correo electrónico" and "Contraseña", followed by a green "Acceder" button. Below the buttons is a link "¿No tienes cuenta? Crear una".

Crear usuario:



The registration form is titled "EcoRide REGISTRO". It includes fields for "Nombre", "Apellido", "Correo", and "Contraseña", each with its own input box. Below these fields is a blue "REGISTRAR" button.

Perfil de usuario:

The screenshot shows the user profile section of the EcoRide application. At the top, there's a teal header bar with the logo 'EcoRide' on the left and navigation links 'Reservar', 'Devolver', 'Historial', and 'Perfil' on the right. Below the header is a large, light gray rectangular area containing a white card. The card has a title 'Perfil del Usuario' at the top. Underneath the title, there are four lines of text: 'Nombre: Juan Pérez', 'Correo: juan@example.com', and 'Estado: Activo'. At the bottom of the card is a blue rectangular button with the text 'CERRAR SESIÓN'.

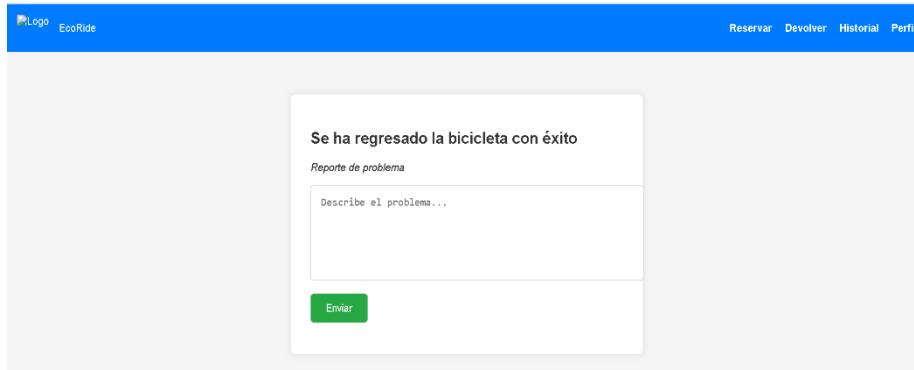
Reserva:

The screenshot shows the 'Reservar Bicicleta' (Reserve Bicycle) page. It features a light gray background with a central white card. At the top of the card, the title 'Reservar Bicicleta' is displayed. Below the title are two dropdown menus: 'Origen' (set to 'Terminal Central') and 'Destino' (set to 'Terminal Central'). To the right of these dropdowns is a blue rectangular button labeled 'Confirmar Reserva'.

Devolver:

The screenshot shows a confirmation dialog box. At the top, it asks the question '¿Deseas devolver la bicicleta?' (Do you want to return the bicycle?). Below the question, there is a small explanatory text: 'Cuando hayas asegurado la bicicleta en una terminal, presiona el botón.' (Once you have secured the bicycle at a terminal, press the button). At the bottom of the dialog is a green rectangular button labeled 'Confirmar Devolución'.

Reporte:



Historial:

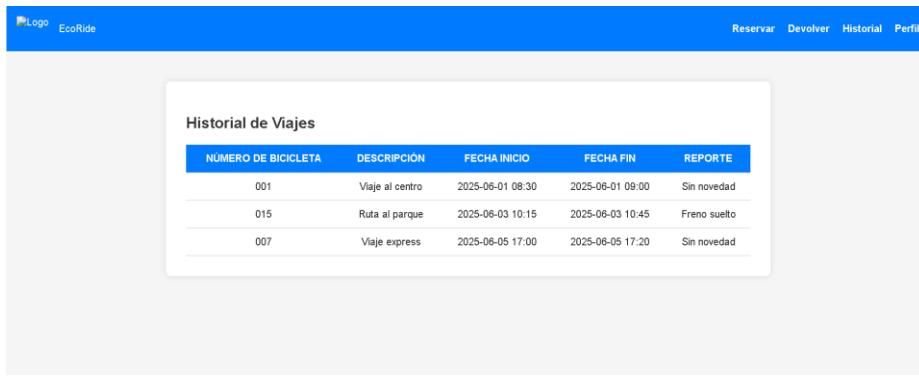
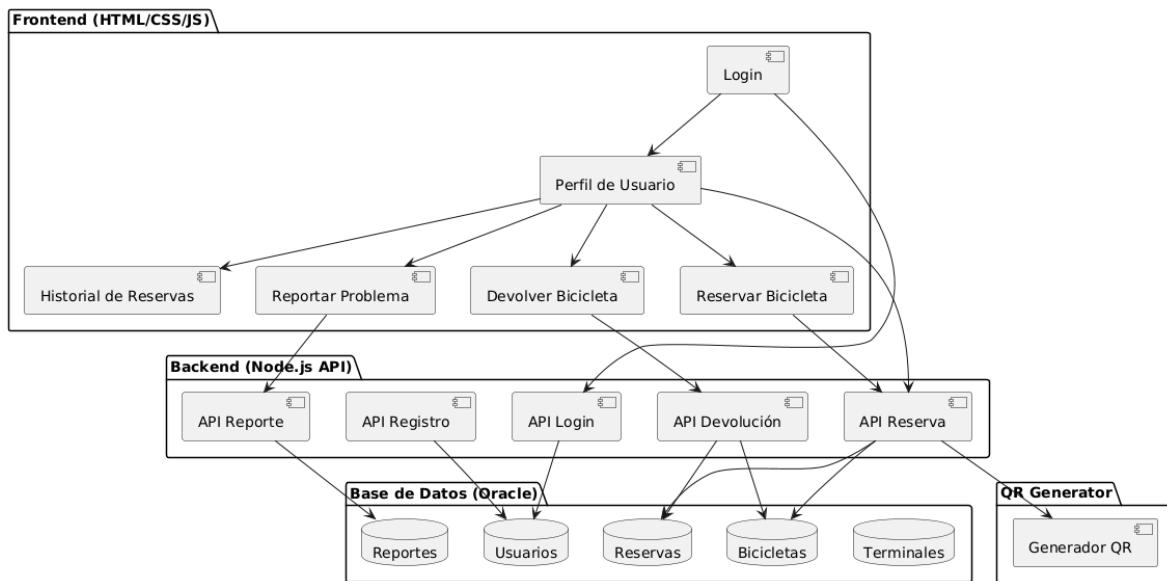


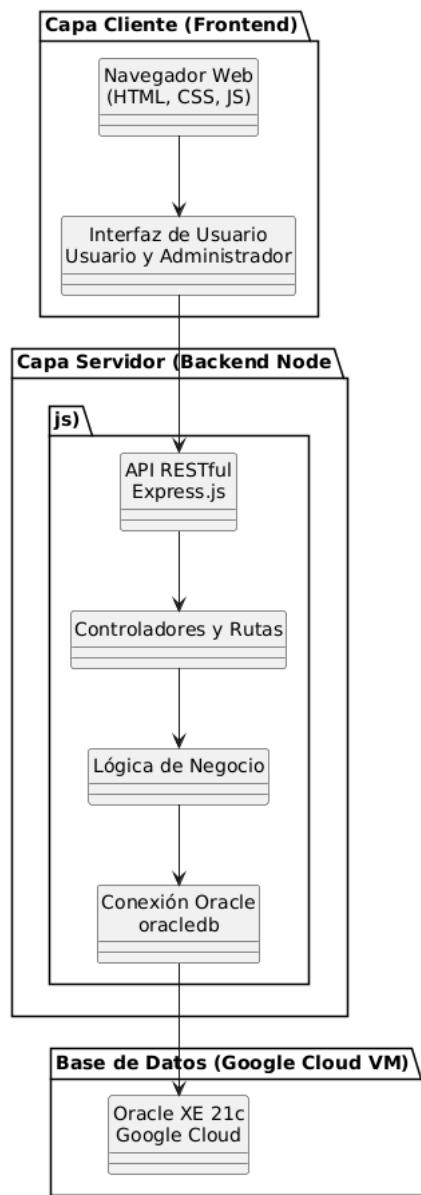
Diagrama de componentes:



Definición de Tecnologías

Componente	Tecnología	Descripción
Frontend	HTML5, CSS3, JavaScript	Interfaz gráfica del sistema para usuarios. Diseño responsivo.
Backend	Node.js (Express)	Lógica del servidor, rutas API RESTful.
Base de Datos	Oracle Database 21c XE	Almacena datos de usuarios, bicicletas, reservas, reportes y terminales.
Servidor Web	Nginx	Servidor HTTP para publicar frontend y redirigir peticiones a la API.
QR Generator	qrcode (librería Node.js)	Generación de códigos QR para confirmar reservas de bicicletas.
Sistema Operativo	Oracle Linux (en VPS de Google Cloud)	Base sobre la que corren los contenedores y servicios web del proyecto.
Hosting Dominio	Hostinger + Namecheap	Dominio personalizado y publicación web.
Editor de Código	Visual Studio Code	Desarrollo del frontend y backend del sistema.
Control de versiones	Git (GitHub opcional)	Para control de cambios durante el desarrollo.
Herramientas de prueba	Navegador web + consola del navegador	Para verificar funcionamiento del sistema y depuración.

Diagrama de arquitectura:



Patrones de diseño aplicados

Patrón MVC (Modelo - Vista - Controlador)

Aplicado en: Arquitectura general del sistema.

Explicación: Express con estructura estricta de carpetas models/, views/, controllers/, el sistema sigue la lógica del patrón MVC:

- **Modelo:** Oracle DB como fuente de datos (tablas, triggers, procedimientos).
- **Vista:** Archivos HTML y CSS como interfaz para usuarios.
- **Controlador:** Endpoints en app.js que controlan la lógica de negocio.

Patrón Singleton (conexión a base de datos)

Aplicado en: Manejador de conexión a Oracle.

Explicación: Se reutiliza la instancia de conexión a la base de datos mediante oracledb.getConnection(...), evitando múltiples conexiones simultáneas innecesarias.

Patrón Módulo

Aplicado en: Inclusión de librerías y organización de lógica.

Explicación: Se usa require() para importar módulos como express, oracledb, qrcode, encapsulando su funcionalidad y evitando variables globales.

Patrón de Fachada (Backend)

Aplicado en: Endpoints RESTful.

Explicación: Cada endpoint en el backend actúa como una "fachada" que abstrae al frontend de la complejidad del acceso a la base de datos, validaciones y lógica empresarial.

Patrón Transaction Script

Aplicado en: Reserva

Explicación: Este es el patrón más explícito aquí. Cada endpoint actúa como un "script de transacción", es decir, un bloque de lógica que se ejecuta paso a paso para realizar una operación de negocio.

Ejemplo en /api/reservar:

- Verifica si el usuario ya tiene una reserva activa.
- Busca una bicicleta disponible.
- Inserta la reserva.
- Actualiza el estado de la bicicleta.
- Actualiza el número de bicicletas disponibles.

Cada endpoint sigue este patrón, procedimental, centrado en la lógica paso a paso.

Patrón Repository (implícito)

Aplicado en: Código/sintaxis

Explicación: Este patrón encapsula el acceso a los datos. El código que accede a la base de datos está contenido en cada endpoint.

Patrón Controller (implícito)

Aplicado en: Código/sintaxis

Explicación: Cada endpoint en Express (app.get, app.post, etc.) actúa como un controlador que recibe una solicitud, procesa la lógica de negocio, y devuelve una respuesta.

Este es un Controlador RESTful.

Diccionario de datos

Tabla/Elemento	Campo	Tipo	Descripción
Usuario	id_usuario	NUMBER	Identificador único del usuario (PK)
Usuario	nombre	VARCHAR2(100)	Nombre completo del usuario
Usuario	email	VARCHAR2(100)	Correo electrónico del usuario
Usuario	estado	VARCHAR2(20)	Estado del usuario (activo/inactivo)
Usuario	contrasena	VARCHAR2(100)	Contraseña encriptada del usuario
Terminal	id_terminal	NUMBER	Identificador único del terminal (PK)
Terminal	ubicacion	VARCHAR2(100)	Ubicación física del terminal
Terminal	capacidad	NUMBER	Capacidad máxima de bicicletas en el terminal
Terminal	bicicletas_disponibles	NUMBER	Cantidad actual de bicicletas disponibles
Bicicleta	id_bicicleta	NUMBER	Identificador único de la bicicleta (PK)
Bicicleta	estado	VARCHAR2(20)	Estado de la bicicleta (activo/inactivo)
Bicicleta	ubicacion	VARCHAR2(100)	Ubicación actual de la bicicleta
Bicicleta	modelo	VARCHAR2(50)	Modelo de la bicicleta
Bicicleta	fecha_mante	DATE	Fecha del último mantenimiento
Bicicleta	id_terminal	NUMBER (FK)	Terminal donde se encuentra (FK)
Reserva	id_reserva	NUMBER	Identificador único de la reserva (PK)
Reserva	fecha_inicio	DATE	Fecha y hora de inicio de la reserva
Reserva	fecha_fin	DATE	Fecha y hora de fin de la reserva
Reserva	estado	VARCHAR2(20)	Estado de la reserva (activa, finalizada)
Reserva	codigo_desbloqueo	VARCHAR2(50)	Código QR o PIN generado para desbloquear
Reserva	id_usuario	NUMBER (FK)	Usuario que hizo la reserva (FK)
Reserva	id_bicicleta	NUMBER (FK)	Bicicleta reservada (FK)
Reporte	id_reporte	NUMBER	Identificador único del reporte (PK)
Reporte	descripcion	VARCHAR2(200)	Descripción del problema reportado
Reporte	fecha_reporte	DATE	Fecha del reporte
Reporte	estado	VARCHAR2(20)	Estado del reporte (pendiente, resuelto)
Reporte	id_bicicleta	NUMBER (FK)	Bicicleta afectada (FK)
Reporte	id_usuario	NUMBER (FK)	Usuario que reportó (FK)
Mantenimiento	id_mantenimiento	NUMBER	Identificador único del mantenimiento (PK)
Mantenimiento	descripcion	VARCHAR2(200)	Descripción de la actividad de mantenimiento
Mantenimiento	fecha_mant	DATE	Fecha de mantenimiento
Mantenimiento	tipo	VARCHAR2(50)	Tipo de mantenimiento (correctivo/preventivo)
Mantenimiento	id_bicicleta	NUMBER (FK)	Bicicleta intervenida (FK)
Procedimiento	p_id_bicicleta	NUMBER	ID numérico de la bicicleta a actualizar
Procedimiento	p_nuevo_estado	VARCHAR2	Texto que indica el nuevo estado (activo/inactivo)
Trigger	v_conteo	NUMBER	Variable local en el trigger: almacena el número total de reportes pendientes para una bicicleta

Script de la base de datos:

-- TABLA: Usuario

```
CREATE TABLE Usuario (
    id_usuario    NUMBER PRIMARY KEY,
    nombre        VARCHAR2(50),
    apellido      VARCHAR2(50),
    correo        VARCHAR2(100) UNIQUE,
    contrasena    VARCHAR2(100),
    administrador NUMBER(1) DEFAULT 0
);
```

-- TABLA: Terminal

```
CREATE TABLE Terminal (
    id_terminal     NUMBER PRIMARY KEY,
    nombre_terminal VARCHAR2(100),
    ubicacion       VARCHAR2(100),
    bicicletas_disponibles NUMBER DEFAULT 0
);
```

-- TABLA: Bicicleta

```
CREATE TABLE Bicicleta (
    id_bicicleta   NUMBER PRIMARY KEY,
    estado         VARCHAR2(20),
    descripcion    VARCHAR2(100),
    modelo         VARCHAR2(50),
    fecha_registro DATE,
    id_terminal    NUMBER,
    FOREIGN KEY (id_terminal) REFERENCES Terminal(id_terminal));
);
```

-- TABLA: Reserva

```
CREATE TABLE Reserva (
    id_reserva      NUMBER PRIMARY KEY,
    fecha_inicio    DATE,
    fecha_fin       DATE,
    estado          VARCHAR2(20),
    codigo_desbloqueo VARCHAR2(20),
    qr_imagen       BLOB,
    id_usuario       NUMBER,
    id_bicicleta    NUMBER,
    FOREIGN KEY (id_usuario) REFERENCES Usuario(id_usuario),
    FOREIGN KEY (id_bicicleta) REFERENCES Bicicleta(id_bicicleta)
);
```

-- TABLA: Reporte

```
CREATE TABLE Reporte (
    id_reporte      NUMBER PRIMARY KEY,
    descripcion     VARCHAR2(200),
    fecha_reporte   DATE DEFAULT SYSDATE,
    id_usuario       NUMBER,
    id_bicicleta    NUMBER,
    FOREIGN KEY (id_usuario) REFERENCES Usuario(id_usuario),
    FOREIGN KEY (id_bicicleta) REFERENCES Bicicleta(id_bicicleta)
);
```

-- SECUENCIAS

CREATE SEQUENCE Usuario_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE Bicicleta_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE Reserva_seq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE Reporte_seq START WITH 1 INCREMENT BY 1;

Documentación de APIs y Servicios – EcoRide

Método	Endpoint	Descripción	Parámetros	Ejemplo de uso	Respuesta esperada	Posibles errores
POST	/api/login	Iniciar sesión	email, password	{"email": "juan@mail.com", "password": "1234"}	200 OK: usuario	401: credenciales inválidas
POST	/api/registro	Registrar nuevo usuario	nombre, apellido, correo, contraseña	{"nombre": "Juan", "apellido": "Lopez", "correo": "juan@mail.com", "contraseña": "1234"}	201 Created	400: campos vacíos o correo duplicado
POST	/api/reservar	Reservar bicicleta	id_usuario, id_origen, id_destino	{"id_usuario": 1, "id_origen": 2, "id_destino": 3}	200 OK: qr_path, código_desbloqueo	400: no hay bicicletas / ya tiene reserva
POST	/api/devolver	Devolver bicicleta	id_usuario	{"id_usuario": 1}	200 OK: Bicicleta devuelta	400: no hay reserva activa
GET	/api/historial/:id_usuario	Ver historial de reservas	id_usuario en URL	/api/historial/1	200 OK: array de reservas	404: usuario no encontrado
POST	/api/reporte	Reportar problema	id_usuario, id_bicicleta, descripción	{"id_usuario": 1, "id_bicicleta": 5, "descripción": "Cadena rota"}	201 Created: Reporte enviado	400: datos incompletos

Instalación y Configuración del Sistema – EcoRide

Recurso	Versión recomendada
SO	Linux (Ubuntu / Oracle Linux)
Node.js	≥ v18.x
Oracle Database	Oracle XE 21c
Navegador	Chrome / Firefox
VPS o Hosting	Con acceso SSH

Estructura del Proyecto:

EcoRide/

```
└── backend
    ├── app.js
    └── public/qrs/
        └── frontend/
            ├── login/
            ├── usuarios/
            ├── admin/
            ├── css/
            └── index.html
    └── init.sql
└── docker-compose.yml
```

Guía de Instalación del Sistema

1. Clonar el proyecto:

```
git clone https://github.com/AEIL2203/Proyecto_1
```

```
cd ecoride
```

2. Instalar dependencias del backend:

```
cd backend
```

```
npm install
```

3. Configurar la base de datos Oracle XE:

A)Instalar Oracle Database XE 21c.

B)Crear usuario y ejecutar init.sql:

```
CONNECT system/password;
```

```
CREATE USER BICICLETAS_APP IDENTIFIED BY 12;
```

```
GRANT CONNECT, RESOURCE TO BICICLETAS_APP;
```

```
@/ruta/a/init.sql
```

Guía de Configuración del Entorno

1. Variables de entorno en .env:

```
ORACLE_USER=BICICLETAS_APP
```

```
ORACLE_PASSWORD=12
```

```
ORACLE_CONNECTION_STRING=localhost/XEPDB1
```

```
PUERTO=3000
```

2.Ejecutar el servidor:

```
node app.js
```

3. Verificar funcionamiento:

Visita en navegador:

<http://localhost:3000> o <http://tu-dominio.com>

Etiquetado de Versiones

Cada entrega estable fue marcada internamente en la producción con una etiqueta para identificar su estado:

Versión	Descripción	Fecha
v1.0.0	Inicio del proyecto: estructura inicial	21/05/2025
v1.1.0	Módulo de login y registro	21/05/2025
v1.2.0	Módulo de reserva de bicicletas	26/05/2025
v1.3.0	Módulo de devolución y reporte	26/05/2025
v1.4.0	Panel administrador y control terminales	27/05/2025
v1.5.0	Conexión con Oracle + QR funcional	27/06/2025
v2.0.0	Versión final integrada y funcional	08/06/2025 

Evidencia del Control

- Sistema versionado en **Git**
- Uso de ramas temporales para evitar conflictos
- Commit frecuentes con mensajes descriptivos

Notas de Versión – v1.0.0

Funcionalidades Incluidas

- Módulo de registro de usuarios con validación de campos.
- Sistema de inicio de sesión para usuarios y administradores.
- Interfaz de reserva de bicicletas con generación de código QR.
- Devolución de bicicletas con opción de reporte de problemas.
- Interacción directa con Oracle Database desde el backend (Node.js).
- Vista de historial de reservas por usuario.

Errores Corregidos

- Solución al problema de reinserción de bicicletas devueltas.
- Ajuste de visualización del estado de bicicletas en tiempo real.
- Corrección del endpoint de login para usuarios no administradores.
- Sincronización de estado entre terminales y disponibilidad de bicicletas.

Limitaciones

- No se cuenta con geolocalización de terminales.
- No hay recuperación de contraseña implementada.
- Solo soporta un idioma (Español).
- No se ha probado en navegadores móviles extensivamente.

Próximas Mejoras

- Implementación de notificaciones por correo al finalizar la reserva.
- Reporte automático por bicicleta inactiva luego de 24h sin devolución.
- Dashboard visual para administración con estadísticas en tiempo real.
- Integración de API para soporte de geolocalización de terminales.

Matriz de Trazabilidad de Requisitos

ID	Requisito Funcional	Módulo Implementado	Verificado
RF1	Registro de usuarios	registro.html, API /api/registro	<input checked="" type="checkbox"/>
RF2	Inicio de sesión	login.html, API /api/login	<input checked="" type="checkbox"/>
RF3	Reserva de bicicleta disponible	reserva.html, API /api/reservar	<input checked="" type="checkbox"/>
RF4	Generación de código QR por reserva	Backend qrcode, guardado en Oracle	<input checked="" type="checkbox"/>
RF5	Devolución de bicicleta	devolver.html, API /api/devolver	<input checked="" type="checkbox"/>
RF6	Registro de reporte de problemas	reporte.html, API /api/reporte	<input checked="" type="checkbox"/>
RF7	Visualización de historial de reservas	historial.html, API /api/historial	<input checked="" type="checkbox"/>
RF8	Administración de terminales y bicicletas	admin_bicicletas.html, API REST	<input checked="" type="checkbox"/>

Documentación del sistema

Archivo: app.js – Backend de EcoRide (Node.js + OracleDB)

Configuración Inicial:

```
const express = require('express');
const oracledb = require('oracledb');
const cors = require('cors');
const app = express();
const PORT = 3000;
const path = require('path');
const fs = require('fs');
const QRCode = require('qrcode');
```

Descripción:

Importa los módulos necesarios para crear el servidor, conectar con Oracle, permitir solicitudes desde el frontend (CORS), manipular archivos del sistema y generar códigos QR.

Middleware y configuración de rutas estáticas:

```
app.use(cors());  
app.use(express.json());  
app.use('/qrs', express.static(path.resolve(__dirname, 'public', 'qrs')));
```

Descripción:

- cors(): Permite llamadas desde cualquier origen.
- express.json(): Interpreta cuerpos JSON.
- /qrs: Permite servir imágenes QR desde una carpeta pública.

Configuración de conexión con Oracle:

```
const oracleConfig = {  
  user: 'BICICLETAS_APP',  
  password: '123456',  
  connectString: '34.9.82.233:1521/XEPDB1'  
};
```

Descripción:

Parámetros de conexión para acceder a la base de datos Oracle hospedada en una IP pública.

Autenticación:

- **POST /api/login:** Valida el usuario y contraseña desde la tabla Usuario.
Retorna datos del usuario si es válido.
- **POST /api/registro:** Registra un nuevo usuario con nombre completo, correo, contraseña y estado activo.

Gestión de Bicicletas:

- **GET /api/bicicletas:** Devuelve todas las bicicletas registradas (modelo, estado y ubicación).

- **POST /api/bicicletas** :Inserta una nueva bicicleta con estado "activo".
- **PUT /api/bicicletas/:id** :Edita datos de una bicicleta existente.
- **DELETE /api/bicicletas/:id** :Elimina una bicicleta por su ID.

Reservas:

- **POST /api/reservar:**

1. Verifica que el usuario no tenga una reserva activa.
2. Selecciona una bicicleta activa de la terminal origen.
3. Genera código QR con información de la reserva.
4. Guarda la reserva en la base de datos.
5. Actualiza el estado de la bicicleta y terminal.

- **POST /api/devolver**

1. Verifica que exista una reserva activa.
2. Cambia el estado de la reserva a “completada”.
3. Cambia el estado de la bicicleta a “activo”.
4. Suma una bicicleta disponible a la terminal.
5. Guarda los datos en la tabla Historial.

Reportes

- **POST /api/reporte**

1. Inserta un reporte con estado pendiente.
2. Si hay más de 3 reportes pendientes para una bicicleta, se marca como inactivo.

Gestión de Usuarios

- **GET /api/usuarios** : Obtiene todos los usuarios, mostrando: ID,Nombre,Email,Estado,Rol
- **PUT /api/usuarios/:id** : Actualiza el estado o rol (administrador/usuario) de un usuario.
- **DELETE /api/usuarios/:id** : Elimina completamente un usuario.

Historial de uso

- **GET /api/historial/:id_usuario:** Devuelve el historial completo del usuario:
 - Bicicleta utilizada
 - Fechas de inicio y fin
 - Descripción del viaje
 - Reporte (si existe)

Servidor:

```
app.listen(PORT, () => {  
    console.log(`Servidor backend corriendo en http://localhost:${PORT}`);  
});
```

Descripción:

Inicia el servidor Express en el puerto 3000.

Archivo: devolver.html

Objetivo:

Permite a un usuario autenticado devolver una bicicleta previamente reservada. Informa que la bicicleta debe estar asegurada en una terminal antes de confirmar la devolución.

Script JavaScript:

1. Obtener información del usuario:

Se accede a `localStorage` para obtener el objeto `usuario`. Si no está presente, se redirige al login.

2. Llamada al backend:

Se hace una petición POST al endpoint `/api/devolver` enviando el `id_usuario`.

3. Manejo de respuesta:

Si es exitosa, se guarda el `id_bicicleta` en `localStorage` y se redirige a `reporte.html`. Si hay error, se muestra una alerta.

4. Validación de errores:

Errores de red o respuesta son capturados y notificados al usuario.

Archivo: perfil.html

Objetivo: Mostrar los datos del usuario autenticado, incluyendo nombre, correo y estado de la cuenta. Permite cerrar sesión.

Script JavaScript:

1. Verificación de sesión:

Se verifica si existe un objeto `usuario` en `localStorage`, de lo contrario redirige al login.

2. Renderizado de datos:

Se muestran los campos nombre, correo y estado del usuario en pantalla.

3. Cierre de sesión:

El botón elimina el usuario del localStorage y redirige al login.

Archivo: registro.html

Objetivo: Permite el registro de nuevos usuarios al sistema EcoRide a través de un formulario con validaciones básicas.

Script JavaScript:

1. Validación de formulario:

Verifica que todos los campos estén completos antes de enviar la solicitud.

2. Llamada al backend:

Hace una petición POST al endpoint `/api/registro` con los datos del nuevo usuario.

3. Manejo de respuesta:

Si el registro es exitoso, redirige al login; de lo contrario, muestra un error.

Archivo: reporte.html

Objetivo: Permite al usuario reportar problemas al devolver una bicicleta. Solo está disponible tras haber devuelto una bicicleta.

Script JavaScript:

1. Validación inicial:

Verifica que existan datos del usuario y de la bicicleta devuelta.

2. Validación de contenido:

Evita que el reporte se envíe vacío.

3. Envío del reporte:

Realiza una petición POST a `/api/report` con los datos del reporte.

4. Opción de cancelar:

Permite omitir el reporte y volver a la vista de reserva.

Archivo: reserva.html

Objetivo: Interfaz para reservar una bicicleta seleccionando origen y destino. Muestra código de desbloqueo y QR al confirmar la reserva.

Script JavaScript:

1. Validación de sesión:

Comprueba que el usuario esté autenticado mediante localStorage.

2. Envío de reserva:

Envía los datos del usuario, origen y destino al endpoint `/api/reservar`.

3. Manejo de respuesta:

Muestra disponibilidad actualizada, el código de desbloqueo y botón para ver el QR generado.

4. Visualización del QR:

Al pulsar en 'Ver QR', se muestra el código QR en un modal con imagen.

Archivo: historial.html

Objetivo: Su objetivo es mostrar al usuario autenticado un historial detallado de los viajes realizados con bicicletas.

Script JavaScript

1. Obtener información del usuario:

Se recupera el objeto `usuario` desde el localStorage, que contiene la información del usuario que ha iniciado sesión. Si no existe, se redirige al login para proteger el acceso a la vista.

2. Llamada al backend para obtener historial:

Se hace una petición `fetch` al endpoint `/api/historial/:id_usuario` del backend. La respuesta contiene un arreglo con los viajes que se mostrará en una tabla HTML.

3. Renderizado dinámico de la tabla:

Cada elemento del historial (bicicleta, descripción, fecha de inicio/fin y reporte) se convierte en una fila HTML y se añade al `<tbody>` de la tabla.

4. Manejo de errores:

Si ocurre un error en la petición (por red, servidor u otros), se muestra un mensaje de alerta y se imprime el error en consola para depuración.

Login

Dentro de la etiqueta <script> hay una función que maneja el evento de clic en el botón "Acceder":

```
document.getElementById('loginBtn').addEventListener('click', async () => {
```

1. Obtiene los datos del formulario:

- Captura el correo y la contraseña escritos por el usuario.

2. Envía una solicitud POST al backend (API):

3. Procesa la respuesta del servidor:

- Si la respuesta es exitosa (response.ok):

- Guarda los datos del usuario en localStorage.

- Redirige:

- A la página de administración si el usuario es administrador.

- A la página de perfil de usuario si no lo es.

- Si ocurre un error (usuario inválido, contraseña incorrecta, etc.):

- Muestra el mensaje de error en el elemento con id="mensaje".

4. Manejo de errores:

- Si ocurre un fallo en la conexión con el servidor, muestra: "Error en el servidor".

Permite al usuario ingresar su correo y contraseña.

Envía esos datos a una API externa para autenticarlo.

Guarda los datos en localStorage si inicia sesión correctamente.

Redirige al usuario según su tipo (administrador o no).

Muestra mensajes de error si algo falla.

Admin Bicicletas

Muestra una tabla con bicicletas registradas.

Permite al administrador agregar una bicicleta con modelo y ubicación.

Ofrece opciones para editar y eliminar bicicletas.

Usa JavaScript y fetch API para comunicarse con el servidor (REST API).

Visualización dinámica de bicicletas:

- Al cargar la página, se obtiene automáticamente una lista de bicicletas desde el backend mediante una llamada a la API REST (GET /api/bicicletas).
- Los datos se muestran en una tabla con columnas como ID, Modelo, Estado y Ubicación.

Agregar nueva bicicleta:

- El administrador puede abrir un formulario para registrar una nueva bicicleta ingresando su modelo y ubicación.
- Al presionar "Guardar", se envía una solicitud POST a la API para almacenar la bicicleta en el servidor.

Editar bicicleta existente:

- Cada fila de bicicleta incluye un botón "Editar", que solicita al usuario ingresar nuevos valores para el modelo, ubicación y estado mediante prompt().
- Los datos actualizados se envían al servidor usando una petición PUT.

Eliminar bicicleta:

- Cada bicicleta también tiene un botón "Eliminar".
- Tras confirmar, se envía una solicitud DELETE a la API para eliminarla del sistema.

Navegación administrativa:

- En la parte superior se encuentra un menú que permite navegar a otros módulos como "Usuarios", "Reportes" o cerrar sesión.

Al cargar la página (window.onload = cargarBicicletas), se consulta la API y se muestran los datos. Los botones "Agregar", "Editar" y "Eliminar" disparan llamadas a la API. Todo el HTML es generado de forma estática, y el contenido dinámico es inyectado vía JS.

Admin reportes:

Este archivo es parte de la interfaz administrativa del sistema **EcoRide**. Está enfocado en la **gestión de reportes generados por usuarios**, lo cual permite al administrador monitorear y resolver incidencias o comentarios relacionados con el uso de las bicicletas.

Funciones principales:

1. Visualización de reportes:

- Al cargar la página, la función cargarReportes() realiza una solicitud GET a la API en la ruta:
- Los reportes recibidos en formato JSON se renderizan dinámicamente en una tabla HTML.
- Cada reporte muestra:
 - ID del reporte
 - Usuario que lo generó
 - Fecha del reporte
 - Descripción del incidente
 - Estado actual (pendiente o resuelto)
 - Botones para editar o eliminar

2. Edición del estado del reporte:

- El botón "Editar" permite cambiar el estado de un reporte a través de un prompt(), enviando un PUT a la API.
- Se puede actualizar el estado a "pendiente" o "resuelto".

3. Eliminación de reportes:

- El botón "Eliminar" permite borrar un reporte del sistema, tras confirmación, mediante un DELETE a la API.

Admin usuarios:

- Este código permite al administrador del sistema EcoRide:
- Ver una lista de usuarios registrados.
- Editar el estado (activo/inactivo).
- Cambiar si es administrador o no.
- Todo de forma dinámica, interactiva y usando JavaScript moderno con async/await.