🚜 Arquitectura del Proyecto SENAParking 📳

1. Introducción

El proyecto **SENAParking** es un sistema de gestión de parqueaderos desarrollado bajo la arquitectura **MVC** (**Modelo-Vista-Controlador**), con **Node.js** en el backend y **HTML**, **CSS y Bootstrap** en el frontend. Utiliza **MariaDB** como sistema de base de datos para almacenar y administrar la información de los usuarios y vehículos.

Este documento detalla la arquitectura del sistema, los módulos que lo componen, su flujo de datos y las tecnologías utilizadas. *

🔀 2. Arquitectura General

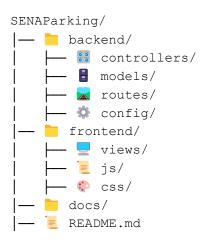
2.1. Modelo-Vista-Controlador (MVC)

SENAParking sigue el patrón MVC, dividiendo su estructura en:

- **Modelo** (models/): Gestiona la comunicación con la base de datos y representa las entidades del sistema.
- **Vista** (frontend/): Interfaz de usuario desarrollada con **HTML**, **CSS** y **JavaScript**.
- **Controlador** (controllers/): Contiene la lógica de negocio y gestiona la comunicación entre modelo y vista.

星 2.2. Diagrama General

La estructura del sistema es la siguiente:



• 3. Descripción de Módulos

El sistema se divide en los siguientes módulos:

3.1. Gestión de Usuarios del Sistema

- Administra los roles de **administrador** y **supervisor**.
- Permite la **creación**, **edición y eliminación** de usuarios.
- **K** Implementado en **usuarioSistemaController.js**.

3.2. Gestión de Usuarios del Parqueadero

- Permite el **registro y administración** de usuarios del parqueadero.
- **K** Implementado en usuarioParqueaderoController.js.

3.3. Gestión de Vehículos

- Registra los vehículos asociados a los usuarios.
- K Controlado por vehiculoController.js.

3.4. Registro de Accesos

- **K** Registra **entradas y salidas** de vehículos.
- **K** Gestionado por **accesoController.js**.

3.5. Gestión de Permisos

- P Define y administra **permisos** según el rol del usuario.
- **K** Implementado en **permisosController.js**.

📜 3.6. Registro de Actividad

- Guarda el historial de acciones de los usuarios.
- K Implementado en actividadController.js.

3.7. Reportes y Estadísticas

- **ii** Genera **informes** sobre el uso del parqueadero.
- K Gestionado por reportesController.js.

🔁 4. Flujo de Datos

∅ 4.1. Interacción entre el Backend y el Frontend

El sistema sigue un modelo cliente-servidor, donde:

- 1. **El usuario** interactúa con la interfaz (Frontend).
- 2. Se envía una solicitud HTTP al backend vía API REST.
- 3. **El backend procesa** la solicitud, accede a la base de datos y devuelve una respuesta.
- 4. La vista se actualiza según la respuesta del backend.

📝 4.2. Diagrama del Flujo de Datos

P Esquema básico de comunicación:

[\$\mathbb{L}\$ Usuario] → [\$\bullet\$ Interfaz Web] → [\$\bullet\$ Backend (API REST)] → [\$\bullet\$ Base de Datos]

🛠 5. Tecnologías Utilizadas

- **Backend**: Node.js, Express.js
- Base de Datos: 🖥 MariaDB
- Frontend: PHTML, CSS, Bootstrap, JavaScript
- **Autenticación**: i JWT (JSON Web Tokens)
- Seguridad: P Encriptación de contraseñas con berypt

📌 6. Casos de Uso

Registro de Vehículo

- 1. El usuario **ingresa** a la plataforma. 🏠
- 2. Accede a la sección de Registro de Vehículos.
- 3. Completa el **formulario** y lo envía.
- 4. El **backend almacena** la información en la base de datos.
- 5. Se confirma el registro en la interfaz. ✓

Caso 2: Entrada y Salida de Vehículo

1 Un **guardia** escanea la placa del vehículo. • 2 El sistema **verifica** si está registrado. • 3 Se almacena el **acceso** en la base de datos. • 4 Al salir, se registra la salida y se genera un **reporte**. • 1

• 7. Escalabilidad y Seguridad

7.1. Buenas Prácticas de Escalabilidad

- ✓ Uso de **MVC** para facilitar la modularidad. <u>►</u>
- ✓ **API REST** para futuras integraciones con apps móviles. ■
- ✓ Posibilidad de migración a microservicios. <a>□

1 7.2. Medidas de Seguridad Implementadas

JWT para autenticación de usuarios. bcrypt para encriptación de contraseñas. Validación de entradas para evitar inyecciones SQL.

6 8. Conclusión

El sistema **SENAParking** ha sido diseñado siguiendo **buenas prácticas de desarrollo**, con una estructura **modular y escalable**. La arquitectura **MVC** y la separación entre frontend y backend garantizan un desarrollo **mantenible y flexible**.

📢 ¡Preparado para futuras mejoras e implementación exitosa! 🎉