**Universidad Tecnológica de Aguascalientes**

**Diego Daniel Jiménez Esparza**

**Profesor:**

**Lic. Mario Villavicencio Rodríguez**

**Desarrollo Web Integral**



Contenido

[Caso de estudio 3](#_Toc205727388)

[Objetivo 3](#_Toc205727389)

[Funcionalidades Clave 3](#_Toc205727390)

[Plan del Proceso de Desarrollo 4](#_Toc205727391)

[Justificación de la Arquitectura 5](#_Toc205727392)

[Escalabilidad y mantenibilidad: 6](#_Toc205727393)

[Seguridad estructurada: 6](#_Toc205727394)

[Flexibilidad tecnológica: 6](#_Toc205727395)

[Diagrama de la arquitectura seleccionada. 7](#_Toc205727396)

[Patrones de diseño 7](#_Toc205727397)

[Singleton 7](#_Toc205727398)

[Factory Method 7](#_Toc205727399)

[Strategy 7](#_Toc205727400)

[Observer 7](#_Toc205727401)

[Framework a utilizar 8](#_Toc205727402)

[React.js 8](#_Toc205727403)

[Node.js 8](#_Toc205727404)

[MongoDB 8](#_Toc205727405)

[JSON Web Tokens (JWT) 8](#_Toc205727406)

# Caso de estudio

## Objetivo

Desarrollar un sistema seguro para gestionar usuarios, roles y catálogos jerárquicos aplicado a la información de vehículos automotrices, incluyendo características técnicas y mantenimiento.

## Funcionalidades Clave

1. Login y Autenticación

* Autenticación con usuarios y contraseñas encriptada (ByCript).
* Validación de credenciales.
* Dashboard diferenciado por roles
* Usuario estándar: Solo lectura de vehículos y catálogos.
* Mecánico (opcional): Acceso a mantenimiento y registros técnicos.

1. Gestión de Usuarios

* CRUD de usuarios (nombre, correo, contraseña, rol).
* Validación de correo único.
* Formato de correo válido.
* Contraseña con complejidad (mínimo 8 caracteres, mayúsculas, números).

1. Roles (Administrador y Usuario)

* Administrador:
* Crear/eliminar usuarios.
* Gestionar catálogos maestros (ej: marcas, modelos).
* Acceso a reportes (ej: vehículos registrados por mes).
* Usuario estándar:
* Solo consultar información (vehículos, catálogos).
* No puede eliminar registros.
* Mecánico (si aplica):
* Agregar registros de mantenimiento a vehículos.

1. Maestro-Detalle

* Maestro: Marcas de Vehículos
* Detalle: Modelos (ej: Toyota
* Corolla, Hilux).
* Maestro: Tipos de Servicio
* Detalle: Cambio de aceite, Alineación.
* Validaciones:
* No eliminar una marca si tiene modelos asociados.
* Búsqueda en cascada (ej: seleccionar marca y filtrar modelos).

1. Tecnologías Sugeridas

* Back: Node.js (Express).
* Front: React.
* Base de datos: MongoDB.
* Autenticación: JWT.

# Plan del Proceso de Desarrollo

|  |  |  |
| --- | --- | --- |
| Semana | Actividades | Entregables |
| 1 | Definición del proyecto, tareas iniciales instalación de herramientas | Documento de visión, tareas priorizadas. |
| 2 | Sprint 1: Diseño de la arquitectura y prototipo de UI | Diagrama de arquitectura, Prototipo de interfaz. |
| 3-4 | Sprint 2: Implementación de autenticación (Login) | Módulo de login funcional con roles (Admin/Usuario/Mecánico). |
| 5-6 | Sprint 3: CRUD de usuarios con validaciones (correo, contraseña). | Formulario de usuarios |
| 7-8 | Sprint 4: Maestro-detalle de marcas/modelos (MongoDB). | Catálogos jerárquicos con validaciones (no eliminar marcas con modelos). |
| 9-10 | Sprint 5: Dashboard por roles (React + permisos). | Vistas diferenciadas (Admin: reportes; Usuario: solo lectura). |
| 11-12 | Sprint 6: Registro de mantenimiento (mecánicos) y búsquedas. | Formulario de mantenimiento + filtros en cascada. |
| 13 | Sprint 7: Pruebas unitarias (Jest) e integración. | Reporte de pruebas (80%) |
| 14 | Sprint 8: Despliegue | Sistema en producción, manual técnico y de usuario. |
| 15 | Retrospectiva y cierre. | Informe final, lecciones aprendidas. |

# Justificación de la Arquitectura

Se selecciono la arquitectura MVC (Modelo-Vista-Controlador) ya que cumple con las siguientes características:

1. Backend: Node.js + Express (Controlador)

* Controlador (Express.js): Administrara la lógica de la autenticación y comunicación con la base de datos.

1. Frontend: React (Vista)

* Vista (React): Manejara la interfaz dinámica y la interacción del usuario: (dashboard, formularios).

1. Base de datos: MongoDB (Modelo)

* Modelo (MongoDB): Gestionara los datos: (usuarios, vehículos, catálogos).

## Escalabilidad y mantenibilidad:

1. La arquitectura MVC permite realizar modificaciones en este caso a ciertos componentes (ej: la vista con React) sin afectar la lógica del servidor (Express) o la estructura de datos (MongoDB).
2. También se tomó en cuenta esta arquitectura porque es Ideal para futuras integraciones.

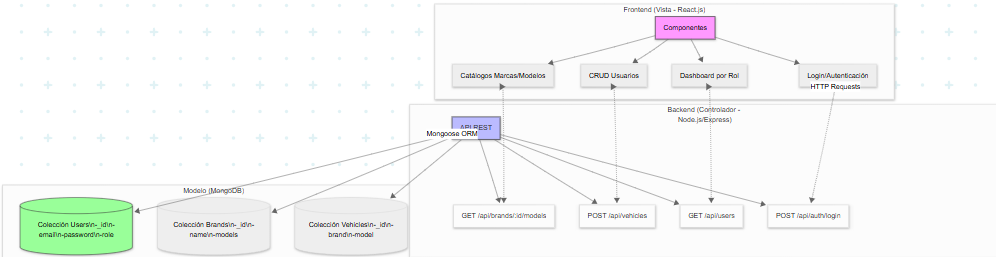
## Seguridad estructurada:

1. En este caso el controlador se centraliza en las validaciones (ej: roles, permisos) antes de interactuar con el Modelo o la Vista.
2. JWT y ByCrypt se implementan en el controlador, siguiendo buenas prácticas de capas.

## Flexibilidad tecnológica:

1. Otro punto por el que se tomó en cuenta esta arquitectura es porque MongoDB funciona como Modelo, pero podría migrarse a SQL (ej: PostgreSQL) si los catálogos requieren más relaciones complejas.

# Diagrama de la arquitectura seleccionada.



# Patrones de diseño

## Singleton

1. Uso: Conexión a la base de datos.
2. Justificación:

* Evita múltiples conexiones a MongoDB, optimizando recursos.
* Centraliza la configuración (timeouts, pools).

## Factory Method

1. Uso: Creación de dashboards y componentes UI según rol.
2. Justificación:

* Simplifica la generación de vistas complejas sin acoplar la lógica al cliente.
* Escalable para nuevos roles (ej: `supervisor`).

## Strategy

1. Uso: Validación de contraseñas y políticas de seguridad.
2. Justificación:

* Permite cambiar reglas de validación dinámicamente (ej: migrar de "basic" a "strict").
* Ideal para requisitos cambiantes de seguridad.

## Observer

1. Uso: Notificaciones en tiempo real (ej: actualización de catálogos).
2. Justificación:

* Notifica a componentes React cuando hay cambios en catálogos (sin polling).
* Bajo acoplamiento entre backend y frontend.

Esta combinación de patrones ayuda a:

1. Reducir la complejidad mediante la separación de responsabilidades.
2. Facilitar la escalabilidad (ej: añadir nuevos roles o políticas).
3. Optimizar el rendimiento (Singleton para DB, Observer para actualizaciones).

# Framework a utilizar

## React.js

Razones por que se seleccionaron:

1. Componentes reutilizables
2. Librerías
3. Facilitar la gestión de autenticaciones es decir compatibilidad con JWT

## Node.js

Razones por que se seleccionaron:

1. Escalabilidad para Apis
2. Integración con seguridad (ByCrypt, Helmet).

## MongoDB

Razones por que se seleccionaron:

1. Flexibilidad para jerarquías
2. Rendimiento, es decir consultas rápidas

## JSON Web Tokens (JWT)

Razones por que se seleccionaron:

1. No requiere almacenar sesiones en el servidor.
2. Seguridad digitalmente y compatible con ByCrypt.
3. Permite el control de accesos a roles mediane el token.

