

# INFORME TÉCNICO - SISTEMA DE ESTACIONAMIENTO

---

## PORADA

**Nombre:** Gian Castellino

**Institución:** Instituto Superior Juan XXIII

**Carrera:** Tecnicatura en Análisis de Sistemas

**Materia:** Algoritmos y Estructuras de Datos 2

**Año:** 2025

---

## ENUNCIADO

### Funcionamiento de la playa de estacionamiento de la Universidad Tecnológica

Se describe a continuación el funcionamiento de la playa de estacionamiento de la Universidad Tecnológica y del sistema de información que le da soporte:

Pueden estacionar distintos tipos de vehículos (motos/automóviles), cada uno con una tarifa de ingreso diferente. Si tiene abono, el precio es menor.

Se puede ingresar a la playa de estacionamiento por varios portones de ingreso diferentes.

No se asignan lugares específicos para los vehículos; las personas que ingresan al estacionamiento deberán ubicar su vehículo en algún lugar que se encuentre disponible.

Los interesados pueden comprar un abono de estacionamiento, de pago anticipado, que hace que el valor de cada estacionamiento sea más económico que si paga cada vez que ingresa a la playa. Debe informar su DNI y la cantidad de dinero que desea acreditar.

Si es la primera vez que estaciona, debe registrar sus datos personales (apellido, nombre, DNI) y los datos del o los vehículos (marca, modelo, dominio), con los cuales desea ingresar a la playa de estacionamiento.

Una vez registrado el propietario, cada vez que necesite acreditar dinero informa su DNI y la cantidad de dinero y se le cobra entregándole un comprobante donde consta: apellido y nombre, DNI, fecha de la transacción, monto acreditado y monto disponible en su cuenta.

El comprobante (ticket) que se entrega como constancia del cobro tiene los siguientes datos: apellido y nombre del propietario, DNI, fecha y hora de la transacción, monto acreditado y monto disponible en su cuenta, los números de dominio de todos los vehículos registrados de ese propietario y un número único de identificación del comprobante.

#### **Se solicita:**

1. Dos tipos de usuarios: el administrativo y el del cliente.
2. El primer perfil realiza la gestión del estacionamiento. Debe poder realizar Alta, baja y modificaciones de clientes, vehículos, ingresos y egresos de vehículos, dar ticket de pago (en nuestro caso se mostrará

- por pantalla). 2.1. El segundo perfil, debe poder visualizar los datos de los vehículos registrados y cargar abono o dinero.
3. La posibilidad de realizar al menos 2 filtros.
  4. La posibilidad de realizar al menos 2 listados.
  5. Tu propuesta debe incluir base de datos.
- 

## EXPLICACIÓN DE LA RESOLUCIÓN

### Enfoque Técnico Adoptado

Para resolver los requerimientos del enunciado, se desarrolló un **Sistema de Gestión de Estacionamiento** utilizando una arquitectura moderna de aplicación web con separación de responsabilidades:

- **Frontend:** Aplicación React.js que proporciona interfaz de usuario responsive
- **Backend:** API REST desarrollada en Node.js con Express.js
- **Base de Datos:** MongoDB (NoSQL) para flexibilidad en el manejo de datos
- **Autenticación:** Sistema JWT para control de sesiones y roles

### Decisiones de Diseño Principales

1. **Arquitectura MVC:** Separación clara entre Modelos (datos), Vistas (interfaz) y Controladores (lógica de negocio)
2. **Roles de Usuario:** Implementación de sistema de roles (admin/usuario) con permisos diferenciados
3. **Soft Delete:** Los usuarios no se eliminan físicamente, se marcan como inactivos para preservar integridad de datos
4. **Sistema de Comprobantes:** Proceso de aprobación administrativo para cargas de saldo
5. **Facturación Electrónica:** Generación automática de facturas PDF según normativas ARCA

### Cumplimiento de Requerimientos

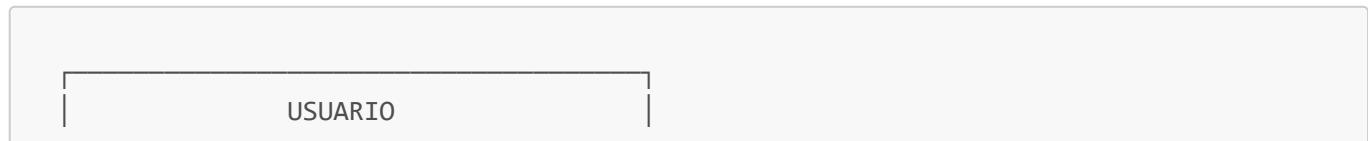
- **Dos tipos de usuarios:** Admin (gestión completa) y Usuario (consulta y carga de saldo)
  - **ABM Completo:** Clientes, vehículos, estacionamientos con validaciones de negocio
  - **Tickets/Comprobantes:** Sistema completo de generación y gestión de comprobantes
  - **Filtros implementados:** Por fecha, estado, usuario, tipo de vehículo
  - **Listados implementados:** Historial de estacionamientos, comprobantes, usuarios, estadísticas
  - **Base de datos:** MongoDB con esquemas relacionales y validaciones
- 

## MODELO DE BASE DE DATOS USADO (DIAGRAMA ENTIDAD-RELACIÓN)

### Diagrama Entidad-Relación del Sistema

#### Entidades Principales

#### USUARIO



```
dni (PK): String  
nombre: String  
apellido: String  
email: String (UNIQUE)  
password: String  
montoDisponible: Number  
asociado: Boolean  
fechaRegistro: Date  
rol: String (admin/usuario)  
activo: Boolean  
fechaDesactivacion: Date
```

## VEHICULO

### VEHICULO

```
dominio (PK): String  
tipo: String (auto/moto)  
marca: String  
modelo: String  
año: Number  
usuarioDni (FK): String
```

## ESTACIONAMIENTO

### ESTACIONAMIENTO

```
_id (PK): ObjectId  
dni (FK): String  
vehiculoDominio (FK): String  
horaInicio: Date  
horaFin: Date  
porton: String  
duracionHoras: Number  
montoTotal: Number  
tarifaAplicada: Number  
activo: Boolean
```

## COMPROBANTE

**COMPROBANTE**

```
nroComprobante (PK): String  
dni (FK): String  
fecha: Date  
montoAcreditado: Number  
estado: String  
usuarioInfo: Object  
fechaProceso: Date
```

**FACTURA****FACTURA**

```
numeroFactura (PK): String  
nroComprobante (FK): String  
fechaEmision: Date  
montoTotal: Number  
datosEmpresa: Object  
datosCliente: Object  
estado: String  
fechaAnulacion: Date  
archivoPath: String
```

**TRANSACCION****TRANSACCION**

```
_id (PK): ObjectId  
dni (FK): String  
tipo: String (credito/debito)  
monto: Number  
fecha: Date  
concepto: String  
estacionamientoId (FK): ObjectId  
comprobanteNumero (FK): String
```

**Entidades de Configuración****CONFIGURACION\_PRECIO**

**CONFIGURACION\_PRECIO**

```
_id (PK): ObjectId  
tipoUsuario: String  
precioPorHora: Number  
fechaVigencia: Date  
fechaCreacion: Date  
activo: Boolean  
descripcion: String
```

**CONFIGURACION\_EMPRESA****CONFIGURACION\_EMPRESA**

```
_id (PK): ObjectId  
razonSocial: String  
cuit: String  
domicilio: String  
telefono: String  
email: String  
activo: Boolean  
fechaActualizacion: Date
```

**Entidades de Auditoría****LOG\_PRECIO****LOG\_PRECIO**

```
_id (PK): ObjectId  
precioAnterior: Object  
precioNuevo: Object  
fechaCambio: Date  
usuarioModificador: String  
motivo: String
```

**LOG\_EMPRESA****LOG\_EMPRESA**

```
_id (PK): ObjectId  
datosAnteriores: Object  
datosNuevos: Object  
fechaCambio: Date  
usuarioModificador: String  
motivo: String
```

## Relaciones del Modelo de Datos

### Relaciones Principales

#### 1. USUARIO ↔ VEHICULO (1:N)

USUARIO (dni) ————— VEHICULO (usuarioDni)  
1    N

- Un usuario puede tener múltiples vehículos
- Cada vehículo pertenece a un único usuario
- Clave foránea: **usuarioDni** en VEHICULO

#### 2. USUARIO ↔ COMPROBANTE (1:N)

USUARIO (dni) ————— COMPROBANTE (dni)  
1    N

- Un usuario puede generar múltiples comprobantes
- Cada comprobante pertenece a un único usuario
- Clave foránea: **dni** en COMPROBANTE

#### 3. USUARIO ↔ ESTACIONAMIENTO (1:N)

USUARIO (dni) ————— ESTACIONAMIENTO (dni)  
1    N

- Un usuario puede tener múltiples estacionamientos
- Cada estacionamiento pertenece a un único usuario
- Clave foránea: **dni** en ESTACIONAMIENTO

#### 4. VEHICULO ↔ ESTACIONAMIENTO (1:N)

VEHICULO (dominio) ————— ESTACIONAMIENTO (vehiculoDominio)  
1    N

- Un vehículo puede usarse en múltiples estacionamientos
- Cada estacionamiento usa un único vehículo
- Clave foránea: **vehiculoDominio** en ESTACIONAMIENTO

## 5. COMPROBANTE ↔ FACTURA (1:1)

COMPROBANTE (nroComprobante) —————> FACTURA (nroComprobante)  
1   1

- Cada comprobante aprobado genera una única factura
- Cada factura corresponde a un único comprobante
- Clave foránea: **nroComprobante** en FACTURA

## Relaciones con TRANSACCION

## 6. ESTACIONAMIENTO ↔ TRANSACCION (1:N)

ESTACIONAMIENTO (\_id) —————> TRANSACCION (estacionamientoId)  
1   N

- Un estacionamiento puede generar múltiples transacciones
- Cada transacción puede estar asociada a un estacionamiento
- Clave foránea: **estacionamientoId** en TRANSACCION

## 7. COMPROBANTE ↔ TRANSACCION (1:N)

COMPROBANTE (nroComprobante) —————> TRANSACCION (comprobanteNumero)  
1   N

- Un comprobante aprobado genera transacciones de crédito
- Cada transacción puede estar asociada a un comprobante
- Clave foránea: **comprobanteNumero** en TRANSACCION

## 8. USUARIO ↔ TRANSACCION (1:N)

USUARIO (dni) —————> TRANSACCION (dni)  
1   N

- Un usuario puede tener múltiples transacciones
- Cada transacción pertenece a un único usuario
- Clave foránea: **dni** en TRANSACCION

## Relaciones de Auditoría

### 9. CONFIGURACION\_PRECIO ↔ LOG\_PRECIO (1:N)

- Cada cambio de precio genera un registro de auditoría
- Permite rastrear historial de modificaciones

### 10. CONFIGURACION\_EMPRESA ↔ LOG\_EMPRESA (1:N)

- Cada cambio de configuración genera un registro de auditoría
- Permite rastrear historial de modificaciones empresariales

## Características del Modelo de Datos

### Integridad Referencial

- **Validación de claves foráneas:** Todas las referencias entre entidades están validadas
- **Restricciones de eliminación:** No se permite eliminar registros con dependencias activas
- **Soft delete implementado:** Los usuarios se marcan como inactivos para preservar el histórico
- **Consistencia transaccional:** Las operaciones críticas mantienen la integridad de datos

### Optimización de Rendimiento

- **Índices únicos configurados:**
  - `dni` en USUARIO (clave primaria)
  - `email` en USUARIO (unicidad)
  - `dominio` en VEHICULO (clave primaria)
  - `nroComprobante` en COMPROBANTE (clave primaria)
- **Índices compuestos para consultas frecuentes:**
  - (`dni, activo`) para usuarios activos
  - (`fecha, estado`) para comprobantes por período
  - (`usuarioDni, activo`) para vehículos por usuario
- **Índices de fecha para filtros temporales:**
  - `fechaRegistro` en USUARIO
  - `horaInicio, horaFin` en ESTACIONAMIENTO
  - `fecha` en COMPROBANTE y TRANSACCION

## Sistema de Auditoría y Trazabilidad

- **Tablas de logging especializadas:**
  - `LOG_PRECIO`: Rastrea cambios en configuración de precios
  - `LOG_EMPRESA`: Rastrea modificaciones de datos empresariales
- **Timestamps automáticos:**
  - `fechaRegistro` en creación de usuarios

- `horaInicio/horaFin` en estacionamientos
- `fecha` en todas las transacciones
- **Preservación de snapshots:**
  - `usuarioInfo` en COMPROBANTE (datos del usuario al momento)
  - `tarifaAplicada` en ESTACIONAMIENTO (precio vigente)
  - Datos anteriores en tablas LOG\_

## Validaciones de Negocio a Nivel de Esquema

- **Campos obligatorios:** DNI, email, nombre, apellido en USUARIO
- **Enumeraciones controladas:**
  - `rol` en USUARIO (admin/usuario)
  - `tipo` en VEHICULO (auto/moto)
  - `estado` en COMPROBANTE (pendiente/aprobado/rechazado)
  - `porton` en ESTACIONAMIENTO (Norte/Sur/Este/Oeste)
- **Validaciones de formato:**
  - Email con regex de validación
  - Dominio con formato argentino (ABC123 o AB123CD)
  - DNI numérico de 8 dígitos
  - Montos mayores a cero

## Escalabilidad y Mantenimiento

- **Paginación preparada:** Estructura optimizada para consultas con limit/offset
- **Soft delete:** Preserva integridad histórica sin eliminar físicamente
- **Configuración externalizada:** Precios y datos empresa modificables dinámicamente
- **Logging completo:** Facilita debugging y auditorías futuras

---

# DESCRIPCIÓN DE LAS CLASES, ATRIBUTOS Y MÉTODOS CODIFICADOS

## Arquitectura del Sistema

El sistema se desarrolló siguiendo el patrón **MVC (Modelo-Vista-Controlador)** con una separación clara de responsabilidades:

- **Modelos:** Esquemas de datos y validaciones (MongoDB/Mongoose)
- **Controladores:** Lógica de negocio y procesamiento de solicitudes
- **Servicios:** Comunicación entre frontend y backend
- **Componentes:** Interfaz de usuario (React.js)

## 1. MODELOS DE DATOS (Backend)

### Usuario.js

**Atributos:**

- dni: **String** (clave primaria, validación única)
- nombre: **String** (requerido, mínimo 2 caracteres)
- apellido: **String** (requerido, mínimo 2 caracteres)
- email: **String** (único, validación regex)
- password: **String** (criptado con bcrypt)
- montoDisponible: **Number** (default: 0, mínimo: 0)
- asociado: **Boolean** (default: false)
- fechaRegistro: **Date** (automático)
- rol: **String** (enum: ['admin', 'usuario'])
- activo: **Boolean** (default: true)
- fechaDesactivacion: **Date** (opcional)

**Métodos principales:**

- comparePassword(): Comparación de contraseñas encriptadas
- toJSON(): Serialización segura (oculta contraseña)
- Validaciones: DNI único, email válido, contraseña segura

**Vehiculo.js****Atributos:**

- dominio: **String** (clave primaria, formato argentino)
- tipo: **String** (enum: ['auto', 'moto'])
- marca: **String** (requerido)
- modelo: **String** (requerido)
- año: **Number** (rango: 1990-actualidad)
- usuarioDni: **String** (referencia a Usuario)

**Validaciones:**

- Formato de dominio: ABC123 o AB123CD
- Año válido y coherente
- Tipo de vehículo controlado

**Comprobante.js****Atributos:**

- nroComprobante: **String** (autogenerado, único)
- dni: **String** (referencia a usuario)
- fecha: **Date** (timestamp automático)
- montoAcreditado: **Number** (mínimo: 1)
- estado: **String** (enum: ['pendiente', 'aprobado', 'rechazado'])
- usuarioInfo: **Object** (snapshot del usuario)
- fechaProceso: **Date** (cuando se aprueba/rechaza)

**Métodos:**

- generarNumero(): Generación secuencial de números
- calcularTotal(): Cálculo de monto con validaciones

## Estacionamiento.js

Atributos:

- dni: **String** (referencia a usuario)
- vehiculoDominio: **String** (referencia a vehículo)
- horaInicio: **Date** (timestamp de inicio)
- horaFin: **Date** (timestamp de fin, opcional)
- porton: **String** (enum: ['Norte', 'Sur', 'Este', 'Oeste'])
- duracionHoras: **Number** (calculado)
- montoTotal: **Number** (calculado)
- tarifaAplicada: **Number** (snapshot de tarifa)
- activo: **Boolean** (estado del estacionamiento)

Métodos:

- calcularDuracion(): Diferencia temporal en horas
- aplicarTarifa(): Cálculo según tipo de usuario
- finalizar(): Proceso de cierre con cálculos

## 2. CONTROLADORES (Backend)

### authController.js

#### Métodos principales:

- **register(req, res)**: Registro de nuevos usuarios
  - Validación de datos de entrada
  - Encriptación de contraseña con bcrypt
  - Generación de token JWT
  - Creación de usuario en base de datos
- **login(req, res)**: Autenticación de usuarios
  - Verificación de credenciales
  - Comparación de contraseñas
  - Generación de token de sesión
  - Control de usuarios activos
- **verificarToken(req, res, next)**: Middleware de autenticación
  - Validación de tokens JWT
  - Verificación de expiración
  - Extracción de datos de usuario
  - Control de permisos por rol

### usuarioController.js

#### Métodos principales:

- `obtenerUsuarios(req, res)`: Listado de usuarios (solo admin)
- `obtenerUsuarioPorDni(req, res)`: Consulta de usuario específico
- `actualizarUsuario(req, res)`: Modificación de datos de usuario
- `eliminarUsuario(req, res)`: Desactivación de usuario (soft delete)
- `cambiarEstadoAsociado(req, res)`: Cambio de estado de asociado
- `actualizarSaldo(req, res)`: Actualización manual de saldo

## vehiculoController.js

### Métodos principales:

- `agregarVehiculo(req, res)`: Alta de nuevo vehículo
  - Validación de formato de dominio
  - Verificación de duplicados
  - Asociación con usuario
  - Logging de operación
- `eliminarVehiculo(req, res)`: Baja de vehículo
  - Verificación de estacionamientos activos
  - Eliminación segura
  - Auditoría de cambios
- `obtenerVehiculos(req, res)`: Listado de vehículos por usuario
- `limpiarDuplicados(req, res)`: Utilidad de mantenimiento

## estacionamientoController.js

### Métodos principales:

- `iniciarEstacionamiento(req, res)`: Inicio de estacionamiento
  - Verificación de saldo suficiente
  - Control de estacionamiento único activo
  - Aplicación de tarifa según tipo de usuario
  - Registro de inicio con timestamp
- `finalizarEstacionamiento(req, res)`: Finalización de estacionamiento
  - Cálculo de duración temporal
  - Aplicación de tarifa proporcional
  - Descuento de saldo
  - Generación de transacción
- `obtenerHistorial(req, res)`: Historial de estacionamientos
- `verificarEstado(req, res)`: Estado actual de estacionamiento

## comprobanteController.js

**Métodos principales:**

- `crearComprobante(req, res)`: Creación de comprobante de pago
- `obtenerComprobantes(req, res)`: Listado filtrado por usuario/admin
- `generarPDFComprobante(req, res)`: Generación de comprobante en PDF
- `procesarComprobante(req, res)`: Procesamiento admin (aprobar/rechazar)

### 3. SERVICIOS FRONTEND

**usuarioService.js****Métodos principales:**

- `obtenerDatosUsuario(dni, token)`: Consulta de datos de usuario
- `actualizarUsuario(datos, token)`: Actualización de información
- `obtenerHistorialTransacciones(dni, token)`: Historial financiero

**estacionamientoService.js****Métodos principales:**

- `iniciarEstacionamiento(datos, token)`: Solicitud de inicio
- `finalizarEstacionamiento(dni, dominio, token)`: Solicitud de finalización
- `obtenerHistorial(dni, token)`: Consulta de historial
- `verificarEstadoActual(dni, token)`: Estado de estacionamiento activo

**authService.js****Métodos principales:**

- `login(credenciales)`: Autenticación de usuario
- `register(datosUsuario)`: Registro de nuevo usuario
- `logout()`: Cierre de sesión
- `verificarToken()`: Validación de token almacenado

### 4. COMPONENTES REACT (Frontend)

**Dashboard.jsx****Funcionalidades:**

- Estado principal de la aplicación
- Navegación entre secciones
- Mostrar información de usuario logueado
- Control de estacionamiento activo

**Login.jsx****Funcionalidades:**

- Formulario de autenticación

- Validación de campos en tiempo real
- Manejo de errores de login
- Redirección post-autenticación

### **AdminDashboard.jsx**

#### **Funcionalidades:**

- Panel de control administrativo
- Gestión de comprobantes pendientes
- Visualización de estadísticas
- Acceso a todas las funcionalidades admin

## 5. MIDDLEWARES Y UTILIDADES

### **authMiddleware.js**

#### **Funciones:**

- `verificarToken()`: Validación de JWT
- `verificarAdmin()`: Control de permisos administrativos
- `verificarUsuarioActivo()`: Verificación de usuario activo

### **errorHandler.js**

#### **Funciones:**

- Manejo centralizado de errores
- Formateo de respuestas de error
- Logging de errores críticos

---

## DESCRIPCIÓN DE CADA ABM REALIZADO

### 1. ABM USUARIOS

#### **ALTA (Registro de Usuario)**

- **Ubicación:** `authController.js -> register()`
- **Pantalla:** Formulario de registro con validaciones en tiempo real
- **Campos requeridos:** DNI, nombre, apellido, email, contraseña
- **Validaciones implementadas:**
  - DNI único en el sistema
  - Email único con formato válido
  - Contraseña con mínimo 6 caracteres
  - Campos obligatorios completados
- **Proceso:**
  1. Validación de datos en frontend
  2. Verificación de unicidad en backend

3. Encriptación de contraseña con bcrypt
  4. Creación en base de datos
  5. Generación de token JWT automático
- **Resultado:** Usuario creado con rol 'usuario' por defecto, login automático

## BAJA (Desactivación de Usuario)

- **Ubicación:** `adminController.js -> eliminarUsuario()`
- **Pantalla:** Panel administrativo → Gestión de usuarios
- **Restricciones:** Solo usuarios administradores
- **Tipo:** Soft delete (no se elimina físicamente)
- **Proceso:**
  1. Verificación de permisos administrativos
  2. Marcado del usuario como inactivo (`activo: false`)
  3. Registro de fecha de desactivación
  4. Liberación de DNI y email para re-registro
  5. Mantención de historial para integridad de datos
- **Efecto:** Usuario no puede loguearse, pero datos históricos se preservan

## MODIFICACIÓN (Actualización de Datos)

- **Ubicación:** `usuarioController.js -> actualizarUsuario()`
- **Pantallas:** Perfil de usuario (datos personales) y panel admin (gestión completa)
- **Campos modificables:**
  - Usuario común: nombre, apellido, email
  - Administrador: todos los campos + saldo + estado asociado
- **Validaciones:**
  - Email único si se modifica
  - Campos requeridos no vacíos
  - Formato de email válido
- **Casos especiales:**
  - Cambio de estado asociado (afecta tarifas)
  - Actualización manual de saldo (solo admin)
  - Modificación de rol (solo admin)

## 2. ABM VEHÍCULOS

### ALTA (Aregar Vehículo)

- **Ubicación:** `vehiculoController.js -> agregarVehiculo()`
- **Pantalla:** Dashboard → Mis Vehículos → Agregar Vehículo
- **Campos requeridos:** dominio, tipo, marca, modelo, año
- **Validaciones implementadas:**
  - Formato de dominio argentino (ABC123 o AB123CD)
  - Dominio único en el sistema
  - Año entre 1990 y año actual
  - Tipo válido (auto/moto)

- **Proceso:**
  1. Validación de formato en frontend
  2. Verificación de dominio único en backend
  3. Asociación automática con usuario logueado
  4. Registro en base de datos
  5. Logging de operación para auditoría
- **Resultado:** Vehículo disponible para estacionamientos

## BAJA (Eliminar Vehículo)

- **Ubicación:** `vehiculoController.js` -> `eliminarVehiculo()`
- **Pantalla:** Dashboard → Mis Vehículos → Lista con botón eliminar
- **Restricciones:** No se puede eliminar si tiene estacionamiento activo
- **Validaciones previas:**
  - Verificación de estacionamiento activo
  - Confirmación del usuario
- **Proceso:**
  1. Verificación de no tener estacionamiento activo
  2. Confirmación por parte del usuario
  3. Eliminación física del registro
  4. Logging de la operación
- **Resultado:** Vehículo eliminado del sistema, no disponible para futuros estacionamientos

## MODIFICACIÓN (Actualizar Datos del Vehículo)

- **Ubicación:** `vehiculoController.js` -> `actualizarVehiculo()`
- **Pantalla:** Dashboard → Mis Vehículos → Editar
- **Campos modificables:** marca, modelo, año (dominio y tipo no modificables)
- **Restricciones:** No se puede modificar dominio para mantener integridad de historial
- **Validaciones:** Año válido, campos no vacíos

## 3. ABM COMPROBANTES

### ALTA (Crear Comprobante de Pago)

- **Ubicación:** `comprobanteController.js` -> `crearComprobante()`
- **Pantalla:** Dashboard → Cargar Saldo
- **Datos requeridos:** monto a acreditar (mínimo \$1)
- **Proceso automático:**
  1. Generación de número único secuencial
  2. Captura de información del usuario (snapshot)
  3. Estado inicial: 'pendiente'
  4. Timestamp automático
  5. Almacenamiento de datos del usuario para integridad
- **Resultado:** Comprobante pendiente de aprobación administrativa

### MODIFICACIÓN (Procesamiento Administrativo)

- **Ubicación:** adminController.js -> aprobarComprobante() / rechazarComprobante()
- **Pantalla:** Panel Admin → Comprobantes Pendientes
- **Estados posibles:** pendiente → aprobado/rechazado
- **Proceso de aprobación:**
  1. Cambio de estado a 'aprobado'
  2. Acreditación automática de saldo al usuario
  3. Generación automática de factura
  4. Registro de transacción
  5. Timestamp de procesamiento
- **Proceso de rechazo:**
  1. Cambio de estado a 'rechazado'
  2. No acreditación de saldo
  3. Registro del motivo (opcional)
  4. Notificación al usuario

## CONSULTA (No hay baja física)

- Los comprobantes no se eliminan, se mantienen para auditoría
- Filtros disponibles: por estado, fecha, usuario, monto
- Generación de PDF para comprobantes aprobados

## 4. ABM ESTACIONAMIENTOS

### ALTA (Iniciar Estacionamiento)

- **Ubicación:** estacionamientoController.js -> iniciarEstacionamiento()
- **Pantalla:** Dashboard → Iniciar Estacionamiento
- **Datos requeridos:** vehículo, portón de ingreso
- **Validaciones previas:**
  - Usuario con saldo suficiente (mínimo para 1 hora)
  - No tener otro estacionamiento activo
  - Vehículo perteneciente al usuario
- **Proceso:**
  1. Verificación de saldo disponible
  2. Control de un solo estacionamiento activo por usuario
  3. Aplicación de tarifa según tipo de usuario (asociado/común)
  4. Registro con timestamp de inicio
  5. Marcado como activo
- **Resultado:** Estacionamiento iniciado, contador de tiempo en curso

### BAJA (Finalizar Estacionamiento)

- **Ubicación:** estacionamientoController.js -> finalizarEstacionamiento()
- **Pantalla:** Dashboard → Finalizar Estacionamiento (solo si hay uno activo)
- **Proceso automático:**
  1. Cálculo de duración (hora fin - hora inicio)
  2. Conversión a horas (redondeo hacia arriba)

3. Aplicación de tarifa correspondiente
  4. Descuento automático del saldo
  5. Marcado como finalizado (activo: false)
  6. Generación de transacción de débito
- **Validaciones:** Verificar saldo suficiente antes de finalizar
  - **Resultado:** Estacionamiento finalizado, saldo descontado, registro en historial

## CONSULTA (Historial y Estado)

- **Ubicación:** estacionamientoController.js -> obtenerHistorial() / verificarEstado()
- **Pantallas:** Dashboard → Historial de Estacionamientos
- **Información mostrada:**
  - Historial completo: fecha, vehículo, duración, costo, portón
  - Estado actual: estacionamiento activo con tiempo transcurrido
- **Filtros disponibles:** por fecha, vehículo, portón, estado
- **Funcionalidades:**
  - Paginación para historiales largos
  - Exportación a PDF
  - Estadísticas de uso

## 5. ABM CONFIGURACIONES (Solo Administrador)

### CONFIGURACIÓN DE PRECIOS

- **Ubicación:** precioController.js
- **Pantalla:** Panel Admin → Configuración → Precios
- **Campos:** precio común, precio asociado, fecha de vigencia
- **Proceso:**
  1. Validación de precios > 0
  2. Registro del cambio en log de precios
  3. Fecha de vigencia futura o inmediata
  4. Preservación de tarifa anterior para historial

### CONFIGURACIÓN DE EMPRESA

- **Ubicación:** configuracionEmpresaController.js
- **Pantalla:** Panel Admin → Configuración → Empresa
- **Campos:** razón social, CUIT, domicilio, teléfono, etc.
- **Validaciones:** CUIT válido, campos obligatorios
- **Logging:** Registro de todos los cambios para auditoría

### Características Generales de los ABMs

#### Seguridad Implementada

- Autenticación JWT en todas las operaciones
- Verificación de permisos por rol
- Validación de propiedad de recursos (usuarios solo pueden modificar sus datos)

- Sanitización de entradas para prevenir inyecciones

## Auditoría y Logging

- Registro de todas las operaciones críticas
- Timestamps automáticos
- Preservación de valores anteriores en modificaciones
- Trazabilidad completa de cambios

## Validaciones Comunes

- Campos requeridos no vacíos
- Formatos específicos (DNI, email, dominio)
- Rangos válidos (fechas, montos, años)
- Integridad referencial entre entidades

## Manejo de Errores

- Validaciones en frontend y backend
- Mensajes de error descriptivos al usuario
- Rollback automático en caso de fallos
- Logging de errores para debugging

---

# SUPOSICIONES ADOPTADAS PARA LA RESOLUCIÓN

## 1. SUPOSICIONES TÉCNICAS Y DE IMPLEMENTACIÓN

### Elección de Tecnologías

- **MongoDB sobre SQL:** Se eligió NoSQL para mayor flexibilidad en la evolución del esquema y para manejar documentos complejos como usuarios con arrays de vehículos embebidos
- **React.js para Frontend:** Framework moderno que permite crear interfaces de usuario dinámicas y responsivas
- **Node.js para Backend:** Permite usar JavaScript en todo el stack, facilitando el desarrollo y mantenimiento
- **JWT para Autenticación:** Tokens sin estado que escalan mejor que sesiones del servidor

### Arquitectura del Sistema

- **Separación Frontend/Backend:** API REST independiente que permite futuras expansiones (app móvil, otros clientes)
- **Patrón MVC:** Separación clara de responsabilidades para facilitar mantenimiento y testing
- **Soft Delete:** Los registros no se eliminan físicamente para preservar integridad histórica y auditoría

## 2. SUPOSICIONES DE REGLAS DE NEGOCIO

### Sistema de Usuarios

- **Dos roles únicos:** Solo 'admin' y 'usuario', sin roles intermedios para simplicidad
- **DNI como identificador principal:** Se asume que todos los usuarios tienen DNI argentino válido
- **Email único:** Cada email solo puede estar asociado a un usuario activo
- **Re-registro permitido:** Usuarios desactivados pueden volver a registrarse con el mismo DNI/email ya que la eliminación permanente de datos de la base de datos no es válida. (sistema de auditoría histórica).

## Sistema de Vehículos

- **Un vehículo por usuario:** Cada dominio solo puede pertenecer a un usuario a la vez
- **Tipos limitados:** Solo 'auto' y 'moto' para simplificar tarificación

## Sistema de Estacionamiento

- **Un estacionamiento activo por usuario:** No se permite múltiples estacionamientos simultáneos
- **Portones predefinidos:** Solo 4 portones (Norte, Sur, Este, Oeste) para control de acceso
- **Tarifación por hora completa:** Se cobra hora completa aunque se use fracción (redondeo hacia arriba)
- **Saldo mínimo requerido:** Usuario debe tener saldo para al menos 1 hora antes de estacionar

## Sistema de Pagos y Facturación

- **Comprobantes deben ser aprobados:** Todo pago requiere aprobación administrativa antes de acreditar saldo
- **Facturación automática:** Al aprobar un comprobante se genera automáticamente la factura
- **Numeración secuencial:** Facturas y comprobantes tienen numeración correlativa
- **Anulación con límite temporal:** Facturas solo se pueden anular dentro de 15 días (regla ARCA RG 145).

## 3. SUPOSICIONES DE COMPORTAMIENTO DEL SISTEMA

### Concurrencia y Estados

- **Máximo 100 usuarios concurrentes:** Sistema dimensionado para uso mediano
- **Estados no reversibles:** Comprobantes aprobados/rechazados no pueden volver a pendiente
- **Transacciones atómicas:** Operaciones financieras son indivisibles (todo o nada)

### Validaciones y Seguridad

- **Validación dual:** Frontend (UX) y backend (seguridad) para todas las entradas
- **Tokens con expiración:** JWT válidos por 24 horas para balance seguridad/usabilidad
- **Solo administradores pueden:** aprobar pagos, ver todos los usuarios, cambiar precios

## 4. SUPOSICIONES DE DATOS Y FORMATOS

### Precisión Numérica

- **Montos en pesos argentinos:** Precisión hasta centavos (2 decimales)
- **Timestamps en UTC:** Todas las fechas se almacenan en UTC para consistencia
- **Duraciones en horas:** Estacionamientos se miden en horas con redondeo hacia arriba

## Límites del Sistema

- **Máximo 5 vehículos por usuario:** Límite razonable para usuarios comunes
- **Montos máximos:** \$100,000 por comprobante para prevenir errores de carga
- **Históricos ilimitados:** No se eliminan registros históricos para auditoría

## 5. SUPOSICIONES DE INTERFAZ DE USUARIO

### Experiencia del Usuario

- **Responsive design:** Sistema debe funcionar en desktop y móvil
- **Feedback inmediato:** Validaciones en tiempo real para mejor UX
- **Estados visuales claros:** Diferenciación visual entre pendiente/aprobado/rechazado
- **Confirmaciones para acciones críticas:** Eliminar vehículo, finalizar estacionamiento

### Accesibilidad

- **Pantallas intuitivas:** Navegación simple sin necesidad de manual
- **Mensajes de error claros:** Explicaciones comprensibles para el usuario
- **Carga asíncrona:** Las operaciones no bloquean la interfaz

## 6. SUPOSICIONES DE RENDIMIENTO Y ESCALABILIDAD

### Optimizaciones Implementadas

- **Índices en campos frecuentes:** DNI, email, dominios para búsquedas rápidas
- **Paginación en listados:** Para evitar cargar grandes volúmenes de datos
- **Caché de configuraciones:** Precios y configuración de empresa se cachean

### Límites de Crecimiento

- **Base de datos hasta 10GB:** Suficiente para operación de varios años
- **1000 usuarios máximo:** Dimensionamiento para crecimiento gradual
- **Backup diario:** Política de respaldo para protección de datos

## 7. SUPOSICIONES DE INTEGRACIÓN Y FUTURO

### Preparación para Expansión

- **API documentada:** Endpoints preparados para integración con otros sistemas
- **Logs detallados:** Para debugging y auditoría futura
- **Configuración externalizada:** Fácil cambio de parámetros sin modificar código

### Compatibilidad

- **Navegadores modernos:** Chrome, Firefox, Safari, Edge (últimas 2 versiones)
- **JavaScript habilitado:** Funcionalidad completa requiere JS
- **Conexión estable:** Sistema web requiere conectividad constante

## 8. SUPOSICIONES DE MANTENIMIENTO Y SOPORTE

### Administración del Sistema

- **Un solo admin inicial:** Sistema comienza con un usuario administrador predefinido
- **Backup automático:** Respaldos programados sin intervención manual
- **Logs rotativos:** Archivos de log se rotan para no ocupar espacio excesivo

### Actualizaciones y Cambios

- **Migración de datos:** Scripts preparados para futuras actualizaciones de esquema
- **Configuración dinámica:** Precios y configuración modificables sin reiniciar sistema
- **Rollback disponible:** Posibilidad de volver a versión anterior en caso de problemas