

# Manual Técnico - Frontend del Sistema de Gestión de Multas

## Versión 1.0

---

### Información del Documento

**Sistema:** Frontend - Sistema de Gestión de Multas y Acuerdos de Pago **Versión:** 1.0.0 **Framework:** Angular 19.0.0 **Fecha de**

**actualización:** Octubre 2025 **Tipo de documento:** Manual Técnico - Frontend **Repositorio:**

<https://github.com/ingrid0208/Proyecto-FRONT>

---

### Tabla de Contenidos

1. [Introducción](#)
  2. [Arquitectura del Sistema](#)
  3. [Stack Tecnológico](#)
  4. [Estructura del Proyecto](#)
  5. [Configuración del Entorno](#)
  6. [Módulos y Componentes](#)
  7. [Servicios y API](#)
  8. [Rutas y Navegación](#)
  9. [Gestión de Estado](#)
  10. [Autenticación y Seguridad](#)
  11. [Integración con Backend](#)
  12. [Base de Datos y Modelos](#)
  13. [Despliegue](#)
  14. [Testing](#)
  15. [Mantenimiento y Troubleshooting](#)
  16. [API Reference](#)
  17. [Guías de Desarrollo](#)
- 

## 1. Introducción

### 1.1 Propósito del Manual

Este manual técnico proporciona información detallada sobre la arquitectura, implementación y mantenimiento de la aplicación Frontend del Sistema de Gestión de Multas. Está dirigido a:

- Desarrolladores Frontend
- Arquitectos de Software
- Personal de mantenimiento y soporte técnico

### 1.2 Alcance del Documento

Este manual cubre únicamente el desarrollo Frontend del sistema, incluyendo:

- Interfaz de usuario con Angular 19
- Componentes y módulos de la aplicación
- Servicios de comunicación con la API
- Gestión de estado y autenticación
- Despliegue y configuración del frontend
- Testing y buenas prácticas de desarrollo

### 1.3 Convenciones del Documento

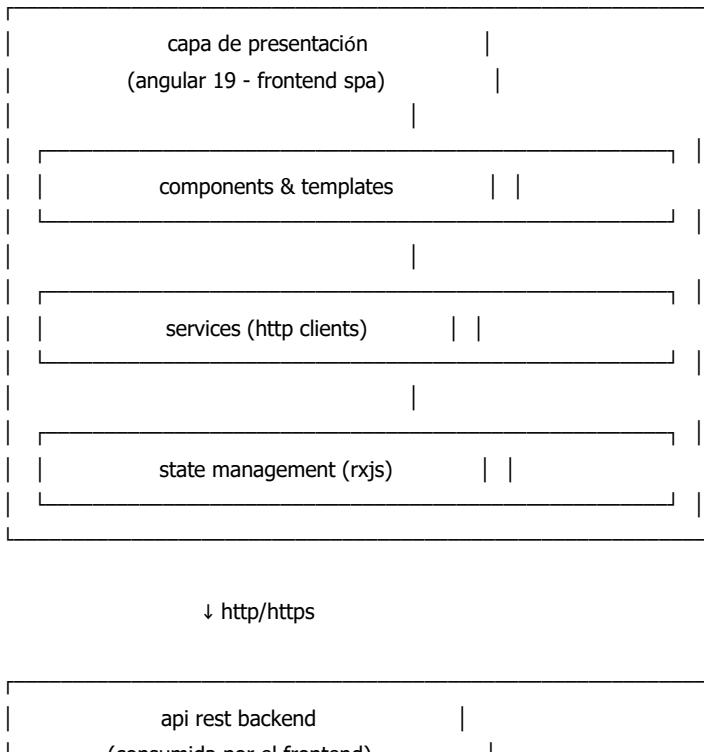
- `código` : Fragmentos de código inline

- **Negrita**: Términos importantes
  - *Cursiva*: Énfasis
  - **Nota**: Información relevante
  - **Advertencia**: Precauciones importantes
  - **Buena práctica**: Recomendaciones
- 

## 2. Arquitectura del Sistema

### 2.1 Arquitectura Frontend

El frontend es una **Single Page Application (SPA)** desarrollada con Angular 19 que consume una API REST:



### 2.2 Estructura de Capas Frontend

**Patrón de Diseño: Feature-Based Architecture**

```

src/app/
├── core/                      # Servicios singleton y configuración
|   ├── config/                # Configuración de la aplicación
|   ├── services/              # Servicios globales
|   ├── interceptors/          # HTTP Interceptors
|   └── guards/                # Route Guards
├── features/                  # Módulos funcionales (lazy-loaded)
|   ├── auth/                  # Autenticación
|   ├── multas/                # Gestión de multas
|   ├── admin/                 # Administración
|   └── profile/               # Perfil de usuario
└── shared/                    # Componentes compartidos
    ├── components/            # Componentes reutilizables
    ├── directives/            # Directivas
    ├── pipes/                 # Pipes
    └── models/                 # Interfaces y tipos
└── layout/                    # Componentes de layout
    ├── header/                # Componentes de layout
    ├── sidebar/               # Componentes de layout
    └── footer/                # Componentes de layout
  
```

## 2.3 Patrones de Arquitectura Implementados

### 2.3.1 Dependency Injection (DI)

Angular utiliza DI para gestionar dependencias:

```
// Decorador Injectable que marca esta clase como un servicio inyectable
@Injectable({
  providedIn: 'root' // Singleton a nivel de aplicación - solo existe una instancia en toda la app
```

```

})
export class CustomerService {
  // Constructor con inyección de dependencias del HttpClient
  constructor(private http: HttpClient) {}
}

```

### 2.3.2 Observable Pattern (RxJS)

Para programación reactiva:

```

// Método que obtiene la lista de clientes desde el servidor
getCustomers(): Observable<Customer[]> {
  // Retorna un Observable que emitirá un array de Customer cuando la petición HTTP complete
  return this.http.get<Customer[]>(`${this.apiUrl}/customers`);
}

```

### 2.3.3 Smart/Dumb Components Pattern

- **Smart Components**: Contienen lógica de negocio
- **Dumb Components**: Solo presentación, reciben datos vía @Input

### 2.3.4 Lazy Loading

Los módulos se cargan bajo demanda:

```
{
  path: 'multas', // Ruta URL para acceder al módulo de multas
  // Lazy loading: carga el módulo solo cuando el usuario accede a esta ruta
  loadChildren: () => import('./features/multas/multas.routes')
}
```

## 3. Stack Tecnológico

### 3.1 Frontend

Tecnología	Versión	Propósito
<b>Angular</b>	19.0.0	Framework principal
<b>TypeScript</b>	~5.6.2	Lenguaje de programación
<b>PrimeNG</b>	19.1.4	Biblioteca de componentes UI
<b>Angular Material</b>	19.0.4	Componentes UI adicionales
<b>RxJS</b>	~7.8.0	Programación reactiva
<b>Chart.js</b>	4.4.2	Gráficos y visualizaciones
<b>SweetAlert2</b>	^11.23.0	Modales y alertas
<b>TailwindCSS</b>	^3.4.17	Framework CSS utility-first
<b>SASS/SCSS</b>	^1.73.0	Preprocesador CSS

### 3.2 Herramientas de Desarrollo

Herramienta	Versión	Propósito

<b>Angular CLI</b>	^19.0.6	Herramienta de línea de comandos
<b>ESLint</b>	^9.14.0	Linter de código
<b>Prettier</b>	^3.0.0	Formateador de código
<b>Karma</b>	~6.4.0	Test runner
<b>Jasmine</b>	~5.4.0	Framework de testing

### 3.3 Configuración de API Backend

El frontend consume una API REST. Para conectar con el backend:

```
// environment.development.ts - Configuración para el entorno de desarrollo
export const environment = {
  production: false, // Indica que NO es un entorno de producción
  apiURL: 'https://localhost:7286/api' // URL base de la API en desarrollo local
};
```

## 4. Estructura del Proyecto

### 4.1 Estructura de Directorios Completa

```
Proyecto-FRONT/
├── src/
│   ├── app/
│   │   ├── core/                      # Módulo Core (Singleton)
│   │   │   ├── config/                 # Configuración de la app
│   │   │   │   └── app.config.ts
│   │   │   ├── guards/
│   │   │   │   ├── auth.guard.ts      # Guard de autenticación
│   │   │   │   └── role.guard.ts     # Guard de roles
│   │   │   ├── interceptors/
│   │   │   │   └── auth-expired.interceptor.ts
│   │   │   ├── services/
│   │   │   │   ├── api/              # Servicios API
│   │   │   │   │   ├── customer.service.ts
│   │   │   │   │   ├── department.service.ts
│   │   │   │   │   ├── document-type.service.ts
│   │   │   │   │   ├── municipality.service.ts
│   │   │   │   │   ├── payment-frequency.service.ts
│   │   │   │   │   ├── persona.service.ts
│   │   │   │   │   ├── roles.service.ts
│   │   │   │   │   └── rol-user.service.ts
│   │   │   │   └── user-infraction.service.ts
│   │   │   └── utils/                # Utilidades
│   │   │   │   ├── icon.service.ts
│   │   │   │   ├── photo.service.ts
│   │   │   │   ├── recaptcha.service.ts
│   │   │   │   └── session-ping.service.ts
│   │   │   └── models/               # Interfaces y modelos
│   │   └── features/               # Módulos funcionales
│   │       ├── auth/                # Autenticación
│   │       └── pages/
```



```

|   └── assets/                      # Recursos estáticos
|       ├── images/
|       ├── icons/
|       └── styles/
|
|   └── environments/                # Configuración de entornos
|       ├── environment.ts
|       ├── environment.development.ts
|       └── environment.prod.ts
|
|   └── styles.scss                  # Estilos globales
|   └── index.html                   # HTML principal
|   └── main.ts                      # Punto de entrada
|
└── angular.json                   # Configuración Angular
└── package.json                   # Dependencias npm
└── tsconfig.json                  # Configuración TypeScript
└── tailwind.config.js             # Configuración Tailwind
└── vercel.json                    # Configuración Vercel
└── README.md                       # Documentación

```

## 4.2 Descripción de Módulos Principales

### 4.2.1 Core Module

- **Propósito:** Servicios singleton y configuración global
- **Importación:** Solo una vez en `main.ts`
- **Contenido:** Guards, interceptors, servicios HTTP

### 4.2.2 Features Modules

- **Propósito:** Funcionalidades específicas del dominio
- **Carga:** Lazy loading para optimización
- **Independencia:** Cada feature es autónomo

### 4.2.3 Shared Module

- **Propósito:** Componentes, pipes y directivas reutilizables
- **Importación:** En cualquier módulo que los necesite

## 5. Configuración del Entorno

### 5.1 Requisitos Previos

#### Software Necesario

```

# Node.js (versión LTS recomendada)
node --version # v18.x o superior

# npm
npm --version # v9.x o superior

# Angular CLI
npm install -g @angular/cli@19
ng version

```

#### Sistema Operativo

- Windows 10+

- Linux (Ubuntu 20.04+, Debian 11+)

## 5.2 Instalación del Proyecto

### Paso 1: Clonar el Repositorio

```
git clone <https://github.com/ingrid0208/Proyecto-FRONT.git>
cd Proyecto-FRONT
```

### Paso 2: Instalar Dependencias

```
npm install
```

### Paso 3: Configurar Variables de Entorno

Crear archivo `src/environments/environment.development.ts` :

```
// Exporta la configuración del entorno de desarrollo
export const environment = {
  production: false, // Indica que es entorno de desarrollo
  apiURL: 'https://localhost:7286/api', // URL de la API backend en desarrollo local
  //apiURL: '/api/', // URL alternativa comentada para proxy
  recapchat: { // Configuración de Google reCAPTCHA
    sitekey: '6LcEs7grAAAAANT_r2A-jLRXQN0A-fEEm1aLqGgX' // Clave pública del sitio para reCAPTCHA
  }
};
```

## 5.3 Scripts de Desarrollo

### package.json Scripts

```
{
  "scripts": {
    "ng": "ng", // Ejecuta el CLI de Angular
    "start": "ng serve", // Inicia el servidor de desarrollo de Angular
    "build": "ng build", // Compila la aplicación
    "build:prod": "ng build --configuration production", // Compila en modo producción
    "watch": "ng build --watch --configuration development", // Compila y observa cambios en desarrollo
    "test": "ng test", // Ejecuta los tests de la aplicación
    "lint": "ng lint", // Revisa la calidad del código
    "format": "prettier --write \"**/*.{js,ts,html,scss}\"" // Formatea archivos con Prettier
  }
}
```

### Comandos Comunes

```
# Desarrollo
npm start          # http://localhost:4200

# Desarrollo con configuración específica
ng serve --configuration development

# Build de producción
npm run build:prod
```

```
# Tests
npm test

# Linter
npm run lint

# Formatear código
npm run format
```

## 5.4 Configuración de Angular

### angular.json - Configuraciones Principales

```
{
  "projects": {
    "sakai-ng": { // Nombre del proyecto Angular
      "architect": {
        "build": { // Configuración de la tarea de construcción (build)
          "options": {
            "outputPath": "dist/sakai-ng", // Carpeta donde se guardan los archivos generados al compilar
            "index": "src/index.html", // Archivo HTML principal de la app
            "browser": "src/main.ts", // Punto de entrada principal de la app (TypeScript)
            "styles": [
              "@angular/material/prebuilt-themes/azure-blue.css", // Tema de Angular Material que se incluye
              "src/styles.scss" // Hoja de estilos principal del proyecto
            ]
          },
          "configurations": {
            "production": { // Configuración específica para el entorno de producción
              "budgets": [
                {
                  "type": "initial", // Tipo de presupuesto (tamaño del bundle inicial)
                  "maximumWarning": "1mb", // Lanza advertencia si el bundle inicial supera 1MB
                  "maximumError": "5mb" // Lanza error si el bundle inicial supera 5MB
                }
              ],
              "outputHashing": "all" // Agrega hash a los archivos generados para mejor cacheo
            }
          }
        }
      }
    }
  }
}
```

## 6. Módulos y Componentes

### 6.1 Módulo de Autenticación

#### Estructura

```
auth/
|--- pages/
```

```

|   └── login/
|   |   ├── login.component.ts
|   |   ├── login.component.html
|   |   └── login.component.scss
|   └── register/
|   |   ├── register.component.ts
|   |   └── register.component.html
|   └── identificacion/
└── auth.routes.ts

```

#### **login.component.ts (Ejemplo)**

```

// Importa los módulos y servicios necesarios de Angular
import { Component, inject } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { AuthService } from '@core/services/auth.service';

@Component({
  selector: 'app-login', // Selector del componente
  templateUrl: './login.component.html', // Ruta de la plantilla HTML
  styleUrls: ['./login.component.scss'] // Ruta de los estilos
})

export class LoginComponent {
  // Inyección de dependencias usando el método inject
  private fb = inject(FormBuilder); // Para construir formularios reactivos
  private authService = inject(AuthService); // Servicio de autenticación
  private router = inject(Router); // Servicio de navegación

  // Definición del formulario de login con validaciones
  loginForm: FormGroup = this.fb.group({
    email: ['', [Validators.required, Validators.email]], // Campo email requerido y con formato de email
    password: ['', [Validators.required, Validators.minLength(6)]] // Campo password requerido y mínimo 6 caracteres
  });

  // Método que se ejecuta al enviar el formulario
  onSubmit(): void {
    if (this.loginForm.valid) { // Verifica si el formulario es válido
      this.authService.login(this.loginForm.value).subscribe({
        next: (response) => {
          localStorage.setItem('token', response.token); // Guarda el token en localStorage
          this.router.navigate(['/dashboard']); // Redirige al dashboard
        },
        error: (error) => {
          console.error('Login failed:', error); // Muestra error en consola si falla el login
        }
      });
    }
  }
}

```

## **6.2 Módulo de Multas**

### **Componentes Principales**

### 6.2.1 Consultar Multas

```
// consultar-ingresar.component.ts - Componente para consultar e ingresar multas
@Component({
  selector: 'app-consultar-ingresar', // Selector del componente para usar en HTML
  templateUrl: './consultar-ingresar.component.html' // Plantilla HTML del componente
})
export class ConsultarIngresarComponent implements OnInit {
  // Observable que contendrá el stream de multas
  multas$: Observable<Multta[]>;
  // Número de documento para buscar multas
  documentNumber: string = '';

  // Constructor con inyección de dependencias
  constructor(
    private multasService: MultasService, // Servicio para gestionar multas
    private route: ActivatedRoute // Servicio para acceder a la ruta activa
  ) {}

  // Método del ciclo de vida que se ejecuta al inicializar el componente
  ngOnInit(): void {
    this.loadMultas(); // Carga las multas al iniciar
  }

  // Método para cargar las multas por número de documento
  loadMultas(): void {
    // Obtiene las multas del servicio y las asigna al observable
    this.multas$ = this.multasService.getMultasByDocument(this.documentNumber);
  }

  // Método para filtrar multas por estado
  filterMultas(status: string): void {
    // Filtra las multas según el estado proporcionado
    this.multas$ = this.multasService.getMultasByStatus(status);
  }
}
```

### 6.2.2 Acuerdos de Pago

```
// formulario-acuerdo-pago.component.ts - Componente para generar acuerdos de pago
@Component({
  selector: 'app-formulario-acuerdo-pago', // Selector del componente
  templateUrl: './formulario-acuerdo-pago.component.html' // Plantilla HTML
})
export class FormularioAcuerdoPagoComponent {
  // Formulario reactivo para el acuerdo de pago
  acuerdoForm: FormGroup;
  // Array de multas seleccionadas para el acuerdo
  selectedMultas: Multta[] = [];
  // Monto total del acuerdo
  totalAmount: number = 0;

  // Constructor con inyección de dependencias
  constructor(
    private fb: FormBuilder, // Constructor de formularios reactivos
    private acuerdoService: AcuerdoPagoService, // Servicio para gestionar acuerdos de pago
  ) {
```

```

    private router: Router // Servicio de navegación
) {
  this.initForm(); // Inicializa el formulario al crear el componente
}

// Método para inicializar el formulario con validaciones
initForm(): void {
  this.acuerdoForm = this.fb.group({
    // Campo cuota inicial: requerido y mínimo 0
    cuotaInicial: ['', [Validators.required, Validators.min(0)]],
    // Campo número de cuotas: requerido y mínimo 1
    numeroCuotas: ['', [Validators.required, Validators.min(1)]],
    // Campo frecuencia de pago: requerido
    frecuenciaPago: ['', Validators.required],
    // Campo fecha del primer pago: requerido
    fechaPrimerPago: ['', Validators.required]
  });
}

// Método para calcular el plan de pagos basado en los valores del formulario
calcularPlanPagos(): void {
  const valores = this.acuerdoForm.value; // Obtiene los valores del formulario
  // Lógica de cálculo del plan de pagos (implementar según necesidad)
}

// Método para generar el acuerdo de pago
generarAcuerdo(): void {
  // Verifica que el formulario sea válido antes de enviar
  if (this.acuerdoForm.valid) {
    // Llama al servicio para crear el acuerdo con los valores del formulario
    this.acuerdoService.createAcuerdo(this.acuerdoForm.value).subscribe({
      next: (response) => {
        // Si la creación es exitosa, navega a la página de éxito con el ID del acuerdo
        this.router.navigate(['/acuerdos/success', response.id]);
      }
    });
  }
}
}

```

## 6.3 Módulo de Administración

### Gestión de Usuarios

```

// usuarios-page.component.ts - Componente para la gestión de usuarios
@Component({
  selector: 'app-usuarios-page', // Selector del componente
  template: `
    <!-- Tabla de PrimeNG para mostrar usuarios con paginación -->
    <p-table [value]="users" [paginator]="true" [rows]="10">
      <!-- Plantilla para el encabezado de la tabla -->
      <ng-template pTemplate="header">
        <tr>
          <th>Nombre</th>
          <th>Email</th>
          <th>Roles</th>

```

```

        <th>Estado</th>
        <th>Acciones</th>
    </tr>
</ng-template>
<!-- Plantilla para cada fila de datos (let-user crea variable local) --&gt;
&lt;ng-template pTemplate="body" let-user&gt;
    &lt;tr&gt;
        <!-- Muestra el nombre del usuario --&gt;
        &lt;td&gt;{{user.name}}&lt;/td&gt;
        <!-- Muestra el email del usuario --&gt;
        &lt;td&gt;{{user.email}}&lt;/td&gt;
        <!-- Une los roles con coma y espacio --&gt;
        &lt;td&gt;{{user.roles.join(', ')}}&lt;/td&gt;
        &lt;td&gt;
            <!-- Tag de PrimeNG con color según el estado del usuario --&gt;
            &lt;p-tag [value]="user.status"
                   [severity]="getSeverity(user.status)"&gt;
            &lt;/p-tag&gt;
        &lt;/td&gt;
        &lt;td&gt;
            <!-- Botón para editar usuario --&gt;
            &lt;button pButton icon="pi pi-pencil" (click)="editUser(user)"&gt;&lt;/button&gt;
            <!-- Botón para eliminar usuario --&gt;
            &lt;button pButton icon="pi pi-trash" (click)="deleteUser(user)"&gt;&lt;/button&gt;
        &lt;/td&gt;
    &lt;/tr&gt;
&lt;/ng-template&gt;
&lt;/p-table&gt;
`)

}

export class UsuariosPageComponent implements OnInit {
    // Array que contendrá la lista de usuarios
    users: User[] = [];

    // Constructor con inyección del servicio de usuarios
    constructor(private usuariosService: UsuariosService) {}

    // Método del ciclo de vida que se ejecuta al inicializar
    ngOnInit(): void {
        this.loadUsers(); // Carga los usuarios al iniciar el componente
    }

    // Método para cargar todos los usuarios desde el servicio
    loadUsers(): void {
        this.usuariosService.getAll().subscribe(users =&gt; {
            this.users = users; // Asigna los usuarios obtenidos al array local
        });
    }

    // Método para editar un usuario
    editUser(user: User): void {
        // Lógica de edición (implementar según necesidad)
    }

    // Método para eliminar un usuario
    deleteUser(user: User): void {
</pre>

```

```

    // Lógica de eliminación (implementar según necesidad)
}

// Método que determina el color del tag según el estado del usuario
getSeverity(status: string): string {
    return status === 'active' ? 'success' : 'danger'; // Verde si activo, rojo si inactivo
}
}

```

## 7. Servicios y API

### 7.1 Arquitectura de Servicios

#### Jerarquía de Servicios

```

services/
├── api/                  # Servicios que consumen API
│   ├── customer.service.ts
│   ├── roles.service.ts
│   └── persona.service.ts
├── utils/                 # Servicios utilitarios
│   ├── icon.service.ts
│   └── photo.service.ts
└── generic/               # Servicio genérico base
    └── service-generic.service.ts

```

### 7.2 Servicio Genérico Base

```

// service-generic.service.ts - Servicio genérico base para operaciones CRUD
import { Injectable, inject } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';
import { Observable } from 'rxjs';
import { environment } from '@env/environment';

// Servicio genérico que puede ser heredado por servicios específicos
@Injectable({
    providedIn: 'root' // Singleton a nivel de aplicación
})
export class ServiceGenericService<T> { // <T> indica que es un servicio genérico de tipo T
    // Inyección del cliente HTTP usando la función inject()
    protected http = inject(HttpClient);
    // URL base de la API desde la configuración del entorno
    protected apiUrl = environment.apiUrl;

    // Constructor que recibe el endpoint específico para cada servicio
    constructor(protected endpoint: string) {}

    // GET /api/{endpoint} - Obtiene todos los registros
    getAll(): Observable<T[]> {
        // Retorna un Observable que emitirá un array de tipo T
        return this.http.get<T[]>(`${this.apiUrl}/${this.endpoint}`);
    }

    // GET /api/{endpoint}/{id} - Obtiene un registro por ID
}

```

```

getById(id: number | string): Observable<T> {
  // Retorna un Observable que emitirá un objeto de tipo T
  return this.http.get<T>(`${this.apiUrl}/${this.endpoint}/${id}`);
}

// POST /api/{endpoint} - Crea un nuevo registro
create(entity: T): Observable<T> {
  // Envía el objeto entity al servidor y retorna el objeto creado
  return this.http.post<T>(`${this.apiUrl}/${this.endpoint}`, entity);
}

// PUT /api/{endpoint}/{id} - Actualiza un registro existente
update(id: number | string, entity: T): Observable<T> {
  // Envía el objeto entity actualizado al servidor
  return this.http.put<T>(`${this.apiUrl}/${this.endpoint}/${id}`, entity);
}

// DELETE /api/{endpoint}/{id} - Elimina un registro
delete(id: number | string): Observable<void> {
  // Elimina el registro y no retorna datos (void)
  return this.http.delete<void>(`${this.apiUrl}/${this.endpoint}/${id}`);
}

// GET con parámetros - Obtiene registros con parámetros de consulta
getWithParams(params: HttpParams): Observable<T[]> {
  // Retorna un Observable con los resultados filtrados según los parámetros
  return this.http.get<T[]>(`${this.apiUrl}/${this.endpoint}`, { params });
}
}

```

### 7.3 Servicios Específicos

#### **CustomerService**

```

// customer.service.ts - Servicio para gestionar clientes
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { ServiceGenericService } from '../utils/generic/service-generic.service';
import { Customer } from '@shared/models/customer.model';

@Injectable({
  providedIn: 'root' // Singleton a nivel de aplicación
})
// Extiende el servicio genérico especificando Customer como tipo
export class CustomerService extends ServiceGenericService<Customer> {
  constructor() {
    // Llama al constructor padre con el endpoint 'customers'
    super('customers');
  }

  // Métodos adicionales específicos del servicio de clientes

  // Obtiene un cliente por su número de documento
  getByDocument(documentNumber: string): Observable<Customer> {
    // Realiza una petición GET al endpoint específico de búsqueda por documento
    return this.http.get<Customer>(

```

```

` ${this.apiUrl}/${this.endpoint}/document/${documentNumber}`
);
}

// Obtiene todas las multas asociadas a un cliente
getMultasByCustomer(customerId: number): Observable<any[]> {
  // Realiza una petición GET para obtener las multas del cliente
  return this.http.get<any[]>(
    `${this.apiUrl}/${this.endpoint}/${customerId}/multas`
  );
}

// Actualiza parcialmente el perfil de un cliente
updateProfile(customerId: number, profile: Partial<Customer>): Observable<Customer> {
  // PATCH permite actualizar solo los campos especificados en 'profile'
  return this.http.patch<Customer>(
    `${this.apiUrl}/${this.endpoint}/${customerId}/profile`,
    profile // Objeto con los campos a actualizar
  );
}
}
}

```

## RolesService

```

// roles.service.ts - Servicio para gestionar roles de usuario
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { ServiceGenericService } from '../utils/generic/service-generic.service';
import { Role } from '@shared/models/role.model';

@Injectable({
  providedIn: 'root' // Singleton a nivel de aplicación
})
// Extiende el servicio genérico especificando Role como tipo
export class RolesService extends ServiceGenericService<Role> {
  constructor() {
    // Llama al constructor padre con el endpoint 'roles'
    super('roles');
  }

  // Obtiene todos los permisos asociados a un rol específico
  getRolePermissions(roleId: number): Observable<any[]> {
    // Realiza una petición GET al endpoint de permisos del rol
    return this.http.get<any[]>(
      `${this.apiUrl}/${this.endpoint}/${roleId}/permissions`
    );
  }

  // Asigna una lista de permisos a un rol
  assignPermissions(roleId: number, permissionIds: number[]): Observable<void> {
    // Realiza una petición POST enviando los IDs de permisos a asignar
    return this.http.post<void>(
      `${this.apiUrl}/${this.endpoint}/${roleId}/permissions`,
      { permissionIds } // Objeto con el array de IDs de permisos
    );
  }
}

```

```

    }

    // Obtiene todos los usuarios que tienen un rol específico
    getUsersByRole(roleId: number): Observable<any[]> {
        // Realiza una petición GET para obtener usuarios del rol
        return this.http.get<any>(`${this.apiUrl}/${this.endpoint}/${roleId}/users`);
    }
}
}

```

### PersonaService

```

// persona.service.ts - Servicio para gestionar personas
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { ServiceGenericService } from '../utils/generic/service-generic.service';
import { Persona } from '@shared/models/persona.model';

@Injectable({
    providedIn: 'root' // Singleton a nivel de aplicación
})
// Extiende el servicio genérico especificando Persona como tipo
export class PersonaService extends ServiceGenericService<Persona> {
    constructor() {
        // Llama al constructor padre con el endpoint 'personas'
        super('personas');
    }

    // Busca una persona por tipo y número de documento
    searchByDocument(documentType: string, documentNumber: string): Observable<Persona> {
        // Realiza una petición GET con parámetros de consulta
        return this.http.get<Persona>(`${this.apiUrl}/${this.endpoint}/search`,
            {
                params: { // Parámetros de la consulta: ?type=...&number=...
                    type: documentType, // Tipo de documento (ej: CC, TI, CE)
                    number: documentNumber // Número del documento
                }
            }
        );
    }

    // Crea una persona junto con su usuario asociado en una sola operación
    createWithUser(persona: Persona, userData: any): Observable<any> {
        // Realiza una petición POST con los datos de persona y usuario
        return this.http.post<any>(`${this.apiUrl}/${this.endpoint}/with-user`,
            { persona, user: userData } // Objeto que combina datos de persona y usuario
        );
    }
}

```

## 7.4 Servicios Utilitarios

## PhotoService

```
// photo.service.ts - Servicio para gestionar fotos y archivos
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root' // Singleton a nivel de aplicación
})
export class PhotoService {
  // URL base de la API
  private apiUrl = 'http://localhost:7262/api';

  // Constructor con inyección del HttpClient
  constructor(private http: HttpClient) {}

  // Sube una foto al servidor asociada a una entidad específica
  uploadPhoto(file: File, entityType: string, entityId: number): Observable<any> {
    // FormData permite enviar archivos mediante peticiones HTTP
    const formData = new FormData();
    formData.append('file', file); // Agrega el archivo
    formData.append('entityType', entityType); // Tipo de entidad (ej: 'customer', 'persona')
    formData.append('entityId', entityId.toString()); // ID de la entidad

    // Envía el FormData al servidor
    return this.http.post(`${this.apiUrl}/photos/upload`, formData);
  }

  // Obtiene una foto del servidor como Blob (datos binarios)
  getPhoto(photoId: number): Observable<Blob> {
    // responseType: 'blob' indica que esperamos datos binarios
    return this.http.get(`${this.apiUrl}/photos/${photoId}`, {
      responseType: 'blob'
    });
  }

  // Elimina una foto del servidor
  deletePhoto(photoId: number): Observable<void> {
    // Realiza una petición DELETE para eliminar la foto
    return this.http.delete<void>(`${this.apiUrl}/photos/${photoId}`);
  }

  // Convierte un archivo File a formato Base64 (útil para previsualización)
  fileToBase64(file: File): Promise<string> {
    // Retorna una Promise que se resuelve con el string Base64
    return new Promise((resolve, reject) => {
      const reader = new FileReader(); // API del navegador para leer archivos
      reader.readAsDataURL(file); // Lee el archivo como Data URL (Base64)
      reader.onload = () => resolve(reader.result as string); // Éxito: resuelve con el resultado
      reader.onerror = error => reject(error); // Error: rechaza con el error
    });
  }
}
```

## SessionPingService

```
// session-ping.service.ts - Servicio para mantener activa la sesión del usuario
import { Injectable, OnDestroy } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { interval, Subscription } from 'rxjs';
import { switchMap } from 'rxjs/operators';

@Injectable({
  providedIn: 'root' // Singleton a nivel de aplicación
})
export class SessionPingService implements OnDestroy {
  // Intervalo de ping: 5 minutos en milisegundos
  private pingInterval = 5 * 60 * 1000; // 5 minutos
  // Suscripción al intervalo de ping
  private subscription?: Subscription;

  // Constructor con inyección del HttpClient
  constructor(private http: HttpClient) {}

  // Inicia el ping periódico al servidor para mantener la sesión activa
  startPing(): void {
    // interval() emite valores cada 5 minutos
    this.subscription = interval(this.pingInterval)
      .pipe(
        // switchMap cancela la petición anterior si aún está en curso
        switchMap(() =>
          this.http.get('/api/auth/ping') // Petición GET al endpoint de ping
        )
      )
      .subscribe({
        next: () => console.log('Session ping successful'), // Ping exitoso
        error: (error) => {
          console.error('Session ping failed:', error); // Error en el ping
          this.handleSessionExpired(); // Maneja la sesión expirada
        }
      });
  }

  // Detiene el ping periódico
  stopPing(): void {
    // Cancela la suscripción si existe
    this.subscription?.unsubscribe();
  }

  // Maneja la expiración de sesión
  private handleSessionExpired(): void {
    localStorage.removeItem('token'); // Elimina el token de autenticación
    window.location.href = '/login'; // Redirige al login
  }

  // Hook del ciclo de vida: se ejecuta cuando el servicio se destruye
  ngOnDestroy(): void {
    this.stopPing(); // Asegura detener el ping al destruir el servicio
  }
}
```

```
    }
}
```

## 7.5 Manejo de Errores en Servicios

```
// error-handler.service.ts - Servicio centralizado para el manejo de errores HTTP
import { Injectable } from '@angular/core';
import { HttpErrorResponse } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { MessageService } from 'primeng/api';

@Injectable({
  providedIn: 'root' // Singleton a nivel de aplicación
})
export class ErrorHandlerService {
  // Constructor con inyección del servicio de mensajes de PrimeNG
  constructor(private messageService: MessageService) {}

  // Método principal para manejar errores HTTP
  handleError(error: HttpErrorResponse): Observable<never> {
    let errorMessage = 'Ocurrió un error'; // Mensaje por defecto

    // Determina si el error es del lado del cliente o del servidor
    if (error.error instanceof ErrorEvent) {
      // Error del lado del cliente (red, JavaScript, etc.)
      errorMessage = `Error: ${error.error.message}`;
    } else {
      // Error del lado del servidor - evalúa el código de estado HTTP
      switch (error.status) {
        case 400:
          errorMessage = 'Solicitud inválida'; // Bad Request
          break;
        case 401:
          errorMessage = 'No autorizado'; // Unauthorized
          break;
        case 403:
          errorMessage = 'Acceso denegado'; // Forbidden
          break;
        case 404:
          errorMessage = 'Recurso no encontrado'; // Not Found
          break;
        case 500:
          errorMessage = 'Error interno del servidor'; // Internal Server Error
          break;
        default:
          errorMessage = `Error: ${error.message}`; // Otros errores
      }
    }

    // Muestra el mensaje de error al usuario usando PrimeNG Toast
    this.messageService.add({
      severity: 'error', // Tipo de mensaje: error (rojo)
      summary: 'Error', // Título del mensaje
      detail: errorMessage // Detalle del error
    });
  }
}
```

```

    // Retorna un Observable que emite un error
    return throwError(() => new Error(errorMessage));
}
}

```

## 8. Rutas y Navegación

### 8.1 Configuración de Rutas Principales

```

// app.routes.ts - Configuración de rutas principales de la aplicación
import { Routes } from '@angular/router';
import { AuthGuard } from '@core/guards/auth.guard';
import { RoleGuard } from '@core/guards/role.guard';

// Array de configuración de rutas de la aplicación
export const routes: Routes = [
{
  path: '', // Ruta raíz (http://localhost:4200/)
  redirectTo: 'inicio', // Redirige a la página de inicio
  pathMatch: 'full' // Coincide solo si la ruta es exactamente ''
},
{
  path: 'inicio', // Ruta pública de página de inicio
  // Lazy loading del componente - se carga solo cuando se accede a la ruta
  loadComponent: () =>
    import('./features/auth/pages/pagina-Inicio/inicio.component')
      .then(m => m.InicioComponent)
},
{
  path: 'login', // Ruta de inicio de sesión
  loadComponent: () =>
    import('./features/auth/pages/login/login.component')
      .then(m => m.LoginComponent)
},
{
  path: 'register', // Ruta de registro de usuarios
  loadComponent: () =>
    import('./features/auth/pages/register/register.component')
      .then(m => m.RegisterComponent)
},
{
  path: 'identificacion', // Ruta para identificación
  // loadChildren carga un conjunto de rutas hijo de forma lazy
  loadChildren: () =>
    import('./features/auth/pages/identificacion/identificacion.routes')
},
{
  path: 'dashboard', // Ruta principal del dashboard (protegida)
  canActivate: [AuthGuard], // Guard que verifica si el usuario está autenticado
  loadComponent: () =>
    import('./layout/app.layout.component')
      .then(m => m.AppLayoutComponent),
  children: [ // Rutas hijas dentro del dashboard

```

```

{
  path: 'multas', // Módulo de gestión de multas
  loadChildren: () =>
    import('./features/multas/multas.routes')
},
{
  path: 'acuerdos', // Módulo de acuerdos de pago
  loadChildren: () =>
    import('./features/multas/acuerdos-pago/acuerdo-pago.routes')
},
{
  path: 'admin', // Módulo de administración (solo para ADMIN)
  canActivate: [RoleGuard], // Guard que verifica roles del usuario
  data: { roles: ['ADMIN'] }, // Metadata: roles requeridos
  loadChildren: () =>
    import('./features/admin/admin.routes')
},
{
  path: 'profile', // Ruta de perfil de usuario
  loadChildren: () =>
    import('./features/profile/perfil.routes')
},
{
  path: 'parameters', // Ruta de parámetros del sistema (solo ADMIN)
  canActivate: [RoleGuard], // Protegido por roles
  data: { roles: ['ADMIN'] }, // Solo accesible para rol ADMIN
  loadChildren: () =>
    import('./features/parameters.routes')
}
]
},
{
  path: 'access-denied', // Página de acceso denegado
  loadComponent: () =>
    import('./features/auth/pages/access-denied/access.component')
    .then(m => m.AccessComponent)
},
{
  path: '**', // Wildcard: captura cualquier ruta no definida
  redirectTo: 'inicio' // Redirige a inicio si la ruta no existe
}
];

```

## 8.2 Rutas de Módulos Específicos

### Multas Routes

```

// multas.routes.ts - Configuración de rutas del módulo de multas
import { Routes } from '@angular/router';

// Exporta las rutas del módulo de multas
export default [
{
  path: 'consultar', // Ruta para consultar multas
  // Carga el componente de forma lazy (solo cuando se accede)
  loadComponent: () =>

```

```

    import('../consultas/consultar-ingresar/consultar-ingresar.component')
      .then(m => m.ConsultarIngresarComponent)
},
{
  path: 'notificaciones', // Ruta para notificaciones de multas
  // Carga las rutas hijas del módulo de notificaciones
  loadChildren: () =>
    import('../notificaciones/notificaciones.routes')
},
{
  path: 'tipos', // Ruta para tipos de multas
  // Carga las rutas hijas del módulo de tipos de multas
  loadChildren: () =>
    import('../tipos/tipos-multas.routes')
},
{
  path: 'anexar', // Ruta para anexar multas
  // Carga las rutas desde un módulo hermano
  loadChildren: () =>
    import('../anexar-multas/anexar-multas.routes')
}
] as Routes; // Cast explícito a tipo Routes

```

## Admin Routes

```

// admin.routes.ts - Configuración de rutas del módulo de administración
import { Routes } from '@angular/router';

// Exporta las rutas del módulo de administración
export default [
{
  path: 'usuarios', // Ruta para gestión de usuarios
  // Carga las rutas del módulo de usuarios de forma lazy
  loadChildren: () =>
    import('../users/usuarios-page.routes')
},
{
  path: 'roles', // Ruta para gestión de roles
  // Carga las rutas del módulo de roles
  loadChildren: () =>
    import('../roles/roles-page.routes')
},
{
  path: 'personas', // Ruta para gestión de personas
  // Carga las rutas del módulo de personas
  loadChildren: () =>
    import('../personas/personas-page/personas-page.routes')
},
{
  path: 'permisos', // Ruta para gestión de permisos
  // Carga las rutas del módulo de permisos
  loadChildren: () =>
    import('../permissions/permisos-page.routes')
},
{

```

```

    path: 'formularios', // Ruta para gestión de formularios
    // Carga las rutas del módulo de formularios
    loadChildren: () =>
      import('./forms/form.routes')
},
{
  path: 'modulos', // Ruta para gestión de módulos del sistema
  // Carga las rutas del módulo de módulos
  loadChildren: () =>
    import('./module/module.routes')
}
] as Routes; // Cast explícito a tipo Routes

```

## 8.3 Guards

### AuthGuard

```

// auth.guard.ts - Guard para proteger rutas que requieren autenticación
import { inject } from '@angular/core';
import { Router, CanActivateFn } from '@angular/router';
import { AuthService } from '@core/services/auth.service';

// Función guard moderna de Angular (Functional Guard)
export const AuthGuard: CanActivateFn = (route, state) => {
  // Inyecta el servicio de autenticación usando inject()
  const authService = inject(AuthService);
  // Inyecta el router para navegación
  const router = inject(Router);

  // Verifica si el usuario está autenticado
  if (authService.isAuthenticated()) {
    return true; // Permite el acceso a la ruta
  }

  // Guardar URL para redireccionar después del login
  authService.redirectUrl = state.url;
  // Redirige al login si no está autenticado
  router.navigate(['/login']);
  return false; // Bloquea el acceso a la ruta
};

```

### RoleGuard

```

// role.guard.ts - Guard para proteger rutas por roles de usuario
import { inject } from '@angular/core';
import { Router, CanActivateFn } from '@angular/router';
import { AuthService } from '@core/services/auth.service';

// Función guard para verificar roles del usuario
export const RoleGuard: CanActivateFn = (route, state) => {
  // Inyecta el servicio de autenticación
  const authService = inject(AuthService);
  // Inyecta el router para navegación
  const router = inject(Router);

```

```

// Obtiene los roles requeridos desde la metadata de la ruta
const requiredRoles = route.data['roles'] as string[];
// Obtiene los roles del usuario autenticado
const userRoles = authService.getUserRoles();

// Verifica si el usuario tiene al menos uno de los roles requeridos
const hasRole = requiredRoles.some(role =>
  userRoles.includes(role) // Retorna true si encuentra el rol
);

// Si tiene el rol requerido, permite el acceso
if (hasRole) {
  return true; // Permite el acceso a la ruta
}

// Si no tiene el rol, redirige a página de acceso denegado
router.navigate(['/access-denied']);
return false; // Bloquea el acceso a la ruta
};

```

## 9. Gestión de Estado

### 9.1 Estrategia de Estado

El sistema utiliza **RxJS** y **BehaviorSubject** para gestión de estado reactivo.

#### StateService (Ejemplo)

```

// state.service.ts
import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable } from 'rxjs';

export interface AppState {
  user: User | null;
  multas: Multa[];
  notifications: Notification[];
  loading: boolean;
}

@Injectable({
  providedIn: 'root'
})
export class StateService {
  private initialState: AppState = {
    user: null,
    multas: [],
    notifications: [],
    loading: false
  };

  private state$ = new BehaviorSubject<AppState>(this.initialState);

  // Observables públicos
  readonly user$ = this.state$.pipe(map(state => state.user));
  readonly multas$ = this.state$.pipe(map(state => state.multas));
}

```

```

readonly notifications$ = this.state$.pipe(map(state => state.notifications));
readonly loading$ = this.state$.pipe(map(state => state.loading));

// Métodos para actualizar estado
setUser(user: User): void {
  this.updateState({ user });
}

setMultas(multas: Multa[]): void {
  this.updateState({ multas });
}

addNotification(notification: Notification): void {
  const currentNotifications = this.state$.value.notifications;
  this.updateState({
    notifications: [...currentNotifications, notification]
  });
}

 setLoading/loading: boolean): void {
  this.updateState({ loading });
}

resetState(): void {
  this.state$.next(this.initialState);
}

private updateState(partial: Partial<AppState>): void {
  this.state$.next({
    ...this.state$.value,
    ...partial
  });
}

// Getter para obtener estado actual (snapshot)
get currentState(): AppState {
  return this.state$.value;
}
}

```

## 9.2 Uso en Componentes

```

// component.ts
@Component({
  selector: 'app-dashboard',
  template: `
    <div *ngIf="loading$ | async">Cargando...</div>
    <div *ngIf="user$ | async as user">
      Bienvenido {{ user.name }}
    </div>
  `
})
export class DashboardComponent implements OnInit {
  user$ = this.stateService.user$;
  loading$ = this.stateService.loading$;
}

```

```

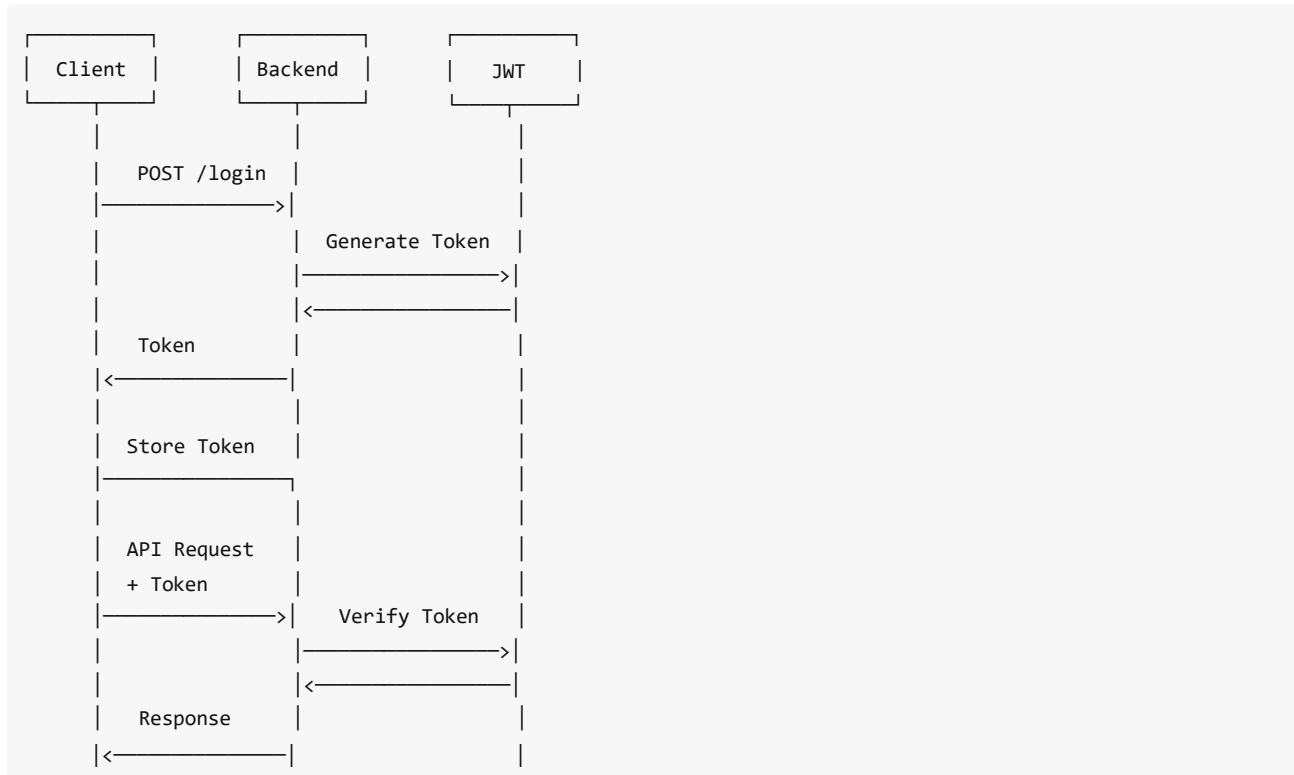
constructor(private stateService: StateService) {}

ngOnInit(): void {
  // El componente reacciona automáticamente a cambios de estado
}
}

```

## 10. Autenticación y Seguridad

### 10.1 Flujo de Autenticación



### 10.2 AuthService

```

// auth.service.ts
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, BehaviorSubject, tap } from 'rxjs';
import { Router } from '@angular/router';
import { jwtDecode } from 'jwt-decode';

export interface LoginRequest {
  email: string;
  password: string;
}

export interface AuthResponse {
  token: string;
  refreshToken: string;
  user: User;
}

```

```

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private apiUrl = 'http://localhost:8080/api/auth';
  private tokenKey = 'auth_token';
  private refreshTokenKey = 'refresh_token';

  private currentUserSubject = new BehaviorSubject<User | null>(null);
  public currentUser$ = this.currentUserSubject.asObservable();

  redirectUrl: string = '/dashboard';

  constructor(
    private http: HttpClient,
    private router: Router
  ) {
    this.loadUserFromToken();
  }

  login(credentials: LoginRequest): Observable<AuthResponse> {
    return this.http.post<AuthResponse>(`${this.apiUrl}/login`, credentials)
      .pipe(
        tap(response => {
          this.setSession(response);
          this.currentUserSubject.next(response.user);
        })
      );
  }

  register(userData: any): Observable<any> {
    return this.http.post(`.${this.apiUrl}/register`, userData);
  }

  logout(): void {
    localStorage.removeItem(this.tokenKey);
    localStorage.removeItem(this.refreshTokenKey);
    this.currentUserSubject.next(null);
    this.router.navigate(['/login']);
  }

  refreshToken(): Observable<AuthResponse> {
    const refreshToken = this.getRefreshToken();
    return this.http.post<AuthResponse>(`.${this.apiUrl}/refresh`, { refreshToken })
      .pipe(
        tap(response => {
          this.setSession(response);
        })
      );
  }

  isAuthenticated(): boolean {
    const token = this.getToken();
    if (!token) return false;
  }
}

```

```

    try {
      const decoded: any = jwtDecode(token);
      const expirationDate = new Date(decoded.exp * 1000);
      return expirationDate > new Date();
    } catch {
      return false;
    }
  }

  getToken(): string | null {
    return localStorage.getItem(this.tokenKey);
  }

  getRefreshToken(): string | null {
    return localStorage.getItem(this.refreshTokenKey);
  }

  getUserRoles(): string[] {
    const token = this.getToken();
    if (!token) return [];

    try {
      const decoded: any = jwtDecode(token);
      return decoded.roles || [];
    } catch {
      return [];
    }
  }

  hasRole(role: string): boolean {
    return this.getUserRoles().includes(role);
  }

  private setSession(authResult: AuthResponse): void {
    localStorage.setItem(this.tokenKey, authResult.token);
    localStorage.setItem(this.refreshTokenKey, authResult.refreshToken);
  }

  private loadUserFromToken(): void {
    const token = this.getToken();
    if (token && this.isAuthenticated()) {
      try {
        const decoded: any = jwtDecode(token);
        this.currentUserSubject.next(decoded.user);
      } catch {}
    }
  }
}

```

### 10.3 HTTP Interceptor

```

// auth.interceptor.ts
import { HttpInterceptorFn } from '@angular/common/http';
import { inject } from '@angular/core';
import { AuthService } from '@core/services/auth.service';

```

```

import { catchError, switchMap, throwError } from 'rxjs';

export const authInterceptor: HttpInterceptorFn = (req, next) => {
  const authService = inject(AuthService);
  const token = authService.getToken();

  // Clonar request y agregar token
  let authReq = req;
  if (token) {
    authReq = req.clone({
      setHeaders: {
        Authorization: `Bearer ${token}`
      }
    });
  }

  return next(authReq).pipe(
    catchError(error => {
      // Si es 401, intentar refresh token
      if (error.status === 401 && !req.url.includes('/auth/')) {
        return authService.refreshToken().pipe(
          switchMap(() => {
            // Reintentar request con nuevo token
            const newToken = authService.getToken();
            const retryReq = req.clone({
              setHeaders: {
                Authorization: `Bearer ${newToken}`
              }
            });
            return next(retryReq);
          }),
          catchError(refreshError => {
            authService.logout();
            return throwError(() => refreshError);
          })
        );
      }
      return throwError(() => error);
    })
  );
}

```

## 10.4 Seguridad en el Frontend

### Mejores Prácticas Implementadas

1. **Tokens JWT**: Autenticación basada en tokens
2. **HttpOnly Cookies**: Para refresh tokens (si aplica)
3. **Guards**: Protección de rutas
4. **Interceptors**: Manejo automático de tokens
5. **CSRF Protection**: Tokens anti-CSRF
6. **XSS Prevention**: Sanitización de inputs
7. **Timeout de Sesión**: Cierre automático por inactividad

### Sanitización de Inputs

```
// sanitize.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizer, SafeHtml } from '@angular/platform-browser';

@Pipe({
  name: 'sanitize'
})
export class SanitizePipe implements PipeTransform {
  constructor(private sanitizer: DomSanitizer) {}

  transform(value: string): SafeHtml {
    return this.sanitizer.sanitize(SecurityContext.HTML, value) || '';
  }
}
```

## 11. Integración con Backend

### 11.1 Comunicación con API REST

El frontend se comunica con un backend a través de servicios HTTP que consumen una API REST.

#### Endpoints Consumidos por el Frontend

Método	Endpoint	Descripción
POST	/api/auth/login	Autenticación de usuario
POST	/api/auth/register	Registro de nuevo usuario
GET	/api/customers	Obtener todos los clientes
GET	/api/customers/{id}	Obtener cliente por ID
POST	/api/multas	Crear nueva multa
GET	/api/multas/document/{doc}	Multas por documento
POST	/api/acuerdos	Crear acuerdo de pago
GET	/api/roles	Obtener todos los roles
POST	/api/users/{id}/roles	Asignar rol a usuario

### 11.2 Configuración de Entorno

```
// environment.ts
export const environment = {
  production: false,
  apiUrl: 'http://localhost:8080/api',
  apiVersion: 'v1',
  endpoints: {
    auth: '/auth',
    customers: '/customers',
    multas: '/multas',
    acuerdos: '/acuerdos',
    roles: '/roles',
    users: '/users',
```

```
    parameters: '/parameters'
},
timeout: 30000, // 30 segundos
retryAttempts: 3
};
```

## 11.3 Modelos de Datos

### Customer Model

```
// customer.model.ts
export interface Customer {
  id?: number;
  personId: number;
  email: string;
  password?: string;
  status: 'ACTIVE' | 'INACTIVE' | 'SUSPENDED';
  emailVerified: boolean;
  verificationCode?: string;
  createdAt?: Date;
  updatedAt?: Date;

  // Relaciones
  person?: Person;
  roles?: Role[];
  multas?: Multa[];
}

export interface Person {
  id?: number;
  documentTypeId: number;
  documentNumber: string;
  fullName: string;
  phone: string;
  gender: 'M' | 'F' | 'OTHER';
  cityId: number;
  address: string;
  photoUrl?: string;
}
```

### Multa Model

```
// multa.model.ts
export interface Multa {
  id?: number;
  comparendoNumber: string;
  infractionTypeId: number;
  personId: number;
  vehiclePlate?: string;
  infractionDate: Date;
  infractionLocation: string;
  amount: number;
  status: 'PENDING' | 'PAID' | 'IN AGREEMENT' | 'EXPIRED';
  dueDate: Date;
  discount?: number;
```

```

interest?: number;
totalAmount: number;
observations?: string;

// Relaciones
infractionType?: InfractionType;
person?: Person;
paymentAgreement?: PaymentAgreement;
}

export interface InfractionType {
  id?: number;
  code: string;
  description: string;
  category: 'LEVE' | 'GRAVE' | 'MUY_GRAVE';
  amountInSMMLV: number;
  articleNumber: string;
  licensePoints?: number;
}

```

## PaymentAgreement Model

```

// payment-agreement.model.ts
export interface PaymentAgreement {
  id?: number;
  customerId: number;
  agreementNumber: string;
  totalAmount: number;
  initialPayment: number;
  numberOfInstallments: number;
  installmentAmount: number;
  paymentFrequencyId: number;
  firstPaymentDate: Date;
  status: 'ACTIVE' | 'COMPLETED' | 'DEFAULTED' | 'CANCELLED';
  createdAt?: Date;

  // Relaciones
  multas?: Multa[];
  installments?: Installment[];
  paymentFrequency?: PaymentFrequency;
}

export interface Installment {
  id?: number;
  agreementId: number;
  installmentNumber: number;
  amount: number;
  dueDate: Date;
  paidDate?: Date;
  status: 'PENDING' | 'PAID' | 'OVERDUE';
  lateFee?: number;
}

```

## 11.4 DTOs (Data Transfer Objects)

```
// dtos.ts

// Login DTO
export interface LoginDTO {
  email: string;
  password: string;
}

// Register DTO
export interface RegisterDTO {
  // Person data
  documentTypeId: number;
  documentNumber: string;
  fullName: string;
  phone: string;
  gender: string;
  cityId: number;
  address: string;

  // Customer data
  email: string;
  password: string;
  confirmPassword: string;
}

// Create Payment Agreement DTO
export interface CreateAgreementDTO {
  multaIds: number[];
  initialPayment: number;
  numberOfInstallments: number;
  paymentFrequencyId: number;
  firstPaymentDate: string;
}

// Update Profile DTO
export interface UpdateProfileDTO {
  phone?: string;
  email?: string;
  address?: string;
  cityId?: number;
  photoUrl?: string;
}
```

## 12. Modelos de Datos Frontend

### 12.1 Interfaces TypeScript

El frontend trabaja con interfaces TypeScript que representan los datos recibidos de la API.

#### Person Interface

```
// person.model.ts
export interface Person {
  id?: number;
```

```
documentTypeId: number;
documentNumber: string;
fullName: string;
phone: string;
gender: 'M' | 'F' | 'OTHER';
cityId: number;
address: string;
photoUrl?: string;
createdAt?: Date;
updatedAt?: Date;
}
```

## Customer Interface

```
// customer.model.ts
export interface Customer {
  id?: number;
  personId: number;
  email: string;
  password?: string;
  status: 'ACTIVE' | 'INACTIVE' | 'SUSPENDED';
  emailVerified: boolean;
  verificationCode?: string;
  createdAt?: Date;
  updatedAt?: Date;
  person?: Person;
  roles?: Role[];
}
```

## Multa Interface

```
// multa.model.ts
export interface Multa {
  id?: number;
  comparendoNumber: string;
  infractionTypeId: number;
  personId: number;
  vehiclePlate?: string;
  infractionDate: Date;
  infractionLocation: string;
  amount: number;
  status: 'PENDING' | 'PAID' | 'IN AGREEMENT' | 'EXPIRED';
  dueDate: Date;
  discount?: number;
  interest?: number;
  totalAmount: number;
  observations?: string;
}
```

## Payment Agreement Interface

```
// payment-agreement.model.ts
export interface PaymentAgreement {
  id?: number;
```

```
customerId: number;
agreementNumber: string;
totalAmount: number;
initialPayment: number;
numberOfInstallments: number;
installmentAmount: number;
paymentFrequencyId: number;
firstPaymentDate: Date;
status: 'ACTIVE' | 'COMPLETED' | 'DEFAULTED' | 'CANCELLED';
createdAt?: Date;
}
```

## 13. Despliegue del Frontend

### 13.1 Build de Producción

El frontend se compila a archivos estáticos (HTML, CSS, JavaScript) para su despliegue:

```
# Build optimizado para producción
npm run build:prod

# Output en dist/sakai-ng/browser/
```

#### Configuración de Build

```
// angular.json - Configuración para producción
{
  "configurations": {
    "production": {
      "budgets": [
        {
          "type": "initial",
          "maximumWarning": "1mb",
          "maximumError": "5mb"
        }
      ],
      "outputHashing": "all",
      "optimization": true,
      "sourceMap": false,
      "namedChunks": false,
      "aot": true,
      "buildOptimizer": true
    }
  }
}
```

**Nota:** La carpeta `dist/` contiene todos los archivos estáticos listos para servir desde un servidor web.

### 13.2 Despliegue en Vercel

#### vercel.json

```
{
  "version": 2,
```

```

"builds": [
  {
    "src": "package.json",
    "use": "@vercel/static-build",
    "config": {
      "distDir": "dist/sakai-ng/browser"
    }
  }
],
"routes": [
  {
    "src": "/(.*)",
    "dest": "/index.html"
  }
]
}

```

#### Script de Deploy

```

# Instalar Vercel CLI
npm install -g vercel

# Deploy
vercel

# Deploy a producción
vercel --prod

```

### 13.3 Despliegue en AWS S3 + CloudFront

#### Script de Deploy

```

#!/bin/bash

# Build
npm run build:prod

# Upload a S3
aws s3 sync dist/sakai-ng/ s3://my-bucket-name --delete

# Iniciar caché de CloudFront
aws cloudfront create-invalidation \
  --distribution-id YOUR_DISTRIBUTION_ID \
  --paths "/*"

```

### 13.4 Despliegue con Docker (Frontend)

#### Dockerfile para Frontend

```

# Stage 1: Build de la aplicación Angular
FROM node:18-alpine AS build

WORKDIR /app

```

```

COPY package*.json .
RUN npm ci

COPY . .
RUN npm run build:prod

# Stage 2: Servir con Nginx
FROM nginx:alpine

# Copiar archivos compilados del frontend
COPY --from=build /app/dist/sakai-ng/browser /usr/share/nginx/html

# Copiar configuración de nginx
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

```

#### **nginx.conf (Frontend)**

```

server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    # Servir archivos estáticos del frontend
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Habilitar compresión
    gzip on;
    gzip_types text/css application/javascript application/json;
}

```

#### **Construcción de la imagen**

```

# Construir la imagen Docker del frontend
docker build -t multas-frontend:latest .

# Ejecutar contenedor
docker run -p 80:80 multas-frontend:latest

```

#### **13.5 CI/CD con GitHub Actions (Frontend)**

##### **.github/workflows/deploy.yml**

```

name: Deploy Frontend to Production

on:
  push:
    branches: [main]

```

```

jobs:
  build-and-deploy-frontend:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test -- --watch=false --browsers=ChromeHeadless

      - name: Build Frontend
        run: npm run build:prod

      - name: Deploy to Vercel
        uses: amondnet/vercel-action@v25
        with:
          vercel-token: ${{ secrets.VERCEL_TOKEN }}
          vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}
          vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID }}
          vercel-args: '--prod'

```

## 14. Testing del Frontend

### 14.1 Configuración de Testing

El frontend utiliza **Jasmine** como framework de testing y **Karma** como test runner.

#### karma.conf.js

```

module.exports = function(config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-coverage'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      jasmine: {
        random: false
      },
      clearContext: false
    }
  })
}

```

```

},
jasmineHtmlReporter: {
  suppressAll: true
},
coverageReporter: {
  dir: require('path').join(__dirname, './coverage'),
  subdir: '.',
  reporters: [
    { type: 'html' },
    { type: 'text-summary' },
    { type: 'lcovonly' }
  ]
},
reporters: ['progress', 'kjhtml'],
port: 9876,
colors: true,
logLevel: config.LOG_INFO,
autoWatch: true,
browsers: ['Chrome'],
singleRun: false,
restartOnFileChange: true
});
}
);

```

## 14.2 Unit Tests

### Service Test

```

// customer.service.spec.ts
import { TestBed } from '@angular/core/testing';
import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';
import { CustomerService } from './customer.service';

describe('CustomerService', () => {
  let service: CustomerService;
  let httpMock: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      providers: [CustomerService]
    });
    service = TestBed.inject(CustomerService);
    httpMock = TestBed.inject(HttpTestingController);
  });

  afterEach(() => {
    httpMock.verify();
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

  it('should retrieve all customers', () => {

```

```

const mockCustomers = [
  { id: 1, name: 'Test User', email: 'test@test.com' }
];

service.getAll().subscribe(customers => {
  expect(customers.length).toBe(1);
  expect(customers).toEqual(mockCustomers);
});

const req = httpMock.expectOne(` ${service['apiUrl']} /customers`);
expect(req.request.method).toBe('GET');
req.flush(mockCustomers);
});

it('should handle error on getAll', () => {
  service.getAll().subscribe(
    () => fail('should have failed'),
    (error) => {
      expect(error.status).toBe(500);
    }
  );
});

const req = httpMock.expectOne(` ${service['apiUrl']} /customers`);
req.flush('Error', { status: 500, statusText: 'Internal Server Error' });
});
});

```

## Component Test

```

// login.component.spec.ts
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { ReactiveFormsModule } from '@angular/forms';
import { Router } from '@angular/router';
import { of, throwError } from 'rxjs';

import { LoginComponent } from './login.component';
import { AuthService } from '@core/services/auth.service';

describe('LoginComponent', () => {
  let component: LoginComponent;
  let fixture: ComponentFixture<LoginComponent>;
  let mockAuthService: jasmine.SpyObj<AuthService>;
  let mockRouter: jasmine.SpyObj<Router>;

  beforeEach(async () => {
    mockAuthService = jasmine.createSpyObj('AuthService', ['login']);
    mockRouter = jasmine.createSpyObj('Router', ['navigate']);

    await TestBed.configureTestingModule({
      imports: [ReactiveFormsModule, LoginComponent],
      providers: [
        { provide: AuthService, useValue: mockAuthService },
        { provide: Router, useValue: mockRouter }
      ]
    }).compileComponents();
  });

  it('should log in user', () => {
    const email = 'test@test.com';
    const password = 'password';
    const expectedUser = { id: 1, name: 'Test User', email: email };

    fixture.detectChanges();

    const inputEmail = fixture.debugElement.nativeElement.querySelector('#email');
    const inputPassword = fixture.debugElement.nativeElement.querySelector('#password');

    inputEmail.value = email;
    inputPassword.value = password;

    fixture.detectChanges();

    const button = fixture.debugElement.nativeElement.querySelector('button');
    button.click();

    await fixture.whenStable();

    expect(mockAuthService.login).toHaveBeenCalledWith(email, password);
    expect(mockRouter.navigate).toHaveBeenCalledWith(['profile']);
  });
});

```

```
fixture = TestBed.createComponent(LoginComponent);
component = fixture.componentInstance;
fixture.detectChanges();
});

it('should create', () => {
  expect(component).toBeTruthy();
});

it('should have invalid form when empty', () => {
  expect(component.loginForm.valid).toBeFalsy();
});

it('should validate email field', () => {
  const email = component.loginForm.controls['email'];

  expect(email.valid).toBeFalsy();
  expect(email.errors?.['required']).toBeTruthy();

  email.setValue('invalid-email');
  expect(email.errors?.['email']).toBeTruthy();

  email.setValue('valid@email.com');
  expect(email.valid).toBeTruthy();
});

it('should call authService.login on valid form submit', () => {
  const mockResponse = { token: 'fake-token', user: {} };
  mockAuthService.login.and.returnValue(of(mockResponse));

  component.loginForm.setValue({
    email: 'test@test.com',
    password: 'password123'
  });

  component.onSubmit();

  expect(mockAuthService.login).toHaveBeenCalled();
  expect(mockRouter.navigate).toHaveBeenCalledWith(['/dashboard']);
});

it('should handle login error', () => {
  mockAuthService.login.and.returnValue(
    throwError(() => new Error('Login failed'))
  );

  component.loginForm.setValue({
    email: 'test@test.com',
    password: 'wrongpassword'
  });

  component.onSubmit();

  expect(mockAuthService.login).toHaveBeenCalled();
  expect(mockRouter.navigate).not.toHaveBeenCalled();
});
```

```
});  
});
```

### 14.3 E2E Tests (Opcional)

```
// e2e/login.e2e-spec.ts  
import { browser, by, element } from 'protractor';  
  
describe('Login Page', () => {  
  beforeEach(() => {  
    browser.get('/login');  
  });  
  
  it('should display login form', () => {  
    expect(element(by.css('form')).isPresent()).toBe(true);  
  });  
  
  it('should login successfully with valid credentials', () => {  
    element(by.css('input[formControlName="email"]')).sendKeys('test@test.com');  
    element(by.css('input[formControlName="password"]')).sendKeys('password123');  
    element(by.css('button[type="submit"]')).click();  
  
    expect(browser.getCurrentUrl()).toContain('/dashboard');  
  });  
});  
});
```

## 15. Mantenimiento y Troubleshooting del Frontend

### 15.1 Logs y Debugging

#### Logger Service (Frontend)

Servicio para registrar eventos y errores en el frontend:

```
// logger.service.ts  
import { Injectable } from '@angular/core';  
import { environment } from '@env/environment';  
  
export enum LogLevel {  
  DEBUG,  
  INFO,  
  WARN,  
  ERROR  
}  
  
@Injectable({  
  providedIn: 'root'  
})  
export class LoggerService {  
  private enabled = !environment.production;  
  
  debug(message: string, ...args: any[]): void {  
    this.log(LogLevel.DEBUG, message, args);  
  }
```

```

info(message: string, ...args: any[]): void {
  this.log(LogLevel.INFO, message, args);
}

warn(message: string, ...args: any[]): void {
  this.log(LogLevel.WARN, message, args);
}

error(message: string, error?: any): void {
  this.log(LogLevel.ERROR, message, [error]);

  // Enviar a servicio de monitoring
  if (environment.production) {
    this.sendToMonitoring(message, error);
  }
}

private log(level: LogLevel, message: string, args: any[]): void {
  if (!this.enabled && level < LogLevel.WARN) return;

  const timestamp = new Date().toISOString();
  const prefix = `[${timestamp}] [${LogLevel[level]}]`;

  switch (level) {
    case LogLevel.DEBUG:
      console.debug(prefix, message, ...args);
      break;
    case LogLevel.INFO:
      console.info(prefix, message, ...args);
      break;
    case LogLevel.WARN:
      console.warn(prefix, message, ...args);
      break;
    case LogLevel.ERROR:
      console.error(prefix, message, ...args);
      break;
  }
}

private sendToMonitoring(message: string, error: any): void {
  // Integración con Sentry, LogRocket, etc.
}
}

```

## 15.2 Problemas Comunes del Frontend

### 15.2.1 Error: CORS

**Problema:** Las peticiones HTTP desde el frontend fallan con error CORS

**Causa:** El backend no está configurado para aceptar peticiones desde el origen del frontend

**Solución (Frontend - Desarrollo local):**

Durante el desarrollo, puede configurar un proxy en Angular:

```
// proxy.conf.json
{
  "/api": {
    "target": "https://localhost:7286",
    "secure": false,
    "changeOrigin": true
  }
}
```

```
# Ejecutar con proxy
ng serve --proxy-config proxy.conf.json
```

**Nota:** En producción, el backend debe tener CORS configurado correctamente.

### 15.2.2 Error: Token Expirado

**Problema:** El token JWT expira y el usuario no puede hacer requests

**Solución:** Implementado en auth.interceptor.ts con refresh token automático

### 15.2.3 Error: Memory Leak

**Problema:** Subscripciones no se destruyen

**Solución:** Usar takeUntil pattern

```
@Component({...})
export class MyComponent implements OnDestroy {
  private destroy$ = new Subject<void>();

  ngOnInit(): void {
    this.myService.getData()
      .pipe(takeUntil(this.destroy$))
      .subscribe(data => {
        // handle data
      });
  }

  ngOnDestroy(): void {
    this.destroy$.next();
    this.destroy$.complete();
  }
}
```

## 15.3 Monitoreo y Performance del Frontend

### Performance Monitoring

```
// performance.service.ts
@Injectable({
  providedIn: 'root'
})
export class PerformanceService {
  measureNavigationTiming(): void {
    window.addEventListener('load', () => {
      const perfData = window.performance.timing;
```

```

const pageLoadTime = perfData.loadEventEnd - perfData.navigationStart;

console.log(`Page Load Time: ${pageLoadTime}ms`);

// Enviar a analytics (Google Analytics, etc.)
this.sendMetric('page_load_time', pageLoadTime);
});

}

private sendMetric(name: string, value: number): void {
    // Implementar integración con herramientas de monitoreo
    // Ejemplos: Google Analytics, Sentry, New Relic
}
}

```

### Optimizaciones de Performance Frontend

- **Lazy Loading:** Módulos cargados bajo demanda
- **AOT Compilation:** Compilación ahead-of-time
- **Tree Shaking:** Eliminación de código no usado
- **Minificación:** Reducción del tamaño de archivos
- **Gzip:** Compresión de archivos en producción

## 16. API Endpoints Consumidos por el Frontend

### 16.1 Lista de Endpoints

El frontend consume los siguientes endpoints de la API REST:

#### Authentication

POST	/api/auth/login	- Iniciar sesión
POST	/api/auth/register	- Registrar nuevo usuario
POST	/api/auth/refresh	- Renovar token
POST	/api/auth/logout	- Cerrar sesión
POST	/api/auth/verify-email	- Verificar email
POST	/api/auth/forgot-password	- Recuperar contraseña
POST	/api/auth/reset-password	- Restablecer contraseña

#### Customers

GET	/api/customers	- Listar clientes
GET	/api/customers/{id}	- Obtener cliente
POST	/api/customers	- Crear cliente
PUT	/api/customers/{id}	- Actualizar cliente
DELETE	/api/customers/{id}	- Eliminar cliente
GET	/api/customers/document/{number}	- Buscar por documento
GET	/api/customers/{id}/multas	- Multas del cliente
PATCH	/api/customers/{id}/profile	- Actualizar perfil

#### Multas

GET	/api/multas	- Listar multas
GET	/api/multas/{id}	- Obtener multa
POST	/api/multas	- Crear multa
PUT	/api/multas/{id}	- Actualizar multa

```

DELETE /api/multas/{id}           - Eliminar multa
GET    /api/multas/person/{personId} - Multas por persona
GET    /api/multas/comparendo/{number}- Buscar por comparendo
GET    /api/multas/status/{status}   - Filtrar por estado

```

### Payment Agreements

```

GET  /api/acuerdos          - Listar acuerdos
GET  /api/acuerdos/{id}      - Obtener acuerdo
POST /api/acuerdos          - Crear acuerdo
PUT  /api/acuerdos/{id}      - Actualizar acuerdo
GET  /api/acuerdos/customer/{customerId}- Acuerdos del cliente
GET  /api/acuerdos/{id}/installments - Cuotas del acuerdo
POST /api/acuerdos/{id}/pay-installment - Pagar cuota

```

### Roles & Permissions

```

GET  /api/roles          - Listar roles
GET  /api/roles/{id}      - Obtener rol
POST /api/roles          - Crear rol
PUT  /api/roles/{id}      - Actualizar rol
DELETE /api/roles/{id}    - Eliminar rol
GET  /api/roles/{id}/permissions - Permisos del rol
POST /api/roles/{id}/permissions - Asignar permisos

```

## 16.2 Ejemplos de Request/Response desde el Frontend

### Ejemplo: Login desde Angular

#### Código del Servicio (Frontend):

```

// auth.service.ts
login(credentials: LoginRequest): Observable<AuthResponse> {
  return this.http.post<AuthResponse>(`${this.apiUrl}/auth/login`, credentials);
}

```

#### Request enviado:

```

POST /api/auth/login
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "password123"
}

```

#### Response recibido:

```
{
  "token": "eyJhbGciOiJIUzI1NiIs...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIs...",
  "user": {
    "id": 1,
    "email": "user@example.com",
    "name": "John Doe",
    "role": "User"
  }
}
```

```
        "roles": ["USER"]
    }
}
```

#### Ejemplo: Crear Acuerdo de Pago desde Angular

##### Código del Servicio (Frontend):

```
// acuerdo-pago.service.ts
createAcuerdo(dto: CreateAgreementDTO): Observable<PaymentAgreement> {
    return this.http.post<PaymentAgreement>(`${this.apiUrl}/acuerdos`, dto);
}
```

##### Request enviado:

```
POST /api/acuerdos
Content-Type: application/json
Authorization: Bearer {token}

{
    "multaIds": [1, 2, 3],
    "initialPayment": 50000,
    "numberOfInstallments": 6,
    "paymentFrequencyId": 3,
    "firstPaymentDate": "2025-11-01"
}
```

##### Response recibido:

```
{
    "id": 10,
    "agreementNumber": "AGR-2025-00010",
    "totalAmount": 300000,
    "initialPayment": 50000,
    "numberOfInstallments": 6,
    "installmentAmount": 41666.67,
    "status": "ACTIVE",
    "createdAt": "2025-10-01T10:00:00Z"
}
```

---

## 17. Guías de Desarrollo del Frontend

### 17.1 Agregar un Nuevo Módulo Frontend

#### 1. Generar módulo con Angular CLI:

```
ng generate module features/nuevo-modulo --routing
cd src/app/features/nuevo-modulo
ng generate component pages/nuevo-componente
```

#### 2. Configurar rutas:

```
// nuevo-modulo.routes.ts
import { Routes } from '@angular/router';

export default [
  {
    path: '',
    loadComponent: () =>
      import('./pages/nuevo-componente/nuevo-componente.component')
        .then(m => m.NuevoComponenteComponent)
  }
] as Routes;
```

### 3. Agregar al app.routes.ts:

```
{
  path: 'nuevo-modulo',
  loadChildren: () =>
    import('./features/nuevo-modulo/nuevo-modulo.routes')
}
```

## 17.2 Crear un Nuevo Servicio

```
ng generate service core/services/api/nuevo-servicio
```

```
import { Injectable } from '@angular/core';
import { ServiceGenericService } from '../utils/generic/service-generic.service';

@Injectable({
  providedIn: 'root'
})
export class NuevoServicioService extends ServiceGenericService<MiModelo> {
  constructor() {
    super('mi-endpoint');
  }

  // Métodos específicos
}
```

## 17.3 Convenciones de Código

### Naming Conventions

- **Componentes:** nombre.component.ts (kebab-case)
- **Servicios:** nombre.service.ts (kebab-case)
- **Modelos:** nombre.model.ts (kebab-case)
- **Clases:** PascalCase
- **Variables/Funciones:** camelCase
- **Constantes:** UPPER\_SNAKE\_CASE

### Estructura de Componente

```
@Component({
  selector: 'app-mi-componente',
```

```

templateUrl: './mi-componente.component.html',
styleUrls: ['./mi-componente.component.scss']
})
export class MiComponenteComponent implements OnInit, OnDestroy {
// 1. Propiedades públicas
title: string = 'Mi Componente';

// 2. Propiedades privadas
private destroy$ = new Subject<void>();

// 3. Constructor con DI
constructor(
  private miServicio: MiServicio,
  private router: Router
) {}

// 4. Lifecycle hooks
ngOnInit(): void {
  this.loadData();
}

ngOnDestroy(): void {
  this.destroy$.next();
  this.destroy$.complete();
}

// 5. Métodos públicos
loadData(): void {
  // ...
}

// 6. Métodos privados
private processData(): void {
  // ...
}
}

```

## 17.4 Git Workflow

### Branch Strategy

```

main           Production
└── develop    Development
    ├── feature/* New features
    ├── bugfix/* Bug fixes
    └── hotfix/* Production hotfixes

```

### Commit Messages

```

feat: Add payment agreement module
fix: Resolve CORS issue in authentication
docs: Update API documentation
style: Format code with Prettier
refactor: Simplify customer service logic

```

```
test: Add unit tests for login component
chore: Update dependencies
```

## Apéndices

### A. Variables de Entorno

```
// environment.prod.ts
export const environment = {
  production: true,
  apiUrl: 'https://api.sistemamultas.com',
  apiVersion: 'v1',
  enableDebug: false,
  tokenKey: 'auth_token',
  sessionTimeout: 1800000,
  recaptcha: {
    siteKey: 'YOUR_PRODUCTION_RECAPTCHA_KEY'
  },
  analytics: {
    googleAnalyticsId: 'UA-XXXXX-X'
  },
  sentry: {
    dsn: 'https://xxx@sentry.io/xxx'
  }
};
```

### B. Comandos Útiles

```
# Desarrollo
ng serve                                # Servidor de desarrollo
ng serve --open                            # Abrir en navegador
ng serve --port 4300                      # Puerto personalizado

# Build
ng build                                    # Build desarrollo
ng build --configuration production        # Build producción
ng build --watch                           # Build continuo

# Testing
ng test                                     # Unit tests
ng test --code-coverage                     # Con coverage
ng test --watch=false --browsers=Chrome    # Una sola ejecución

# Code Quality
ng lint                                     # Linter
npm run format                             # Formatear código

# Generadores
ng g component features/nuevo/pages/page # Componente
ng g service core/services/api/servicio   # Servicio
ng g guard core/guards/mi-guard           # Guard
ng g pipe shared/pipes/mi-pipe             # Pipe
ng g directive shared/directives/dir      # Directiva
```

## C. Recursos y Referencias

### Documentación Oficial

- [Angular](#)
- [PrimeNG](#)
- [RxJS](#)
- [TypeScript](#)

### Herramientas

- [Angular DevTools](#)
- [Augury](#)
- [VSCode Extensions](#)

# PARTE II: BACKEND - SISTEMA DE CONTROL DE MULTAS CIUDADANAS

## 18. Información General del Backend

### 18.1 Descripción

Sistema backend para la gestión y control de multas ciudadanas, desarrollado con arquitectura multicapa en .NET 8. El sistema permite administrar infracciones ciudadanas, usuarios, pagos, acuerdos de pago, y genera reportes de inspectoría.

### 18.2 Tecnologías Principales del Backend

- **Framework:** .NET 8.0
- **Lenguaje:** C#
- **Arquitectura:** Clean Architecture / N-Capas
- **Bases de Datos:** SQL Server, PostgreSQL, MySQL (soporte multi-base de datos)
- **ORM:** Entity Framework Core 9.0.4
- **Autenticación:** JWT (JSON Web Tokens)
- **Validación:** FluentValidation 12.0.0
- **Mapeo de Objetos:** AutoMapper 14.0.0
- **Documentación API:** Swagger/OpenAPI
- **Contenedores:** Docker

## 19. Arquitectura del Backend

### 19.1 Estructura de Capas

El proyecto backend sigue una arquitectura en capas claramente definida:

```
taller/
└── Web/                      # Capa de Presentación (API REST)
└── Business/                  # Capa de Lógica de Negocio
└── Data/                      # Capa de Acceso a Datos
└── Entity/                    # Capa de Entidades y DTOs
└── Utilities/                 # Utilidades y Helpers
└── Helpers/                   # Funciones auxiliares
└── Template/                  # Plantillas (emails, PDFs)
└── Templates/                 # Plantillas adicionales
└── Strategy/                  # Patrones de estrategia
└── ControlDeComparando.Tests/ # Pruebas unitarias
```

## 19.2 Proyectos Adicionales

```
Merge_Back/
├── taller/          # Sistema principal
├── Gateway/         # API Gateway
└── PublicAPI/       # API pública
```

# 20. Capa Web Backend (Presentación)

## 20.1 Configuración del Proyecto Backend

Archivo: Web.csproj

**Paquetes NuGet principales:**

- Microsoft.AspNetCore.Authentication.JwtBearer 8.0.15
- Microsoft.AspNetCore.Authentication.Google 8.0.15
- FluentValidation.AspNetCore 11.3.1
- Swashbuckle.AspNetCore 6.6.2
- Microsoft.EntityFrameworkCore.SqlServer 9.0.4
- Npgsql.EntityFrameworkCore.PostgreSQL 9.0.4
- Pomelo.EntityFrameworkCore.MySql 9.0.0

## 20.2 Configuración de la Aplicación Backend

Archivo: Program.cs

**Servicios configurados:**

1. Controllers y Swagger
2. FluentValidation
3. Servicios de aplicación personalizados
4. Configuración JWT
5. Base de datos dinámica (multi-provider)
6. Autenticación y autorización
7. CORS
8. Caché en memoria

**Middleware Pipeline:**

1. Archivos estáticos
2. Swagger (Dev/Prod)
3. CORS
4. Autenticación
5. Autorización
6. Controladores
7. Migraciones automáticas

## 20.3 Configuración Backend (appsettings.json)

**Configuraciones clave:**

**Bases de Datos**

```
"MigrationProvider": "SqlServer",
"ConnectionStrings": {
    "SqlServer": "Server=localhost,1433;Database=controlComparendo;...",
    "Postgres": "Host=localhost;Port=5433;Database=controlComparendo;..."}
```

```
" MySql": "Server=127.0.0.1;Port=3307;Database=controlComparendo;..."  
}
```

## JWT

```
"Jwt": {  
    "Issuer": "WebCDCP",  
    "Audience": "WebCDCP",  
    "AccessTokenExpirationMinutes": 15,  
    "RefreshTokenExpirationDays": 7  
}
```

## Sesiones de Usuario

```
"Auth": {  
    "IdleMinutes": 1,  
    "AbsoluteMinutes": 120  
}
```

## reCAPTCHA

```
"Recaptcha": {  
    "SecretKey": "6LcEs7grAAAAAPaoAy-k5kx1pNhWfsEihOph6bJ_ ",  
    "MinScore": 0.5  
}
```

## SMTP (Correo electrónico)

```
"SmtpSettings": {  
    "Host": "smtp.gmail.com",  
    "Port": 587,  
    "EnableSsl": true,  
    "Email": "camiloandreslosada901@gmail.com"  
}
```

a

## Scheduler (Tareas programadas)

```
"Scheduler": {  
    "TzId": "America/Bogota",  
    "Hour": 16,  
    "Minute": 21  
}
```

## Acuerdos de Pago

```
"PaymentAgreementInterestOptions": {  
    "MonthlyRate": 0.02,  
    "GracePeriodDays": 30,
```

```
        "DailyDivisor": 30  
    }  
}
```

## 20.4 Controladores Backend

**Ubicación:** Web/Controllers/

### Controladores Principales

1. **AuthController.cs** - Autenticación
2. **LoginController.cs** - Login y gestión de sesiones
3. **TouristicAttractionsController.cs** - Atracciones turísticas
4. **VerificationController.cs** - Verificaciones

### Controladores de Entidades

**Ubicación:** Web/Controllers/Implements/Entities/

- Gestión de documentos de infracción
- Gestión de tipos de infracción
- Gestión de pagos
- Gestión de acuerdos de pago
- Gestión de usuarios
- Gestión de notificaciones

### Controladores de Seguridad

**Ubicación:** Web/Controllers/Implements/Security/

- Gestión de usuarios
- Gestión de roles
- Gestión de permisos
- Gestión de módulos y formularios

---

## 21. Capa Business (Lógica de Negocio)

### 21.1 Configuración del Proyecto Business

**Archivo:** Business.csproj

#### Paquetes principales:

- AutoMapper 14.0.0
- FluentValidation 12.0.0
- Google.Apis.Auth 1.69.0
- Microsoft.Playwright 154.0 (automatización web)

#### Referencias de proyecto:

- Data
- Entity
- Helpers
- Template
- Utilities

### 21.2 Servicios de Negocio

#### Servicios de Entidades

**Ubicación:** Business/Services/Entities/

1. **DocumentInfractionServices.cs** - Gestión de documentos de infracciones

2. **FineCalculationDetailService.cs** - Cálculo de detalles de multas
3. **InspectoraReportService.cs** - Reportes de inspectoría
4. **PaymentAgreementServices.cs** - Acuerdos de pago e intereses
5. **TypeInfractionService.cs** - Tipos de infracciones
6. **TypePaymentServices.cs** - Tipos de pago
7. **UserInfractionServices.cs** - Infracciones de usuarios
8. **UserNotificationService.cs** - Notificaciones a usuarios
9. **ValueSmldvService.cs** - Valores de SMLDV

### Servicios de Seguridad

**Ubicación:** Business/Services/Security/

1. **AuthService.cs** - Autenticación (login, registro, Google Auth)
2. **AuthSessionService.cs** - Gestión de sesiones de usuario
3. **UserService.cs** - CRUD de usuarios
4. **PersonService.cs** - Gestión de personas
5. **RolService.cs** - Gestión de roles
6. **RolUserService.cs** - Asignación de roles a usuarios
7. **RolFormPermissionService.cs** - Permisos de formularios por rol
8. **FormService.cs** - Gestión de formularios
9. **ModuleService.cs** - Gestión de módulos
10. **FormModuleService.cs** - Relación formularios-módulos
11. **PermissionService.cs** - Gestión de permisos

### Otros Servicios

1. **ApiColombiaGatewayService.cs** - Integración con API de Colombia
2. **DiscountService.cs** - Cálculo de descuentos en multas
3. **TouristicAttractionService.cs** - Atracciones turísticas

### Servicios PDF

**Ubicación:** Business/Services/PDF/

Generación de documentos PDF para el sistema.

---

## 22. Capa Data (Acceso a Datos)

### 22.1 Configuración del Proyecto Data

**Archivo:** Data.csproj

**Paquetes principales:**

- AutoMapper 14.0.0
- Microsoft.EntityFrameworkCore 9.0.4
- Microsoft.EntityFrameworkCore.SqlServer 9.0.4
- Npgsql.EntityFrameworkCore.PostgreSQL 9.0.4

**Referencia de proyecto:**

- Entity
- Utilities

### 22.2 Estructura Data

```
Data/
|--- Interfaces/      # Interfaces de repositorios
```

```
|── Repository/      # Implementación de repositorios  
└── Services/       # Servicios de datos
```

## 22.3 Patrón Repository

El proyecto utiliza el patrón Repository para abstraer el acceso a datos, permitiendo:

- Desacoplamiento de la lógica de negocio
- Facilidad para cambiar el proveedor de base de datos
- Mejor testabilidad

---

## 23. Capa Entity (Entidades y DTOs)

### 23.1 Configuración del Proyecto Entity

**Archivo:** Entity.csproj

**Paquetes principales:**

- Microsoft.EntityFrameworkCore 9.0.4
- Microsoft.AspNetCore.Authentication 2.3.0
- Microsoft.AspNetCore.Http.Abstractions 2.3.0
- Newtonsoft.Json 13.0.3
- Pomelo.EntityFrameworkCore.MySql 9.0.0

### 23.2 Estructura Entity

```
Entity/  
|── Domain/  
|   |── Enums/      # Enumeraciones  
|   |── Interfaces/  # Interfaces de dominio  
|   |── Models/  
|   |   |── Base/      # Modelos base  
|   |   |── Implements/ # Implementaciones  
|   |       |── Entities/    # Entidades de negocio  
|   |       |── ModelSecurity/ # Entidades de seguridad  
|   |── DTOS/        # Data Transfer Objects  
|   |── ConfigurationsBase/ # Configuraciones de EF Core  
|   |── Migrations/   # Migraciones de base de datos  
|   |── Init/         # Datos iniciales  
|   |── Infrastructure/ # Infraestructura de datos
```

### 23.3 Enumeraciones Backend

**Ubicación:** Entity/Domain/Enums/

1. **DatabaseType.cs** - Tipos de base de datos (SqlServer, PostgreSQL, MySql)
2. **DeleteType.cs** - Tipos de eliminación (lógica/física)
3. **EstadoMultas.cs** - Estados de multa
4. **GetAllType.cs** - Tipos de consulta
5. **TipoUsuario.cs** - Tipos de usuario

### 23.4 Entidades Principales Backend

**Entidades de Negocio**

**Ubicación:** Entity/Domain/Models/Implements/Entities/

1. **DocumentInfraction** - Documentos de infracciones

2. **TypeInfraction** - Tipos de infracciones
3. **UserInfraction** - Infracciones de usuarios
4. **TypePayment** - Tipos de pago
5. **PaymentAgreement** - Acuerdos de pago
6. **FineCalculationDetail** - Detalles de cálculo de multas
7. **InspectorReport** - Reportes de inspectoría
8. **AddFines** - Adición de multas

### Entidades de Seguridad Backend

**Ubicación:** Entity/Domain/Models/Implements/ModelSecurity/

1. **User** - Usuarios del sistema
2. **Person** - Personas
3. **Rol** - Roles
4. **RolUser** - Relación Usuario-Rol
5. **Module** - Módulos del sistema
6. **Form** - Formularios
7. **FormModule** - Relación Formulario-Módulo
8. **Permission** - Permisos
9. **RolFormPermission** - Permisos por rol y formulario

### 23.5 Modelos Base Backend

**BaseModel** y  **BaseModelGeneric**: Proporcionan propiedades comunes como:

- Id
- CreatedAt
- UpdatedAt
- DeletedAt (para borrado lógico)
- State

### 23.6 Interfaces de Dominio Backend

1. **IApplicationDbContext** - Interfaz del contexto de base de datos
2. **IAuditService** - Servicio de auditoría
3. **IDbContextFactory** - Fábrica de contextos de base de datos
4. **IHasId** - Interface para entidades con Id
5. **ISupportLogicalDelete** - Soporte para borrado lógico

---

## 24. Capa Utilities Backend

### 24.1 Estructura Utilities

```
Utilities/  
|__ Custom/          # Utilidades personalizadas  
|__ Exceptions/     # Excepciones personalizadas
```

### 24.2 Funcionalidades Utilities

- Manejo de excepciones personalizadas
- Utilidades de validación
- Helpers de formato y conversión
- Constantes del sistema

---

## 25. Capa Helpers Backend

Funciones auxiliares para:

- Operaciones comunes
  - Formateo de datos
  - Conversiones
  - Validaciones adicionales
- 

## 26. Capa Template Backend

Gestión de plantillas para:

- Emails (notificaciones, verificación, recuperación de contraseña)
  - PDFs (comprobados, reportes, acuerdos de pago)
  - Reportes del sistema
- 

## 27. Seguridad Backend

### 27.1 Autenticación JWT

**Configuración:**

- Issuer: WebCDCP
- Audience: WebCDCP
- Access Token: 15 minutos
- Refresh Token: 7 días

### 27.2 Gestión de Sesiones Backend

**Archivo:** AuthSessionService.cs

**Características:**

- Control de sesiones activas por usuario
- Timeout de inactividad: 1 minuto (configurable)
- Timeout absoluto: 120 minutos
- Validación de tokens

### 27.3 OAuth 2.0 - Google

Integración con Google Authentication mediante:

- Microsoft.AspNetCore.Authentication.Google
- Google.Apis.Auth

### 27.4 reCAPTCHA Backend

Protección contra bots con Google reCAPTCHA v3:

- Score mínimo: 0.5
- Validación en endpoints críticos

### 27.5 CORS Backend

Configuración de orígenes permitidos:

- Frontend: <http://localhost:4200>
- 

## 28. Base de Datos Backend

### 28.1 Soporte Multi-Base de Datos

El sistema soporta 3 proveedores de bases de datos:

1. **SQL Server** (puerto 1433)
2. **PostgreSQL** (puerto 5433)
3. **MySQL** (puerto 3307)

## 28.2 Migraciones Backend

**Estrategia:** Migraciones automáticas al iniciar la aplicación

**Configuración:**

```
"MigrateOnStartupTargets": ["SqlServer", "Postgres", " MySql"]
```

**Proveedor por defecto:** SqlServer

## 28.3 Entity Framework Core

**Versión:** 9.0.4

**Características utilizadas:**

- Code First
- Migrations
- Fluent API
- Data Annotations
- Change Tracking
- Lazy/Eager Loading

---

## 29. Patrones de Diseño Utilizados en Backend

### 29.1 Repository Pattern

Abstracción del acceso a datos en la capa Data.

### 29.2 Strategy Pattern

**Carpeta:** Strategy/

Implementación de diferentes estrategias de negocio.

### 29.3 Dependency Injection

Inyección de dependencias nativa de .NET Core en todos los servicios.

### 29.4 Unit of Work

Gestión transaccional en las operaciones de base de datos.

### 29.5 DTO Pattern

Separación entre modelos de dominio y objetos de transferencia de datos.

### 29.6 Factory Pattern

Creación dinámica de contextos de base de datos según el proveedor.

---

## 30. Integración de Servicios Externos Backend

### 30.1 API Colombia Gateway

**Servicio:** ApiColombiaGatewayService

Integración con servicios de datos de Colombia (municipios, departamentos, etc.).

### 30.2 SMTP - Gmail

Envío de correos electrónicos:

- Notificaciones
- Recuperación de contraseña
- Confirmación de registro
- Alertas de pagos

### 30.3 Google Authentication Backend

Autenticación mediante cuentas de Google.

### 30.4 Microsoft Playwright

**Paquete:** Microsoft.Playwright 1.54.0

**Uso:** Automatización de navegadores para scraping o generación de capturas.

---

## 31. Validaciones Backend

### 31.1 FluentValidation

**Versión:** 12.0.0

**Validadores implementados:**

#### 1. DocumentInfraction Validator

- Validación de documentos de infracciones

#### 2. InspectorReport Validator

- Validación de reportes de inspectoría

**Ubicación:** Business/validaciones/

**Características:**

- Validaciones fluidas y legibles
- Mensajes personalizados
- Validaciones asíncronas
- Validaciones condicionales

---

## 32. Mapeo de Objetos Backend

### 32.1 AutoMapper

**Versión:** 14.0.0

**Ubicación:** Web/AutoMapper/

**Perfiles de mapeo:**

- Entity → DTO
- DTO → Entity
- Entity → ViewModel

- 
- Relaciones complejas
- 

## 33. Funcionalidades del Sistema Backend

### 33.1 Gestión de Infracciones

1. Registro de comparendos
2. Tipos de infracciones
3. Documentos adjuntos
4. Estados de multas
5. Cálculo de valores (SMLDV)

### 33.2 Gestión de Pagos Backend

1. Tipos de pago
2. Acuerdos de pago
3. Cálculo de intereses (2% mensual)
4. Descuentos por pronto pago
5. Período de gracia (30 días)

### 33.3 Gestión de Usuarios Backend

1. Registro y autenticación
2. Roles y permisos
3. Sesiones activas
4. Perfil de usuario
5. Notificaciones

### 33.4 Reportes Backend

1. Reportes de inspectoría
2. Reportes de multas
3. Reportes de pagos
4. Estadísticas del sistema

### 33.5 Sistema de Permisos Backend

**Modelo:** RBAC (Role-Based Access Control)

**Niveles:**

- Usuario → Rol(es)
  - Rol → Permisos sobre Formularios
  - Formularios → Módulos
- 

## 34. Middleware Backend

### 34.1 Middleware Implementados

**Ubicación:** Web/Middleware/

Posibles middlewares personalizados:

- Validación de sesiones
  - Logging
  - Manejo de excepciones
  - Rate limiting
- 

## 35. Workers y Background Services

## **35.1 WebBackgroundService**

**Ubicación:** Web/WebBackgroundService/

**Tareas programadas:**

- Verificación de acuerdos de pago vencidos
  - Cálculo automático de intereses
  - Envío de notificaciones programadas
  - Horario configurado: 16:21 (America/Bogota)
- 

## **36. Pruebas Backend**

### **36.1 Proyecto de Pruebas**

**Nombre:** ControlDeComparando.Tests

**Ubicación:** ControlDeComparando.Tests/

**Framework de pruebas:** xUnit / NUnit / MSTest

**Alcance:**

- Pruebas unitarias de servicios
  - Pruebas de validadores
  - Pruebas de lógica de negocio
  - Mocks de repositorios
- 

## **37. Docker Backend**

### **37.1 Dockerfile**

**Ubicación:** Web/Dockerfile

**Características:**

- Target OS: Linux
- Multi-stage build
- Optimización de capas

### **37.2 Docker Ignore**

**Archivo:** .dockerignore

Exclusión de archivos innecesarios en la imagen Docker.

---

## **38. Swagger / OpenAPI**

### **38.1 Configuración Swagger**

**Disponible en:** /swagger

**Versión de API:** v1

**Entornos activos:** Development y Production

**Características:**

- Documentación automática de endpoints
- Pruebas interactivas

- Esquemas de modelos
  - Autenticación JWT integrada
- 

## 39. Infraestructura Backend

### 39.1 Servicios de Infraestructura

**Ubicación:** Web/Infrastructure/

Configuración de:

- DbContext Factory
  - Migraciones automáticas
  - Conexión a múltiples bases de datos
  - Caché
- 

## 40. Configuraciones Personalizadas Backend

### 40.1 Web Configurations

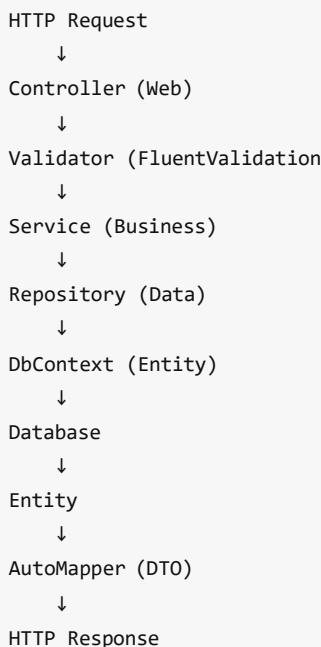
**Ubicación:** Web/Configurations/

Extensiones de configuración para:

- Servicios de aplicación
  - CORS
  - Autenticación JWT
  - Base de datos
  - Validadores
- 

## 41. Flujo de Datos Backend

### 41.1 Request Pipeline



## 42. Modelos de Datos Principales Backend

### 42.1 Sistema de Infracciones Backend

**UserInfraction** (Infracción de Usuario)

- Id del usuario
- Tipo de infracción
- Fecha
- Estado
- Valor SMLDV
- Documentos asociados

**DocumentInfraction** (Documento de Infracción)

- Referencia a UserInfraction
- Tipo de documento
- Ruta del archivo
- Fecha de carga

**TypeInfraction** (Tipo de Infracción)

- Código
- Descripción
- Valor base en SMLDV

**FineCalculationDetail** (Detalle de Cálculo de Multa)

- Valor base
- Descuentos aplicados
- Intereses calculados
- Total a pagar

### 42.2 Sistema de Pagos Backend

**PaymentAgreement** (Acuerdo de Pago)

- Usuario
- Infracción
- Número de cuotas
- Tasa de interés (2% mensual)
- Fecha de inicio
- Estado

**TypePayment** (Tipo de Pago)

- Efectivo
- Transferencia
- Tarjeta
- Acuerdo de pago

### 42.3 Sistema de Seguridad Backend

**User** (Usuario)

- Username
- Email
- Password (hasheado)
- Estado
- Roles asociados

**Person** (Persona)

- Datos personales
- Relación con User

#### Rol (Rol)

- Nombre
- Descripción
- Permisos

#### RolFormPermission (Permiso)

- Rol
- Formulario
- Permisos (CRUD)

---

## 43. Estados del Sistema Backend

### 43.1 Estados de Multa (EstadoMulta)

1. **Pendiente:** Multa registrada, pendiente de pago
2. **Pagada:** Multa completamente pagada
3. **En Acuerdo:** Con acuerdo de pago activo
4. **Vencida:** Pasó el período de gracia sin pago
5. **Coactivo:** En proceso coactivo

---

## 44. Lógica de Negocio Crítica Backend

### 44.1 Cálculo de Intereses

**Servicio:** PaymentAgreementServices

**Fórmula:**

- Tasa mensual: 2%
- Días de gracia: 30
- Divisor diario: 30
- Interés = Valor \* (0.02 / 30) \* días\_mora

### 44.2 Descuentos Backend

**Servicio:** DiscountService

**Reglas:**

- Pronto pago: Descuento según días desde infracción
- Pago total: Mayor descuento
- Acuerdos de pago: Sin descuento de pronto pago

### 44.3 Generación de Reportes Backend

**Servicio:** InspectorReportService

**Tipos:**

- Reportes diarios
- Reportes mensuales
- Reportes por usuario
- Reportes por tipo de infracción

## 45. Notificaciones Backend

### 45.1 Canales de Notificación

1. Email (SMTP)
  - Confirmación de registro
  - Recuperación de contraseña
  - Notificaciones de pagos
  - Recordatorios de vencimiento
  - Alertas administrativas

### 45.2 UserNotificationService

Gestión de notificaciones a usuarios dentro del sistema.

---

## 46. Consideraciones de Seguridad Backend

### 46.1 Buenas Prácticas Implementadas en Backend

1. Contraseñas hasheadas
  2. JWT con expiración
  3. Refresh tokens
  4. HTTPS redirection
  5. CORS configurado
  6. reCAPTCHA en formularios
  7. Validación de entrada (FluentValidation)
  8. SQL Injection protection (EF Core)
  9. Control de sesiones
  10. Borrado lógico de datos
- 

## 47. Despliegue Backend

### 47.1 Prerrequisitos Backend

1. .NET 8.0 SDK
2. Una de las siguientes bases de datos:
  - SQL Server 2019+
  - PostgreSQL 13+
  - MySQL 8+
3. Docker (opcional)

### 47.2 Configuración Inicial Backend

1. Clonar repositorio
2. Configurar connection strings en appsettings.json
3. Configurar secretos SMTP, JWT, reCAPTCHA
4. Ejecutar migraciones: `dotnet ef database update`
5. Ejecutar aplicación: `dotnet run --project Web`

### 47.3 Variables de Entorno Backend

Configurar en el servidor:

- ♦ `ASPNETCORE_ENVIRONMENT`
- ♦ Connection strings
- ♦ Claves de API
- ♦ Configuración SMTP

## 47.4 Docker Compose Backend

Possible configuración de servicios:

- API (.NET)
  - SQL Server
  - PostgreSQL
  - MySQL
  - Redis (caché)
  - Nginx (reverse proxy)
- 

## 48. Mantenimiento Backend

### 48.1 Logs Backend

Implementar logging con:

- Consola (Development)
- Archivos (Production)
- Application Insights / Seq (Monitoreo)

### 48.2 Backups Backend

**Base de datos:**

- Backups diarios automatizados
- Retención de 30 días
- Pruebas de restauración mensuales

**Archivos:**

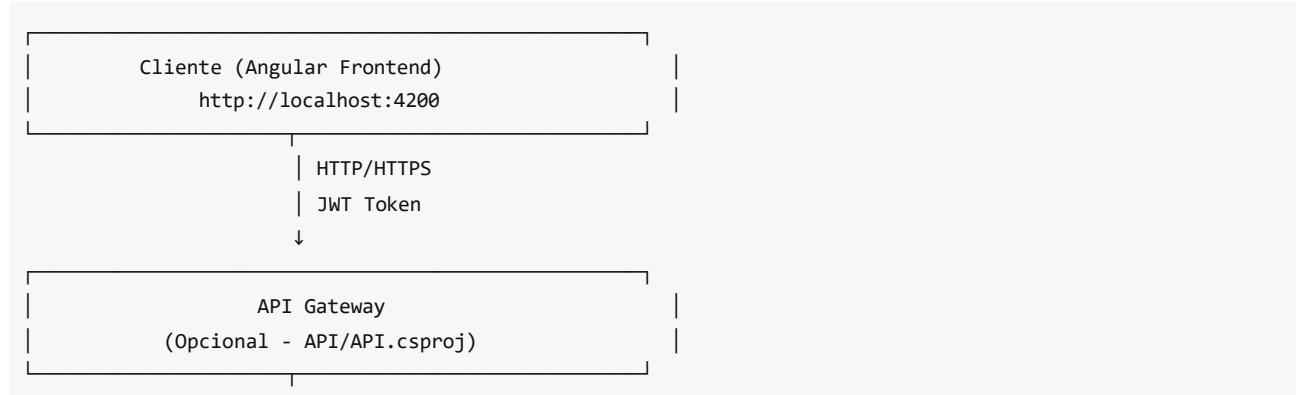
- Documentos de infracciones
- Reportes generados
- Logs del sistema

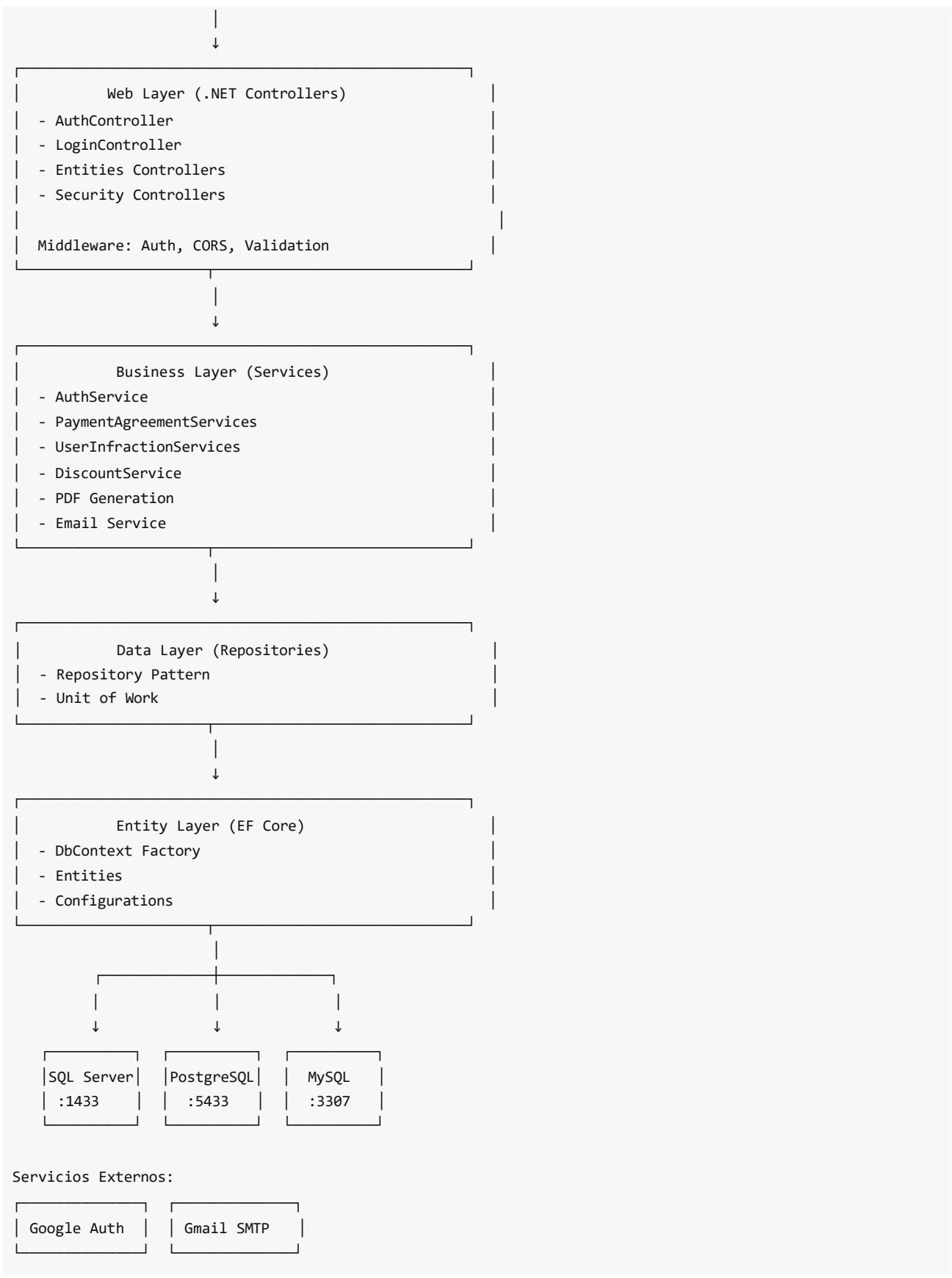
### 48.3 Monitoreo Backend

**Métricas clave:**

- Tiempo de respuesta de API
  - Tasa de errores
  - Uso de CPU/Memoria
  - Conexiones a BD
  - Tamaño de cola de trabajos
- 

## 49. Diagrama de Arquitectura Completa





## 50. Endpoints Principales Backend

### 50.1 Autenticación Endpoints

```
POST /api/auth/login
POST /api/auth/register
POST /api/auth/refresh-token
POST /api/auth/logout
POST /api/auth/forgot-password
POST /api/auth/reset-password
POST /api/auth/google-login
```

## 50.2 Usuarios Endpoints

```
GET /api/users
GET /api/users/{id}
POST /api/users
PUT /api/users/{id}
DELETE /api/users/{id}
```

## 50.3 Infracciones Endpoints

```
GET /api/infractions
GET /api/infractions/{id}
POST /api/infractions
PUT /api/infractions/{id}
DELETE /api/infractions/{id}
GET /api/infractions/types
GET /api/infractions/user/{userId}
```

## 50.4 Pagos Endpoints

```
GET /api/payments
POST /api/payments
GET /api/payment-agreements
POST /api/payment-agreements
GET /api/payment-agreements/{id}/calculate
```

## 50.5 Reportes Endpoints

```
GET /api/reports/inspector
GET /api/reports/fines
GET /api/reports/payments
POST /api/reports/generate
```

---

# 51. Integración Frontend-Backend

## 51.1 Comunicación

### Frontend (Angular) ↔ Backend (.NET)

- **Protocolo:** HTTP/HTTPS
- **Formato de datos:** JSON
- **Autenticación:** JWT Bearer Token
- **Puerto Frontend:** 4200
- **Puerto Backend:** 5000/5001 (HTTP/HTTPS)

## 51.2 Flujo de Autenticación Completo

1. Usuario ingresa credenciales en Angular
2. Angular envía POST a /api/auth/login
3. Backend valida credenciales
4. Backend genera JWT token
5. Backend retorna token al frontend
6. Angular guarda token en localStorage
7. Angular incluye token en todas las peticiones subsiguientes
8. Backend valida token en cada petición

## 51.3 Manejo de Errores

**Backend → Frontend:**

- Códigos HTTP estándar
- Mensajes de error estructurados
- Validaciones de FluentValidation
- Excepciones personalizadas

---

# 52. Comandos Útiles Backend

## 52.1 Comandos .NET

**Ejecutar aplicación**

```
dotnet run --project Proyecto-BACK/Merge_Back/taller/Web
```

**Crear migración**

```
dotnet ef migrations add NombreMigracion --project Entity --startup-project Web
```

**Aplicar migración**

```
dotnet ef database update --project Entity --startup-project Web
```

**Ejecutar pruebas**

```
dotnet test
```

**Build del proyecto**

```
dotnet build
```

**Publicar**

```
dotnet publish -c Release -o ./publish
```

**Docker build**

```
docker build -t control-multas-ciudadanas-api .
```

#### Docker run

```
docker run -p 8080:80 control-multas-ciudadanas-api
```

### 52.2 Puertos por Defecto

- **Frontend Angular:** <http://localhost:4200>
- **Backend API:** <https://localhost:5001> o <http://localhost:5000>
- **SQL Server:** 1433
- **PostgreSQL:** 5433
- **MySQL:** 3307
- **Swagger:** <http://localhost:5000/swagger>

### 52.3 Credenciales por Defecto Backend

#### Base de datos:

- Usuario: sa (SQL Server) / postgres (PostgreSQL) / root (MySQL)
- Password: Admin123.

⚠ **IMPORTANTE:** Cambiar en producción

---

## 53. Conclusiones del Sistema Completo

### 53.1 Fortalezas del Sistema Integrado

#### Frontend (Angular 19):

1. Framework moderno y reactivo
2. Componentes reutilizables
3. Interfaz responsive con PrimeNG
4. Guards de seguridad
5. Lazy loading de módulos
6. Interceptores HTTP

#### Backend (.NET 8):

1. Arquitectura limpia y bien estructurada
2. Soporte multi-base de datos
3. Seguridad robusta (JWT, reCAPTCHA, sesiones)
4. Validaciones con FluentValidation
5. Documentación con Swagger
6. Patrones de diseño apropiados
7. Escalabilidad horizontal
8. Preparado para Docker

#### Integración:

1. Comunicación REST API
2. Autenticación JWT unificada
3. Manejo de errores consistente
4. Documentación completa (Frontend + Backend)

### 53.2 Stack Tecnológico Completo

#### Frontend:

- Angular 19
- TypeScript 5.6
- RxJS 7.8
- PrimeNG 17
- Chart.js

**Backend:**

- .NET 8
- C#
- Entity Framework Core 9
- AutoMapper
- FluentValidation

**Bases de Datos:**

- SQL Server
- PostgreSQL
- MySQL

**Infraestructura:**

- Docker
- Nginx (opcional)
- Redis (opcional)

### 53.3 Casos de Uso del Sistema

Este sistema completo es ideal para:

- Gestión municipal de multas ciudadanas
- Control de infracciones de convivencia
- Gestión de acuerdos de pago
- Reportes de inspectoría
- Notificaciones automatizadas
- Sanciones administrativas
- Portal ciudadano web

---

## Glosario Técnico

- **AOT:** Ahead-of-Time Compilation
- **DI:** Dependency Injection
- **DTO:** Data Transfer Object
- **Guard:** Protector de rutas
- **Interceptor:** Middleware HTTP
- **JWT:** JSON Web Token
- **Lazy Loading:** Carga diferida de módulos
- **Observable:** Flujo de datos asíncrono
- **Pipe:** Transformador de datos en templates
- **Resolver:** Pre-cargador de datos para rutas
- **RxJS:** Reactive Extensions for JavaScript
- **Service:** Servicio singleton inyectable
- **SPA:** Single Page Application
- **Tree Shaking:** Eliminación de código no usado

---

## Control de Versiones del Documento

Versión	Fecha	Autor	Cambios

1.0	2025-10-01	Equipo Dev	Versión inicial
-----	------------	------------	-----------------

---

© 2025 Sistema de Gestión de Multas - Manual Técnico del Frontend

**Repositorio Frontend:** <https://github.com/ingrid0208/Proyecto-FRONT>

---

**Nota Importante:** Este manual cubre únicamente la implementación del Frontend (Angular 19). Para información sobre el backend, consulte la documentación correspondiente del servidor.