

# Sistema de Control Escolar

**Versión:** 1.0.0

**Fecha:** 2024

**Autor:** Omar Bermejo Osuna

## Tabla de Contenidos

1. [Visión General del Sistema](#)
2. [Arquitectura del Sistema](#)
3. [Stack Tecnológico Detallado](#)
4. [Estructura de Directorios](#)
5. [Modelo de Datos](#)
6. [Backend - Arquitectura y Componentes](#)
7. [Frontend - Arquitectura y Componentes](#)
8. [API REST - Especificación Completa](#)
9. [Sistema de Autenticación y Autorización](#)
10. [Flujos de Datos](#)
11. [Seguridad](#)
12. [Performance y Optimización](#)
13. [Docker y Deployment](#)
14. [Testing](#)
15. [Manejo de Errores](#)
16. [Convenciones de Código](#)

## 1. Visión General del Sistema

### 1.1 Propósito

Sistema web full-stack para la gestión integral de calificaciones escolares, diseñado para manejar tres roles principales:

- **Control Escolar (Admin):** Gestión completa del sistema
- **Maestros:** Registro y gestión de calificaciones
- **Alumnos:** Consulta de calificaciones propias

### 1.2 Características Principales

- Gestión completa de materias, maestros y alumnos
- Sistema de calificaciones por unidades (1-5)

- Asignación de materias a maestros y alumnos
- Reportes estadísticos y promedios
- Soft delete para preservación de datos
- Interfaz moderna con animaciones fluidas
- Sistema de autenticación JWT
- Arquitectura escalable y mantenible

### 1.3 Requisitos del Sistema

- **Node.js:** >= 18.0.0
- **PostgreSQL:** >= 13
- **Docker:** >= 20.10 (opcional pero recomendado)
- **Navegadores:** Chrome 90+, Firefox 88+, Safari 14+, Edge 90+

## 2. Arquitectura del Sistema

### 2.1 Arquitectura General



SQL Queries

Base de Datos (PostgreSQL 15)

- Tablas relacionales
- Índices optimizados
- Soft delete (deleted\_at)

## 2.2 Patrones de Diseño Utilizados

### Backend

- **MVC (Model-View-Controller)**: Separación clara de responsabilidades
- **Middleware Pattern**: Procesamiento de requests en cadena
- **Repository Pattern**: Abstracción de acceso a datos (Sequelize)
- **Singleton Pattern**: Conexión única a base de datos

### Frontend

- **Component-Based Architecture**: Componentes React reutilizables
- **Service Layer Pattern**: Separación de lógica de negocio y UI
- **Container/Presentational Pattern**: Separación de lógica y presentación
- **Custom Hooks Pattern**: Lógica reutilizable encapsulada

## 2.3 Flujo de Request

1. Cliente → Request HTTP
2. Express Router → Enruta a controlador específico
3. Middleware Stack:
  - a. CORS
  - b. Helmet (Seguridad)
  - c. Morgan (Logging)
  - d. Body Parser
  - e. Auth Middleware (si requiere autenticación)
  - f. Validation Middleware (si requiere validación)
4. Controller → Procesa lógica de negocio
5. Model (Sequelize) → Interactúa con BD
6. Response → JSON al cliente
7. Error Handler → Captura errores y responde apropiadamente

## 3. Stack Tecnológico Detallado

### 3.1 Backend

#### Runtime y Framework

- **Node.js v22+**: Runtime JavaScript del lado del servidor

- Motor V8 optimizado
- Event Loop asíncrono
- Módulos ES6 nativos
- **Express.js v5.2:** Framework web minimalista
  - Routing flexible
  - Middleware extensible
  - Manejo de errores integrado

## Base de Datos

- **PostgreSQL v15:** Base de datos relacional
  - ACID compliance
  - Transacciones robustas
  - Índices B-tree optimizados
  - Full-text search capabilities
- **Sequelize v6.37:** ORM (Object-Relational Mapping)
  - Migraciones automáticas
  - Validaciones a nivel de modelo
  - Relaciones definidas
  - Query builder avanzado
  - Hooks y callbacks

## Autenticación y Seguridad

- **JWT (jsonwebtoken v9.0.3):** Tokens de autenticación
  - Algoritmo: HS256
  - Payload: { id, email, rol }
  - Expiración configurable
- **bcryptjs v3.0.3:** Hash de contraseñas
  - Salt rounds: 10
  - Algoritmo bcrypt
  - Resistente a rainbow tables
- **helmet v8.1.0:** Seguridad HTTP
  - Headers de seguridad
  - Protección XSS
  - Prevención clickjacking

## Validación

- **express-validator v7.3.1:** Validación de requests
  - o Validación de body, params, query
  - o Sanitización de datos
  - o Mensajes de error personalizados

## Utilidades

- **morgan v1.10.1:** HTTP request logger
  - o Formato: 'combined'
  - o Logging en desarrollo
- **dotenv v17.2.3:** Variables de entorno
  - o Configuración centralizada
  - o Secrets management
- **cors v2.8.5:** Cross-Origin Resource Sharing
  - o Configuración de orígenes permitidos
  - o Headers CORS apropiados

## 3.2 Frontend

### Framework y Biblioteca

- **React v19.2:** Biblioteca UI
  - o Componentes funcionales
  - o Hooks (useState, useEffect, useCallback)
  - o Virtual DOM optimizado
  - o Server Components (futuro)
- **TypeScript v5.9:** Superset de JavaScript
  - o Tipado estático
  - o Interfaces y tipos
  - o Autocompletado mejorado
  - o Detección temprana de errores

### Build Tools

- **Vite v7.2:** Build tool y dev server
  - o HMR (Hot Module Replacement) rápido
  - o Build optimizado con Rollup
  - o Tree-shaking automático

- Code splitting

## Estilos

- **Tailwind CSS v3.4:** Framework utility-first
  - Clases utilitarias
  - PurgeCSS integrado
  - Configuración personalizada
  - Responsive design
- **PostCSS v8.5:** Procesador CSS
  - Autoprefixer
  - Transformaciones CSS

## HTTP Client

- **Axios v1.13:** Cliente HTTP
  - Interceptores de request/response
  - Manejo automático de tokens
  - Transformación de datos
  - Cancelación de requests

## Routing

- **React Router DOM v7.10:** Enrutamiento
  - Navegación declarativa
  - Protected routes
  - History API

## Desarrollo

- **ESLint v9.39:** Linter
  - Reglas TypeScript
  - Reglas React
  - Auto-fix

## 3.3 DevOps

### Containerización

- **Docker:** Containerización de aplicaciones
  - Imágenes optimizadas
  - Multi-stage builds
  - Volúmenes persistentes

- **Docker Compose:** Orquestación

- Servicios definidos
- Networking automático
- Health checks

## 4. Estructura de Directorios

## 4.1 Estructura Completa del Proyecto

```
prueba-tecnica-fullstack/
├── backend/ # Aplicación Backend
│   ├── src/
│   │   ├── config/
│   │   │   └── database.js # Configuración Sequelize
│   │   ├── controllers/ # Lógica de Negocio
│   │   │   ├── admin.controller.js # Controlador Admin (1404 líneas)
│   │   │   ├── alumno.controller.js # Controlador Alumno
│   │   │   ├── auth.controller.js # Autenticación
│   │   │   ├── general.controller.js # Endpoints generales
│   │   │   └── maestro.controller.js # Controlador Maestro
│   │   ├── middlewares/ # Middlewares
│   │   │   ├── auth.middleware.js # Verificación JWT
│   │   │   ├── errorHandler.middleware.js # Manejo de errores
│   │   │   └── validate.middleware.js # Validación de requests
│   │   ├── models/ # Modelos Sequelize
│   │   │   ├── Alumno.js # Modelo Alumno
│   │   │   ├── Asignacion.js # Modelo Asignación
│   │   │   ├── Calificacion.js # Modelo Calificación
│   │   │   ├── Materia.js # Modelo Materia
│   │   │   ├── Usuario.js # Modelo Usuario
│   │   │   └── index.js # Relaciones entre modelos
│   │   ├── routes/ # Definición de Rutas
│   │   │   ├── admin.routes.js # Rutas Admin (154 líneas)
│   │   │   ├── alumno.routes.js # Rutas Alumno (42 líneas)
│   │   │   ├── auth.routes.js # Rutas Auth
│   │   │   ├── general.routes.js # Rutas Generales
│   │   │   └── maestro.routes.js # Rutas Maestro
│   │   ├── migrations/ # Migraciones de BD
│   │   │   ├── addSemestreColumn.js # Agregar columna semestre
│   │   │   └── addUnidadColumn.js # Agregar columna unidad
│   │   ├── seeders/ # Datos de Prueba
│   │   │   ├── generateSeeds.js # Generador de seeds
│   │   │   └── runSeeds.js # Ejecutor de seeds
│   │   ├── scripts/ # Scripts utilitarios
│   │   │   └── fixUsuarioBEA0101.js # Script de corrección
│   └── index.js # Punto de entrada
```

package.json	# Dependencias
Dockerfile	# Imagen Docker
frontend/	# Aplicación Frontend
src/	
components/	# Componentes React
AdminDashboard.tsx	# Panel Admin (3350 líneas)
AlumnoDashboard.tsx	# Panel Alumno (359 líneas)
MaestroDashboard.tsx	# Panel Maestro
pages/	# Páginas/Vistas
Dashboard.tsx	# Layout principal
Login.tsx	# Página de login
services/	# Servicios API
admin.service.ts	# Servicios Admin
alumno.service.ts	# Servicios Alumno
api.ts	# Configuración Axios
general.service.ts	# Servicios generales
maestro.service.ts	# Servicios Maestro
types/	# Tipos TypeScript
index.ts	# Definiciones de tipos
App.tsx	# Componente raíz
main.tsx	# Punto de entrada
index.css	# Estilos globales
tailwind.config.cjs	# Config Tailwind
vite.config.ts	# Config Vite
tsconfig.json	# Config TypeScript
package.json	# Dependencias
Dockerfile	# Imagen Docker
docker-compose.yml	# Orquestación Docker
.env.example	# Variables de entorno ejemplo
README.md	# Documentación general
DOCUMENTACION_TECNICA.md	# Este documento

## 4.2 Descripción de Directorios Clave

### Backend/src/controllers/

Contiene la lógica de negocio de cada módulo:

- **admin.controller.js**: Funciones para gestión completa del sistema
- **maestro.controller.js**: Funciones para gestión de calificaciones
- **alumno.controller.js**: Funciones para consulta de calificaciones
- **auth.controller.js**: Login y registro
- **general.controller.js**: Endpoints compartidos

### Backend/src/models/

Define las entidades de la base de datos:

- Cada modelo extiende `sequelize.Model`

- Define campos, tipos, validaciones
- **index.js**: Configura todas las relaciones entre modelos

## Backend/src/middlewares/

Procesamiento de requests antes de llegar a controllers:

- **auth.middleware.js**: Verifica JWT y roles
- **validate.middleware.js**: Valida datos de entrada
- **errorHandler.middleware.js**: Captura y formatea errores

## Frontend/src/components/

Componentes React reutilizables:

- **AdminDashboard.tsx**: Panel completo de administración (3350 líneas)
- **AlumnoDashboard.tsx**: Vista de alumno (359 líneas)
- **MaestroDashboard.tsx**: Panel de maestro

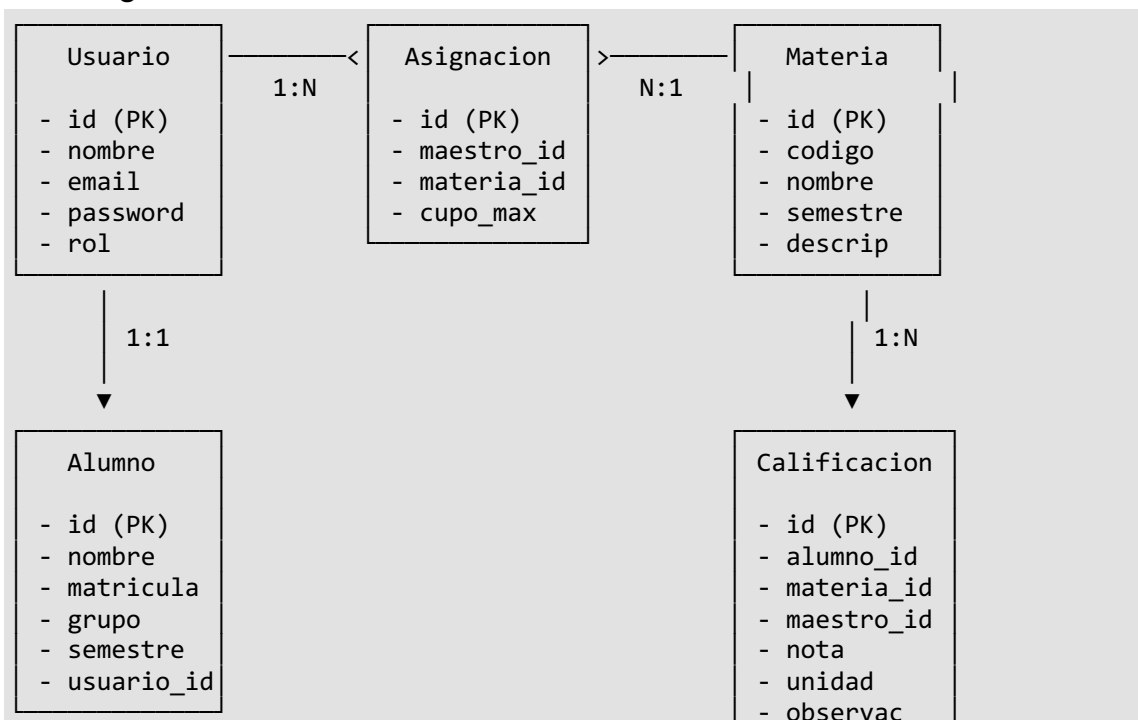
## Frontend/src/services/

Capa de comunicación con API:

- Cada servicio corresponde a un módulo del backend
- **api.ts**: Configuración centralizada de Axios con interceptores

## 5. Modelo de Datos

### 5.1 Diagrama Entidad-Relación



## 5.2 Especificación Detallada de Modelos

### 5.2.1 Usuario

```
{
  id: INTEGER (PK, AUTO_INCREMENT),
  nombre: STRING(255) NOT NULL,
  email: STRING(255) UNIQUE NOT NULL,
  password_hash: STRING(255) NOT NULL,
  rol: ENUM('MAESTRO', 'CONTROL_ESCOLAR', 'ALUMNO') NOT NULL,
  created_at: TIMESTAMP,
  updated_at: TIMESTAMP
}
```

#### Índices:

- PRIMARY KEY: **id**
- UNIQUE: **email**

#### Relaciones:

- **hasOne** Alumno (cuando rol = 'ALUMNO')
- **hasMany** Asignacion (como maestro)
- **hasMany** Calificacion (como maestro que registra)

#### Validaciones:

- Email: formato válido, lowercase, trim
- Password: mínimo 6 caracteres (antes de hash)
- Rol: debe ser uno de los valores permitidos

### 5.2.2 Alumno

```
{
  id: INTEGER (PK, AUTO_INCREMENT),
  nombre: STRING(255) NOT NULL,
  matricula: STRING(50) UNIQUE NOT NULL,
  grupo: STRING(10) NOT NULL,
  semestre: INTEGER (1-8, opcional),
  fecha_nacimiento: DATE (opcional),
  usuario_id: INTEGER (FK → Usuario.id, opcional),
  created_at: TIMESTAMP,
  updated_at: TIMESTAMP
}
```

#### Índices:

- PRIMARY KEY: **id**
- UNIQUE: **matricula**
- FOREIGN KEY: **usuario\_id** → **Usuario.id**

#### Relaciones:

- **belongsTo** Usuario (opcional, 1:1)
- **hasMany** Calificacion

#### Validaciones:

- Matricula: único, no nulo
- Grupo: no nulo
- Semestre: rango 1-8 si está presente

### 5.2.3 Materia

```
{
  id: INTEGER (PK, AUTO_INCREMENT),
  codigo: STRING(50) UNIQUE NOT NULL,
  nombre: STRING(255) NOT NULL,
  descripcion: TEXT (opcional),
  semestre: INTEGER (1-8) NOT NULL,
  estatus: INTEGER DEFAULT 1,
  created_at: TIMESTAMP,
  updated_at: TIMESTAMP
}
```

#### Índices:

- PRIMARY KEY: **id**
- UNIQUE: **codigo**
- INDEX: **semestre** (para búsquedas rápidas)

#### Relaciones:

- **hasMany** Asignacion
- **hasMany** Calificacion

#### Validaciones:

- Codigo: único, no nulo
- Nombre: no nulo
- Semestre: rango 1-8

### 5.2.4 Asignacion

```
{
  id: INTEGER (PK, AUTO_INCREMENT),
  maestro_id: INTEGER (FK → Usuario.id) NOT NULL,
  materia_id: INTEGER (FK → Materia.id) NOT NULL,
  cupo_maximo: INTEGER DEFAULT 40,
  created_at: TIMESTAMP,
  updated_at: TIMESTAMP
}
```

#### Índices:

- PRIMARY KEY: **id**

- UNIQUE: (`maestro_id`, `materia_id`) (un maestro no puede estar asignado dos veces a la misma materia)
- FOREIGN KEY: `maestro_id` → `Usuario.id`
- FOREIGN KEY: `materia_id` → `Materia.id`

#### Relaciones:

- `belongsTo` Usuario (como maestro)
- `belongsTo` Materia

#### Validaciones:

- `Cupo_maximo`: mínimo 1

### 5.2.5 Calificacion

```
{
  id: INTEGER (PK, AUTO_INCREMENT),
  alumno_id: INTEGER (FK → Alumno.id) NOT NULL,
  materia_id: INTEGER (FK → Materia.id) NOT NULL,
  maestro_id: INTEGER (FK → Usuario.id) NOT NULL,
  nota: DECIMAL(4,2) NOT NULL,
  unidad: INTEGER (1-5) NOT NULL,
  observaciones: TEXT (opcional),
  deleted_at: TIMESTAMP (opcional, Soft Delete),
  created_at: TIMESTAMP,
  updated_at: TIMESTAMP
}
```

#### Índices:

- PRIMARY KEY: `id`
- FOREIGN KEY: `alumno_id` → `Alumno.id`
- FOREIGN KEY: `materia_id` → `Materia.id`
- FOREIGN KEY: `maestro_id` → `Usuario.id`
- INDEX: (`alumno_id`, `materia_id`, `unidad`) (para búsquedas rápidas)
- INDEX: `deleted_at` (para queries con soft delete)

#### Relaciones:

- `belongsTo` Alumno
- `belongsTo` Materia
- `belongsTo` Usuario (como Maestro)

#### Validaciones:

- Nota: rango 0.00 - 10.00
- Unidad: rango 1-5
- Soft delete: `deleted_at IS NULL` para registros activos

## 5.3 Relaciones Detalladas

### Usuario ↔ Alumno (1:1 Opcional)

- Un Usuario con rol 'ALUMNO' puede tener un registro Alumno asociado
- Un Alumno puede tener un Usuario asociado (para login)
- Relación opcional: permite alumnos sin cuenta de usuario

### Usuario ↔ Asignacion (1:N)

- Un Usuario con rol 'MAESTRO' puede tener múltiples Asignaciones
- Una Asignacion pertenece a un Usuario (maestro)

### Materia ↔ Asignacion (1:N)

- Una Materia puede tener múltiples Asignaciones (diferentes maestros)
- Una Asignacion pertenece a una Materia

### Alumno ↔ Calificacion (1:N)

- Un Alumno puede tener múltiples Calificaciones
- Una Calificacion pertenece a un Alumno

### Materia ↔ Calificacion (1:N)

- Una Materia puede tener múltiples Calificaciones
- Una Calificacion pertenece a una Materia

### Usuario ↔ Calificacion (1:N)

- Un Usuario (maestro) puede registrar múltiples Calificaciones
- Una Calificacion es registrada por un Usuario (maestro)

## 5.4 Constraints y Reglas de Negocio

1. **Unicidad de Email:** Cada usuario debe tener un email único
2. **Unicidad de Matrícula:** Cada alumno debe tener una matrícula única
3. **Unicidad de Código de Materia:** Cada materia debe tener un código único
4. **Asignación Única:** Un maestro no puede estar asignado dos veces a la misma materia
5. **Calificación por Unidad:** Un alumno solo puede tener una calificación por materia por unidad (1-5)
6. **Soft Delete:** Las calificaciones eliminadas se marcan con `deleted_at`, no se eliminan físicamente
7. **Rango de Notas:** Las calificaciones deben estar entre 0.00 y 10.00
8. **Semestres:** Las materias y alumnos pueden tener semestres del 1 al 8

## 6. Backend - Arquitectura y Componentes

### 6.1 Punto de Entrada (index.js)

```
// Estructura básica del servidor Express
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const morgan = require('morgan');
const { sequelize } = require('./src/models');

const app = express();

// Middlewares globales
app.use(helmet());           // Seguridad
app.use(cors());             // CORS
app.use(morgan('combined')); // Logging
app.use(express.json());     // Body parser

// Rutas
app.use('/api/auth', authRoutes);
app.use('/api/controlescolar', adminRoutes);
app.use('/api/maestro', maestroRoutes);
app.use('/api/alumno', alumnoRoutes);
app.use('/api', generalRoutes);

// Error handler
app.use(errorHandler);

// Inicialización
sequelize.sync({ alter: true }).then(() => {
  app.listen(PORT);
});
```

### 6.2 Configuración de Base de Datos

**Archivo:** `backend/src/config/database.js`

```
const { Sequelize } = require('sequelize');

const sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    host: process.env.DB_HOST,
    dialect: 'postgres',
    logging: process.env.NODE_ENV === 'development' ? console.log : false,
    pool: {
      max: 5,
      min: 0,
      acquire: 30000,
      idle: 10000
    }
  }
);

module.exports = sequelize;
```

**Configuración:**

- **Dialect:** PostgreSQL
- **Pool:** Máximo 5 conexiones simultáneas
- **Logging:** Solo en desarrollo
- **Sync:** `alter: true` (modifica tablas existentes)

## 6.3 Middlewares

### 6.3.1 Auth Middleware

**Archivo:** `backend/src/middlewares/auth.middleware.js`

**Funciones:**

1. **verifyToken(req, res, next)**
  - o Extrae token del header `Authorization: Bearer <token>`
  - o Verifica token con `JWT_SECRET`
  - o Adjunta información del usuario a `req.user`
  - o Retorna 401 si no hay token
  - o Retorna 403 si token inválido/expirado
2. **checkRole(roles)**
  - o Middleware factory que retorna un middleware
  - o Verifica que `req.user.role` esté en el array de roles permitidos
  - o Retorna 403 si el rol no está permitido

**Uso:**

```
router.get('/endpoint', verifyToken, checkRole(['CONTROL_ESCOLAR']),
controller);
```

### 6.3.2 Validation Middleware

**Archivo:** `backend/src/middlewares/validate.middleware.js`

**Funcionalidad:**

- Usa `express-validator` para validar requests
- Valida body, params, query según reglas definidas
- Sanitiza datos de entrada
- Retorna errores formateados si la validación falla

**Ejemplo:**

```
const { body, validationResult } = require('express-validator');

const validateLogin = [
  body('email').isEmail().normalizeEmail(),
  body('password').isLength({ min: 6 }),
  (req, res, next) => {
```

```

const errors = validationResult(req);
if (!errors.isEmpty()) {
  return res.status(400).json({ errors: errors.array() });
}
next();
}
];

```

### 6.3.3 Error Handler Middleware

**Archivo:** `backend/src/middlewares/errorHandler.middleware.js`

**Funcionalidad:**

- Captura todos los errores no manejados
- Formatea respuestas de error consistentes
- Logs de errores para debugging
- Diferencia entre errores de desarrollo y producción

**Estructura de Error:**

```

{
  message: "Mensaje de error legible",
  error: "Detalles técnicos (solo en desarrollo)",
  status: 500
}

```

## 6.4 Controladores

### 6.4.1 Admin Controller

**Archivo:** `backend/src/controllers/admin.controller.js` (1404 líneas)

**Funciones Principales:**

1. **getReporteGlobal**
  - Obtiene promedio general de todos los alumnos
  - Calcula promedio por alumno
  - Retorna lista ordenada por promedio
2. **getReporteMateria(materialId)**
  - Obtiene todas las calificaciones de una materia
  - Incluye información de alumnos y maestro asignado
  - Agrupa por alumno con sus unidades
3. **crearMateria**
  - Crea nueva materia
  - Opcionalmente asigna maestro al crear
  - Valida código único y semestre válido
4. **asignarMateriaMaestro**

- Asigna un maestro a una materia
- Crea registro en tabla Asignacion
- Valida que no exista asignación duplicada

#### 5. **asignarAlumnosAMateria**

- Asigna múltiples alumnos a una materia
- Valida cupo máximo
- Crea registros de asignación

#### 6. **asignarMateriasAAlumno**

- Asigna múltiples materias a un alumno
- Útil para inscripción masiva

#### 7. **crearUsuario**

- Crea usuario (Maestro, Alumno, Admin)
- Si es Alumno, opcionalmente crea registro Alumno
- Hashea contraseña con bcrypt

#### 8. **getDetalleAlumno(alumnoid)**

- Obtiene información completa de un alumno
- Incluye calificaciones, materias cursando, cursadas, faltantes
- Agrupa por semestre

#### 9. **updateCalificacionAdmin**

- Actualiza calificación por unidad específica
- Valida rango de nota (0-10)
- Permite actualizar observaciones

#### 10. **deleteCalificacionAdmin**

- Soft delete de calificación
- Marca **deleted\_at** en lugar de eliminar físicamente

#### 11. **updateMateriaAdmin**

- Actualiza información de materia
- Valida código único si se cambia

#### 12. **updateMaestroAdmin**

- Actualiza información de maestro
- Opcionalmente actualiza contraseña

#### 13. **updateAlumnoAdmin**

- Actualiza información de alumno

- Permite vincular/desvincular usuario

#### 14. **deleteMateriaAdmin**

- Soft delete de materia
- Marca estatus como inactivo

#### 15. **deleteAlumnoAdmin**

- Soft delete de alumno
- Marca registro como eliminado

### 6.4.2 Maestro Controller

**Archivo:** `backend/src/controllers/maestro.controller.js`

**Funciones:**

#### 16. **getMateriasAsignadas**

- Obtiene materias asignadas al maestro autenticado
- Incluye información de cupo y alumnos inscritos

#### 17. **getAlumnosPorMateria(materialId)**

- Obtiene alumnos inscritos en una materia
- Valida que la materia esté asignada al maestro

#### 18. **registrarCalificacion**

- Registra nueva calificación
- Valida que el maestro esté asignado a la materia
- Valida rango de nota y unidad (1-5)
- Permite agregar observaciones

### 6.4.3 Alumno Controller

**Archivo:** `backend/src/controllers/alumno.controller.js`

**Funciones:**

#### 19. **getMisCalificaciones**

- Obtiene todas las calificaciones del alumno autenticado
- Agrupa por materia
- Incluye información de unidades

#### 20. **getCalificacionPorMateria(materialId)**

- Obtiene calificaciones de una materia específica
- Incluye todas las unidades (1-5)

#### 21. **getMiPromedio**

- Calcula promedio general del alumno
- Considera solo calificaciones activas (deleted\_at IS NULL)

#### 6.4.4 Auth Controller

**Archivo:** `backend/src/controllers/auth.controller.js`

**Funciones:**

##### 22. login

- Autentica usuario con email y password
- Verifica password con bcrypt
- Genera token JWT
- Retorna token y información del usuario

**Flujo:**

```
1. Recibe email y password
2. Busca usuario por email
3. Compara password con hash almacenado
4. Si coincide, genera JWT con { id, email, rol }
5. Retorna token y datos del usuario
```

### 6.5 Rutas

#### 6.5.1 Admin Routes

**Archivo:** `backend/src/routes/admin.routes.js` (154 líneas)

**Endpoints:**

```
// Reportes
GET    /api/controlescolar/reporte
GET    /api/controlescolar/reporte/:materiaID

// Materias
POST   /api/controlescolar/materias
PATCH /api/controlescolar/materias/:materiaID
DELETE /api/controlescolar/materias/:materiaID

// Asignaciones
POST   /api/controlescolar/asignacion
POST   /api/controlescolar/materias/:materiaID/alumnos
POST   /api/controlescolar/alumnos/:alumnoID/materias

// Usuarios
POST   /api/controlescolar/usuarios

// Alumnos
GET    /api/controlescolar/alumnos/:alumnoID/detalle
PATCH /api/controlescolar/alumnos/:alumnoID
DELETE /api/controlescolar/alumnos/:alumnoID

// Maestros
PATCH /api/controlescolar/maestros/:maestroID
```

```
// Calificaciones
PATCH /api/controlescolar/calificaciones/:materiaID/:alumnoID/:unidadID
DELETE /api/controlescolar/calificaciones/:materiaID/:alumnoID/:unidadID
```

**Protección:**

- Todas las rutas requieren `verifyToken`
- Todas las rutas requieren rol `CONTROL_ESCOLAR`

## 6.5.2 Maestro Routes

**Archivo:** `backend/src/routes/maestro.routes.js`

**Endpoints:**

```
GET /api/maestro/materias
GET /api/maestro/alumnos/:materiaID
POST /api/maestro/calificaciones
```

**Protección:**

- Todas requieren `verifyToken`
- Todas requieren rol `MAESTRO`

## 6.5.3 Alumno Routes

**Archivo:** `backend/src/routes/alumno.routes.js` (42 líneas)

**Endpoints:**

```
GET /api/alumno/calificaciones
GET /api/alumno/calificaciones/:materiaID
GET /api/alumno/promedio
```

**Protección:**

- Todas requieren `verifyToken`
- Todas requieren rol `ALUMNO`

## 6.5.4 Auth Routes

**Archivo:** `backend/src/routes/auth.routes.js`

**Endpoints:**

```
POST /api/auth/login
```

**Protección:**

- Público (no requiere autenticación)

## 6.5.5 General Routes

**Archivo:** `backend/src/routes/general.routes.js`

**Endpoints:**

```
GET /api/materias
GET /api/usuarios/list
GET /api/alumnos/list
GET /api/maestros/list
```

**Protección:**

- Todas requieren `verifyToken`
- Accesibles para todos los roles autenticados

## 6.6 Seeders

**Archivo:** `backend/src/seeders/runSeeds.js`

**Funcionalidad:**

- Crea datos de prueba iniciales
- 50 maestros, 50 materias, 50 alumnos
- Usuario admin por defecto
- Asignaciones y calificaciones de ejemplo

**Ejecución:**

- Automática al iniciar el servidor si no existe admin
- Forzada con `SEED_FORCE=true` en `.env`

## 7. Frontend - Arquitectura y Componentes

### 7.1 Configuración de Axios

**Archivo:** `frontend/src/services/api.ts`

**Configuración:**

```
const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL || 'http://localhost:3000/api',
  headers: {
    'Content-Type': 'application/json',
  },
});
```

**Interceptores:****1. Request Interceptor:**

- o Adjunta token JWT automáticamente desde `localStorage`
- o Formato: `Authorization: Bearer <token>`

**2. Response Interceptor:**

- o Captura errores 401/403 (token expirado/inválido)
- o Limpia `localStorage`
- o Redirige a `/login`

## 7.2 Servicios

### 7.2.1 Admin Service

**Archivo:** frontend/src/services/admin.service.ts

**Funciones:**

```
// Reportes
getReporteGlobal(): Promise<ReporteItem[]>
getReporteMateria(materiaId: number): Promise<CalificacionMateriaItem[]>

// Materias
crearMateria(data: CrearMateriaPayload): Promise<Response>
updateMateriaAdmin(materiaId: number, data: UpdateMateriaPayload):
Promise<void>
deleteMateriaAdmin(materiaId: number): Promise<void>

// Asignaciones
asignarMateriaMaestro(data: AsignarMateriaPayload): Promise<void>
asignarAlumnosAMateria(materiaId: number, data: AsignarAlumnosPayload):
Promise<Response>
asignarMateriasAAlumno(alumnoId: number, data: AsignarMateriasPayload):
Promise<Response>

// Usuarios
crearUsuario(data: CrearUsuarioPayload): Promise<Response>

// Alumnos
getDetalleAlumnoAdmin(alumnoId: number): Promise<DetalleAlumno>
updateAlumnoAdmin(alumnoId: number, data: UpdateAlumnoPayload): Promise<void>
deleteAlumnoAdmin(alumnoId: number): Promise<void>

// Maestros
updateMaestroAdmin(maestroId: number, data: UpdateMaestroPayload):
Promise<void>

// Calificaciones
updateCalificacionAdmin(materiaId: number, alumnoId: number, nota: number,
obs: string, unidad: number): Promise<void>
deleteCalificacionAdmin(materiaId: number, alumnoId: number, unidad?:
number): Promise<void>
```

### 7.2.2 Maestro Service

**Archivo:** frontend/src/services/maestro.service.ts

**Funciones:**

```
getMateriasAsignadas(): Promise<Materia[]>
getAlumnosPorMateria(materiaId: number): Promise<AlumnoListItem[]>
registrarCalificacion(data: RegistrarCalificacionPayload): Promise<void>
```

### 7.2.3 Alumno Service

**Archivo:** frontend/src/services/alumno.service.ts

**Funciones:**

```
getMisCalificaciones(): Promise<CalificacionAlumno[]>  
getCalificacionPorMateria(materiaId: number): Promise<CalificacionMateria>  
getMiPromedio(): Promise<{ promedio: number }>
```

## 7.2.4 General Service

**Archivo:** `frontend/src/services/general.service.ts`

**Funciones:**

```
getMaterias(): Promise<Materia[]>  
getMaestros(): Promise<MaestroListItem[]>  
getAlumnos(): Promise<AlumnoListItem[]>  
getUsuariosAlumnosDisponibles(): Promise<Usuario[]>
```

## 7.3 Componentes Principales

### 7.3.1 AdminDashboard

**Archivo:** `frontend/src/components/AdminDashboard.tsx` (3350 líneas)

**Responsabilidades:**

- Gestión completa del sistema
- CRUD de materias, maestros, alumnos
- Asignación de materias y maestros
- Gestión de calificaciones
- Reportes y estadísticas
- Creación de usuarios

**Estados Principales:**

```
// Datos  
const [reporteGlobal, setReporteGlobal] = useState<ReporteItem[]>([]);  
const [materiasList, setMateriasList] = useState<Materia[]>([]);  
const [maestrosList, setMaestrosList] = useState<MaestroListItem[]>([]);  
const [alumnosList, setAlumnosList] = useState<AlumnoListItem[]>([]);  
  
// Vista principal  
const [selectedMateriaId, setSelectedMateriaId] = useState<number | null>(null);  
const [calificacionesMateria, setCalificacionesMateria] =  
  useState<CalificacionMateriaItem[] | null>(null);  
const [maestroAsignado, setMaestroAsignado] = useState<{ id: number; nombre: string } | null>(null);  
  
// UI  
const [loading, setLoading] = useState(true);  
const [loadingMateria, setLoadingMateria] = useState(false);  
const [gestionError, setGestionError] = useState<string | null>(null);  
const [successMsg, setSuccessMsg] = useState<string | null>(null);
```

**Funciones Clave:**

1. **loadInitialData:** Carga inicial de catálogos
2. **loadMateriaDetails:** Carga detalles de materia seleccionada

3. **handleMateriaChange**: Cambia materia seleccionada
4. **handleAsignacionSubmit**: Asigna maestro a materia
5. **handleSaveEdit**: Guarda edición de calificación
6. **handleDelete**: Elimina calificación
7. **renderMateriaView**: Renderiza vista de materia

**Características Especiales:**

- Selectores de materias agrupados por semestre
- Dropdowns personalizados con búsqueda
- Filtrado en tiempo real
- Modales para edición/creación
- Confirmaciones para eliminaciones

### 7.3.2 MaestroDashboard

**Archivo:** `frontend/src/components/MaestroDashboard.tsx`

**Responsabilidades:**

- Visualización de materias asignadas
- Gestión de calificaciones por unidad
- Registro de observaciones

**Estados:**

```
const [materias, setMaterias] = useState<Materia[]>([]);
const [selectedMateriaId, setSelectedMateriaId] = useState<number | null>(null);
const [alumnos, setAlumnos] = useState<AlumnoListItem[]>([]);
const [calificaciones, setCalificaciones] = useState<Calificacion[]>([]);
```

### 7.3.3 AlumnoDashboard

**Archivo:** `frontend/src/components/AlumnoDashboard.tsx` (359 líneas)

**Responsabilidades:**

- Visualización de calificaciones propias
- Promedio general y por materia
- Detalle por materia con unidades

**Estados:**

```
const [calificaciones, setCalificaciones] =
  useState<CalificacionAlumno[]>([]);
const [promedio, setPromedio] = useState<number>(0);
const [loading, setLoading] = useState(true);
```

## 7.4 Páginas

### 7.4.1 Login

**Archivo:** `frontend/src/pages/Login.tsx`

**Funcionalidad:**

- Formulario de login
- Validación de campos
- Llamada a API de autenticación
- Almacenamiento de token en localStorage
- Redirección según rol

### 7.4.2 Dashboard

**Archivo:** `frontend/src/pages/Dashboard.tsx`

**Funcionalidad:**

- Layout principal
- Renderiza componente según rol
- Manejo de rutas protegidas
- Logout

## 7.5 Tipos TypeScript

**Archivo:** `frontend/src/types/index.ts`

**Tipos Principales:**

```
// Usuario
interface Usuario {
  id: number;
  nombre: string;
  email: string;
  rol: 'MAESTRO' | 'CONTROL_ESCOLAR' | 'ALUMNO';
}

// Materia
interface Materia {
  id: number;
  codigo: string;
  nombre: string;
  descripcion?: string;
  semestre: number;
  estatus: number;
}

// Alumno
interface AlumnoListItem {
  id: number;
  nombre: string;
  matricula: string;
  grupo: string;
}
```

```

    semestre?: number;
    usuario_id?: number;
}

// Calificación
interface Calificacion {
    id: number;
    nota: number;
    unidad: number;
    observaciones?: string;
    materia: Materia;
    maestro: Usuario;
}

// Payloads
interface CrearMateriaPayload {
    nombre: string;
    codigo: string;
    descripcion?: string;
    semestre: number;
    maestro_id?: number;
    cupo_maximo?: number;
}

// ... más tipos

```

## 8. API REST - Especificación Completa

### 8.1 Autenticación

#### POST /api/auth/login

**Descripción:** Inicia sesión y obtiene token JWT

**Request:**

```

{
  "email": "admin@escuela.com",
  "password": "password123"
}

```

**Response (200):**

```

{
  "message": "Login exitoso",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 1,
    "nombre": "Admin Control Escolar",
    "email": "admin@escuela.com",
    "rol": "CONTROL_ESCOLAR"
  }
}

```

**Errores:**

- **400:** Email o password inválidos

- 401: Credenciales incorrectas

## 8.2 Control Escolar (Admin)

### GET /api/controlescolar/reporte

**Descripción:** Obtiene reporte global de promedios de todos los alumnos

**Headers:**

Authorization: Bearer <token>

**Response (200):**

```
[
  {
    "alumno": {
      "id": 1,
      "nombre": "Juan Pérez",
      "matricula": "A1001",
      "grupo": "A"
    },
    "promedio_general": 8.5
  },
  ...
]
```

### GET /api/controlescolar/reporte/:materialID

**Descripción:** Obtiene detalle de calificaciones por materia

**Response (200):**

```
{
  "data": [
    {
      "alumno": {
        "id": 1,
        "nombre": "Juan Pérez",
        "matricula": "A1001",
        "grupo": "A"
      },
      "unidades": [
        {
          "unidad": 1,
          "nota": 8.5,
          "observaciones": "Buen trabajo"
        },
        ...
      ],
      "promedio_materia": 8.5
    },
    ...
  ],
  "maestro": {
    "id": 2,
    "nombre": "Prof. García"
  }
}
```

## POST /api/controlescolar/materias

**Descripción:** Crea nueva materia

**Request:**

```
{
  "nombre": "Matemáticas I",
  "codigo": "MAT101",
  "descripcion": "Álgebra básica",
  "semestre": 1,
  "maestro_id": 2,
  "cupo_maximo": 40
}
```

**Response (201):**

```
{
  "message": "Materia creada exitosamente",
  "data": {
    "materia": {
      "id": 1,
      "nombre": "Matemáticas I",
      "codigo": "MAT101",
      "semestre": 1
    }
  }
}
```

## POST /api/controlescolar/materias/:materialID/alumnos

**Descripción:** Asigna alumnos a una materia

**Request:**

```
{
  "alumno_ids": [1, 2, 3],
  "maestro_id": 2
}
```

**Response (200):**

```
{
  "message": "Alumnos asignados exitosamente",
  "data": {
    "alumnos_asignados": 3
  }
}
```

## POST /api/controlescolar/alumnos/:alumnoID/materias

**Descripción:** Asigna múltiples materias a un alumno

**Request:**

```
{
  "materia_ids": [1, 2, 3]
}
```

**Response (200):**

```
{
  "message": "Materias asignadas exitosamente",
}
```

```
"data": {  
  "materias_asignadas": 3  
}
```

GET /api/controlescolar/alumnos/:alumnoID/detalle

**Descripción:** Obtiene detalle completo de un alumno

**Response (200):**

```
{  
  "alumno": {  
    "id": 1,  
    "nombre": "Juan Pérez",  
    "matricula": "A1001",  
    "grupo": "A",  
    "semestre": 1  
  },  
  "calificaciones": [...],  
  "materias_cursando": [  
    {  
      "materia": {...},  
      "activa": 1,  
      "promedio_materia": 8.5,  
      "unidades": [...]  
    }  
  ],  
  "materias_por_semestre": {  
    "1": {  
      "cursando": [...],  
      "cursadas": [...],  
      "faltantes": [...]  
    }  
  }  
}
```

PATCH /api/controlescolar/calificaciones/:materialID/:alumnoID/:unidadID

**Descripción:** Actualiza calificación por unidad

**Request:**

```
{  
  "nota": 9.0,  
  "observaciones": "Excelente trabajo"  
}
```

**Response (200):**

```
{  
  "message": "Calificación actualizada exitosamente"  
}
```

DELETE

/api/controlescolar/calificaciones/:materialID/:alumnoID/:unidadID

**Descripción:** Elimina calificación (soft delete)

**Response (200):**

```
{
  "message": "Calificación eliminada exitosamente"
}
```

### 8.3 Maestro

#### GET /api/maestro/materias

**Descripción:** Obtiene materias asignadas al maestro

**Response (200):**

```
[
  {
    "id": 1,
    "nombre": "Matemáticas I",
    "codigo": "MAT101",
    "cupo_maximo": 40,
    "alumnos_inscritos": 25
  },
  ...
]
```

#### GET /api/maestro/alumnos/:materialID

**Descripción:** Obtiene alumnos inscritos en una materia

**Response (200):**

```
[
  {
    "id": 1,
    "nombre": "Juan Pérez",
    "matricula": "A1001",
    "grupo": "A"
  },
  ...
]
```

#### POST /api/maestro/calificaciones

**Descripción:** Registra nueva calificación

**Request:**

```
{
  "materia_id": 1,
  "alumno_id": 1,
  "nota": 8.5,
  "unidad": 1,
  "observaciones": "Buen trabajo"
}
```

**Response (201):**

```
{
  "message": "Calificación registrada exitosamente"
}
```

## 8.4 Alumno

### GET /api/alumno/calificaciones

**Descripción:** Obtiene todas las calificaciones del alumno

**Response (200):**

```
[
  {
    "materia": {
      "id": 1,
      "nombre": "Matemáticas I",
      "codigo": "MAT101"
    },
    "calificaciones": [
      {
        "unidad": 1,
        "nota": 8.5,
        "observaciones": "Buen trabajo"
      },
      ...
    ],
    "promedio": 8.5
  },
  ...
]
```

### GET /api/alumno/promedio

**Descripción:** Obtiene promedio general del alumno

**Response (200):**

```
{
  "promedio": 8.5
}
```

## 8.5 Códigos de Estado HTTP

- **200 OK:** Request exitoso
- **201 Created:** Recurso creado exitosamente
- **400 Bad Request:** Error en los datos enviados
- **401 Unauthorized:** No autenticado o token inválido
- **403 Forbidden:** No tiene permisos para el recurso
- **404 Not Found:** Recurso no encontrado
- **500 Internal Server Error:** Error del servidor

## 9. Sistema de Autenticación y Autorización

### 9.1 Flujo de Autenticación

1. Usuario ingresa email y password
2. Frontend envía POST /api/auth/login

3. Backend verifica credenciales:
  - a. Busca usuario por email
  - b. Compara password con hash (bcrypt)
4. Si válido:
  - a. Genera JWT con payload { id, email, rol }
  - b. Retorna token al frontend
5. Frontend almacena token en localStorage
6. Frontend adjunta token en header Authorization de requests subsecuentes

## 9.2 Estructura del Token JWT

### Header:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

### Payload:

```
{
  "id": 1,
  "email": "admin@escuela.com",
  "rol": "CONTROL_ESCOLAR",
  "iat": 1234567890,
  "exp": 1234571490
}
```

### Firma:

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  JWT_SECRET
)
```

## 9.3 Verificación de Token

### Proceso:

1. Middleware `verifyToken` extrae token del header
2. Verifica firma con `JWT_SECRET`
3. Verifica expiración
4. Adjunta información del usuario a `req.user`
5. Continúa al siguiente middleware/controller

## 9.4 Control de Acceso por Roles

### Roles Disponibles:

- `CONTROL_ESCOLAR`: Acceso completo al sistema
- `MAESTRO`: Gestión de calificaciones de materias asignadas
- `ALUMNO`: Consulta de calificaciones propias

### Middleware de Roles:

```
checkRole(['CONTROL_ESCOLAR']) // Solo admin
checkRole(['MAESTRO', 'CONTROL_ESCOLAR']) // Admin o maestro
```

## 9.5 Seguridad de Contraseñas

### Proceso:

1. Usuario proporciona password en texto plano
2. Backend hashea con bcrypt (10 salt rounds)
3. Hash se almacena en `password_hash`
4. Password original nunca se almacena

### Verificación:

```
bcrypt.compare(password, password_hash)
```

## 10. Flujos de Datos

### 10.1 Flujo de Creación de Materia

1. AdminDashboard → `handleCreateMateria()`
2. Frontend valida datos
3. `admin.service.ts` → `crearMateria(payload)`
4. Axios → `POST /api/controlescolar/materias`
5. Backend: `admin.routes.js` → `router.post('/materias', ...)`
6. Middleware: `verifyToken, checkRole(['CONTROL_ESCOLAR'])`
7. `admin.controller.js` → `crearMateria()`
8. Sequelize → `Materia.create()`
9. PostgreSQL → `INSERT INTO materias`
10. Response → Frontend
11. Frontend actualiza estado y UI

### 10.2 Flujo de Registro de Calificación

1. MaestroDashboard → `handleSaveCalificacion()`
2. `maestro.service.ts` → `registrarCalificacion(payload)`
3. Axios → `POST /api/maestro/calificaciones`
4. Backend: `maestro.routes.js` → `router.post('/calificaciones', ...)`
5. Middleware: `verifyToken, checkRole(['MAESTRO'])`
6. `maestro.controller.js` → `registrarCalificacion()`
7. Validaciones:
  - a. Maestro asignado a materia
  - b. Alumno inscrito en materia
  - c. Nota en rango 0-10
  - d. Unidad en rango 1-5
8. Sequelize → `Calificacion.create()`
9. PostgreSQL → `INSERT INTO calificaciones`
10. Response → Frontend
11. Frontend actualiza lista de calificaciones

### 10.3 Flujo de Consulta de Calificaciones (Alumno)

1. AlumnoDashboard → `useEffect(() => loadCalificaciones())`
2. `alumno.service.ts` → `getMisCalificaciones()`
3. Axios → `GET /api/alumno/calificaciones`
4. Backend: `alumno.routes.js` → `router.get('/calificaciones', ...)`
5. Middleware: `verifyToken, checkRole(['ALUMNO'])`
6. `alumno.controller.js` → `getMisCalificaciones()`
7. Sequelize query:
  - Busca calificaciones donde `alumno_id = req.user.id`

- Incluye información de materia
  - Agrupa por materia
  - Calcula promedios
8. PostgreSQL → SELECT con JOINS
  9. Response → Frontend
  10. Frontend renderiza calificaciones agrupadas por materia

## 10.4 Flujo de Asignación Maestro-Materia

1. AdminDashboard → handleAsignacionSubmit()
2. admin.service.ts → asignarMateriaMaestro(payload)
3. Axios → POST /api/controlescolar/asignacion
4. Backend: admin.routes.js → router.post('/asignacion', ...)
5. admin.controller.js → asignarMateriaMaestro()
6. Validaciones:
  - a. Maestro existe
  - b. Materia existe
  - c. No existe asignación duplicada
7. Sequelize → Asignacion.create()
8. PostgreSQL → INSERT INTO asignaciones
9. Response → Frontend
10. Frontend recarga detalles de materia si está seleccionada

## 11. Seguridad

### 11.1 Medidas de Seguridad Implementadas

#### Backend

1. **Helmet.js**
  - Headers de seguridad HTTP
  - Prevención XSS
  - Prevención clickjacking
  - HSTS (HTTP Strict Transport Security)
2. **CORS**
  - Orígenes permitidos configurados
  - Headers CORS apropiados
3. **Bcrypt**
  - Hash de contraseñas con 10 salt rounds
  - Resistente a rainbow tables
4. **JWT**
  - Tokens firmados con secreto
  - Expiración configurable
  - Verificación en cada request protegido
5. **Validación de Inputs**

- express-validator
- Sanitización de datos
- Validación de tipos y rangos

#### 6. **Soft Delete**

- Preservación de datos eliminados
- Posibilidad de recuperación

### Frontend

#### 7. **Almacenamiento Seguro**

- Token en localStorage (considerar httpOnly cookies en producción)
- Limpieza automática en logout

#### 8. **Interceptores Axios**

- Manejo automático de tokens expirados
- Redirección a login en 401/403

#### 9. **Validación de Formularios**

- Validación client-side
- Validación server-side (doble capa)

## 11.2 Vulnerabilidades Conocidas y Recomendaciones

### Mejoras Recomendadas para Producción

#### 10. **Tokens JWT**

- Implementar refresh tokens
- Reducir tiempo de expiración
- Rotación de JWT\_SECRET

#### 11. **Contraseñas**

- Política de contraseñas más estricta
- Implementar cambio de contraseña
- Implementar recuperación de contraseña

#### 12. **Rate Limiting**

- Implementar límites de requests
- Prevenir brute force attacks

#### 13. **HTTPS**

- Forzar HTTPS en producción
- Certificados SSL válidos

#### 14. Logging y Monitoreo

- Logs de seguridad
- Alertas de intentos de acceso no autorizados

#### 15. SQL Injection

- Sequelize previene la mayoría de casos
- Validar queries raw si se usan

#### 16. XSS

- Sanitizar inputs del usuario
- CSP (Content Security Policy) headers

## 12. Performance y Optimización

### 12.1 Optimizaciones Backend

#### 17. Índices de Base de Datos

- Índices en campos de búsqueda frecuente
- Índices en foreign keys
- Índices compuestos para queries complejas

#### 18. Queries Optimizadas

- Uso de **include** en Sequelize para evitar N+1 queries
- Selección de campos específicos (no SELECT \*)
- Paginación para listas grandes

#### 19. Connection Pooling

- Pool de conexiones configurado (max: 5)
- Reutilización de conexiones

#### 20. Caching (Futuro)

- Cache de reportes frecuentes
- Redis para sesiones

### 12.2 Optimizaciones Frontend

#### 21. Code Splitting

- Vite realiza code splitting automático
- Lazy loading de componentes pesados

#### 22. Memoización

- **useCallback** para funciones estables

- `useMemo` para cálculos costosos
- `React.memo` para componentes puros

### 23. Optimización de Re-renders

- Estado local cuando es posible
- Evitar prop drilling excesivo

### 24. Bundle Size

- Tree shaking automático
- PurgeCSS para Tailwind

## 12.3 Métricas de Performance

### Backend:

- Tiempo de respuesta promedio: < 200ms
- Queries a BD: < 50ms
- Uso de memoria: < 500MB

### Frontend:

- First Contentful Paint: < 1.5s
- Time to Interactive: < 3s
- Bundle size: < 500KB (gzipped)

## 13. Docker y Deployment

### 13.1 Docker Compose

Archivo: `docker-compose.yml`

### Servicios:

#### 1. db (PostgreSQL)

- Imagen: `postgres:15-alpine`
- Puerto: 5432
- Volumen persistente: `pgdata`
- Health check configurado

#### 2. backend (Node.js)

- Build: `./backend`
- Puerto: 3000 (configurable)
- Depende de: `db` (health check)
- Volúmenes: código fuente y `node_modules`

### 3. frontend (React)

- Build: `./frontend`
- Puerto: 5173
- Depende de: `backend`
- Volúmenes: código fuente y `node_modules`

## 13.2 Dockerfiles

### Backend Dockerfile

```
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["npm", "start"]
```

### Frontend Dockerfile

```
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 5173
CMD ["npm", "run", "dev"]
```

## 13.3 Variables de Entorno

### Backend (.env):

```
NODE_ENV=development
PORT=3000
DB_HOST=db
DB_USER=postgres
DB_PASSWORD=postgres
DB_NAME=sistema_escolar
JWT_SECRET=tu_secret_key_super_segura
SEED_FORCE=false
```

### Frontend (.env):

```
VITE_API_URL=http://localhost:3000/api
```

## 13.4 Deployment en Producción

### Recomendaciones:

#### 4. Base de Datos

- Usar servicio gestionado (AWS RDS, Google Cloud SQL)
- Backups automáticos
- Replicación para alta disponibilidad

#### 5. Backend

- Usar PM2 o similar para gestión de procesos
- Nginx como reverse proxy
- SSL/TLS con Let's Encrypt

## 6. Frontend

- Build de producción: `npm run build`
- Servir con Nginx o CDN
- Compresión gzip

## 7. Monitoreo

- Logs centralizados
- Alertas de errores
- Métricas de performance

# 14. Testing

## 14.1 Estrategia de Testing

### Backend:

- Unit tests para controladores
- Integration tests para rutas
- Tests de modelos con Sequelize

### Frontend:

- Unit tests para componentes
- Integration tests para flujos
- E2E tests con Cypress (futuro)

## 14.2 Herramientas

- **Jest:** Framework de testing
- **Supertest:** Testing de APIs HTTP
- **React Testing Library:** Testing de componentes React

## 14.3 Ejemplos de Tests

### Test de Controlador

```
describe('Admin Controller', () => {
  test('should create materia', async () => {
    const response = await request(app)
      .post('/api/controlescolar/materias')
      .set('Authorization', `Bearer ${adminToken}`)
      .send({
        nombre: 'Test Materia',
```

```
        codigo: 'TEST101',
        semestre: 1
    });

    expect(response.status).toBe(201);
    expect(response.body.data.materia.nombre).toBe('Test Materia');
  });
});
```

## 15. Manejo de Errores

### 15.1 Backend

#### Error Handler Middleware:

- Captura todos los errores no manejados
- Formatea respuestas consistentes
- Logs de errores para debugging
- Diferencia entre errores de desarrollo y producción

#### Tipos de Errores:

1. **Errores de Validación (400)**
  - Datos inválidos
  - Campos requeridos faltantes
2. **Errores de Autenticación (401)**
  - Token no proporcionado
  - Token inválido
3. **Errores de Autorización (403)**
  - Rol insuficiente
  - Recurso no accesible
4. **Errores de Recurso (404)**
  - Recurso no encontrado
5. **Errores del Servidor (500)**
  - Errores inesperados
  - Errores de base de datos

### 15.2 Frontend

#### Manejo de Errores:

- Try-catch en funciones async
- Interceptores de Axios para errores globales

- Mensajes de error amigables al usuario
- Logs de errores en consola (desarrollo)

#### Estados de Error:

```
const [error, setError] = useState<string | null>(null);
```

## 16. Convenciones de Código

### 16.1 Backend (JavaScript)

#### Nomenclatura:

- Variables y funciones: **camelCase**
- Constantes: **UPPER\_SNAKE\_CASE**
- Clases: **PascalCase**
- Archivos: **kebab-case.js**

#### Estructura:

- Imports al inicio
- Funciones exportadas al final
- Comentarios JSDoc para funciones públicas

#### Ejemplo:

```
const { Usuario, Materia } = require('../models');

const crearMateria = async (req, res) => {
  // Lógica aquí
};

module.exports = { crearMateria };
```

### 16.2 Frontend (TypeScript/React)

#### Nomenclatura:

- Componentes: **PascalCase**
- Funciones y variables: **camelCase**
- Tipos e interfaces: **PascalCase**
- Archivos de componentes: **PascalCase.tsx**

#### Estructura de Componente:

```
// Imports
import React, { useState, useEffect } from 'react';

// Tipos
interface Props {
  // ...
}
```

```
// Componente
const ComponentName: React.FC<Props> = ({ prop1 }) => {
  // Estados
  const [state, setState] = useState<Type>(initialValue);

  // Efectos
  useEffect(() => {
    // ...
  }, [dependencies]);

  // Funciones
  const handleAction = () => {
    // ...
  };

  // Render
  return (
    // JSX
  );
};

export default ComponentName;
```

## 16.3 Git

### Commits:

- Formato: **tipo: descripción breve**
- Tipos: **feat, fix, docs, style, refactor, test, chore**

### Ejemplos:

```
feat: agregar funcionalidad de asignación masiva
fix: corregir cálculo de promedios
docs: actualizar documentación de API
```

## 17. Consideraciones Futuras

### 17.1 Mejoras Planificadas

#### 1. Funcionalidades

- Sistema de notificaciones
- Exportación de reportes (PDF/Excel)
- Dashboard con gráficas
- Sistema de mensajería

#### 2. Técnicas

- Implementar tests automatizados
- CI/CD pipeline
- Monitoreo y alertas

- Caching con Redis
- WebSockets para actualizaciones en tiempo real

### 3. Seguridad

- Refresh tokens
- Rate limiting
- 2FA (Two-Factor Authentication)
- Auditoría de acciones

### 4. Performance

- Paginación en todas las listas
- Lazy loading de componentes
- Optimización de imágenes
- CDN para assets estáticos

## 18. Glosario de Términos

- **API:** Application Programming Interface
- **JWT:** JSON Web Token
- **ORM:** Object-Relational Mapping
- **CRUD:** Create, Read, Update, Delete
- **Soft Delete:** Eliminación lógica (marca como eliminado sin borrar físicamente)
- **HMR:** Hot Module Replacement
- **CORS:** Cross-Origin Resource Sharing
- **XSS:** Cross-Site Scripting
- **CSRF:** Cross-Site Request Forgery
- **HSTS:** HTTP Strict Transport Security

## 19. Referencias y Recursos

### 19.1 Documentación Oficial

- [React Documentation](#)
- [Express.js Guide](#)
- [Sequelize Documentation](#)
- [PostgreSQL Documentation](#)
- [TypeScript Handbook](#)

- [Tailwind CSS Documentation](#)

## 19.2 Herramientas

- [Docker Documentation](#)
- [Axios Documentation](#)
- [JWT.io](#) - Debugger de tokens JWT

## 20. Contacto y Soporte

Para preguntas técnicas o problemas:

1. Revisar esta documentación
2. Revisar logs del servidor
3. Consultar código fuente comentado
4. Revisar issues en el repositorio

**Última actualización:** 2024

**Versión del documento:** 1.0.0

*Este documento técnico proporciona una visión completa y detallada del Sistema de Control Escolar. Para información más específica, consultar el código fuente y los comentarios inline.*