

Documentación Completa - Sistema de Créditos Backend

Información del Proyecto

Nombre: Créditos Backend

Versión: 1.0.0

Autor: Proyecto Práctica Empresarial 2025

Repositorio: [jesebe4991/creditos-backend](#)

Licencia: ISC

Tabla de Contenidos

1. Descripción General
 2. Características Principales
 3. Arquitectura del Sistema
 4. Tecnologías Utilizadas
 5. Instalación y Configuración
 6. Modelo de Base de Datos
 7. Sistema de Autenticación
 8. Sistema de Roles y Permisos
 9. Endpoints de la API
 10. Validaciones de Negocio
 11. Integraciones Externas
 12. Cron Jobs y Notificaciones
 13. Manejo de Errores
 14. Docker y Despliegue
 15. Guía de Desarrollo
 16. Casos de Uso Implementados
 17. Próximos Pasos
-

1. Descripción General

Sistema backend completo para la gestión de créditos rápidos desarrollado con Node.js, Express y PostgreSQL. El sistema implementa Clean Architecture, proporcionando una separación clara entre las capas de dominio, aplicación, infraestructura y presentación.

Propósito

Gestionar el ciclo completo de créditos desde la solicitud hasta el pago, incluyendo: - Registro y autenticación de usuarios - Gestión de clientes - Creación y aprobación de créditos - Registro de pagos con generación de comprobantes - Notificaciones automáticas por WhatsApp - Sistema de roles y permisos granular

2. Características Principales

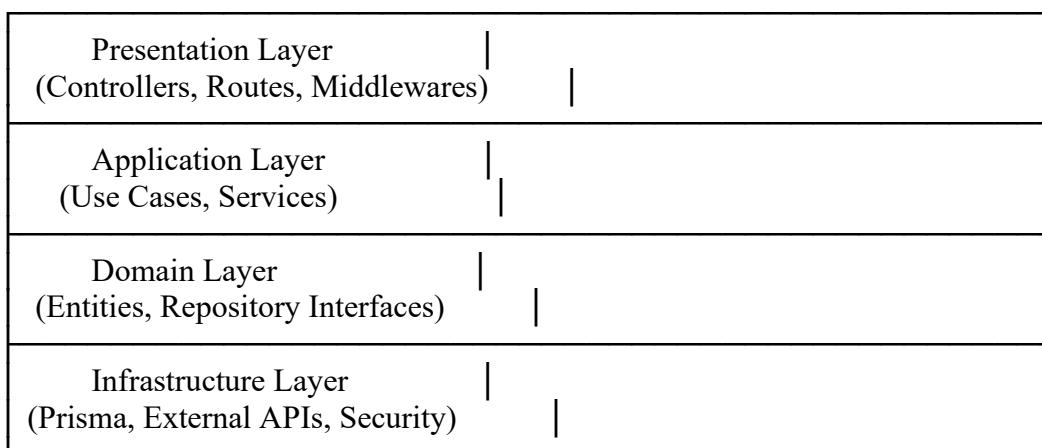
Implementado

Característica	Descripción	Estado
Clean Architecture	Separación en capas bien definidas	<input checked="" type="checkbox"/> Completo
Prisma ORM	ORM moderno con type-safety	<input checked="" type="checkbox"/> Completo
Autenticación JWT	Access + Refresh Tokens	<input checked="" type="checkbox"/> Completo
Roles y Permisos	Sistema granular y dinámico	<input checked="" type="checkbox"/> Completo
WhatsApp Integration	Notificaciones automáticas	<input checked="" type="checkbox"/> Completo
PDF Generation	Comprobantes de pago	<input checked="" type="checkbox"/> Completo
Cron Jobs	Recordatorios automáticos	<input checked="" type="checkbox"/> Completo
Validación Única	Un crédito activo por cliente	<input checked="" type="checkbox"/> Completo
Auditoría	Tracking completo de cambios	<input checked="" type="checkbox"/> Completo
Docker Support	Contenedorización	<input checked="" type="checkbox"/> Completo

3. Arquitectura del Sistema

3.1 Clean Architecture

El proyecto sigue los principios de Clean Architecture con cuatro capas principales:



3.2 Estructura de Directorio

creditos-backend/

```
  prisma/
    schema.prisma      # Esquema de base de datos
    seed.js            # Datos iniciales
    migrations/        # Migraciones generadas
```

```
src/
  domain/
    entities/      # Entidades de negocio
      User.js
      Client.js
      Credit.js
      Payment.js
    repositories/   # Interfaces de repositorios
  application/
    use-cases/     # Casos de uso
      LoginUserUseCase.js
      RefreshTokenUseCase.js
      LogoutUserUseCase.js
      CreateCreditUseCase.js
      CreatePaymentUseCase.js
      CreateUserUseCase.js
      UpdateUserUseCase.js
      DeleteUserUseCase.js
      ListUsersUseCase.js
      GetUserByIdUseCase.js
      ManageCreditStatusUseCase.js
  infrastructure/
    database/
      prismaClient.js
    repositories/   # Implementaciones Prisma
      PrismaUserRepository.js
      PrismaClientRepository.js
      PrismaCreditRepository.js
      PrismaPaymentRepository.js
      PrismaRefreshTokenRepository.js
    security/       # JWT, Password
      JwtService.js
      PasswordService.js
    integrations/   # APIs externas
      whatsappService.js
      pdfService.js
      emailService.js
  presentation/
    controllers/   # Controladores HTTP
      AuthController.js
      UsersController.js
      CreditsController.js
    middlewares/   # Middlewares
      authMiddleware.js
      authorize.js
      validate.js
      errorHandler.js
```

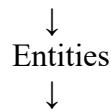
```

    └── asyncHandler.js
    └── routes/      # Definición de rutas
        ├── authRoutes.js
        ├── userRoutes.js
        └── creditRoutes.js
    └── cron/       # Jobs programados
        └── notificationsCron.js
    └── app.js       # Configuración Express
    └── server.js   # Punto de entrada
    └── .env.example # Variables de entorno
    └── docker-compose.yml # Docker config
    └── Dockerfile     # Docker image
    └── package.json   # Dependencies

```

3.3 Flujo de Datos

Request → Route → Middleware → Controller → Use Case → Repository → Database



4. Tecnologías Utilizadas

4.1 Core Technologies

Tecnología	Versión	Propósito
Node.js	18+	Runtime JavaScript
Express	5.1.0	Framework web
PostgreSQL	12+	Base de datos relacional
Prisma	6.2.0	ORM moderno
TypeScript	-	Type safety (opcional)

4.2 Authentication & Security

Tecnología	Versión	Propósito
jsonwebtoken	9.0.2	JWT tokens
bcrypt	6.0.0	Hash de contraseñas
express-validator	7.0.0	Validación de entrada

4.3 Integrations & Services

Tecnología	Versión	Propósito
axios	1.7.9	Cliente HTTP
pdfkit	0.15.2	Generación de PDFs
node-cron	3.0.3	Tareas programadas

Tecnología	Versión	Propósito
nodemailer	7.0.10	Envío de emails

4.4 Development Tools

Tecnología	Versión	Propósito
nodemon	3.1.10	Hot reload
Docker	-	Contenedorización
dotenv	17.2.3	Variables de entorno

5. Instalación y Configuración

5.1 Requisitos Previos

- Node.js v18 o superior
- PostgreSQL v12 o superior
- npm o yarn
- Docker (opcional)

5.2 Instalación Paso a Paso

Opción A: Instalación Manual

1. Clonar el repositorio

```
git clone https://github.com/jesebe4991/creditos-backend.git
```

```
cd creditos-backend
```

2. Instalar dependencias

```
npm install
```

3. Configurar variables de entorno

```
cp .env.example .env
```

Editar .env con tus configuraciones

4. Crear base de datos

```
psql -U postgres
```

```
CREATE DATABASE creditos_db;
```

```
\q
```

5. Generar Prisma Client

```
npx prisma generate
```

6. Ejecutar migraciones

```
npx prisma migrate dev --name init
```

7. Cargar datos iniciales

```
npm run prisma:seed
```

8. Iniciar servidor

npm run dev

Opción B: Instalación con Docker

1. Clonar el repositorio

git clone https://github.com/jesebe4991/creditos-backend.git

cd creditos-backend

2. Levantar contenedores

docker-compose up -d

3. Copiar y configurar .env

cp .env.docker .env

4. Instalar dependencias (en host)

npm install

5. Generar Prisma Client

npx prisma generate

6. Ejecutar migraciones

npx prisma migrate dev --name init

7. Cargar datos iniciales

npm run prisma:seed

8. Iniciar servidor

npm run dev

5.3 Variables de Entorno

```
# =====
# APlicacióN
# =====
NODE_ENV=development
PORT=3000

# =====
# BASE DE DATOS (PRISMA)
# =====
DATABASE_URL=postgresql://postgres:postgres123@localhost:5432/creditos_db

# =====
# AUTENTICACIóN JWT
# =====
JWT_SECRET=tu-clave-secreta-muy-segura-cambiala-en-produccion
JWT_EXPIRATION=15m
REFRESH_TOKEN_EXPIRATION_DAYS=30
```

```

# =====
# WHATSAPP CLOUD API
# =====
WHATSAPP_TOKEN=EAxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
WHATSAPP_PHONE_ID=123456789012345
WHATSAPP_API_URL=https://graph.facebook.com/v18.0

# =====
# CONFIGURACIÓN GLOBAL
# =====
GLOBAL_INTEREST_RATE=0.20
CRON_SCHEDULE=0 8 * * *

# =====
# EMAIL (OPCIONAL)
# =====
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USER=tu-email@gmail.com
EMAIL_PASSWORD=tu-app-password
EMAIL_FROM=noreply@creditos.com

```

5.4 Verificación de Instalación

Health check

curl http://localhost:3000/health

Login con usuario por defecto

```
curl -X POST http://localhost:3000/api/auth/login \
-H "Content-Type: application/json" \
-d '{"email":"admin@creditos.com","password":"Admin123!"'}
```

5.5 Credenciales por Defecto

Usuario Administrador: - Email: admin@creditos.com - Password: Admin123!

⚠ IMPORTANTE: Cambiar esta contraseña en producción.

6. Modelo de Base de Datos

6.1 Esquema Completo

El sistema utiliza 11 tablas principales con relaciones bien definidas:

Usuarios y Autenticación

- **users** - Usuarios del sistema
- **roles** - Roles disponibles

- **permissions** - Permisos granulares
- **role_permissions** - Relación roles-permisos
- **user_roles** - Relación usuarios-roles
- **refresh_tokens** - Tokens de renovación

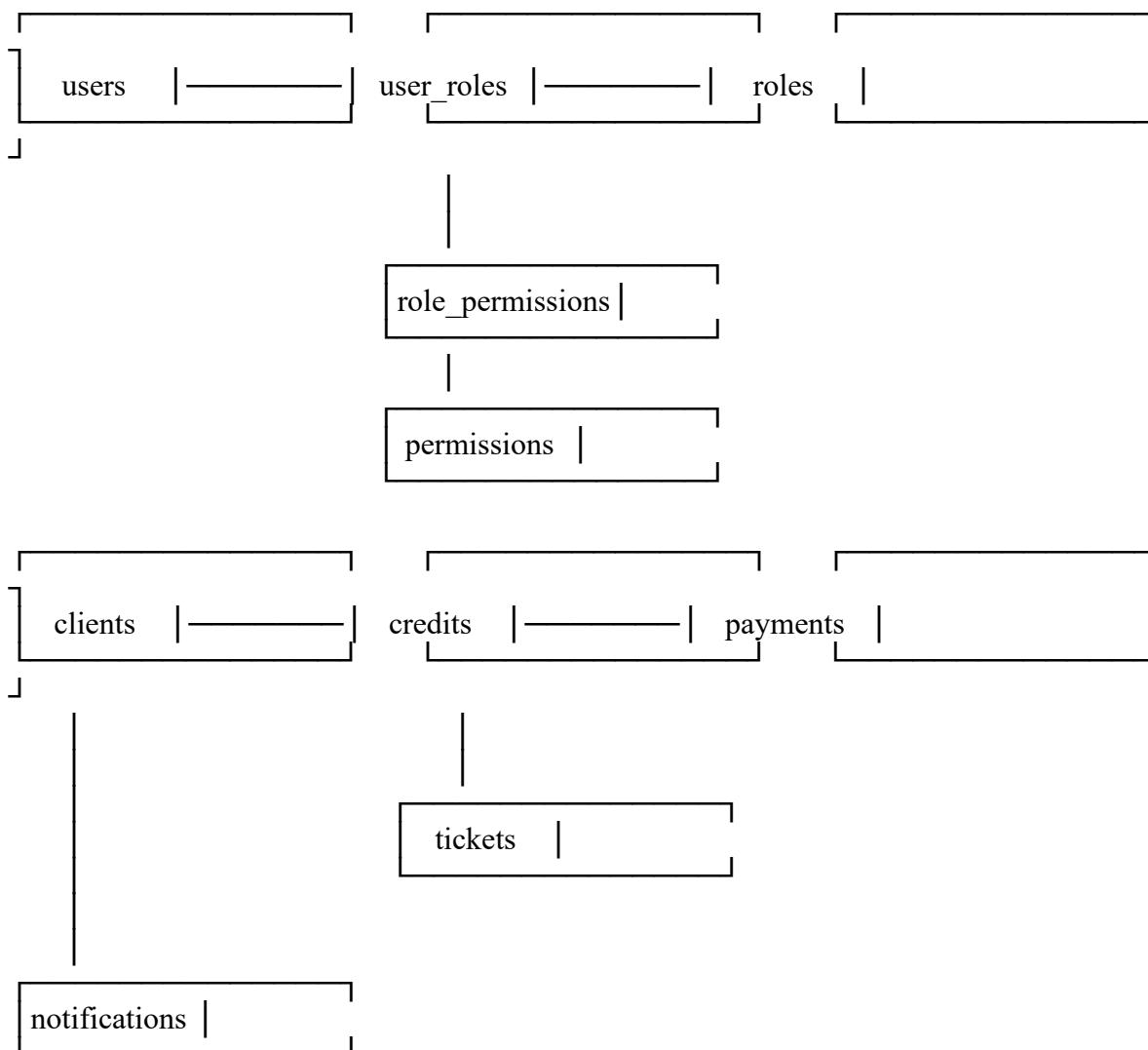
Gestión de Créditos

- **clients** - Clientes
- **credits** - Créditos otorgados
- **payments** - Pagos realizados
- **tickets** - Comprobantes de pago
- **notifications** - Historial de notificaciones

Configuración

- **config** - Configuraciones globales

6.2 Diagrama Entidad-Relación



6.3 Tablas Principales

users

```
CREATE TABLE "users" (
    "id" SERIAL PRIMARY KEY,
    "nombre" VARCHAR(255) NOT NULL,
    "email" VARCHAR(255) UNIQUE NOT NULL,
    "passwordHash" VARCHAR(255) NOT NULL,
    "telefono" VARCHAR(20),
    "is_active" BOOLEAN DEFAULT true,
    "created_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    "created_by" INTEGER REFERENCES users(id),
    "updated_at" TIMESTAMP,
    "updated_by" INTEGER REFERENCES users(id)
);
```

clients

```
CREATE TABLE "clients" (
    "id" SERIAL PRIMARY KEY,
    "nombre" VARCHAR(255) NOT NULL,
    "cedula" VARCHAR(50) UNIQUE NOT NULL,
    "direccion" TEXT,
    "telefono" VARCHAR(20) NOT NULL,
    "referencias" TEXT,
    "modalidad_pago" VARCHAR(100),
    "assigned_to" INTEGER REFERENCES users(id),
    "created_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    "created_by" INTEGER,
    "updated_at" TIMESTAMP,
    "updated_by" INTEGER
);
```

credits

```
CREATE TABLE "credits" (
    "id" SERIAL PRIMARY KEY,
    "numero_credito" VARCHAR(50) UNIQUE NOT NULL,
    "cliente_id" INTEGER REFERENCES clients(id),
    "monto_principal" DECIMAL(15,2) NOT NULL,
    "cuotas" INTEGER NOT NULL,
    "tasa_interes_aplicada" DECIMAL(5,4) NOT NULL,
    "fecha_vencimiento" TIMESTAMP NOT NULL,
    "estado" "EstadoCredito" DEFAULT 'PENDIENTE',
    "created_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    "created_by" INTEGER REFERENCES users(id),
    "updated_at" TIMESTAMP,
    "updated_by" INTEGER REFERENCES users(id)
);
```

```

payments
CREATE TABLE "payments" (
    "id" SERIAL PRIMARY KEY,
    "credit_id" INTEGER REFERENCES credits(id),
    "cliente_id" INTEGER REFERENCES clients(id),
    "user_id" INTEGER REFERENCES users(id),
    "monto" DECIMAL(15,2) NOT NULL,
    "fecha_pago" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    "metodo_pago" "MetodoPago" NOT NULL,
    "cuota_numero" INTEGER NOT NULL,
    "comprobante_referencia" VARCHAR(255),
    "created_at" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    "created_by" INTEGER,
    "updated_at" TIMESTAMP,
    "updated_by" INTEGER
);

```

6.4 Enums

```

enum EstadoCredito {
    PENDIENTE
    ACTIVO
    PAGADO
    INCUMPLIDO
    RENOVADO
    RECHAZADO
}

```

```

enum MetodoPago {
    EFECTIVO
    TRANSFERENCIA
    CHEQUE
    TARJETA
}

```

```

enum TipoNotificacion {
    RECORDATORIO_PAGO
    VENCIMIENTO_PROXIMO
    CREDITO_VENCIDO
    PAGO_REGISTRADO
    CREDITO_APROBADO
    CREDITO_RECHAZADO
}

```

```

enum MedioNotificacion {
    WHATSAPP
    EMAIL
    SMS
}

```

```
}
```

```
enum EstadoEnvio {
    SIN_ENVIAR
    ENVIADO
    FALLIDO
}
```

6.5 Índices Importantes

-- *Usuarios*

```
CREATE INDEX idx_users_email ON users(email);
```

-- *Clientes*

```
CREATE INDEX idx_clients_cedula ON clients(cedula);
CREATE INDEX idx_clients_telefono ON clients(telefono);
CREATE INDEX idx_clients_assigned_to ON clients(assigned_to);
```

-- *Créditos*

```
CREATE INDEX idx_credits_numero_credito ON credits(numero_credito);
CREATE INDEX idx_credits_cliente_id ON credits(cliente_id);
CREATE INDEX idx_credits_estado ON credits(estado);
CREATE INDEX idx_credits_fecha_vencimiento ON credits(fecha_vencimiento);
```

-- *Pagos*

```
CREATE INDEX idx_payments_credit_id ON payments(credit_id);
CREATE INDEX idx_payments_cliente_id ON payments(cliente_id);
CREATE INDEX idx_payments_fecha_pago ON payments(fecha_pago);
```

7. Sistema de Autenticación

7.1 JWT + Refresh Tokens

El sistema implementa autenticación moderna con dos tipos de tokens:

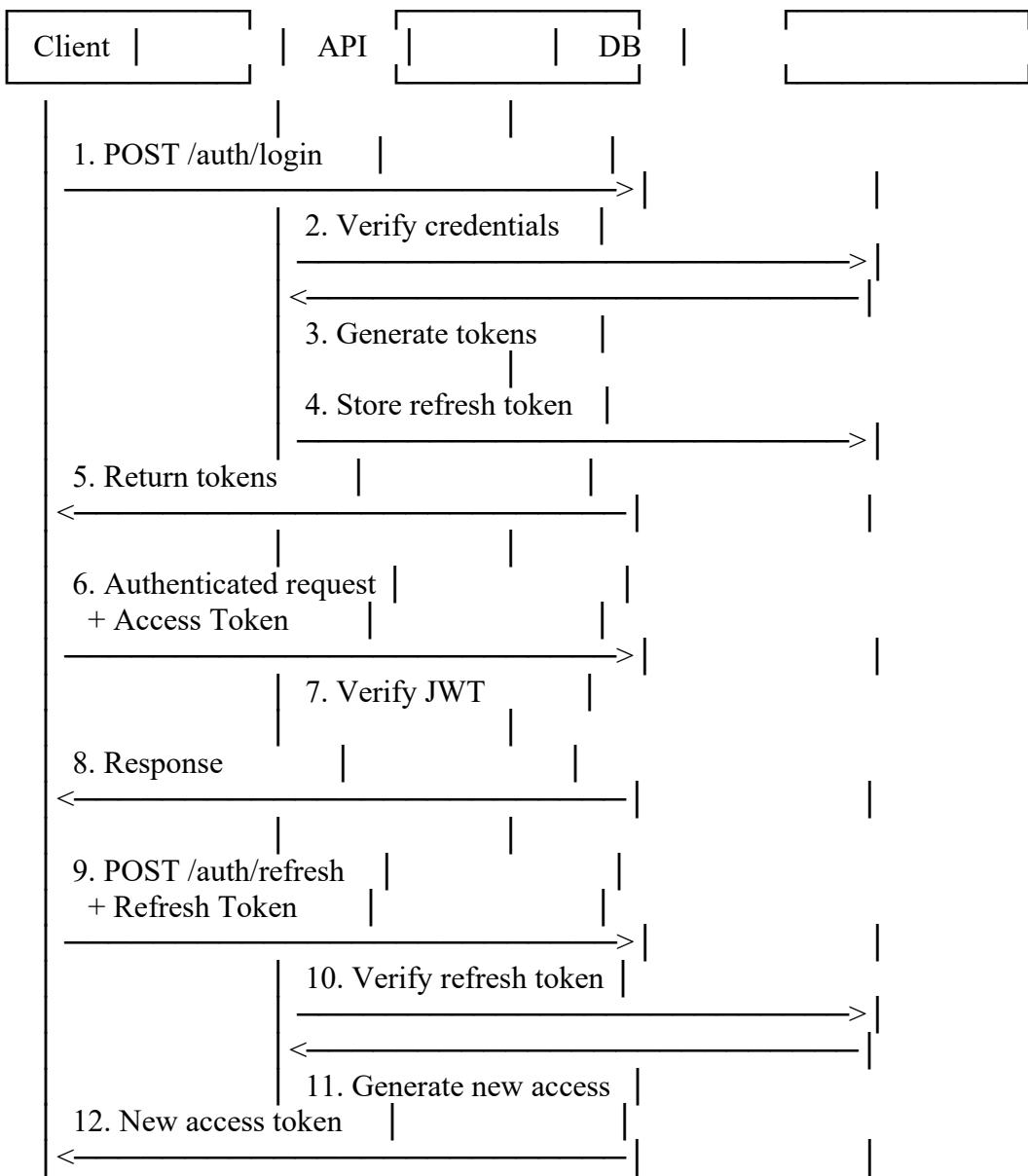
Access Token

- **Duración:** 15 minutos (configurable)
- **Propósito:** Autenticación de requests
- **Storage:** Cliente (memory/state)
- **Renovable:** Sí, con refresh token

Refresh Token

- **Duración:** 30 días (configurable)
- **Propósito:** Renovar access token
- **Storage:** Base de datos + cliente
- **Revocable:** Sí

7.2 Flujo de Autenticación



7.3 Implementación

Login
// POST /api/auth/login
{
 "email": "admin@creditos.com",
 "password": "Admin123!"
}

// Response
{
 "success": true,

```

"message": "Login exitoso",
"data": {
  "user": {
    "id": 1,
    "email": "admin@creditos.com",
    "nombre": "Administrador",
    "roles": ["ADMIN"],
    "permissions": [...]
  },
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6...",
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}
}

```

Refresh Token

```

// POST /api/auth/refresh
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}

```

// Response

```

{
  "success": true,
  "message": "Token renovado",
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
  }
}

```

Logout

```

// POST /api/auth/logout
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."
}

```

// Response

```

{
  "success": true,
  "message": "Logout exitoso"
}

```

7.4 Seguridad

- Contraseñas hasheadas con bcrypt (10 salt rounds)
- JWT firmados con secret key
- Refresh tokens persistidos en BD

- Refresh tokens revocables
 - Refresh token rotation (opcional)
 - Validación de expiración
 - Protección contra timing attacks
-

8. Sistema de Roles y Permisos

8.1 Arquitectura RBAC

El sistema implementa Role-Based Access Control (RBAC) con:

- **Roles:** Grupos de permisos (ADMIN, COBRADOR)
- **Permisos:** Acciones específicas (users.create, credits.approve)
- **Usuarios:** Pueden tener múltiples roles
- **Dinámico:** Roles y permisos configurables en BD

8.2 Roles Disponibles

ADMIN

- **Descripción:** Administrador con acceso total
- **Permisos:** Todos (21 permisos)
- **Capacidades:**
 - Gestión completa de usuarios
 - Gestión de roles y permisos
 - Aprobar/rechazar créditos
 - Acceso a reportes y dashboard

COBRADOR

- **Descripción:** Cobrador de créditos
- **Permisos:** Limitados (7 permisos)
- **Capacidades:**
 - Ver y crear clientes
 - Ver créditos
 - Registrar pagos
 - Ver dashboard básico

8.3 Lista de Permisos

Módulo	Acción	Permiso	Descripción
users	create	users.create	Crear usuarios
users	read	users.read	Ver usuarios
users	update	users.update	Actualizar usuarios
users	delete	users.delete	Eliminar usuarios
clients	create	clients.create	Crear clientes

Módulo	Acción	Permiso	Descripción
clients	read	clients.read	Ver clientes
clients	update	clients.update	Actualizar clientes
clients	delete	clients.delete	Eliminar clientes
credits	create	credits.create	Crear créditos
credits	read	credits.read	Ver créditos
credits	update	credits.update	Actualizar créditos
credits	delete	credits.delete	Eliminar créditos
credits	approve	credits.approve	Aprobar créditos
credits	reject	credits.reject	Rechazar créditos
payments	create	payments.create	Registrar pagos
payments	read	payments.read	Ver pagos
reports	read	reports.read	Ver reportes
dashboard	read	dashboard.read	Ver dashboard
roles	create	roles.create	Crear roles
roles	read	roles.read	Ver roles
roles	update	roles.update	Actualizar roles
roles	delete	roles.delete	Eliminar roles

8.4 Uso de Middleware

```

authorize (Permisos)
// Requiere uno o más permisos específicos
router.post('/credits',
  authMiddleware,
  authorize('credits.create'),
  controller.store
);

// OR logic - requiere al menos uno
router.get('/dashboard',
  authMiddleware,
  authorize('dashboard.read', 'reports.read'),
  controller.dashboard
);

authorizeRole (Roles)
// Requiere rol específico
router.post('/users',
  authMiddleware,
  authorizeRole('ADMIN'),
  controller.store
);

```

```
// OR logic - requiere al menos uno
router.get('/admin-panel',
  authMiddleware,
  authorizeRole('ADMIN', 'SUPERVISOR'),
  controller.adminPanel
);
```

8.5 Asignación de Roles

// Crear usuario con rol

```
POST /api/users
{
  "nombre": "Juan Cobrador",
  "email": "cobrador@creditos.com",
  "password": "password123",
  "roleIds": [2] // COBRADOR
}
```

// Asignar rol a usuario existente

```
POST /api/users/2/roles
{
  "roleId": 1 // ADMIN
}
```

// Remover rol

```
DELETE /api/users/2/roles/1
```

9. Endpoints de la API

9.1 Autenticación

POST /api/auth/login

Descripción: Autenticar usuario y obtener tokens

Request:

```
{
  "email": "admin@creditos.com",
  "password": "Admin123!"
}
```

Response (200):

```
{
  "success": true,
  "message": "Login exitoso",
  "data": {
    "token": "JWTToken1234567890"
  }
}
```

```
"user": {  
    "id": 1,  
    "email": "admin@creditos.com",  
    "nombre": "Administrador",  
    "roles": ["ADMIN"],  
    "permissions": [...]  
},  
"accessToken": "eyJhbGci...",  
"refreshToken": "eyJhbGci..."  
}  
}
```

POST /api/auth/refresh

Descripción: Renovar access token

Request:

```
{  
    "refreshToken": "eyJhbGci..."  
}
```

POST /api/auth/logout

Descripción: Cerrar sesión y revocar refresh token

Request:

```
{  
    "refreshToken": "eyJhbGci..."  
}
```

GET /api/auth/profile

Descripción: Obtener perfil del usuario autenticado

Headers:

Authorization: Bearer {accessToken}

9.2 Usuarios

GET /api/users

Descripción: Listar usuarios con paginación

Query Params: - page (number): Página (default: 1) - limit (number): Límite (default: 10, max: 100) - search (string): Búsqueda por nombre/email - isActive (boolean): Filtrar por estado - roleId (number): Filtrar por rol

Headers:

Authorization: Bearer {accessToken}

Response (200):

```
{  
  "success": true,  
  "data": [...],  
  "meta": {  
    "total": 100,  
    "page": 1,  
    "limit": 10,  
    "totalPages": 10  
  }  
}
```

GET /api/users/:id

Descripción: Obtener usuario por ID

POST /api/users

Descripción: Crear nuevo usuario

Request:

```
{  
  "nombre": "Juan Pérez",  
  "email": "juan@example.com",  
  "password": "password123",  
  "telefono": "555-1234",  
  "roleIds": [2]  
}
```

PUT /api/users/:id

Descripción: Actualizar usuario

DELETE /api/users/:id

Descripción: Desactivar usuario (soft delete)

DELETE /api/users/:id/hard

Descripción: Eliminar permanentemente

POST /api/users/:id/roles

Descripción: Asignar rol a usuario

DELETE /api/users/:id/roles/:roleId

Descripción: Remover rol de usuario

9.3 Créditos

GET /api/credits

Descripción: Listar créditos con filtros

Query Params: - page, limit: Paginación - estado: Filtrar por estado - clientId: Filtrar por cliente - cobradorId: Filtrar por cobrador

GET /api/credits/:id

Descripción: Obtener crédito por ID con detalles

Response:

```
{  
  "success": true,  
  "data": {  
    "id": 1,  
    "numeroCredito": "CRE-2025-000001",  
    "estado": "ACTIVO",  
    "montoPrincipal": "5000.00",  
    "cuotas": 12,  
    "montoTotal": "6000.00",  
    "valorCuota": "500.00",  
    "totalInteres": "1000.00",  
    "totalPagado": "1500.00",  
    "saldoPendiente": "4500.00",  
    ...  
  }  
}
```

POST /api/credits

Descripción: Crear nuevo crédito

Request:

```
{  
  "clientId": 1,  
  "montoPrincipal": 5000,  
  "cuotas": 12,  
  "tasaInteresAplicada": 0.20,  
  "fechaVencimiento": "2026-01-03"  
}
```

POST /api/credits/:id/approve

Descripción: Aprobar crédito pendiente

Validaciones: - ✓ Crédito debe estar en estado PENDIENTE - ✓ Cliente no debe tener otro crédito activo - ✓ Usuario debe tener permiso credits.approve

Response:

```
{  
  "success": true,  
  "message": "Crédito aprobado exitosamente",  
  "data": {  
    "credit": {...},  
    "notificacionEnviada": true  
  }  
}
```

POST /api/credits/:id/approve

Descripción: Aprobar crédito pendiente

Request:

```
{  
  "motivo": "Documentación incompleta"  
}
```

9.4 Health Check

GET /health

Descripción: Verificar estado del servidor

Response:

```
{  
  "success": true,  
  "message": "Server is running",  
  "timestamp": "2025-11-03T10:00:00.000Z",  
  "env": "development"  
}
```

10. Validaciones de Negocio

10.1 Regla Crítica: Un Crédito Activo por Cliente

Descripción: Un cliente solo puede tener UN crédito en estado ACTIVO o INCUMPLIDO a la vez.

Estados bloqueantes: - ACTIVO - INCUMPLIDO

Estados permitidos: - PENDIENTE - PAGADO - RENOVADO - RECHAZADO

Implementación:

// En CreateCreditUseCase

```
const hasActiveCredit = await creditRepository.hasActiveCredit(clienteId);

if (hasActiveCredit) {
    throw new Error('El cliente ya tiene un crédito activo');
}
```

// En ManageCreditStatusUseCase.approve()

```
const hasActiveCredit = await creditRepository.hasActiveCredit(credit.clienteId);

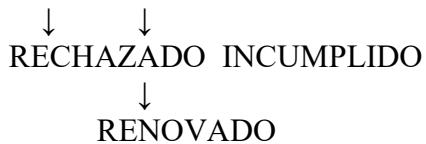
if (hasActiveCredit) {
    throw new Error('El cliente ya tiene un crédito ACTIVO o INCUMPLIDO');
}
```

Query de validación:

```
SELECT COUNT(*) FROM credits
WHERE cliente_id = ?
    AND estado IN ('ACTIVO', 'INCUMPLIDO')
```

10.2 Estados de Crédito

PENDIENTE → ACTIVO → PAGADO



10.3 Validaciones de Pago

- ✓ El crédito debe existir
- ✓ El monto debe ser mayor a 0
- ✓ El número de cuota debe ser válido
- ✓ El método de pago es requerido
- ✓ El crédito se marca como PAGADO automáticamente cuando el total pagado
>= monto total

10.4 Validaciones de Usuario

- ✓ Email único
 - ✓ Formato de email válido
 - ✓ Contraseña mínimo 6 caracteres
 - ✓ No se puede eliminar al admin principal
 - ✓ No se puede eliminar a sí mismo
-

11. Integraciones Externas

11.1 WhatsApp Cloud API

Propósito: Envío de notificaciones automáticas a clientes

Configuración:

WHATAPP_TOKEN=EAxxxxxxxxxxxxxx
WHATAPP_PHONE_ID=123456789012345
WHATAPP_API_URL=https://graph.facebook.com/v18.0

Mensajes enviados: - Crédito aprobado - Crédito rechazado - Pago registrado -
 Recordatorio de vencimiento próximo - Alerta de crédito vencido

Ejemplo de implementación:

```
// whatsappService.js
async sendTextMessage(phoneNumber, message) {
  const response = await axios.post(
    `${WHATAPP_API_URL}/${WHATAPP_PHONE_ID}/messages`,
    {
      messaging_product: 'whatsapp',
      to: phoneNumber,
      type: 'text',
      text: { body: message }
    },
    {
      headers: {
        'Authorization': `Bearer ${WHATAPP_TOKEN}`,
        'Content-Type': 'application/json'
      }
    }
  );

  return {
    success: true,
    messageId: response.data.messages[0].id
  };
}
```

Registro de notificaciones:

Todas las notificaciones se registran en la tabla notifications con: - Cliente receptor - Tipo de notificación - Mensaje enviado - Medio de envío (WHATAPP) - Estado de envío (ENVIADO/FALLIDO) - Response de la API - Fecha de envío

11.2 PDF Generation (PDFKit)

Propósito: Generar comprobantes de pago

Características: - ✓ PDF guardado como bytea en BD - ✓ Metadata en JSON - ✓ Diseño profesional - ✓ Información completa del pago

Contenido del ticket: - Datos del cliente - Datos del crédito - Datos del pago - Número de comprobante - Fecha de emisión - Monto destacado

Implementación:

```
async generateTicketPDF(payment, client, credit, numeroComprobante) {  
    const doc = new PDFDocument();  
  
    // Encabezado  
    doc.fontSize(20).text('COMPROBANTE DE PAGO', { align: 'center' });  
  
    // Información del cliente  
    doc.fontSize(14).text('DATOS DEL CLIENTE');  
    doc.fontSize(11).text(`Nombre: ${client.nombre}`);  
  
    // ... resto del contenido  
  
    return pdfBuffer;  
}
```

11.3 Email (Nodemailer) - Opcional

Propósito: Envío de recordatorios por email

Configuración:

```
EMAIL_HOST=smtp.gmail.com  
EMAIL_PORT=587  
EMAIL_USER=tu-email@gmail.com  
EMAIL_PASSWORD=tu-app-password
```

Estado: Implementado pero opcional

12. Cron Jobs y Notificaciones

12.1 Notifications Cron Job

Propósito: Enviar recordatorios automáticos de pago

Schedule: Diario a las 8 AM (configurable)

Configuración:

```
CRON_SCHEDULE=0 8 * * *  
DIAS_RECORDATORIO_PREVIO=3
```

Funcionalidades:

1. Recordatorios de Vencimiento Próximo
// Busca créditos que vencen en 3 días
const creditosProximos = **await** creditRepository.findUpcomingDue(3);

// Envía recordatorio por WhatsApp
const mensaje = `

🔔 Recordatorio de Pago

Su crédito está próximo a vencer.
Vence en: \${diasRestantes} día(s)
`;

await whatsappService.sendTextMessage(client.telefono, mensaje);
await prisma.notification.create({...});

2. Alertas de Crédito Vencido

// Busca créditos vencidos

const creditosVencidos = **await** creditRepository.findOverdue();

// Envía alerta y actualiza estado
const mensaje = `⚠️ CRÉDITO VENCIDO
Su crédito se encuentra VENCIDO.
Días vencidos: \${diasVencidos}
`;

await whatsappService.sendTextMessage(client.telefono, mensaje);

// Actualiza estado a INCUMPLIDO
await prisma.credit.update({
 where: { id: credit.id },
 data: { estado: 'INCUMPLIDO' }
});

12.2 Ejecución Manual

Ejecutar manualmente para testing
node src/jobs/notifyPayments.js --once

12.3 Logs del Cron

- ⌚ Cron job de notificaciones iniciado. Schedule: 0 8 * * *
- ⌚ Ejecutando tarea de notificaciones...
- 📅 Enviando recordatorios - 2025-11-03T08:00:00.000Z
- 📋 Créditos próximos a vencer: 5
- ⚠️ Créditos vencidos: 2
- ✓ Recordatorio enviado a Juan Pérez (CRE-2025-000001)
- ✓ Tarea de notificaciones completada

13. Manejo de Errores

13.1 Códigos de Error

Código	HTTP	Descripción
VALIDATION_ERROR	400	Error de validación
UNAUTHORIZED	401	No autenticado
FORBIDDEN	403	Sin permisos
NOT_FOUND	404	Recurso no encontrado
DUPLICATE_ENTRY	409	Registro duplicado
INVALID_REFERENCE	400	Foreign key inválida
DATABASE_ERROR	500	Error de BD
INTERNAL_ERROR	500	Error interno

13.2 Formato de Respuesta de Error

```
{  
  "success": false,  
  "error": "Mensaje descriptivo del error",  
  "code": "VALIDATION_ERROR",  
  "field": "email" // (opcional)  
}
```

13.3 Errores de Prisma

El sistema maneja automáticamente errores específicos de Prisma:

- **P2002** (Unique constraint): 409 - Email/cédula duplicada
- **P2025** (Record not found): 404 - Registro no encontrado
- **P2003** (Foreign key constraint): 400 - Referencia inválida
- **P2014** (Relation violation): 400 - No se puede eliminar

13.4 Middleware Global

```
const errorHandler = (err, req, res, next) => {  
  console.error('Error:', err);
```

```
// Errores de Prisma  
if (err.code?.startsWith('P')) {  
  return handlePrismaError(err, res);  
}
```

```
// Errores de JWT  
if (err.name === 'JsonWebTokenError') {  
  return res.status(401).json({  
    success: false,  
    error: 'Token inválido',
```

```

        code: 'INVALID_TOKEN'
    });
}

// Error por defecto
res.status(err.statusCode || 500).json({
    success: false,
    error: err.message,
    code: err.code || 'INTERNAL_ERROR'
});

```

13.5 Validaciones con express-validator

```

const validate = (req, res, next) => {
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        return res.status(400).json({
            success: false,
            error: 'Errores de validación',
            code: 'VALIDATION_ERROR',
            errors: errors.array().map(err => ({
                field: err.path,
                message: err.msg,
                value: err.value
            }))
        });
    }

    next();
};

```

14. Docker y Despliegue

14.1 Docker Compose

Servicios: - PostgreSQL 15 - pgAdmin (interfaz web)

Configuración:

```

services:
  postgres:
    image: postgres:15-alpine
    container_name: creditos_postgres
    ports:
      - "5432:5432"
    environment:

```

```

POSTGRES_USER: postgres
POSTGRES_PASSWORD: postgres123
POSTGRES_DB: creditos_db
volumes:
  - postgres_data:/var/lib/postgresql/data
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U postgres"]
  interval: 10s

pgadmin:
image: dpage/pgadmin4:latest
container_name: creditos_pgadmin
ports:
  - "5050:80"
environment:
  PGADMIN_DEFAULT_EMAIL: admin@creditos.com
  PGADMIN_DEFAULT_PASSWORD: admin123

```

14.2 Comandos Docker

Levantar servicios
 docker-compose up -d

Ver logs
 docker-compose logs -f

Detener servicios
 docker-compose stop

Eliminar todo (incluyendo datos)
 docker-compose down -v

Acceder a PostgreSQL
 docker exec -it creditos_postgres psql -U postgres -d creditos_db

Backup de BD
 docker exec -t creditos_postgres pg_dump -U postgres creditos_db > backup.sql

Restaurar backup
 docker exec -i creditos_postgres psql -U postgres -d creditos_db < backup.sql

14.3 pgAdmin

Acceso: http://localhost:5050

Credenciales: - Email: admin@creditos.com - Password: admin123

Conectar a PostgreSQL: 1. Servers → Register → Server 2. Name: Creditos DB 3. Connection: - Host: postgres - Port: 5432 - Database: creditos_db - Username: postgres - Password: postgres123

14.4 Dockerfile para Producción

```
FROM node:20-bullseye AS builder
WORKDIR /app
COPY package*.json .
RUN npm ci --prefer-offline
COPY ..
RUN npx prisma generate

FROM node:20-bullseye-slim AS runner
WORKDIR /app
ENV NODE_ENV=production
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/prisma ./prisma
COPY --from=builder /app/src ./src
COPY --from=builder /app/package*.json .
EXPOSE 3000
USER app
CMD ["node", "src/server.js"]
```

14.5 CI/CD con GitHub Actions

name: CI - Creditos Backend

```
on:
  push:
    branches: [main, develop]
  pull_request:

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 20
      - run: npm ci
      - run: npm run lint --if-present
      - run: docker build -t creditos-backend .
```

15. Guía de Desarrollo

15.1 Flujo de Trabajo

1. Crear rama para feature

```
git checkout -b feature/nueva-funcionalidad
```

2. Desarrollar

```
# ... código ...
```

3. Ejecutar tests (cuando estén implementados)

```
npm test
```

4. Commit

```
git add .
```

```
git commit -m "feat: descripción del cambio"
```

5. Push

```
git push origin feature/nueva-funcionalidad
```

6. Crear Pull Request en GitHub

15.2 Convenciones de Código

Nombres de Archivos

- **Controllers:** UsersController.js
- **Use Cases:** CreateUserUseCase.js
- **Repositories:** PrismaUserRepository.js
- **Services:** whatsappService.js
- **Middlewares:** authMiddleware.js

Nombres de Funciones

- **camelCase:** getUserById(), createCredit()
- **Descriptivos:** Nombres que explican qué hacen

Estructura de Controller

```
class UsersController {  
    // GET /api/users  
    index = [validations, validate, asyncHandler(async (req, res) => {...})];  
  
    // GET /api/users/:id  
    show = [validations, validate, asyncHandler(async (req, res) => {...})];  
  
    // POST /api/users  
    store = [validations, validate, asyncHandler(async (req, res) => {...})];  
  
    // PUT /api/users/:id  
    update = [validations, validate, asyncHandler(async (req, res) => {...})];
```

```
// DELETE /api/users/:id
destroy = [validations, validate, asyncHandler(async (req, res) => {...})];
}
```

15.3 Prisma Workflows

Crear nueva migración

```
npx prisma migrate dev --name nombre_migracion
```

Aplicar migraciones en producción

```
npx prisma migrate deploy
```

Generar Prisma Client

```
npx prisma generate
```

Ver estado de migraciones

```
npx prisma migrate status
```

Abrir Prisma Studio

```
npx prisma studio
```

Resetear BD (desarrollo)

```
npx prisma migrate reset
```

Formatear schema

```
npx prisma format
```

15.4 Testing (Pendiente)

Instalar dependencias de testing

```
npm install --save-dev jest supertest
```

Ejecutar tests

```
npm test
```

Coverage

```
npm run test:coverage
```

15.5 Logging

Desarrollo:

```
console.log('✅ Acción exitosa');
console.error('❌ Error:', error);
console.warn('⚠️ Advertencia');
```

Producción (próximo):

```
// Winston o Pino
logger.info('User logged in', { userId: 1 });
logger.error('Database error', { error: err.message });
```

16. Casos de Uso Implementados

16.1 Autenticación

LoginUserUseCase

Input: email, password

Output: user, accessToken, refreshToken

Validaciones: - Usuario existe - Usuario activo - Contraseña correcta

RefreshTokenUseCase

Input: refreshToken

Output: accessToken, refreshToken (nuevo si rotation)

Validaciones: - Token válido - Token no revocado - Token no expirado

LogoutUserUseCase

Input: refreshToken

Output: success

Acción: Revoca refresh token

16.2 Usuarios

CreateUserUseCase

Input: userData, createdBy

Output: newUser

Validaciones: - Email único - Formato válido - Password ≥ 6 caracteres

UpdateUserUseCase

Input: userId, userData, updatedBy

Output: updatedUser

Validaciones: - Usuario existe - Email único (si cambia)

DeleteUserUseCase

Input: userId, deletedBy

Output: success

Validaciones: - No eliminar a sí mismo - No eliminar admin principal

ListUsersUseCase

Input: filters (page, limit, search, isActive, role)

Output: users[], meta

Características: - Paginación - Búsqueda - Filtros

GetUserByIdUseCase

Input: userId

Output: user con roles y permisos

16.3 Créditos

CreateCreditUseCase

Input: creditData, createdBy

Output: newCredit

Validaciones: - Cliente existe - Cliente sin crédito activo - Monto > 0 -
Cuotas > 0

Cálculos automáticos: - Número de crédito único - Tasa de interés (si no se proporciona) - Fecha de vencimiento (si no se proporciona) - Monto total -
Valor cuota

ManageCreditStatusUseCase.approve()

Input: creditId, approvedBy

Output: credit, notificacionEnviada

Validaciones: - Crédito existe - Estado = PENDIENTE - Cliente sin otro crédito activo

Acciones: - Cambia estado a ACTIVO - Envía WhatsApp - Registra notificación - Auditoría (updated_by)

ManageCreditStatusUseCase.reject()

Input: creditId, rejectedBy, motivo

Output: credit, notificacionEnviada

Validaciones: - Crédito existe - Estado = PENDIENTE

Acciones: - Cambia estado a RECHAZADO - Envía WhatsApp con motivo -
Registra notificación - Auditoría

16.4 Pagos

CreatePaymentUseCase

Input: paymentData, createdBy

Output: payment, ticket, creditoActualizado

Proceso (Transacción): 1. Crear pago 2. Generar PDF del ticket 3. Guardar ticket en BD (bytea) 4. Calcular total pagado 5. Actualizar estado crédito si aplica 6. Enviar WhatsApp 7. Registrar notificación

17. Próximos Pasos

17.1 Alta Prioridad

- **Endpoints REST completos**
 - Clientes (CRUD)
 - Pagos (CRUD)
 - Tickets (descarga PDF)
 - Notificaciones (historial, reenvío)
- **Dashboard**
 - Métricas calculadas
 - Gráficas de créditos
 - Indicadores KPI
- **Sistema de Reportes**
 - Reporte de créditos
 - Reporte de pagos
 - Reporte de cobranza
 - Exportación PDF/Excel

17.2 Media Prioridad

- **Testing**
 - Tests unitarios (Jest)
 - Tests de integración (Supertest)
 - Coverage > 80%
- **Documentación**
 - Swagger/OpenAPI
 - Postman Collection
 - Ejemplos de uso
- **Seguridad**
 - Rate limiting
 - Helmet.js
 - CORS configuración avanzada
 - Input sanitization

17.3 Baja Prioridad

- **Monitoreo**

- Winston/Pino logging
 - Health checks detallados
 - Alertas automáticas
- **Performance**
 - Caché con Redis
 - Query optimization
 - Índices adicionales
- **Features Adicionales**
 - Renovación de créditos
 - Cálculo de intereses moratorios
 - Estadísticas por cobrador
 - Sistema de multas

17.4 Refactoring

- TypeScript migration
 - Event-driven architecture
 - Queue system (Bull/BullMQ)
 - Microservices (si escala)
-

Anexos

A. Scripts Útiles

Desarrollo

npm run dev	# Iniciar con nodemon
npm start	# Iniciar en producción

Prisma

npm run prisma:generate	# Generar client
npm run prisma:migrate	# Crear migración
npm run prisma:studio	# Abrir studio
npm run prisma:seed	# Ejecutar seed

Testing (pendiente)

npm test	# Ejecutar tests
npm run test:watch	# Watch mode
npm run test:coverage	# Coverage report

B. Variables de Entorno Completas

Ver archivo .env.example en el repositorio.

C. Recursos Adicionales

- **Prisma Docs:** <https://www.prisma.io/docs>

- **Express Docs:** <https://expressjs.com/>
- **JWT Best Practices:** <https://jwt.io/introduction>
- **WhatsApp Cloud API:** <https://developers.facebook.com/docs/whatsapp>

D. Contribución

Para contribuir al proyecto:

1. Fork del repositorio
2. Crear rama feature
3. Hacer commits descriptivos
4. Push a tu fork
5. Crear Pull Request

E. Licencia

ISC License - Ver LICENSE file

Manual de Usuario

Ingresa con tu credencial Usuario y contraseña

The image shows a login form titled "Crédito Rápido" with the sub-instruction "Inicia sesión en tu cuenta". It features two input fields: "Correo Electrónico" containing "ejemplo@correo.com" and "Contraseña" containing "*****". There is a "Recordarme" checkbox and a blue "Iniciar Sesión" button. Below the button is a link "¿Olvidaste tu contraseña?".

Correo Electrónico	ejemplo@correo.com
Contraseña	*****
<input type="checkbox"/> Recordarme	
Iniciar Sesión	
¿Olvidaste tu contraseña?	

Navega por el menú

Pantalla 2

Dashboard Principal

Crédito Rápido

- Dashboard**
- Cuentas
- Créditos
- Aprobar Créditos
- Pagos
- Calendario
- Usuarios
- Reportes
- Control de Cartera
- Configuración

Dashboard

Resumen general de la plataforma

¡Hola de nuevo, Administrador!

Aquí tienes un resumen de tu actividad reciente

Total Clientes **248**

Créditos Activos **45**

Pagos Hoy **12**

Total Cartera **\$2.5M**

Navega en el modulo de clientes y adminístralos agregándolos, editándolo según tu necesidad

Pantalla 3

Gestión de Clientes

Crédito Rápido

- Clientes**
- Créditos
- Aprobar Créditos
- Pagos
- Calendario
- Usuarios
- Reportes
- Control de Cartera
- Configuración

Gestión de Clientes

Lista de todos los clientes registrados

Nuevo Cliente

Cliente	Cédula	Teléfono	Ciudad	Cobrador	Estado	Acciones
Carlos Ramirez	1234567890	+57 300 123 4567	Calli	Juan Pérez	Activo	
Ana López	9876543210	+57 310 987 6543	Bogotá	Maria García	Activo	
Pedro Martinez	5555555555	+57 320 555 5555	Medellín	Juan Pérez	Inactivo	

Navega y visualiza La gestión de crédito, el total de crédito, pagos y total

Gestión de Créditos

The screenshot shows the 'Gestión de Créditos' (Credit Management) dashboard. On the left is a sidebar with navigation links: Crédito Rápido, Dashboard, Clientes, Créditos (selected), Aprobar Créditos, Pagos, Calendario, Usuarios, Reportes, Control de Cartera, and Configuración. The main area has a title 'Gestión de Créditos' and a subtitle 'Lista de todos los créditos del sistema'. It features three summary cards: 'Créditos Activos' (45), 'Cartera Total' (\$2.5M), and 'Créditos Vencidos' (12). Below these are dropdown filters for 'Todos los estados', 'Todos los clientes', and a search bar 'Buscar crédito...'. A table lists three credit entries with columns: #ID, Cliente, Monto, Cuotas, Saldo Pendiente, Estado, and Acciones. The first entry is 'Activado', the second is 'Pagado', and the third is 'Incumplido'.

Si necesitas crear un nuevo crédito solo asegúrate de que la información este ingresada correctamente y no des espacio en blanco y listo

Crear Nuevo Crédito

The screenshot shows the 'Crear Nuevo Crédito' (Create New Credit) form. The sidebar is identical to the previous dashboard. The main form has sections: 'Seleccionar Cliente' (Client Selection) with a search bar, 'Información del Crédito' (Credit Information) with fields for 'Monto a Prestar' (\$500,000), 'Tasa de Interés (%)' (20%), 'Número de Cuotas' (12), 'Modalidad de Pago' (Select payment mode), and 'Fecha de Inicio' (01/01/2024); and a 'Resumen del Crédito' (Credit Summary) table showing details like 'Monto Primitivo' (\$500,000), 'Interés (20%)' (\$100,000), 'TOTAL A PAGAR' (\$600,000), 'Valor por Cuota' (\$50,000), and 'Número de Cuotas' (12). At the bottom are 'Cancelar' and 'Crear Crédito' buttons.

Así mismo puedes registrar un o varios pagos

Panel 7

Registrar Pago

Registrar Pago
Registrar el pago del cliente

Buscar Crédito
Buscar por Cliente o ID de Crédito *

Información del Crédito

Cliente:	Carlos Ramón González
Crédito ID:	PCR-00123
Saldo Pendiente:	\$100.000
Próxima Cuenta:	#7
Valor Cuenta:	\$50,000

Datos del Pago

Monto del Pago *
50000

Fecha del Pago *
dd/mm/aaaa

Método de Pago *
Seleccionar método

Número de Cuenta +
7

Observaciones:
El pago recibido en la oficina

Cancelar **Registrar Pago**

Ahora si necesitas ver un reporte consolidado con las estadísticas de cada crédito o en general abres el modulo de reportes

Panel 8

Reportes y Estadísticas

Reportes y Estadísticas
informe financiero del sistema

Resumen Financiero

Monto Total Prestado:	\$25,000,000
Monto Cobrado:	\$18,000,000
Saldo Pendiente:	\$7,000,000
Tasa de Recuperación:	72%

Dashboard
Resumen general de la plataforma

Administrador
Administrador

¡Hola de nuevo, Administrador!
Aquí tienes los resúmenes de tu actividad reciente

Total Clientes: 248 **Creditos Activos:** 45 **Pagos Hoy:** 12 **Total Cartera:** \$2.5M

Por ultimo recuerda que tu tienes el control de los clientes nuevos y antiguos solo ingresando al modulo de clientes, los puedes editar y administrar según sea la necesidad

Gestión de Clientes

Cliente	Dirección	Teléfono	Ciudad	Colectrador	Estado	Acciones
Carlos Ramirez	123 Hacienda	+57 300 123 4567	Cali	Juan Perez	Activo	
Ana Lopez	9876543210	+57 310 987 6543	Bogotá	Maria Garcia	Activo	
Pedro Martinez	09876543210	+57 320 500 0000	Medellín	Juan Perez	Inactivo	

Si deseas registrar uno nuevo siempre lo único que debes hacer es agregar un nuevo cliente y te cargara esta pantalla para que cargues la información

Registrar Nuevo Cliente

Contacto y Soporte

Proyecto: Práctica Empresarial 2025

Repositorio: <https://github.com/jesebe4991/creditos-backend>

Issues: <https://github.com/jesebe4991/creditos-backend/issues>

Última actualización: 11 de noviembre de 2025

Versión del documento: 1.0.0