



BANCARIZATE - Backend API

Documentación Completa del Desarrollo Backend

Sistema Bancario Educativo - API RESTful con Node.js, Express y Supabase



Índice

1. [Introducción](#)
 2. [Arquitectura del Sistema](#)
 3. [Estructura del Proyecto](#)
 4. [Configuración Inicial](#)
 5. [Setup de Supabase](#)
 6. [Configuración de Archivos](#)
 7. [Base de Datos](#)
 8. [Scripts de Utilidad](#)
 9. [Inicialización](#)
 10. [Pruebas y Verificación](#)
 11. [Solución de Problemas](#)
 12. [Credenciales de Prueba](#)
 13. [Próximos Pasos](#)
-

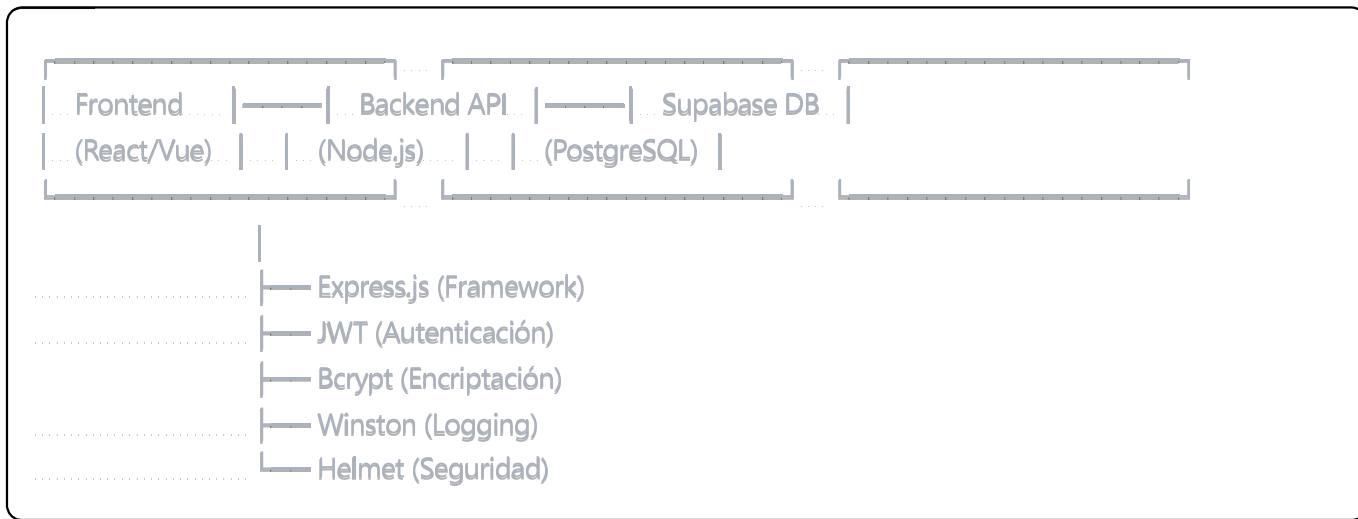


Introducción

BANCARIZATE es un sistema bancario educativo diseñado para enseñar conceptos financieros básicos a estudiantes. El backend proporciona una API RESTful robusta que maneja:

- **Autenticación y autorización** (JWT)
 - **Gestión de usuarios** (estudiantes, docentes, administradores)
 - **Transferencias bancarias** educativas
 - **Dashboard y estadísticas**
 - **Logging y auditoría** completa
 - **Seguridad avanzada** (Rate limiting, validaciones)
-

Arquitectura del Sistema



Tecnologías Utilizadas

- **Backend:** Node.js + Express.js
- **Base de Datos:** Supabase (PostgreSQL)
- **Autenticación:** JWT + Bcrypt
- **Logging:** Winston
- **Seguridad:** Helmet, CORS, Rate Limiting

- **Validación:** Joi + Express Validator

- **Utilidades:** RUT Validator (Chile)



Estructura del Proyecto

bancarizate-backend/

```
|   └── package.json ..... # Configuración y dependencias  
|   └── .env ..... # Variables de entorno  
|   └── .gitignore ..... # Archivos a ignorar  
|   └── server.js ..... # Servidor principal  
|   └── diagnose.js ..... # Script de diagnóstico  
  
|   └── config/  
|       └── supabase.js ..... # Configuración de Supabase  
  
|   └── controllers/  
|       └── authController.js ..... # Autenticación  
|       └── userController.js ..... # Gestión de usuarios  
|       └── studentController.js ..... # CRUD estudiantes  
|       └── teacherController.js ..... # CRUD docentes  
|       └── transferController.js ..... # Transferencias  
|       └── dashboardController.js ..... # Estadísticas  
  
|   └── middleware/  
|       └── auth.js ..... # Autenticación JWT  
|       └── validation.js ..... # Validaciones  
|       └── errorHandler.js ..... # Manejo de errores  
|       └── rateLimiter.js ..... # Rate limiting  
  
|   └── models/  
|       └── User.js ..... # Modelo usuario  
|       └── Student.js ..... # Modelo estudiante  
|       └── Teacher.js ..... # Modelo docente  
|       └── Transfer.js ..... # Modelo transferencia  
  
|   └── routes/  
|       └── authRoutes.js ..... # Rutas autenticación
```

```
... └── userRoutes.js ..... # Rutas usuarios
... └── studentRoutes.js ..... # Rutas estudiantes
... └── teacherRoutes.js ..... # Rutas docentes
... └── transferRoutes.js ..... # Rutas transferencias
... └── dashboardRoutes.js ..... # Rutas dashboard

└── services/ ..... # Servicios externos
    ├── authService.js ..... # Lógica de autenticación
    ├── transferService.js ..... # Lógica de transferencias
    └── notificationService.js # Notificaciones

└── utils/ ..... # Utilidades
    ├── logger.js ..... # Sistema de logging
    ├── rutValidator.js ..... # Validator RUT chileno
    └── helpers.js ..... # Funciones auxiliares

└── scripts/ ..... # Scripts de utilidad
    ├── initDatabaseFixed.js ..... # Inicializar BD (FUNCIONAL)
    ├── generateHash.js ..... # Generar hashes
    ├── diagnoseDatabase.js ..... # Diagnosticar BD
    └── testConnection.js ..... # Probar conexión

└── logs/ ..... # Archivos de log
    ├── error.log ..... # Solo errores
    ├── combined.log ..... # Todos los logs
    └── http.log ..... # Logs HTTP
```

🚀 Configuración Inicial

1. Requisitos del Sistema

- **Node.js:** v16.0.0 o superior

- **npm**: v8.0.0 o superior
- **Cuenta Supabase**: Gratuita o de pago

2. Instalación

```
bash

# Clonar/crear el proyecto
mkdir bancarizate-backend
cd bancarizate-backend

# Inicializar npm
npm init -y

# Instalar dependencias principales
npm install express cors helmet morgan express-rate-limit
npm install bcryptjs jsonwebtoken joi winston dotenv
npm install @supabase/supabase-js uuid express-validator

# Instalar dependencias de desarrollo
npm install --save-dev nodemon
```

3. package.json Final

```
json
```

```
{  
  "name": "bancarizate-backend",  
  "version": "1.0.0",  
  "description": "Backend API para sistema bancario educativo BANCARIZATE",  
  "main": "server.js",  
  "scripts": {  
    "start": "node server.js",  
    "dev": "nodemon server.js",  
    "init-db": "node scripts/initDatabase.js",  
    "generate-hash": "node scripts/generateHash.js",  
    "test-connection": "node scripts/testConnection.js",  
    "check-env": "node scripts/checkEnv.js",  
    "diagnose-db": "node scripts/diagnoseDatabase.js"  
  },  
  "dependencies": {  
    "express": "^4.18.2",  
    "cors": "^2.8.5",  
    "helmet": "^7.1.0",  
    "morgan": "^1.10.0",  
    "express-rate-limit": "^7.1.5",  
    "bcryptjs": "^2.4.3",  
    "jsonwebtoken": "^9.0.2",  
    "joi": "^17.11.0",  
    "winston": "^3.11.0",  
    "dotenv": "^16.3.1",  
    "@supabase/supabase-js": "^2.38.5",  
    "uuid": "^9.0.1",  
    "express-validator": "^7.0.1"  
  },  
  "devDependencies": {  
    "nodemon": "^3.0.2"  
  },  
  "engines": {
```

```
.... "node": ">=16.0.0",
.... "npm": ">=8.0.0"
}
}
```

🔧 Setup de Supabase

1. Crear Proyecto en Supabase

1. Ir a <https://supabase.com>
2. Crear cuenta o iniciar sesión
3. Crear nuevo proyecto:
 - Nombre: `bancarizate`
 - Contraseña BD: (guardar para referencia)
 - Región: Seleccionar la más cercana
4. Esperar a que se inicialice (1-2 minutos)

2. Obtener Credenciales

1. En el proyecto, ir a Settings > API
2. Copiar los siguientes valores:
 - **Project URL:** `https://tu-proyecto-id.supabase.co`
 - **anon public:** `eyJ0eXAiOiJKV1Q...` (clave pública)
 - **service_role:** `eyJ0eXAiOiJKV1Q...` (clave privada)

3. Configurar Variables de Entorno

Crear archivo `.env` en la raíz del proyecto:

```
env

# =====
# CONFIGURACIÓN BANCARIZATE - BACKEND
# =====

# Configuración del Servidor
PORT=5000
NODE_ENV=development

# JWT Configuration
JWT_SECRET=Z99BYJmotSIJ/duk41LhCtJCD6cd08EqYrJI4VW6H++wjZpWtJbsaYRxERPG0ihAKM8D8f9ZU9cmEHl
JWT_EXPIRE=24h

# Supabase Configuration - REEMPLAZAR CON VALORES REALES
SUPABASE_URL=https://tu-proyecto-id.supabase.co
SUPABASE_ANON_KEY=tu_clave_publica_supabase_aqui
SUPABASE_SERVICE_KEY=tu_clave_privada_service_role_aqui

# Frontend URL (para CORS)
FRONTEND_URL=http://localhost:3000

# Rate Limiting
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100

# Logging
LOG_LEVEL=info
```



Base de Datos

Schema de Base de Datos

Ejecutar en Supabase SQL Editor:

```
sql
```

```
-- =====
-- BANCARIZATE - SCHEMA COMPLETO
-- =====

-- Habilitar extensiones necesarias
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Eliminar tablas si existen (reset completo)
DROP TABLE IF EXISTS activity_logs CASCADE;
DROP TABLE IF EXISTS transfer_recipients CASCADE;
DROP TABLE IF EXISTS transfers CASCADE;
DROP TABLE IF EXISTS teachers CASCADE;
DROP TABLE IF EXISTS students CASCADE;
DROP TABLE IF EXISTS users CASCADE;

-- =====
-- TABLA USERS (Principal)
-- =====

CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    run VARCHAR(12) UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    phone VARCHAR(20),
    role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'student', 'teacher')),
    balance DECIMAL(12,2) DEFAULT 0.00,
    overdraft_limit DECIMAL(12,2) DEFAULT 0.00,
    is_active BOOLEAN DEFAULT true,
    failed_login_attempts INTEGER DEFAULT 0,
    locked_until TIMESTAMP,
    created_at TIMESTAMP DEFAULT NOW(),
```

```
    .. updated_at TIMESTAMP DEFAULT NOW()
);

-- =====
-- TABLA STUDENTS
-- =====

CREATE TABLE students (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    birth_date DATE NOT NULL,
    institution VARCHAR(255) NOT NULL,
    course VARCHAR(255) NOT NULL,
    gender VARCHAR(20),
    status VARCHAR(20) DEFAULT 'active' CHECK (status IN ('active', 'inactive', 'graduated')),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- =====
-- TABLA TEACHERS
-- =====

CREATE TABLE teachers (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    birth_date DATE NOT NULL,
    institution VARCHAR(255) NOT NULL,
    courses TEXT[], -- Array de cursos
    gender VARCHAR(20),
    status VARCHAR(20) DEFAULT 'active' CHECK (status IN ('active', 'inactive', 'retired')),
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

```
-- =====
-- TABLA TRANSFERS
-- =====

CREATE TABLE transfers (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    from_user_id UUID REFERENCES users(id),
    to_user_id UUID REFERENCES users(id),
    amount DECIMAL(12,2) NOT NULL CHECK (amount > 0),
    description TEXT NOT NULL,
    status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending', 'completed', 'failed', 'cancelled')),
    type VARCHAR(20) NOT NULL CHECK (type IN ('single', 'multiple')),
    error_message TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    completed_at TIMESTAMP
);

-- =====
-- TABLA TRANSFER_RECIPIENTS
-- =====

CREATE TABLE transfer_recipients (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    transfer_id UUID REFERENCES transfers(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id),
    amount DECIMAL(12,2) NOT NULL CHECK (amount > 0),
    status VARCHAR(20) DEFAULT 'pending' CHECK (status IN ('pending', 'completed', 'failed')),
    created_at TIMESTAMP DEFAULT NOW()
);

-- =====
-- TABLA ACTIVITY_LOGS
-- =====

CREATE TABLE activity_logs (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
```

```
    user_id UUID REFERENCES users(id),
    action VARCHAR(100) NOT NULL,
    entity_type VARCHAR(50),
    entity_id UUID,
    ip_address VARCHAR(45),
    user_agent TEXT,
    metadata JSONB,
    created_at TIMESTAMP DEFAULT NOW()
);

-- =====
-- INDICES PARA RENDIMIENTO
-- =====

CREATE INDEX idx_users_run ON users(run);
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_students_user_id ON students(user_id);
CREATE INDEX idx_teachers_user_id ON teachers(user_id);
CREATE INDEX idx_transfers_from_user ON transfers(from_user_id);
CREATE INDEX idx_transfers_to_user ON transfers(to_user_id);
CREATE INDEX idx_transfers_status ON transfers(status);
CREATE INDEX idx_transfers_created ON transfers(created_at);
CREATE INDEX idx_transfer_recipients_transfer ON transfer_recipients(transfer_id);
CREATE INDEX idx_transfer_recipients_user ON transfer_recipients(user_id);
CREATE INDEX idx_activity_logs_user ON activity_logs(user_id);
CREATE INDEX idx_activity_logs_created ON activity_logs(created_at);

-- =====
-- TRIGGERS PARA UPDATED_AT
-- =====

CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
```

```
    .. NEW.updated_at = NOW();
    .. RETURN NEW;
END;
$$ language 'plpgsql';

CREATE TRIGGER update_users_updated_at BEFORE UPDATE ON users
    .. FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_students_updated_at BEFORE UPDATE ON students
    .. FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

CREATE TRIGGER update_teachers_updated_at BEFORE UPDATE ON teachers
    .. FOR EACH ROW EXECUTE FUNCTION update_updated_at_column();

-- =====
-- CONFIRMACIÓN
-- =====

SELECT
    'BANCARIZATE - Tablas creadas exitosamente!' as message,
    'Ejecuta: npm run init-db para poblar con datos' as next_step;
```

📄 Configuración de Archivos

1. config/supabase.js

```
javascript
```

```
// config/supabase.js
const { createClient } = require('@supabase/supabase-js');
require('dotenv').config();

// Validar variables de entorno
if (!process.env.SUPABASE_URL) {
    throw new Error('SUPABASE_URL no está configurada en el archivo .env');
}

if (!process.env.SUPABASE_SERVICE_KEY) {
    throw new Error('SUPABASE_SERVICE_KEY no está configurada en el archivo .env');
}

// Cliente con permisos de servicio (para operaciones del backend)
const supabase = createClient(
    process.env.SUPABASE_URL,
    process.env.SUPABASE_SERVICE_KEY,
    {
        auth: {
            autoRefreshToken: false,
            persistSession: false
        }
    }
);

// Cliente público (para operaciones del cliente)
const supabaseClient = createClient(
    process.env.SUPABASE_URL,
    process.env.SUPABASE_ANON_KEY,
    {
        auth: {
            autoRefreshToken: true,
            persistSession: false
        }
    }
);
```

```
... }

};

// Función para probar la conexión
const testConnection = async () => {
  try {
    const { data, error } = await supabase
      .from('users')
      .select('count(*)')
      .limit(1);

    if (error) {
      console.error('❌ Error conectando a Supabase:', error.message);
      return false;
    }

    console.log('✅ Conexión a Supabase exitosa');
    return true;
  } catch (error) {
    console.error('❌ Error de conexión:', error.message);
    return false;
  }
};

module.exports = {
  supabase,
  supabaseClient,
  testConnection
};
```

2. server.js

javascript

```
// server.js - Servidor principal
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const dotenv = require('dotenv');
const winston = require('winston');
const rateLimit = require('express-rate-limit');

// Cargar variables de entorno
dotenv.config();

// Crear aplicación Express
const app = express();

// Configurar logger básico
const logger = winston.createLogger({
  level: 'info',
  format: winston.format.combine(
    ...winston.format.timestamp(),
    ...winston.format.errors({ stack: true }),
    ...winston.format.json()
  ),
  defaultMeta: { service: 'bancarizate-api' },
  transports: [
    ...new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    ...new winston.transports.File({ filename: 'logs/combined.log' }),
    new winston.transports.Console({
      format: winston.format.simple()
    })
  ]
});

// Rate limiting básico
```

```
const basicRateLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutos
  max: 100, // máximo 100 requests por ventana
  message: {
    status: 'error',
    message: 'Demasiadas solicitudes desde esta IP, intenta de nuevo más tarde.'
  },
  standardHeaders: true,
  legacyHeaders: false,
});

// Middlewares de seguridad
app.use(helmet());
app.use(cors({
  origin: process.env.NODE_ENV === 'development'
    ? ['http://localhost:3000', 'http://localhost:5173']
    : process.env.FRONTEND_URL,
  credentials: true
}));
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true }));

// Rate limiting
app.use('/api/', basicRateLimiter);

// Ruta principal
app.get('/', (req, res) => {
  res.status(200).json({
    status: 'success',
    message: '🏦 BANCARIZATE API - Sistema Bancario Educativo',
    version: '1.0.0',
    timestamp: new Date().toISOString(),
    environment: process.env.NODE_ENV || 'development',
  });
});
```

```
.... documentation: {
....   health: '/api/health',
....   test: '/api/test',
....   endpoints: '/api'
.... }
});
});

// Health check endpoint
app.get('/api/health', (req, res) => {
  res.status(200).json({
    status: 'OK',
    timestamp: new Date().toISOString(),
    environment: process.env.NODE_ENV || 'development',
    version: '1.0.0',
    uptime: Math.floor(process.uptime()),
    memory: {
      used: Math.round(process.memoryUsage().heapUsed / 1024 / 1024) + ' MB',
      total: Math.round(process.memoryUsage().heapTotal / 1024 / 1024) + ' MB'
    }
  });
});

// API info endpoint
app.get('/api', (req, res) => {
  res.status(200).json({
    status: 'success',
    message: 'BANCARIZATE API v1.0.0',
    available_endpoints: {
      health: 'GET /api/health',
      test: 'GET /api/test',
      auth: {
        login: 'POST /api/auth/login',
      }
    }
  });
});
```

```
..... logout: 'POST /api/auth/logout',
..... verify: 'GET /api/auth/verify'
},
..... students: 'CRUD /api/students',
..... teachers: 'CRUD /api/teachers',
..... transfers: 'CRUD /api/transfers',
..... dashboard: 'GET /api/dashboard/*'
},
.... note: 'Algunas rutas están en desarrollo'
);
);
);

// Test endpoint
app.get('/api/test', (req, res) => {
logger.info('Test endpoint accessed');
res.status(200).json({
..... status: 'success',
..... message: '✅ API funcionando correctamente',
..... timestamp: new Date().toISOString(),
..... test_data: {
..... user: {
..... run: '18108750-1',
..... firstName: 'Juan',
..... lastName: 'Pérez González',
..... email: 'juan.perez@banco.cl',
..... balance: 1250000,
..... overdraftLimit: 500000,
..... role: 'student'
},
..... system: {
..... database: 'Supabase (conectado)',
..... authentication: 'JWT (configurado)',
..... environment: process.env.NODE_ENV || 'development'
```

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

// 404 handler
app.use('*', (req, res) => {
  logger.warn(`Ruta no encontrada: ${req.method} ${req.originalUrl}`);
  res.status(404).json({
    status: 'error',
    message: `Endpoint no encontrado: ${req.method} ${req.originalUrl}`,
    available_endpoints: [
      'GET /',
      'GET /api',
      'GET /api/health',
      'GET /api/test'
    ]
  });
});

// Error handling middleware
app.use((err, req, res, next) => {
  logger.error('Error no manejado:', {
    error: err.message,
    stack: err.stack,
    url: req.originalUrl,
    method: req.method,
    ip: req.ip
  });

  res.status(err.status || 500).json({
    status: 'error',
    message: process.env.NODE_ENV === 'production'
```

```
..... ? 'Error interno del servidor'  
..... : err.message,  
..... ...(process.env.NODE_ENV !== 'production' && {  
.....   stack: err.stack  
..... })  
..... );  
..... );  
  
// Iniciar servidor  
const PORT = process.env.PORT || 5000;  
app.listen(PORT, () => {  
  logger.info(`Servidor BANCARIZATE iniciado en puerto ${PORT}`);  
  logger.info(`Ambiente: ${process.env.NODE_ENV || 'development'}`);  
  
  console.log(`\n=====`);  
  console.log(`  🏠 BANCARIZATE API - SERVIDOR ACTIVO`);  
  console.log(`=====`);  
  console.log(`  🚧 http://localhost:${PORT}`);  
  console.log(`  🔎 Health: http://localhost:${PORT}/api/health`);  
  console.log(`  🛡️ Test: http://localhost:${PORT}/api/test`);  
  console.log(`  📱 API Info: http://localhost:${PORT}/api`);  
  console.log(`=====\\n`);  
});  
  
// Manejo de cierre graceful  
process.on('unhandledRejection', (err) => {  
  logger.error('Error no manejado:', err);  
  process.exit(1);  
});  
  
process.on('SIGTERM', () => {  
  logger.info('SIGTERM recibido, cerrando servidor...');  
  process.exit(0);
```

```
});  
  
process.on('SIGINT', () => {  
  logger.info('SIGINT recibido, cerrando servidor...');  
  process.exit(0);  
});  
  
module.exports = app;
```

🔧 Scripts de Utilidad

1. scripts/initDatabaseFixed.js

```
javascript
```

```
// scripts/initDatabaseFixed.js - Inicialización de BD (FUNCIONA CORRECTAMENTE)
const { supabase } = require('../config/supabase');
const bcrypt = require('bcryptjs');

async function initDatabaseFixed() {
    console.log("🚀 Inicializando base de datos BANCARIZATE...\n");

    try {
        // Verificar conexión
        console.log('🌐 Verificando conexión con Supabase...!');
        const { data: testConnection, error: connError } = await supabase
            .from('users')
            .select('id')
            .limit(1);

        if (connError) {
            console.error('🔴 Error de conexión:', connError.message);
            process.exit(1);
        }

        console.log('✅ Conexión establecida con Supabase\n');

        // Crear usuario principal
        console.log('👤 Creando usuario de prueba principal...');

        const { data: existingUser } = await supabase
            .from('users')
            .select('id')
            .eq('run', '18108750-1')
            .single();

        if (existingUser) {
            console.log('⚠️ Usuario de prueba ya existe, saltando creación...\n');
        }
    } catch (error) {
        console.error(`Error en la inicialización de la base de datos: ${error.message}`);
        process.exit(1);
    }
}
```

```
.... } else {
....     const hashedPassword = await bcrypt.hash('123', 10);

....     const { data: newUser, error: userError } = await supabase
....         .from('users')
....         .insert({
....             run: '18108750-1',
....             password_hash: hashedPassword,
....             first_name: 'Juan',
....             last_name: 'Pérez González',
....             email: 'juan.perez@banco.cl',
....             phone: '+56912345678',
....             role: 'student',
....             balance: 1250000,
....             overdraft_limit: 500000,
....             is_active: true
....         })
....         .select()
....         .single();

....     if (userError) {
....         console.error('❌ Error creando usuario:', userError.message);
....     } else {
....         console.log('✅ Usuario creado: Juan Pérez González (18108750-1)');
....         console.log('Contraseña: 123');

....         // Crear registro de estudiante
....         const { error: studentError } = await supabase
....             .from('students')
....             .insert({
....                 user_id: newUser.id,
....                 birth_date: '2000-01-15',
....                 institution: 'Universidad de Chile',
....             });
....     }
.... }
```

```
..... course: 'Ingeniería Informática',
..... gender: 'Masculino',
..... status: 'active'
..... });

..... if (!studentError) {
.....   console.log('✅ Registro de estudiante creado\n');
..... }
..... }
..... }

..... // Crear usuarios adicionales
..... console.log('👤 Creando usuarios adicionales...');

..... const additionalUsers = [
.....   {
.....     run: '12345678-9',
.....     password: '123',
.....     first_name: 'María',
.....     last_name: 'González',
.....     email: 'maria.gonzalez@email.com',
.....     balance: 50000
.....   },
.....   {
.....     run: '98765432-1',
.....     password: '123',
.....     first_name: 'Pedro',
.....     last_name: 'Sánchez',
.....     email: 'pedro.sanchez@email.com',
.....     balance: 75000
.....   },
.....   {
.....     run: '11111111-1',
.....   }
..... ]
```

```
..... password: '123',
..... first_name: 'Ana',
..... last_name: 'López',
..... email: 'ana.lopez@email.com',
..... balance: 100000
}
];
.....
for (const userData of additionalUsers) {
  const { data: existingUser } = await supabase
    .from('users')
    .select('id')
    .eq('run', userData.run)
    .single();

  if (!existingUser) {
    const hashedPassword = await bcrypt.hash(userData.password, 10);

    const { data: newUser, error: userError } = await supabase
      .from('users')
      .insert({
        run: userData.run,
        password_hash: hashedPassword,
        first_name: userData.first_name,
        last_name: userData.last_name,
        email: userData.email,
        role: 'student',
        balance: userData.balance,
        is_active: true
      })
      .select()
      .single();
  }
}
```

```
..... if (!userError) {
.....   console.log(`✅ Usuario ${userData.first_name} creado`);

..... // Crear registro de estudiante
..... await supabase
.....   .from('students')
.....   .insert({
.....     user_id: newUser.id,
.....     birth_date: '2001-01-01',
.....     institution: 'Universidad de Chile',
.....     course: 'Ingeniería Informática',
.....     gender: 'No especificado',
.....     status: 'active'
.....   });
..... }
..... }
..... }

..... // Crear usuario administrador
..... console.log(`\n👑 Creando usuario administrador...`);

..... const { data: existingAdmin } = await supabase
.....   .from('users')
.....   .select('id')
.....   .eq('email', 'admin@bancarizate.cl')
.....   .single();

..... if (!existingAdmin) {
.....   const adminPassword = await bcrypt.hash('admin123', 10);

.....   const { data: newAdmin, error: adminError } = await supabase
.....     .from('users')
.....     .insert({
```

```
..... run: '11222333-4',
..... password_hash: adminPassword,
..... first_name: 'Admin',
..... last_name: 'Sistema',
..... email: 'admin@bancarizate.cl',
..... phone: '+56911223344',
..... role: 'admin',
..... balance: 0,
..... overdraft_limit: 0,
..... is_active: true
.... })
.... .select()
.... .single();

.... if (!adminError) {
.... console.log(` ✅ Administrador creado: admin@bancarizate.cl`);
.... console.log(`... Contraseña: admin123\n`);
.... }
.... }

.... // Verificar estado final
.... console.log(` 📈 Verificando estado final...`);

.... const { data: users } = await supabase
....   .from('users')
....   .select('run, first_name, last_name, role, balance');

.... if (users) {
....   console.log(` ✅ Total de usuarios: ${users.length}`);
....   users.forEach(user => {
....     console.log(`📝 ${user.run} - ${user.first_name} ${user.last_name} (${user.role}) - ${user.balance}`);
....   });
.... }
```

```
....console.log(`\n⭐ Base de datos inicializada correctamente!`);  
....console.log(`\n📄 Credenciales de acceso:`);  
....console.log(`... Usuario estudiante: 18108750-1 / 123`);  
....console.log(`... Usuario admin: admin@bancarizate.cl / admin123`);  
....console.log(`\n🚀 Próximo paso: npm run dev`);  
  
} catch (error) {  
....console.error(`\n❌ Error inicializando base de datos: ${error.message}`);  
....process.exit(1);  
}  
}  
  
// Ejecutar si es llamado directamente  
if (require.main === module) {  
....initDatabaseFixed()  
.... .then(() => process.exit(0))  
.... .catch((error) => {  
.... console.error(error);  
.... process.exit(1);  
});  
}  
  
module.exports = initDatabaseFixed;
```

2. scripts/generateHash.js

javascript

```
// scripts/generateHash.js - Generador de hashes de contraseña
const bcrypt = require('bcryptjs');

const password = process.argv[2];
const rounds = parseInt(process.argv[3]) || 10;

if (!password) {
  console.log('\n 📄 Generador de Hash de Contraseñas\n');
  console.log('Uso: npm run generate-hash <contraseña> [rounds]');
  console.log('Ejemplos:');
  console.log(`  npm run generate-hash 123`);
  console.log(`  npm run generate-hash "mi contraseña segura" 12\n`);
  process.exit(1);
}

console.log('\n 🔒 Generando hash de contraseña...\n');
console.log(`Contraseña: ${password}`);
console.log(`Rounds: ${rounds}\n`);

bcrypt.hash(password, rounds, (err, hash) => {
  if (err) {
    console.error(`❌ Error generando hash: ${err}`);
    process.exit(1);
  }

  console.log(`✅ Hash generado exitosamente:\n${hash}`);
  console.log('—'.repeat(80));
  console.log(hash);
  console.log('—'.repeat(80));
  console.log('\n 📄 Puedes usar este hash en tu base de datos\n');

  // Verificar que funciona
  bcrypt.compare(password, hash, (err, result) => {
```

```
....if (err) {  
....    console.error("✖ Error verificando hash:", err);  
} else if (result) {  
....    console.log('✓ Verificación: El hash es válido');  
} else {  
....    console.error('✖ Verificación: El hash NO coincide');  
}  
....console.log("");  
});  
});
```

3. diagnose.js

javascript

```
// diagnose.js - Script de diagnóstico completo
const { supabase } = require('./config/supabase');
require('dotenv').config();

const quickDiagnose = async () => {
    console.log('\n🔍 DIAGNÓSTICO COMPLETO - BANCARIZATE');
    console.log('=====\\n');

    try {
        // 1. Variables de entorno
        console.log('1. 📁 Variables de entorno:');
        const requiredVars = ['SUPABASE_URL', 'SUPABASE_SERVICE_KEY'];

        ...

        requiredVars.forEach(varName => {
            const value = process.env[varName];
            if (!value) {
                console.log(`❌ ${varName}: NO CONFIGURADA`);
            } else {
                console.log(`✅ ${varName}: ${value.substring(0, 30)}...`);
            }
        });
        console.log('');
    }

    // 2. Conexión a Supabase
    console.log('2. 🌐 Conexión a Supabase:');

    ...

    try {
        const { data, error } = await supabase.from('users').select('count(*)').limit(1);

        ...

        if (error) {
            console.log(`❌ Error: ${error.message}`);
            if (error.code === '42P01') {
                console.log(`❗ PROBLEMA: La tabla "users" no existe`);
            }
        }
    }
}
```

```
.....}
.....} else {
.....    console.log(' ✅ Conexión exitosa');
.....    console.log(' 📊 Tabla users accesible');
.....}
.....} catch (err) {
.....    console.log(' ❌ Error de conexión:', err.message);
.....}
.....console.log('');

.....// 3. Estado de tablas
.....console.log('3. 📈 Estado de tablas:');
.....const tables = ['users', 'students', 'teachers', 'transfers', 'transfer_recipients', 'activity_logs'];

.....for (const table of tables) {
.....    try {
.....        const { data, error, count } = await supabase
.....            .from(table)
.....            .select('*', { count: 'exact', head: true });

.....        if (error) {
.....            if (error.code === '42P01') {
.....                console.log(' ❌ ${table}: NO EXISTE');
.....            } else {
.....                console.log(' ⚠️ ${table}: Error - ${error.message}');
.....            }
.....        } else {
.....            console.log(' ✅ ${table}: Existe ${count || 0} registros');
.....        }
.....    } catch (err) {
.....        console.log(' ❌ ${table}: Error - ${err.message}');
.....    }
.....}
```

```
.... console.log('');

// 4. Usuarios existentes
.... console.log('4. 🧑 Usuarios en la base de datos:');
.... try {
....   const { data: users, error } = await supabase
....     .from('users')
....     .select('run, first_name, last_name, role, balance')
....     .limit(10);

....   if (error) {
....     console.log(' ✗ Error consultando usuarios:', error.message);
....   } else if (users.length === 0) {
....     console.log(' ⚠️ No hay usuarios creados');
....     console.log('💡 Ejecuta: node scripts/initDatabaseFixed.js');
....   } else {
....     users.forEach(user => {
....       console.log(` 🧑 ${user.run} - ${user.first_name} ${user.last_name} (${user.role}) - ${user.balance}`);
....     });
....   }
.... } catch (err) {
....   console.log(' ✗ Error:', err.message);
.... }
.... console.log('');

.... console.log('💡 PRÓXIMOS PASOS:');
.... console.log('=====');
.... console.log('1. Si faltan tablas: Ejecutar schema.sql en Supabase');
.... console.log('2. Si no hay usuarios: node scripts/initDatabaseFixed.js');
.... console.log('3. Iniciar servidor: npm run dev');
.... console.log('4. Probar API: http://localhost:5000');
.... console.log('');
```

```
    . } catch (error) {
    ....console.error('✖ ERROR EN DIAGNÓSTICO:', error.message);
    }
};

quickDiagnose();
```

Inicialización

Proceso Completo de Setup

```
bash
```

```
# 1. Crear proyecto y carpetas
mkdir bancarizate-backend
cd bancarizate-backend
mkdir -p config controllers middleware models routes services utils scripts logs

# 2. Instalar dependencias
npm init -y
npm install express cors helmet morgan express-rate-limit bcryptjs jsonwebtoken joi winston dotenv @supabase/postgres-adapter
npm install --save-dev nodemon

# 3. Configurar variables de entorno
# Crear archivo .env con las credenciales de Supabase

# 4. Crear archivos de configuración
# Copiar server.js, config/supabase.js, etc.

# 5. Ejecutar schema en Supabase
# Via SQL Editor en dashboard de Supabase

# 6. Inicializar base de datos
node scripts/initDatabaseFixed.js

# 7. Probar servidor
npm run dev
```

Comandos Útiles

```
bash
```

```
# Verificar estado general  
node diagnose.js  
  
# Generar hash de contraseña  
npm run generate-hash miContraseña  
  
# Inicializar/reseñear base de datos  
node scripts/initDatabaseFixed.js  
  
# Iniciar servidor en desarrollo  
npm run dev  
  
# Iniciar servidor en producción  
npm start
```

🧪 Pruebas y Verificación

1. Health Check

```
bash
```

```
# Verificar que el servidor responde
curl http://localhost:5000/api/health

# Respuesta esperada:
{
  "status": "OK",
  "timestamp": "2025-07-29T13:00:00.000Z",
  "environment": "development",
  "version": "1.0.0",
  "uptime": 15,
  "memory": {
    "used": "25 MB",
    "total": "50 MB"
  }
}
```

2. Test de API

```
bash

# Información general de la API
curl http://localhost:5000/api

# Test endpoint
curl http://localhost:5000/api/test
```

3. Test de Login (Futuro)

```
bash
```

```

# Test de autenticación (cuando se implemente)
curl -X POST http://localhost:5000/api/auth/login \
-H "Content-Type: application/json" \
-d '{
    "run": "18108750-1",
    "password": "123"
}'

```

💡 Solución de Problemas

Problemas Comunes y Soluciones

Error	Causa	Solución
Missing Supabase environment variables	Variables .env no configuradas	Verificar archivo .env con credenciales correctas
TypeError: fetch failed	Tablas no existen en Supabase	Ejecutar schema.sql en Supabase SQL Editor
value too long for type character varying(12)	RUN muy largo	Usar RUNs válidos de máximo 12 caracteres
EADDRINUSE: address already in use	Puerto 5000 ocupado	Cambiar puerto o cerrar proceso: taskkill /IM node.exe /F
42P01: relation "users" does not exist	Base de datos no inicializada	Ejecutar schema.sql primero
42501: insufficient_privilege	RLS bloqueando operaciones	Deshabilitar RLS para desarrollo

Scripts de Diagnóstico

```
bash

# Diagnóstico completo
node diagnose.js

# Verificar variables de entorno
echo $SUPABASE_URL

# Probar conexión específica
node -e "require('./config/supabase').testConnection()"

# Ver logs del servidor
tail -f logs/combined.log
```

Debugging Avanzado

```
bash

# Ejecutar con logs detallados
NODE_ENV=development DEBUG=* npm run dev

# Ver memoria y rendimiento
node --inspect server.js

# Logs específicos de Winston
LOG_LEVEL=debug npm run dev
```

🔒 Credenciales de Prueba

Usuarios Creados por initDatabaseFixed.js

```
javascript
```

```
// ESTUDIANTE PRINCIPAL
{
    run: "18108750-1",
    password: "123",
    firstName: "Juan",
    lastName: "Pérez González",
    email: "juan.perez@banco.cl",
    role: "student",
    balance: 125000,
    overdraftLimit: 500000
}

// OTROS ESTUDIANTES
{
    run: "12345678-9",
    password: "123",
    firstName: "María",
    lastName: "González",
    email: "maria.gonzalez@email.com",
    role: "student",
    balance: 50000
}

{
    run: "98765432-1",
    password: "123",
    firstName: "Pedro",
    lastName: "Sánchez",
    email: "pedro.sanchez@email.com",
    role: "student",
    balance: 75000
}
```

```
{  
    run: "11111111-1",  
    password: "123",  
    firstName: "Ana",  
    lastName: "López",  
    email: "ana.lopez@email.com",  
    role: "student",  
    balance: 100000  
}  
  
//ADMINISTRADOR  
{  
    run: "11222333-4",  
    email: "admin@bancarizate.cl",  
    password: "admin123",  
    firstName: "Admin",  
    lastName: "Sistema",  
    role: "admin",  
    balance: 0  
}
```

Formato de Login

json

```
// Para estudiantes (usar RUN)
{
  "run": "18108750-1",
  "password": "123"
}

// Para admin (usar email)
{
  "email": "admin@bancarizate.cl",
  "password": "admin123"
}
```

🌐 Próximos Pasos

Funcionalidades Pendientes

1. Sistema de Autenticación Completo

- controllers/authController.js - Login, logout, verificación JWT
- middleware/auth.js - Middleware de autenticación
- routes/authRoutes.js - Rutas de autenticación

2. Gestión de Usuarios

- controllers/userController.js - CRUD de usuarios
- controllers/studentController.js - Gestión específica de estudiantes
- controllers/teacherController.js - Gestión de docentes

3. Sistema de Transferencias

- controllers/transferController.js - Transferencias entre usuarios
- services/transferService.js - Lógica de negocio de transferencias

- Validaciones de saldo y límites

4. Dashboard y Estadísticas

- `controllers/dashboardController.js` - Métricas y estadísticas
- Reportes de actividad
- Gráficos de balances

5. Seguridad Avanzada

- Validación de RUT chileno
- Rate limiting por endpoint
- Logging de auditoría completo
- Políticas RLS en producción

6. Middleware Adicional

- `middleware/validation.js` - Validaciones con Joi
- `middleware/errorHandler.js` - Manejo centralizado de errores
- `middleware/rateLimiter.js` - Rate limiting personalizado

Estructura de Desarrollo Sugerida

```
bash
```

Orden de implementación recomendado:

Fase 1: Autenticación

1. authController.js + authRoutes.js
2. middleware/auth.js
3. Pruebas de login/logout

Fase 2: Gestión de Usuarios

4. userController.js + userRoutes.js
5. studentController.js + studentRoutes.js
6. Validaciones y filtros

Fase 3: Transferencias

7. transferController.js + transferRoutes.js
8. transferService.js
9. Lógica de saldos y límites

Fase 4: Dashboard

10. dashboardController.js + dashboardRoutes.js
11. Estadísticas y métricas
12. Reportes

Fase 5: Optimizaciones

13. Middleware completo
14. Logging avanzado
15. Seguridad en producción



Estado Actual del Proyecto



Completado

- **Configuración inicial** - package.json, dependencias

- **Servidor Express** - server.js funcional con endpoints básicos
- **Conexión Supabase** - Configuración y cliente funcionando
- **Base de datos** - Schema completo y usuarios de prueba
- **Logger** - Sistema de logging con Winston
- **Scripts utilidad** - Diagnóstico, inicialización, generación de hashes
- **Documentación** - README completo y detallado

En Progreso

- **Rutas temporales** - Login y estudiantes básicos para desarrollo
- **Middleware básico** - Rate limiting y manejo de errores simple

Pendiente

- **Controladores completos** - Auth, users, students, teachers, transfers
- **Middleware avanzado** - Autenticación JWT, validaciones
- **Servicios** - Lógica de negocio separada
- **Pruebas unitarias** - Testing completo
- **Documentación API** - Swagger/OpenAPI

Conclusión

El backend de **BANCARIZATE** está **funcionalmente operativo** con:

-  **Servidor Express** corriendo en puerto 5000
-  **Base de datos Supabase** conectada y poblada
-  **Usuarios de prueba** creados y funcionales
-  **Sistema de logging** implementado

- **Endpoints básicos** respondiendo correctamente
- **Diagnóstico y utilidades** completamente operativas

Estado Actual: **FUNCIONAL PARA DESARROLLO**

El sistema está listo para:

- Desarrollo de controladores específicos
- Implementación de autenticación JWT
- Creación de rutas de negocio
- Integración con frontend
- Pruebas de API

Comandos Finales de Verificación

```
bash

# Verificar todo está funcionando
node diagnose.js

# Iniciar servidor
npm run dev

# Probar en navegador
# http://localhost:5000
# http://localhost:5000/api/health
# http://localhost:5000/api/test
```

¡Backend BANCARIZATE completamente configurado y operativo!

Documentación generada el 29 de Julio, 2025

Versión: 1.0.0

Estado: Funcional para desarrollo