MANUAL TÉCNICO

TODO SOBRE EL DESARROLLO DE LA APLICACIÓN

Reacti

Sistema de Generación de Exámenes con IA

DESARROLLADA POR:

CASTRO AGUILAR EDER JOEL
MARTÍNEZ PEÑAFIEL ADRIÁN FERNANDO
VALENCIA OROPEZA ANGEL YAHIR

Versión 1.0

19 de junio de 2025

Desarrollado con React y Node.js
Integración con Supabase y Google Gemini AI

Índice general

Capítulo 1

Introducción

1.1. Tecnologías

El Sistema de Generación de Exámenes con IA (Reacti) es una plataforma web moderna diseñada para automatizar la creación de exámenes personalizados utilizando inteligencia artificial. El sistema permite a los usuarios subir documentos, generar preguntas automáticamente, realizar exámenes con temporizador y recibir retroalimentación detallada.

1.1.1. Stack Tecnológico Detallado

Frontend

- **React 19.0.0**: Framework principal para la interfaz de usuario
- **TypeScript**: Lenguaje de programacion con tipado estático
- **Vite**: Herramienta de build rápida y servidor de desarrollo
- **Tailwind CSS**: Framework de CSS para diseño responsive
- **React Router**: Librería de enrutamiento para SPA
- **Motion**: Librería de animaciones para React
- **SweetAlert2**: Librería para alertas y modales elegantes
- **React Markdown**: Renderizador de contenido Markdown
- **KaTeX**: Renderizador de fórmulas matemáticas

Backend

- **Node.js**: Runtime de JavaScript del lado del servidor
- **Express.js**: Framework web minimalista y flexible
- **Google Generative AI**: Integración con Gemini AI
- **Multer**: Middleware para manejo de archivos multipart

- **CORS**: Middleware para habilitar cross-origin requests
- **dotenv**: Gestión de variables de entorno

Base de Datos y Servicios

- **Supabase**: Plataforma Backend-as-a-Service (PostgreSQL)
- **Google Gemini AI**: Servicio de inteligencia artificial
- **Hostinger**: Plataforma de hosting para el frontend

Capítulo 2

Arquitectura del Sistema

2.1. Patrón Arquitectónico

El sistema Reacti sigue una arquitectura de tres capas bien definidas, separando las responsabilidades entre presentación, lógica de negocio y acceso a datos.

2.1.1. Capas de la Arquitectura

Capa de Presentación (Frontend)

- **Componentes React**: Elementos de interfaz reutilizables
- **Páginas**: Vistas principales de la aplicación
- **Context API**: Gestión de estado global
- **Hooks personalizados**: Lógica reutilizable

Capa de Lógica de Negocio (Backend)

- **API REST**: Endpoints para operaciones CRUD
- **Middleware**: Autenticación y validación
- **Servicios**: Integración con APIs externas
- **Controladores**: Manejo de la lógica de aplicación

Capa de Datos

- **Supabase**: Base de datos PostgreSQL
- **Autenticación**: Sistema de usuarios integrado
- **Storage**: Almacenamiento de archivos
- **Real-time**: Actualizaciones en tiempo real

2.1.2. Estructura de Directorios

Frontend

```
frontend/
+-- src/
    +-- API/
        +-- Gemini.tsx          # Configuracion API Gemini
    +-- components/
        +-- Main/               # Componentes principales
        +-- Examen/             # Componentes de examenes
        +-- Navbar.tsx          # Barra de navegacion
    +-- context/
        +-- AuthContext.tsx     # Context de autenticacion
    +-- pages/
        +-- Login.tsx           # Pagina de login
        +-- Examenes.tsx        # Pagina de examenes
        +-- Dashboard.tsx       # Panel principal
    +-- routers/
        +-- routes.tsx          # Configuracion de rutas
    +-- utils/                  # Utilidades
    +-- styles/                 # Estilos globales
```

Backend

```
backend/
+-- src/
    +-- index.js               # Servidor principal
    +-- reqAuthMiddleware.js   # Middleware de autenticacion
    +-- reqSupabase.js         # Integracion con Supabase
    +-- local.js               # Configuracion local
    +-- analize.js             # Analisis de documentos
    +-- routes/                # Rutas de la API
    +-- services/              # Servicios externos
    +-- utils/                 # Utilidades del servidor
+-- uploads/                   # Archivos temporales
+-- package.json               # Dependencias
```

2.2. Gestión de Estado

2.2.1. Context API de React

El sistema utiliza React Context API para manejar el estado global de la aplicación, especialmente para:

- **Autenticación**: Estado del usuario y sesión
- **Datos de examenes**: Información de examenes activos
- **Configuración**: Preferencias del usuario
- **Notificaciones**: Sistema de mensajes y alertas

2.2.2. Providers Principales

AuthProvider

```
// Manejo de autenticacion y sesion de usuario
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  // Metodos de login, logout, registro
  const login = async (email, password) => { ... };
  const logout = async () => { ... };
  const register = async (userData) => { ... };

  return (
    <AuthContext.Provider value={{
      user, login, logout, register, loading
    }}>
      {children}
    </AuthContext.Provider>
  );
};
```

2.3. Componentes Principales

2.3.1. Integración con Gemini AI

El sistema integra Google Gemini AI para la generación automática de contenido educativo:

Configuración del Modelo

```
import { GoogleGenerativeAI } from "@google/generative-ai";

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);

// Configuracion para examenes faciles
const model_flash = genAI.getGenerativeModel({
  model: "gemini-2.0-flash"
});

// Configuracion para examenes complejos
const model_pro = genAI.getGenerativeModel({
  model: "gemini-2.5-pro-exp-03-25"
});
```

Funcionalidades de IA

- **Generación de preguntas**: A partir de documentos subidos
- **Análisis de contenido**: Extracción de conceptos clave
- **Retroalimentación personalizada**: Explicaciones detalladas
- **Adaptación de dificultad**: Ajuste automático del nivel

2.3.2. Sistema de Autenticación

Flujo de Autenticación

1. Usuario ingresa credenciales
2. Validación con Supabase Auth
3. Generación de JWT token
4. Almacenamiento seguro en localStorage
5. Protección de rutas privadas

Middleware de Seguridad

```
// Middleware para verificar autenticacion
export const getUserFromRequest = async (req) => {
  const token = req.headers.authorization?.split(' ')[1];

  if (!token) {
    throw new Error('Token no proporcionado');
  }

  const { data: user, error } = await supabase.auth.getUser(token);

  if (error || !user) {
    throw new Error('Token invalido');
  }

  return user;
};
```

Capítulo 3

Pantallas y Funcionalidades

3.1. Dashboard Principal

El dashboard es la pantalla principal después del login, proporcionando una vista general del sistema:

3.1.1. Componentes del Dashboard

- **Panel de estadísticas**: Resumen de examenes realizados
- **Examenes recientes**: Lista de examenes completados
- **Accesos rápidos**: Botones para funciones principales
- **Notificaciones**: Mensajes y actualizaciones del sistema

3.1.2. Métricas Mostradas

- Total de examenes realizados
- Promedio de calificaciones
- Tiempo promedio por examen
- Progreso de aprendizaje

3.2. Sistema de Examenes

3.2.1. Categorías Disponibles

El sistema permite generar examenes en diferentes modalidades:

Por Tipo de Contenido

- **Documentos PDF**: Análisis de archivos subidos
- **Texto libre**: Generación a partir de prompts
- **Historial**: Basado en examenes anteriores

- **Temas específicos**: Materias predefinidas

Por Nivel de Dificultad

- **Fácil**: Preguntas básicas y conceptos fundamentales
- **Medio**: Aplicación de conocimientos
- **Difícil**: Análisis crítico y síntesis
- **Mixto**: Combinación de todos los niveles

3.2.2. Estructura de Examenes

Configuración de Examen

```
const examenConfig = {
  titulo: "Examen de Programacion",
  descripcion: "Conceptos basicos de JavaScript",
  numeroPreguntas: 10,
  tiempoLimite: 1800, // 30 minutos en segundos
  dificultad: "medio",
  tipoRespuesta: "multiple_choice",
  puntajeTotal: 100
};
```

Formato de Preguntas

```
const pregunta = {
  id: 1,
  pregunta: "Cual es la diferencia entre let y var?",
  opciones: [
    "No hay diferencia",
    "let tiene scope de bloque, var de funcion",
    "var es mas moderno que let",
    "let no se puede redeclarar"
  ],
  correcta: 1,
  explicacion: "let tiene scope de bloque mientras que var...",
  puntaje: 10
};
```

3.3. Sistema de Retroalimentación

3.3.1. Funcionalidades Avanzadas

Análisis de Respuestas

- **Evaluación automática**: Calificación instantánea
- **Explicaciones detalladas**: Para cada respuesta incorrecta
- **Sugerencias de mejora**: Areas de estudio recomendadas

- **Recursos adicionales**: Enlaces y materiales de apoyo

Reportes Personalizados

- Análisis de fortalezas y debilidades
- Comparación con examenes anteriores
- Tendencias de aprendizaje
- Recomendaciones personalizadas

3.3.2. Generación de Feedback con IA

```
const generateFeedback = async (examenId) => {
  const prompt = `
    Analiza los resultados del examen y genera:
    1. Resumen de desempeno
    2. Areas de mejora
    3. Fortalezas identificadas
    4. Plan de estudio sugerido
  `;

  const result = await model.generateContent(prompt);
  return result.response.text();
};
```

Capítulo 4

Configuración y Desarrollo

4.1. Requisitos del Sistema

4.1.1. Entorno de Desarrollo

Requisitos Mínimos

- **Node.js**: Versión 18 o superior
- **npm**: Versión 8 o superior (incluido con Node.js)
- **Git**: Para control de versiones
- **Editor de código**: VSCode recomendado
- **Navegador**: Chrome, Firefox o Safari actualizado

Herramientas Adicionales

- **Postman**: Para pruebas de API
- **Chrome DevTools**: Para debugging del frontend
- **Supabase CLI**: Para gestión de base de datos

4.1.2. Variables de Entorno

Frontend (.env)

```
# Configuracion de Supabase
VITE_SUPABASE_URL=https://tu-proyecto.supabase.co
VITE_SUPABASE_ANON_KEY=tu_clave_anonima

# URL del backend
VITE_BACKEND_URL=http://localhost:3001

# Configuracion de desarrollo
VITE_APP_ENV=development
VITE_DEBUG_MODE=true
```

Backend (.env)

```
# API Key de Google Gemini
GEMINI_API_KEY=tu_api_key_de_gemini

# Configuracion de Supabase
SUPABASE_URL=https://tu-proyecto.supabase.co
SUPABASE_SERVICE_ROLE_KEY=tu_service_role_key

# Configuracion del servidor
PORT=3001
NODE_ENV=development

# Configuracion de CORS
CORS_ORIGIN=http://localhost:5173
```

4.2. Dependencias del Proyecto

4.2.1. Dependencias de Producción

Frontend

```
{
  "dependencies": {
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "react-router-dom": "^6.8.0",
    "@supabase/supabase-js": "^2.38.0",
    "framer-motion": "^10.16.4",
    "sweetalert2": "^11.7.32",
    "react-markdown": "^8.0.7",
    "katex": "^0.16.8",
    "react-katex": "^3.0.1"
  }
}
```

Backend

```
{
  "dependencies": {
    "express": "^4.18.2",
    "@google/generative-ai": "^0.1.3",
    "@supabase/supabase-js": "^2.38.0",
    "multer": "^1.4.5-lts.1",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1"
  }
}
```

4.2.2. Dependencias de Desarrollo

```
{
  "devDependencies": {
    "@types/react": "^18.2.37",
    "@types/react-dom": "^18.2.15",
    "@vitejs/plugin-react": "^4.1.0",
    "vite": "^4.5.0",
    "tailwindcss": "^3.3.0",
    "autoprefixer": "^10.4.16",
    "postcss": "^8.4.31",
    "eslint": "^8.53.0",
    "prettier": "^3.0.3"
  }
}
```

4.3. Proceso de Build

4.3.1. Comandos de Desarrollo

Instalación

```
# Clonar el repositorio
git clone https://github.com/tu-usuario/reacti.git
cd reacti

# Instalar dependencias del frontend
cd frontend
npm install

# Instalar dependencias del backend
cd ../backend
npm install
```

Ejecución en Desarrollo

```
# Terminal 1: Ejecutar backend
cd backend
npm start

# Terminal 2: Ejecutar frontend
cd frontend
npm run dev
```

4.3.2. Build de Producción

Frontend

```
# Generar build optimizado
cd frontend
npm run build

# El directorio 'dist' contiene los archivos optimizados
ls dist/
```

Backend

```
# Configurar variables de entorno de produccion
export NODE_ENV=production
export PORT=3001

# Ejecutar en produccion
node src/index.js
```

Capítulo 5

Backend y Base de Datos

5.1. Supabase como Backend

5.1.1. Componentes de Supabase

Supabase proporciona una solución completa de Backend-as-a-Service:

Base de Datos PostgreSQL

- **Esquema relacional**: Tablas optimizadas para el dominio
- **Índices**: Para consultas eficientes
- **Triggers**: Para lógica automática
- **Functions**: Procedimientos almacenados

Autenticación

- **JWT Tokens**: Autenticación segura
- **Row Level Security**: Políticas de acceso granular
- **Providers sociales**: Login con Google, GitHub, etc.
- **Gestión de sesiones**: Manejo automático de tokens

API REST Automática

- **CRUD operations**: Operaciones automáticas
- **Filtros avanzados**: Consultas complejas
- **Paginación**: Para grandes conjuntos de datos
- **Ordenamiento**: Múltiples criterios

5.1.2. Estructura de la Base de Datos

Tabla: users

```
CREATE TABLE public.users (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    full_name VARCHAR(255),
    avatar_url TEXT,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);
```

Tabla: exams

```
CREATE TABLE public.exams (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    user_id UUID REFERENCES public.users(id) ON DELETE CASCADE,
    titulo VARCHAR(500) NOT NULL,
    descripcion TEXT,
    preguntas JSONB NOT NULL,
    dificultad VARCHAR(50) DEFAULT 'medium',
    numero_preguntas INTEGER DEFAULT 5,
    tiempo_limite INTEGER DEFAULT 1800,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Indice para busquedas por usuario
CREATE INDEX idx_exams_user_id ON public.exams(user_id);
-- Indice para busquedas por fecha
CREATE INDEX idx_exams_created_at ON public.exams(created_at);
```

Tabla: exam_results

```
CREATE TABLE public.exam_results (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
    exam_id UUID REFERENCES public.exams(id) ON DELETE CASCADE,
    user_id UUID REFERENCES public.users(id) ON DELETE CASCADE,
    respuestas JSONB NOT NULL,
    puntaje INTEGER NOT NULL DEFAULT 0,
    puntaje_maximo INTEGER NOT NULL DEFAULT 100,
    tiempo_empleado INTEGER, -- en segundos
    completed_at TIMESTAMP DEFAULT NOW(),

    UNIQUE(exam_id, user_id) -- Un usuario solo puede hacer un examen
   una vez
);
```

Tabla: feedback

```
CREATE TABLE public.feedback (
    id UUID DEFAULT gen_random_uuid() PRIMARY KEY,
```

```
    exam_result_id UUID REFERENCES public.exam_results(id) ON DELETE
   CASCADE,
    user_id UUID REFERENCES public.users(id) ON DELETE CASCADE,
    feedback_data JSONB NOT NULL,
    ai_analysis TEXT,
    recommendations TEXT[],
    created_at TIMESTAMP DEFAULT NOW()
);
```

5.1.3. Políticas de Seguridad

Row Level Security (RLS)

```
-- Habilitar RLS en todas las tablas
ALTER TABLE public.users ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.exams ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.exam_results ENABLE ROW LEVEL SECURITY;
ALTER TABLE public.feedback ENABLE ROW LEVEL SECURITY;

-- Politica para que usuarios solo vean sus propios datos
CREATE POLICY "Users can view own data" ON public.users
    FOR ALL USING (auth.uid() = id);

CREATE POLICY "Users can manage own exams" ON public.exams
    FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can view own results" ON public.exam_results
    FOR ALL USING (auth.uid() = user_id);

CREATE POLICY "Users can view own feedback" ON public.feedback
    FOR ALL USING (auth.uid() = user_id);
```

5.2. Integración con Google Gemini

5.2.1. Configuración del Modelo

Inicialización del Cliente

```
import { GoogleGenerativeAI } from "@google/generative-ai";

class GeminiService {
    constructor() {
        this.genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
        this.modelFlash = this.genAI.getGenerativeModel({
            model: "gemini-2.0-flash"
        });
        this.modelPro = this.genAI.getGenerativeModel({
            model: "gemini-2.5-pro-exp-03-25"
        });
    }

    async generateExam(content, difficulty = 'medium') {
        const model = difficulty === 'hard' ? this.modelPro : this.
   modelFlash;
```

```
        const prompt = this.buildExamPrompt(content, difficulty);

        try {
            const result = await model.generateContent(prompt);
            return this.parseExamResponse(result.response.text());
        } catch (error) {
            throw new Error(`Error generando examen: ${error.message}`);
        }
    }
}
```

5.2.2. Funcionalidades de IA

Generación de Preguntas

```
buildExamPrompt(content, difficulty) {
    return `
        Analiza el siguiente contenido y crea un examen con estas especificaciones:

        Contenido: ${content}
        Dificultad: ${difficulty}

        Genera un JSON con la siguiente estructura:
        {
            "titulo": "Titulo del examen",
            "descripcion": "Descripcion breve",
            "numero_preguntas": 5,
            "dificultad": "${difficulty}",
            "preguntas": [
                {
                    "id": 1,
                    "pregunta": "Texto de la pregunta",
                    "opciones": ["opcion1", "opcion2", "opcion3", "opcion4"],
                    "correcta": 0,
                    "explicacion": "Explicacion de la respuesta correcta"
                }
            ]
        }

        Asegurate de que las preguntas sean relevantes y del nivel solicitado.
    `;
}
```

Análisis de Resultados

```
async generateFeedback(examResult) {
    const prompt = `
        Analiza los resultados de este examen y proporciona feedback personalizado:

        Puntaje obtenido: ${examResult.puntaje}/${examResult.puntaje_maximo}
```

```
        Tiempo empleado: ${examResult.tiempo_empleado} segundos
        Respuestas: ${JSON.stringify(examResult.respuestas)}

        Genera un analisis que incluya:
        1. Resumen del desempeno
        2. Fortalezas identificadas
        3. Areas de mejora
        4. Recomendaciones especificas de estudio
        5. Proximos pasos sugeridos

        El feedback debe ser constructivo y motivador.
    `;

    const result = await this.modelPro.generateContent(prompt);
    return result.response.text();
}
```

Capítulo 6

API Endpoints

6.1. Endpoints de Autenticación

6.1.1. Registro de Usuario

```
POST /api/auth/register
Content-Type: application/json

{
    "email": "usuario@ejemplo.com",
    "password": "password123",
    "full_name": "Nombre Completo"
}

Response:
{
    "success": true,
    "user": {
        "id": "uuid",
        "email": "usuario@ejemplo.com",
        "full_name": "Nombre Completo"
    },
    "token": "jwt_token"
}
```

6.1.2. Inicio de Sesión

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

{
    "email": "usuario@ejemplo.com",
    "password": "password123"
}

Response:
{
    "success": true,
    "user": { ... },
    "token": "jwt_token"
```

```
}
```

6.2. Endpoints de Examenes

6.2.1. Generar Examen desde Archivo

```
POST /api/upload_files
Authorization: Bearer jwt_token
Content-Type: multipart/form-data

Files: archivo.pdf
Body:
{
    "prompt": "Genera preguntas sobre este documento",
    "tiempo_limite_segundos": 1800,
    "numero_preguntas": 10,
    "dificultad": "medium"
}

Response:
{
    "success": true,
    "exam": {
        "id": "uuid",
        "titulo": "Examen generado",
        "preguntas": [...],
        "tiempo_limite": 1800
    }
}
```

6.2.2. Generar Examen desde Texto

```
POST /api/generate-content
Authorization: Bearer jwt_token
Content-Type: application/json

{
    "prompt": "Crea un examen sobre JavaScript",
    "dificultad": "medium",
    "numero_preguntas": 8,
    "tiempo_limite_segundos": 1200
}

Response:
{
    "success": true,
    "exam": { ... }
}
```

6.2.3. Obtener Examenes del Usuario

```
GET /api/exams
Authorization: Bearer jwt_token

Response:
{
    "success": true,
    "exams": [
        {
            "id": "uuid",
            "titulo": "Examen de JavaScript",
            "created_at": "2025-06-19T10:00:00Z",
            "numero_preguntas": 10,
            "dificultad": "medium"
        }
    ]
}
```

6.3. Endpoints de Resultados

6.3.1. Enviar Respuestas de Examen

```
POST /api/exam-results
Authorization: Bearer jwt_token
Content-Type: application/json

{
    "exam_id": "uuid",
    "respuestas": [
        {
            "pregunta_id": 1,
            "respuesta_seleccionada": 2
        }
    ],
    "tiempo_empleado": 900
}

Response:
{
    "success": true,
    "result": {
        "id": "uuid",
        "puntaje": 85,
        "puntaje_maximo": 100,
        "respuestas_correctas": 8,
        "total_preguntas": 10
    }
}
```

6.3.2. Generar Retroalimentación

```
POST /api/generate-feedback
Authorization: Bearer jwt_token
Content-Type: application/json

```

```
{
    "exam_result_id": "uuid"
}

Response:
{
    "success": true,
    "feedback": {
        "analysis": "Analisis detallado del desempeno...",
        "strengths": ["Conceptos basicos", "Sintaxis"],
        "improvements": ["Programacion asincrona"],
        "recommendations": ["Practicar con async/await"]
    }
}
```

Capítulo 7

Seguridad y Optimización

7.1. Medidas de Seguridad

7.1.1. Autenticación y Autorización

- **JWT Tokens**: Tokens seguros con expiración
- **Validación de entrada**: Sanitización de datos
- **Rate limiting**: Límites de peticiones por IP
- **CORS configurado**: Solo orígenes permitidos

7.1.2. Protección de Datos

- **Encriptación en tránsito**: HTTPS obligatorio
- **Variables de entorno**: Credenciales seguras
- **Row Level Security**: Políticas de acceso granular
- **Limpieza de archivos**: Eliminación automática de temporales

7.2. Optimización de Rendimiento

7.2.1. Frontend

- **Lazy loading**: Carga bajo demanda de componentes
- **Memoización**: Evitar re-renders innecesarios
- **Bundle splitting**: Código dividido por rutas
- **Compresión**: Assets optimizados para producción

7.2.2. Backend

- **Conexión pool**: Reutilización de conexiones DB
- **Caché**: Resultados frecuentes en memoria
- **Compresión gzip**: Respuestas comprimidas
- **Índices de BD**: Consultas optimizadas

Capítulo 8

Despliegue y Mantenimiento

8.1. Proceso de Despliegue

8.1.1. Frontend (Hostinger)

```
# 1. Generar build de produccion
npm run build

# 2. Configurar variables de entorno de produccion
VITE_SUPABASE_URL=https://prod.supabase.co
VITE_BACKEND_URL=https://api.reacti.com

# 3. Subir archivos a hosting
# Copiar contenido de 'dist' al directorio public_html
```

8.1.2. Backend (Servidor VPS)

```
# 1. Clonar repositorio en servidor
git clone https://github.com/usuario/reacti.git
cd reacti/backend

# 2. Instalar dependencias
npm install --production

# 3. Configurar variables de entorno
export NODE_ENV=production
export GEMINI_API_KEY=tu_api_key
export SUPABASE_URL=https://prod.supabase.co

# 4. Ejecutar con PM2
pm2 start src/index.js --name "reacti-api"
pm2 save
pm2 startup
```

8.2. Monitoreo y Mantenimiento

8.2.1. Métricas Importantes

- **Tiempo de respuesta**: APIs ¡500ms
- **Tasa de errores**: ¡1 % de requests fallidos
- **Uso de CPU**: ¡70 % promedio
- **Memoria**: ¡80 % del total disponible
- **Uso de API Gemini**: Dentro de las cuotas

8.2.2. Logs y Debugging

```
# Ver logs del backend
pm2 logs reacti-api

# Logs de errores especificos
pm2 logs reacti-api --err

# Monitoreo en tiempo real
pm2 monit
```

8.2.3. Backup y Recuperación

- **Base de datos**: Backup diario automático en Supabase
- **Código**: Repositorio Git como backup
- **Configuración**: Variables de entorno documentadas
- **Assets**: Copia de seguridad en almacenamiento externo

Capítulo 9

Conclusión

9.1. Resumen del Sistema

El Sistema de Generación de Examenes con IA (Reacti) representa una solución moderna y escalable para la automatización de evaluaciones educativas. Utilizando tecnologías de vanguardia como React, Node.js, Supabase y Google Gemini AI, el sistema proporciona una experiencia completa desde la generación automática de preguntas hasta la retroalimentación personalizada.

9.1.1. Características Principales

- **Generación automática**: Preguntas basadas en documentos o texto libre
- **Múltiples dificultades**: Adaptación automática del nivel
- **Retroalimentación IA**: Análisis detallado del desempeno
- **Interfaz moderna**: Diseño responsive y accesible
- **Seguridad robusta**: Protección de datos y autenticación segura

9.1.2. Beneficios Técnicos

- **Arquitectura escalable**: Fácil mantenimiento y extensión
- **Tecnologías modernas**: Stack actualizado y soporte a largo plazo
- **Base de datos optimizada**: Consultas eficientes y seguras
- **API bien diseñada**: Endpoints claros y documentados

9.2. Trabajo Futuro

9.2.1. Mejoras Planificadas

- **Soporte multiidioma**: Internacionalización completa
- **Análisis avanzado**: Métricas de aprendizaje más sofisticadas

- **Integración LMS**: Compatibilidad con sistemas existentes
- **App movil**: Versión nativa para iOS y Android

9.2.2. Escalabilidad

- **Microservicios**: Migración gradual de monolito
- **Cache distribuido**: Redis para mejor rendimiento
- **CDN**: Distribución global de contenido
- **Auto-scaling**: Escalado automático basado en demanda

Versión del Manual: 1.0
Fecha: 19 de junio de 2025
Desarrollado por: Castro Aguilar Eder Joel, Martínez Peñafiel Adrián Fernando, Valencia Oropeza Angel Yahir