

MANUAL TÉCNICO

TODO SOBRE EL DESARROLLO DE LA APLICACIÓN

ExamGenAI

Sistema de Generación de Exámenes con IA

DESARROLLADA POR:

HERNÁNDEZ TELLEZ HÉCTOR FIDEL
RIVERO FLORES VICTOR GABRIEL
Versión 1.0

19 de junio de 2025

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

Índice general

1. Introducción	4
1.1. Tecnologías	4
1.1.1. Stack Tecnológico Detallado	4
2. Arquitectura del Sistema	6
2.1. Patrón Arquitectónico	6
2.1.1. Capas de la Arquitectura	6
2.1.2. Estructura de Directorios	7
2.2. Gestión de Estado	8
2.2.1. Context API de React	8
2.2.2. Providers Principales	8
2.3. Componentes Principales	9
2.3.1. Integración con Gemini AI	9
2.3.2. Sistema de Autenticación	9
3. Pantallas y Funcionalidades	11
3.1. Dashboard Principal	11
3.1.1. Componentes del Dashboard	11
3.1.2. Métricas Mostradas	11
3.2. Sistema de Exámenes	12
3.2.1. Categorías Disponibles	12
3.2.2. Estructura de Exámenes	12
3.3. Sistema de Retroalimentación	13
3.3.1. Funcionalidades Avanzadas	13
3.3.2. Generación de Feedback con IA	13
3.3.3. Gestión de Exámenes	14
4. Configuración y Desarrollo	19
4.1. Requisitos del Sistema	19
4.1.1. Entorno de Desarrollo	19
4.1.2. Variables de Entorno	19
4.2. Dependencias del Proyecto	20
4.2.1. Dependencias de Producción	20
4.2.2. Dependencias de Desarrollo	20
4.3. Proceso de Build	21
4.3.1. Comandos de Desarrollo	21
4.3.2. Build de Producción	21

5. Backend y Base de Datos	23
5.1. Supabase como Backend	23
5.1.1. Componentes de Supabase	23
5.1.2. Estructura de la Base de Datos	24
5.1.3. Políticas de Seguridad	25
5.2. Integración con Google Gemini	25
5.2.1. Configuración del Modelo	25
5.2.2. Funcionalidades de IA	26
6. API Endpoints y Comunicación	28
6.1. Endpoints de Autenticación	28
6.1.1. Registro de Usuario	28
6.1.2. Inicio de Sesión	28
6.2. Endpoints de Exámenes	29
6.2.1. Generar Examen desde Archivo	29
6.2.2. Generar Examen desde Texto	29
6.2.3. Obtener Exámenes del Usuario	29
6.3. Endpoints de Resultados	30
6.3.1. Enviar Respuestas de Examen	30
6.3.2. Generar Retroalimentación	30
7. Costos y Viabilidad Económica	32
7.1. Análisis Detallado de Costos	32
7.1.1. Estructura de Costos	32
7.1.2. Ventajas del Modelo de Costos	33
7.1.3. Comparación con Alternativas	33
7.2. Modelo de Negocio	33
7.2.1. Opciones de Monetización	33
7.2.2. Proyección de Ingresos	33
8. Seguridad y Optimización	34
8.1. Medidas de Seguridad	34
8.1.1. Autenticación y Autorización	34
8.1.2. Protección de Datos	34
8.2. Optimización de Rendimiento	34
8.2.1. Frontend	34
8.2.2. Backend	35
9. Despliegue, Monitoreo y Mantenimiento	36
9.1. Proceso de Despliegue	36
9.1.1. Frontend (Hostinger)	36
9.1.2. Backend (Servidor VPS)	36
10. Exposición Móvil con Flutter WebView	37
10.1. Estrategia y Justificación	37
10.2. Implementación Técnica	37
10.2.1. Manejo Avanzado: Autenticación OAuth con Deep Links	38
10.3. Ventajas y Limitaciones de la Estrategia	40

11. Conclusión	41
11.1. Resumen del Sistema	41
11.1.1. Características Principales	41
11.1.2. Beneficios Técnicos	41
11.2. Trabajo Futuro	41
11.2.1. Escalabilidad	42

Capítulo 1

Introducción

1.1. Tecnologías

El Sistema de Generación de Exámenes con IA (ExamGenAI) 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 programación 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

Cliente Móvil (Wrapper)

- **Flutter:** Framework para el desarrollo de aplicaciones móviles multiplataforma.
- **webview_flutter:** Paquete para integrar una vista web (WebView) dentro de la aplicación Flutter, permitiendo mostrar la aplicación web de React como si fuera una aplicación nativa.

Capítulo 2

Arquitectura del Sistema

2.1. Patrón Arquitectónico

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

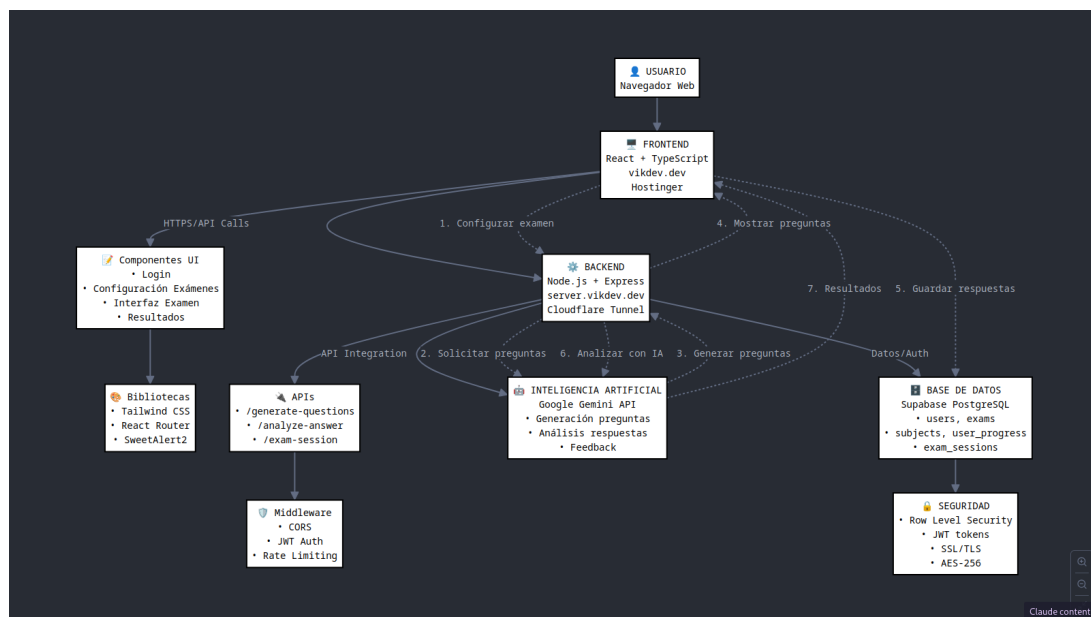


Figura 2.1: Diagrama de Arquitectura del Sistema ExamGenAI

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

Cliente Móvil (Wrapper con Flutter)

Adicionalmente, se ha desarrollado un *wrapper* o contenedor móvil utilizando Flutter. Esta capa no contiene lógica de negocio, sino que su única función es encapsular la aplicación web de React (Capa de Presentación) dentro de una WebView. Esto permite distribuir la aplicación en tiendas de aplicaciones móviles y ofrecer una experiencia de usuario más integrada en dispositivos móviles, aprovechando el 100 % del código y funcionalidades ya desarrolladas para la web.

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

```
1 frontend/
2 +-- src/
3     +-- API/
4         +-- Gemini.tsx           # Configuración API Gemini
5     +-- components/
6         +-- Main/               # Componentes principales
7         +-- Examen/             # Componentes de exámenes
8         +-- Navbar.tsx         # Barra de navegación
9     +-- context/
10        +-- AuthContext.tsx     # Context de autenticación
11    +-- pages/
12        +-- Login.tsx           # Página de login
13        +-- Exámenes.tsx       # Página de exámenes
14        +-- Dashboard.tsx      # Panel principal
15    +-- routers/
16        +-- routes.tsx         # Configuración de rutas
17    +-- utils/                  # Utilidades
18    +-- styles/                 # Estilos globales
```


Backend

```
1 backend/
2 +-- src/
3     +-- index.js           # Servidor principal
4     +-- reqAuthMiddleware.js # Middleware de autenticacion
5     +-- reqSupabase.js     # Integracion con Supabase
6     +-- local.js           # Configuracion local
7     +-- analyze.js         # Analisis de documentos
8     +-- routes/            # Rutas de la API
9     +-- services/          # Servicios externos
10    +-- utils/              # Utilidades del servidor
11 +-- uploads/              # Archivos temporales
12 +-- 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 exámenes:** Información de exámenes activos
- **Configuración:** Preferencias del usuario
- **Notificaciones:** Sistema de mensajes y alertas

2.2.2. Providers Principales

AuthProvider

```
1 // Manejo de autenticacion y sesion de usuario
2 export const AuthProvider = ({ children }) => {
3   const [user, setUser] = useState(null);
4   const [loading, setLoading] = useState(true);
5
6   // Metodos de login, logout, registro
7   const login = async (email, password) => { ... };
8   const logout = async () => { ... };
9   const register = async (userData) => { ... };
10
11   return (
12     <AuthContext.Provider value={{
13       user, login, logout, register, loading
14     }}>
15       {children}
16     </AuthContext.Provider>
17   );
18 };
```

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

```
1 import { GoogleGenerativeAI } from "@google/generative-ai";
2
3 const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
4
5 // Configuración para exámenes fáciles
6 const model_flash = genAI.getGenerativeModel({
7   model: "gemini-2.0-flash"
8 });
9
10 // Configuración para exámenes complejos
11 const model_pro = genAI.getGenerativeModel({
12   model: "gemini-2.5-pro-exp-03-25"
13 });
```

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

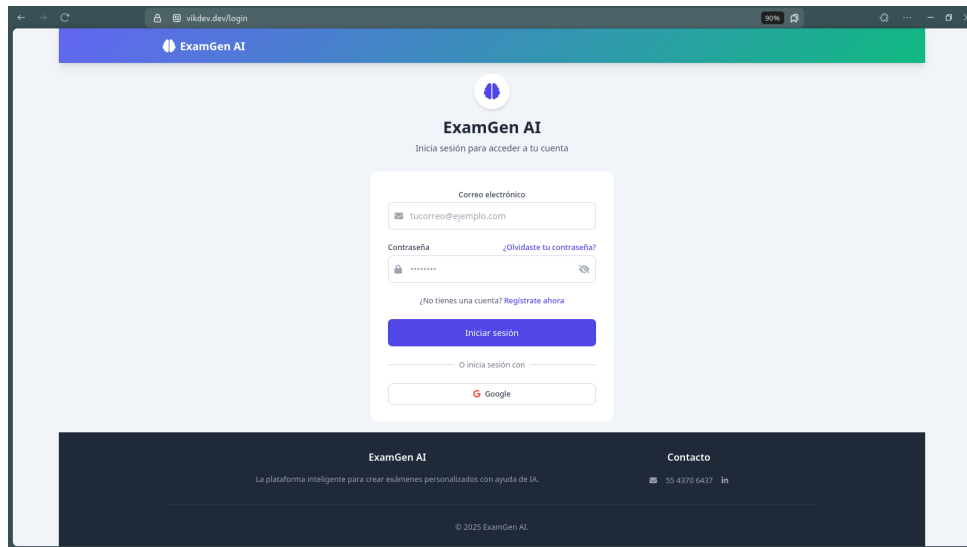


Figura 2.2: Pantalla de Login del Sistema ExamGenAI

Middleware de Seguridad

```
1 // Middleware para verificar autenticacion
2 export const getUserFromRequest = async (req) => {
3   const token = req.headers.authorization?.split(' ')[1];
4
5   if (!token) {
6     throw new Error('Token no proporcionado');
7   }
8
9   const { data: user, error } = await supabase.auth.getUser(token);
10
11   if (error || !user) {
12     throw new Error('Token invalido');
13   }
14
15   return user;
16 };
```

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 exámenes realizados
- **Exámenes recientes:** Lista de exámenes completados
- **Accesos rápidos:** Botones para funciones principales
- **Notificaciones:** Mensajes y actualizaciones del sistema

3.1.2. Métricas Mostradas

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

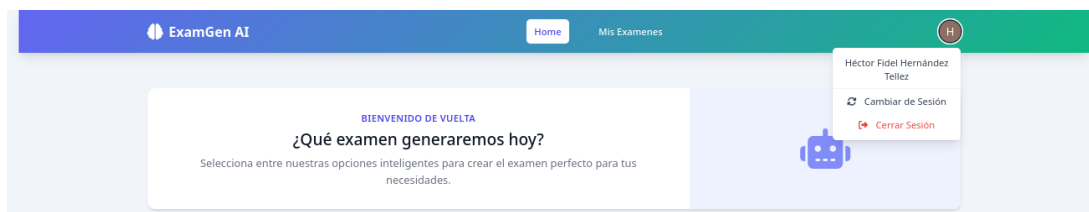


Figura 3.1: Vista del Dashboard con Menú de Usuario

3.2. Sistema de Exámenes

3.2.1. Categorías Disponibles

El sistema permite generar exámenes en diferentes modalidades:

Por Tipo de Contenido

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

Configuración de Examen

```
1 const examenConfig = {  
2   titulo: "Examen de Programacion",  
3   descripcion: "Conceptos basicos de JavaScript",  
4   numeroPreguntas: 10,  
5   tiempoLimite: 1800, // 30 minutos en segundos  
6   dificultad: "medio",  
7   tipoRespuesta: "multiple_choice",  
8   puntajeTotal: 100  
9 };
```

Formato de Preguntas

```
1 const pregunta = {  
2   id: 1,  
3   pregunta: "Cual es la diferencia entre let y var?",  
4   opciones: [  
5     "No hay diferencia",  
6     "let tiene scope de bloque, var de funcion",  
7     "var es mas moderno que let",  
8     "let no se puede redeclarar"  
9   ],  
10  correcta: 1,  
11  explicacion: "let tiene scope de bloque mientras que var...",  
12  puntaje: 10  
13 };
```

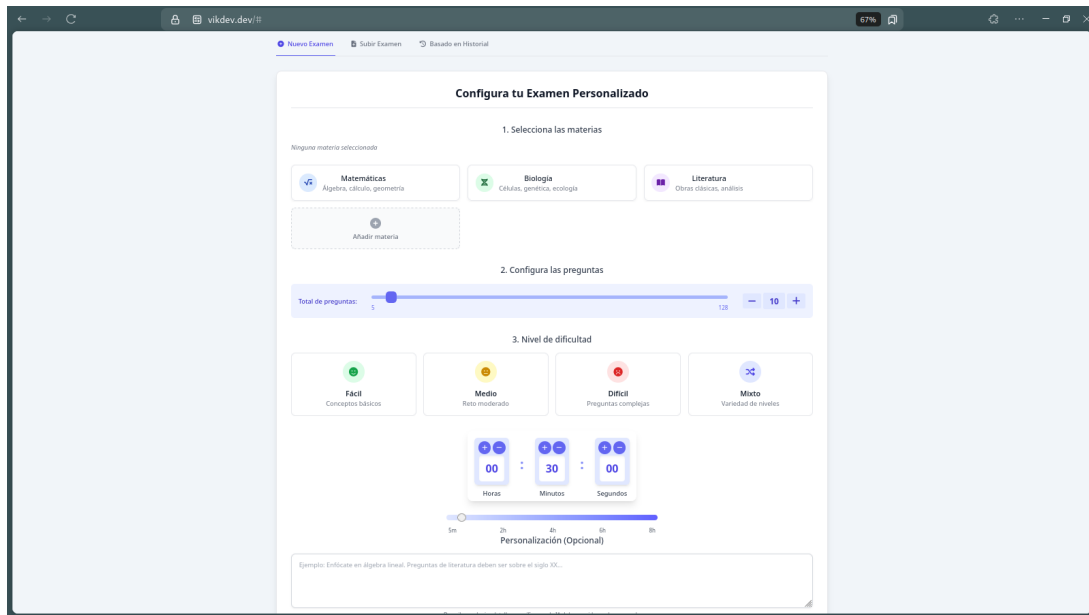


Figura 3.2: Configuración Avanzada de Exámenes Personalizados

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:** Áreas de estudio recomendadas
- **Recursos adicionales:** Enlaces y materiales de apoyo

Reportes Personalizados

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

3.3.2. Generación de Feedback con IA

```
1 const generateFeedback = async (examenId) => {  
2   const prompt = '  
3     Analiza los resultados del examen y genera:  
4     1. Resumen de desempeño  
5     2. Áreas de mejora  
6     3. Fortalezas identificadas
```

Genera Exámenes desde Contenido

Arrastra y suelta tus archivos aquí
o
Seleccionar archivos

o

Pega tu texto directamente

Pega aquí el contenido...

03 : 00 : 00
Horas Minutos Segundos

5m 2h 4h 6h 8h
Personalización (Opcional)

Ejemplo: Enfócate en álgebra lineal. Preguntas de literatura deben ser sobre el siglo XX...

Describe cualquier detalle específico que la IA deba considerar al generar el examen.

Generar Examen

Figura 3.3: Generación de Exámenes desde Contenido

```
7     4. Plan de estudio sugerido
8     ' ;
9
10    const result = await model.generateContent(prompt);
11    return result.response.text();
12  };
```

3.3.3. Gestión de Exámenes

Lista de Exámenes Recientes

El sistema mantiene un historial completo de todos los exámenes realizados por el usuario:

Interfaz de Examen en Ejecución

Durante la realización del examen, la interfaz proporciona herramientas avanzadas de navegación:

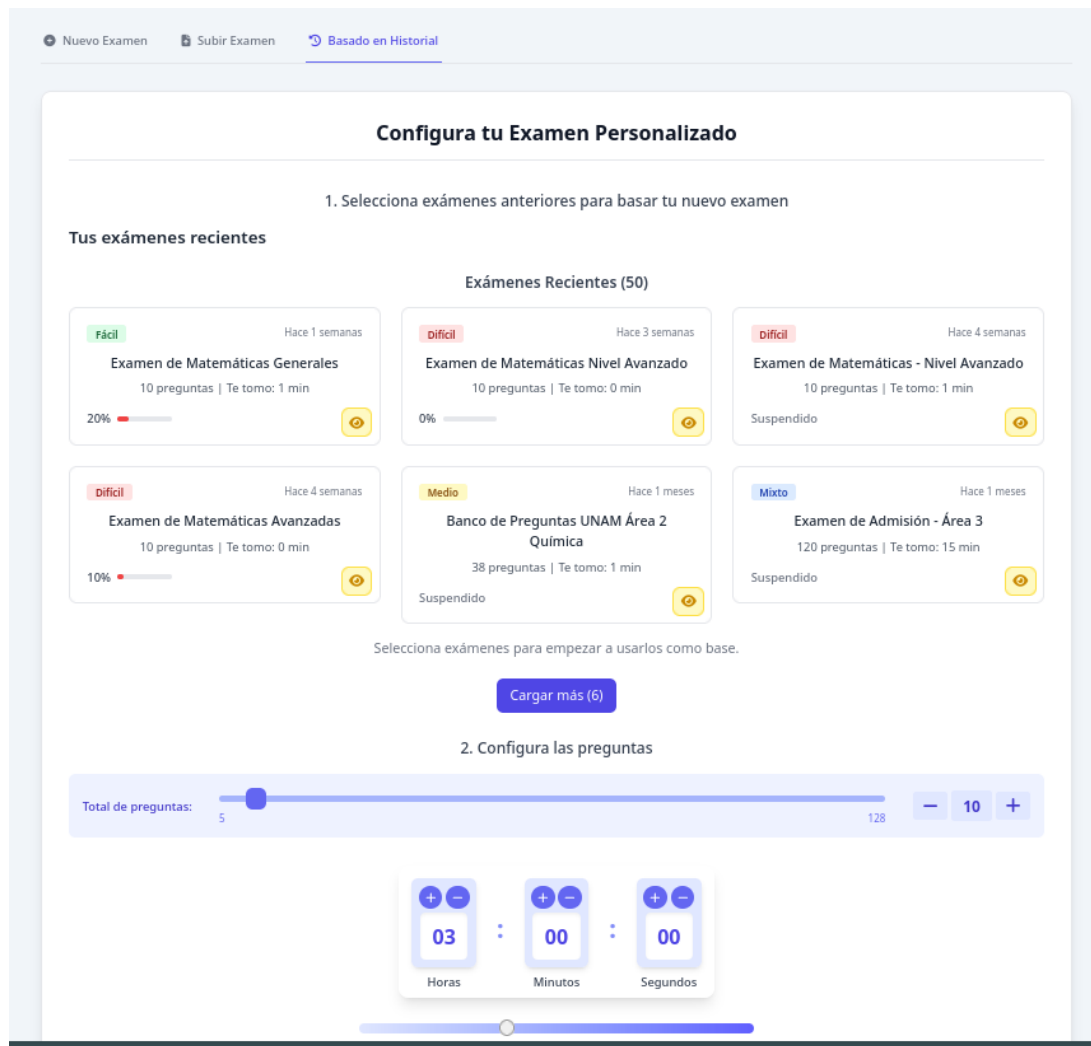


Figura 3.4: Configuración de Exámenes Basados en Historial

Progreso y Seguimiento

El sistema incluye una vista completa del progreso durante el examen:

Finalización del Examen

Al completar el examen, el usuario recibe opciones para revisar y obtener retroalimentación:

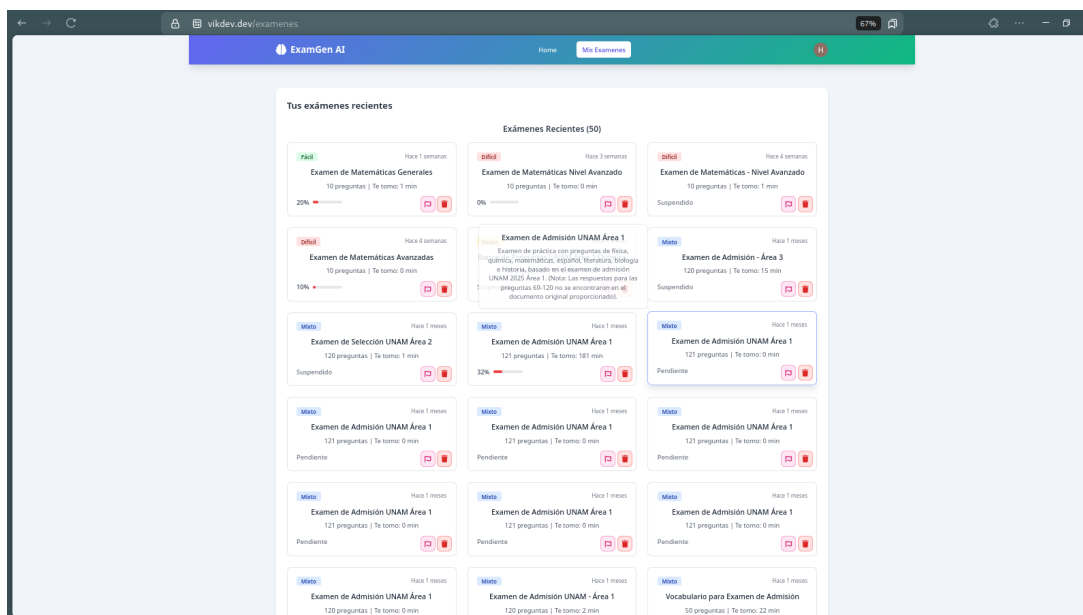


Figura 3.5: Lista de Exámenes Recientes del Usuario

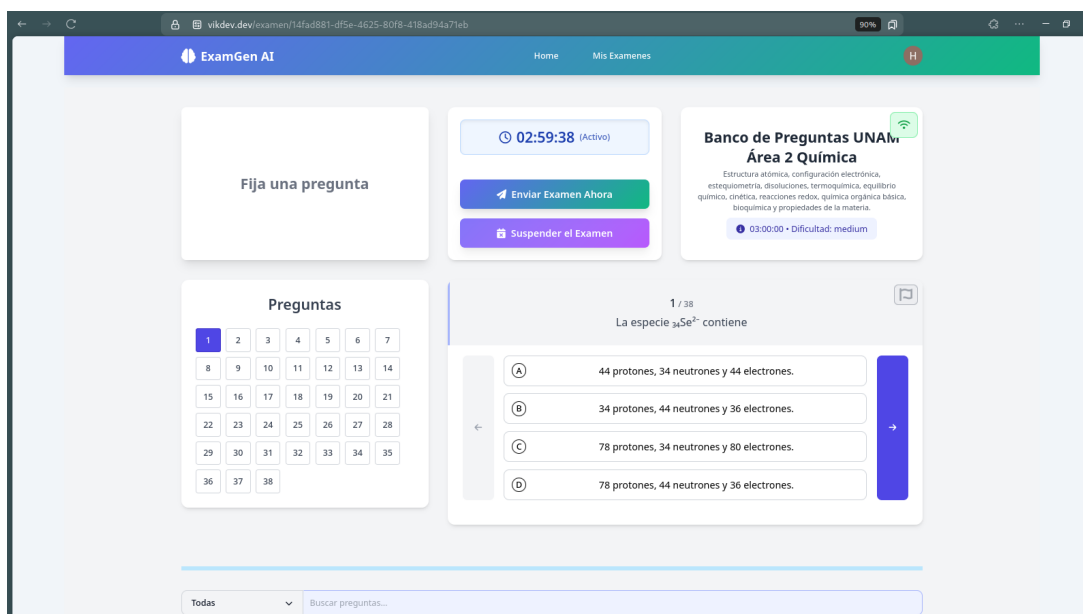


Figura 3.6: Interfaz de Examen con Navegador de Preguntas

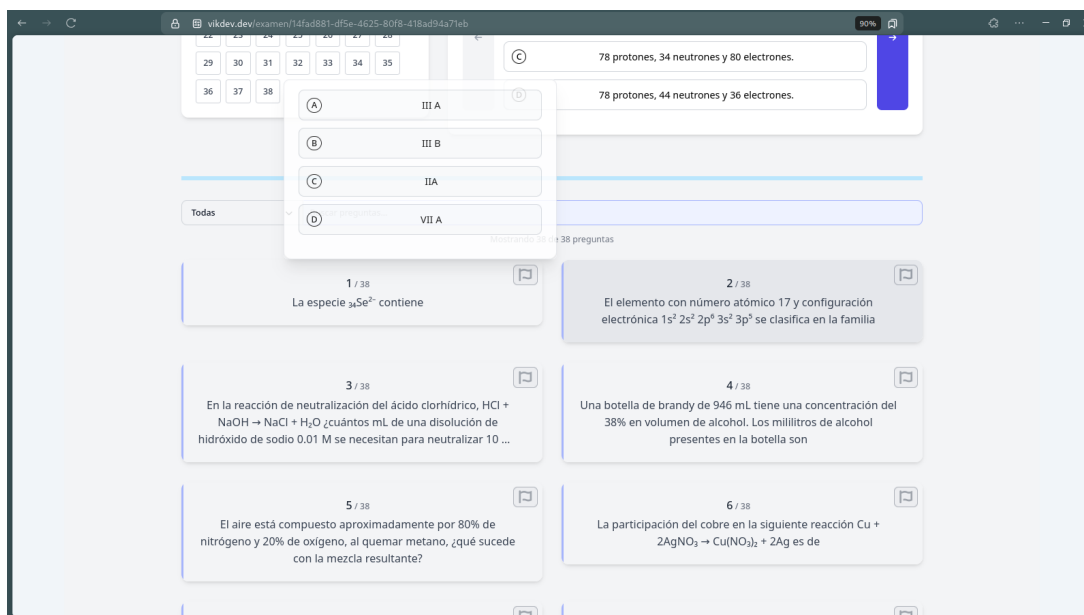


Figura 3.7: Vista Detallada de Preguntas con Opciones Múltiples

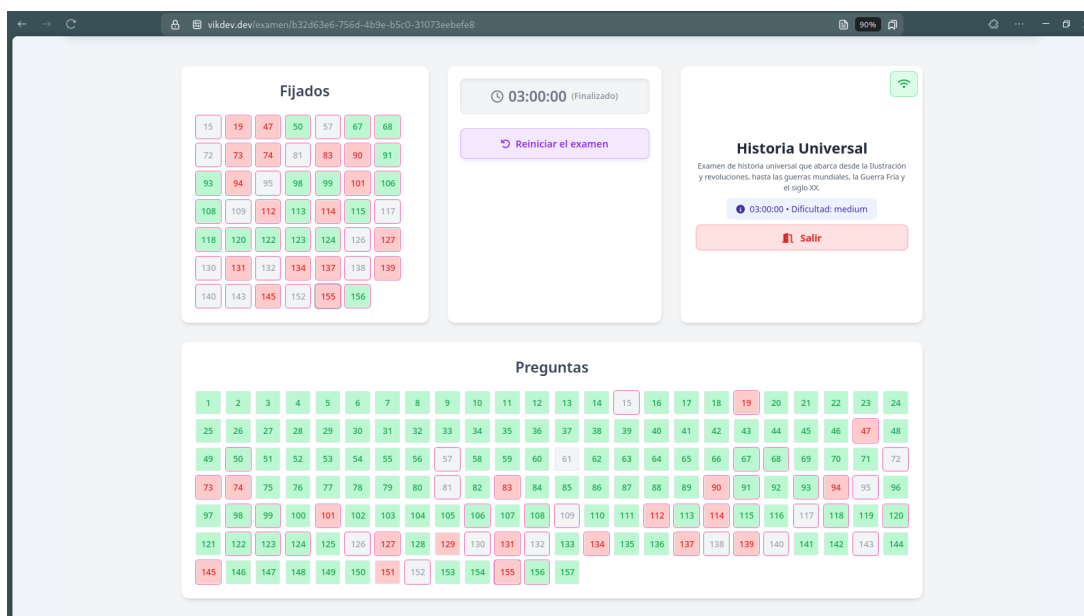


Figura 3.8: Vista de Progreso y Estado del Examen

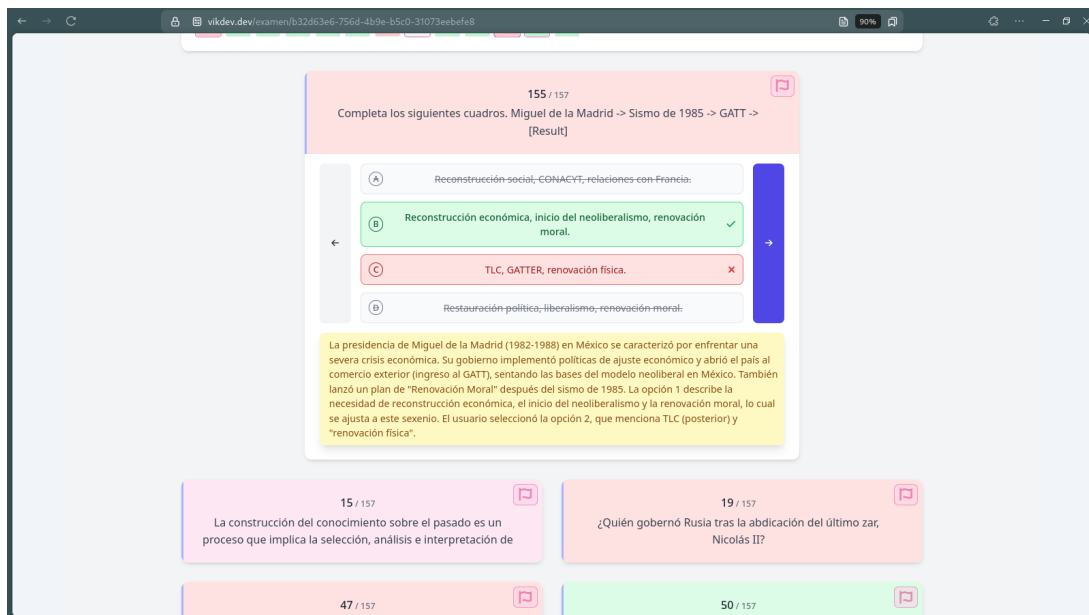


Figura 3.9: Visualización de Respuestas y Retroalimentación

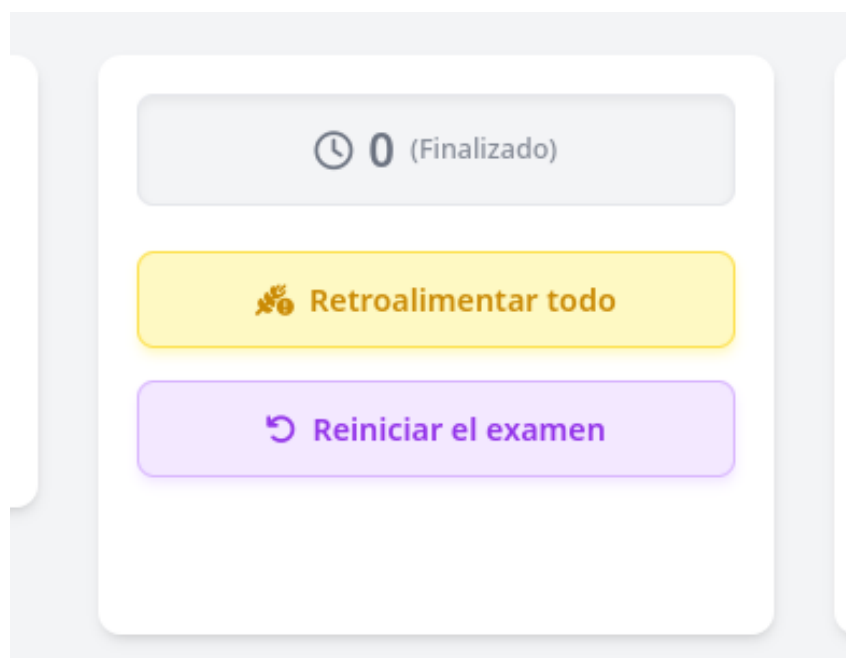


Figura 3.10: Pantalla de Finalización del Examen

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
- **Flutter SDK:** Para el wrapper móvil

4.1.2. Variables de Entorno

Frontend (.env)

```
1 # Configuración de Supabase
2 VITE_SUPABASE_URL=https://tu-proyecto.supabase.co
3 VITE_SUPABASE_ANON_KEY=tu_clave_anonima
4
5 # URL del backend
6 VITE_BACKEND_URL=http://localhost:3001
7
8 # Configuración de desarrollo
9 VITE_APP_ENV=development
10 VITE_DEBUG_MODE=true
```

Backend (.env)

```
1 # API Key de Google Gemini
2 GEMINI_API_KEY=tu_api_key_de_gemini
3
4 # Configuración de Supabase
5 SUPABASE_URL=https://tu-proyecto.supabase.co
6 SUPABASE_SERVICE_ROLE_KEY=tu_service_role_key
7
8 # Configuración del servidor
9 PORT=3001
10 NODE_ENV=development
11
12 # Configuración de CORS
13 CORS_ORIGIN=http://localhost:5173
```

4.2. Dependencias del Proyecto

4.2.1. Dependencias de Producción

Frontend

```
1 {
2   "dependencies": {
3     "react": "^19.0.0",
4     "react-dom": "^19.0.0",
5     "react-router-dom": "^6.8.0",
6     "@supabase/supabase-js": "^2.38.0",
7     "framer-motion": "^10.16.4",
8     "sweetalert2": "^11.7.32",
9     "react-markdown": "^8.0.7",
10    "katex": "^0.16.8",
11    "react-katex": "^3.0.1"
12  }
13 }
```

Backend

```
1 {
2   "dependencies": {
3     "express": "^4.18.2",
4     "@google/generative-ai": "^0.1.3",
5     "@supabase/supabase-js": "^2.38.0",
6     "multer": "^1.4.5-lts.1",
7     "cors": "^2.8.5",
8     "dotenv": "^16.3.1"
9   }
10 }
```

4.2.2. Dependencias de Desarrollo

```
1 {  
2   "devDependencies": {  
3     "@types/react": "^18.2.37",  
4     "@types/react-dom": "^18.2.15",  
5     "@vitejs/plugin-react": "^4.1.0",  
6     "vite": "^4.5.0",  
7     "tailwindcss": "^3.3.0",  
8     "autoprefixer": "^10.4.16",  
9     "postcss": "^8.4.31",  
10    "eslint": "^8.53.0",  
11    "prettier": "^3.0.3"  
12  }  
13 }
```

4.3. Proceso de Build

4.3.1. Comandos de Desarrollo

Instalación

```
1 # Clonar el repositorio  
2 git clone https://github.com/tu-usuario/reacti.git  
3 cd reacti  
4  
5 # Instalar dependencias del frontend  
6 cd frontend  
7 npm install  
8  
9 # Instalar dependencias del backend  
10 cd ../backend  
11 npm install
```

Ejecución en Desarrollo

```
1 # Terminal 1: Ejecutar backend  
2 cd backend  
3 npm start  
4  
5 # Terminal 2: Ejecutar frontend  
6 cd frontend  
7 npm run dev
```

4.3.2. Build de Producción

Frontend

```
1 # Generar build optimizado  
2 cd frontend  
3 npm run build  
4  
5 # El directorio 'dist' contiene los archivos optimizados  
6 ls dist/
```

Backend

```
1 # Configurar variables de entorno de produccion
2 export NODE_ENV=production
3 export PORT=3001
4
5 # Ejecutar en produccion
6 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

```
1 CREATE TABLE public.users (  
2     id UUID DEFAULT gen_random_uuid() PRIMARY KEY,  
3     email VARCHAR(255) UNIQUE NOT NULL,  
4     full_name VARCHAR(255),  
5     avatar_url TEXT,  
6     created_at TIMESTAMP DEFAULT NOW(),  
7     updated_at TIMESTAMP DEFAULT NOW()  
8 );
```

Tabla: exams

```
1 CREATE TABLE public.exams (  
2     id UUID DEFAULT gen_random_uuid() PRIMARY KEY,  
3     user_id UUID REFERENCES public.users(id) ON DELETE CASCADE,  
4     titulo VARCHAR(500) NOT NULL,  
5     descripcion TEXT,  
6     preguntas JSONB NOT NULL,  
7     dificultad VARCHAR(50) DEFAULT 'medium',  
8     numero_preguntas INTEGER DEFAULT 5,  
9     tiempo_limite INTEGER DEFAULT 1800,  
10    created_at TIMESTAMP DEFAULT NOW(),  
11    updated_at TIMESTAMP DEFAULT NOW()  
12 );  
13  
14 -- Indice para busquedas por usuario  
15 CREATE INDEX idx_exams_user_id ON public.exams(user_id);  
16 -- Indice para busquedas por fecha  
17 CREATE INDEX idx_exams_created_at ON public.exams(created_at);
```

Tabla: exam_results

```
1 CREATE TABLE public.exam_results (  
2     id UUID DEFAULT gen_random_uuid() PRIMARY KEY,  
3     exam_id UUID REFERENCES public.exams(id) ON DELETE CASCADE,  
4     user_id UUID REFERENCES public.users(id) ON DELETE CASCADE,  
5     respuestas JSONB NOT NULL,  
6     puntaje INTEGER NOT NULL DEFAULT 0,  
7     puntaje_maximo INTEGER NOT NULL DEFAULT 100,  
8     tiempo_empleado INTEGER, -- en segundos  
9     completed_at TIMESTAMP DEFAULT NOW(),  
10  
11     UNIQUE(exam_id, user_id) -- Un usuario solo puede hacer un examen  
12     una vez  
12 );
```

Tabla: feedback

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

```

3      exam_result_id UUID REFERENCES public.exam_results(id) ON DELETE
      CASCADE,
4      user_id UUID REFERENCES public.users(id) ON DELETE CASCADE,
5      feedback_data JSONB NOT NULL,
6      ai_analysis TEXT,
7      recommendations TEXT[],
8      created_at TIMESTAMP DEFAULT NOW()
9  );

```

5.1.3. Políticas de Seguridad

Row Level Security (RLS)

```

1  -- Habilitar RLS en todas las tablas
2  ALTER TABLE public.users ENABLE ROW LEVEL SECURITY;
3  ALTER TABLE public.exams ENABLE ROW LEVEL SECURITY;
4  ALTER TABLE public.exam_results ENABLE ROW LEVEL SECURITY;
5  ALTER TABLE public.feedback ENABLE ROW LEVEL SECURITY;
6
7  -- Politica para que usuarios solo vean sus propios datos
8  CREATE POLICY "Users can view own data" ON public.users
9      FOR ALL USING (auth.uid() = id);
10
11 CREATE POLICY "Users can manage own exams" ON public.exams
12     FOR ALL USING (auth.uid() = user_id);
13
14 CREATE POLICY "Users can view own results" ON public.exam_results
15     FOR ALL USING (auth.uid() = user_id);
16
17 CREATE POLICY "Users can view own feedback" ON public.feedback
18     FOR ALL USING (auth.uid() = user_id);

```

5.2. Integración con Google Gemini

5.2.1. Configuración del Modelo

Inicialización del Cliente

```

1  import { GoogleGenerativeAI } from "@google/generative-ai";
2
3  class GeminiService {
4      constructor() {
5          this.genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
6          this.modelFlash = this.genAI.getGenerativeModel({
7              model: "gemini-2.0-flash"
8          });
9          this.modelPro = this.genAI.getGenerativeModel({
10             model: "gemini-2.5-pro-exp-03-25"
11         });
12     }
13
14     async generateExam(content, difficulty = 'medium') {
15         const model = difficulty === 'hard' ? this.modelPro : this.
            modelFlash;

```

```

16     const prompt = this.buildExamPrompt(content, difficulty);
17
18     try {
19         const result = await model.generateContent(prompt);
20         return this.parseExamResponse(result.response.text());
21     } catch (error) {
22         throw new Error('Error generando examen: ${error.message}');
23     }
24 }
25 }

```

5.2.2. Funcionalidades de IA

Generación de Preguntas

```

1 buildExamPrompt(content, difficulty) {
2     return `
3         Analiza el siguiente contenido y crea un examen con estas
4         especificaciones:
5
6         Contenido: ${content}
7         Dificultad: ${difficulty}
8
9         Genera un JSON con la siguiente estructura:
10        {
11            "titulo": "Titulo del examen",
12            "descripcion": "Descripcion breve",
13            "numero_preguntas": 5,
14            "dificultad": "${difficulty}",
15            "preguntas": [
16                {
17                    "id": 1,
18                    "pregunta": "Texto de la pregunta",
19                    "opciones": ["opcion1", "opcion2", "opcion3", "
20                    opcion4"],
21                    "correcta": 0,
22                    "explicacion": "Explicacion de la respuesta correcta
23                    "
24                }
25            ]
26        }
27
28        Asegurate de que las preguntas sean relevantes y del nivel
29        solicitado.
30    `;
31 }

```

Análisis de Resultados

```

1 async generateFeedback(examResult) {
2     const prompt = `
3         Analiza los resultados de este examen y proporciona feedback
4         personalizado:
5
6         Puntaje obtenido: ${examResult.puntaje}/${examResult.
7         puntaje_maximo}
8     `;
9 }

```

```
6      Tiempo empleado: ${examResult.tiempo_empleado} segundos
7      Respuestas: ${JSON.stringify(examResult.respuestas)}
8
9      Genera un analisis que incluya:
10     1. Resumen del desempeno
11     2. Fortalezas identificadas
12     3. Areas de mejora
13     4. Recomendaciones especificas de estudio
14     5. Proximos pasos sugeridos
15
16     El feedback debe ser constructivo y motivador.
17     ‘;
18
19     const result = await this.modelPro.generateContent(prompt);
20     return result.response.text();
21 }
```

Capítulo 6

API Endpoints y Comunicación

6.1. Endpoints de Autenticación

6.1.1. Registro de Usuario

```
1 POST /api/auth/register
2 Content-Type: application/json
3
4 {
5     "email": "usuario@ejemplo.com",
6     "password": "password123",
7     "full_name": "Nombre Completo"
8 }
9
10 Response:
11 {
12     "success": true,
13     "user": {
14         "id": "uuid",
15         "email": "usuario@ejemplo.com",
16         "full_name": "Nombre Completo"
17     },
18     "token": "jwt_token"
19 }
```

6.1.2. Inicio de Sesión

```
1 POST /api/auth/login
2 Content-Type: application/json
3
4 {
5     "email": "usuario@ejemplo.com",
6     "password": "password123"
7 }
8
9 Response:
10 {
11     "success": true,
12     "user": { ... },
13     "token": "jwt_token"
14 }
```

```
14 }
```

6.2. Endpoints de Exámenes

6.2.1. Generar Examen desde Archivo

```
1 POST /api/upload_files
2 Authorization: Bearer jwt_token
3 Content-Type: multipart/form-data
4
5 Files: archivo.pdf
6 Body:
7 {
8     "prompt": "Genera preguntas sobre este documento",
9     "tiempo_limite_segundos": 1800,
10    "numero_preguntas": 10,
11    "dificultad": "medium"
12 }
13
14 Response:
15 {
16     "success": true,
17     "exam": {
18         "id": "uuid",
19         "titulo": "Examen generado",
20         "preguntas": [...],
21         "tiempo_limite": 1800
22     }
23 }
```

6.2.2. Generar Examen desde Texto

```
1 POST /api/generate-content
2 Authorization: Bearer jwt_token
3 Content-Type: application/json
4
5 {
6     "prompt": "Crea un examen sobre JavaScript",
7     "dificultad": "medium",
8     "numero_preguntas": 8,
9     "tiempo_limite_segundos": 1200
10 }
11
12 Response:
13 {
14     "success": true,
15     "exam": { ... }
16 }
```

6.2.3. Obtener Exámenes del Usuario

```
1 GET /api/exams
2 Authorization: Bearer jwt_token
3
4 Response:
5 {
6     "success": true,
7     "exams": [
8         {
9             "id": "uuid",
10            "titulo": "Examen de JavaScript",
11            "created_at": "2025-06-19T10:00:00Z",
12            "numero_preguntas": 10,
13            "dificultad": "medium"
14        }
15    ]
16 }
```

6.3. Endpoints de Resultados

6.3.1. Enviar Respuestas de Examen

```
1 POST /api/exam-results
2 Authorization: Bearer jwt_token
3 Content-Type: application/json
4
5 {
6     "exam_id": "uuid",
7     "respuestas": [
8         {
9             "pregunta_id": 1,
10            "respuesta_seleccionada": 2
11        }
12    ],
13     "tiempo_empleado": 900
14 }
15
16 Response:
17 {
18     "success": true,
19     "result": {
20         "id": "uuid",
21         "puntaje": 85,
22         "puntaje_maximo": 100,
23         "respuestas_correctas": 8,
24         "total_preguntas": 10
25     }
26 }
```

6.3.2. Generar Retroalimentación

```
1 POST /api/generate-feedback
2 Authorization: Bearer jwt_token
3 Content-Type: application/json
4
```

```
5 {  
6   "exam_result_id": "uuid"  
7 }  
8  
9 Response:  
10 {  
11   "success": true,  
12   "feedback": {  
13     "analysis": "Análisis detallado del desempeño...",  
14     "strengths": ["Conceptos basicos", "Sintaxis"],  
15     "improvements": ["Programacion asincrona"],  
16     "recommendations": ["Practicar con async/await"]  
17   }  
18 }
```


Capítulo 7

Costos y Viabilidad Económica

7.1. Análisis Detallado de Costos

El Sistema de Generación de Exámenes con IA (ExamGenAI) ha sido desarrollado con un enfoque de costos optimizados, ideal para equipos pequeños y proyectos educativos con presupuesto limitado.

7.1.1. Estructura de Costos

Costos de Desarrollo

- **Tiempo de desarrollo:** 4 meses aproximadamente
- **Herramientas de desarrollo:** Gratuitas (VS Code, Git, etc.)

Costos de Infraestructura

Servicio	Proveedor	Costo Mensual
Base de Datos	Supabase (Tier gratuito)	\$0.00
Hosting Frontend	Hostinger	\$2.99
API Gemini	Google (cuota gratuita)	\$0.00
Servidor Backend	VPS básico	\$5.00
Dominio	Registro anual	\$1.00
Total Mensual		\$8.99

Cuadro 7.1: Desglose Completo de Costos de Infraestructura

Costos de Escalamiento

- **Supabase Pro:** \$25/mes (para más de 500MB de DB)
- **Google Gemini Pro:** \$0.002 por 1K tokens de entrada
- **VPS Pro:** \$20/mes (para mayor tráfico)
- **CDN:** \$10/mes (para distribución global)

7.1.2. Ventajas del Modelo de Costos

- **Inicio gratuito:** Posible comenzar sin inversión inicial
- **Escalamiento gradual:** Costos crecen con el uso
- **Sin licencias costosas:** Tecnologías open source
- **Mantenimiento mínimo:** Servicios administrados

7.1.3. Comparación con Alternativas

Solución	Costo Inicial	Costo Mensual	Costo Anual
ExamGenAI	\$0	\$8.99	\$107.88
Moodle + Hosting	\$500	\$25.00	\$800.00
Canvas LMS	\$0	\$50.00	\$600.00
Blackboard	\$2000	\$100.00	\$3200.00

Cuadro 7.2: Comparación de Costos con Plataformas Educativas

7.2. Modelo de Negocio

7.2.1. Opciones de Monetización

- **Freemium:** Versión básica gratuita, funciones avanzadas de pago
- **Suscripción institucional:** Planes para escuelas y universidades
- **API como servicio:** Integración con otras plataformas
- **Consultoría:** Implementación personalizada

7.2.2. Proyección de Ingresos

Plan	Precio Mensual	Usuarios Objetivo	Ingreso Mensual
Básico	\$0	1000	\$0
Pro	\$9.99	100	\$999
Institucional	\$49.99	20	\$999.80
Enterprise	\$199.99	5	\$999.95
Total Proyectado		1125	\$2998.75

Cuadro 7.3: Proyección de Ingresos Mensuales

Capítulo 8

Seguridad y Optimización

8.1. Medidas de Seguridad

8.1.1. 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

8.1.2. 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

8.2. Optimización de Rendimiento

8.2.1. Frontend

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

8.2.2. Backend

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

Capítulo 9

Despliegue, Monitoreo y Mantenimiento

9.1. Proceso de Despliegue

9.1.1. Frontend (Hostinger)

```
1 # 1. Generar build de produccion
2 npm run build
3
4 # 2. Configurar variables de entorno de produccion
5 VITE_SUPABASE_URL=https://prod.supabase.co
6 VITE_BACKEND_URL=https://api.reacti.com
7
8 # 3. Subir archivos a hosting
9 # Copiar contenido de 'dist' al directorio public_html
```

9.1.2. Backend (Servidor VPS)

```
1 # 1. Clonar repositorio en servidor
2 git clone https://github.com/usuario/reacti.git
3 cd reacti/backend
4
5 # 2. Instalar dependencias
6 npm install --production
7
8 # 3. Configurar variables de entorno
9 export NODE_ENV=production
10 export GEMINI_API_KEY=tu_api_key
11 export SUPABASE_URL=https://prod.supabase.co
12
13 # 4. Ejecutar con PM2
14 pm2 start src/index.js --name "reacti-api"
15 pm2 save
16 pm2 startup
```

Capítulo 10

Exposición Móvil con Flutter WebView

Para expandir el alcance de la plataforma ExamGenAI a dispositivos móviles de manera rápida y eficiente, se adoptó la estrategia de encapsular la aplicación web existente dentro de un contenedor móvil. La tecnología elegida para esta tarea fue Flutter, debido a su naturaleza multiplataforma y su facilidad para integrar WebViews.

10.1. Estrategia y Justificación

La adopción de un enfoque de WebView, en lugar de desarrollar una aplicación nativa desde cero, se basó en las siguientes consideraciones clave:

- **Rapidez de lanzamiento (Time-to-Market):** Permitió tener una versión móvil funcional en una fracción del tiempo que tomaría un desarrollo nativo completo.
- **Reutilización del código:** Se aprovecha el 100% del frontend desarrollado en React, incluyendo toda la lógica, componentes y comunicación con el backend.
- **Mantenimiento centralizado:** Cualquier actualización en la aplicación web se refleja inmediatamente en la aplicación móvil, sin necesidad de publicar una nueva versión en las tiendas.
- **Costo-efectividad:** Minimiza los costos de desarrollo al no requerir un equipo de desarrollo móvil dedicado.

10.2. Implementación Técnica

El núcleo de la aplicación móvil es el paquete `webview_flutter`. La implementación consiste en una aplicación de Flutter mínima cuya pantalla principal es un widget `WebViewWidget` que carga la URL de producción del frontend.

```
1 import 'package:flutter/material.dart';
2 import 'package:webview_flutter/webview_flutter.dart';
3
4 void main() {
5   runApp(const MyApp());
6 }
```

```
7
8 class MyApp extends StatelessWidget {
9   const MyApp({super.key});
10
11   @override
12   Widget build(BuildContext context) {
13     return MaterialApp(
14       title: 'ExamGenAI',
15       theme: ThemeData(
16         primarySwatch: Colors.blue,
17         useMaterial3: true,
18       ),
19       home: const WebViewPage(),
20       debugShowCheckedModeBanner: false,
21     );
22   }
23 }
24
25 class WebViewPage extends StatefulWidget {
26   const WebViewPage({super.key});
27
28   @override
29   State<WebViewPage> createState() => _WebViewPageState();
30 }
31
32 class _WebViewPageState extends State<WebViewPage> {
33   late final WebViewController _controller;
34   bool _isLoading = true;
35
36   @override
37   void initState() {
38     super.initState();
39     _controller = WebViewController()
40       ..setJavaScriptMode(JavascriptMode.unrestricted)
41       ..setNavigationDelegate(
42         NavigationDelegate(
43           onPageStarted: (String url) {
44             setState(() {
45               _isLoading = true;
46             });
47           },
48           onPageFinished: (String url) {
49             setState(() {
50               _isLoading = false;
51             });
52           },
53         ),
54       )
55       ..loadRequest(Uri.parse('https://vikdev.dev/')); // URL de
        Producci n
56   }
57
58   @override
59   Widget build(BuildContext context) {
60     return Scaffold(
61       body: SafeArea(
62         child: Stack(
63           children: [
```

```
64     WebViewWidget(controller: _controller),
65     if (_isLoading)
66       const Center(
67         child: CircularProgressIndicator(),
68       ),
69   ],
70 ),
71 ),
72 );
73 }
74 }
```

Listing 10.1: Implementación básica del WebView en Flutter

10.3. Ventajas y Limitaciones de la Estrategia

Ventajas

- **Desarrollo acelerado:** La principal ventaja es la velocidad de implementación.
- **Experiencia de usuario consistente:** La interfaz es idéntica en la web y en el móvil.
- **Un solo codebase para la UI/UX:** Reduce la complejidad del proyecto.

Limitaciones

- **Rendimiento:** El rendimiento puede no ser tan fluido como una aplicación nativa.
- **Dependencia de la conexión:** La aplicación requiere una conexión a internet constante.
- **Dificultad con flujos complejos:** Flujos de autenticación de terceros (OAuth) son problemáticos de implementar y a menudo son bloqueados por los proveedores por seguridad, requiriendo soluciones más complejas.
- **Acceso limitado a APIs nativas:** El acceso a hardware del dispositivo (cámara, GPS, etc.) es limitado.

Capítulo 11

Conclusión

11.1. Resumen del Sistema

El Sistema de Generación de Exámenes con IA (ExamGenAI) representa una solución moderna y escalable para la automatización de evaluaciones educativas. Utilizando tecnologías de vanguardia como React, Node.js, y Supabase, el sistema proporciona una experiencia web completa, extendida a dispositivos móviles a través de un wrapper básico con Flutter.

11.1.1. Características Principales

- **Generación automática:** Preguntas basadas en documentos o texto libre.
- **Múltiples dificultades:** Adaptación automática del nivel.
- **Retroalimentación IA:** Análisis detallado del desempeño.
- **Interfaz moderna:** Diseño responsive y accesible.
- **Seguridad robusta:** Protección de datos y autenticación segura.
- **Acceso multiplataforma:** Disponible en la web y como aplicación móvil (iOS/Android) a través de un WebView.

11.1.2. 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.

11.2. Trabajo Futuro

El enfoque futuro del proyecto se centra en la expansión de funcionalidades y la optimización de la infraestructura para un crecimiento sostenido.

11.2.1. Mejoras Planificadas

- **Soporte multiidioma:** Internacionalización completa de la plataforma.
- **Análisis avanzado:** Incorporar métricas de aprendizaje más sofisticadas y reportes visuales.
- **Integración con LMS:** Desarrollar compatibilidad con sistemas de gestión de aprendizaje (LMS) como Moodle o Canvas.
- **App móvil nativa:** Evolucionar el actual wrapper de WebView a una aplicación completamente nativa (en Flutter o nativo) para mejorar el rendimiento, habilitar funciones offline y permitir una integración robusta con flujos de autenticación de terceros y APIs del dispositivo.

11.2.2. Escalabilidad

- **Microservicios:** Migración gradual del backend a una arquitectura de microservicios.
- **Cache distribuido:** Implementación de Redis para almacenar en caché resultados de API frecuentes.
- **CDN (Content Delivery Network):** Uso de una CDN para distribuir globalmente los assets del frontend.
- **Auto-scaling:** Configuración de la infraestructura del backend para escalar automáticamente según la demanda.

Versión del Manual: 1.0

Fecha: 19 de junio de 2025

Desarrollado por: Hernández Tellez Héctor Fidel, Rivero Flores Victor Gabriel