|  |
| --- |
| DUOC UC - ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES |
| InterviewAI: Simulador de Entrevistas Laborales con IA |
| ***(SAD) Software Architecture Document***  *Versión 1.0* |
|  |
|  |
|  |

Especificación de Requisitos según estándar de IEEE 830.

**Equipo de Desarrollo:**

* Cristian Álvarez - AI Specialist & QA Lead
* Matías Coloma Contreras - Project Leader & Backend Developer
* Ulises Muñoz - Frontend Developer & UX/UI Designer

**Sección:** 008V  
**Profesor Guía:** Félix Eduardo Cifuentes

# Identificación de Documento

|  |  |
| --- | --- |
| **Identificación** | IAI\_DAS\_01 |
| **Proyecto** | InterviewAI - Simulador de Entrevistas con IA |
| **Versión** | 1.0 |
| **Documento mantenido por** | Matías Coloma Contreras |
| **Fecha última revisión** | 17/11/2025 |
| **Fecha próxima revisión** | 01/12/2025 |
| **Documento aprobado por** | Félix Eduardo Cifuentes |
| **Fecha última aprobación** | Pendiente |

# Historia de Revisiones

| **Fecha** | **Versión** | **Descripción** | **Autor** |
| --- | --- | --- | --- |
| 17/11/2025 | 1.0 | Inicio del documento DAS | Equipo InterviewAI |

|  |
| --- |
|  |

**Contenido**

Contenido

[Identificación de Documento 2](#_Toc215761920)

[Historia de Revisiones 2](#_Toc215761921)

[1. Introducción 4](#_Toc215761922)

[1.1. Contexto del Problema 4](#_Toc215761923)

[1.2. Propósito 4](#_Toc215761924)

[1.3. Definiciones, Acrónimos y Abreviaturas 5](#_Toc215761925)

[1.4. Referencias 5](#_Toc215761926)

[1.5. Resumen Ejecutivo 6](#_Toc215761927)

[2. Representación 6](#_Toc215761928)

[3. Metas y Restricciones de la Arquitectura 7](#_Toc215761929)

[3.1 Metas y Restricciones de la Arquitectura 7](#_Toc215761930)

[3.2 Restricciones de la Arquitectura 8](#_Toc215761931)

[3.3 Otros Antecedentes y Consideraciones 8](#_Toc215761932)

[4. Vista de Casos de Uso y Escenarios de Calidad 9](#_Toc215761933)

[4.1 Modelo de Casos de Uso 9](#_Toc215761934)

[4.2 Especificación de Casos de Uso Relevantes 9](#_Toc215761935)

[4.3 Especificación de Escenarios de Calidad Relevantes 10](#_Toc215761936)

[5. Vista Lógica 13](#_Toc215761937)

[5.1 Diagrama de Clases Principal 13](#_Toc215761938)

[5.2 Diagrama de Secuencia: Realizar Entrevista 14](#_Toc215761939)

[6. Vista de Procesos 15](#_Toc215761940)

[6.1 Proceso: Realizar Entrevista Completa 15](#_Toc215761941)

[6.2 Proceso: Registro de Usuario 16](#_Toc215761942)

[7. Vista de Implementación 17](#_Toc215761943)

[7.1 Arquitectura General 17](#_Toc215761944)

[7.2 Servicios Externos Integrados 17](#_Toc215761945)

[7.3 Estructura de Módulos 18](#_Toc215761946)

[8. Vista de Despliegue 19](#_Toc215761947)

[8.1 Arquitectura de Despliegue 19](#_Toc215761948)

[8.2 Flujo de Comunicación entre Nodos 20](#_Toc215761949)

[9. Diagrama de Componentes 22](#_Toc215761950)

[10. Decisiones de Diseño y Selección de Alternativas 23](#_Toc215761951)

[10.1 Selección de Arquitectura Cliente-Servidor 23](#_Toc215761952)

[10.2 Selección de Stack Tecnológico 23](#_Toc215761953)

[10.3 Patrón de Autenticación JWT 23](#_Toc215761954)

[10.4 Estrategia de Integración con APIs Externas 24](#_Toc215761955)

[10.5 Diseño de Base de Datos Relacional 24](#_Toc215761956)

[10.6 Gestión de Estado en Frontend: Context API vs Redux 24](#_Toc215761957)

[10.7 Almacenamiento de Tokens JWT 25](#_Toc215761958)

[10.8 Estrategia de Fallback para APIs Externas 26](#_Toc215761959)

[10.9 Decisión de No Almacenar Audio 28](#_Toc215761960)

[10.10 Componentes de Código Reutilizables 30](#_Toc215761961)

[11. Análisis de Reutilización 32](#_Toc215761962)

[11.1 Componentes de Código Reutilizables 32](#_Toc215761963)

[11.2 Patrones de Diseño Aplicados 32](#_Toc215761964)

[11.3 Consideraciones para Migración Futura 33](#_Toc215761965)

[12. Análisis de Riesgos Técnicos y Estrategias de Mitigación 33](#_Toc215761966)

[12.1 Introducción 33](#_Toc215761967)

[12.2 Matriz de Riesgos 34](#_Toc215761968)

[12.3 Riesgos Críticos y Mitigación Detallada 34](#_Toc215761969)

[12.4 Plan de Acción y Responsabilidades 44](#_Toc215761970)

[12.5 Conclusión del Análisis de Riesgos 44](#_Toc215761971)

[13. Conclusión 45](#_Toc215761972)

# 1. Introducción

## 1.1. Contexto del Problema

El mercado laboral actual, especialmente en el sector tecnológico, presenta desafíos significativos para los candidatos que buscan empleo. Las entrevistas laborales son un componente crítico del proceso de selección, pero muchos profesionales, particularmente aquellos en inicio de carrera o en transición profesional, carecen de oportunidades para practicar y desarrollar sus habilidades de entrevista de manera efectiva.

**Los problemas identificados incluyen:**

* Acceso limitado a simulaciones realistas de entrevistas
* Falta de retroalimentación personalizada y constructiva
* Ansiedad y nerviosismo causados por la falta de preparación
* Alto costo de servicios de coaching profesional
* Dificultad para acceder a preguntas relevantes y actualizadas del sector

tecnológico

InterviewAI surge como respuesta a estas necesidades, proporcionando una plataforma accesible, inteligente y efectiva para la práctica de entrevistas laborales mediante el uso de inteligencia artificial avanzada.

## 1.2. Propósito

La intención del presente documento es capturar las definiciones estructurales y decisiones arquitectónicas que en conjunto dan como resultado el compendio de la solución técnica para el sistema InterviewAI.

**Este documento sirve como:**

* Guía técnica para el equipo de desarrollo durante la implementación
* Referencia para la toma de decisiones arquitectónicas
* Documentación para evaluadores académicos
* Base para futuras ampliaciones y mantenimiento del sistema

## 1.3. Definiciones, Acrónimos y Abreviaturas

| **Término** | **Descripción** |
| --- | --- |
| **IA** | Inteligencia Artificial |
| **API** | Application Programming Interface - Interfaz de programación de aplicaciones |
| **STT** | Speech-to-Text - Conversión de voz a texto |
| **TTS** | Text-to-Speech - Conversión de texto a voz |
| **LLM** | Large Language Model - Modelo de lenguaje extenso |
| **REST** | Representational State Transfer - Arquitectura de servicios web |
| **JWT** | JSON Web Token - Token de autenticación |
| **HTTPS** | Hypertext Transfer Protocol Secure - Protocolo seguro de transferencia |
| **MVC** | Model-View-Controller - Patrón de arquitectura de software |
| **SPA** | Single Page Application - Aplicación de página única |
| **UI/UX** | User Interface / User Experience - Interfaz y experiencia de usuario |

## 1.4. Referencias

**A continuación se listan las referencias a otros documentos del proyecto:**

* Especificación de Requisitos de Software (ERS) - InterviewAI v1.0
* Documento de Casos de Uso - InterviewAI
* Modelo de Análisis y Diseño
* Catálogo de Actores y Escenarios de Calidad
* Plan de Pruebas y Testing

## 1.5. Resumen Ejecutivo

InterviewAI es una plataforma web innovadora que utiliza inteligencia artificial para simular entrevistas laborales del sector tecnológico. El sistema permite a los usuarios practicar sus habilidades de entrevista mediante interacciones con un avatar conversacional impulsado por IA, recibiendo retroalimentación personalizada y análisis detallado de su desempeño.

El proyecto se estructura sobre una arquitectura cliente-servidor moderna, utilizando React para el frontend, Node.js/Express para el backend, y PostgreSQL como base de datos. La integración con servicios de IA externos (OpenAI GPT-4, Google Speech-to-Text, D-ID Avatar) proporciona capacidades avanzadas de procesamiento de lenguaje natural, reconocimiento de voz y generación de avatar conversacional.

# 2. Representación

La arquitectura del sistema InterviewAI está representada siguiendo el enfoque del framework 4+1 y las recomendaciones del proceso unificado. Las vistas incluidas en este documento son:

**Vista de Casos de Uso y Escenarios de Calidad**: Describe los casos de uso más significativos, presenta los actores y una descripción de sus casos de uso asociados. También describe los escenarios de calidad más relevantes para la arquitectura.

**Vista de Metas y Restricciones**: Describe restricciones tecnológicas, normativas, estándares, etc., los cuales influyen sobre las decisiones arquitectónicas del producto y del proceso de desarrollo.

**Vista Lógica**: Describe la arquitectura del sistema presentando varios niveles de refinamiento. Indica los módulos lógicos principales, sus responsabilidades y dependencias.

**Vista de Procesos**: Describe los procesos involucrados para darle sentido a la ejecución del sistema, así como sus relaciones de comunicación y sincronización.

**Vista de Implementación**: Describe los componentes de deployment construidos y sus dependencias.

# 3. Metas y Restricciones de la Arquitectura

## 3.1 Metas y Restricciones de la Arquitectura

De acuerdo con las reuniones y al análisis de los requerimientos, se listan los principales conductores iniciales de la arquitectura:

**Desempeño**:

Proveer un sistema de información capaz de soportar los requerimientos funcionales declarados, con tiempos de respuesta óptimos. La generación de preguntas por IA debe completarse en menos de 3 segundos, y la sincronización del avatar debe ser fluida (menos de 500ms de latencia).

**Escalabilidad**:

La arquitectura debe soportar el crecimiento incremental de usuarios. Inicialmente diseñado para 100 usuarios concurrentes, debe poder escalar a 1000+ usuarios mediante optimización de recursos Cloud y balanceo de carga.

**Seguridad**:

Tener la capacidad de administrar los datos de los usuarios con alta seguridad. Las contraseñas deben ser encriptadas usando bcrypt con factor de costo 12. Implementar autenticación JWT con tokens de corta duración (15 minutos) y refresh tokens. Todo tráfico debe ser HTTPS con certificado SSL/TLS válido.

**Usabilidad**:

Interfaz intuitiva y accesible que permita a usuarios de diferentes niveles técnicos utilizar la plataforma sin dificultad. Cumplimiento con estándares WCAG 2.1 nivel AA para accesibilidad web.

**Mantenibilidad**:

Código modular y bien documentado que facilite futuras ampliaciones y correcciones. Uso de patrones de diseño reconocidos (MVC, Repository Pattern) y separación clara de responsabilidades entre capas.

**Interoperabilidad**:

Integración efectiva con APIs externas (OpenAI, Google Cloud, D-ID) mediante interfaces REST estandarizadas. Uso de formatos de intercambio comunes (JSON) y manejo robusto de errores de servicios externos.

## 3.2 Restricciones de la Arquitectura

Existen restricciones que han sido levantadas con los stakeholders:

**Tiempo de construcción**:

Se cuenta con un plazo de 18 semanas para completar el desarrollo del proyecto, desde el análisis inicial hasta el deployment en producción.

**Presupuesto**:

Proyecto académico con presupuesto limitado. Se utilizarán servicios con tier gratuito o de bajo costo (Vercel, Railway, servicios cloud con créditos educativos).

**Tecnologías**:

El stack tecnológico está definido por el expertise del equipo: React para frontend, Node.js/Express para backend, PostgreSQL para base de datos.

**APIs** **Externas**:

Dependencia de servicios externos (OpenAI, Google Cloud, D-ID) que tienen limitaciones de rate limiting y costos por uso. Se debe implementar manejo de cuotas y fallbacks apropiados.

**Equipo**:

Equipo de 3 desarrolladores con roles especializados, lo que requiere coordinación efectiva y división clara de responsabilidades.

## 3.3 Otros Antecedentes y Consideraciones

El equipo cuenta con las siguientes fortalezas tecnológicas:

* React ofrece desarrollo ágil de interfaces modernas y reactivas, con amplio ecosistema de librerías
* Node.js/Express proporciona un backend eficiente y escalable con JavaScript full-stack
* PostgreSQL ofrece robustez, ACID compliance y excelente soporte para datos estructurados
* OpenAI GPT-4 proporciona capacidades avanzadas de procesamiento de lenguaje natural
* Google Cloud Speech-to-Text ofrece reconocimiento de voz preciso en múltiples idiomas

# 4. Vista de Casos de Uso y Escenarios de Calidad

## 4.1 Modelo de Casos de Uso

El modelo completo de casos de uso puede ser encontrado en el documento 'Especificación de Requisitos de Software (ERS)' y en el 'Documento de Casos de Uso'.

## 4.2 Especificación de Casos de Uso Relevantes

Los casos de uso más relevantes para la arquitectura fueron determinados considerando:

* Implementación que involucra múltiples componentes y servicios externos
* Alto riesgo técnico de implementación
* Críticos para la propuesta de valor del sistema
* Incluyen escenarios de calidad significativos

Casos de uso arquitectónicamente significativos:

| **Código** | **Nombre** | **Actores** | **Prioridad** |
| --- | --- | --- | --- |
| CU-01 | Realizar Entrevista con IA | Usuario, Sistema, IA | Alta |
| CU-02 | Registrar Usuario | Usuario, Sistema | Alta |
| CU-03 | Obtener Retroalimentación IA | Usuario, Sistema, IA | Alta |
| CU-04 | Gestionar Perfil Profesional | Usuario, Sistema | Media |
| CU-05 | Visualizar Historial de Entrevistas | Usuario, Sistema | Media |

## 4.3 Especificación de Escenarios de Calidad Relevantes

Después de un análisis con los stakeholders, los escenarios de calidad más relevantes son:

**ID: QS1**

**Nombre**: Desempeño en Generación de Preguntas IA

**Sinopsis**: Tiempo de respuesta en la generación de preguntas contextualizadas por IA.

**Entorno**: Usuario inicia una sesión de entrevista.

**Cambio en el entorno**: Sistema solicita generación de pregunta a OpenAI GPT-4.

**Comportamiento esperado**: La pregunta se genera y muestra al usuario en menos de 3 segundos.

**Medida**: 95% de las solicitudes completan en < 3 segundos.

**Prioridad Arquitectónica**: Alta

**Aplicación**: Global

**ID: QS2**

**Nombre**: Seguridad de Datos de Usuario

**Sinopsis**: Protección de información sensible de usuarios mediante encriptación y autenticación robusta.

**Entorno**: Sistema opera normalmente.

**Cambio en el entorno**: Intento de acceso no autorizado o interceptación de datos.

**Comportamiento esperado**: Sistema detecta y bloquea intentos no autorizados. Contraseñas encriptadas con bcrypt. Comunicación HTTPS.

**Medida**: 0% de brechas de seguridad. 100% de contraseñas encriptadas. Todos los endpoints requieren autenticación JWT.

**Prioridad Arquitectónica**: Alta

**Aplicación**: Global

**ID: QS3**

**Nombre**: Disponibilidad y Tolerancia a Fallos

**Sinopsis**: Sistema mantiene disponibilidad ante fallos de servicios externos.

**Entorno**: Sistema opera con dependencias externas (OpenAI, Google Cloud, D-ID).

**Cambio en el entorno**: Fallo temporal de servicio externo (timeout, rate limit, error 5xx).

**Comportamiento esperado**: Sistema implementa retry logic con backoff exponencial. Muestra mensajes de error amigables. Permite al usuario reintentar.

**Medida**: Uptime del 99% durante horario de uso. Máximo 3 intentos de retry antes de fallo.

**Prioridad Arquitectónica**: Alta

**Aplicación**: Global

**ID: QS4**

**Nombre**: Usabilidad e Interfaz Intuitiva

**Sinopsis**: Usuarios pueden navegar y usar el sistema sin capacitación previa.

**Entorno**: Usuario nuevo accede al sistema por primera vez.

**Cambio en el entorno**: Usuario intenta completar una entrevista completa.

**Comportamiento esperado**: Usuario completa exitosamente sin ayuda externa. Interfaz clara con feedback visual.

**Medida**: 80% de usuarios nuevos completan entrevista sin ayuda. Tiempo promedio de primera entrevista < 10 minutos.

**Prioridad Arquitectónica**: Alta

**Aplicación**: Frontend

**ID: QS5**

**Nombre**: Mantenibilidad del Código

**Sinopsis**: Sistema diseñado para facilitar mantenimiento y futuras ampliaciones.

**Entorno**: Sistema en producción requiere nueva funcionalidad.

**Cambio en el entorno**: Desarrollador debe agregar nuevo tipo de entrevista o integrar nueva API.

**Comportamiento esperado**: Código modular permite agregar funcionalidad sin afectar componentes existentes.

**Medida**: Tiempo para implementar cambio < 3 días. Cobertura de tests > 70%.

**Prioridad Arquitectónica**: Media

**Aplicación**: Global

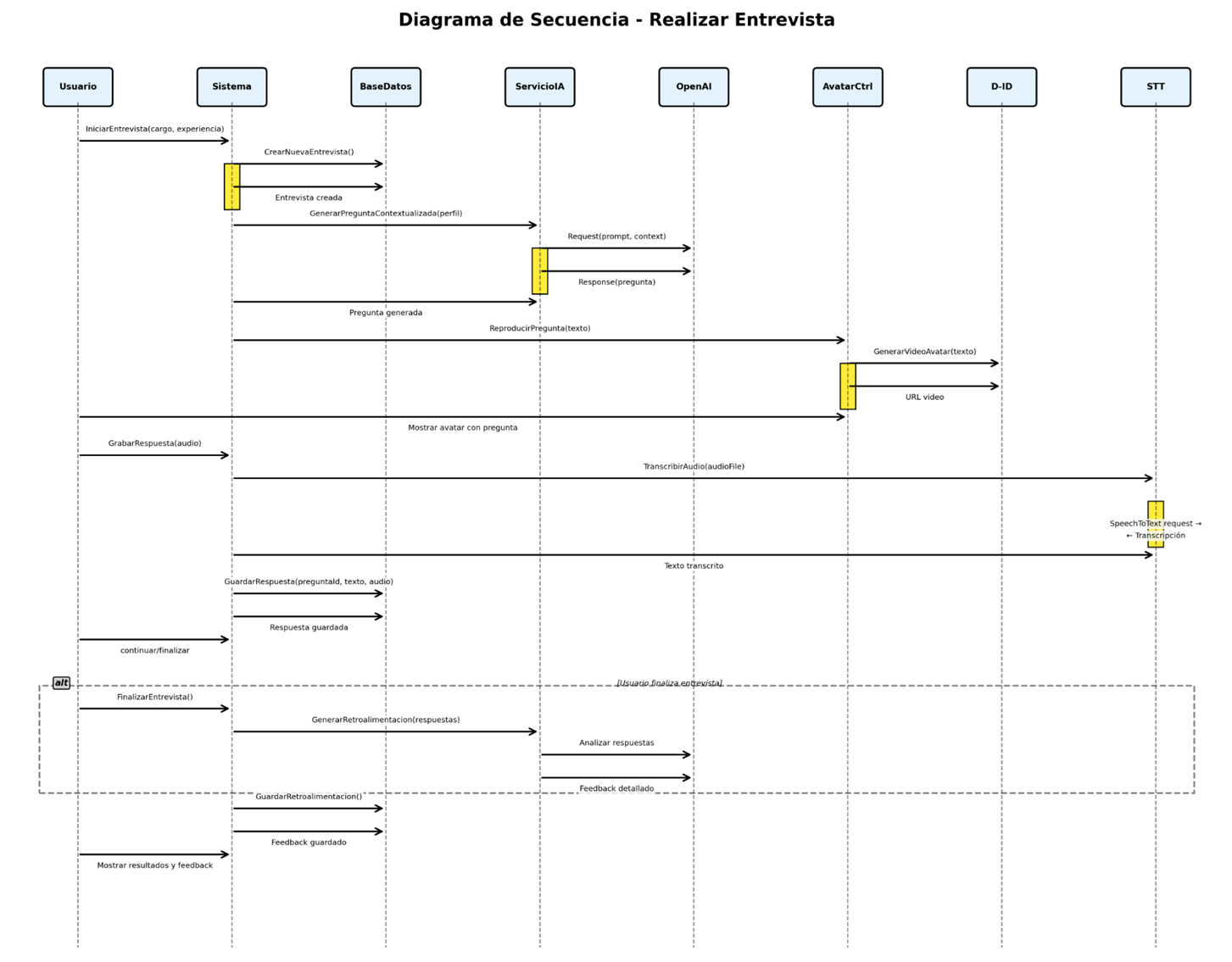
# 5. Vista Lógica

A continuación se presenta la vista lógica de InterviewAI, expresada mediante diagramas de clases principales, diagramas de secuencia y comunicación.

## 5.1 Diagrama de Clases Principal



## 5.2 Diagrama de Secuencia: Realizar Entrevista



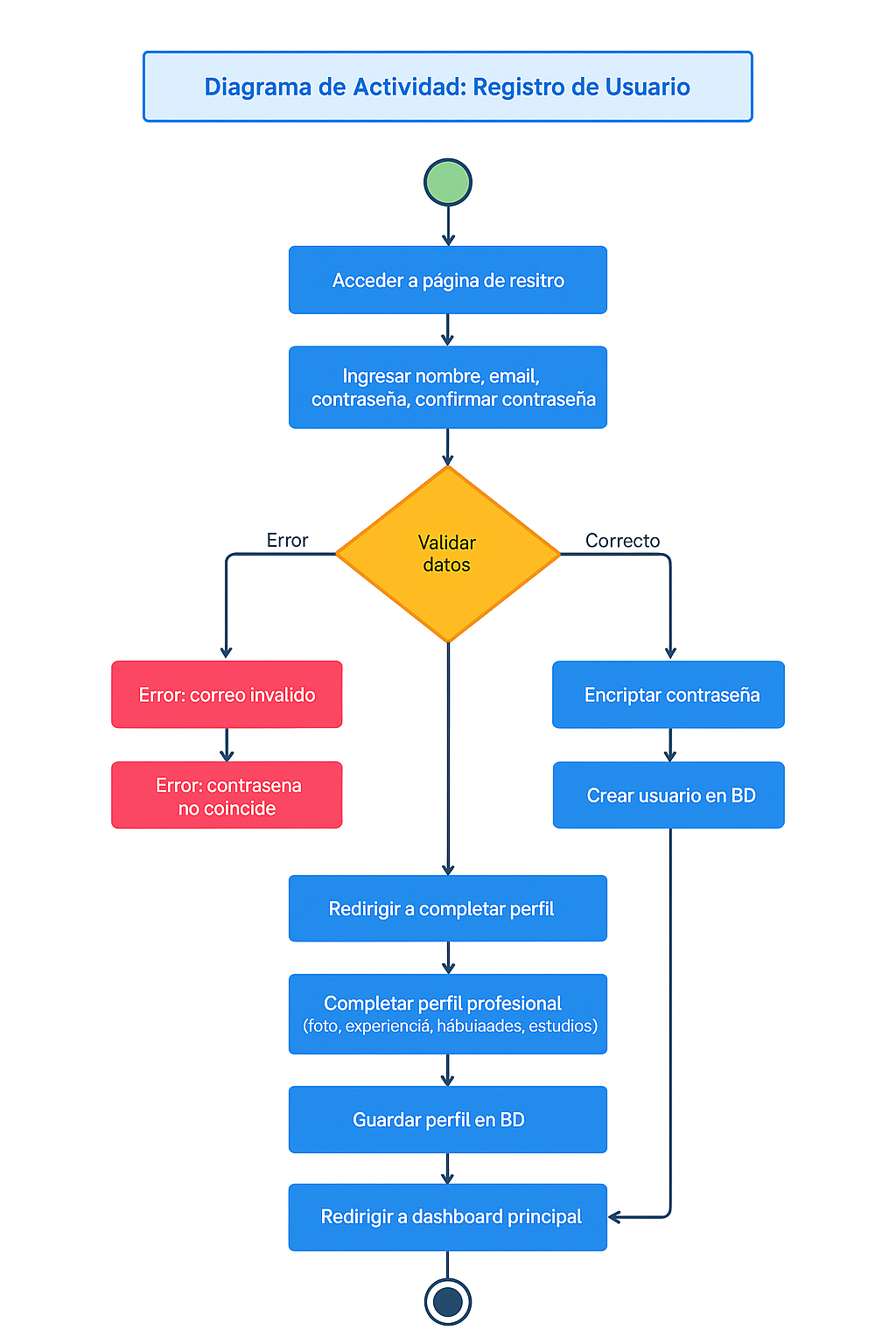
# 6. Vista de Procesos

Esta vista describe los principales procesos del sistema y su flujo de ejecución.

## 6.1 Proceso: Realizar Entrevista Completa



## 6.2 Proceso: Registro de Usuario



# 7. Vista de Implementación

Esta vista describe la estructura de implementación del sistema, incluyendo módulos principales y sus dependencias.

## 7.1 Arquitectura General

InterviewAI implementa una arquitectura cliente-servidor de tres capas:

**Capa de Presentación (Frontend)**

* Tecnología: React 18
* Librerías: React Router, Axios, TailwindCSS
* Estado: Context API + Hooks
* Deployment: Vercel

**Capa de Lógica de Negocio (Backend)**

* Tecnología: Node.js 18 + Express.js
* Arquitectura: RESTful API
* Autenticación: JWT + bcrypt
* ORM: Sequelize
* Deployment: Railway / Render

**Capa de Datos**

* Tecnología: PostgreSQL 15
* ORM: Sequelize
* Backups: Automáticos diarios
* Deployment: Railway / Supabase

## 7.2 Servicios Externos Integrados

**OpenAI GPT-4**

Propósito: Generación de preguntas contextualizadas y análisis de respuestas

Modelo: gpt-4-turbo

Rate Limit: 500 requests/minuto

**Google Cloud Speech-to-Text**

Propósito: Transcripción de audio de respuestas

Idiomas: Español (es-CL, es-ES), Inglés (en-US)

Formato audio: WebM, WAV

**D-ID Avatar API**

Propósito: Generación de avatar conversacional con sincronización labial

Avatar: Profesional masculino/femenino

Voz: Elevenlabs integration

## 7.3 Estructura de Módulos

**Frontend (React)**

/src

/components

/auth: Login, Register, ProtectedRoute

/interview: InterviewRoom, Avatar, QuestionDisplay, AudioRecorder

/dashboard: UserDashboard, InterviewHistory, Stats

/profile: ProfileForm, ProfileView

/feedback: FeedbackDisplay, FeedbackCard

/common: Button, Modal, Loading, ErrorBoundary

/contexts: AuthContext, InterviewContext

/services: api.js, auth.js, interview.js

/utils: validators.js, formatters.js, constants.js

/pages: Home, Login, Register, Dashboard, Interview, Profile, History

**Backend (Node.js/Express)**

/src

/controllers: authController, userController, interviewController

/models: User, Profile, Interview, Question, Response, Feedback

/routes: authRoutes, userRoutes, interviewRoutes

/services

openaiService.js: Integración con OpenAI GPT-4

sttService.js: Integración con Google Speech-to-Text

avatarService.js: Integración con D-ID

/middleware: auth.js, errorHandler.js, rateLimiter.js

/config: database.js, constants.js

/utils: validators.js, logger.js

# 8. Vista de Despliegue

Esta vista describe la configuración de despliegue del sistema y la infraestructura física/virtual.

## 8.1 Arquitectura de Despliegue

El sistema se despliega en una infraestructura cloud distribuida:

**Nodo 1: Cliente (Navegador Web)**

* Componentes: Aplicación React (SPA)
* Requisitos: Navegador moderno (Chrome, Firefox, Safari, Edge)
* Conectividad: HTTPS

**Nodo 2: Servidor Frontend (Vercel)**

* Componentes: Build estático de React, CDN global
* Características: SSL automático, edge caching, DDoS protection
* Ubicación: Global (múltiples regiones)

**Nodo 3: Servidor Backend (Railway/Render)**

* Componentes: Node.js runtime, Express server
* Características: Auto-scaling, health checks, rolling deployments
* Recursos: 512MB RAM, 0.5 CPU (escalable)
* Ubicación: US East (Virginia) / EU West

**Nodo 4: Base de Datos (PostgreSQL)**

* Componentes: PostgreSQL 15, pgAdmin (opcional)
* Características: Backups automáticos diarios, conexiones SSL, replicación
* Storage: 1GB inicial (escalable a 10GB)
* Ubicación: Co-located con backend

**Nodo 5: Servicios Externos**

* OpenAI API (api.openai.com)
* Google Cloud Speech-to-Text (speech.googleapis.com)
* D-ID Avatar API (api.d-id.com)

## 8.2 Flujo de Comunicación entre Nodos

1. **Cliente ↔ Frontend Server (Vercel)**

Protocolo: HTTPS

Puerto: 443

Datos: Archivos estáticos (HTML, CSS, JS)

2. **Cliente ↔ Backend Server**

Protocolo: HTTPS / WebSocket (para avatar)

Puerto: 443

Datos: JSON (requests/responses), audio files

Autenticación: JWT Bearer token

3. **Backend Server ↔ PostgreSQL**

Protocolo: PostgreSQL protocol over SSL

Puerto: 5432

Datos: SQL queries, transacciones

4. **Backend Server ↔ OpenAI API**

Protocolo: HTTPS

Datos: JSON (prompts, completions)

Autenticación: API Key

5. **Backend Server ↔ Google Cloud STT**

Protocolo: HTTPS / gRPC

Datos: Audio files (WebM), transcripciones (JSON)

Autenticación: Service Account credentials

6. **Backend Server ↔ D-ID Avatar API**

Protocolo: HTTPS

Datos: Texto, configuración avatar, video URLs

Autenticación: API Key

Diagrama

El contenido generado por IA puede ser incorrecto.

# 9. Diagrama de Componentes

El diagrama de componentes muestra los módulos principales del sistema y sus relaciones:

**Componentes Frontend**

* Módulo Autenticación: Login, Registro, Gestión de sesión
* Módulo Perfil: Visualización y edición de perfil profesional
* Módulo Entrevista: Sala de entrevista, avatar, grabación
* Módulo Dashboard: Estadísticas, historial, métricas
* Módulo Retroalimentación: Visualización de feedback y análisis

**Componentes Backend**

* API Gateway: Punto de entrada, routing, rate limiting
* Módulo Auth: Autenticación JWT, validación de usuarios
* Módulo User: CRUD usuarios y perfiles
* Módulo Interview: Gestión de entrevistas, preguntas, respuestas
* Módulo IA: Integración con OpenAI, generación de contenido
* Módulo STT: Integración con Google Speech-to-Text
* Módulo Avatar: Integración con D-ID, gestión de videos
* Data Access Layer: Repositorios, ORM Sequelize

**Componentes de Datos**

* Base de Datos PostgreSQL: Almacenamiento persistente
* Esquema Users: Datos de usuarios y autenticación
* Esquema Profiles: Información profesional
* Esquema Interviews: Sesiones de entrevista
* Esquema Questions: Banco de preguntas
* Esquema Responses: Respuestas de usuarios
* Esquema Feedback: Retroalimentación generada

# 10. Decisiones de Diseño y Selección de Alternativas

## 10.1 Selección de Arquitectura Cliente-Servidor

**Decisión**: Arquitectura cliente-servidor de tres capas con frontend SPA.

**Justificación**:

* Separación clara de responsabilidades entre presentación, lógica y datos
* Escalabilidad independiente de cada capa
* Frontend SPA proporciona experiencia fluida sin recargas
* Facilita mantenimiento y testing por capas

**Alternativas consideradas**:

* Arquitectura monolítica: Descartada por limitaciones de escalabilidad
* Microservicios: Demasiado complejo para el alcance y equipo actual

## 10.2 Selección de Stack Tecnológico

**Decisión**: React + Node.js/Express + PostgreSQL

**Justificación**:

* JavaScript full-stack facilita desarrollo y mantenimiento
* React ofrece componentes reutilizables y estado reactivo
* Node.js/Express proporciona alto rendimiento para I/O intensivo
* PostgreSQL ofrece robustez, ACID, y excelente para datos relacionales
* Expertise del equipo en estas tecnologías

**Alternativas consideradas**:

* Vue.js: Menor ecosistema que React
* Python/Django: Menos adecuado para real-time features
* MongoDB: Menos apropiado para datos estructurados y relacionales

## 10.3 Patrón de Autenticación JWT

**Decisión**: JSON Web Tokens (JWT) con refresh tokens

**Justificación**:

* Stateless: No requiere almacenamiento de sesión en servidor
* Escalable: Múltiples instancias de backend sin sincronización
* Seguro: Tokens firmados, expiración corta (15 min)
* Refresh tokens permiten sesiones largas sin comprometer seguridad

**Alternativas consideradas**:

* Sesiones del servidor: Requiere state, complica escalado
* OAuth2 Third-party: Agrega complejidad innecesaria

## 10.4 Estrategia de Integración con APIs Externas

**Decisión**: Capa de abstracción para servicios externos con retry logic

**Justificación**:

* Abstracción permite cambiar proveedores sin afectar lógica de negocio
* Retry con backoff exponencial maneja fallos temporales
* Circuit breaker previene cascada de fallos
* Manejo centralizado de rate limits y cuotas

**Patrones implementados**:

* Retry Pattern: Máximo 3 intentos con backoff exponencial
* Circuit Breaker: Abre tras 5 fallos consecutivos
* Timeout: 10 segundos para OpenAI, 30 segundos para D-ID

## 10.5 Diseño de Base de Datos Relacional

**Decisión**: Modelo relacional normalizado con PostgreSQL

**Justificación**:

* Datos altamente relacionales (usuarios, perfiles, entrevistas, respuestas)
* ACID compliance crítico para integridad de datos
* Normalización reduce redundancia y mantiene consistencia
* PostgreSQL ofrece tipos de datos avanzados (JSONB para metadata)

**Optimizaciones**:

* Índices en campos frecuentemente consultados (userId, entrevistaId)
* JSONB para almacenar metadata flexible
* Particionamiento por fecha para tabla de respuestas

## 10.6 Gestión de Estado en Frontend: Context API vs Redux

**Decisión**: Context API con React Hooks para gestión de estado global

**Justificación**:

* Estado global relativamente simple (usuario autenticado, sesión de entrevista activa, configuración)
* Redux agregaría complejidad innecesaria para un equipo de 3 personas en 18 semanas
* Context API es nativo de React, sin dependencias adicionales
* Menor curva de aprendizaje para el equipo

**Trade-offs reconocidos:**

**Desventaja**: Potencial de re-renders innecesarios con Context API

**Mitigación**:

* Separación de contextos independientes (AuthContext, InterviewContext)
* Uso de useMemo y useCallback para optimizar renders
* Memoización de componentes costosos con React.memo

**Plan de contingencia**: Si se identifican problemas de performance en testing, migración a Zustand (librería más ligera que Redux)

**Alternativas consideradas:**

* Redux: Demasiado boilerplate, excesivo para el alcance
* Zustand: Opción viable, pero Context API es suficiente inicialmente
* Recoil: Experimental, menor estabilidad en producción

## 10.7 Almacenamiento de Tokens JWT

**Decisión**: Access tokens en memoria + Refresh tokens en cookies httpOnly

**Justificación:**

* Seguridad: Previene ataques XSS al no exponer tokens en localStorage
* Usabilidad: Mantiene sesiones activas sin reautenticación constante
* Balance: Access tokens de corta vida (15 min) limitan ventana de exposición

**Estrategia de rotación:**

* Cada uso de refresh token genera uno nuevo (Refresh Token Rotation)
* Invalidación de refresh tokens antiguos después de usar
* Máximo 1 refresh token activo por dispositivo

**Protección contra CSRF:**

* Cookies con flag sameSite=strict
* Validación de origen en backend
* Token CSRF adicional para operaciones críticas (cambio de contraseña, eliminación de cuenta)

**Alternativas consideradas:**

* localStorage: Vulnerable a XSS, descartado
* sessionStorage: Se pierde al cerrar tab, mala UX
* Cookies para todo: Vulnerable a CSRF sin medidas adicionales

## 10.8 Estrategia de Fallback para APIs Externas

**Decisión**: Sistema de múltiples niveles de fallback con banco de preguntas pre-generadas

**Problema identificado**: OpenAI, Google Cloud y D-ID son dependencias críticas. Sus fallos (timeout, rate limit, downtime) pueden detener completamente el servicio.

**Solución implementada:**

**Nivel 1:** Retry con Backoff Exponencial

// Configuración de reintentos

const retryConfig = {

maxRetries: 3,

initialDelay: 1000, // 1 segundo

maxDelay: 8000, // 8 segundos

backoffMultiplier: 2

};

**Secuencia de reintentos:**

1. Primer intento → Fallo → Espera 1s
2. Segundo intento → Fallo → Espera 2s
3. Tercer intento → Fallo → Espera 4s
4. Activar fallback

**Nivel 2: Circuit Breaker**

const circuitBreakerConfig = {

failureThreshold: 5, *// Abre tras 5 fallos consecutivos*

timeout: 60000, *// 60 segundos abierto*

resetTimeout: 30000 *// 30 segundos en half-open*

};

**Estados:**

* **Cerrado:** Sistema normal, permite llamadas
* **Abierto:** Tras 5 fallos consecutivos, bloquea llamadas durante 60s
* **Semi-abierto:** Tras 60s, permite 1 llamada de prueba
  + Si éxito → Cerrado
  + Si fallo → Abierto nuevamente

**Nivel 3: Banco de Preguntas Pre-generadas**

**Implementación:**

* **250-300 preguntas curadas por área técnica y nivel**
* **Almacenadas en PostgreSQL**
* **Categorizadas por:** 
  + **Área: Frontend, Backend, Full-Stack, DevOps, QA, Data Science**
  + **Nivel: Junior, Semi-Senior, Senior**
  + **Tipo: Técnica, Comportamental, Situacional**

**Estructura de datos:**

CREATE TABLE fallback\_questions (

id SERIAL PRIMARY KEY,

question TEXT NOT NULL, area VARCHAR(50) NOT NULL,

level VARCHAR(20) NOT NULL,

type VARCHAR(30) NOT NULL,

difficulty\_score INT CHECK (difficulty\_score BETWEEN 1 AND 10),

created\_at TIMESTAMP DEFAULT NOW()

);

**Lógica de selección:**

* Si OpenAI disponible → Pregunta generada por IA
* Si OpenAI falla → Pregunta aleatoria del banco que coincida con perfil de usuario
* Rotación inteligente: No repetir preguntas de las últimas 3 entrevistas

**Nivel 4: Provider Alternativo (Futuro)**

**Plan para v2.0:**

* Integración con Google Gemini como proveedor secundario
* Switching automático si OpenAI falla consistentemente
* Configuración por variable de entorno

**Trade-offs:**

* Ventaja: Sistema funcional incluso con APIs caídas
* Desventaja: Preguntas del banco son menos contextualizadas que las generadas por IA
* Compromiso aceptado: Es mejor una experiencia con preguntas genéricas que ninguna experiencia

## 10.9 Decisión de No Almacenar Audio

**Decisión:** Transcribir audio en tiempo real y descartar archivos de audio después de procesamiento

**Justificación:**

**Razones técnicas:**

1. **Costo de almacenamiento:**

* Promedio 50-100 MB por entrevista de 10 minutos
* 1000 entrevistas = 50-100 GB
* Costo mensual estimado: $5-10 USD (significativo para proyecto académico)

1. **Privacidad:**

* Usuarios pueden sentirse incómodos con grabaciones permanentes de su voz
* Menor riesgo de filtración de información sensible
* Cumplimiento más sencillo con regulaciones de privacidad

1. **Rendimiento:**

* Transferencia de archivos grandes afecta latencia
* Procesamiento en tiempo real más eficiente

**Información capturada:**

* Transcripción completa (texto)
* Duración de respuesta
* Timestamp de respuesta
* Palabras clave identificadas
* ❌ Tono de voz
* ❌ Pausas y titubeos
* ❌ Velocidad de habla
* ❌ Entonación

**Limitaciones reconocidas:** No se pueden analizar aspectos paraverbales críticos en entrevistas:

* Seguridad/confianza transmitida por el tono
* Nerviosismo detectado por pausas excesivas
* Claridad de expresión oral vs escrita
* Uso excesivo de "muletillas" (eh, este, o sea)

**Plan para versión futura (v2.0):**

***// Análisis en tiempo real sin almacenamiento permanente***

const audioAnalytics = {

speechRate: calculateWordsPerMinute(audioBuffer),

pauseDuration: detectPauses(audioBuffer),

fillerWords: countFillerWords(transcription),

confidenceScore: analyzeTone(audioBuffer)

};

***// Guardar solo métricas, no audio***

await saveMetrics(audioAnalytics);

discardAudio(audioBuffer);

**Alternativa considerada pero descartada:**

* Almacenamiento temporal (7 días): Agrega complejidad de garbage collection, costos similares
* Análisis diferido: Requiere almacenar audio, contradice objetivo de privacidad

**Compromiso aceptado:** Priorizar privacidad y costos sobre análisis avanzado en v1.0. Funcionalidad completa planificada para v2.0 con análisis en tiempo real.

## 10.10 Componentes de Código Reutilizables

**Desafío:** SLA de 500ms de latencia para sincronización de avatar con audio

**Análisis realista de latencia:**

* Latencia de red del usuario: 50-300ms (variable, fuera de control)
* Tiempo de procesamiento OpenAI: 800-2500ms (generación de pregunta)
* Tiempo de procesamiento D-ID: 200-600ms (generación de video)
* Latencia de red hacia D-ID: 50-150ms
* Buffering del navegador: 100-200ms

**Latencia total realista:** 1200-3750ms en condiciones normales

**Reconocimiento de SLA no alcanzable:** El objetivo inicial de 500ms fue basado en expectativas optimistas. La latencia real promedio será de **2-3 segundos**.

**Estrategias de mitigación implementadas:**

**1. Precarga y Pipeline Asíncrono**

javascript

*// Mientras usuario responde pregunta N, precargar pregunta N+1*

async function pipelineOptimization() {

*// Usuario respondiendo pregunta actual*

const nextQuestionPromise = generateNextQuestion(); *// Inicia precarga*

await waitForUserResponse();

*// Pregunta siguiente ya lista (ahorro de 2s)*

const nextQuestion = await nextQuestionPromise;

displayQuestion(nextQuestion);

}

**Ahorro:** Hasta 2 segundos en transiciones entre preguntas

**2. Indicadores Visuales de Progreso**

const loadingStates = {

thinking: "🤔 El entrevistador está pensando...",

generating: "📝 Generando pregunta personalizada...",

preparing: "🎬 Preparando avatar...",

ready: "✅ ¡Pregunta lista!"

};

**Beneficio: Usuario percibe espera como procesamiento valioso, no como fallo**

**3.Optimización de Payload:**

* Compresión de video: Usar formato WebM con VP9 (30% más pequeño que H.264)
* Resolución adaptativa: 720p en conexiones buenas, 480p en lentas
* Streaming progresivo: Iniciar reproducción antes de descarga completa

**4.Fallback a Texto:**

javascript

const TIMEOUT\_AVATAR = 3000; *// 3 segundos*

const avatarPromise = generateAvatar(question);

const result = await Promise.race([

avatarPromise,

timeout(TIMEOUT\_AVATAR)

]);

if (!result) {

*// Fallback: mostrar texto + voz sin avatar*

displayTextQuestion(question);

playTextToSpeech(question);

}

**Condiciones de fallback:**

* Latencia > 3 segundos
* D-ID API no disponible
* Conexión del usuario < 2 Mbps detectada

**5. Caché de Avatars Comunes**

javascript

*// Cachear videos de avatar para preguntas frecuentes*

const commonQuestionCache = new Map();

if (isCommonQuestion(question)) {

const cachedAvatar = commonQuestionCache.get(question.id);

if (cachedAvatar) return cachedAvatar; *// Latencia: ~100ms*

}

**Preguntas cacheables:**

* Preguntas de apertura estándar ("Cuéntame sobre ti")
* Preguntas de cierre ("¿Tienes preguntas para mí?")
* Top 20 preguntas más frecuentes por área

**Ahorro: Hasta 2.5 segundos en preguntas comunes**

**Métricas de éxito ajustadas:**

* (Objetivo original no realista)
* 80% de transiciones en < 3 segundos (Objetivo realista)
* 95% con indicador visual de progreso (Experiencia aceptable)
* Fallback funcional en 100% de timeouts (Resiliencia garantizada)

**Trade-off aceptado:** Sacrificar latencia mínima por experiencia consistente y feedback visual claro. Es mejor un sistema con 2-3 segundos predecibles que uno que aspira a 500ms pero falla frecuentemente.

# 11. Análisis de Reutilización

## 11.1 Componentes de Código Reutilizables

Los siguientes componentes han sido diseñados con reutilización en mente:

**Frontend - Componentes React**

* Button, Input, Modal, Card: Componentes UI reutilizables en toda la app
* AudioRecorder: Reutilizable para cualquier funcionalidad de grabación
* ProtectedRoute: Reutilizable para cualquier ruta autenticada
* ErrorBoundary: Manejo de errores aplicable globalmente

**Backend - Módulos de Servicio**

* authMiddleware: Reutilizable para cualquier endpoint protegido
* errorHandler: Manejo centralizado de errores
* validators: Funciones de validación reutilizables
* logger: Sistema de logging centralizado

## 11.2 Patrones de Diseño Aplicados

**Repository Pattern**

Abstrae acceso a datos, permitiendo cambiar ORM o DB sin afectar lógica.

**Service Layer Pattern**

Encapsula lógica de negocio, facilitando testing y reutilización.

**Factory Pattern**

Creación de objetos complejos (entrevistas, feedback) de manera consistente.

**Adapter Pattern**

Adaptadores para APIs externas, permitiendo cambiar proveedores fácilmente.

## 11.3 Consideraciones para Migración Futura

El diseño actual facilita futuras migraciones y ampliaciones:

**Escalado a Microservicios**

Módulos actuales (Auth, Interview, IA) pueden extraerse a servicios independientes sin modificar contratos de API.

**Migración de Proveedores IA**

Capa de abstracción permite cambiar de OpenAI a alternativas (Anthropic, Cohere) modificando solo servicios.

**Internacionalización**

Strings externalizados facilitan agregar soporte multiidioma sin refactorizar código.

**Nuevos Tipos de Entrevista**

Sistema modular permite agregar nuevas categorías (entrevistas de comportamiento, técnicas específicas) sin afectar funcionalidad existente.

# 12. Análisis de Riesgos Técnicos y Estrategias de Mitigación

## 12.1 Introducción

Este apartado identifica los riesgos técnicos más significativos del proyecto InterviewAI, analiza su probabilidad e impacto, y establece estrategias concretas de mitigación. El análisis se realizó considerando las restricciones de tiempo (18 semanas), presupuesto (créditos educativos) y equipo (3 desarrolladores).

## 12.2 Matriz de Riesgos

| **ID** | **Riesgo** | **Probabilidad** | **Impacto** | **Severidad** | **Estrategia** |
| --- | --- | --- | --- | --- | --- |
| **R1** | Fallo crítico de APIs externas (OpenAI, D-ID) | Alta | Alto | CRÍTICO | Fallback multi-nivel |
| **R2** | Superación de cuotas gratuitas de APIs | Media | Alto | CRÍTICO | Monitoreo + límites |
| **R3** | Latencia excesiva en generación de avatar | Alta | Medio | ALTO | Optimización + fallback |
| **R4** | Inconsistencia en feedback de IA | Media | Alto | ALTO | Prompts estructurados |
| **R5** | Vulnerabilidad de seguridad en autenticación | Baja | Crítico | ALTO | Buenas prácticas |
| **R6** | Problemas de escalabilidad con 100+ usuarios | Media | Medio | MEDIO | Arquitectura escalable |
| **R7** | Desviación del cronograma de 18 semanas | Media | Alto | ALTO | Metodología ágil |
| **R8** | Incompatibilidad cross-browser | Baja | Bajo | BAJO | Testing multi-navegador |

## 12.3 Riesgos Críticos y Mitigación Detallada

**R1: Fallo Crítico de APIs Externas**

**Descripción del riesgo:** OpenAI, Google Cloud Speech-to-Text y D-ID son servicios de terceros fuera de nuestro control. Pueden experimentar:

* Downtime no planificado
* Rate limiting inesperado
* Cambios en términos de servicio
* Deprecación de endpoints

**Probabilidad:** Alta (70-80%)

* APIs cloud tienen uptime típico de 99.5-99.9%, lo que significa 43-216 minutos de downtime mensual
* Durante desarrollo, es casi seguro experimentar al menos un incidente

**Impacto:** Alto

* Sin OpenAI: No se pueden generar preguntas contextualizadas
* Sin Google STT: No se pueden transcribir respuestas
* Sin D-ID: No hay avatar conversacional

**Resultado:** Sistema completamente inoperable si falla OpenAI

**Estrategias de mitigación:**

1. **Sistema de fallback en cascada (Descrito en 10.8)**

* Retry con backoff exponencial
* Circuit breaker
* Banco de preguntas pre-generadas
* Provider alternativo planificado

1. **Monitoreo proactivo**

// Healthcheck de APIs cada 5 minutos

const apiHealthMonitor = {

openai: checkOpenAIStatus(),

googleSTT: checkGoogleSTTStatus(),

dID: checkDIDStatus()

};

// Alertas automáticas

if (anyAPIDown(apiHealthMonitor)) {

notifyTeam("API down detected");

activateFallbackMode();

}

1. **Degradación elegante del servicio**

* Prioridad 1: Funcionalidad core (preguntas y respuestas)
* Prioridad 2: Características avanzadas (avatar, análisis detallado)
* Prioridad 3: Nice-to-have (gamificación, estadísticas avanzadas)

1. **Documentación de procedimientos**

* Runbook con pasos exactos ante fallo de cada API
* Contactos de soporte de cada proveedor
* Scripts de activación de fallback

**Métricas de éxito:**

* Sistema sigue funcional (modo degradado) durante downtime de APIs
* Tiempo medio de detección de fallo < 2 minutos
* Activación automática de fallback < 30 segundos

**R2: Superación de Cuotas Gratuitas**

**Descripción del riesgo:** Las APIs tienen límites en tiers gratuitos o créditos estudiantiles:

* OpenAI: $5-10 USD créditos gratis → ~200-400 entrevistas
* Google Cloud: $300 créditos iniciales → ~5000 minutos transcripción
* D-ID: 20 créditos gratis → ~20 videos de prueba

**Probabilidad:** Media (40-60%)

* Durante testing intensivo, es probable agotar créditos
* 10 testers haciendo 5 entrevistas diarias = 50 entrevistas/día
* Créditos agotados en 4-8 días de testing

**Impacto:** Alto

* Interrupción completa del desarrollo
* Imposibilidad de demostrar sistema en presentación final
* Necesidad de inversión económica no planificada

**Estrategias de mitigación:**

1. **Sistema de cuotas por usuario**

javascript

*// Límites durante testing*

const quotas = {

testing: {

interviewsPerDay: 3,

interviewsPerWeek: 10

},

production: {

interviewsPerDay: 5,

interviewsPerMonth: 30

}

};

1. **Monitoreo en tiempo real de costos**

javascript

*// Dashboard de costos*

const costMonitor = {

openai: {

used: calculateTokensUsed(),

remaining: calculateTokensRemaining(),

estimatedCost: calculateCost()

},

alerts: {

threshold50: notifyAt50Percent(),

threshold80: notifyAt80Percent()

}

};

1. **Optimización de prompts**

* Reducir tokens en prompts a OpenAI (de 500 a 200 tokens promedio)
* Usar GPT-3.5-turbo para testing, GPT-4 solo para demo
* Cachear respuestas de preguntas comunes

1. **Mock services para desarrollo**

javascript

*// Modo desarrollo sin APIs reales*

if (process.env.NODE\_ENV === 'development') {

useOpenAIMock(); *// Respuestas predefinidas*

useSTTMock(); *// Transcripciones fake*

useDIDMock(); *// Avatares de prueba*

}

1. **Plan de contingencia financiera**

* Reserva de $50 USD para compra de créditos adicionales si necesario
* Solicitud de créditos educativos adicionales a proveedores
* Fundraising interno del equipo si crítico

**Métricas de éxito:**

* Costos mensuales < $10 USD durante desarrollo
* Créditos suficientes hasta presentación final
* Alertas activadas al 50% y 80% de cuota

**R4: Inconsistencia en Feedback de IA**

**Descripción del riesgo:** GPT-4 es un modelo de lenguaje probabilístico. Dos respuestas idénticas pueden recibir feedback diferente, afectando:

* Credibilidad del sistema
* Experiencia de usuario
* Validez pedagógica del feedback

**Probabilidad:** Media (40-50%)

* Modelos de lenguaje tienen inherente variabilidad
* Sin prompts estructurados, variabilidad alta

**Impacto:** Alto

* Usuarios pierden confianza en el sistema
* Feedback contradictorio es peor que no tener feedback
* Puede perpetuar sesgos del modelo

**Estrategias de mitigación:**

1. **Prompts estructurados con formato de salida definido**

javascript

const feedbackPrompt = `

Eres un evaluador experto de entrevistas técnicas. Analiza la siguiente respuesta según los criterios EXACTOS:

RESPUESTA DEL CANDIDATO:

"${userResponse}"

PREGUNTA REALIZADA:

"${question}"

**CRITERIOS DE EVALUACIÓN (escala 1-10):**

1. CLARIDAD: ¿La respuesta es coherente y fácil de entender?

2. PROFUNDIDAD TÉCNICA: ¿Demuestra conocimiento sólido del tema?

3. ESTRUCTURA: ¿Usa metodología STAR (Situación, Tarea, Acción, Resultado)?

4. EJEMPLOS CONCRETOS: ¿Proporciona ejemplos específicos y relevantes?

5. RELEVANCIA: ¿Responde directamente a la pregunta?

Responde ÚNICAMENTE en este formato JSON:

{

"puntuaciones": {

"claridad": <número 1-10>,

"profundidad\_tecnica": <número 1-10>,

"estructura": <número 1-10>,

"ejemplos\_concretos": <número 1-10>,

"relevancia": <número 1-10>

},

"puntuacion\_total": <promedio de las 5 métricas>,

"fortalezas": [

"<fortaleza 1>",

"<fortaleza 2>"

],

"areas\_mejora": [

"<área 1>",

"<área 2>"

],

"sugerencias\_concretas": [

"<sugerencia accionable 1>",

"<sugerencia accionable 2>"

],

"ejemplo\_respuesta\_mejorada": "<versión mejorada de la respuesta>"

}

NO incluyas texto fuera del JSON. NO uses markdown. Responde solo JSON válido.

`;

1. **Validación de estructura de respuesta**

javascript

function validateFeedback(apiResponse) {

const required = ['puntuaciones', 'fortalezas', 'areas\_mejora'];

const parsed = JSON.parse(apiResponse);

for (const field of required) {

if (!parsed[field]) {

throw new Error(`Missing required field: ${field}`);

}

}

*// Validar rangos numéricos*

for (const score of Object.values(parsed.puntuaciones)) {

if (score < 1 || score > 10) {

throw new Error(`Score out of range: ${score}`);

}

}

return parsed;

}

1. **Testing de consistencia**

javascript

*// Generar feedback 5 veces para misma respuesta*

async function testConsistency(question, response) {

const feedbacks = [];

for (let i = 0; i < 5; i++) {

feedbacks.push(await generateFeedback(question, response));

}

*// Calcular desviación estándar de puntuaciones*

const avgScores = calculateAverageScores(feedbacks);

const stdDev = calculateStdDev(feedbacks);

console.log(`Average score: ${avgScores}`);

console.log(`Std deviation: ${stdDev}`); *// Debe ser < 1.0*

}

1. **Parámetros de modelo para reducir variabilidad**

javascript

const openaiConfig = {

model: "gpt-4-turbo",

temperature: 0.3, *// Baja temperatura = más determinístico*

top\_p: 0.8, *// Reduce variabilidad adicional*

frequency\_penalty: 0,

presence\_penalty: 0

};

1. **Banco de respuestas de referencia**

* 20-30 respuestas "gold standard" por área técnica
* Feedback experto humano pre-validado
* Comparación de feedback de IA vs experto humano
* Ajuste de prompts basado en discrepancias

**Métricas de éxito:**

* Desviación estándar de puntuaciones < 1.0 para misma respuesta
* 90% de feedbacks generan JSON válido al primer intento
* Correlación > 0.85 entre feedback de IA y experto humano

**R5: Vulnerabilidad de Seguridad**

**Descripción del riesgo:** Sistema maneja datos sensibles de usuarios (correos, contraseñas, grabaciones de voz). Posibles vulnerabilidades:

* SQL Injection
* XSS (Cross-Site Scripting)
* CSRF (Cross-Site Request Forgery)
* Exposición de tokens JWT
* Man-in-the-middle attacks
* Brute force en login

**Probabilidad:** Baja (15-20%)

* Con buenas prácticas, riesgo bajo
* Sin experiencia extensa en seguridad, riesgo aumenta

**Impacto:** Crítico

* Filtración de datos personales
* Acceso no autorizado a cuentas
* Daño reputacional irreparable
* Posibles implicaciones legales (Ley 19.628 Chile)

**Estrategias de mitigación:**

1. **Encriptación de contraseñas con bcrypt**

javascript

const bcrypt = require('bcrypt');

const SALT\_ROUNDS = 12; *// Factor de costo alto*

async function hashPassword(plainPassword) {

return await bcrypt.hash(plainPassword, SALT\_ROUNDS);

}

async function verifyPassword(plainPassword, hashedPassword) {

return await bcrypt.compare(plainPassword, hashedPassword);

}

1. **Prevención de SQL Injection con ORM**

javascript

*// NUNCA hacer esto*

const query = `SELECT \* FROM users WHERE email = '${userEmail}'`;

*// SIEMPRE usar ORM con parámetros*

const user = await User.findOne({

where: { email: userEmail }

});

1. **Sanitización de inputs**

javascript

const validator = require('validator');

function sanitizeInput(input) {

return validator.escape(input.trim());

}

function validateEmail(email) {

return validator.isEmail(email);

}

1. **Headers de seguridad HTTP**

javascript

const helmet = require('helmet');

app.use(helmet());

*// Content Security Policy*

app.use(helmet.contentSecurityPolicy({

directives: {

defaultSrc: ["'self'"],

scriptSrc: ["'self'", "'unsafe-inline'"],

styleSrc: ["'self'", "'unsafe-inline'"],

imgSrc: ["'self'", "data:", "https:"],

}

}));

1. **Rate limiting contra brute force**

javascript

const rateLimit = require('express-rate-limit');

const loginLimiter = rateLimit({

windowMs: 15 \* 60 \* 1000, *// 15 minutos*

max: 5, *// máximo 5 intentos*

message: "Demasiados intentos de login. Intenta en 15 minutos."

});

app.post('/api/auth/login', loginLimiter, loginController);

1. **HTTPS obligatorio**

javascript

*// Redirect HTTP to HTTPS*

app.use((req, res, next) => {

if (!req.secure && process.env.NODE\_ENV === 'production') {

return res.redirect(`https://${req.headers.host}${req.url}`);

}

next();

});

1. **Auditoría de dependencias**

bash

*# Ejecutar semanalmente*

npm audit

npm audit fix

*# Actualizar dependencias con vulnerabilidades*

npm update

1. **Testing de seguridad**

* OWASP ZAP scan automatizado
* Penetration testing básico antes de producción
* Code review enfocado en seguridad

**Métricas de éxito:**

* 0 vulnerabilidades críticas en npm audit
* 100% de contraseñas encriptadas
* 100% de tráfico sobre HTTPS
* Rate limiting activo en endpoints sensibles

**R7: Desviación del Cronograma**

**Descripción del riesgo:** Proyecto de 18 semanas con alcance ambicioso. Riesgos de retraso:

* Complejidad técnica subestimada
* Bloqueos por dependencias externas
* Enfermedad o ausencia de miembros del equipo
* Scope creep (agregar funcionalidades no planificadas)

**Probabilidad:** Media (50-60%)

* 18 semanas es ajustado para el alcance
* Primer proyecto de esta envergadura para el equipo

**Impacto:** Alto

* No cumplir deadline académico
* Presentación con funcionalidad incompleta
* Estrés y burnout del equipo

**Estrategias de mitigación:**

1. **Metodología ágil con sprints de 2 semanas**

Sprint 1-2: Setup inicial, autenticación básica

Sprint 3-4: Generación de preguntas con OpenAI

Sprint 5-6: Integración Speech-to-Text

Sprint 7-8: Sistema de feedback

Sprint 9-10: Avatar conversacional

Sprint 11-12: Dashboard y métricas

Sprint 13-14: Testing y bug fixes

Sprint 15-16: Optimización y pulido

Sprint 17-18: Documentación y presentación

1. **Priorización MoSCoW**

* **Must have:** Auth, preguntas IA, transcripción, feedback básico
* **Should have:** Avatar, dashboard, historial
* **Could have:** Gamificación, estadísticas avanzadas
* **Won't have (v1.0):** Multi-idioma, integración LinkedIn, app móvil

1. **Buffer time (20% del cronograma)**

Desarrollo planificado: 14 semanas (78% del tiempo)

Buffer: 4 semanas (22% del tiempo)

Total: 18 semanas

1. **Reuniones diarias de seguimiento**

* Daily standup de 15 minutos
* ¿Qué hice ayer?
* ¿Qué haré hoy?
* ¿Tengo bloqueos?

1. **Métricas de progreso semanales**

* Velocity de sprints (story points completados)
* Burndown chart
* Identificación temprana de desviaciones

1. **Plan de contingencia por feature**

| Feature | Tiempo Estimado | Fallback si retraso | |---|---|---| | Avatar D-ID | 3 semanas | Usar solo voz sin video | | Feedback IA avanzado | 2 semanas | Feedback basado en keywords | | Dashboard con gráficos | 2 semanas | Lista simple de entrevistas |

**Métricas de éxito:**

* Velocity estable entre sprints (±20%)
* 90% de features "Must have" completados en semana 14
* Buffer no completamente agotado al final

## 12.4 Plan de Acción y Responsabilidades

| **Riesgo** | **Responsable Primario** | **Revisión** | **Frecuencia Monitoreo** |
| --- | --- | --- | --- |
| R1: APIs externas | Cristian (AI Specialist) | Matías | Diaria |
| R2: Cuotas APIs | Matías (Project Leader) | Equipo completo | Semanal |
| R3: Latencia avatar | Ulises (Frontend) | Cristian | Sprint review |
| R4: Feedback IA | Cristian (AI Specialist) | Profesor guía | Cada iteración |
| R5: Seguridad | Matías (Backend) | Auditoría externa | Pre-producción |
| R6: Escalabilidad | Matías (Backend) | Testing de carga | Pre-producción |
| R7: Cronograma | Matías (Project Leader) | Profesor guía | Semanal |
| R8: Cross-browser | Ulises (Frontend) | Testing manual | Sprint review |

## 12.5 Conclusión del Análisis de Riesgos

Este análisis de riesgos ha identificado 8 riesgos técnicos significativos, de los cuales 2 son críticos (R1, R2) y 3 son de alta prioridad (R3, R4, R5, R7). Cada riesgo cuenta con estrategias de mitigación concretas, métricas de éxito y responsables asignados.

El equipo de InterviewAI se compromete a:

1. Monitorear activamente todos los riesgos identificados
2. Actualizar este documento cuando se identifiquen nuevos riesgos
3. Ejecutar los planes de contingencia de manera oportuna
4. Comunicar proactivamente al profesor guía sobre riesgos materializados

**Próximos pasos:**

* Implementar sistema de monitoreo de APIs (Sprint 2)
* Configurar alertas de costos (Sprint 2)
* Realizar pruebas de consistencia de IA (Sprint 4)
* Auditoría de seguridad (Sprint 12)
* Testing de carga (Sprint 14)

# 13. Conclusión

Este Documento de Arquitectura del Sistema (DAS) ha presentado la arquitectura completa de InterviewAI, describiendo las decisiones técnicas, patrones de diseño y estructura del sistema.

**La arquitectura propuesta cumple con los objetivos del proyecto:**

* Provee una solución escalable y mantenible
* Garantiza seguridad y privacidad de datos de usuarios
* Facilita integración con servicios de IA externos
* Permite futuras ampliaciones y migraciones

El equipo de InterviewAI está comprometido con mantener este documento actualizado a medida que el proyecto evoluciona, asegurando que refleje fielmente la implementación real del sistema.