**Resumen Frontend**

1. \*\*Autenticación y Registro:\*\*

- Sistema de login (`Login.tsx`)

- Sistema de registro (`Register.tsx`)

2. \*\*Navegación:\*\*

- Sistema de navegación con tabs

- Rutas protegidas y públicas

- Manejo de layouts con `\_layout.tsx`

3. \*\*Funcionalidades para Profesores:\*\*

- Vista principal del profesor (`ProfesorHome.tsx`)

- Creación de quizzes (`CrearQuizz.tsx`)

4. \*\*Funcionalidades para Alumnos:\*\*

- Vista principal del alumno (`AlumnoHome.tsx`)

- Vista de preguntas (`VistaPreguntas.tsx`)

- Acceso a quizzes específicos (`quizId.tsx`)

5. \*\*Estructura del Proyecto:\*\*

- Organización en carpetas (apis, components, context, hooks, etc.)

- Configuración de TypeScript

- Sistema de estilos y temas

- Manejo de estado global

\*\*Lo que parece estar incompleto o falta por implementar:\*\*

1. \*\*Gestión de Cursos:\*\*

- Aunque hay carpetas `cursoHome` y `cursoalumno`, podrían necesitar más funcionalidades

- Falta implementación completa de gestión de cursos

2. \*\*Funcionalidades de Quiz:\*\*

- Sistema de calificaciones

- Estadísticas y reportes

- Historial de intentos

3. \*\*Características Adicionales:\*\*

- Sistema de notificaciones

- Perfiles de usuario completos

- Recuperación de contraseña

- Validaciones más robustas

4. \*\*Mejoras Técnicas:\*\*

- Testing (no se ve configuración de tests)

- Documentación más detallada

- Optimización de rendimiento

- Manejo de errores más robusto

5. \*\*UI/UX:\*\*

- Posiblemente faltan animaciones y transiciones

- Mejoras en la accesibilidad

- Temas oscuro/claro

- Diseño responsive más completo

El proyecto parece ser una aplicación educativa con roles de profesor y alumno, construida con React Native y Expo, utilizando TypeScript. Tiene una base sólida con las funcionalidades principales implementadas, pero hay espacio para expandir y mejorar varias características.

**Microservicio Usuarios**

1. **Framework: FastAPI**

* **Framework moderno de Python para APIs**
* **Soporte nativo para async/await**
* **Documentación automática con Swagger/OpenAPI**

1. **Base de Datos: SQLAlchemy**

* **ORM (Object-Relational Mapping)**
* **Manejo de sesiones y transacciones**
* **Migraciones de base de datos**

1. **Autenticación: JWT (JSON Web Tokens)**

* **Implementación usando la biblioteca python-jose**
* **Tokens con información del usuario y rol**
* **Sistema de OAuth2 para protección de rutas**

1. \*\*Autenticación (`/auth`)\*\*:

- `POST /auth/register`: Registro de nuevos usuarios

- `POST /auth/login`: Inicio de sesión

- `GET /auth/validate-token`: Validación de tokens JWT

2. \*\*Usuarios (`/users`)\*\*:

- `GET /users/me`: Obtener información del usuario actual

3. \*\*Sistema\*\*:

- `GET /health`: Verificación de estado del servicio

Integración con el Frontend:

El microservicio se integra con el frontend de la siguiente manera:

1. \*\*Autenticación\*\*:

- El frontend debe hacer una petición POST a `/auth/login` con las credenciales del usuario

- Recibe un token JWT que debe almacenar y usar en todas las peticiones subsecuentes

- El token contiene información del usuario como ID, email, nombre y rol

2. \*\*Validación de Sesión\*\*:

- El frontend puede validar el token en cualquier momento usando `/auth/validate-token`

- Esto es útil para mantener la sesión activa y verificar permisos

3. \*\*Información del Usuario\*\*:

- El frontend puede obtener la información del usuario actual usando `/users/me`

- Esta información incluye datos básicos y el rol del usuario

### Funcionalidades Pendientes o Mejoras Posibles:

1. \*\*Gestión de Usuarios\*\*:

- No hay endpoints para actualizar información del usuario

- No hay endpoints para eliminar usuarios

- No hay endpoints para listar usuarios (útil para administradores)

2. \*\*Gestión de Roles\*\*:

- No hay endpoints para gestionar roles

- No hay endpoints para asignar/cambiar roles a usuarios

3. \*\*Recuperación de Contraseña\*\*:

- No hay implementación de recuperación de contraseña

- No hay verificación de email

4. \*\*Seguridad\*\*:

- El CORS está configurado para permitir todos los orígenes (`"\*"`), lo cual no es recomendable para producción

- No hay rate limiting implementado

- No hay validación de complejidad de contraseñas

### Recomendaciones:

1. Implementar endpoints para:

- Actualización de perfil de usuario

- Recuperación de contraseña

- Gestión de roles

- Listado de usuarios (para administradores)

2. Mejorar la seguridad:

- Configurar CORS específicamente para los dominios permitidos

- Implementar rate limiting

- Agregar validación de contraseñas

- Implementar verificación de email

3. Documentación:

- Agregar documentación más detallada de los endpoints

- Incluir ejemplos de uso

- Documentar los códigos de error

**Microservicio Cursos**

### Arquitectura del Microservicio

- \*\*Framework\*\*: FastAPI

- \*\*Base de datos\*\*: SQLAlchemy (ORM)

- \*\*Autenticación\*\*: Sistema de autenticación implementado (get\_current\_user)

### Endpoints Disponibles

#### Cursos (`/cursos`)

1. `POST /cursos/`

- Crea un nuevo curso

- Requiere autenticación

- Body: CursoBase

- Response: CursoOut

2. `GET /cursos/`

- Lista todos los cursos disponibles

- No requiere autenticación

- Response: List[CursoOut]

3. `GET /cursos/{curso\_id}/existe`

- Verifica si existe un curso específico

- No requiere autenticación

- Response: boolean

#### Inscripciones (`/inscripciones`)

1. `POST /inscripciones/inscribirse/{codigo}`

- Permite inscribirse a un curso usando un código

- Requiere autenticación

- Response: InscripcionOut

2. `GET /inscripciones/mis-inscripciones`

- Lista los cursos en los que está inscrito el usuario

- Requiere autenticación

- Response: List[InscripcionOut]

### Aspectos Técnicos Importantes

1. \*\*Seguridad\*\*:

- Implementación de autenticación mediante `get\_current\_user`

- Endpoints protegidos que requieren autenticación

2. \*\*Base de Datos\*\*:

- Uso de SQLAlchemy como ORM

- Inicialización automática de la base de datos al inicio de la aplicación

3. \*\*Validación de Datos\*\*:

- Uso de schemas (Pydantic) para validación de entrada/salida

- Modelos definidos para la base de datos

### Integración con Frontend

Según el documento adjunto, este microservicio se integra con:

1. \*\*Frontend\*\*: Aplicación web que consume estos endpoints para:

- Mostrar listado de cursos

- Permitir inscripciones

- Gestionar cursos del usuario

2. \*\*API Gateway\*\*: Actúa como punto de entrada único para todas las peticiones

### Elementos No Implementados o Incompletos

1. \*\*Gestión de Cursos\*\*:

- No hay endpoints para actualizar cursos

- No hay endpoints para eliminar cursos

- No hay endpoints para obtener detalles específicos de un curso

2. \*\*Gestión de Inscripciones\*\*:

- No hay endpoints para cancelar inscripciones

- No hay endpoints para verificar el estado de una inscripción

3. \*\*Validaciones Adicionales\*\*:

- No se observa validación de cupos en cursos

- No hay manejo de fechas límite para inscripciones

### Recomendaciones de Mejora

1. Implementar endpoints CRUD completos para cursos

2. Agregar endpoints para gestión de inscripciones

3. Implementar validaciones de negocio (cupos, fechas, etc.)

4. Agregar documentación de API más detallada

5. Implementar manejo de errores más robusto

6. Agregar tests unitarios y de integración

Microservicio Evaluaciones

Este es un microservicio de evaluaciones que forma parte de un sistema más grande de gestión de cursos. Se integra con otros servicios como el de usuarios y cursos.

### Endpoints Principales

#### 1. Gestión de Preguntas (`/preguntas`)

- `POST /`: Crear pregunta manualmente (profesor)

- `POST /generar-con-ia/`: Generar preguntas usando IA

- `POST /guardar-seleccionadas/`: Guardar preguntas generadas por IA

- `GET /`: Obtener preguntas por tema

#### 2. Gestión de Quizzes (`/quizzes`)

- `POST /`: Crear nuevo quiz (profesor)

- `GET /{quiz\_id}/pregunta`: Obtener pregunta actual del quiz (alumno)

- `POST /{quiz\_id}/respuesta`: Guardar respuesta y avanzar (alumno)

- `GET /activos-programados/{curso\_id}`: Listar quizzes activos/programados

#### 3. Gestión de Respuestas (`/respuestas`)

- `POST /evaluar`: Evaluar respuestas del alumno

- `GET /{quiz\_id}/estadisticas`: Ver estadísticas del quiz (profesor)

### Sistema de Generación de Preguntas

El sistema utiliza OpenAI (GPT-3.5-turbo) para generar preguntas automáticamente:

- Permite especificar tema y cantidad de preguntas

- Genera preguntas de opción múltiple con 4 opciones

- Incluye explicación de la respuesta correcta

- Maneja errores de cuota y conexión con la API

### Integración con Otros Servicios

1. \*\*Servicio de Usuarios\*\*:

- Validación de tokens de autenticación

- Verificación de roles (profesor/alumno)

- URL configurable mediante variable de entorno

2. \*\*Servicio de Cursos\*\*:

- Verificación de existencia de cursos

- Integración para validar IDs de cursos

- URL configurable mediante variable de entorno

### Base de Datos

Utiliza MongoDB con las siguientes colecciones:

- `preguntas`: Almacena las preguntas

- `quizzes`: Almacena los quizzes

- `respuestas`: Almacena las respuestas de los alumnos

### Características Destacadas

1. \*\*Sistema de Progreso\*\*:

- Seguimiento de respuestas por alumno

- Control de preguntas respondidas

- Cálculo de porcentaje de avance

2. \*\*Estadísticas\*\*:

- Promedio de puntuación por quiz

- Identificación de preguntas problemáticas

- Total de alumnos por quiz

3. \*\*Seguridad\*\*:

- Validación de roles

- Verificación de tokens

- Control de acceso por tipo de usuario

### Endpoints Pendientes o Quemados

- El sistema de progreso está parcialmente implementado (usa variables en memoria)

- La función `guardar\_respuesta\_pregunta` tiene valores hardcodeados que deberían venir de la base de datos

### Integración con Frontend

El servicio está diseñado para ser consumido por un frontend que:

1. Permite a profesores:

- Crear quizzes

- Generar preguntas con IA

- Ver estadísticas

- Gestionar preguntas

2. Permite a alumnos:

- Ver quizzes activos

- Responder preguntas

- Ver su progreso

- Recibir resultados

### Recomendaciones

1. Implementar el almacenamiento persistente del progreso

2. Mejorar el manejo de errores en las respuestas

3. Agregar más validaciones de datos

4. Implementar un sistema de caché para optimizar consultas frecuentes

### 1. Servicio de Autenticación (`auth.py`)

- Maneja la autenticación y autorización de usuarios

- Integra con un servicio externo de usuarios para validar tokens

- Verifica la existencia de cursos en el sistema

- Proporciona middleware para proteger rutas

### 2. Servicio de Quizzes (`quiz\_service.py`)

Funcionalidades principales:

- Creación de quizzes con título, tema, preguntas y fechas

- Gestión del progreso de los alumnos durante el quiz

- Obtención de quizzes activos y programados

- Manejo de respuestas pregunta por pregunta

- Control de estados de los quizzes (activo, programado, finalizado)

### 3. Servicio de Preguntas (`question\_service.py`)

- Creación de preguntas individuales

- Almacenamiento en base de datos MongoDB

- Gestión de preguntas con opciones múltiples

- Asociación de preguntas con cursos específicos

### 4. Servicio de IA (`ia\_service.py`)

Características avanzadas:

- Generación automática de preguntas usando GPT-3.5

- Generación de feedback personalizado para respuestas incorrectas

- Manejo de errores y límites de la API de OpenAI

- Personalización de preguntas por tema

### 5. Servicio de Respuestas (`respuesta\_service.py`)

Funcionalidades:

- Evaluación de respuestas de alumnos

- Cálculo de puntuaciones

- Generación de estadísticas para profesores

- Identificación de preguntas problemáticas

- Almacenamiento de historial de respuestas

### Integración con el Frontend

La aplicación se integra con el frontend a través de endpoints REST bien definidos:

1. \*\*Endpoints de Quizzes\*\*:

- `POST /quizzes/` - Crear nuevo quiz

- `GET /quizzes/{quiz\_id}/pregunta` - Obtener pregunta actual

- `POST /quizzes/{quiz\_id}/respuesta` - Guardar respuesta

- `GET /quizzes/activos-programados/{curso\_id}` - Listar quizzes disponibles

2. \*\*Endpoints de Preguntas\*\*:

- `POST /preguntas/` - Crear pregunta manual

- `POST /preguntas/generar-con-ia/` - Generar preguntas con IA

- `POST /preguntas/guardar-seleccionadas/` - Guardar preguntas generadas

3. \*\*Endpoints de Respuestas\*\*:

- `POST /respuestas/evaluar` - Evaluar respuestas completas

- `GET /respuestas/{quiz\_id}/estadisticas` - Obtener estadísticas

### Características Técnicas

1. \*\*Base de Datos\*\*:

- Utiliza MongoDB para almacenamiento

- Colecciones principales: quizzes, preguntas, respuestas

2. \*\*Seguridad\*\*:

- Autenticación basada en tokens

- Validación de roles (profesor/alumno)

- Verificación de permisos por endpoint

3. \*\*Integración con Servicios Externos\*\*:

- Servicio de Usuarios

- Servicio de Cursos

- API de OpenAI para generación de contenido

4. \*\*Manejo de Errores\*\*:

- Validaciones robustas

- Mensajes de error descriptivos

- Manejo de excepciones específicas

Esta arquitectura permite una separación clara de responsabilidades y facilita la integración con el frontend, proporcionando una API RESTful bien estructurada y documentada.