Documentación Tutorlink-Conexión back-front

Documentación para el Frontend del Proyecto Tutorlink

Nombre del Proyecto: Tutorlink
Versión: 1.0
Backend: Saúl Angulo Triana (Cayn)
Fecha: 19 de noviembre de 2025

---

1. Introducción

Este documento está diseñado para guiar al equipo de frontend en la integración con el backend del proyecto **Tutorlink**. Proporciona una referencia clara sobre los endpoints de la API REST, las estructuras de datos (DTOs), el flujo de autenticación y las mejores prácticas para consumir los servicios del backend.

El objetivo es que el frontend pueda desarrollar una interfaz de usuario rica e interactiva, aprovechando todas las funcionalidades del sistema sin necesidad de un conocimiento profundo de la implementación interna del backend.

---

2. Tecnologías del Backend

- **Framework Principal:** Spring Boot 3.3.5
- **Base de Datos:** PostgreSQL
- **Modelo de IA:** Ollama
- **Despliegue:** Amazon Web Services (AWS EC2)
- **Lenguaje:** Java 17
- **IDE:** Visual Studio Code

---

3. Arquitectura General del Backend

El backend sigue una arquitectura de **4 capas**:

1. **Controlador:** Recibe solicitudes HTTP.
2. **Servicio:** Coordina operaciones.
3. **Negocio:** Contiene toda la lógica de negocio (validación de reglas, integración con Ollama).
4. **Repositorio:** Acceso a la base de datos.

Esto garantiza que la API REST sea estable, segura y escalable.

---

4. Flujo de Trabajo Clave

Entender estos flujos te ayudará a diseñar la UI:

1. **Inicio de Sesión:**
   - El usuario ingresa correo y contraseña.
   - El frontend envía una solicitud POST a `/api/auth/login`.
   - Si es exitoso, recibe un token JWT.
   - Este token debe almacenarse (por ejemplo, en `localStorage` o `sessionStorage`) y enviarse en cada solicitud posterior.
2. **Creación de Pregunta:**
   - Un estudiante escribe una pregunta y selecciona un alcance (GENERAL, FACULTAD, PLAN).
   - Al enviar, el frontend llama a `POST /api/preguntas`.
   - El backend guarda la pregunta como "PENDIENTE" y activa el flujo de generación de respuesta por el LLM.
3. **Flujo de Revisión (Tutor):**
   - Un tutor inicia sesión y ve un panel con preguntas pendientes de sus tutorados (`GET /api/tutores/{id}/pendientes`).
   - Puede aprobar (`POST /api/respuestas/{id}/aprobar`) o rechazar (`POST /api/respuestas/{id}/rechazar`) una respuesta.
   - Si se aprueba, la pregunta y la respuesta se vuelven visibles para el estudiante.
   - Si se rechaza, el LLM genera una nueva versión para revisar.

---

5. Autenticación y Autorización

La autenticación se maneja mediante tokens JWT (JSON Web Token).

- **Login:**
  - Endpoint: `POST /api/auth/login`
  - Body: `{ "correo": "usuario@dominio.com", "contrasena": "contraseña" }`
  - Respuesta: `{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." }`
  - **Acción del Frontend:** Guarda el token en el almacenamiento local.
- **Solicitudes Protegidas:**
  - Todas las solicitudes a endpoints bajo `/api/**` (excepto `/auth/login` y `/auth/register`) requieren el header:
    ```
    Authorization: Bearer <tu_token_aqui>
    ```
  - Si el token expira o no es válido, el backend responde con un error `401 Unauthorized`.
- **Obtener Información del Usuario:**
  - Endpoint: `GET /api/auth/me`
  - Header: `Authorization: Bearer <token>`
  - Respuesta: `{ "username": "Ana Pérez López", "authorities": ["ESTUDIANTE"] }`

---

6. API REST - Endpoints Principales

Todos los endpoints están bajo la URL base: `http://localhost:8080/api` (en desarrollo) o la URL de producción en AWS.

Autenticación y Registro

Método	Endpoint	Descripción
`POST`	`/auth/register`	Registra un nuevo usuario (estudiante).
`POST`	`/auth/login`	Inicia sesión y obtiene un JWT.
`GET`	`/auth/me`	Obtiene el perfil del usuario autenticado.

Preguntas

Método	Endpoint	Parámetros	Descripción
`POST`	`/preguntas`	Body: JSON con `titulo`, `texto`, `scope_tipo`, `id_facultad_scope` (opcional), `id_plan_educativo_scope` (opcional).	Crea una nueva pregunta.
`GET`	`/preguntas`	Query: `estado`, `scope_tipo`, `id_facultad_scope`, `id_plan_educativo_scope`, `page`, `size`, `sort`.	Lista preguntas con filtros y paginación.
`GET`	`/preguntas/sugerencias?q={texto}`	Query: `q` (texto de búsqueda).	Busca preguntas existentes similares a `texto`.

Respuestas

Método	Endpoint	Descripción
`POST`	`/respuestas/{id}/aprobar`	Aprobar una respuesta específica.
`POST`	`/respuestas/{id}/rechazar`	Rechazar una respuesta específica (genera una nueva versión).

Usuarios y Perfil

Método	Endpoint	Descripción
`PUT`	`/usuarios/{id}/foto`	Actualiza la foto de perfil del usuario con ID `{id}`.
`GET`	`/usuarios/tutores/{id}/tutorados`	Lista todos los estudiantes asignados como tutorados al tutor con ID `{id}`.
`GET`	`/usuarios/tutores/{id}/tutorados/buscar?q={query}`	Busca tutorados por nombre/apellidos.
`GET`	`/usuarios/tutores/{id}/tutorados/filtrar?idPlan=&semestre=`	Filtra tutorados por plan educativo y/o semestre.

Administración (Solo ADMIN)

Método	Endpoint	Descripción
`POST`	`/admin/usuarios/{id}/aprobar`	Aprueba el registro de un usuario.
`POST`	`/admin/estudiantes/{idEst}/asignar-tutor/{idTutor}`	Asigna un tutor a un estudiante.
`DELETE`	`/admin/preguntas/{id}`	Elimina una pregunta.

---

7. Estructuras de Datos (DTOs)

El backend utiliza DTOs para enviar y recibir datos. Aquí tienes ejemplos clave:

LoginRequest

```
{
  "correo": "string",
  "contrasena": "string"
}
```

UserDTO

```
{
  "idUsuario": 1,
  "nombre": "Ana",
  "apellidos": "Pérez López",
  "correo": "ana@uv.mx",
  "rol": "ESTUDIANTE",
  "fotoPerfilUrl": "https://example.com/foto.jpg",
  "telefono": "+521234567890"
}
```

QuestionDTO (Crear/Actualizar)

```
{
  "titulo": "Duda sobre herencia",
  "texto": "¿Cómo funciona super()?",
  "scopeTipo": "FACULTAD",
  "idFacultadScope": 1
}
```

QuestionSummaryDto (Lista Paginada)

```
{
  "idPregunta": 5,
  "titulo": "Duda sobre herencia",
  "estado": "PENDIENTE",
  "fechaCreacion": "2025-11-19T10:00:00",
  "autor": {
    "nombre": "Ana",
    "apellidos": "Pérez López"
  }
}
```

Page<> Response (para listados)

```
{
  "content": [...], // Array de objetos (e.g., QuestionSummaryDto)
  "totalElements": 25,
  "totalPages": 3,
  "number": 0,
  "size": 10,
  "first": true,
```

```
  "last": false
}
```

8. Errores Comunes y Manejo

El backend devuelve respuestas de error estandarizadas. Ejemplos comunes:

- `400 Bad Request`: Validación fallida (campos vacíos, formato incorrecto).
- `401 Unauthorized`: Token JWT faltante, inválido o expirado.
- `403 Forbidden`: El usuario no tiene permiso para realizar la acción (por ejemplo, un estudiante intenta aprobar una respuesta).
- `404 Not Found`: El recurso solicitado no existe (por ejemplo, ID de pregunta no encontrado).
- `409 Conflict`: Conflicto de negocio (por ejemplo, correo ya registrado).
- `500 Internal Server Error`: Error interno del servidor.

Consejo para el Frontend: Muestra mensajes de error amigables al usuario basados en el código de estado HTTP.

---

9. Pruebas Locales

Para probar el backend localmente:

1. Asegúrate de tener instalado Java 17 y Maven.
2. Navega a la carpeta `backend/`.
3. Ejecuta `mvn spring-boot:run` para iniciar el servidor.
4. La aplicación estará disponible en `http://localhost:8080`.

Nota sobre Docker: Aunque el README menciona Docker, el despliegue final será directamente en AWS EC2. Para desarrollo local, puedes usar el comando Maven anterior.

---

10. Conexión con el Modelo de IA (Ollama)

- El backend se comunica automáticamente con Ollama cuando se crea una pregunta.
- No necesitas llamar directamente a Ollama desde el frontend.
- El flujo es: Frontend → Backend → Ollama → Backend → Base de Datos → Frontend (cuando el tutor aprueba).

---

11. Próximos Pasos para el Frontend

1. Implementar el formulario de login y registro.
2. Crear la página principal para crear y listar preguntas.
3. Diseñar el panel de tutor para revisar respuestas.
4. Implementar el perfil de usuario y gestión de foto.
5. Integrar notificaciones (visuales en la UI, complementarias al correo).

¡Buena suerte! Estoy disponible para resolver cualquier duda sobre la API.