Documentación backend Tutorlink- Welcome

Documentación del Backend del Proyecto Tutorlink

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

---

1. Introducción

Este documento proporciona una descripción detallada y completa de la arquitectura, diseño e implementación del backend para el proyecto **Tutorlink**, una plataforma de tutorías académicas que utiliza inteligencia artificial generativa para mejorar la experiencia educativa.

El sistema permite a los estudiantes realizar preguntas sobre temas específicos, las cuales son respondidas por un modelo de lenguaje (LLM). Las respuestas generadas pasan por un proceso de revisión realizado por tutores asignados antes de ser publicadas. Los administradores tienen herramientas para gestionar usuarios, preguntas y el flujo de registro.

Esta documentación tiene como objetivo servir como referencia para desarrolladores, testers y otros miembros del equipo, explicando la estructura del código, las decisiones de diseño, la API REST y el flujo de trabajo del backend.

---

2. Tecnologías Utilizadas

- **Lenguaje de Programación:** Java 17
- **Framework Principal:** Spring Boot 3.3.5
- **Base de Datos:** PostgreSQL
- **Modelo de IA:** Ollama
- **Despliegue:** Amazon Web Services (AWS EC2)
- **Entorno de Desarrollo:** Visual Studio Code
- **Control de Versiones:** GitHub
- **Dependencias Clave:**
  - `spring-boot-starter-web`: Para servicios web REST.
  - `spring-boot-starter-data-jpa`: Para acceso a datos con JPA/Hibernate.
  - `spring-boot-starter-security`: Para autenticación y autorización.
  - `spring-boot-starter-mail`: Para envío de notificaciones por correo.
  - `io.jsonwebtoken` (jjwt): Para la generación y validación de tokens JWT.
  - `org.projectlombok`: Para reducir código boilerplate.
  - `org.modelmapper`: Para mapeo de objetos entre entidades y DTOs.
- **IDE:** Visual Studio Code (con extensión Java y Lombok).

---

3. Arquitectura General

El backend sigue una arquitectura de 4 capas clásica, promoviendo la separación de responsabilidades y facilitando el mantenimiento y testing:

1. Capa de Controlador (`controller/`): Recibe solicitudes HTTP del frontend (React), valida datos de entrada, autentica al usuario mediante JWT y delega la lógica de negocio a la capa de servicio.
2. Capa de Servicio (`service/`): Coordina las operaciones entre la capa de controlador y la capa de negocio. Actúa como intermediario, delegando la lógica específica del dominio a la capa de negocio.
3. Capa de Negocio (`business/`): Contiene toda la lógica de negocio central del sistema. Aquí se validan reglas de negocio, se interactúa con los repositorios y se orquesta la integración con servicios externos (Ollama, Notificaciones).
4. Capa de Acceso a Datos (`repository/`): Se encarga exclusivamente de la persistencia y recuperación de datos en la base de datos PostgreSQL mediante Spring Data JPA.

---

4. Modelo de Datos

La estructura de la base de datos está diseñada para ser normalizada y escalable, cumpliendo con todos los requisitos funcionales del sistema.

Tablas Principales:

- `rol`: Define los tipos de usuario (`ESTUDIANTE`, `TUTOR`, `ADMIN`, `SUDO`, `LLM`).
- `usuario`: Tabla central que almacena a todos los usuarios del sistema (humanos y bots). Tiene una clave foránea a `rol`.
- `detalle_estudiante`, `detalle_tutor`, `detalle_admin`: Almacenan información específica de cada tipo de usuario, relacionada uno a uno con `usuario`.
- `pregunta`: Representa una consulta realizada por un estudiante. Tiene un estado (`PENDIENTE`, `PUBLICADA`, `RECHAZADA`) y un alcance definido (`GENERAL`, `FACULTAD`, `PLAN`).
- `respuesta`: Representa una respuesta a una pregunta, generada por el LLM. Puede tener múltiples versiones y un estado (`PENDIENTE_REVISION`, `PUBLICADA`).
- Tablas de Alcance (`region`, `area_academica`, `facultad`, `programa_educativo`, `plan_educativo`): Definen la jerarquía académica del sistema para permitir preguntas con alcances específicos.

Relaciones Clave:

- Un `Usuario` con rol `ESTUDIANTE` tiene un `DetalleEstudiante` asociado, que define su `PlanEducativo` y su `tutor_asignado`.
- Una `Pregunta` pertenece a un `autor` (estudiante) y puede estar asociada a una `Facultad` o `PlanEducativo` según su `scope_tipo`.
- Una `Respuesta` pertenece a una `Pregunta` y tiene un `autor` (el `LLM`).

---

5. Flujo de Trabajo Principal

El flujo principal del sistema es el siguiente:

1. Autenticación:
   - Un estudiante o tutor accede al sistema mediante `/api/auth/login`.
   - El backend valida credenciales contra la base de datos.

- Si son correctas, se genera un token JWT firmado y se devuelve al cliente.

2. **Creación de Pregunta:**
   - Un estudiante autenticado realiza una solicitud POST a `/api/preguntas` con el título, texto y alcance.
   - La solicitud incluye el token JWT en el header `Authorization`.
   - El `QuestionController` valida los datos y delega a `QuestionService`, que a su vez llama a `QuestionBusiness`.
   - `QuestionBusiness.crearPregunta()` valida el alcance (por ejemplo, si el estudiante pertenece al `PlanEducativo` especificado) y guarda la pregunta en estado `PENDIENTE`.
3. **Generación y Revisión de Respuesta:**
   - El sistema automáticamente invoca `AnswerBusiness.generarRespuestaConOllama()`.
   - Este método prepara un prompt basado en la pregunta y su alcance, y llama al `OllamaService` para obtener una respuesta del modelo de IA.
   - La respuesta generada se guarda en la base de datos con estado `PENDIENTE_REVISION`.
   - Se envía una notificación por correo al tutor asignado del estudiante (`RF-10`).
   - El tutor, al iniciar sesión, puede ver las preguntas pendientes de sus tutorados (`RF-11`).
   - El tutor aprueba (`aprobarRespuesta`) o rechaza (`rechazarYRegenerarRespuesta`) la respuesta.
   - Si se aprueba, la pregunta pasa a estado `PUBLICADA`, la respuesta a `PUBLICADA`, y se notifica al estudiante (`RF-09`).
   - Si se rechaza, se genera una nueva versión de la respuesta y vuelve a entrar en revisión.

---

6. API REST

La API expone endpoints para todas las funcionalidades del sistema. Todos los endpoints protegidos requieren un token JWT válido en el header `Authorization: Bearer <token>`.

Método	Endpoint	Autenticación	Descripción
`POST`	`/api/auth/register`	Pública	Registra un nuevo usuario (estudiante/tutor).
`POST`	`/api/auth/login`	Pública	Inicia sesión y devuelve un token JWT.
`GET`	`/api/auth/me`	Protegida	Devuelve información del usuario autenticado.
`POST`	`/api/preguntas`	Protegida	Crea una nueva pregunta.
`GET`	`/api/preguntas`	Protegida	Lista preguntas con filtros (estado, alcance, paginación).
`GET`	`/api/preguntas/sugerencias?q={texto}`	Pública	Busca preguntas existentes similares a `texto`.
`GET`	`/api/tutores/{id}/pendientes`	Protegida (Rol TUTOR)	Lista preguntas pendientes de revisión para el tutor con ID `{id}`.
`POST`	`/api/respuestas/{id}/aprobar`	Protegida (Rol TUTOR)	Aprueba una respuesta, haciendo pública la pregunta.
`POST`	`/api/respuestas/{id}/rechazar`	Protegida (Rol TUTOR)	Rechaza una respuesta, generando una nueva versión.
`PUT`	`/api/usuarios/{id}/foto`	Protegida	Actualiza la URL de la foto de perfil del usuario.

Método	Endpoint	Autenticación	Descripción
`GET`	`/api/usuarios/tutores/{id}/tutorados`	Protegida (Rol TUTOR)	Lista los estudiantes asignados como tutorados al tutor con ID `{id}`.
`POST`	`/api/admin/usuarios/{id}/aprobar`	Protegida (Rol ADMIN)	Aprueba el registro de un usuario.
`DELETE`	`/api/admin/preguntas/{id}`	Protegida (Rol ADMIN)	Elimina una pregunta.

(Para detalles completos de los cuerpos de solicitud y respuesta, consultar el código fuente o considerar integrar Swagger UI).

---

7. Seguridad

El sistema implementa medidas de seguridad robustas:

- **Autenticación con JWT:** Todos los accesos a recursos protegidos se validan mediante tokens JSON Web Token.
- **Autorización por Roles:** El sistema distingue entre roles (`ESTUDIANTE`, `TUTOR`, `ADMIN`). Spring Security y anotaciones como `@PreAuthorize` garantizan que solo los usuarios con el rol adecuado puedan ejecutar ciertas acciones.
- **Contraseñas Hashed:** Las contraseñas de los usuarios se almacenan en la base de datos utilizando hash (BCrypt), nunca en texto plano.
- **Validación de Entrada:** Se utilizan anotaciones de `jakarta.validation` (como `@NotBlank`, `@Email`) para validar los datos recibidos desde el frontend.
- **Protección CSRF:** Deshabilitada porque el sistema es stateless (no usa sesiones HTTP).
- **CORS:** Configurado para permitir solicitudes desde el origen del frontend.

---

8. Integración con Servicios Externos

- **Ollama:** El backend se comunica directamente con una instancia de Ollama a través de su API REST (`POST /api/generate`). El `OllamaService` maneja la construcción del payload, el envío de la solicitud y el procesamiento de la respuesta.
- **Notificaciones por Correo:** El `NotificationService` utiliza `JavaMailSender` para enviar correos electrónicos cuando ocurren eventos importantes (respuesta aprobada, nueva respuesta pendiente de revisión). Está configurado para usar cualquier servidor SMTP (Gmail, etc.).

---

9. Pruebas

El proyecto incluye pruebas de integración para verificar el comportamiento de los componentes principales (controladores, servicios) con una base de datos H2 en memoria.

- **Configuración:** Se utiliza el perfil de Spring `test` y un archivo `application-test.properties`.
- **Aislamiento:** La anotación `@DirtiesContext(classMode = AFTER_EACH_TEST_METHOD)` reinicia la base de datos en memoria después de cada prueba.
- **Ejecución:** `mvn test` desde la carpeta `backend/`.

---

10. Despliegue en AWS

El despliegue se realizará en una instancia gratuita de AWS EC2.

1. **Preparación:**
   - Empaquetar la aplicación: `mvn clean package`.
   - Generar un archivo `.jar` ejecutable (`tutorlink-backend.jar`).
2. **Configuración del Entorno:**
   - Crear una instancia RDS (PostgreSQL) y una instancia EC2 (Linux).
   - Configurar grupos de seguridad para permitir tráfico entre EC2 y RDS (puerto 5432) y tráfico HTTP/HTTPS (puertos 80/443) hacia EC2.
3. **Despliegue:**
   - Conectarse a la instancia EC2 via SSH.
   - Instalar Java 17 en la instancia.
   - Copiar el archivo `.jar` al servidor.
   - Configurar `application-prod.properties` con las credenciales de RDS y otras variables de entorno.
   - Ejecutar la aplicación: `java -jar tutorlink-backend.jar`.
4. **Producción:**
   - Considerar el uso de `systemd` para gestionar el proceso del backend.
   - Usar Nginx como servidor proxy inverso para servir el frontend y enrutar solicitudes al backend.

---

11. Conclusiones

El backend de Tutorlink ha sido diseñado e implementado siguiendo buenas prácticas de desarrollo de software, arquitectura limpia y estándares de seguridad. La separación clara de capas, la implementación de lógica de negocio bien definida y la integración fluida con Ollama crean una base sólida para una plataforma de tutorías innovadora.

El sistema está listo para ser consumido por el frontend React y para su despliegue en producción en AWS.