Registro de un nuevo tópico

La API debe contar con un *endpoint* (punto final) para el registro de tópicos, y debe aceptar solicitudes del tipo **POST** para la URI **/tópicos**.

Los datos del tópico (título, mensaje, autor y curso) deben ser enviados en el cuerpo de la solicitud, en formato JSON.

→ No olvides utilizar la anotación @RequestBody para que tu proyecto Spring reciba correctamente los datos del cuerpo de la solicitud.

→ Además, recuerda que el tópico debe ser guardado en la **base de datos** construida para el proyecto, así que aquí tienes el recordatorio de utilizar el método *save* del *JpaRepository* para realizar la **persistencia de los datos** del tópico creado.

**Sugerencia**: para ayudar en la validación de los datos, intenta utilizar la anotación Java integrada en Spring @Valid.

## Reglas de negocio

* Todos los campos son obligatorios, por lo tanto, es necesario verificar si todos los campos se están ingresando correctamente.
* La API no debe permitir el registro de tópicos duplicados (con el mismo título y mensaje).

Listado de tópicos

La API debe contar con un punto final para el listado de todos los tópicos, y debe aceptar solicitudes del tipo **GET** para la URI **/tópicos**.

Los datos de los tópicos (título, mensaje, fecha de creación, estado, autor y curso) deben ser devueltos en el cuerpo de la respuesta, en formato JSON.

→ Recordando que al tratar con el CRUD es necesario trabajar con JpaRepository asociado al tópico, especialmente en la lista de datos de la base de datos utilizamos el método findAll.

## Opcionales:

1. ¿Qué hay de mostrar los primeros 10 resultados ordenados por fecha de creación del tópico en orden ASC?
2. ¿Qué tal mostrar los resultados usando un criterio de búsqueda? Sugerimos listar por: nombre de curso y año específico.

**Otra opción opcional**: ¿Qué tal practicar el listado de resultados con paginación? Utilizando la anotación @PageableDefault en tu código para manejar el volumen de datos presentados al usuario.

**Detalle de tópicos**

La API debe contar con un *endpoint* (punto final) para el listado de todos los tópicos, y debe aceptar solicitudes del tipo **GET** para la URI **/tópicos/{id}**.

Los datos de los tópicos (título, mensaje, fecha de creación, estado, autor y curso) deben ser devueltos en el cuerpo de la respuesta, en formato JSON.

→ Recuerda utilizar la anotación @‌PathVariable en tu código para recibir el campo de ID de la solicitud GET.

**Reglas de negocio**

* Solicitar el campo ID para realizar la consulta es una acción obligatoria, ya que tu usuario necesita poder visualizar los detalles de un tópico solicitando una consulta a los datos en la base de datos. En este caso, es necesario verificar si el campo ID se ingresó correctamente.

Actualización de tópico

La API debe contar con un endpoint (punto final) para la actualización de los datos de un determinado tópico, y debe aceptar solicitudes del tipo PUT para la URI /tópicos/{id}.

Observación: las mismas reglas de negocio del registro de un tópico deben aplicarse también en su actualización.

Dado que estamos realizando una consulta en la base de datos para luego actualizar un tópico, es necesario solicitar y verificar el campo ID de su solicitud.

En el código del proyecto, sugerimos, al igual que en la tarjeta de Detalle de Tópicos, el uso de la anotación @PathVariable para obtener el ID de la solicitud PUT.

**Eliminación de tópico**

La API debe contar con un endpoint para la eliminación de un tópico específico, el cual debe aceptar solicitudes del tipo **DELETE** para la URI **/tópicos/{id}**.

Dado que estamos realizando una consulta en la base de datos para luego actualizar un tópico, es necesario solicitar y verificar el campo ID de su solicitud.

En el código del proyecto, sugerimos, al igual que en la tarjeta de Detalle de Tópicos, el uso de la anotación @PathVariable para obtener el ID de la solicitud PUT.

→ Recuerda verificar si el tópico existe en la base de datos antes de realizar su actualización. En este caso, sugerimos el uso del método isPresent() de la clase Java llamada Optional.

Por último, al tratarse de la eliminación de un elemento específico de la base de datos, es importante destacar la importancia del uso del método deleteById del *JpaRepository*.

**Pruebas de la API**

Las pruebas de las funcionalidades de la API pueden realizarse utilizando alguna herramienta de pruebas de API, como **Postman** o **Insomnia**.

Postman: [https://www.postman.com](https://www.postman.com/)

Insomnia: [https://insomnia.rest](https://insomnia.rest/)

Aquí tienes algunas sugerencias de contenido para ayudarte a utilizar Postman en esta tarea:

* [Postman: aprenda cómo instalar y dar sus primeros pasos | Alura Cursos Online](https://www.aluracursos.com/blog/postman-aprenda-como-instalar-y-dar-sus-primeros-pasos)
* Git y GitHub son herramientas muy útiles para cualquier desarrollador, así que debes acostumbrarte a trabajar con ellas.
* Ahora que ha desarrollado muchas novedades en su aplicación, es el momento de volver a **actualizar su repositorio**. También recuerde **actualizar README** para incluir las nuevas características de su aplicación.

**Autenticación**

A partir de ahora, solo los usuarios autenticados pueden interactuar con la API.

Implementa un mecanismo de autenticación en la API para que los usuarios puedan autenticarse y enviar solicitudes a ella.

→ Recuerda agregar la dependencia "Spring Security" en tu archivo pom.xml (si aún no lo has hecho en la etapa de configuración del entorno Java y Spring).

**Configuración de seguridad**

Para configurar la autenticación en tu proyecto, es necesario definir tu clase SecurityConfigurations con información para el acceso a través de solicitudes http, utilizando anotaciones como @Configuration y @EnableWebSecurity, así como la clase spring HttpSecurity.

→ Te dejamos como sugerencia la **documentación** oficial de Spring Security (recuerda que el contenido está en inglés, pero puedes traducir la página gracias al atajo de Google Traductor en tu navegador): [Spring Security](https://spring.io/projects/spring-security)

**Autenticación en el código Java**

El proceso de autenticación en la API se realiza con la implementación de un *controller* responsable de recibir las solicitudes de inicio de sesión. Asegúrate de utilizar las anotaciones @RestController y @RequestMapping para definir la URL del *controller*.

Además, utilizamos una clase DTO (en el curso implementada como instancia Record en Java) para recibir los datos de inicio de sesión y contraseña, y luego autenticar al usuario en el método AuthenticationManager presente en la clase SecurityConfigurations.

→ Recuerda utilizar las anotaciones @PostMapping, @RequestBody y @Valid para recibir y validar los datos de la solicitud.

Sugerencia: Contenido del curso "Spring Boot 3: aplique las mejores prácticas y proteja una API Rest" para ayudarte en esta etapa: <https://app.aluracursos.com/course/spring-boot-3-aplique-practicas-proteja-api-rest/task/83804>

**Te recomendamos hacer una prueba de la aplicación en *Insomnia* o *Postman* para verificar si esta etapa se implementó correctamente.**

**Base de datos**

Para implementar el mecanismo de autenticación en la API, deberás modificar su estructura de base de datos, incluyendo una nueva tabla para almacenar los **datos de autenticación de los usuarios**.

En este caso, necesitarás crear o modificar la migración referente a los datos de los usuarios. Esta modificación es importante para gestionar los datos de inicio de sesión y contraseña de cada usuario.

→ Recuerda siempre pausar/detener la ejecución del proyecto Spring para modificar las migraciones.

Sugerencia de **artículos de Alura Latam** para ayudarte en esta tarea:

* [Como crear una buena contraseña | Alura Cursos Online](https://www.aluracursos.com/blog/como-crear-una-buena-contrasena)
* [Tipos de Autenticación: Contraseña, Token, JWT, Dos Factores y Más. | Alura Cursos Online](https://www.aluracursos.com/blog/tipos-de-autenticacion)

**Sugerencia**: Contenido del curso "Spring Boot 3: aplique las mejores prácticas y proteja una API Rest" para ayudarte en esta etapa:

**Token JWT**

Para agregar mayor seguridad a tu aplicación, una opción muy ventajosa es requerir tokens para **autenticación**. El JWT (JSON Web Token) es un estándar utilizado para compartir información entre **cliente y servidor** que será muy útil en esta tarea.

→ Para poder utilizar el JWT es necesario agregar su biblioteca en nuestro pom.xml como dependencia. Accede al sitio para obtener la biblioteca en Java de Auth0: [JWT.IO](https://jwt.io/)

Esta biblioteca es importante precisamente para poder generar el token en el **estándar JWT** y así agregarlo en la configuración de seguridad de nuestro proyecto, creando una clase DTO UsernamePasswordAuthenticationToken para recibir el nombre de **usuario y contraseña**.

**Generar y validar token**

Además, es necesaria la construcción de una clase de servicio, TokenService, para aislar la generación y validación del token.

En la clase, se ha implementado el método "generarToken()", utilizando la biblioteca JWT para crear un token con el algoritmo HMAC256 y una contraseña. También se ha añadido la funcionalidad de configurar la fecha de expiración del token.

Dentro de esta clase, se debe implementar el método "generarToken()" que utiliza la biblioteca JWT para generar el token con el **algoritmo HMAC256** y una **contraseña secreta**. Además, también es importante definir la **fecha de expiración** del token.

Por último, es necesario inyectar esta clase en tu controlador de autenticación, y así obtener el token retornado en la respuesta de la solicitud de inicio de sesión.

→ Recuerda que en esta etapa utilizamos atributos definidos en nuestro application.properties como jwt.secret y jwt.expiration.

Para ayudarte en esta etapa, te **recomendamos** este curso “Spring Boot 3: aplique las mejores prácticas y proteja una API Rest“ en la clase llamada "JSON Web Token" y otros contenidos de nuestra plataforma:

* Curso: <https://app.aluracursos.com/course/spring-boot-3-aplique-practicas-proteja-api-rest/task/83805>
* Artículo: [¿Qué es JSON Web Token? | Alura Cursos Online](https://www.aluracursos.com/blog/que-es-json-web-token)

## Control de acceso

Después de la generación del token JWT, este debe ser utilizado para autenticar la gestión de registros de los tópicos, incluyendo acciones como creación, consulta, actualización o eliminación. La API debe ser configurada para responder con el resultado de cada solicitud, **siempre y cuando el token proporcionado sea válido**.

Antes de configurar más detalles del control de acceso, es válido agregar una nueva solicitud con una URL y un archivo JSON que contenga el nombre de usuario y contraseña para la generación del token. ¿Qué te parece si la URL es "<http://localhost:8080/login>"?

→ Destacamos la importancia de almacenar y enviar el token junto con las próximas solicitudes.

En principio, debemos **mapear las URLs y validar los tokens** en nuestro controlador. Recomendamos crear una clase separada para validar los tokens y llamarla antes de las solicitudes en el controlador para **evitar la repetición de código**.

→ Reforzamos la necesidad de crear un filter o interceptor en el proyecto para validar el token en cada solicitud.

Ahora, manos a la obra y continúa con esta etapa crucial del proceso de autenticación de nuestro FórumHub. No olvides manejar los errores y los estados de las solicitudes de la API utilizando las excepciones de nuestro querido Java.

### Te recomendamos realizar una prueba de la aplicación en Insomnia o Postman para verificar si esta etapa se implementó correctamente.

Para ayudarte en esta tarea, te recomendamos el curso “Spring Boot 3: aplique las mejores prácticas y proteja una API Rest“ en su clase final "Control de Acceso": <https://app.aluracursos.com/course/spring-boot-3-aplique-practicas-proteja-api-rest/task/83806>

README

Uno de los pasos más importantes al participar en un proceso de selección es resolver un desafío propuesto por la empresa, y generalmente esto debe estar descrito en el README. ¿Y qué es el **README**? Es un archivo con extensión **.md** y es un documento con la descripción del proyecto.

Ahora que estamos en la recta final del proyecto, vamos a empezar a desarrollar archivos README para nuestros últimos desafíos.

Aquí tienes un artículo con los pasos para crear un README increíble:

**Recuerda que esto es un desafio opcional y que puedes hacer otras implementaciones incluso si no están siendo sugeridas aqui**

Implementa otras rutas en tu aplicación

Para nuestro foro estar completo deberíamos tener otras rutas o endopints que nos permitirán crear, listar, actualizar y eliminar otras informaciones además de los tópicos como:

* /usuario
* /respuestas

## Documentación

## → Recuerda que este es un desafío opcional y que puedes hacer otras implementaciones aunque no sean sugeridas aquí.

Ahora es el momento de documentar tu API con SpringFox Swagger. Swagger es una herramienta extremadamente ventajosa para tu API, ya que ofrece una interfaz amigable y accesible, así como una forma de documentación automática generada simultáneamente con el desarrollo de la API.

Además de incorporar la dependencia de SpringFox Swagger en tu archivo pom.xml, ¡recuerda probar los endpoints de la API en la interfaz gráfica que esta herramienta ofrece!