¡Te damos la bienvenida a nuestro más reciente Challenge Back End!

Un **foro** es un lugar donde todos los participantes de una plataforma  
pueden colocar sus preguntas sobre determinados asuntos. Aquí en Alura, los estudiantes utilizan el foro para resolver sus dudas sobre los cursos y proyectos en los que están participando. Este lugar mágico está lleno de mucho aprendizaje y colaboración entre estudiantes, profesores y moderadores.

Ya sabemos para que sirve el foro y sabemos cómo se ve, pero ¿sabemos cómo funciona por detrás? Es decir ¿dónde se almacenan las informaciones? ¿cómo se tratan esos datos para que se relacione un **tópico** con una **respuesta**, o como se relacionan los **usuarios** con las respuestas de un tópico?

Ese es nuestro desafío, que se llama **Foro Hub**: vamos a replicar a nivel de back end este proceso, y para eso crearemos una **API REST usando Spring**.

Nuestra API va a centrarse específicamente en los **tópicos**, y debe permitir a los usuarios:

* Crear un nuevo tópico
* Mostrar todos los tópicos creados
* Mostrar un tópico específico
* Actualizar un tópico
* Eliminar un tópico

Es lo que conocemos comúnmente como CRUD (CREATE, READ, UPDATE, DELETE) y es muy parecido con lo que desarrollamos en el Hotel Alura, pero ahora usando un framework que va a facilitar mucho nuestro trabajo.

Al final de nuestro desarrollo tendremos una API REST con las siguientes funcionalidades

1. API con rutas implementadas siguiendo las mejores prácticas del modelo REST;
2. Validaciones realizadas según reglas de negocio;
3. Implementación de una base de datos para la persistencia de la información;
4. Servicio de autenticación/autorización para restringir el acceso a la información.

**¡Manos a la obra!**

Tenemos un periodo de tiempo de tres semanas para desarrollar esta primera parte del proyecto y por eso en esta oportunidad también disponibilizaremos la parte del modelo de las entidades del proyecto, para que puedas centrarte en desarrollar las otras funcionalidades y hagas la API funcionar, no olvides que también puedes usar tu creatividad y hacer las modificaciones que creas convenientes, y de la misma forma en la que venimos trabajando usaremos el sistema ágil de desarrollo, utilizando el Trello de la siguiente forma:

1. La columna **Listos para iniciar** presenta las tarjetas con elementos que aun no fueron desarrollados.
2. En la columna **En Desarrollo** estarán los elementos que estés desarrollando en el momento. Al iniciar una tarea, podrás mover la tarjeta que contiene dicha tarea para esta columna.
3. En la columna **Pausado** estarán los elementos que comenzaste a desarrollar, pero necesitaste parar por algún motivo.
4. Por fin, en la columna **Concluido** estarán los elementos ya concluidos.

El Trello es una herramienta de uso individual para que puedas controlar el progreso de tus actividades, pero no será evaluada.

Buen proyecto!

Git y GitHub son herramientas muy útiles para cualquier desarrollador, así que debes acostumbrarte a trabajar con ellas.

En este desafío queremos proponer que el primer paso a dar sea la creación de este repositorio en GitHub. Aunque aún no hayas desarrollado ningún código, lo importante es que tengas al menos una carpeta específica para tu proyecto, y puedas actualizarlo gradualmente. El material siguiente te ayudará en esta actividad:

* <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Configurando-Git-por-primera-vez>
* <https://www.youtube.com/watch?v=dw04N616Abw&t=5s>
* <https://www.youtube.com/watch?v=bhKkeOMysuw>

Utilicemos nuestro README para detallar las funcionalidades de nuestro sistema, capturas de pantalla e incluso un vídeo de tu aplicación funcionando. De esta manera podrás mostrar tu proyecto a la comunidad 😄

Además, tenemos algunos **artículos** que pueden ayudarte a crear y gestionar un repositorio en GitHub:

* <https://www.aluracursos.com/blog/iniciando-repositorio-con-git>
* <https://www.aluracursos.com/blog/creando-repositorio-remoto-en-github>
* <https://www.aluracursos.com/blog/clonando-un-repositorio-remoto>
* <https://www.aluracursos.com/blog/github-pages>
* <https://www.aluracursos.com/blog/git-y-los-nuevos-comandos>

En esta primera fase, nos sumergiremos en la configuración del entorno de desarrollo Java para nuestro desafío de construir el **ForoHub**. Asegúrate de tener los siguientes programas, archivos y versiones:

* Java JDK: versión 17 en adelante - Descarga la última versión LTS de Java gratuita
* Maven: versión 4 en adelante
* Spring Boot: versión 3 en adelante - <https://start.spring.io/>
* MySQL: versión 8 en adelante \*- [MySQL :: Download MySQL Installer](https://dev.mysql.com/downloads/installer/) (Instalador para **Windows**)
* IDE (Entorno de desarrollo integrado) IntelliJ IDEA - opcional - [Icono

  El contenido generado por IA puede ser incorrecto.Descargar IntelliJ IDEA](https://www.jetbrains.com/es-es/idea/download/?section=windows)

\*Aquí tienes un **artículo Alura Latam** sobre la instalación de **MySQL** en Windows: [Logotipo

El contenido generado por IA puede ser incorrecto.MySQL: desde la descarga e instalación hasta su primera tabla | Alura Cursos Online](https://www.aluracursos.com/blog/mysql-desde-la-descarga-e-instalacion-hasta-su-primera-tabla)

Configuración al crear el proyecto con [Spring Initializr](https://start.spring.io/):

* Java (versión 17 en adelante)
* Maven (Initializr utiliza la versión 4)
* Spring Boot
* Proyecto en formato JAR

Dependencias para agregar al crear el proyecto con [Spring Initializr](https://start.spring.io/):

* Lombok
* Spring Web
* Spring Boot DevTools
* Spring Data JPA
* Flyway Migration
* MySQL Driver
* Validation
* Spring Security

**Diagrama de Base de Datos**

Necesitarás utilizar una base de datos para almacenar la información de la aplicación.

Para crear un tópico necesitan las siguientes informaciones:

* id
* título
* mensaje
* fecha de creación
* status (estado del tópico)
* autor
* curso

En la siguiente imagen te proporcionamos una sugerencia en forma de **diagrama** para la creación de tu base de datos con los campos requeridos por nuestro cliente.

**Observación: Nuestro challenge se centra en los tópicos, por lo que el diagrama representa una base de datos más completa pero no es obligatorio implementar todas las tablas presentes en él - es suficiente centrarse en la tabla de tópicos.**

La modelización de la(s) tabla(s) queda a tu criterio.

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

* [Logotipo

  El contenido generado por IA puede ser incorrecto.Guía de Instalación de MySQL Server y MySQL Workbench en Diferentes Sistemas Operativos" | Alura Cursos Online](https://www.aluracursos.com/blog/descargar-mysql-serve)
* [Logotipo

  El contenido generado por IA puede ser incorrecto.Bases de datos relacionales | Alura Cursos Online](https://www.aluracursos.com/blog/base-de-datos-relacional)
* [Logotipo

  El contenido generado por IA puede ser incorrecto.Normalización en base de datos - Estructura | Alura Cursos Online](https://www.aluracursos.com/blog/normalizacion-en-base-de-datos)

También te proporcionamos enlaces a los softwares para la creación del modelo físico de la base de datos:

* [MySQL :: MySQL Workbench](https://www.mysql.com/products/workbench/)
* [Download Astah Software](https://astah.net/downloads/)
* [fabFORCE.net](https://www.fabforce.net/dbdesigner4/downloads.php)
* <https://lucid.app/documents#/dashboard>

**Instalación de MySQL**

Para ayudarte en esta etapa de construcción de la base de datos para nuestro proyecto ForoHub, nos gustaría dejarte algunos enlaces importantes tanto para la instalación de MySQL como para la configuración de la base de datos a través del proyecto Spring.

Enlace para descargar el **instalador** para Windows: [MySQL :: Download MySQL Installer](https://dev.mysql.com/downloads/installer/)

→ Tranquilo, aquí tienes un **artículo Alura Latam** sobre cómo instalar no solo en Windows: [Logotipo

El contenido generado por IA puede ser incorrecto.Guía de Instalación de MySQL Server y MySQL Workbench en Diferentes Sistemas Operativos" | Alura Cursos Online](https://www.aluracursos.com/blog/descargar-mysql-serve)

**Configuración en el proyecto**

Para integrar una base de datos a nuestro proyecto Spring, debemos agregar algunas **dependencias** en nuestro pom.xml (en caso de que aún no las hayas agregado al configurar el entorno Java y Spring):

* Validation
* MySQL Driver
* Spring Data JPA
* Flyway Migration

Además, también es importante recordar la configuración necesaria en nuestro application.properties con los datos de url, nombre de usuario y contraseña de nuestra base de datos, lo que implica definir el driver de la base de datos, así como los datos de inicio de sesión con usuario y contraseña.

**Sugerencia**: Antes de pasar a la etapa de migraciones del proyecto, te sugerimos crear la base de datos y configurarla según se mencionó anteriormente.

**Migración en el proyecto**

Las migraciones son comandos en lenguaje SQL para la configuración de la base de datos creada.

Es necesario definir las migraciones, por ejemplo, para la construcción de las tablas de tu proyecto en archivos con extensión .sql. Además, es importante reforzar el uso de las anotaciones @‌PostMapping y @‌Transactional en tu clase controladora para realizar la persistencia de los datos.

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

Si lo deseas, realiza pruebas de la aplicación en Insomnia o Postman.

**Sugerencia**: Contenido del curso "Spring Boot 3: desarrolla una API Rest en Java" para ayudarte en esta etapa: <https://app.aluracursos.com/course/spring-boot-3-desarrollar-api-rest-java/task/84642>

**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).

**Sugerencias complementarias**

Aquí te dejamos algunas **sugerencias** de la **documentación** de *Spring Data* y del blog *Baeldung* (recuerda que el contenido está en inglés, pero puedes traducir la página con el atajo de Google Traductor en tu navegador):

* Para empezar con JpaRepository: [Getting Started :: Spring Data JPA](https://docs.spring.io/spring-data/jpa/reference/jpa/getting-started.html)
* Más detalles sobre JpaRepository: [2. JPA Repositories](https://docs.spring.io/spring-data/jpa/docs/1.6.0.RELEASE/reference/html/jpa.repositories.html)
* Sobre la validación de datos en REST Controller: [Validation in Spring Boot | Baeldung](https://www.baeldung.com/spring-boot-bean-validation)

Si no sabes cómo realizar esta actividad, te sugerimos que veas el siguiente contenido: [Spring Boot 3: desarrolla una API REST en Java: Aula 2 - Actividad 1 Proyecto del aula anterior | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-dessarrola-api-rest-java/task/91193)

**Descripción**

Editar

**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.

Si no sabes cómo realizar esta actividad, te sugerimos que veas el siguiente contenido: [Spring Boot 3: desarrolla una API REST en Java: Aula 4 - Actividad 1 Proyecto del aula anterior | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-dessarrola-api-rest-java/task/91195)

**Descripción**

Editar

**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.

Si no sabes cómo realizar esta actividad, te sugerimos que veas el siguiente contenido: [Spring Boot 3: desarrolla una API REST en Java: Aula 4 - Actividad 1 Proyecto del aula anterior | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-dessarrola-api-rest-java/task/91195)

**Descripción**

Editar

**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.

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

Si no sabes cómo realizar esta actividad, te sugerimos que veas el siguiente contenido: [Spring Boot 3: desarrolla una API REST en Java: Aula 5 - Actividad 1 Proyecto del aula anterior | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-dessarrola-api-rest-java/task/91196)

**Descripción**

Editar

**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*.

Si no sabes cómo realizar esta actividad, te sugerimos que consultes el siguiente contenido:

[Spring Boot 3: desarrolla una API REST en Java: Aula 5 - Actividad 6 Requests DELETE | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-dessarrola-api-rest-java/task/91112)

**Descripción**

Editar

**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: [Spring Boot 3: aplique las mejores prácticas y proteja una API Rest: Aula 3 - Actividad 1 Proyecto del aula anterior | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-aplique-mejores-practicas-proteja-api-rest/task/91699)

**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:

* <https://www.aluracursos.com/blog/como-crear-una-buena-contrasena>
* <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: [Spring Boot 3: aplique las mejores prácticas y proteja una API Rest: Aula 3 - Actividad 5 Entidad usuario y migration | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-aplique-mejores-practicas-proteja-api-rest/task/91595)

**Descripción**

Editar

**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: [Spring Boot 3: aplique las mejores prácticas y proteja una API Rest: Aula 4 - Actividad 1 Proyecto del aula anterior | Alura Latam - Cursos online de tecnologia](https://app.aluracursos.com/course/spring-boot-3-aplique-mejores-practicas-proteja-api-rest/task/91700)
* Artículo: <https://www.aluracursos.com/blog/que-es-json-web-token>

### Descripción

Editar

## 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.

**Descripción**

Editar

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:

**Descripción**

Editar

**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

### Descripción

Editar

## 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!