Challenge técnico

¡Bienvenidos al challenge técnico de Rooftop Academy! Estamos contentos de que nos elijas como motor de crecimiento para tu futuro. En este documento encontrarás toda la información relacionado a este desafío (los ejercicios en sí, formato de entrega, etc). Por eso, **te recomendamos que leas todo el documento detenidamente antes de comenzar**.

Para aprobar el ingreso, deberás solucionar la consigna propuesta.

Entrega

Para entregar la solución deberás crear un repositorio en tu cuenta de Github, o del servicio Git que utilices. **Asegúrate que el repositorio sea público para que podamos verlo.**

Solamente aceptaremos links a repositorios, no es necesario que envíes links de Google Drive, ni que adjuntes archivos comprimidos tipo Zip en el mensaje. Tampoco requerimos el ejecutable (.jar) de JAVA (aunque puedes incluir un fat jar -ver explicación debajo). No descargaremos ningún tipo de archivo en ningún caso.

En el caso de que se reciba correctamente tu solución (con un asunto válido), **se responderá un email automático que te confirmará que hemos recibido correctamente la entrega**.

Sólo se tomará en cuenta los commits realizados hasta la fecha de cierre. Si agregas cambios en el repositorio luego, no será tomado en cuenta para la validación.

CONDICIONES:

- Resolver la consigna en lenguaje **Java**.
- Deberás usar una base de datos para guardar registros. La misma debe ser H2.
- Incluir en el README del proyecto los pasos a seguir para que podamos levantar/ejecutar el proyecto. Si usas Spring Boot será fácil para nosotros, pero si usas algún otro framework nos servirá de mucho esa guía.
- Aunque no es obligatorio, obtendrás puntos adicionales si dentro del proyecto incluyes un fat jar, el cual tenga un servidor Tomcat embebido y nos facilite el poder levantar el servidor para que podamos probarlo.
- La honestidad es un valor fundamental para nosotros. **Si detectamos soluciones iguales, todas las personas implicadas serán bloqueadas de Rooftop Academy y no podrán ingresar a ningún programa en el futuro**. No intentes copiar ni prestar la solución a alguien, porque ambos resultarán perjudicados.

- Si no se respeta el formato de entrega (asunto correcto, cuerpo de email con nombre y apellido y una url) se descontarán puntos.

Consigna 🤓

El objetivo de este challenge es desarrollar una API REST siguiendo las especificaciones que indicaremos luego. Este servicio analizará textos y devolverá conteos acerca de caracteres y palabras. Todas las interacciones tendrán efecto sobre la base de datos para alguna operación de inserción, lectura o eliminación.

La API REST consiste de 4 endpoints. Todas las rutas de los endpoints deben llamarse **"/text"**. Solo se aceptarán peticiones con los métodos GET, POST, DELETE.

Esta API es como una API pública. Es decir, no requiere de ningún mecanismo de autenticación de usuarios. Tampoco es necesario validar más de lo que te pidamos, ni implementar seguridad. Lo único que pedimos es que configures CORS para que cualquier frontend pueda llamarla desde cualquier URL, sin ser bloqueado por el navegador. (Más información sobre CORS)

La base de datos deberás hacerla según lo que consideres necesario para cumplir la consigna. ¡Cada detalle es importante! Todas las respuestas de todos los endpoints deben devolver JSON siempre, incluso si sucede un error.

¿Cómo es el análisis del texto?

Supongamos como ejemplo que el texto es: "*solo se que nada se*"

Del texto recibido se tomará 2 caracteres, empezando por los primeros (es decir, "so"). Hay que buscar a lo largo de todo el texto si aparecen nuevamente las letras "so". En el ejemplo vemos que no se repite.

Luego se realiza nuevamente el proceso con las siguientes dos letras "ol", que tampoco se repiten. Así sigue con "lo", "o " (fijarse que es una "o" y un espacio), etc.

Cuando llegue a las letras " s", buscará y encontrará 1 repetición adicional a lo último. Lo mismo pasará con "se".

Con este ejemplo, y recorriendo todo el texto, la respuesta sería la siguiente:

```
{
   "so": 1,
   "ol": 1,
```

```
    "lo": 1,
    "o ": 1,
    " s": 2,
    "se": 2,
    "e ": 2,
    " q": 1,
    "qu": 1,
    "ue": 1,
    " n": 1,
    "na": 1,
    "ad": 1,
    "da": 1,
    "a ": 1
}
```

Si el parámetro que indica la cantidad de caracteres a tomar fuese 6, entonces devolvería lo siguiente:

```
{
    "solo s": 1,
    "olo se": 1,
    "lo se ": 1,
    "o se q": 1,
    " se qu": 1,
    "se que": 1,
    "e que ": 1,
    " que n": 1,
    "que na": 1,
    "ue nad": 1,
    "e nada": 1,
    " nada ": 1,
    "nada s": 1,
    "ada se": 1
}
```

Consideraciones a tener en cuenta:

- No importa si son mayúsculas o minúsculas (case insensitive: “Lo” y “lo” son lo mismo).
- Los espacios también se consideran como letras.
- La API siempre recibirá caracteres ASCII, por lo que no deberás preocuparte por acentos o diacríticos.
- Si la cantidad de caracteres tomados es más grande que el texto, entonces se contará el texto completo. Por ejemplo, si el valor es 100 y el texto es “*solo se que nada se*”, el resultado será:

```
{
   "solo se que nada se": 1
}
```

Endpoints

POST /text

Recibe un texto para analizarlo y guardarlo en la base de datos.

Si el mismo texto y parámetro de cantidad de letras ya existe en la base de datos, entonces no deberá crear otro registro. Para esto deberás utilizar un hash con MD5 (o algún otro hash) y guardarlo en la base de datos. Para verificar si el texto y parámetro es idéntico utiliza el hash, no el contenido.

En la petición (POST) para analizar el texto puede incluirse un parámetro adicional llamado “chars” que indique la cantidad de caracteres que deberían ser tomados para el análisis. El mismo es opcional. En el caso que no se envíe, se usará por defecto el valor de “2”. Si el valor es menor a “2” (lo cual es incorrecto), usar “2” por defecto.

Paso por paso:

1. El usuario envía la petición con el texto a analizar.
2. La aplicación debe hashear el texto junto con la cantidad de letras a analizar, y buscar con el hash resultante si ya existe en la base de datos.
3. En caso que ya exista, devolverá el id existente como respuesta al cliente.
4. En caso que no, analiza el texto, guarda toda la información en la db, y devuelve el ID existente.
5. Siempre se responde con el id y la url a consultar para saber el resultado, sin importar si ya había sido analizado previamente o no.

En el siguiente ejemplo usa el parámetro “chars”. Aunque se envía el número 2 (que es el valor por defecto), tener en cuenta que puede ser otro valor (como 4).

Request

```
{
   "text": "solo se que nada se",
   "chars": 2
}
```

Response

```
{
   "id": 2345,
```

```
    "url": "/text/2345"
}
```

GET /text/{id}
Devuelve información detallada acerca de un texto en particular identificado por el ID de la base de datos.

Teniendo en cuenta la petición anterior, supongamos que hacemos una consulta a "*/text/2345*". La respuesta deberá ser:

```
{
    "id": 2345,
    "hash": "6c37815dafc28e44ded5f7bc827d15b2",
    "chars": 2,
    "result": {
        "so": 1,
        "ol": 1,
        "lo": 1,
        "o ": 1,
        " s": 2,
        "se": 2,
        "e ": 2,
        " q": 1,
        "qu": 1,
        "ue": 1,
        " n": 1,
        "na": 1,
        "ad": 1,
        "da": 1,
        "a ": 1
    }
}
```

El campo "hash" es el hash que se guarda para evitar tener que analizar nuevamente un texto que ya lo ha sido.

En el caso que el ID no exista en la base de datos, o haya sido borrado, deberá devolver una respuesta HTTP 404 con el siguiente cuerpo:

```
{
    "error" : true,
    "message" : "Text not found",
    "code" : 404
```

```
}
```

GET /text

Devuelve todos los textos previamente enviados, de forma paginada. Los textos que se hayan marcado como *borrados* no deberían aparecer aquí.

Parámetros del query string:

- **chars** = Parámetro opcional. Debe ser un número mayor o igual a 2. Si el número es menor a 2, o si no se envía, se usará como valor por defecto 2. Este número determina con qué cantidad de caracteres es la búsqueda.
- **page** = Parámetro opcional. Debe ser un número para indicar la página que se solicita. Si no se envía o es menor a “1”, el valor por defecto es “1”. La primer página es la “1”
- **rpp** = Parámetro opcional. Debe ser un número para indicar Son los resultados por página, ahí el usuario puede indicar un número entre 10 y 100. Si indica un número fuera del rango simplemente debe devolver el mínimo o máximo por defecto según el caso. Si fuera “3” devuelve 10, y si fuera “500” usará 100.

Supongamos que hacemos una consulta a *“/text?chars=6”* y ya hay dos análisis efectuados previamente usando 6 caracteres como parámetro. Una respuesta de ejemplo sería:


```
[
    {
        "id": 1646656897698,
        "hash": "6c37815dafc28e44ded5f7bc827d15b2",
        "chars": 6,
        "result": {
            "solo s": 1,
            "olo se": 1,
            "lo se ": 1,
            "o se q": 1,
            " se qu": 1,
            "se que": 1,
            "e que ": 1,
            " que n": 1,
            "que na": 1,
            "ue nad": 1,
            "e nada": 1,
            " nada ": 1,
            "nada s": 1,
            "ada se": 1
        }
    },
    {
```

```
        "id": 1646656911363,
        "hash": "f260d5903f9fa42c255a261ed8b25422",
        "chars": 6,
        "result": {
            "asasas": 2,
            "sasasa": 1,
            "sasase": 1,
            "asasee": 1
        }
    }
]
```


DELETE /text/{id}

Borra el texto por el id especificado en la URL. El borrado es “lógico” (soft delete). Es decir, que no se elimina físicamente la información sino que se utiliza una columna de la base de datos para saber si fue “borrado” o no.

En el caso que sea exitoso, devolver un JSON vacío “`{}`. En el caso que no exista ningún registro con ese ID para borrar, devolver el siguiente error con un código HTTP 404:


```
{
   "error" : true,
   "message" : "Text not found",
   "code" : 404
}
```


PARA TODOS LOS ENDPOINTS

El manejo de errores deberá lograr que el cliente siempre reciba un JSON como respuesta. Los errores deberán tener este formato. Si tu aplicación tiene una excepción no controlada debido a un bug, devolver el siguiente JSON:


```
{
   "error": true,
   "message": "An error occurred when processing the text",
   "code": 422
}
```


Dejamos a tu criterio el texto del mensaje, pero para el código de estado utilizar los códigos estándares de HTTP.
