Taller de Programación 2

Examen Final (3 de julio, 2023)

Enunciado

Se desea realizar un sistema para procesar texto que forme parte del desarrollo de un juego tipo «Cadáver exquisito», implementando los siguientes requerimientos:

- Poder ingresar palabras aisladas que compondrán un texto final.
- Poder listar las palabras almacenadas en forma de frase.
- Contar con un canal que indique la cantidad de palabras ingresadas y las veces que se encuentran repetidas.
- Borrar del sistema una palabra.

1. Implementar los endpoints que permitan:
    a. Ingresar una palabra, validando en el lugar adecuado que sea válida: no vacía, sólo con caracteres alfabéticos. Retornar la palabra con el código de estado de ingreso:
        - status **200**: ok
        - status **422**: no válida
    b. Listar la frase completa en modo texto uniendo las palabras ingresadas.
    c. Eliminar una palabra indicada por parámetro de ruta, retornando dicha palabra e indicando en el código de estado de respuesta el éxito / falla de la operación:
        - status **200**: palabra eliminada correctamente.
        - status **404**: palabra no encontrada.
        - status **422**: palabra no válida.

    Si hubiesen varias palabras que coinciden con el parámetro, se borran todas.

2. El sistema contará con un servicio externo que podrá consumir, el cual le devuelve la cantidad de palabras aleatorias que el cliente necesite generar. La URL externa de la API es: https://texto.deno.dev/palabras?cantidad=x donde x será la cantidad de palabras a devolver. La idea es utilizarlo bajo un endpoint provisto en el servidor que le indique a esta API cuantas palabras quiero generar y luego las incorpore a las existentes.

    Formato de la respuesta de la API:

    ```
    {
      cantidad: x,                  // cantidad de palabras
      palabras: [p1, p2, p3, ...]   // array con cada una de las palabras generadas
    }
    ```

3. Implementar un endpoint que permita obtener la información de todas las palabras ingresadas hasta ese momento, mediante un objeto de información que indique cuántas veces aparece cada palabra en el texto completo utilizando el siguiente formato json:

    ```
    {
      p1: x,
      p2: y,
      p3: z,
      ... : ...
    }
    ```

    donde x y z … son las veces que aparecen las palabras p1, p2, p3 … respectivamente.

En caso de ser necesario, el servidor recibirá desde el cliente los datos requeridos en formato JSON. En caso de inconvenientes, el servidor responderá con un objeto con un campo **'errorMsg'** informando el motivo de la falla. Todas las respuestas deberán estar correctamente adosadas con su código de estado correspondiente, según el resultado de la operación. En el caso de realizar notificaciones, modularizar adecuadamente. La notificación en sí puede simularse con un mensaje por terminal.

Aclaraciones sobre el desarrollo esperado:

1. El proyecto debe incluir únicamente el backend del sistema, utilizando Node.js + express. El formato del servidor es de tipo RESTful. Tener en cuenta los lineamientos de dicho formato, especialmente a la hora de elegir los nombres de las rutas de acceso al sistema, los verbos que accionan sobre ellas, y cómo pasar datos adicionales a la consulta.
2. El sistema debe estar correctamente separado en capas y componentes, y esta separación debe estar claramente puesta de manifiesto en la estructura de carpetas y archivos. Entre los componentes que esperamos que estén presentes encontramos: router/controlador, casos de uso, modelo/s, DAO/s, repositorios, servicios de terceros, factories, commands (los que correspondan de acuerdo al sistema modelado y se hayan visto en cursada).
3. La *validación de datos* es una parte importante del negocio, por lo tanto, observar cómo y dónde realizarla.
4. *No es necesario utilizar una conexión a base de datos* real; es posible persistir en el DAO/Repositorio usando memoria del servidor.
5. Recordar el rol de las factories, que nos permiten desacoplar las dependencias de nuestros componentes a la hora necesitar una instancia de los mismos. Recordar esto especialmente a la hora de decidir cómo obtener los casos de uso para invocarlos desde la capa de ruteo.
6. Considerar la inclusión de algún test funcional (usando axios, por ejemplo) para verificar el correcto funcionamiento de lo que se está desarrollando.
7. Pueden reutilizar código de proyectos realizados durante el cuatrimestre, siempre y cuando el código se utilice y realmente aporte al desarrollo de las funcionalidades pedidas. **La inclusión de código innecesario o fuera de lugar será penalizada**.