[v3] Full Stack JS - Code Challenge - Español

Code Challenge

El JavaScript *challenge* para Full Stack esta dividido en 3 partes:

1. Un API usando Node + Express
2. Un *frontend* cliente usando Bootstrap + React
3. Puntos opcionales para lucir tus conocimientos.

¿Qué tomamos en cuenta al revisar el *challenge*?

- El código entregado tiene que funcionar y cumplir con todo lo solicitado.
- Las instrucciones y la documentación deben ser prolijas y claras.
- Debe cumplir con los requisitos técnicos solicitados (incluir las librerías/frameworks especificados, versiones pedidas, tests indicados, instrucciones para correrlo, documentación, etc.).
- En caso de no entregarse o de no cumplir lo esperado, los puntos opcionales no influyen en la evaluación que haremos . Pero suman mucho para nosotros en caso de estar bien.

¿Qué hacer si tengo dudas sobre algo de lo pedido?

Ponte en contacto con la persona que te envió este *challenge*. Es preferible aclarar las dudas antes de enviar un resultado de *challenge* incorrect o o incompleto. Las consultas también nos ayudan a modo de retroalimentación para mejorar las consignas.

¿Cómo envío el *challenge*?

Debes dejar tu código en un repo git público (o uno al cual podamos acceder) y luego enviarle la información a la persona que te envió este examen (URL y datos de acceso).

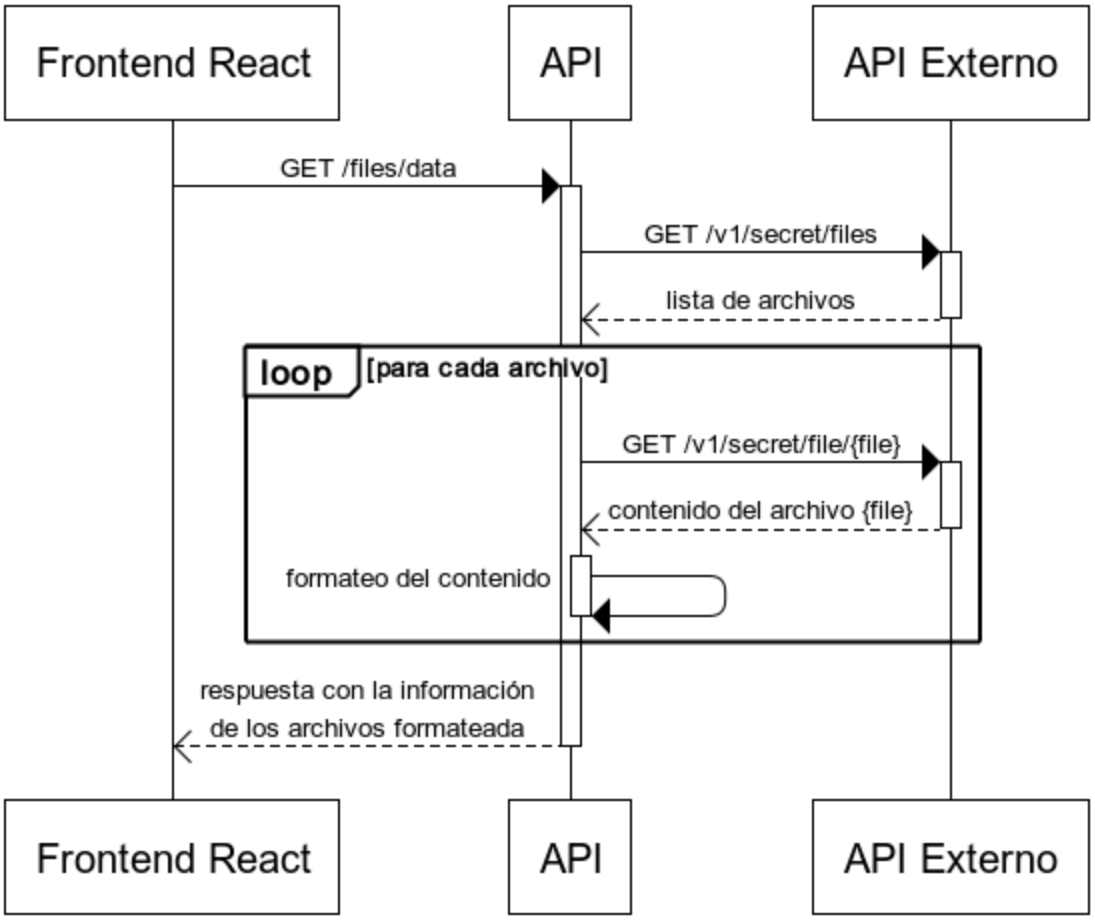
Una vez enviado el acceso, no deberás realizar ningún cambio en el repo. La evaluación puede comenzar en cualquier momento desde que recibimos los accesos.

Challenge

Este challenge te pide a desarrollar un *frontend* que consumirá datos de un API que también deberás desarrollar. El *frontend* deberá mostrar la información obtenida respetando las pautas que se describirán más abajo. Asimismo la API también debe ser desarrollada siguiendo requisitos explícitos.

EL API Externo es provisto por nosotros y sólo debe ser consumido.

El siguiente diagrama de secuencia muestra cómo es el el flujo de las peticiones de información para este ejercicio. Si bien el formateo del contenido de archivos no es una llamada REST, el diagrama la incluye porque es una parte fundamental de este desafío.



1. API

El API a desarrollar, es un API REST que toma información de un API externa y la reformatea para exponerla.

El API Externo

El API Externo de la cual se toma la información está documentada en el siguiente Swagger: https://echo-serv.tbxnet.com/explorer/#/Secret

Para poder utilizarla, la API Key es: "Bearer aSuperSecretKey".

Los métodos a utilizar están en la sección "Secret" de la documentación del Swagger, pero a modo de resumen se indican a continuación:

Para listar los archivos:

```
$ curl -X GET https://echo-serv.tbxnet.com/v1/secret/files -H
'authorization: Bearer aSuperSecretKey'
{
   "files":["file1.csv",....]
}
```

Para descargar un archivo:

```
$ curl -X GET https://echo-serv.tbxnet.com/v1/secret/file/file1.csv -H
'authorization: Bearer aSuperSecretKey'

....
```

Los archivos siguen el formato CSV estricto con las siguientes columnas:

- **file**: el nombre del archivo.
- **text**: un texto de largo variable
- **number**: un numero
- **hex**: un hexadecimal de 32 dígitos

Ejemplo del contenido de un archivo con información correctamente formateada:

```
file,text,number,hex
file1.csv,RgTya,64075909,70ad29aacf0b690b0467fe2b2767f765
file1.csv,AtjW,6,d33a8ca5d36d3106219f66f939774cf5
file1.csv,PNzRfORtKtEDOzmIVrQuSh,74088708,3
e29651a63a5202a5661e05a060401fb
file1.csv,d,6173,f9e1bcdb9e3784acc448af34f4727252
```

Cómo procesar la información

1. Se deben llamar al listado de archivos /v1/secret/files
2. Descargar cada file usando /v1/secret/file/{file}
3. Formatear la información en los CSV:
    a. Tener en cuenta que pueden existir archivos vacíos y líneas con error (que no tenga la cantidad de datos suficientes).
    b. Si una línea tiene error se debe descartar.
    c. También pueden existir errores al descargar un archivo.
4. Por cada archivo, se debe crear un objeto JSON que contenga las líneas válidas.

Un objeto JSON pedido a partir de un archivo debe seguir el siguiente *schema*:

```
{
   "file": "file1.csv",
   "lines": [
      {
         "text" :"RgTya",
         "number": 64075909,
         "hex": "70ad29aacf0b690b0467fe2b2767f765"
      },
      . . .
   ]
}
```

Usando NodeJs + ExpressJs, se debe crear el API para funcionar desde el siguiente *endpoint*:

```
GET /files/data
```

Este *endpoint* es el encargado de buscar los archivos y formatear la información tal como se indicó en los pasos descriptos previamente.

Toda la información generada por el API deberá ser definida como `content-type: application/json`.

Una respuesta 200 en caso de éxito se deberá ver como sigue:

```
[
   {
      "file": "file1.csv",
      "lines": [
         {
            "text" :"RgTya",
            "number": 64075909,
            "hex": "70ad29aacf0b690b0467fe2b2767f765"
         },
         . . .
      ]
   }
]
```

Ejemplo usando curl (llamada y respuesta):

```
$ curl -v -X GET "http://apihost/files/data" -H "accept: application
/json"

> GET /files/data HTTP/1.1
> Host: apihost
> User-Agent: curl/7.68.0
> accept: application/json
>
< HTTP/1.1 200 OK
< Date: Mon, 19 Oct 2020 15:18:53 GMT
< Content-Type: application/json; charset=utf-8
< Content-Length: 15
< Connection: keep-alive
[
   {
      "file": "file1.csv",
      "lines": [
         {
            "text" :"RgTya",
            "number": 64075909,
            "hex": "70ad29aacf0b690b0467fe2b2767f765"
         },
         . . .
      ]
   }
]
```

También se deben crear los tests que validan el API usando Mocha + Chai.

Los tests deben poder correrse usando `npm test` y el API debe poder iniciarse usando `npm start`.

Requisitos para el código del API

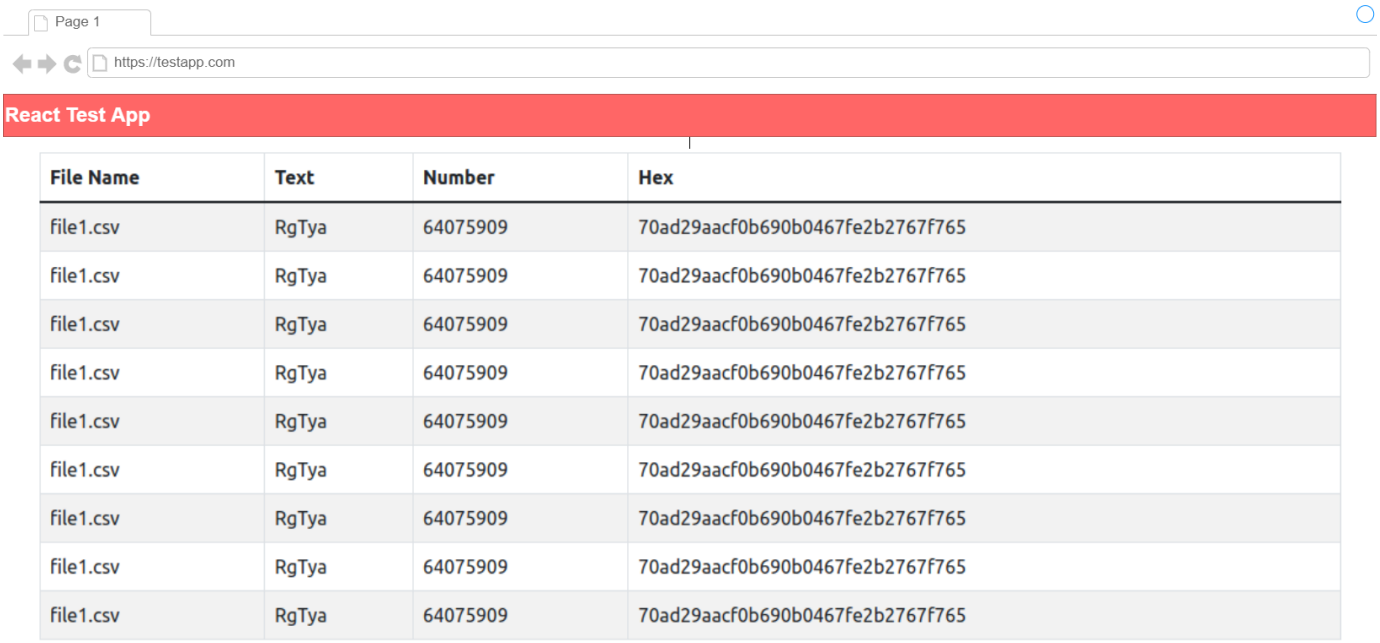
- El código que envíes debe correr usando NodeJS 14 y no depender de librerías que están instaladas de forma global, variables de entorno o configuraciones de algún sistema operativo especifico.
- El código debe ser escrito en JavaScript (ES6+), no utilizar: Babel, TypeScript, Dart, Elm, etc
- En cuanto a las librerías y frameworks, puede usar la versión que consideres apropiadas:
  - ExpressJs https://expressjs.com/
  - Mocha https://mochajs.org/
  - Chai https://www.chaijs.com/

2. Frontend

Deberás desarrollar una App en React que deberá actuar como cliente del API ya desarrollado y que permita ver la información de `/files/data` de manera ordenada en pantalla.

Layout

Usando React + React Bootstrap se debe crear una pantalla similar a la que se muestra en el siguiente *wireframe*:



Requisitos para el código del *frontend*

- Se deberá usar programación funcional y Hook Effects en React.
- El código que envíes debe correr usando NodeJS 16 y no depender de librerías instaladas de forma global, variables de entorno o configuraciones de algún sistema operativo especifico.
- El código debe ser escrito en JavaScript (ES6+).
- No están permitidas las siguientes herramientas: TypeScript, Dart, Elm, ni similares.
- En cuanto a las librerías y *frameworks*, puedes usar la versión que consideres apropiadas de:
  - Webpack https://webpack.js.org/
  - React https://reactjs.org/
  - React Bootstrap https://react-bootstrap.github.io/

3. Puntos opcionales

Puntos opcionales para el API

- Un endpoint `GET /files/list` que dé como respuesta la lista de archivos disponibles tal cual como se la muestra en el API Externa.
- Agregar un filtro por *queryparam* para poder pedir los datos de un archivo especifico: `/files/data?fileName=<Nombre del archivo>`.
- Usar StandarJs https://standardjs.com/.

Puntos opcionales para el *frontend*

- Usar Redux https://redux.js.org/.
- Test unitarios usando Jest https://jestjs.io/.
- Poder filtrar por "fileName" usando el punto opcional del API de listado de archivos y filtro por *queryparam*.

Punto opcional global

- Usar Docker o Docker Compose para correr las apps.