RESOLUCIÓN EXAMEN

Andrés Belhot

Consideraciones generales

La resolución de la prueba está compuesta por los siguientes elementos del repositorio:

https://github.com/landru4/rest-api-js

1. Este documento: *'Documentacion API.pdf'*
2. El código contenido en este repositorio: https://github.com/landru4/rest-api-js
3. Las instrucciones para ejecutar la solución se encuentran en el readme.md del repositorio.

Sobre las herramientas:

- Se codificó utilizando *Visual Studio Code*.
- Se utilizó la herramienta *Postman* para realizar las pruebas de las peticiones tanto de la API solicitada como de la provista para obtener el archivo y los datos de los clientes.
- En la raíz del repositorio se encuentra el archivo: '*PruebasAPI.postman_collection.json'*

Sobre la API:

- Está desarrollada en JavaScript utilizando node.js, adonis, sqlite3.
- La arquitectura utilizada se realizó siguiendo el patrón MVC provisto por adonis.
- Se agrega la subruta */api/v1/* para simular la ubicación dentro de otro sistema posiblemente en el mismo servidor y con su respectiva versión 1.
- El código en términos generales consta de un controlador por cada entidad: Cliente, Pago, Transaccion, Descuento y Usuario con sus respectivos modelos.
- Adicionalmente se utiliza un controlador auxiliar a modo de encapsular y modularizar las operaciones de inicio y finalización del loop infinito de la API:
- Todas las operaciones (excepto el registro) requieren autenticación utilizando el Bearer token provisto al loguearse.

Requisitos:

1. Uso de Postman necesario para ejecutar las solicitudes, en el archivo que se encuentra en la raíz, se tienen ejemplos de cada solicitud que se realiza a la API.
2. Los requisitos de ejecución se encuentran en el archivo readme.me dentro del repositorio.

Utilización de la API:

1. Registro, login y logout

a. Registro: El usuario debe registrarse previamente en la API a través del siguiente endpoint: *localhost:3333/api/v1/usuarios/signup* indicando '*email*' y '*password*' en el body del request.
b. La API realiza el registro y el login en conjunto y responde con un Bearer token, el cual debe utilizarse en todas las solicitudes restante del sistema.
c. Login: Alternativamente si el usuario ya se encuentra registrado, puede realizar el login a través del endpoint: *localhost:3333/api/v1/usuarios/login* indicando '*email*' y '*password*' en el body del request.

d. Logout: Cuando el usuario está logueado previamente, enviando el Bearer token que tiene asociado a: *localhost:3333/api/v1/usuarios/logout*, la API lo desloguea, necesitando loguearse nuevamente para realizar futuras peticiones ya que el sistema invalida todas las tokens asociadas al usuario.

2. Inicializar y finalizar el proceso automático que obtiene el archivo 'file' cada 10 minutos

a. Inicializar el proceso infinito de leer el archivo:
Endpoint: *localhost:3333/api/v1/iniProceso*

La API cuenta con la inicialización del proceso de obtener el archivo *'file.txt'* desde la API provista siguiendo las indicaciones y formato indicados. Al inicializarse, el proceso se ejecuta en un loop infinito cada **10 minutos** para obtener nuevamente el archivo. Esto lo realiza de esa forma para considerar que el archivo permanece en caché durante 10 minutos y luego se pierde.
Entonces la API se encarga de leer cada 10 minutos el archivo *'file.txt'* y volcar los datos en la base de datos local sqlite3.
En caso de ocurrir un error al obtener el archivo de la API, la solución reintenta conectarse nuevamente para obtener el archivo en cuestión a los **30 segundos**.
Lo mismo espera para volver a solicitar en caso de ocurrir un error al intentar obtener los datos de un cliente.
Todos estos valores están configurados de manera independiente de forma general para que puedan ser fácilmente modificados en caso de ser necesario.

b. Finalizar el proceso infinito de leer el archivo:
Acceder a *localhost:3333/api/v1/finProceso*

La API continúa la ejecución hasta su próximo ciclo. Es decir, no intentará más obtener el archivo a los **10 minutos** pero sigue procesando el archivo que está actualmente disponible en la API provista.

Sobre los endpoints requeridos:

Al no tener los detalles del formato de los datos, ni de los endpoints solicitados, se resolvió desarrollar los siguientes endpoints para cumplir con los requisitos del ejercicio.

1. Información de los clientes
Se desarrollaron dos endpoints:

a. *localhost:3333/api/v1/clientes*: Retorna la información guardada en el sistema de todos los clientes de forma histórica. El formato es una lista de JSON.

Ejemplo con una lista de 1 cliente:


```
[{
        "id": "6b43736c8c5a46a0bff26203a47ee16f",
        "email": "frankie@beatty.io",
        "first_name": "Franklyn",
        "last_name": "Botsford",
        "job": "Customer Healthcare Administrator",
```

```
            "country": "Saint Helena",
            "address": "21337 Muriel Drives",
            "zip_code": "69367",
            "phone": "243-903-6015",
            "created_at": "2022-04-18 17:29:51",
            "updated_at": "2022-04-18 17:29:52"
        }]
```


b. *localhost:3333/api/v1/clientes/cliente/:id*: Retorna la información guardada en el sistema del cliente con id solicitado en el parámetro *'id'*. El formato del resultado es un elemento JSON con el mismo formado del ejemplo anterior. Si no encuentra el cliente, retorna *null*.

2. Dinero que los clientes cobraron y el dinero que van a cobrar

Se desarrolló un endpoint para obtener los datos de los pagos de un cliente:

localhost:3333/api/v1/pagos/cliente/:id

Se realiza la solicitud al endpoint indicando el id del cliente deseado.

Los montos se guardan en la base en el mismo formato que los consume de la API provista (file.txt), es decir en centésimos.

La API retorna en formato JSON una lista de los pagos del cliente discriminado en pagos anteriores y futuros (a la fecha del momento de la consulta) así como por moneda (ars y usd). También agrega el dato *'total con descuento'* por cada moneda y si es del pasado o del futuro.

Ejemplo:


```
{
    "anteriores": {
        "ars": [
                {
                    "id": "d8974120c3ff4310810f81695ec9876b",
                    "moneda": 0,
                    "monto_total": 217440275,
                    "total_descuento": 1708011,
                    "total_con_descuento": 215732264,
                    "fecha_pago": 20220307,
                    "id_cliente": "6b43736c8c5a46a0bff26203a47ee16f",
                    "created_at": "2022-04-18 17:29:51",
                    "updated_at": "2022-04-18 17:29:51"
                }
        ],
        "total_ars": [
            {
                "total_con_descuento_ar": 215732264
            }
        ],
        "usd": [],
        "total_usd": [
            {
```

```
                "total_con_descuento_usd": null
            }
        ]
    },
    "futuros": {
        "ars": [
        {
            "id": "c3974120c3ff4310810f81695ec9876a",
            "moneda": 0,
            "monto_total": 217590275,
            "total_descuento": 1557978,
            "total_con_descuento": 216032297,
            "fecha_pago": 20220507,
            "id_cliente": "6b43736c8c5a46a0bff26203a47ee16f",
            "created_at": "2022-04-18 17:29:51",
            "updated_at": "2022-04-18 17:29:51"
        }
        ],
        "total_ars": [
            {
                "total_con_descuento_ar": 216032297
            }
        ],
        "usd": [],
        "total_usd": [
            {
                "total_con_descuento_usd": null
            }
        ]
    }
}
```


3. Transacciones de los clientes

Se desarrolló un endpoint para obtener los datos de las transacciones de un cliente:

localhost:3333/api/v1/transacciones/cliente/:id

La API retorna una lista en formato JSON con todas las transacciones de los clientes registradas en el sistema.

Ejemplo:


```
        [
            {
                "id": "aa23dc48434748caa8c9b1924ecafbb5",
                "monto_total": 7423720,
                "tipo": 1,
                "id_cliente": "6b43736c8c5a46a0bff26203a47ee16f",
                "id_pago": "c3974120c3ff4310810f81695ec9876a",
                "created_at": "2022-04-18 17:29:51",
                "updated_at": "2022-04-18 17:29:51"
            },
```

```
    {
        "id": "aece6f28400242c6af19e98e828b63ba",
        "monto_total": 8605382,
        "tipo": 1,
        "id_cliente": "6b43736c8c5a46a0bff26203a47ee16f",
        "id_pago": "c3974120c3ff4310810f81695ec9876a",
        "created_at": "2022-04-18 17:29:51",
        "updated_at": "2022-04-18 17:29:51"
    }
]
```


4. Borrar datos para pruebas
   Se desarrolló un endpoint de pruebas para borrar los datos de la base de datos relacionados a las entidades registradas: Clientes, Pagos, Descuentos, Transacciones. Dejando intactos los datos de Usuarios y tokens habilitadas y revocadas.

   localhost:3333/api/v1/borrarTodo

Respecto a la base de datos:

Se utilizan los archivos de migraciones contenidos en el código, donde se representan los modelos para cada una de las tablas.

Las tablas generadas son las siguientes: users, clients, pagos, transaccions, descuentos.

Mejoras:

Como siempre sabemos es posible mejorar una aplicación, descontando que los clientes pueden ir variando sus requerimientos o teniendo nuevos, un sistema puede mejorar para cumplir más requisitos de performance permitiendo que sea una aplicación escalable y mantenible entre otras cualidades que puede tener un sistema.

Es importante cumplir con los estándares acordados tanto con el cliente como con el team de trabajo para obtener un código legible, fácil y uniforme para toda la solución.

En este caso algunas mejoras posibles son:

- Definir un modelo diferente de controladores, tal vez otra jerarquía según los requisitos adicionales que puedan surgir en una aplicación real.
- Agregar más comentarios en el código para explicar mejor lo que realiza cada función.

Testing:

- Utilizar alguna herramienta para realizar el testing, definir un plan de pruebas, en lo posible automatizar las pruebas.

Mejoras en los endpoints:

- Cambiaría algunos endpoints para mejorar según los requisitos, por ejemplo, si se quiere que la API esté siempre actualizándose cada 10 minutos del archivo file.txt y los datos de los clientes.
- Agregar más parámetros a la hora de consultar los datos de pagos. Por ejemplo, sumarizar distintos montos (no solo el monto total).

- Tener otro endpoint para los datos de los descuentos o agregarlos a los pagos.
- En el endpoint de transacciones, agregar un parámetro para discriminar entre transacciones aprobadas o rechazadas.

Algunas consideraciones tomadas por falta de información son:

- Los pagos no toman en cuenta si las transacciones son aprobadas o no para sumar el monto total. Los únicos datos que toman en cuenta son la moneda y la fecha de pago.
- No se brindan datos de los descuentos, por lo que la API utiliza una tabla dentro de la base de datos para los descuentos pero no tiene un endpoint para brindar la información.
- Dependiendo de los distintos requerimientos del cliente, podrían agregarse más endpoints para obtener la información adaptada a las posibles consultas.