Prueba Técnica

BackEnd Developer

Rubén Ramírez

HABITS.AI

Ejercicio 1

En (Javascript vailla o Typescript) realice el siguiente programa, documente el proceso de solució n(descripció n), algoritmo(los pasos que realiza el programa) y documente las pruebas unitarias realizadas al programa.
Realice una calculadora de operaciones bá sicas que resuelva las operaciones dada una cadena. Por ejemplo:
Si el usuario introduce 4-7+8+9/2*3 el programa deberá mostrar como resultado 6.5

Consideraciones.

- Deberárealizar las 4 operaciones básicas en la misma cadena suma+,resta-, Multiplicació n *, Divisió n /
- Lalongitud maxima a ingresar de lacadenas erán 20 caracteres.
- Se deberán respetar la prioridad de los operadores.
- Punto extra si se utilizan paréntesis para agrupamiento y multiplicació n
- Punto Extra si se realiza potencia o raıź cuadrada
- Punto Extra si se realizan operaciones de 2 o mas números (23-14+123/3*49)
- Queda prohibido utilizar la funció n EVAL o equivalentes, o en su defecto incluir alguna librerıá que realice todo el proceso.

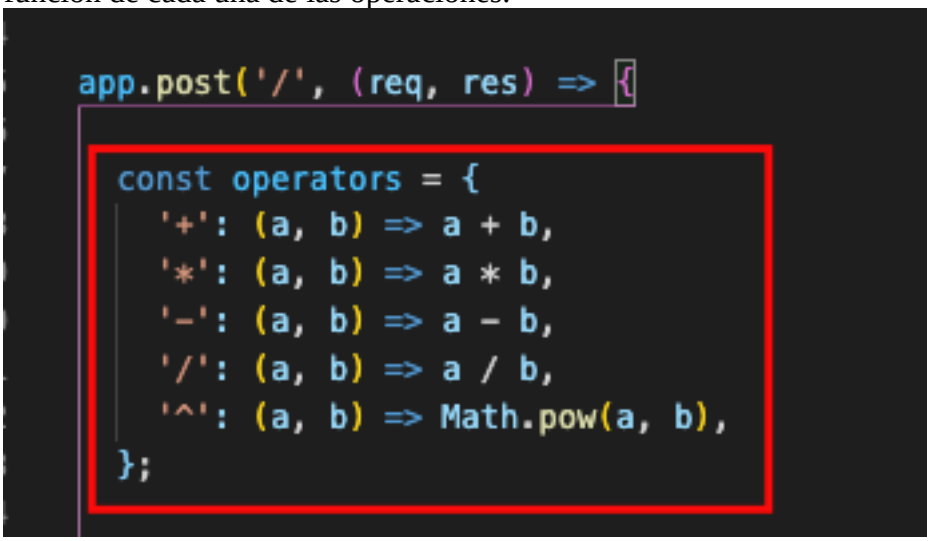
Solución.

Se plantea la utilización de expresión regular para la identificación de operandos, diviendo la frase en operaciónes individuales. Los pasos para realizar la solución son los siguientes:

- Crear un proyecto nuevo de Node, utilizando los siguientes paquetes:
  - Express JS
  - Body-Parser
- Agregamos nuestro punto de entrada tipo post.

```
const express = require('express');
const app = express();
var bodyParser = require('body-parser');

// parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: false }));

// parse application/json
app.use(bodyParser.json());

app.get('/', function (req, res) {
  res.send('Hello World');
});

app.post('/', (req, res) => {
```

- Declaramos un arreglo, donde se tiene por llave el string del operador, con el valor en función de cada una de las operaciones.



```
app.post('/', (req, res) => {

  const operators = {
    '+': (a, b) => a + b,
    '*': (a, b) => a * b,
    '-': (a, b) => a - b,
    '/': (a, b) => a / b,
    '^': (a, b) => Math.pow(a, b),
  };
```

- Se declara constante con el valor de la expresión regular para obtener el operador del string, el objetivo es identificar los operandos (+-/*^); ejem. 4-7+8+9/2*3.

```
const operators_regex = /[+\-*/]/;
```

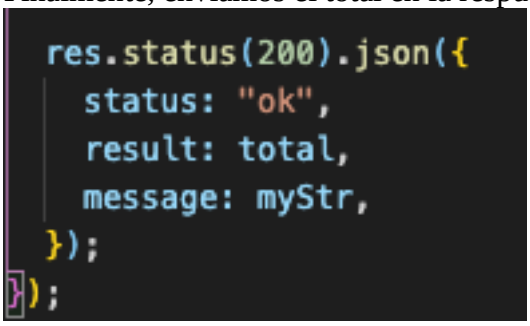
- Declaramos otra constante con expresión regular para obtner los pares de operación de la cadena de caracteres, el objetivo es obtner operador con número; ejem. 4-7+8+9/2*3.

```
const general_regex = /[+\-*/]*(\.\d+|\d+(\.\d+)?)/g;
```

- Declaramos una variable total, en la cual vamos a guardar el valor que se va obteniendo de las operaciones.
- Hacemos match de la frase del usuario con la expresión regular que obtiene el par de operador y número. Y luego recorremos el arreglo que se obtiene del match. En cada iteración, separamos el operador del número y mandamos el valor de total más operador y número de la iteración. Con esto vamos realizando las operaciones de Izquierda a Derecha.

```
let myStr = req.body.op;
myStr = myStr.match(general_regex).join("");

let str_op = myStr.match(general_regex);
let total = 0;

for (let i = 0; i < str_op.length; i++) {
  if (i == 0 && parseFloat(str_op[i])) {
    total = parseFloat(str_op[i]);
  }else{
    let op = str_op[i].match(operators_regex);
    if(op){
      total = operators[op[0]](total, parseFloat(str_op[i].replace(op[0], "")));
    }
  }
}
```

- Finalmente, enviamos el total en la respuesta de la solicitud.



Ejercicio 2

Con ayuda del Framework ExpressJS realice un ejemplo básico de una API REST que permita realizar los siguiente. Simular un sistema de inventario de farmacia.

- POST->Registrar un medicamento o Nombre o Tipo de medicamento o Cantidad o Fecha de registro o Precio o Ubicación
- GET->Traer la lista de medicamentos guardados
- GET->Traer un elemento por id o nombre
- PUT->Editar un documento de la API (por ID)
- DELETE-> Eliminar un documento de la API (por ID)

Podrá utilizar SQLITE, MONGODB, O OBJETOS en memoria para almacenar los datos de la API.

Solución.

Para este ejercicio se crea un nuevo proyecto de node, y se utilizan los siguiente paquetes:

- Axios
- Body-parser
- Express
- Mongoose
- Oauth2-server
- Pug

Se crea un nuevo server, al cual se le configura OAuth 2.0 con Client Credentials como metodo de autenticación, se crea entrada para obtener token, y función middleware para verificar el token en las solicitudes.

```
app.oauth = new OAuth2Server({
    model: require('./model.js'),
    accessTokenLifetime: 60,
    debug: true,
    allowBearerTokensInQueryString: true
});

const server = app.listen(APP_PORT, () => {
    console.log(`App running on port ${APP_PORT}`)
});
```

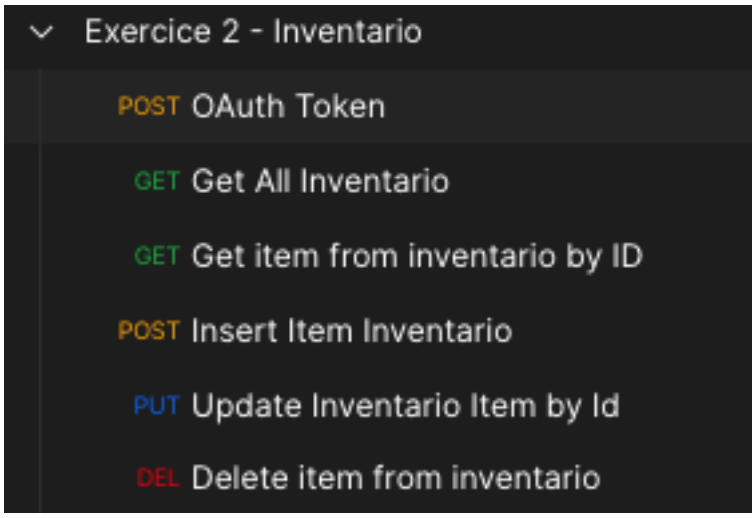
Se crea modelo para la base de datos con mongoose, de los valores en el modelo se asignan el tipo y si son requerido:

```
const mongoose = require('mongoose');


const InventarioSchema = mongoose.Schema({
    nombre: { type: String, required: true },
    tipo_medicamento: { type: String, required: true },
    cantidad: {type: Number, required: true},
    precio: {type: Number, required: true},
    ubicacion: { type: String, required: true },
}, {
    timestamps: true
});

module.exports = mongoose.model('Inventario', InventarioSchema);
```

Se crean archivos de rutas y controlador de las funciones. De igual forma, se agrega el middleware del access token en cada una de las rutas para el CRUD, para que solo se puedan realizar las solicitudes de manera autenticada.

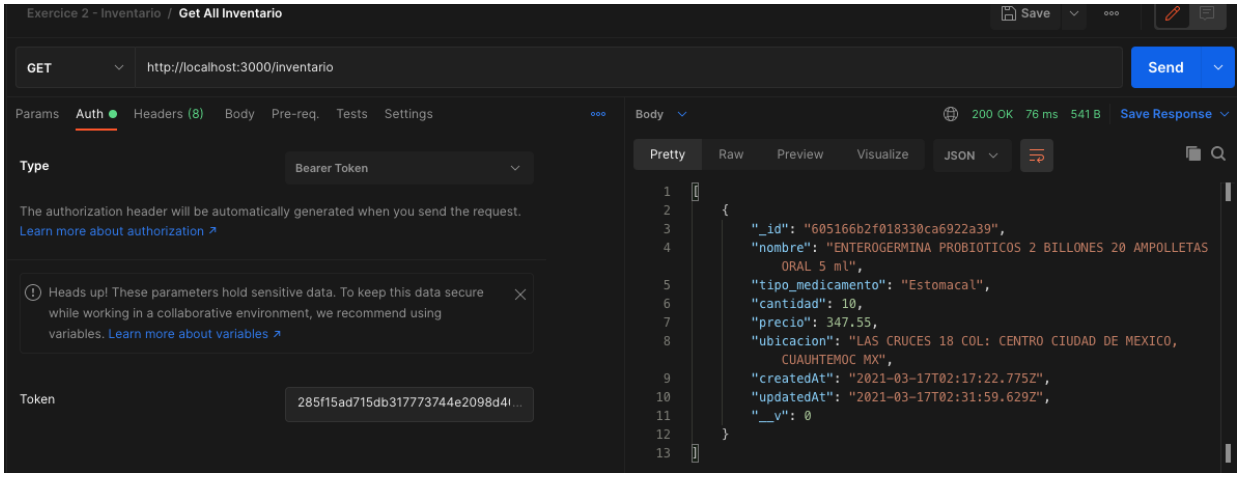
```
// Create a new item
app.post('/inventario', authenticateRequest, inventario.create);

// Retrieve all items
app.get('/inventario', authenticateRequest, inventario.findAll);

// Retrieve a single item with itemId
app.get('/inventario/:itemId', authenticateRequest, inventario.findOne);

// Update a item with itemId
app.put('/inventario/:itemId', authenticateRequest, inventario.update);

// Delete a item with itemId
app.delete('/inventario/:itemId', authenticateRequest, inventario.delete);
```

Se comparte colección de Postman, la cual contiene las siguientes solicitudes:



El proceso para poder realizar cualquier operación del CRUD es:

1. Solicitar Bearer Token con la solicitud “Oauth Token”. Las credenciales para obtner el token son las siguiente:
    a. client_id: UkkaPo4BAe
    b. client_secret: uGddKT1eID>0Op:Ff|7&9fVC#)Dr~R
    c. grant_type: client_credentials
2. Copiar el token que se obtiene de la solcitud, y agregarlo en el apartado de Auth de la solicitud a realizar.
3. Obtener el resultado una vez validado el token.



Ejercicio 3

Con ayuda de Socket.io emitir un evento llamado notificación cada que se crea o se elimina un documento del registro de farmacia.

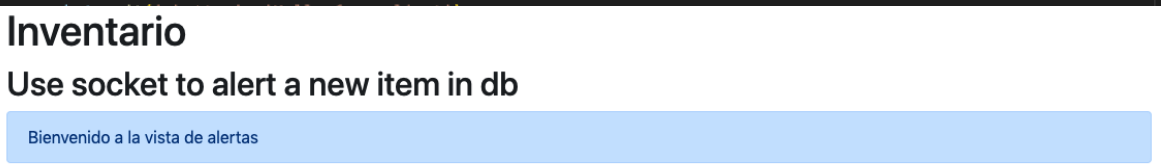
Instrucciones extra que serán tomadas en cuenta para la evaluación (Punto extras)

- Deseable documentar en POSTMAN o SWAGGER la funcionalidad de la API •
- Documentación de EVENTOS y EMITS
- Generación de una pantalla donde se muestre la notificación del EMIT en tiempo real
- Integración de JWT y autenticación para la API (user:test, password:test)

Solución.

Este ejercicio se complementa con el ejercicio 2, complementando el ejercicio anterior se inicia intalando el paquete de Socket io. Una vez instalado, se crea la vista utilizando Pug. En esta vista es donde se van a mostrar las notificaciones.

```
doctype html
html
  head
    title Socket App Test

    link(rel='stylesheet', href='css/main.css')
    link(rel='stylesheet' href='https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css' integrity='sha384-

    //- script(src='https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.0.3/socket.io.js')
    script(src='https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js')

  body
    .container
        h1.title Inventario
        h2 Use socket to alert a new item in db
        #alerts-div
            .alert.alert-primary(role='alert') Bienvenido a la vista de alertas

    script(src='https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js' integrity='sha384-ApNbgh9B+Y1Q
    script(src='https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js' integrity='sha384-JZR6Spejh4U02d8jOt
    script(src='http://localhost:3000/socket.io/socket.io.js')

    script.
        const socket = io.connect('http://localhost:3000/')
```



Se destina un div con el ID “alerts-div” para poder agregar las alertas desde el socket. En el servidor se hace set up de socket io.

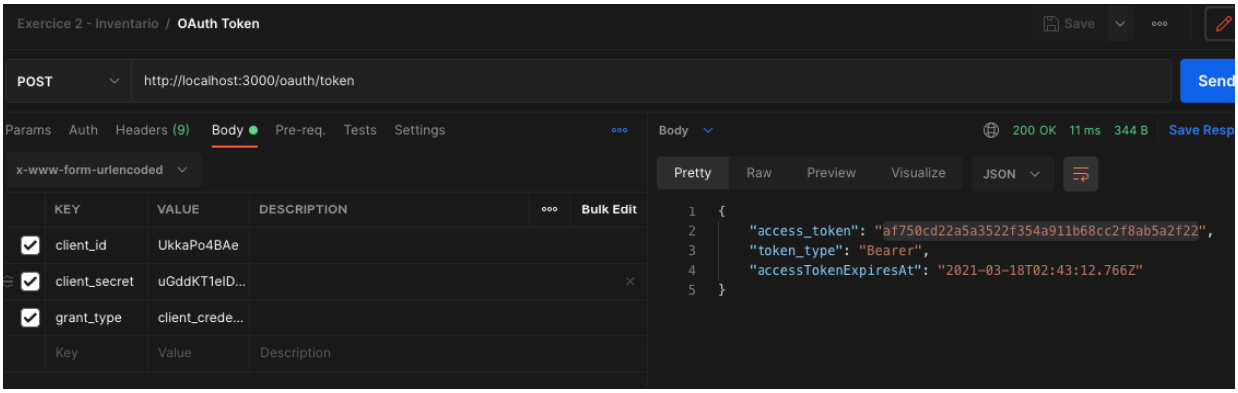
```
const io = require('socket.io')(server);

io.on('connection', (socket) => {
    console.log('a user connected')
    socket.on('chatter', function(message) {
        console.log('message : ' + message);
    });
});

app.set('socketio', io);
```

Y desde la vista se realiza la conexión con las funciones correspondientes para escuchar cuando se disparen los eventos de añadir un nuevo item a la bd y el de eliminar.

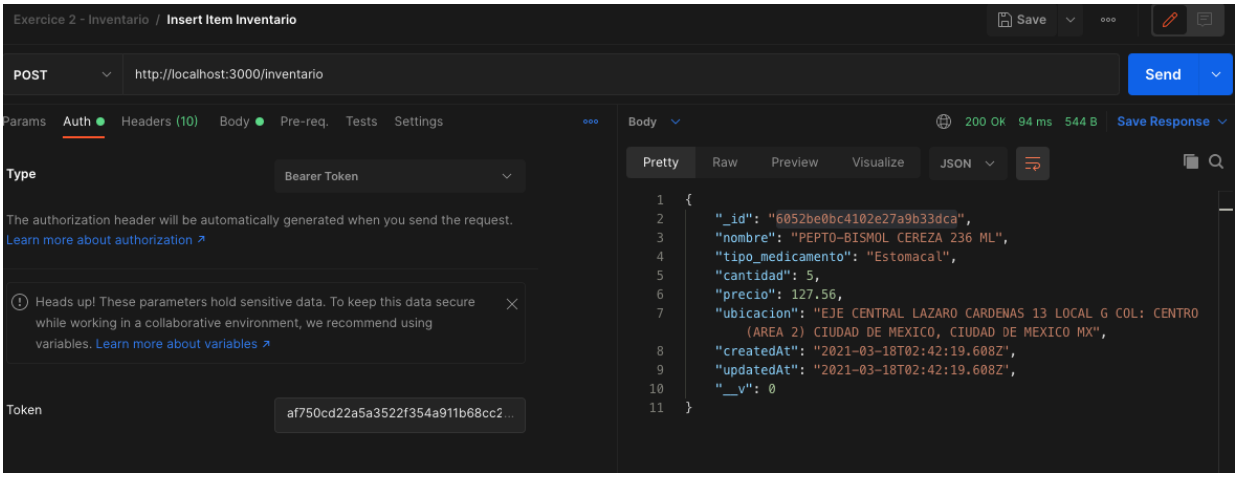
```
script.
    const socket = io.connect('http://localhost:3000/')
    socket.emit('chatter', 'Hello from client');
    socket.on('Alert New Item', function(msg) {
        var html_alert = `<div class="alert alert-success" role="alert">
                    <h4 class="alert-heading">${msg.message}</h4>
                    <hr>
                    <p class="mb-0">ID Item: ${msg.item._id}.</p>

                </div>`;
        $("#alerts-div").append(html_alert);
    });

    socket.on('Alert Item Deleted', function(msg) {
        var html_alert = `<div class="alert alert-danger" role="alert">
                    <h4 class="alert-heading">${msg.message}</h4>
                    <hr>
                    <p class="mb-0">ID Item: ${msg.item._id}.</p>
                </div>`;
        $("#alerts-div").append(html_alert);
    });
```

Se realizan las pruebas de insertar y eliminar items en la base de datos desde Postman y se valida con la vista de que se estan mostrando las alertas correspondientes:

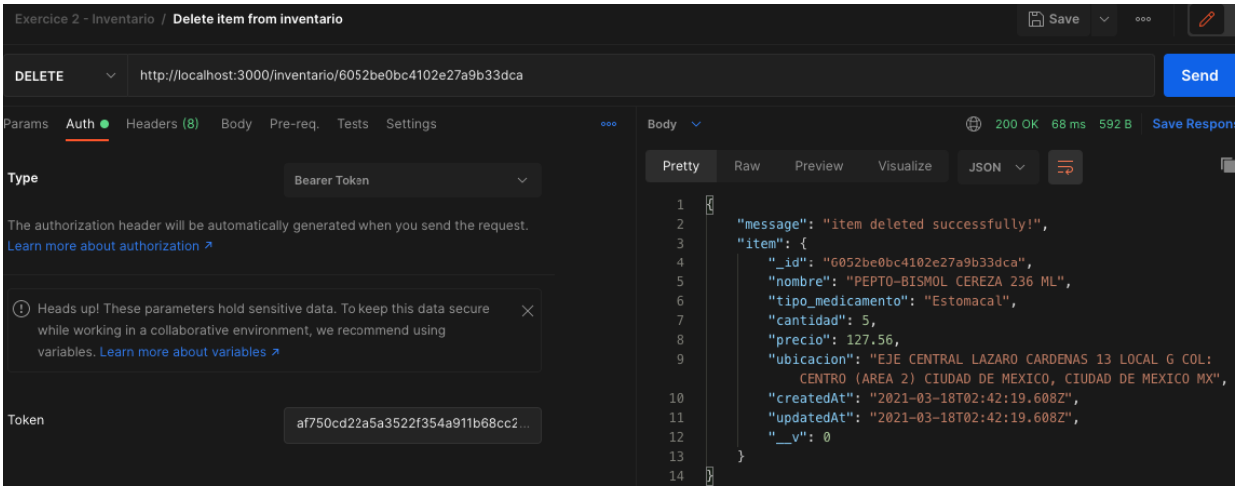
- Se obtiene Token



- Se inserta item a la base de datos



- Se elimina de la base de datos



- Se muestran las alertas en la vista.