ANEXO D
SOFTWARE SSPA

Versión *1.0*

28/04/2024

INFORMACIÓN DEL PROYECTO

Autores			
Nombre completo	Capozzelli, Lucas Santiago	Camargo, Julián	Tedesco, Facundo
Legajo	42894	42741	42742
e-mail	lucascapozzelli@gmail.com	julicmrgo@gmail.com	Facu.Tedesco96@gmail.com

Tutor	Ing. Néstor Manzur
Director	Ing. Carlos Taffernaberry
Jurado	Ing. Carlos Taffernaberry
Año Académico	2024
Responsable de la cátedra	Ing. Antonio Álvarez Abril

Empresa / Cliente / Laboratorio	Instituto Nacional de Tecnología Agropecuaria (INTA)
Patrocinador (Sponsor)	Bodega Experimental del INTA

1- BACKEND-SPA

1.1- Estructura del código fuente (index.js)

El archivo index.js sirve como el punto de entrada principal para el servidor backend. En este archivo, se configuran y definen los elementos esenciales para el funcionamiento del servidor. Define cómo se manejan las solicitudes, qué funcionalidades se aplican a través de middlewares y cómo se gestionan los puntos finales de la API.

A continuación, se proporciona una descripción más detallada de su funcionalidad:

Al principio del archivo, se crea una instancia de la aplicación Express utilizando express(). Esta instancia actúa como el objeto central para la definición de rutas, middlewares y la configuración del servidor. Se establece un puerto en el que el servidor Express estará escuchando las solicitudes entrantes. El valor del puerto se obtiene de la variable de entorno PORT si está definida; de lo contrario, se utiliza el puerto predeterminado 8080. Se importan los módulos path y fs para trabajar con rutas de archivos y el sistema de archivos, respectivamente. Estos módulos son esenciales para la lectura de archivos estáticos y la gestión de rutas en el servidor.

```
const express = require('express');
const { Pool } = require('pg');
const app = express();
const port = process.env.PORT ?? 8080;
const path = require('path');
const fs = require('fs');
```

Se define una clase de excepción personalizada para manejar errores de autenticación no autorizada y una función genérica para manejar errores, clasificando adecuadamente las excepciones y respondiendo con mensajes de error apropiados. Tambien una función para autenticar las solicitudes utilizando un token básico proporcionado en las cabeceras HTTP.

Se define un objeto que actúa como mapeo de nombres cortos a nombres completos de columnas en la base de datos. Facilita la interpretación de datos recibidos.

```
class UnauthorizedException extends Error {
  constructor(message) {
    super(message);
    this.name = this.constructor.name;
    this.code = 401;
  }
}
function errorHandler(error, res) {
  // Manejo de errores, respondiendo con el código de error apropiado
y un mensaje JSON.
}
function authenticate(basic_token) {
  // Verificación de la autenticación básica mediante comparación del
token proporcionado.
```

```
}
const columnName = {
  // Mapeo de nombres cortos a nombres completos de columnas en la
base de datos.
}
```

Se configuran middlewares utilizando el método app.use(). En particular, se utiliza express.json() para analizar datos JSON en las solicitudes entrantes y express.static("public") para servir archivos estáticos desde la carpeta 'public'. Los middlewares se ejecutan antes de manejar las solicitudes y permiten realizar acciones adicionales, como el análisis de datos o la gestión de archivos estáticos.

```
app.use(express.json());
app.use(express.static("public"));
```

Se definen los puntos finales de la API del servidor Express. Cada punto final está asociado con un tipo de solicitud HTTP (GET, POST, etc.) y tiene una función de manejo específica. Por ejemplo, el punto final /insert maneja las solicitudes POST para la inserción de datos en la base de datos.

```
app.post('/insert', async (req, res) => {
  // Lógica para el endpoint de inserción de datos.
});
app.post('/log', async (req, res) => {
  // Lógica para el endpoint de inserción de logs.
});
app.get('/etc', async (req, res) => {
  // Lógica para el endpoint de consulta de datos específicos.
});
app.get('/', (req, res) => {
  // Lógica para el endpoint raíz, sirviendo la página de inicio.
});
app.get('/ping', async (req, res) => {
  // Lógica para el endpoint de recepción de alarmas de ping.
});
```

Se utiliza el método app.listen() para iniciar el servidor Express en el puerto configurado. Una vez que el servidor está en funcionamiento, se imprime un mensaje en la consola indicando el puerto en el que está escuchando. Además de gestionar los puntos finales, el servidor también sirve la página de inicio desde la carpeta 'public'. El código está diseñado para capturar cualquier excepción que pueda ocurrir durante la ejecución y gestionarla de manera adecuada mediante la función errorHandler.

A continuación, el código fuente del archivo index.js:

```
const express = require('express');
const { Pool } = require('pg');

const app = express();
const port = process.env.PORT ?? 8080;
const path = require('path');
```

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

const WETWEIGHT_QUERY = `
      SELECT wetweight
      FROM spa.wetweights
      ORDER BY id DESC
      LIMIT 1
    `;

class UnauthorizedException extends Error {
  constructor(message) {
    super(message);
    this.name = this.constructor.name;
    this.code = 401;
  }
}

function errorHandler (error, res) {
  if(error instanceof UnauthorizedException) {
    error.message += " at authentication";
    console.error(error.message);
    res.status(error.code).json({error: error.message})
  } else {
    console.error(error.message);
    res.status(500).json({ error: 'Internal Server Error' });
  }
}

function authenticate(basic_token) {
  if(basic_token != process.env.BASIC_AUTH) {
    throw new UnauthorizedException("Unauthorized exception");
  }
  console.log("Login succesfully!");
}

const columnName = {
  pl: "pluviometer",
  ws: "windspeed",
  wd: "winddirection",
  l: "leafmoisture",
  h: "humidity",
  r: "radiation",
  t: "temperature",
  pr: "pressure",
  wh: "weight",
  etc: "etc",
  wwh: "wetweight",
  hc: "httpcode",
  msg: "message",
  lv: "level",
  src: "source"
}

app.use(express.json());
```

```
app.use(express.static("public"));

// Context to receive data and insert into the specified table
app.post('/insert', async (req, res) => {
  try {
    authenticate(req.headers.authorization);
    console.debug(`Incoming body: ${JSON.stringify(req.body)}`);
    const { tb, fr } = req.body;
    const frame = fr.substring(tb.indexOf(">") + 1, fr.indexOf("<"));
    var [command, finalFrame] = frame.split("+");
    const sensors = finalFrame.split(";");
    const columns = [];
    const values = [];

    sensors.forEach((sensor) => {
      const [name, value] = sensor.split(":");
      console.log(name + " -> " + value);
      columns.push(columnName[name]);
      values.push(value);
    });

    console.log("Comando: " + command);
    console.log("Tabla: " + tb);
    console.log("Columnas: " + columns);
    console.log("Valores: " + values);

    const database = process.env.PG_DB
    console.debug(`Trying to insert to: ${database}`);

    const pool = new Pool({
      user: process.env.PG_USER,
      host: process.env.PG_HOST,
      database: process.env.PG_DB,
      password: process.env.PG_PASS,
      port: process.env.PG_PORT,
      ssl: require
    });

    const query = `INSERT INTO ${tb} (${columns.join(', ')}) VALUES
(${values.map((_, index) => `$${index + 1}`).join(', ')})`;
    console.log("Query: " + query);
    console.log("Values: " + values);
    await pool.query(query, values);

    res.status(201).json({ message: 'Data inserted successfully' });
    console.debug(`Data inserted succesffuly: "${values}" in
"${columns}" from "${tb}"`);
  } catch (error) {
    console.error(error);
    error.message = "Error on inserting data";
    errorHandler(error, res);
  }
});
```

```
app.post('/log', async (req, res) => {
  try {
    authenticate(req.headers.authorization);
    console.debug(`Incoming log: ${JSON.stringify(req.body)}`);
    const { fr } = req.body;
    const frame = fr.substring(fr.indexOf(">") + 1,
fr.lastIndexOf("<")+1);
    console.log("Incoming frame: " + frame);
    const fields = ['hc', 'msg', 'lv', 'src'];
    const columns = [];
    const values = [];

    fields.forEach((column, index) => {
      if(index === fields.length - 1) {
        const value = frame.substring(frame.indexOf(column) +
column.length + 1, frame.lastIndexOf("<"));
        values.push(value);
      } else {
        const value = frame.substring(frame.indexOf(column) +
column.length + 1, frame.indexOf(fields[index+1]) - 1);
        values.push(value);
      }
      columns.push(columnName[column]);
    })

    console.debug(`Trying to insert to: ${process.env.PG_DB}`);

    const pool = new Pool({
      user: process.env.PG_USER,
      host: process.env.PG_HOST,
      database: process.env.PG_DB,
      password: process.env.PG_PASS,
      port: process.env.PG_PORT,
      ssl: require
    });

    // Construct the SQL query dynamically based on the columns and
values
    const query = `INSERT INTO spa.logs (${columns.join(', ')}) VALUES
(${values.map((_, index) => `$${index + 1}`).join(', ')})`;

    // Execute the query with the values
    await pool.query(query, values);

    res.status(201).json({ message: 'Log inserted successfully' });
    console.debug("Log inserted succesffuly.");
  } catch (error) {
    console.error(error);
    error.message = 'Error inserting log';
    errorHandler(error, res);
  }
});

// Start the server
```

```
app.listen(port, () => {
  console.log(`Server is listening on port ${port}`);
});

app.get('/etc', async (req, res) => {
  try {
    authenticate(req.headers.authorization);
    const pool = new Pool({
      user: process.env.PG_USER,
      host: process.env.PG_HOST,
      database: process.env.PG_DB,
      password: process.env.PG_PASS,
      port: process.env.PG_PORT,
      ssl: require
    });

    const wetweight_result = await pool.query(WETWEIGHT_QUERY);

    const finalResponse = {
      wetweight: wetweight_result.rows[0].wetweight
    };

  console.debug("Returning wetweight value: " +
JSON.stringify(finalResponse));

  res.json(finalResponse);

  } catch (error) {
    error.message = 'Error on query execution';
    errorHandler(error, res);
  }
});

app.get('/',(req, res) => {
  authenticate(req.headers.authorization);
  const indexPath = path.join(__dirname, '../public', 'index.html');
  console.log(indexPath);
  fs.readFile(indexPath, 'utf8', (err, data) => {
    if (err) {
      console.error('Error reading HTML file:', err);
      res.status(500).json({ error: 'Internal Server Error' });
      return;
    }
    res.send(data).status(200);
  });
});

app.get('/ping', async (req, res) => {
  try {
    authenticate(req.headers.authorization);
    const pool = new Pool({
      user: process.env.PG_USER,
      host: process.env.PG_HOST,
      database: process.env.PG_DB,
```

```
      password: process.env.PG_PASS,
      port: process.env.PG_PORT,
      ssl: require
    });

    console.debug("Incoming ping alarm.");

    res.status(200).json({ message: 'Ping alarm received' });

  } catch (error) {
    error.message = 'Error on query execution';
    errorHandler(error, res);
  }
});
```