METEOLOGICA

MANUAL DE INSTRUCCIONES DE LA APLICACIÓN

METEOLIVE

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

Este manual contiene las instrucciones sobre como lanzar la aplicación web de lectura de datos de la prueba de selección para el puesto de Frontend/Fullstack de la empresa Meteologica.

El manual está dividido en dos apartados, el primero (como lanzar la aplicación) describe como descargar los archivos necesarios e instalar los módulos para que la aplicación funcione correctamente. El segundo apartado contiene una descripción pormenorizada de la metodología utilizada para desarrollar el código.

2. CÓMO LANZAR LA APLICACIÓN

2.1. Archivos a descargar

Se puede descargar el proyecto desde mis repositorios en GitHub.

Enlace repositorio GitHub Frontend: https://github.com/Edu-mt/Prueba-Meteologica-Front.git

Enlace repositorio GitHub Backend: https://github.com/Edu-mt/Prueba-Meteologica-Back.git

2.2. Instalación de los módulos y bibliotecas

Esta aplicación ha sido desarrollada utilizando React (framework Javascript) para el frontend y el servidor en NodeJs (Express). Por lo tanto, se describirán los pasos a seguir para tener los paquetes y módulos instalados correctamente.

Lo primero a instalar será node.js: https://nodejs.org/en/. Luego hay que hacer diferentes instalaciones para el Frontend y el Backend (servidor) como se describe a continuación.

FRONTEND

Después de haber descargado los archivos (2.1. Archivos a descargar), se puede abrir el directorio en la carpeta del Frontend en el terminal de un editor de código (ejemplo: Visual Studio Code) y utilizar el comando **npm install** para descargar todas las dependencias necesarias para el funcionamiento de la aplicación.

Para los gráficos se ha utilizado la biblioteca react-chartjs-2.

BACKEND

Como comentado anteriormente, el servidor está desarrollado en NodeJs (Express). Para la instalación se puede abrir el directorio en la carpeta del Backend en el terminal y utilizar el comando **npm install**.

Al terminar todas las instalaciones hay que lanzar el servidor utilizando el comando node server.js.

3. METODOLOGÍA DE DESARROLLO DEL CÓDIGO

En este apartado, se explica el proceso de desarrollo del código.

3.1. Frontend

La aplicación está desarrollada en React, y prácticamente toda la lógica ha sido creada dentro de un componente de clase.

A continuación, se puede observar los componentes a importar y todos los estados creados para el funcionamiento del código.

Se hace la importación del Chart para poder utilizar el gráfico comentado anteriormente. La URL al principio del código se utilizará para realizar las peticiones al servidor, como veremos más adelante.

```
import React, { Component } from "react";
import { Chart } from "../Chart";
import "./Aplicacion.css";

const URL = "http://localhost:3001/";

class Aplicacion extends Component {
  constructor(props) {
    super(props);
    this.state = {
      inicializado: false,
      ultimaTemperatura: "",
      ultimaPotencia: "",
      arrayTemperaturas: [],
      arrayEnergia: [],
      promedioTemperatura: [],
      promedioEnergia: [],
      myIntervalID: 0,
      horaLectura: [],
      numeroPeticiones: 0,
      graficoEnergia: {},
      graficoTemperatura: {},
      avisoTemperaturas: [],
      avisoPotencia: [],
    };
  }
```

A continuación, se observa la función "_datosGrafico". Esta función tiene como objetivo recibir las lecturas de los promedios de las temperaturas, la energía producida y las horas de las lecturas de estos valores y enviárselas al componente Chart, responsable por construir los gráficos. Abajo se encuentra el código del Chart (archivo Chart.js), que contiene la configuración necesaria para que se pinte bien los gráficos en la pantalla.

```
import { Line } from "react-chartjs-2";
export const Chart = ({ chartData }) => {
  return (
    <div>
```

```
      <Line
        data={chartData}
        options={{
          animation: {
            duration: 1,
            },
          responsive: true,
          interaction: {
            mode: "index",
            intersect: false
          },
          stacked: false,
          aspectRatio: 6,
          plugins: {
            legend: false
          },
          scales: {
            y: {
              type: "linear",
              display: true,
              position: "left",
            },
          }
        }}
      />
    </div>
  );
};
```

Más adelante, para hacer la petición al servidor, se utilizará la hora actual, para eso se utiliza la función “new Date()”. Sin embargo esta función retorna los valores de hora, minuto y segundo sin los ceros delante, cuando son menores que 10. Por lo cual se utiliza la función “_addCero”, que tiene la finalidad de añadir ceros a estos valores.

```
_addCero = (i) => {
    if (i >= 0 && i < 10) {
      i = "0" + i;
    }
    return i;
  };
```

La web hace las peticiones al servidor con un intervalo de 5 segundos, pero la representación gráfica se hace con intervalos minutales, por esa razón, se ha creado una función llamada “_promedio” que calcula el promedio de todos los valores de las lecturas realizadas en el periodo de 1 minuto. En el caso de que se reciba un valor “null” (cuando el servidor no encuentra los datos) la función calcula el promedio de los valores restantes y, en el caso de que no haya ninguna lectura dentro del intervalo

de 1 minuto, retorna como valor del promedio “null”. Esto hace con que no se dibuje datos en el gráfico en este periodo.

```
_promedio = (array) => {
    let suma = 0;
    let promedio;
    let longitud = array.length;
    for (let i = 0; i < array.length; i++) {
      if (array[i] === null) {
        longitud = longitud - 1;
      } else {
        suma = suma + parseFloat(array[i]);
      }
    }
    if (longitud === 0) {
      promedio = "null";
    } else {
      promedio = suma / longitud;
    }
    return promedio;
  };
```

Esta función promedio es llamada dentro de la función “_datosMinutales”, que recibe los valores de temperatura, potencia y hora de la lectura, obtiene los promedios de los datos de temperatura y energía producida y los almacena en arreglos (arrays) que se pintarán en los gráficos. Para el caso de la energía, se convierte los datos recibidos de potencia (kW) a energía producida (kWh). En general, dado que las peticiones son realizadas de 5 en 5 segundos, la función está hecha para que a cada 12 peticiones (numero de intervalos de 5 segundos en 1 minuto) se limpie el array auxiliar responsable por rellenar el array de los promedios y entonces todo el proceso vuelve a repetirse.

```
  _datosMinutales = (temperatura, potencia, hora) => {
    let temperaturas = this.state.arrayTemperaturas;
    let arrayMediaTemperatura = this.state.promedioTemperatura;
    let energias = this.state.arrayEnergia;
    let arrayMediaEnergia = this.state.promedioEnergia;
    let horaEntrada = this.state.horaLectura;

    temperaturas.push(temperatura);
    this.setState({ arrayTemperaturas: temperaturas });

    let energia = potencia / (60 * 12);
    energias.push(energia);
    this.setState({ arrayEnergia: energias });

    if (this.state.numeroPeticiones === 12) {
      this.setState({ arrayTemperaturas: [] });
```

```
      this.setState({ arrayEnergia: [] });
      this.setState({ numeroPeticiones: 0 });

      arrayMediaTemperatura.push(this._promedio(temperaturas));
      this.setState({ promedioTemperatura: arrayMediaTemperatura });

      arrayMediaEnergia.push(this._promedio(energias));
      this.setState({ promedioEnergia: arrayMediaEnergia });

      horaEntrada.push(hora);
      this.setState({ horaLectura: horaEntrada });

      this._datosGrafico();
    }
  };
```

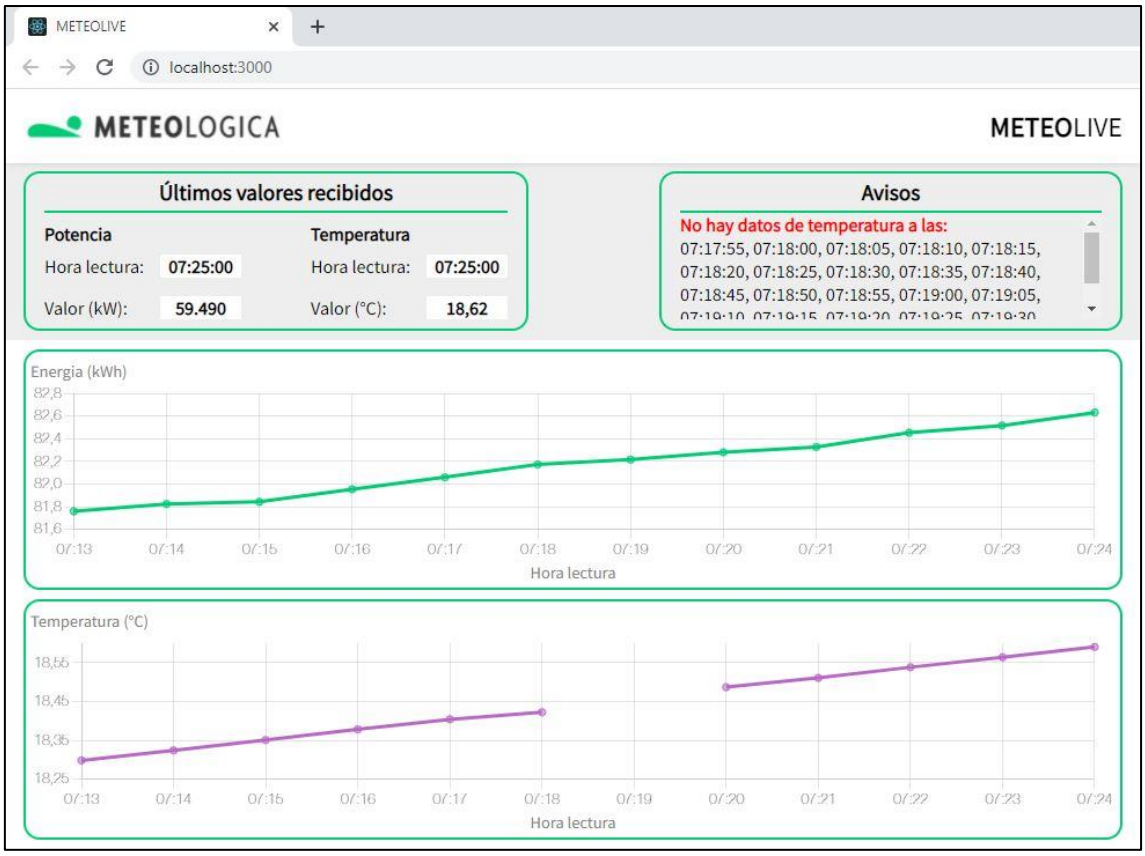
Con todas estas funciones declaradas anteriormente, se utiliza la función "_recibirDatos" que tiene por objetivo hacer las peticiones al servidor, convertir los datos recibidos (temperatura de dK a grados Celcius y potencia de MW a kW) y envasárselos a la función "_datosMinutales" comentada anteriormente.

```
  _recibirDatos = async () => {
    this.setState({ numeroPeticiones: this.state.numeroPeticiones + 1 });
    let alertTemperatura = this.state.avisoTemperaturas;
    let alertPotencia = this.state.avisoPotencia;
    let temperaturaCelsius;
    let potenciaKW;
    let horaMin;

    let fecha = new Date();
    let hora = this._addCero(fecha.getHours());
    let min = this._addCero(fecha.getMinutes());
    let seg = this._addCero(fecha.getSeconds());

    if (seg % 5 !== 0) {
      let redondeado = Math.floor(seg / 5) * 5;
      seg = this._addCero(redondeado);
    }

    let horaActual = `${hora}:${min}:${seg}`;

    let promesaResueltaPeticion = await fetch(`${URL}${horaActual}`);

    let json = await promesaResueltaPeticion.json();

    if (json.temperatura === null) {
      alertTemperatura.push(horaActual);
      this.setState({ avisoTemperaturas: alertTemperatura });
      temperaturaCelsius = null;
```

```
    } else {
      temperaturaCelsius = (
        parseFloat(json.temperatura.value) / 10 - 273.15).toFixed(2);
      this.setState({
        ultimaTemperatura: {
          time: json.temperatura.time,
          value: Intl.NumberFormat().format(temperaturaCelsius),
        },
      });
    }

    if (json.potencia === null) {
      alertPotencia.push(horaActual);
      this.setState({ avisoPotencia: alertPotencia });
      potenciaKW = null;
    } else {
      potenciaKW = (parseFloat(json.potencia.value) * 1000).toFixed(0);
      this.setState({
        ultimaPotencia: {
          time: json.potencia.time,
          value: Intl.NumberFormat().format(potenciaKW),
        },
      });
    }

    horaMin = `${hora}:${min}`;

    this._datosMinutales(temperaturaCelsius, potenciaKW, horaMin);
  };
```

Hay algunos puntos importantes en esta función: en ella se redondea el valor de los segundos a múltiplos de 5, para que coincida con las lecturas del archivo de datos; en este punto también se generan los **avisos*** de cuando faltan lecturas; por fin, esta función también tiene la finalidad de obtener las ultimas lecturas de datos que saldrán en la pantalla.

***avisos:** se pintan en la pantalla en un cuadro a la derecha los horarios en los cuales no hubo lecturas, como se puede observar en la imagen siguiente (simulación realizada eliminando datos de las 07:17:55 a las 07:19:40).



Después hay el "componentDidMount()" que se invoca inmediatamente después de que el componente se monte, en este método se inicia el temporizador (setInterval) responsable por ejecutar la función "_recibirDatos" en intervalos de 5 segundos.

Al final, está el componentWillUnmount() que, para el caso en que se desmonte el componente, se realice la invalidación del temporizador iniciado anteriormente.

```
componentDidMount() {
   let myIntervalID = setInterval(this._recibirDatos, 5000);
   this.setState({ myIntervalID: myIntervalID });
 }

 componentWillUnmount() {
   clearInterval(this.state.myIntervalID);
 }
```

Al final del código está el render, con todos los elementos que se renderizarán en la pantalla durante el proceso ya explicado.

3.2. Backend

Como comentado anteriormente, para el servidor se utiliza Express "const express = require("express")". Se define también el puerto 3001, donde el servidor escuchará las peticiones. Luego se define la constante para usar la biblioteca js-yaml "const yaml = require("js-yaml")" y poder leer el archivo .yml.

A continuación, se configura las cabeceras y el acceso CORS para no haber problemas con el acceso CORS y que se fallen las peticiones.

El servidor entonces recibe la hora actual desde el Frontend, busca esta hora en los datos y devuelve la temperatura y la potencia encontrada. En el caso de que no se encuentra, devuelve “null” al front.

```
const express = require("express");
const app = express();
const port = 3001;

const fs = require("fs");
const yaml = require("js-yaml");

app.use((req, res, next) => {
  res.header("Access-Control-Allow-Origin", "*");
  res.header(
    "Access-Control-Allow-Headers",
    "Authorization, X-API-KEY, Origin, X-Requested-With, Content-Type,
Accept, Access-COntrol-Allow-Request-Method"
  );
  res.header("Access-Control-Allow-Methods", "GET, POST, OPTIONS, PUT,
DELETE");
  res.header("Allow", "GET, POST, OPTIONS, PUT, DELETE");
  next();
});

app.get("/:hora", async (req, res) => {
  let hora = req.params.hora;
  try {
    let fileContents = fs.readFileSync("./data.yml", "utf8");
    let data = yaml.load(fileContents);

    function buscarTemperatura(hora) {
      return new Promise((resolve, reject) => {
        let resultado = null;
        for (let i = 0; i < data.temperature.values.length; i++) {
          if (data.temperature.values[i].time === hora) {
            resultado = data.temperature.values[i];
            break;
          }
        }
        resolve(resultado);
      });
    }

    function buscarPotencia(hora) {
      return new Promise((resolve, reject) => {
        let resultado = null;
```

```
        for (let i = 0; i < data.power.values.length; i++) {
          if (data.power.values[i].time === hora) {
            resultado = data.power.values[i];
            break;
          }
        }
        resolve(resultado);
      });
    }
    let resultadoTemperatura = await buscarTemperatura(hora);
    let resultadoPotencia = await buscarPotencia(hora);

    res.status(200).json({
      temperatura: resultadoTemperatura,
      potencia: resultadoPotencia,
    });
  } catch (e) {
    console.log(e);
  }
});

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