Proyecto 4

Eliaser Alejandro Concha Sepúlveda

1. Acerca de este proyecto

1.1. Requisitos

Adicional a Python y su gestor de paquetes pip, es necesario contar con NodeJS instalado.

1.2. Instalación

Para entornos de desarrollo, es muy recomendado trabajar con entornos virtuales, como por ejemplo, virtualenv para Python.

Instalar las dependencias necesarias para Python con el comando: *pip install -r requirements.txt*

Navegar hasta el directorio llamado frontend, con el comando: *cd frontend*

Instalar React y otras dependencias necesarias para NodeJS: *npm i*

1.3. Ejecución

Una vez instalado los módulos necesarios para el proyecto, navegamos un directorio atrás con el comando *cd ..* y ejecutamos *python index.py* para iniciar el entorno del backend.

Una vez arrancado el backend, procedemos a navegar hasta el directorio frontend, utilizando el comando: *cd frontend* y utilizamos el comando *npm run dev* para ejecutar el cliente desarrollado para el frontend, utilizamos la dirección http que devuelve la consola para acceder mediante un navegador al proyecto.

2. Monitoreo Ambiental

2.1. Backend con Flask

Inicialmente, se definió una lista de elementos, que almacenan las estaciones ambientales presentes en la ciudad de Temuco, junto con sus ubicaciones. Luego, se define la ruta "get-stations", junto con el método get para devolver la lista de estaciones generadas anteriormente. Para consumir desde el frontend y poder generar los marcadores en el mapa de la ciudad para cada estación.

```
stations = [
    {"id": 1, "x": -38.738223, "y": -72.601198},
    {"id": 2, "x": -38.725602, "y": -72.573003},
    {"id": 3, "x": -38.734508, "y": -72.583217},
    {"id": 4, "x": -38.731846, "y": -72.605189},
    {"id": 5, "x": -38.731712, "y": -72.588324}
]

@Station.route("/get-stations", methods=["GET"])
def get_stations():
    return jsonify(stations)
```

Con respecto a la información medio ambiental a entregar, se genera una ruta llamada "set-data". Esta recibe como paramétros de la URL tanto el id de la estación como la cantidad de datos a entregar por cada material particulado, generando valores aleatorios entre 5 y 20 para cada material particulado, devolviendo un JSON para ser consumido desde el frontend,

```
@Station.route("/set-data/<id_station>/<number>", methods=["GET"])
def generate_data(id_station, number):
    material = {
        "station_id": id_station,
        "01": [ra.randint(5,20) for e in range(int(number))],
        "25": [ra.randint(5,20) for e in range(int(number))],
        "10": [ra.randint(5,20) for e in range(int(number))],
        "labels": [e for e in range(int(number))]
    }
    if id_station != "0":
        return jsonify(material)
    return jsonify({"msg": "error"})
```

2.2. Frontend con React

El frontend es la parte que llevó mayor trabajo, A partir de aqupi lo quise realizar con React, ya que es una librería de JavaScript que me gusta bastante, y es bastante popular a la hora de generar interfaces gráficas basadas en componentes reutilizables.

2.2.1. Módulos

Con respecto a los principales módulos de terceros para el desarrollo de esta parte del proyecto se utilizó un paquete de Chart JS adaptado para utilizarse con React, su finalidad es generar gráficos de líneas para mostrar la información cuando se hace click sobre cualquier marcador disponible en el mapa.

El segundo módulo que se utilizó fue React Leaflet, una librería bastante amigable para generar mapas interactivos con el usuario.

2.2.2. Resumen de la estructura del proyecto

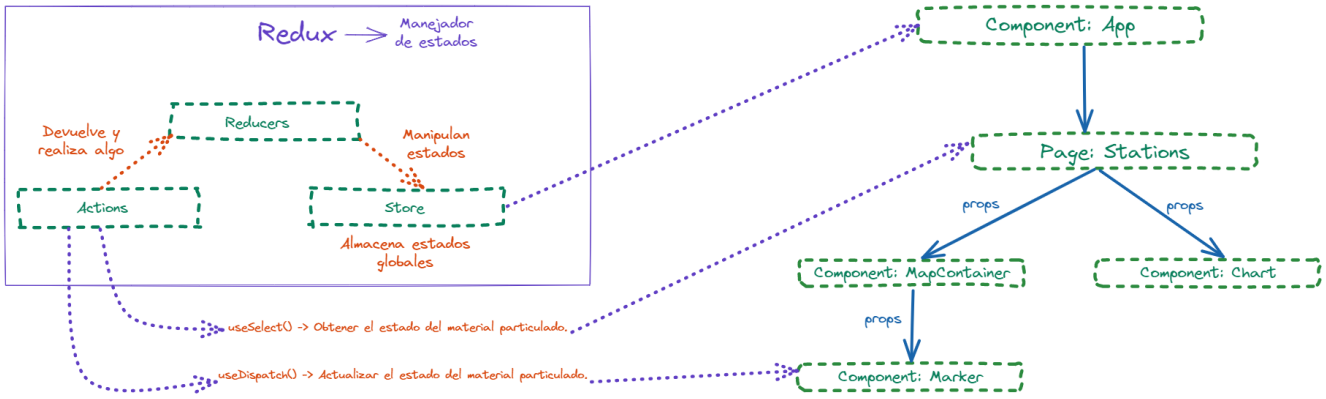

Figura 1: Resumen de la estructura del frontend)

Estructura principal: Desde la parte derecha de la figura 1, se visualiza por medio de un árbol, la estructura con los componentes principales de la aplicación, Primero que nada, incorpora de un componente padre conocido como App, en él se renderiza las rutas y/o componentes asociados a vistas principales, como el componente Stations, el cual es una vista generada para renderizar tanto el mapa como el gráfico con la información de la estación ambiental seleccionada. El componente Stations renderiza dos componentes hijos llamados MapContainer y Chart. MapContainer es el componente principal dónde se renderiza el mapa con la ciudad de Temuco, en ella se renderizan otros componentes a utilizar como los son los marcadores o markers, que proviene a través del recorrido de un custom hook conocido como useStations. El marcador recibe props o propiedades para renderizar información en el componente, principalmente la posición para indicar en que parte del mapa se debe renderizar y la id de la estación, que será utilizada para identificar los marcadores y obtener la información según sea el caso. Por último, el

componente Chart, cuya implementación está generada para renderizar cualquier tipo de información en un gráfico de tipo linea, este componente recibe algunas props como lo son los respectivos datasets conformados por una lista de JSON con sus respectivas propiedas para personalizar el gráfico y mostrar los datos provenientes del manejador de estados, y otras props como labels y el id de la estación ambiental para identificar a qué id hace referencia la información que se está renderizando.

Motivación para utilizar Redux: Redux es una herramienta bastante popular para el manejo de estados en React y otros frameworks de JavaScript. En la parte izquierda de la figura 1 se puede visualizar la estructura de esta herramienta y su aplicación en esta parte del proyecto.En primer lugar, Redux es una herramienta para manejar la mutabilidad en los estados de nuestra aplicación. El problema que presenté al desarrollar el frontend, es que a partir de un componente superior en la jerarquía no encontraba una manera eficiente de poder actualizar un estado importante que está relacionado al manejo de los datos. Si bien, al incorporar un estado al componente marcador y solicitar los datos al hacer click, mi problema se resumía en que no disponía de una solución para actualizar el componente Chart que se encuentra renderizado fuera del componente Marcador.

¿Cómo funciona Redux? Redux incorpora de 3 elementos esenciales, Store, Reducers y Actions. Los reducers corresponden a una serie de funciones que permiten la mutabilidad de un estado en específico, estas funciones explican cómo determinado estado es actualizado, estos reciben como argumento el state (opcional) y un action, del cual solamente interesa su propiedad payload el cual corresponde al nuevo valor del estado. El segundo elemento son los actions o acciones, los cuales son objetos JSON de Javascript pero cuya particularidad radica en que gestiona el envío y obtención de información al store mediante los métodos dispatch y select. El store en terminos sencillos administra todos los estados que nosotros especifiquemos permitiendo el acceso a los actions para obtener y manipular estos. Dispatch se utiliza para actualizar estados, mientras que Select se utiliza para obtener un estado.

2.3. Componentes

2.3.1. Marker.jsx (Marcador)

```
export const Marker = ({position, id}) => {

    // Hook de Redux para actualizar un estado almacenado en el store
    const dispatch = useDispatch()

    /*
    Custom hook que realiza una petici n a la ruta set data del backend y
    obtener 30 elementos para cada material particulado con respecto al id
    asociado al marcador de una estaci n ambiental.
    */
    const mat = useMaterial(id,30)

    /*
    Returnar el marcador en la ubicaci n de la estaci n ambiental dada por el backend,
    React Leaflet ofrece una propiedad eventHandlers para que mediante un JSON se definan
    todos los eventos como una propiedad y su respectiva funcionalidad, para este caso,
    actualizar el estado del material particulado definido en el store.
    */
    return <Marcador position={position} eventHandlers={{
        click: () => {
            dispatch(updateMaterialParticulado(mat))
        }
    }}>
        <Popup>
            <h2 style={{textAlign: 'center'}}>Estaci n Ambiental: {id}</h2>
            <span>Ubicaci n: {position[0]},{position[1]}</span>
            <br />
        </Popup>
    </Marcador>
}
```

2.3.2. Chart.jsx (Gráfico)

```
// Definir los componentes de CHART JS a utilizar globalmente en todos los gr ficos que se
    generen
ChartJS.register(
    CategoryScale, LinearScale, PointElement, LineElement, Title, Tooltip, Legend
)
```

```
/*
Mediante desestructuraci n , se definen las propiedades correspondientes para que este componente
    funcione.
@data -> lista de objetos JSON con las propiedades como tipo de gr fico y color de su respectiva
    informaci n a representar
@labels -> Valores a insertar en el eje horizontal (x)
@station_id -> El n mero que representa a la estaci n ambiental
*/
export const Chart = ({data, labels, station_id}) => {

    // Configurar y/o personalizar el gr fico
    const options = {
        responsive: true, // Gr fico adaptable a distintas resoluciones de pantalla
        plugins: {
            legend: {
                position: 'top', // Posicionar la leyenda en la parte superior del gr fico
            },
            title: { // Mostrar y definir un t tulo al gr fico
                display: true,
                text: `Estaci n Ambiental: ${station_id}`,
            },
        },
        scales : { // Definir o mostrar un t tulo a cada eje
            y: {
                title : {
                    display: true,
                    text: 'ug/m3'
                }
            },
            x: {
                title : {
                    display: true,
                    text: 't(d)'
                }
            }
        }
    }

    const datasets = { // Las propiedades de cada datasets vienen incluidas en una lista que viene
     en data
        labels,
        datasets: data
    }

    return <Line options={options} data={datasets} />

}

// Definir el tipo de dato de cada propiedad y solicitarlos obligatoriamente
Chart.propTypes = {
    data: PropTypes.array.isRequired,
    labels: PropTypes.array.isRequired
}
```

2.3.3. Ruta: Stations.jsx (Vista principal)

```
export const Stations = () => {

    // Obtener de forma din mica cada una de las estaciones
    const { stations } = useStations()

    // Obtener el estado actual del material particulado.
    const materialState = useSelector(state => state.material)
    console.log(materialState)

    /*
    Para cada material particulado, se desarrolla un JSON con propiedades como su label,
    su informaci n, el color de fondo y borde de la l nea.
    */
    const datos = [
        {
```

```
            label: 'Material Particulado 1',
            data: materialState['01'],
            borderColor: 'rgb(144, 190, 109)',
            backgroundColor: 'rgba(144, 190, 109, 0.5)',
        },
        {
            label: 'Material Particulado 2.5',
            data: materialState['25'],
            borderColor: 'rgb(248, 150, 30)',
            backgroundColor: 'rgba(248, 150, 30, 0.5)',
        },
        {
            label: 'Material Particulado 10',
            data: materialState['10'],
            borderColor: 'rgb(249, 65, 68)',
            backgroundColor: 'rgba(249, 65, 68, 0.5)',
        }
    ]

    return <div className="stations-container">

        <MapContainer center={[-38.7379, -72.6005]} zoom={14.5} scrollWheelZoom={false} style={{
    height: '400px'}}>
            <TileLayer
                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            />

            {
                /*
                Dentro del mapa, se obtiene de forma din mica cada una de las estaciones y para
    cada una se renderiza
                su respectivo marcador en la posici n dada.
                */
                stations.map((element, key) => (<Marker
                    position={[element['x'], element['y']]}
                    key={key}
                    id={element['id']}
                />))
            }
        </MapContainer>

        {
            /*
            Si al obtener el estado del material particulado, existe informaci n cargada.
            S lo en ese casom se renderiza el comoponente del gr fico con las propiedades
    solicitadas
            para el componente.
            */
            materialState['station_id'] !== '' && <Chart data={datos} labels={materialState['
    labels']} station_id={materialState['station_id']} />
        }

    </div>
}
```

3. Rutas GPS Vehículos Ciudad de Temuco

3.1. Backend

3.2. Frontend

3.2.1. Resumen de la estructura del proyecto

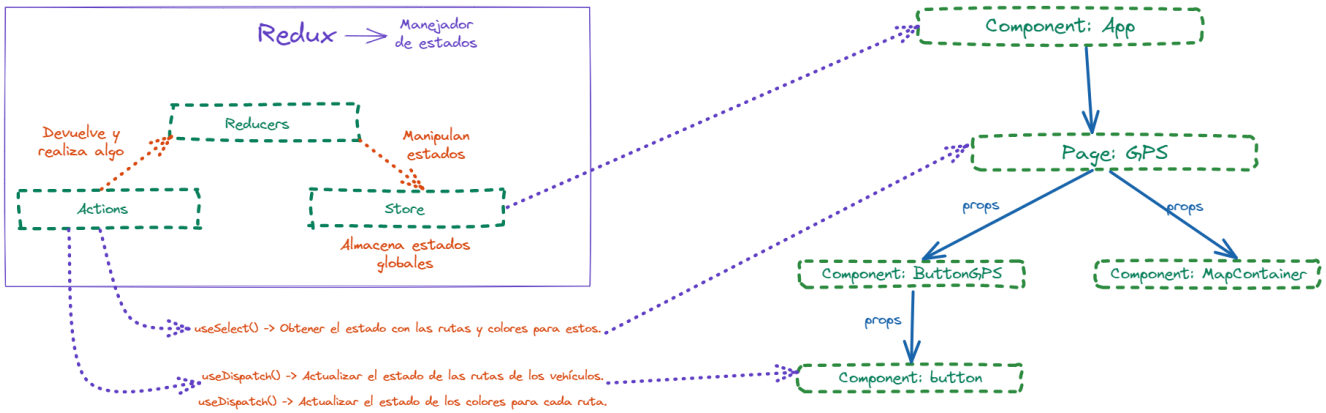

Figura 2: Resumen de la estructura del frontend)

Estructura principal: Desde la parte derecha de la figura 1, se visualiza por medio de un árbol, la estructura con los componentes principales de la aplicación, Primero que nada, incorpora de un componente padre conocido como App, en él se renderiza las rutas y/o componentes asociados a vistas principales, como el componente GPS, el cual es una vista generada para renderizar tanto el mapa como los botones con la ruta del vehículo seleccionado. El componente GPS renderiza dos componentes hijos llamados MapContainer y ButtonGPS. MapContainer es el componente principal dónde se renderiza el mapa con la ciudad de Temuco, en ella se renderizan otros componentes a utilizar como los son los Polyline, un componente importado desde React Leaflet para dibujar líneas dentro de un mapa.

Motivación para utilizar Redux: Redux es una herramienta bastante popular para el manejo de estados en React y otros frameworks de JavaScript. En la parte izquierda de la figura 1 se puede visualizar la estructura de esta herramienta y su aplicación en esta parte del proyecto.En primer lugar, Redux es una herramienta para manejar la mutabilidad en los estados de nuestra aplicación. El problema que presenté al desarrollar el frontend, es que a partir de un componente superior en la jerarquía no encontraba una manera eficiente de poder actualizar un estado importante que está relacionado al manejo de los datos. Si bien, al incorporar un estado al componente ButtonGPS y solicitar los datos al hacer click, mi problema se resumía en que no disponía de una solución para actualizar el componente MapContainer que se encuentra renderizado en una jerarquía superior a ButtonGPS.

3.3. Componentes

3.3.1. GPS.jsx

```
export const GPS = () => {

    // Custom Hook que realiza una petici n al backend y obtiene todos los veh culos
    const vehicles = useAllVehicles()
    // Obtener el estado global para la ruta del veh culo
    const route = useSelector(state => state.route)
    // Obtener el estado global para el color de la ruta
    const color = useSelector(state => state.color)

    return <div>
        <MapContainer center={[-38.7379, -72.6005]} zoom={14.5} scrollWheelZoom={false} style={{
    height: '100vh'}}>
            <TileLayer
                url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
            />
```

```
            <Polyline pathOptions={color} positions={route} />
        </MapContainer>
        {
            /*
            Obtener los veh culos y de manera din mica renderizar un bot n para cada uno de
    ellos.
            */
            vehicles.map((value, key) => {
                return <ButtonGPS key={key} id={value} />
            })
        }
    </div>
}
```