Taller de Programación 2

Examen Final (2 de febrero, 2026)

Enunciado: Sistema de seguimiento de colectivos por paradas

Se desea implementar un backend que reciba y procese reportes de ubicación enviados por colectivos que participan del seguimiento en tiempo real de una línea de transporte público.

Cada colectivo se identifica por un **id alfanumérico de 6 caracteres** (ej.: BUS105) y envía su ubicación en formato GPS (**latitud** y **longitud**) junto con su **velocidad** en km/h.

El sistema debe **almacenar el último reporte** de cada colectivo. Si ingresa un nuevo reporte con el mismo **id**, se debe **actualizar únicamente** su última posición, velocidad y fecha de actualización.

Además, el sistema debe detectar si un colectivo se encuentra **a menos de 80 metros de una parada**. Para esto se utilizará la fórmula de **Haversine** para calcular la distancia entre dos coordenadas GPS:

```
function distanciaGPS(lat1, lon1, lat2, lon2) {
    const R = 6371000; // radio terrestre en metros
    const dLat = (lat2 - lat1) * Math.PI / 180;
    const dLon = (lon2 - lon1) * Math.PI / 180;
    const lat1Rad = lat1 * Math.PI / 180;
    const lat2Rad = lat2 * Math.PI / 180;

    const a =
        Math.sin(dLat / 2) ** 2 +
        Math.cos(lat1Rad) *
        Math.cos(lat2Rad) *
        Math.sin(dLon / 2) ** 2;

    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

    return R * c; // distancia en metros
}
```

Requerimientos del backend:

- Almacenar, actualizar y listar colectivos.
- Validar id (6 alfanumérico), lat/lon numéricos, velocidad numérica >= 0.
- Al recibir una ubicación, calcular la distancia (Haversine) a todas las paradas.
- Si está a < 80m de una parada, devolver alerta con la parada más cercana.

Endpoints requeridos:

1. **POST** /colectivos
   a. Recibe: { id: "BUS105", latitud: -34.6037, longitud: -58.3816, velocidad: x }

      Tras ingresar el colectivo, comparar su posición con la lista de paradas:
      Respuesta: { guardado: true, alerta: { idParada, nombre, distancia } | null }

   b. Validaciones: id de 6 caracteres alfanuméricos, latitud y longitud numéricas y velocidad >= 0. Si no cumple con alguna validación la respuesta será:
      { guardado: false }

2. **GET** /colectivos
   Lista todos los colectivos con su última posición.

3. **GET** /paradas
   Lista paradas configuradas.

Datos del sistema:

El backend debe tener un listado fijo (hardcodeado) de paradas, por ejemplo:

```
[
    { idParada: "P01", nombre: "Plaza Central", latitud: -34.6037, longitud: -58.3816 },
    { idParada: "P02", nombre: "Hospital", latitud: -34.6060, longitud: -58.3850 },
    { idParada: "P03", nombre: "Estación", latitud: -34.6095, longitud: -58.3920 },
    { idParada: "P04", nombre: "Universidad", latitud: -34.5982, longitud: -58.3731 }
]
```

Ubicar este listado en la capa correspondiente dentro del servidor.

El servidor recibirá y responderá desde y hacia el frontend con los datos requeridos en formato JSON. Todas las respuestas deberán estar correctamente adosadas con su código de estado correspondiente, según el resultado de la operación.

Aclaraciones sobre el desarrollo esperado:

1. El proyecto debe incluir únicamente el backend del sistema, utilizando Node.js + express. El formato del servidor es de tipo REST utilizando patrón arquitectónico MVC. Tener en cuenta los lineamientos que propone este desarrollo, especialmente a la hora de elegir las rutas de acceso al sistema.
2. El sistema debe estar correctamente separado en capas y componentes, y esta separación debe estar claramente puesta de manifiesto en la estructura de carpetas y archivos. Entre los componentes que esperamos que estén presentes encontramos: router/controlador, casos de uso, modelo/s, DAO/s y factories (los que correspondan de acuerdo al sistema modelado).
3. Prestar atención al sentido de las dependencias entre los componentes, recordando que las capas más cercanas al negocio no deben estar acopladas a las capas más externas (usualmente de infraestructura). Con esto en mente, *importar módulos o inyectar dependencias según corresponda*.
4. La *validación de datos* es una parte importante del negocio, por lo tanto, observar cómo y dónde realizarla.
5. *No es necesario utilizar una conexión a base de datos real, persistir en el DAO usando memoria ram del servidor.*
6. Recordar el rol de las factories, que nos permiten desacoplarnos de las dependencias de nuestros componentes a la hora necesitar una instancia de los mismos. Recordar esto especialmente a la hora de decidir cómo obtener los casos de uso para invocarlos desde la capa de ruteo.