IIC3103 - Taller de Integración
Departamento Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica

Enunciado Tarea 3

Objetivo

El objetivo de esta tarea es consumir un websocket, para mostrar datos en tiempo real. Esta tarea debe ser resuelta ocupando algún framework de frontend tales como React, Vue, Gatsby, Angular o Svelte.

Trabajo a realizar

Deberán **consumir un websocket** que entregará datos de un centro de control de una flota de camiones mineros. En su página web deberán mostrar la posición en tiempo real de los camiones sobre un mapa, la información del camión y sus posibles fallas, y por último un chat global para los operadores del centro de control.

No se debe guardar información en bases de datos, solo se debe mostrar la información que se alcanzó a consumir desde que se inicia la conexión con el websocket.

Websocket

Para esta entrega se ocupará únicamente la información entregada por el websocket. Para conectarse a este se ocupan los siguientes datos:

Conexión con Websocket

- **Protocolo:** wss://
- **Servidor:** tarea-3-websocket.2021-2.tallerdeintegracion.cl
- **Ruta:** /trucks

Eventos emitidos por el Websocket

POSITION Envía posición actual de un camión.

```
{
    code: String,
    position: [Float, Float]
}
```

Donde **code** hace referencia a un camión y **position**, a la posición actual de este (lat, long). Esta información debe ser mostrada sobre un mapa, mostrando la ruta realizada por el camión desde que se realizó la conexión al websocket. Sobre el camión, o al hacer hover sobre este se debe mostrar su código.

TRUCKS Envía información sobre los camiones

```
[{
    code: String,
    origin: [Float, Float] ,
    destination: [Float, Float],
    driver_name: String,
    status: String
}]
```

Se deberá desplegar en el mapa para cada camión su posición de inicio y fin. Además se deberá poder revisar la información de cada camión de manera intuitiva en la página web.

CHAT: Envía mensajes de los operadores de otros centros de control.

```
{
    date: String
    message: String
    name: String
}
```

Se deberá mostrar los mensajes recibidos en una lista en la aplicación web. Se debe mostrar toda la información del mensaje, tanto el nombre como la fecha y mensaje.

FAILURE: Envía información sobre las fallas que presentan los camiones, donde source indica el origen del problema.

```
{
    code: String
    source: String
}
```

Junto con la información de los camiones, se debe poder ver claramente cuando uno de ellos ha presentado una falla, indicando cuál fue el origen de esta.

FIX: Indica que el camión ha sido atendido por un operador y será reparado.

```
{
    code: String
}
```

Si un camión presentaba una falla al momento de recibir este evento, se debe mostrar como “sin fallas” nuevamente.

Eventos que recibe el Websocket

TRUCKS: gatilla el envío de la información de todos los camiones.

CHAT: recibe un mensaje y envía este mensaje a todos los clientes conectados al websocket. El evento que recibe debe ser enviado en el siguiente formato:

```
{
    message: String
    name: String
}
```

En su solución debe haber un chat, donde se pueden enviar mensajes. También se debe poder elegir un nickname para el usuario.

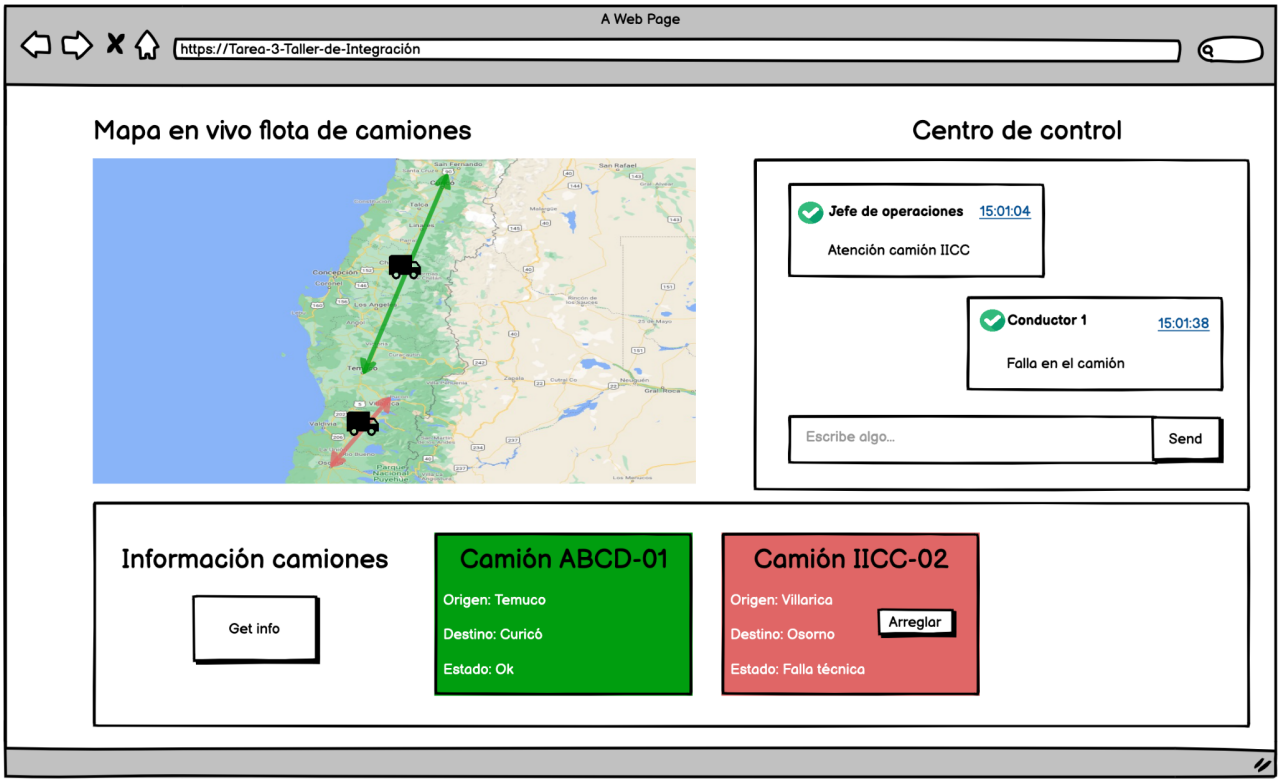
FIX: al igual que el chat, recibe un evento FIX y lo envía a todos los clientes conectados. El evento que recibe debe ser enviado en el siguiente formato:

```
{
    code: String
}
```

Cuando un cambión presenta una falla, el usuario debe poder atender la falla en la aplicación web, enviando este evento al Websocket, que a su vez lo envía a todos los clientes, notificando que el camión ha sido atendido.

Visualización

A continuación se presenta un ejemplo de cómo se podría visualizar esta información:



Como se evidencia en el ejemplo, se debe visualizar en un dashboard un mapa que muestre la posición de los camiones, una ventana de información de estos diferenciando aquellos camiones en buen estado con aquellos que presenten fallas técnicas y finalmente incorporar un chat donde el usuario pueda enviar y recibir mensajes. Cabe mencionar que el ejemplo anterior es solo una referencia gráfica y son libres de posicionar o diseñar los elementos a su gusto.

Entregables

Cada alumno deberá entregar, mediante un formulario publicado en el sitio del curso, las siguientes *url's*:

- URL de acceso a sitio en Github Pages
- URL del repositorio Github

Además cada alumno deberá responder todas las evaluaciones de pares que se le asignen. Se descontará un punto a la tarea por cada evaluación no contestada.

Recomendaciones

Para la resolución de esta tarea se recomienda usar la librería Socket.io, pueden encontrar la documentación en https://socket.io/.

Para desplegar la información en un mapa se recomienda utilizar Open Street Maps, estos son mapas de código abierto gratis para su uso. En react existe React Leaflet, librería que facilita el uso de estos mapas, se recomienda fuertemente buscar una librería como esta ya que facilita la interacción con la api de estos mapas. Pueden encontrar la librería en https://react-leaflet.js.org/.

Fecha de entrega

La tarea se deberá entregar el día 29 de octubre antes de las 18 hrs.

Requisitos mínimos

Las tareas que no cumplan las siguientes condiciones no serán corregidas y serán evaluados con la nota mínima:

- La página web deberá ser pública, accesible desde cualquier dispositivo conectado a internet.
- El código deberá estar versionado en su totalidad en un repositorio Git
- El sitio debe reflejar fielmente el código entregado en el repositorio
- Se recomienda configurar un deploy automático de página estática con Github Pages

Penalizaciones

Se descontarán 0,2 puntos de la nota de la tarea por cada hora de atraso en la entrega, contados a partir de la fecha estipulada en el punto anterior.

Se descontará 1 punto por cada revisión entre pares no contestada.

Cualquier intento de copia, plagio o acto deshonesto en el desarrollo de la tarea, será penalizado con nota 1,1 de acuerdo a la política de integridad académica del DCC.