FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

eolian
UNIVERSIDAD DE CHILE

Reporte de Avance

Telemetría

Área:	Eléctrica
Encargada:	Catalina Domínguez
Subárea:	Telemetría
Encargado:	Maximiliano Vargas
Integrantes:	Analía Banura
	Maximiliano Vargas
Fecha:	27 de Abril, 2020

1. Resumen de avance

Resumen

Se añade el linearProgress de Quasar (con limitaciones de color) para mostrar el SOC (Se buscan alternativas pero sin éxito). Se hace interpolación HSL de color para temperaturas y voltajes como corresponde, en forma genérica con formulas de javascript y con v-bind. Se logra obtener reactividad de ambos gráficos con los states de Vuex (se usa librería chartjs-plugin-streaming cambiando el método *onrefresh* en el obejto *options* de cada gráfico, mediante un *watcher* en las variables computada de Vuex. Se crea un array de datos para cada gráfico en Vuex por una limitación de uso). Por último se logra conectar Socket.io con Quasar, escuchando *eventos* enviados desde la API exterior, y realizar un *dispatch* desde el *listener* en el fornt-end para hacer update en Vuex con los valores nuevos recibidos.

Tareas planteadas (Brainstorm)

1. Usar socket.io en Java para evitar la comunicación innecesaria Java->API->Front, y reemplazarla por Java->Front.
2. Analizar la encriptación de la Xbee. ¿Que ciphers usa? ¿LLave simétrica o asimética? ¿Acuerdo de llaves?.
3. Analizar el comportamiento de calor de la Raspberry y buscar soluciones.
4. Actualizar el diagrama de cableado para la Raspberry Pi 4 y no la 3B+, por el uso de microHDMI y otras cosas. Además es necesario para hacer la placa de telemetría.
5. Analizar frecuencia de mensajes repetidos, para crear una tabla de Huffman dinámica y comprimir envío de datos.
6. Analizar máxima derivada para cada dato. Así implementar protocolo tipo UDP que manda datos descomprimidos una sola vez y luego sólo manda las variaciones de cada dato. Receptor puede pedir datos descomprimidos si lo desea, pero un vez enviados los datos descomprimidos, se empiezan a enviar en forma comprimida automáticamente.
7. Analizar si Websocket del front puede conectarse a una API en la nube, en internet. Esto podría llevar a desarrollos como: Una App de celular para hacer monitoreo por internet a eolian (y como usamos responsiveness basta compilar el front actual para móviles). O bien, desde eolian.cl/seguimiento hacer monitoreo del auto. Requisito: eolian conectado a internet.

Avances

1. Se da con el Linear Progress nativo de Quasar para ser candidato de la barra que muestra el SOC del banco de baterías. Sin embargo no se logra hacer el CSS dinámico para aplicar una interpolación de color. Se prueba y queda reactivo con Vuex, eso es bueno. Alternativas se probaron muchas (como esta o esta) pero no se consiguió el efecto esperado. Así que se decide usar la de Quasar y avanzar en otras cosas.

2. Se indaga aún más en interpolaciones de colores. Se aprende sobre el modelo HSL y RGB para representar colores. Nos damos cuenta que queremos una interpolación HSL para mostrar temperaturas (Que vaya de rojo a azul o viceversa pasando por amarillo, varde y cian). Ahora que tenemos claro el objetivo es fácil econtrar lo que se quiere. Se da con un ejemplo en javascript de interpolación HSL y RGB que será la base para lo que se quiere. Se prueba para temperaturas y queda genial. Por su simple uso, se reemplaza la interpolación para voltajes por esta nueva y también queda genial. Código legible y ordenado. Queda la interpolación de color resuelta.

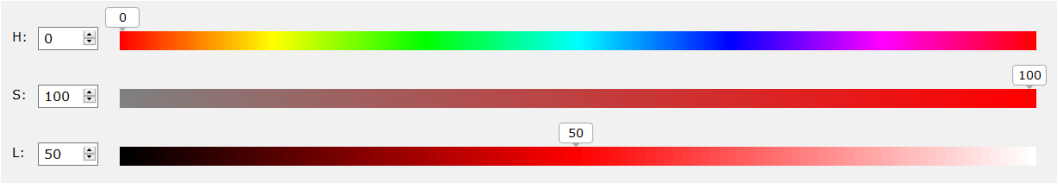

Figura 1: Color en formato HSL

3. Ahora se busca implementar la reactividad de los gráficos de ChartJS, es decir, conectar Vuex (datos lógicos, estado común) con el display de los gráficos. Luego de intentar durante mucho tiempo con la forma nativa o por defecto de realizar esto con la librería vue-cahtjs que usamos antes, no se logra el objetivo. El uso de mixins no funciona, ni tampoco con variables computadas. Además queremos que el gráfico se vea en el tiempo, y estar agregando/quitando puntos en forma manual no daría una visualización fluida porque dependería de a cada cuanto llegan los datos. Se buscan soluciones sin éxito. Se encuentra la librería chartjs-plugin-streaming con una simple instalación se pueden agregar puntos en forma dinámica. Después de indagar nos damos cuenta que la clave esta en el método `onRefresh()` del parámetro `options` que se pasa al gráfico. Este define la forma de cada cuanto, cómo y qué se pone en el gráfico. Entonces se implementa la siguiente solución: Se usará un **watcher** sobre una variable computada de Vuex. Cada vez que la variable cambia se ejecuta el watcher. Entonces, dentro del watcher pondremos que al cambiar la variable en Vuex, el onRefresh del gráfico ahora mostrará el último valor recibido y dejaremos todo lo demás (ventana de tiempo, frecuencia de actualización, etc) intacto. Se prueba y funciona perfectamente. Sin embargo, cuando se intenta implementar en el segundo gráfico, que depende de varios arrays de datos en Vuex (bms, kelly_der, kelly_izq, etc) vemos que el display falla, se debugea con vue-tools y algunos datos quedan en *undefinded*. Se concluye que no pueden haber referencias mixtas (a diferentes partes del state) en el watcher, por eso se decide crear un array exclusivo de datos en Vuex para cada gráfico, y por ejemplo, cuando se actualice el bms, se actualiza el array correspondiente y también el del gráfico donde corresponda. Se prueba tener un array en el state de Vuex por gráfico y funciona de maravilla. Ahora tenemos reactividad de los gráficos!

4. Ahora se quiere implementar Socket.io (webSockets) en el front-end. Esto permite una conexión constante cliente-servidor (full duplex), lo cual permite enviar datos y refrescarlos visualmente sin tener que recargar la página [Referencia]. Para utilizar esto en Vue, existe la librería vue-socket.io que ya se había probado con éxito en el pasado (Diciembre de 2019) para un ejemplo básico. Sin embargo, la forma probada no contemplaba la conexión con Vuex, sino que una actualización simple del campo `data` de cada Componente de Vue, y además estaba siendo utilizada sobre un proyecto en Vue CLI, no Quasar, por lo tanto la instalación cambia. Para instalar socket.io en Quasar se encuentra la siguiente solución, aporte del usuario "lalo.dev". Básicamente hay que crear un archivo `boot` para el socket y activar su uso en `quasar.conf.js`. Se prueba recibir eventos básicos, colocando los métodos en el componente de Vue `App.vue` y se reciben exitosamente, esto quiere decir, que la interfaz del socket queda correcta. Para conectar el evento recibido en el front-end (Cliente) con una `mutation` o `action` de Vuex bastaría aplicar una regla de prefijos como se describe en la documentación de vue-socket.io, sección "Vuex integration". Sin embargo, bajo hartas horas de pruebas no se logra conseguir. Incluso usando librerías alternativas como vue-socket.io-extended e intentado soluciones como esta o esta. Se ocurre la idea de utilizar `dispatch()` para llamar a las **actions** de Vuex desde el método que recibe el evento del socket. Para hacer eso bastaría tener el `this` de Vue para llamar al dispatch. Se verifica que existe esa conexión, bien!. Pero recordemos que nuestra store está modularizada, por lo tanto hay que encontrar una forma de desencadenar una action de un módulo. En este hilo se da la idea de hacer algo como `dispatch('/othermodule/action', null /*or whatever payload*/, root: true)`. Se copia la idea, utilizando `dispatch('/fenix/updateMainData', nuevos_datos)`, donde "fenix" es el módulo en la store, "updateMainData" la action y "nuevos_datos" el array con los datos nuevos que llegan por el socket, y la action efectivamente se ejecuta!!. **En este punto podemos recibir un payload (datos) desde el socket mediante un evento, y mediante un dispatch ejecutar una action de un módulo de Vuex, que hace update de su state (datos internos).** Es lo último que faltaba para completar toda la cadena de transmisión de datos, desde los Arduinos, hasta el front-end.

5. Cambios guardados en GitHub

Figura 2: Visualización anterior de eolian Fénix

Figura 3: Progress Bar, gráficos responsive, CSS dinámico