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: | 08 de Abril, 2020 |

# 1. Resumen de avance

Resumen

Se añaden estilos dinámicos para el Componente del banco de baterías. Se usa correctamente la reactividad de Vue con Vuex. Se hace formateo a tres decimales a los valores de voltaje en la visualización. El front-end queda listo para recibir arrays de datos con socket.io. Se instala vue-dev-tools en Firefox para debuggear mejor.

Tareas planteadas

1. Agregar un Componente para el SOC, puede ser con CSS o un componente de VueJS
2. Ver como arreglar la reactividad (Que no funcionaba, pero se arregla, detalles en Avances)

Avances

1. Se busca intensivamente como hacer el `background-color` del CSS del banco de baterías dinámico entre rojo y verde según el valor guardado en el store de Vuex del módulo correspondiente. Finalmente se encuentra que Vue tiene CSS dinámico por lo cual puede bindearse el estilo a una variable computada. No es precisamente lo que se necesita porque la variable está en el store de Vuex. Pero recordar que el store queda precisamente en variables computadas con `computed: { ...mapState('fenix', ['bms_volt', 'bms_temp'])}`, por lo cual podría pasarse al `v-bind:style` el valor de la variable del store. Pero esta variable devuelve el valor y no un estilo tipo `background-color: rgb(255, 0, 0)`, por lo que hay que hacer algo más. Por otro lado, hay que calcular el color (entre verde y rojo) correspondiente al valor del módulo entre (máx voltaje y mín voltaje), es decir, una interpolación del color. Luego de mucho tiempo buscando, se da con que puede calcularse con un `method` que recibe como input el valor del módulo. Puede retornar en formato HEX o RGB. (En la búsqueda se encontraron cosas bacanes como progressbar con SCSS dinámico en el texto según el background. Ahora bien, enlazar el estilo con un método que depende de una variable computada no es precisamente lo que se sabe que funciona (enlazar directamente con una variable computada), pero vale la pena intentarlo. Se transcribe el ponderado de RGB a JS (Estaba en Perl), se enlaza el estilo y chan chan!. Funciona! Además si se cambia el valor en el store y se recarga la página, también cambia el color. Sin embargo, se crea un botón que llame a una mutación del valor del módulo en el store y el valor no cambia :( (tampoco el color por ende). Es más, se intenta cambiar la velocidad con otro botón y la reactividad tampoco cambia)
2. Para arreglar la reactividad también se busca exhaustivamente por soluciones, pero al final la solución la da la documentación de Vue en la sección de los Arrays. Hay que usar `Vue.set` de la forma `Vue.set(vm.items, indexOfItem, newValue)`, diciendo explícitamente que la forma es estábamos usando: `vm.items[1] = 'x' // is NOT reactive` no funciona. Se implementa con un detalle, hay que importar Vue en las mutations. Se prueba y funciona!! La reactividad ha vuelto :D
3. Con la reactividad resuelta el valor del módulo en el Componente cambia correctamente, y el color también!! Por lo que enlazar el estilo a un método que depende de una variable computada y que retorna un estilo sirve!

4. Por otro lado, al estar correcta la reactividad y con el botón cambiar de a 0.01 un módulo, nos encontramos con que se ven valores del módulo como 3.99999998 y queda mal visualizado, entonces se busca una forma de hacer *format* a los valores del array. Se encuentra una solución y se implementa en las mutations antes de cambiar el store.

5. Ahora bien, no sólo queremos aumentar ciertos valores en el array, queremos reemplazar el array completo (porque los datos vendrán como arrays completos), para eso usamos nuevamente `Vue.set` pero de la siguiente forma: `Vue.set(state, 'mainData', [...nuevoArray])` según las sugerencias encontradas en Stackoverflow que evita mutaciones externas del array ingresado al copiarlo en uno nuevo. Se verifica que funciona lo anterior y el front-end queda reactivo y listo para recibir arrays de datos.

6. Aparte de todo lo anterior, se decubre Vue-dev-tools Lo cual es tremendamente útil para debuggear la App. Es más se uso bastante para identificar problemas de reactividad.

7. Cambios guardados en GitHub


Figura 1: Visualización anterior de eolian Fénix

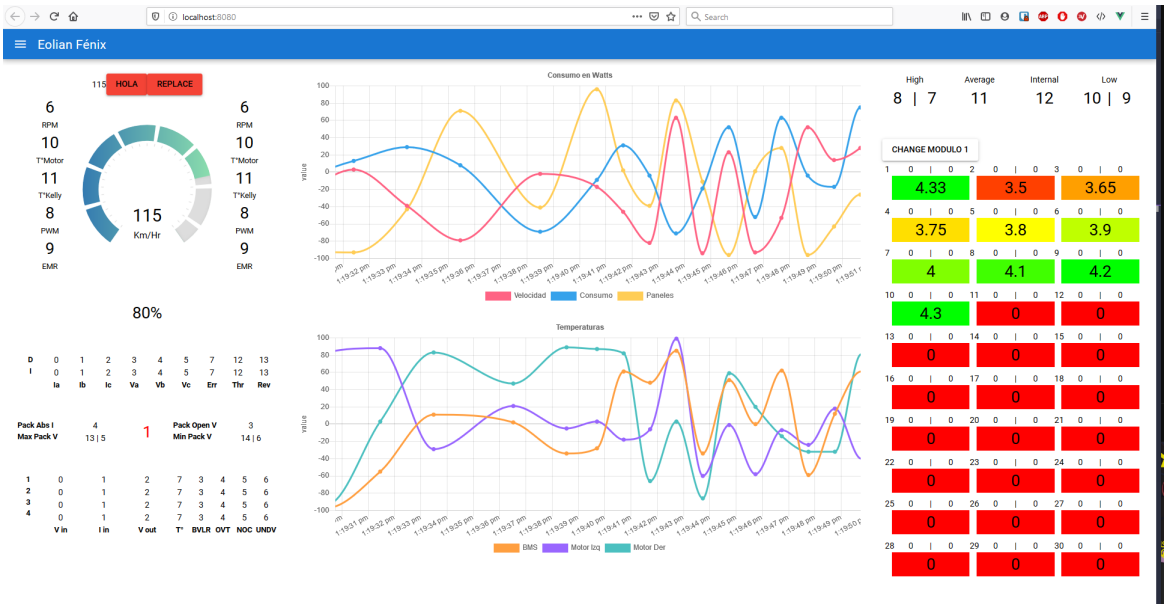


Figura 2: App en VueJS, con CSS dinámico y botones (de más) para debug