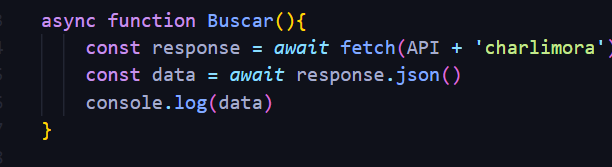
**02- Ejercicio Práctico de Introducción a Vue: Métodos y Eventos**

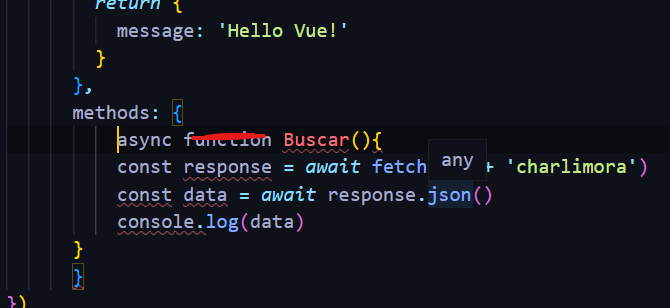
Recursos cortesía y propiedad de Escuela Vue (Juan Andrés Núñez), link GitHub <https://github.com/escuelavue/curso-vue-3-desde-cero/tree/L2_start>

1. Nos gustaría que la siguiente sección de código se activara cuando se oprime el botón de búsqueda del formulario del index:

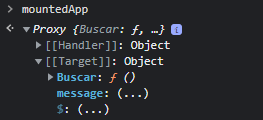


Para lograrlo es necesario añadir este método buscar a la instancia (si es que así puede llamársele):

1. Para esto creamos un nuevo objeto llamado methods y allí dentro copiamos el código cortado de la función Buscar(), pero quitamos la palabra function:

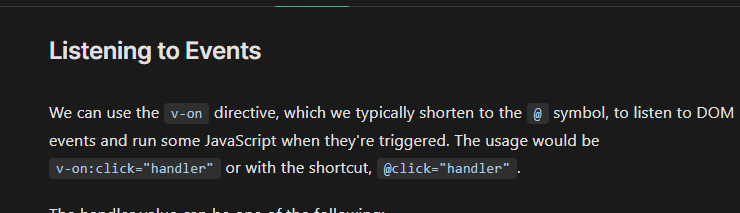


1. Podemos comprobar que ya está anclado este método a la página index, revisando en la consola con la variable mountedApp que creamos previamente (desplegar Proxy->Target:



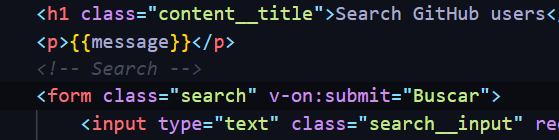
1. Esto mismo se puede implementar ya en el index, ya que ha sido anclado al div identificado con app. Pero antes vamos a revisar la información de eventos en la documentación oficial:

<https://vuejs.org/guide/essentials/event-handling.html#listening-to-events>

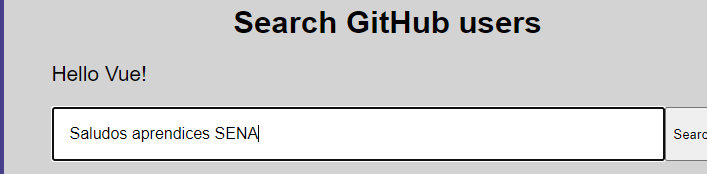
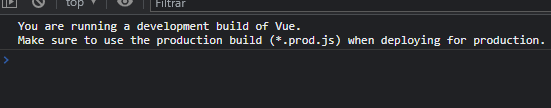


v-on le dará super poderosos a los elementos html. Con v-on se “escucharán” los eventos.

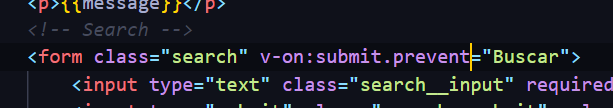
1. Vamos a indicar dentro del formulario del index que cuando se desencadene el evento submit se llame al método ya definido anteriormente: Buscar(), de la siguiente manera:



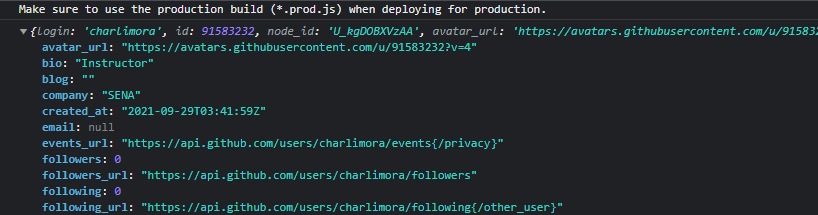
1. Si probamos cómo se comporta ahora la página en el navegador, nos damos cuenta que no alcanzamos a ver la información en consola pues la página se recarga.

1. ¿Pero cómo detenemos ese comportamiento por defecto al oprimir el botón del formulario? Usando prevent en el código así:



1. SI volvemos al formulario en el navegador, escribimos algo y oprimimos el botón, nos percatamos que funciona en consola. Se verán los datos públicos de la cuenta asociada a github:

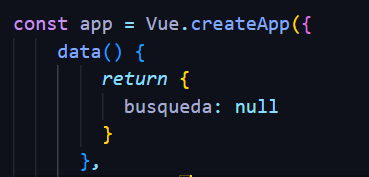


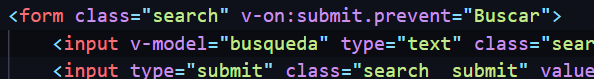
¿Qué sigue? Conectar el input que es de tipo texto, con el método Buscar, para poder buscar perfiles a partir de lo ingresado.

**Reactividad en 2 sentidos.**

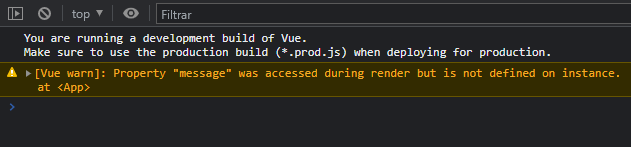
Para este caso particular usaremos v-model. Esto tiene que ver con el modelo de instancia que está dentro de data que siempre retorna un objeto con un mensaje.

1. Primero en el modelo definimos el respectivo objeto a utilizar. Reemplazamos message por busqueda (sin tilde). Lo ponemos null. Y en el index, hacemos la respectiva interpolación:



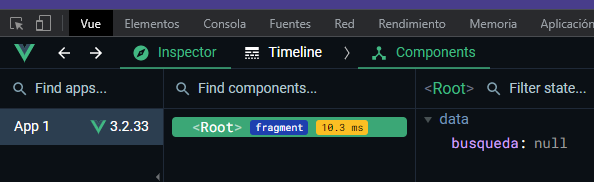


1. Nos damos cuenta que en la consola del navegador nos avisa que se está invocando la propiedad message, pero no existe



Entonces borramos dicha propiedad del html

1. Ahora inspeccionamos, usando el plugin de vue que ya se instaló en el navegador. Allí observamos el comportamiento del objeto búsqueda que ya se modeló dentro del input:



Se detecta, por ahora, un error con la inspección a través del plugin Vue Devptools, ya que no actualiza la información en tiempo real. Para esto se debe forzar un refresh en el plugin cada vez:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

1. Una vez verificado que la reactividad funciona en ambos sentidos, tanto en la vista como en modelo, podemos añadir la búsqueda a la dirección de la API de github (almacenada previamente en la constante API):

Texto

Descripción generada automáticamente

Como vemos, this permite acceder a objetos que están dentro de data.

1. Hacemos la búsqueda de un usuario (dar clic en el botón) y verificamos que efectivamente en la consola se ve la información en formato json:

Captura de pantalla de un celular

Descripción generada automáticamente