

## PRUEBA DE PROGRAMACIÓN FRONTEND

Por Canhui Huang

La aplicación es desarrollada en Vue.js, no se utilizó ninguna librería o preprocesadores de estilos. Se usó font-awesome para para los íconos. Y se utilizaron estas librerías: vue-router para el routing, axios para http requests. No se utilizó ninguna herramienta de manejo de estados.

### Instrucciones para ejecutar

Descarga y descomprime el zip o clona el repo.

Abre la consola, CMD o bash en el directorio donde está el proyecto.

(.../mobilendervuetask-master)

Para asegura que la consola esté abierta en el directorio del proyecto. Copia el directorio de tu carpeta mobilendervuetask-master y escribe esto en la consola:

```
cd <directorio>
```

Ejemplo:

```
cd C:\descargas\mobilendervuetask
```

Ejecuta los siguientes comandos.

(npm install puede tomar un buen tiempo instalando las dependencias)

```
npm install  
npm run serve
```

Después de la espera, esto aparecerá en la consola:

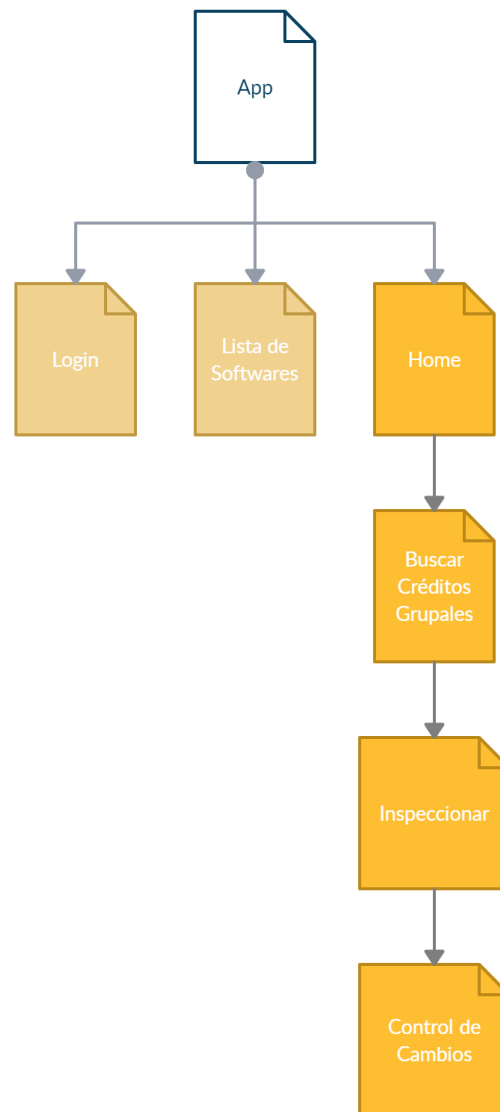
```
App running at:  
- Local:    http://localhost:8080/  
- Network:  http://192.168.1.13:8080/
```

Abre el navegador y entra a una de esas direcciones.

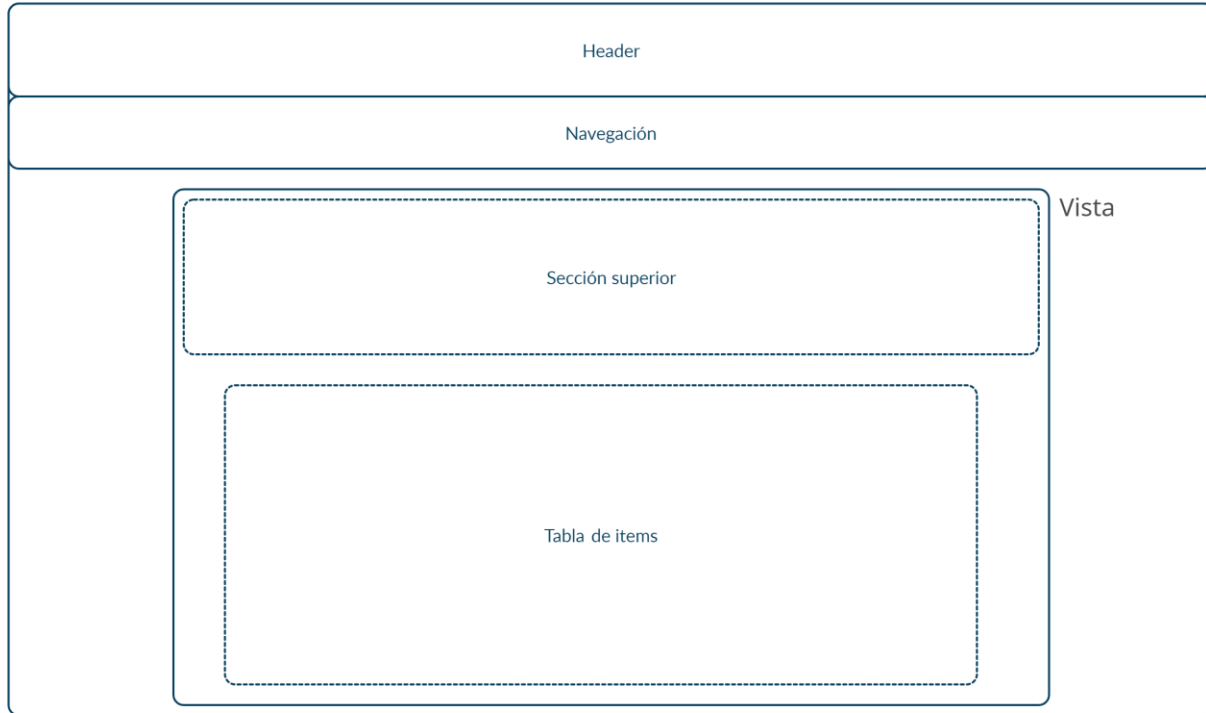
También puedes ver el demo online aquí: [frontendui.netlify.app/](https://frontendui.netlify.app/)

## Detalles

La aplicación tiene 6 vistas, organizadas así:



Este es el layout/diseño de la aplicación:



Al navegar por diferentes vistas, lo único que cambiaría en la aplicación es la sección de vista, donde todas nuestras vistas tienen un diseño similar. Todas las vistas tienen una sección superior que contiene: título y/o botones, barras de búsqueda, etc. Debajo de sección superior está la tabla de ítems, donde cada ítem es un componente reutilizable llamado **RowItem** para todas las vistas.

Por lo tanto, el componente de las vistas tienen un template más o menos así:

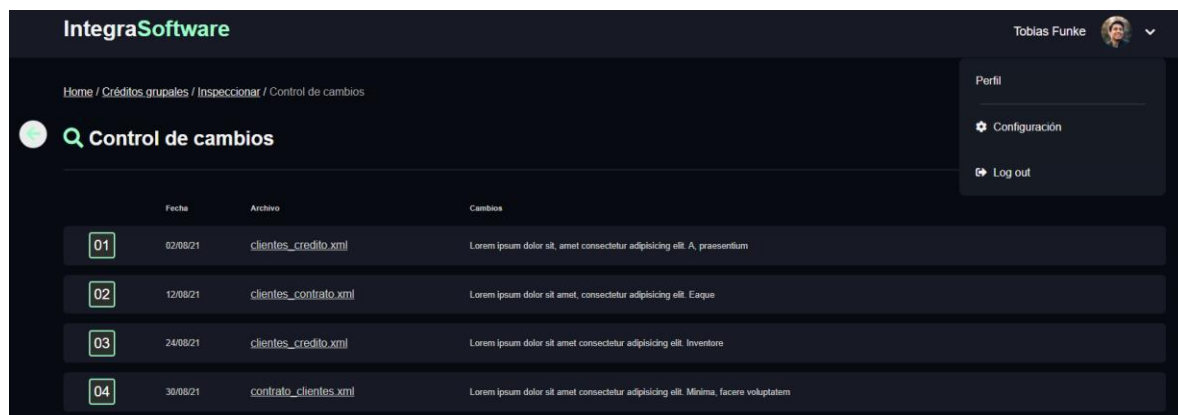
```
<div class="vista">
  <div class="topbar">
    ...
  </div>
  <table class="items">
    <RowItem
      v-for="(item) in mis_items "
      :data="item"
      :fractions="[1, 3, 14, 1]"
    />
  </table>
</div>
```

Login es la única vista que no tiene este diseño, que consiste en un simple form que toma input, y tiene la funcionalidad de loguear.

Los componentes vistas de Crédito Grupales, Control de cambios, y Listado de Softwares crean sus items con los datos consumiendo de un backend, en este caso, es un restful API falso (<https://my-json-server.typicode.com/canhuiHuang/vuetask/>) que tiene los datos mostrados en las imágenes del documento de **PRUEBA DE PROGRAMACIÓN FRONTEND** para hacer GET requests y simular POST requests. Se usó la librería axios para hacer los http calls.

Se puede simular el logueo con el correo de cualquier usuario de <https://regres.in/> con cualquier contraseña. La sesión es persistente con la información guardada en localStorage.

Al estar logueado, se podrá desloguearse con el menú desplegable del component de perfil en la parte superior derecha.



Se usó vue-utils, vue-jest y se realizó pruebas unitarias simples del UI

```
PASS tests/unit/Home.spec.js
Home Component
  ✓ Tiene rutas para creditosGrupales & listaSoftwares. (36ms)

PASS tests/unit/VistasGenerales.spec.js
Vistas Generales tienen una table de items
  ✓ CreditosGrupales_componente tiene <table> con class="items" (5ms)
  ✓ Inspeccionar_componente tiene <table> con class="items"
  ✓ ControlDeCambios_componente tiene <table> con class="items" (1ms)
  ✓ ListaSoftwares_componente tiene <table> con class="items"

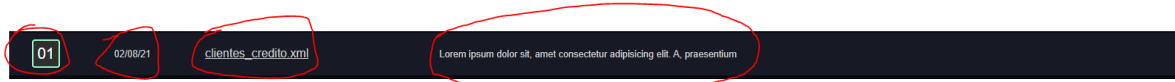
PASS tests/unit/Inspeccionar.spec.js
Inspeccionar Component
  ✓ Tiene barra de busqueda (3ms)

Test Suites: 3 passed, 3 total
Tests:       6 passed, 6 total
Snapshots:   0 total
Time:        3.217s
```

## Componente RowItem

El componente RowItem toma 2 props obligatorios: data y fraction, y 2 opciones: labelMode y link.

data es una lista de elementos td:



Y cada element td es un objeto:

```
{
  type: "text",
  content: "02/08/21",
  src: "",
  classes: "no_left_padding big_font bright_font",
  action: null,
  label: "Fecha",
}
```

RowItem tomará esta lista y creará un row relleno con esos elementos td.



### El objeto td


Key	value: args	Description
type	<code>&lt;String&gt;</code> : "text", "numberBox", "file", "imgUrl", "icon"	Tipo de contenido del td.
content	<code>&lt;String&gt;</code>	Es el contenido de <code>text</code> , el nombre del <code>file</code> , el numero de <code>numberBox</code> , y las <code>classes</code> para el icono font-awesome.
src	<code>&lt;String&gt;</code>	El link de la imagen, en caso de type: <code>"imgUrl"</code> .
classes	<code>&lt;String&gt;</code>	Personaliza posición o tamaño del contenido con args/classes como <code>no_left_padding</code> o <code>big_font</code> .
action	<code>Function</code>	Por el momento, esto no hace nada, pero está reservado para accionar funciones al momento de interactuar con este td.
label	<code>&lt;String&gt;</code>	El label de este td.

## Props de RowItem

Prop Mandatorio	value	Description
data	List<objecto td>	RowItem tomará esta lista y creará un row relleno con esos elementos td.
fraction	List<number>	Fraction es una lista de números que indica la proporción del ancho del row que cada elemento td de la lista data va a tomar. La lista fraction tiene que tener el mismo tamaño (fractionList.length == data.length) que la lista de data.

01	02/06/21	clientes_creado.sql	Lorem ipsum dolor sit, amet consectetur adipiscing elit. A, praesentium
----	----------	---------------------	---

Por ejemplo, para un crear un RowItem de proporciones similares, la lista fraction sería: [1,2,5,12] .

Prop Opcional	value	Description
labelMode	<boolean> default: false	Al pasar labelMode: true , se creara un RowItem de labels. 
link	<String> default: ""	Al pasar un link, RowItem sera clickeable, y redireccionara el browser al link indicado.