Vue.js - Componentes

...

Resumen en base a https://vuejs.org/

Vue.js The official Vue project scaffolding tool

Para crear una aplicación vue.js (hay que tener node.js instalado), se puede usar:

npm init vue@latest

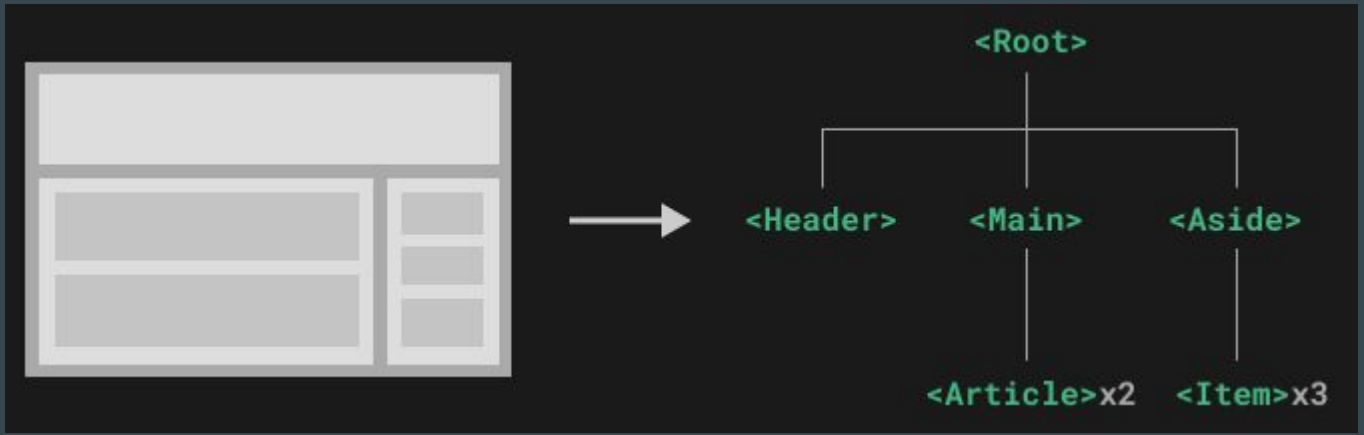
Para comenzar, con estos valores por defecto:

```
✓ Project name: … <your-project-name>
✓ Add TypeScript? … No / Yes
✓ Add JSX Support? … No / Yes
✓ Add Vue Router for Single Page Application development? … No / Yes
✓ Add Pinia for state management? … No / Yes
✓ Add Vitest for Unit testing? … No / Yes
✓ Add Cypress for both Unit and End-to-End testing? … No / Yes
✓ Add ESLint for code quality? … No / Yes
✓ Add Prettier for code formatting? … No / Yes
```

Luego, cambiar al directorio del proyecto, ejecutar npm install y npm run dev (crea un servidor de desarrollo)

Componentes

Los componentes nos permiten dividir la interfaz de usuario en piezas independientes y reutilizables, y pensar en cada pieza de forma aislada. Vue implementa su propio modelo de componentes que nos permite encapsular contenido y lógica personalizados en cada componente. Es común que una aplicación se organice en un árbol de componentes anidados:



Definición de un componente

Cuando usamos un paso de compilación, normalmente definimos cada componente de Vue en un archivo dedicado usando la .vue extensión, conocida como componente de un solo archivo (SFC, por sus siglas en inglés):

https://vuejs.org/guide/essentials/component-basics.html#defining-a-component

Definición de un componente

Por ejemplo este componente, con nombre: Sistema.vue

```
<template>
    <div>
        <h2>Sistema</h2>
    </div>
</template>
<script>
export default {
    data() {
        return {
        }
    }
}
</script>
```

Llamado a un componente

Desde un componente “padre” en este caso App.vue:

```
<script>
  import Sistema from './components/Sistema.vue'
  export default {
    components: {
      Sistema
    }
  }
</script>

<template>
  Hola mundo
  <Sistema/>
</template>
```

Paso de datos a un componente

Ejemplo de componente padre que llama con datos:

```
<script>
  import Sistema from './components/Sistema.vue'
  import Pie from './components/Pie.vue'
  export default {
    components: {
      Sistema,Pie
    }
  }
</script>
<template>
  <Sistema/>
  <Pie autor="Adrian"/>
</template>
```

Componentes que recibe datos

Ejemplo archivo llamado Pie.vue, con una props llamada autor:

```
<template>
    <div>
        <h4>Autor {{ autor }}</h4>
    </div>
</template>
<script>
export default {
    data() {
        return {
        }
    },
    props: ['autor']
}
</script>
```

Escuchar eventos de un componente hijo

Componente que tiene los datos “padre”

```
<template>
    <div>
        <h2>Sistema</h2>
        {{ lista }}
        <Detalle @agregardetalle="agregar"/>
    </div>
</template>
<script>
import Detalle from './Detalle.vue'
export default {
    components: { Detalle },
    data() {
        return { lista: [] } },
    methods: {
        agregar(elemento) { this.lista.push(elemento); }
    }
}
</script>
```

Emitir eventos del componente padre

Para emitir un evento del componente padre, desde el componente hijo, usamos “emit”

```
<template>
    <div>
        <p>Detalle</p>
        Elemento <input type="text" v-model="elemento">
        <button v-on:click="agregar">Agregar</button>
    </div>
</template>
<script>
export default {
    data() {
        return { elemento: '' }
    },
    emits: ['agregardetalle'],
    methods: {
        agregar() {
            this.$emit('agregardetalle',this.elemento);
        }
    }
}
</script>
```

Ejercicio

Realizar un sistema en vue 3, que contenga los siguientes componentes:

Head, System, Detail, Footer

Consignas:

Desde app.vue pasar el autor al componente Footer.

Desde el componente Detail, emitir un evento llamando a un método de System que recibe un parámetro con un elemento a agregar a una lista que está en el componente System.