

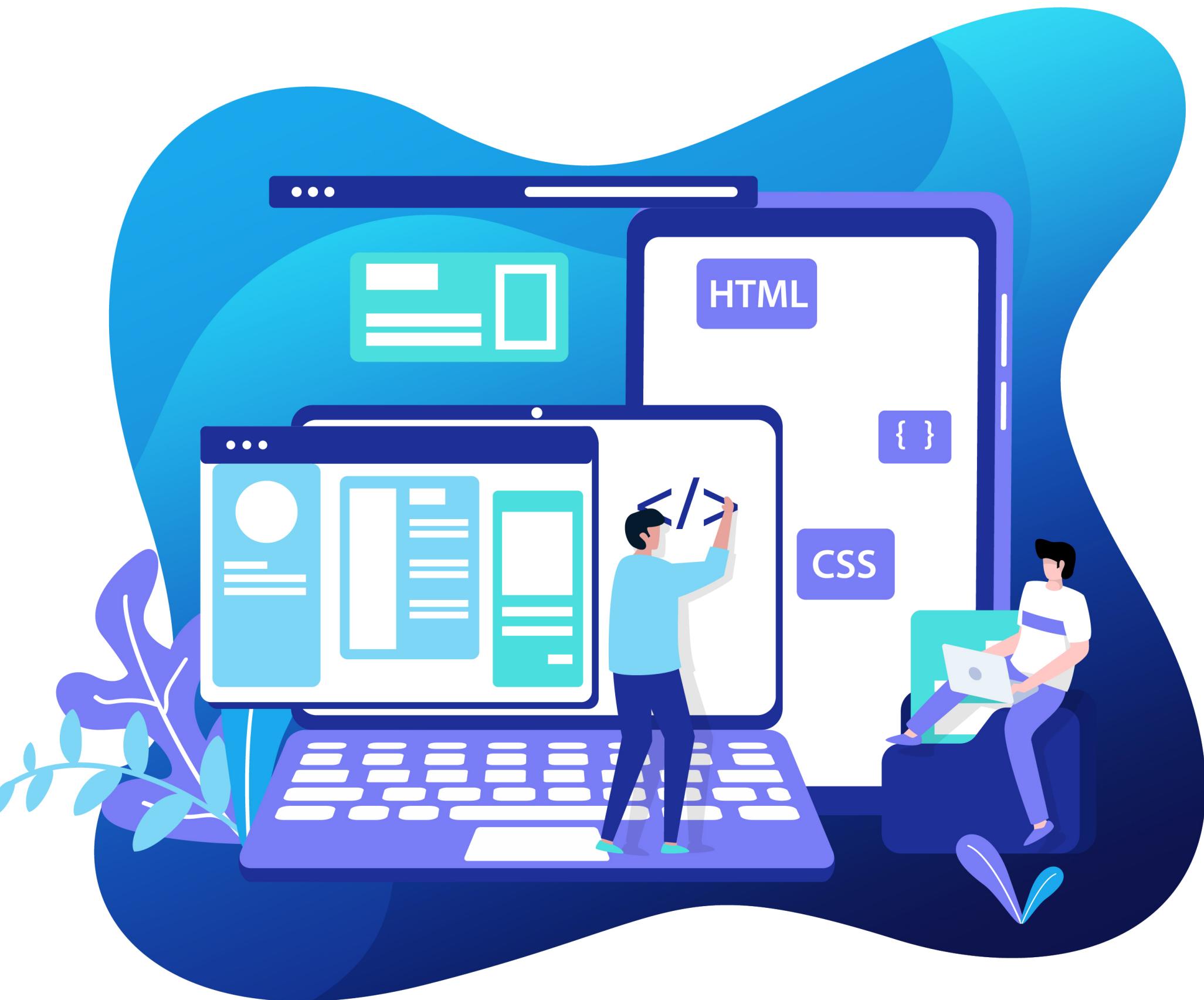


DESARROLLO WEB FRONTEND

M.I.C Normando Ali Zubia Hernández

TEMAS DE LA UNIDAD

1. Introducción
2. SSR vs CSR
3. Introducción a Vue
4. Vue CLI
5. Manejo de Rutas
6. Uso de componentes
7. Vuex
8. Websockets
9. Autenticación

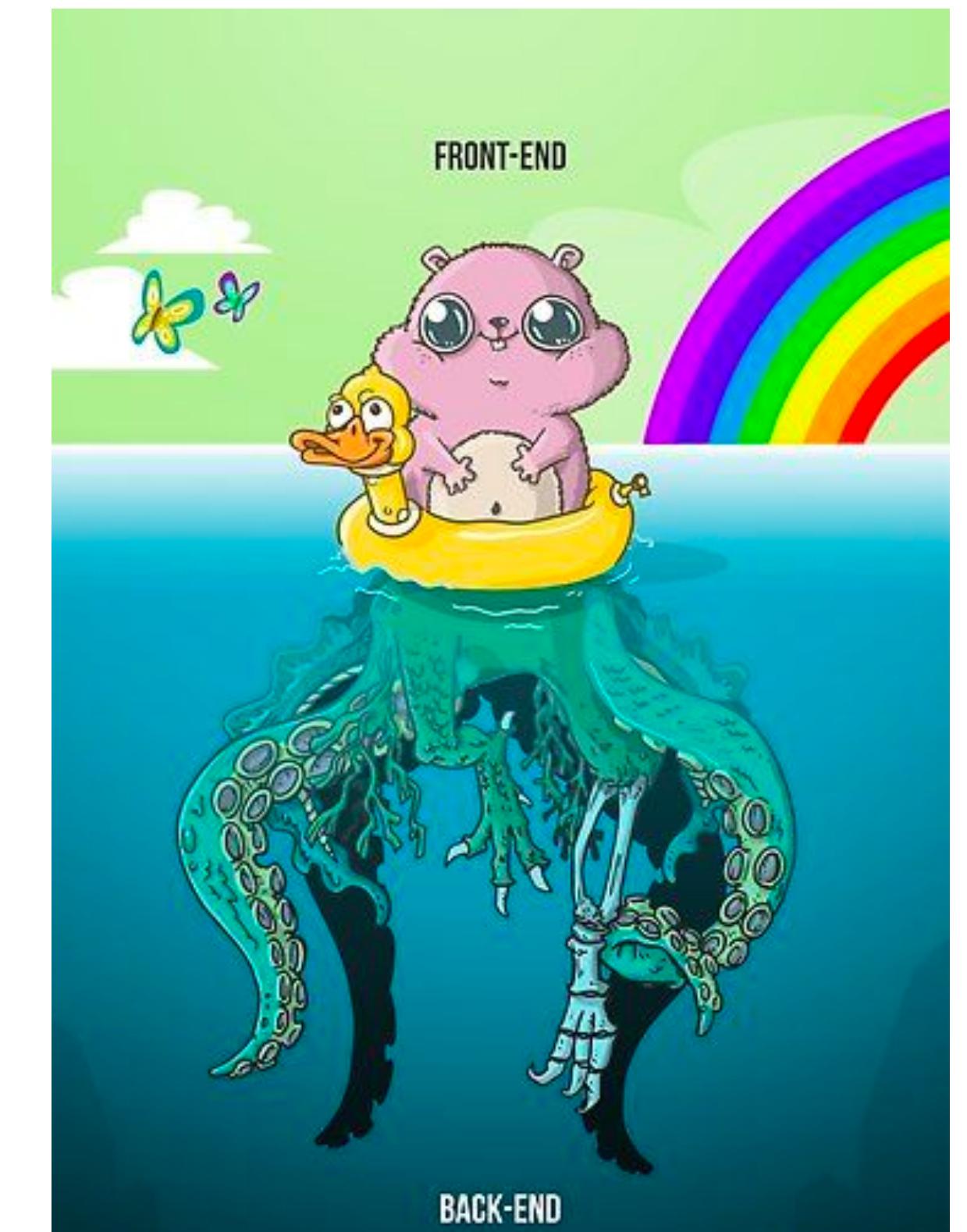


INTRODUCCIÓN



¿QUÉ ES EL DESARROLLO FRONTEND?

- Es la parte con la que interactúa el usuario final, se compone de vistas y elementos que permiten que el usuario realice acciones a través de un navegador web.
- Puede contener algo de lógica, pero regularmente es para manejar las interacciones con el usuario.
- Lenguajes utilizados:



¿QUÉ ES HTML?

- Es el lenguaje que determina/define la estructura de una página web.
- Dicho lenguaje utiliza etiquetas para definir todos los elementos que integraran a la plataforma.



¿QUÉ ES CSS?

- Es el lenguaje que permite darle estilo a los elementos/etiquetas HTML que contiene una página.
- Para poder otorgarle dicho estilo, CSS hace uso de selectores que identifican la etiqueta HTML deseada para a



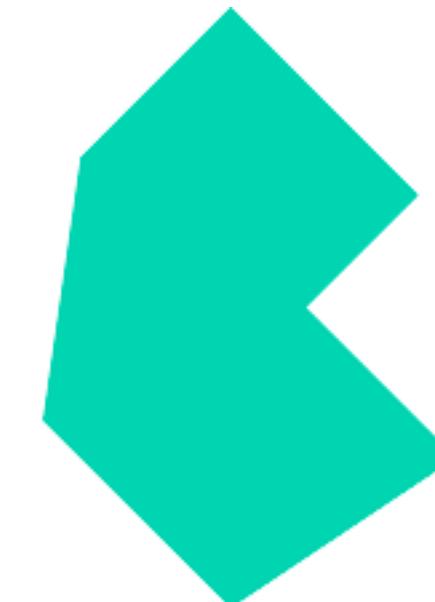
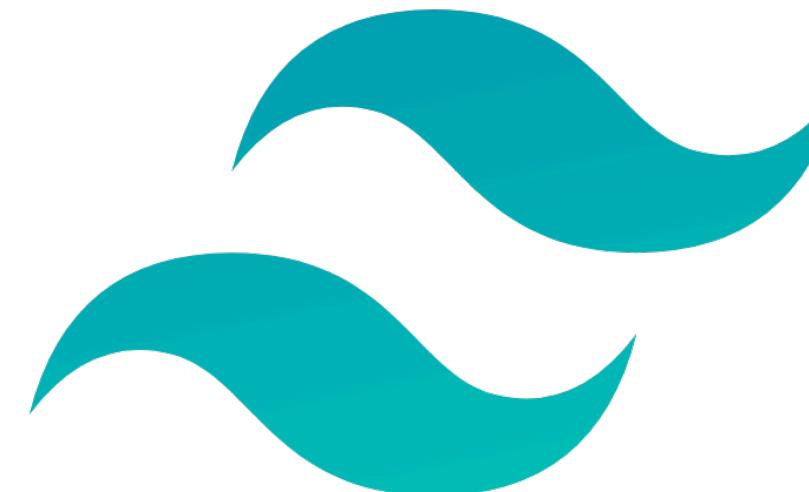
¿QUÉ ES JAVASCRIPT?

- Es el lenguaje que permite realizar interacciones complejas con el cliente a través de una página web.
- Este lenguaje permite añadir lógica a la estructura HTML, permitiendo agregar eventos, acciones e interacciones a los elementos HTML que contenga la página.



¿QUÉ ES UN FRAMEWORK CSS?

- En este caso un **Framework CSS** es aquel que nos brinda una serie de lineamientos para crear componentes que ya contienen estilo y que por defecto se verán bien.
- Regularmente contiene una serie de elementos comunes que al ser utilizados se puede fácilmente desarrollar una plataforma completa.
- Entre los más comunes encontramos los siguientes:



CV

Práctica en clase



CV

- En base a algún diseño en internet, realizar su CV para practicar el uso de frameworks CSS.

RENDERIZADO DE LADO DEL SERVIDOR VS LADO DEL CLIENTE

- Actualmente existen dos tipos renderizado por parte de los clientes, el **SSR** o **Server Side Rendering** y el **CSR** también conocido como aplicaciones **SPA** o **Single Page Applications**.
- Ambas formas son válidas y muy utilizadas hoy en día, la diferencia más grande reside en como el cliente estará relacionado/comunicado con el servidor.
- Mientras que una **SPA** es una aplicación totalmente independiente y estática, el uso de **SSR** implica que toda la lógica de mostrado de la página residirá en el servidor, es decir que aparte de las rutas de funcionalidades, se tendrán rutas de renderizado.
- Más información:
 - <https://www.freecodecamp.org/news/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d/>
 - <https://ssr.vuejs.org/#what-is-server-side-rendering-ssr>

TEMPLATE ENGINE (1/2)

- Un **template engine** es una herramienta que permite desarrollar páginas de una forma más rápida con la ventaja de poder injectar valores desde el lado del servidor, su funcionamiento consiste en transformar comandos de texto en código html por medio de la compilación de sus archivos.
- Esta herramienta es mayormente utilizada en el **SSR** debido a que existe la necesidad de mostrar la información que probablemente hemos obtenido desde la ruta de nuestro servidor, así mismo permite separar la vista de la lógica de la vista.

handlebars



<% = EJS %>

mastering context

TEMPLATE ENGINE (2/2)

- Más información:
 - <https://colorlib.com/wp/top-template-engines-for-javascript/>
 - <https://expressjs.com/en/resources/template-engines.html>
 - <https://expressjs.com/en/guide/using-template-engines.html>
 - <https://vue-loader.vuejs.org/guide/pre-processors.html#pug>
 - <https://codeburst.io/vue-components-with-pug-and-stylus-556508b74c96>

handlebars



<%= EJS %>

mastering context

CV CON SSR

Práctica en clase

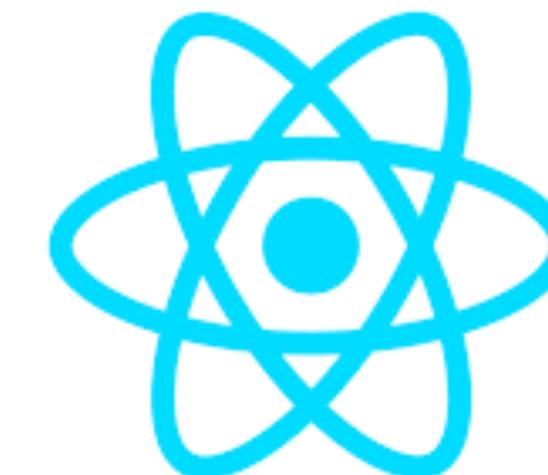


CV CON SSR

- En base a algún diseño en internet, realizar su CV para practicar el uso de frameworks CSS.
- Implementar una ruta index que permita ingresar a su cv, adicionalmente genere otras rutas, puedes ser por ejemplo:
 - Portafolio
 - Información personal
- Implementar todo lo mencionado anteriormente con renderizado de parte del servidor

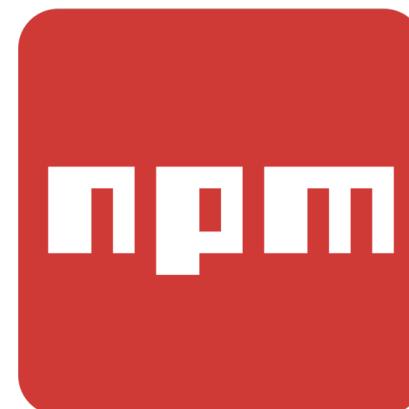
¿QUÉ ES UN FRAMEWORK?

- Un Framework es una herramienta o marco de trabajo que permite desarrollar una aplicación de manera más sencilla.
- Dicha herramienta otorga los lineamientos y la estructura para tener una aplicación mejor organizada y proporcionar un desarrollo más ágil.
- Entre los más comunes encontramos los siguientes:



¿QUÉ ES UN MANEJADOR DE DEPENDENCIAS?

- Un **Manejador de dependencias** es una manera de automatizar el proceso de instalación, actualización y configuración de herramientas/librerías externas que se están utilizando en el proyecto:
- **NPM** (Node package manager) es el gestor de dependencias para Node.js. Es un proyecto de código abierto que ayuda a los desarrolladores javascript a manejar y compartir los árboles de dependencias de sus aplicaciones. <https://www.npmjs.com/>
- **Yarn** es un administrador de paquetes que también se desempeña como administrador de proyectos. Ya sea que trabaje en proyectos pequeños o grandes, como aficionado o usuario empresarial, lo tenemos cubierto. <https://yarnpkg.com/>
- **Bower** es un administrador de dependencias html, css y javascript: <https://bower.io/>



BASE DE NUESTRO PROYECTO: VUEJS + BOOTSTRAP + NPM

- **Vue js** es un framework progresivo en Javascript para desarrollar interfaces de usuario. <https://vuejs.org/>
- **Bootstrap** es uno de los frameworks css más populares hoy en día, contiene una serie de herramientas y componentes que permiten un desarrollo rápido de interfaces. <https://getbootstrap.com/>
- **NPM** (Node package manager) es el gestor de dependencias para Node.js. Es un proyecto de código abierto que ayuda a los desarrolladores javascript a manejar y compartir los árboles de dependencias de sus aplicaciones. <https://www.npmjs.com/>



CONOCIENDO VUE



UTILIZANDO VUE EN PÁGINA ESTÁTICA



Práctica en clase

UTILIZANDO VUE EN PÁGINA ESTÁTICA (1/2)

- Crear un documento html con una estructura inicial básica

- Añadimos el archivo de modo desarrollo que nos proporciona Vue en la cabecera del documento.



A screenshot of a terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top. The terminal displays the following code:

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
5   <title>Hello Vue</title>
6 </head>
7 <body>
8
9 </body>
10 </html>
```

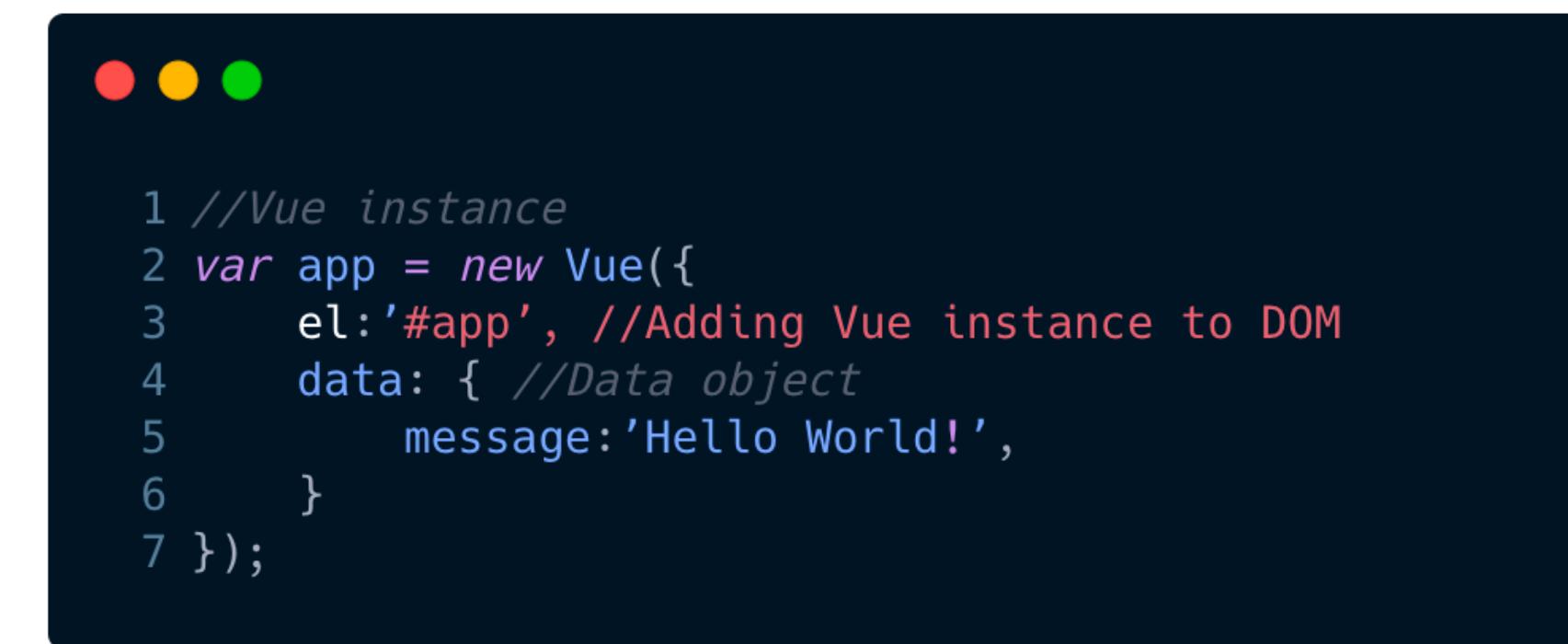
UTILIZANDO VUE EN PÁGINA ESTÁTICA (2/2)

- Implementamos Vue dentro de nuestra página por medio del siguiente código:

```
● ● ●  
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4   <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>  
5   <title>Hello Vue</title>  
6 </head>  
7 <body>  
8   <div id="app">  
9     {{ message }}  
10  </div>  
11  
12  <script type="text/javascript">  
13    var app = new Vue({  
14      el: '#app',  
15      data: {  
16        message: 'Hello Vue!'  
17      }  
18    })  
19  </script>  
20 </body>  
21 </html>
```

INSTANCIA DE VUE

- Es la raíz de nuestra aplicación. Es creada en base a opciones que son provistas a través de un objeto.
- Dentro de las opciones principales encontramos:
 - **data**: Utilizada para definición de variables a utilizar dentro de la instancia
 - **computed**: Utilizada para definición de variables auto calculadas
 - **methods**: Utilizada para definir las funciones a utilizar dentro de la instancia
- Más información: <https://vuejs.org/v2/api/#Options-Data>



```
1 //Vue instance
2 var app = new Vue({
3   el:'#app', //Adding Vue instance to DOM
4   data: { //Data object
5     message:'Hello World!',
6   }
7 });
```

RENDERIZADO DECLARATIVO

- El renderizado declarativo nos permite injectar valores al documento html de forma dinámica.
- Con el renderizado declarativo el DOM y los datos de la aplicación están relacionado y todo es reactivo!

```
● ● ●  
1 //Vue instance  
2 var app = new Vue({  
3   el:'#app', //Adding Vue instance to DOM  
4   data: { //Data object  
5     message:'Hello World!',  
6   }  
7 });
```

JAVASCRIPT

```
● ● ●  
1 <!DOCTYPE html>  
2 <html>  
3 <!-- More code -->  
4 <body>  
5   <div id="app">  
6     {{ message }}  
7   </div>  
8  
9   <!-- More code -->  
10 </body>  
11 </html>
```

HTML

ENLACE DE DATOS: V-BIND DIRECTIVE

- Al igual que el renderizado declarativo, el enlace de datos permite relacionar el valor de las variables contenidas en la propiedad **data** de la instancia de Vue con los atributos DOM.
- Cuando vemos la directiva **v-bind** sabemos que el valor usado en dicho atributo es reactivo.

```
1 //Vue instance
2 var app = new Vue({
3   el:"#app",
4   data: {
5     image:"http://img.com",
6   }
7});
```

JAVASCRIPT

```
1 <!DOCTYPE html>
2 <html>
3 <!-- More code -->
4 <body>
5   <div id="app">
6     
7     
8   </div>
9   <!-- More code -->
10 </body>
11 </html>
```

HTML

RENDERIZADO CONDICIONAL

- El renderizado condicional nos permite mostrarle al usuario diferentes tipos de contenido dependiendo de una condición dada. Dentro de las directivas encontramos:

- v-if
- v-else
- v-else-if
- v-show



```
● ● ●

1 <!DOCTYPE html>
2 <html>
3 <!-- More code -->
4 <body>
5   <div id="app">
6     <p v-if="inStock">In Stock</p>
7     <p v-else>Out of Stock</p>
8   </div>
9   <!-- More code -->
10 </body>
11 </html>
```

CICLOS

- La directiva **v-for** nos permite iterar ciertos objetos y renderizarlos en nuestro DOM.



```
1 //Vue instance
2 var app = new Vue({
3   el:"#app",
4   data: {
5     todos: [
6       { text:"TODO 1" },
7       { text:"TODO 2" },
8       { text:"TODO 3" },
9     ],
10   }
11});
```



```
1 <!DOCTYPE html>
2 <html>
3 <!-- More code -->
4 <body>
5   <div id="app">
6     <ol>
7       <li v-for="todo in todos">
8         {{ todo.text }}
9       </li>
10      </ol>
11    </div>
12    <!-- More code -->
13  </body>
14 </html>
```

JAVASCRIPT

HTML

EVENTOS

- La directiva **v-on** nos permite escuchar eventos en el DOM y ejecutar un método cuando el evento es recibido.
- <https://learnvue.co/2020/01/a-vue-event-handling-cheatsheet-the-essentials/>

```
 1 //Vue instance
 2 var app = new Vue({
 3   el:"#app",
 4   data: {
 5     numVar: 1,
 6   },
 7   methods: {
 8     addOne(){
 9       this.numVar += 1
10     }
11   }
12 });
```

JAVASCRIPT

```
 1 <!DOCTYPE html>
 2 <html>
 3 <!-- More code -->
 4 <body>
 5   <div id="app">
 6     <p>{{ numVar }}</p>
 7     <button v-on:click="addOne"> Add One </button>
 8   </div>
 9   <!-- More code -->
10 </body>
11 </html>
```

HTML

COMPUTED PROPERTY

- En Vue una **computed property** calcula un valor en lugar de guardarlo.

```
1 //Vue instance
2 var app = new Vue({
3   el:"#app",
4   data: {
5     firstName:"text",
6     lastName:"text",
7   },
8   computed: {
9     completeName(){
10       return this.firstName + this.lastName
11     }
12   }
13});
```

JAVASCRIPT

```
1 <!DOCTYPE html>
2 <html>
3 <!-- More code -->
4 <body>
5   <div id="app">
6     {{ completeName }}
7   </div>
8   <!-- More code -->
9 </body>
10 </html>
```

HTML

COMPONENTES

- Los componentes son bloques de código reusables que pueden tener tanto estructura html como funcionalidad.

```
● ● ●  
1 //Vue instance  
2 Vue.component("component1", {  
3   el:"#app",  
4   data(){  
5     return {  
6       //data  
7     }  
8   },  
9   methods:{  
10    //methods  
11  },  
12   template:{  
13    //html template  
14  }  
15});
```

JAVASCRIPT

```
● ● ●  
1 <!DOCTYPE html>  
2 <html>  
3 <!-- More code -->  
4 <body>  
5   <div id="app">  
6     <component1></component1>  
7   </div>  
8   <!-- More code -->  
9 </body>  
10 </html>
```

HTML

MATERIAL COMPLEMENTARIO

- Documentación de Vuejs
 - <https://vuejs.org/v2/guide/>
- Documentación de API
 - <https://vuejs.org/v2/api/>
- Cursos de Vuejs
 - <https://www.vuemastery.com/courses/>
 - <https://www.vuemastery.com/courses/intro-to-vue-js/vue-instance/>

BASE CLIENT



VUE CLI

- Es una herramienta que nos permite realizar la inicialización de un proyecto de Vue
- Dicha herramienta provee una estructura sólida para un cliente web
- Más información:
 - <https://cli.vuejs.org/>
 - <https://cli.vuejs.org/guide/>



Vue CLI

CREANDO LA BASE DE UN CLIENTE WEB



Práctica en clase

USANDO VUE CLI

- Instalamos Vue CLI por medio de la terminal



```
npm install -g @vue/cli
```

- Crear estructura base de proyecto



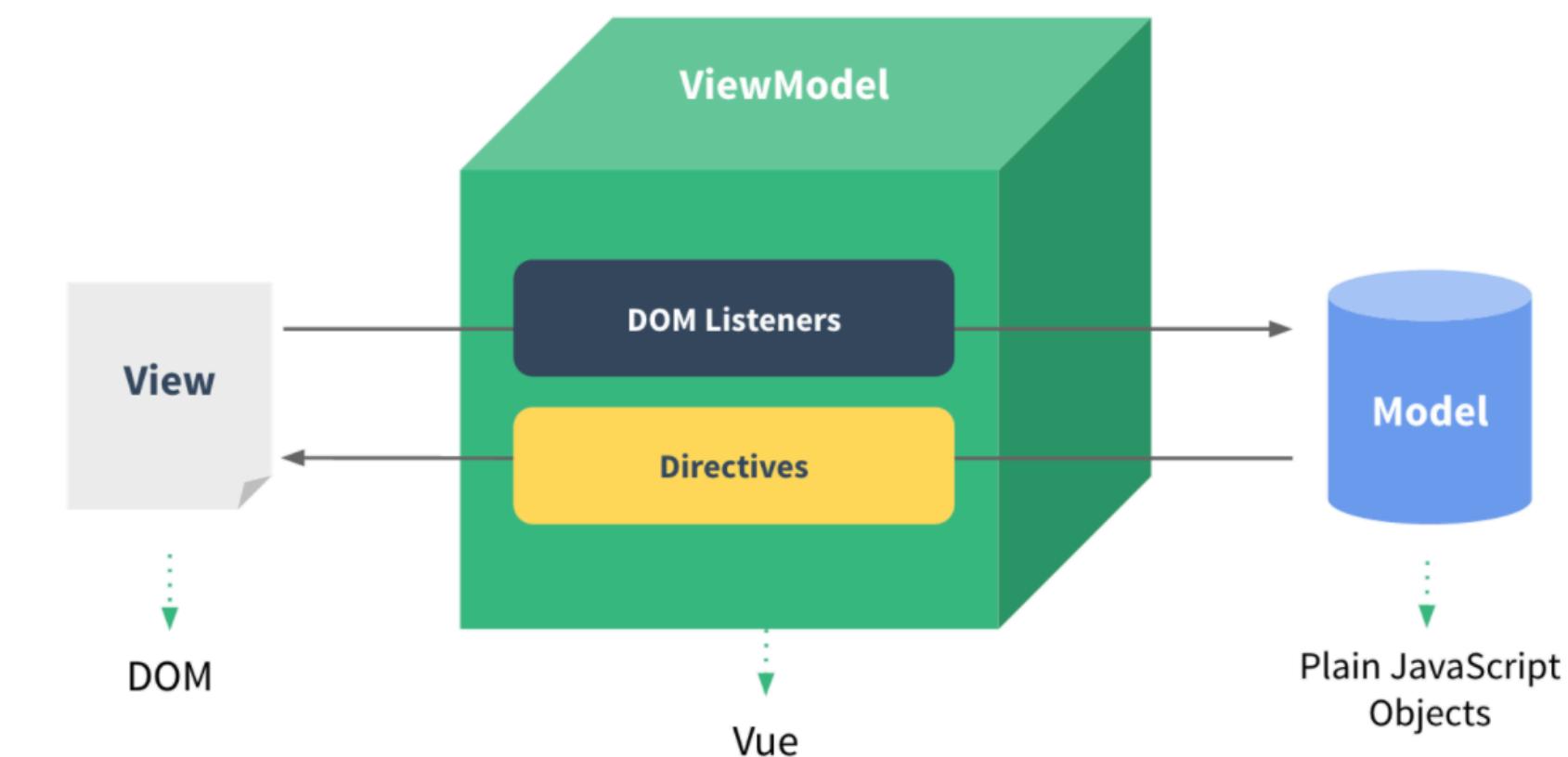
```
vue create my-first-client
```

ESTRUCTURA CLIENTE

- Al usar **Vue CLI** y elegir las configuraciones pertinentes a **Router** y **Vuex** podrá observar la siguiente estructura:
 - **assets** -> Este directorio guardará todos los recursos públicos de nuestro cliente
 - **router**-> Este directorio organizará todas las rutas en nuestro cliente
 - **views**-> Aquí residirán todas las vistas de nuestro cliente
 - **main.js**->Archivo donde implementaremos la lógica de inicio de nuestro cliente
 - **App.vue**->Junto con el archivo **index.html** conforman la vista principal de nuestro cliente
 - **package.json**->Archivo que administra el funcionamiento del cliente
 - **components**->Este directorio contendrá todos los **componentes atómicos** de nuestro cliente
 - **store**-> Este directorio contendrá todos los **store** de nuestro cliente

MVVM

- Vue está basado en el patrón **Model-View-ViewModel**, el cual separa los objetos en tres grupos diferentes:
 - **Modelo.**- El cual mantiene la información del cliente
 - **Vista.**- La cual muestra todos los elementos visuales y controla las interacciones en la pantalla
 - **VistaModelo.**- Es un puente que permite mostrar la información que tiene el modelo en la vista.



CICLO DE VIDA DE UNA INSTANCIA DE VUE

- Cada instancia de Vue contiene diferentes etapas en su existencia:
 - **beforeCreate**: En esta etapa se definen todas las operaciones que son necesarias antes de que inicie la instancia de Vue
 - **created**: En esta etapa la instancia de Vue ya ha sido creada.
 - **beforeMount**: Etapa previa a crear toda la estructura dentro de la instancia.
 - **mounted**: La estructura del componente ha sido creada.
 - **beforeDestroy**: Operaciones que son necesarias antes de destruir la instancia.
 - **destroyed**: Operaciones ejecutadas después de destruir la instancia.
- <https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram>

SINGLE FILE COMPONENTS

- Los componentes en vue son llamados **single file** debido a que la lógica y la estructura del mismo se manejan en un mismo archivo.
- Partes de un **SFC**:
 - Template.- Cuerpo HTML
 - Script.- Funcionalidad
 - Style.- Estilos



```
1 <template>
2   <p>{{ greeting }} World!</p>
3 </template>
4
5 <script>
6 module.exports = {
7   data: function() {
8     return {
9       greeting: "Hello"
10    };
11  }
12};
13 </script>
14
15 <style scoped>
16 p {
17   font-size: 2em;
18   text-align: center;
19 }
20 </style>
```

LANDING PAGE

Práctica en clase

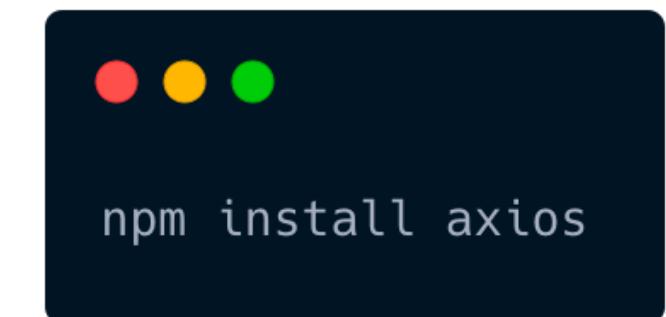


LANDING PAGE

- En base a lo aprendido desarrollar una landing page que vaya conforme a las definiciones que se han realizado en clase
- Usar Bootstrap para generar los estilos de la plataforma
- Contenido de la landing page:
 - Listado de servicios
 - Listado de productos
 - Información importante de la empresa

CONSUMO DE SERVICIOS API CON AXIOS

- Axios es una librería que nos permite realizar peticiones a APIs, dicha librería funciona perfectamente tanto para clientes como para servidores.
- Instalación de Axios: <https://www.npmjs.com/package/axios>
- Uso de axios



```
// GET petition
async function getUser() {
  try {
    const response = await axios.get('/user', {
      params: {
        ID: 12345
      }
    });

    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

```
// POST petition
async function addUser() {
  try {
    const response = await axios.post('/user', {
      firstName: 'Fred',
      lastName: 'Flintstone'
    });

    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

CREACIÓN DE SERVICIOS DE PRUEBA CON JSON SERVER

- Json Server es una librería que nos permite crear servicios REST de prueba para simular la conexión con un servidor por medio del cliente.
- Instalación Json Server: <https://github.com/typicode/json-server>
- Uso de Json Server



```
● ● ●
npm install -g json-server
```



```
{  
  "posts": [  
    { "id": 1, "title": "json-server", "author": "typicode" }  
  ],  
  "comments": [  
    { "id": 1, "body": "some comment", "postId": 1 }  
  ],  
  "profile": { "name": "typicode" }  
}
```

DB.JSON



```
● ● ●
json-server --watch db.json
```

CREACIÓN DE SERVIDOR DE PRUEBA

Práctica



CONSUMO DE API

- En base a la landing page vista anteriormente realizar lo siguiente:
 - Crear las funciones que obtengan la lista de servicios y productos de nuestra API por medio de Axios
 - Ejecutar las funciones para obtener la información antes de la etapa **mounted**
 - Renderizar la lista de servicios y productos por medio de un ciclo

CONSUMO DE API

Práctica



CONSUMO DE API

- En base a la landing page vista anteriormente realizar lo siguiente:
 - Crear las funciones que obtengan la lista de servicios y productos de nuestra API por medio de Axios
 - Ejecutar las funciones para obtener la información antes de la etapa **mounted**
 - Renderizar la lista de servicios y productos por medio de un ciclo

VUE ROUTER



VUE ROUTER

- Vue Router es el enrutador oficial para Vuejs, el cual permite la navegación de los usuarios a través de nuestro cliente. <https://router.vuejs.org/>
- Sus funciones permiten afectar la navegación mediante el template de la página como por medio de la lógica del componente.
- Al utilizarlo toda la lógica estará concentrada en el archivo **router.js**, en el cual se definirá las rutas de nuestro cliente y el componente que le corresponde a cada una de ellas.
- Para renderizar las vistas se utilizará la etiqueta **<router-view>**, donde se alojarán todas las vistas que usemos.

DEFINICIÓN DE RUTAS E HIJOS

- El archivo **router.js** es donde definiremos todas las rutas de nuestro cliente.
- Para definir una ruta anidada basta con usar la propiedad **children** dentro de la definición de nuestra ruta:



```
1 const router = new VueRouter({
2   routes: [
3     { path: '/user/:id', component: User,
4       children: [
5         {
6           // UserProfile will be rendered inside User's <router-view>
7           // when /user/:id/profile is matched
8           path: 'profile',
9           component: UserProfile
10        },
11        {
12          // UserPosts will be rendered inside User's <router-view>
13          // when /user/:id/posts is matched
14          path: 'posts',
15          component: UserPosts
16        }
17      ]
18    }
19  ]
20 })
```

NAVEGACIÓN POR MEDIO DEL TEMPLATE

- Para navegar por medio del template se puede utilizar la etiqueta `<router-link>`, el cual será convertido en un link interno de navegación:



```
1 <script src="https://unpkg.com/vue/dist/vue.js"></script>
2 <script src="https://unpkg.com/vue-router/dist/vue-
  router.js"></script>
3
4 <div id="app">
5   <h1>Hello App!</h1>
6   <p>
7     <!-- use router-link component for navigation. -->
8     <!-- specify the link by passing the `to` prop. -->
9     <!-- `<router-link>` will be rendered as an `<a>` tag
  by default -->
10    <router-link to="/foo">Go to Foo</router-link>
11    <router-link to="/bar">Go to Bar</router-link>
12  </p>
13  <!-- route outlet -->
14  <!-- component matched by the route will render here -->
15  <router-view></router-view>
16 </div>
```



```
1 // 1. Define route components.
2 // These can be imported from other files
3 const Foo = { template: '<div>foo</div>' }
4 const Bar = { template: '<div>bar</div>' }
5
6 // 2. Define some routes
7 // Each route should map to a component. The "component" can
8 // either be an actual component constructor created via
9 // `Vue.extend()`, or just a component options object.
10 // We'll talk about nested routes later.
11 const routes = [
12   { path: '/foo', component: Foo },
13   { path: '/bar', component: Bar }
14 ]
```

HTML VIEW

ROUTER.JS

NAVEGACIÓN PROGRAMÁTICA

- Además de la etiqueta `<router-link>` es posible hacer que el usuario navegue dentro de nuestra plataforma por medio de javascript por medio del método `router.push()`
- <https://router.vuejs.org/guide/essentials/navigation.html>

A screenshot of a terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top. The terminal displays several lines of code demonstrating different ways to use the `router.push()` method:

```
1 // literal string path
2 router.push('home')
3
4 // object
5 router.push({ path: 'home' })
6
7 // named route
8 router.push({ name: 'user', params: { userId: '123' } })
9
10 // with query, resulting in /register?plan=private
11 router.push({ path: 'register', query: { plan: 'private' } })
```

DEFINICIÓN DE RUTAS

Práctica en clase



DEFINICIÓN DE RUTAS

- En base a la landing page construida anteriormente, diseñar las siguientes vistas:
 - Ver todos los productos
 - Ver todos los servicios
 - About us
 - Contacto
- Diseñar la navegación por medio de Vue router

PARA INVESTIGAR...

- Oficial documentation: <https://router.vuejs.org/>
- Navigation Guards: <https://router.vuejs.org/guide/advanced/navigation-guards.html>
- Lazy loading routes: <https://router.vuejs.org/guide/advanced/lazy-loading.html>



PRÁCTICA DE REFORZAMIENTO

Práctica en clase



TODO LIST

- En base a los conocimientos adquiridos a lo largo del semestre, realizar lo siguiente:
 - Crear una Servidor web que permita realizar un CRUD de tareas con el stack Nodejs + Express
 - Crear un cliente web que tenga una vista donde se puedan realizar las siguientes tareas:
 - Dar de alta una tarea
 - Observar una lista de tareas por completar
 - Poder marcar una tarea como completada y realizar una acción en base a dicha acción
 - Poder eliminar o modificar tareas

USO DE COMPONENTES



CREACIÓN DE COMPONENTES

Práctica en clase



CREACIÓN DE COMPONENTES (1/2)

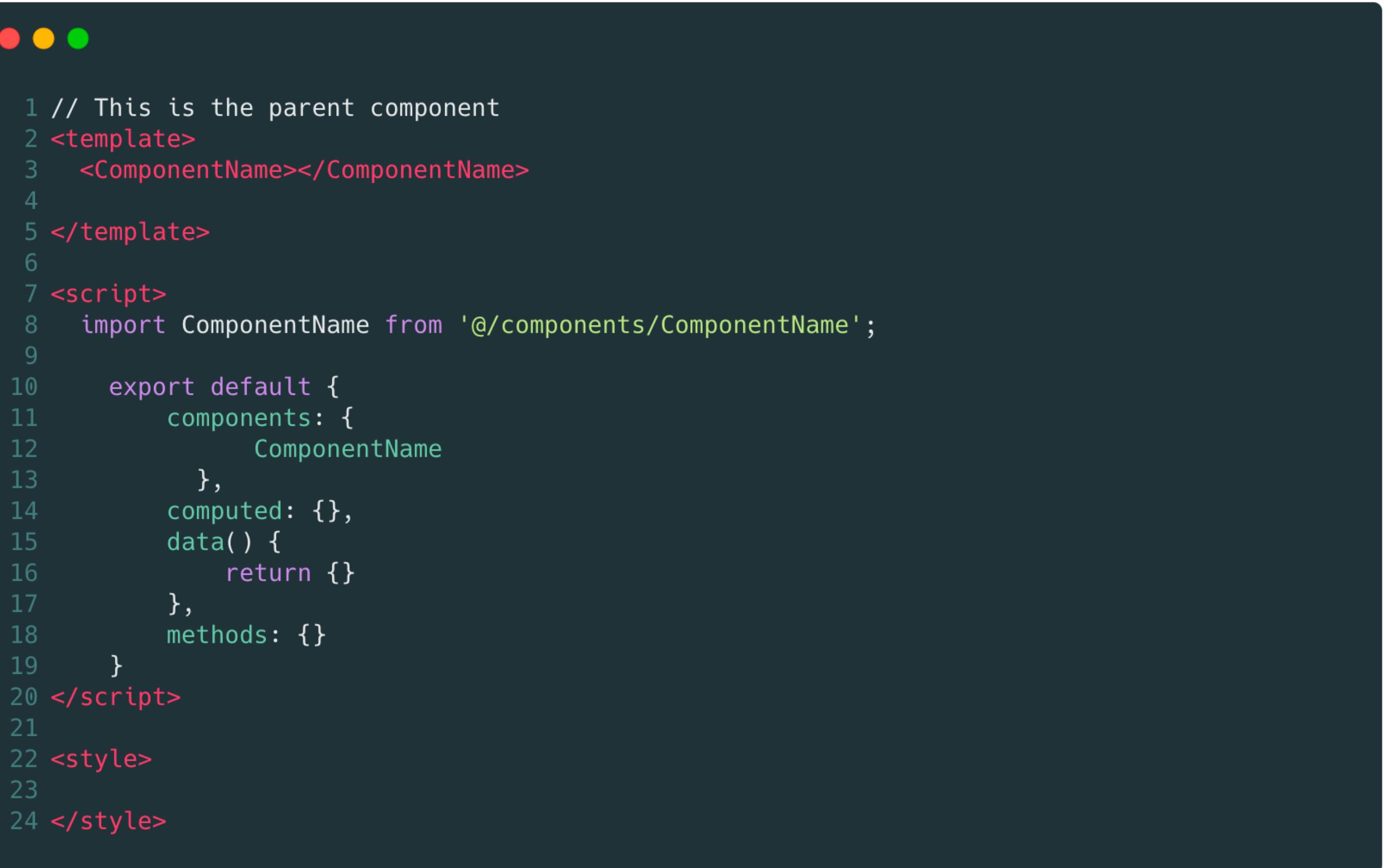
- Creación de estructura base para Single File Component, utilizando la propiedad **name** para la identificación del componente. Es recomendable ubicar dicho archivo en la carpeta de componentes de nuestro proyecto.



```
1 <template>
2
3 </template>
4
5 <script>
6   export default {
7     name: "ComponentName",
8     computed: {},
9     data() {
10       return {}
11     },
12     methods: {}
13   }
14 </script>
15
16 <style>
17
18 </style>
```

CREACIÓN DE COMPONENTES (2/2)

- Una vez realizado esto, podremos usarlo importandolo por medio del nombre de su archivo en otro componente. Esto debe ser realizado dentro de la etiqueta <script> del componente padre.



A screenshot of a code editor window showing a component definition. The code is written in a syntax similar to Vue.js. It includes a template section with a placeholder for another component, a script section where the component is imported, and a style section. The code is numbered from 1 to 24.

```
1 // This is the parent component
2 <template>
3   <ComponentName></ComponentName>
4
5 </template>
6
7 <script>
8   import ComponentName from '@/components/ComponentName';
9
10  export default {
11    components: {
12      ComponentName
13    },
14    computed: {},
15    data() {
16      return {}
17    },
18    methods: {}
19  }
20 </script>
21
22 <style>
23
24 </style>
```

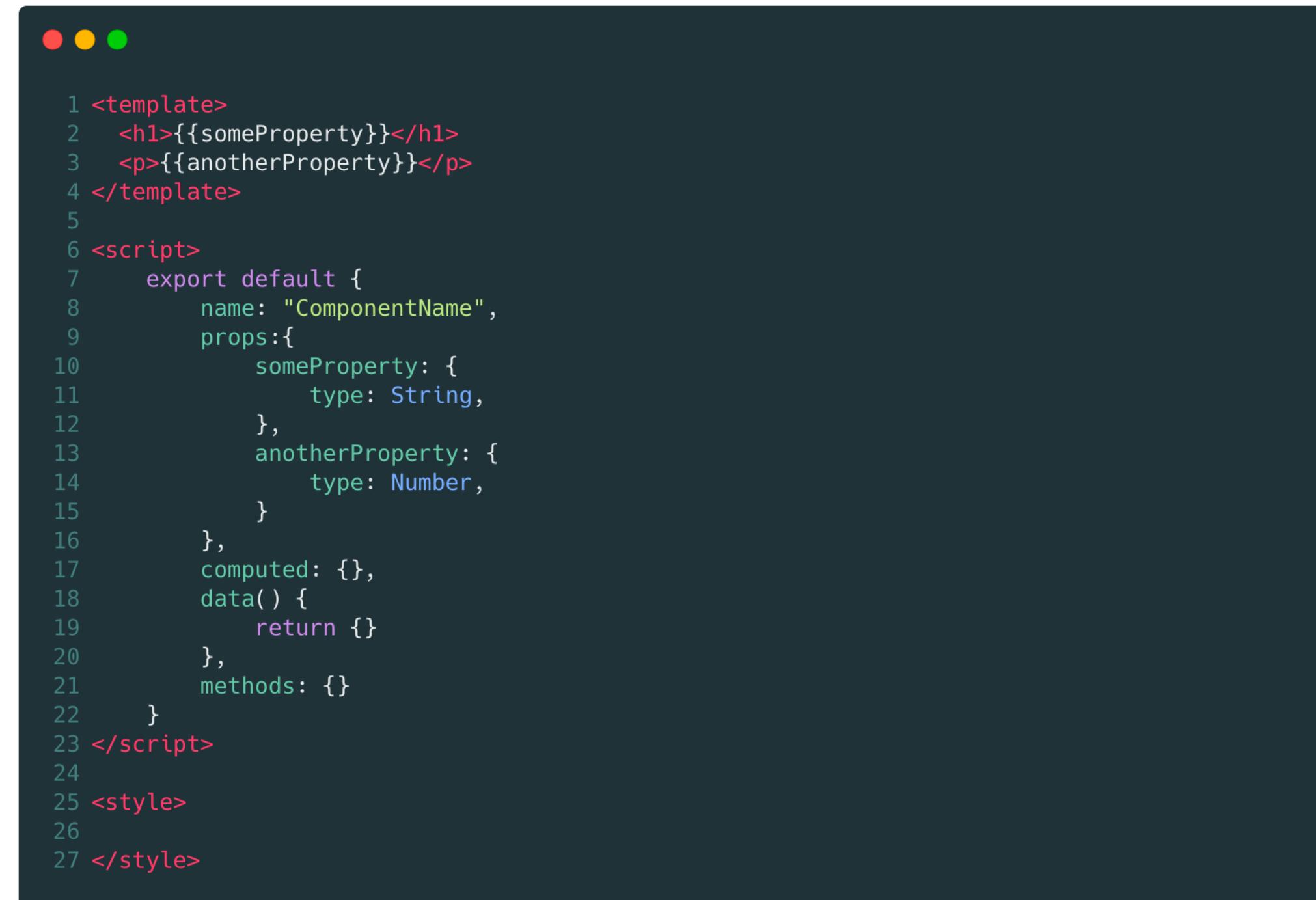
PROPIEDADES

- En ciertas ocasiones en los cuales generamos componentes, es necesario proveer a los componentes hijos con valores que se encuentran a el padre, para esto Vue tiene las llamadas **propiedades**.
- <https://vuejs.org/v2/guide/components-props.html>



```
1 <template>
2 <ComponentName
3   someProperty="SomeText"
4   anotherProperty="100"
5 ></ComponentName>
6
7
8 </template>
```

USO



```
1 <template>
2   <h1>{{someProperty}}</h1>
3   <p>{{anotherProperty}}</p>
4 </template>
5
6 <script>
7   export default {
8     name: "ComponentName",
9     props: {
10       someProperty: {
11         type: String,
12       },
13       anotherProperty: {
14         type: Number,
15       }
16     },
17     computed: {},
18     data() {
19       return {}
20     },
21     methods: {}
22   }
23 </script>
24
25 <style>
26
27 </style>
```

USO DE EVENTOS

- En ciertas ocasiones en los cuales generamos componentes, es necesario indicar al padre que ciertos sucesos han pasado en el hijo, para esto Vue hace uso de **eventos**.
- <https://vuejs.org/v2/guide/components-custom-events.html>

```
 1 <template>
 2   <ComponentName
 3     v-on:someEvent="someAction"
 4   >
 5   </ComponentName>
 6 </template>
 7
 8 <script>
 9   export default {
10     computed: {},
11     data() {
12       return {}
13     },
14     methods: {
15       someAction(){
16         // Some code
17       }
18     }
19   }
20 </script>
21
22 <style>
23
24 </style>
```

```
 1 <template>
 2   <button
 3     v-on:click="emitEvent"
 4   >Save</button>
 5
 6 </template>
 7
 8 <script>
 9   export default {
10     name: "ComponentName"
11     computed: {},
12     data() {
13       return {}
14     },
15     methods: {
16       emitEvent(){
17         this.$emit("someEvent");
18       }
19     }
20   }
21 </script>
22
23 <style>
24
25 </style>
```

SLOTS

- La etiqueta `<slot>` dentro de los componentes nos permiten tener componentes más dinámicos, los cuales pueden requerir del cambio de su contenido dependiendo del caso de uso.
- La etiqueta `slot` nos permite injectar contenido html/web desde el padre hacia el hijo.
- Así mismo es posible nombrar los slots para asignar la inyección a áreas específicas de nuestros componente.
- <https://vuejs.org/v2/guide/components-slots.html>

EVENT BUS

- Existen ocasiones en las cuales la emisión de eventos de un hijo a un padre no es suficiente y requerimos una emisión más extensa.
- Para ello existe el **EventBus** con el cual podremos mandar un evento que trascenderá hacia los padres hasta que uno de ellos procese el evento.
- En el caso de los hijos se tendrá un **EventBus.\$emit** que mandará el evento y en el padre tendremos un **EventBus.\$on** que lo procesará
- Es importante que si se hace el uso de este tipo de eventos se debe de definir un **EventBus.\$off** en el componente que definimos el listener, ya que es posible que entremos en una problemática de repetición de listeners.
- <https://medium.com/@andrejsabrickis/https-medium-com-andrejsabrickis-create-simple-eventbus-to-communicate-between-vue-js-components-cdc11cd59860>

VUEX



VUEX

- Sirve como un archivo centralizado para todos los componentes en una aplicación
- Permite compartir información entre componentes sin la necesidad de llamar al servidor en cada uno de ellos.
- Se pueden tener varios archivos de información dependiendo de la lógica de la aplicación.



CONCEPTOS CLAVE

- Un store se compone de 4 elementos principales:
 - **State:** Es el espacio donde tenemos alojada nuestra información, esta variable está en formato json
 - **Getters:** Es la forma de acceder desde los componentes a la información que se encuentra dentro de la variable state
 - **Actions:** Son operaciones que pueden ser realizadas desde los componentes para interactuar con el estado, aquí también se pueden hacer peticiones al servidor.
 - **Mutations:** Son las operaciones que nos permiten modificar el valor de una variable dentro del estado.
- **Módulos:** Es el acomodo que nos permite realizar un store para delimitar el alcance de un componente

CREANDO UN STORE

Práctica en clase



CREACIÓN DE STORE (1/3)

- Si hemos instalado Vuex desde Vue Cli deberemos tener una carpeta store con un archivo index.js dentro de ella. En caso contrario es necesario instalar la librería:
<https://vuex.vuejs.org/installation.html>
- Dentro de la carpeta store crearemos una carpeta llamada **modules**, la cual nos permitirá manejar los diferentes store que tengamos en nuestra aplicación.
- Crear un archivo llamado **tasks.store.js**

CREACIÓN DE STORE (2/3)

- Asignar la siguiente estructura al archivo `tasks.store.js`

```
1 const state = {};
2
3 const getters = {};
4
5 const actions = {};
6
7 const mutations = {};
8
9
10
11 export default {
12     namespaced: true,
13     state: {
14         ...state
15     },
16     getters: {
17         ...getters
18     },
19     actions: {
20         ...actions
21     },
22     mutations: {
23         ...mutations
24     }
25 };
```

CREACIÓN DE STORE (3/3)

- Dentro del archivo **index.js** importar el archivo **task.store.js** y asignarlo a la variable **modules**



```
1 import Vue from 'vue'  
2 import Vuex from 'vuex'  
3  
4 import tasks from '@/store/modules/tasks.store';  
5  
6 Vue.use(Vuex)  
7  
8 export default new Vuex.Store({  
9   state: {  
10     },  
11   mutations: {  
12     tasks  
13   },  
14   actions: {  
15   },  
16   modules: {  
17   }  
18 })  
19
```

MAPGETTERS

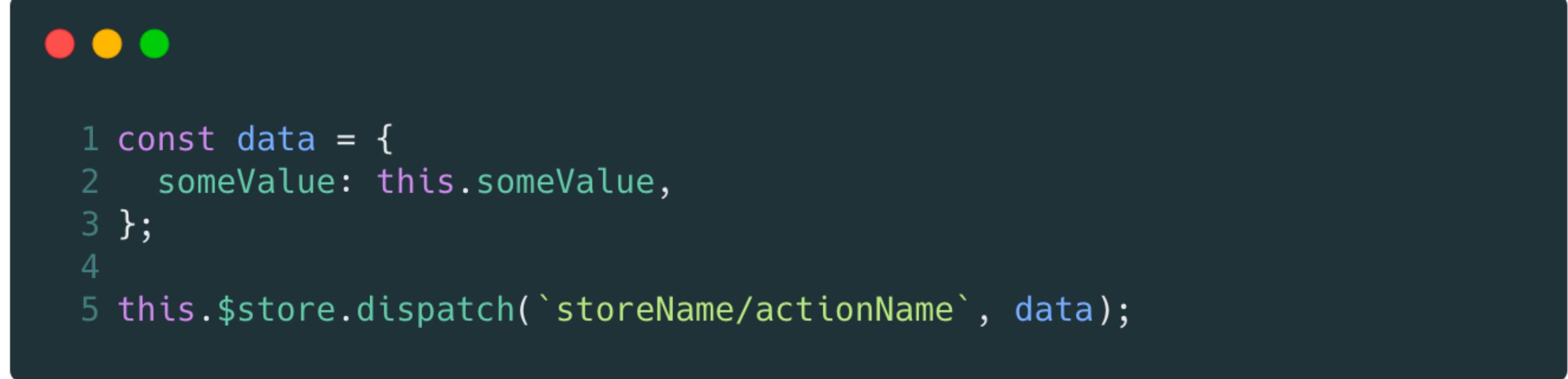
- Para interactuar con las variables dentro de nuestro store podremos hacer uso de los mapGetters, para ello solo necesitaremos indicar el store que estaremos utilizando y los métodos que deseamos utilizar.
- En la vista sólo es necesario el nombre del método para invocarlo
- Dentro de los métodos de nuestros componente será necesario accederlo por medio de la siguiente dirección: `this.$store.getters['storeName/getterName'];`



```
1 computed: {
2     ...mapGetters('storeName', [
3         'getterName',
4     ]),
5 },
```

ACTIONS

- Para poder ejecutar actions desde nuestro componentes podemos hacer uso del método dispatch, el cual nos permitirá llamar a una **action** dentro de nuestro store.



The screenshot shows a dark-themed code editor window with three circular status indicators at the top: red, yellow, and green. The code itself is written in a light-colored font and consists of five numbered lines:

```
1 const data = {  
2   someValue: this.someValue,  
3 };  
4  
5 this.$store.dispatch(`storeName/actionName`, data);
```

PRÁCTICA DE REFORZAMIENTO

Práctica en clase



CALCULADORA

- En base a los conocimientos adquiridos a lo largo del semestre, realizar lo siguiente:
 - Creación de una calculadora basándonos en el siguiente ejemplo: <https://codepen.io/kalpeshsingh/pen/wMNpLp>
 - Actividades:
 - Creación de proyecto por medio de CLI
 - Asignación de eventos a cada botón para poder realizar las acciones pertinentes
 - Asignación de eventos para obtener input del teclado
 - Creación de vista que muestre el historial de transacciones por medio del botón <, ocultando el teclado numérico y mostrando allí el historial de operaciones
 - Utilizar el botón "diferente de" para mostrar o no la venta que muestra la operación actual