UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL SOFTWARE PARA LA WEB

Trabajo Fin de Máster

ESTUDIO DE LA FUNCIONALIDAD DE VUE 3 EN APLICACIONES WEB

Juan José Navarro Morales

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN

INGENIERÍA DEL SOFTWARE PARA LA WEB

Trabajo Fin de Máster

"ESTUDIO DE LA FUNCIONALIDAD DE VUE 3 EN APLICACIONES WEB"

Autor: **Juan José Navarro Morales**

Director: **Carlos Delgado**

Tribunal:

Presidente: .....

Vocal 1º: ..... Vocal 2º: .....

Calificación: .....

Fecha: ..... de ..... de .....

En agradecimiento a los compañeros y profesores que estuvieron durante todo el año. Así como a mis amigos y familia que siempre han estado y están.

ÍNDICE RESUMIDO

ÍNDICE DETALLADO

ÍNDICE DE FIGURAS

INTRODUCCIÓN

OBJETIVOS DEL PROYECTO

ESTADO DEL ARTE

Vue 3

Desarrollo prototipo

Conclusión

Anexo

INTRODUCCIÓN

A medida que la tecnología ha evolucionado y se ha extendido el uso de internet por el mundo, se ha incrementado el desarrollo de aplicaciones con tecnologías web que junto con el creciente número de usuarios que hacen un uso diario de este tipo de aplicaciones, nos encontramos en un punto álgido de esta tecnología. Cada vez son más las facilidades proporcionadas por este tipo de tecnologías encontrándonos una gran variedad de frameworks, tanto para back-ends como para front-ends, donde elegir según las necesidades requeridas por los clientes.

Dentro de esta tecnología para la web, nos encontramos con el desarrollo web de tipo front-ends el cual se encarga de la interacción con el usuario mediante una interfaz gráfica. Para ello podemos encontrar un sinfín de frameworks que nos ayuden en nuestro propósito como pueden ser Angular, React Vue, Ember, etcétera.

Uno de estos frameworks que más ha evolucionado en estos tiempos ha sido el denominado "Vue". Éste se está haciendo un hueco importante en el mercado de aplicaciones web con cada versión que va desplegando, mejorando así la experiencia recopilada durante el uso de los desarrolladores de la versión anterior. La última versión publicada fue en septiembre del 2020 con "Vue 3.0".

En esta última versión "Vue 3.0", podemos observar un gran conjunto de mejoras en diferentes aspectos, como son la usabilidad del framework, nivel de rendimiento, eliminación de plugins que ahora se incluyen de forma nativa, así como, composition API el cual es el mayor cambio que nos encontramos en esta versión frente a "Vue 2".

Por lo tanto, se propone la investigación, análisis y comparación del frameworks front-ends Vue 3 y, además, la realización de un pequeño prototipo de aplicación web realizada con esta herramienta siguiendo las fases de análisis, diseño, implementación y pruebas.

OBJETIVOS DEL PROYECTO

El objetivo principal de este Trabajo Fin de Máster es poder poner en práctica conocimientos adquiridos durante el curso. Para ello veremos el estado del arte de las aplicaciones web en la actualidad y observaremos los diferentes frameworks más utilizados en la actualidad para el desarrollo front-ends. Además, la realización de un estudio sobre las funcionalidades, novedades, desventajas y comparación del framework Vue en su última versión.

Objetivos específicos derivados del objetivo principal:

1. Estudio del ámbito del proyecto.
2. Profundización en el framework front-end Vue en su última versión.
3. Desarrollo de un prototipo con Vue 3.
4. Elaborar la memoria final del Trabajo Fin de Máster.

ESTADO DEL ARTE

En este primer capítulo nombrado *Estado del Arte,* se pretende explicar el estado actual de las tecnologías en las que se basa el tema central de este proyecto. Explicando de forma clara y concisa todo lo que engloba las aplicaciones web actualmente, pasando por alguna anotación histórica para poder comprender la evolución. Además, presentaremos la información obtenida tras una pequeña investigación sobre distintos frameworks front-ends que posean características similares al que en este documento presentamos.

1.1 Aplicaciones web

En esta sección podremos ver alguno de los hitos más importantes en la historia de internet como tecnología esencial para las aplicaciones web, así como, entender el concepto de aplicación web junto con la estructura utilizada.

1.1.1 Historia de Internet

Para partir de un contexto común, la Real Academia Española nos proporciona la siguiente definición de Internet: "Red informática mundial, descentralizada, formada por la conexión directa entre computadoras mediante un protocolo especial de comunicación."[1] .

El desarrollo de lo que conocemos como Internet ha sido fruto del trabajo de miles de personas, aunque hay algunas de esas personas que han conseguido dejar su nombre marcado en el hilo de hitos conseguidos.

Lawrence Roberts ha sido denominado comúnmente como "el padre de Internet" debido a que fue el director del equipo que creo "ARPANET", una red de computadores conectados en diferentes localizaciones, el precedente del actual Internet. ARPANET fue desarrollado por la ARPA (Agencia para Proyectos de Investigación Avanzados), institución dependiente del departamento de defensa de los EEUU. El 5 de diciembre de 1969 se estableció la primera interconexión de ARPANET entre los diferentes nodos localizados en la Universidad de Utah, la Universisdad de California en los Ángeles, el Stanford Research Institute y la Universidad de California en Santa Barbara.

Vinton Cerf y Robert Kahn contribuyeron con el desarrollo del protocolo TCP/IP (Transmission Control Protocol / Internet Protocol) en la década de 1970, con el cual todo ordenador pudiese comunicarse con cualquier otro, sin depender del hardware o software. Este protocolo es usado actualmente en las comunicaciones de internet permitiendo conectar redes independientes entre sí.

Por último, nos encontramos con Tim Berners-Lee denominado como "el padre de la Web". Tim creó el lenguaje de marcado HTML, sobre 1980, con el que se generan los documentos para la Web, el protocolo de comunicación en la web HTTP (Hyper Text Transfer Protocol) entre los años 1989-1991, así como, las URL (Universal Resource Locator) como identificador de recursos en Internet en 1991 [2].

1.1.2 Qué es una aplicación web

Una aplicación web es un software desarrollado para ser ejecutado por diferentes navegadores de internet, intranet o redes locales. Este tipo de software, que es ejecutado por medio de un navegador web como cliente. no necesita ser instalado en un dispositivo físico para su uso, debido a que la información se encuentra almacenada en una red o en la nube.

Aunque los conceptos de sitio web y aplicación web puedan tener cierto parecido, estos no son exactamente lo mismo. El objetivo de un sitio web es aportar información que, a diferencia de una aplicación web, es orientada a la ejecución de ciertas acciones. Por lo tanto, podemos observar como el propósito de las aplicaciones web es prestar un servicio online ofreciendo soluciones a problemas concretos.

Una aplicación web elimina la responsabilidad de desarrollar el software para un determinado hardware o sistema operativo, teniendo un abanico muy amplio de compatibilidad entre los diferentes navegadores posibles.

1.1.3 Estructura de las aplicaciones web

Las aplicaciones web actualmente se modelizan mediante el denominado modelo de capas, donde cada capa representa un procesamiento o tratamiento de información. Podemos encontrarnos distintos tipos:

- Modelo de dos capas:
  - Capas:
    - Cliente: Gestiona la lógica de negocio en el lado del cliente (navegador).
    - Servidor: Administra los datos.
  - Limitaciones:
    - Escasamente escalable.
    - Número bajo de conexiones.
    - Funcionalidad limitada.

- Modelo de n-capas:
  - El más común es el modelo de tres capas.

El modelo de tres capas fue diseñado para sobrepasar las limitaciones que nos encontrábamos trabajando con el modelo de dos capas introduciendo una tercera capa. Obteniendo las siguientes capas [4]:

- Capa de presentación: Denominada como interfaz de usuario es la visión del sistema por parte del usuario, mostrándole información, así como, interactuando con el usuario. Esta capa se comunica únicamente con la capa de negocio.

- Capa de negocio: Responde y recibe las peticiones realizadas por el usuario (como puede ser un clic o introducir información). Esta capa es la intermediaria entre la capa de presentación para recibir solicitudes o presentar resultados y de la capa de datos para almacenar o recuperar los datos.

- Capa de datos: Compuesta por algún gestor de bases de datos (Oracle, MySQL, Cassandra, SQLite, mongoDB) donde se retienen los datos y se reciben las solicitudes para almacenar o recuperar información desde la capa de negocio.

Un esbozo de una arquitectura en tres capas se puede visualizar en la Figura 1 a continuación.

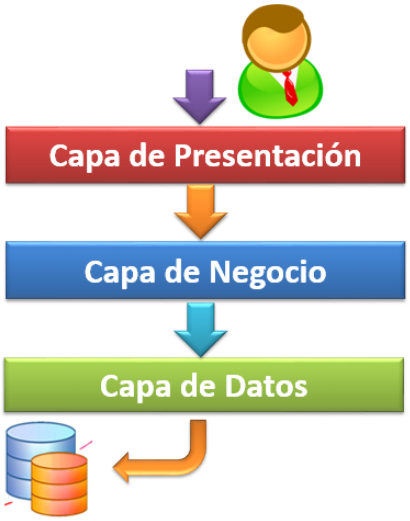


Figura 1. Esquema de una arquitectura en tres capas. Fuente: [3]

1.2 Capa de presentación

Una vez conocido el modelo de capas, en este apartado nos centraremos en la capa de presentación de las aplicaciones web. Para ello partiremos de una definición común sobre qué es un framework.

Un framework es "un patrón o esquema que ayuda a la programación a estructurar el código y a ahorrar tiempo y esfuerzo a los programadores. Se trata de una herramienta versátil, ya que está incompleta y, al añadirle líneas de código, la convertimos en una determinada aplicación"[5].

1.2.1 Framewowrks front-ends

Los frameworks frontends nos ofrecen la base para crear y, además, nos permiten ser flexible respecto a los diseños a implementar. Actualmente, encontramos una multitud de diferentes frameworks con los que poder realizar un mismo trabajo, aunque cada uno dispone de una curva de aprendizaje distinta, así como sus peculiaridades, conformando diferentes ventajas y desventajas.

En la "Figura 2" que se muestra a continuación, se puede visualizar cuáles han sido los frameworks más de "moda" y cómo han ido variando en los últimos años, según el porcentaje de preguntas obtenidas en el famoso dominio "Stack Overflow".

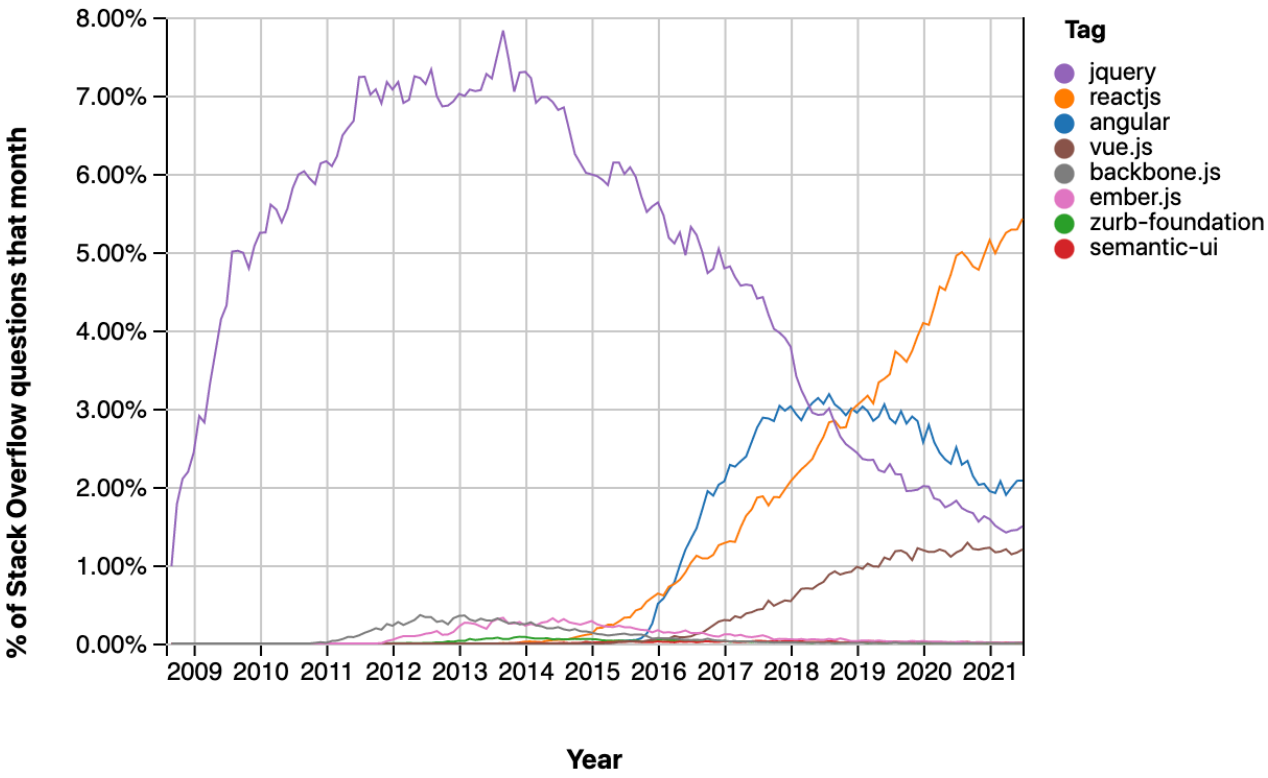


Figura 2. Porcentaje de preguntas en Stack Overflow sobre distintos frameworks. Fuente: [6]

En la figura anterior, también se observa que, actualmente, los frameworks más usados son "React" y "Angular", sin embargo, "Vue" se va haciendo un hueco cada vez mayor en el mercado.

A continuación, procederemos a ver algunas de las características más importantes de Angular y React. En el siguiente capítulo de este trabajo, nos centraremos en Vue y sus características más importantes.

- Angular creado por Google es un framework para la web, open-source, con una arquitectura modelo-vista-controlador y basado en TypeScript. Su popularidad apareció para el desarrollo de las páginas de tipo SPA (Single Page Applicaions). Usado por Sony, PayPal, The Guardian, …[9]

- React creada por Facebook es una librería Javascript open-source con el objetivo de construir interfaces de usuarios. Trabaja con un DOM (Document Object Model) virtual y ha sido destacado por la capacidad de generar interfaces de usuarios complejas sin bajar el rendimiento. Solo ofrece la capa de vista del MVC (modelo-vista-controlador), las otras capas tienen que agregarse mediante el uso de otras librerías. Usado por AirBnB, Instagram, UberEats, … [10]

En la Figura 3, podemos observar alguna de las diferencias existentes entre Angular y React con respecto a un conjunto de características principales.

Atributos	Angular	React
Tipo	JavaScript framework	Open Source JavaScript Library
Número de descargas semanales en NPM (2018)	444,794	5,036,078
Tamaño	167 KB production 1,2 MB development	109,7 KB production 774,7 KB development
Data Binding	Bi-direccional	Uni-direccional
Aplicaciones web completas	Se puede usar de forma independiente.	Necesita ser integrado con otras herramientas.
Rendering	En el lado del cliente.	En el lado del servidor.
Modelo	MVC	DOM Virtual
Código reutilizable	Si	No, solo CSS

Figura 3. Comparación Angular vs React. Fuente: [11]

VUE 3

En este capitulo, denominado "Vue 3", conoceremos qué es Vue, qué novedades obtenemos en su última revisión y, además, se añadirá Vue a la comparación de los frameworks ya comentados en el capítulo anterior: Angular y React.

1.1 ¿Qué es Vue?

Vue es un framework progresivo JavaScript, esto quiere decir que las principales características, como pueden ser el renderizado y el sistema de componentes, se encuentran ubicadas en una pequeña biblioteca, aunque es posible añadir todas las funcionalidades que se vayan requiriendo según el tipo de proyecto. Algunas de esas características que pueden ser añadidas, que ya traen de partida otros frameworks, son el routing, en el lado del cliente, o manejo de estados mediante vuex o redux. Esta filosofía permite incluir en la aplicación solo las funcionalidades requeridas.

El principal objetivo que nos encontramos con Vue es la creación de interfaces de usuarios, es por eso por lo que la palabra Vue pronunciado /vju:/, como view, es como se denomina a la capa de presentación del modelo MVC (modelo-vista-controlador).

Una de las características, que hace muy cómodo el desarrollo en este framework, es que permite dividir las aplicaciones en distintos bloques funcionales independientes llamados componentes, esto se suele denominar arquitectura de componentes. Esta arquitectura nos permite generar aplicaciones a base de la reutilización de estos componentes, evitando tener código redundante.

Otra de las virtudes de Vue es la capacidad de que las interfaces sean reactivas. Esto significa que se actualiza el HTML y CSS cuando se modifican datos de la aplicación sin que los programadores tengan que realizar una propagación de los cambios de datos de manera manual.

1.1.1 Ecosistema de Vue

Dentro del conjunto de herramientas de Vue que podemos ir incluyendo en los proyectos según las distintas necesidades donde cada uno realiza una determinada tarea, podemos distinguir los principales:

- Vue: Núcleo del framework, en el nos encontramos las funciones principales.
- Vue CLI: Gestor para crear y administrar desde el entorno gráfico o la terminal de comandos los proyectos de Vue.
- Vue Router: Sistema para crear y gestionar el enrutado de las URL del navegador.
- Vuex: Gestor de estados para las aplicaciones SPA (single page application).
- Vue Test Utils: API para poder realizar test sobre la aplicación Vue.

1.2 Novedades Vue 3

En septiembre del 2020 se liberó lo que conocemos como Vue 3. Todas las novedades de esta nueva versión han relanzado el framework como uno de los más provechosos, obteniendo los desarrolladores una mayor agilidad en su uso.

En esta versión todo el framework ha sido implementado bajo TypeScript, el cual ahora se incorpora por defecto en Vue. Esto facilita la reutilización de código y una detección temprana de errores.

Otro punto que todo desarrollador tiene en cuenta es el rendimiento obtenido del conjunto de herramientas empleadas, para ello Vue ha realizado mejoras para obtener un rendimiento superior. Algunas de las mejoras obtenidas se deben a [13]:

- El núcleo se ha visto reducido a la mitad aproximadamente.

- El API de Vue 3 es "tree-shakeable", esto significa que todo aquel código que no se utilice podrá ser retirado. Esto provoca que no ocupe espacio o tiempo de procesamiento en las aplicaciones compiladas.

- Capacidad de detectar y pausar procesos de renderizaciones innecesarios, al introducir un nuevo método del ciclo de vida en los componentes.

- Se ha implementado mediante un nuevo API el motor de reactividad. Una de las nuevas características que nos proporciona es la posibilidad de extraer diferentes partes estáticas de los componentes, para que así no tengan que procesarse cada vez que se modifican los datos. Esto permite disminuir la memoria usada y incrementar la velocidad de las renderizaciones de las vistas.

- En las antiguas versiones, cuando se generaba un componente solo podía estar enlazado a un elemento padre. Esto es debido a que toda instancia de Vue que representa un componente tiene que estar asociado a un solo elemento del DOM. La única manera de poder crear un componente con diferentes nodos DOM es generando un componente funcional que no dependa de una instancia de Vue. Este proceso se realizaba mediante el plugin "vue-fragments", pero en Vue 3 esta característica de los fragments se ha incluido de forma nativa.

- Los v-model son empleados para poder tener una conexión de datos bidireccional, pero en Vue 2 solo podíamos tener un v-model por componente. Esto en Vue 3 ha cambiado pudiendo tener tantos v-model como necesitamos. Cada v-model debe tener un nombre asignado diferente a los otros. Un ejemplo de su uso sería:

```
<LoginForm v-model:user="userName" v-model:passwd="userPasswd" />
```

En el archivo src/main.js o src/main.ts (si usamos TypeScript) el cual es el fichero principal, ya podemos observar cambios en esta nueva versión. En la Figura 4. Sintaxis main.js Vue2 vs Vue3, que se muestra a

continuación, podemos visualizar como se hace lo mismo en ambas versiones, pero en Vue 3 obtenemos una sintaxis mucho más agradable de leer y fácil de entender.

Vue 2	**Vue 3**

`const options = {`
`router: router,`
`store: store,`
`render: function(createElement) {`
`return createElement(App);`
`}`
`});`
`const app = new Vue(options);`
`app.$mount("#app");`
	`createApp(App)`
	`.use(store)`
	`.use(router)`
	`.mount("#app");`

Figura 4. Sintaxis main.js Vue2 vs Vue3

1.2.1 Composition API

Vue nos permite mediante la creación de componentes tener ciertas partes de la aplicación reutilizable, esto por sí solo ya nos ayuda en términos de facilidad de mantenimiento y flexibilidad. Aunque los creadores de Vue han llegado a la conclusión de que hace falta algo más cuando las aplicaciones incrementan y obtenemos cientos de componentes. Para ello en Vue 3 han introducido lo que denominan como "Composition API", cambiando la forma en la que se programaba, obteniendo un nuevo enfoque para la reutilización de código y la gestión de los proyectos.

Composition API ha sido diseñado para aumentar las diferentes capacidades de crecimiento de los proyectos, ayudando también en la reutilización de componentes.

Según con la versión que trabajemos podremos hacer uso de una u otra API:

- En Vue 2 solo se puede emplear Options API.
- En Vue 3 podemos utilizar Options API o Composition API.

En la Figura 5. Options API vs Composition API. Fuente: [12], se pueden observar las principales diferencias entre las dos APIs.

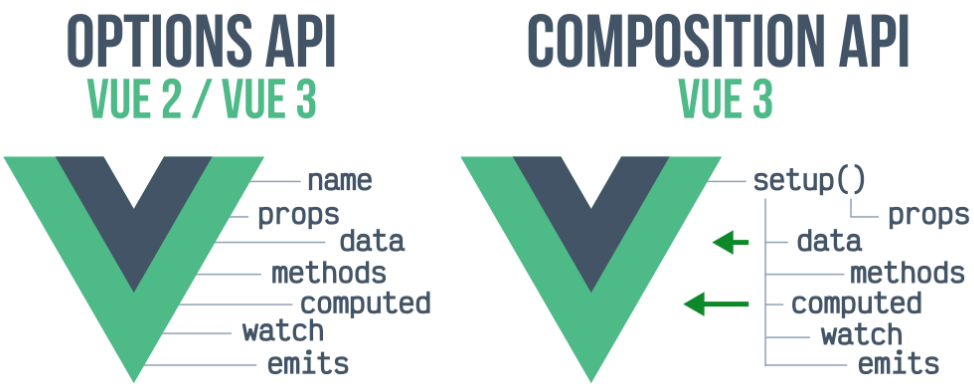


Figura 5. Options API vs Composition API. Fuente: [12]

Options API trabaja con un conjunto de "opciones", esto permite separar la información por nombre de estructuras. Esto hacia que sea muy sencillo para todo tipo de usuarios con o sin experiencia.

En contraposición Composition API trabaja con un método llamado "setup()", este método es como si fuese el constructor del propio componente. El método realizara un return con un object dónde encontraremos todos los elementos que queramos usar en el resto del componente.

En la Figura 6. Sintaxis Options API vs Sintaxis Composition API. se puede visualizar las sintaxis de ambas API.

Options API	**Composition API**

`export default {`
`  name: "ComponentName",`
`  props: { },`
`  data() { },`
`  computed: { },`
`  methods: { },`
`  mounted() { }`
`}`
	`export default {`
	`  name: "ComponentName",`
	`  props: {  },`
	`  setup() {`
	`  return {`
	
	`  }`
	`  }`
	`}`

Figura 6. Sintaxis Options API vs Sintaxis Composition API.

1.2.1.1 Setup

El método setup() es el cambio más característico de esta Composition API frente a la Options API, ya que se encargará de lanzarse durante la fase de inicialización del componente. Cuando setup() es ejecutado,

la instancia del componente aún no ha sido creada, por lo tanto, solo es posible acceder a las propiedades props, attrs, slots y emit. Las únicas a las que no tendríamos acceso en ese momento serían data, computed y methods. Dentro de setup(), "this" no contendrá una referencia a la instancia activa actual del componente debido a como comentábamos antes que setup() será ejecutado antes de que haya sido creada la instancia del componente completamente.

Setup() podrá contener dos parámetros opcionales:

- context es un objeto javascript no reactivo que incluye los attrs, emit y slot del componente.

- props es un objeto que contiene los props que hayan sido definidos previamente en el componente.

```
<template>
  <div>{{ total }}</div>
</template>


<script>
export default {
  props: {
    coste: Number
  },
  setup(props, context) {
    const total = `El precio total con IVA es de: ${props.coste * 0,21}`;
    return { total }
  }
}
</script>
```

Figura 7. Ejemplo simple setup(props, context) en Vue 3.

En la Figura 7. Ejemplo simple setup(props, context) en Vue 3. se puede observar un sencillo ejemplo de un componente de Vue con setup(), dónde nos encontramos con que la opción "props" se mantiene para definir las props del componente pero ha sido añadido setup() para generar la lógica de inicialización del componente. Así en setup() se realizarían las siguientes principales acciones:

- Definición: Todas las variables establecidas previamente en "data", las funciones definidas en "methods, las propiedades computadas de "computed" o los watchers de "watch" se convierten en variables o funciones definidos dentro del setup() del componente.

- Return: Todos aquellos objetos o funciones devueltos en el return del setup() podrán ser utilizados en todo el componente como por ejemplo en la sección "<template>" dónde nos encontraremos el HTML del componente.

- Inicialización: Todas las tareas previas a la inicialización del componente que antes se realizaban en "created(), mounted(), updated()" ahora son declaradas en setup().

1.2.1.2 Ciclo de vida

Los ciclos de vida han sufrido ciertas modificaciones. Antes eran definidas como métodos que se usaban como opciones, ahora en Vue 3 son un conjunto de funciones que son importadas desde el paquete principal de vue. La mayoría de los métodos han conservado el mismo nombre, aunque le han añadido un "on" antes del nombre del método. En la Figura 8, puede observarse el conjunto de métodos del ciclo de vida de un componente, así como las diferencias de nomenclatura por el cambio de versión.

Options API (Vue 2)	Options API (Vue 3)	Composition API (Vue 3)
beforeCreate	beforeCreate	Se implementa directamente en setup()
created	created	Se implementa directamente en setup()
beforeMount	beforeMount	onBeforeMount
mounted	Mounted	onMounted
beforeUpdate	beforeUpdate	onBeforeUpdate
Updated	Updated	onUpdated
beforeDestroy	beforeUnmount	onBeforeUnmount
destroyed	unmounted	onUnmounted
activated	activated	onActivated
desactivated	desactivated	onDesactivated

Figura 8. Comparación ciclo de vida en Vue 2 y Vue 3. Fuente: [12]

En la Figura 9, podemos ver un sencillo ejemplo de como utilizar algunos de los métodos comentados anteriormente del ciclo de vida. Solo es necesario importarlos para poder ser utilizados en setup().

```
<template>
  <div>{{ total }}</div>
</template>

<script>
import { onBeforeMount, onMounted } from "vue";
export default {
  props: {
    coste: Number
  },
  setup(props, context) {
    const total = `El precio total con IVA es de: ${props.coste * 0,21}`;
    onMounted(() => {console.log("mounted")});
    onBeforeMount(()=> {console.log("before mount")});
    return { total }
  }
}
</script>
```

Figura 9. Ejemplo setup() con funciones del ciclo de vida.

1.2.2 API de Reactividad

Otra de las modificaciones más significativas de Vue 3 es sobre el manejo de la reactividad dónde se ha convertido en algo más "explicito". En Vue 3 partimos de un conjunto de métodos a usar para generar variables u objetos reactivos. Los principales los podemos encontrar en la Figura 10, que se muestra a continuación.

Método	Información
ref()	Devuelve un number, string o boolean como objeto reactivo
reactive()	Devuelve el object como objeto reactivo.

readonly()	Devuelve el object como un objeto que no puede ser mutado.
isRef()	Verifica si una variable es una referencia reactiva de un dato primitivo.
isReactive()	Verifica si una variable es un object reactivo.
isReadonly()	Verifica si una variable es un object inmutable de sólo lectura.
isProxy()	Verifica si una variable es un proxy generado por reactive o readonly.

Figura 10. Métodos reactivos de Vue 3. Fuente: [12]

1.2.2.1 Función ref()

Esta función ref() es solo valida para tipo primitivo de datos como son los números, booleanos o strings. La función devuelve el dato en un objeto con la propiedad "value", la cual es a la que accederemos para transformar el valor de la variable reactiva. En la Figura 11, se puede observar un ejemplo de la función ref(), la cual solo necesita ser importada de la librería principal de Vue para poder utilizarla.

```
<template>
  <div @click="sumar">{{ contReactivo }}</div>
</template>
<script>
import { ref } from "vue";
export default {
  props: {
    cont: Number
  },
  setup(props) {
    const contReactivo = ref(props.cont);
    const sumar = () => contReactivo.value++;
    return { contReactivo, sumar }
  }
```

```
}
</script>
```

Figura 11. Ejemplo función ref().

1.2.2.2 Función reactive()

Esta función realiza el mismo trabajo que la función ref() vista anteriormente, con la diferencia de que ref() se aplicaba a tipos primitivos y reactive() se utiliza para aplicar reactividad a objetos, arrays, etc… Una diferencia entre estas dos funciones es que con la función reactive() no accedemos a la propiedad ".value" si no que accedemos directamente al objeto completo. En la Figura 12, podemos visualizar el mismo ejemplo visto anteriormente pero con las modificaciones necesarias para utilizar sobre un objeto la función reactive().

```
<template>
  <div @click="sumar">{{ contReactivo }}</div>
</template>
<script>
import { ref } from "vue";
export default {
  props: {
    cont: Number
  },
  setup(props) {
    const contReactivo = reactive({
      nombre: "Contador"
      cont: props.cont
    });
    const sumar = () => contReactivo.value++;
    return { contReactivo, sumar }
  }
}
</script>
```

Figura 12. Ejemplo función reactive().

1.2.2.3 Funciones computed(), watch() y watchEffect()

En Composition API seguimos teniendo las propiedades computadas y watchers, todas estas ahora se declaran en la función setup().

- computed() es una propiedad computada, es decir, cachea los datos hasta que los valores de las variables reactivas con las que se conformen no cambien. Esto evita realizar cálculos innecesarios que obtengan un mismo resultado, ya que solo se realizara cuando cambien los valores de las variables reactivas que lo componen.

- watchEffect() es una función la cual se ejecutará nada más generarse, y volverá a ejecutarse cada vez que se modifiquen aquellas dependencias reactivas que contenga.

- watch() realiza un tratamiento similar a la función whatchEffect() pero es posible indicar a que variable reactiva vamos a estar pendientes de si es modificada. También se puede obtener el valor previo y posterior de la variable reactiva modificada.

1.3 Comparaciones con otros frameworks (react, angular)

Como vimos en el capitulo del estado del arte, actualmente, los frameworks más usados son "React" y "Angular", sin embargo, "Vue" se va haciendo un hueco cada vez mayor en el mercado según el porcentaje de preguntas al portal Stack Overflow.

Para poder tener otra evidencia sobre el reciente crecimiento en popularidad que vemos en el framework Vue hemos utilizado la herramienta Google Trends. En la Figura 13, se puede observar como en los últimos 12 meses ha ido fluctuando el interés de los tres frameworks, pero destacamos como Vue esta cerca de sobrepasar a React el cual se encontraría actualmente en primer lugar.

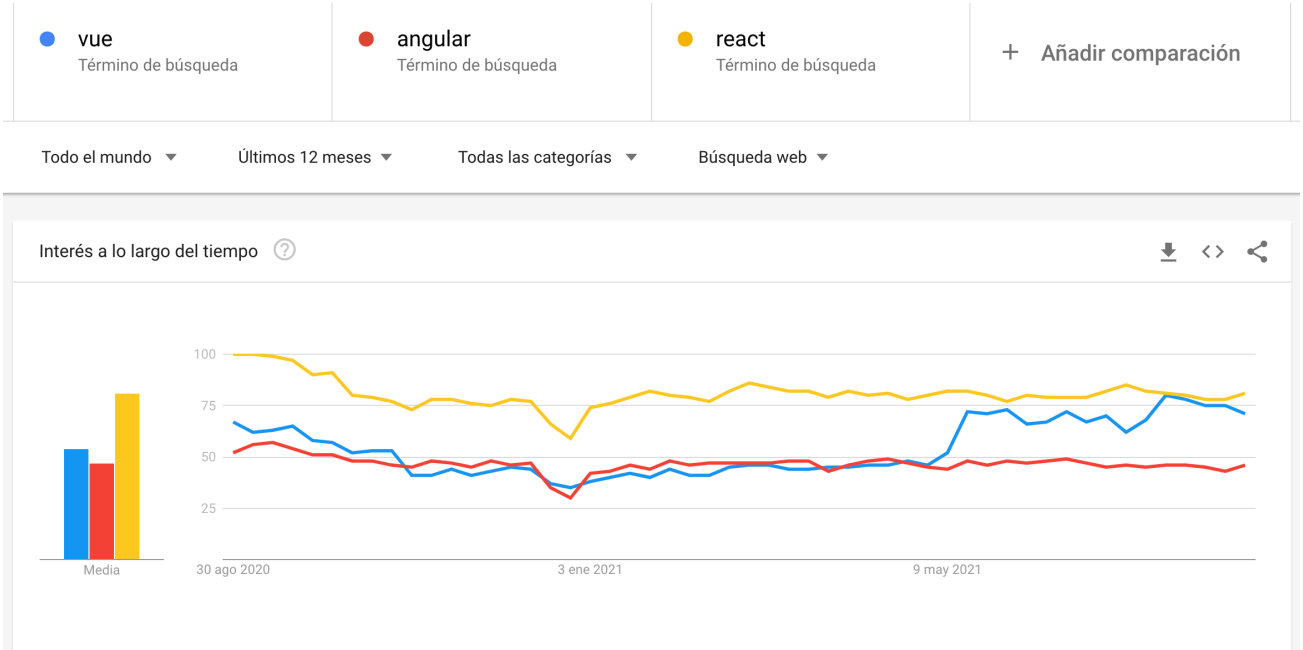


Figura 13. Comparación en Google Trends del interés sobre los términos Vue, Angular y React. Fuente: [7]

También en la Figura 3, pudimos observar alguna de las diferencias existentes entre Angular y React con respecto a un conjunto de características principales, pero habiendo visto más en profundidad Vue en la Figura 14, vemos la misma tabla de comparación con la diferencia de que le añadimos una nueva columna para poder visualizar de una forma sencilla a grandes rasgos las principales diferencias de los tres frameworks.

Atributos	Angular	React	Vue
Tipo	Framework JavaScript	Open Source JavaScript Library	Framework progresivo JavaScript
Número de descargas semanales en NPM (2018)	444,794	5,036,078	996,293
Tamaño	167 KB production 1,2 MB development	109,7 KB production 774,7 KB development	30,67 KB production 279 KB development
Data Binding	Bi-direccional	Uni-direccional	Bi-direccional
Aplicaciones web completas	Se puede usar de forma independiente.	Necesita ser integrado con otras herramientas.	Necesario utilizar herramientas de terceros.
Rendering	En el lado del cliente.	En el lado del servidor.	En el lado del servidor.
Modelo	MVC	DOM Virtual	DOM Virtual
Código reutilizable	Si	No, solo CSS	Si, CSS y HTML

Figura 14. Comparación Angular, React y Vue. Fuente: [11]

DESARROLLO PROTOTIPO

En este capitulo, veremos la información acerca del prototipo desarrollado mediante Vue 3. A modo resumen podemos ver que el prototipo es una aplicación web responsive para la compra de productos dónde en todo momento se visualizará la cesta de la compra junto con los productos que los usuarios añadan, pudiendo eliminarlos del carrito en cualquier momento. Todos los productos estarán clasificados en categorías y un administrador de la página podrá añadir nuevos productos y categorías dinámicamente.

1.1 Diagrama de casos de uso

Los diagramas de casos de usos son unos tipos de diagrama dónde poder observar el comportamiento esperado del sistema, desde la visión de los usuarios, sin llegar a un alto nivel de especificación acerca de como se implementan las acciones. Los principales elementos que nos encontramos en este tipo de diagramas son:

- Sistema: Representa con un rectángulo los limites del sistema, los actores se ubican fuera de la figura.

- Caso de uso: Representado con un óvalo y un texto identificativo, identifican una determinada acción.

- Actor: Son las entidades externas al sistema que interactúan con el.

Los diagramas de casos de uso del prototipo desarrollado los puedes encontrar en la Figura 15, que se muestra a continuación.

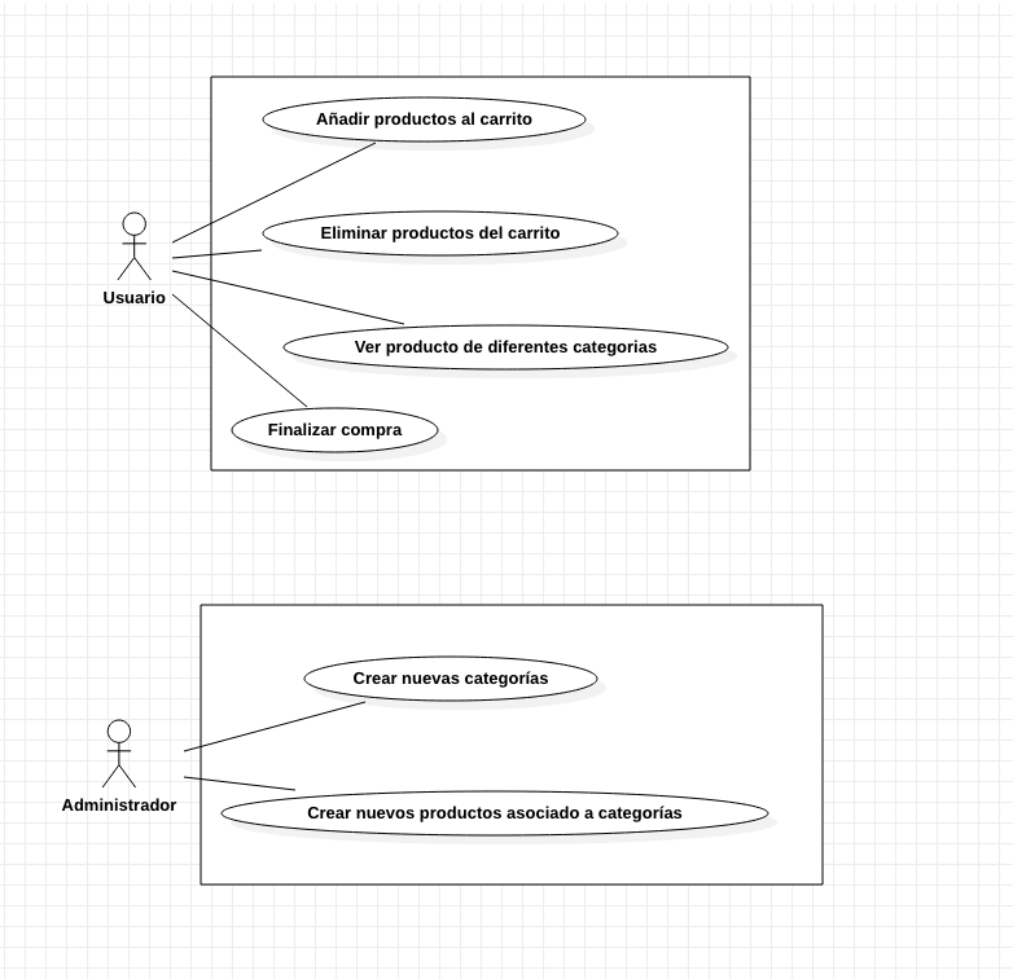


Figura 15. Diagrama casos de usos prototipo.

1.2 Diseño

Para la generación de la interfaz de usuario, se ha utilizado el uso de bocetos. Los bocetos se emplean para poder realizar diseños iniciales de aquellos elementos que queremos visualizar por pantalla.

1.2.1 Diseño de bocetos

Los bocetos iniciales tenían se realizaron con la intención encontrar la mejor ubicación de los diferentes elementos en pantalla. Así como unificar elementos comunes como son las categorías, productos o el carrito de la compra. En la Figura 16 y Figura 17, se puede observar la distribución realizada de los diferentes elementos en los bocetos realizados.

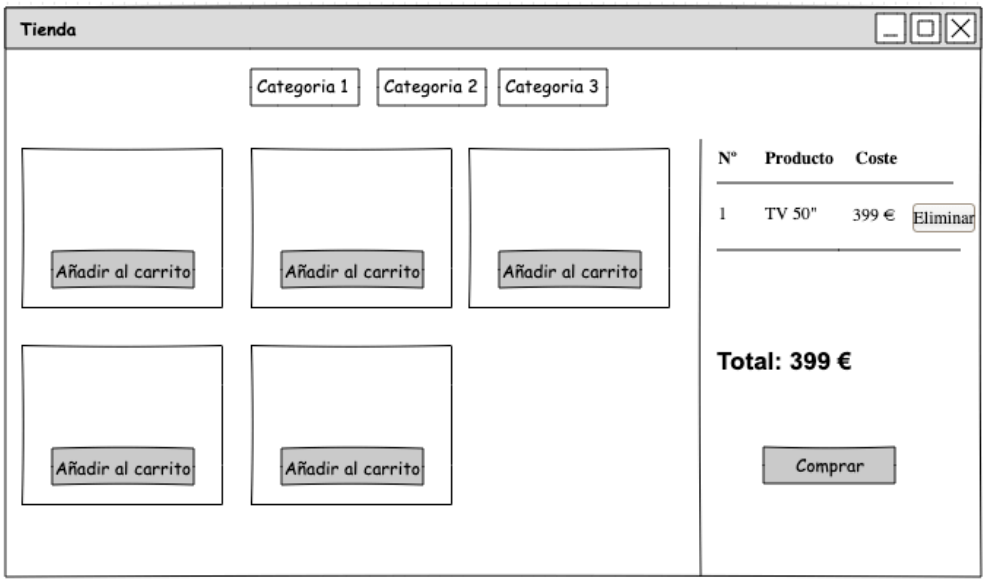


Figura 16. Boceto de la vista de usuario.

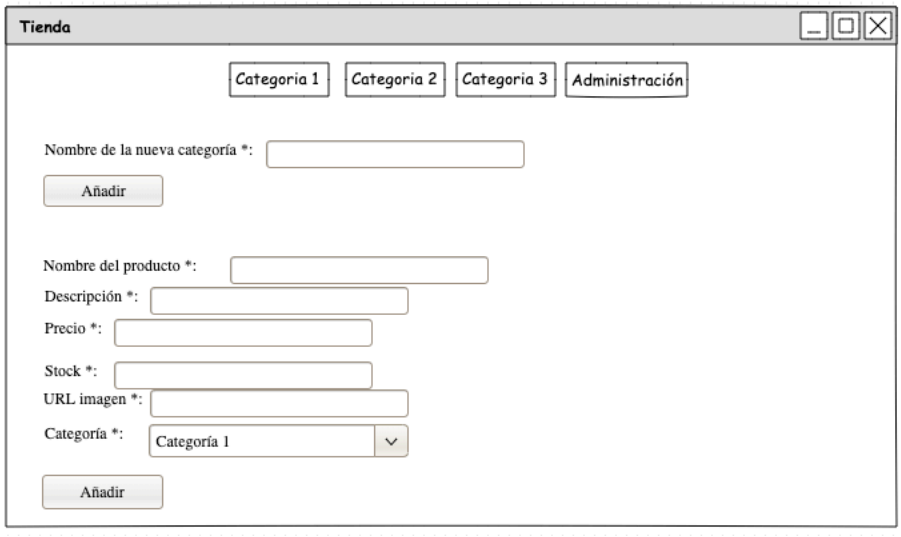


Figura 17. Boceto de la vista del administrador.

1.3 Implementación

En este apartado se comentan algunas de las partes de la implementación realizada para el desarrollo del prototipo con Vue 3.

1.3.1 Estructura

En la estructura del proyecto podemos diferenciar varios aspectos importantes:

- Fichero "app.vue", es el fichero principal que se lanza al cargar la aplicación en el que se encuentra la organización/estructuración principal de la página mediante dos columnas para los productos y el carrito, así como la sección para la cabecera dónde ubicamos la cabecera con las categorías.

- Store/index.js, en este fichero es dónde gracias a vuex implementamos lo que es la "lógica" común que necesitan tener varios componentes para su comunicación.

- En el directorio components se encuentran todos los componentes reutilizables, cada fichero con la extensión ".vue" contiene su propio html, css y javascript intependiente.

Esta estructura se puede visualizar en la Figura 18.

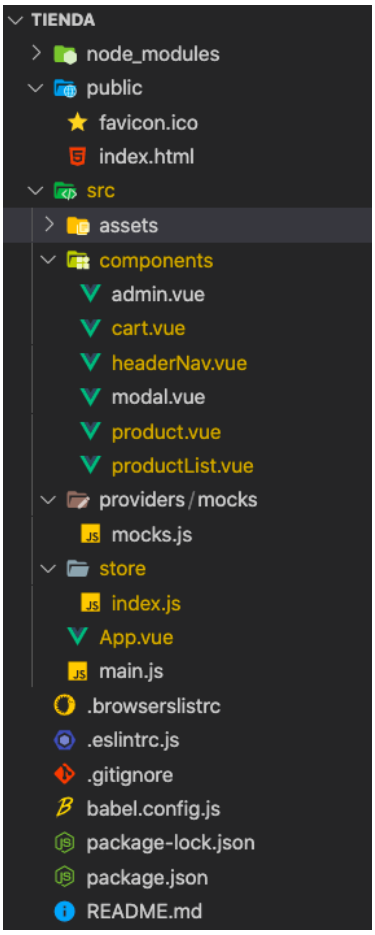


Figura 18. Estructura del Proyecto.

1.3.2 Aspectos relevantes

En la Figura 19, se puede observar el código del fichero principal del prototipo desarrollado denominado "app.vue". Que procedemos a comentar los puntos más destacado de este fichero y realizar referencias con alguna de las características de Vue 3 ya comentadas en este mismo documento.

El fichero esta dividido en tres secciones:

- <template> dónde incorporamos el código HTML.
    - En esta sección hacemos uso de los componentes reutilizables, alguno de ellos con renderización condicional usando "v-if". Para los estilos se ha utilizado el framework Boostrap, con el cual se ha implementado un grid responsive para obtener la estructura de la página con una fila para la cabecera, y otra fila con dos columnas una para los productos o la gestión del administrador, y otra columna para el carrito de la compra.

- <script> dónde encontramos el código JavaScript. En este apartado nos encontramos con los siguiente:
    - Importación de librerías y componentes a utilizar.

    - data(), se ubican un objeto con datos para la instancia del componente. Es una función para que así cada instancia pueda tener una copia independiente del objeto de datos devueltos en el return.

    - components, sección dónde se realiza el registro de los componentes que se van a utilizar en el actual.

    - Composition API, el cambio más característico de esta versión de Vue lo utilizamos con la función setup(), la cual se ejecuta al inicializar el componente. En el return de la función devolvemos todos aquellos objetos o funciones que queramos usar en el componente.

    - Ciclo de vida, usamos onMounted() para inicializar algunas propiedades que requieren tener valor nada más haya sido montado el componente.

    - Propiedades computadas, computed() cachea los datos hasta que los valores de las variables reactivas con las que se conforma no cambien. Esto nos evita realizar cálculos innecesarios.

    - Vuex, useStore se implementa para hacer tener una gestión de estados centralizada.

- <style> dónde se implementa el CSS de la página.

```
<template>
  <div>
    <modal v-if="showModal" :title="modalTitle" :description="modalDescription"/>
  </div>
  <div class="container">
    <div class="row">
      <headerNav/>
    </div>
    <div class="row mt-4">
      <div class="col-sm-9">
        <productList v-if="currentCategory !== 'Administración'"/>
        <admin v-if="currentCategory === 'Administración'"/>
      </div>
      <div class="col-sm-3">
        <cart/>
      </div>
    </div>
  </div>
</template>
<script>
// Importamos bootstrap a nivel del proyecto así todos los componentes hijos lo tendrán sin necesidad de importarlo
import "bootstrap/dist/css/bootstrap.min.css";
// Importamos componentes
import cart from './components/cart.vue';
import admin from './components/admin.vue';
import productList from './components/productList.vue';
import headerNav from './components/headerNav.vue';
import modal from './components/modal.vue';
import { useStore } from 'vuex';
import { computed, onMounted } from 'vue';
export default {
  name: "App",
  data () {
```

```
    return {
      modalTitle: 'Compra realizada',
      modalDescription: 'Vuelve a buscar entre nuestras ofertas cuando quieras.'
    }
  },
  components: {
    cart,
    productList,
    headerNav,
    admin,
    modal
  },
  setup() { // Composition API
      const store = useStore();
      onMounted(() => { // Ciclo de vida
        store.dispatch('initProductos');
        store.dispatch('initCategorias');
      })
      const showModal = computed(() => store.getters.getShowModal);
      const currentCategory = computed(() => store.state.currentCategory);
      return {currentCategory, showModal}; // devolvemos los objetos que se usaran
en el html
    }
};
</script>
<style>
#app {
}
</style>
```

Figura 19. Código del fichero principal del prototipo "App.vue"

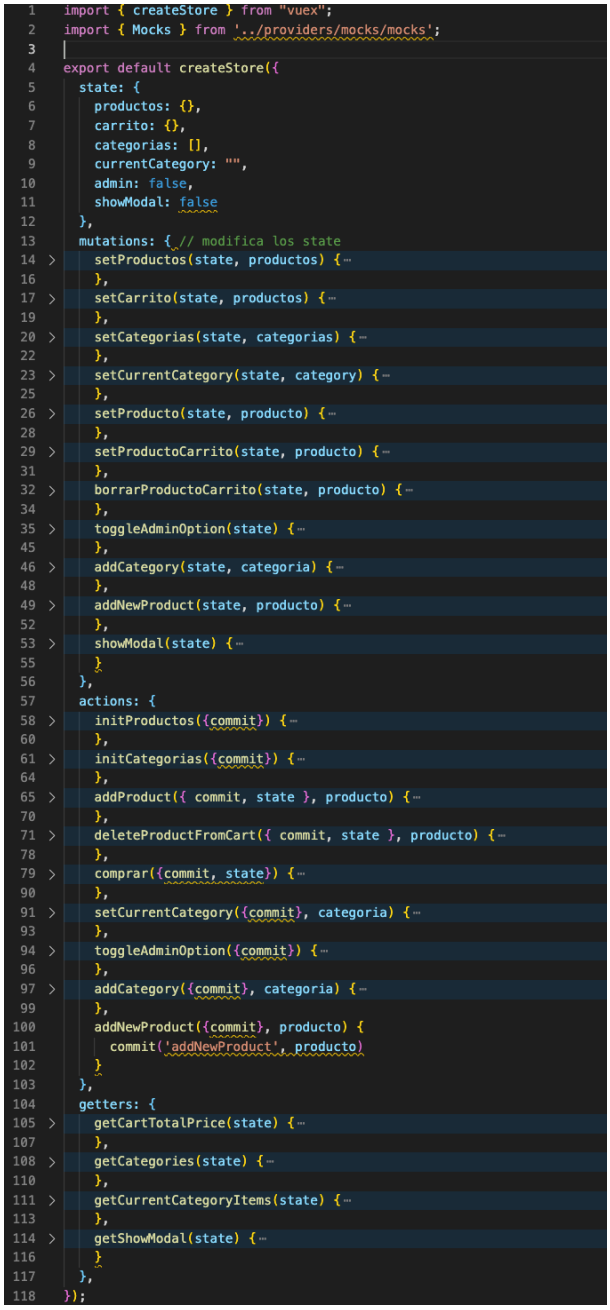
Como hemos visto anteriormente se hace uso de Vuex, mediante su API de referencia Store que se implementa para hacer tener una gestión de estados centralizada. En la Figura 20, se puede observar como esta estructurado el fichero en el cual nos encontramos cuatros secciones:

- state, nos encontramos con los datos/atributos/estados. Por ejemplo, un array con las categorías de los productos, un objeto que contenga los productos a comprar por un usuario.

- mutations, son las funciones que pueden modificar los valores de los state. Por ejemplo, modificar los productos del carrito.

- actions, realiza el conjunto de acciones necesarias haciendo uso de las mutations para actualizar posteriormente los state con los resultados. Por ejemplo, añadir un nuevo producto o categoría.

- getters, usado para generar algunos datos que necesitan algunos cálculos. Por ejemplo, obtener el coste total de los productos del carro de la compra.



```
import { createStore } from "vuex";
import { Mocks } from '../providers/mocks/mocks';

export default createStore({
  state: {
    productos: {},
    carrito: {},
    categorias: [],
    currentCategory: "",
    admin: false,
    showModal: false
  },
  mutations: { // modifica los state
    setProductos(state, productos) {
    },
    setCarrito(state, productos) {
    },
    setCategorias(state, categorias) {
    },
    setCurrentCategory(state, category) {
    },
    setProducto(state, producto) {
    },
    setProductoCarrito(state, producto) {
    },
    borrarProductoCarrito(state, producto) {
    },
    toggleAdminOption(state) {
    },
    addCategory(state, categoria) {
    },
    addNewProduct(state, producto) {
    },
    showModal(state) {
    }
  },
  actions: {
    initProductos({commit}) {
    },
    initCategorias({commit}) {
    },
    addProduct({ commit, state }, producto) {
    },
    deleteProductFromCart({ commit, state }, producto) {
    },
    comprar({commit, state}) {
    },
    setCurrentCategory({commit}, categoria) {
    },
    toggleAdminOption({commit}) {
    },
    addCategory({commit}, categoria) {
    },
    addNewProduct({commit}, producto) {
      commit('addNewProduct', producto)
    }
  },
  getters: {
    getCartTotalPrice(state) {
    },
    getCategories(state) {
    },
    getCurrentCategoryItems(state) {
    },
    getShowModal(state) {
    }
  },
});
```

Figura 20. Fichero con createStore de la utilidad Vuex.

CONCLUSIÓN

Los frameworks frontend llegaron y cambiaron todo el ecosistema actual existente en la web, proponiendo conjuntos de herramientas para facilitar el desarrollo de todo tipo de aplicaciones web. Además, esto propicio el aumento del número de desarrollos realizados de todo tipo de software web, debido a la reducción de tiempo y costo de los proyectos.

Durante la confección de este proyecto se ha podido ahondar en las diferentes etapas para el desarrollo de un prototipo. El cual ha permitido el aprendizaje en profundidad de algunas tecnologías vistas durante el plan de estudio como ha sido principalmente en este caso del framework Vue en su última versión, pero trabajando también con diferentes herramientas como son Git para el control de versiones, herramientas de diseño de interfaces mediante bocetos para un estudio sobre la colocación de los elementos en pantalla, etc…

Tras analizar y estudiar este frameworks podemos confirmar que se encuentra ya lo suficientemente estable para un uso generalizado de desarrollos para la mayoría de los proyectos que solemos denominar de tamaños pequeños y medianos, ya que nos garantiza una buena calidad, con amplias funcionalidades a poder incluir en el proyecto desde un repositorio oficial y solo lo requerido por el proyecto, ahorrándonos tener basura y un software de menos tamaño. Para grandes proyectos todavía puede quedarle alguna revisión más y poder acortar distancias como puede ser con Angular un peso pesado muy pensado para grandes aplicaciones, pero saber encontrar la para un correcto funcionamiento en todo tipo de proyectos es muy complicado.

Como hemos visto, en la actualidad, cada vez son más personas las que se introducen en el mundo de Vue gracias a su rendimiento y buenas practicas. Esto ha provocado que la comunidad de desarrolladores haya incrementado muy rápidamente, esto puede marcar una gran diferencia con otros competidores directos a la hora de escoger un framework u otro.

Tal ha sido el crecimiento de Vue, que frameworks muy utilizados como "Ionic", el cual es un SDK frontend para el desarrollo de aplicaciones híbridas basado en tecnologías web ya da soporte a Vue junto con React, Angular y JavaScript. Esto nos permite aumentar de una forma fácil el número de tipos de proyectos que podemos desarrollar con Vue, pudiendo tener con un mismo código un entorno web y apps tanto para Android como para iOS.

BIBLIOGRAFÍA

[1] Real Academia Española. Diccionario de la lengua española. Recuperado el 25 de agosto de 2021, de https://dle.rae.es/

[2] Sergio Luján Mora. Programación de aplicaciones web: historia, principios básicos y clientes web. Editorial Club Universitario, Alicante, 2002. ISBN: 978-84-8454-206-3. Recuperado el 15 de agosto de 2021 de https://sergiolujanmora.es/verpdf/42.

[3] Walter R. Ojeda Valiente, Entendiendo la programación en 3 (o más). Recuperado el 16 de agosto de 2021 de https://vfpavanzado.wordpress.com/2017/10/31/entendiendo-la-programacion-en-3-o-mas-capas/

[4] Wikipedia, Programación por capas. Recuperado el 20 de agosto de 2021 de https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas

[5] UOC, Qué es un framework en programación. Recuperado el 27 de agosto de 2021 de https://fp.uoc.fje.edu/blog/que-es-un-framework-en-programacion/

[6] Stack Overflow, % de preguntas en Stack Overflow respecto a diferentes frameworks. Recuperado el 27 de agosto de 2021 de https://insights.stackoverflow.com/trends?tags=angular%2Creactjs%2Czurb-foundation%2Csemantic-ui%2Cjquery%2Cvue.js%2Cember.js%2Cbackbone.js

[7] Google Trends, Comparación del interés a lo largo del tiempo de los términos Vue, Angular y React. Recuperado el 28 de agosto de 2021 de https://trends.google.es/trends/explore?q=vue,angular,react

[8] Vue.js, Documentation. Recuperado el 1 de julio de 2021 de https://v3.vuejs.org/guide/introduction.html

[9] Angular, Documentation. Recuperado el 25 de agosto de 2021 de https://angular.io/docs

[10] React, Documentation. Recuperado el 25 de agosto de 2021 de https://es.reactjs.org/docs/getting-started.html

[11] Anand Mahajan, React vs Angular vs Vue. Recuperado el 28 de agosto de 2021 de https://unpocodejava.com/2020/01/15/react-vs-angular-vs-vue/

[12] Framework Vue.js. Recuperado el 5 de julio de 2021 de https://lenguajejs.com/vuejs/

[13] Fernán García de Zúñiga, Vue 3: Estas son sus principales novedades. Recuperado el 5 de julio de 2021 de https://www.arsys.es/blog/vue3/

[14] Scott Chacon y Ben Straub, "Pro Git". Recuperado el 30 de agosto de 2021 de https://www.git-scm.com/book/en/v2

[15] Wikipedia, GitLab. Recuperado el 30 de agosto de 2021 de https://es.wikipedia.org/wiki/GitLab

ANEXO

A continuación, se visualizan capturas del prototipo desarrollado.

En las Figura 21 y Figura 22, se observa la configuración de la cabecera para usuario administrador y no administrador.

Figura 21. Cabecera con las categorías en modo administrador.

Figura 22. Cabecera con las categorías en modo usuario normal.

El siguiente modulo que se observa en la Figura 23, muestra todos los productos con los que un usuario puede interactuar.

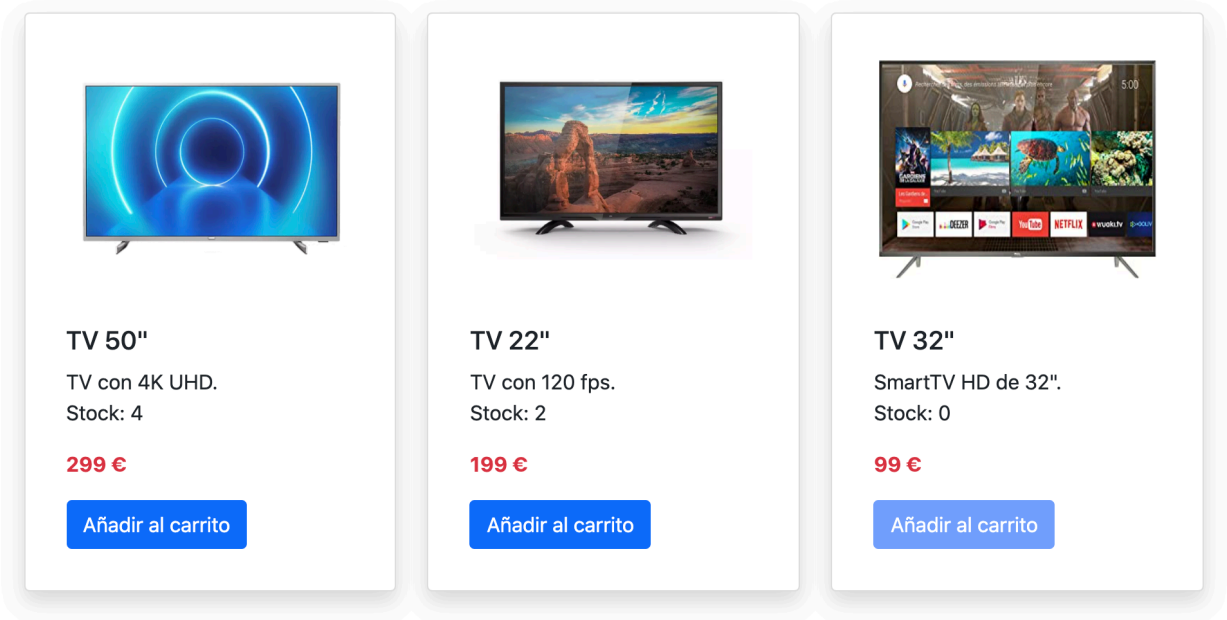


Figura 23. Componente que muestra el conjunto de productos.

El componente que se muestra en la Figura 24, se visualiza en la columna de la derecha de la pantalla. Este componente muestra a los usuarios los productos que ha añadido al carrito junto con la cantidad de cada uno, el coste individual, el coste total. Además, puede realizar la acción de eliminar un producto del carrito haciendo click en el icono con el símbolo menos o finalizar la compra pulsando el botón "comprar".

Figura 24. Componente con la configuración del carrito.

A continuación, en la Figura 25, se puede visualizar los formularios para la creación de nuevas categorías y productos de forma dinámica por parte de un usuario administrador.

Figura 25. Formularios para la creación de nuevos productos y categorías por parte del usuario administrador.

Esta última imagen que se visualizamos en la Figura 26, es un componente modal utilizado para indicar al usuario el fin del proceso de la compra, lanzado cuando el usuario realiza click en el botón "comprar" del carrito.

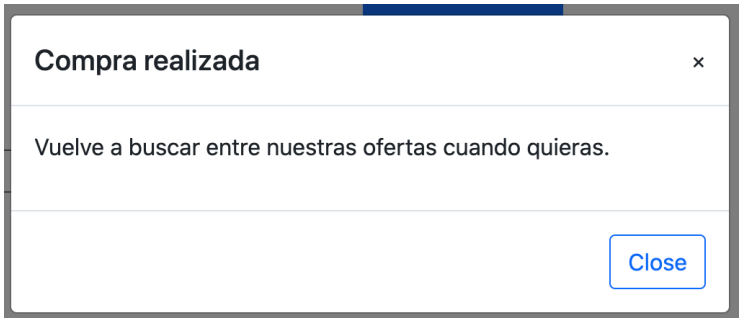


Figura 26. Componente modal que se visualiza cuando el usuario pulsa el botón "comprar" del carrito para notificar el fin de la compra.