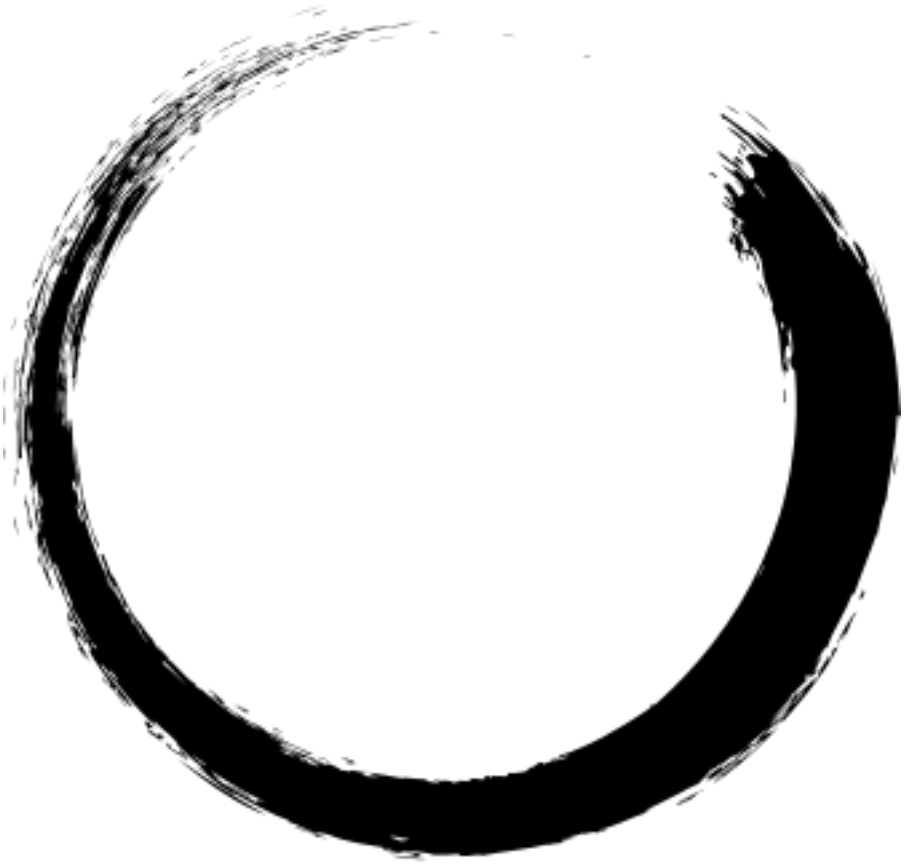
WhiteMarket

Oscar LLorens Lurbe

May 28, 2018



PROYECTO FINAL DE DAW DEL
I.E.S L'ESTACIO

WhiteMarket

Realizado por: Oscar LLorens Lurbe

May 28, 2018

Contents

1 Introducción

1 Idea del proyecto

La idea nace de la necesidad de comprar productos de segunda mano en diferentes localidades, en las actuales plataformas de compra y venta en línea, el comprador está muy perjudicado porque debe comprar el producto y rezar para que el vendedor sea buena persona y le envíe el producto. De la anterior necesidad nace la idea de este proyecto para hacer una aplicación en la que se pueda comprar y vender productos, nuevos o de segunda mano directamente con el cliente, y hacer un sistema de pago en el que el dinero no llega directamente al vendedor sin que el comprador tenga su producto.

2 Que puede hacer el usuario?

Como he explicado anteriormente referidos al usuario podrá comprar y venderá productos, por lo tanto podrá administrar sus productos y comprar productos de otros usuarios en caso de estar registrado , si el usuario no está registrado, solo podrá ver el listado de productos, y buscarlos. El listado será ordenado por localidades de las más próximas a las más alejadas. Tampoco podrá añadirlos a la lista de favoritos o al carrito. El usuario que compra podrá marcar el producto como recibido, cuando le llega y esté en perfectas condiciones. Cuando el producto esté marcado como recibido el dinero se desbloqueará y será enviado al vendedor inmediatamente. El vendedor en caso de que el comprador no marque el producto como recibo podrá hacer una reclamación en la que a dinero quedarían bloqueados hasta que un administrador revise las dos parte y decida quién tiene razón

2 Tecnologies emprades

1 Vue

Vue es un framework open-source el cual es fácil de usar y de manera progresiva. Vue fue diseñado por un ex desarrollador de google en la intención de mejorar las carencias de angular 1.X y actualmente en evolucion adquiriendo lo que le parece más útil y simple. Pues no ha sido un decisión fácil aunque si ha sido fácil descartar angular 2 porque te obliga a usar Typescript al cual no tengo nada en contra pero si puedo eligir entre usarlo o no me quedo con no usarlo, aunque Typescript también se puede usar en vue, no estas obligado a usarlo. Finalmente he elegido Vue por la forma de gestionar los estados ya que vue es un framework y automáticamente te crea los getters y setters del estado para que se pueda cambiar de forma casi automática, también porque muchas de las librerías más utilizadas son oficiales de vue como pueden ser vuex , vue-router ...

Empresas que usan Vue: Xiaomi, Gitlab, Alibaba.

2 Django

He decidido gastar un framework porque simplifica el trabajo, me hace ahorrar mucho tiempo y lineas de codigo, también tiene una amplia oferta de funciones predeterminadas por tanto no tengo que estar buscando en otras librerías o estar reinventando la rueda. He escogido Django porque utiliza un lenguaje de alto nivel, este lenguaje es Python un lenguaje fácil de aprender y limpio en su estructura tambien tiene una gran variedad de librerías. Django está diseñado exclusivamente para agilizar las tareas de la programación web. Tiene un gestor de base de datos (ORM) que te evade de la creación de la estructura de la base de datos, tablas, columnas ... simplemente debes crear el modelo y Django crear la tabla y las columnas en la base de datos correspondiente.

Empresas que usan Django: Instagram.

3 Preparacion entorno desarrollo

1 Django

1 Preparacion entorno virtual de python con venv

Antes de instalar Django, instalaremos una herramienta extremadamente útil que ayudará a mantener tu entorno de desarrollo ordenado en tu computadora. Es posible saltarse este paso, pero es altamente recomendable. Esta herramienta nos permitira instalar diferentes paquetes con diferentes versiones en diferentes entornos virtuales sin que entren en conflicto. Para ello lo primero que tendremos que hace es crear el entorno virtual Si lo tenemos instalado cuando creemos el entorno virtual no nos mostrara ningun mensaje por pantalla y ademas creara las diferentes carpetas.

```
$ python3 -m venv .
$ ls
bin  include  lib  lib64  pyvenv.cfg  share
```

En caso de que no estuviese instalado , el mimso te diria como instalarlo desde tu plataforma

```
The virtual environment was not created successfully because ensurepip is not
available.  On Debian/Ubuntu systems, you need to install the python3-venv
package using the following command.

    apt-get install python3-venv

You may need to use sudo with that command.  After installing the python3-venv
package, recreate your virtual environment.
```

Ahora que hemos creado un entorno virtual, ejecutaremos un script para activarlo. Después de activar el entorno virtual, los paquetes que instalemos solo estarán disponibles en este entorno

virtual. De esta forma, trabajaremos en un entorno aislado en el que todos los paquetes que instalemos no afectarán a nuestro entorno principal de Python ni a otros entornos virtuales.

```
$ source bin/activate
(venv) $
```

Una vez que desactive un entorno virtual, volverá al entorno predeterminado de Python. En macOS o Linux, simplemente escriba deactivate y presione Enter.

```
(venv) $ deactivate
$
```

2 Estructura directorios backend django

```
WhiteMarket/
└── WhiteMarket
    └── apps
        ├── images
        │   └── static
        ├── products
        └── user
```

He elegido esta estructura de directorios porque es la que me parece más lógica. Rompe con la filosofía de django que es hacer cada modulo como si fuera un aplicación, pero yo he elegido poner todos los módulos o apps dentro de la misma aplicación porque realmente no pueden trabajar de forma independientemente, porque esta relacionadas entre ellas, la única que si podría considerarse como una app independiente seria images pero está aquí porque en un futuro estaría relacionada con el usuario y también por tenerlo un poco organizado todo en el mismo sitio.

2 Vue

1 Preparacion entorno vue

Aunque se puede desarrollar una aplicacion Vue sin Node, utilizare Node a lo largo del desarrollo del proyecto para lanzar el servidor de desarrollo , para inicializar el proyecto con vue-cli i para gestionar la dependencias. Para ello primero tendremos que instalar Node

```
$ apt-get update && apt-get upgrade -y
$ apt-get install curl -y
$ curl -sL https://deb.nodesource.com/setup_9.x | bash -
$ apt-get install nodejs
```

Despues solo quedara inicializar el projecto con vue-cli y responder a las preguntas que nos haga como las siguientes.

```
$ vue init webpack whitemarket

? Project name whitemarket
? Project description A Vue.js project
? Author oscll <oscllweb@gmail.com>
```

```
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? No
? Set up unit tests Yes
? Pick a test runner karma
? Setup e2e tests with Nightwatch? Yes
? Should we run `npm install` for you after the project has been created? (recommended) npm
```

2 Estructura archivos frontend

```
WhiteMarket/
├── build
├── config
├── dist
│   └── static
├── src
│   ├── assets
│   │   └── scss
│   ├── components
│   │   ├── auth
│   │   ├── commons
│   │   ├── header
│   │   └── products
│   ├── router
│   ├── store
│   │   └── modules
│   │       ├── auth
│   │       ├── cart
│   │       └── products
│   └── utils
└── static
```

Aquí se puede apreciar uno de los puntos fuertes de Vue que es la casi libertad que permite porque se puede estructurar como más te guste y también te permite separar el html , javascript y css. La estructura que he elegido está inspirada en la estructura que genera vue-cli y en los proyectos que he realizado anteriormente. En la carpeta dist contendrá la aplicación ya compilada lista para ser lanzada en el servidor. Ahora pasare a explicar la carpeta src la cual contiene los archivos que forman la aplicación , en la carpeta assets esta els scss que se usara en toda la aplicación, si el scss solo se usara en un componente vue será incluido en el componente y no en la carpeta assets. He decido no separar el scss porque solo se usara en el componente y no será muy extenso. El javascript y el html tampoco lo he separado porque me resulta más fácil de ver si están en el mismo archivo, ya que en la mayoría de components el html llaman a funciones de javascript. Después la carpeta components la he organizado como si fueran módulos, la cual dentro contiene los módulos(auth, products ...) y dentro de cada modulo están las vistas , las rutas y en la carpeta components contiene los componentes que solo se usan en este modulo en caso contrario estarían en commons. He estructurado la carpeta de components en módulos porque he intentado que sean totalmente independientes y así poder reutilizarlos en otros proyectos. En la carpeta router está el archivo index.js que se encarga de importar las rutas de los módulos comentfuncionados anteriormente

que están en la carpeta components. Después está la carpeta store la cual contiene los módulos vuex. Vuex es una librería que gestiona el estado de forma centralizada y garantiza que el estado solo se puede mutar de manera predecible. Esta carpeta la explicare más adelante en profundidad. Finalmente está la carpeta utils la cual contendrá funciones necesarias en toda la aplicación , en este caso tengo la función de Axios para hacer peticiones a la api django

3 Postman

Postman es una herramienta que nos permite hacer peticiones a una api de forma muy fácil y por esto la usare para comprobar que la api django funcione correctamente antes de implementar la funcionalidad en el cliente.

4 Editor

Se puede usar una gran variedad de editores, como Visual Studio Code, Sublime Text, Atom y WebStorm. Yo recomiendo VisualStudio Code ya que tiene una alta frecuencia de actualizaciones y una gran cantidad de extensiones para Vue y podemos usar para mejorar nuestro ritmo de trabajo.

4 Flujo aplicación

1 Flujo frontend

Para explicar el flujo del frontend explicare todo el recorrido que hace la aplicación desde que se inicia hasta muestra los productos por pantalla. Lo primero que se ejecuta es el archivo main.js, este archivo lo que hace es cargar el componente vue que contendrá la aplicación es decir un componente que tendra el header, footer y el router-view, por ello también tendrá que importar la store("vuex") y las rutas, también puedes ser que importe algunos componentes pero no es obligatorio.

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
/* eslint-disable */
import App from './App'
import router from './router'
import store from './store'
import VueMarkdown from 'vue-markdown'
```

```
import VeeValidate from 'vee-validate';

require('es6-promise').polyfill()

Vue.config.productionTip = false
Vue.config.devtools = true
Vue.config.silent = false
Vue.config.debug = process.env.NODE_ENV !== 'production';

Vue.use(VueMarkdown)
Vue.use(VeeValidate)

/* eslint-disable no-new */
new Vue({
  el: '#app',
  store,
  beforeCreate(){
    this.$store.commit('initialiseStore')
  },
  components: { App },
  template: '<App/>',
  router,
})
```

Luego llega al componente App.vue que he mencionado anteriormente, este componente es el único que es estático porque dentro de este contendrá los componentes header, footer y los demás componentes irán alternando según la ruta en la que este. En el scss importo las librerías de bootstrap, fontawesome ... y finalmente el scss que uso para toda la aplicación, lo importo el último para que sobrescriba las librerías anteriores

```
<template>
  <div id="app">
    <main-header> </main-header>
    <router-view/>
    <main-footer></main-footer>
  </div>
</template>

<script>
/* eslint-disable */
import MainHeader from '@/components/header/Header'
import MainFooter from '@/components/footer/Footer'
export default {
  name: 'App',
  components: {
    MainHeader,
    MainFooter
  },
}
```

```
</script>

<style lang="scss">
  @import './../node_modules/bootstrap/dist/css/bootstrap.css';
  @import url("https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css");
  @import url("https://fonts.googleapis.com/css?family=Gugi");
  @import url("../static/fontawesome/web/css/fontawesome.css");
  @import url("./assets/scss/style.scss");
</style>
```

1 Routes

Los componentes que irán alternándose en router-view dependiendo de la ruta se administran desde el index.js que está en la carpeta routes, el cual tendrá la ruta por defecto e importara las rutas de los módulos.

```
//index.js
export default new Router({
  mode: 'history',
  base: '/',
  routes: [
    {
      path: '/',
      redirect: '/products'
    },
    ...authRoutes,
    ...productsRoutes
  ]
})

// productsRoutes rutas modulo products
export default
[
  {
    path: '/products',
    name: 'products-list',
    component: list,
  },
  {
    path: '/myproducts',
    name: 'myproducts-list',
    component: list,
  },
  {
    path: '/myfavorited',
    name: 'favorited-list',
    component: list,
  },
```

```
  {
    path: '/product/add',
    component: edit,
  },
  {
    path: '/product/edit/:pk',
    component: edit,
  },
  {
    path: '/products/detail/:pk',
    component: detail,
  }
]
```

La librería vue-routes tiene un modo muy peculiar en el cual la aplicación es una single page aplication pero a simple vista se comporta como si no fuera, este modo se llama history.

2 Componentes

Como anteriormente he explicado la estructura de los archivos de frontend e intentado tratar cada carpeta de components como si fuera un modulo la cual contien "vistas", componentes y rutas ya que aqui aparece la anarquia de vue.

```
components/
├── auth                              // modulo auth
│   ├── auth.routes.js                // rutas modulo auth
│   ├── components                    // componentes modulo auth
│   │   └── vue-change-password.vue
│   ├── login.vue                     // vista modulo auth
│   └── register.vue                  // vista modulo auth
├── commons                           // componentes usados mas de un modulo
│   ├── vue-mapleaflet.vue
│   ├── vue-qrcode.vue
│   └── vue-uploader-image.vue
├── footer                            // modulo footer
│   └── Footer.vue                    // vista modulo footer
├── header                            // modulo header
│   ├── components                    // componentes modulo header
│   │   ├── auth.vue
│   │   └── cart.vue
│   └── Header.vue                    // vista modulo header
└── products                          // modulo products
    ├── components                    // componentes modulo products
    │   └── vue-slider-upload.vue
    ├── details.vue                   // vista modulo products
    ├── edit.vue                      // vista modulo products
    ├── list.vue                      // vista modulo products
    └── products.routes.js            // rutas modulo products
```

Un componente de vue puede estar formado por una template que puede ser html o jsx

```
<template>
<div >
  <div class="row">
    <div class="col-sm-2">
      <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" id="customCheck1">
        <label class="custom-control-label" for="customCheck1">Boots</label>
      </div>
    </div>
    <div class="col-sm-10 flexy" v-if="!(products == 0)">
        <div class="card-deck">
            <div class="card mt-5" v-for="product in products"
            v-bind:key="product.pk">
              <div class="contain-img">
                <img class="card-img-top"
                :src="'http://localhost:8000'+product.images[0].thumbnail"
                width="250px" height="250px" alt="Card image cap">
                <span v-if="product.favorited" class="like"
                @click="like(product.pk)"></span>
                <span v-else class="unlike"
                @click="like(product.pk)">{{product.total_likes}}</span>
                <span class="views">{{product.total_views}}</span>
              </div>
              <div class="card-body">
              <h5 class="card-title">{{product.title}}</h5>
              <h5 class="card-title">{{product.price}}$</h5>
              <div v-if="edit" class="btn btn-danger delete-list"
              @click="remove(product.pk)">
              </div>
              <router-link v-if="edit" :to='"/product/edit/"+product.pk' tag="div"
              class="btn btn-primary" >
                Modificar
              </router-link>

              <router-link v-else :to='"/products/detail/"+product.pk' tag="div"
              class="btn btn-primary" >
                Ver mas
              </router-link>
              <p class="card-text"><small
              class="text-muted">{{date(product.created)}}</small></p>
              </div>
            </div>
        </div>
    </div>
    <div class="col-sm-10" v-if="products == 0">
      We couldn't find any repositories matching
    </div>
```

```
  </div>
  <button v-if="next" class="btn btn-primary my-5" @click="nextProducts()">
  NEXT_PAGE </button>
</div>
</template>
```

Como se puede apreciar hay algunos elementos que no concuerdan con el html , esto es porque son de vue

- {{}} Sirven para hacer referencia a un estado o una variable
- : o v-bind es lo mismo y sirve para bindear los datos por ejemplo

  ```
  <div class="card mt-5" v-for="product in products" v-bind:key="product.pk">
  ```

  Lo que hace es un referencia entre el atributo pk del producto y el valor que tiene en key, para cada producto hace un referencia por eso es recomendable hace el bind en las keys , pero si coges el producto exacto y le cambias el atributo pk deberías ver como cambia también en la key.
- v-model la directiva anterior sirve para crear un enlace de datos entre el valor del atributo y el estado pero si tú cambias el valor del atributo , el valor del estado no cambia , para ello se utiliza v-model para crear enlaces entre datos bidireccionales. Altamente útil en los campos de los formularios

  ```
  <input type="password" placeholder="Password" v-model="password">
  ```
- @ o v-on sirven para capturar los eventos y llamar a una función o realizar una pequeña acción

  ```
  <span v-if="product.favorited" class="like" @click="like(product.pk)"></span>
  ```

  Cuando hagan clic sobre el span llamara al una función llamada like que está en methods.Vue permite capturar algunas pulsaciones de teclas y también puedes añadir preventDefault() de formar rápida y corta simplemente añadiéndolo después del evento esto es altamente aconsejable en formularios

  ```
  <form class="form-login" autocomplete="off"@submit.prevent="Submit()">
  ```

Despues de la etiquta template viene la etiqueta script la cual contiene los componentes que importa, los stados del componente, los methods('funciones'), computed ...

```
<script>
import { GET_PRODUCTS, ADD_FAVORITED, GET_PRODUCTS_FAVORITED, GET_MYPRODUCTS,
REMOVE_PRODUCT, NEXT_PAGE } from '@/store/modules/products'
import {mapGetters} from 'vuex'
export default {
  data(){
    return{
      edit:false,
    }
  },
  computed: {
      ...mapGetters([
```

```
        'products',
        'token',
        'next',
      ])
  },
  methods: {
    date(date){
      let diference = new Date() - new Date(date)

      switch (true) {
        case (diference/1000) > 0 && (diference/1000) < 60  :
            return `Hace ${Math.round(diference/1000)} segundos.`
          break;
        case ((diference/1000)/60) > 0 && ((diference/1000)/60) < 60  :
            return `Hace ${Math.round((diference/1000)/60)} minutes.`
          break;
        case (((diference/1000)/60)/60) > 0 && (((diference/1000)/60)/60) < 60  :
            return `Hace ${Math.round(((diference/1000)/60)/60)} hours.`
          break;
        case ((((diference/1000)/60)/60)/24) > 0 &&
        ((((diference/1000)/60)/60)/24) < 7  :
            return `Hace ${Math.round((((diference/1000)/60)/60)/24)} days.`
          break;
        case (((((diference/1000)/60)/60)/24)/7) > 0 &&
        (((((diference/1000)/60)/60)/24)/7) < 13  :
            return `Hace ${Math.round(((((diference/1000)/60)/60)/24)/7)} weeks.`
          break;

        default:
            return `Hace mucho tiempo
            ${date.getFullYear}-${date.getMonth}-${date.getDate}`
          break;
      }

      return date;
    },
    nextProducts(){
      this.$store.dispatch(NEXT_PAGE)
    },
    like(pk){
      this.$store.dispatch(ADD_FAVORITED, pk).then(
          () => {
          },
          (error) => {
          }
      )
    },
    remove(pk){
```

```
      this.$store.dispatch(REMOVE_PRODUCT, pk).then(
          () => {
            toastr.error("Product removed","Product")
          },
          (error) => {
          }
      )
    },
    getProducts(){
      switch (this.$route.name) {
        case 'products-list':
          this.$store.dispatch(GET_PRODUCTS)
          this.edit = false
          break;
        case 'myproducts-list':
          console.log('my products')
          this.$store.dispatch(GET_MYPRODUCTS)
          this.edit = true
          break;
        case 'favorited-list':
          console.log('my favorited list')
          this.$store.dispatch(GET_PRODUCTS_FAVORITED)
          this.edit = false
          break;
        default:
          this.$store.dispatch(GET_PRODUCTS)
          this.edit = false
          break;
      }
    }
  },
  beforeMount: function(){
    console.log(this.$route)
    this.getProducts()
  },
  watch:{
    token: function(){
      this.getProducts()
    },
    $route: function(){
      this.getProducts()
    },
    products: function(){
      console.log(this.products)
    }
  }
};
</script>
```

Voy a pasar a explicar los apartados más importantes

- Data contiene los estados del componente, debe devolver un objeto. En este ejemplo solo tiene el estado de edit que por defecto es false
- Computed suelen ser pequeñas funciones declarativas que devuelven un valor basado en elementos dentro de nuestro modelo de datos.
- Methods contiene funciones que pueden llamarse desde todo el componente, principalmente desde el html de la etiqueta template. PD No confundir el this que genera vue.
- Watch observa las propiedades que le indiques y cuando cambian llama a una función
- BeforeMount es una función que se ejecuta antes de montar el componente

3 Vuex

Como se ha podido observar en el componente, antes de montarlo realiza un dispatch

```
this.$store.dispatch(GET_PRODUCTS)
```

Y también las propiedades calculadas o computed son un poco especial

```
  computed: {
      ...mapGetters([
        'products',
        'token',
        'next',
      ])
  },
```

Esto se debe porque esta relacionados con la liberia vuex para entender esto primero debemos entender como funciona vuex.

Los archivos de vuex se almacenan en una carpeta llamada store

- Actions.js se encarga de administrar las mutations y hacer las peticiones a la api. Se accede a los actions a traves de dispatch()
- Mutations.js se encarga de manipular el estado. Se accede a las mutaciones a traves de commit()
- Getters.js almacena los getters para devolver el estado
- Mutation-types.js tener todas las constantes en un solo archivo permite obtener una vista rápida de las mutaciones posibles en toda la aplicación, lo recomiendan como buena practica.
- Index.js Aquí se define el estado y une los archivos anteriores, también importa los modulos

Mantendre la estructura de modulos

```
store/
├── actions.js
├── getters.js
├── index.js
├── modules
│   ├── auth
│   │   ├── actions.js
│   │   ├── getters.js
│   │   ├── index.js
```

```
│   │   ├── mutations.js
│   │   └── mutation-types.js
│   ├── cart
│   │   ├── actions.js
│   │   ├── getters.js
│   │   ├── index.js
│   │   ├── mutations.js
│   │   └── mutation-types.js
│   └── products
│       ├── actions.js
│       ├── getters.js
│       ├── index.js
│       ├── mutations.js
│       └── mutation-types.js
├── mutations.js
└── mutation-types.js
```

Entonces ahora ya comprendemos que cuando llamaba a 'this.$store.dispatch(GET_PRODUCTS)'

```
  async [types.GET_PRODUCTS]({ commit },data) {
    let order = ''
    let user = JSON.parse(localStorage.getItem('user'))
    if(user){

    }
    API.get(`/products/${order}`)
    .then(response => {
       commit(types.CHANGE_PRODUCTS, response.data)
       this.dispatch(types.GET_CATEGORIES);
    }).catch(err => {
      err.response ? this.dispatch(ERRORS, err.response.data) : ""
      reject(err)
    })
  },
```

Se puede observar que realizar un peticion api la cual si no tiene ningun error llamara a la mutacion de CHANGE_PRODUCTS pasandole como datos la respuesta recivida por la api y tambien llamaria a la action GET_CATEGORIES.

```
  [types.CHANGE_PRODUCTS](state, response) {
    console.log(response)
    state.products = response.results;
    state.next = response.next;
    state.count = response.count;
  },
```

En caso contrario de que hubirea algun error saltaria un dispatch hacia ERRORS que esta en el modulo global , el cual haria un commit ha ERRORS que mostraria el error por pantalla.

```
export default {
  [types.ERRORS](state, errors) {
```

```
    state.errors.push(errors);
    switch (typeof errors) {
      case 'string':
        toastr.error(errors,'Error upload image');
        break;
      case 'object':
        for(let key in errors) {
            if(errors.hasOwnProperty(key)) {
                toastr.error(errors[key],key);
            }
        }
        break;

      default:
        break;
    }
  },
}
```

Finalmente . . . mapGetters hace refencia a la funciones getters.

```
    products(state) {
      return state.products;
    },
```

2 Flujo backend

Para explicar el flujo del backend explicare todo el recorrido que hace la aplicación desde que recibe un petición get para obtener los productos hasta que devuelve todo los productos en un JSON.

1 Models

Antes de poder crear la vista para que devuelva los productos tenemos que tener un modelo el cual serán los campos que forman un producto. Estos campos pueden ser validados como cadenas de texto, números, decimales o también pueden ser relaciones a otros modelos , en el caso de ser relaciones pueden ser de uno a muchos o de muchos a muchos. Primero explicare los más sencillos y algunas de las opciones que uso y luego explicare las relaciones. Para definir el tipo de "variable" hay que revisar la documentación de django y comprobar como llama al tipo de dato. Por ejemplo si queremos una cadena de texto o string es models.CharField, un entero es models.PositiveSmallIntegerField y date es models.DateTimeField. Luego estos campos admiten unas opciones para validar el contenido algunas de las más usadas son :

- Max_length y min_length para limitar en uso de caracteres en una cadena
- Default es el valor que contendrá por defecto
- Editable por defecto es True el cual permite que se cambie
- Blank por defecto es False y si es True permite que el campo este vació en la base de datos
- Db_index es para acelerar las búsquedas en esa columna .
- Max_digits es el numero máximo de dígitos que puede tener un numero

- Validators sirven para validar el valor del campo en este caso para que no sea mayor de 3 ni menor de 0

```
    created = models.DateTimeField(default=datetime.now, editable=False)
    title = models.CharField(db_index=True, max_length=50, blank=False)
    price = models.DecimalField(db_index=True, max_digits=30, decimal_places=2)
    stock = models.PositiveIntegerField(db_index=True, default=1)
    description = models.CharField(db_index=True, max_length=10000, blank=False)
    state = models.PositiveSmallIntegerField(db_index=True, default=0,
    validators=[MaxValueValidator(3), MinValueValidator(0)])
```

Un claro caso de muchos a muchos son los users_like que son los usuarios que le han dado me gusta al producto , porque muchos usuarios le pueden gustar mucho productos y un producto puede tener muchos usuarios. Aquí se puede comprobar como se crea la relación con el modelo User

```
    users_like = models.ManyToManyField(
        User,
        related_name='products_like',
        blank=True
    )
```

Otro caso claro par ver la relaciones de uno a muchos es el campo owner , aquí está el usuario que ha creado el producto. Es fácil de ver porque un usuario puede crear muchos productos pero un producto solo puede haber sido creado por un usuario. Aquí se puede comprobar como se crea la relación con el modelo User

```
    owner = models.ForeignKey(
        User,
        related_name='products',
        on_delete='models.CASCADE',
        editable=False
    )
```

Se puede observar que a diferencia de la relacion muchos a muchos contiene el campo on_delete , este campo permite elegir que accion hacer cuando se borre el usuario que esta asociado en esta relacion. Este ejemplo cuando se borre el usuario se borrara tambien el producto porque esta CASCADE.
Ahora que se entiende mejor , se puede ver el modelo de productos el cual está formado por los siguientes campos.

```
class Product(models.Model):
    created = models.DateTimeField(default=datetime.now, editable=False)
    title = models.CharField(db_index=True, max_length=50, blank=False)
    price = models.DecimalField(db_index=True, max_digits=30, decimal_places=2)
    stock = models.PositiveIntegerField(db_index=True, default=1)
    description = models.CharField(db_index=True, max_length=10000, blank=False)
    state = models.PositiveSmallIntegerField(db_index=True, default=0,
    validators=[MaxValueValidator(3), MinValueValidator(0)])
    images = models.ManyToManyField(
        Image,
        related_name='images_products',
    )
```

```
    category = models.ForeignKey(
        ProductCategory,
        related_name='products',
        on_delete='models.CASCADE',
    )
    owner = models.ForeignKey(
        User,
        related_name='products',
        on_delete='models.CASCADE',
        editable=False
    )
    users_like = models.ManyToManyField(
        User,
        related_name='products_like',
        blank=True
    )
    total_likes = models.PositiveIntegerField(db_index=True, default=0)
    total_views = models.PositiveIntegerField(db_index=True, default=0, editable=False)
```

2 Serializers

Después de tener el modelo tenemos que tener el Serializer que es el intermediario entre los datos que recibe de la base de datos o de una petición y convertirlos en un objeto para poder manipularlos desde python. Los campos que no estén relacionado y queramos que se muestren en el producto que devolvemos bastara con simplemente declararlos en fields

```
class ProductSerializer(serializers.ModelSerializer):
    category = serializers.SlugRelatedField(queryset=ProductCategory.objects.all()
    ,slug_field='name')
    owner = UserRelatedField(many=False)
    favorited = serializers.SerializerMethodField()
    images = ImageRelatedField(many=True)

    class Meta:
        model = Product
        fields = (
            'pk',
            'created',
            'title',
            'description',
            'images',
            'price',
            'state',
            'stock',
            'category',
            'owner',
            'total_likes',
            'total_views',
```

```
            'favorited',
        )

       def get_favorited(self, instance):
        request = self.context.get('request', None)
        if request is None:
            return False
        if not request.user.is_authenticated:
            return False
        if request.user in instance.users_like.all():
            return True
        return False
```

Como se puede comprobar aparte de los campos declarados hay variables declaradas al principio y una función, esto se debe porque son los campos relacionados que explique anterior mente en los modelos.
El campo category utiliza un serializer que lo proporciona el django-rest-framework, SlugRelatedField se usa para representar un campo del modelo de la relación y queryset es para saber donde tiene que buscarlo en esta caso en todos los objectos ProductCategory.

```
category = serializers.SlugRelatedField(queryset=ProductCategory.objects.all()
    ,slug_field='name')
```

El campo owner y images son muy parecidos lo único que cambia es el tipo de modelo. Estos campos utilizan una función que es importada desde relations.py , utilizan esta función porque necesito obtener más de un parámetro del modelo relacionado en el caso del owner necesito el email, latitude y longitude. También se podría utilizar un serializer que proporciona django-rest-framework pero este no permite obtener solo algunos campos y también a la hora crear es más difícil, en cambio en la función de relations solo necesito un serializer y para crear el producto basta con pasarle id o pk del modelo.

```
owner = UserRelatedField(many=False)

 relations.py
class UserRelatedField(serializers.RelatedField):
    def get_queryset(self):
        return User.objects.all()

    def to_internal_value(self, data):
        user = get_object_or_404(Image, id=data)
        return user

    def to_representation(self, value):
        return UserSerializer(value).data
```

Aparte de la variable, debe estar declarada en fields y la opción que le pasa en many indica si debe recorrerlo sirve para cuando es una relacion de mucho a mucho como en las images , en caso contrario como en owner está en False porque solo tiene un usuario.

Finalmente el campo favorited que seria el campo users_like que contendría los usuarios que le han dado me gusta. Aquí no interesaría utilizar una relación porque no quiero que devuelva

todos los usuarios que le han dado me gusta sino que quiero que devuelva True si el usuario que pide el producto le ha dado me gusta, sino que devuelva Fasle. Para ello utilizo un serializer de django-rest-framework que busca un función que se llame get_favorited , busca get_favorited porque en este caso se llama favorited. Esta función comprueba que este loggeado y que le haya dado me gusta.

```
    favorited = serializers.SerializerMethodField()

       def get_favorited(self, instance):
        request = self.context.get('request', None)
        if request is None:
            return False # Si no esta request no puede obtener si esta loggeado
        if not request.user.is_authenticated:
            return False # Si no esta loggeado
        if request.user in instance.users_like.all():
            return True # Si el usuario esta en el campo users_like()
        return False
```

3 Views

Una de las principales diferencias entre las vistas genéricas de DRF y las vistas genéricas de Django es cómo combinan múltiples operaciones en una única clase de vista. Por ejemplo, DRF ofrece la ListCreateAPIView clase pero Django solo ofrece ListView clase y CreateView clase. DRF ofrece una ListCreateAPIView clase porque espera una petición POST para crear un modelo y con una petición GET devuelve una lista de todos los objectos del modelo.

Al principio la vista ProductList fue como se puede apreciar en el código de abajo es bastante simple y devuelve todas las instancias de product y crea una instancia de product

```
class ProductList(generics.ListCreateAPIView):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    name = 'product-list'
```

Después me di cuenta que quería que solo los usuarios registrados pudieran crear productos. Para ello importe IsAuthenticatedOrReadOnly del django-rest-framework y añadí el siguiente código

```
    permission_classes = (
        permissions.IsAuthenticatedOrReadOnly,
    )
```

Ahora se pueden crear productos por usuarios loggeados pero no sirve de nada porque el usuario puede poner en el campo owner el usuario que quiera , para arreglar esto lo que hice fue que al crear el producto el campo owner fuera el usuario loggeado ignorando los que envié el usuario en el campo owner.

```
    def perform_create(self, serializer):
        serializer.save(owner=self.request.user, total_views=0)
```

Hasta aquí todo perfecto pero que pasa si quieres buscar, ordenar o filtrar por campos ?
Pues aquí es donde entra la librería django_filters que permite filtrar, ordenar y buscar en los

diferentes campos del modelo que le indiques

```
    filter_fields = (
        'title',
        'category',
        'owner',
        'state',
        )
    search_fields = (
        'title',
        'description',
        )
    ordering_fields = (
        'created',
        'price',
        'total_views',
        'total_likes',
        )
```

Aunque la librería django_filters te permite filtrar por campos no puede ordenar por latitud y longitud para solucionar esto lo tuve que hacer manualmente, si recibe una petición con los campos latitude, longitude y distancia hace una consulta a la base de datos que le devuelve el id de los productos que están en la zona y ordenados por cercanía. Luego devuelve todas las instancias filtrando por los ids recibidos de la base de datos.

```
def get_queryset(self):
        request = self.request
        if request.query_params.__contains__('latitude') &
        request.query_params.__contains__('longitude') &
        request.query_params.__contains__('distance'):
            latitude = request.query_params['latitude']
            longitude = request.query_params['longitude']
            radius = request.query_params['distance']
            radius = float(radius) / 1000.0

            query = """SELECT p.id (6367*acos(cos(radians(%2f))
            *cos(radians(latitude))*cos(radians(longitude)-radians(%2f))
            +sin(radians(%2f))*sin(radians(latitude)))) AS distance FROM
            products_product as p INNER JOIN user_user as u ON u.id = p.owner_id
            WHERE (6367*acos(cos(radians(%2f))
            *cos(radians(latitude))*cos(radians(longitude)-radians(%2f))
            +sin(radians(%2f))*sin(radians(latitude)))) < %2f ORDER BY distance
            """ % (
                float(latitude),
                float(longitude),
                float(latitude),
                float(latitude),
                float(longitude),
                float(latitude),
                radius
```

```
            )
            ids = [p.id for p in Product.objects.raw(query)]
            return Product.objects.filter(id__in=ids)
        else:
            return Product.objects.all()
```

Finalmente después de implementar todas las características la vista ProductList quedaría así

```
class ProductList(generics.ListCreateAPIView):

    def get_queryset(self):
        request = self.request
        if request.query_params.__contains__('latitude') &
        request.query_params.__contains__('longitude') &
        request.query_params.__contains__('distance'):
            latitude = request.query_params['latitude']
            longitude = request.query_params['longitude']
            radius = request.query_params['distance']
            radius = float(radius) / 1000.0

            query = """SELECT p.id (6367*acos(cos(radians(%2f))
            *cos(radians(latitude))*cos(radians(longitude)-radians(%2f))
            +sin(radians(%2f))*sin(radians(latitude)))) AS distance FROM
            products_product as p INNER JOIN user_user as u ON u.id = p.owner_id
            WHERE (6367*acos(cos(radians(%2f))
            *cos(radians(latitude))*cos(radians(longitude)-radians(%2f))
            +sin(radians(%2f))*sin(radians(latitude)))) < %2f ORDER BY distance
            """ % (
                float(latitude),
                float(longitude),
                float(latitude),
                float(latitude),
                float(longitude),
                float(latitude),
                radius
            )
            ids = [p.id for p in Product.objects.raw(query)]
            return Product.objects.filter(id__in=ids)
        else:
            return Product.objects.all()
    serializer_class = ProductSerializer
    name = 'product-list'
    filter_fields = (
        'title',
        'category',
        'owner',
        'state',
        )
    search_fields = (
```

```
        'title',
        'description',
        )
    ordering_fields = (
        'created',
        'price',
        'total_views',
        'total_likes',
        )
    permission_classes = (
        permissions.IsAuthenticatedOrReadOnly,
    )

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user, total_views=0)
```

4 Urls

Finalmente solo faltar vincular la vista con un ruta para que pueda acceder desde la api , ello se hace desde el archivo urls.py.

```
urlpatterns = [
    path('products/', views.ProductList.as_view(), name=views.ProductList.name),

    path('products/<int:pk>', views.ProductDetail.as_view(), name=views.ProductDetail.name),

]
```

Aunque teóricamente ya estaría todo debes comprobar que el archivo urls del modulo esta importado en el archivo urls de toda la aplicación y tambien que el modulo o "app" esta importada en el archivo settings.py

```
urls.py
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('WhiteMarket.apps.products.urls')),
    path('api/', include('WhiteMarket.apps.images.urls')),
]
```

```
settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
    'corsheaders',
    'rest_framework',
    'rest_framework.authtoken',
    'djoser',

    'WhiteMarket.apps.products.apps.ProductsConfig',
]
```

```
├── Dockerfile
├── manage.py
├── postgresql
├── requirements.txt
└── WhiteMarket
    ├── apps
    │   ├── images
    │   ├── products
    │   │   ├── admin.py
    │   │   ├── apps.py
    │   │   ├── models.py
    │   │   ├── relations.py
    │   │   ├── serializers.py
    │   │   ├── signals.py
    │   │   ├── tests.py
    │   │   ├── urls.py # <-- urls modulo
    │   │   └── views.py
    │   └── user
    ├── custompermission.py
    ├── __init__.py
    ├── settings.py # <-- settings applicacion importar ruta modulo
    ├── urls.py # <-- urls applicacion importar urls modulo
    └── wsgi.py
```

5 Resumen flujo para obtener los productos

Finalmente para terminar haré un pequeño resumen para que quede claro el flujo del backend desde que se realiza un petición hasta que devuelve los datos.

1. Urls de toda la aplicación recibe un petición a '/api/products/' redirecciona a urls del modulo
2. Urls del modulo lo redirecciona a la vista ProductsList
3. La vista aplica los filtros pertinentes y obtiene las instancias(modelo)
4. La vista serializa las instancias y las devuelve como un json

5 Problemas encotrados y mejoras tecnicas

1 Vuex carrito

Este problema aparece cuando asignamos un Map de js en el estado de vuex, el problema en si consiste en que no detecta cuando cambian los valores y por tanto no se actualiza automanticamte en el componente cart del modulo header. Para solucionarlo he tenido que guardar el Map como un array en el estado.

```
// mutations.js
export default {
  [types.ADD_ITEM_CART](state, response) {
    console.log('mutation')
    let cart = new Map(state.cart)
    cart.set(response.pk,response)
    state.cart = [...cart]
    localStorage.setItem('cart',JSON.stringify(state.cart))
  },
  [types.REMOVE_ITEM_CART](state, pk) {
    let cart = new Map(state.cart)
    cart.delete(pk)
    state.cart = [...cart]
    localStorage.setItem('cart',JSON.stringify(state.cart))
  },
};



// index.js
    cart: JSON.parse(localStorage.getItem('cart')) || [],
```

2 Buenas practicas con Axios

Primero declaro Axios como una constante que se puede exportar y esta centralizada en un archivo.

```
import axios from 'axios'

export const API = axios.create({
  baseURL: `http://localhost:8000/api/`,
  Accept: 'application/json',
})
```

Y también intento no hacer las peticiones con axios desde los componentes porque incrementa el numero de lineas de código y dificulta la legibilidad del mismo. Finalmente están casi todas concentradas en los actions por si he de realizar un petición desde otro componente , no estar duplicando el código.

3 Signals

Sirve para que cada vez que un usuario le de a me gusta o no me gusta cuente los usuario que tiene en users_likes y lo ponga en total_likes

```
from django.db.models.signals import m2m_changed
from django.dispatch import receiver
from WhiteMarket.apps.products.models import Product

@receiver(m2m_changed, sender=Product.users_like.through)
def users_like_changed(sender, instance, **kwargs):
    instance.total_likes = instance.users_like.count()
    instance.save()
```

4 Control login rutas

El problema que tenia era que cuando el usuario estaba registrado aun podia seguir accediendo a las vistas de login y registro, para solucionarlo cree las siguientes funcions que compruebas si esta loggeado o no

```
const isLogged = function(to,from,next) {
  if(localStorage.getItem('token')){
    next('/')
    return
  }
  next()
}
const notLogged = function(to,from,next) {
  if(!localStorage.getItem('token')){
    next('/')
    return
  }
  next()
}

export default
  [
    {
      path: '/login',
      component: login,
      beforeEnter: isLogged,
    },
    {
      path: '/register',
      component: register,
      beforeEnter: isLogged,
    },
    {
      path: '/account',
```

```
      component: register,
      beforeEnter: notLogged,
    }
  ]
```

5 Creación componente map

Cree un componente que se reutiliza dependiendo de los parámetros que le pases, en login los uso para que le usuario marque donde está ubicado, el cual geolocaliza y con cada clic lanza un envento con la posicion y mueve el icon. Y en la vista del producto solo muestra la ubicación del producto y no hace caso a los clics

```
<template>
  <div id="map">
  </div>
</template>

<script>
export default {
    props: {
        latitude: {
            type: Number,
            default: 38.821103
        },
        longitude: {
            type: Number,
            default: -0.609543
        },
        clickLatLong: {
            type: String,
            default: 'false',
        },
        items: {
            type: Object,
        },
        localizame:{
            type: String,
            default: 'true'
        }
    },
    watch:{
        latitude(){
            this.markerLatLong({latlng:
            {lat:this.latitude,lng:this.longitude},located: true})
        },
        longitude(){
            this.markerLatLong({latlng:
            {lat:this.latitude,lng:this.longitude},located: true})
```

```
        },
    },
    methods:{
        markerLatLong(e) {
            if(this.marker){
               // if(this.clickLatLong != 'false') {
                    if(!e.located)
                        this.$emit('location', e.latlng)
                    this.marker.setLatLng(e.latlng)
                    this.map.setView(e.latlng)
                //}
            }
        },

        mountmap(){
            this.map = L.map('map').setView([this.latitude, this.longitude], 13);
            L.tileLayer('http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
                maxZoom: 18,
                attribution: 'Map data &copy; <a
                href="http://openstreetmap.org">OpenStreetMap</a>',
                id: 'mapbox.streets'
            }).addTo(this.map);
            // Icon options
            let iconOptions = {
                iconUrl: '/static/logo.png',
                iconSize: [25, 25]
            }
            let customIcon = L.icon(iconOptions);
            this.marker = new L.marker({lat: this.latitude, lng: this.longitude},
            {title:'located', clickable: true, icon: customIcon,
            draggable:true});

            this.marker.addTo(this.map)
            if(this.clickLatLong != 'false'){
                this.map.on('click', this.markerLatLong);
            }
        }
    },

    mounted() {
        this.mountmap()
        if(this.localizame != 'false'){
            if(navigator.geolocation){
                navigator.geolocation.getCurrentPosition((data)=>{
                    this.$emit('location', {lat: data.coords.latitude, lng:
                    data.coords.longitude})
                });
            }else{
```

```
                console.log('No located')
            }
        }
    },
}
</script>
```

6 Slider para subir imagenes

Se trata de un slider que puede contener componentes el cual modifique para que puediera emitir los eventos que recive el componente y tambien para que cuando le para un array con imagenes las pusiera cada una en un componente y totalmente funcional.

```
  mounted () {
    let that = this
    setTimeout(function () {
      if(that.images){
        that.someList = []
        let images = JSON.parse(that.images)
        images.forEach((image, index) => {
          if(index == 0){
            that.someList.push(
              {
                component:{
                  components:{
                    uploader
                  },
                  template: `<uploader v-on="$listeners" name="img${that.sliderinit.currentPage}"
                }
              }
            )
          }else{
            that.sliderinit.currentPage++
            that.someList.push(
              {
                component:{
                  components:{
                    uploader
                  },
                  template: `<uploader v-on="$listeners" name="img${that.sliderinit.currentPage}"
                }
              }
            )
          }
        });
        that.sliderinit.currentPage++
        that.someList.push(
          {
```

```
            component:{
              components:{
                uploader
              },
              template: `<uploader v-on="$listeners" name="img${that.sliderinit.currentPage}"> </u
            }
          }
        )
      }else{
        that.someList = [
          {
            component:{
              components:{
                uploader
              },
              template: `<uploader v-on="$listeners" name="img${that.sliderinit.currentPage}"> </u
            }
          },
        ]
      }
    }, 2000)
  },
```

7 Contador de visitas

Se trata de un problema que no sabia como hacer que cada vez realizaran una peticion a un productos este fuera incrementado las visitas, finalmente la solucione de la suigiente manera

```
    def get(self, request, pk):
        pk = self.kwargs.get('pk', None)
        obj = get_object_or_404(Product, pk=pk)
        obj.total_views += 1
        obj.save()
        return Response(ProductSerializer(obj, context={'request':
        self.request}).data)
```

8 Custom permission

Cree un permisson para que solo se pueda modificar o eliminar la instancia si eres el creador , en caso contrario devuelve un error de autenticacion.

```
from rest_framework import permissions

class IsCurrentUserOwnerOrReadOnly(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            # The method is a safe method
```

```
            return True
        else:
            # The method isn't a safe method
            # Only owners are granted permissions for unsafe methods
            return obj.owner == request.user
```

6 Conclusión

Creo que la experiencia ha sido buena, porque he aprendido bastante sobre vue y django-rest-framework, pero creo que al final me ha faltado un poco de tiempo para poder terminar el proyecto. Bueno si volvería a hacer el proyecto no elegiría unas tecnologías que no conozco porque se tiene el tiempo justo para hacer el proyecto, como para investigar sobre unas tecnologías que no conoces.