

# PYTHON

PROYECTO FINAL

Oriol Marco Sánchez



**Oriol Marco Sanchez**

RESPONSABLE DEL DESARROLLO DE LA APLICACIÓN

**Fecha:** 24 de Agosto de 2020



## 1- Contexto proyecto

La empresa de suministros informáticos TECLA2 (ficticia) nos solicita, como desarrolladores, una aplicación web para la gestión de las ventas y compras de la compañía.

El cliente solicita que la aplicación pueda cumplir diferentes aspectos de su funcionamiento habitual, incluyendo los siguientes módulos:

- 1- Gestión de clientes.
- 2- Gestión de proveedores.
- 3- Gestión de productos.
- 4- Gestión de categorías de productos.
- 5- Gestión de ventas.
- 6- Gestión de compras.
- 7- Reportes de ventas y compras.
- 8- Gestión de permisos de usuarios.

## 2- Objetivo principal proyecto

El objetivo principal de este proyecto es facilitar al cliente una herramienta de gestión empresarial, más conocida como ERP (Enterprise Resource Planning). Para ello el proyecto a desarrollar gira entorno a la idea de una aplicación web, ya que actualmente no poseen y así poder automatizar e informatizar tareas que actualmente se realizan de forma manual.

Este portal Web debe estar orientado sobre todo a la gestión interna de las tareas diarias de la empresa (compras y ventas), sin olvidarnos de poder facilitar información a personas externas a la empresa, como pueden ser los clientes y/o proveedores, ofreciéndoles un buen diseño, que lo haga atractivo y sencillo.

Con todo esto pretendemos hacer que sea fácil de usar, que nos proporcione información importante, veraz y actualizada.

La aplicación contendrá diferentes páginas para mostrar toda la información necesaria, según los módulos descritos en el apartado 1.



### 3- Enfoque y método de trabajo

Con el objetivo de poder cumplir los requisitos presentados por el cliente y satisfacer sus necesidades de usabilidad se determina un enfoque de diseño de la aplicación siguiendo las siguientes características:

- 1- Se determina el desarrollo del proyecto para realizar el mismo en un entorno de producción de una aplicación web, mediante el Framework Django.
- 2- Se determina como lenguaje principal de programación el lenguaje Python, ya que se utilizará como framework, Django. En este proyecto se han realizado partes del mismo en Javascript para poder efectuar los formularios y otras partes del proyecto de cara a definir la parte Frontend.
- 3- Durante el desarrollo del proyecto se realiza un análisis estático del código con el objetivo de ir verificando que éste cumpla las reglas descritas para su uso.
- 4- Se realizan tests del código para poder supervisar el correcto funcionamiento de las diferentes funcionalidades del proyecto y evitar errores y bugs si se deseara añadir nuevas funcionalidades al mismo.
- 5- A la hora de codificar el proyecto se seguirán las normas descritas en la guía de producción de Python PEP-8. Estas guías están controladas, de forma automática por el editor de código escogido para llevar a cabo el proyecto. En esta ocasión, debido a su versatilidad y facilidad de uso se ha escogido el IDE de Python Pycharm Professional.

### 4- Entorno de desarrollo

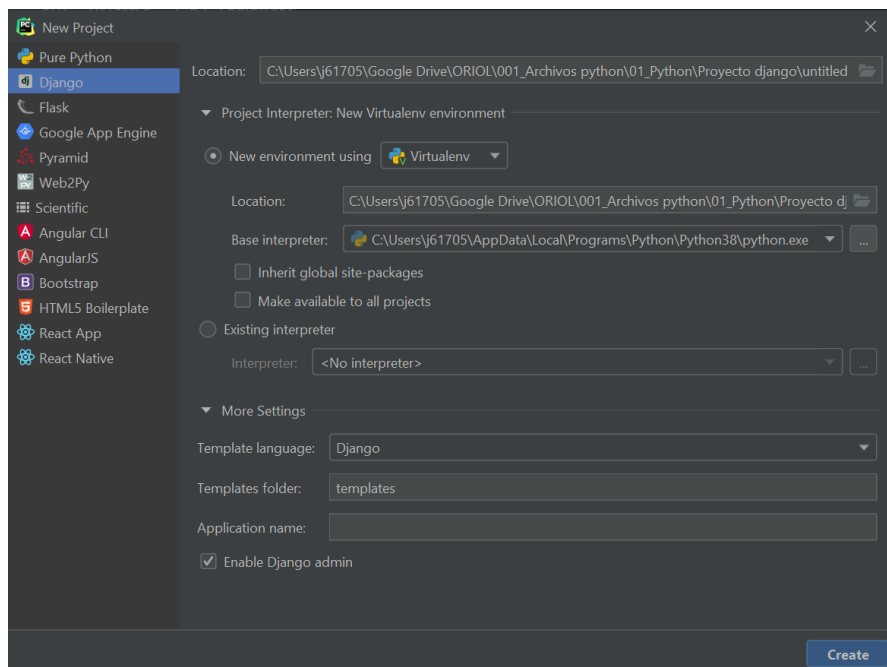
Con el objetivo de llevar a cabo el proyecto siguiendo las directrices generales marcadas en el apartado anterior, a continuación se definen, en profundidad, las características del entorno de desarrollo escogido para realizar la aplicación web.

Para llevar a cabo la creación de la aplicación web siguiendo una organización del código bien estructurado y conseguir que la aplicación sea 100% funcional y escalable para posibles futuras modificaciones, se decide realizar el proyecto mediante el framework DJANGO, escrito principalmente en lenguaje de programación PYTHON 3.

Django es un framework escrito en Python y se trata de un proyecto open source cuyo origen se remonta a 2005. En 2008 se crea la Django Software Foundation que se encarga del desarrollo y mejora continua del proyecto, ampliando funcionalidades e implementando mejoras en la seguridad.

El framework a día de hoy ha evolucionado mucho y es un marco de trabajo que proporciona una alta capacidad de producción de software, potenciando la reutilización, haciendo fácil y uniforme la conexión a diferentes motores de bases de datos con su ORM (Object Relationship Management).

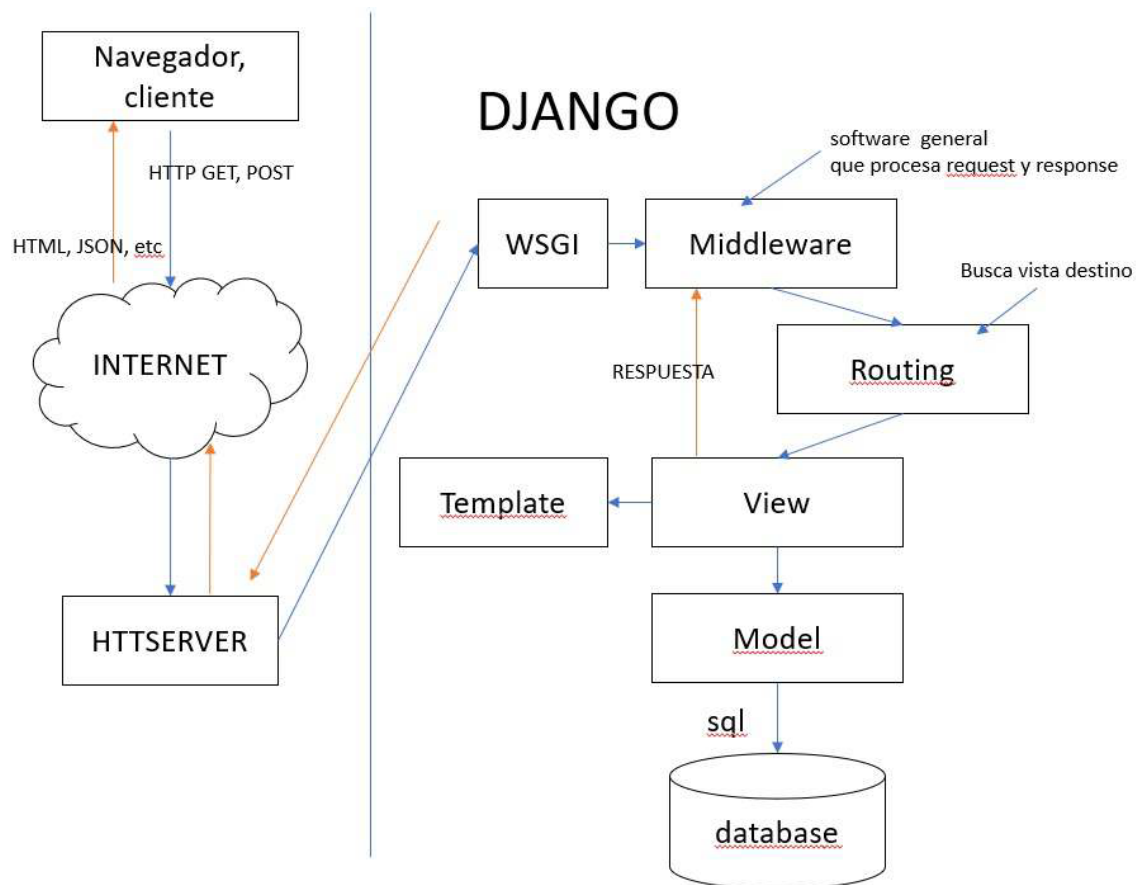
Para llevar a cabo el proyecto de forma que cualquier desarrollador pueda realizar futuras ampliaciones y/o modificaciones del proyecto, éste se lleva a cabo dentro de un entorno virtual (Virtualenv). Para la creación de un entorno virtual donde se determinará la instalación de la versión de Python, Django y todas las librerías utilizadas para este proyecto, se efectúa mediante la creación directa del proyecto Django en Pycharm Professional, tal y como se indica a continuación:



Con la creación del entorno virtual conseguimos que cualquier desarrollador pueda disponer de las versiones utilizadas para proceder con el proyecto y no deba instalar en su pc todas las librerías y versiones diferentes de Python y Django en cada uno de los proyectos que desarrolle.

## 5- Estructura y desarrollo de la aplicación Django

En la figura siguiente se define la estructura básica de una aplicación web Django:

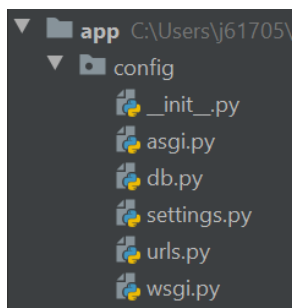


<b>Navegador cliente</b>	Es el agente que solicita datos al servidor donde en la mayoría de los casos será un navegador. Se comunica con el servidor mediante el protocolo http.
<b>Internet</b>	Red de comunicación que permite alcanzar el servidor. Será una red TCP/IP
<b>Http Server</b>	Es el servidor que escucha las peticiones
<b>WSGI</b>	Modelo de interacción entre servidores web y aplicaciones
<b>Middleware</b>	Software Python dentro de Django, definido en settings y que permite alterar las request y/o response

<b>Routing</b>	Módulo de Django que usa las definiciones en el fichero urls.py para seleccionar la vista
<b>View</b>	Código Python que recibe la request y la procesa y usando las templates genera la respuesta
<b>Model</b>	La vista usa las clases que representan el modelo de datos. Estas clases del modelo accederán a la base de datos mediante SQL
<b>Template</b>	En conjunción con la vista describe la salida HTML. Usará datos proporcionados por la vista con origen de la base de datos.

## 5.1- Creación aplicación web Django

El primer paso para iniciar el proyecto después de la creación del entorno virtual, es la creación del proyecto en sí. En este caso, tal y como se ha indicado con anterioridad, éste se ha creado de forma automática mediante el IDE Pycharm Profesional. Después de realizar la creación, se obtienen los siguientes ficheros del proyecto:



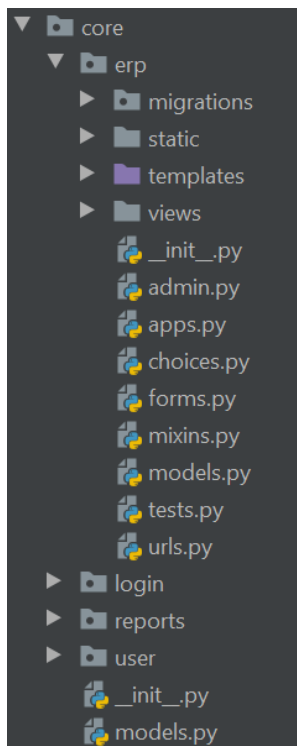
- `__init__.py` = Fichero que indica que el directorio es un paquete de Django
- `asgi.py` = Standard emergente de Python para aplicaciones y servidores web asíncronos.
- `db.py` = archivo de configuración de la conexión con las bases de datos.
- `Settings.py` = fichero de configuración de parámetros generales como las aplicaciones instaladas.
- `Urls.py` = fichero de gestión de urls's.
- `Wsgi.py` = fichero de configuración del protocolo del servidor wsgi.



Un proyecto Django contiene de diferentes aplicaciones. Para crear una nueva aplicación dentro de nuestro proyecto se deberá realizar mediante el siguiente script:

```
python manage.py startapp core
```

Este script crea nuestra aplicación principal dentro de nuestro proyecto donde irán almacenadas las otras aplicaciones con la siguiente estructura:



**erp = aplicación principal del proyecto.**

**login = aplicación de control de la función login.**

**reports = aplicación de gestión de los reportes.**

**User = aplicación de gestión de usuarios.**

- Todos las aplicaciones están formadas por la misma estructura:

- migrations = gestión de las migraciones de la base de datos.
- static = directorio ficheros estáticos.
- templates = directorio ficheros de los templates html.
- views = directorio conjunto de todas las vistas.

- admin.py = registro de los modelos.
- apps.py = configuración de las app's.
- choices.py = fichero de elección de parámetro según modelo "client".
- forms.py = fichero de registro de los formularios.
- mixins = fichero gestión permisos y validaciones.
- models.py = registro de todos los modelos relacionales.
- test.py = fichero de de gestión de los tests.
- urls.py = fichero registro urls de la aplicación.





Tenemos que añadir todas las aplicaciones que vayamos creando en el archivo de settings.py:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    # Librerías  
    'widget_tweaks',  
    # Apps  
    'core.erp',  
    'core.login',  
    'core.user',  
    'core.reports',  
]
```

Una vez que hemos terminado la configuración del proyecto, debemos lanzar la aplicación al servidor para comprobar su correcto funcionamiento. Esta acción la llevamos a cabo con el siguiente script:

```
python manage.py runserver
```

## 5.2- Modelos

Es una de las partes más importantes del framework de Django. El ORM de Django nos permite, a través de código Python, definir las tablas, relaciones e índices. Una vez definidas las clases, éstas son transformadas en código SQL que permite la creación de las tablas y relaciones de la base de datos.

El ORM permite la modificación del modelo, pudiendo añadir, modificar o eliminar campos. Estas definiciones, realizadas en código Python en el framework producirán los cambios en el modelo de la base de datos escogida. También pueden crearse o modificarse relaciones del modelo.

A la operación de sincronizar el modelo en Python con el modelo físico de la base de datos se denomina migración.

Hay que destacar que para el desarrollador será mas fácil mantener el código Python que el modelo de base de datos con SQL.

El sistema de migración sincroniza el modelo de Django con las tablas en SQL mediante la siguiente orden:

```
python manage.py makemigrations
```

Se traducirá el código de las clases del modelo en sentencias SQL. Cabe destacar que el mecanismo de migración añade un id a la tabla declarado como Primary Key.

Lo que en realizada hace el comando “makemigrations” es crear un archivo dentro de migrations que contiene las clase Python con los modelos. Para que este modelo se cree en la base de datos deberemos aplicar el siguiente comando:

```
python manage.py migrate
```

El comando crea en la base de datos la tabla correspondiente. Pero, ¿En que base de datos ha grabado la tabla? En el fichero **db.py** tenemos la siguiente definición:

```
SQLITE = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# psycopg2

POSTGRESQL = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'db',
        'USER': 'postgres',
        'PASSWORD': '123',
        'HOST': 'localhost',
        'PORT': '5432',
        'OPTIONS': {
            'options': '-c search_path=prueba'
        },
    }
}
```

En ese caso se han configurado dos opciones diferentes para poder gestionar la base de datos, una mediante SQLITE, o bien POSTGRESQL. De todas formas, para poder trabajar, en el fichero **settings.py** se deberá indicar con cual de los gestores se quiere trabajar. En mi caso, finalmente decidí trabajar con SQLITE:

```
DATABASES = db.SQLITE
```



BASE\_DIR representa el directorio del proyecto y está definida en el mismo fichero settings.py:

```
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
```

Con lo comentado anteriormente, se procede con la creación de los diferentes modelos y sus relaciones, creando el modelo relacional.

Para llevar a cabo el proyecto, se crea una aplicación Django principal denominada **Core**. En dicha aplicación se realiza la creación de las clases principales de nuestro ERP. Las diferentes clases del modelo son las siguientes, indicando su relaciones principales con las Primary Keys y Foreign Keys:

- PROVIDER = hace referencia a la clase proveedor.

  - PK = Name

- CATEGORY = hace referencia a la clase categoría y poder clasificar los productos.

  - PK = Name

- PRODUCT = hace referencia a los productos de nuestra tienda.

  - PK = Name

  - FK = Cat (CATEGORY)

- PURCHASE = hace referencia a las compras a proveedores.

  - PK = ID Purchase

  - FK = Prov (PROVIDER)

- INVENTORY = hace referencia al control de los productos y sus compras.

  - PK = ID Inventory

  - FK = Purch (PURCHASE)

  - FK = Prod (PRODUCT)

- CLIENT = hace referencia a los clientes y sus compras.

  - PK = Names

- SALE = hace referencia a las ventas realizadas a los clientes.

  - PK = ID Sale

  - FK = Cli (CLIENT)

- DETSALE = hace referencia al detalle de cada una de las ventas.

PK = ID Det Sale

FK = Sale (SALE)

FK = Prod (PRODUCT)

A continuación queda detallado el ejemplo del modelo de CLIENT donde se realiza la creación de la tabla de nuestros clientes:

```
class Client(models.Model):
    names = models.CharField(max_length=150, verbose_name='Nombres')
    surnames = models.CharField(max_length=150, verbose_name='Apellidos')
    dni = models.CharField(max_length=10, unique=True, verbose_name='Dni')
    date_birthday = models.DateField(default=datetime.now, verbose_name='Fecha de nacimiento')
    address = models.CharField(max_length=150, null=True, blank=True, verbose_name='Dirección')
    gender = models.CharField(max_length=10, choices=gender_choices, default='male', verbose_name='Sexo')

    def __str__(self):
        return self.names

    def toJSON(self):
        item = model_to_dict(self)
        item['gender'] = {'id': self.gender, 'name': self.get_gender_display()}
        item['date_birthday'] = self.date_birthday.strftime('%Y-%m-%d')
        return item

    class Meta:
        verbose_name = 'Cliente'
        verbose_name_plural = 'Clientes'
        ordering = ['id']
```

En la definición del modelo se pueden observar diferentes métodos y clases hijas, dando características al propio modelo:

- Método `__str__` : es un método mágico de Python que devuelve una representación de cadena de cualquier objeto, en este caso `self`, retornando los nombres de los clientes.
- `def toJson`: es una función creada para poder dar formato a nuestro listado de clientes según nuestras claves.
- `def META`: es el método que hace referencia a los metadatos de la tabla de la base de datos y en este caso hemos indicado el nombre de la tabla, tanto en singular, como en plural. Por otra parte hemos indicado la columna por la cual queremos que nos ordene los datos de la tabla (id).

Para la definición de las otras clases indicadas se ha seguido la misma metodología y realizando algunos cambios y/o añadiendo otros métodos para ampliar funcionalidades a la clase.

A continuación quedan indicados los diferentes modelos creados:

Class PROVIDER:

```
class Provider(models.Model):
    name = models.CharField(max_length=100, unique=True, verbose_name='Nombre')
    CIF = models.CharField(max_length=13, unique=True, verbose_name='CIF')
    mobile = models.CharField(max_length=10, null=True, blank=True, verbose_name='Teléfono móvil')
    address = models.CharField(max_length=500, null=True, blank=True, verbose_name='Dirección')
    email = models.CharField(max_length=500, null=True, blank=True, verbose_name='Email')

    def __str__(self):
        return self.name

    def toJSON(self):
        item = model_to_dict(self)
        return item

    class Meta:
        verbose_name = 'Proveedor'
        verbose_name_plural = 'Proveedores'
        ordering = ['-id']
```

Class CATEGORY:

```
class Category(models.Model):
    name = models.CharField(max_length=150, unique=True, verbose_name='Nombre')
    desc = models.CharField(max_length=500, null=True, blank=True, verbose_name='Descripción')

    def __str__(self):
        return self.name

    def toJSON(self):
        item = model_to_dict(self)
        return item

    class Meta:
        verbose_name = 'Categoría'
        verbose_name_plural = 'Categorías'
        ordering = ['-id']
```

## Class PRODUCT:

```
class Product(models.Model):
    name = models.CharField(max_length=150, unique=True, verbose_name='Nombre')
    cat = models.ForeignKey(Category, on_delete=models.CASCADE, verbose_name='Categoría')
    cost = models.DecimalField(max_digits=9, decimal_places=2, default=0.00, verbose_name='Coste')
    pvp = models.DecimalField(max_digits=9, decimal_places=2, default=0.00, verbose_name='PVP')
    presentation = models.CharField(max_length=50, null=True, blank=True, verbose_name='Presentación')
    image = models.ImageField(upload_to='product/%Y/%m/%d', verbose_name='Imagen', null=True, blank=True)

    def __str__(self):
        return self.name

    def toJSON(self):
        item = model_to_dict(self)
        item['cat'] = self.cat.toJSON()
        item['cost'] = format(self.cost, '.2f')
        item['pvp'] = format(self.pvp, '.2f')
        item['image'] = self.get_image()
        return item

    def get_image(self):
        if self.image:
            return '{}{}'.format(MEDIA_URL, self.image)
        return '{}{}'.format(STATIC_URL, 'img/empty.png')

    def get_stock(self):
        return int(self.inventory_set.filter(saldo__gt=0).aggregate(total=Coalesce(Sum('saldo'), 0)).get('total', 0))

    class Meta:
        verbose_name = 'Producto'
        verbose_name_plural = 'Productos'
        ordering = ['-name']
```

## Class PURCHASE:

```
class Purchase(models.Model):
    prov = models.ForeignKey(Provider, on_delete=models.CASCADE)
    date_joined = models.DateField(default=datetime.now)
    subtotal = models.DecimalField(max_digits=9, decimal_places=2, default=0.00)
    iva = models.DecimalField(max_digits=9, decimal_places=2, default=0.00)
    total = models.DecimalField(max_digits=9, decimal_places=2, default=0.00)

    def __str__(self):
        return self.prov.name
```



```
def calculate_invoice(self):
    subtotal = 0.00
    for d in self.inventory_set.all():
        subtotal += float(d.price) * int(d.cant)
    self.subtotal = subtotal
    self.iva = 0.00
    self.total = float(self.subtotal)
    self.save()

def toJSON(self):
    item = model_to_dict(self)
    item['date_joined'] = self.date_joined.strftime('%Y-%m-%d')
    item['prov'] = self.prov.toJSON()
    item['subtotal'] = format(self.subtotal, '.2f')
    item['iva'] = format(self.iva, '.2f')
    item['total'] = format(self.total, '.2f')
    item['det'] = [v.toJSON() for v in self.inventory_set.all()]
    return item

class Meta:
    verbose_name = 'Compra'
    verbose_name_plural = 'Compras'
    ordering = ['-id']
```

En este caso se ha procedido a crear una nuevo método para el cálculo de la factura de cada una de las compras realizadas y su detalle.

Class Inventory:

```
class Inventory(models.Model):
    purch = models.ForeignKey(Purchase, on_delete=models.CASCADE)
    prod = models.ForeignKey(Product, on_delete=models.CASCADE)
    cant = models.IntegerField(default=0)
    saldo = models.IntegerField(default=0)
    price = models.DecimalField(max_digits=9, decimal_places=2, default=0.00)
    total = models.DecimalField(max_digits=9, decimal_places=2, default=0.00)

    def __str__(self):
        return self.prod.name

    def toJSON(self):
        item = model_to_dict(self, exclude=['purch'])
        item['prod'] = self.prod.toJSON()
        item['price'] = format(self.price, '.2f')
        item['total'] = format(self.total, '.2f')
        return item

    class Meta:
        verbose_name = 'Inventario'
        verbose_name_plural = 'Inventarios'
        ordering = ['-id']
```

## Class CLIENT:

```
class Client(models.Model):
    names = models.CharField(max_length=150, verbose_name='Nombres')
    surnames = models.CharField(max_length=150, verbose_name='Apellidos')
    dni = models.CharField(max_length=10, unique=True, verbose_name='Dni')
    date_birthday = models.DateField(default=datetime.now, verbose_name='Fecha de nacimiento')
    address = models.CharField(max_length=150, null=True, blank=True, verbose_name='Dirección')
    gender = models.CharField(max_length=10, choices=gender_choices, default='male', verbose_name='Sexo')

    def __str__(self):
        return self.names

    def toJSON(self):
        item = model_to_dict(self)
        item['gender'] = {'id': self.gender, 'name': self.get_gender_display()}
        item['date_birthday'] = self.date_birthday.strftime('%Y-%m-%d')
        return item

    class Meta:
        verbose_name = 'Cliente'
        verbose_name_plural = 'Clientes'
        ordering = ['id']
```



## Class SALE:

```
class Sale(models.Model):
    cli = models.ForeignKey(Client, on_delete=models.CASCADE)
    date_joined = models.DateField(default=datetime.now)
    subtotal = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)
    iva = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)
    total = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)

    def __str__(self):
        return self.cli.names

    def toJSON(self):
        item = model_to_dict(self)
        item['cli'] = self.cli.toJSON()
        item['subtotal'] = format(self.subtotal, '.2f')
        item['iva'] = format(self.iva, '.2f')
        item['total'] = format(self.total, '.2f')
        item['date_joined'] = self.date_joined.strftime('%Y-%m-%d')
        item['det'] = [i.toJSON() for i in self.detsale_set.all()]
        return item

    class Meta:
        verbose_name = 'Venta'
        verbose_name_plural = 'Ventas'
        ordering = ['id']
```

## Class DETSALE:

```
class DetSale(models.Model):
    sale = models.ForeignKey(Sale, on_delete=models.CASCADE)
    prod = models.ForeignKey(Product, on_delete=models.CASCADE)
    price = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)
    cant = models.IntegerField(default=0)
    subtotal = models.DecimalField(default=0.00, max_digits=9, decimal_places=2)

    def __str__(self):
        return self.prod.name

    def toJSON(self):
        item = model_to_dict(self, exclude=['sale'])
        item['prod'] = self.prod.toJSON()
        item['price'] = format(self.price, '.2f')
        item['subtotal'] = format(self.subtotal, '.2f')
        return item

    class Meta:
        verbose_name = 'Detalle de Venta'
        verbose_name_plural = 'Detalle de Ventas'
        ordering = ['id']
```



## 5.3- Vistas y formularios

En una aplicación web, el usuario interactúa con la página, lo cual produce una petición http que el servidor recoge. En el caso de Django esta petición es recogida por el componente VIEW.

Las vistas permiten procesar peticiones http, como son los métodos POST y GET. Y Esta vista conecta con el TEMPLATE. La Template contienen el HTML, que va a ser la estructura base para formar el HTML de respuesta.

En nuestro proyecto se ha llevado a cabo la misma estructura para la creación de las diferentes vistas en todos los módulos. Por este motivo a continuación explicaré uno de los más completos, el modelo de SALE.

Tanto en el caso del módulo de Sale, como Purchase, se han creado 5 clases diferentes para las vistas:

- 1- SaleListView: para listar todas las ventas realizadas.
- 2- SaleCreateView: para crear una nueva venta.
- 3- SaleUpdateView: para poder modificar una venta ya creada.
- 4- SaleDeleteView: para poder eliminar una venta existente.
- 5- SaleInvoicePdfView: para crear las facturas de cada una de las ventas.

```
class SaleListView(LoginRequiredMixin, ValidatePermissionRequiredMixin, ListView):
    model = Sale
    template_name = 'sale/list.html'
    permission_required = 'erp.view_sale'

    @method_decorator(csrf_exempt)
    def dispatch(self, request, *args, **kwargs):
        return super().dispatch(request, *args, **kwargs)
```

La parte principal de la vista está definida en la clase y sus parámetros:

- 1- Nombre de la vista: SaleListView.
- 2- LoginRequiredMixin: Este mixin debe estar en la posición más a la izquierda de la herencia y hace la función de que todas las solicitudes de usuarios no logeados serán redirigidas a la página de inicio de sesión.

- 3- `ValidatePermissionRequiredMixin`: comprueba si el usuario que accede a una vista tiene todos los permisos otorgados para acceder a ella.

`PermissionRequired: 'erp.view_sale'`

- 4- `ListView`: parámetro de la vista.

A continuación se debe indicar el modelo al que hace referencia la vista: `Sale`. Y también se debe indicar el `Template` que hace referencia: `'sale/list.html'`

El **method\_decorator** transforma un decorador de funciones en un decorador de métodos para que pueda usarse en un método de instancia. En este caso el **csrf\_exempt** marca una vista como exenta de la protección garantizada por el middleware.

Siguientemente se realiza la creación de las funciones con el método `POST` y el `context` data.

```
def post(self, request, *args, **kwargs):
    data = {}
    try:
        action = request.POST['action']
        if action == 'searchdata':
            data = []
            for i in Sale.objects.all():
                data.append(i.toJSON())
        elif action == 'search_details_prod':
            data = []
            for i in DetSale.objects.filter(sale_id=request.POST['id']):
                data.append(i.toJSON())
        else:
            data['error'] = 'Ha ocurrido un error'
    except Exception as e:
        data['error'] = str(e)
    return JsonResponse(data, safe=False)

def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['title'] = 'Listado de Ventas'
    context['create_url'] = reverse_lazy('erp:sale_create')
    context['list_url'] = reverse_lazy('erp:sale_list')
    context['entity'] = 'Ventas'
    return context
```

En el `context` introducimos los objetos que necesitamos en la vista.

El siguiente paso es crear el fichero HTML para crear la estructura de la vista web. Se deberá crear un fichero HTML para cada una de las vistas basadas en clases. Siguiendo el ejemplo anterior, a continuación queda indicado el Template de sale/list.html:

```
{% extends 'list.html' %}
{% load static %}
{% block head_list %}
    <script src="{% static 'sale/js/list.js' %}"></script>
{% endblock %}

{% block columns %}
    <tr>
        <th scope="col">Nro</th>
        <th scope="col">Cliente</th>
        <th scope="col">Fecha de registro</th>
        <th scope="col">Subtotal</th>
        <th scope="col">Iva</th>
        <th scope="col">Total</th>
        <th scope="col">Opciones</th>
    </tr>
{% endblock %}
```

En el apartado “extends” se hace referencia al archivo html básico creado con la estructura común para todas las clases. De esta forma se consigue realizar el proyecto de forma escalable y de menor dificultad a la hora de modificar el mismo.

Después se hace referencia a la carga de los archivos “static” donde se indica el origen del fichero.

Para finalizar el html de la vista se realiza la estructura de toda la página, indicando los diferentes elementos, como las columnas y formato de las tablas, mediante el primer bloque y el bloque de JavaScript.

A la hora de crear los archivos estáticos, éstos se han realizado en el lenguaje de programación JavaScript. De esta forma, leyendo documentación, y a título personal he podido aprender las bases de un segundo lenguaje de programación.

Se ha utilizado JavaScript para la creación de los archivos estáticos, tanto para los formularios, como el del listado.

En el caso del HTML de CreateView, tal y como se indica a continuación se han cargado las referencias a las librerías y utilidades utilizadas para llevar a cabo el proyecto:

```
{% extends 'list.html' %}
{% load static %}
{% block head_list %}
    <link href="{% static 'lib/jquery-ui-1.12.1/jquery-ui.min.css' %}" rel="stylesheet"/>
    <script src="{% static 'lib/jquery-ui-1.12.1/jquery-ui.min.js' %}"></script>

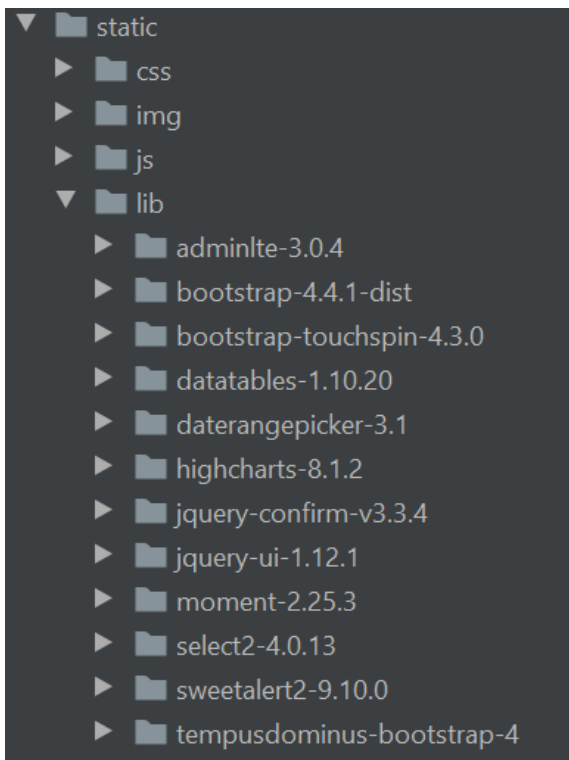
    <link href="{% static 'lib/select2-4.0.13/css/select2.min.css' %}" rel="stylesheet"/>
    <link href="{% static 'lib/select2-4.0.13/css/select2-bootstrap4.min.css' %}" rel="stylesheet"/>
    <script src="{% static 'lib/select2-4.0.13/js/select2.min.js' %}"></script>
    <script src="{% static 'lib/select2-4.0.13/js/i18n/es.js' %}"></script>

    <script src="{% static 'lib/moment-2.25.3/moment-with-locales.js' %}"></script>
    <script src="{% static 'lib/tempusdominus-bootstrap-4/tempusdominus-bootstrap-4.min.js' %}"></script>
    <link href="{% static 'lib/tempusdominus-bootstrap-4/tempusdominus-bootstrap-4.min.css' %}" rel="stylesheet"/>

    <link href="{% static 'lib/bootstrap-touchspin-4.3.0/jquery.bootstrap-touchspin.css' %}" rel="stylesheet"/>
    <script src="{% static 'lib/bootstrap-touchspin-4.3.0/jquery.bootstrap-touchspin.js' %}"></script>

    <script src="{% static 'sale/js/form.js' %}"></script>
{% endblock %}
```

Todas las librerías se han cargado en la carpeta Lib, siguiendo el siguiente enlace:





Siendo las principales librerías y recursos externos utilizados los siguientes:

- 1- Bootstrap: librería principal con la estructura html y otra utilidades.
- 2- AdminLTE: Librería base en referencia al diseño del Dashboard de nuestro proyecto.
- 3- Datatable: usado para la creación de las tablas relacionadas con cada una de las vistas, realizando las consultas mediante **Ajax** y **Jquery**.

### 5.3.2- Relación entre vistas y url's

Como sabemos, cuando el usuario hace click en un link, se ejecuta una petición http en la que viaja una url con parámetros. Si se envían datos de un formulario se ejecuta una petición GET o POST según esté definido y va a una url con parámetros o con un body.

Cuando este contenido llegue a Django, éste será analizado y usando la tabla de relación url's con views se obtendrá la vista deseada.

Este listado queda indicado en el fichero **urls.py**

```
app_name = 'erp'

urlpatterns = [
    # category
    path('category/list/', CategoryListView.as_view(), name='category_list'),
    path('category/add/', CategoryCreateView.as_view(), name='category_create'),
    path('category/update/<int:pk>/', CategoryUpdateView.as_view(), name='category_update'),
    path('category/delete/<int:pk>/', CategoryDeleteView.as_view(), name='category_delete'),
```

### 5.3.3- Formularios

Los formularios son una parte esencial de las aplicaciones web, ya que el usuario puede introducir información mediante ellos.

En cuenta a la programación web suelen ser una de las partes más complejas del sistema. Django suministra una serie de ClassViews orientadas a trabajar con los formularios.

Django dispone de vistas para realizar diferentes acciones en los formularios, como puede ser la creación, modificación, listado y eliminación de un objeto.

CreateView	Para crear un registro
DetailView	Para ver los detalles
DeleteView	Para pedir confirmación al borrado de un registro
ListView	Para ver la lista de registros

En este proyecto se ha llevado a cabo la creación de los formularios según las vistas basadas en clases y se ha realizado su desarrollo en el lenguaje JavaScript.

Siguiendo con el ejemplo de la clase Sale, a continuación queda indicado el archivo **form.js** para las vistas Sale.

```
var tblProducts;
var vents = {
  items: {
    cli: '',
    date_joined: '',
    subtotal: 0.00,
    iva: 0.00,
    total: 0.00,
    products: []
  },
  calculate_invoice: function () {
    var subtotal = 0.00;
    var iva = $('input[name="iva"]').val();
    $.each(this.items.products, function (pos, dict) {
      dict.pos = pos;
      dict.subtotal = dict.cant * parseFloat(dict.pvp);
      subtotal += dict.subtotal;
    });
    this.items.subtotal = subtotal;
    this.items.iva = this.items.subtotal * iva;
    this.items.total = this.items.subtotal + this.items.iva;

    $('input[name="subtotal"]').val(this.items.subtotal.toFixed( fractionDigits: 2));
    $('input[name="ivacalc"]').val(this.items.iva.toFixed( fractionDigits: 2));
    $('input[name="total"]').val(this.items.total.toFixed( fractionDigits: 2));
  },
}
```

En la primera parte se realiza la configuración del formulario con todos los parámetros necesarios para poder realizar la creación de una entidad, definiendo el nombre de las tablas y sus ítems.

Seguidamente se realiza la creación de la función necesaria para la creación de la factura asociada a una venta nueva.

A continuación se realiza la creación de las funciones para llevar a cabo diferentes acciones, como pueden ser el listado de la venta o la creación del report de esa venta.

```
list: function () {
  this.calculate_invoice();
  tblProducts = $('#tblProducts').DataTable( opts: {
    responsive: true,
    autoWidth: false,
    destroy: true,
    data: this.items.products,
    columns: [
      {"data": "id"},
      {"data": "name"},
      {"data": "cat.name"},
      {"data": "pvp"},
      {"data": "cant"},
      {"data": "subtotal"},
    ],
    columnDefs: [
      {
        targets: [0],
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {
          return '<a rel="remove" class="btn btn-danger btn-xs btn-flat" style="color: #f
        }
      },
      {
        targets: [-3],
```

```
        targets: [-3],
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {
          return '€' + parseFloat(data).toFixed( fractionDigits: 2);
        }
      },
      {
        targets: [-2],
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {
          return '<input type="text" name="cant" class="form-control form-control-sm inpu
        }
      },
      {
        targets: [-1],
        class: 'text-center',
        orderable: false,
        render: function (data, type, row) {
          return '€' + parseFloat(data).toFixed( fractionDigits: 2);
        }
      },
    ],
    rowCallback(row, data, displayNum, displayIndex, dataIndex) {
```



Cada uno de los formularios va asociado a la clase de formulario correspondiente según se indica en el fichero **forms.py**. En el caso de ventas, a continuación se indica el mismo:

```
class SaleForm(ModelForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    class Meta:
        model = Sale
        fields = '__all__'
        widgets = {
            'cli': Select(attrs={
                'class': 'form-control select2',
                'style': 'width: 100%'
            }),
            'date_joined': DateInput(
                format='%Y-%m-%d',
                attrs={
                    'value': datetime.now().strftime('%Y-%m-%d'),
                    'autocomplete': 'off',
                    'class': 'form-control datetimepicker-input',
                    'id': 'date_joined',
                    'data-target': '#date_joined',
                    'data-toggle': 'datetimepicker'
                }
            ),
            'iva': TextInput(attrs={
                'class': 'form-control',
            }),
            'subtotal': TextInput(attrs={
                'readonly': True,
                'class': 'form-control',
            }),
            'total': TextInput(attrs={
                'readonly': True,
                'class': 'form-control',
            })
        }
```



## 6- Aplicaciones secundarias

Para la creación del proyecto se ha realizado el desarrollo de otras aplicaciones secundarias, pero de importancia vital para el correcto funcionamiento de la aplicación principal. Para el desarrollo de todas ellas se ha seguido la misma estructura indicada en los apartados anteriores. Estas aplicaciones secundarias son:

- 1- Report: se ha realizado la aplicación report con el objetivo de poder crear los reports, tanto de ventas como de las compras y así poder efectuar un estudio diferencial.
- 2- Login: aplicación de vital importancia para efectuar la creación de permisos para cada grupo diferente de usuarios.
- 3- Usuarios: creación y modificación de usuarios del portal y administrar sus permisos.

En este punto se podría incluir la creación de un “Superuser” con permisos maestros para poder administrar la aplicación web.

## 7- Usabilidad aplicación web ERP

En el punto en el que nos encontramos, se va a mostrar el funcionamiento de la aplicación y su usabilidad.

Como primer punto vamos a mostrar la pantalla inicial del Login. En este punto se debe considerar que el desarrollador principal ha creado un perfil como Superuser para poder administrar otros usuarios que se hayan ido creando.

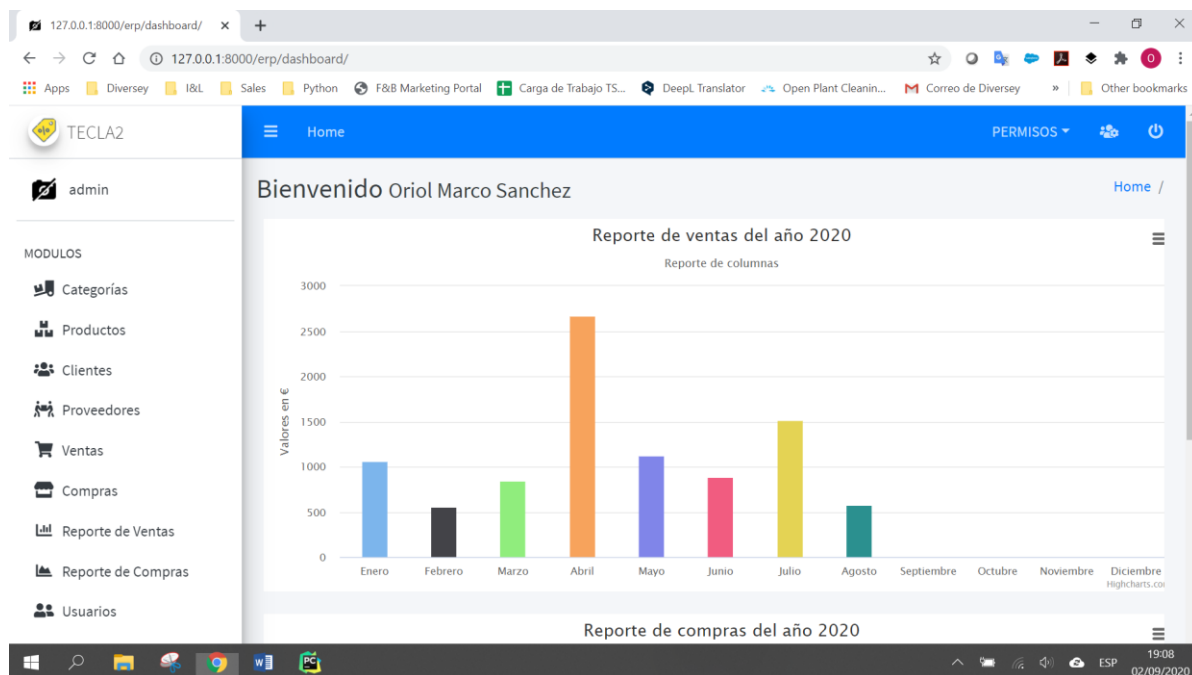
Inicie sesión con sus credenciales

admin

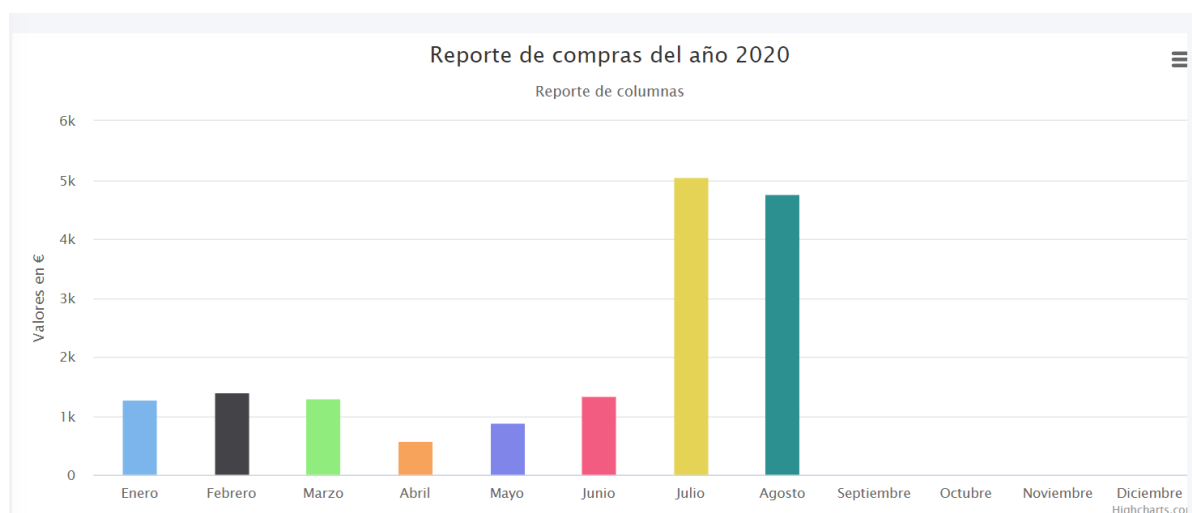
.....

➔ Iniciar sesión

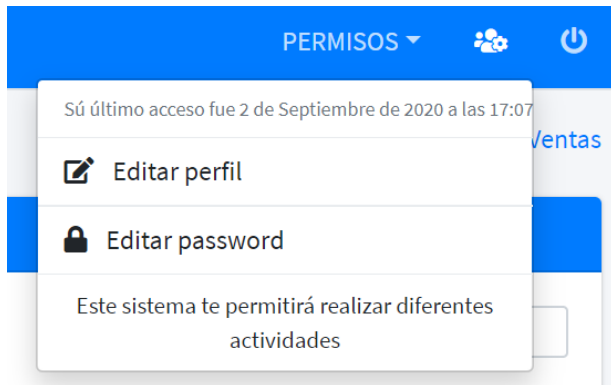
Con la introducción de las credenciales se accede al portal de gestión. Una vez se han introducido las credenciales correctas y no se ha recibido un mensaje de error, se accede a la página principal, donde se observa el dashboard y todos los módulos.



En el dashboard se pueden observar en la parte izquierda un desplegable con todos los módulos del sistema, según las clases y vistas creadas y a la derecha se pueden observar los gráficos, tanto de ventas, como de compras.



En la parte superior se pueden observar las opciones de gestión de la plataforma, como pueden ser los permisos y edición de perfiles, accediendo al formulario de modificación de perfiles:



Bienvenido Oriol Marco Sanchez [Home](#) / [Perfil](#)

### Edición de perfil

**Nombre:**

**Apellidos:**

**Dirección de correo electrónico:**

**Nombre de usuario:**

**Contraseña:**

**Image:**

No file chosen

Siguiendo con la guía del desarrollo del software, clickaremos en el módulo de ventas y veremos todas las funcionalidades y vistas asociadas.



En la vista principal de la venta encontramos diferentes elementos y posibilidades para crear una nueva venta:

- 1- Buscador de productos para el listado de los mismos.
- 2- Botón eliminación de producto escogido.
- 3- Listado de los productos que hemos escogido y formaran parte de la venta.

En la parte derecha observamos la parte del cálculo de la venta y la creación de los datos que conformaran la factura:

**Buscador de productos:**  
  

Eliminar todos mis items

  
Mostrar  registros      Buscar:   

Eliminar	Producto	Categoría	PVP	Cantidad	Subtotal
	Samsung 970 EVO Plus 1TB SSD NVMe M.2	Discos Duros	€219.90	<input type="text" value="-"/> <input type="text" value="1"/> <input type="text" value="+"/>	€219.90
	Gigabyte GeForce RTX 2060 OC 6GB GDDR6	Targetas Gráficas	€329.51	<input type="text" value="-"/> <input type="text" value="1"/> <input type="text" value="+"/>	€329.51

Mostrando registros del 1 al 2 de un total de 2 registros      Anterior 1 Siguiente

**Fecha de venta:**  
  
**Cliente:**  
  
**Subtotal:**  
  
**IVA:**  
  %   
**IVA Calculado:**  
  
**Total a pagar:**

En este punto se indican, tanto la fecha de la venta, el cliente que realiza la compra, subtotal de la venta, el iva escogido y el calculado y el total a pagar por parte del cliente.

SSD NVMe M.2  
Giga...  
GeFo...  
2060...  
GDD...

**Notificación**  
¿Estas seguro de realizar la siguiente acción?  

SI NO

Mostrando registros de...      Anterior 1 Siguiente

Guardar registro

Cancelar

**IVA Calculado:**  
  
**Total a pagar:**



En finalizar la venta, obtendremos un aviso indicando si realmente queremos finalizar la misma, o bien queremos cancelar. Seguidamente obtendremos un segundo aviso solicitando si queremos imprimir el detalle de la venta. En caso afirmativo, obtendremos el pdf de la factura asociada a la venta:

TECLA2 S.A.				
BARCELONA, ESPAÑA				
FACTURA: 136				
FECHA DE VENTA: 3 de Enero de 2020				
CLIENTE: MARÍA				
DNI: 35609456L				
CATEGORIA	PRODUCTO	CANT	P.UNITARIO	TOTAL
Discos Duros	Samsung 970 EVO Plus 1TB SSD NVMe M.2	1	€219,90	€219,90
Teclados	Tempest K9 RGB	1	€23,60	€23,60
SUBTOTAL				€243,50
IVA 21%				€51,14
TOTAL A PAGAR				€294,64
****GRACIAS POR SU COMPRA****				
NO SE ACEPTAN CAMBIOS NI DEVOLUCIONES				

Volviendo al listado de ventas, si procedemos a clicar el botón de la columna NRO, obtendremos un detalle rápido de la venta.

Nro	↕	Cliente	↕	Fecha de registro	↕	Subtotal	Iva	Total	Opciones
		María		2020-01-03		€243.50	€51.14	€294.64	
Producto						Categoría	PVP	Cantidad	Subtotal
Samsung 970 EVO Plus 1TB SSD NVMe M.2						Discos Duros	219.90	1	219.90
Tempest K9 RGB						Teclados	23.60	1	23.60

En la página principal, tenemos dos módulos que hacen referencia a los reportes de ventas y de compras. Si accedemos a ellos, obtendremos un resumen de las ventas realizadas según el periodo de tiempo escogido:

Bienvenido Oriol Marco Sanchez

Home / Reportes

Reporte de Ventas

Rango de fechas:  
2020-01-02 - 2020-09-02

Descargar Excel

Descargar Pdf

Nro	Cliente	Fecha de registro	Subtotal	Iva	Total
136	María	2020-01-03	€243.50	€51.14	€294.64
137	Juan	2020-01-15	€636.06	€133.57	€769.63
138	Santi	2020-02-13	€463.49	€97.33	€560.82

El mismo lo podremos descargar, tanto en Excel, como en pdf.

Volviendo al dashboard, accederemos al módulo de usuarios, donde obtendremos el listado de los mismos y que permisos tienen asociados a cada uno de ellos:

Home

PERMISOS










Bienvenido Oriol Marco Sanchez

Home / Usuarios

Listado de Usuarios

Mostrar 10 registros

Buscar:

Nro	Nombres	Username	Fecha de registro	Imagen	Grupos	Opciones
9	Oriol Marco Sanchez	admin	2020-07-22		Administrador Gestor	 
10	Juan Costa García	juan	2020-08-09		Proveedor	 
11	Marisa García Moreno	marisa	2020-08-24		Cliente	 

Mostrando registros del 1 al 3 de un total de 3 registros

Anterior 1 Siguiente

+ Nuevo registro

Actualizar





Se debe tener en cuenta que los otros módulos se han creado de la misma forma que los mostrados a la largo de la memoria, presentando funcionalidades parecidas o iguales.



co para la localización, información de los  
últimos eventos de la empresa y un pequeño catálogo.

Aunque se muestre en el catálogo un subconjunto de todos los productos, no se podrán  
comprar todos por Internet. Esto se debe a las características de los productos que se venden,  
ya que muchos de ellos no podrán ser transportados por paquetería convencional. Además

6

la empresa quiere poder hacer cambios en el futuro tanto en el portal como en el catálogo,  
por lo que habrá que escribir un pequeño manual con los conceptos básicos para el manejo y  
actualización de su contenido.



## CONCLUSIONES