

Trabajo de Ciclo - Kai Rodríguez García

Importante

La versión de la documentación en MkDocs, una página web, se encuentra en este link:

<https://yooookai.github.io/tfc-docu/>

ÍNDICE

Importante

Proyecto de Final de Ciclo

DSW

Introducción:

Librerías usadas en el backend

APLICACIONES

MAIN

Aplicación Post

Modelo Post

Modelo PostImage

Vistas

Serializadores

Tests

URLs

Aplicación Event

Aplicación Portfolio

Aplicación Personal

Modelo About

Modelo Commissions

Modelo FAQ

Modelo Contact

Vistas (APIView)

Serializadores

Formulario

Urls

Tests

Aplicación Product

Modelo Category

Modelo Product

Modelo ProductImage

Modelo ProductOption

Vistas del módulo de productos

LatestProductsList (APIView)

ProductDetail (APIView)

CategoryDetail (APIView)

search (Función con decorador @api_view(['POST']))

Serializadores del módulo de productos

ProductImageSerializer

ProductOptionSerializer

ProductSerializer

CategorySerializer

URLs de la aplicación Product

Tests

test_latest_products_list

test_product_detail

test_category_detail

test_search_products

test_search_no_query

Aplicación Order

Modelo

Campos principales:

Comportamiento:

Modelo OrderItem

Campos principales:

Comportamiento:

Vistas

checkout (Función vista)

OrdersList (Clase basada en vista - APIView)

Tests

Setup inicial

Test: Éxito en checkout

Test: Checkout con campos faltantes

Test: Listado de pedidos del usuario

Urls

EMPRESAS

PLAN DE MARKETING

1.1 Análisis de mercado
1.2 Propuesta de valor
1.3 Estrategias de promoción y publicidad
1.4 Estrategia de ventas y precios
1.5 Medición y KPIs
PLAN DE SOSTENIBILIDAD
2.1 Sostenibilidad económica
2.2 Sostenibilidad social
2.3 Sostenibilidad ambiental

DOR

Adaptabilidad
Accesibilidad y Usabilidad
Diseño Responsive
Home
Navbar
Galería:
INFO - Comisiones
Caja. Desglose de pedido

DEW

Dependencias ([dependencies](#))
Componentes y vistas
Navbar.vue (componente)
Login.vue
SignUp.vue
Modal.vue
AuthModals.vue
Store.vue
CategoriesSidebar.vue
Search.vue
ProductBox.vue
ProductsGrid.vue
Category
CategoryView
Contact

Cart.vue

CartItem.vue

Checkout.vue

MyAccount
OrderSummary
Success
BaseTitle

[CardPost](#)
[SocialLinks](#)
[SolidButton](#)
[About](#)
[Commissions](#)

[EventsView.vue](#)

[FAQView.vue](#)

[Funcionalidades principales](#)

[PostsView.vue](#)

[Funcionalidades principales](#)

[PostDetail.vue](#)

[Funcionalidades principales](#)

[Product.vue](#)

[Funcionalidades principales](#)

[mainStore.js](#)

[1. Estado \(`state` \)](#)

[2. Getters](#)

[3. Actions](#)

[Uso típico](#)

[Router:](#)

[Tests](#)

[Login](#)

[Contacto](#)

[Checkout](#)

[DPL](#)

[Explicación detallada paso a paso del despliegue con Gunicorn, Supervisor, Nginx y Certbot en Ubuntu](#)

[1. Conectarse al servidor por SSH](#)

[2. Actualizar paquetes del sistema](#)

[3. Instalar software necesario para el proyecto](#)

[4. Crear base de datos y usuario en PostgreSQL](#)

[5. Instalar herramientas para Python y entornos virtuales](#)

[6. Crear estructura de carpetas y usuarios para la app](#)

[7. Crear entorno virtual Python para el proyecto](#)

[8. Preparar archivo con dependencias del proyecto](#)

[9. Subir dependencias al servidor y instalar](#)

[10. Subir proyecto Django al servidor](#)

[11. Configurar Django para producción](#)

[12. Migrar la base de datos y permisos en PostgreSQL](#)

[13. Instalar Gunicorn y preparar script de inicio](#)

- [14. Instalar y configurar Supervisor para mantener Gunicorn vivo](#)
- [15. Configurar Nginx para servir la app y hacer proxy a Gunicorn](#)
- [16. Comprar dominio y configurar SSL con Certbot](#)

Configuración de seguridad y dominio en Django ([settingsprod.py](#))

Añadir dominios al proyecto:

[Reiniciar Gunicorn vía Supervisor](#)

[Crear superusuario de Django](#)

[Cambiar URL hardcodeada en modelos](#)

Despliegue del Frontend Vue

[Configurar el endpoint del backend en Vue](#)

[Compilar y empaquetar Vue](#)

[Enviar el build al servidor remoto](#)

[Desempaquetar en el servidor](#)

Configuración del servidor NGINX para el Frontend

[Crear archivo de configuración NGINX](#)

[Activar el sitio en NGINX](#)

[Reiniciar NGINX para aplicar cambios](#)

Verificación final

Proyecto de Final de Ciclo

Mi proyecto consiste en el desarrollo de una plataforma web integral para una ilustradora profesional. La plataforma actúa como un escaparate digital multifuncional que incluye: portfolio artístico, tienda online de productos, información sobre comisiones, blog personal, formulario de contacto, calendario de eventos y una sección informativa completa. Esta solución personalizada responde directamente a las necesidades específicas de la artista, centralizando todas sus actividades profesionales en un único espacio digital accesible y fácil de gestionar.

DSW

Introducción:

El backend del proyecto está desarrollado con **Django** y se encarga de gestionar la lógica de negocio y los datos esenciales de la aplicación: productos, usuarios, pedidos, publicaciones de blog, eventos, imágenes y más. Mediante **Django REST Framework**, se crean las APIs que comunican esta información con el frontend. Django facilita el desarrollo gracias a sus herramientas integradas como el sistema de administración, el manejo de usuarios y su conexión con bases de datos.

Librerías usadas en el backend

El proyecto ha experimentado y explorado posibilidades, descubriendo librerías nuevas que me ayudaron a aumentar la rapidez para en la creación de apis.

- **Django** y **DRF** forman la base del backend y la API REST.
- **djoser** y **simplejwt** facilitan la autenticación de usuarios mediante tokens JWT.
- **django-cors-headers** permite la comunicación con el frontend en Vue configurando el acceso entre orígenes distintos.
- **pillow** gestiona la carga y edición de imágenes.
- **stripe** permite integrar pagos seguros en la web.
- **cryptography** y **defusedxml** refuerzan la seguridad del proyecto, protegiendo datos sensibles y evitando vulnerabilidades XML.
- **Kubi** - personalizar el panel de administración
- También se incluyen bibliotecas como **requests**, **oauthlib**, o **social-auth**.

APLICACIONES

Este proyecto cuenta con varias aplicaciones. Main, Post, Event, order, personal, portfolio, product, order. A continuación hablaré resumidamente de cada una de ellas.

Cuenta con la carpeta media, donde se guardan todas las imágenes subidas al proyecto.

MAIN

La app `main` centraliza la configuración general del proyecto, como rutas, bases de datos, correo y middlewares. Incluye la configuración de **envío de correos mediante Gmail** para facilitar la comunicación en el formulario de contacto, y define los **endpoints principales bajo `/api/v1/`**, lo que permite escalar futuras versiones (como `v2`). También gestiona la carga de archivos multimedia y estática, y define las rutas que enlazan con otras apps como `post`, `product`, `order`, etc.

```
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include('djoser.urls')),
    path('api/v1/', include('djoser.urls.authtoken')),
    path('api/v1/', include('post.urls')),
    path('api/v1/', include('product.urls')),
    path('api/v1/', include('order.urls')),
    path('api/v1/', include('event.urls')),
    path('api/v1/', include('portfolio.urls')),
    path('api/v1/', include('personal.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Les he escrito el prefijo v1 para que no haya conflicto en caso de una nueva actualización de la api, a la que llamaría v2, etc...

Djoser: `path('api/v1/', include('djoser.urls')),`

Incluye las URLs estándar de Djoser para manejar autenticación (registro, login, logout, cambio de contraseña, etc).

`path('api/v1/', include('djoser.urls.authtoken')),`

Incluye rutas específicas para el **token de autenticación** cuando usas el sistema de tokens con Django REST Framework.

Aplicación Post

Esta aplicación gestiona publicaciones o entradas de blog, permitiendo almacenar contenido textual junto con imágenes relacionadas, organizadas en categorías definidas.

Modelo `Post`

Representa una entrada o publicación en el blog. Incluye los siguientes campos:

- `title`: Título del post, máximo 256 caracteres.
- `slug`: Cadena única generada automáticamente a partir del título para URLs amigables.
- `content`: Contenido principal del post (texto libre).
- `created_at`: Fecha y hora de creación, asignada automáticamente.
- `category`: Categoría del post, con opciones predefinidas (`Artist Alley`, `Daily Life`, `Announcement`, etc.).

El método `save()` sobreescrito asegura que el campo `slug` se actualice automáticamente usando la función `slugify` con base en el título, facilitando la generación de URLs limpias.

La clase `Meta` ordena los posts por fecha de creación descendente para mostrar primero los más recientes.

```
class Post(models.Model):  
    class Category(models.TextChoices):  
        ARTIST_ALLEY = 'AL', 'Artist Alley'  
        DAILY_LIFE = 'DL', 'Daily Life'  
        OTHER = 'OT', 'Other'  
        ANNOUNCEMENT = 'AN', 'Announcement'  
  
        title = models.CharField(max_length=256)  
        slug = models.SlugField(unique=True)
```

```

content = models.TextField()
created_at = models.DateTimeField(auto_now_add=True)
category = models.CharField(max_length=2, choices=Category.choices,
default=Category.OTHER)

def save(self, *args, **kwargs):
    self.slug = slugify(self.title)
    super().save(*args, **kwargs)

def __str__(self):
    return self.title

def get_absolute_url(self):
    return f'/{self.slug}/'

class Meta:
    ordering = ['-created_at']

```

Modelo PostImage

Permite asociar múltiples imágenes a un post, con los siguientes campos:

- `post`: Relación Many-to-One con el modelo `Post`.
- `image`: Archivo de imagen que se guarda en la carpeta `uploads/blog_posts/`.
- `caption`: Texto descriptivo o pie de foto, opcional.

```

class PostImage(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='images')
    image = models.ImageField(upload_to='uploads/blog_posts/')
    caption = models.CharField(max_length=256, blank=True)

    def __str__(self):
        return f"Image for: {self.post.title}"

```

Vistas

Se utilizan vistas basadas en clases (APIView) para ofrecer acceso a las publicaciones a través de una API RESTful.

- **PostList** : Devuelve una lista de posts, con opción a filtrar por categoría a través de un parámetro GET `category` .
- **PostDetail** : Devuelve los detalles completos de un post identificado por su `slug` .

Ambas usan serializers para estructurar la respuesta JSON.

```
class PostList(APIView):  
    def get(self, request, format=None):  
        category = request.GET.get('category', None)  
        posts = Post.objects.all()  
        if category:  
            posts = posts.filter(category=category)  
        serializer = PostListSerializer(posts, many=True)  
        return Response(serializer.data)  
  
class PostDetail(APIView):  
    def get_object(self, slug):  
        try:  
            return Post.objects.get(slug=slug)  
        except Post.DoesNotExist:  
            raise Http404  
  
    def get(self, request, slug, format=None):  
        post = self.get_object(slug)  
        serializer = PostDetailSerializer(post)  
        return Response(serializer.data)
```

Serializadores

Transforman los modelos en datos JSON para la API.

- **PostImageSerializer** : Serializa imágenes con su ruta y caption.
- **PostListSerializer** : Resume los posts con título, slug, fecha, categoría legible, resumen del contenido y lista de imágenes.

- `PostDetailSerializer`: Muestra todos los detalles del post, incluyendo contenido completo e imágenes.

```

class PostImageSerializer(serializers.ModelSerializer):
    class Meta:
        model = PostImage
        fields = ['image', 'caption']

class PostListSerializer(serializers.ModelSerializer):
    summary = serializers.SerializerMethodField()
    category_display = serializers.CharField(source='get_category_display', r
ead_only=True)
    images = PostImageSerializer(many=True, read_only=True)

    class Meta:
        model = Post
        fields = ['title', 'slug', 'created_at', 'category', 'category_display', 'sum
mary', 'images']

    def get_summary(self, obj):
        return ' '.join(obj.content.split()[:25]) + '...'

class PostDetailSerializer(serializers.ModelSerializer):
    images = PostImageSerializer(many=True, read_only=True)
    category_display = serializers.CharField(source='get_category_display', r
ead_only=True)

    class Meta:
        model = Post
        fields = ['title', 'slug', 'content', 'created_at', 'category', 'category_displ
ay', 'images']

```

Tests

Se valida el comportamiento de la API para obtener la lista de posts y el detalle de un post específico, incluyendo la gestión de errores para slugs no existentes.

- Se crea un post de prueba en `setUp`.
- Se verifica que la lista de posts incluya dicho post.
- Se consulta el detalle por slug y se confirma la respuesta.
- Se prueba que un slug inexistente devuelve un error 404.

```
class PostAPITest(TestCase):
    def setUp(self):
        self.client = APIClient()
        self.post = Post.objects.create(
            title="Test Post",
            content="Contenido de prueba para el post.",
            category=Post.Category.ANNOUNCEMENT
        )

    def test_post_list(self):
        response = self.client.get('/api/v1/posts/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertTrue(any(p['title'] == "Test Post" for p in response.data))

    def test_post_detail(self):
        response = self.client.get(f'/api/v1/posts/{self.post.slug}/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(response.data['title'], "Test Post")

    def test_post_not_found(self):
        response = self.client.get('/api/v1/posts/slug-inexistente/')
        self.assertEqual(response.status_code, status.HTTP_404_NOT_FOUND)
```

URLs

Se definen rutas RESTful para la API de posts:

```
urlpatterns = [
    path('posts/', PostList.as_view(), name='post-list'),
```

```
    path('posts/<slug:slug>', PostDetail.as_view(), name='post-detail'),  
]
```

Aplicación Event

Esta aplicación contiene el modelo UpcomingEvent que representa eventos próximos con detalles básicos para su presentación.

- **Campos principales:**

- `title` (CharField): Título del evento.
- `image` (ImageField): Imagen asociada al evento, guardada en la carpeta `uploads/events/`.
- `start_date` (DateField): Fecha de inicio del evento.
- `end_date` (DateField, opcional): Fecha de finalización, permite eventos de varios días.
- `location` (CharField): Lugar donde se realizará el evento.
- `link` (URLField, opcional): Enlace externo relacionado con el evento (por ejemplo, página oficial).
- `location_link` (URLField, opcional): Enlace para la ubicación (como un mapa o dirección web).

- **Comportamiento:**

- Ordena los eventos automáticamente por fecha de inicio (`start_date`).
- Método `is_multi_day()` que devuelve `True` si el evento dura más de un día.

- **Representación:**

- El método `_str_` retorna el título para facilitar su identificación en interfaces administrativas.

Tiene una vista basada en APIView que crea un endpoint para obtener la lista completa de eventos próximos. Al recibir una petición GET, consulta todos los registros del modelo

`UpcomingEvent`, la serializa con `UpcomingEventSerializer` y devuelve los datos en formato JSON.

La url para esta es:

```
path('events/', UpcomingEventList.as_view(), name='upcoming-events'),
```

También tiene un test que crea un evento de prueba, hace una petición GET al endpoint `/events/`, verifica que la respuesta es 200 OK y comprueba que el evento creado aparece en la respuesta.

Aplicación Portfolio

Representa una entrada del portafolio.

Modelo

- `title` (CharField): Título de la entrada del portafolio, con un máximo de 255 caracteres.
- `image` (ImageField): Imagen asociada a la entrada, almacenada en la carpeta `uploads/portfolio/`.
- `category` (CharField): Categoría de la entrada, con opciones predefinidas: Concept Art, Animación, Ilustración, y Otros (por defecto).
- `created_at` (DateTimeField): Fecha y hora en que se creó la entrada, se asigna automáticamente al crear el registro.

Las entradas se ordenan por fecha de creación descendente (`ordering = ['-created_at']`), mostrando primero las más recientes.

Como en el caso anterior, es una aplicación simple con una vista basada en APIView que obtiene una lista de registros que son serializados. Y una URL para acceder a los datos.

```
urlpatterns = [  
    path('portfolio/', PortfolioEntryList.as_view(), name='portfolio-list'),  
]
```

Aplicación Personal

Contiene modelos que gestionan la información personal, comisiones, preguntas frecuentes y contactos de la web.

Modelo About

- `title` (CharField, opcional): Título descriptivo, puede quedar vacío.
- `content` (TextField): Texto con información sobre la persona o proyecto.
- `image` (ImageField, opcional): Imagen relacionada, almacenada en `media/about/`.

Representa la sección "Acerca de mí" o información general.

Modelo Commissions

- `title` (CharField, opcional): Título del servicio o comisión ofrecida.
- `description` (TextField): Descripción detallada de la comisión.
- `price` (DecimalField): Precio de la comisión, con hasta 10 dígitos y 2 decimales.
- `slots_left` (PositiveIntegerField): Número de plazas disponibles para la comisión.
- `image` (ImageField, opcional): Imagen ilustrativa guardada en `media/comissions/`.

Gestiona las ofertas y disponibilidad de comisiones personalizadas.

Modelo FAQ

- `question` (CharField): Pregunta frecuente.
- `answer` (TextField): Respuesta correspondiente a la pregunta.

Modelo para mostrar las preguntas frecuentes y sus respuestas.

Modelo Contact

- `name` (CharField): Nombre del usuario que contacta.
- `email` (EmailField): Correo electrónico del usuario.
- `message` (TextField): Mensaje enviado por el usuario.
- `created_at` (DateTimeField): Fecha y hora en que se creó el contacto (registro automático).

Permite registrar los mensajes y consultas recibidas desde la sección de contacto.

Vistas (APIView)

- **ContactView (POST)**

Recibe nombre, email y mensaje, valida campos, guarda contacto y envía email notificando el mensaje.

- **AboutView (GET)**

Devuelve la primera instancia de About serializada.

- **CommissionsList (GET)**

Devuelve la lista completa de comisiones serializadas.

- **FAQList (GET)**

Devuelve todas las preguntas frecuentes serializadas.

Serializadores

Se basan en `ModelSerializer` y exponen todos los campos de sus modelos:

`AboutSerializer`, `ComissionsSerializer`, `FAQSerializer`

Formulario

ContactForm, un sencillo formulario para recibir mensajes desde el front.

Campos: `name`, `email`, `message` con validación básica, usado para validar datos de contacto.

Urls

```
urlpatterns = [
    path('contact/', ContactView.as_view(), name='contact'),
    path('about/', AboutView.as_view(), name='about'),
    path('commissions/', ComissionsList.as_view(), name='commissions'),
    path('faq/', FAQList.as_view(), name='faq'),
]
```

Tests

He añadido unos test de comprobación sencillos en esta app:

- **test_get_about:** Verifica que la vista de "About" responde con éxito (200 OK) y que el contenido incluye el título esperado.
- **test_get_commissions:** Comprueba que la lista de comisiones se obtiene correctamente y que devuelve al menos un elemento.
- **test_get_faq:** Asegura que la lista de preguntas frecuentes (FAQ) se recupera correctamente y no está vacía.
- **test_post_contact_success:** Prueba que al enviar un mensaje de contacto válido, la API responde con éxito, guarda el mensaje en la base de datos y confirma el envío.

```
def test_post_contact_success(self):
    url = reverse('contact')
    data = {
        'name': 'Sergio',
        'email': 'matraca@example.com',
        'message': 'Hola matraca, este es un mensaje de prueba.'
    }
    response = self.client.post(url, data)
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(response.data['message'], '¡Se ha enviado el mensaje!')
    self.assertTrue(Contact.objects.filter(email='matraca@example.com').exists())
```

- **test_post_contact_missing_fields:** Verifica que si faltan datos obligatorios en el formulario de contacto, la API responde con un error 400 y muestra un mensaje de error.

Aplicación Product

Modelo Category

Este modelo representa las categorías de productos en la tienda. Cada categoría tiene un nombre (`name`) y un `slug` único que se utiliza para construir URLs amigables. Las categorías se ordenan alfabéticamente por nombre para facilitar su visualización.

- **Campos:**

- `name` : Nombre descriptivo de la categoría.

- `slug` : Identificador único para URLs.

- **Métodos importantes:**

- `get_absolute_url()` : Devuelve la URL relativa basada en el slug, facilitando la navegación hacia la categoría.

- **Meta:**

- Ordena las categorías por nombre (`ordering = ('name',)`).
-

Modelo Product

Este modelo define los productos que se ofrecen en la tienda, cada uno vinculado a una categoría específica mediante una relación de clave foránea (`ForeignKey`). Incluye detalles básicos como nombre, descripción, precio y fechas de creación. También permite gestionar imágenes principales y miniaturas, generando automáticamente una miniatura si no existe para optimizar la carga.

- **Campos principales:**

- `category` : Categoría a la que pertenece el producto.

- `name` : Nombre del producto.

- `slug` : Identificador único para URLs.

- `description` : Descripción detallada (opcional).

- `price` : Precio del producto con dos decimales.

- `image` : Imagen principal del producto.

- `thumbnail` : Imagen en miniatura, optimizada para cargas rápidas.

- `date_added` : Fecha en que se añadió el producto.

- `weight` : Peso base del producto en gramos (importante para envíos y logística).

- **Métodos relevantes:**

- `get_absolute_url()` : Construye la URL del producto usando el slug de la categoría y del producto.

- `get_image()` : Retorna la URL completa de la imagen principal.

- `get_thumbnail()` : Retorna la URL de la miniatura, generándola automáticamente si no existe.
- `make_thumbnail(image, size)` : Crea una miniatura redimensionando y optimizando la imagen principal.

- **Meta:**

- Ordena los productos por fecha de adición, mostrando primero los más recientes (`ordering = ('-date_added')`).

```
class Product(models.Model):
    category = models.ForeignKey(Category, related_name='products', on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    slug = models.SlugField(unique=True)
    description = models.TextField(blank=True, null=True)
    price = models.DecimalField(max_digits=6, decimal_places=2)
    image = models.ImageField(upload_to='uploads/', blank=True, null=True)
    thumbnail = models.ImageField(upload_to='uploads/', blank=True, null=True)
    date_added = models.DateTimeField(auto_now_add=True)
    weight = models.PositiveIntegerField(default=0, help_text="Peso base en gramos")

    class Meta:
        ordering = ('-date_added',)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return f'/{self.category.slug}/{self.slug}/'

    def get_image(self):
        if self.image:
            return 'http://127.0.0.1:8000' + self.image.url
        return ''

    def get_thumbnail(self):
        if self.thumbnail:
            return 'http://127.0.0.1:8000' + self.thumbnail.url
        return ''
```

```

else:
    if self.image:
        self.thumbnail = self.make_thumbnail(self.image)
        self.save()
        return 'http://127.0.0.1:8000' + self.thumbnail.url
    else:
        return ""

def make_thumbnail(self, image, size=(600, 400)):

    img = Image.open(image)
    img = img.convert('RGB')
    img.thumbnail(size)

    thumb_io = BytesIO()
    img.save(thumb_io, 'JPEG', quality=85)

    thumbnail = File(thumb_io, name= image.name)
    return thumbnail

```

Modelo ProductImage

Este modelo permite añadir múltiples imágenes adicionales para un producto, mejorando la presentación visual en la tienda.

- **Campos:**

- `product` : Producto asociado (FK).
- `image` : Imagen adicional.
- `alt_text` : Texto alternativo para accesibilidad y SEO (opcional).

Modelo ProductOption

Define opciones o variantes de un producto que pueden afectar su precio y peso, como diferentes tamaños, colores o configuraciones personalizadas.

- **Campos:**

- `product` : Producto asociado (FK).

- `name` : Nombre de la opción o variante.
- `additional_price` : Precio adicional que suma esta opción al producto base.
- `additional_weight` : Incremento o decremento de peso en gramos que añade esta opción.

Este conjunto de modelos está diseñado para gestionar la tienda en línea con productos categorizados, número de imágenes flexible y variantes personalizables, proporcionando una base robusta para manejar catálogo, presentación y cálculo de precios/pesos para envíos. Además, incluye funciones para optimizar la carga de imágenes con miniaturas generadas automáticamente.

Vistas del módulo de productos

LatestProductsList (APIView)

- **Descripción:**

Proporciona una lista completa de todos los productos disponibles en la tienda.

- **Métodos:**

- `get` : Recupera todos los objetos `Product`, los serializa y devuelve los datos en formato JSON.

- **Uso:**

Endpoint para mostrar todos los productos recientes o disponibles.

ProductDetail (APIView)

- **Descripción:**

Muestra el detalle de un producto específico, identificándolo por el slug de la categoría y el slug del producto.

- **Métodos:**

- `get_object(category_slug, product_slug)` : Busca el producto que corresponde a la categoría y slug indicados. Si no existe, lanza un error 404.
- `get` : Usa `get_object` para obtener el producto y devuelve su información serializada.

- **Uso:**

Endpoint para obtener la información completa de un producto en particular.

CategoryDetail (APIView)

- **Descripción:**

Proporciona la información detallada de una categoría identificada por su slug.

- **Métodos:**

- `get_object(category_slug)` : Busca la categoría por slug o lanza error 404 si no existe.
- `get` : Devuelve los datos serializados de la categoría encontrada.

- **Uso:**

Endpoint para ver detalles de una categoría específica.

search (Función con decorador `@api_view(['POST'])`)

- **Descripción:**

Permite buscar productos mediante una consulta de texto, que compara el término con los campos `name` y `description`.

- **Parámetros:**

- Recibe un JSON con el campo `query` que contiene la palabra o frase a buscar.

- **Funcionamiento:**

- Filtra los productos que contengan el texto buscado en su nombre o descripción (case-insensitive).
- Devuelve la lista de productos que coinciden, o una lista vacía si no hay término de búsqueda.

- **Uso:**

Endpoint para implementar funcionalidades de búsqueda dinámica en la tienda.

```
@api_view(['POST'])
def search(request):
    query = request.data.get('query', "")
    if query:
        products = Product.objects.filter(Q(name__icontains=query) | Q(descripti
            serializer = ProductSerializer(products, many=True)
            return Response(serializer.data)
    else:
        return Response({'products': []})
```

Serializadores del módulo de productos

ProductImageSerializer

- **Descripción:**

Serializa las imágenes asociadas a un producto.

- **Campos:**

- `image`: archivo de imagen.
- `alt_text`: texto alternativo para accesibilidad.
- `full_image_url`: URL completa absoluta generada dinámicamente para acceder a la imagen.

- **Métodos especiales:**

- `get_full_image_url`: Construye la URL completa de la imagen usando el contexto de la petición para devolver un enlace absoluto.

```
class ProductImageSerializer(serializers.ModelSerializer):
    full_image_url = serializers.SerializerMethodField()

    class Meta:
        model = ProductImage
        fields = ['image', 'alt_text', 'full_image_url']
```

```
def get_full_image_url(self, obj):
    request = self.context.get('request')
    if request:
        return request.build_absolute_uri(obj.image.url)
    return obj.image.url
```

ProductOptionSerializer

- **Descripción:**

Serializa las opciones adicionales de un producto, como variantes con precio y peso extra.

- **Campos:**

- `name` : nombre de la opción.
- `additional_price` : incremento en el precio para esta opción.
- `additional_weight` : incremento en el peso para esta opción.

ProductSerializer

- **Descripción:**

Serializa un producto, incluyendo sus datos básicos, imágenes y opciones relacionadas.

- **Campos:**

- `id` , `name` , `price` , `weight` : información básica del producto.
- `get_absolute_url` : URL relativa al detalle del producto.
- `description` : descripción del producto.
- `get_image` , `get_thumbnail` : métodos que devuelven las URLs de imagen principal y miniatura.
- `images` : lista de imágenes adicionales, serializadas con `ProductImageSerializer` .
- `options` : lista de opciones del producto, serializadas con `ProductOptionSerializer` .

CategorySerializer

- **Descripción:**

Serializa una categoría, incluyendo la lista de productos que pertenecen a ella.

- **Campos:**

- `id`, `name`: datos básicos de la categoría.
- `get_absolute_url`: URL relativa para acceder a la categoría.
- `products`: lista de productos relacionados, serializados con `ProductSerializer`.

URLs de la aplicación Product

```
from django.urls import path
from product import views

urlpatterns = [
    path('latest-products/', views.LatestProductsList.as_view(), name='latest-pr
    path('products/search/', views.search, name='product-search'),
    path('products/<slug:category_slug>/<slug:product_slug>/', views.ProductI
    path('products/<slug:category_slug>/', views.CategoryDetail.as_view(), nam
]
```

Tests

Utilizo `APITestCase` de Django REST Framework para validar las principales funcionalidades de la API de productos, asegurando que las vistas respondan correctamente y que los datos devueltos sean los esperados.

```
from rest_framework.test import APITestCase
from rest_framework import status
from django.urls import reverse
from .models import Category, Product

class ProductAPITest(APITestCase):
```

```
def setUp(self):
    # Crear una categoría y un producto para usar en los tests
    self.category = Category.objects.create(name="Test Category", slug
= "test-category")
    self.product = Product.objects.create(
        category=self.category,
        name="Test Product",
        slug="test-product",
        price=9.99,
        weight=100
    )
```

test_latest_products_list

Este test verifica que la vista de productos más recientes (`latest-products`) responda con éxito (código 200) y que el producto creado en `setUp` aparezca en la lista.

```
def test_latest_products_list(self):
    url = reverse('latest-products')
    # Construcción dinámica de la URL usando el nombre de la ruta
    response = self.client.get('/api/v1/latest-products/')
    # Se puede usar directamente la URL
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    # Comprobamos respuesta exitosa
    # Confirmamos que "Test Product" esté en los datos recibidos
    self.assertTrue(any(p['name'] == "Test Product" for p in response.data))
```

test_product_detail

Comprueba que la vista de detalle de producto funcione correctamente, devolviendo el producto esperado cuando se consulta por categoría y slug.

```
def test_product_detail(self):
    url = f'/api/v1/products/{self.category.slug}/{self.product.slug}/'
    response = self.client.get(url)
    self.assertEqual(response.status_code, status.HTTP_200_OK)
```

```
    self.assertEqual(response.data['name'], "Test Product")
    # Validamos que el producto sea el correcto
```

test_category_detail

Valida que la vista de detalle de categoría responde correctamente e incluye la lista de productos asociados.

```
def test_category_detail(self):
    url = f'/api/v1/products/{self.category.slug}/'
    response = self.client.get(url)
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(response.data['name'], "Test Category")
    # Confirmamos la categoría
    self.assertTrue('products' in response.data)
    # Verificamos que se devuelvan los productos relacionados
```

test_search_products

Test para la funcionalidad de búsqueda. Envía una consulta y verifica que el producto que contiene el término se incluya en la respuesta.

```
def test_search_products(self):
    url = '/api/v1/products/search/'
    response = self.client.post(url, {'query': 'Test'}, format='json')
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertTrue(any(p['name'] == "Test Product" for p in response.data))
```

test_search_no_query

Asegura que si no se envía ningún término de búsqueda, la respuesta sea exitosa y se retorne una lista vacía.

```
def test_search_no_query(self):
    url = '/api/v1/products/search/'
    response = self.client.post(url, {}, format='json')
```

```
self.assertEqual(response.status_code, status.HTTP_200_OK)
self.assertEqual(response.data['products'], [])
```

Aplicación Order

Modelo

Representa una orden de compra realizada por un usuario en la tienda. Contiene información del cliente, detalles del pago y estado del envío.

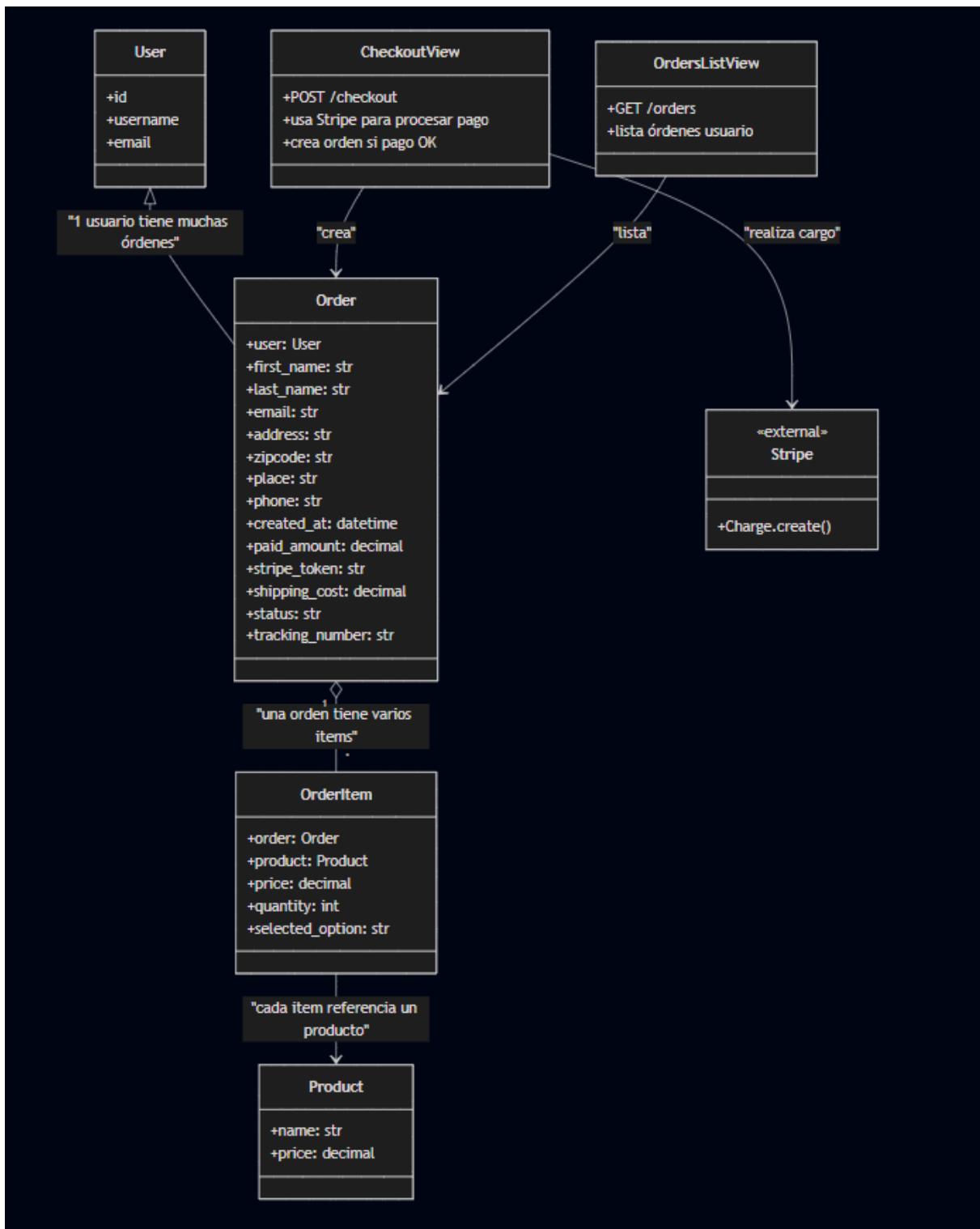
Campos principales:

- `user`: Relación con el modelo `User` de Django. Indica el cliente que realizó el pedido.
- `first_name`, `last_name`, `email`, `address`, `zipcode`, `place`, `phone`: Datos personales y de contacto del comprador para la entrega.
- `created_at`: Fecha y hora en que se creó el pedido. Se asigna automáticamente.
- `paid_amount`: Monto total pagado por el pedido. Puede estar vacío si el pago no se ha procesado aún.
- `stripe_token`: Token asociado a Stripe para el pago, usado para validación y seguimiento del pago.
- `shipping_cost`: Costo del envío, que puede variar según el destino o método.
- `status`: Estado actual del pedido. Puede ser:
 - `"processing"` (En proceso)
 - `"shipped"` (Enviado)
 - `"delivered"` (Entregado)
- `tracking_number`: Número de seguimiento para el envío, opcional y se asigna cuando está disponible.

Comportamiento:

- Los pedidos se ordenan por fecha de creación, mostrando primero las más recientes (`ordering = ['-created_at']`).

- La representación en texto (`__str__`) devuelve el nombre del comprador, para identificación rápida en el admin o logs.



Modelo OrderItem

Representa un ítem individual dentro de un pedido, es decir, un producto comprado.

Campos principales:

- `order` : Relación con `Order` . Un ítem pertenece a un pedido específico.
- `product` : Relación con `Product` . El producto que fue comprado.
- `price` : Precio unitario del producto en el momento de la compra. Esto garantiza que cambios futuros en precios no afecten el histórico de órdenes.
- `quantity` : Cantidad comprada de ese producto.
- `selected_option` : Opcional, describe alguna variante o opción elegida para el producto (por ejemplo, talla, color).

Comportamiento:

- La representación en texto devuelve el `id` del ítem, para referencias sencillas.
-

Esta aplicación permite gestionar pedidos complejos, donde cada orden puede contener múltiples productos con opciones específicas. Se almacena información del cliente y del pago para facilitar la gestión y seguimiento, así como el estado de envío.

Vistas

`checkout` (Función vista)

Descripción:

Permite a un usuario autenticado realizar el pago y crear una nueva orden.

Método:

`POST`

Autenticación y permisos:

- Solo usuarios autenticados mediante token pueden acceder.

Flujo principal:

1. Recibe los datos de la orden desde el cliente.
2. Valida los datos con `OrderSerializer`.
3. Calcula el total a cobrar sumando el costo de los productos y el envío.
4. Realiza el cobro mediante la API de Stripe usando el token recibido.
5. Si el pago es exitoso, guarda la orden asociándola al usuario autenticado.
6. Devuelve la orden creada con estado HTTP 201.

Errores:

- Si falla la validación o el cobro con Stripe, devuelve error 400 con detalles.

```

@api_view(['POST'])
@authentication_classes([authentication.TokenAuthentication])
@permission_classes([permissions.IsAuthenticated])
def checkout(request):
    serializer = OrderSerializer(data=request.data)

    if serializer.is_valid():
        stripe.api_key = settings.STRIPE_SECRET_KEY

        items = serializer.validated_data['items']
        shipping_cost = serializer.validated_data.get('shipping_cost', 0)

        paid_amount = sum(item.get('quantity') * item.get('product').price for item in items)
        total_to_charge = paid_amount + shipping_cost

        try:
            charge = stripe.Charge.create(
                amount=int(total_to_charge * 100),
                currency='EUR',
                description='Charge from Charge from TangerineMessStore',
                source=serializer.validated_data['stripe_token']
            )

            serializer.save(user=request.user, paid_amount=total_to_charge)
        except stripe.error.StripeError as e:
            return Response({'error': str(e)}, status=400)
    else:
        return Response(serializer.errors, status=400)

```

```
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    except Exception:
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

OrdersList (Clase basada en vista - APIView)

Descripción:

Lista todas las órdenes realizadas por el usuario autenticado.

Método:

GET

Autenticación y permisos:

- Solo usuarios autenticados mediante token pueden acceder.

Flujo principal:

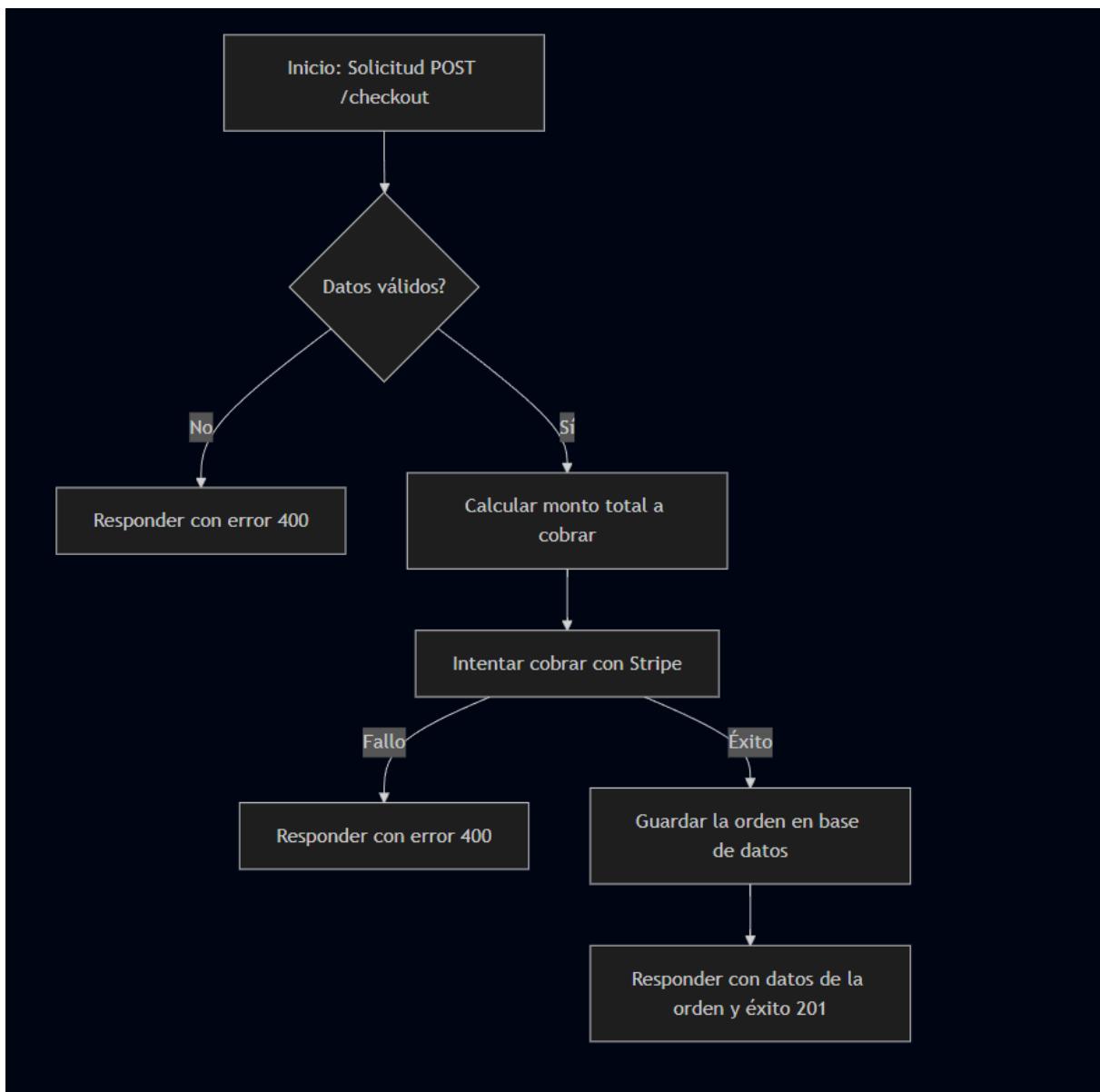
1. Filtra las órdenes por el usuario autenticado.
2. Serializa la lista con `MyOrderSerializer`.
3. Devuelve la lista de órdenes en formato JSON.

```
class OrdersList(APIView):
    authentication_classes = [authentication.TokenAuthentication]
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request, format=None):
        orders = Order.objects.filter(user=request.user)
        serializer = MyOrderSerializer(orders, many=True)
        return Response(serializer.data)
```

- Se usa Stripe para procesar pagos, configurando la clave secreta desde la configuración del proyecto.
- La lógica de cobro es manejada dentro de la vista `checkout`, asegurando que solo se cree la orden tras confirmarse el pago.

- El uso de autenticación por token garantiza seguridad y control de acceso a las órdenes y pagos.



Tests

Estos tests comprueban el correcto funcionamiento de los endpoints relacionados con la creación de pedidos (checkout) y la consulta de pedidos existentes para un usuario autenticado.

```

from rest_framework.test import APITestCase
from rest_framework import status
  
```

```
from django.contrib.auth.models import User
from rest_framework.authtoken.models import Token
from django.urls import reverse
from .models import Order, OrderItem
from product.models import Product, Category
```

Setup inicial

Se crea un usuario de prueba con token de autenticación, así como una categoría y producto para simular la compra.

```
def setUp(self):
    self.user = User.objects.create_user(username='testuser', password='testpass')
    self.token = Token.objects.create(user=self.user)
    self.client.credentials(HTTP_AUTHORIZATION='Token ' + self.token.key)

    self.category = Category.objects.create(name="Test Category", slug="test-category")
    self.product = Product.objects.create(
        category=self.category,
        name="Test Product",
        slug="test-product",
        price=10.00,
        weight=100
    )
```

Test: Éxito en checkout

Este test envía una solicitud POST válida a la ruta de checkout para crear un pedido. Se valida que:

- La respuesta sea HTTP 201 (creado).
- Se haya creado un pedido y un ítem relacionado.
- El monto pagado sea el correcto (precio productos + envío).

```
def test_checkout_success(self):
    url = reverse('checkout')
```

```

data = {
    "first_name": "Sergio",
    "last_name": "Delgado",
    "email": "matraca@example.com",
    "address": "123 Matracazo",
    "zipcode": "12345",
    "place": "City",
    "phone": "1234567890",
    "stripe_token": "tok_visa",
    "shipping_cost": "5.00",
    "items": [
        {
            "product": self.product.id,
            "quantity": 2,
            "price": "10.00"
        }
    ]
}
response = self.client.post(url, data, format='json')
self.assertEqual(response.status_code, status.HTTP_201_CREATED)
self.assertEqual(Order.objects.count(), 1)
self.assertEqual(OrderItem.objects.count(), 1)
self.assertEqual(Order.objects.first().paid_amount, 25.00) # (2*10) + 5 en envío

```

Test: Checkout con campos faltantes

Verifica que la API rechace solicitudes incompletas o inválidas, devolviendo un error HTTP 400.

```

def test_checkout_missing_fields(self):
    url = reverse('checkout')
    data = {
        "first_name": "Sergio",
        "stripe_token": "tok_visa",
        "items": []
    }
    response = self.client.post(url, data, format='json')

```

```
    self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST)
```

Test: Listado de pedidos del usuario

Simula la consulta GET para obtener los pedidos creados por el usuario autenticado. Verifica que la respuesta sea correcta y que se devuelva la cantidad adecuada de pedidos.

```
def test_orders_list(self):
    order = Order.objects.create(
        user=self.user,
        first_name="Sergio",
        last_name="Doe",
        email="matraca@example.com",
        address="123 Matracazo",
        zipcode="12345",
        place="City",
        phone="1234567890",
        stripe_token="tok_visa",
        paid_amount=15.00
    )
    response = self.client.get(reverse('orders-list'))
    self.assertEqual(response.status_code, status.HTTP_200_OK)
    self.assertEqual(len(response.data), 1)
```

Urls

```
from django.urls import path
from . import views

urlpatterns = [
    path('checkout/', views.checkout, name='checkout'),
    path('orders/', views.OrdersList.as_view(), name='orders-list'),
]
```

EMPRESAS

PLAN DE MARKETING

1.1 Análisis de mercado

El público objetivo comprende principalmente personas entre 15 y 40 años, apasionadas por el anime, los videojuegos, la música y las series. Son fans de artistas independientes que buscan productos originales y personalizados que reflejen sus intereses.

El producto estrella de la artista —ilustraciones convertidas en merchandising como llaveros, pines, peluches y pegatinas— conecta con ese perfil de consumidor que busca piezas únicas ausentes en el merchandising oficial, frecuentemente limitado o genérico.

En cuanto a la competencia, numerosos artistas utilizan plataformas como Etsy, Ko-fi o redes sociales (Instagram, Twitter) para promocionarse. Sin embargo, pocos disponen de una web profesional que centralice su presencia: tienda, portfolio, contacto y eventos. Esta web personal marca la diferencia al ofrecer una experiencia de usuario más coherente, profesional y directa. Además, reduce comisiones de terceros y permite adaptar el diseño a la estética de la artista.

En los últimos años ha crecido la visibilidad del **Artist Alley** en ferias de manga y cómic por toda España, y el público valora cada vez más las obras únicas. El cliente ideal aprecia el contacto directo con la artista, por lo que la web incluye un formulario para pedidos personalizados y comisiones.

1.2 Propuesta de valor

La tienda ofrece **productos ilustrados de forma original**, creados por una artista independiente, con un estilo visual propio y una conexión emocional con personajes y universos que los fans adoran.

Además de productos artesanales únicos, la web funciona como escaparate del talento de la artista: incluye portfolio, blog, calendario de eventos, contacto directo y una tienda profesional.

Todo está centralizado, accesible desde cualquier red social o tarjeta de visita en eventos. Esto brinda al público una experiencia más directa, personalizada y confiable, permitiendo a la artista construir una comunidad fiel.

1.3 Estrategias de promoción y publicidad

- **Redes sociales (Instagram, Tumblr, Twitter):** Publicaciones periódicas mostrando nuevos productos, procesos creativos, contenido detrás de cámaras ("work in progress") y participación en eventos.
- **Blog en la web:** Artículos sobre la inspiración de sus personajes, diseño de productos y experiencias en ferias, fortaleciendo el vínculo con la comunidad.
- **Colaboraciones con tiendas o artistas** para ampliar alcance y diversificar el público.
- **Promociones:** Pegatinas de regalo en primeras compras online, descuentos por combo o durante eventos especiales.
- **Difusión cruzada:** Incluir la dirección web en tarjetas, flyers, packaging y redes para redirigir tráfico a la tienda online desde mercadillos físicos.

1.4 Estrategia de ventas y precios

- **Estructura de precios:** Basada en tipo de producto, tamaño y materiales. Ej: pegatinas (1-2 €), llaveros (5-8 €), peluches (20-40 €), comisiones digitales por encargo (según detalle y tipo).
- **Upselling:** Incentivar la compra de productos de mayor valor, como packs o productos premium (ediciones especiales).
- **Cross-selling:** Combos de productos relacionados (ej: "combo anime X" con llavero, postal y pin).
- **Stock B:** Venta con descuento de productos con imperfecciones mínimas.
- **Ediciones limitadas:** Productos únicos o con tiradas cortas para crear exclusividad ("solo 20 unidades disponibles").

1.5 Medición y KPIs

Para evaluar el rendimiento del proyecto se emplearán estos indicadores medibles:

- **Conversiones:** Relación entre visitas, registros y ventas en la tienda.
- **Engagement en redes sociales:** Número de likes, comentarios, compartidos y mensajes directos.

- **Crecimiento de comunidad:** Seguidores en Instagram/X o visitas recurrentes.
- **Clientes recurrentes:** Usuarios que recompran en un periodo determinado e incremento de clientes en eventos físicos gracias a la visibilidad web.
- **ROI (Retorno de la inversión):** Medición del rendimiento de acciones promocionales, como ofrecer productos extra en pedidos online o recompensar en eventos físicos a compradores previos de la web. El ROI comparará el gasto en promociones con los beneficios obtenidos en ventas directas, fidelización y aumento del tráfico web, evaluando el impacto positivo real y sostenible de estas estrategias.

PLAN DE SOSTENIBILIDAD

2.1 Sostenibilidad económica

La web propia permite reducir costes de intermediarios (como Etsy, Ko-fi) y controlar directamente el canal de ventas, aumentando el margen de beneficio por producto.

Centralizar la actividad reduce gastos en publicidad dispersa y herramientas externas. La gestión bajo demanda evita exceso de stock, y la estructura web está diseñada para crecer en funcionalidad según el negocio evoluciona, sin grandes inversiones iniciales.

La plataforma facilita la implementación de estrategias de venta como packs, promociones y comisiones personalizadas.

2.2 Sostenibilidad social

El diseño web incorpora navegación accesible, textos inclusivos y lenguaje cercano, creando una experiencia positiva para públicos diversos.

La sección de blog y portfolio divulga contenidos que conectan con intereses culturales y educativos, promoviendo valores de creatividad, identidad personal y comunidad.

La web potencia el trabajo de una creadora canaria, impulsando el talento local y facilitando conexiones con otros artistas y eventos, tanto en Canarias como en la península.

La creación de cuentas de usuario con beneficios exclusivos fomenta una comunidad cercana y duradera.

2.3 Sostenibilidad ambiental

La web está optimizada para ser ligera y eficiente, con imágenes y recursos optimizados para reducir el consumo energético. Se utilizará un proveedor de hosting verde o de bajas emisiones, alineado con buenas prácticas ecológicas.

La disponibilidad digital del catálogo, información de eventos y contacto reduce la necesidad de materiales impresos, minimizando residuos.

En conjunto, la plataforma promueve un consumo más consciente, donde cada pedido es informado, meditado y gestionado con mejor trazabilidad que los canales masivos o impersonales.

DOR

Adaptabilidad

Para este proyecto he elegido una paleta de colores basada en azules y naranjas, representativos de la artista a quien está dedicada la web. Estos colores están definidos como variables CSS en un archivo separado para facilitar su uso consistente en todo el proyecto y permitir un mantenimiento sencillo:

```
:root {  
    --color-light-orange: #ffe2bb;  
    --color-primary: #F59300;  
    --color-secondary: #FF9100;  
    --color-secondary-hover: #cc6300;  
    --color-bg: #377AB9;  
    --color-dark-blue: #0056b3;  
    --color-darker-blue: #034184;  
    --color-darkest-blue: #1d1d35;
```

```
--color-warm-cream: #ffffbf8;  
--color-warmer-cream:#fdfdeb;  
--color-yellow: #fdeaa5;  
--color-light-blue: #81aedb;  
--color-lighter-blue: #cbe5ff;  
--color-grey: #838897;  
--color-light-grey: #b8bfd3;  
}
```

El uso de colores con buen contraste garantiza que los elementos sean fácilmente localizables y legibles, mejorando la experiencia visual y funcional del usuario. Se han seleccionado tonos que combinan bien para destacar botones, textos y otros componentes clave.

En cuanto a la fuente, he utilizado principalmente dos familias, Degular, y ObviouslyNarrow. La primera para textos descriptivos, y la segunda para titulares, por su carácter alargado. Ambas son letras redondeadas. Además, modiflico esta según ayude a la legibilidad, como por ejemplo, añadiendo la propiedad de letter-spacing en números o menús de navegación para hacer a todo el mundo entender bien lo que está leyendo. En general, uso tamaños de fuente grandes para permitir a los visitantes leer sin problema.

```
@font-face {  
    font-family: 'Degular';  
    src: url('../fonts/DegularDisplay-Regular.otf') format('opentype');  
    font-weight: normal;  
    font-style: normal;  
}  
  
@font-face {  
    font-family: 'ObviouslyNarrow';  
    src: url('../fonts/ObviouslyNarrow-Regular.otf') format('opentype');  
    font-weight: normal;  
    font-style: normal;  
}  
  
@font-face {  
    font-family: 'ObviouslyNarrowBold';  
    src: url('../fonts/ObviouslyNarrow-Bold.otf') format('opentype');  
    font-weight: bold;  
    font-style: bold;
```

}

Accesibilidad y Usabilidad

Se ha prestado especial atención a la accesibilidad para asegurar que la web sea usable para la mayor cantidad de personas posible:

- Se emplean etiquetas `<label>` correctamente asociadas con todos los campos de formulario, facilitando el uso con lectores de pantalla y mejorando la navegación por teclado.
- Todas las imágenes cuentan con atributos `alt` descriptivos, incluyendo las imágenes de productos, para ofrecer contexto a usuarios con discapacidades visuales.
- Los indicadores visuales para la interacción, como cambios de cursor, efectos hover en enlaces y botones, o animaciones sutiles en tarjetas, ofrecen señales claras sobre la interactividad de los elementos.
- La navegación es clara, con botones y enlaces intuitivos, con tooltips informativos en iconos y elementos que requieren explicación adicional.
- Se han implementado elementos como contadores visibles en el carrito y botones de fácil acceso para acciones comunes (volver arriba, navegación atrás), que mejoran la experiencia del usuario.
- La interfaz es simple y consistente, reduciendo la curva de aprendizaje y haciendo que la web sea accesible para usuarios con diferentes niveles de habilidad digital.
- Utilizo toast o mensajes de información al usuario tras haber realizado acciones como añadir al carrito, registrarse o comprar.

Diseño Responsive

El diseño es completamente responsive, adaptándose fluidamente a distintos tamaños y dispositivos:

- Se utiliza una combinación de grid flexible (`flexbox`, CSS grid) y el sistema de columnas de Bootstrap (`col-12`, `col-md-6`, etc.) para que los componentes y las tarjetas se reorganizan y escalen correctamente en móviles, tablets y escritorios.

- Los menús de navegación se transforman en menús desplegables o hamburguesa en dispositivos móviles para optimizar el espacio y facilitar el acceso.
- Los tamaños de fuentes y botones escalan de forma proporcional según el dispositivo para mantener una legibilidad óptima y facilitar la interacción táctil.
- Las imágenes y demás contenidos gráficos se ajustan para evitar desbordamientos o distorsiones, manteniendo la coherencia visual.
- Se han incorporado media queries para modificar estilos específicos en función del ancho de pantalla, logrando un comportamiento adaptado sin sacrificar la estética en dispositivos grandes.

```

<!-- Productos y búsqueda -->
<div class="col-12 col-md-9 col-sm-12">
<form method="get" action="/search" class="mb-4 search-form ms-auto">
  <div class="input-group">
    <input
      type="text"
      class="form-control"
      placeholder="¿Qué estás buscando?"
      name="query"
    />
    <button class="btn btn-custom" type="submit">
      <i class="fas fa-search"></i>
    </button>
  </div>
</form>

```

```

@media (max-width: 992px) {
  .portfolio-grid {
    column-count: 2;
  }
}

@media (max-width: 576px) {
  .portfolio-grid {
    column-count: 1;
  }
}

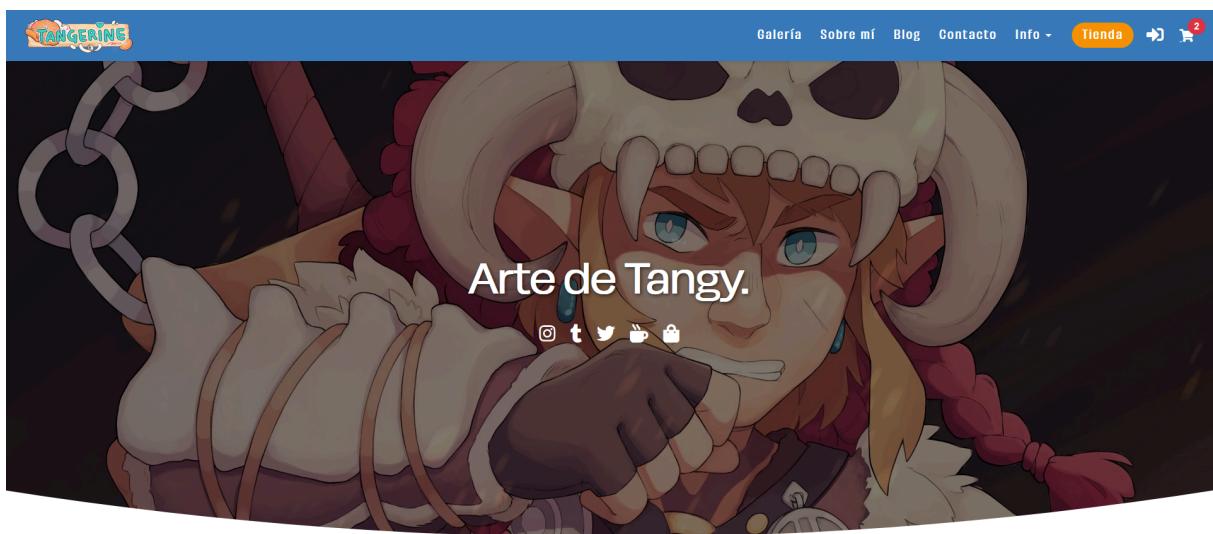
@media (min-width: 768px) {
  .search-form {
    width: 50%;
  }
}

@media (min-width: 1200px) {
  .search-form {
    width: 25%;
  }
}

```

Home

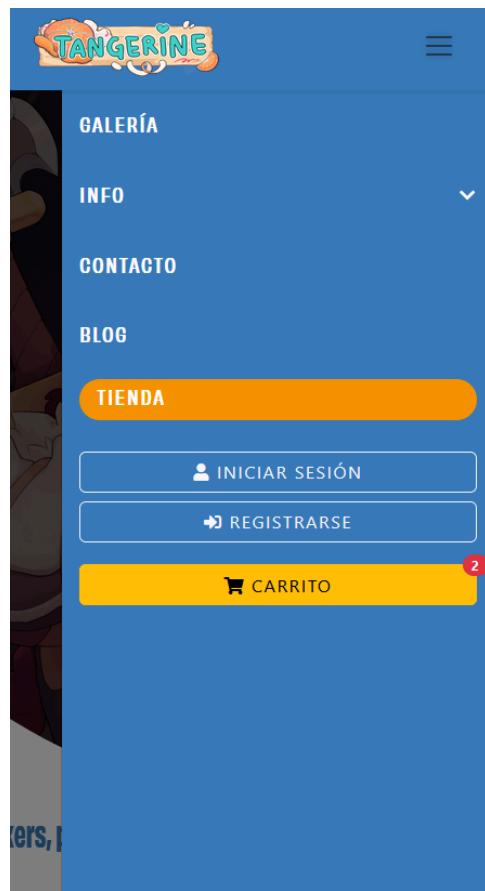
En este caso, tengo un texto que se desplaza a lo largo de la pantalla, y he agregado un evento para evitar que se sobreponga en pantallas más pequeñas, editando su ancho.





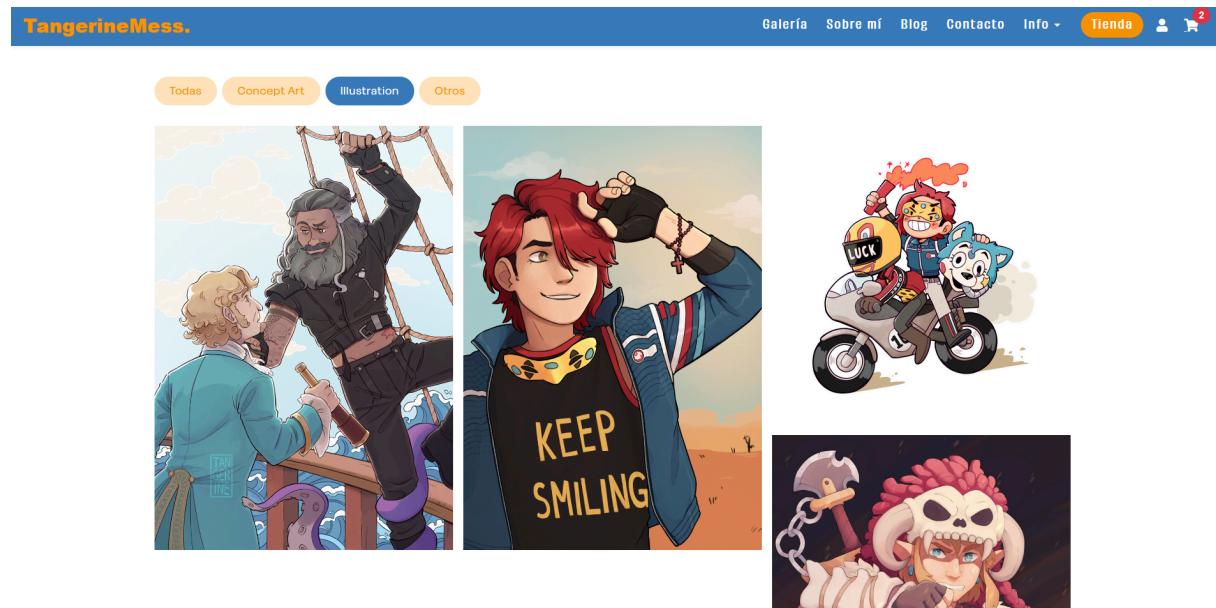
Navbar

Este se convierte en menú hamburguesa en dispositivos pequeños.

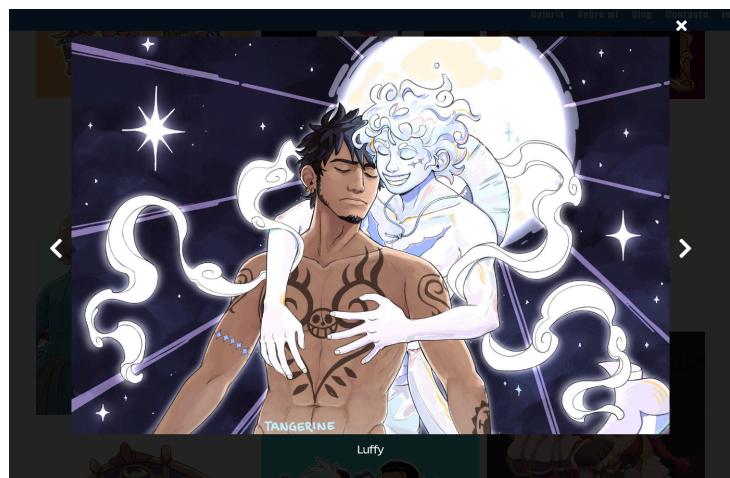
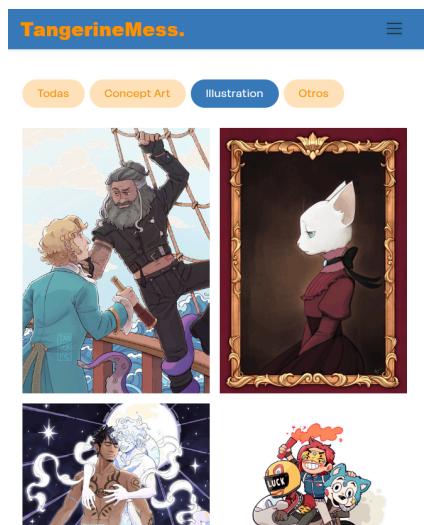


Galería:

Tiene etiquetas para las categorías que filtran, y estos botones activos están en azul.

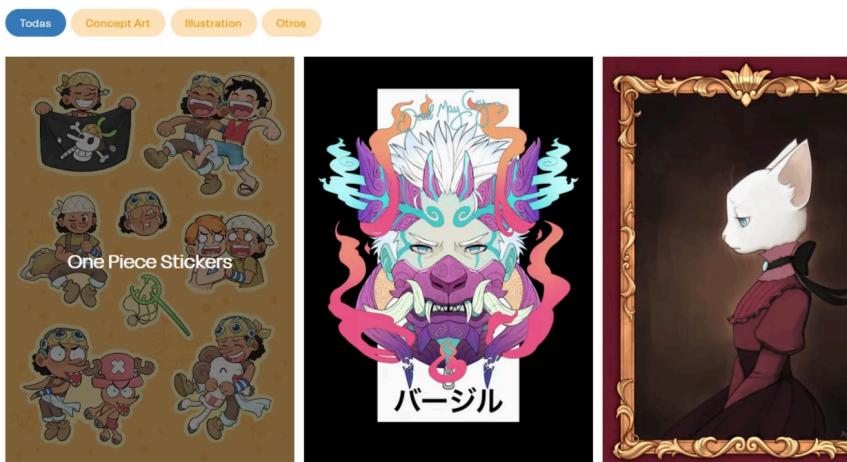


Pantalla pequeña



Se puede navegar entre elementos por flechas, y cerrar al darle click a la x, o cualquier parte fuera de la imagen.

Al hacer hover, se muestra el título de la imagen.



INFO - Sobre mí

TANGERINE

Galería Sobre mí Blog Contacto Info Tienda  

Hola!

Hola, soy Aurora, ilustradora freelance afincada en España. Trabajo principalmente en ilustración digital, aunque también me muevo entre el diseño gráfico, la narrativa visual y algún que otro experimento creativo.

Tengo tres tatuajes de My Chemical Romance, lo que ya dice bastante sobre mí, y un perro precioso que probablemente sea más popular que yo. Además de dibujar, me paso la vida entre música, cómics y teorías innecesarias sobre personajes que me gustan demasiado (sí, me shippeo con Corazón, y no me arrepiento).

Trabajo por encargo, colaboro en proyectos personales y también me gusta publicar cosas que nacen simplemente porque me apetece. Si te interesa lo emocional, lo raro, o lo que está entre ambos, probablemente encuentres algo aquí que te llame la atención.

Estoy siempre abierta a nuevas ideas, propuestas y colaboraciones.



TANGERINE

Hola!

Hola, soy Aurora, ilustradora freelance afincada en España. Trabajo principalmente en ilustración digital, aunque también me muevo entre el diseño gráfico, la narrativa visual y algún que otro experimento creativo.

Tengo tres tatuajes de My Chemical Romance, lo que ya dice bastante sobre mí, y un perro precioso que probablemente sea más popular que yo. Además de dibujar, me paso la vida entre música, cómics y teorías innecesarias sobre personajes que me gustan demasiado (sí, me shippeo con Corazón, y no me arrepiento).

Trabajo por encargo, colaboro en proyectos...



INFO - Comisiones

Se adapta a las pantallas.

Comisiones Disponibles



Coloured sketches

Slots disponibles: 5

Uno o dos personajes por dibujo.

Puede ser cuerpo completo o medio cuerpo.

35.00 €

Coloured sketch icon

Slots disponibles: 3

Uno o dos personajes en el mismo dibujo.

Puede ser un ícono (cabeza y cuello), o busto (parte de hombros y pecho también).

25.00 €



Comisiones Disponibles



Coloured sketches

Slots disponibles: 5

Uno o dos personajes por dibujo.

Puede ser cuerpo completo o medio cuerpo.

35.00 €

Coloured sketch icon

Slots disponibles: 3

Uno o dos personajes en el mismo dibujo.

Puede ser un ícono (cabeza y cuello), o busto (parte de hombros y pecho también).

25.00 €



Coloured sketch icon

Slots disponibles: 3

Uno o dos personajes en el mismo dibujo.

Puede ser un ícono (cabeza y cuello), o busto (parte de hombros y pecho también).

25.00 €

Info - Eventos

TangerineMess.

Galería Sobre mi Blog Contacto Info Tienda 2



Próximos eventos



16 may - 13 may 2025
TGG
La Laguna, Recinto
[Localización](#) [Web](#)



16 jul - 19 jul 2025
TLP SUMMER CON
Santa Cruz de Tenerife
[Localización](#) [Web](#)

TangerineMess.



Próximos eventos



16 may - 13 may 2025
TGG
La Laguna, Recinto
[Localización](#) [Web](#)

Info - Preguntas y Respuestas

Las preguntas desplegadas cambian de color e icono.

TangerineMess.

Galería Sobre mí Blog Contacto Info Tienda

Preguntas Frecuentes

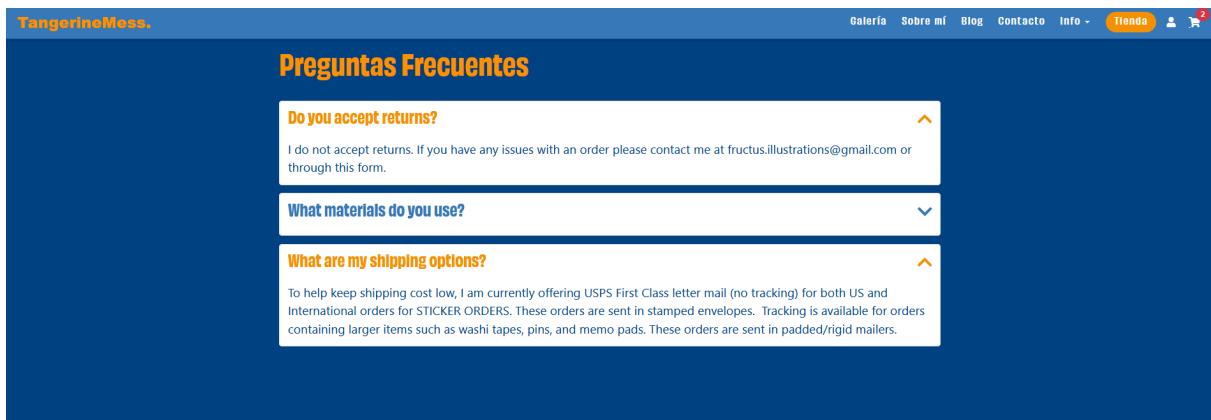
Do you accept returns?

I do not accept returns. If you have any issues with an order please contact me at fructus.illustrations@gmail.com or through this form.

What materials do you use?

What are my shipping options?

To help keep shipping cost low, I am currently offering USPS First Class letter mail (no tracking) for both US and International orders for STICKER ORDERS. These orders are sent in stamped envelopes. Tracking is available for orders containing larger items such as washi tapes, pins, and memo pads. These orders are sent in padded/rigid mailers.



TangerineMess.

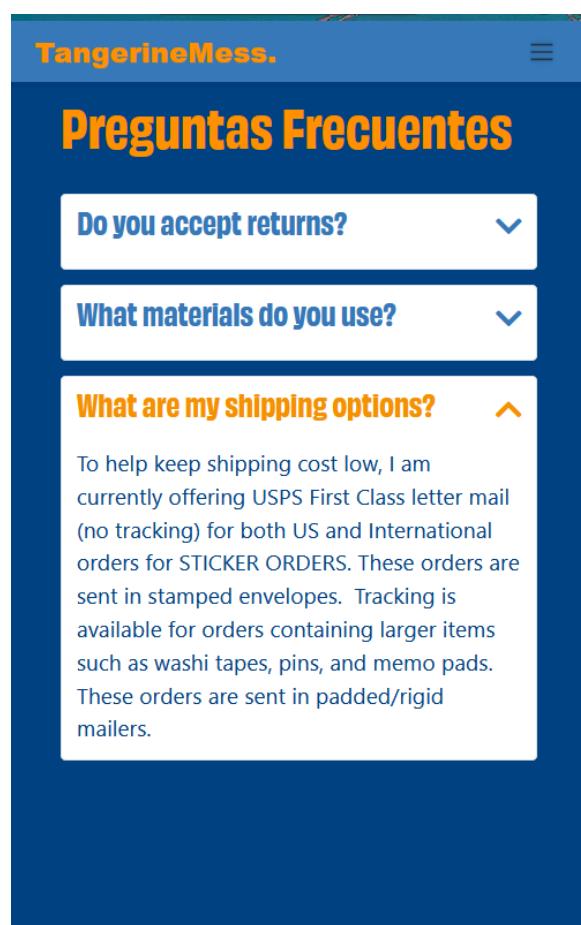
Preguntas Frecuentes

Do you accept returns?

What materials do you use?

What are my shipping options?

To help keep shipping cost low, I am currently offering USPS First Class letter mail (no tracking) for both US and International orders for STICKER ORDERS. These orders are sent in stamped envelopes. Tracking is available for orders containing larger items such as washi tapes, pins, and memo pads. These orders are sent in padded/rigid mailers.



Contacto

¡HABLEMOS!

Cada gran colaboración comienza con una conversación.

¿Quieres colaborar conmigo o pedir una comisión? Ponte en contacto enviando un correo [aquí](#) o completa el siguiente formulario:

También puedes encontrarme en mis redes:



Nombre (requerido)

Email (requerido)

Mensaje (requerido)

Enviar

¡HABLEMOS!

Cada gran colaboración comienza con una conversación.

¿Quieres colaborar conmigo o pedir una comisión? Ponte en contacto enviando un correo [aquí](#) o completa el siguiente formulario:

También puedes encontrarme en mis redes:



Nombre (requerido)

¡HABLEMOS!

Cada gran colaboración comienza con una conversación.

¿Quieres colaborar conmigo o pedir una comisión? Ponte en contacto enviando un correo [aquí](#) o completa el siguiente formulario:

También puedes encontrarme en mis redes:



Ko-fi

Nombre (requerido)

Tiene tooltip para indicar el significado del ícono.

Blog

TangerineMess.

Últimos posts

[Todas las Categorías ▾](#)

10 MAY 2025

OTHER

NUEVO DISEÑO

10 MAY 2025

DAILY LIFE

PIN DE PIRATAS

26 ABR 2025

ARTIST ALLEY

CINE DE VERANO

TangerineMess.

Últimos posts

[Todas las Categorías ▾](#)

10 MAY 2025

OTHER

NUEVO DISEÑO

10 MAY 2025

DAILY LIFE

PIN DE PIRATAS

26 ABR 2025

ARTIST ALLEY

CINE DE VERANO

Todos los días, a partir de las 20.30H, habrá una proyección totalmente abierta y gratuita para todas las personas...

[Ler más >](#)

TangerineMess.

10 MAY 2025

OTHER

NUEVO DISEÑO

10 MAY 2025

DAILY LIFE

PIN DE PIRATAS

26 ABR 2025

ARTIST ALLEY

CINE DE VERANO

Todos los días, a partir de las 20.30H, habrá una proyección totalmente abierta y gratuita para todas las personas...

[Ler más >](#)

[Todas las Categorías ▾](#)

[Todas las Categorías](#)

[Artist Alley](#)

[Vida Diaria](#)

[Otro](#)

[Anuncio](#)

Blog/post

TangerineMess.

PIN DE PIRATAS
10 may 16:45

n muchos casos, es causado por un complemento del navegador (Ej, un bloqueador de anuncios o un protector de privacidad) que blokea la solicitud.

Otras causas posibles:

Intentar acceder a un recurso https que tenga un certificado no válido, causará este error.
 Intentar acceder a un recurso http desde una página con un origen https también causará este error.
 A partir de Firefox 68, las páginas https no pueden acceder a http://localhost, aunque esto puede ser modificado por el Error 1488740.
 El servidor no respondió a la solicitud actual (incluso si respondió la solicitud Preflight. Un escenario podría ser un servicio HTTP en desarrollo que "entró en pánico" sin devolver ningún dato.)

[← Volver a la lista de posts](#)

TangerineMess.

NUEVO DISEÑO
10 may 16:48

n muchos casos, es causado por un complemento del navegador (Ej, un bloqueador de anuncios o un protector de privacidad) que blokea la solicitud.

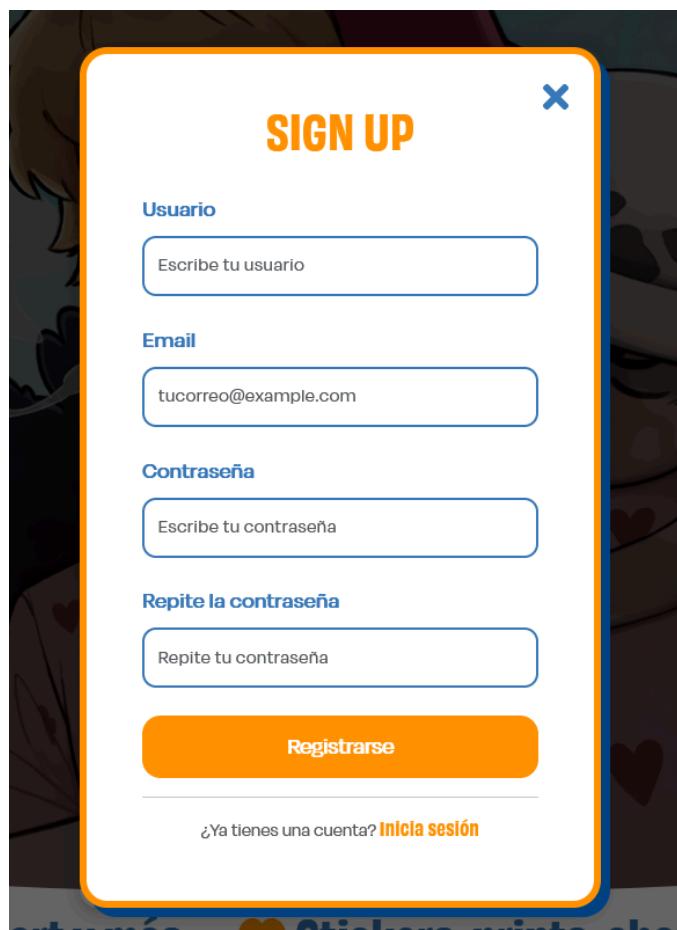
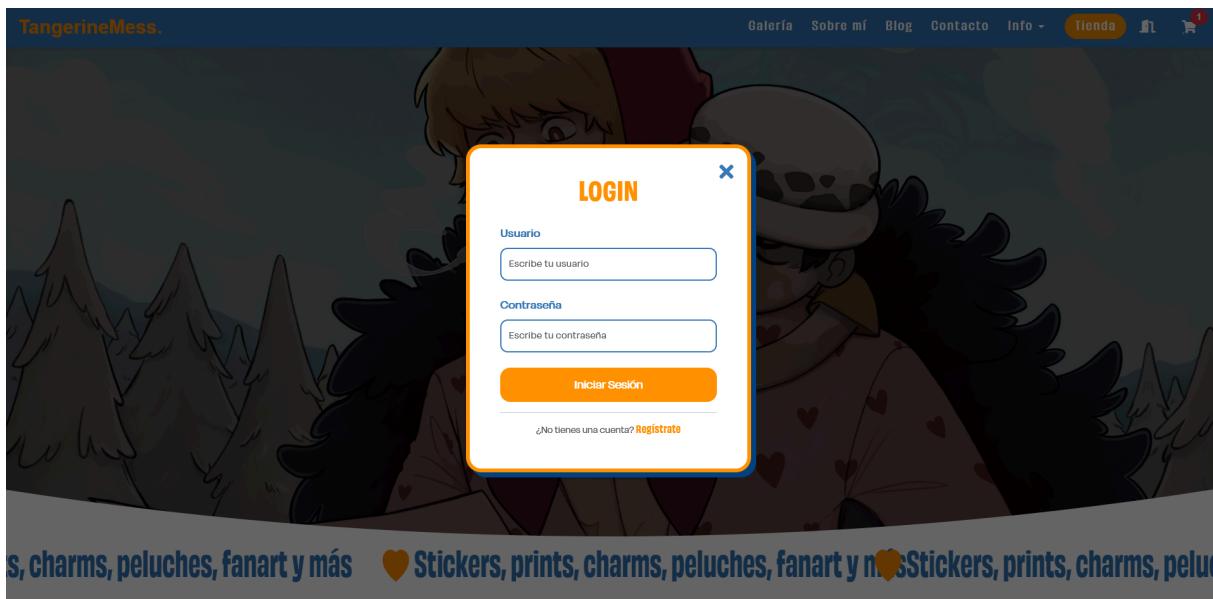
Otras causas posibles:

Intentar acceder a un recurso https que tenga un certificado no válido, causará este error.
 Intentar acceder a un recurso http desde una página con un origen https también causará este error.
 A partir de Firefox 68, las páginas https no pueden acceder a http://localhost, aunque esto puede ser modificado por el Error 1488740.
 El servidor no respondió a la solicitud actual (incluso si respondió la solicitud Preflight. Un escenario podría ser un servicio HTTP en desarrollo que "entró en pánico" sin devolver ningún dato.)

[← Volver a la lista de posts](#)

Login Modal

Fácil de abrir y cerrar, opaca el fondo de la página. Contiene texto en los inputs para especificar qué debe escribir el usuario.



Móvil:

TangerineMess.

LOGIN

Usuario

Contraseña

Iniciar Sesión

¿No tienes una cuenta? [Regístrate](#)

SIGN UP

Usuario

Email

Contraseña

Repite la contraseña

Registrarse

¿Ya tienes una cuenta? [Inicia sesión](#)

Mi cuenta

TangerineMess.

[Galería](#) [Sobre mí](#) [Blog](#) [Contacto](#) [Info](#) [Tienda](#)

Mi cuenta

[Salir](#)

Mis Pedidos

Pedido #13
Fecha: 2/05/2025
Estado: En proceso

Producto	Precio	Cantidad	Total
Ancient Hero and Zelda small keychains - Ancient Hero	6,00 €	1	6,00 €
My Chemical Romance Danger Days keychains and pins - Pistola de Ray	6,00 €	1	6,00 €
Aziraphale and Crowley enamel pin	12,00 €	1	12,00 €
Total productos			26,00 €
Coste envío			5,00 €
Total con envío			31,00 €

[Hola, papa](#)

Mis Pedidos

Pedido #13

Fecha: 21/05/2025

Estado: En proceso

Producto	Precio	Cantidad	Total
Ancient Hero and Zelda small keychains - Ancient Hero	6,00 €	1	6,00 €
My Chemical Romance Danger Days keychains and pins - Pistola de Ray	6,00 €	1	6,00 €
Aziraphale and Crowley enamel pin	12,00 €	1	12,00 €
Total productos	26,00 €		
Coste envío	5,00 €		
Total con envío	31,00 €		

Carrito:

Mensajes explican el estado del carrito si no hay añadidos

Tu carrito

Aún no tienes ningún producto en el carrito...

Tu carrito

	Link paraglider keychain Opción: N/A 8€	8,00€
	Do flamingo corazón peluches Opción: Cora 40€	40,00€

Resumen del pedido

Subtotal	48,00 €
Total	48,00 €

Ir a la caja

The screenshot shows a mobile application interface for a store named "TangerineMess". The top navigation bar is blue with the store name in orange. Below it, the title "Tu carrito" (Your Cart) is displayed in large orange text. The cart contains two items:

- Link paraglider keychain**: An image of a colorful paraglider keychain, labeled "Opción: N/A" and "8€". It includes quantity controls (-, +, 1) and a trash icon.
- Do flamingo corazón peluches**: An image of two plush flamingos, labeled "Opción: Cora" and "40€". It includes quantity controls (-, +, 1) and a trash icon.

Below the cart, a summary section titled "Resumen del pedido" (Order Summary) provides the total cost:

Subtotal	48.00 €
Total	48.00 €

A large orange button at the bottom right contains the text "→ Ir a la caja" (Go to the cash register).

Caja. Desglose de pedido

Caja

Producto	Precio	Cantidad	Total
Link paraglider keychain - N/A	8,00 €	1	8,00 €
Do flamingo corazón peluches - Cora	40,00 €	1	40,00 €
Total		2	48,00 €
Peso total (productos + opciones + sobre)			290 g
Coste envío			5,00 €
Total + Envío			53,00 €

Datos de envío

* Todos los campos son obligatorios

Nombre*	Dirección*
<input type="text"/>	<input type="text"/>
Apellido*	Código postal*
<input type="text"/>	<input type="text"/>
Correo electrónico*	Ciudad / Localidad*
<input type="text"/>	<input type="text"/>
Teléfono*	
<input type="text"/>	

TangerineMess.
≡

Apellido*

Correo electrónico*

Teléfono*

Dirección*

Código postal*

Ciudad / Localidad*

Número de tarjeta

MM / AA CVC

Pagar

La tienda

TangerineMess.

Galería Sobre mí Blog Contacto Info Tienda 2

Categorías

Todos los productos

Llaveros

Peluches

Prints

Pins

Textil

Standees

Pegatinas

¿Qué estás buscando?

ANCIENT HERO AND ZELDA SMALL KEYCHAINS 6.00 €	LINK PARAGLIDER KEYCHAIN 8.00 €	DO FLAMINGO CORAZÓN PELUCHES 40.00 €	ENAMEL PIN - 6 CM - 2 WHITE RUBBER CLUTCHES TRACKED SHIPPING INCLUDED 7.00 €
NIER EMIL ENAMEL PIN	AZIRAPHALE AND CROWLEY ENAMEL PIN	MY CHEMICAL ROMANCE DANGER DAYS KEYCHAINS AND PINS	OUR FLAG MEANS DEATH BLACKBONNET ENAMEL PIN

Animación en la que se amplia el tamaño de la tarjeta y animación de línea bajo el título.

Permite hacer búsquedas claras.



Resultados de la búsqueda

Término buscado: "link"



LINK PARAGLIDER KEYCHAIN
8.00 €

[← Ver todos los productos](#)



TangerineMess Peluche de Luffy

20.00€ EUR

Cantidad

[Añadir al carrito](#)

Some specifications:

- 25 cm tall
- vest and pants are detachable
- extra dressosa shirt for free!
- every plushie will come with a surprise freebie
- tracked shipping will be the only available option this time!!!

[↑ Volver a la lista de productos](#)

Vista móvil:

TangerineMess.

- [Peluches](#)
- [Prints](#)
- [Pins](#)
- [Textil](#)
- [Standees](#)
- [Pegatinas](#)

¿Qué estás buscando?

ANCIENT HERO AND ZELDA SMALL KEYCHAINS
6.00 €

TangerineMess.

TangerineMess
Ancient Hero and Zelda small keychains
6.00€ EUR

Opciones

Ancient Hero (6.00€)

Cantidad

[Añadir al carrito](#)

DEW

He utilizado **Vue.js** como framework principal para el desarrollo del frontend, aprovechando su estructura basada en componentes para construir una interfaz modular, clara y fácilmente mantenible. Cada funcionalidad está dividida en componentes reutilizables que encapsulan lógica, estilos y estructura, lo que facilita la escalabilidad y mejora la organización del código.

Durante el desarrollo, se contempló inicialmente la integración con **TypeScript** para aprovechar sus beneficios en cuanto a tipado estático y detección temprana de errores. No obstante, debido a problemas de compatibilidad al migrar componentes ya desarrollados en JavaScript y a restricciones de tiempo antes de la entrega, se optó por continuar con JavaScript puro para garantizar la estabilidad y funcionalidad del proyecto.

El proyecto hace un uso intensivo de **Pinia** como gestor de estado global, lo que permite centralizar y sincronizar datos críticos como la gestión del carrito de compras, el estado de autenticación y el control de carga, proporcionando una experiencia de usuario fluida y reactiva.

Entre las funcionalidades destacadas implementadas con Vue, se incluyen:

- Gestión dinámica de productos con opciones configurables y cantidades ajustables, integrando validaciones y actualización automática de precios.
- Visualización avanzada de imágenes mediante una galería modal con navegación, para mejorar la experiencia visual.
- Persistencia del estado crítico (carrito, sesión, usuario) mediante almacenamiento local (`localStorage`), lo que garantiza la continuidad de la sesión y los datos entre recargas o cierres del navegador.
- Uso de métodos y propiedades computadas para mantener la lógica de negocio desacoplada de la presentación y optimizar el rendimiento.

Como mejora futura, se plantea una migración progresiva a TypeScript, comenzando por los componentes y módulos más críticos y con mayor lógica,

para reforzar la robustez del código sin comprometer la base establecida.

Dependencias (`dependencies`)

- **vue** (`^3.2.13`)

Framework progresivo para construir interfaces de usuario. Versión 3, la más reciente y con Composition API.

- **vue-router** (`^4.0.3`)

Librería oficial para manejar rutas y navegación en aplicaciones Vue 3.

- **pinia** (`^3.0.2`)

Alternativa moderna y más ligera a Vuex para gestión de estado en Vue 3.

- **axios** (`^1.8.4`)

Cliente HTTP para hacer peticiones AJAX a APIs, muy popular y sencillo de usar.

- **aos** (`^2.3.4`)

Animate On Scroll, librería para animar elementos cuando aparecen al hacer scroll en la página.

- **core-js** (`^3.8.3`)

Biblioteca para polyfills de JavaScript moderno, asegurando compatibilidad con navegadores antiguos.

Conexión de vue al backend de Django:

Se realiza la conexión del frontend (Vue) con el backend (Django REST Framework) a través de la librería

`axios`. Las rutas protegidas están autenticadas con tokens JWT, y los datos de usuarios, productos y carrito son intercambiados entre cliente y servidor de forma segura y asincrónica.

Componentes y vistas

Este proyecto cuenta con una serie de componentes y vistas:

Vistas: About, Cart, Category, CategoryView, Checkout, Comissions, Contact, Events, FAQ, Home, LogIn, MyAccount, Portfolio, PostDetail, PostList, Product, Search, SignUp, Store, Success.

Componentes: AuthModals, BaseTitle, CardPost, CartItem, CategorySidebar, Marquee, Modal, Navbar, OrderSummary, ProductBox, ProductGrid, Socials, SolidButton.

A continuación hablaré de cada uno de ellos:

Navbar.vue (componente)

Este componente gestiona la barra de navegación, mostrando enlaces a las distintas vistas, el carrito y los controles de autenticación. Además, al montarse carga el token de la store y lo añade a los headers de Axios para que todas las llamadas al backend Django incluyan autenticación.

- **Composition API & SFC:** usa `<script setup>` con `ref`, `computed` y `onMounted`.

```
<script setup>
import { ref, computed, onMounted } from 'vue'
// ...
const showLogin = ref(false)
const cart = computed(() => mainStore.cart)
onMounted(() => {
  const token = mainStore.token
  axios.defaults.headers.common['Authorization'] = token
    ? 'Token ' + token
    : ''
})
</script>
```

- **Conexión a Django:** inyecta el token en Axios para llamadas al API de Django.

```
axios.defaults.headers.common['Authorization'] = 'Token ' + token
```

- **Pinia (composable):** importa y utiliza la store (`useMainStore`) para estado global (auth, carrito).

```
import { useMainStore } from './store/mainstore'
const mainStore = useMainStore()
```

- **Vue Router:** usa `useRouter()` y `<router-link>` para navegación SPA.

```
<router-link class="navbar-brand" to="/">TangerineMess.</router-link>
```

- **Directivas:** muestra/oculta elementos según `isAuthenticated` con `v-if`, controla clics con `@click`.

```
<template v-if="isAuthenticated">
  <router-link ...>Mi cuenta</router-link>
</template>
<button @click="showLogin = true">...</button>
```

Home.vue (vista)

Pantalla de portada con un “hero” a toda pantalla, un título y dos componentes hijos (`Marquee` y `Socials`). Define el título de la pestaña del navegador y usa SVG para una curva decorativa.

- **SFC:** todo en un `.vue` con `<template>`, `<script>` y `<style scoped>`.
- **Components:** importa y registra `Marquee` y `Socials`.

```
<script>
import Marquee from '@/components/Marquee.vue'
import Socials from '@/components/Socials.vue'
document.title = 'Home | TangerineMess'
export default { components: { Marquee, Socials } }
</script>
```

- **Estilos scoped:** aplica estilos locales al hero y al texto con sombras CSS.

Marquee.vue (componente)

Genera una cinta de texto que se desplaza indefinidamente, adaptando la duración de la animación al ancho del contenido.

- **SFC & Options API:** usa `mounted` y `beforeUnmount` para hooks de ciclo de vida, y `methods` para lógica.
- **Refs con `$refs`:** accede directamente a los nodos DOM para medir ancho y posicionar el segundo bloque.

```
mounted() {
  this.adjustAnimation()
  window.addEventListener('resize', this.adjustAnimation)
},
methods: {
  adjustAnimation() {
    const width = this.$refs.marquee1.offsetWidth
    // ...
    this.$refs.marquee2.style.left = `${width}px`
  }
}
```

- **Animación dinámica:** inyecta un bloque `<style>` en el `<head>` con `@keyframes` que trasladan el texto según el ancho calculado.

Login.vue

Este componente implementa la funcionalidad de inicio de sesión para la aplicación. Utiliza la Options API de Vue para manejar el estado local y métodos. Cuando el usuario envía el formulario, se hace una petición POST al endpoint `/api/v1/token/login/` de la API Django REST para autenticar. Si la autenticación es exitosa, se recibe un token que se almacena en `localStorage` y en un store global con Pinia (`useMainStore`). Además, el token se configura en los headers de Axios para futuras peticiones autenticadas. Se redirige al usuario a la ruta `/store`. En caso de error, se muestran los mensajes recibidos desde el backend.

Se usan datos reactivos para el formulario (`username`, `password`) y para almacenar errores que se presentan en la UI. La interacción con el store global permite centralizar el estado del usuario.

Fragmento clave del método que envía el formulario:

```
async submitForm() {
  this.errors = []
```

```

axios.defaults.headers.common["Authorization"] = ""
localStorage.removeItem("token")

const formData = { username: this.username, password: this.password }
try {
  const response = await axios.post("/api/v1/token/login/", formData)
  const token = response.data.auth_token

  localStorage.setItem("username", this.username)
  localStorage.setItem("token", token)
  this.mainStore.setUsername(this.username)
  this.mainStore.setToken(token)
  axios.defaults.headers.common["Authorization"] = "Token " + token
  this.mainStore.initializeStore()

  this.$router.push('/store')
} catch (error) {
  if (error.response) {
    for (const property in error.response.data) {
      this.errors.push(`${property}: ${error.response.data[property]}`)
    }
  } else {
    this.errors.push('Something went wrong. Please try again')
  }
}
}

```

SignUp.vue

Este componente permite registrar un nuevo usuario. Incluye un formulario con campos para usuario, email, contraseña y confirmación de contraseña. Antes de enviar, se valida en el cliente que no falten campos y que las contraseñas coincidan, mostrando errores si es necesario.

Al enviar el formulario, se realiza una petición POST al endpoint `/api/v1/users/` para crear el usuario en el backend. Si la petición es exitosa, se emite un evento hacia el componente padre para indicar que el registro fue exitoso y cambiar la vista a login. En caso de errores de validación o de servidor, los mensajes se muestran en pantalla.

Fragmento del método que valida y envía los datos:

```
async submitForm() {
  this.errors = []

  if (!this.username) this.errors.push('El nombre de usuario es obligatorio')
  if (!this.email) this.errors.push('El email es obligatorio')
  if (!this.password) this.errors.push('La contraseña es obligatoria')
  if (this.password !== this.password2) this.errors.push('Las contraseñas no coinciden')

  if (this.errors.length) return

  try {
    await axios.post('/api/v1/users/', {
      username: this.username,
      email: this.email,
      password: this.password
    })
    this.$emit('signup-success')
  } catch (error) {
    if (error.response) {
      for (const property in error.response.data) {
        this.errors.push(`${property}: ${error.response.data[property]}`)
      }
    } else {
      this.errors.push('Algo salió mal. Inténtalo de nuevo.')
    }
  }
}
```

Modal.vue

Un componente modal reutilizable que sirve como contenedor para mostrar contenido sobre una capa oscura en la pantalla. Recibe una propiedad booleana `show` que controla la visibilidad. Cuando el usuario hace click fuera del contenido (overlay) o en el botón de cerrar, se emite un evento `close` para que el componente padre controle el estado y oculte el modal.

Usa un slot para permitir que cualquier contenido dinámico se inserte dentro del modal, por ejemplo, los formularios de login y signup. El modal está estilizado con bordes redondeados, sombra y animación de aparición.

Código clave:

```
<template>
  <div v-if="show" class="modal-overlay" @click.self="close">
    <div class="modal-content cartoon-card animate-pop">
      <button class="modal-close" @click="close"><i class="fas fa-times"></i></button>
      <slot></slot>
    </div>
  </div>
</template>

<script>
export default {
  name: 'Modal',
  props: {
    show: { type: Boolean, required: true }
  },
  emits: ['close'],
  methods: {
    close() {
      this.$emit('close')
    }
  }
}
</script>
```

AuthModals.vue

Este componente es el encargado de controlar cuál modal se muestra: el de login o el de registro (signup). Mantiene un estado local `showSignUp` que indica qué modal debe estar visible. Importa y usa los componentes `Modal`, `Login` y `SignUp`.

Cuando se quiere cambiar entre login y signup, se invocan métodos que actualizan `showSignUp`. Además, maneja el evento de registro exitoso para cerrar el modal de signup, abrir el de login y mostrar un toast con Bootstrap para notificar al usuario que la cuenta fue creada.

El componente también incluye el contenedor para los toasts de Bootstrap, y genera dinámicamente los toasts para los mensajes.

Fragmento que controla el estado y muestra los modales:

```
data() {
  return {
    showSignUp: false
  },
},
methods: {
  switchToSignUp() {
    this.showSignUp = true
  },
  switchToLogin() {
    this.showSignUp = false
  },
  handleSignupSuccess() {
    this.showSignUp = false
    this.showBootstrapToast('¡Cuenta creada! Ahora inicia sesión.', 'success')
  },
  showBootstrapToast(message, type = 'success') {
    const toastContainer = document.getElementById('toast-container')
    if (!toastContainer) return

    const toastEl = document.createElement('div')
    toastEl.className = `toast align-items-center text-white bg-primary border-0`
    toastEl.style.minWidth = '300px'
    toastEl.style.fontSize = '1.1rem'

    toastEl.setAttribute('role', 'alert')
    toastEl.setAttribute('aria-live', 'assertive')
    toastEl.setAttribute('aria-atomic', 'true')
```

```

toastEl.innerHTML = `
  <div class="d-flex">
    <div class="toast-body fs-5">${message}</div>
    <button type="button" class="btn-close btn-close-white me-2 m-auto" data-bs-dismiss="toast" aria-label="Close"></button>
  </div>
` 

toastContainer.appendChild(toastEl)
const bsToast = new bootstrap.Toast(toastEl, { delay: 3000 })
bsToast.show()
toastEl.addEventListener('hidden.bs.toast', () => { toastEl.remove() })
}
}

```

Store.vue

Este componente es la página principal de la tienda. Su estructura HTML está dividida en un sidebar con las categorías y un área principal con la búsqueda y el listado de productos.

- Usa un componente `CategoriesSidebar` para mostrar la lista de categorías disponibles, destacando la categoría activa.
- Tiene un formulario de búsqueda que redirige a la ruta `/search` enviando el término en query string.
- Los productos más recientes se cargan desde la API (`/api/v1/latest-products/`) al montar el componente, almacenándose en `latestProducts`.
- Usa el componente `ProductsGrid` para renderizar la grilla de productos.
- Además, usa un store global `useMainStore` para manejar el estado de carga (`isLoading`) mientras se obtiene la información.

Fragmento clave de la llamada a la API para obtener productos recientes:

```

async fetchLatestProducts() {
  const store = useMainStore()
  store.setIsLoading(true)

  try {
    const res = await axios.get("/api/v1/latest-products/")
  }
}

```

```

    this.latestProducts = res.data
  } catch (error) {
    console.error(error)
  }

  store.setIsLoading(false)
}

```

El CSS se encarga de estilos responsivos para el formulario de búsqueda y la disposición general, con colores personalizados.

CategoriesSidebar.vue

Este componente es una barra lateral que lista las categorías de productos disponibles para filtrar.

- Recibe por prop `activeCategorySlug` para saber qué categoría está activa y aplicar estilos especiales.
- Usa enlaces `router-link` para navegar entre categorías, cada uno con una ruta distinta.
- Tiene estilos personalizados para los enlaces, con efecto hover y resaltado para la categoría activa.
- Emplea el componente `BaseTitle` para mostrar un título estilizado "Categorías".
- El método `isActive(slug)` devuelve true si el slug coincide con la categoría activa, para aplicar la clase CSS `active`.

Fragmento del método que determina si una categoría está activa:

```

methods: {
  isActive(slug) {
    return this.activeCategorySlug === slug;
  }
}

```

Estilísticamente destaca la usabilidad con colores, transición y tipografía custom.

Search.vue

Este componente muestra los resultados de búsqueda de productos basados en el término que el usuario ingresó.

- Obtiene el parámetro de búsqueda desde la URL con `useRoute` y lo almacena en una variable reactiva `query`.
- Cada vez que cambia `query` o al montar el componente, se hace una petición POST al backend (`/api/v1/products/search/`) enviando el término de búsqueda.
- Los productos encontrados se almacenan en `products`.
- Si no hay productos, se muestra un mensaje de aviso.
- Se usan componentes `ProductBox` para mostrar cada producto, `BaseTitle` para el encabezado y `SolidButton` para un botón que vuelve a la tienda completa.

Fragmento que realiza la búsqueda:

```
async function performSearch() {
  try {
    const response = await axios.post('/api/v1/products/search/', { query: query.value })
    products.value = response.data
  } catch (error) {
    console.error(error)
  }
}
```

Además, hay un watcher que responde a cambios en la `query` para actualizar los resultados sin recargar.

ProductBox.vue

Componente que representa una tarjeta individual de producto.

- Muestra la imagen principal, el nombre y el precio del producto.
- El nombre y la imagen son enlaces que llevan a la página del producto (usando `product.get_absolute_url`).
- La imagen tiene un efecto de zoom al pasar el mouse.

- La tarjeta tiene efectos visuales para mejorar la experiencia, como sombra y desplazamiento al hacer hover.
- Estilizado con colores cálidos y tipografías definidas para mantener el estilo visual del sitio.

Fragmento de la estructura principal:

```
<div class="card h-100 border-0 product-card">
  <router-link :to="product.get_absolute_url" class="image-link">
    
  </router-link>
  <div class="card-body d-flex flex-column px-2 py-2">
    <router-link :to="product.get_absolute_url" class="product-name text-up
percase fw-bold mb-2 mt-2">
      {{ product.name }}
    </router-link>
    <p class="product-price mt-auto mb-3">{{ product.price }} €</p>
  </div>
</div>
```

ProductsGrid.vue

Este componente es un contenedor que recibe un array de productos y renderiza una grilla con tarjetas `ProductBox`.

- Recibe por prop `products` la lista de productos.
- Usa un `v-for` para iterar y renderizar cada producto.
- Facilita la reutilización y organización, permitiendo mostrar grillas de productos en cualquier parte del proyecto.

Fragmento de la plantilla:

```
<div class="row g-4 container-custom">
  <ProductBox
    v-for="product in products"
    :key="product.id"
    :product="product"
```

```
/>  
</div>
```

No tiene lógica extra, solo la estructura y la importación de `ProductBox`.

Category

Este componente es como la ventana que muestra los productos de una categoría específica. Cuando entras, hace una llamada al backend con Axios para traer la info: el nombre de la categoría y sus productos.

- Usa `useMainStore` (Pinia) para manejar un loading que le avisa al usuario que está cargando datos.
- Escucha cuando cambias de categoría en la URL y automáticamente recarga los datos.
- También actualiza el título de la pestaña del navegador con el nombre de la categoría.

Un pedacito importante:

```
async getCategory() {  
  const categorySlug = this.$route.params.category_slug  
  store.setIsLoading(true)  
  try {  
    const response = await axios.get(`/api/v1/products/${categorySlug}`)  
    this.category = response.data  
    document.title = this.category.name + ' | TangerinMess'  
  } catch (error) {  
    toast({ message: 'Algo salió mal, intenta de nuevo.', type: 'is-danger' })  
  }  
  store.setIsLoading(false)  
}
```

En resumen: cuando abres una categoría, este componente se encarga de mostrar el título y una lista de productos con un componente `ProductBox` que recibe cada producto como prop.

CategoryView

Esta vista es como la versión más completa y pulida del `Category`. Aquí tienes el sidebar con todas las categorías para navegar, un buscador para filtrar productos y el listado de productos en un grid.

- El sidebar se hace con un componente `CategoriesSidebar` que recibe todas las categorías.
- El buscador está preparado para enviar la consulta a la ruta `/search`.
- Si la categoría no tiene productos, muestra un mensaje amigable avisando que no hay productos.
- Igual que en el otro, hace la llamada con Axios para traer la categoría y productos y actualiza el título.

Un fragmento clave del template para el buscador y productos:

```
<form method="get" action="/search" class="mb-4 search-form ms-aut  
o">  
  <div class="input-group">  
    <input type="text" class="form-control" placeholder="¿Qué estás busca  
ndo?" name="query" />  
    <button class="btn btn-custom" type="submit">  
      <i class="fas fa-search"></i>  
    </button>  
  </div>  
</form>  
  
<div class="row row-cols-1 row-cols-md-2 row-cols-lg-3 row-cols-xl-4 g-  
4">  
  <ProductBox  
    v-for="product in category.products"  
    :key="product.id"  
    :product="product"  
  />  
  </div>  
  
<div v-if="category.products.length === 0" class="alert alert-warning text-  
center mt-4">  
  No hay productos disponibles en esta categoría.  
</div>
```

Este componente te da la experiencia completa para navegar, buscar y visualizar productos en una categoría, con estilos responsivos y sidebar para facilitar la navegación.

Contact

Aquí tienes el formulario para que los usuarios puedan enviarte mensajes o pedir colaboraciones. La magia es que todo se maneja con Vue: el formulario está ligado con `v-model` para capturar cada dato, y cuando envías, hace un POST a la API con Axios.

- Si el envío es exitoso, muestra un toast verde con el mensaje de éxito.
- Si falla, muestra un toast rojo con el error.
- También tienes enlaces a redes sociales con iconos muy visibles para que te encuentren fácilmente.
- El formulario es súper sencillo, con validaciones básicas (todos los campos son requeridos).

Este método muestra cómo se hace el envío y el manejo de los mensajes:

```
async handleSubmit() {
  try {
    const response = await axios.post('http://127.0.0.1:8000/api/v1/contact/',
    this.formData)
    this.message = response.data.message || '¡Mensaje enviado correctamente!'
    this.messageType = 'success'
    this.resetForm()
  } catch (error) {
    this.message = 'Error al enviar. Intenta de nuevo.'
    this.messageType = 'error'
  }
  this.showToast = true
  setTimeout(() => { this.showToast = false }, 4000)
},
```

En pocas palabras: este componente da una experiencia limpia y sencilla para que cualquier persona pueda contactarte fácilmente, con mensajes claros de confirmación o error.

Claro, aquí te dejo una documentación más personal, explicando qué hace cada componente (Cart, CartItem y Checkout) con fragmentos relevantes de código para que lo entiendas bien y puedas usarlo o modificarlo con confianza.

Cart.vue

Este es el contenedor principal del carrito de compras. Su función es mostrar todos los productos que el usuario ha agregado, mostrar el resumen del pedido con el subtotal y total, y ofrecer un botón para ir al checkout.

- Muestra una lista de productos con el componente `CartItem`.
- Calcula y muestra el total del pedido.
- Permite eliminar productos del carrito usando el método `removeFromCart` del store.
- Si el carrito está vacío, muestra un mensaje informativo.

```
<CartItem
  v-for="item in cart.items"
  :key="item.product.id"
  :item="item"
  @removeFromCart="removeFromCart"
/>

<div class="summary-row">
  <span>Subtotal</span>
  <span>{{ cartTotalPrice.toFixed(2) }} €</span>
</div>

<SolidButton
  text="Ir a la caja"
  iconClass="fas fa-arrow-right"
  routeTo="/cart/checkout"
/>
```

CartItem.vue

Este componente representa cada producto dentro del carrito. Se encarga de mostrar detalles del producto, su imagen, precio, opción seleccionada, cantidad y el total de ese producto. Además, tiene controles para incrementar o decrementar la cantidad, y eliminar el producto.

- Muestra información clave del producto.
- Permite cambiar la cantidad con botones + y -.
- Elimina el producto del carrito si la cantidad baja a cero o si el usuario lo elimina explícitamente.
- Actualiza el localStorage para mantener la persistencia del carrito.

```
<button class="qty-btn" @click="decrementQuantity(item)" :disabled="item.quantity <= 1">-</button>
<span class="quantity">{{ item.quantity }}</span>
<button class="qty-btn" @click="incrementQuantity(item)">+</button>

<span class="total">{{ getItemTotal(item).toFixed(2) }}€</span>

function incrementQuantity(item) {
  item.quantity += 1
  mainStore.updateLocalStorageCart()
}

function decrementQuantity(item) {
  if (item.quantity > 1) {
    item.quantity -= 1
    mainStore.updateLocalStorageCart()
  } else {
    removeFromCart(item)
  }
}
```

Este componente es muy útil porque permite al usuario controlar exactamente qué quiere comprar, y cuántos. Además, mantiene todo sincronizado con el store y el almacenamiento local, asegurando que los datos no se pierdan.

Checkout.vue

Esta es la página de pago, donde el usuario introduce sus datos de envío y puede revisar el pedido antes de confirmar el pago.

- Muestra un resumen detallado del pedido, con productos, cantidades, precios, peso y coste de envío.
- Calcula el peso total con el sobre y ajusta el coste de envío según ese peso.
- Recoge la información del usuario (nombre, email, dirección, etc.) para el envío.
- Valida que todos los campos estén completos.
- Integra Stripe para gestionar el pago con tarjeta.

```
<tbody>
  <tr v-for="item in cart.items" :key="item.product.id">
    <td>{{ item.product.name }} <span class="option-highlight"> - {{ item.product.selectedOption?.name || 'N/A' }}</span></td>
    <td class="price-text">{{ formatPrice(item.product.price) }}</td>
    <td>{{ item.quantity }}</td>
    <td class="price-text">{{ formatPrice(getItemTotal(item)) }}</td>
  </tr>
</tbody>

const totalWeightWithEnvelope = computed(() => {
  const baseEnvelopeWeight = 30
  const totalProductWeight = store.cart.items.reduce((acc, item) => {
    return acc + item.quantity * (item.product.totalWeight || 0)
  }, 0)
  return baseEnvelopeWeight + totalProductWeight
})

const shippingCost = computed(() => {
  const weight = totalWeightWithEnvelope.value
  if (weight <= 500) return 5
  else if (weight <= 1000) return 8
```

```

    return 12
  })

function submitForm() {
  errors.value = []

  if (!first_name.value) errors.value.push('The first name field is missing!')
  // ... validations para otros campos

  if (!errors.value.length) {
    store.setIsLoading(true)

    stripe.value.createToken(card.value).then(result => {
      if (result.error) {
        store.setIsLoading(false)
        errors.value.push('Something went wrong with Stripe. Please try again')
      } else {
        stripeTokenHandler(result.token)
      }
    })
  }
}

```

Esta pantalla es fundamental para cerrar la compra. Combina la validación del formulario, cálculo de costes y la integración con Stripe para que todo el proceso sea seguro y sencillo para el usuario.

```

function incrementQuantity(item) {
  item.quantity += 1
  mainStore.updateLocalStorageCart()
}

function decrementQuantity(item) {
  if (item.quantity > 1) {
    item.quantity -= 1
    mainStore.updateLocalStorageCart()
  } else {

```

```
    removeFromCart(item)
  }
}
```

Este código permite aumentar o disminuir la cantidad del producto y asegura que los datos se mantengan guardados para que el usuario no pierda el carrito al recargar la página.

MyAccount

Este componente Vue usa SFC y mezcla Options API con Composition API para acceder a un store Pinia (`useMainStore`). Se conecta al backend mediante llamadas axios para obtener los pedidos y maneja la sesión usando localStorage y Vue Router para la navegación.

- **Conexión al backend (Django REST API):**

Se realiza una llamada a la API para obtener pedidos:

```
async getMyOrders() {
  const response = await axios.get('/api/v1/orders/')
  this.orders = response.data
}
```

- **Uso parcial de Composition API y SFC:**

El componente es un SFC y usa `setup()` para Pinia:

```
setup() {
  const mainStore = useMainStore()
  return { mainStore }
}
```

- **Uso de directivas Vue:**

En template se usan `v-if`, `v-for` y eventos con `@click`:

```
<button @click="logout">Salir</button>
<div v-for="order in orders" :key="order.id">...</div>
<p v-if="username">Hola, {{ username }}</p>
```

- **Uso de Pinia (store) para estado global:**

```
import { useMainStore } from '../store/mainstore'
```

- **Uso de localStorage para mantener sesión y logout:**

```
logout() {
  localStorage.removeItem("token")
  localStorage.removeItem("username")
  localStorage.removeItem("userid")
}
```

- **Uso de Vue Router para redireccionar tras logout:**

```
this.$router.push('/')
```

OrderSummary

Este componente Vue recibe una prop `order` y muestra los detalles del pedido. Está en formato SFC y usa solo Options API con propiedades computadas para cálculos y formato.

- **Uso de props:**

Recibe la orden como prop:

```
props: {
  order: Object
}
```

- **Uso de computed properties:**

Para formatear fecha y calcular totales:

```
computed: {
  formattedOrderDate() {
    const date = new Date(this.order.created_at)
    return date.toLocaleDateString('es-ES')
  },
}
```

```

orderTotalPrice() {
  return this.order.items.reduce((total, item) => total + Number(item.price) * item.quantity, 0)
}

```

- **Uso de métodos para cálculos y formateo:**

```

methods: {
  getItemTotal(item) {
    return item.quantity * item.product.price
  },
  formatPrice(value) {
    return new Intl.NumberFormat('es-ES', { style: 'currency', currency: 'EUR' }).format(value)
  }
}

```

- **Uso de directivas Vue:**

`v-for` para listar productos:

```
<tr v-for="item in order.items" :key="item.product.id">
```

Success

Componente simple SFC que muestra mensaje estático. Usa Options API y Vue Router para cambiar título en `mounted`. Se usa para redirigir a esta vista cuando se confirma una compra.

- **Uso de lifecycle hook para document title:**

```

mounted() {
  document.title = 'Success'
}

```

BaseTitle

Este componente es un título reutilizable y personalizable para mostrar encabezados en la aplicación.

- **Qué hace:**

Renderiza un `<h2>` con texto dinámico recibido por props y permite personalizar el color, la fuente y el tamaño del texto mediante props también. Esto facilita que el mismo componente pueda usarse en distintos contextos con estilos diferentes sin modificar el componente base.

- **Características importantes:**

- Recibe las props `text` (obligatoria), `color`, `font` y `size`.
- Usa binding dinámico en el atributo `style` para aplicar los estilos recibidos.
- Está estructurado como un Single File Component (SFC) típico en Vue.

- **Código clave:**

```
props: {
  text: { type: String, required: true },
  color: { type: String, default: 'var(--color-secondary)' },
  font: { type: String, default: "'ObviouslyNarrow', sans-serif" },
  size: { type: String, default: '3.6rem' }
}
```

- **Uso del binding dinámico:**

```
<h2 :style="{ color: color, fontFamily: font, fontSize: size }">{{ text }}</h2>
```

CardPost

Este componente representa una tarjeta que muestra un post o artículo de blog con su título, extracto, fecha, género y una imagen.

- **Qué hace:**

Muestra la información básica de un post en un formato visual atractivo, con imagen, fecha resaltada y un enlace para ir al post completo. Al pasar el cursor sobre la tarjeta, la imagen y el contenido cambian suavemente, mejorando la experiencia visual.

- **Características importantes:**

- Recibe varias props para su contenido (`title`, `excerpt`, `date`, `genre`, `image`, `slug`).
- Utiliza **computed properties** para:
 - Separar la fecha en día, mes y año (`dateParts`).
 - Mostrar una imagen por defecto si no se pasa ninguna (`imageToShow`).
- Usa `<router-link>` para enlazar a la página del post, integrándose con Vue Router para navegación SPA.
- Tiene efectos visuales en hover mediante CSS para mejorar la interacción.

- **Código clave:**

```
props: {
  title: String,
  excerpt: String,
  date: String,
  genre: String,
  image: String,
  slug: { type: String, required: true }
},
computed: {
  dateParts() { return this.date.split(' '); },
  imageToShow() { return this.image || 'https://via.placeholder.com/300x20
0?text=No+Image'; }
}
```

- **Enlace dinámico con router:**

```
<router-link :to={`/blog/posts/${slug}`} class="read-more">Leer más</rout
er-link>
```

SocialLinks

Componente simple que muestra iconos sociales con enlaces a diferentes plataformas.

- **Qué hace:**

Renderiza una barra de iconos que enlazan a redes sociales específicas. Es un componente estático sin lógica compleja, ideal para colocarse en pies de página o secciones de contacto.

- **Características importantes:**

- Usa etiquetas `<a>` con atributos para accesibilidad (`aria-label`) y seguridad (`rel="noopener"`).
- Contiene iconos con clases de FontAwesome.
- Está estructurado como un SFC básico sin props ni estados.

- **Código clave:**

```
<a href="https://www.instagram.com/..." target="_blank" rel="noopener" aria-label="Instagram">
  <i class="fab fa-instagram"></i>
</a>
```

SolidButton

Botón reutilizable con texto, ícono y acción para volver o navegar a una ruta específica.

- **Qué hace:**

Muestra un botón con un ícono (por defecto una flecha hacia la izquierda) y un texto. Al hacer clic, navega programáticamente a la ruta especificada (por defecto `/posts`). Es útil para acciones de navegación dentro de la app.

- **Características importantes:**

- Recibe las props `text`, `iconClass` y `routeTo` para personalización.
- Usa el método `goBack` que utiliza Vue Router (`this.$router.push`) para cambiar de ruta sin recargar la página.
- Tiene estilos para efectos hover y desplazamiento del ícono.

- **Código clave:**

```
methods: {
  goBack() {
```

```
    this.$router.push(this.routeTo);
}
}
```

- **Uso de evento en template:**

```
<button @click="goBack">
  <i :class="iconClass"></i>
  <span>{{ text }}</span>
</button>
```

About

Página que muestra información "Sobre mí" obtenida desde un backend.

- **Qué hace:**

Al montarse, hace una petición HTTP usando axios para obtener contenido de la sección "About" desde una API REST. Si hay contenido, lo muestra con texto e imagen. Si no, muestra un mensaje de alerta.

- **Características importantes:**

- Usa el ciclo de vida `mounted()` para hacer la llamada axios.
- Guarda el contenido en `data` y actualiza el DOM con `v-if` para condicionar la presentación.
- Cambia el título del documento para SEO y usabilidad.
- Método para convertir rutas relativas de imágenes a URLs absolutas.

- **Código clave:**

```
async getAboutContent() {
  try {
    const response = await axios.get('/api/v1/about/');
    this.aboutContent = response.data;
  } catch (error) {
    console.error('Error fetching about content:', error);
  }
}
```

- **Condicional en template:**

```
<div v-if="aboutContent">
  <p>{{ aboutContent.content }}</p>
  
</div>
<div v-else>
  <p>About content is not available.</p>
</div>
```

Commissions

Página que muestra una lista de comisiones disponibles, obtenidas desde backend.

- **Qué hace:**

Hace una petición HTTP para obtener una lista de comisiones, luego las renderiza en una lista con imagen, descripción, precio y slots disponibles. Usa diseño responsive y alterna el orden de imagen/texto en cada elemento para variedad visual.

- **Características importantes:**

- Usa `axios` para cargar datos desde API en `mounted()`.
- Usa `v-for` para iterar dinámicamente las comisiones.
- Utiliza `v-html` para renderizar contenido HTML seguro dentro de la descripción.
- Cambia el orden de elementos con clases dinámicas para que la presentación sea más atractiva.
- Método para generar URLs absolutas de imágenes.

- **Código clave:**

```
async getCommissions() {
  try {
    const response = await axios.get('/api/v1/commissions/');
    this.commissions = response.data;
  } catch (error) {
```

```
        console.error('Error fetching commissions:', error);
    }
}
```

- **Iteración con v-for y clases dinámicas:**

```
<div v-for="(commission, index) in commissions" :key="commission.id" :class="{ 'flex-row-reverse': index % 2 === 1 }">
    <!-- contenido -->
</div>
```

EventsView.vue

Este componente Vue muestra una lista de próximos eventos obtenidos desde un backend (API REST de Django) usando Axios para hacer la petición.

Se estructura como un **Single File Component (SFC)** con su template, script y estilos encapsulado

- **Carga de eventos al montar el componente:**

En el ciclo de vida `mounted()`, se llama al método `getEvents()` que hace una petición GET y actualiza el array reactivo `events`:

```
mounted() {
    this.getEvents();
    document.title = 'Upcoming Events';
},
methods: {
    async getEvents() {
        try {
            const response = await axios.get('/api/v1/events/');
            this.events = response.data;
        } catch (error) {
            console.error('Error fetching events:', error);
        }
    },
    ...
}
```

```
}
```

- **Renderizado dinámico con directivas:**

Se usa `v-if` para mostrar una alerta cuando no hay eventos:

```
v v-if="events.length === 0" class="alert alert-warning" role="alert">>
  No hay futuros eventos actualmente.
</div>
```

Y `v-for` para listar eventos en tarjetas:

```
or="event in events" :key="event.id" class="col-md-6 mb-5">
  <div class="card shadow border-0 overflow-hidden flex-md-row h-100">
    
    <div class="p-4 d-flex flex-column justify-content-between flex-grow-
1">
      <h3 class="event-title">{{ event.title }}</h3>
      <p class="event-location mb-3">{{ event.location }}</p>
    </div>
  </div>
</div>
```

- **Métodos para formateo y construcción de URLs:**

```
methods: {
  getImageUrl(imagePath) {
    return `http://localhost:8000${imagePath}`;
  },
  formatShortDayMonth(dateStr) {
    const date = new Date(dateStr);
    return date.toLocaleDateString('es-ES', { day: '2-digit', month: 'short' });
  },
  getYear(dateStr) {
    return new Date(dateStr).getFullYear();
```

```
},  
...  
}
```

FAQView.vue

Este componente muestra una lista de preguntas frecuentes (FAQ) obtenidas desde un backend Django mediante Axios.

Se estructura como un **Single File Component (SFC)** y utiliza las funcionalidades básicas de Vue para mostrar/ocultar respuestas con interacción del usuario.

Funcionalidades principales

- **Carga de FAQs al montar el componente:**

En `mounted()`, se obtiene la lista de preguntas frecuentes desde la API y se inicializa cada ítem con la propiedad reactiva `isOpen` para controlar la visibilidad de la respuesta.

```
async getFAQ() {  
  try {  
    const response = await axios.get('/api/v1/faq/');  
    this.faq = response.data.map(item => ({ ...item, isOpen: false }));  
  } catch (error) {  
    console.error('Error fetching FAQs:', error);  
  }  
},  
mounted() {  
  this.getFAQ();  
  document.title = 'Preguntas Frecuentes';  
},
```

- **Toggle para mostrar/ocultar respuestas:**

Al hacer clic en una pregunta, se cambia el estado `isOpen` del ítem correspondiente para mostrar u ocultar la respuesta.

```
toggleAnswer(id) {  
  const selectedItem = this.faq.find(item => item.id === id);  
  if (selectedItem) {  
    selectedItem.isOpen = !selectedItem.isOpen;  
  }  
}
```

- **Uso de directivas y bindings:**

- `v-for` para listar las preguntas
- `v-if` para mostrar la respuesta solo si está abierta
- `@click` para manejar la interacción
- `:class` para cambiar el estilo según el estado

```
<div  
  v-for="item in faq"  
  :key="item.id"  
  class="faq-item p-3 mb-3 rounded"  
  @click="toggleAnswer(item.id)"  
  :class="{ 'open': item.isOpen }"  
>  
  <div class="d-flex justify-content-between align-items-center faq-questi  
on">  
    <h4 class="mb-0">{{ item.question }}</h4>  
    <i :class="['fas', item.isOpen ? 'fa-chevron-up' : 'fa-chevron-down']" cla  
ss="faq-icon"></i>  
  </div>  
  <div v-if="item.isOpen" class="faq-answer mt-2">  
    {{ item.answer }}  
  </div>  
</div>
```

PostsView.vue

Este componente muestra una lista filtrable de posts obtenidos desde un backend Django usando Axios.

Se utiliza un dropdown para seleccionar la categoría y filtrar los posts mostrados. Los posts se visualizan en tarjetas mediante el componente hijo `CardPost`.

Funcionalidades principales

- **Carga inicial de posts:**

Al montar el componente, se cargan todos los posts disponibles desde la API.

```
async getPosts() {
  try {
    const response = await axios.get('/api/v1/posts/');
    this.posts = response.data;
    this.filteredPosts = this.posts;
  } catch (error) {
    console.error('Error fetching posts:', error);
  }
},
mounted() {
  this.getPosts();
  document.title = 'Posts | TangerineMess';
},
```

- **Filtrado por categoría:**

Al seleccionar una categoría en el dropdown, se consulta la API con un parámetro para filtrar posts y actualizar la lista visible.

```
async filterByCategory() {
  try {
    const response = await axios.get('/api/v1/posts/', {
      params: { category: this.selectedCategory },
    });
    this.filteredPosts = response.data;
  } catch (error) {
```

```

        console.error('Error fetching filtered posts:', error);
    }
},
};

selectCategory(code) {
    if (this.selectedCategory !== code) {
        this.selectedCategory = code;
        this.filterByCategory();
    }
},

```

- **Renderizado con `v-for` y bindings:**

Se muestran tarjetas para cada post filtrado, pasando props con información relevante.

```

<div class="row g-4 mt-5">
    <div class="col-12 col-md-6 col-lg-4" v-for="post in filteredPosts" :key
        ="post.id">
        <CardPost
            :title="post.title"
            :excerpt="post.summary"
            :date="formatDate(post.created_at)"
            :genre="post.category_display"
            :image="getImageUrl(post.images?.[0]?.image)"
            :slug="post.slug"
        />
    </div>
</div>

```

- **Transformación y formato de datos:**

- Método para formatear fechas a un estilo corto y claro.
- Método para construir URLs completas de imágenes o usar un placeholder.

```

formatDate(dateStr) {
    const date = new Date(dateStr);
    const day = String(date.getDate()).padStart(2, '0');

```

```

    const month = date.toLocaleString('default', { month: 'short' }).toUpperCase();
    const year = date.getFullYear();
    return `${day} ${month} ${year}`;
},
getimageUrl(imagePath) {
    return imagePath ? `http://localhost:8000${imagePath}` : 'https://via.placeholder.com/300x200';
},

```

PostDetail.vue

Este componente muestra los detalles completos de un post específico, recuperando los datos del backend mediante el parámetro `slug` de la URL.

Funcionalidades principales

- **Carga de datos según slug de ruta:**

En `created()`, se obtiene el post correspondiente con el slug extraído del router.

```

fetchPost() {
    const slug = this.$route.params.slug;
    axios
        .get(`/api/v1/posts/${slug}`)
        .then((response) => {
            this.post = response.data;
        })
        .catch((error) => {
            console.error('Error al obtener los detalles del post:', error);
        });
},
created() {
    this.fetchPost();
},

```

- **Renderizado condicional:**

Mientras no se haya cargado el post, se muestra un texto de "Cargando...". Cuando está listo, se muestra el contenido completo.

```
<div v-if="post" class="container-wide py-5" id="container">
  <div class="text-center mb-4">
    <h1 class="text-primary">{{ post.title }}</h1>
    <p class="text-muted date-large">{{ formatDate(post.created_at) }}</p>
  </div>
  ...
</div>
<div v-else>
  <p class="text-center">Cargando...</p>
</div>
```

- **Visualización de imágenes:**

Se muestra una imagen principal fija y, si hay más imágenes, se muestran como miniaturas debajo.

```
<div v-if="post.images && post.images.length > 0" class="text-center mb-4">
  
</div>

<div v-if="post.images && post.images.length > 1" class="d-flex justify-content-center flex-wrap mb-4 gap-3">
  <div v-for="(image, index) in post.images.slice(1)" :key="index" class="col-md-4 mb-3">
    
  </div>
</div>
```

- **Formato personalizado de fecha:**

Para mostrar la fecha en formato local y con horas.

```
formatDate(dateString) {  
  const options = {  
    day: '2-digit',  
    month: 'short',  
    hour: '2-digit',  
    minute: '2-digit',  
    hour12: false,  
  };  
  const date = new Date(dateString);  
  return new Intl.DateTimeFormat('es-ES', options).format(date).replace(';',  
  '');  
},
```

- **Botón para volver a la lista de posts:**

Usa un componente `SolidButton` con routing para facilitar la navegación.

```
<SolidButton  
  text="Volver a la lista de posts"  
  iconClass="fas fa-arrow-left"  
  routeTo="/posts"  
/>
```

Product.vue

Este componente muestra la página de detalle de un producto con imágenes, opciones seleccionables, cantidad y posibilidad de añadirlo al carrito. Incluye una galería modal para ver imágenes ampliadas.

Funcionalidades principales

- **Carga de producto por parámetros de URL:**

En el `mounted()`, se obtiene el producto desde la API REST usando los slugs de categoría y producto de la ruta.

```

async getProduct() {
  const store = useMainStore();
  store.setIsLoading(true);
  const category_slug = this.$route.params.category_slug;
  const product_slug = this.$route.params.product_slug;

  try {
    const response = await axios.get(
      `/api/v1/products/${category_slug}/${product_slug}`
    );
    this.product = response.data;
    document.title = this.product.name + ' | TangerineMess';
    this.displayedPrice = parseFloat(this.product.price);

    if (this.product.options && this.product.options.length > 0) {
      this.selectedOption = this.product.options[0];
      this.updatePrice();
    }
  } catch (error) {
    console.error(error);
  } finally {
    store.setIsLoading(false);
  }
},
mounted() {
  this.getProduct();
},

```

- **Visualización de imágenes y galería modal:**

- Imagen principal con zoom al hacer clic.
- Miniaturas de otras imágenes que abren la galería.
- Modal con navegación entre imágenes (anterior, siguiente) y botón de cierre.

```



<div v-if="product.images && product.images.length > 0" class="d-flex flex-wrap gap-2 mt-3 justify-content-center">
    <div v-for="img in product.images" :key="img.id" style="width: 48%;">
        
    </div>
</div>

<div v-if="galleryOpen" class="gallery-modal" @click.self="closeGallery">
    <button class="btn-close btn-close-white gallery-close-btn" @click="closeGallery"></button>
    
    <button class="gallery-nav prev" @click.stop="prevImage">#10094;</button>
    <button class="gallery-nav next" @click.stop="nextImage">#10095;</button>
</div>

```

- **Selección de opciones del producto:**

Si el producto tiene opciones (e.g., tamaño, color), se muestran en un select. Al cambiar la opción, se actualiza el precio mostrado.

```
<div v-if="product.options && product.options.length > 0" class="mb-3">
    <label for="product-option-select" class="form-label">Opciones</label>
```

```

<select
  id="product-option-select"
  class="form-select"
  v-model="selectedOption"
  @change="onOptionChange"
>
  <option v-for="option in product.options" :key="option.id" :value="option">
    {{ option.name }} ({{ parseFloat(option.additional_price).toFixed(2) }}€)
  </option>
</select>
</div>

```

- **Cantidad con botones + / - y validación:**

El usuario puede modificar la cantidad con los botones o escribiendo directamente. No permite cantidades menores que 1.

```

<div class="mb-3">
  <label class="form-label">Cantidad</label>
  <div class="input-group quantity-input">
    <button class="btn quantity-btn" @click="quantity > 1 && quantity--">
      <i class="fas fa-minus"></i>
    </button>
    <input
      type="number"
      class="form-control text-center quantity-field"
      min="1"
      v-model.number="quantity"
    />
    <button class="btn quantity-btn" @click="quantity++">
      <i class="fas fa-plus"></i>
    </button>
  </div>
</div>

```

- **Añadir al carrito y notificación (toast):**

Se crea un objeto con el producto seleccionado, cantidad y opción elegida, y se añade al store global. Luego se muestra un toast de confirmación que desaparece tras 2 segundos o al cerrarlo manualmente.

```
addToCart() {
  if (isNaN(this.quantity) || this.quantity < 1) {
    this.quantity = 1;
  }

  const item = {
    product: {
      ...this.product,
      selectedOption: this.selectedOption,
      price: this.selectedOption
        ? parseFloat(this.selectedOption.additional_price)
        : parseFloat(this.product.price),
      totalWeight: this.selectedOption
        ? parseFloat(this.product.weight || 0) +
          parseFloat(this.selectedOption.additional_weight || 0)
        : parseFloat(this.product.weight || 0),
    },
    quantity: this.quantity,
  };

  const store = useMainStore();
  store.addToCart(item);

  this.showToast = true;

  if (this.toastTimeoutId) clearTimeout(this.toastTimeoutId);
  this.toastTimeoutId = setTimeout(() => {
    this.showToast = false;
  }, 2000);
},
```

- **Navegación de la galería:**

```

openGallery(img) {
  const mainImage = {
    full_image_url: this.product.get_image,
    alt_text: 'Imagen principal',
    id: 'main',
  };
  this.galleryImages = [mainImage, ...this.product.images];
  this.currentGalleryIndex = this.galleryImages.findIndex(
    i => i.id === img.id || i.full_image_url === img.full_image_url
  );
  this.currentGalleryImage = this.galleryImages[this.currentGalleryIndex];
  this.galleryOpen = true;
},

closeGallery() {
  this.galleryOpen = false;
  this.currentGalleryImage = null;
  this.galleryImages = [];
  this.currentGalleryIndex = 0;
},

nextImage() {
  this.currentGalleryIndex = (this.currentGalleryIndex + 1) % this.galleryImages.length;
  this.currentGalleryImage = this.galleryImages[this.currentGalleryIndex];
},

prevImage() {
  this.currentGalleryIndex = (this.currentGalleryIndex - 1 + this.galleryImages.length) % this.galleryImages.length;
  this.currentGalleryImage = this.galleryImages[this.currentGalleryIndex];
},

```

- **Botón para volver a la lista de productos:**

Usa un componente `SolidButton` con routing hacia `/store`.

```
<SolidButton  
  text="Volver a la lista de productos"  
  iconClass="fas fa-arrow-up"  
  routeTo="/store"  
>
```

mainStore.js

Este store es el corazón de la gestión del estado global en tu aplicación usando Pinia. Básicamente, aquí guardas y manejas cosas esenciales como:

- El carrito de compras (`cart`),
- El estado de autenticación del usuario (`isAuthenticated` , `token` , `username`),
- El estado de carga (`isLoading`).

1. Estado (`state`)

El estado inicial guarda:

- `cart` : Un objeto con un array `items` donde vas almacenando los productos añadidos al carrito.
- `isAuthenticated` : Para saber si el usuario está logueado o no.
- `token` : El token de autenticación JWT o similar.
- `username` : El nombre del usuario conectado, para manejar datos específicos, por ejemplo, el carrito por usuario.
- `isLoading` : Para mostrar indicadores de carga en la UI cuando estás esperando respuestas de APIs o procesos.

```
state: () => ({  
  cart: { items: [] },  
  isAuthenticated: false,  
  token: '',  
  isLoading: false,
```

```
    username: ''  
  },
```

2. Getters

Son como propiedades calculadas que te permiten obtener datos derivados del estado de forma sencilla y reactiva.

- `cartTotalLength`: Suma total de unidades que hay en el carrito.
- `cartTotalPrice`: Precio total acumulado de todos los productos en el carrito.

```
getters: {  
  cartTotalLength(state) {  
    return state.cart.items.reduce((acc, item) => acc + item.quantity, 0)  
  },  
  cartTotalPrice(state) {  
    return state.cart.items.reduce(  
      (acc, item) => acc + item.quantity * item.product.price,  
      0  
    )  
  }  
}
```

Así, por ejemplo, cuando quieras mostrar el total de productos o el total a pagar, solo llamas a estas propiedades sin tener que recalcular a mano.

3. Actions

Aquí están todas las funciones que modifican el estado, que también pueden hacer operaciones asíncronas o lógicas complejas.

- **initializeStore:**

Al iniciar la app, esta función carga el estado guardado en `localStorage`.

Por ejemplo, recupera el token, el usuario y el carrito asociado a ese usuario.

```
initializeStore() {  
  const token = localStorage.getItem('token')  
  if (token) {
```

```

    this.token = token
    this.isAuthenticated = true
} else {
    this.token = ""
    this.isAuthenticated = false
}

const storedUsername = localStorage.getItem('username')
if (storedUsername) {
    this.username = storedUsername
}

const cartKey = this.username ? `cart_${this.username}` : 'cart'
const storedCart = localStorage.getItem(cartKey)
this.cart = storedCart ? JSON.parse(storedCart) : { items: [] }
},

```

- **addToCart:**

Añade un producto al carrito. Si el producto ya está (considerando la opción seleccionada), solo aumenta la cantidad. Luego actualiza el `localStorage`.

```

addToCart(item) {
    const existing = this.cart.items.find(i =>
        i.product.id === item.product.id &&
        JSON.stringify(i.product.selectedOption) === JSON.stringify(item.product.selectedOption)
    )

    if (existing) {
        existing.quantity += item.quantity
    } else {
        this.cart.items.push(item)
    }

    const cartKey = this.username ? `cart_${this.username}` : 'cart'

```

```
localStorage.setItem(cartKey, JSON.stringify(this.cart))  
},
```

- **removeFromCart:**

Elimina un producto del carrito (también compara opciones para distinguir productos iguales con diferentes opciones).

- **clearCart:**

Vacia el carrito y sincroniza `localStorage`.

- **setToken, removeToken, setUsername, setIsLoading:**

Métodos para actualizar estados relacionados con la sesión y la carga.

- **updateLocalStorageCart:**

Solo para sincronizar el carrito en `localStorage` cuando se quiera sin modificar el estado.

Uso típico

En la aplicación, cuando el usuario añade algo al carrito, se llama a:

```
store.addToCart({ product: productoSeleccionado, quantity: cantidad })
```

Y cuando la aplicación arranca, cargas el estado guardado con:

```
store.initializeApp()
```

Router:

Para la navegación dentro de mi aplicación, utilizo **Vue Router 4** configurado con el modo historial (`createWebHistory`) para que las URLs se vean limpias, sin el típico `#`.

He definido todas las rutas necesarias para las distintas vistas, como la página principal (`Home`), la tienda (`Store`), los productos, el blog, el carrito, y muchas otras. Cada ruta está asociada a un componente Vue en formato **Single File Component (SFC)**.

Además, algunas rutas requieren que el usuario esté autenticado para acceder (por ejemplo, "Mi cuenta" y el proceso de "Checkout"). Para esto, utilizo el campo `meta: { requireLogin: true }` dentro de las rutas que deben estar protegidas.

Para controlar el acceso, implemento un **guard global** con `router.beforeEach`, que antes de cada cambio de ruta revisa si la ruta requiere login y si el usuario está autenticado. Esto lo verifico consultando el estado de autenticación desde la store principal (Pinia). Si no está autenticado, redirijo al usuario a la página de login y le paso la ruta original en la query para que, una vez inicie sesión, pueda regresar a donde quería ir.

Esto me asegura que solo usuarios con sesión activa puedan acceder a partes privadas de la app, manteniendo una buena experiencia de usuario y seguridad.

En resumen, con esta configuración de Vue Router logro:

- Navegación fluida y organizada entre las distintas vistas.
- URLs amigables y fáciles de leer.
- Control de acceso basado en autenticación, integrado con la store global.
- Uso de rutas dinámicas para mostrar productos y categorías según la URL.

Parte del Código comentado de router:

```
import { createRouter, createWebHistory } from 'vue-router'
import { useMainStore } from '../store/mainstore'

// Importación de las vistas (componentes) para cada ruta
import Home from '@/views/Home.vue'
import Product from '../views/Product.vue'
// ... otras importaciones omitidas para brevedad

// Definición de rutas
const routes = [
  {
    path: '/',
    name: 'home',
    component: Home
  },
  {
    path: '/product/:id',
    name: 'product',
    component: Product
  }
]

// Guard global para controlar el acceso
router.beforeEach((to, from, next) => {
  const requireAuth = to.meta.requireLogin
  if (requireAuth) {
    const store = useMainStore()
    if (!store.isAuth) {
      next('/login?from=' + to.fullPath)
    } else {
      next()
    }
  } else {
    next()
  }
})
```

```

path: '/store',
name: 'Store',
component: Store
},
{
path: '/:category_slug/:product_slug/',
name: 'Product',
component: Product
},
// Otras rutas...
{
path: '/my-account',
name: 'MyAccount',
component: MyAccount,
meta: { requireLogin: true } // Ruta protegida, requiere login
},
{
path: '/cart/checkout',
name: 'Checkout',
component: Checkout,
meta: { requireLogin: true } // Ruta protegida, requiere login
},
]

```

// Creación del router con historial HTML5 (sin # en URL)

```

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
})

```

// Guard global antes de cada navegación para proteger rutas con requireLogin

```

router.beforeEach((to, from, next) => {
  const store = useMainStore() // Accede a la store Pinia para estado global
  store.initializeApp() // Inicializa la store (ej. carga token/localStorage)

  if (to.matched.some(record => record.meta.requireLogin) && !store.isAuthenticated)
    // Si la ruta requiere login y el usuario NO está autenticado, redirige a Login
    next({ name: 'Login', query: { to: to.path } })
})

```

```
    } else {
      // Si está autenticado o la ruta no requiere login, continúa normalmente
      next()
    }
  })

export default router
```

Tests

Para asegurarme de que las partes más importantes de la aplicación funcionen correctamente, he implementado algunos tests unitarios con **Jest** y **@vue/test-utils**. Estos tests me ayudan a validar el comportamiento esperado de los componentes clave como el login, el formulario de contacto y el checkout.

Login

El test del login se encarga de comprobar que el formulario se renderiza correctamente y que, al hacer submit, se llama al método de autenticación del store. También incluye un caso en el que simulo un error de login para asegurarme de que se muestra un mensaje apropiado si las credenciales son incorrectas.

Contacto

Aquí testeo el formulario de contacto que se encuentra en la sección "¡Hablemos!". El objetivo principal es asegurarme de que:

- Se puede enviar el formulario con datos válidos.
- Se muestra un mensaje de éxito si la petición se completa correctamente.
- En caso de error (por ejemplo, si falla la conexión con la API), se notifica adecuadamente al usuario.
- Además, el formulario se limpia después de enviarlo con éxito.

Checkout

En el checkout, me enfoqué en testear el flujo más crítico: que los campos obligatorios se validen correctamente antes de continuar. El test comprueba que:

- Si falta algún campo importante (nombre, email, dirección...), se muestran errores y **no** se continúa con la petición a Stripe.
- Stripe **no** se ejecuta si hay errores de validación.
- (Más adelante podría incluir un test para simular una compra exitosa con redirección al "success").

DPL

Explicación detallada paso a paso del despliegue con Gunicorn, Supervisor, Nginx y Certbot en Ubuntu

Me he informado y he explorado opciones y maneras de despliegue. He elegido **digitalocean**.

En primer lugar, creé un Droplet tras registrarme, eligiendo un plan de acuerdo a mis necesidades. Elegí el más sencillo.

and dev/test environments.

CPU options

<input checked="" type="radio"/> Regular Disk type: SSD	<input type="radio"/> Premium Intel Disk: NVMe SSD	<input type="radio"/> Premium AMD Disk: NVMe SSD			
\$6/mo \$0.009/hour 1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer	\$12/mo \$0.018/hour 2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer	\$18/mo \$0.022/hour 2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer	\$24/mo \$0.036/hour 4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer	\$48/mo \$0.071/hour 8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer	\$96/mo \$0.143/hour 16 GB / 8 CPUs 320 GB SSD Disk 6 TB transfer

Show all plans

Additional Storage

The screenshot shows the DigitalOcean interface for a project titled "TangerineMessWeb". On the left, there's a sidebar with "Resources", "Activity", and "Settings". Below that is a list of "DROPLETS (1)" containing one entry: "ubuntu-tangy" with IP "178.128.35.47". To the right is a sidebar with various creation options: "Droplets" (Create cloud servers), "Agents" (Create GenAI Agents), "Knowledge Bases" (Create GenAI Knowledge Bases), "GPU Droplets" (Create cloud servers with GPUs), "Kubernetes" (Create Kubernetes clusters), "App Platform" (Deploy your code), "Functions" (Create Cloud Functions), "SaaS Add-Ons" (Deploy Marketplace software), "Databases" (Create database clusters), and "Volumes Block Storage" (Add storage to Droplets). At the top, there's a search bar and a "Create" button.

Y comprar un dominio, para lo que elegí namecheap, pues como el nombre indica, es económico. Lo conecté con la IP del droplet de digitalocean.

The screenshot shows the Namecheap domain management interface for the domain "tangerinemess.com". The left sidebar has links for Dashboard, Expiring / Expired, Domain List (selected), Hosting List, Private Email, SSL Certificates, Apps, My Offers (NEW), and Profile. The main area shows the "Domains → Details" for "tangerinemess.com". The "Advanced DNS" tab is selected. Under "HOST RECORDS", there are two entries:

Type	Host	Value	TTL
A Record	@	178.128.35.47	Automatic
A Record	apitangy	178.128.35.47	Automatic

At the bottom, there's a "ADD NEW RECORD" button.

1. Conectarse al servidor por SSH

```
ssh root@178.128.35.47
```

- Te conectas al servidor remoto usando el usuario root.

- La IP es la del servidor en DigitalOcean (o similar).
- Se te pedirá la contraseña (`6cR@66)X0c`).

2. Actualizar paquetes del sistema

```
sudo apt-get update  
sudo apt-get upgrade
```

- `update` : actualiza la lista de paquetes disponibles.
- `upgrade` : instala las actualizaciones disponibles para paquetes ya instalados.
- Mantiene el sistema seguro y con las últimas versiones.

3. Instalar software necesario para el proyecto

```
sudo apt install python3-pip python3-dev libpq-dev postgresql postgresql-contrib nginx  
sudo apt install certbot python3-certbot-nginx
```

- `python3-pip` : gestor de paquetes Python.
- `python3-dev` : herramientas para compilar paquetes Python con extensiones nativas.
- `libpq-dev` : librerías para PostgreSQL, necesarias para la conexión desde Python.
- `postgresql` , `postgresql-contrib` : base de datos PostgreSQL y herramientas adicionales.
- `nginx` : servidor web que actuará como proxy inverso.
- `certbot` y `python3-certbot-nginx` : para obtener y gestionar certificados SSL de Let's Encrypt.

4. Crear base de datos y usuario en PostgreSQL

```
sudo -u postgres psql
```

- Accedemos al prompt de PostgreSQL con usuario administrador `postgres` .

Dentro de PostgreSQL:

```
CREATE DATABASE tangy;
CREATE USER tangyuser WITH PASSWORD 'tangypassword';
ALTER ROLE tangyuser SET client_encoding TO 'utf8';
GRANT ALL PRIVILEGES ON DATABASE tangy TO tangyuser;
\q
```

- Creamos la base de datos llamada `tangy` para el proyecto.
- Creamos un usuario `tangyuser` con contraseña para conectarse a la DB.
- Configuramos la codificación UTF-8 para manejar acentos y caracteres.
- Le damos todos los permisos sobre la base de datos.

5. Instalar herramientas para Python y entornos virtuales

```
sudo -H pip3 install --upgrade pip
sudo -H pip3 install --virtualenv pip
sudo apt install python3-venv python3-pip
```

- Actualizamos `pip` (gestor de paquetes Python).
- Instalamos `virtualenv` para crear entornos aislados.
- Instalamos el paquete `python3-venv` para crear entornos virtuales con `venv`.

6. Crear estructura de carpetas y usuarios para la app

```
mkdir -p /webapps/tangy
cd /webapps/tangy

sudo groupadd --system webapps
sudo useradd --system --gid webapps --shell /bin/bash --home /webapps/
tangy tangy
```

- Creamos carpeta `/webapps/tangy` donde vivirá la app.
- Creamos grupo de sistema `webapps` para manejar permisos.

- Creamos usuario sistema `tangy` con ese grupo, con home en la carpeta anterior, para correr la app con menos privilegios que root (por seguridad).

7. Crear entorno virtual Python para el proyecto

```
python3 -m venv environment_3_8_2  
source environment_3_8_2/bin/activate  
pip install --upgrade pip
```

- Creamos entorno virtual con Python 3.8.2 (o el que uses).
- Activamos el entorno para usar sus paquetes aislados.
- Actualizamos `pip` dentro del entorno.

8. Preparar archivo con dependencias del proyecto

En tu entorno local (donde desarrollas):

```
pip freeze > req.txt
```

- Esto genera una lista de todas las librerías usadas con sus versiones.

Ejemplo de dependencias (tu archivo `req.txt`):

```
asgi==3.8.1  
certifi==2025.1.31  
cffi==1.17.1  
charset-normalizer==3.4.1  
cryptography==44.0.2  
defusedxml==0.7.1  
Django==5.2  
django-cors-headers==4.7.0  
django-rest-framework==0.1.0  
djangorestframework==3.16.0  
djangorestframework_simplejwt==5.5.0  
djoser==2.3.1  
idna==3.10  
oauthlib==3.2.2  
pillow==11.2.1
```

```
pycparser==2.22
PyJWT==2.9.0
python3-openid==3.2.0
requests==2.32.3
requests-oauthlib==2.0.0
social-auth-app-django==5.4.3
social-auth-core==4.5.6
sqlparse==0.5.3
stripe==12.0.0
typing_extensions==4.13.2
tzdata==2025.2
urllib3==2.4.0
```

9. Subir dependencias al servidor y instalar

En servidor:

```
touch req.txt
vi req.txt
# Pegar el contenido del archivo req.txt aquí
:wq # para guardar y salir de vi

pip install -r req.txt
pip install psycopg2-binary
```

- Creamos el archivo `req.txt` con todas las dependencias del proyecto.
- Instalamos todas las librerías.
- `psycopg2-binary` es el adaptador para PostgreSQL en Python.

10. Subir proyecto Django al servidor

En local:

```
cd ..
zip -r djackets_tangy.zip tangy_django
scp djackets_tangy.zip root@178.128.35.47:
```

- Salimos de la carpeta del proyecto y comprimimos todo.

- Subimos el zip al servidor con `scp`.

En servidor:

```
apt install unzip  
unzip djackets_tangy.zip  
rm djackets_tangy.zip  
  
ls -larth  
chown -R tangy:webapps .
```

- Instalamos unzip para descomprimir el zip.
- Descomprimimos y borramos el zip para ahorrar espacio.
- Cambiamos permisos para que el usuario tangy y grupo webapps sean dueños de los archivos.

11. Configurar Django para producción

```
cd tangy_django/main  
  
cp settings.py settingsprod.py  
vi settingsprod.py
```

- Creamos archivo `settingsprod.py` para la configuración de producción.

Dentro de `settingsprod.py`, configurar base de datos:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql_psycopg2',  
        'NAME': 'tangy',  
        'USER': 'tangyuser',  
        'PASSWORD': 'tangypassword',  
        'HOST': 'localhost',  
        'PORT': '',  
    }  
}
```

- Configuras el acceso a la base de datos que creaste antes.

Luego:

```
cd ..  
cp manage.py manageprod.py  
vi manageprod.py
```

- Modifica `manageprod.py` para que use `settingsprod.py` en lugar de `settings.py` (cambia el `DJANGO_SETTINGS_MODULE`).

12. Migrar la base de datos y permisos en PostgreSQL

```
python manageprod.py makemigrations
```

- Detecta cambios en los modelos Django y crea migraciones.

Luego, en PostgreSQL:

```
sudo -u postgres psql  
\c tangy  
ALTER SCHEMA public OWNER TO tangyuser;  
GRANT ALL PRIVILEGES ON SCHEMA public TO tangyuser;  
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO tangyuser;  
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO tangy  
user;  
\q
```

- Cambia el propietario del esquema para que el usuario tenga permisos completos.

Finalmente:

```
python manageprod.py migrate
```

- Aplica las migraciones, creando tablas en la base.

13. Instalar Gunicorn y preparar script de inicio

```
pip install gunicorn  
cd ..  
vi environment_3_8_2/bin/gunicorn_start
```

Escribir script `gunicorn_start`:

```
#!/bin/sh

NAME='tangy_django'
DJANGODIR=/webapps/tangy/tangy_django
SOCKFILE=/webapps/tangy/environment_3_8_2/run/gunicorn.sock
USER=tangy
GROUP=webapps
NUM_WORKERS=3
DJANGO_SETTINGS_MODULE=tangy_django.settingsprod
DJANGO_WSGI_MODULE=tangy_django.wsgi
TIMEOUT=120

cd $DJANGODIR
source ../environment_3_8_2/bin/activate
export DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
export PYTHONPATH=$DJANGODIR:$PYTHONPATH

RUNDIR=$(dirname $SOCKFILE)
test -d $RUNDIR || mkdir -p $RUNDIR

exec ../environment_3_8_2/bin/gunicorn ${DJANGO_WSGI_MODULE}:application \
--name $NAME \
--workers $NUM_WORKERS \
--timeout $TIMEOUT \
--user=$USER --group=$GROUP \
--bind=unix:$SOCKFILE \
--log-level=debug \
--log-file=-
```

- Este script inicia Gunicorn con la configuración correcta, usando socket Unix, usuario y grupo específicos, con 3 trabajadores.

Luego:

```
chmod +x environment_3_8_2/bin/gunicorn_start
```

- Hacemos el script ejecutable.

14. Instalar y configurar Supervisor para mantener Gunicorn vivo

```
apt install supervisor  
cd /etc/supervisor/conf.d/  
touch tangy.conf  
vi tangy.conf
```

Escribir en `tangy.conf` :

```
[program:tangy_django]  
command = /webapps/tangy/environment_3_8_2/bin/gunicorn_start  
user = tangy  
stdout_logfile = /webapps/tangy/environment_3_8_2/logs/supervisor.log  
redirect_stderr = true  
environment=LANG=en_US.UTF-8,LC_ALL=en_US.UTF-8
```

- Configuramos Supervisor para que controle el proceso Gunicorn y lo reinicie si falla.

Luego:

```
mkdir /webapps/tangy/environment_3_8_2/logs/  
  
supervisorctl reread  
supervisorctl update  
supervisorctl status
```

- Recargamos Supervisor para que detecte la nueva configuración, la aplique y verifiquemos el estado.

15. Configurar Nginx para servir la app y hacer proxy a Gunicorn

```
cd /etc/nginx/sites-available  
vi apitangy.tangerinemess.com
```

Poner esta configuración:

```
upstream tangy_app_server {  
    server unix:/webapps/tangy/environment_3_8_2/run/gunicorn.sock fail_ti  
meout=0;  
}  
  
server {  
    listen 80;  
    server_name tangy.tangerinemess.com;  
    return 301 https://tangy.tangerinemess.com$request_uri;  
}  
  
server {  
    listen 443 ssl;  
    server_name tangy.tangerinemess.com;  
  
    client_max_body_size 4G;  
  
    access_log /webapps/tangy/environment_3_8_2/logs/nginx-django-acce  
ss.log;  
    error_log /webapps/tangy/environment_3_8_2/logs/nginx-django-error.lo  
g;  
  
    ssl_certificate /etc/letsencrypt/live/tangy.tangerinemess.com/fullchain.p  
em;  
    ssl_certificate_key /etc/letsencrypt/live/tangy.tangerinemess.com/privke  
y.pem;  
  
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;  
    ssl_prefer_server_ciphers on;  
    ssl_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+
```

```

EDH';

location /static/ {
    alias /webapps/tangy/environment_3_8_2/tangy_django/static/;
}

location /media/ {
    alias /webapps/tangy/tangy_django/media/;
}

location / {
    root /webapps/tangy/tangy_vue/dist;
    try_files $uri $uri/ /index.html;
}

location /api/ {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;
    proxy_pass http://tangy_app_server;
}
}

```

- Definimos upstream con el socket de Gunicorn.
- Redirigimos HTTP a HTTPS.
- Configuramos SSL con certificados de Let's Encrypt.
- Sirve archivos estáticos y media directamente para optimizar.
- Sirve frontend Vue desde carpeta `/webapps/tangy/tangy_vue/dist`.
- Reenvía las peticiones que empiezan por `/api/` a Gunicorn/Django.

Luego:

```

cd ../sites-enabled/
ln -s ../sites-available/apitangy.tangerinemess.com .
ls -larth
service nginx restart

```

- Creamos enlace simbólico para activar la configuración.
- Reiniciamos nginx para aplicar cambios.

16. Comprar dominio y configurar SSL con Certbot

```
sudo certbot -d tangy.tangerinemess.com  
service nginx restart
```

- Certbot obtiene certificado gratuito para tu dominio.
- Reinicia nginx para aplicar el certificado y habilitar HTTPS.

Con esto tenemos Django en producción, con base de datos PostgreSQL, servidor Gunicorn gestionado por Supervisor, y nginx como proxy inverso con SSL.

Configuración de seguridad y dominio en Django (`settingsprod.py`)

Archivo: `tangy_django/main/settingsprod.py`

Añadir dominios al proyecto:

```
ALLOWED_HOSTS = [  
    "tangy.tangerinemess.com",  
    "apitangy.tangerinemess.com",  
]  
  
CORS_ALLOWED_ORIGINS = [  
    "https://tangy.tangerinemess.com",  
    "https://apitangy.tangerinemess.com",  
]
```

Reiniciar Gunicorn vía Supervisor

```
supervisorctl restart tangy_django
```

Crear superusuario de Django

```
source .../environment_3_8_2/bin/activate  
python manageprod.py createsuperuser
```

Sigue los pasos para configurar un usuario administrador.

Cambiar URL hardcodeada en modelos

Archivo: [product/models.py](#)

Reemplaza cualquier IP local (como <http://127.0.0.1:8000>) por:

```
"https://apitangy.tangerinemess.com"
```

Después:

```
supervisorctl restart tangy_django
```

Despliegue del Frontend Vue

Configurar el endpoint del backend en Vue

Archivo: [src/main.js](#)

```
axios.defaults.baseURL = "https://apitangy.tangerinemess.com";
```

Compilar y empaquetar Vue

```
npm run build  
zip -r dist.zip dist
```

Enviar el build al servidor remoto

```
scp dist.zip root@178.128.35.47
```

Desempaquetar en el servidor

```
cp /root/dist.zip .
unzip dist.zip

mkdir tangy_vue
mv dist tangy_vue/dist

chown -R tangy:webapps .
```

Configuración del servidor NGINX para el Frontend

Crear archivo de configuración NGINX

```
cd /etc/nginx/sites-available
touch tangy.tangerinemess.com
vi tangy.tangerinemess.com
```

Contenido:

```
server {
    listen 80;
    server_name tangy.tangerinemess.com;
    return 301 https://tangy.tangerinemess.com$request_uri;
}

server {
    listen 443 ssl;
    server_name tangy.tangerinemess.com;

    client_max_body_size 4G;

    error_log /webapps/tangy/environment_3_8_2/logs/nginx-vue-error.log;
    access_log /webapps/tangy/environment_3_8_2/logs/nginx-vue-access.log;

    ssl_certificate /etc/letsencrypt/live/tangy.tangerinemess.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/tangy.tangerinemess.com/privkey.pem;
```

```
y.pem;

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;
ssl_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH: AES256+
EDH';

charset utf-8;
root /webapps/tangy/tangy_vue/dist;
index index.html index.htm;

location / {
    root /webapps/tangy/tangy_vue/dist;
    try_files $uri /index.html;
}
}
```

Activar el sitio en NGINX

```
cd /etc/nginx/sites-enabled
ln -s ../sites-available/tangy.tangerinemess.com .
```

Reiniciar NGINX para aplicar cambios

```
supervisorctl restart nginx
```

Verificación final

- Backend en: <https://apitangy.tangerinemess.com>
- Frontend en: <https://tangy.tangerinemess.com>

Abrir las URLs en el navegador y verifica que:

- Vue renderiza correctamente el frontend.
- Las llamadas a la API funcionan sin error de CORS o redirección.
- Django Admin está accesible vía <https://apitangy.tangerinemess.com/admin/>.