Software Requirements Specification

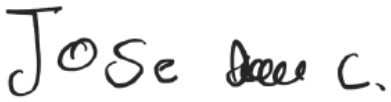
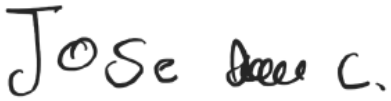
Project: MX Discovery
Module: Web Applications

Members:
Pulido Larios Cristopher
Cruz Arredondo José David
González Hernández Cruz Omar
Salas Figueroa Jesús Nahataen

Grade: 3-F

Career: ENVD (Entornos Virtuales y Negocios Digitales)

Revision History

Date	Revision	AUTHOR	Verified by:
10/10/23	1	Pulido Larios Cristopher: Descriptions of functional and non-functional requirements.	
28/10/23	2	González Hernández Cruz Omar: Development of class and use case diagrams.	
10/11/23	3	All members: backend connection to the database	**All members of MX Discovery**
16/11/23	4	Cruz Arredondo Jose David: Programming classes, serializers, views (backend)	
25/11/23	5	Salas Figueroa Jesús Nahataen: frontend development (js, css, ts)	
29/11/23	6	All members: Last revision of the project, beta phase.	**All members of MX Discovery**

Document validated by the parties on date: 30/11/23.

Client	Enterprise
UTT (Universidad Tecnológica de Tijuana)	Inspirarts Interactive (MX Discovery development company)

1. Introduction

Below, an overview of the Software Requirements Specifications that are part of MX Discovery will be shown in the sections.

1.1 Purpose

The purpose of this project is to create an e-commerce that consists of the distribution, sale, purchase, marketing, and provision of information about products or services over the Internet.
In the case of MX Discovery, it was requested to create the website with different sections such as: Home, us, workshops, contact, etc. In addition, the option to create a new user and be able to enter the page with said user was requested, as well as to be able to purchase products that are registered in the project database.

This product is aimed at merchants who would like to purchase an art product or one of the services that we can offer.

1.2 Product scope

The “MX Discovery” e-commerce is expected to be a store where users can purchase any artistic product or learn about one of our workshops that we offer.

The software is always functional, if the non-functional requirements are met, that is, the requirements that we suggest for its correct functioning. The e-commerce will be able to make sales, reviews, show information about our location, workshops and about the company, it will also have a form where users can enter their basic information so that we can contact the user in a more personal way.

1.3 Personnel involved

Name	Salas Figueroa Jesús Nahataen
Role	Project leader / Programmer
Professional category	TSU EVND
Responsibilities	Organize team activities
Contact information	664 264 4830

Name	González Hernández Cruz Omar
Role	Designer / Marketing
Professional category	TSU EVND
Responsibilities	Web content creator and advertising manager
Contact information	664 477 2061

Name	Cruz Arredondo Jose David
Role	Programmer / Assistant Manager / Designer
Professional category	TSU EVND
Responsibilities	Check program errors, code, and design
Contact information	664 447 5743

Name	Pulido Larios Cristopher
Role	Programmer / Documenter
Professional category	TSU EVND
Responsibilities	Programmer and person in charge of project documentation
Contact information	664 447 5743

1.4 Definitions, acronyms and abbreviations

- MX: It refers to the country of Mexico, origin of our product.
- TS: Refers to the TypeScript programming language.
- PY: Refers to the Python programming language.
- JS: Refers to the JavaScript programming language.
- MySQL: Database engine
- DC: Diagram Cases

1.5 References

Reference	Title	Date	Author/Organization
ISBN 978-0672327124	MySQL Crash Course	02/November/2023	Ben Forta
ISBN 978-1775093305	Python Tricks: A Buffet of Awesome Python Features	11/November/2023	Dan Bader

1.6 Summary

Next, an explanation will be given about the functionality of the product, its operation, which user it is intended for and what preparation is required to use the product.

2 Overview

2.1 Product perspective

It is an independent product.

2.2 Product functions

This product can sell art products to the public, as well as sell tickets for different workshops that we offer. Some other functionalities that e-commerce offers are the following:

• Intuitive Navigation: The website will ensure an intuitive user interface, allowing visitors to easily navigate the different sections and exhibits.

• Profile Customization: Users will be able to create profiles that will allow them to modify some data and add a profile photo.

• Device Compatibility: It will be ensured that the website is compatible with laptops and mobile devices.

• Comments and Feedback: Option for users to leave comments, rate exhibits and provide feedback, encouraging participation and online community.

• Secure shopping cart and payment process: Facilitate shopping by allowing users to add artwork to a cart, offer different secure and clear payment methods, and ensure the security of financial data.

• Educational content: Offer blogs, articles or videos about prominent artists, artistic techniques or information relevant to the art world, adding value beyond the simple sale.

2.3 User classes and characteristics

Type of user	General public, collectors, buyers.
Training	Knowledge of online shopping
Skills	Knowledge of purchases, use of web browser
Activities	Users will be able to purchase art items, view product information, as well as product information. You will also have the possibility of purchasing tickets for one of our workshops that we offer.

2.4 Restrictions

E-commerce restrictions are related to factors external to our product, such as the user's internet speed, the user's home if it is difficult for the package to go, and the type of internet browser used. It can be viewed on both PC and cell phones, but you may have incompatibility problems if you use less secure web browsers.

Because our product is recent, we only have the payment method through PayPal, we will continue to implement other payment methods so that users have other options to make transactions.

2.5 Assumptions and dependencies

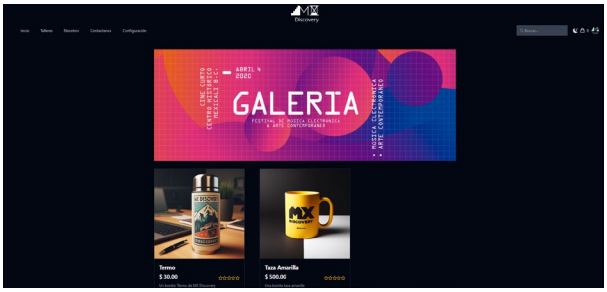
Due to the nature of our product, downloading your own software is not required to use it, however, we suggest users use one of the following internet browsers:

- Firefox
- Microsoft Edge
- Safari
- Google Chrome

Additionally, for the payment method, it is necessary to have a PayPal account to carry out transactions in the store.

2.6 Foreseeable evolution of the system

For future improvements or implementations to the shopping site, we have some ideas proposed by some of the MX Discovery members:



1) Implement a virtual tour where users can see from their devices a 3D tour showing different artistic sculptures

2) For administrators: We will implement in the future a control over the reviews that users publish on products, with the possibility of verifying if the user has tried the product, if so, they can publish the review, this to avoid fake reviews

3) We would improve the design of the beginning of the store, as well as the visualization of the products and the workshops, adding more detailed sample images.

4) Store performance in web browsers: it is an improvement that we will be working hard on, for the correct display of the sections, and a faster loading speed.

3 Specific requirements

Requirement number	SF1		
Requirement name	Account creation		
Type	Requirement		
Requirement source	At user request		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF1: The shop will be able to create an account in the store, it will be necessary to purchase any of the items offered by the store, however, it is not necessary to view the page.

Requirement number	SF1.2		
Requirement name	User profile customization		
Type	Requirement		
Requirement source	At user request		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF1.2: The user will have the possibility of personalizing their profile if they wish, where they can modify their first and last name, as well as their profile image. It is not a necessary requirement; it is completely optional.

Requirement number	SF1.3		
Requirement name	Post reviews of articles		
Type	Requirement		
Requirement source	At user request		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF1.3: The online store will allow users to leave a review of any item, rating it on a scale of 0 to 5 stars, as well as being able to add a review if desired.

Requirement number	SF1.4		
Requirement name	Shopping cart		
Type	Requirement		
Requirement source	At user request		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF1.4: The online store will allow users to add their items to a virtual cart, where the products they wish to purchase will be stored and will remain in their account until the user removes them or until they are no longer in stock.

Requirement number	SF1.5		
Requirement name	Payment method for purchase		
Type	Requirement		
Requirement source	At user request		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF1.5: The online store will allow users to proceed with the purchase of their items, it will be necessary to have a payment method, in this case, a PayPal account to proceed with the transaction.

Requirement number	SF1.6		
Requirement name	Company contact via email		
Type	Requirement		
Requirement source	At user request		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF1.6: The online store will have a contact us section, where the user will have to fill out a form that will ask for their name, email, and their main reason why they want to contact the company. This message will be received in the email address of the developer of MX Discovery.

Requirement number	SF2		
Requirement name	CRUD (Only Administrator)		
Type	Restrict		
Requirement source	Database		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF2: This specific requirement is only for administrators, it allows to make a CRUD to different sections of the store, some of them are:

- Delete, consult, modify, or add new products.
- Delete or consult the accounts of users who have registered, without being able to modify personal information of this, for this, the user must do it from your own account.
- Change the status of orders, whether delivered or not. Orders will be visible once the user's transaction has been successfully completed.

Requirement number	SF2.2		
Requirement name	Token Authentication		
Type	Requirement		
Requirement source	Server		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF2.2: The site's server will have a security token generator. Each user registered in the online store, will be generated automatically, this to increase the security of your account.

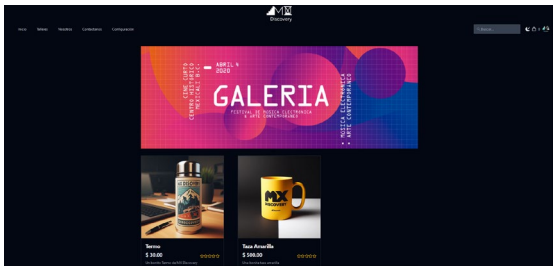
Requirement number	SF2.3		
Requirement name	Creation of administrator accounts		
Type	Restrict		
Requirement source	Server		
Requirement Priority	High/Essential	Average/Desired	Low/Optional

Description SF2.3: To access the configuration of the online store (CRUD), you must log in to the site with an administrator account, if you do not have one, you can create one from the backend server.

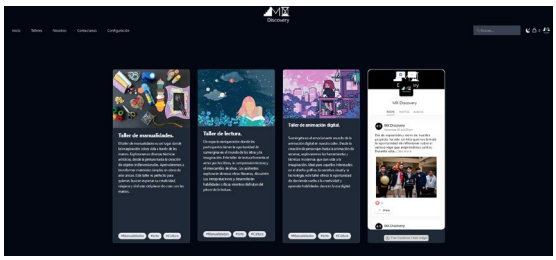
3.1 Common interface requirements

The inputs and outputs that the software can perform will be described.

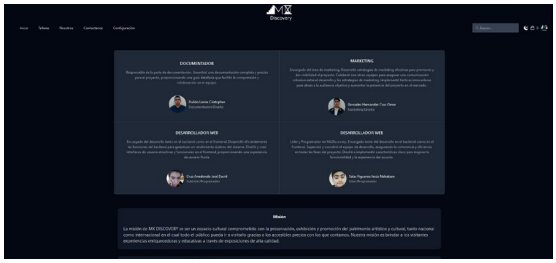
3.1.1 User Interfaces



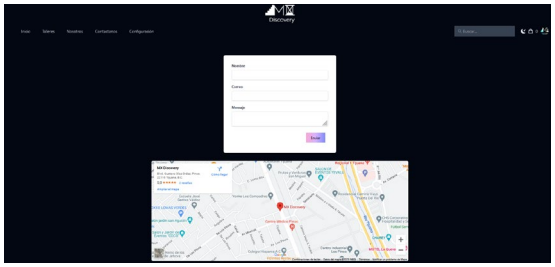
Home: In this section a carrousel will be displayed showing the news of the store. At the bottom will be displayed the items we have for sale



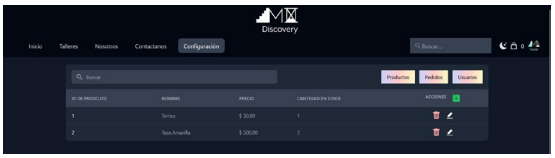
Workshops: Here will be displayed the workshops that MX Discovery offers to the public, on the right side, there will be a box that will show the publications of the Facebook page



Contact Us: A form will be displayed in case a registered user wants to communicate with us, besides showing the location of MX Discovery headquarters using Google Maps.



Configuration (Administrators only): This section will only be visible to site administrators, where they will be able to access the CRUD.

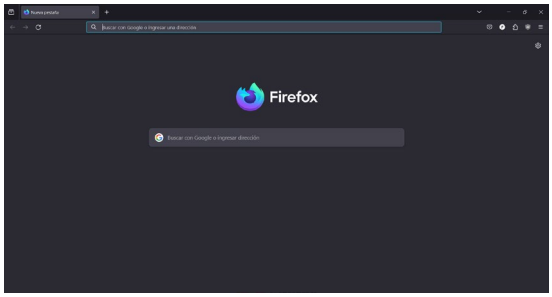


3.1.2 Hardware interfaces

The interfaces where the administrator will be able to interact with the web site will be only by means of a physical keyboard and a monitor.

3.1.3 Software interfaces

A software interface could be the internet browser, where the website can be viewed. We recommend one of the browsers that we mention in the



3.1.4 Communication interfaces

```
VITE v4.4.4  ready in 1055 ms

➜  Local:   http://localhost:5173/
➜  Network: http://192.168.3.12:5173/
➜  press h to show help
```

The VITE server is where the entire frontend of the page is hosted. Running 'npm run dev -- --host' will run the server using the computer's IP address as host.

```
System check identified no issues (0 silenced).
December 03, 2023 - 03:10:14
Django version 4.2.7, using settings 'backend.settings'
Starting development server at http://192.168.3.12:8000/
Quit the server with CONTROL-C.
```

The backend server runs thanks to the Django Framework, where the classes, views, and models are stored where the information is stored and redirects the views at the user's request through the frontend.

```
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.mysql',
'NAME': 'mxdiscovery',
'USER': 'usuario',
'PASSWORD': '',
'HOST': '192.168.3.12',
'PORT' : '3306',
}
}
```

The database server is provided by the MySQL database engine, where the data of users, products, accounts, orders, etc. will be stored.

These three servers must be running with the same IP address of the PC that acts as the host, in this way, communication between the 3 servers is achieved simultaneously.

3.2 Functional requirements

The functional requirements of the software are:

3.2.1 Functional requirement 1

It is recommended to use a monitor that has a resolution greater than HD (1280X720 pixels) to display the information correctly when using the software.

3.2.2 Functional requirement 2

All the servers must always be connected to the database assigned to it, in addition to the backend and frontend servers being running, otherwise, the page will not be able to be viewed or it will not be able to perform any operation.

3.2.3 Functional requirement 3

For security, the page has a security timer for the user. If you have an active session and have not performed any action in the browser for 5 minutes, your session will be automatically closed.

3.2.4 Functional requirement 4

It is recommended to use the following browsers to view the page correctly:

- Firefox
- Microsoft Edge
- Safari
- Google Chrome

Preferably, always keep it updated to the latest version.

3.3 Non-functional requirements

3.3.1 Performance requirements

1) Servers should preferably run on a computer that has at least 4GB of RAM, a quad-core processor and at least 20GB of memory. This to guarantee optimal performance.

2) An internet connection of at least 10 Mbps is required to avoid slow loading of online store content, such as the integrated Google Maps or Facebook APIs.

3) On mobile phones, it is recommended to have an Android device with version 7.0 or later and IOS 8.0 or later, due to the protocols and interfaces used by the page.

4) If the page has an error in certain actions, a message will be displayed in the browser visually that there was an error.

3.3.2 Security

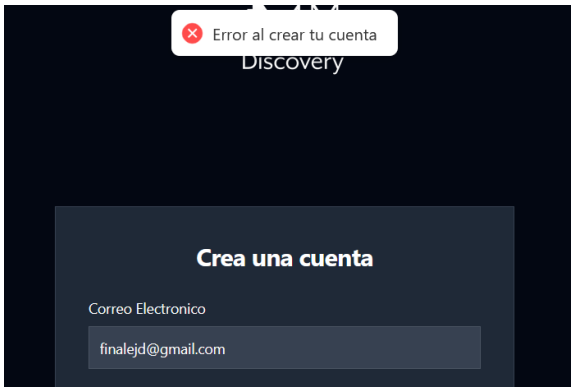
```
// con este fragmento de codigo, desde el token de un usuario, permite evaluar
// el tiempo que este usuario o administrador estara con su sesion abierta,
// una vez expirado el tiempo de 5 min, se refrescara la pagina en automatico,
// y sera devuelto a la pagina de inicio de sesion
authAxios.interceptors.request.use(async (config) => {
    const token: string = useAuthStore.getState().access;
    config.headers = {
        Authorization: `Bearer ${token}`,
    } as AxiosRequestHeaders;

    const tokenDecoded: Token = jwt_decode(token)

    const expiration = new Date(tokenDecoded.exp * 1000); // expiracion del token
    const now = new Date(); // la expiracion del token comienza apartir de la hora actual
    const fiveMin = 1000 * 60 * 5; // representa los 5 minutos en milisegundos, actua como un temporizador

    if (expiration.getTime() - now.getTime() < fiveMin)
        try {
            const response = await axi.post('/users/refresh/', { refresh: useAuthStore.getState().refresh })
            useAuthStore.getState().setToken(response.data.access, response.data.refresh)
        } catch (err) {
            logout()
        }
        return config
});
```

For the security of the page, through a function in TypeScript and using the 'jwt_decode' standard that Node.js provides us, we generate an authentication token for users who register on the website, in this way, the integrity of the page is ensured. your account against malicious attacks that try to obtain our database.

```
export interface Token {
  user_id: number;
  exp: number;
  is_staff: boolean;
  email: string;
  name: string;
  last_name: string;
  avatar: File | null;
}
```

The token is generated according to the information that the user provides, therefore, each token is unique and unrepeatable.

3.3.3 Reliability



The website has validations to prevent the creation of accounts using already registered emails.
Additionally, you will be required to enter all requested fields to create the account.

3.3.4 Availability

It is required to run the backend, frontend, and database servers, in addition to an internet connection for the correct functioning of the APIs integrated into the frontend.

3.3.5 Maintainability

Maintenance tasks should only be carried out by authorized personnel of Inspirarts Interactive (MX Discovery development company)

Due to the nature of the product, the status of the servers must be constantly reviewed, as well as the operation of the computer that acts as the physical server.

3.3.6 Portability

a) Servers running for operation: 100%

b) The page can theoretically be viewed on any device, although its optimal functioning is not guaranteed due to different external factors.

c) The website can be viewed on any operating system.

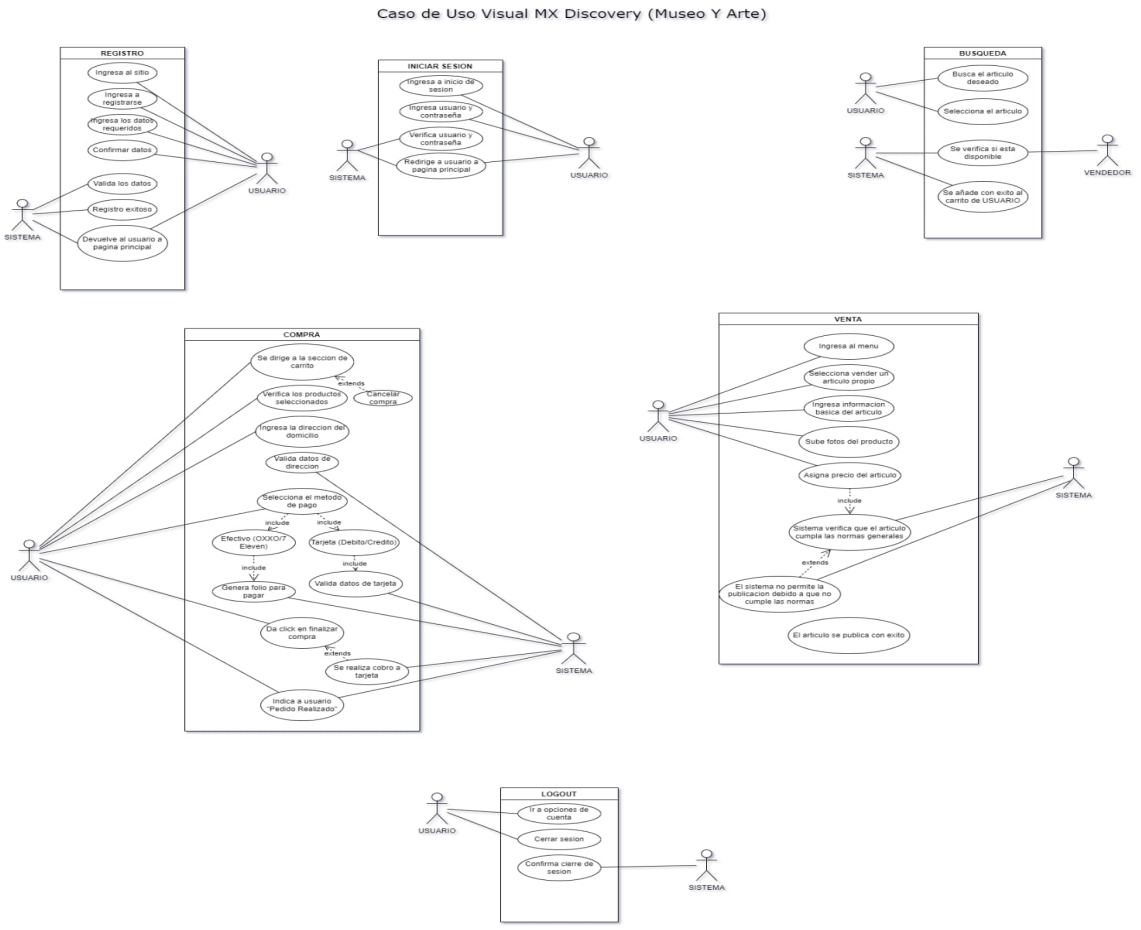
d) The language used (TypeScript), focused on web use, is compatible with internet browsers in both mobile and desktop versions.

3.4 Other requirements

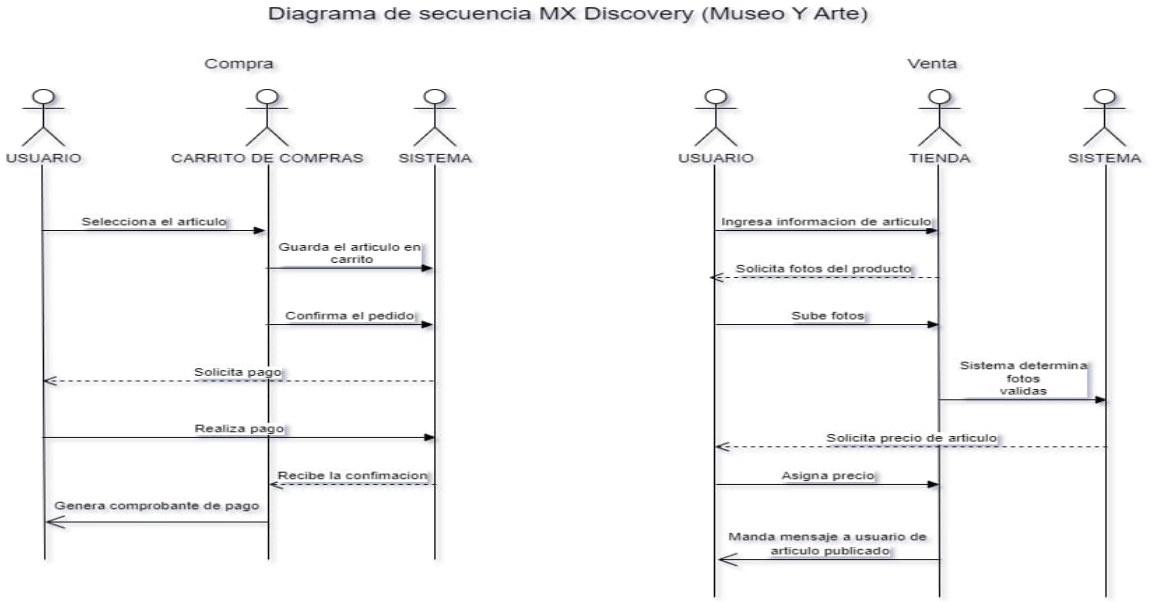
Manipulation, piracy, plagiarism, improper use of the "MX Discovery" computer software will be duly condemned and judged according to the crimes committed under the regimes established by the United Mexican States.

4 Appendix B: Analysis Models

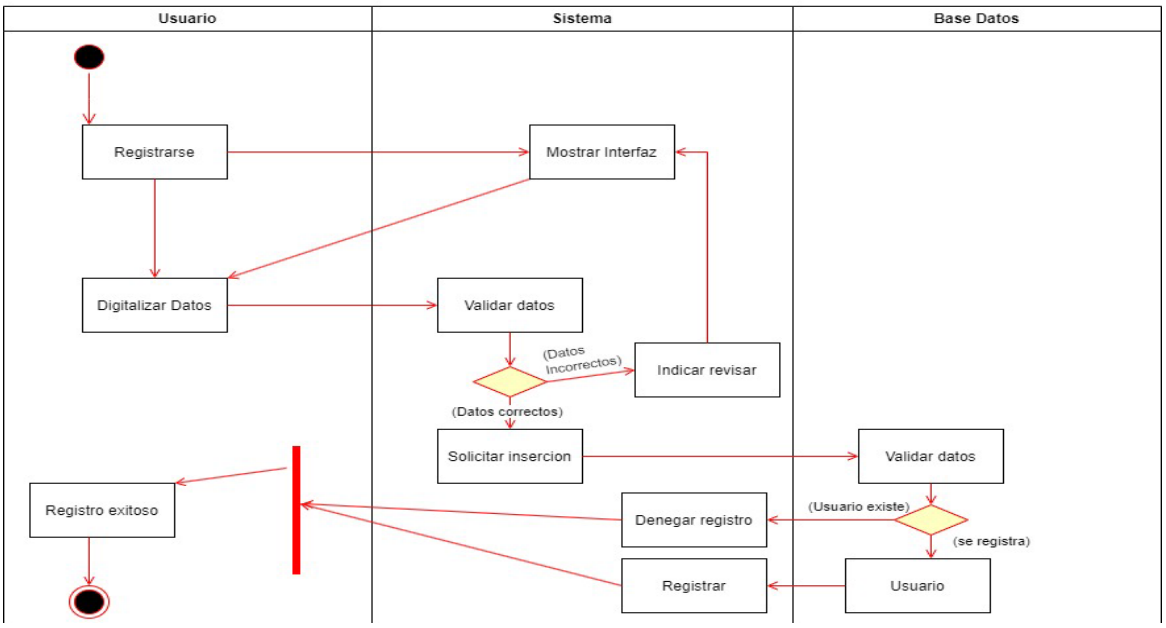
USE CASE DIAGRAM:



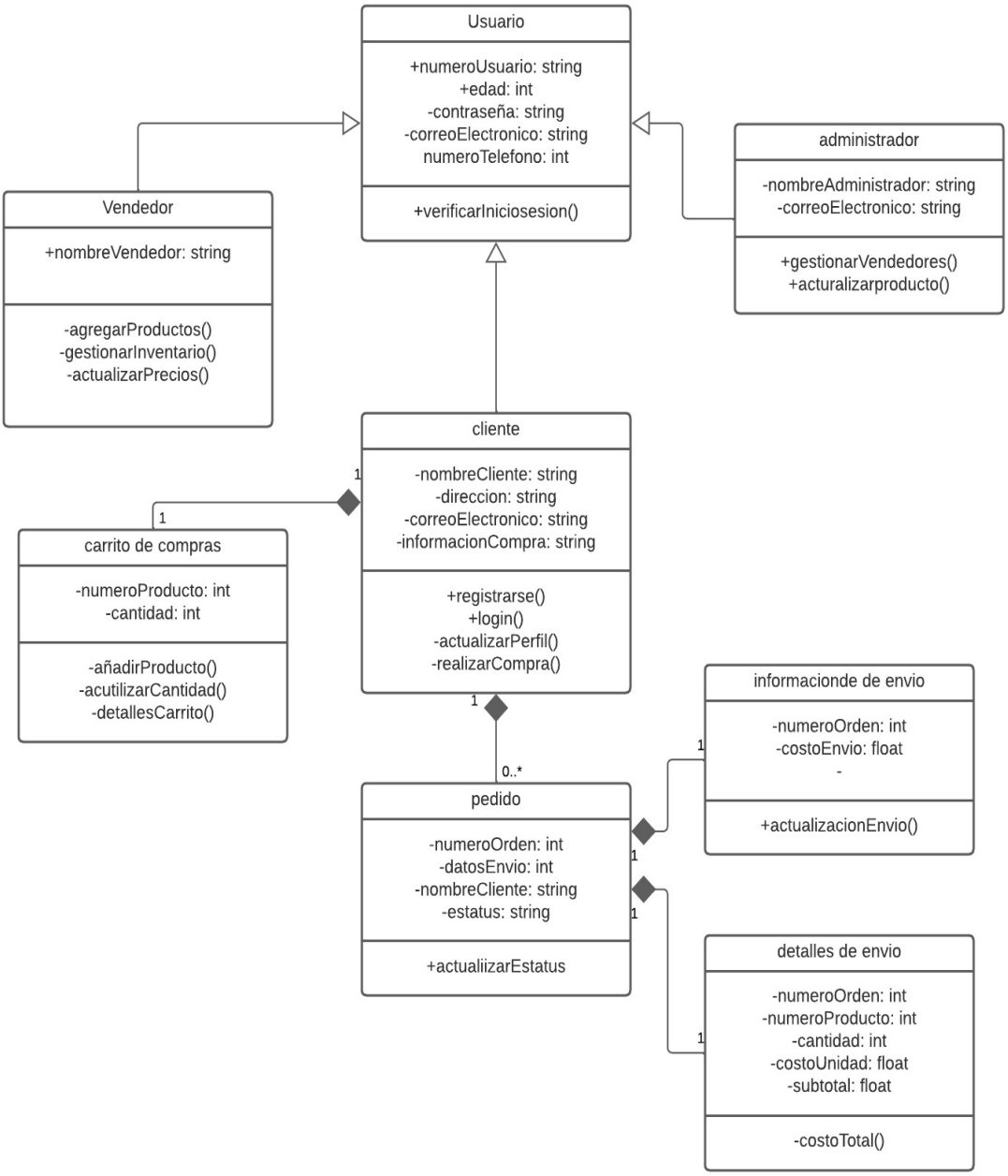
SEQUENCE DIAGRAM:



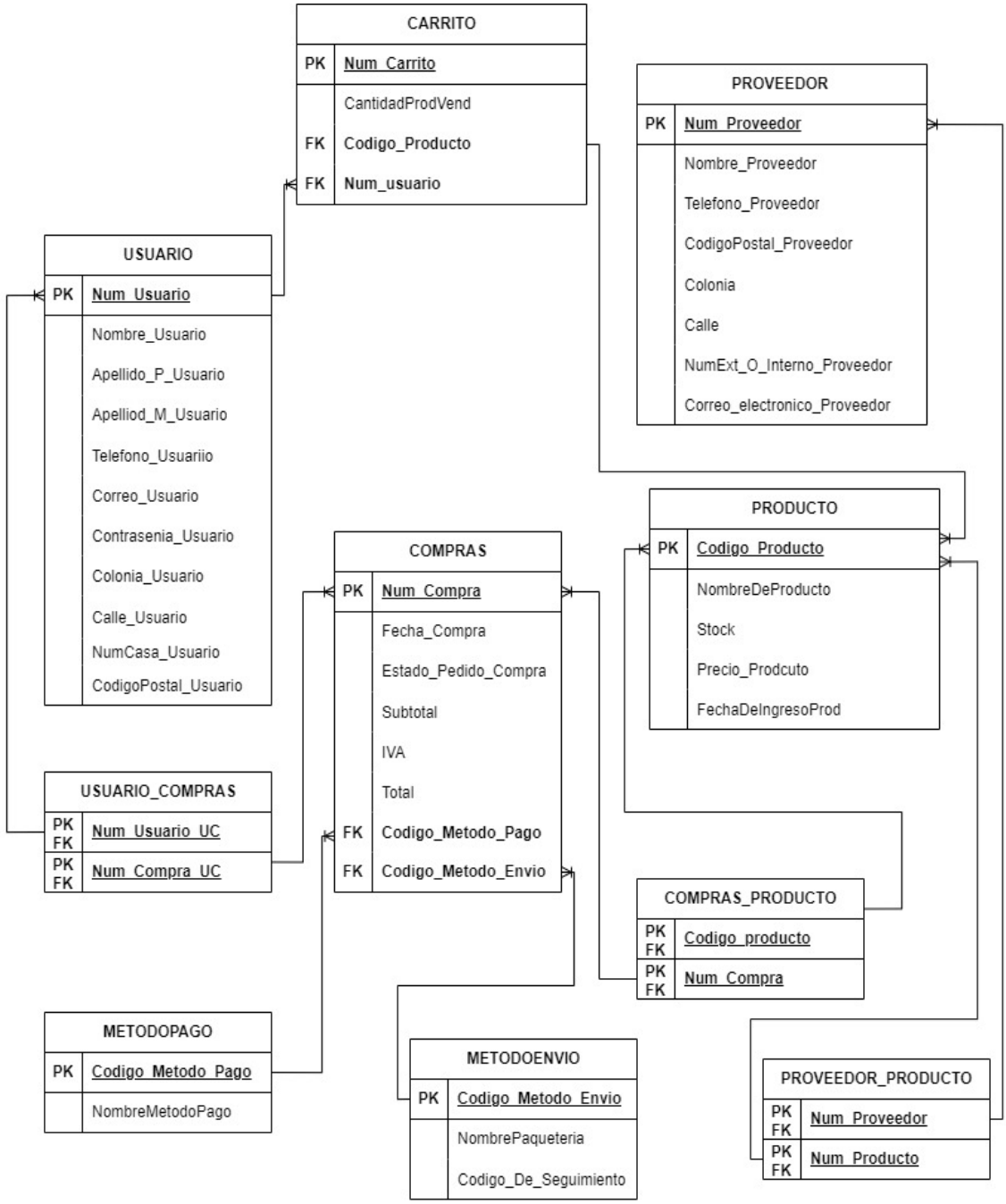
WORKDIAGRAM:



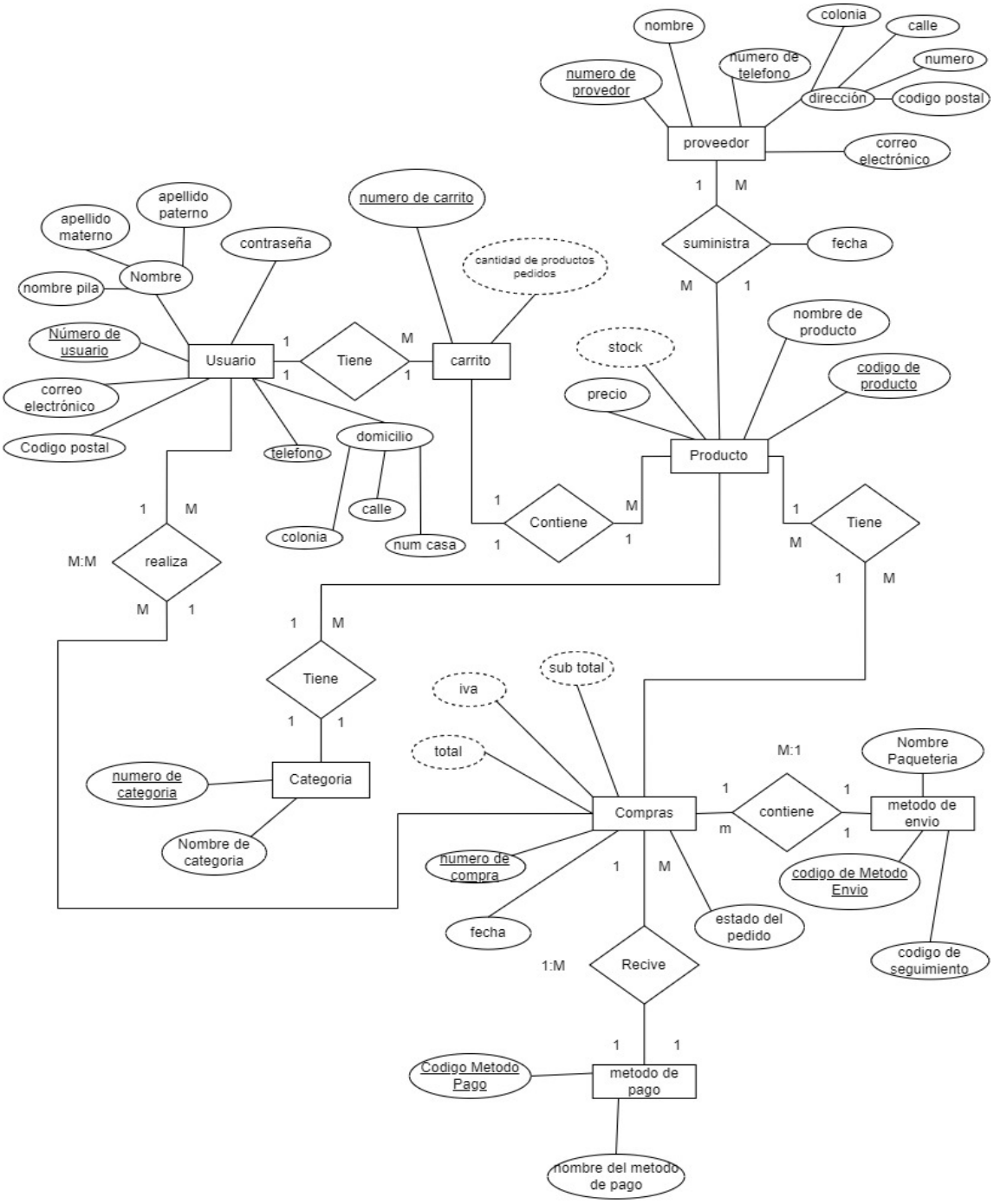
CLASS DIAGRAMS:



RELATIONAL MODEL:

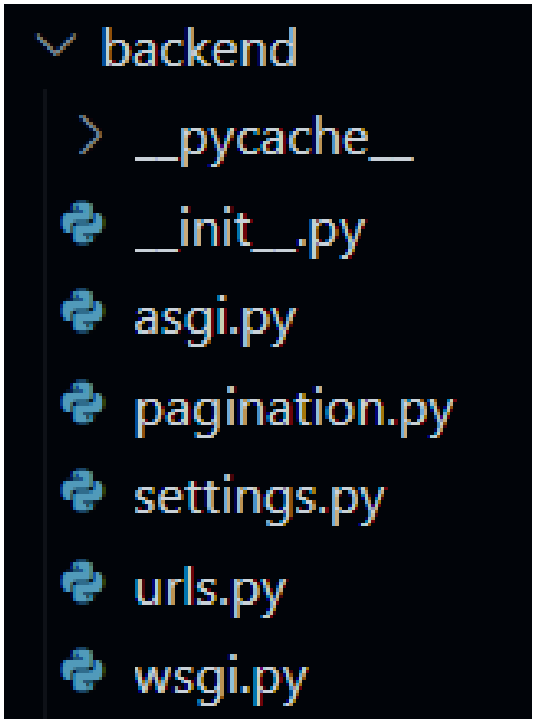


ENTITY-RELATIONSHIP DIAGRAM



5 Appendix B: Analysis Backend

The virtual store backend is made up of the following files, an overview of each file of the Django 'Backend' project will be displayed:



The Django project called 'Backend' generates the following python files:

__init__.py: This method is a special method called every time a class is instance and serves to initialize the object that is created.

asgi.py: Contains the configuration for optional implementation to the Asynchronous Server Linking Interface

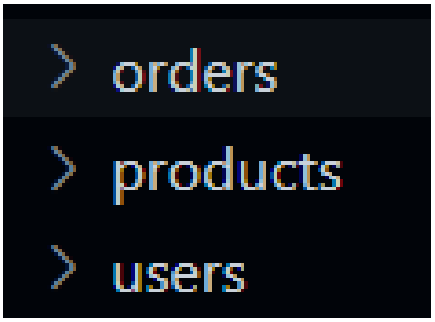
pagination.py (manually created): It is a file or module that is often used in web programming to handle the division and presentation of large data sets in several pages.

setitngs.py: key configurations are defined, such as the database that is being used, the application configuration, the Middleware's, the configuration of static files, the installed applications, the authentication configuration, the language, the time zone, and Other essential parameters for the operation of the web application.

urls.py: This file is used to define the routes or URLs of the application and map those routes to the functions or views that will oversee handling the incoming requests.

wsgi.py: It is a file in Django projects that acts as an entry point for web servers compatible with the WSGI specification.

A quick view of Python applications that are required for the operation of the website will be shown:

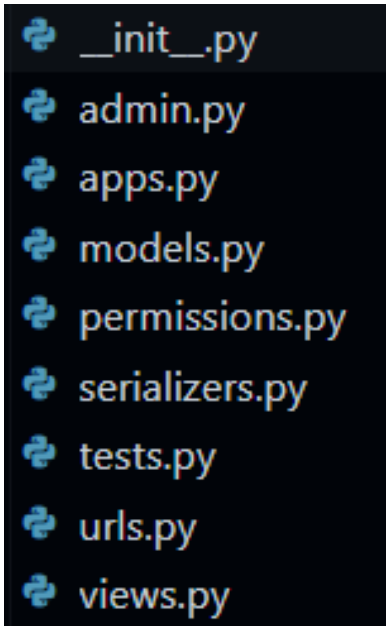


Application 'orders': This application will be responsible for carrying out operations in relation to the purchase, cart, and information shipping information.

Application 'products': This application will oversee storing information about the items/products that are uploaded to the store, it will also support the website that users can publish reviews to a specific product.

Application 'users': This application is the most important, since it will be the one that will do the user creation tasks, login, and will be the basis for generating the authentication token of new users. You can also identify whether the user is an administrator or not, which will allow the online store's raw control.

Each application will contain the following python files:



admin.py: This file allows you to register models so that they can be managed through the Django admin site.

apps.py: This file primarily contains the application configuration class that inherits from django.apps.AppConfig. This class provides metadata and configurations for the Django app.

models.py: File is a core component within each app's directory. It defines the data models for the app using Django's Object-Relational Mapping (ORM) system.

permissions.py: Permissions in Django determine whether a user can perform certain actions or access certain parts of an application.

serializers.py: Is a file used in building APIs to serialize and deserialize complex data types, such as Django model instances or querysets, into Python data types that can be easily rendered into JSON, XML, or other content types.

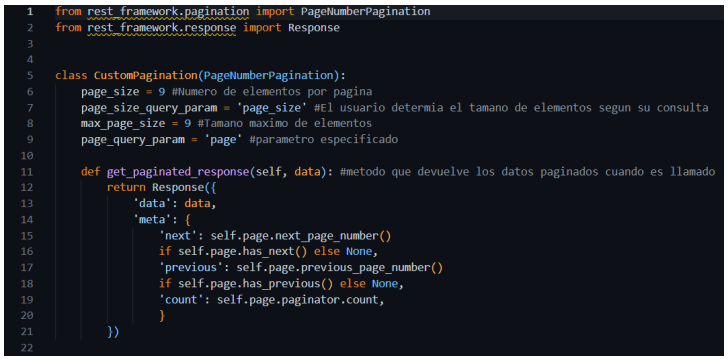
tests.py: Is a file is a standard location within each app's directory where you can define test cases and run unit tests for that specific app.

urls.py: is a file used to define URL patterns and route incoming requests to the appropriate views or resources within your Django application.

views.py: Is a file within each app's directory contains Python functions or classes that handle HTTP requests and generate HTTP responses.

Backend internal code

/backend/pagination.py



```python
from rest_framework.pagination import PageNumberPagination
from rest_framework.response import Response


class CustomPagination(PageNumberPagination):
    page_size = 9 #Numero de elementos por pagina
    page_size_query_param = 'page_size' #El usuario determia el tamano de elementos segun su consulta
    max_page_size = 9 #Tamano maximo de elementos
    page_query_param = 'page' #parametro especificado

    def get_paginated_response(self, data): #metodo que devuelve los datos paginados cuando es llamado
        return Response({
            'data': data,
            'meta': {
                'next': self.page.next_page_number()
                if self.page.has_next() else None,
                'previous': self.page.previous_page_number()
                if self.page.has_previous() else None,
                'count': self.page.paginator.count,
                }
        })

```

This code defines a custom pagination class that adjusts the pagination behavior in Django REST Framework views by providing additional metadata and returning the paginated data in a specific format.

/backend/settings.py

`path:` allows us to manipulate file paths, images, etc.
`timedelta:` Represents a specific time duration.
`os, sys:` Python provides an interface that interacts with the operating system.
`dotenv:` Allow loading of .env environment variables

```python
from pathlib import Path #nos permite manipular rutas de archivo, imagenes, etc
from datetime import timedelta #Representa una duracion de tiempo en especifico
import os,sys #Python proporcina una interfaz que interactua con el sistema operativo
from dotenv import load_dotenv #permite la carga de variables de entorno .env
```

`SECRET KEY`: Adds security to the application, without it, the server does not run.
`ALLOWED_HOST`: With the "*", it allows us to connect to the Django server from another pc.

```python
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get("SECRET_KEY") #anade seguiridad a la aplicacion, sin ella, la app no ejecuta

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ["*"] #con el "*", nos permite conectarnos al servidor de django desde otra pc
```

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework_simplejwt.token_blacklist',
    'users',
    'products',
    'orders',
]
```

It is a list that specifies which applications are available and active in a particular Django project, allowing its functionalities to be used in its development.

In this case, the **`rest_framework`** applications have been grouped, and the personalized apps, in this case they are **`users`**, **`products`** and **`orders.`**

```
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.mysql',
'NAME': 'mmx',
'USER': 'usuarionata',
'PASSWORD': 'nata123',
'HOST': 'localhost',
'PORT' : '3306',
}
}
```

The server database is provided by the MySQL database management system. Information about the database **name**, user and **password**, **host**, and **port** is also displayed.

This configuration allows us to connect to the MySQL server.

These settings allow Django to serve and store static files, such as CSS style sheets, JavaScript files, images, etc., efficiently, while **MEDIA** is used to store and serve media files uploaded by users, such as profile images, attachments, etc.

CORS_ALLOW_ALL_ORIGINS = True: This setting allows all cross-origin requests to be allowed from any origin. Basically, any website (origin) can make requests to your Django server without CORS restrictions.

CORS_ALLOW_CREDENTIALS = True: This setting allows the server to include credentials (such as cookies, authentication headers, etc.) in cross-origin requests.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'dist/static')
]
MEDIA_URL = 'media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

AUTH_USER_MODEL = "users.User"

CORS_ALLOW_ALL_ORIGINS = True

CORS_ALLOW_CREDENTIALS = True
```

These configurations allow you to customize the behavior of JWT token-based authentication in the Django REST Framework Simple JWT, defining lifetimes, signing algorithms, keys, claims, among other key aspects of authentication.

```
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=50),
    'REFRESH_TOKEN_LIFETIME': timedelta(minutes=50),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
    'UPDATE_LAST_LOGIN': False,

    'ALGORITHM': 'HS256',

    'VERIFYING_KEY': SECRET_KEY,
    'AUDIENCE': None,
    'ISSUER': None,
    'JWK_URL': None,
    'LEEWAY': 0,

    'AUTH_HEADER_TYPES': ('Bearer',),
    'AUTH_HEADER_NAME': 'HTTP_AUTHORIZATION',
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
    'USER_AUTHENTICATION_RULE': 'rest_framework_simplejwt.authentication.default_user_authentication_rule',

    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',
    'TOKEN_USER_CLASS': 'rest_framework_simplejwt.models.TokenUser',

    'JTI_CLAIM': 'jti',
}
```

/backend/urls.py

This file defines URL paths that direct incoming HTTP requests to the corresponding views in Django applications. In this case, the application routes for our project have been defined.

```
from django.contrib import admin
from django.urls import path, include, re_path
from django.views.generic import TemplateView
from django.conf import settings
from django.conf.urls.static import static


#Con path, indicamos la ubicacion de las paginas mediante las urls de la aplicaciones
urlpatterns = [
    path('admin/', admin.site.urls),
    path('users/', include('users.urls')),
    path('products/', include('products.urls')),
    path('orders/', include('orders.urls')),
]

#urlpatterns += [re_path(r'^.*', TemplateView.as_view(template_name='index.html'))]
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

/orders/models.py

models.py is a file in an application directory that contains database model definitions. Models in Django are Python classes that represent tables in the database. Each attribute of the model class represents a column in the database table.

In **models.py** of the orders application, there are the classes that will be related to the orders that are generated on the page. The user attribute is declared as ForeignKey because it will be related to an order number, since a user can have several orders.

The **`order`** class will store the order that is generated for the administrator once the user has made the payment.

The **`Orderitem`** class is what the order includes, such as the products, order number, quantity, and total price.

The **`ShippingAddress`** class will save information about the address of the user who entered to generate the order.

```python
from django.db import models
from users.models import User
from products.models import Product


class Order(models.Model):
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    total_price = models.CharField(max_length=250, blank=True)
    is_delivered = models.BooleanField(default=False)
    delivered_at = models.DateTimeField(auto_now_add=False, null=True, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)


class Orderitem(models.Model):
    product = models.ForeignKey(Product, on_delete=models.SET_NULL, null=True)
    order = models.ForeignKey(Order, on_delete=models.SET_NULL, null=True)
    quantity = models.IntegerField(null=True, blank=True, default=0)
    price = models.CharField(max_length=250, blank=True)


class ShippingAddress(models.Model):
    order = models.OneToOneField(Order, on_delete=models.CASCADE, null=True, blank=True)
    address = models.CharField(max_length=250, blank=True)
    city = models.CharField(max_length=100, blank=True)
    postal_code = models.CharField(max_length=100, blank=True)
```

/orders/serializers.py

```python
from rest_framework import serializers
from .models import Order, Orderitem, ShippingAddress


class ShippingSerializer(serializers.ModelSerializer):
    class Meta:
        model = ShippingAddress
        fields = '__all__'


class OrderItemSerializer(serializers.ModelSerializer):

    product = serializers.ReadOnlyField(source='product.name')

    class Meta:
        model = Orderitem
        fields = '__all__'


class OrderSerializer(serializers.ModelSerializer):

    user = serializers.ReadOnlyField(source='user.email')
    order_items = serializers.SerializerMethodField(read_only=True)
    shipping_address = serializers.SerializerMethodField(read_only=True)

    class Meta:
        model = Order
        fields = '__all__'

    def get_order_items(self, obj):
        items = obj.orderitem_set.all()
        serializer = OrderItemSerializer(items, many=True)
        return serializer.data

    def get_shipping_address(self, obj):
        try:
            address = ShippingSerializer(
                obj.shippingaddress, many=False).data
        except:
            address = False
        return address
```

This file is serialized to the **`ShippingAddress`**, **`product`**, and **`Orderitem`** classes to become a JSON and can be rendered in any other external application. In this case, this is done so that it can be displayed on the frontend from TypeScript.

In **`OrderSerializers`**, defines how an **`Order`** object will be represented when serialized to a format such as JSON on an API endpoint. Include specific fields and use custom methods to obtain and represent additional information associated with each order, such as order items and shipping address.

Additionally, through a function, you obtain the shipping address associated with this order (obj). Attempts to obtain the shipping address of the order and, if there is an address, serializes it using the **`ShippingSerializer`** to return the serialized shipping address data.

/orders/urls.py

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

urlpatterns = [
    path('', views.get_orders),
    path('search/', views.search),
    path('create/', views.create_order),
    path('my/orders/', views.my_orders),
    path('deliver/<int:pk>/', views.delivered),
    path('solo/<int:pk>/', views.solo_order),
]
```

The **urls** that will be related to the orders on the page are defined, the routes will be made for the **`search`**, **`creation`** of orders and the **`orders`** of a user.

The **`delivery`** route and **`solo_order`** will be only for administrators, where the information about the order of a specific user will be displayed.

/orders/views.py

In this first part of the code, **APIs** are created so that they can be connected to other external applications.

The API with the **HTTP** 'GET' method **`@permision_clases`**, searches for orders based on the user's email provided in the query parameter 'query' and returns the results as a JSON response with the serialized data of the orders found. Additionally, only users with the administrator role have permission to access this view. The following API performs the same function as the previous one, only this API view gets all the requests from the database.

The API with the **HTTP** 'PUT' method **`@permision_clases`** view handles POST requests to create a new order, verifying user authentication and performing operations related to order creation in the database.

```python
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated, IsAdminUser
from rest_framework.response import Response
from rest_framework import status
from datetime import datetime

from .models import Order, Orderitem, ShippingAddress
from .serializers import OrderSerializer
from products.models import Product


@api_view(['GET'])
@permission_classes([IsAdminUser])
def search(request):
    query = request.query_params.get('query')
    if query is None:
        query = ''
    order = Order.objects.filter(user__email__icontains=query)
    serializer = OrderSerializer(order, many=True)
    return Response({'orders': serializer.data})


@api_view(['GET'])
@permission_classes([IsAdminUser])
def get_orders(request):
    orders = Order.objects.all()
    serializer = OrderSerializer(orders, many=True)
    return Response(serializer.data)


@api_view(['POST'])
@permission_classes([IsAuthenticated])
def create_order(request):
    user = request.user
    data = request.data
    orderItems = data['order_items']
    total_price = data['total_price']

    sum_of_prices = sum(int(float(item['price'])) * item['quantity'] for item in orderItems)

    if total_price == sum_of_prices:
        order = Order.objects.create(
            user=user,
            total_price=total_price
        )
```

The first **API** view gets specific details of an order based on its primary key (**pk**) and checks if the authenticated user has permission to view that order. If the order exists and the user has the appropriate permissions, the serialized order information is returned. If the conditions are not met, responses are sent with appropriate messages and corresponding **HTTP** status codes.

The second **API** view gets all orders associated with the user who made the request.

The third view of the **API** allows an administrator to mark a specific order as delivered. Checks if the user has administrator permissions, updates the delivery status of the order, and returns a message indicating that the order has been delivered successfully.

```python
@api_view(['GET'])
@permission_classes([IsAuthenticated])
def solo_order(request, pk):
    user = request.user
    try:
        order = Order.objects.get(pk=pk)
        if user.is_staff or order.user == user:
            serializer = OrderSerializer(order, many=False)
            return Response(serializer.data)
        else:
            Response({'detail': 'No access to view orders'},
                     status=status.HTTP_401_UNAUTHORIZED)
    except:
        return Response({'detail': 'Order does not exist'}, status=status.HTTP_400_BAD_REQUEST)


@api_view(['GET'])
@permission_classes([IsAuthenticated])
def my_orders(request):
    user = request.user
    orders = user.order_set.all()
    serializer = OrderSerializer(orders, many=True)
    return Response(serializer.data)


@api_view(['PUT'])
@permission_classes([IsAdminUser])
def delivered(request, pk):
    order = Order.objects.get(pk=pk)
    order.is_delivered = True
    order.delivered_at = datetime.now()
    order.save()
    return Response('Order was delivered')
```

/products/models.py

In **models.py** of the products application, two classes are created that will be related to the products in the store and the reviews that users leave about a specific product.

In **`class Product`**, information about the product is saved, such as the name, the user (this is in reference to the fact that a product can be purchased by several users), the image of the item, category, description, rating, number of reviews, price, quantity in stock, and creation date.

In **`class Reviews`**, the information is saved according to the product, so this attribute is declared as Foreign Key, user of the person who makes the review, rating, opinion or comment and the date of creation of the review.

```python
class Product(models.Model):
    slug = models.SlugField(max_length=50, null=True, blank=True)
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    name = models.CharField(max_length=100, blank=True)
    image = models.ImageField(default='placeholder.png')
    category = models.CharField(max_length=100, blank=True)
    description = models.CharField(max_length=100, blank=True)
    rating = models.DecimalField(max_digits=10,
                                 decimal_places=2,
                                 null=True, blank=True)
    num_reviews = models.IntegerField(default=0)
    price = models.DecimalField(max_digits=10,
                                decimal_places=2,
                                null=True, blank=True)
    count_in_stock = models.IntegerField(default=0)
    created = models.DateTimeField(auto_now_add=True)


class Reviews(models.Model):
    product = models.ForeignKey(Product, on_delete=models.SET_NULL, null=True)
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    rating = models.DecimalField(max_digits=10,
                                 decimal_places=2,
                                 null=True, blank=True)
    description = models.CharField(max_length=100, blank=True)
    created = models.DateTimeField(auto_now_add=True)
```

/products/urls.py

The routes created for the products app are configured for different product-related actions, such as search, get, create, edit, delete, and manage revisions.

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

urlpatterns = [
    path('search/', views.search),
    path('', views.get_products),
    path('get/<slug:slug>/', views.get_product),
    path('get/admin/<int:id>/', views.get_product_admin),
    path('post/', views.create_product),
    path('edit/<int:pk>/', views.edit_product),
    path('delete/<int:pk>/', views.delete_product),
    path('cate/<str:category>/', views.get_prod_by_cate),

    path('review/<int:pk>/', views.create_review),
]
```

/products/views.py

In the product views, the APIs are created that will serve the CRUD functionality in relation to the products, such as consulting the name, quantity, category, description, and image.

This API view is more focused on the page administrator, for example, different APIS are created with PUT, GET, DELETE, AND POST methods.

DELETE: allows us to delete an item from the store.

POST: allows you to create a product and publish it in the store.

PUT: allows us to replace the information of an existing product

GET: with this method, we obtain information about a product from all its data or those that are required

```python
from rest_framework.response import Response
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated
from django.utils.text import slugify
from rest_framework import status

from . models import Product
from . serializers import ProductSerializer, ReviewSerializer
from backend.pagination import CustomPagination


@api_view(['POST'])
@permission_classes([IsAuthenticated])
def create_review(request, pk):
    serializer = ReviewSerializer(data=request.data)
    product = Product.objects.get(pk=pk)
    if serializer.is_valid():
        serializer.save(user=request.user, product=product)
        return Response(serializer.data, status=status.HTTP_201_CREATED)
    else:
        return Response(serializer.data, status=status.HTTP_400_BAD_REQUEST)


@api_view(['GET'])
def get_prod_by_cate(request, category):
    products = Product.objects.filter(category=category)
    serializer = ProductSerializer(products, many=True)
    return Response(serializer.data)


@api_view(['GET'])
def search(request):
    query = request.query_params.get('query')
    if query is None:
        query = ''
    product = Product.objects.filter(name__icontains=query)
    serializer = ProductSerializer(product, many=True)
    return Response({'products': serializer.data})
```

```python
@api_view(['GET'])
def get_products(request):
    products = Product.objects.all()
    paginator = CustomPagination()
    paginated_products = paginator.paginate_queryset(products, request)
    serializer = ProductSerializer(paginated_products, many=True)
    return paginator.get_paginated_response(serializer.data)


@api_view(['GET'])
def get_product_admin(request, id):
    products = Product.objects.get(id=id)
    serializer = ProductSerializer(products, many=False)
    return Response(serializer.data)


@api_view(['GET'])
def get_product(request, slug):
    products = Product.objects.get(slug=slug)
    serializer = ProductSerializer(products, many=False)
    return Response(serializer.data)


@api_view(['POST'])
def create_product(request):
    if request.user.is_staff:
        serializer = ProductSerializer(data=request.data)
        if serializer.is_valid():
            name = serializer.validated_data['name']
            category = serializer.validated_data['category']
            s = name + category
            slug = slugify(s)
            if serializer.Meta.model.objects.filter(slug=slug).exists():
                return Response(serializer.data, status=status.HTTP_400_BAD_REQUEST)
            serializer.save(user=request.user, slug=slug)
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.data, status=status.HTTP_400_BAD_REQUEST)
    else:
        return Response(serializer.data, status=status.HTTP_401_UNAUTHORIZED)
```

```python
@api_view(['PUT'])
def edit_product(request, pk):
    product = Product.objects.get(pk=pk)
    if request.user.is_staff:
        serializer = ProductSerializer(product, data=request.data)
        if serializer.is_valid():
            name = serializer.validated_data['name']
            category = serializer.validated_data['category']
            s = name + category
            slug = slugify(s)
            serializer.save(user=request.user, slug=slug)
            return Response(serializer.data)
        return Response(status=status.HTTP_400_BAD_REQUEST)
    else:
        return Response(status=status.HTTP_401_UNAUTHORIZED)


@api_view(['DELETE'])
def delete_product(request, pk):
    product = Product.objects.get(pk=pk)
    if request.user.is_staff:
        product.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
    else:
        return Response(status=status.HTTP_401_UNAUTHORIZED)
```

/users/models.py

In **models.py** of the user application, the first **`CustomUserManager`** class is created, where the user's essential information will be saved: email, first name, last name, image (this is added by default by the system), creation date (assigned in automatic), a verification if the user is an administrator or not. As a requirement, it is mandatory to enter a valid email.

In this file, import the users who are registered from the Django administrator, so they will be able to log in to the store, and when validated as administrators, they will be able to view the **'Settings'** section in the store bar.

To carry out this verification, the **`is_staff`** attribute of type Boolean will change according to the origin of the account, if it comes from Django users, it will be **`True`**, so it will be an administrator. If the account comes from the website, it will be a user, so the value will be **`False`**, and it will not have administrator permissions.

```python
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import (
    AbstractBaseUser,
    PermissionsMixin,
    UserManager
)


class CustomUserManager(UserManager):
    def _create_user(self, email, password, **extra_fields):
        if not email:
            raise ValueError("Debes tener un correo electronico")

        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)

        return user

    def create_user(self, email=None, password=None, **extra_fields):
        extra_fields.setdefault("is_staff", False)
        return self._create_user(email, password, **extra_fields)

    def create_superuser(self, email=None, password=None, **extra_fields):
        extra_fields.setdefault("is_staff", True)
        return self._create_user(email, password, **extra_fields)


class User(AbstractBaseUser, PermissionsMixin):
    email = models.CharField(max_length=100, unique=True)
    name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    avatar = models.ImageField(default="avatar.png")
    date_joined = models.DateTimeField(default=timezone.now)
    is_staff = models.BooleanField(default=False)
    objects = CustomUserManager()
    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = []

    class Meta:
        ordering = ["-date_joined"]
```

/users/urls.py

The **user application** routes are in relation to functions such as creating, editing, deleting, and obtaining data from an account. As well as the '**login/**', where the account can be logged in and the '**register/**' route, where a user can create an account.

The '**refresh/**' route is a security measure that has been implemented on the site. If a user or administrator has an active session and has not done any action within the site in the next 5 minutes, the session will be automatically closed.

```python
from django.urls import path
from . import views
from rest_framework_simplejwt.views import TokenRefreshView

urlpatterns = [
    path('search/', views.search),
    path('register/', views.register),
    path('login/', views.LoginView.as_view()),
    path('refresh/', TokenRefreshView.as_view()),
    path('get/', views.get_users),
    path('delete/<int:pk>/', views.delete_user),
    path('edit/<str:email>/', views.edit_profile),
    path('get/solo/<int:pk>/', views.get_solo_user),
]
```

/users/views.py

In the user views, the views of the APIs that will have the **HTTP** methods to perform the different operations in relation to the users are saved.

The first API uses the '**GET**' method that will obtain the user according to their primary key.

The second API uses the '**PUT**' method that will modify the information of an account depending on the email, it has validations in case the account does not exist or does not have access to modify said information.

```python
from rest_framework_simplejwt.views import TokenObtainPairView
from django.contrib.auth.hashers import make_password
from rest_framework.response import Response
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated
from rest_framework import status

from . models import User
from . serializers import RegisterUserSerializer, MyTokenObtainPairSerializer, UserSerializer


@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_solo_user(request, pk):
    user = User.objects.get(pk=pk)
    serializer = UserSerializer(user)
    return Response(serializer.data)


@api_view(['PUT'])
def edit_profile(request, email):
    try:
        user = User.objects.get(email=email)
    except User.DoesNotExist:
        return Response(status=status.HTTP_404_NOT_FOUND)

    if request.user == user:
        serializer = UserSerializer(user, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
    else:
        return Response(status=status.HTTP_401_UNAUTHORIZED)
```

In the following image, there are the API views that will perform the operations and will be used for both administrators and users.

DELETE: will allow you to delete an account. This action is only allowed for the administrator.

GET: from the **CRUD**, the administrator will obtain the email, first and last name of the users who have registered on the site.

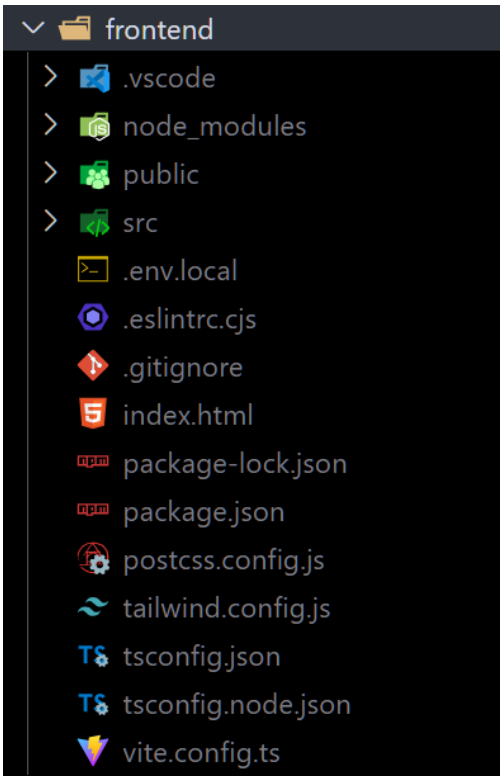
POST: this method will create an account from the '**`register/`**' route.

The last class called **`LoginView`** will be the one that provides security when a user or administrator logs in, it will be required to obtain the user's token to verify its authenticity.

```python
@api_view(['GET'])
def search(request):
    query = request.query_params.get('query')
    if query is None:
        query = ''
    user = User.objects.filter(email__icontains=query)
    serializer = UserSerializer(user, many=True)
    return Response({'users': serializer.data})


@api_view(['DELETE'])
def delete_user(request, pk):
    user = User.objects.get(pk=pk)
    if request.user.is_staff:
        user.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
    return Response(status=status.HTTP_401_UNAUTHORIZED)


@api_view(['GET'])
def get_users(request):
    if request.user.is_staff:
        users = User.objects.exclude(email='admin@admin.com')
        serializer = UserSerializer(users, many=True)
        return Response(serializer.data)
    return Response(serializer.data, status=status.HTTP_401_UNAUTHORIZED)


@api_view(['POST'])
def register(request):
    data = request.data
    user = User.objects.create(
        email=data['email'],
        name=data['name'],
        last_name=data['last_name'],
        password=make_password(data['password'])
    )
    serializer = RegisterUserSerializer(user, many=False)
    return Response(serializer.data)


class LoginView(TokenObtainPairView):
    serializer_class = MyTokenObtainPairSerializer
```

6 Appendix C: Analysis Frontend

The structure of the virtual store's frontend is comprised of several key files. Below, an overview of each file in the 'Frontend' project will be provided. These files play specific roles in ensuring proper functionality and the visual appearance of the user interface. The frontend is developed using ReactJS, with additional support from Tailwind for stylesheets, HTML5, and Node.js.



Main files and folders in front end:

.vscode: User configuration for Visual Studio Code.
Defines editor preferences and specific settings.

node_modules: Contains Node.js modules used in the project.
Node modules are importable code packages in Node.js programs.

public: Holds files served directly to users.
May include HTML, CSS, and JavaScript files for the web interface.

.env.local: Contains environment variables to configure the project (e.g., database address).

.eslintrc.cjs: ESLint configuration, a code linter detecting errors and style issues in JavaScript.

gitignore: List of files/directories ignored when tracking changes in Git.

index.html: Main web page of the application.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <!-- Set the character set for the document -->
  <meta charset="UTF-8" />

  <!-- Set the viewport to control layout on different devices -->
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Set the title of the document -->
  <title>MX Discovery - Museo y Arte</title>

  <!-- Set the favicon for the website -->
  <link rel="icon" type="image/png" sizes="16x16" href="/favicon.png">
</head>

<body class="">
  <!-- Root element where the React application will be mounted -->
  <div id="root"></div>

  <!-- Include the main TypeScript file for the React application -->
  <script type="module" src="/src/main.tsx"></script>
</body>

</html>
```

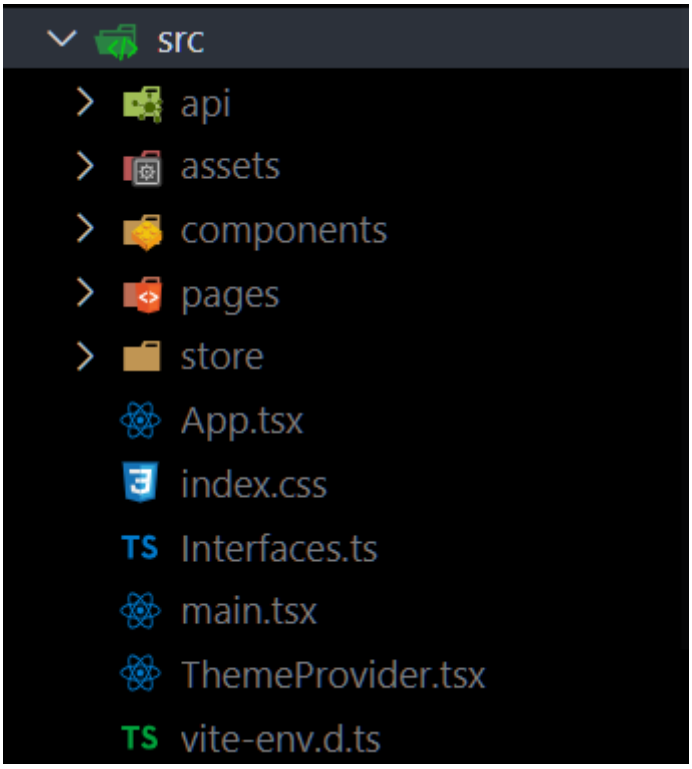
package-lock.json: Lists Node.js modules and their specific versions.

package.json: Project information (name, version, dependencies).

postcss.config.js: PostCSS configuration, a CSS preprocessor.

tailwind.config.js: Tailwind CSS configuration, a framework for web interfaces.

src: Contains five folders and six files.



app.tsx:

The file app.tsx is the main component of the React application. It is used to define the structure and behavior of the application. This file typically contains code to render the web interface of the application, as well as code to handle user events.

index.css:

The file index.css contains the CSS styles for the application. It is used to define the appearance of the application. This file usually contains CSS rules to define the font, size, color, and layout of the web interface elements.

interfaces.ts:

The file interfaces.ts defines interfaces for the components and classes of the application. It is used to ensure code consistency. Interfaces define the data types of the components and classes of the application. This helps prevent errors and makes the code more understandable.

main.tsx:

The file main.tsx is the entry point of the application. It is used to start the application and load the necessary components. This file typically contains code to initialize the React environment and load the application's components.

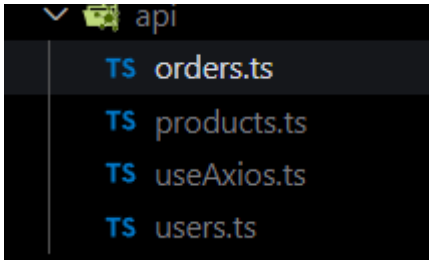
theme-provider.tsx:

The file theme-provider.tsx provides a theme for the application. It is used to centralize the styling configuration of the application. This file usually contains code to define colors, fonts, and other styles of the application.

vite-env.d.ts:

The file vite-env.d.ts contains type definitions for the Vite environment. It is used to facilitate the use of Vite modules from TypeScript. This file typically contains type definitions for objects and functions provided by the Vite environment.

api: contains four files.



/frontend/src/api/orders.ts:

```
// Allows editing an order based on its order number (only for administrators)
export const edit_order = async (id: number) => {
    await authAxios.put(`/orders/deliver/${id}/`);
};

// Retrieves all orders in general
export const get_orders = async () => {
    const response = await authAxios.get(`/orders/`);
    return response.data;
};

// Allows selecting a specific order based on its order number
export const solo_order = async (id: number) => {
    const response = await authAxios.get(`/orders/solo/${id}/`);
    return response.data;
};

// Allows a user to view their own orders
export const my_orders = async () => {
    const response = await authAxios.get('orders/my/orders/');
    return response.data;
};

// Creates an order when the user selects "save to cart"
export const create_order = async (data: Order) => {
    await authAxios.post('/orders/create/', data);
};
```

/frontend/src/api/products.ts:

In this file, you will find all the functions and the general code for conducting searches, creating reviews, selecting categories, obtaining all the information about a specific product, modifying the specific product, deleting a product, etc.

```
import { Product } from "../Interfaces";
import { authAxios, axi } from "./useAxios";

//funcion to allow the creation of a new review
export const create_review = async (
  description: string,
  rating: number,
  productId: number
) => {
  await authAxios.post(`/products/review/${productId}/`, {
    description,
    rating,
  });
};

//allow to show and select a new category
export const cate_api = async (cateogry: string) => {
  const response = await authAxios.get(`/products/cate/${cateogry}/`);
  return response.data;
};
```

```
//Initiates a search request to find a specific product based on the query entered in the search bar
export const search_prod = async (query: string) => {
  const response = await authAxios.get(`/products/search/?query=${query}`);
  return response.data;
};

//"It obtains the general name of the products."
export const get_solo = async (slug: string) => {
  const response = await authAxios.get(`/products/get/${slug}/`);
  return response.data;
};

//This function allows obtaining the complete information of a specific product.
export const get_solo_prod = async (id: number) => {
  const response = await authAxios.get(`/products/get/admin/${id}/`);
  return response.data;
};

//Function that allows modifying the information of a product, such as name, description, etc.
export const edit_product = async (data: Product) => {
  const formData = new FormData();
  formData.append("name", data.name);
  formData.append("description", data.description);
  formData.append("count_in_stock", data.count_in_stock.toString());
  formData.append("category", data.category);
  formData.append("price", data.price.toString());
  if (data.image) {
    formData.append("image", data.image);
  }
  await authAxios.put(`/products/edit/${data.id}/`, formData);
};
```

```
//Allows deleting a specific product.
export const delete_product = async (id: number) => {
  await authAxios.delete(`/products/delete/${id}/`);
};

//Same parameters as the edit function, but here the information is published, and a new product is added.
export const post_product = async (data: Product) => {
  const formData = new FormData();
  formData.append("name", data.name);
  formData.append("description", data.description);
  formData.append("count_in_stock", data.count_in_stock.toString());
  formData.append("category", data.category);
  formData.append("price", data.price.toString());
  if (data.image) {
    formData.append("image", data.image);
  }
  await authAxios.post("/products/post/", formData);
};

export const get_products = async ({ pageParam = 1 }) => {
  const response = await axi.get(`/products/?page=${pageParam}&pages=9`);
  return response.data;
};
```

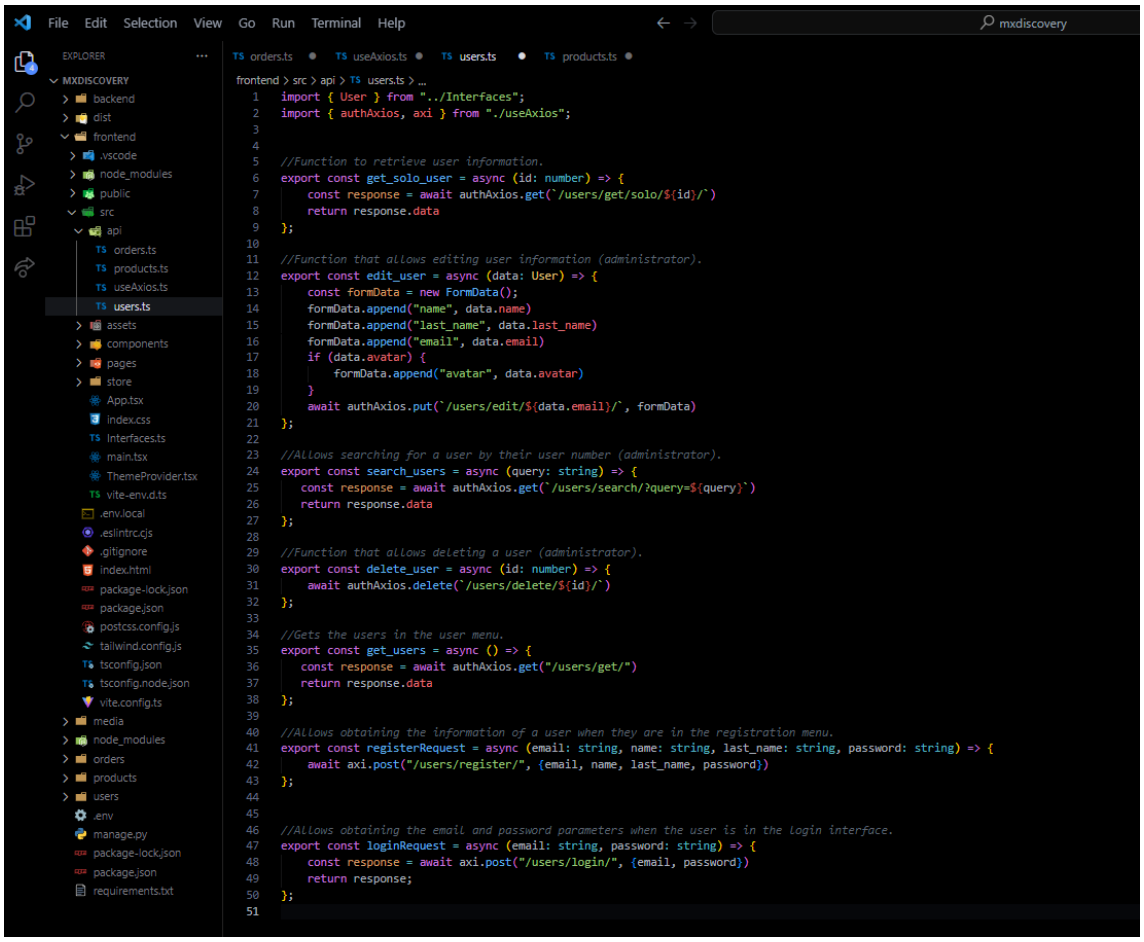
/frontend/src/api/userAxios.ts:

It is used to communicate with the backend and validate the login. When attempting to log in, the security token is verified since the user has their account protected. Not even the developer can view the password, thanks to the randomly generated token that safeguards user passwords.

```
import axios, { AxiosRequestHeaders } from "axios";
import { useAuthStore } from "../store/auth"; // Call a function to determine if the user is an administrator in the store.
import jwt_decode from "jwt-decode"; // Allows encoding tokens using the JWT library.
import { Token } from "../Interfaces"; //Imports to "Interfaces" to generate the token.


//Enables logging out by calling the "logout" function.
function logout() {
    useAuthStore.getState().logout()
    window.location.href = '/login'
}

//Provides a link between the baseURL and the environment variable.
const baseURL = import.meta.env.VITE_BACKEND_URL

export const axi = axios.create({
    baseURL
});

export const authAxios = axios.create({
    baseURL,
    withCredentials: true
});

// With this code snippet, using a user's token, it allows evaluating
// the time that this user or administrator will have their session open.
// Once the 5-minute time limit expires, the page will automatically refresh,
// and the user will be returned to the login page.
authAxios.interceptors.request.use(async (config) => {
    const token: string = useAuthStore.getState().access;
    config.headers = {
        Authorization: `Bearer ${token}`,
    } as AxiosRequestHeaders;

    const tokenDecoded: Token = jwt_decode(token)

    const expiration = new Date(tokenDecoded.exp * 1000); // Token expiration.
    const now = new Date(); // The token expiration starts from the current time.
    const fiveMin = 1000 * 60 * 5; // Represents 5 minutes in milliseconds, functioning as a timer.

    if (expiration.getTime() - now.getTime() < fiveMin)
        try {
            const response = await axi.post('/users/refresh/', { refresh: useAuthStore.getState().refresh })
            useAuthStore.getState().setToken(response.data.access, response.data.refresh)
        } catch (err) {
            logout()
        }
        return config
});
```

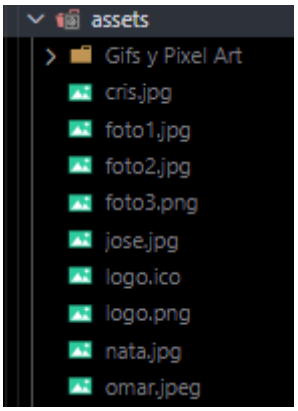
/frontend/src/api/users.ts:

It is used to obtain information about users, allowing for editing names, surnames, deleting, or creating users in administrator mode.



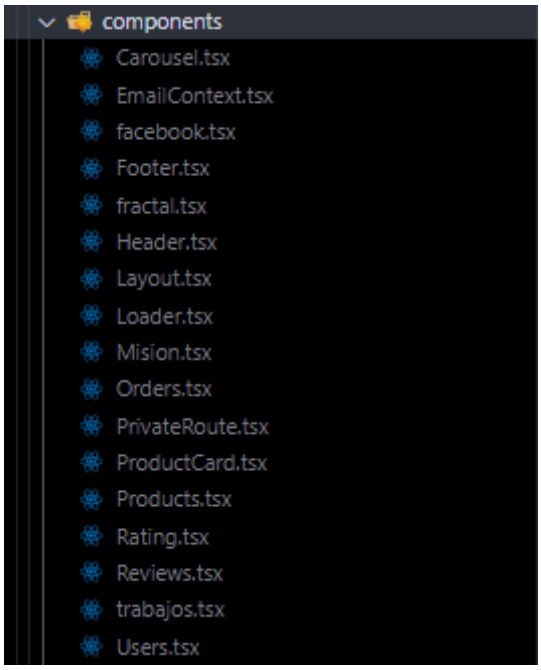
/frontend/src/assets:

This folder contains all the photos and media that will be used on the website to decorate it.



/frontend/src/components:

In the components folder, as the name suggests, it houses all the components of the page, each of which has its own function and style sheet, the folder has seventeen files.



/frontend/src/components/Carousel.tsx:

This component contains the code to create a carousel, which will be called later on a page or route.

The component has tree images and it is created with Slider, a npm complement.

/frontend/src/components/EmailContext.tsx:

In the second component, the email functionality is created using a small framework called emails to send emails to the company. A form is created with three text inputs: one for the email, one for the name, and another for the message.

The code includes a function to send the email, another function to reset the form when one is sent (clearing the text), and an alert indicating whether the message was successfully sent or not.

```
import { useRef, useState, LegacyRef, useEffect } from 'react';
import emailjs from '@emailjs/browser';

export const ContactUs = () => {
    const form = useRef<HTMLFormElement>(null);
    const [isSubmitted, setIsSubmitted] = useState(false);
    const [isError, setIsError] = useState(false);
    const [alertMessage, setAlertMessage] = useState('');

    const sendEmail = (e: { preventDefault: () => void; }) => {···
    };

    const resetForm = () => {···
    };

    useEffect(() => {···
    }, [alertMessage]);

    return (···
    );
};
```

/frontend/src/components/facebook.tsx

It calls a widget to display Facebook posts, which uses a script from an external page.

```tsx
import React, { useEffect } from 'react';

const FacebookPage = () => {
    useEffect(() => {
        const script = document.createElement('script');
        script.src = 'https://static.elfsight.com/platform/platform.js';
        script.defer = true;
        script.setAttribute('data-use-service-core', 'true');

        document.body.appendChild(script);

        // add styles
        const style = document.createElement('style');
        style.innerHTML = `
            .elfsight-app-1cdb1302-4e5b-4900-bfae-bc64750fb0e9 .elfsight-popup {
                display: none !important;
            }
        `;
        document.head.appendChild(style);

        return () => {
            // Cleaning up the script on component unmount to avoid issues with multiple loading.
            document.body.removeChild(script);
            // Cleaning up the styles on component unmount.
            document.head.removeChild(style);
        };
    }, []);

    return (
        <div className="elfsight-app-1cdb1302-4e5b-4900-bfae-bc64750fb0e9" data-elfsight-app-lazy />
    );
};

export default FacebookPage;
```

/frontend/src/components/Footer.tsx:

It contains a footer element that displays links to different types of social media along with their respective styles using Tailwind CSS.

/frontend/src/components/fractal.tsx:

It contains code to draw a fractal as a component using a JavaScript framework called p5.js.

```tsx
import React, { useEffect, useRef } from 'react';
import p5 from 'p5';
import 'tailwindcss/tailwind.css';

const FractalTreeComponent = () => {
    const fractalContainer = useRef(null);

    class FractalTree {
        position: p5.Vector;
        angle: number;
        branchLength: number;
        branchWidth: number;
        angleOffset: number;
        level: number;
        changeColor: any;

        constructor(position: p5.Vector, angle: number, branchLength:
            this.position = position.copy();
            this.angle = angle;
            this.branchLength = branchLength;
            this.branchWidth = branchWidth;
            this.angleOffset = angleOffset;
            this.level = level;
        }

        show(p: p5) {
            p.strokeWeight(this.branchWidth);
            p.stroke(0);
```

/frontend/src/components/Header.tsx

The code is a component responsible for managing the header section of a website. Its key functionalities include the ability to toggle between light and dark modes, handle user authentication, display navigation links, manage the user's shopping cart, and provide a search function.

```tsx
<div className="space-y-1 px-2 pb-3 pt-2">
  {isAuth ? (
    <div className="w-full grid grid-cols-1">
      <Link
        to={"/"}
        className="bg-slate-400 p-2 px-4 rounded-lg text-black dark:bg-gray-900 dark:text-white"
      >
        Inicio
      </Link>


      <Link
        to={"/Servicios"}
        className="text-black p-2 px-4 rounded-lg hover:bg-slate-400 dark:text-gray-300 dark:hover:bg-gray-700 dark:hover:text-white"
      >
        Servicios
      </Link>

      <Link
        to={"/Nosotros"}
        className="text-black p-2 px-4 rounded-lg hover:bg-slate-400 dark:text-gray-300 dark:hover:bg-gray-700 dark:hover:text-white"
      >
        Nosotros
      </Link>

      <Link
        to={"/Contacto"}
        className="text-black p-2 px-4 rounded-lg hover:bg-slate-400 dark:text-gray-300 dark:hover:bg-gray-700 dark:hover:text-white"
      >
        Contactanos
      </Link>
    </div>
  ) : (
    <div className="w-full grid grid-cols-1">
      <Link
        to={"/login"}
        className="text-black p-2 px-4 rounded-lg hover:bg-slate-400 dark:text-gray-300 dark:hover:bg-gray-700 dark:hover:text-white"
      >
        Iniciar Sesion
      </Link>

      <Link
        to={"/register"}
        className="text-black p-2 px-4 rounded-lg hover:bg-slate-400 dark:text-gray-300 dark:hover:bg-gray-700 dark:hover:text-white"
```

/frontend/src/components/Layout.tsx:

```tsx
import { Outlet } from "react-router-dom";
import Header from "./Header";
import { Toaster } from "react-hot-toast";

const Layout = () => {
  return (
    <div>
      <Toaster />
      <Header />
      <div className=" bg-white dark:bg-gray-950">
        <Outlet />
      </div>
    </div>
  );
};

export default Layout;
```

This component is responsible for the overall structure of the page, as it includes other components such as Header and Toaster. It also renders the main content through Outlet.

/frontend/src/components/Loader.tsx:

```tsx
const Loader = () => (
<div className="flex min-h-full items-center justify-center py-14  ">
    <div className=" m-5 p-10">
      <div className="w-full max-w-md space-y-8">
        <div className="flex justify-center items-center py-3">
          <div className="animate-spin rounded-full h-32 w-32 border-b-4 border-white" />
        </div>
      </div>
    </div>
  </div>
);
export default Loader;
```

It contains only HTML and Tailwind CSS code to create a simple loader.

/frontend/src/components/Mision.tsx:

Contains the information about us, and pictures, is only html and taildwind.

```tsx
import { useDarkMode } from "../store/theme";
import { Accordion, AccordionHeader, AccordionBody } from "@material-tailwind/react";

const Mision = () => {
    const { toggleDarkMode, darkMode } = useDarkMode();
    const [open, setOpen] = React.useState(1);

    const handleOpen = (value: React.SetStateAction<number>) => setOpen(open === value ? 0 : value);

    return (
        <>
            <div className={`flex items-center justify-center ${darkMode.valueOf() ? 'dark' : ''}`}>
                <div className="container p-4">
                    <article className={`rounded-xl ${darkMode.valueOf() ? 'bg-gray-900' : 'bg-gradient-to-r from-green-300 via-blue-500 to-purple-600'} p-0.5 flex items-center justify-center`}>
                        <div className={`rounded-[10px] ${darkMode.valueOf() ? 'bg-gray-900' : 'bg-white'} p-4 !pt-5 sm:p-6`}>
                            <h1 className={`mt-0.5 text-lg font-bold text-center ${darkMode.valueOf() ? 'text-white' : 'text-gray-900'} text-5xl`}>
                                Misión
                                <br />
                                <br />
                            </h1>
                            <p className={`text-xl ${darkMode.valueOf() ? 'text-gray-300' : 'text-gray-900'}`}>
                                La misión de MX DISCOVERY es ser un espacio cultural comprometido con la preservación, exhibición y promoción del patrimonio artístico y cultural, tanto nacional como internacional en

                            </p>
                        </div>
                    </article>
                </div>
            </div>

            {/* Asegúrate de cambiar el código para la sección de Visión de manera similar */}
            <div className={`flex items-center justify-center ${darkMode.valueOf() ? 'dark' : ''}`}>
                <div className="container p-4">
                    <article className={`rounded-xl ${darkMode.valueOf() ? 'bg-gray-900' : 'bg-gradient-to-r from-green-300 via-blue-500 to-purple-600'} p-0.5 flex items-center justify-center`}>
                        <div className={`w-full rounded-[10px] ${darkMode.valueOf() ? 'bg-gray-900' : 'bg-white'} p-4 !pt-5 sm:p-6`}>
                            <h1 className={`mt-0.5 text-lg font-bold text-center ${darkMode.valueOf() ? 'text-white' : 'text-gray-900'} text-5xl`}>
                                Visión
                                <br />
                                <br />
                            </h1>
                            <p className={`text-xl ${darkMode.valueOf() ? 'text-gray-300' : 'text-gray-900'}`}>
                                La visión de MX Discovery es ser un faro de cultura y conocimiento, un lugar donde las obras de arte cobren vida y las historias detrás de ellas se cuenten con pasión. Nuestra misión se resu

                                <p> 1.- Preservar y Celebrar el Arte: Nuestro objetivo principal es preservar y celebrar el arte en todas sus formas, desde las antiguas reliquias hasta las expresiones artísticas más contemporán

                                <p> 2.- Ofrecer Experiencias Enriquecedoras: Buscamos proporcionar experiencias enriquecedoras y educativas para nuestros visitantes, inspirándolos a explorar, aprender y apreciar el arte y la cultura.<
                            </p>
                        </div>
                    </article>
                </div>
            </div>
            <br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br /><br />
        </>
    );
};
```

/frontend/src/components/Orders.tsx:

This component is responsible for displaying a list of orders, allowing the editing of orders with a checkbox, and showing key details for each order. Additionally, it uses the react-query and react-hot-toast libraries to manage query and user feedback.

```
return (
  <div className="relative overflow-x-auto shadow-md sm:rounded-lg">
    <table className="w-full text-sm text-left text-gray-500 dark:text-gray-400">
      <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400">
        <tr>
          <th scope="col" className="p-4">
            <div className="flex items-center">
              <input
                id="checkbox-all-search"
                type="checkbox"
                className="w-4 h-4 text-blue-600 bg-gray-100 border-gray-300 rounded focus:ring-blue-500 dark:focus:ring-blue-600 dark:ring-offset-gray-800 dark:focus:ring-offset-gray-80
              />
              <label htmlFor="checkbox-all-search" className="sr-only">
                checkbox
              </label>
            </div>
          </th>
          <th scope="col" className="px-6 py-3">
            Numero de orden
          </th>
          <th scope="col" className="px-6 py-3">
            Creado en
          </th>
          <th scope="col" className="px-6 py-3">
            Entregado en
          </th>
          <th scope="col" className="px-6 py-3">
            Usuario
          </th>
          <th scope="col" className="px-6 py-3">
            Precio Total
          </th>
          <th scope="col" className="px-6 py-3">
            Productos
          </th>
          <th scope="col" className="px-6 py-3">
            Direccion de envio
          </th>
        </tr>
      </thead>
```

/frontend/src/components/PrivateRoute.tsx:

```tsx
import { Outlet, Navigate } from "react-router-dom";
import { useAuthStore } from "../store/auth";
import { Token } from "../Interfaces";
import jwt_decode from "jwt-decode";

export const PrivateRoute = () => {
  const { isAuth } = useAuthStore();
  return isAuth ? <Outlet /> : <Navigate to="/login" />;
};

export const AdminPrivateRoute = () => {
  const token: string = useAuthStore.getState().access;
  const tokenDecoded: Token = jwt_decode(token);
  const isAdmin = tokenDecoded.is_staff;
  return isAdmin ? <Outlet /> : <Navigate to="/" />;
};
```

Private Route allows access only to authenticated users and redirects to the login page if they are not authenticated.

Admin Private Route allows access only to users with administrator privileges and redirects to the home page if the user does not have these privileges.

Both components make use of the react-router-dom library and the useAuthStore authentication store to manage the authentication state and user privileges. These are examples of private routes that control access to certain parts of the application based on user authentication and role.

/frontend/src/components/ProductCard.tsx:

This "ProductCard" component is used to visually represent the information of a product, including its image, name, price, rating, description, and buttons for actions such as adding to the cart or viewing more details.

```tsx
import { Link } from "react-router-dom";
import { Product } from "../Interfaces";
import { useCartStore } from "../store/cart";
import Rating from "./Rating";

interface Props {
  product: Product;
}

const ProductCard = ({ product }: Props) => {
  const addToCart = useCartStore((state) => state.addToCart);

  return (
    <div>
      <div classNam="max-w-sm bg-white border border-gray-200 shadow dark:bg-gray-950 dark:border-gray-800">
        <Link to={`/product/${product.slug}`}>
          <img
            src={`${import.meta.env.VITE_BACKEND_URL}${product.image}`}
            alt=""
          />
        </Link>
        <div classNam="p-5 ">
          <Link to={`/product/${product.name}`}>
            <h5 classNam="mb-2 text-2xl font-bold tracking-tight text-gray-900 dark:text-white">
              {product.name}
            </h5>
            <div cl (parameter) product: Product old tracking-tight text-gray-900 dark:text-slate-200">
              $ {product.price}
            </h5>
            <div classNam="flex items-center">
              <span classNam="ml-1 text-gray-500 dark:text-gray-400">
                {product.rating === null && <Rating value={product.rating} />}
              </span>
            </div>
          </div>
        </Link>
        <p classNam="mb-3 font-normal text-gray-700 dark:text-gray-400">
          {product.description}
        </p>
```

/frontend/src/components/Products.tsx:

It is responsible for displaying a paginated list of products, allowing the deletion of products, and navigating to the product editing page. It also includes a function to load more products when the user scrolls down.

```tsx
return (
  <div classNam="overflow-x-auto">
    <table classNam="w-full text-sm text-left text-gray-500 dark:text-gray-400">
      <thead classNam="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400">
        <tr>
          <th scope="col" classNam="px-4 py-3">
            ID de prodcuto
          </th>
          <th scope="col" classNam="px-4 py-3">
            Nombre
          </th>

          <th scope="col" classNam="px-4 py-3">
            Precio
          </th>

          <th scope="col" classNam="px-4 py-3">
            Cantidad en stock
          </th>

          <th scope="col" classNam="px-4 py-3 flex justify-center gap-4">
            Acciones
            <Link to="add">
              <AiFillPlusSquare
                size={25}
                classNam="text-green-500 cursor-pointer rounded-full"
              />
            </Link>
          </th>
        </tr>
      </thead>
```

/frontend/src/components/Raiting.tsx:

It allows users to rate products using stars, ranging from 1 to 5 for the rating. It also aggregates the votes and displays the total value.

```tsx
import { faStar, faStarHalfAlt } from "@fortawesome/free-solid-svg-icons";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faStar as farStar } from "@fortawesome/free-regular-svg-icons";

interface Props {
  value: number;
}

function Rating({ value }: Props) {
  return (
    <div className="rating text-[#ffc107]">
      {value >= 1 ? (
        <FontAwesomeIcon icon={faStar} />
      ) : value >= 0.5 ? (
        <FontAwesomeIcon icon={faStarHalfAlt} />
      ) : (
        <FontAwesomeIcon icon={farStar} />
      )}

      {value >= 2 ? (
        <FontAwesomeIcon icon={faStar} />
      ) : value >= 1.5 ? (
        <FontAwesomeIcon icon={faStarHalfAlt} />
      ) : (
        <FontAwesomeIcon icon={farStar} />
      )}

      {value >= 3 ? (
        <FontAwesomeIcon icon={faStar} />
      ) : value >= 2.5 ? (
        <FontAwesomeIcon icon={faStarHalfAlt} />
      ) : (
        <FontAwesomeIcon icon={farStar} />
      )}

      {value >= 4 ? (
        <FontAwesomeIcon icon={faStar} />
      ) : value >= 3.5 ? (
        <FontAwesomeIcon icon={faStarHalfAlt} />
      ) : (
        <FontAwesomeIcon icon={farStar} />
      )}

      {value >= 5 ? (
        <FontAwesomeIcon icon={faStar} />
      ) : value >= 4.5 ? (
        <FontAwesomeIcon icon={faStarHalfAlt} />
      ) : (
        <FontAwesomeIcon icon={farStar} />
      )}
    </div>
  );
}

export default Rating;
```

/frontend/src/components/Reviews.tsx:

It manages and displays product reviews, allowing users to create new reviews through an interactive form.

```tsx
                <button
                  type="submit"
                  className="text-white inline-flex items-center bg-primary-700 hover:bg-primary-800 focus:rin
                >
                  <svg
                    className="mr-1 -ml-1 w-6 h-6"
                    fill="currentColor"
                    viewBox="0 0 20 20"
                    xmlns="http://www.w3.org/2000/svg"
                  >
                    <path
                      fill-rule="evenodd"
                      d="M10 5a1 1 0 011 1v3h3a1 1 0 110 2h-3v3a1 1 0 11-2 0v-3H6a1 1 0 110-2h3V6a1 1 0 011-1z
                      clip-rule="evenodd"
                    ></path>
                  </svg>
                  Publicar reseña
                </button>
              </form>
            </div>
          </div>
        </div>
      </div>
    )}
    <section className="bg-white dark:bg-gray-900">
      <div className="py-8 px-4 mx-auto max-w-screen-xl text-center lg:py-16 lg:px-6">
        <div className="mx-auto max-w-screen-sm">
          <h2 className="mb-4 text-4xl tracking-tight font-extrabold text-gray-900 dark:text-white">
            Reviews Generales
          </h2>
          <p className="mb-8 font-light text-gray-500 lg:mb-16 sm:text-xl dark:text-gray-400">
            Reseñas sobre este producto
          </p>
          <button
            onClick={() => {
              setShow(true);
            }}
            className="inline-flex items-center mx-3 px-3 py-2 text-sm font-medium text-center text-white bg-b
          >
            Crear una reseña
          </button>
        </div>
      </div>
    </section>
```

/frontend/src/components/trabajos.tsx:

Simply displays a grid of static images on a page. Each image is contained within a styled box. You can adjust the amount of space between the grid and other elements according to your needs.

```tsx
import React from "react";
import imagenuno from "../assets/Gifs y Pixel Art/1.jpeg";
import imagendos from "../assets/Gifs y Pixel Art/2.jpeg";
import imagentres from "../assets/Gifs y Pixel Art/3.jpg";
import imagencuatro from "../assets/Gifs y Pixel Art/4.jpeg";
import imagencinco from "../assets/Gifs y Pixel Art/5.jpeg";
import imagenseis from "../assets/Gifs y Pixel Art/6.gif";
import imagensiete from "../assets/Gifs y Pixel Art/7.gif";
import imagenocho from "../assets/Gifs y Pixel Art/8.gif";
import imagenueve from "../assets/Gifs y Pixel Art/9.jpg";
import imagendiez from "../assets/Gifs y Pixel Art/10.png";
import imagonce from "../assets/Gifs y Pixel Art/11.gif";
import imagendoce from "../assets/Gifs y Pixel Art/12.png";
import imagentrece from "../assets/Gifs y Pixel Art/13.gif";
import imagencatorce from "../assets/Gifs y Pixel Art/14.gif";

const Gallery = () => {
    const images = [
        imagenuno,
        imagendos,
        imagentres,
        imagencuatro,
        imagencinco,
        imagenseis,
        imagensiete,
        imagenocho,
        imagenueve,
        imagendiez,
        imagonce,
        imagendoce,
        imagentrece,
        imagencatorce,
    ];

    return (<>
        <br /><br />
        <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 lg:grid-cols-4 gap-4">
            {images.map((imagen, index) => (
                <div key={index} className="overflow-hidden bg-gray-200 rounded-lg">
                    <img className="h-64 w-full object-cover" src={imagen} alt={`Imagen ${index + 1}`} />
                </div>
            ))}
        </div>
        <br /><br /><br /><br /><br /><br /><br /><br /><br />
    </>

    );

};

export default Gallery;
```

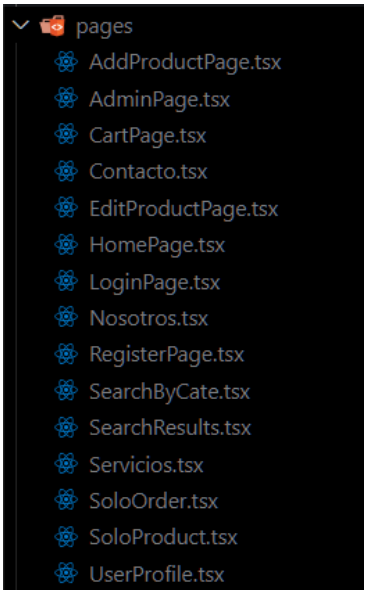
/frontend/src/components/Users.tsx:

Displays a user table with the ability to delete users. Utilizes React Query to manage queries and mutations related to users.

```tsx
const Users = ({ results }: Props) => {
  const queryClient = useQueryClient();

  const { data, isError, isLoading } = useQuery({
    queryKey: ["users"],
    queryFn: get_users,
  });

  const deleteUserMut = useMutation({
    mutationFn: delete_user,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["users"] });
      toast.success("Usuario eliminado");
    },
    onError: () => {
      toast.error("Error!");
    },
  });

  if (isError) return toast.error("Error!");
  if (isLoading) return <Loader />;

  return (
    <div className="overflow-x-auto">
      <table className="w-full text-sm text-left text-gray-500 dark:text-gray-400">
        <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400">
          <tr>
            <th scope="col" className="px-4 py-3">
              ID Usuario
            </th>
            <th scope="col" className="px-4 py-3">
              Correo Electronico
            </th>
            <th scope="col" className="px-4 py-3">
              Nombre(s)
            </th>
            <th scope="col" className="px-4 py-3">
              Apellidos
            </th>
            <th
              scope="col"
              className="px-4 py-3 flex items-center justify-center gap-4"
            >
              Acciones
            </th>
          </tr>
        </thead>
```

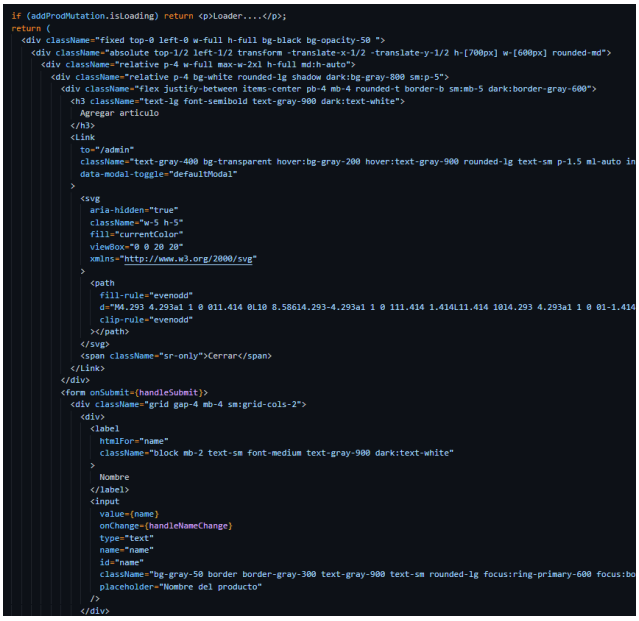
Pages folder:

The pages folder contains five teen files where the code for the pages is hosted. These files represent the pages to which a user is redirected when navigating through the menus.



/frontend/src/pages/AddProductPage.tsx:

Is a form for adding new products. It utilizes various state variables to manage input fields for product details such as name, count in stock, category, description, price, and image. The component also handles file input for image uploading, including drag-and-drop functionality.



```
if (addProdMutation.isLoading) return <p>Loader....</p>;
return (
  <div className="fixed top-0 left-0 w-full h-full bg-black bg-opacity-50 ">
    <div className="absolute top-1/2 left-1/2 transform -translate-x-1/2 -translate-y-1/2 h-[700px] w-[600px] rounded-md">
      <div className="relative p-4 w-full max-w-2xl h-full md:h-auto">
        <div className="relative p-4 bg-white rounded-lg shadow dark:bg-gray-800 sm:p-5">
          <div className="flex justify-between items-center pb-4 mb-4 rounded-t border-b sm:mb-5 dark:border-gray-600">
            <h3 className="text-lg font-semibold text-gray-900 dark:text-white">
              Agregar articulo
            </h3>
            <Link
              to="/admin"
              className="text-gray-400 bg-transparent hover:bg-gray-200 hover:text-gray-900 rounded-lg text-sm p-1.5 ml-auto in
              data-modal-toggle="defaultModal"
            >
              <svg
                aria-hidden="true"
                className="w-5 h-5"
                fill="currentColor"
                viewBox="0 0 20 20"
                xmlns="http://www.w3.org/2000/svg"
              >
                <path
                  fill-rule="evenodd"
                  d="M4.293 4.293a1 1 0 011.414 0L10 8.58614.293-4.293a1 1 0 111.414 1.414L11.414 1014.293 4.293a1 1 0 01-1.414
                  clip-rule="evenodd"
                ></path>
              </svg>
              <span className="sr-only">Cerrar</span>
            </Link>
          </div>
          <form onSubmit={handleSubmit}>
            <div className="grid gap-4 mb-4 sm:grid-cols-2">
              <div>
                <label
                  htmlFor="name"
                  className="block mb-2 text-sm font-medium text-gray-900 dark:text-white"
                >
                  Nombre
                </label>
                <input
                  value={name}
                  onChange={handleNameChange}
                  type="text"
                  name="name"
                  id="name"
                  className="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg focus:ring-primary-600 focus:bo
                  placeholder="Nombre del producto"
                />
              </div>
```

Is a form for adding new products. It utilizes various state variables to manage input fields for product details such as name, count in stock, category, description, price, and image. The component also handles file input for image uploading, including drag-and-drop functionality.

The component uses the useMutation hook from the react-query library to handle the asynchronous product creation process. Upon successful submission, it triggers a query invalidation to update the product list, displays a success toast, and navigates to the admin page. In case of an error, it displays an error toast and navigates to the admin page.

The form includes input fields for product details, a file input for image uploading, and conditional rendering for either displaying a file upload prompt or previewing the selected image. Stylistically, it utilizes Tailwind CSS classes for responsive and modern UI design.

The component provides a user-friendly interface for adding new products with real-time feedback through toasts and visual cues for image upload.

/frontend/src/pages/AdminPage.tsx:

The AdminPage React.js component is a dashboard for managing products, orders, and users in an administration interface. It includes a search bar to filter results based on user input. The component uses the useQuery hook from the react-query library to fetch and display data based on the selected category (products, orders, or users).

Key features:

State Management: Utilizes the useState hook to manage the state for the selected category (show) and the search input (search).

Search Functionality: Enables users to search for specific items (products, orders, or users) using the search bar. It dynamically fetches and updates the displayed data based on the search input.

Category Selection: Provides buttons to switch between different categories (products, orders, and users). Clicking on a button triggers a re-fetch of data based on the selected category.

Conditional Rendering: Renders different components (Products, Orders, or Users) based on the selected category (show). This allows for a dynamic display of content.

Styling: Uses Tailwind CSS classes for responsive and visually appealing design, including dark mode support.

User Interaction: Buttons for each category enable users to easily switch between product, order, and user views.

Components Integration: Embeds Products, Orders, and Users components to display the respective data based on the selected category.

Overall, the AdminPage serves as a central hub for managing and viewing products, orders, and users, providing a user-friendly and visually consistent interface.

```
return (
  <section className="bg-gray-50 dark:bg-gray-900 p-3 sm:p-5">
    <div className="mx-auto max-w-screen-xl px-4 lg:px-12">
      <div className="bg-white dark:bg-gray-800  dark: sm:rounded-lg overflow-hidden">
        <div className="flex flex-col md:flex-row items-center justify-between space-y-3 md:space-y-0 md:space-x-4 p-4">
          <div className="w-full md:w-1/2">
            <form className="flex items-center">
              <label htmlFor="simple-search" className="sr-only">
                Buscar
              </label>
              <div className="relative w-full">
                <div className="absolute inset-y-0 left-0 flex items-center pl-3 pointer-events-none">
                  <svg
                    aria-hidden="true"
                    className="w-5 h-5 text-gray-500 dark:text-gray-400"
                    fill="currentColor"
                    xmlns="http://www.w3.org/2000/svg"
                  >
                    <path
                      fill-rule="evenodd"
                      d="M8 4a4 4 0 100 8 4 4 0 000-8zM2 8a6 6 0 1110.89 3.47614.817 4.817a1 1 0 01-1.414 1.4141-4.816-4.816A6 6 0 012 8z"
                      clip-rule="evenodd"
                    />
                  </svg>
                </div>
                <input
                  value={search}
                  onChange={(e) => setSearch(e.target.value)}
                  type="text"
                  className="bg-gray-50 border border-gray-300 text-gray-900 text-sm rounded-lg focus:ring-primary-500 focus:border-primary-500 block w-full pl-10 p-2 dark:bg-gray-700 dark:border-
                  placeholder="Buscar"
                />
              </div>
            </form>
          </div>
          <div className="w-full md:w-auto flex flex-col md:flex-row space-y-2 md:space-y-0 items-stretch md:items-center justify-end md:space-x-3 flex-shrink-0">
            <button
              onClick={() => setShow(0)}
              type="button"
              className="flex items-center justify-center text-gray bg-gradient-to-r from-indigo-200 via-red-200 to-yellow-100 hover:bg-primary-800 focus:ring-4 focus:ring-primary-300 font-medium
            >
              Productos
            </button>
            <button
              onClick={() => setShow(1)}
              type="button"
              className="flex items-center justify-center text-gray bg-gradient-to-r from-indigo-200 via-red-200 to-yellow-100 hover:bg-primary-800 focus:ring-4 focus:ring-primary-300 font-medium
            >
              Pedidos
            </button>
            <button
              onClick={() => setShow(2)}
              type="button"
              className="flex items-center justify-center text-gray bg-gradient-to-r from-indigo-200 via-red-200 to-yellow-100 hover:bg-primary-800 focus:ring-4 focus:ring-primary-300 font-medium
```

/frontend/src/pages/CartPage.tsx:

This component is a page that displays the items in the shopping cart, allows users to modify the quantity of items, provides a form for entering shipping details, and integrates with PayPal for payment. Here's an overview of the key features:

State Management:

Uses the useCartStore custom hook to manage cart-related state, such as adding, removing, and clearing items from the cart.
Local states (address, city, postal_code) are used to store user input for shipping details.

Mutation for Creating Orders:
Utilizes the useMutation hook from react-query to handle asynchronous creation of orders.
Defines createOrderMut mutation with onSuccess and onError handlers to handle the success and failure scenarios.

Cart Display:
Displays a table with information about each item in the cart, including product name, category, quantity, price, and total price.
Shipping Details Form:
Provides a form for users to enter their shipping details, including address, city, and postal code.

PayPal Integration:

Utilizes the PayPalScriptProvider and PayPalButtons components from the @paypal/react-paypal-js library for PayPal integration.
The createOrder and onApprove functions are defined to handle the order creation process and order approval process, respectively.
On successful order creation, it invalidates the query for orders, displays a success message, clears the cart, and redirects the user to the home page.

Styling and Layout:

Uses Tailwind CSS classes for styling, including dark mode support.

User Interaction:
Users can modify the quantity of items in the cart using increment and decrement buttons.
PayPal buttons are integrated to provide a seamless payment experience.

Form Submission:
Submits the shipping details and order information to the server using the handleSubmit function.

Note:
Ensure that the PayPal client ID (clientId) used in the PayPalScriptProvider is valid and corresponds to your PayPal account.
This component provides a complete flow for managing the shopping cart, entering shipping details, and processing payments through PayPal.

/frontend/src/pages/Contacto.tsx:

The page calls the previously created component (EmailContext.tsx) to send emails.

```
import { ContactUs } from "../components/EmailContext";
const Contacto = () => {
    return (
        <>
            <body>
                <br />
                <ContactUs />
                <br /> <br /> <br />
                <br />
                <br /><br />
            </body >

        </>

    );
};
export default Contacto;
```

/frontend/src/pages/EditProductsPage.tsx:

It contains code for editing products, such as the name, price, or image, in addition to having an attractive design for editing these attributes.

```
const EditProductPage = () => {
  const [name, setName] = useState<string>("");
  const [countInStock, setCountInStock] = useState<number>(0);
  const [category, setCategory] = useState<string>("");
  const [description, setDescription] = useState<string>("");
  const [price, setPrice] = useState<number>(0);
  const [image, setImage] = useState<File | null>(null);
  const [filePreview, setFilePreview] = useState<string>("");
  const inputRef = React.useRef<HTMLInputElement>(null);
  const [isHovered, setIsHovered] = useState(false);

  const { id } = useParams();
  let prodId: number;

  if (id !== undefined) {
    prodId = Number(id);
  }

  const { data } = useQuery({
    queryKey: ["products", id],
    queryFn: () => get_solo_prod(prodId),
  });

  useEffect(() => {
    if (data) {
      setName(data.name);
      setCountInStock(data.count_in_stock);
      setDescription(data.description);
      setCategory(data.category);
      setPrice(data.price);
    }
  }, [data]);

  const navigate = useNavigate();
  const queryClient = useQueryClient();

  const editProdMutation = useMutation({
    mutationFn: edit_product,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["products"] });
      toast.success("Product edited!");
      navigate("/admin");
    },
    onError: () => {
      toast.error("Error!");
      navigate("/admin");
    },
  });

  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    editProdMutation.mutate({
      name: name,
      count_in_stock: countInStock,
```

/frontend/src/pages/HomePage.tsx:

```
import { get_products } from "../api/products";
import ProductCard from "../components/ProductCard";
import { Product } from "../Interfaces";
import { useInfiniteQuery } from "@tanstack/react-query";
import { useInView } from "react-intersection-observer";
import React, { useEffect } from "react";
import Loader from "../components/Loader";
import toast from "react-hot-toast";
import { useSearchStore } from "../store/search";
import SearchResults from "./SearchResults";
import Carousel from "../components/Carousel";

const HomePage = () => {
  const { ref, inView } = useInView();
  const searchTerm = useSearchStore((state) => state.searchTerm);

  useEffect(() => {
    if (inView) {
      fetchNextPage();
    }
  }, [inView]);

  const {
    data,
    isLoading,
    error,
    isFetchingNextPage,
    fetchNextPage,
    hasNextPage,
  } = useInfiniteQuery(["products"], get_products, {
    getNextPageParam: (page: any) => page.meta.next,
  });

  if (searchTerm) return <SearchResults />;
  if (error instanceof Error) return <>{toast.error(error.message)}</>;

  return (
    <>
      <Carousel />
      {data?.pages.map((page: any) => (
        <>
          <div className="flex justify-center">
            <div
              key={page.meta.next}
              className="p-8 grid grid-cols-1 md:grid-cols-3 lg:grid-cols-3 gap-16"
            >
              {page.data.map((product: Product) => (
                <ProductCard key={product.id} product={product} />
              ))}
            </div>
          </div>
```

The HomePage component is a React component responsible for rendering a dynamic and paginated display of products.

Imports:

The component imports various modules, components, and utility functions.

Hooks and State:

Uses the useInView hook to detect when a specified element comes into view.
Utilizes the useInfiniteQuery hook from react-query for fetching products in an infinite scroll manner.

Data Fetching:

Fetches products with pagination support using the get_products function.
Uses the useInfiniteQuery hook to handle paginated data.

Infinite Scroll Logic:

Utilizes the fetchNextPage function to load the next page of products when the target element (ref) comes into view.

Conditional Rendering:

- Checks for ongoing search (searchTerm) and renders the SearchResults component if true.
- Displays an error message if there is an error during data fetching.
- Renders a carousel followed by a grid of product cards.

Rendering:

- Renders a Carousel component.
- Renders a grid of product cards with three columns for small to large screens.
- Displays a loader during data fetching or when the next page is being loaded.
- Shows a message when there are no results.

/frontend/src/pages/LoginPage.tsx:

This page contains all the components responsible for the login options, implementing an aesthetically pleasing design and making calls to these components.

```
import React, { useState } from "react";
import { toast } from "react-hot-toast";
import { useAuthStore } from "../store/auth";
import Logo from "../assets/logo.png";

// Definir el componente funcional LoginPage
const LoginPage = () => {
  // Utilizar el hook useNavigate para la navegación
  const navigate = useNavigate();

  // Obtener el estado de autenticación y la función para establecer el token desde el store de autenticación
  const { isAuth } = useAuthStore();
  const setToken = useAuthStore((state) => state.setToken);

  // Estado local para almacenar el correo electrónico y la contraseña del usuario
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  // Utilizar el hook useMutation para gestionar la lógica de la mutación de inicio de sesión
  const loginMutation = useMutation({
    mutationFn: () => loginRequest(email, password),
    onSuccess: (response) => {
      // En caso de éxito, establecer el token y redirigir a la página principal
      setToken(response.data.access, response.data.refresh);
      toast.success("Inicio de sesión con éxito!");
      navigate("/");
    },
    onError: () => {
      // En caso de error, mostrar un mensaje de error
      toast.error("Error al iniciar sesión");
    },
  });

  // Manejar el envío del formulario
  const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    // Llamar a la mutación de inicio de sesión
    loginMutation.mutate();
  };

  // Si la mutación está en curso, mostrar un mensaje de carga
  if (loginMutation.isLoading) return <p>Loading...</p>;

  // Si el usuario ya está autenticado, redirigir a la página principal
  if (isAuth) return <Navigate to="/" />;

  // Renderizar la interfaz de inicio de sesión
  return (
    <div className="flex flex-col items-center justify-center px-6 py-8 mx-auto md:h-[800px] lg:py-0">

      <div className="w-full md:w-[400px] lg:w-[500px] bg-slate-300 shadow dark:border md:mt-0 sm:max-w-md xl:p-0 dark:bg-gray-800 dark:border-gray-700">
        <div className="p-6 space-y-4 md:space-y-6 sm:p-8">
          <h1 className="text-xl text-center font-bold leading-tight tracking-tight text-gray-900 md:text-2xl dark:text-white">
            Inicio de sesión
```

/frontend/src/pages/Nosotros.tsx:

Calls the components fractal.tsx, Mision.tsx, and trabajos.tsx to display them on this page.

```
import React from "react";
import Mision from "../components/Mision";
import cris from "../assets/cris.jpg";
import omar from "../assets/omar.jpeg";
import jose from "../assets/jose.jpg";
import nataen from "../assets/nata.jpg";
import FacebookPage from "../components/facebook";
import Trabajos from "../components/trabajos";
import FractalTreeComponent from "../components/fractal";
const Nosotros = () => {


  return (
    <>
      <br />
      <br />
      <br />

      <div className="container mx-auto">
        <div className="grid mb-8 border border-gray-200 rounded-lg shadow-sm dark:border-gray-700 md:mb-12 md:grid-cols-2 bg-white dark:bg-gray-800">

          {/* Perfil 1 */}
          <figure className="flex flex-col items-center justify-center p-8 text-center bg-white border-b border-gray-200 rounded-t-lg md:rounded-t-none md:rounded-ss-lg md:border-e dark:bg-gray-800 dark:border-gray-700">
            <blockquote className="max-w-2xl mx-auto mb-4 text-gray-500 lg:mb-8 dark:text-gray-400">
              <h3 className="text-lg font-semibold text-gray-900 dark:text-white">DOCUMENTADOR</h3>
              <p classname="my-4">Responsable de la parte de documentación.
                Garantizó una documentación completa y precisa para el proyecto, proporcionando una guía detallada que facilitó la comprensión y colaboración en el equipo.</p>
            </blockquote>
            <figcaption className="flex items-center justify-center ">
              <img className="rounded-full w-20 h-20" src={cris} alt="profile picture" />
              <div className="space-y-0.5 font-medium dark:text-white text-left rtl:text-right ms-3">
                <div>Pulido Larios Cristopher</div>
                <div className="text-sm text-gray-500 dark:text-gray-400">Documentación/Diseño</div>
              </div>
            </figcaption>
          </figure>
```

/frontend/src/pages/Register.tsx:

It contains the components to register a user, along with the respective functions to create the user by communicating with the database.

```
  >
    Nombre
  </label>
  <input
    value={name}
    required
    onChange={(e) => setName(e.target.value)}
    type="name"
    name="name"
    id="name"
    className="bg-gray-50 border border-gray-300 text-gray-900 sm:text-sm focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5 dark:bg-gray-700 dark:border-gray-600
    placeholder="Nombre(s)"
  />
</div>

<div>
  <label
    htmlFor="last_name"
    className="block mb-2 text-sm font-medium text-gray-900 dark:text-white"
  >
    Apellidos
  </label>
  <input
    value={last_name}
    required
    onChange={(e) => setLastName(e.target.value)}
    type="last_name"
    name="last_name"
    id="last_name"
    className="bg-gray-50 border border-gray-300 text-gray-900 sm:text-sm  focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5 dark:bg-gray-700 dark:border-gray-600
    placeholder="Apellidos"
  />
</div>

<div>
  <label
    htmlFor="password"
    className="block mb-2 text-sm font-medium text-gray-900 dark:text-white"
  >
    Contraseña
  </label>
  <input
    value={password}
    required
    onChange={(e) => setPassword(e.target.value)}
    type="password"
    name="password"
    id="password"
    placeholder="*********"
    className="bg-gray-50 border border-gray-300 text-gray-900 sm:text-sm  focus:ring-primary-600 focus:border-primary-600 block w-full p-2.5 dark:bg-gray-700 dark:border-gray-600
  />
```

/frontend/src/pages/SearchByCate.tsx:

The code in this file contains components created to display the search bar. Additionally, it implements logic to call products from the database and display them.

```
return (
  <div className="flex justify-center">
    <div className="p-8 grid grid-cols-1 md:grid-cols-3 lg:grid-cols-3 gap-16">
      {data &&
        data.map((product: Product) => (
          <div className="max-w-sm bg-white border border-gray-200 rounded-lg shadow dark:bg-gray-800 dark:border-gray-700">
            <Link to={`/product/${product.slug}`}>
              <img
                className="rounded-t-lg"
                src={`${import.meta.env.VITE_BACKEND_URL}${product.image}`}
                alt=""
              />
            </Link>
            <div className="p-5 ">
              <Link to={`/product/${product.name}`}>
                <h5 className="mb-2 text-2xl font-bold tracking-tight text-gray-900 dark:text-white">
                  {product.name}
                </h5>
                <div className="flex justify-between">
                  <h5 className="mb-2 text-2xl font-bold tracking-tight text-gray-900 dark:text-slate-200">
                    $ {product.price}
                  </h5>
                  <div className="flex items-center">
                    <span className="ml-1 text-gray-500 dark:text-gray-400">
                      {product.rating === null && (
                        <Rating value={product.rating} />
                      )}
                    </span>
                  </div>
                </div>
              </Link>
              <p className="mb-3 font-normal text-gray-700 dark:text-gray-400">
                {product.description}
              </p>

              <button
                onClick={() => addToCart(product)}
                className="inline-flex items-center mx-3 px-3 py-2 text-sm font-medium text-center text-white bg-blue-700 rounded-lg hover:bg-blue-800 focus:ring-4
              >
                Añadir al carrito
                <svg
                  aria-hidden="true"
                  className="w-4 h-4 ml-2 -mr-1"
                  fill="currentColor"
                  viewBox="0 0 20 20"
                  xmlns="http://www.w3.org/2000/svg"
                >
                  <path
                    fill-rule="evenodd"
                    d="M10.293 3.293a1 1 0 011.414 016 6a1 1 0 010 1.414l-6 6a1 1 0 01-1.414-1.414L14.586 11H3a1 1 0 110-2h11.586l-4.293-4.293a1 1 0 010-1.414z"
                    clip-rule="evenodd"
                  ></path>
                </svg>
              </button>
```

/frontend/src/pages/SearchResult.tsx:
It searches for products when you enter something general without specifying the category.

```tsx
import { search_prod } from "../api/products";
import { useQuery } from "@tanstack/react-query";
import { useSearchStore } from "../store/search";
import ProductCard from "../components/ProductCard";
import { Product } from "../Interfaces";

const SearchResults = () => {
  const searchTerm = useSearchStore((state) => state.searchTerm);

  const { data } = useQuery({
    queryKey: ["products", searchTerm],
    queryFn: () => {
      if (searchTerm) {
        return search_prod(searchTerm);
      }
      return { products: [] };
    },
  });

  return (
    <div className="flex justify-center">
      <div className="p-8 grid grid-cols-1 md:grid-cols-3 lg:grid-cols-3 gap-16">
        {data &&
          data.products.map((product: Product) => (
            <ProductCard product={product} />
          ))}
      </div>
    </div>
  );
};
export default SearchResults;
```

/frontend/src/pages/Servicios.tsx:
is a static page that contains our services of mxdiscovery, has text and images like gifs, other option here is the facebook widget , is implemented here, call the component and show on the page.

```tsx
import FacebookPage from "../components/facebook";
const Servicios = () => {
  return (
    <body>
      <br />
      <br />
      <br /> <br /> <br />
      <br />
      <div className="container mx-auto">
        <div className="grid lg:grid-cols-4 md:grid-cols-2 sm:grid-cols-1 gap-4">
          <div>
            <div className="ml-5 max-w-sm rounded overflow-hidden max-w-sm bg-white border border-gray-200 rounded-lg shadow dark:bg-gray-800 dark:border-gray-700">
              <img
                className="w-full"
                src="https://www.asafes.org/wp-content/uploads/2022/03/manualidades-1.jpg"
                alt="Sunset in the mountains"
              ></img>
              <div className="px-6 py-4 ">
                <div className="mb-2 text-2x1 font-bold tracking-tight text-gray-900 dark:text-white">
                  Taller de manualidades.
                </div>
                <p className="text-gray-700 text-base tracking-tight text-gray-900 dark:text-white">
                  El taller de manualidades es un lugar donde la imaginación
                  cobra vida a través de las manos. Exploraremos diversas
                  técnicas artísticas, desde la pintura hasta la creación de
                  objetos tridimensionales. Aprenderemos a transformar
                  materiales simples en obras de arte únicas. Este taller es
                  perfecto para quienes buscan expresar su creatividad,
                  relajarse y disfrutar del placer de crear con las manos.
                  <br />
                  <br />
                  <br />
                  <br /> <br />
                  <br /> <br />
                </p>
              </div>
              <div className="px-6 pt-4 pb-2 ">
                <span className="inline-block bg-gray-200 rounded-full px-3 py-1 text-sm font-semibold text-gray-700 mr-2 mb-2">
                  #Manualidades
                </span>
                <span className="inline-block bg-gray-200 rounded-full px-3 py-1 text-sm font-semibold text-gray-700 mr-2 mb-2">
                  #Arte
                </span>
                <span className="inline-block bg-gray-200 rounded-full px-3 py-1 text-sm font-semibold text-gray-700 mr-2 mb-2">
                  #Cultura
                </span>
              </div>
            </div>
          </div>
        </div>
```

/frontend/src/pages/SoloOrder.tsx:
displaying detailed information about a single order.

```
return (
  <div className="overflow-x-auto container mx-auto px-4 pt-11">
    <table className="w-full text-sm text-left text-gray-500 dark:text-gray-400">
      <thead className="text-xs text-gray-700 uppercase bg-gray-50 dark:bg-gray-700 dark:text-gray-400">
        <tr>
          <th scope="col" className="px-4 py-3">
            Precio total
          </th>
          <th scope="col" className="px-4 py-3">
            Entregado
          </th>

          <th scope="col" className="px-4 py-3">
            Creado
          </th>

          <th scope="col" className="px-4 py-3">
            Entregado a
          </th>

          <th scope="col" className="px-4 py-3">
            Ciudad
          </th>

          <th scope="col" className="px-4 py-3">
            Direccion
          </th>

          <th scope="col" className="px-4 py-3">
            Codigo Postal
          </th>
        </tr>
      </thead>
```

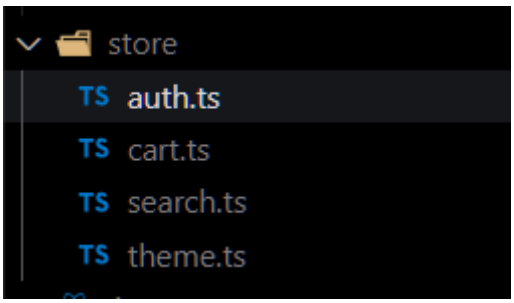
/frontend/src/pages/SoloProduct.tsx:
It is used to display detailed information of a specific product in an application, displays the details of a specific product, allowing users to view the product information and reviews associated with it.

```
return (
  <div className="bg-white dark:bg-gray-900">
    <div className="gap-16 items-center py-8 px-4 mx-auto max-w-screen-xl lg:grid lg:grid-cols-2 lg:py-16 lg:px-6">
      <div className="font-light text-gray-500 sm:text-lg dark:text-gray-400">
        <h2 className="mb-4 text-4xl tracking-tight font-extrabold text-gray-900 dark:text-white">
          {data.name}
          <span className="text-green-300 ml-4">${data.price}</span>
        </h2>
        <p className="mb-4 font-bold">{data.description}</p>
        <a
          href="#"
          className="inline-flex items-center px-3 py-2 text-sm font-medium text-center text-white bg-blue-700 rounded-lg hover:bg-blue-800 focus:ring-4 focus:outline-none
        >
          Añadir al carrito
          <svg
            aria-hidden="true"
            className="w-4 h-4 ml-2 -mr-1"
            fill="currentColor"
            viewBox="0 0 20 20"
            xmlns="http://www.w3.org/2000/svg"
          >
            <path
              fill-rule="evenodd"
              d="M10.293 3.293a1 1 0 011.414 016 6a1 1 0 010 1.4141-6 6a1 1 0 01-1.414-1.414L14.586 11H3a1 1 0 110-2h11.5861-4.293-4.293a1 1 0 010-1.414z"
              clip-rule="evenodd"
            ></path>
          </svg>
        </a>
      </div>

      <img
        className="w-full"
        src={`${import.meta.env.VITE_BACKEND_URL}${data.image}`}
        alt="office content 1"
      />
    </div>
    <Reviews productId={data.id} reviews={data.reviews} />
  </div>
);
};
export default SoloProduct;
```

UserProfile.tsx:
This is a React component that displays user profile information and provides the ability to edit some profile fields, such as first name, last name, and profile image, all integrated with queries to an API to obtain and update data.

```tsx
import { useAuthStore } from "../store/auth";
import { Token } from "../Interfaces";
import jwt_decode from "jwt-decode";
import React, { useState, ChangeEvent, useEffect } from "react";
import { useMutation, useQueryClient, useQuery } from "@tanstack/react-query";
import { edit_user, get_solo_user } from "../api/users";
import { toast } from "react-hot-toast";
import { my_orders } from "../api/orders";
import Loader from "../components/Loader";
import { Link } from "react-router-dom";

const UserProfile = () => {
  const [show, setShow] = useState(true);
  const [stateName, setStateName] = useState<string>("");
  const [stateLast, setStateLast] = useState<string>("");
  const [image, setImage] = useState<File | null>(null);
  const [filePreview, setFilePreview] = useState<string>("");
  const inputRef = React.useRef<HTMLInputElement>(null);
  const [isHovered, setIsHovered] = useState(false);

  const token: string = useAuthStore.getState().access;
  const tokenDecoded: Token = jwt_decode(token);
  const id = tokenDecoded.user_id;

  const { data: user } = useQuery({
    queryKey: ["users", id],
    queryFn: () => get_solo_user(id),
  });

  useEffect(() => {
    if (user) {
      setStateName(user.name);
      setStateLast(user.last_name);
      setImage(user.avatar);
    }
  }, [user]);

  const queryClient = useQueryClient();

  const editProfileMut = useMutation({
    mutationFn: edit_user,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["users"] });
      toast.success("Profile updated!");
      setShow(true);
    },
    onError: () => {
      toast.error("Error!");
      setShow(true);
    },
  });
```

Store Folder:
This folder has four files to authentication, cart , search and theme functions.



/frontend/src/store/auth.ts

This code provides a state hook to handle user authentication in a React app using zustand and configures it so that the state persists in the browser.

```ts
import { create } from "zustand";
import { persist } from "zustand/middleware";

type State = {
  access: string;
  refresh: string;
  isAuth: boolean;
};

type Actions = {
  setToken: (access: string, refresh: string) => void;
  logout: () => void;
};

export const useAuthStore = create(
  persist<State & Actions>(
    (set) => ({
      access: "",
      refresh: "",
      isAuth: false,
      setToken: (access: string, refresh: string) =>
        set(() => ({
          access,
          refresh,
          isAuth: !!access && !!refresh,
        })),
      logout: () => set(() => ({ access: "", refresh: "", isAuth: false })),
    }),
    {
      name: "auth",
    }
  )
);
```

/frontend/src/store/cart.ts

This code is another example of a custom hook using the zustand library to manage the shopping cart state in a React application. Like the previous code, this useCartStore hook uses the persist middleware to store and persist the cart state in the browser's local storage.

This code provides a state hook (useCartStore) that manages the state of the shopping cart in a React application, with the ability to add, remove products, and completely empty the cart, keep ing the data saved in the browser's local s torage for persistence.

```
export const useCartStore = create(
  persist<State & Actions>(
    (set, get) => ({
      cart: State.cart,
      totalPrice: State.totalPrice,

      removeAll: () => {
        set({
          cart: [],
          totalPrice: 0,
        });
      },
      addToCart: (product: Product) => {
        const cart = get().cart;
        const cartItem = cart.find((item) => item.id === product.id);

        if (cartItem) {
          const updatedCart = cart.map((item) =>
            item.id === product.id
              ? { ...item, quantity: (item.quantity as number) + 1 }
              : item
          );
          set((state) => ({
            cart: updatedCart,
            totalPrice: state.totalPrice + Number(product.price),
          }));
        } else {
          const updatedCart = [...cart, { ...product, quantity: 1 }];

          set((state) => ({
            cart: updatedCart,
            totalPrice: state.totalPrice + Number(product.price),
          }));
        }
      },

      removeFromCart: (product: Product) => {
        const cart = get().cart;
        const cartItem = cart.find((item) => item.id === product.id);

        if (cartItem && cartItem.quantity && cartItem.quantity > 1) {
          const updatedCart = cart.map((item) =>
            item.id === product.id
              ? { ...item, quantity: (item.quantity as number) - 1 }
              : item
          );
          set((state) => ({
            cart: updatedCart,
            totalPrice: state.totalPrice - Number(product.price),
          }));
        } else {
          set((state) => ({
            cart: state.cart.filter((item) => item.id !== product.id),
            totalPrice: state.totalPrice - Number(product.price),
          }));
```

/frontend/src/store/cart.ts

```
import { create } from "zustand";

interface SearchStore {
  searchTerm: string;
  setSearchTerm: (term: string) => void;
}

export const useSearchStore = create<SearchStore>((set) => ({
  searchTerm: "",
  setSearchTerm: (term) => set({ searchTerm: term }),
}));
```

This code is a custom hook using the zustand library to manage the state of a search term.

Provides a way to manage and update a search term in the application using zustand as a state manager.

/frontend/src/store/theme.ts

This code shows a custom hook called use Dark Mode using the zustand library in React to handle the state of a dark mode in an application.

This useDarkMode hook provides functionality to handle the state of dark mode in a React application, with the ability to turn it on or off, and uses persist to have that setting persist in the browser's local storage.

```
import { create } from "zustand";
import { persist } from "zustand/middleware";

interface DarkModeStore {
  darkMode: boolean;
  toggleDarkMode: () => void;
}

export const useDarkMode = create<DarkModeStore>()(
  persist(
    (set) => ({
      darkMode: true,
      toggleDarkMode: () => set((state) => ({ darkMode: !state.darkMode })),
    }),
    {
      name: "theme",
    }
  )
);
```