Documentación de la Aplicación de Rotisería

Introducción

Esta aplicación es una solución para gestionar una rotisería, desde la visualización de productos hasta la administración de inventario. Con un backend desarrollado en Django que proporciona una API RESTf y un frontend implementado en Vue.js.

Objetivo

Permitir a los propietarios y empleados llevar un registro detallado de los productos disponibles, así como facilita la inclusión de nuevos productos en el menú.

Características

- Visualización de Categorías: La aplicación recupera categorías desde el backend y las muestra de forma organizada en el frontend, lo que facilita la navegación y búsqueda de productos.
- Gestión de Productos: Los usuarios pueden explorar una lista completa de productos disponibles en la rotisería, lo que incluye detalles como nombre, precio y descripción. Además, la aplicación permite a los administradores cargar nuevos productos de manera sencilla.
- Integración Backend-Frontend: El backend, desarrollado en Django, proporciona una API RESTful que permite una comunicación eficiente entre el frontend y el backend, asegurando que los datos se mantengan sincronizados y actualizados.

Backend (Django)

Modelos

- Modelo de Categorías (Category):
  - Este modelo representa las categorías de los productos en tu aplicación. Cada categoría tiene un nombre.
- Modelo de Productos (Product):
  - Este modelo representa los productos disponibles en tu aplicación de rotisería. Los productos tienen un nombre, una imagen, una categoría, una descripción, un precio y un tipo de precio.

Vistas (basadas en clases)

- **Vista de Productos (ProductViewSet)**
  - La vista de Productos (ProductViewSet) es una vista basada en clases que maneja la gestión de los productos en tu aplicación. Utiliza Django REST framework para proporcionar funcionalidad CRUD (Crear, Leer, Actualizar, Borrar) para los productos.
  - Queryset: El conjunto de datos de esta vista incluye todos los productos en la base de datos.

    - Serializer: Utiliza el serializador ProductSerializer para convertir los objetos del modelo Product en datos JSON y viceversa.
    - Método get_queryset: Se sobrescribe el método get_queryset para permitir filtrado personalizado de productos basado en parámetros de consulta, como categoría y búsqueda por nombre o descripción.
- **Vista de Categorías (CategoryViewSet)**
    - La vista de Categorías (CategoryViewSet) es otra vista basada en clases que gestiona las categorías de productos en tu aplicación.
    - Queryset: El conjunto de datos de esta vista incluye todas las categorías disponibles en la base de datos.
    - Serializer: Utiliza el serializador CategorySerializer para convertir los objetos del modelo Category en datos JSON y viceversa.

Serializadores (Serializers)

- **Serializador de Productos (ProductSerializer)**
    - El serializador de productos (ProductSerializer) convierte los objetos del modelo Product en datos JSON para su uso en la API REST. Además de los campos del modelo, agrega información adicional para facilitar la visualización y la comprensión en la interfaz de usuario.
    - Campos Personalizados:
        - category_name: Devuelve el nombre de la categoría a la que pertenece el producto, lo que facilita la visualización en la interfaz de usuario de Vue.js.
        - price_type_description: Proporciona la descripción del tipo de precio en lugar del valor crudo (por ejemplo, "Unidad" en lugar de "unitario").
- **Serializador de Categorías (CategorySerializer)**
    - El serializador de categorías (CategorySerializer) convierte los objetos del modelo Category en datos JSON para su uso en la API REST. Incluye todos los campos del modelo Category para su serialización.
- **Filtros de Productos (ProductFilter)**
    - El filtro de productos (ProductFilter) permite realizar filtrado personalizado de productos a través de la API REST. Proporciona dos opciones de filtrado:
    - category: Permite filtrar productos por la ID de la categoría.
    - name: Permite buscar productos por nombre o descripción, utilizando el operador "icontains" para buscar coincidencias parciales.
    - Además, este filtro incluye una función personalizada filter_by_search para combinar la búsqueda en el nombre y la descripción de los productos.

Frontend (VUE)

Archivo Vue Principal (App.vue)

- **Descripción**
  - El archivo App.vue es el componente raíz de la aplicación Vue.js. Define la estructura general de la aplicación, incluyendo la barra de navegación, la sección de navegación, la vista de enrutamiento y el pie de página.
- **Plantilla (Template)**
  - El componente utiliza una plantilla Vue para definir la estructura de la página.
  - Se verifica la ruta actual utilizando $route.path para decidir si se debe mostrar la barra de navegación y la sección de navegación.
  - Se utiliza <router-view/> para mostrar el contenido de la vista enrutada.
  - Incluye el componente de pie de página al final de la página.
- **Script**
  - En la sección de script, se importan los componentes que se utilizan en la plantilla.
  - Los componentes importados se utilizan en la plantilla como elementos personalizados.

Componentes

- **Componente NavbarComponent**
  - **Descripción**
    - El componente NavbarComponent representa la barra de navegación superior de la aplicación. Proporciona opciones de navegación y una barra de búsqueda para los usuarios.
  - **Plantilla (Template)**
    - La plantilla define la estructura de la barra de navegación, que incluye un logotipo, opciones de menú, un campo de búsqueda y un botón de búsqueda.
    - Las rutas a las que apuntan los enlaces (<a>) se configuran a través de los atributos href.
  - **Script**
    - En la sección de script, se importan las funciones y referencias necesarias.
    - Se define un ref llamado searchText que almacena el texto de búsqueda ingresado por el usuario.
    - Se utiliza defineEmits para definir un evento personalizado llamado getSearchText que se emite cuando se hace clic en el botón de búsqueda.

- **Componente NavigationComponent**
  - **Descripción**
    - El componente NavigationComponent representa la sección de cabecera de la aplicación que muestra el título "Rotiseria" y botones para seleccionar categorías de productos. Esta sección es común en la mayoría de las vistas de la aplicación.
  - **Plantilla (Template)**
    - La plantilla define la estructura de la sección de cabecera.
    - Incluye el título "Rotiseria" y botones para seleccionar categorías de productos, si la ruta actual no es /about ni /upload.
    - Utiliza un bucle v-for para iterar a través de las categorías y crear botones de categoría dinámicamente.
  - **Script**
    - En la sección de script, se importan las funciones y referencias necesarias, como axios para hacer solicitudes a la API de Django.
    - Se define un ref llamado categories para almacenar las categorías de productos.
    - Se define un ref llamado categoryID y categoryName para almacenar la categoría seleccionada.
    - Utiliza defineEmits para definir un evento personalizado llamado getCategoryID que se emite cuando se selecciona una categoría.
    - El método getCategory emite este evento y pasa la ID y el nombre de la categoría seleccionada.
    - Se utiliza el gancho onMounted para cargar las categorías desde la API de Django cuando el componente se monta.

Vistas

- **Vista principal (HomeView.vue)**
  - **Descripción**
    - La vista principal de la aplicación representa la página principal de la tienda de rotisería. Muestra una barra de navegación superior, una sección de categorías y una lista de productos. Los usuarios pueden filtrar productos por categoría o realizar búsquedas por nombre o descripción.
  - **Plantilla (Template)**
    - La plantilla define la estructura de la vista principal, que consta de varios elementos:
    - La barra de navegación superior se muestra en todo momento.
    - La sección de categorías permite a los usuarios seleccionar una categoría específica.
    - Se muestra un mensaje informativo si se ha aplicado un filtro por categoría o búsqueda.
    - Los productos se muestran en tarjetas, con una imagen, nombre, categoría, descripción y precio.

- **Script**
  - En la sección de script, se importan las funciones y referencias necesarias, como axios para hacer solicitudes a la API de Django, y los componentes NavbarComponent y NavigationComponent.
  - Se definen varios ref para almacenar productos, productos filtrados, categorías seleccionadas y texto de búsqueda.
  - Los métodos search, categoryID y resetFilter se utilizan para aplicar filtros y restablecerlos.
  - En el gancho onMounted, se cargan los productos desde la API de Django cuando la vista se monta.

- **Vista de Carga de Producto (UploadProducts.vue)**
  - **Descripción**
    - La vista de carga de producto permite a los usuarios agregar nuevos productos a la tienda de rotisería. Los usuarios pueden ingresar detalles del producto, como nombre, imagen, categoría, descripción, precio y tipo de precio. La vista también muestra mensajes de éxito o error después de enviar un producto.
  - **Plantilla (Template)**
    - La plantilla define la estructura de la vista de carga de producto.
    - Incluye un formulario con campos para ingresar los detalles del producto.
    - Los campos incluyen nombre, imagen, categoría, descripción, precio y tipo de precio.
    - Los mensajes de éxito o error se muestran según el resultado de la operación.
  - **Script**
    - En la sección de script, se importan las funciones y referencias necesarias, como axios para hacer solicitudes a la API de Django.
    - Se definen ref para almacenar los datos del producto, categorías, mensajes de éxito y mensajes de error.
    - La función submitProduct se utiliza para enviar los datos del producto al backend.
    - La función handleImageUpload se utiliza para manejar la carga de imágenes.
    - Se realiza una llamada a la API de Django para cargar las categorías al montar el componente.

Enrutador (router/index.js)

- **Descripción**
  - El archivo index.js del enrutador define las rutas y sus componentes asociados para la navegación en tu aplicación Vue.js. Utiliza la biblioteca vue-router para gestionar las rutas y cargar los componentes correspondientes en función de la URL.
- **Contenido**
  - Importaciones: Se importan las funciones createRouter y createWebHistory de vue-router para crear el enrutador. Además, se importan los componentes asociados a las rutas.
  - Rutas (Routes): Se definen las rutas de la aplicación. Cada ruta se compone de los siguientes campos:
  - path: La URL de la ruta.
  - name: El nombre de la ruta.
  - component: El componente asociado a la ruta. En el caso de la ruta /about y /upload, se utilizan rutas dinámicas con carga diferida (lazy loading) para dividir el código en partes más pequeñas y mejorar el rendimiento.
  - Creación del enrutador: Se crea el enrutador utilizando createRouter y se le pasa la configuración, incluyendo el tipo de historia (en este caso, createWebHistory) y las rutas definidas anteriormente.
  - Exportación del enrutador: El enrutador se exporta para que pueda ser utilizado en otras partes de la aplicación.