Manual Técnico - AppPionono v1.0

6 de septiembre de 2022

La aplicación ha sido desarrollada utilizando el framework Django 4.+, bootstrap 5, con la base de datos postgresql presenta la siguiente estructura de carpetas:

1. aplication

Esta es la carpeta principal, en el interior de la misma los archivos más importantes son los siguientes:

- settings.py: Contiene la configuración de la aplicación, se indican las apps creadas (que estan contenidas en la carpeta apps) y también la configuración de la base de datos.
- urls.py: Contiene las urls del aplicativo:
  - admin : El dashboard default de django desde donde se pueden editar distintos campos de la base de datos, así como crear y dar permisos a los usuarios.
  - raiz: Es la pantalla de inicio de sesión, si el usuario se autentica se redirecciona de forma automática a ventas.
  - ventas: Acá se encuentra el dashboard donde se pueden observar los diferentes pedidos que se obtienen a partir del consumo de web service de square.
- .env: Contiene las variables de entorno del proyecto, como el token de square, usuarios y contraseñas de la base de datos. En produccion la variable realacionada con DEBUG debe ser False.

2. automatizacion

Esta no es una carpeta propia del proyecto Django, pero contiene el service y el timer necesarios para automatizar la sincronización de los productos desde square. Para poder activarlos se realiza un link simbólico a cada uno de estos archivos desde */etc/systemd/system*, y se activa el timer via systemd. En el service dependiendo de la distribución Linux ejecutará el script de automatización con `python3 -m scripts.automaticsinc` o `python -m scripts.automaticsinc`, para esto se tiene que establecer el directorio de trabajo `WorkingDirectory = /var/www/webappPIO/aplication`.

3. static

Contiene 4 carpetas principales:

- admin: Esta carpeta se crea automáticamente y contiene los archivos estáticos para la configuración del panel de Django.
- css: Contiene un archivo llamado custom.css, donde se hacen algunas configuraciones personalizadas a bootstrap.

- img: Contiene el logo de la aplicación y la imágen de inicio de sesión.
- js: Contiene dos archivos:
  - autocomplete.js : Sirve para realizar la busqueda de productos en el modal detail por autocompletado, en la parte inicial se pueden configurar los parámetros de busqueda, como por ejemplo el número de letras a partir del cual ser realizará la busqueda y el número de líneas que se mostrará.
  - custom.js: Se puede encontrar el conjunto de funciones e instrucciones de JavaScript que se encargan de personalizar varias funciones del dashboard de pedidos, además realiza la conexión con la api_rest creada en views para obtener diferentes datos necesarios del aplicativo. Cada una de las funciones estan comentadas en el archivos para comprender la utilidad que desempeñan.

4. apps

Contiene una carpeta por cada app creada para el proyecto, el interior de cada app tiene cuatro archivos principales (views.py, models.py, admin.py,urls.py).

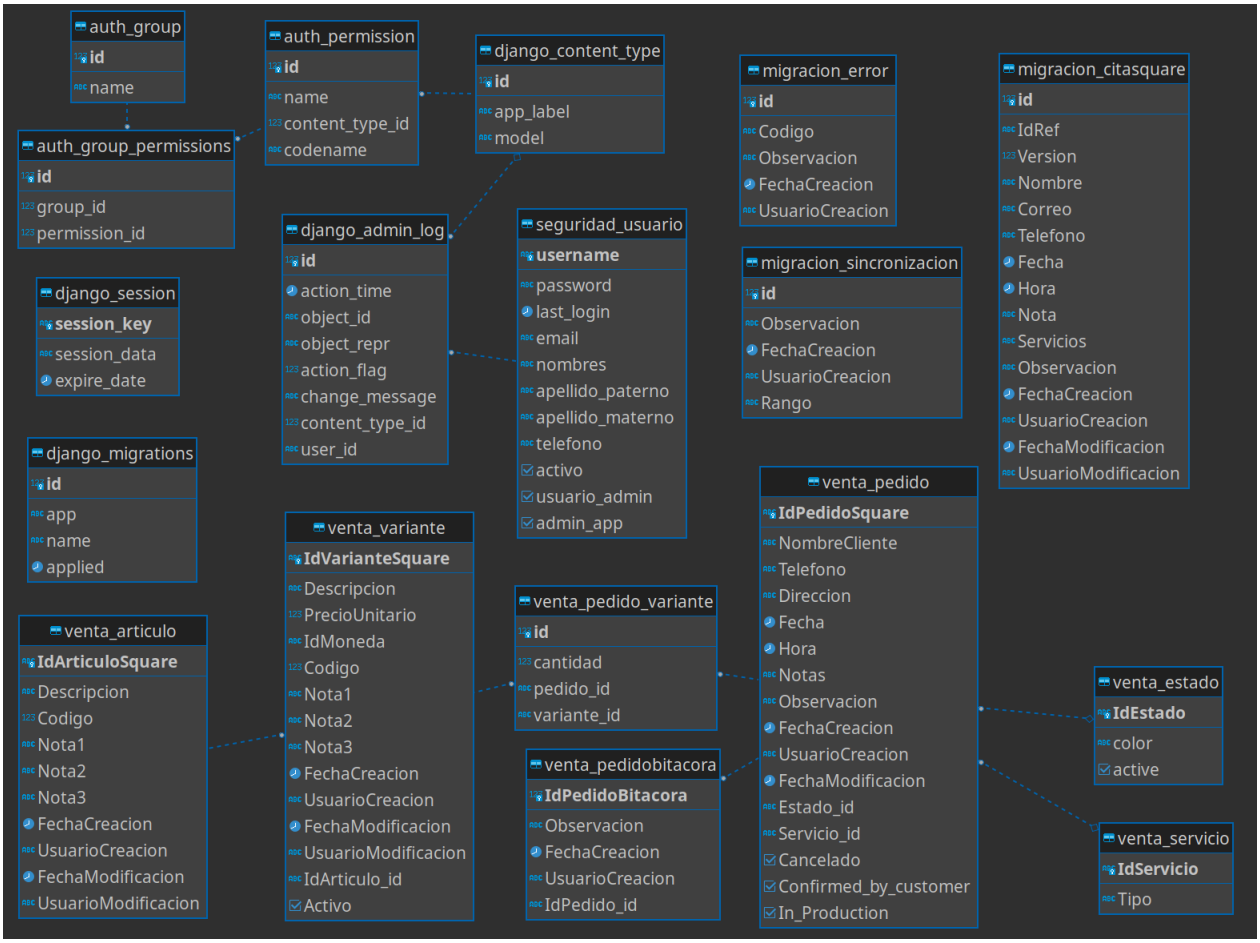
4.1. views

A continuación se detallan las características principales del archivo views.py para cada app

- migración: Solo sirve para modelar las tablas de migraciones, esa configuración de tablas se puede ver en models.py. No se configuran vistas pues las migraciones no tienen templates asociados, funcionan en segundo plano o mediante el botón sinc de la app de ventas.
- seguridad: Acá se configura la redirección 404. En urls se establece el login utilizando la función de logín por defecto que tiene django, es decir no se contruye ninguna vista.
- venta: Es la app principal pues en esta sección se establece la lógina del dashboard de la sección de pedidos. Contiene tres vistas:
  - home: Se encarga de controlar la vista del dashboard de pedidos, envía un contexto dependiendo de la situación, por defecto enviará los datos de los pedidos del día actual y el siguiente. Si recibe un POST ejecutará una acción dependiendo del `request.POST['nombre']`. Este puede tomar los siguientes valores:
    - sinc: Recibe el rango de fechas en el que se desea realizar la sincronización, ejecuta las funciones de sincronizar pedidos de square y variantes y devulve como contexto esos valores.
    - search: Funciona cuando se hace una busqueda usando el boton search, es decir recibe el texto a buscar y devuelve los resultados que se consultan a nivel de base de datos. No sincroniza con square, solo busca resultados ya existentes, el paquete usado se indica en la línea `from django.db.models import Q`
    - addproduct: Desde el modal de añadir productos se recibe una lista de productos añádidos por el usuario y se los almacena en la base de datos.
  - api_productos: Es una api rest que recibe consultas desde la página de ventas, esta api no tiene credenciales, pero solo se puede acceder a ella desde un usuario que haya iniciado sesión. Realiza dos funciones principales:
    - Envía un json con la lista de los detalles de cada producto.
    - Cuando recibe como parámetro `IdPedido` devuelve los detalles de ese pedido, esto se usa con el modal detail.
  - EportaExcel: La librería necesaria para exportar a excel es *openpyxl* y se importa con la línea `from openpyxl import Workbook`

4.2. models

Los modelos se crean en los archivos *models.py* de cada carpeta views. Estos tienen el siguiente diagrama entidad – relación:



5. templates

Acá están contenidos los templates html del aplicativo. Se encuentran tres carpetas principales:

- aplication: Se encuentra el template base.html, este es la base para todos los templates del aplicativo.
- seguridad: Contiene el template home.html usado para el login.
- venta: Contiene los siguientes templates:
  - home.html : Plantilla principal de página pedidos.
  - modaldetail.html : Detalle de pedidos para usuario tipo admin_app.
  - modaldetail_lectura.html: Detalle de pedidos para usuario solo lectura.
  - modalsinc.html: Modal de sincronización de pedidos.
  - navbar.html: Barra superior de navegación, contiene el botón de logout.
  - sidebar.html: Barra lateral de navegación.

6. scripts

Este módulo no es propio de Django, se crea con la intención de realizar diferentes tareas que integran los consumos de web service y la base de datos del aplicativo. Los scripts que contiene son los siguientes:

- consumoapi.py: Contiene tres funciones, que cumplen las tareas de hacer los consumos de la lista de artículos, lista de pedidos y obtención de los datos del customer.
- conexioni_db.py: Establece la conexión con la base de datos postgresql fuera el ORM de django.
- utilidades_db.py: Guarda los datos que se consumen de la api de square en las tablas correspondientes de pedidos y citas_square, tambien se encarga de registrar un historial a modo de bitácoras en la base de datos por cada interacción realizada por el usuario o de forma automática por el sistema.
- automaticsinc.py: Este es el script que se usa para la automatización de la sincronización con la base de datos, acá se puede configurar la ventana de tiempo en la cual se realiza esta sincronización automática.