

Informe de Laboratorio 07

Tema: Django Rest Framework

Nota

Estudiante	Escuela	Asignatura
Hidalgo Chinchay, Paulo Andre phidalgo@unsa.edu.pe Huayhua Mayta, Iván Rodrigo ihuayhuam@unsa.edu.pe Jaita Chura, José Manuel jjaitac@unsa.edu.pe Garcia Valdivia, Ronald Pablo rgarciava@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Semestre: III Código: 1702122

Laboratorio	Tema	Duración
07	Django Rest Framework	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 26 Junio 2023	Al 03 Julio 2023

1. Competencias del curso

- General: C.c. Diseña responsablemente aplicaciones web, sus componentes o procesos para satisfacer necesidades dentro de restricciones realistas: económicas, medio ambientales, sociales, políticas, éticas, de salud, de seguridad, manufacturación y sostenibilidad.
- Específica: C.m. Construye responsablemente soluciones con tecnología web siguiendo un proceso adecuado llevando a cabo las pruebas ajustada a los recursos disponibles del cliente.
- Específica: C.p. Aplica de forma flexible técnicas, métodos, principios, normas, estándares y herramientas del desarrollo web necesarias para la construcción de aplicaciones web e implementación de estos sistemas en una organización.

2. Resultado del estudiante

- RE. 2 La capacidad de aplicar diseño de ingeniería para producir soluciones a problemas y diseñar sistemas, componentes o procesos para satisfacer necesidades específicas dentro de consideraciones realistas en los aspectos de salud pública, seguridad y bienestar; factores globales, culturales, sociales, económicos y ambientales.

- RE. 8 La capacidad de crear, seleccionar y utilizar técnicas, habilidades, recursos y herramientas modernas de ingeniería y tecnologías de la información, incluyendo la predicción y el modelamiento, con una comprensión de las limitaciones.

3. Equipos, materiales y temas

- Sistema Operativo (GNU/Linux de preferencia).
- GNU Vim.
- Python 3.
- Git.
- Cuenta en GitHub con el correo institucional.
- Entorno virtual.
- Django 4.
- djangoestframework.

4. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/123ihuayhua/pweb2-lab-c-23a.git`
- URL para el laboratorio 07 en el Repositorio GitHub.
- `https://github.com/123ihuayhua/pweb2-lab-c-23a/tree/main/Lab7-Pweb2`

5. Admin, user y tokens

- Admin:
 - Usuario: admin
 - Contraseña: 12345
 - Token: eef53ad0f9a274f2d2f27426410abb4fe3fb855
- User:
 - Usuario: Alfonso_M
 - Contraseña: alfonso
 - Token: 8aff82557e15d93fe60bfac6f6f86fa9ae34fc3c8

6. Tarea

- En sus grupos de trabajo correspondientes. Elabore un servicio web que tenga un CRUD con el uso de este framework.
- Create - POST
- Read - GET
- Update - PUT
- Delete - DELETE
- Centrarce en el Core business de su aplicación web. Lo más importante y necesario que este disponible a través de un servicio web.
- Ejemplos: <https://reqbin.com/>, <https://www.googleapis.com/youtube/v3/playlistItems>
- Muestre la funcionalidad consumiéndola desde el cliente Rest de su preferencia.
- El método GET puede ser directamente consumido por un navegador web:
- Por ejemplo: En esta API se puede obtener la temperatura de Arequipa en un rango de fechas: (La versión gratuita tiene un retraso de 7 días, por tanto sólo mostrará la temperatura en Arequipa desde el 01 de Julio hasta el 03 de Julio)
- https://archive-api.open-meteo.com/v1/archive?latitude=-16.39889&longitude=-71.535&start_date=2023-07-01&end_date=2023-07-10&hourly=temperature_2m&daily=temperature_2m_max,temperature_2m_min&timezon

7. Pregunta

- ¿Cuál fué la mayor dificultad del uso de este framework?.
 - Manejo de errores y excepciones: Es importante manejar correctamente los errores y excepciones en una API REST para proporcionar respuestas adecuadas a los usuarios o clientes de la API. (Huayhua Mayta, Iván Rodrigo)
 - Validación y autenticación: La validación y autenticación de las solicitudes y respuestas pueden ser desafiantes, especialmente cuando se trata de lidiar con diferentes niveles de permisos y roles de usuario. (García Valdivia, Ronald Pablo)
 - Integración con otras aplicaciones: Al integrar Django REST Framework con otras aplicaciones y módulos, pueden surgir problemas de compatibilidad o conflictos con otras bibliotecas o extensiones utilizadas en el proyecto. (Jaita Chura, José Manuel)
 - Configuración compleja: Si bien DRF es poderoso y versátil, puede ser complicado configurar y personalizar, especialmente cuando se necesita una funcionalidad específica o características avanzadas. (Hidalgo Chinchay, Paulo Andre)

8. Entregable

- El informe debe tener un enlace al directorio específico del laboratorio en su repositorio GitHub privado donde esté todo el código fuente y otros que sean necesarios. Evitar la presencia de archivos: binarios, objetos, archivos temporales, cache, librerías, entornos virtuales. Si hay configuraciones particulares puede incluir archivos de especificación como: requirements.txt, o leeme.txt.
- No olvide que el profesor debe ser siempre colaborador a su repositorio (Usuario del profesor @rescobedoq).

- Para ser considerado con la calificación de máxima nota, el informe debe estar elaborado en LATEX
- Usted debe describir sólo los commits más importantes que marcaron hitos en su trabajo, adjuntando capturas de pantalla, del commit, del código fuente, de sus ejecuciones y pruebas.
- En el informe siempre se debe explicar las imágenes (código fuente, capturas de pantalla, commits, ejecuciones, pruebas, etc.) con descripciones puntuales pero precisas.
- Partes de entrega:
 - Django orientado a Usuarios finales
 - Tema para clientes, con funcionalidades importantes.
 - Recomendaciones: CRUD en uno de los procesos para el cliente final
 - Ejemplo 1: Para la WebApp Library (Sistema de Biblioteca virtual), sería .El cliente reserva un ejemplar de un libro determinado”.
 - Ejemplo 2: Para la WebApp Enrollments (Sistema para Inscripciones en laboratorios), sería. El alumno se inscribe en un laboratorio disponible para un curso”.
- carrito.html:
- `% extends 'base.html' %`: Indica que esta plantilla hereda del archivo 'base.html'. Esto significa que la plantilla actual se extiende a partir de otra plantilla base, que contiene la estructura y el diseño común de todas las páginas del sitio.
-
-
- `{% link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/sweetalert2@11.1.4/dist/sweetalert2.min.css" %}`: Se carga la hoja de estilos de la librería Swal (SweetAlert2), que se utiliza para mostrar alertas y mensajes emergentes.
- `{% div class="banner" %}{% enddiv %}`: Aquí se muestra un banner con una imagen y un título “Carrito de Compras”. La imagen del banner se obtiene de la carpeta de archivos estáticos usando la etiqueta `{% static %}`.
- `{% div %}{% enddiv %}`: Esta parte muestra mensajes de alerta (en caso de que haya) utilizando el framework de mensajes de Django.
- `{% div class="container" %}{% enddiv %}`: Aquí se muestra el contenido principal del carrito de compras. Si el carrito no está vacío (es decir, si hay elementos en el carrito), se muestra una tabla con los detalles de los productos en el carrito. Cada fila de la tabla contiene el nombre del producto, la cantidad, el precio unitario, el subtotal y un botón para eliminar el producto del carrito. Además, se muestra el total del carrito al final de la tabla. Los botones de la tabla llaman a funciones JavaScript (sumarCantidad, restarCantidad, actualizarCantidad y eliminarProducto) para actualizar la cantidad de productos en el carrito.
- `{% script %}{% endscript %}`: Este bloque contiene código JavaScript que define funciones utilizadas para gestionar la cantidad de productos en el carrito, eliminar productos, guardar el pedido y cancelar el carrito. También se utiliza la librería Swal para mostrar mensajes emergentes de confirmación y notificación.

- En resumen, esta plantilla HTML muestra el contenido del carrito de compras de una aplicación web, permite al usuario interactuar con los elementos del carrito (agregar o disminuir la cantidad de productos, eliminar productos) y realizar acciones como guardar el pedido o cancelar el carrito, todo ello utilizando JavaScript y alertas visuales a través de SweetAlert2. Además, extiende de una plantilla base, lo que le da una estructura y diseño común en toda la aplicación.

Listing 1: carrito.html

```

1  {% extends 'base.html' %}
2  {% load static %}
3  {% block content %}
4  <link rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/sweetalert2@11.1.4/dist/sweetalert2.min.css">
5  <div class="banner">
6      
7      <h1 class="text-center text-warning text-uppercase fw-bolder"
8          style="position: absolute; top: 50%; left: 50%; transform: translate(-50%,
        -50%);">Carrito de Compras</h1>
9  </div>
10 <div>
11     {% if messages %}
12     <div class="container">
13         <div class="row justify-content-center">
14             <div class="col-md-8">
15                 {% for message in messages %}
16                 <div class="alert alert-danger alert-dismissible fade show" role="alert">
17                     {{ message }}
18                     <button type="button" class="btn-close" data-bs-dismiss="alert"
19                         aria-label="Close"></button>
20                 </div>
21                 {% endfor %}
22             </div>
23         </div>
24     {% endif %}
25 </div>
26 <div class="container">
27     <br>
28     {% csrf_token %}
29     {% if carrito %}
30     <table class="table">
31         <thead>
32             <tr>
33                 <th>Producto</th>
34                 <th>Cantidad</th>
35                 <th>Precio Unitario</th>
36                 <th>Subtotal</th>
37                 <th>Acciones</th>
38             </tr>
39         </thead>
40         <tbody>
41             {% for item in carrito %}
42             <tr>
43                 <td>{{ item.articulo.ArtNom }}</td>
44                 <td>

```

```

45         <div class="input-group">
46             <button class="btn btn-outline-secondary" type="button"
47                 onclick="restarCantidad({{ item.pk }})">-</button>
48             <input type="number" class="form-control" value="{{ item.cantidad }}"
49                 min="1"
50                 max="{{ item.articulo.ArtSto }}" onchange="actualizarCantidad({{
51                     item.pk }}, this.value)">
52             <button class="btn btn-outline-secondary" type="button"
53                 onclick="sumarCantidad({{ item.pk }}, {{ item.articulo.ArtSto
54                     }})">+</button>
55         </div>
56     </td>
57     <td>S/{{ item.articulo.ArtPreUni }}</td>
58     <td>S/{{ item.subtotal }}</td>
59     <td>
60         <a href="{{ url 'eliminar_del_carrito' item.pk }}" class="btn btn-danger"
61             onclick="eliminarProducto({{ item.pk }})">Eliminar</a>
62     </td>
63 </tr>
64 </tbody>
65 </tfoot>
66 </table>
67 <div class="d-flex justify-content-around">
68     <a href="{{ url 'productos' }}" class="btn btn-primary">Seguir comprando</a>
69     <button class="btn btn-success" onclick="guardarPedido()">Guardar pedido</button>
70     <button class="btn btn-danger" onclick="cancelarCarrito()">Cancelar</button>
71 </div>
72 {% else %}
73 <br>
74 <p>No hay productos en el carrito.</p>
75 <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
76 <script>
77     Swal.fire({
78         icon: 'info',
79         title: 'Carrito vaco',
80         text: 'No hay productos en el carrito.',
81         showConfirmButton: false,
82         timer: 2000
83     });
84 </script>
85 {% endif %}
86 </div>
87 <br>
88 <script>
89     function sumarCantidad(itemPk, maxCantidad) {
90         var inputCantidad = document.querySelector('[data-item-pk="{{ itemPk }}"]');
91         var cantidad = parseInt(inputCantidad.value);
92         if (cantidad < maxCantidad) {
93             cantidad++;

```

```
98         inputCantidad.value = cantidad;
99         actualizarCantidad(itemPk, cantidad);
100     }
101 }
102
103 function restarCantidad(itemPk) {
104     var inputCantidad = document.querySelector('[data-item-pk="${itemPk}"]');
105     var cantidad = parseInt(inputCantidad.value);
106     if (cantidad > 1) {
107         cantidad--;
108         inputCantidad.value = cantidad;
109         actualizarCantidad(itemPk, cantidad);
110     }
111 }
112
113 function actualizarCantidad(itemPk, cantidad) {
114     fetch('/actualizar_cantidad/${itemPk}/${cantidad}/', {
115         method: 'POST',
116         headers: {
117             'Content-Type': 'application/json',
118             'X-CSRFToken': '{ csrf_token }'
119         },
120         body: JSON.stringify({})
121     })
122     .then(response => {
123         if (response.ok) {
124             location.reload();
125         } else {
126             console.error('Error:', response.statusText);
127         }
128     })
129     .catch(error => {
130         console.error('Error:', error);
131     });
132 }
133
134 function eliminarProducto(itemPk) {
135     event.preventDefault();
136     const swalWithBootstrapButtons = Swal.mixin({
137         customClass: {
138             confirmButton: 'btn btn-success',
139             cancelButton: 'btn btn-danger'
140         },
141         buttonsStyling: false
142     });
143
144     swalWithBootstrapButtons.fire({
145         title: 'Ests seguro de que quieres eliminar este producto del carrito?',
146         text: "Esta accin no se puede deshacer.",
147         icon: 'warning',
148         showCancelButton: true,
149         confirmButtonText: 'S, eliminar',
150         cancelButtonText: 'No, cancelar',
151         reverseButtons: true,
152         timer: null
153     }).then((result) => {
154         if (result.isConfirmed) {
```

```
154     fetch('/eliminar_del_carrito/${itemPk}/', {
155         method: 'POST',
156         headers: {
157             'Content-Type': 'application/json',
158             'X-CSRFToken': '{ csrf_token }'
159         },
160         body: JSON.stringify({})
161     })
162     .then(response => {
163         if (response.ok) {
164             swalWithBootstrapButtons.fire({
165                 icon: 'success',
166                 title: 'El producto ha sido eliminado del carrito.',
167                 showConfirmButton: true,
168                 backdrop: 'rgba(30, 12, 204, 0.49)'
169                 url('https://media.giphy.com/media/3oEjI6SIHbDRxXI40/giphy.gif")
170                 center center
171                 no-repeat'
172             })
173             .then(() => {
174                 location.reload();
175             });
176         } else {
177             console.error('Error:', response.statusText);
178         }
179     })
180     .catch(error => {
181         console.error('Error:', error);
182     });
183 } else if (
184     result.dismiss === Swal.DismissReason.cancel
185 ) {
186     swalWithBootstrapButtons.fire({
187         icon: 'error',
188         title: 'El producto no ha sido eliminado del carrito.',
189         backdrop: 'rgba(255, 47, 79, 0.42)'
190         url('https://media.giphy.com/media/3oEjI6SIHbDRxXI40/giphy.gif")
191         center center
192         no-repeat'
193     });
194 }
195 });
196 }
197 function guardarPedido() {
198     fetch('/guardar_pedido/', {
199         method: 'POST',
200         headers: {
201             'Content-Type': 'application/json',
202             'X-CSRFToken': '{ csrf_token }'
203         },
204         body: JSON.stringify({})
205     })
206     .then(response => {
207         if (response.ok) {
208             Swal.fire({
209                 icon: 'success',
```



```

210         title: 'El pedido ha sido guardado exitosamente.',
211         showConfirmButton: true,
212         backdrop: 'rgba(0, 128, 0, 0.49)'
213         url("https://media.giphy.com/media/3oEjI6SIHBdRxXI40/giphy.gif")
214         center center
215         no-repeat'
216     }).then(() => {
217         location.href = '{% url "guardar_pedido" %}';
218     });
219 } else {
220     console.error('Error:', response.statusText);
221 }
222 })
223 .catch(error => {
224     Swal.fire({
225         icon: 'error',
226         title: 'Error al guardar el pedido',
227         text: 'La cantidad en el pedido supera el stock disponible.',
228         showConfirmButton: true
229     });
230     console.error('Error:', error);
231 });
232 }
233
234 function cancelarCarrito() {
235     const swalWithBootstrapButtons = Swal.mixin({
236         customClass: {
237             confirmButton: 'btn btn-success',
238             cancelButton: 'btn btn-danger'
239         },
240         buttonsStyling: false
241     });
242
243     swalWithBootstrapButtons.fire({
244         title: ' ¿ Ests  seguro de que quieres cancelar el carrito?',
245         text: 'Se eliminarn todos los productos del carrito.',
246         icon: 'warning',
247         showCancelButton: true,
248         confirmButtonText: 'S, cancelar',
249         cancelButtonText: 'No, mantener',
250         reverseButtons: true
251     }).then((result) => {
252         if (result.isConfirmed) {
253             fetch('/cancelar_carrito/', {
254                 method: 'POST',
255                 headers: {
256                     'Content-Type': 'application/json',
257                     'X-CSRFToken': '{{ csrf_token }}'
258                 },
259                 body: JSON.stringify({})
260             })
261                 .then(response => {
262                     if (response.ok) {
263                         swalWithBootstrapButtons.fire({
264                             icon: 'success',
265                             title: 'El carrito ha sido cancelado.',

```

```

266         showConfirmButton: true,
267         backdrop: 'rgba(30, 12, 204, 0.49)'
268         url("https://media.giphy.com/media/3oEjI6SIIHBdRxXI40/giphy.gif")
269         center center
270         no-repeat'
271     }).then(() => {
272         location.reload();
273     });
274     } else {
275         console.error('Error:', response.statusText);
276     }
277 })
278 .catch(error => {
279     console.error('Error:', error);
280 });
281 } else if (result.dismiss === Swal.DismissReason.cancel) {
282     swalWithBootstrapButtons.fire({
283         icon: 'error',
284         title: 'El carrito no ha sido cancelado.',
285         backdrop: 'rgba(255, 47, 79, 0.42)'
286         url("https://media.giphy.com/media/3oEjI6SIIHBdRxXI40/giphy.gif")
287         center center
288         no-repeat'
289     });
290 }
291 });
292 }
293 </script>
294 <script
295     src="https://cdn.jsdelivr.net/npm/sweetalert2@11.1.4/dist/sweetalert2.min.js"></script>
    {% endblock %}

```

- pedido_detalle.html.
- En el bloque de contenido, se define una estructura con contenedores para mostrar los datos del pedido.
- La primera parte muestra el encabezado "DATOS DEL PEDIDO" y presenta una tabla con detalles como el cliente, la fecha y el total del pedido.
- La segunda parte muestra una tabla con los detalles de los artículos del pedido (detalles del pedido). Utiliza un bucle `% for detalle in detalles %` para iterar sobre cada detalle del pedido.
- Dentro del bucle, se muestra la información de cada detalle del pedido, incluyendo el artículo, la cantidad, el precio unitario y el subtotal.
- Para mostrar los valores de cada detalle del pedido, se utilizan etiquetas de interpolación de Django, como `pedido.PedCabCodCli` y `detalle.PedDetArtCod`, que acceden a los campos de los modelos relacionados.
- En resumen, esta plantilla se utiliza para mostrar los detalles de un pedido específico en una página web. Los datos del pedido se presentan en una tabla con información como el cliente, la fecha, el total y los detalles de los artículos del pedido.

Listing 2: pedido_detalle.html

```
1 {% extends 'base.html' %}
2
3 {% block content %}
4 <div class="container">
5     <div class="container">
6         <h2 class="text-center fw-bolder">DATOS DEL PEDIDO</h2>
7         <table class="table">
8             <tbody>
9                 <tr>
10                     <th scope="row">CLIENTE</th>
11                     <td>{{ pedido.PedCabCodCli }}</td>
12                 </tr>
13                 <tr>
14                     <th scope="row">FECHA</th>
15                     <td>{{ pedido.PedCabFec }}</td>
16                 </tr>
17                 <tr>
18                     <th scope="row">TOTAL</th>
19                     <td>S/. {{ pedido.PedDetTot }}</td>
20                 </tr>
21             </tbody>
22         </table>
23     </div>
24
25     <div class="container">
26         <table class="table">
27             <thead class="table-dark">
28                 <tr>
29                     <th>Articulo</th>
30                     <th>Cantidad</th>
31                     <th>Precio unitario</th>
32                     <th>Subtotal</th>
33                 </tr>
34             </thead>
35             <tbody class="table-group-divider">
36                 {% for detalle in detalles %}
37                 <tr>
38                     <td>{{ detalle.PedDetArtCod }}</td>
39                     <td>{{ detalle.PedDetCantidad }}</td>
40                     <td>S/. {{ detalle.PedDetPreUniArt }}</td>
41                     <td>S/. {{ detalle.PedDetSubtotal }}</td>
42                 </tr>
43                 {% endfor %}
44             </tbody>
45         </table>
46     </div>
47 </div>
48 {% endblock %}
```

- A continuación se detalla la función de cada parte del código api.py:
- VendedorViewSet: Permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en el modelo Vendedor. Está configurado para que solo los usuarios administradores puedan realizar todas las operaciones, mientras que los usuarios autenticados pueden leer los datos (operaciones

de solo lectura) y los usuarios no autenticados pueden acceder solo para lectura (operaciones de solo lectura).

- **ClienteViewSet:** Similar al **VendedorViewSet**, pero este maneja el modelo **Cliente** y solo permite que los usuarios administradores realicen operaciones CRUD.
- **MarcaViewSet:** Similar a los anteriores, pero se enfoca en el modelo **Marca** y permite que los usuarios administradores realicen todas las operaciones CRUD, mientras que los usuarios autenticados pueden realizar solo operaciones de lectura.
- **TipoArticuloViewSet:** Igual que los anteriores, pero maneja el modelo **TipoArticulo**.
- **ArticuloViewSet:** Permite operaciones CRUD en el modelo **Articulo**, al igual que los otros ViewSets, restringiendo algunas operaciones a los usuarios administradores y permitiendo que los usuarios autenticados realicen operaciones de lectura.
- **PedidoViewSet:** Similar a los anteriores, pero para el modelo **Pedido**, permitiendo que los usuarios administradores realicen todas las operaciones CRUD y que los usuarios autenticados solo realicen operaciones de lectura.
- **CarritoViewSet:** Permite a los usuarios autenticados realizar operaciones CRUD en el modelo **Carrito**.
- **PedidoView (APIView):**
 - **post:** Esta vista permite crear un nuevo objeto **Pedido** utilizando el serializador **PedidoSerializer** y los datos proporcionados en el cuerpo de la solicitud POST. Luego devuelve los datos del pedido creado con el estado HTTP 201 Created.
 - **delete:** Esta vista permite eliminar un objeto **Pedido** específico mediante una solicitud DELETE con el ID del pedido a eliminar (pk). Después de eliminar el pedido, devuelve una respuesta con el estado HTTP 204 No Content.
- Este además nos define quienes pueden ver los datos como re artículos, clientes y vendedores.

Listing 3: api.py

```
1 from django.shortcuts import get_object_or_404
2 from rest_framework.response import Response
3 from .models import *
4 from rest_framework import viewsets, permissions, status
5 from .serializers import *
6 from rest_framework import status
7 from rest_framework.response import Response
8 from rest_framework.views import APIView
9
10 #ViewSets
11 class VendedorViewSet(viewsets.ModelViewSet):
12     queryset = Vendedor.objects.all()
13     serializer_class = VendedorSerializer
14     permission_classes = [(permissions.IsAdminUser & permissions.IsAuthenticated) |
15                          permissions.IsAuthenticatedOrReadOnly]
16
17 class ClienteViewSet(viewsets.ModelViewSet):
18     queryset = Cliente.objects.all()
19     permission_classes = [(permissions.IsAdminUser & permissions.IsAuthenticated)]
20     serializer_class = ClienteSerializer
```

```
20
21 class MarcaViewSet(viewsets.ModelViewSet):
22     queryset = Marca.objects.all()
23     permission_classes = [(permissions.IsAdminUser & permissions.IsAuthenticated) |
24         permissions.IsAuthenticatedOrReadOnly]
25     serializer_class = MarcaSerializer
26
27 class TipoArticuloViewSet(viewsets.ModelViewSet):
28     queryset = TipoArticulo.objects.all()
29     permission_classes = [(permissions.IsAdminUser & permissions.IsAuthenticated) |
30         permissions.IsAuthenticatedOrReadOnly]
31     serializer_class = TipoArticuloSerializer
32
33 class ArticuloViewSet(viewsets.ModelViewSet):
34     queryset = Articulo.objects.all()
35     permission_classes = [(permissions.IsAdminUser & permissions.IsAuthenticated) |
36         permissions.IsAuthenticatedOrReadOnly]
37     serializer_class = ArticuloSerializer
38
39 class PedidoViewSet(viewsets.ModelViewSet):
40     queryset = Pedido.objects.all()
41     serializer_class = PedidoSerializer
42     permission_classes = [(permissions.IsAdminUser & permissions.IsAuthenticated) |
43         permissions.IsAuthenticated]
44
45 class CarritoViewSet(viewsets.ModelViewSet):
46     queryset = Pedido.objects.all()
47     serializer_class = CarritoSerializer
48     permission_classes = [permissions.IsAuthenticated]
49
50 class PedidoView(APIView):
51     def post(self, request):
52         serializer = PedidoSerializer()
53         pedido = serializer.create_pedido(request.data)
54
55         return Response(PedidoSerializer(pedido).data, status=status.HTTP_201_CREATED)
56
57     def delete(self, request, pk):
58         pedido = Pedido.objects.get(pk=pk)
59         pedido.eliminar()
60         return Response(status=status.HTTP_204_NO_CONTENT)
```

- La funcionalidad de models.py es la siguiente:
- Vendedor: Representa un vendedor en la aplicación. Tiene campos como VenDNI (documento de identidad), VenApePat (apellido paterno), VenNom (nombre), y VenEstReg (estado de registro). El método `__str__` devuelve el nombre completo del vendedor.
- Cliente: Hereda de `AbstractUser`, lo que significa que es un modelo de usuario personalizado. Tiene campos adicionales como CliDNI (documento de identidad), CliApePat (apellido paterno), CliNom (nombre), y CliEstReg (estado de registro). También tiene los campos `password` y `username`, pero se definen como campos adicionales en lugar de utilizar los campos ya existentes de `AbstractUser`. El método `__str__` devuelve el nombre completo del cliente. Además, se define el método `generate_auth_token`, que crea un token de autenticación asociado al cliente.
- Marca: Representa una marca en la aplicación. Tiene campos como `MarNom` (nombre de la

marca), MarImg (imagen de la marca), y MarEstReg (estado de registro). El método `__str__` devuelve el nombre de la marca.

- **TipoArticulo:** Representa un tipo de artículo en la aplicación. Tiene campos como `TipArtNom` (nombre del tipo de artículo) y `TipArtEstReg` (estado de registro). El método `__str__` devuelve el nombre del tipo de artículo.
- **Articulo:** Representa un artículo en la aplicación. Tiene campos como `ArtMarCod` (clave externa que vincula el artículo con una marca), `ArtTipCod` (clave externa que vincula el artículo con un tipo de artículo), `ArtNom` (nombre del artículo), `ArtDes` (descripción del artículo), `ArtImg` (imagen del artículo), `ArtSto` (stock disponible), `ArtPreUni` (precio unitario), y `ArtEstReg` (estado de registro). Los métodos `__str__`, `disminuir_stock` y `aumentar_stock` se utilizan para obtener una representación de texto del artículo y actualizar el stock del mismo.
- **Pedido:** Representa un pedido realizado por un cliente. Tiene campos como `PedCabCodCli` (clave externa que vincula el pedido con un cliente), `PedCabFec` (fecha del pedido, se establece automáticamente al crear el pedido), `PedDetArtCod` (clave externa que vincula el pedido con un artículo), `PedDetCantidad` (cantidad del artículo en el pedido), `PedDetPreUniArt` (precio unitario del artículo en el momento del pedido), `PedDetSubtotal` (subtotal del artículo en el pedido), `PedDetTot` (total del pedido), y `PedDetEstReg` (estado de registro). El método `save` se utiliza para calcular el subtotal y el total del pedido y actualizar el precio unitario. También se definen los métodos `__str__`, `total`, y `eliminar` para obtener una representación de texto del pedido, calcular el total del pedido y eliminar el pedido.
- **Carrito:** Representa el carrito de compras de un cliente. Tiene campos como `usuario` (clave externa que vincula el carrito con un cliente), `articulo` (clave externa que vincula el carrito con un artículo), `cantidad` (cantidad del artículo en el carrito), y `pedido` (clave externa que vincula el carrito con un pedido, si el artículo se ha comprado). Los métodos `__str__`, `subtotal`, y `total` se utilizan para obtener una representación de texto del carrito, calcular el subtotal y el total del artículo en el carrito.
- En resumen, este código define los modelos y métodos necesarios para implementar una aplicación web que maneja vendedores, clientes, marcas, tipos de artículos, artículos, pedidos y carritos de compras, y realiza algunas funcionalidades adicionales, como el cálculo del total y el subtotal de los pedidos y carritos. También proporciona un método para generar tokens de autenticación para los clientes.

Listing 4: models.py

```
1 from django.db import models
2 from django.contrib.auth.models import User, AbstractUser
3 from rest_framework.auth_token.models import Token
4 # Create your models here.
5
6 # Vendedor
7 class Vendedor(models.Model):
8     VenDNI = models.CharField(max_length=8, unique=True)
9     VenApePat = models.CharField(max_length=20)
10    VenNom = models.CharField(max_length=20)
11    VenEstReg = models.BooleanField(default=True)
12    #Mostrar nombre del vendedor
13    def __str__(self):
14        nombre = self.VenNom + ' ' + self.VenApePat
15        return nombre
16
```

```
17 #Cliente
18 class Cliente(AbstractUser):
19     CliDNI = models.CharField(max_length=8)
20     CliApePat = models.CharField(max_length=20)
21     CliNom = models.CharField(max_length=20)
22     CliEstReg = models.BooleanField(default=True)
23
24     # Mostrar nombre completo del cliente
25     def __str__(self):
26         return self.CliNom + ' ' + self.CliApePat
27
28     password = models.CharField(max_length=128, null=True)
29     username = models.CharField(max_length=150, unique=True, null=True)
30
31     def generate_auth_token(self):
32         token, created = Token.objects.get_or_create(user=self)
33         return token.key
34
35 #Marca
36 class Marca(models.Model):
37     MarNom = models.CharField(max_length=20)
38     MarImg = models.ImageField(upload_to='imagenes/imgs', null=True)
39     MarEstReg = models.BooleanField(default=True)
40     #Mostrar nombre de las marcas
41     def __str__(self):
42         return self.MarNom
43
44 #Tipo Artículo
45 class TipoArticulo(models.Model):
46     TipArtNom = models.CharField(max_length=20)
47     TipArtEstReg = models.BooleanField(default=True)
48     #Mostrar nombre de los tipos de articulos
49     def __str__(self):
50         return self.TipArtNom
51
52 #Articulo
53 class Articulo(models.Model):
54     ArtMarCod = models.ForeignKey(Marca, on_delete=models.CASCADE, null=True)
55     ArtTipCod = models.ForeignKey(TipoArticulo, on_delete=models.CASCADE, null=True)
56     ArtNom = models.CharField(max_length=50, null=True)
57     ArtDes = models.TextField(max_length=1000, help_text='Ingresa la descripción del artículo',
58                               null=True)
59     ArtImg = models.ImageField(upload_to='imagenes/imgs', null=True)
60     ArtSto = models.PositiveSmallIntegerField()
61     ArtPreUni = models.FloatField()
62     ArtEstReg = models.BooleanField(default=True)
63     #Mostrar nombre del Artículo
64     def __str__(self):
65         return self.ArtNom
66
67     def disminuir_stock(self, cantidad):
68         self.ArtSto -= cantidad
69         self.save()
70
71     def aumentar_stock(self, cantidad):
72         self.ArtSto += cantidad
```

```
72         self.save()
73
74     class Pedido(models.Model):
75         PedCabCodCli = models.ForeignKey(Cliente, on_delete=models.CASCADE)
76         PedCabFec = models.DateField(auto_now_add=True)
77         PedDetArtCod = models.ForeignKey(Articulo, on_delete=models.CASCADE)
78         PedDetCantidad = models.PositiveSmallIntegerField(default=1)
79         PedDetPreUniArt = models.FloatField(default=0.0)
80         PedDetSubtotal = models.FloatField(default=0.0)
81         PedDetTot = models.FloatField(default=0.0)
82         PedDetEstReg = models.BooleanField(default=True)
83
84         def save(self, *args, **kwargs):
85             # Obtener el precio del articulo seleccionado
86             precio_articulo = self.PedDetArtCod.ArtPreUni
87
88             # Actualizar el campo PedDetPreUniArt con el precio del articulo
89             self.PedDetPreUniArt = precio_articulo
90
91             # Calcular el subtotal y el total
92             self.PedDetSubtotal = self.PedDetCantidad * self.PedDetPreUniArt
93             self.PedDetTot = self.PedDetSubtotal
94
95             super().save(*args, **kwargs)
96
97         def __str__(self):
98             return f"{self.PedCabCodCli} - {self.PedCabFec}"
99
100        def total(self):
101            detalles = self.detalles.all()
102            return sum(detalle.PedDetTot for detalle in detalles)
103
104        def eliminar(self):
105            # Obtener el articulo correspondiente
106            articulo = self.PedDetArtCod
107
108            # Aumentar el stock del articulo
109            articulo.aumentar_stock(self.PedDetCantidad)
110
111            # Eliminar el pedido
112            self.delete()
113
114    #Carrito de Compras
115    class Carrito(models.Model):
116        usuario = models.ForeignKey(Cliente, on_delete=models.CASCADE)
117        articulo = models.ForeignKey(Articulo, on_delete=models.CASCADE)
118        cantidad = models.PositiveIntegerField(default=1)
119        pedido = models.ForeignKey(Pedido, on_delete=models.CASCADE, null=True, blank=True)
120
121        def __str__(self):
122            return f"{self.usuario} - {self.articulo}"
123
124        def subtotal(self):
125            return self.cantidad * self.articulo.ArtPreUni
126
127        def total(self):
```


128

`return self.subtotal()`

- A continuación, se explican brevemente los serializadores definidos en `serializers.py`:
- `VendedorSerializer`: Se utiliza para serializar y deserializar objetos del modelo `Vendedor`. Los campos que se incluirán en la representación JSON del vendedor son `id`, `VenDNI`, `VenApePat`, `VenNom`, y `VenEstReg`.
- `ClienteSerializer`: Similar al `VendedorSerializer`, pero se utiliza para el modelo `Cliente`. Además de los campos `id`, `CliDNI`, `CliApePat`, `CliNom`, y `CliEstReg`, también incluye los campos `password` y `username`, pero estos campos no serán actualizables en la serialización (operaciones de solo lectura).
- `MarcaSerializer`: Se utiliza para el modelo `Marca` y serializa los campos `id`, `MarNom`, `MarImg`, y `MarEstReg`.
- `TipoArticuloSerializer`: Similar a los anteriores, pero para el modelo `TipoArticulo`. Serializa los campos `id`, `TipArtNom`, y `TipArtEstReg`.
- `ArticuloSerializer`: Se utiliza para el modelo `Articulo`. Serializa los campos `id`, `ArtMarCod`, `ArtTipCod`, `ArtNom`, `ArtDes`, `ArtImg`, `ArtSto`, `ArtPreUni`, y `ArtEstReg`.
- `PedidoSerializer`: Se utiliza para el modelo `Pedido`. Serializa los campos `id`, `PedCabCodCli`, `PedCabFec`, `PedDetArtCod`, `PedDetCantidad`, `PedDetPreUniArt`, `PedDetSubtotal`, `PedDetTot`, y `PedDetEstReg`. El campo `PedCabFec` es de solo lectura y se establece automáticamente al crear el pedido. Además, contiene un método personalizado `create_pedido` que permite crear un nuevo pedido y realizar ciertas operaciones relacionadas con el cálculo del subtotal y la disminución del stock del artículo correspondiente.
- `CarritoSerializer`: Se utiliza para el modelo `Carrito`. Serializa los campos `usuario`, `articulo`, `cantidad`, y `pedido`.
- En resumen, estos serializadores se utilizan para convertir objetos de los modelos Django en representaciones JSON, y también se utilizan para realizar validaciones y crear nuevos objetos, como en el caso del método personalizado `create_pedido` dentro del `PedidoSerializer`. Estos serializadores son útiles para exponer los datos de la aplicación Django a través de una API REST.

Listing 5: `serializers.py`

```
1 from django.shortcuts import get_object_or_404
2 from rest_framework import serializers
3 from .models import *
4
5 class VendedorSerializer(serializers.ModelSerializer):
6     class Meta:
7         model = Vendedor
8         fields = ('id', 'VenDNI', 'VenApePat', 'VenNom', 'VenEstReg')
9         read_only_fields = ('id',)
10
11 class ClienteSerializer(serializers.ModelSerializer):
12     class Meta:
13         model = Cliente
14         fields = ('id', 'CliDNI', 'CliApePat', 'CliNom', 'CliEstReg', 'password', 'username')
15         read_only_fields = ('id',) #Campo no se actualiza
```

```
16
17 class MarcaSerializer(serializers.ModelSerializer):
18     class Meta:
19         model = Marca
20         fields = ('id', 'MarNom', 'MarImg', 'MarEstReg')
21
22 class TipoArticuloSerializer(serializers.ModelSerializer):
23     class Meta:
24         model = TipoArticulo
25         fields = ('id', 'TipArtNom', 'TipArtEstReg')
26         read_only_fields = ('id',)
27
28 class ArticuloSerializer(serializers.ModelSerializer):
29     class Meta:
30         model = Articulo
31         fields = ('id', 'ArtMarCod', 'ArtTipCod', 'ArtNom', 'ArtDes', 'ArtImg', 'ArtSto',
32                 'ArtPreUni', 'ArtEstReg')
33         read_only_fields = ('id',)
34
35 class PedidoSerializer(serializers.ModelSerializer):
36     class Meta:
37         model = Pedido
38         fields = ('id', 'PedCabCodCli', 'PedCabFec', 'PedDetArtCod', 'PedDetCantidad',
39                 'PedDetPreUniArt', 'PedDetSubtotal', 'PedDetTot', 'PedDetEstReg')
40         read_only_fields = ('id', 'PedCabFec',)
41
42     def create_pedido(self, data):
43         # Validar los datos enviados en la solicitud
44         serializer = PedidoSerializer(data=data)
45         serializer.is_valid(raise_exception=True)
46
47         # Obtener los datos validados
48         validated_data = serializer.validated_data
49
50         # Obtener el precio del articulo seleccionado
51         precio_articulo = validated_data['PedDetArtCod'].ArtPreUni
52
53         # Calcular el subtotal y el total
54         subtotal = validated_data['PedDetCantidad'] * precio_articulo
55         total = subtotal
56
57         # Agregar el subtotal y el total a los datos validados
58         validated_data['PedDetSubtotal'] = subtotal
59         validated_data['PedDetTot'] = total
60
61         # Disminuir el stock del articulo correspondiente
62         articulo = validated_data['PedDetArtCod']
63         cantidad = validated_data['PedDetCantidad']
64         articulo.disminuir_stock(cantidad)
65
66         # Crear el pedido
67         pedido = Pedido.objects.create(**validated_data)
68
69         return pedido
70
71 class CarritoSerializer(serializers.ModelSerializer):
```

```
70 class Meta:
71     model = Carrito
72     fields = ('usuario', 'articulo', 'cantidad', 'pedido')
```

- Se importan todos los ViewSets definidos en el archivo api.py utilizando la declaración from .api import *. Esto asume que en el archivo api.py, se han definido los ViewSets con nombres como VendedorViewSet, ClienteViewSet, PedidoViewSet, MarcaViewSet, TipoArticuloViewSet, CarritoViewSet, y ArticuloViewSet.
- Se crea un enrutador de DRF mediante routers.DefaultRouter().
- Se registran los ViewSets y sus rutas correspondientes en el enrutador utilizando el método router.register(). Cada llamada a router.register() asocia un ViewSet a una URL específica de la API.
 - router.register('api/vendedor', VendedorViewSet, 'vendedor'): Asocia el VendedorViewSet a la URL 'api/vendedor/'. Los nombres vendedor se utilizan como prefijos para las rutas de la API asociadas a este ViewSet.
 - De manera similar, otros ViewSets se registran con sus rutas y nombres de prefijo respectivos, como cliente, pedido, tienda, tipoArt, Carrito, y articulo.
- Finalmente, se obtienen todas las rutas registradas en el enrutador mediante router.urls. Estas rutas incluirán las rutas generadas automáticamente para cada ViewSet registrado, siguiendo las convenciones de Django REST Framework.
- En resumen, este código configura automáticamente las rutas y vistas necesarias para implementar una API RESTful utilizando DRF y los ViewSets definidos en el archivo api.py. Los nombres de las rutas y los ViewSets están asociados para exponer los diferentes recursos y operaciones de la API, como obtener una lista de vendedores, crear un nuevo cliente, obtener detalles de un pedido, entre otras acciones.

Listing 6: urls.py

```
1 from rest_framework import routers
2 from .api import *
3
4 router = routers.DefaultRouter()
5
6 router.register('api/vendedor', VendedorViewSet, 'vendedor')
7 router.register('api/cliente', ClienteViewSet, 'cliente')
8 router.register('api/pedido', PedidoViewSet, 'pedido')
9 router.register('api/marca', MarcaViewSet, 'tienda')
10 router.register('api/tipoArt', TipoArticuloViewSet, 'tienda')
11 router.register('api/carrito', CarritoViewSet, 'Carrito')
12 router.register('api/articulo', ArticuloViewSet, 'articulo')
13
14 urlpatterns = router.urls
```

- En el código de view.py:
- ViewSets:

- **VendedorViewSet:** Permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en el modelo Vendedor. Está configurado para que solo los usuarios administradores puedan realizar todas las operaciones, mientras que los usuarios autenticados pueden leer los datos (operaciones de solo lectura).
 - **ClienteViewSet:** Similar al **VendedorViewSet**, pero este maneja el modelo Cliente y solo permite que los usuarios administradores realicen operaciones CRUD.
 - **MarcaViewSet:** Similar a los anteriores, pero se enfoca en el modelo Marca y permite que los usuarios administradores realicen todas las operaciones, mientras que los usuarios autenticados pueden realizar solo operaciones de lectura.
 - **TipoArticuloViewSet:** Igual que los anteriores, pero maneja el modelo TipoArticulo.
 - **ArticuloViewSet:** Permite operaciones CRUD en el modelo Articulo, al igual que los otros ViewSets, restringiendo algunas operaciones a los usuarios administradores.
 - **PedidoViewSet:** Similar a los anteriores, pero para el modelo Pedido, permitiendo que los usuarios administradores realicen todas las operaciones CRUD.
 - **CarritoViewSet:** Permite a los usuarios autenticados realizar operaciones CRUD en el modelo Carrito.
- **PedidoView (APIView):**
- **post:** Esta vista permite crear un nuevo objeto Pedido utilizando el serializador **PedidoSerializer** y los datos proporcionados en el cuerpo de la solicitud POST. Luego devuelve los datos del pedido creado con el estado HTTP 201 Created.
 - **delete:** Esta vista permite eliminar un objeto Pedido específico mediante una solicitud DELETE con el ID del pedido a eliminar (pk). Después de eliminar el pedido, devuelve una respuesta con el estado HTTP 204 No Content.
- Cada ViewSet está configurado para utilizar un serializador específico que define cómo los datos se convierten entre representaciones de Python y formatos de datos como JSON.
- En resumen, este código implementa una API RESTful que permite realizar operaciones CRUD en varios modelos (Vendedor, Cliente, Marca, TipoArticulo, Articulo y Pedido), restringiendo ciertas operaciones solo a los usuarios administradores y proporcionando diferentes niveles de permisos a los usuarios autenticados para lectura y escritura en los recursos de la API. También incluye una vista personalizada **PedidoView** para crear y eliminar objetos Pedido.
- A partir de `@api.view(['POST'])` se encuentra el método para crear un nuevo usuario.

Listing 7: views.py

```
1 from django.shortcuts import render, redirect, get_object_or_404
2 from django.contrib.auth.forms import AuthenticationForm
3 from django.contrib.auth import login, logout, authenticate
4 from django.db import IntegrityError
5 from .forms import PedidoForm, RegistroForm
6 from .models import Carrito, Pedido, Cliente, Marca, Articulo
7 from django.contrib.auth.decorators import login_required
8 from django.contrib.auth.decorators import login_required
9 from django.http import JsonResponse
10 from django.contrib import messages
11 from django.contrib.auth import authenticate
12 from rest_framework.decorators import api_view
13 from rest_framework.response import Response
```

```
14
15 # Create your views here.
16 #Vista principal
17 def Home(request):
18     marcas = Marca.objects.all()
19     arts = Articulo.objects.all()
20     context = {'marcas': marcas, 'arts': arts}
21     return render(request, 'home.html', context)
22
23 #Registrarse (usuario)
24 def Signup(request):
25     if request.method == 'GET':
26         return render(request, 'signup.html', {'form': RegistroForm})
27
28     else:
29         if request.POST['password1'] == request.POST['password2']:
30             try:
31                 # Registrar usuario
32                 user = Cliente.objects.create_user(username=request.POST['username'],
33                                                    password=request.POST['password1'])
34                 # Guardar usuario
35                 user.CliDNI = request.POST['CliDNI']
36                 user.CliApePat = request.POST['CliApePat']
37                 user.CliNom = request.POST['CliNom']
38                 user.CliEstReg = True
39                 user.save()
40                 login(request, user)
41                 return redirect('index')
42             except IntegrityError:
43                 return render(request, 'signup.html', {
44                     'form': RegistroForm,
45                     'error': 'El usuario ya existe'
46                 })
47
48             return render(request, 'signup.html', {
49                 'form': RegistroForm,
50                 'error': 'Las contraseñas no coinciden'
51             })
52
53 @login_required
54 #Ver pedidos
55 def Index(request):
56     pedido = Pedido.objects.filter(PedCabCodCli=request.user)
57     return render(request, 'index.html', {'pedidos' : pedido})
58
59 #Detalle pedido
60 @login_required
61 def DetallePedido(request, pedidoID):
62     pedido_cabecera = get_object_or_404(Pedido, pk=pedidoID, PedCabCodCli=request.user)
63     detalles_pedido = Pedido.objects.filter(PedDetCodCab=pedido_cabecera)
64
65     if detalles_pedido.exists():
66         detalle = detalles_pedido.first()
67     else:
68         detalle = Pedido(PedDetCodCab=pedido_cabecera)
```

```
69     if request.method == 'GET':
70         form = PedidoForm(instance=detalle)
71         carrito = request.session.get('carrito', {})
72         articulos = Articulo.objects.filter(pk__in=carrito.keys())
73         detalles = []
74         for articulo in articulos:
75             cantidad = carrito[str(articulo.pk)][ 'cantidad' ]
76             subtotal = cantidad * articulo.ArtPreUni
77             detalles.append({
78                 'articulo': articulo,
79                 'cantidad': cantidad,
80                 'subtotal': subtotal
81             })
82         return render(request, 'pedido_detalle.html', {
83             'pedido_cabecera': pedido_cabecera,
84             'detalle': detalle,
85             'form': form,
86             'detalles': detalles
87         })
88     else:
89         form = PedidoForm(request.POST, instance=detalle)
90         if form.is_valid():
91             form.save()
92             return JsonResponse({'success': True})
93         else:
94             return JsonResponse({'success': False})
95
96 #Salir de la sesin
97 def Signout(request):
98     logout(request)
99     return redirect('home')
100
101 #Logearse
102 def Signin(request):
103     if request.method == 'GET':
104         return render(request, 'signin.html', {
105             'form' : AuthenticationForm
106         })
107     else:
108         user = authenticate(
109             request, username = request.POST['username'], password = request.POST['password']
110         )
111         if user is None:
112             return render(request, 'signin.html', {
113                 'form' : AuthenticationForm,
114                 'error': 'Usuario o contraseña incorrecta'
115             })
116         else:
117             login(request, user)
118             return redirect('index')
119
120 #Marcas
121 def marca_detail(request, marca_id):
122     marca = get_object_or_404(Marca, pk=marca_id)
123     arts = Articulo.objects.filter(ArtMarCod=marca)
```

```
125     templates = {
126         'Acer': 'marca_acer.html',
127         'Apple': 'marca_apple.html',
128         'Asus': 'marca_asus.html',
129         'Dell': 'marca_dell.html',
130         'HP': 'marca_hp.html',
131         'Lenovo': 'marca_lenovo.html',
132         'MSI': 'marca_msi.html',
133     }
134     templates = templates.get(marca.MarNom, 'marca_default.html')
135     context = {'marca': marca, 'arts': arts}
136     return render(request, f'{templates}', context)
137
138 #Eliminar pedido
139 @login_required
140 def eliminar_pedido(request, pedidoID):
141     pedido_cabecera = get_object_or_404(Pedido, pk=pedidoID, PedCabCodCli=request.user)
142     detalles_pedido = Pedido.objects.filter(pk=pedidoID)
143
144     for detalle in detalles_pedido:
145         articulo = detalle.PedDetArtCod
146         articulo.ArtSto += detalle.PedDetCantidad
147         articulo.save()
148
149     pedido_cabecera.delete()
150
151     return redirect('index')
152
153 #Carrito de compras
154 @login_required
155 def agregar_al_carrito(request, articulo_id):
156     articulo = get_object_or_404(Articulo, pk=articulo_id)
157     carrito, created = Carrito.objects.get_or_create(usuario=request.user, articulo=articulo)
158     if not created:
159         carrito.cantidad += 1
160         carrito.save()
161     return redirect('carrito')
162
163 @login_required
164 def eliminar_del_carrito(request, carrito_id):
165     carrito = get_object_or_404(Carrito, pk=carrito_id, usuario=request.user)
166     carrito.delete()
167     return redirect('carrito')
168
169 @login_required
170 def carrito(request):
171     carrito = Carrito.objects.filter(usuario=request.user)
172     total = sum(item.subtotal() for item in carrito)
173     return render(request, 'carrito.html', {'carrito': carrito, 'total': total})
174
175 def actualizar_cantidad(request, item_pk, cantidad):
176     carrito_item = get_object_or_404(Carrito, pk=item_pk)
177     carrito_item.cantidad = cantidad
178     carrito_item.save()
179     return JsonResponse({'status': 'ok'})
180
```

```
181 def cancelar_carrito(request):
182     carrito = get_object_or_404(Carrito, usuario=request.user)
183     request.session['carrito'] = []
184     request.session.modified = True
185     carrito.delete()
186     return redirect('carrito')
187
188 @login_required
189 def guardar_pedido(request):
190     carrito = Carrito.objects.filter(usuario=request.user)
191     if carrito.exists():
192         pedido_cabecera = Pedido(PedCabCodCli=request.user)
193         # pedido_cabecera.save()
194         for item in carrito:
195             articulo = item.articulo
196             cantidad = item.cantidad
197             if cantidad > articulo.ArtSto:
198                 messages.error(request, f'La cantidad de "{articulo.ArtNom}" en el carrito
199                     supera el stock disponible.')
200                 return redirect('carrito')
201             subtotal = cantidad * articulo.ArtPreUni
202             pedido_cabecera.PedDetArtCod = articulo
203             pedido_cabecera.PedDetCantidad = cantidad
204             pedido_cabecera.PedDetPreUniArt = articulo.ArtPreUni
205             pedido_cabecera.PedDetSubtotal = subtotal
206             pedido_cabecera.PedDetTot += subtotal
207             articulo.ArtSto -= cantidad
208             articulo.save()
209             pedido_cabecera.save()
210             item.pedido = pedido_cabecera
211             item.save()
212             carrito.delete()
213             # request.session['carrito'] = {}
214             return redirect('pedido_detalle', pedidoID=pedido_cabecera.pk)
215
216     else:
217         messages.warning(request, 'No hay articulos en el carrito.')
218         return redirect('index')
219
220 #Pedido detalle
221 def pedido_detalle(request, pedidoID):
222     pedido = get_object_or_404(Pedido, pk=pedidoID)
223     detalles = Pedido.objects.filter(pk=pedidoID, PedCabCodCli=request.user,
224         PedCabFec=pedido.PedCabFec)
225     return render(request, 'pedido_detalle.html', {'pedido': pedido, 'detalles': detalles})
226
227 #Productos
228 def productos(request):
229     productos = Articulo.objects.all()
230     return render(request, 'productos.html', {'productos': productos})
231
232 #Token
233 @api_view(['POST'])
234 def generar_token(request):
235     username = request.data.get('username')
236     password = request.data.get('password')
```



```

235 user = authenticate(username=username, password=password)
236 if user is not None:
237     token = user.generate_auth_token()
238     return Response({'token': token})
239 else:
240     return Response({'error': 'Credenciales invlidas'})

```

■ Links de rest:

- <http://127.0.0.1:8000/api/vendedor/>
- <http://127.0.0.1:8000/api/cliente/>
- <http://127.0.0.1:8000/api/pedido/>
- <http://127.0.0.1:8000/api/marca/>
- <http://127.0.0.1:8000/api/tipoArt/>
- <http://127.0.0.1:8000/api/carrito/>
- <http://127.0.0.1:8000/api/articulo/>

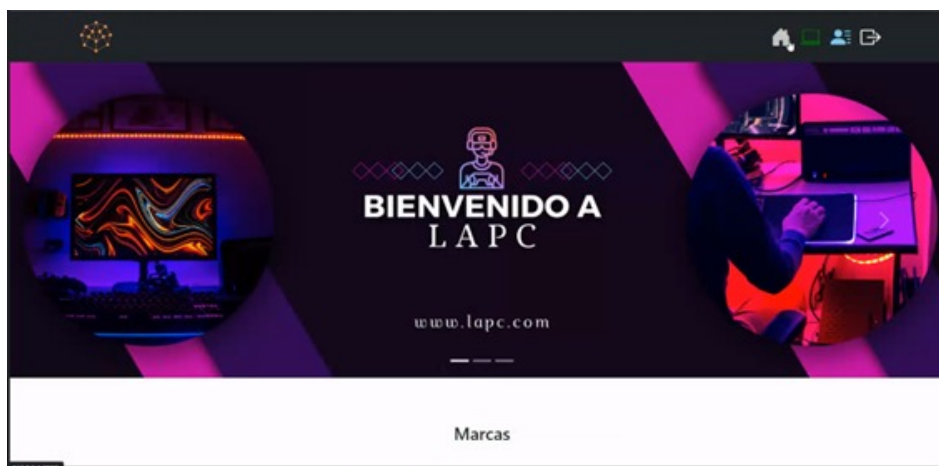


Figura 1: Página principal.

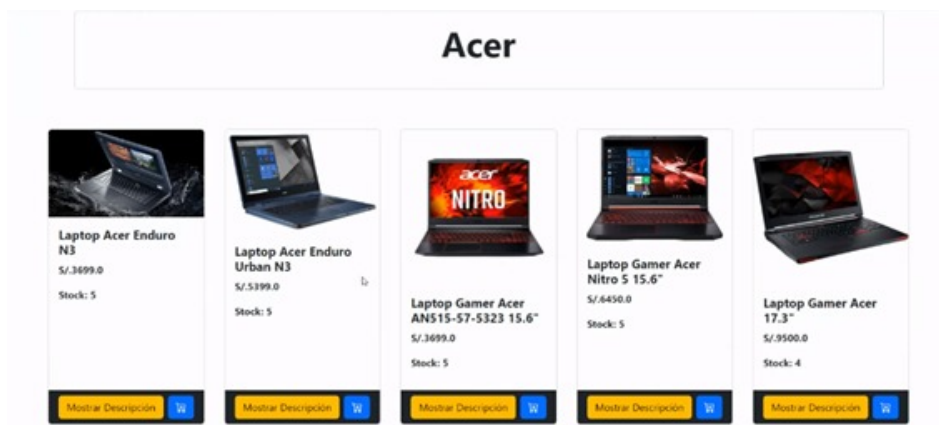


Figura 2: Productos con carritos de compra como opción.

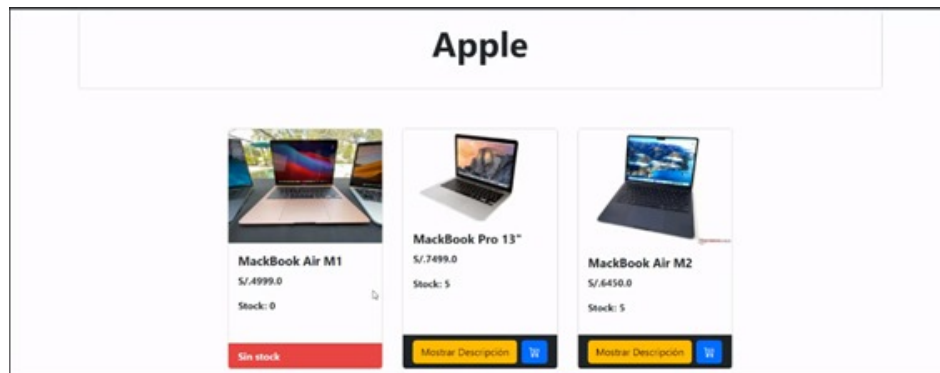


Figura 3: Posible caso de stock de un producto.



Figura 4: Datos de un pedido.

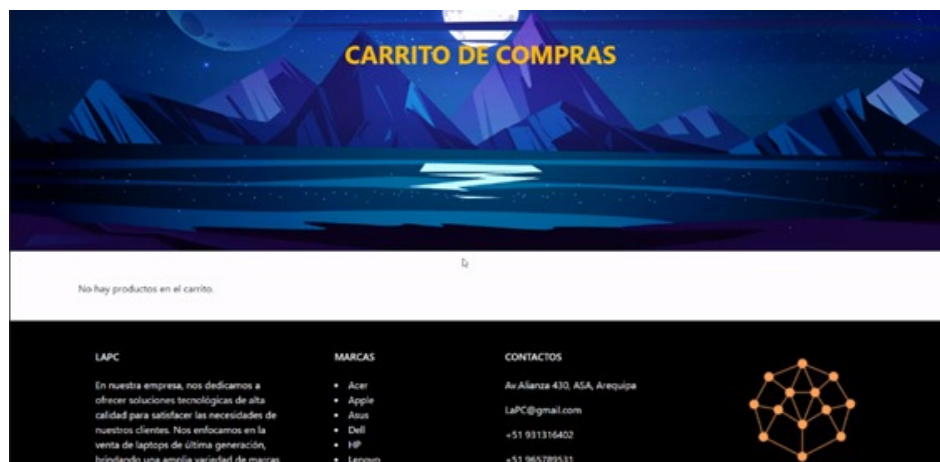
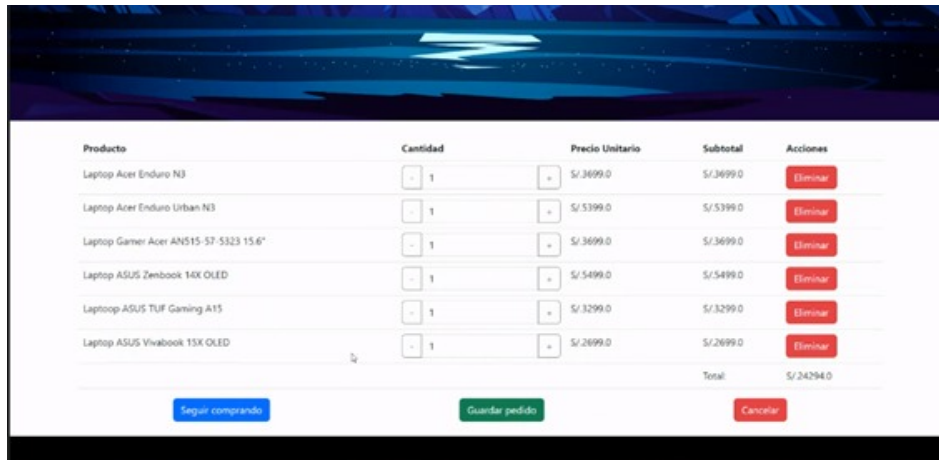


Figura 5: Carrito de compras.



Producto	Cantidad	Precio Unitario	Subtotal	Acciones
Laptop Acer Enduro N3	1	S/ 3699.0	S/ 3699.0	Eliminar
Laptop Acer Enduro Urban N3	1	S/ 5399.0	S/ 5399.0	Eliminar
Laptop Gamer Acer AN515-57-5323 15.6"	1	S/ 3699.0	S/ 3699.0	Eliminar
Laptop ASUS Zenbook 14X OLED	1	S/ 5499.0	S/ 5499.0	Eliminar
Laptop ASUS TUF Gaming A15	1	S/ 3299.0	S/ 3299.0	Eliminar
Laptop ASUS Vivabook 15X OLED	1	S/ 2699.0	S/ 2699.0	Eliminar
			Total:	S/ 24294.0

[Seguir comprando](#)
[Guardar pedido](#)
[Cancelar](#)

Figura 6: Ejemplo de artículos en el carrito de compras.

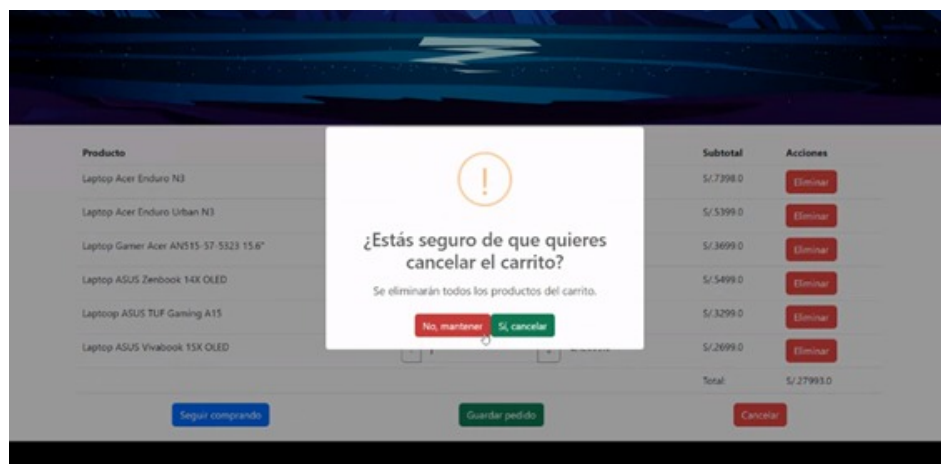


Figura 7: Ejemplo de poder quitar un artículo del carrito de compras.

DATOS DEL PEDIDO			
CLIENTE	Alfonso Mercaderes		
FECHA	July 20, 2023		
TOTAL	S/ 3699.0		
Artículo	Cantidad	Precio unitario	Subtotal
Laptop Acer Enduro N3	1	S/ 3699.0	S/ 3699.0

LAPC En nuestra empresa, nos dedicamos a ofrecer soluciones tecnológicas de alta calidad para satisfacer las necesidades de nuestros clientes. Nos enfocamos en la venta de laptops de última generación, brindando una amplia variedad de marcas y modelos para que nuestros clientes puedan encontrar la opción que mejor se adapte a sus necesidades.	MARCAS <ul style="list-style-type: none"> Acer Apple Asus Dell HP Lenovo MSI 	CONTACTOS Av. Alianza 430, ASA, Arequipa LaPC@gmail.com +51 931316402 +51 965789531	
--	---	--	--

Figura 8: Datos de pedido.

Django REST framework	
Vendedor List	
Vendedor List	OPTIONS GET
GET /api/vendedor/	
HTTP 200 OK Allow: GET, POST, HEAD, OPTIONS Content-Type: application/json Vary: Accept	
[
{"id": 1,	
"vendor": "80122605",	
"vendorPat": "idpat",	
"vendor": "Adriel",	
"vendorTag": true	
},	
{"id": 2,	
"vendor": "80122609",	
"vendorPat": "Narcis",	
"vendor": "Richard",	
"vendorTag": true	
}	
]	

Figura 9: Vendedor List.

Django REST framework	
Cliente List	
Cliente List	GET
GET /api/cliente/	
HTTP 401 Unauthorized Allow: GET, POST, HEAD, OPTIONS Content-Type: application/json Vary: Accept WWW-Authenticate: Token	
{	
"detail": "Authentication credentials were not provided."	
}	

Figura 10: Cliente List.

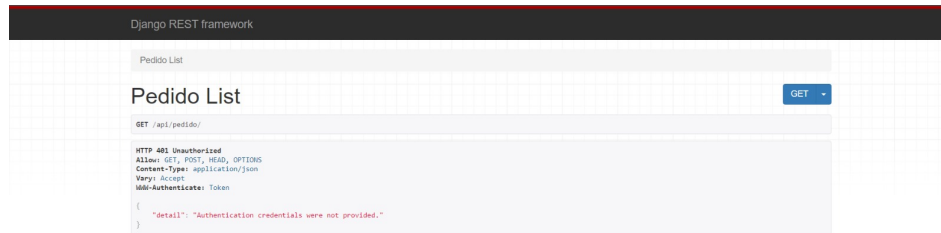


Figura 11: Pedido List.

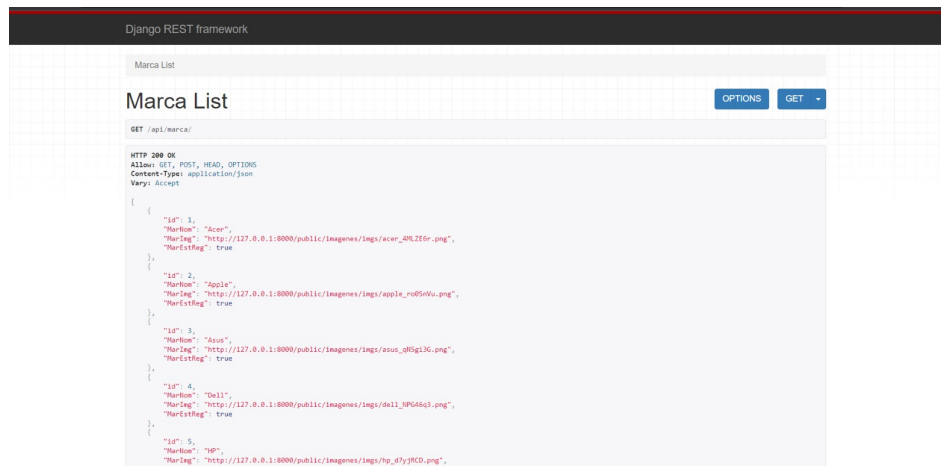


Figura 12: Marca List.



Figura 13: Marca Instance.

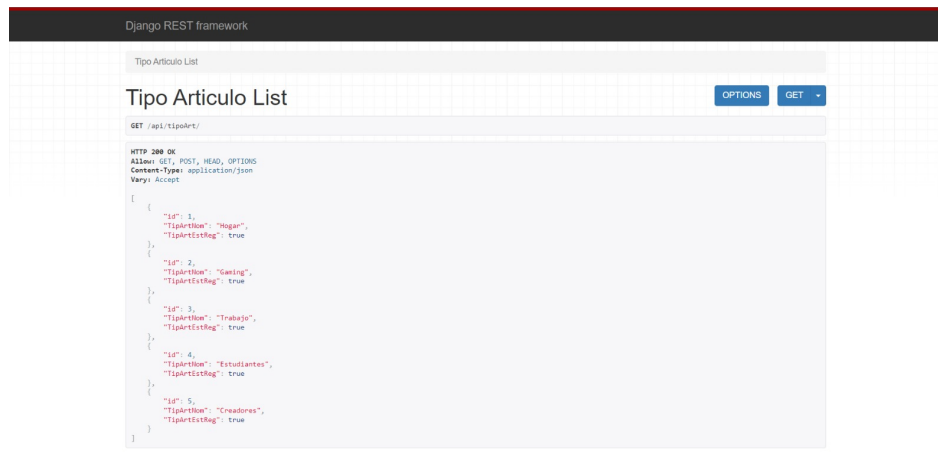


Figura 14: Tipo Artículo List.

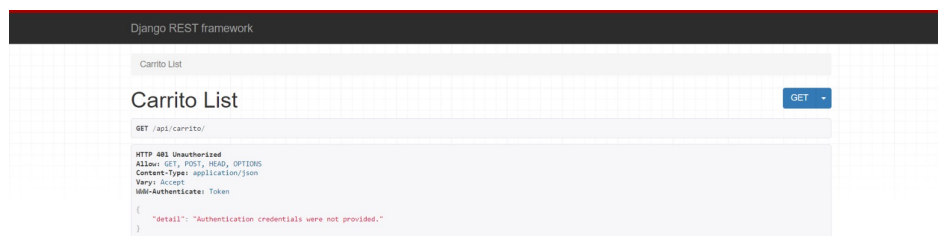


Figura 15: Carrito List.

- Accediento con el token del administrador y usando la extensión de VSC, Thunder Client y GET, vemos la lista de los vendedores actuales.

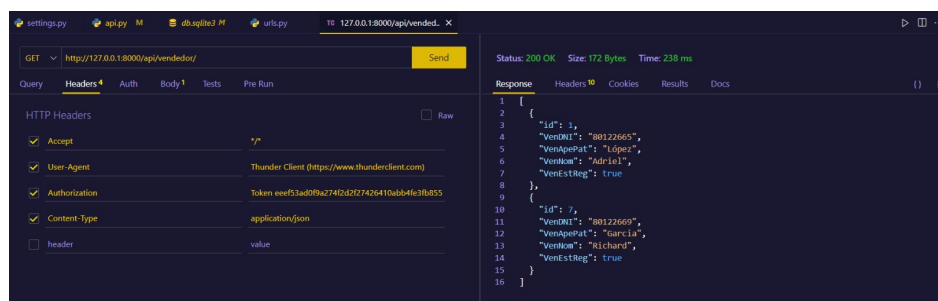


Figura 16: Vendedores.

- Usamos patch para actualizar nombre y ahora se muestra como el vendedor Ronald Garcia está en la lista de Vendedores, tomando el lugar del anterior vendedor Richard.

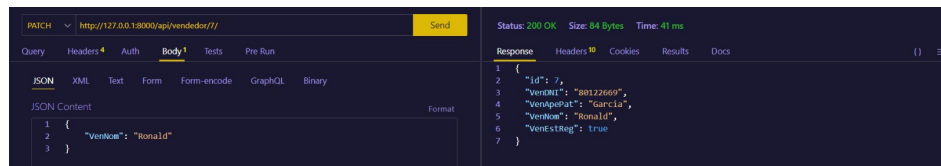


Figura 17: Comprobación de vendedores.

- Ahora eliminaremos al vendedor Ronald usando DELETE.

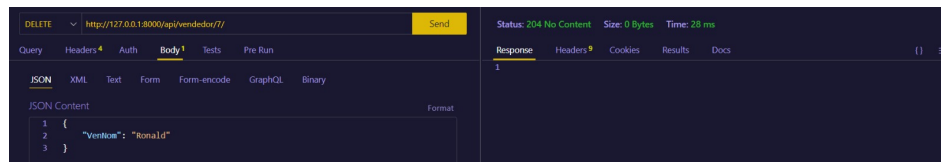


Figura 18: Eliminación de vendedor.

- Ahora creamos una nueva vendedora llamada Maricielo Gamarra usando POST y confirmamos que Maricielo está en la lista de vendedores.

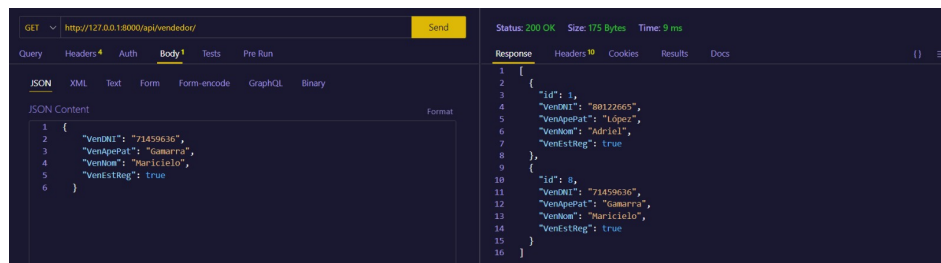


Figura 19: Comprobación de creación de vendedora.

- Ahora veremos los pedidos usando la extensión de Visual Studio Code, Thunder Client.
- Se muestra nuestra autorización con el token de admin.

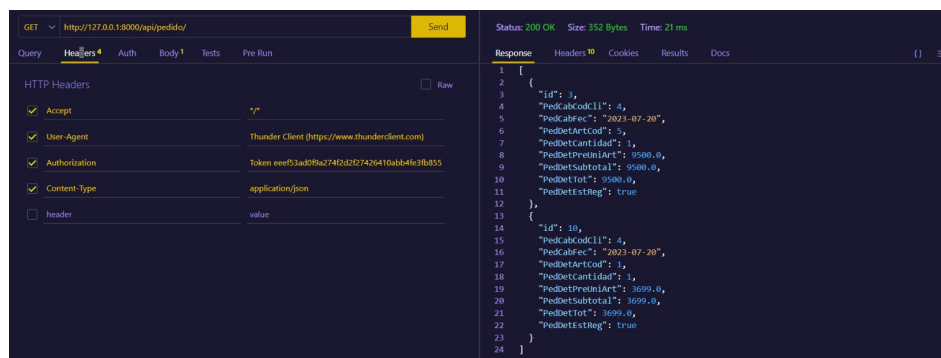


Figura 20: Token admin.

- Se muestra nuestra autorización con el token de Alfonso_M.

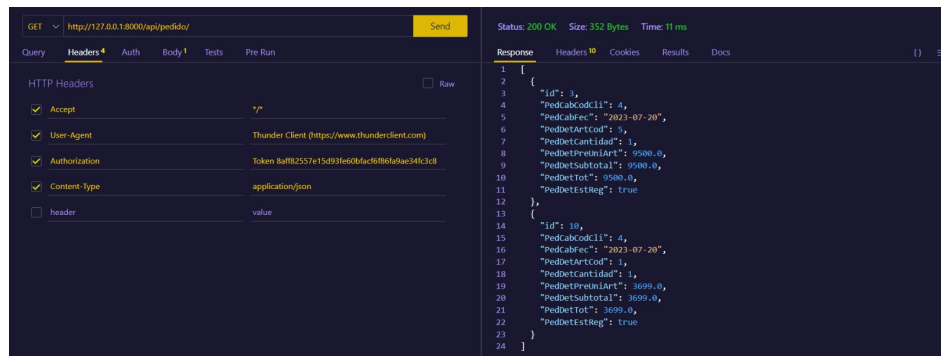


Figura 21: Token Alfonso_M.

- Se muestra cómo es que estamos creando un pedido nuevo.

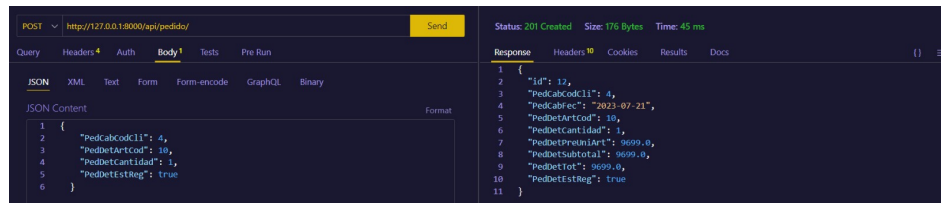


Figura 22: Creación de pedido.

- Se muestra que el pedido ha sido creado.

DATOS DEL PEDIDO			
CLIENTE	Alfonso Mercaderes		
FECHA	July 21, 2023		
TOTAL	S/. 9699.0		
Artículo	Cantidad	Precio unitario	Subtotal
Laptop ASUS Zenbook Pro 16X OLED	1	S/. 9699.0	S/. 9699.0

Figura 23: Pedido.

- A continuación se muestra como obtenemos todos los pedidos usando GET.

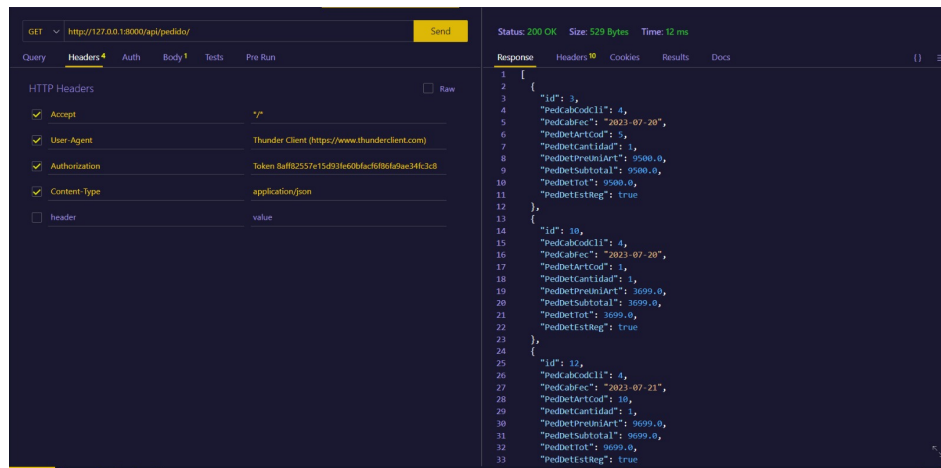


Figura 24: Obtención de pedidos.

- Se muestra cómo se elimina el pedido.

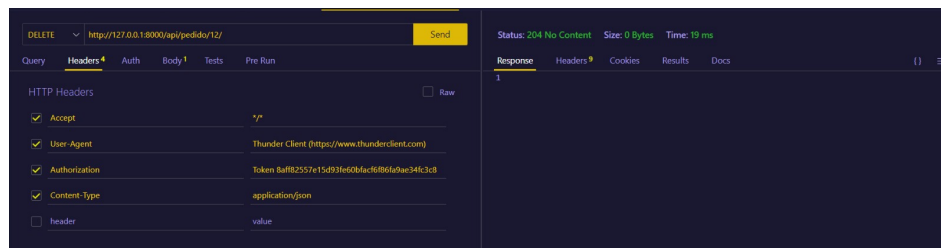


Figura 25: Eliminación de un pedido.

- Ahora comprobamos el estado de nuestro carrito de compras.

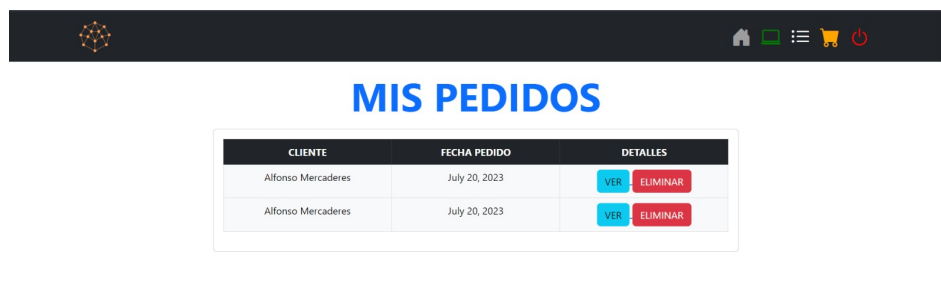


Figura 26: Verificación.

- En la siguiente imagen se muestra la validación y comprobación.

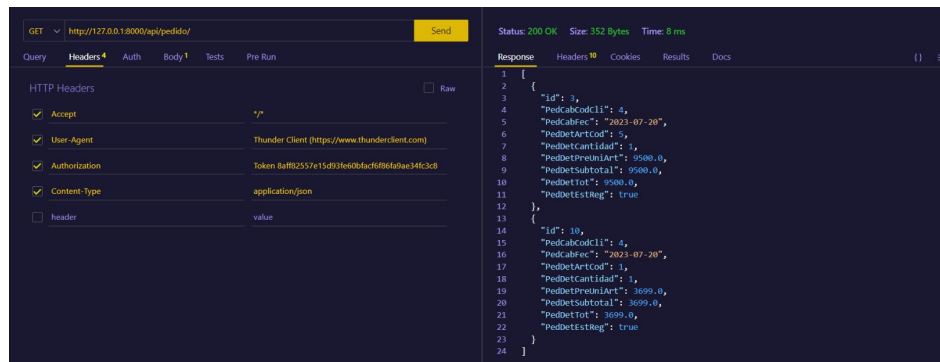








Figura 27: Validación y comprobación.









9. Commits





- Funcionalidad carrito de compras funcional, se realizó la comprobación de la eficacia del carrito de compras.
- Redireccionamiento signin, singup, esto cuando no se halla creado una cuenta previamente.
- Función de Token para cada usuario, se hizo para que cada usuarioi logre ver sus pedidos.
- Los commits son los siguientes:

Commits on Jul 20, 2023	
Estructura 123ihuayhua committed 7 hours ago	b221699
Estructura 123ihuayhua committed 7 hours ago	7d398c1
Reorganizacion 123ihuayhua committed 7 hours ago	57711a6
Fin Proyecto 123ihuayhua committed 7 hours ago	abbe82f
Version 1 Proyecto 123ihuayhua committed 9 hours ago	18e6963
Funcion token para cada usuario 123ihuayhua committed 20 hours ago	bb9ba96
Agregando dependencia autenticacion 123ihuayhua committed 20 hours ago	1379c48












urls include 123ihuayhua committed yesterday	 71ee378 <>
Urls viewsets 123ihuayhua committed yesterday	 a0497ac <>
viewsets 123ihuayhua committed yesterday	 4caa2f1 <>
Clases serializers - models 123ihuayhua committed yesterday	 d992a8f <>
Creación archivo serializers 123ihuayhua committed yesterday	 ab7e3cc <>
Funcionalidad carrito compras funcional 123ihuayhua committed 2 days ago	 2dd8bdd <>

Commits on Jul 17, 2023

Verificacion si desea eliminar producto 123ihuayhua committed 3 days ago	 fc3082c <>
Estilos bootstrap y contador 123ihuayhua committed 4 days ago	 af6e3a4 <>
Agregar notificaciones 123ihuayhua committed 4 days ago	 e87bc39 <>
Redireccionamiento signín a signúp 123ihuayhua committed 4 days ago	 ab2e86a <>
Validación de que usuario esta logeado para agregar productos 123ihuayhua committed 4 days ago	 e9129f2 <>
Agregar funcionalidad - carrito desde el producto 123ihuayhua committed 4 days ago	 ab88ac8 <>
Nueva vista, ver carrito de compras 123ihuayhua committed 4 days ago	 9d11ccc <>
Agregando enlaces 123ihuayhua committed 4 days ago	 57c7e86 <>

Vistas.py agregando carrito 123ihuayhua committed 4 days ago	 22de4ba <>
Mejorando modelo carrito 123ihuayhua committed 4 days ago	 9a9def <>
Carrito de Compras 123ihuayhua committed 4 days ago	 fd85b57 <>
Copia Lab06 123ihuayhua committed 4 days ago	 81cbd3e <>

Commits on Jun 30, 2023

Leeme.txt 123ihuayhua committed 3 weeks ago	 25ce082 <>
Fin 123ihuayhua committed 3 weeks ago	 5d37e24 <>
Informe 3 123ihuayhua committed 3 weeks ago	 8a3520f <>
Informe 2 123ihuayhua committed 3 weeks ago	 976593f <>
Informe 1-D 123ihuayhua committed 3 weeks ago	 b9a36ec <>
gitignore 123ihuayhua committed 3 weeks ago	 f938a8b <>
gitignore 123ihuayhua committed 3 weeks ago	 b6b3d93 <>
Fusion 123ihuayhua committed 3 weeks ago	 0fdcce1 <>
Merge branch 'temp-branch'  123ihuayhua committed 3 weeks ago	 a807bd8 <>
Informe 123ihuayhua committed 3 weeks ago	 a4e5e6f <>

10. Rúbricas

10.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

10.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

	Contenido y demostración	Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	1	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	1	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		17	

11. Referencias

- https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Tutorial_local_library_website
- <https://github.com/rescobedoq/pw2/tree/main/labs/lab05>
William S. Vincent. (2022). Django for Beginners: Build websites with Python. Django 4.0. leanpub.com. [URL]
<https://docs.djangoproject.com/en/4.1/ref/models/fields/>
https://docs.djangoproject.com/en/4.0/topics/db/examples/many_to_many/
https://docs.djangoproject.com/en/4.0/topics/db/examples/many_to_one/
<https://blog.hackajob.co/djangos-new-database-constraints/>
<https://stackoverflow.com/questions/3330435/is-there-an-sqlite-equivalent-to-mysqls-describe-table>
<https://docs.djangoproject.com/en/4.1/ref/validators/#how-validators-are-run>
<https://docs.djangoproject.com/en/4.1/ref/models/instances/>
<https://www.youtube.com/watch?v=rHux0gMZ3Eg>
<https://www.youtube.com/watch?v=OTmQQjsl0eg>
<https://tex.stackexchange.com/questions/34580/escape-character-in-latex>
<https://www.sqlitetutorial.net/sqlite-show-tables/>
<https://www.wplogout.com/export-database-diagrams-erd-from-django/>