**Resumen del Backend "MesaFácil" para el Desarrollador Frontend**

Aquí tienes un resumen de las capacidades actuales del backend, consideraciones importantes y un listado de los endpoints de la API:

**1. Funcionalidades Principales del Backend:**

El backend de "MesaFácil" actualmente soporta:

* **Gestión de Usuarios:**
  + Creación de superusuarios y usuarios con estatus de "staff" (personal) a través del panel de Django o la línea de comandos.
  + Autenticación de usuarios para el acceso a la API.
* **Gestión de Productos:**
  + CRUD completo (Crear, Leer, Actualizar, Eliminar) para productos, incluyendo nombre, descripción, precio, imagen (opcional) y estado de disponibilidad.
  + Filtrado por disponibilidad, nombre, descripción y rango de precios.
  + Búsqueda de texto libre en nombre y descripción.
* **Gestión de Promociones:**
  + CRUD completo para promociones, incluyendo título, descripción, fecha de inicio y fin, y estado activo.
* **Gestión de Mesas:**
  + CRUD para mesas, incluyendo número de mesa y estado (disponible, ocupada, etc.).
  + **Generación de Códigos Dinámicos:** Una acción especial (POST /api/v1/tables/{id}/generate-code/) para que el personal del restaurante genere un código único de 4 dígitos para una mesa específica. Este código se guarda en la mesa junto con la fecha de actualización.
  + **Validación de Códigos por Cliente:** Una acción abierta (POST /api/v1/tables/validate-code/) que permite a un cliente enviar un código de 4 dígitos. Si el código es válido y la mesa está disponible:
    - La mesa se marca como "ocupada".
    - El código se anula (se borra de current\_code).
    - Se devuelven los datos de la mesa.
* **Gestión de Pedidos:**
  + **Creación de Pedidos Detallados:** Un endpoint (POST /api/v1/orders/) que permite crear un pedido completo, especificando la mesa y una lista de productos con sus cantidades.
    - El sistema calcula automáticamente el subtotal de cada línea de producto y el total general del pedido.
    - El pedido se crea inicialmente con el estado 'pending'.
  + **Confirmación de Pedidos:** Una acción especial (POST /api/v1/orders/{id}/confirm/) para que el personal del restaurante cambie el estado de un pedido de 'pending' a 'confirmed'.
* **Autenticación para el Frontend (JWT):**
  + Sistema de autenticación basado en JSON Web Tokens (JWT).
  + Endpoints para obtener tokens (/api/token/) y refrescarlos (/api/token/refresh/).
* **Seguridad:**
  + Permisos definidos para cada endpoint, distinguiendo entre acciones que requieren ser administrador (IsAdminUser), solo estar autenticado (IsAuthenticated), o que cualquiera puede leer pero solo autenticados pueden modificar (IsAuthenticatedOrReadOnly). La validación de códigos de mesa es abierta (AllowAny).
* **Documentación Automática de la API:**
  + Disponible a través de Swagger UI (/api/schema/swagger-ui/) y ReDoc (/api/schema/redoc/). Esta es la **fuente de verdad** para todos los detalles de los endpoints.

**2. Consideraciones Clave para el Frontend:**

* **Autenticación JWT:**
  + El frontend necesitará implementar un flujo de inicio de sesión que llame a POST /api/token/ con username y password.
  + Debe almacenar de forma segura los tokens access y refresh recibidos (normalmente en localStorage o sessionStorage).
  + Para cada petición a endpoints protegidos, debe enviar el token de access en la cabecera Authorization así: Authorization: Bearer <token\_de\_acceso>.
  + Debe implementar la lógica para usar el refresh\_token en POST /api/token/refresh/ para obtener un nuevo access\_token cuando el actual expire (el backend devolverá un error 401 si el token de acceso está expirado o es inválido).
* **Permisos y Roles:**
  + Consultar la documentación de la API para entender qué endpoints requieren autenticación y qué acciones pueden estar restringidas a roles específicos (ej. el personal para generar códigos de mesa o confirmar pedidos).
* **Flujo del Cliente:**
  + El cliente (probablemente anónimo en este punto) escanea un QR general que lo lleva a la sección de productos.
  + Cuando va a realizar un pedido, la app le pide ingresar el código de mesa que ve en una TV.
  + El frontend envía este código a POST /api/v1/tables/validate-code/ con el cuerpo: {"code": "XXXX"}.
  + Si la validación es exitosa (respuesta 200 OK), el backend devuelve los datos de la mesa (incluyendo su id). El frontend debe guardar este id de mesa. La mesa ahora está "ocupada" y el código usado ya no es válido.
  + El cliente selecciona productos. Para crear el pedido, el frontend envía a POST /api/v1/orders/ una petición con el cuerpo:

JSON

{

"table": ID\_DE\_LA\_MESA\_OBTENIDA\_EN\_PASO\_4,

"details\_write": [

{ "product\_id": ID\_PRODUCTO\_1, "quantity": CANTIDAD\_1 },

{ "product\_id": ID\_PRODUCTO\_2, "quantity": CANTIDAD\_2 }

]

}

* + - **Nota Importante sobre el Cliente que Hace el Pedido:** La creación de pedidos (POST /api/v1/orders/) actualmente requiere autenticación (IsAuthenticated). Esto significa que el cliente que hace el pedido necesitaría tener una cuenta y haber iniciado sesión (obtenido un token JWT). Si el diseño es que clientes completamente anónimos puedan hacer pedidos después de validar una mesa, necesitaríamos ajustar los permisos de este endpoint o replantear el flujo de sesión del cliente. Es un punto a discutir.
* **Flujo del Personal (Administrador/Mesero con Tablet):**
  + El personal inicia sesión en su aplicación (tablet) usando POST /api/token/ para obtener sus tokens JWT.
  + Para generar un código para una mesa: POST /api/v1/tables/{table\_id}/generate-code/. No necesita cuerpo la petición.
  + Para confirmar un pedido pendiente: POST /api/v1/orders/{order\_id}/confirm/. No necesita cuerpo la petición (a menos que se decida añadir campos como customer\_name).
* **Manejo de Errores:**
  + La API devuelve códigos de estado HTTP estándar (200, 201, 400, 401, 403, 404, etc.).
  + Los mensajes de error vienen en formato JSON, usualmente en un campo como detail o error (ej. {"detail": "Authentication credentials were not provided."} o {"error": "Código de mesa inválido..."}). El frontend debe estar preparado para interpretar estos mensajes.
* **CORS (Cross-Origin Resource Sharing):**
  + El backend tiene django-cors-headers configurado con CORS\_ALLOW\_ALL\_ORIGINS = True por ahora (en core/settings.py). Para el desarrollo local (Angular corriendo en un puerto y Django en otro), esto está bien. Para producción, CORS\_ALLOW\_ALL\_ORIGINS debería ser False y se debería configurar CORS\_ALLOWED\_ORIGINS (o CORS\_ALLOW\_CREDENTIALS y CORS\_ORIGIN\_WHITELIST/CORS\_ALLOWED\_ORIGIN\_REGEXES) para permitir solo el dominio del frontend.

**3. Listado de Endpoints Principales de la API:**

(Todos bajo el prefijo base: http://127.0.0.1:8000/)

* **Autenticación JWT:**
  + POST /api/token/ (Para obtener tokens de acceso y refresco. Envía username y password.)
  + POST /api/token/refresh/ (Para obtener un nuevo token de acceso. Envía refresh token.)
* **API Principal (prefijo** /api/v1/**):**
  + users/ (CRUD - Solo Admin)
  + products/ (CRUD - Lectura para todos, Escritura para autenticados)
    - Filtros por URL: ?availability=true/false, ?name\_\_icontains=<texto>, ?description\_\_icontains=<texto>, ?price\_\_lte=<numero>, ?price\_\_gte=<numero>
    - Búsqueda por URL: ?search=<texto\_a\_buscar\_en\_nombre\_y\_descripcion>
  + promotions/ (CRUD - Lectura para todos, Escritura para autenticados)
  + tables/ (CRUD - Solo Autenticados)
    - tables/{id}/generate-code/ (POST - Acción del personal, requiere autenticación)
    - tables/validate-code/ (POST - Acción del cliente, abierta. Envía {"code": "XXXX"})
  + orders/ (CRUD - Solo Autenticados)
    - Para crear: POST con {"table": <id>, "details\_write": [{"product\_id": <id>, "quantity": <num>}, ...]}
    - orders/{id}/confirm/ (POST - Acción del personal, requiere autenticación)
  + orderdetails/ (CRUD - Solo Autenticados, aunque generalmente se gestionan a través de orders/)
* **Documentación Automática:**
  + Swagger UI: /api/schema/swagger-ui/
  + ReDoc: /api/schema/redoc/