FastFood.exe - Documentación del Proyecto

Versión: 1.0
Fecha: Diciembre 2025
Autor: Sistema de Gestión de Comida Rápida

Tabla de Contenidos

---

1. Resumen Ejecutivo

FastFood.exe es una aplicación web moderna de gestión de comida rápida que permite a los usuarios explorar productos, realizar pedidos y gestionar su cuenta, mientras que los administradores pueden gestionar productos, categorías y ventas.

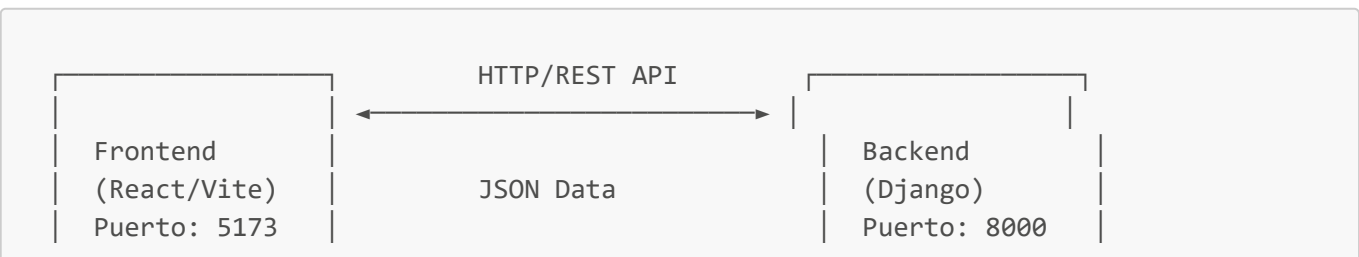
Características Clave

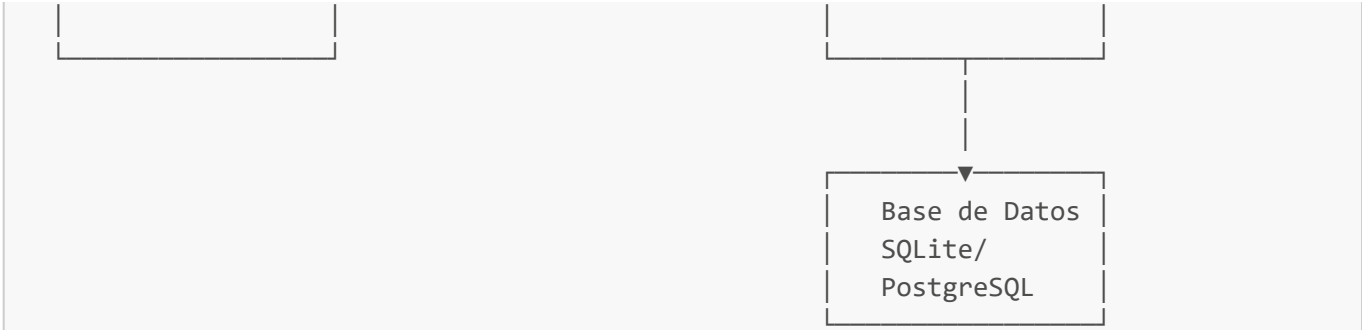
- ☑ Sistema de autenticación con 2FA (Doble Factor de Autenticación)
- ☑ Panel de administración completo
- ☑ Gestión de productos y categorías
- ☑ Sistema de carrito de compras
- ☑ Interfaz moderna y responsiva
- ☑ API RESTful con Django Rest Framework
- ☑ Frontend en React con Vite

---

2. Arquitectura del Sistema

Arquitectura General





Componentes Principales

Backend (Django)

- **Framework:** Django 5.2.7
- **API:** Django Rest Framework 3.16.1
- **Autenticación:** JWT con djangorestframework-simplejwt
- **Base de datos:** SQLite (desarrollo) / PostgreSQL (producción)

Frontend (React)

- **Framework:** React 19.1.1
- **Build Tool:** Vite 7.1.9
- **Routing:** React Router DOM 7.9.3
- **HTTP Client:** Axios 1.12.2
- **Styling:** TailwindCSS 4.1.16

---

3. Tecnologías Utilizadas

Backend

Tecnología	Versión	Propósito
Django	5.2.7	Framework web principal
Django REST Framework	3.16.1	API REST
djangorestframework-simplejwt	5.5.1	Autenticación JWT
django-cors-headers	4.9.0	Manejo de CORS
django-filter	24.3	Filtrado de querysets
gunicorn	23.0.0	Servidor WSGI (producción)
whitenoise	6.11.0	Servir archivos estáticos
psycopg[binary]	3.2.13	Adaptador PostgreSQL
Pillow	12.0.0	Procesamiento de imágenes

Frontend

Tecnología	Versión	Propósito
React	19.1.1	Librería UI
Vite	7.1.9	Build tool y dev server
React Router DOM	7.9.3	Navegación
Axios	1.12.2	Cliente HTTP
TailwindCSS	4.1.16	Framework CSS

---

4. Instalación y Configuración

Prerrequisitos

- Python 3.10 o superior
- Node.js 18 o superior
- npm o yarn
- Git

Instalación del Backend

```
# 1. Clonar el repositorio
git clone <repository-url>
cd mi_proyecto

# 2. Crear entorno virtual
python -m venv venv

# 3. Activar entorno virtual
# Windows:
venv\\Scripts\\activate
# Linux/Mac:
source venv/bin/activate

# 4. Instalar dependencias
pip install -r requirements.txt

# 5. Configurar variables de entorno
# Crear archivo .env con:
# SECRET_KEY=tu-clave-secreta
# DEBUG=True
# DATABASE_URL=sqlite:///db.sqlite3

# 6. Ejecutar migraciones
python manage.py migrate

# 7. Crear superusuario
python manage.py createsuperuser
```

```
# 8. Cargar datos de prueba (opcional)
python create_categories.py
python create_products.py

# 9. Ejecutar servidor
python manage.py runserver
```

Instalación del Frontend

```
# 1. Navegar a la carpeta frontend
cd frontend

# 2. Instalar dependencias
npm install

# 3. Configurar variables de entorno
# Crear archivo .env con:
# VITE_API_URL=http://localhost:8000

# 4. Ejecutar servidor de desarrollo
npm run dev

# El frontend estará disponible en http://localhost:5173
```

---

5. Estructura del Proyecto

```
mi_proyecto/
├── administracion/          # App de administración
│   ├── models.py           # Modelos de Venta y DetalleVenta
│   ├── views.py            # ViewSets para admin
│   ├── serializers.py      # Serializers de ventas
│   └── admin.py            # Configuración admin de Django
│
├── productos/              # App de productos
│   ├── models.py           # Modelos Producto y Categoria
│   ├── views.py            # ViewSets de productos
│   ├── serializers.py      # Serializers de productos
│   └── urls.py             # Rutas de productos
│
├── usuarios/               # App de usuarios
│   ├── models.py           # CustomUser y CodigoVerificacion
│   ├── views.py            # Vistas de autenticación
│   ├── serializers.py      # Serializers de usuarios
│   └── urls.py             # Rutas de usuarios
│
├── mi_proyecto/            # Configuración principal
│   ├── settings.py         # Configuración Django
```

```
│   ├── urls.py             # URLs principales
│   └── wsgi.py             # Configuración WSGI
│
├── frontend/               # Aplicación React
│   ├── src/
│   │   ├── components/     # Componentes reutilizables
│   │   ├── pages/          # Páginas principales
│   │   ├── services/       # API services
│   │   ├── context/        # React Context
│   │   └── assets/         # Imágenes y recursos
│   │
│   ├── public/             # Archivos públicos
│   ├── package.json        # Dependencias npm
│   └── vite.config.js      # Configuración Vite
│
├── requirements.txt        # Dependencias Python
├── manage.py              # CLI Django
└── db.sqlite3             # Base de datos SQLite
```

---

6. Características Principales

6.1 Sistema de Autenticación

Autenticación con 2FA

El sistema implementa autenticación de doble factor:

1. **Login con credenciales**: Usuario y contraseña
2. **Verificación 2FA**: Código de 6 dígitos enviado al email
3. **Generación de tokens JWT**: Access token y refresh token

Flujo de autenticación:

```
Usuario → Login → 2FA Code → Verify → JWT Token → Acceso
```

Tipos de usuarios

- **Cliente**: Puede ver productos, agregar al carrito, realizar compras
- **Administrador**: Acceso completo al panel de administración

6.2 Gestión de Productos

Características

- CRUD completo de productos
- Gestión de categorías
- Filtrado por categoría, disponibilidad

- Búsqueda por nombre y descripción
- Paginación (12 productos por página)
- Imágenes de productos

Campos de Producto

Campo	Tipo	Descripción
nombre	String	Nombre del producto
descripcion	Text	Descripción detallada
precio	Decimal	Precio en moneda local
categoria	ForeignKey	Categoría del producto
imagen	URL	URL de la imagen
disponible	Boolean	Si está disponible
stock	Integer	Cantidad en inventario
calificacion	Float	Calificación de 0-5

6.3 Panel de Administración

Funcionalidades

- Dashboard con estadísticas
- Gestión completa de productos
- Filtrado y búsqueda avanzada
- Paginación
- Creación/Edición/Eliminación de productos
- Visualización de categorías

Características de UI

- Diseño moderno con gradientes
- Animaciones suaves
- Responsive design
- Skeleton loaders para mejor UX
- Modales de confirmación
- Manejo de errores

6.4 Carrito de Compras

- Agregar/Eliminar productos
- Actualizar cantidades
- Persistencia local (localStorage)
- Cálculo automático de totales
- Validación de stock

6.5 API RESTful

Endpoints Principales

Autenticación:

- `POST /api/usuarios/register/` - Registro de usuario
- `POST /api/usuarios/login/` - Login con 2FA
- `POST /api/usuarios/verify-code/` - Verificar código 2FA
- `POST /api/usuarios/admin-login/` - Login de administrador
- `POST /api/token/refresh/` - Refrescar token

Productos (Públicos):

- `GET /api/productos/` - Listar productos
- `GET /api/productos/{id}/` - Detalle de producto
- `GET /api/categorias/` - Listar categorías

Admin (Requiere autenticación de staff):

- `GET /api/admin/productos/` - Listar productos (admin)
- `POST /api/admin/productos/` - Crear producto
- `PUT /api/admin/productos/{id}/` - Actualizar producto
- `DELETE /api/admin/productos/{id}/` - Eliminar producto
- `GET /api/admin/ventas/` - Listar ventas

---

7. API Documentation

Autenticación con JWT

Todas las peticiones protegidas requieren un token JWT en el header:

```
Authorization: Bearer <access_token>
```

Ejemplos de Uso

Registro de Usuario

```
POST /api/usuarios/register/
Content-Type: application/json

{
  "username": "usuario123",
  "email": "usuario@example.com",
  "password": "contraseña_segura"
}
```

Respuesta:

```
{
  "message": "Usuario creado exitosamente",
  "access": "eyJhbGc...",
  "refresh": "eyJhbGc...",
  "username": "usuario123",
  "rol": "cliente"
}
```

Login con 2FA

```
POST /api/usuarios/login/
Content-Type: application/json

{
  "username": "usuario123",
  "password": "contraseña_segura"
}
```

Respuesta:

```
{
  "requires_2fa": true,
  "session_id": "uuid-aqui",
  "message": "Código de verificación enviado",
  "email": "usuario@example.com",
  "debug_code": "123456"
}
```

Verificar Código 2FA

```
POST /api/usuarios/verify-code/
Content-Type: application/json

{
  "session_id": "uuid-aqui",
  "codigo": "123456"
}
```

Respuesta:

```
{
  "access": "eyJhbGc...",
  "refresh": "eyJhbGc...",
  "username": "usuario123",
  "rol": "cliente",
  "message": "Verificación exitosa"
}
```

Listar Productos (Público)

```
GET /api/productos/?page=1&categoria=1
```

Respuesta:

```
{
  "count": 35,
  "next": "http://localhost:8000/api/productos/?page=2",
  "previous": null,
  "results": [
    {
      "id": 1,
      "nombre": "Hamburguesa Clásica",
      "descripcion": "Deliciosa hamburguesa con...",
      "precio": "8500.00",
      "categoria": 1,
      "categoria_nombre": "Hamburguesas",
      "imagen": "https://...",
      "disponible": true,
      "stock": 50,
      "calificacion": 4.5
    }
  ]
}
```

Crear Producto (Admin)

```
POST /api/admin/productos/
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "nombre": "Nueva Pizza",
  "descripcion": "Pizza deliciosa",
  "precio": 12500,
  "categoria": 2,
```

```
  "imagen": "https://ejemplo.com/imagen.jpg",
  "disponible": true,
  "stock": 20,
  "calificacion": 0
}
```

---

8. Seguridad

Medidas de Seguridad Implementadas

1. **Autenticación JWT**
   - Tokens con expiración
   - Refresh tokens para renovación
   - Almacenamiento seguro en localStorage
2. **Autenticación de Doble Factor (2FA)**
   - Códigos de 6 dígitos
   - Expiración de 10 minutos
   - Códigos de un solo uso
3. **CORS**
   - Configurado con django-cors-headers
   - Permite requests desde el frontend
4. **Validación de Datos**
   - Serializers de DRF para validación
   - Validación de campos en backend
   - Sanitización de inputs
5. **Permisos**
   - `IsAdminUser` para endpoints de administración
   - `IsAuthenticatedOrReadOnly` para productos públicos
   - Validación de roles

Recomendaciones de Seguridad

Para producción, asegúrate de:

- ☑ Usar HTTPS
- ☑ Configurar `DEBUG=False`
- ☑ Usar base de datos PostgreSQL
- ☑ Configurar SECRET_KEY segura
- ☑ Implementar rate limiting
- ☑ Configurar ALLOWED_HOSTS correctamente
- ☑ Usar email real para 2FA (no consola)

9. Guía de Usuario

Para Usuarios Finales

Registro y Login

1. Ir a `/register`
2. Completar formulario de registro
3. El sistema crea la cuenta automáticamente
4. Para login posterior:
    - Ir a `/login`
    - Ingresar usuario y contraseña
    - Ingresar código 2FA mostrado en pantalla
    - Acceder al sistema

Explorar Productos

1. La página principal (`/`) muestra todos los productos
2. Usar filtros de categoría
3. Usar búsqueda para encontrar productos específicos
4. Click en un producto para ver detalles

Realizar Compra

1. Agregar productos al carrito
2. Ajustar cantidades si necesario
3. Ver resumen en el carrito
4. Proceder al checkout

Para Administradores

Acceso al Panel

1. Ir a `/admin/login`
2. Ingresar credenciales de administrador
3. Acceder al dashboard

Gestionar Productos

Crear Producto:

1. Click en "Nuevo Producto"
2. Completar formulario
3. Guardar

Editar Producto:

1. Click en "Editar" en el producto deseado

2. Modificar campos
3. Guardar cambios

Eliminar Producto:

1. Click en "Eliminar"
2. Confirmar eliminación

Filtros y Búsqueda

- Usar barra de búsqueda para buscar por nombre
- Filtrar por categoría usando el selector
- Navegar entre páginas con los controles de paginación

---

10. Mantenimiento y Troubleshooting

Problemas Comunes

Error 401: No autorizado

Causa: Token inválido o expirado

Solución:

- Cerrar sesión y volver a iniciar
- Verificar que el token se está enviando correctamente
- Revisar configuración de CORS

Error "Sesión inválida" en 2FA

Causa: Problema con almacenamiento de session_id

Solución:

- Este problema fue corregido usando `session_id` en el modelo
- Asegurarse de que la migración `0004_codigoverificacion_session_id` está aplicada

Productos no cargan en Admin Dashboard

Causa: Token no se guarda en localStorage

Solución:

- Verificar que `AdminLogin.jsx` guarda los tokens
- Revisar consola del navegador para errores
- Cerrar sesión y volver a iniciar

CORS Errors

Causa: Configuración incorrecta de CORS

Solución:

```
# settings.py
CORS_ALLOW_ALL_ORIGINS = True
CORS_ALLOW_CREDENTIALS = True
```

Comandos Útiles

```
# Crear migraciones
python manage.py makemigrations

# Aplicar migraciones
python manage.py migrate

# Crear superusuario
python manage.py createsuperuser

# Verificar configuración
python manage.py check

# Recolectar archivos estáticos
python manage.py collectstatic

# Ejecutar shell de Django
python manage.py shell

# Ver productos en DB
python check_products.py

# Verificar categorías
python check_categories_api.py
```

Logs y Debugging

Los logs se imprimen en consola durante desarrollo. Para producción:

```
# settings.py
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
        'file': {
            'class': 'logging.FileHandler',
            'filename': 'debug.log',
        },
```

```
    },
    'root': {
        'handlers': ['console', 'file'],
        'level': 'INFO',
    },
}
```

Contacto y Soporte

Para preguntas o soporte técnico, contactar al equipo de desarrollo.

Versión del documento: 1.0
Última actualización: Diciembre 2025
