ChatGPT

Simple E-Commerce Django REST Application Guide

This document provides an overview of the **simple_ecommerce_djrest** repository. The project is a Django REST-framework back-end for a small e-commerce store. It includes authentication, product management, inventory control, order processing, multi-currency pricing, and double-entry accounting. The guide below describes the capabilities and suggests how to navigate the API.

Project structure

The Django project is organised into several apps:

Directory	Purpose
`api/`	Contains a sample API with JWT token endpoints and a **dummy** product viewset used for demonstration [1] .
`ecommerce/`	Core e-commerce application. It implements models for products, pricing, inventory, orders, payments, purchases and accounting, along with serializers and viewsets for each domain.
`config/`	Holds project settings and root `urls.py`. It defines installed apps, REST-framework configuration, SimpleJWT options and environment variables, and registers application routes [2] .
`docs/`	Markdown documents explaining design decisions, such as multi-currency pricing, product images, and accounting.

The `ecommerce` app contains submodules by domain: `product`, `purchase`, `order`, `inventory`, `users`, and `accounting`. Each submodule has models, serializers and viewsets tailored to that domain.

Authentication and user management

User registration and login

The application uses `django-rest-auth` with SimpleJWT for authentication. Users register via `ecommerce/auth/registration/`. Registration uses a custom serializer to create a user, customer record and address [3] . Email verification is required; the custom account adapter sends a confirmation email with a React-front-end link [4] .

Login uses a **custom login view** that returns both access and refresh tokens plus the user's profile. After successful login you receive JSON containing `access`, `refresh` and a user object containing id, username, email and role [3]. Tokens are generated using SimpleJWT; token endpoints are exposed under `/api/v1/api/token/` and `/api/v1/api/token/refresh/` [1].

Staff and customer roles

Users have roles defined in the `Role` model (e.g., `customer`, `staff`, `admin`). Permissions restrict access: non-staff users can read products and FX rates but cannot create or update them. Custom permissions `IsStaffOrReadOnly` and `IsStaff` enforce write privileges only for staff [5]. Staff management endpoints allow viewing and creating staff members, while customers can update their addresses and profiles via `/ecommerce/customers/` and `/ecommerce/addresses/`.

JWT token endpoints

To obtain a JWT token, call the token endpoint:

```
POST /api/v1/api/token/
Payload: { "username": "your_user", "password": "your_pass" }
```

You will receive `refresh` and `access` tokens. Use the access token in the `Authorization: Bearer <token>` header for authenticated requests. Refresh tokens can be exchanged for new access tokens via `/api/v1/api/token/refresh/` [1].

Product management

Models

The product domain models include:

- **Category**, **Brand** and **Tag** models for organising products. Each has a name and slug.
- **Product** model stores name, description, SKU, category, brand and tags, plus flags for `is_active` and timestamps [6]. Products are linked to many images and prices.
- **ProductPrice** stores a product's price in a particular currency and date range. The `begin_date` and optional `end_date` allow time-based pricing. A unique constraint ensures only one active price exists per product and currency at a time [6]. Prices are nested in the product serializer, so the API returns the active price automatically [7].
- **ProductImage** stores multiple images for each product. A `tag` field marks images as "icon", "main", etc. Icon images are used for thumbnails and returned in dedicated list endpoints [8].
- **Inventory** records the current quantity for each product and currency. Inventory is updated whenever purchases, sales or manual adjustments occur.

Viewsets & endpoints

Routes are registered under `/ecommerce/` (configured in `ecommerce/urls.py` and `config/urls.py`). Key endpoints include:

Endpoint	Method	Description
`/ecommerce/categories/`, `/brands/`, `/tags/`	GET/POST	List or create categories, brands or tags. Only staff can POST.
`/ecommerce/products/`	GET	List products with nested category, brand, tags, images, price and inventory [7]. Staff can filter by category or search by name.
`/ecommerce/products/<id>/`	GET/PUT/DELETE	Retrieve, update or delete a product. Update operations require staff privileges.
`/ecommerce/product/create/`	POST	Custom APIView for creating a product with price and inventory. Accepts JSON including product details, price (amount and currency), and inventory quantity [9]. Returns the created product with nested data.
`/ecommerce/product/update/<id>/`	PUT	Custom APIView to update a product's fields, price and inventory. It also maintains journal entries to reflect inventory changes [9].
`/ecommerce/product/import/`	POST	Accepts a CSV file containing product information. This endpoint reads the CSV, creates products, sets prices and inventory quantities, and returns a summary of created records [9].
`/ecommerce/products-with-image/`	GET	Lists products with only the icon image for quick display. Useful for a storefront or cart view [9].

The product serializer includes related objects like currency, FX rates and inventory. When retrieving products, you see human-friendly nested objects rather than foreign keys [7].

Multi-currency pricing

The application supports pricing in multiple currencies. The `Currency` model stores currency codes (e.g., USD, JPY) and a boolean flag for the accounting currency. `FXRate` stores exchange rates relative to the accounting currency, along with a date and `source` [10]. Product prices reference a currency, and orders can be placed using any currency.

A utility function `convert_price(amount, source_currency, target_currency, date)` converts amounts using the exchange rate effective on the order date [11]. When an order is created, the system calculates the total in the order currency and also converts it to the accounting currency for journal entries [12]. The project documentation recommends storing order totals in the base currency and converting at payment time [13].

Inventory and purchase management

Inventory is automatically managed through purchases and sales. When staff add inventory via a **Purchase**, the quantity increases and a journal entry records the inventory asset and accounts payable. Purchases include fields like vendor name, date, currency, and a list of items with quantity and price [14]. Endpoints include:

Endpoint	Method	Description
`/ecommerce/purchases/`	GET/POST	List purchases or create a purchase. Creating a purchase updates inventory and records journal entries [15].
`/ecommerce/purchase/<id>/`	GET/PUT/DELETE	Retrieve or modify a purchase. Updating a purchase will adjust inventory accordingly.
`/ecommerce/purchase/import/`	POST	Bulk import purchases from a CSV. This endpoint adds multiple purchases and returns a summary.
`/ecommerce/purchase/by-date/`	GET	Returns aggregated purchase totals grouped by date. You can filter by date range.
`/ecommerce/inventories/`	GET/POST/PUT/DELETE	Staff can list, create, adjust or delete inventory records. Manual adjustments trigger journal entries to balance inventory and adjustment accounts [16].

Order workflow and cart management

Customers can browse products and add items to a cart (client-side). To place an order, send a POST request to `/ecommerce/order/create/` with a list of items and quantities. The order creation view performs several operations:

1. **Calculate prices** – For each item, it fetches the active product price in the order currency, multiplies by quantity, and sums them up [12].
2. **Convert to accounting currency** – The total is converted using the FX rate for the order date [11].
3. **Create order and order items** – Records the order with status `pending` and creates order item records. The user is linked as the customer [12].
4. **Reduce inventory** – Inventory is decreased using a FIFO strategy. The cost of goods sold (COGS) is calculated based on purchase prices. The function `journal_entry_when_product_is_sold_fifo` creates a journal entry debiting COGS and crediting inventory【997665252291883†L174-L233】.
5. **Record payment** – A payment record is created with method and amount. A journal entry records the sale (credit revenue and debit cash) using the accounting currency [12].

Customers can view their orders via `/ecommerce/orders/`. A detail endpoint `/ecommerce/orders/<id>/retrieve-with-items/` returns the order with nested items and product icon images [17].

Accounting system

A simplified double-entry accounting system is integrated into the application. Key models include:

- **Account** – Represents ledger accounts with a `type` (asset, liability, equity, revenue, expense) and a `parent` for hierarchical organisation [18].
- **JournalEntry** – Represents a dated journal entry with a description and link to the user who made the entry [18].
- **JournalEntryLine** – A line in a journal entry with a debit or credit amount and an associated account [18].

When inventory is adjusted or items are sold, helper functions create balanced journal entries:

- **journal_entries_for_direct_inventory_changes** – When inventory is manually changed (e.g., adjusting stock), this function records a debit to inventory and a credit to inventory adjustment or vice versa [16].
- **journal_entry_when_product_is_sold_fifo** – When a sale occurs, it records the cost of goods sold, revenue, and inventory reduction. It uses FIFO to compute the COGS【997665252291883†L174-L233】.

These functions ensure the accounting ledger remains balanced. The `accounting_for_simple_ecommerce.md` document provides additional guidelines on designing accounts and journal entries [19].

Admin interface

The project registers all models with the Django admin interface. Custom list displays show helpful fields such as price, currency, inventory levels, and foreign keys [20]. Staff can manage categories, products, inventory, orders, purchases, payments, accounts and journal entries through `/admin/`.

Navigating the API

1. **Authenticate** – Register a user via `/ecommerce/auth/registration/` and confirm the email. Then log in at `/ecommerce/auth/login/` to obtain JWT tokens or use the token endpoints under `/api/v1/api/token/`. Include the access token in the `Authorization` header for subsequent requests.
2. **Browse products** – Use `/ecommerce/products/` or `/ecommerce/products-with-image/` to list products. Filter by category or search by name as needed. Customers can view details via `/ecommerce/products/<id>/`.
3. **Manage products (staff)** – Staff can create or update products via `/ecommerce/product/create/` and `/ecommerce/product/update/<id>/`. Bulk import using `/ecommerce/product/import/`.
4. **Handle purchases and inventory (staff)** – Add inventory by creating purchases at `/ecommerce/purchases/`. Review and adjust inventory via `/ecommerce/inventories/`. Bulk import purchases using `/ecommerce/purchase/import/`.

5. **Place orders (customers)** – Build a cart on the client side and post to `/ecommerce/order/create/`. Retrieve orders via `/ecommerce/orders/` and view details with `/ecommerce/orders/<id>/retrieve-with-items/`.
6. **Accounting** – Use `/ecommerce/accounts/`, `/ecommerce/journal-entries/` and related endpoints to view ledger accounts and journal entries. Staff can analyse financial data and ensure transactions are balanced.

Conclusion

This repository demonstrates how to build a full-featured e-commerce back-end using Django REST Framework. It covers authentication with JWT, product catalogue management with images and multi-currency pricing, purchases and inventory control, order processing with cost-of-goods and payments, and an integrated double-entry accounting system. The API design leverages serializers and viewsets to return nested data, while permission classes ensure only staff can perform privileged operations. The repository's documentation provides further guidance on multi-currency pricing, image tagging, and accounting design, making it a solid foundation for building a modern e-commerce platform.

---

[1] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/api/urls.py

[2] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/config/settings.py

[3] [4] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/viewsets/user/viewsets.py

[5] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/permissions.py

[6] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/models/product/models.py

[7] [9] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/viewsets/product/viewsets.py

[8] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/docs/product_images_and_cart_management.md

[10] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/docs/multi_currency_prices.md

[11] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/viewsets/order/utils.py

[12] [17] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/viewsets/order/viewsets.py

[13] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/docs/prices_in_different_currencies.md

[14] [15] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/viewsets/purchase/viewsets.py

[16] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/viewsets/accounting/viewsets.py

[18] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/models/accounting/models.py

[19] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/docs/accounting_for_simple_ecommerce.md

[20] raw.githubusercontent.com
https://raw.githubusercontent.com/samrullo/simple_ecommerce_djrest/main/ecommerce/admin.py