Documentación Técnica - Sistema tienda_pro

■ SYSTEM DOCUMENTATION

1. Project Information

Project Name: **Multi-Branch Retail Management System – “tienda_pro”**
Student Name: **Fredy Enrrique Acosta Paz**
Course: **Databases II**
Semester: **7th Semester**
Date: **November 19, 2025**
Instructor: **Jaider Quintero**

Short Project Description

This project consists of a complete **multi-branch retail management system** developed using **Django REST Framework** (backend) and **Angular CLI 20.3.10** (frontend), with **MySQL** as the primary database engine. :contentReference[oaicite:0]{index=0}
The system manages:
- Branches (physical stores)
- Product catalog and categories
- Taxes
- Inventory per branch
- Customers
- Discount campaigns
- Sales orders and their line details
- Payments
- Audit logs of critical changes

The goal is to centralize the management of a retail chain with multiple branches while keeping full traceability of sales and stock movements.
---

2. System Architecture Overview

2.1 Architecture Description

The system follows a **client–server REST architecture**, where:
- The **backend** (Django REST Framework) exposes RESTful API endpoints for all entities: products, customers, orders, payments, etc.
- The **frontend** (Angular SPA) consumes these endpoints and provides the user interface for administrators, cashiers and inventory managers.
- The **MySQL database** `tienda_pro` stores all operational information of the retail business.

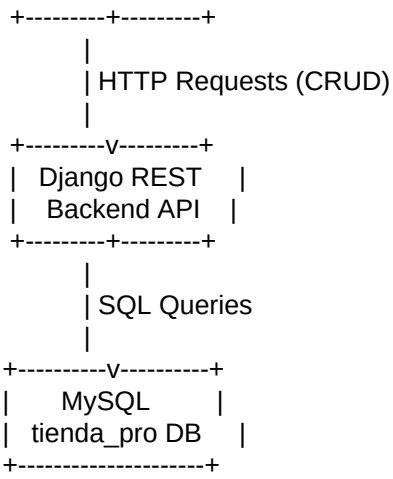
Data flows from the Angular frontend → REST API → Django backend → MySQL database, and responses travel in the opposite direction.

2.2 Technologies Used

- **Frontend:** Angular CLI 20.3.10, TypeScript, HTML, CSS
- **Backend:** Django, Django REST Framework
- **Database Engine:** MySQL
- **Additional Libraries / Tools:**
  - Angular HttpClient
  - Django ORM
  - MySQL Connector

2.3 Visual Explanation of the System’s Operation

```text
        +-------------------+
        |    Angular UI     |
        |  (Frontend SPA)   |
```



3. Database Documentation (ENGLISH)

3.1 Database Description

The tienda_pro database is designed to store and manage all operational information of a multi-branch retail store.

It follows a relational structure to ensure:

Data integrity and consistency through primary and foreign keys.

Efficient querying for reports and day-to-day operations.

Main entities:

Branch – physical stores of the chain.

Category – groups of products.

Tax – tax definitions and percentages.

Product – catalog of items sold.

Inventory – stock and selling prices per product and branch.

Customer – customer master data.

Discount – promotional campaigns and discount rules.

Orders – sales documents.

Order_detail – line items of each order.

Payment – payment records for orders.

Audit_log – audit trail of insert/update/delete operations.

This structure allows full traceability of:

Which products are sold and where,

How much inventory exists in each branch,

Which customers bought what, when and how they paid.

3.2 ERD – Entity Relationship Diagram

(Insert the ERD image exported from MySQL Workbench – tienda_pro.png)

The ERD shows the relationships between: branch, category, tax, product, inventory, customer, discount, orders, order_detail, payment and audit_log.

3.3 Logical Model

Key relationships between entities:

Product – Category – Tax

Each product belongs to one category (product.category_id → category.category_id).

Each product uses one tax type (product.tax_id → tax.tax_id).

Branch – Inventory – Product

Inventory links a product to a branch (inventory.product_id → product.product_id, inventory.branch_id → branch.branch_id).

Stock and selling price are stored at branch level.

Customer – Orders – Branch – Discount

Each order is issued for one customer (orders.customer_id → customer.customer_id).

Each order takes place in one branch (orders.branch_id → branch.branch_id).

Optionally, an order may use a discount campaign (orders.discount_id → discount.discount_id).

Orders – Order_detail – Product

Each order has one or more line items (order_detail.order_id → orders.order_id).
Each line item references a product (order_detail.product_id → product.product_id).
Orders – Payment
Each payment is linked to an order (payment.order_id → orders.order_id).
An order can have one or multiple payments depending on business rules.
Audit_log
Keeps generic references to modified tables and primary key values (table_name, pk_value), independently from the specific schema.
3.4 Physical Model (Tables)
Below is the main table documentation (simplified types).
3.4.1 branch

Table	Column	Type	PK/FK	Description
branch	branch_id	int	PK	Unique branch identifier
branch	branch_name	varchar		Branch name (e.g., “Sucursal Sur #1”)
branch	address	text		Physical address
branch	phone	varchar		Contact phone number
branch	email	varchar		Contact email
branch	manager_name	varchar		Name of the branch manager
branch	opening_date	date		Opening date of the branch
branch	status	varchar		Branch status (active/inactive)
branch	created_at	timestamp		Record creation date

3.4.2 category

Table	Column	Type	PK/FK	Description
category	category_id	int	PK	Unique category identifier
category	category_name	varchar		Category name (e.g., “Computación 1”)
category	description	text		Category description
category	status	varchar		Status (active/inactive)
category	created_at	timestamp		Creation date

3.4.3 tax

Table	Column	Type	PK/FK	Description
tax	tax_id	int	PK	Unique tax identifier
tax	tax_name	varchar		Name (e.g., “IVA 5.00%”)
tax	tax_percentage	decimal		Tax percentage (e.g., 5.00, 19.00)
tax	description	text		Description
tax	status	varchar		Status (active/inactive)
tax	created_at	timestamp		Creation date

3.4.4 product

Table	Column	Type	PK/FK	Description
product	product_id	int	PK	Unique product identifier
product	product_code	varchar		SKU / product code (e.g., SKU-20250001)
product	product_name	varchar		Product name (e.g., “Pro Laptop 101”)
product	description	text		Product description
product	base_price	decimal		Base price (without tax or discounts)
product	category_id	int	FK	References category.category_id
product	tax_id	int	FK	References tax.tax_id
product	weight	decimal		Weight (optional)
product	dimensions	varchar		Dimensions (optional)
product	brand	varchar		Brand name
product	status	varchar		Status (active/inactive)
product	created_at	timestamp		Creation date
product	updated_at	timestamp		Last update date

3.4.5 customer

Table	Column	Type	PK/FK	Description
customer	customer_id	int	PK	Unique customer identifier
customer	document_type	varchar		Document type (id_card, tax_id, passport)
customer	document_number	varchar		Document number

customer	first_name	varchar	First name
customer	middle_name	varchar	Middle name (optional)
customer	last_name	varchar	Last name
customer	second_last_name	varchar	Second last name (optional)
customer	email	varchar	Email address
customer	phone	varchar	Phone number
customer	address	text	Full address
customer	birth_date	date	Birth date (optional)
customer	gender	varchar	Gender (optional, according to design)
customer	status	varchar	Status (active/inactive)
customer	created_at	timestamp	Creation date
customer	updated_at	timestamp	Last update date

3.4.6 discount

Table	Column	Type	PK/FK	Description
discount	discount_id	int	PK	Unique discount identifier
discount	discount_name	varchar		Name (e.g., “Promo 1”)
discount	discount_type	varchar		Type: percentage or fixed_amount
discount	discount_value	decimal		Discount value (percent or fixed amount)
discount	start_date	date		Start date
discount	end_date	date		End date
discount	minimum_amount	decimal		Minimum order total to apply discount
discount	status	varchar		Status (active/inactive)
discount	created_at	timestamp		Creation date

3.4.7 orders

Table	Column	Type	PK/FK	Description
orders	order_id	int	PK	Unique order identifier
orders	order_number	varchar		Human-readable number (e.g., ORD-202500001)
orders	customer_id	int	FK	References customer.customer_id
orders	branch_id	int	FK	References branch.branch_id
orders	discount_id	int	FK	References discount.discount_id (optional)
orders	order_date	timestamp		Date and time of the order
orders	subtotal	decimal		Sum of line subtotals
orders	total_tax	decimal		Total tax amount
orders	total_discount	decimal		Total discount at order level
orders	final_total	decimal		Final amount to be paid
orders	order_status	varchar		Status (pending, completed, cancelled, etc.)
orders	notes	text		Optional comments
orders	salesperson	varchar		Salesperson name
orders	created_at	timestamp		Creation date
orders	updated_at	timestamp		Last update date

3.4.8 order_detail

Table	Column	Type	PK/FK	Description
order_detail	detail_id	int	PK	Unique line identifier
order_detail	order_id	int	FK	References orders.order_id
order_detail	product_id	int	FK	References product.product_id
order_detail	quantity	int		Quantity sold
order_detail	unit_price	decimal		Unit price at the time of sale
order_detail	line_subtotal	decimal		Quantity × unit_price
order_detail	line_tax	decimal		Tax amount for this line
order_detail	line_discount	decimal		Discount applied to this line
order_detail	line_total	decimal		Final line total (subtotal + tax – discount)

3.4.9 inventory

Table	Column	Type	PK/FK	Description
inventory	inventory_id	int	PK	Unique inventory record identifier
inventory	product_id	int	FK	References product.product_id
inventory	branch_id	int	FK	References branch.branch_id

inventory available_quantity int Current stock
inventory minimum_quantity int Minimum stock threshold
inventory selling_price decimal Selling price for this product in this branch
inventory warehouse_location varchar Location in warehouse/store
inventory last_updated timestamp Last inventory update

3.4.10 payment

Table Column Type PK/FK Description
payment payment_id int PK Unique payment identifier
payment order_id int FK References orders.order_id
payment payment_method varchar Method (cash, credit_card, bank_transfer, etc.)
payment cash_amount decimal Amount paid in cash
payment card_amount decimal Amount paid with card
payment transfer_amount decimal Amount paid by bank transfer
payment total_paid decimal Total sum of all methods
payment change_given decimal Change returned to customer
payment authorization_number varchar Authorization code (for card/transfer)
payment payment_date timestamp Date and time of payment
payment payment_status varchar Status (pending, processed, failed, reversed, etc.)
payment payment_notes text Optional notes

3.4.11 audit_log

Table Column Type PK/FK Description
audit_log audit_id bigint PK Unique audit identifier
audit_log table_name varchar Name of the table affected
audit_log pk_value varchar Primary key value of the affected record
audit_log op varchar Operation type (INSERT, UPDATE, DELETE)
audit_log changed_at timestamp Timestamp of the change
audit_log changed_by varchar User or process that performed the change
audit_log before_data json Data before the operation
audit_log after_data json Data after the operation

4. Use Cases – CRUD Operations

4.1 Use Case: CRUD Product

4.1.1 Create Product

Actor: Administrator / Product Manager
Description: Registers a new product in the catalog.
Preconditions: Category and tax must exist.
Postconditions: A new product record is stored in product.
Main Flow:
User selects “Add Product”.
System displays the product registration form.
User fills in SKU, name, category, tax, base price, brand and description.
User submits the form.
System validates the data.
System creates the new product record.
System shows a confirmation message.

4.1.2 Read Product

Actor: Any authorized user
Description: Displays the list of products.
Main Flow:
User selects “Products”.
System loads the product list.
System displays name, SKU, category, price, stock status, etc.

4.1.3 Update Product

Actor: Administrator / Product Manager
Description: Edits information of an existing product.
Main Flow:
User selects a product from the list.

System loads product information.
User edits fields (price, tax, status, etc.).
User saves changes.
System validates and updates the record.
Confirmation message is shown.
4.1.4 Delete / Deactivate Product
Actor: Administrator
Description: Removes or deactivates a product from the catalog.
Main Flow:
User chooses a product.
System asks for confirmation.
User confirms.
System marks the product as inactive or deletes it, according to business rules.
Success notification is shown.
4.2 Use Case: CRUD Customer
Similar structure to Product:
Create Customer: register new customer data.
Read Customer: list all customers.
Update Customer: modify address, contact info or status.
Delete/Deactivate Customer: archive or deactivate customers when needed.
4.3 Use Case: CRUD Branch
Actor: System Administrator
Create new branch (name, address, manager, opening date).
List existing branches.
Update contact information.
Deactivate branch while preserving historical data (orders, inventory).
4.4 Use Case: CRUD Inventory
Actor: Inventory Manager
Create inventory record for a product in a branch.
Read inventory per branch and product.
Update available quantity, minimum quantity and selling price.
Remove or adjust records when stock is restructured.
4.5 Use Case: CRUD Discount
Actor: Marketing Manager
Create discount campaign (discount_name, type, value, dates, minimum amount).
Read active and expired discounts.
Update discount parameters.
Deactivate discounts when campaign ends.
4.6 Use Case: CRUD Order
Create Order
Actor: Cashier / Salesperson
Main Flow:
User selects “New Order”.
Chooses customer and branch.
(Optional) selects a discount.
Adds products and quantities to the order detail.
System calculates subtotal, tax, discount and final total.
User saves the order.
System stores the order in orders and lines in order_detail.
Read / Update / Delete Order
View order history and details.
Modify pending orders (add/remove products, change quantities).
Cancel orders (set status cancelled, revert stock if required).
4.7 Use Case: CRUD Payment
Actor: Cashier
Create Payment: register payment for an existing order with one or several methods.

Read Payment: view payment history per order, per customer or per date.
Update Payment: correct method or status (e.g., mark as reversed).
Delete Payment: remove erroneous payments (with audit logging).
9. Backend Documentation
9.1 Backend Architecture
The backend of the retail system is built using Django and Django REST Framework (DRF), following a modular app-based structure similar to the clinical system.
System_documentation
DRF components used:
Models – map tables in tienda_pro (Product, Branch, Order, etc.).
Serializers – convert models ↔ JSON.
Views / ViewSets – implement business logic and expose CRUD endpoints.
URLs – route HTTP requests to viewsets.
Permissions – control access to APIs (future improvement).
This modular design improves scalability, maintenance and reusability.
9.2 Backend Folder Structure (Suggested)
backend/
■■■ tienda_site/
■ ■■■ settings.py
■ ■■■ urls.py
■ ■■■ wsgi.py
■ ■■■ asgi.py
■
■■■ apps/
■ ■■■ catalog/ # product, category, tax
■ ■■■ location/ # branch, inventory
■ ■■■ customer/ # customer
■ ■■■ sales/ # orders, order_detail, discount
■ ■■■ payment/ # payment
■ ■■■ audit/ # audit_log
■
■■■ manage.py
Each app contains:
app_name/
■■■ models.py
■■■ serializers.py
■■■ views.py or viewsets.py
■■■ urls_viewset.py
■■■ admin.py
9.3 Models by Application
Catalog App = { Product, Category, Tax }
Location App = { Branch, Inventory }
Customer App = { Customer }
Sales App = { Orders, Order_detail, Discount }
Payment App = { Payment }
Audit App = { Audit_log }
9.4 URL Routing Structure
tienda_site/urls.py:
urlpatterns = [
path('admin/', admin.site.urls),
path('api/', include('apps.catalog.urls_viewset')),
path('api/', include('apps.location.urls_viewset')),
path('api/', include('apps.customer.urls_viewset')),
path('api/', include('apps.sales.urls_viewset')),
path('api/', include('apps.payment.urls_viewset')),
path('api/', include('apps.audit.urls_viewset')),

]
Example API endpoints:
/api/products/
/api/categories/
/api/taxes/
/api/branches/
/api/inventory/
/api/customers/
/api/orders/
/api/order-details/
/api/discounts/
/api/payments/
/api/audit-logs/
9.5 API Documentation (REST Examples)
Example 1 – Create Product
Endpoint: POST /api/products/
Body:

```
{
  "product_code": "SKU-20250001",
  "product_name": "Pro Laptop 101",
  "description": "High performance laptop",
  "base_price": 1907.29,
  "category_id": 1,
  "tax_id": 1,
  "brand": "Globex",
  "status": "active"
}
```

Responses:
201 Created – Product successfully created.
400 Bad Request – Validation errors.
Example 2 – List Orders
Endpoint: GET /api/orders/
Description: Returns a list of all orders with customer, branch and totals.
Example 3 – Create Payment for an Order
Endpoint: POST /api/payments/

```
{
  "order_id": 1,
  "payment_method": "credit_card",
  "cash_amount": 0.00,
  "card_amount": 4005.31,
  "transfer_amount": 0.00,
  "total_paid": 4005.31,
  "change_given": 0.00,
  "authorization_number": "AUTH-202500001",
  "payment_status": "processed",
  "payment_notes": "Automatic payment registration"
}
```

Example 4 – Get Inventory by Branch
Endpoint: GET /api/inventory/?branch_id=1
Description: Returns all inventory records for branch 1 (e.g., “Sucursal Sur #1”).
Example 5 – Apply Discount to Order
Endpoint: PUT /api/orders/1/

```
{
  "discount_id": 1
}
```

Backend recalculates total_discount and final_total based on discount rules.

9.6 Authentication
At this stage, a login system may not be implemented for the demo version:
No JWT or session-based authentication.
APIs might be public/unprotected.
For a production system, it is recommended to:
Add JWT authentication (e.g., DRF SimpleJWT).
Define roles (admin, manager, cashier, auditor).
Restrict access to sensitive endpoints.
10. Frontend Documentation
The frontend of the retail system uses Angular CLI 20.3.10 with a modular structure of components, services and models for each entity, similar to the clinical system.
System_documentation
10.1 Technical Frontend Documentation
Framework: Angular CLI 20.3.10
UI Library: PrimeNG, PrimeIcons
Languages: TypeScript, HTML, CSS
10.2 Project Folder Structure (Suggested)
frontend/
■■■ angular.json
■■■ package.json
■■■ tsconfig.json
■
■■■ src/
  ■■■ main.ts
  ■■■ index.html
  ■■■ styles.css
  ■
  ■■■ app/
    ■■■ app.component.ts
    ■■■ app.component.html
    ■■■ app.routes.ts
    ■
    ■■■ layout/
    ■ ■■■ aside/
    ■ ■■■ footer/
    ■ ■■■ header/
    ■
    ■■■ components/
    ■ ■■■ branch/
    ■ ■■■ category/
    ■ ■■■ tax/
    ■ ■■■ product/
    ■ ■■■ inventory/
    ■ ■■■ customer/
    ■ ■■■ order/
    ■ ■■■ order-detail/
    ■ ■■■ payment/
    ■ ■■■ discount/
    ■ ■■■ audit/
    ■
    ■■■ services/
    ■ ■■■ branches.service.ts
    ■ ■■■ categories.service.ts
    ■ ■■■ taxes.service.ts
    ■ ■■■ products.service.ts
    ■ ■■■ inventory.service.ts

- ■■■ customers.service.ts
- ■■■ orders.service.ts
- ■■■ payments.service.ts
- ■■■ discounts.service.ts
- ■■■ audit.service.ts
- ■

■■■ models/

- ■■■ branch.ts
- ■■■ category.ts
- ■■■ tax.ts
- ■■■ product.ts
- ■■■ inventory.ts
- ■■■ customer.ts
- ■■■ order.ts
- ■■■ order-detail.ts
- ■■■ payment.ts
- ■■■ discount.ts
- ■■■ audit.ts

10.3 Services (API Communication)

Each entity communicates with the Django API through an Angular service.

Example: Product Service

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Product, ProductCreate } from '../models/product';
@Injectable({
  providedIn: 'root'
})
export class ProductService {
  private baseUrl = 'http://localhost:8000/api/products';
  constructor(private http: HttpClient) {}
  getProducts(): Observable<Product[]> {
    return this.http.get<Product[]>(`${this.baseUrl}/`);
  }
  getProduct(id: number): Observable<Product> {
    return this.http.get<Product>(`${this.baseUrl}/${id}/`);
  }
  createProduct(payload: ProductCreate): Observable<Product> {
    return this.http.post<Product>(`${this.baseUrl}/`, payload);
  }
  updateProduct(id: number, payload: Partial<ProductCreate>): Observable<Product> {
    return this.http.put<Product>(`${this.baseUrl}/${id}/`, payload);
  }
  deleteProduct(id: number): Observable<void> {
    return this.http.delete<void>(`${this.baseUrl}/${id}/`);
  }
}
```

All other services (Customer, Branch, Order, Payment, etc.) follow a similar pattern.

10.4 Models (TypeScript Interfaces)

Models define the TypeScript interfaces used by components and services.

Example: Product Model

```
export interface Product {
  product_id?: number;
  product_code: string;
  product_name: string;
  description: string;
```

```
  base_price: number;
  category_id: number;
  tax_id: number;
  weight?: number;
  dimensions?: string;
  brand?: string;
  status: 'active' | 'inactive';
  created_at?: string;
  updated_at?: string;
}
export type ProductCreate = Omit<Product, 'product_id' | 'created_at' | 'updated_at'>;
```

Each entity (Branch, Customer, Order, Payment, etc.) has its own interface.

10.5 Components

Each Angular component is responsible for:

Displaying tables with PrimeNG (lists).

Showing forms for creating and editing records.

Handling delete/confirm dialogs.

Navigating between modules through the main layout.

Typical components per module:

product-list, product-form

customer-list, customer-form

order-list, order-detail, order-form

payment-list, payment-form

10.6 Layout System and Visual Explanation

The main layout contains:

A sidebar with navigation links (Products, Customers, Orders, etc.).

A header with system title (e.g., “Multi-Branch Retail System”).

A central router-outlet where module components are rendered.

Screenshots to include in the final documentation:

Dashboard / Main Layout – header + sidebar.

Product Module – product list and create form.

Customer Module – customer list and create form.

Order Module – create order, show order details.

Payment Module – register payment for an order.

Inventory Module – inventory by branch.

Each screenshot should be labeled, e.g.:

Figure X. Product Module – list and CRUD actions.

11. Frontend–Backend Integration

Angular communicates with the Django REST API using:

HttpClient in services (GET, POST, PUT, DELETE).

JSON serialization – DRF sends JSON, Angular models map the fields.

Consistent routing – Angular calls /api/products/, Django exposes /api/products/.

CORS configuration – backend allows requests from http://localhost:4200 during development.

Example call:

```
this.http.get<Product[]>('http://localhost:8000/api/products/');
```

12. Conclusions & Recommendations

The tienda_pro database provides a solid, normalized model for multi-branch retail operations.

The combination of Django REST + Angular forms a clean separation between backend logic and frontend UI.

The modular design (apps/services/components) simplifies maintenance and future extensions.

The system supports complete workflows: product management, inventory by branch, customer management, sales orders, payments and discounts.

For future work it is recommended to:

Implement authentication and role-based permissions.

Add reporting modules (top-selling products, sales per branch, low stock alerts).

Improve audit log visualization on the frontend for better traceability.