# TuTienda — Documentación Final

**Carrera:** Ingeniería en Sistemas de Información
**Materia:** Paradigmas y Lenguajes de Programación
**Estudiante:** Silveyra Mattos Tomás Agustín
**Profesor:** Mgter. Ing. Agustín Encina
**Fecha:** 2025-10-07

## 0) Resumen ejecutivo

TuTienda es una demo de e-commerce construida con **Astro** (front SSR/SSG) y un **API Node/Express + Prisma** (SQLite/Postgres) para administración de productos/categorías vía **JWT**.
El catálogo se puede navegar en tarjetas/tabla, ver detalle, y agregar al carrito (storage local).
El **panel admin** permite crear/editar/borrar productos, marcarlos como destacados y asociarlos a categorías (incluyendo jerarquías padre>hija>nieta).

## 1) Qué cambió respecto a la entrega anterior

- **Backend real** con Express + Prisma (antes era solo mock).
- **Autenticación JWT** (login, me) y helper `public/admin-auth.js`.
- **CRUD de productos**: `GET/POST/PATCH/DELETE /api/products`.
- **Categorías** con jerarquía y tabla pivote `ProductCategory`.
- **CORS** configurado para desarrollo (Astro en 4321, API en 8787).
- **Listado público** ahora **lee de la API** y **hace fallback** a mock si la API no responde.
- **Detalle dinámico** `/producto/[slug]` con API→mock y preserva `?store=`.
- **Featured**: flag `isFeatured` en BD + “Destacados” en Home.

- **Admin UI** mejorado: vista previa, autogeneración de slug, chips de categorías y creación rápida padre>hija>nieta.

## 2) Arquitectura general

Frontend (Astro)
├─ Páginas públicas: index, listado_box, listado_tablas, producto/[slug], comprar
├─ Panel admin: /admin/productos (form + listado)
├─ Carrito: storage local + util tpCart
└─ Helper admin: public/admin-auth.js (JWT + fetch con Authorization)

API (Node/Express)
├─ /api/auth (login, me)
├─ /api/products (GET/POST/PATCH/DELETE)
├─ /api/categories (GET/POST/PATCH)
└─ CORS + JSON + Prisma Client (DB)

Base de datos (Prisma)
├─ Store
├─ Product (→ Store)
├─ Category (→ Store, self parent)
└─ ProductCategory (Product ↔ Category)

# 3) Esquema de datos (ER)


.

## 4) Endpoints principales

### Auth

- `POST /api/auth/login` → `{ token, user }`
  body: `{ email, password }`

- `GET /api/auth/me` → `{ user }` (requiere `Authorization: Bearer <token>`)

### Products

- `GET /api/products?store=:slug[&cat=:catSlug]`
  Lista productos de una tienda (opcional: filtra por categoría).

- `POST /api/products` (JWT)
  body: `{ title, slug, price, description?, imageUrl?, storeSlug, isFeatured?, categorySlugs?[] }`
  • **Valida**: title, slug, price, storeSlug
  • Enlaza categorías por `storeId + slug`

- `PATCH /api/products/:id` (JWT)
  body parcial con campos a actualizar (incluye `isFeatured`).

- `DELETE /api/products/:id` (JWT)

### Categories

- `GET /api/categories?store=:slug` → lista categorías (para el selector/auto-complete).

- `POST /api/categories` (JWT)
  body: `{ storeSlug, name, slug, parentId? }`.

- `PATCH /api/categories/:id` (opcional)

## 5) CORS y configuración

- **API_PORT**: `8787` (por defecto)
- **FRONT_ORIGIN**: `http://localhost:4321` (o múltiples separados por coma)
- CORS en API:
    - `origin`: FRONT_ORIGIN (soporta múltiples)
    - `credentials: true`
    - `allowedHeaders`: `Content-Type`, `Authorization`
    - Métodos: `GET, POST, PATCH, DELETE, OPTIONS`

## 6) Autenticación (JWT) y helper de admin

- El login guarda el token en `localStorage` (clave interna).
- `public/admin-auth.js` expone:
    - `tpAdminAuth.login(email, password)`
    - `tpAdminAuth.me()` (usa `/api/auth/me`)
    - `tpAdminAuth.api(path, { method, body })` → incluye `Authorization` y `Content-Type: application/json`.
- El **panel admin** redirige a `/admin/login` si `me()` no devuelve usuario.

# 7) Frontend (Astro)

## Páginas públicas

- **Home (`src/pages/index.astro`)**
    - Hero configurable, CTAs, y **“Destacados”**.
    - Si hay `featuredSlugs` (config opcional), filtra; sino usa `isFeatured`.
- **Listado Box (`src/pages/listado_box.astro`)**
    - **Fetch a la API** `GET /api/products?store=...`
    - **Fallback a mock** `src/data/products.ts` si falla la API.
    - Filtro por texto, orden por título/precio, “solo destacados”.
- **Listado Tabla (`src/pages/listado_tablas.astro`)**
    - Igual idea que Box, pero maquetado en tabla (comparación rápida).
- **Detalle (`src/pages/producto/[slug].astro`)**
    - Busca en API por tienda (`?store=`). Si falla, **fallback** al mock.
    - Recomendados: 3 del mismo origen (API o mock).
- **Comprar** (simple demo) y util **tpCart** (localStorage).

## Panel admin

- **`/admin/productos`**
    - **Form**: título, slug (auto), precio (normaliza `1.234,56 → 1234.56`), imagen, descripción, **destacado**, **categorías** (chips).
    - **Categorías**: buscador + **creación rápida** escribiendo `padre > hija > nieta` y Enter.

- **Listado** con miniatura, precio, fecha, botón Destacado (toggle), Editar/Eliminar.
- **Validación mínima** en cliente y errores legibles si falta algún campo.

## 8) Flujo: crear un producto (resumen)

1. Admin completa form → JS normaliza `price`, arma `body` y detecta `storeSlug`.
2. (Opcional) Crea categorías nuevas escritas como `padre > hija`.
3. `POST /api/products` con JWT → server valida y crea:
   - Verifica `Store` por `storeSlug`.
   - Enforce slug único por tienda.
   - Crea `Product` y vínculos `ProductCategory`.
4. UI refresca lista y limpia formulario (mantiene `storeSlug`).

# 9) Cómo correr el proyecto (dev)

**Clonar & deps**

```
pnpm install
```

1.

**Variables de entorno (`.env`)**

```
DATABASE_URL="file:./dev.db"            # o Postgres
JWT_SECRET="tu-secreto-fuerte"          # requerido por /auth
FRONT_ORIGIN="http://localhost:4321"    # puede ser lista separada por coma
API_PORT=8787
```

2.

**Prisma**

```
pnpm prisma generate
pnpm prisma migrate dev --name init
```

3.

**Levantar API y Front**

```
# API
pnpm api:watch   # nodemon/tsx server/index.ts

# Front (en otra terminal)
pnpm dev         # astro dev en :4321
```

4.

> **Tip:** si ves 400 `"title, slug, price y storeSlug son requeridos"`, revisá que el **form JS esté enviando `storeSlug`** (se toma de `#storeSlugHidden` o `?store=` o subdominio), y que `Content-Type: application/json` esté bien puesto por `tpAdminAuth.api`.

## 10) Checklist de pruebas

- Login válido devuelve token y `me()` retorna usuario.
- `POST /api/products` con `{ title, slug, price, storeSlug }` **ok**.
- Crear producto con `isFeatured`=true → aparece en Home/box/tabla.
- Crear categorías `padre > hija` desde admin y asignarlas.
- `PATCH /api/products/:id { isFeatured: false }` actualiza.
- `GET /api/products?store=mi-tienda-demo` lista lo creado.
- `GET /api/products?store=...&cat=telefonia` filtra por categoría.
- Borrar un producto → desaparece del listado y del front.
- CORS ok: front 4321 llama a API 8787 sin errores preflight.
- Fallback: si API caída, front usa `src/data/products.ts`.

## 11) Accesibilidad y performance

- Imágenes con `alt`, `loading="lazy"` y wrappers 16:9 con `object-contain`.
- Controles con labels y live-region en detalle para anunciar “agregado al carrito”.
- Build Astro optimizado; CSS simple y diseño responsive.

# 12) Trabajo futuro

- Órdenes/pedidos reales y checkout.
- Paginación y filtros por categoría/price-range del lado del server.
- Panel de categorías dedicado (ABM + drag&drop jerárquico).
- Rate-limiting + refresh tokens.
- Imágenes subidas por el dueño (S3/Cloudinary).

# 13) Créditos y licencias

- Astro, Express, Prisma — MIT.
- Este repo es material académico.

### Anexo A — Secuencia “Crear producto” (Mermaid)

```
sequenceDiagram
  participant Admin
  participant Front as Admin UI (Astro)
  participant API as API Express
  participant DB as Prisma/DB

  Admin->>Front: Completa form + Enter categorías
  Front->>API: POST /api/categories (si hay nuevas)
  API->>DB: Inserta categorías (storeId, slug, parentId)
  DB-->>API: OK
  Front->>API: POST /api/products {title, slug, price, storeSlug, ...}
  API->>DB: Valida store, slug único, inserta producto
  API->>DB: Vincula ProductCategory
  DB-->>API: OK
  API-->>Front: { product }
  Front-->>Admin: Toast “Producto creado” + Refrescar listado
```