Djangify eCommerce Builder

Self-Hosted and Managed Hosting eCommerce Platform for Digital Downloads

Comprehensive Documentation & Technical Reference

Version 1.0 | Django 5.2 LTS | Docker-First Distribution

Table of Contents

1. Introduction & Overview

The Djangify eCommerce Builder (aka eBuilder) is an eCommerce platform specifically designed for selling digital downloads.

Built with Django and distributed via Docker, it provides creators and businesses with complete ownership of their online store, data, and revenue streams.

It is available as a **self-hosted version** that you host yourself or with **hosting (and maintenance) managed by Djangify**.

Unlike SaaS platforms such as Shopify or Gumroad, eBuilder eliminates recurring platform fees, transaction percentages, and vendor lock-in. Once installed, the self-hosted version is yours to host anywhere, change freely, and scale according to your needs.

Philosophy

The eCommerce Builder follows these core principles:

- Ownership First: Your store, your data, your customers, your income
- Docker-First Distribution: Consistent deployments across any environment
- Admin-First Design: Manage everything through Django's powerful admin interface
- SQLite by Default: Simple, reliable, and performant without database server complexity
- No Feature Bloat: Essential ecommerce functionality without unnecessary complexity

What the eCommerce Builder Is NOT

- Not for shared hosting; requires Docker/VPS environment
- Not a page builder it is an eCommerce system with CMS capabilities

- Not for physical products it is for digital downloads only (v1)
- Not a SaaS - you host and own everything (self-hosted)
- Not a SaaS - we maintain everything and you can move your whole site by taking your database with your data - (managed hosting)
- Not a marketplace. It is a single-vendor store only

2. Target Audience

The Djangify eCommerce Builder is designed for creators and businesses who want independence from platform fees and restrictions:

Primary Users

- Digital Product Creators: Selling PDFs, guides, templates, or digital tools
- Course Creators: Offering downloadable educational content
- Digital Artists: Selling artwork, design assets, or creative resources
- Software Developers: Distributing scripts, code templates, or tools
- Musicians & Audio Producers: Selling beats, samples, or audio files
- Photographers: Distributing stock photos or preset packs

Technical Requirements

- Basic command line familiarity
- Access to a VPS (DigitalOcean, Hetzner, Vultr, etc.)
- Domain name
- Stripe account for payment processing

3. Key Features

3.1 Complete Shop System

The shop module provides everything needed for selling digital products:

- Product Management: Create products with titles, descriptions, images, and pricing
- Multiple Downloads Per Product: Attach multiple downloadable files to a single product.
- Download Limits: Configure per-purchase download limits for security
- Categories: Organize products into categories with their own pages
- Secure Downloads: Files served through authenticated endpoints, not direct URLs
- Product Reviews: Customers can leave ratings and reviews
- Wishlist: Users can save products for later
- Order History: Complete purchase history accessible from customer dashboard
- Multi-Currency Support: Configure currency via Site Settings
- Video Embeds: YouTube video support for product demonstrations

3.2 Stripe Integration

Payment processing is handled entirely through Stripe:

- Stripe Checkout: Modern, secure payment flow
- PaymentIntent API: Proper payment handling with webhook verification
- Automatic Receipts: Stripe sends receipts to customers
- Webhook Handling: Secure webhook endpoint for payment confirmations
- Test Mode Support: Full testing capability with Stripe test keys

3.3 Content Management

Beyond the shop, eBuilder includes a complete content management system:

Blog System

- Full blogging with categories
- Featured posts highlighted on the blog index
- YouTube video embeds with automatic thumbnail extraction
- External image URL support
- Meta titles and descriptions for SEO
- Social sharing buttons

Custom Pages (Page Builder)

- Hero Sections: Full-width banners with title, subtitle, body, image, and CTA button
- Page Sections: Rich text content blocks with images
- Three-Column Blocks: Feature cards with icons/images and descriptions
- Gallery Blocks: Image galleries with automatic thumbnail generation
- FAQ Blocks: Expandable question/answer sections
- Template Selection: Homepage, About, Custom Page, or Gallery layouts

InfoPages

- Documentation pages with category organization
- Policy pages (Privacy, Terms, Refunds, etc.)
- Automatic table of contents generation from headings
- Separate URL structures (/docs/ and /policies/)

3.4 SEO & Discovery

Comprehensive SEO features are built into every page:

- Schema.org Structured Data: Product, Article, ItemList, Organization, and WebPage schemas
- Open Graph Tags: Facebook and social media optimization
- Twitter Card Integration: Rich previews on Twitter/X
- AI Search Metadata: Page-type metadata for AI crawlers
- Automatic Sitemap: XML sitemap generation
- Robots.txt: Configurable with AI bot support
- Canonical URLs: Prevent duplicate content issues
- Meta Titles & Descriptions: Per-page SEO customization

3.5 Design & User Interface

- Fully Responsive: Mobile-first design that works on all devices
- Tailwind CSS 4: Modern utility-first styling with CSS custom properties
- Adminita Theme: Beautiful, dark-mode compatible admin interface
- ARIA Compliant: Accessible to screen readers and assistive technologies
- HTMX Integration: Dynamic interactions without page reloads
- Alpine.js: Lightweight JavaScript for UI interactions

4. Architecture & Tech Stack

4.1 Core Technologies

Component	Technology
Framework	Django 5.2 LTS
Frontend CSS	Tailwind CSS 4
Frontend JS	Alpine.js, HTMX
Database	SQLite with WAL mode (default)
Alternative DB	PostgreSQL (optional)
Payments	Stripe
Authentication	django-allauth
Rich Text Editor	TinyMCE
Admin Theme	Adminita
Container	Docker + Gunicorn
Reverse Proxy	Nginx/Caddy
Static Files	WhiteNoise

4.2 Application Architecture

The eCommerce Builder follows Django's app-based architecture:

- accounts: User authentication, registration, customer dashboard
- shop: Products, categories, cart, checkout, orders, reviews, wishlist
- blog: Posts, categories, featured content
- pages: Site settings, custom pages, page builder blocks
- infopages: Documentation and policy pages

- hosting: Managed hosting signup flow (for service providers)

4.3 Database Strategy

SQLite with WAL (Write-Ahead Logging) mode is the default database:

- No separate database server required
- Excellent read performance for typical ecommerce workloads
- WAL mode enables concurrent reads during writes
- Simple backup: just copy the database file
- PostgreSQL can be used if wanted as an alternative for high-concurrency scenarios

5. Project Structure

The project follows Django conventions with Docker-first organization:

```
ebuilder/
├── docker-compose.yml     # Container orchestration
├── Dockerfile             # Application container definition
├── requirements.txt       # Python dependencies
├── manage.py              # Django management script
├── .env.example           # Environment variable template
│
├── ebuilder/              # Project configuration
│   ├── settings.py        # Main Django settings
│   ├── urls.py            # Root URL configuration
│   ├── sitemaps.py        # Sitemap configuration
│   └── wsgi.py            # WSGI entry point
│
├── accounts/              # User authentication app
│   ├── models.py          # Custom user model
│   ├── views.py           # Dashboard, profile views
│   ├── adapters.py        # Allauth adapter customization
│   └── templates/         # Auth templates
│
```

```
├── shop/                # eCommerce app
│   ├── models.py        # Product, Order, Review models
│   ├── views/           # Shop views (products, cart, checkout)
│   ├── cart.py          # Session-based cart
│   ├── emails.py        # Order confirmation emails
│   ├── webhooks.py      # Stripe webhook handling
│   └── templates/       # Shop templates
│
├── blog/                # Blog app
│   ├── models.py        # Post, Category models
│   └── templates/       # Blog templates
│
├── pages/               # Page builder app
│   ├── models.py        # Page, Hero, content block models
│   ├── context_processors.py  # Global template context
│   └── templates/       # Page templates and sections
│
├── infopages/           # Documentation app
│   ├── models.py        # InfoPage, Category models
│   └── templates/       # Doc/policy templates
│
├── templates/           # Global templates
│   ├── base.html        # Base template
│   ├── includes/        # Reusable components
│   │   └── seo/         # SEO schema templates
│   └── account/         # Allauth template overrides
│
├── static/              # Static assets
│   ├── css/             # Stylesheets
│   ├── js/              # JavaScript files
│   └── images/          # Static images
│
└── data/                # Persistent data (mounted volume)
    ├── db/              # SQLite database
    ├── media/           # User uploads
    └── logs/            # Application logs
```

6. Installation & Deployment

6.1 Prerequisites

Before starting, ensure you have:

Requirement	Minimum	Recommended
VPS/Server	2 CPU, 1GB RAM	3 CPU, 2GB RAM
OS	Ubuntu 22.04+	Ubuntu 24.04 LTS
Docker	v24+	Latest
Docker Compose	v2.20+	Latest
Domain	Required	With SSL
Stripe Account	Required	Live mode for production

Suggested VPS providers: Hetzner, DigitalOcean, Linode, Vultr

6.2 Quick Start Installation

The standard installation only takes a few minutes:

1. **Upload the eCommerce Builder folder to your server:**

```
# Using scp from your local machine
scp -r ebuilder-selfhosted/ user@yourserver:/home/user/ebuilder

Or upload via SFTP using FileZilla, WinSCP, etc.
```

2. Configure Environment

```
cd /home/user/ebuilder
cp .env.example .env
nano .env

Minimum required changes
SECRET_KEY=generate-a-long-random-string-here
ALLOWED_HOSTS=yourdomain.com,www.yourdomain.com
CSRF_TRUSTED_ORIGINS=https://yourdomain.com,https://www.yourdomain.com
```

```
STRIPE_PUBLIC_KEY=pk_live_xxx
STRIPE_SECRET_KEY=sk_live_xxx
STRIPE_WEBHOOK_SECRET=whsec_xxx

Generate a secret key:
python3 -c "import secrets; print(secrets.token_urlsafe(50))"

3. Build and Start
docker compose build
docker compose up -d

4. Create Admin User
docker compose exec web python manage.py createsuperuser

5. Visit your site
http://localhost:8000 (or your domain)
```

6.3 Environment Configuration

Essential .env variables:

```
# Django Settings
SECRET_KEY=your-unique-secret-key-here
DEBUG=False
ALLOWED_HOSTS=yourdomain.com,www.yourdomain.com

# Stripe (required for payments)
STRIPE_PUBLIC_KEY=pk_live_...
STRIPE_SECRET_KEY=sk_live_...
STRIPE_WEBHOOK_SECRET=whsec_...

# Email (optional but recommended)
EMAIL_HOST=smtp.example.com
EMAIL_PORT=587
EMAIL_HOST_USER=your@email.com
EMAIL_HOST_PASSWORD=your-password
```

```
# Database (optional - defaults to SQLite)
# DATABASE_URL=postgresql://user:pass@db:5432/ebuilder
```

6.4 SSL/HTTPS Setup

For production, configure SSL through your reverse proxy (Nginx, Caddy, or HestiaCP). The application runs behind the proxy on port 8000.

See the full instructions inside the folder called ***Your Documentation***

7. Configuration

7.1 Initial Site Setup

After installation, configure your site through the admin interface:

1. Visit /admin and log in with your superuser credentials
2. Go to Pages → Site Settings
3. Configure: Business name, Site URL, Support email, Logo, Currency, Default SEO

7.2 Shop Configuration

Configure shop-specific settings in Shop → Shop Settings:

- Hero Section: Title, subtitle, image, call-to-action
- Products Per Page: Pagination settings
- Display Mode: Grid or list view
- Intro Section: Welcome text and styling
- Promo Blocks: Three-column promotional cards
- Product Spotlight: Featured product highlight section

7.3 Homepage Configuration

The homepage is controlled through Pages → Homepage Settings:

- Layout Selection: Shop-focused or content-focused
- Section Visibility: Toggle blog posts, products, gallery
- Section Ordering: Control display order of components
- Section Titles & Descriptions: Customize each section’s heading

8. Admin Interface Guide

The eCommerce Builder uses Django's admin interface enhanced with the Adminita theme for all management tasks.

8.1 Admin Structure

The admin is arranged into logical sections:

- ACCOUNTS: Users, email addresses
- AUTHENTICATION: Groups, permissions
- BLOG: Posts, categories
- INFO PAGES: Documentation, policies, categories
- PAGES: Site settings, pages, hero sections, dashboard settings
- SHOP: Products, categories, orders, reviews, shop settings
- SITES: Django sites framework

8.2 Key Admin Features

- Collapsible Sections: Fieldsets collapse to reduce clutter
- List Filters: Filter products by category, status, featured
- Search: Quick search across titles and content
- Bulk Actions: Select multiple items for batch operations
- Dark Mode: Toggle between light and dark themes

8.3 Content Block Management

Pages use inline content blocks managed within the page admin:

- Hero Sections: Add one hero banner per page
- Page Sections: Rich text content with TinyMCE editor
- Two and Three-Column Blocks: Feature cards with images
- Gallery Blocks: Image galleries (click through to manage images)
- FAQ Blocks: Question/answer pairs (click through to manage items)

9. Shop Module

9.1 Product Management

Products are managed through Shop → Products:

Product Fields

- Title & Slug: Product name and URL-friendly identifier
- Category: Product categorization
- Description: Full product description (TinyMCE rich text)
- Excerpt: Short description for listings
- Price (pence): Price in smallest currency unit
- Sale Price: Optional discounted price
- Preview Image: Product thumbnail for listings
- Video URL: Optional YouTube video embed
- Download Limit: Maximum downloads per purchase
- Featured: Highlight on shop homepage
- Published: Control visibility

Product Downloads

Each product can have multiple downloadable files:

- Title: Display name for the download
- File: The actual downloadable file (stored securely)
- Version: Track file versions
- Order: Display sequence for multiple downloads

9.2 Order Management

Orders are created automatically after successful Stripe payments:

- Order ID: Unique identifier (auto-generated)
- Customer: Linked to user account

- Status: Pending, Completed, Failed, Refunded
- Payment Intent ID: Stripe reference for verification
- Order Items: Products purchased with prices paid
- Downloads Remaining: Track per-item download limits

9.3 Cart System

The cart is session-based and does not require login:

- Add to Cart: Products added via AJAX
- Quantity Updates: Adjust quantities in cart view
- Cart Persistence: Survives browser sessions
- **Checkout Requires Login**: User must authenticate to purchase

9.4 Checkout Flow

4. Customer adds products to cart
5. Proceeds to checkout (login required)
6. Stripe PaymentIntent created with cart total
7. Customer enters payment details in Stripe Elements form
8. Payment processed by Stripe
9. Webhook confirms payment success
10. Order created, confirmation email sent
11. Customer accesses downloads via dashboard

10. Blog Module

10.1 Post Management

- Title & Slug: Post title and URL identifier
- Category: Blog post categorization
- Content: Full post content (TinyMCE rich text)
- Image: Featured image for post and thumbnails
- External Image URL: Alternative to uploaded image
- YouTube URL: Embed video instead of/alongside image
- Publish Date: Schedule posts for future publication
- Status: Draft or Published
- Featured: Highlight on blog index
- Resource: Add a PDF that user can download
- Meta Title/Description: SEO customization

10.2 YouTube Integration

Posts support YouTube video embeds:

- Paste any YouTube URL (watch, share, or embed format)
- Video ID automatically extracted
- Thumbnail pulled from YouTube for listings
- Responsive embed on post detail page
- Falls back to featured image if no video

10.3 Blog Display

- Blog Index: Paginated list of published posts
- Featured Posts: Highlighted section on first page
- Category Filtering: Browse posts by category
- Social Sharing: Share buttons on each post

11. Pages Module (Page Builder)

11.1 Page Types

Each page uses a template that determines its layout:

- Homepage: Dynamic homepage with configurable sections
- About: Standard about page layout
- Custom Page: Flexible template for any content
- Gallery Page: Image-focused layout

11.2 Content Blocks

Hero Section

Full-width banner sections with:

- Title and subtitle
- Body content (rich text)
- Background image
- Call-to-action button with link
- Order position control
- Active/inactive toggle

Text Block (no image)

One column block of text with white background:

- Each column: title and body content
- Perfect for page introduction and adding text sections
- Optional button with link

Two Column Block

Two sections with image and text:

- Title and subtitle

- Body content (TinyMCE rich text)
- Image appears on left hand side
- Optional button with link

Three-Column Block

Feature cards displayed in three columns:

- Each column: title, image, body content
- Perfect for features, benefits, or services
- Images displayed as icons

Gallery Block

Image galleries with:

- Gallery title
- Multiple images with captions in masonry grid
- Automatic thumbnail generation
- Lightbox viewing

FAQ Block

Expandable FAQ sections:

- Block title
- Multiple question/answer pairs
- Accordion-style expansion

11.3 Site Settings

Global settings managed through Pages → Site Settings:

- Business Name: Displayed in header and footer
- Site URL: Used for canonical URLs and schema
- Support Email: Contact email address
- Logo: Site logo for header

- Currency Code: Payment currency
- Site Color: Add your default brand colors
- Social Links: Footer social media links
- Newsletter HTML: Email signup embed code
- Default SEO: Fallback meta title/description

12. InfoPages Module

12.1 Documentation Pages

Documentation pages accessible at /docs/:

- Category Organization: Group docs into categories
- Automatic TOC: Table of contents generated from headings
- Rich Content: TinyMCE editor for formatted content
- SEO Fields: Custom meta title and description

12.2 Policy Pages

Legal/policy pages accessible at /policies/:

- Privacy Policy
- Terms of Service
- Refund Policy
- Cookie Policy
- Any other legal documents

12.3 InfoPage Fields

- Title & Slug: Page title and URL
- Page Type: Documentation or Policy
- Category: Organization (docs only)
- Content: Full page content (rich text)
- Published: Control visibility and Last Updated: Automatic timestamp

13. SEO Features

13.1 Schema.org Structured Data

eBuilder includes reusable schema templates in templates/includes/seo/:

- schema_product.html: Product rich snippets with pricing and reviews
- schema_article.html: Blog post structured data
- schema_itemlist.html: Product and post listing pages
- schema_organization.html: Business/organization data
- schema_webpage.html: Generic page markup
- og_meta.html: Open Graph and Twitter Card tags

13.2 Per-Page SEO

Every content type supports custom SEO:

- Products: Meta title, description, OG image
- Blog Posts: Meta title, description, keywords
- Pages: Meta title, description
- InfoPages: Meta title, description

13.3 Technical SEO

- XML Sitemap: Auto-generated at /sitemap.xml
- Robots.txt: Configurable at /robots.txt
- Canonical URLs: Prevent duplicate content
- AI Bot Support: Metadata for AI crawlers

14. Authentication System

14.1 django-allauth Integration

eBuilder uses django-allauth for authentication:

- Email-based login (no usernames)
- Email verification required
- Password reset via email
- Customizable email templates

14.2 Custom User Model

The accounts app provides a custom user model with:

- Email as primary identifier
- First name and last name fields
- No username field

14.3 Customer Dashboard

Authenticated users access their dashboard at /accounts/dashboard/:

- Order History: List of all purchases
- Downloads: Access purchased files
- Wishlist: Saved products
- Profile: Account settings

14.4 Admin User Creation

Use the custom management command for verified admin accounts:

```
docker compose exec web python manage.py createsuperuser_verified
```

This creates the user AND verifies their email address automatically.

15. Theming & Customization

15.1 CSS Custom Properties

Colors are controlled via CSS variables in static/css/base.css for self-hosted and also through the Site Settings page:

```
:root {
 --color-primary: #1e3a8a;
 --color-secondary: #3b82f6;
 --color-accent: #60a5fa;
 }
```

15.2 Tailwind CSS 4

The frontend uses Tailwind CSS 4 with:

- Utility-first styling
- CSS variable integration
- Responsive design utilities
- Dark mode support

15.3 Template Customization

Templates are organized by app:

- templates/: Global templates and includes
- shop/templates/: Shop-specific templates
- blog/templates/: Blog templates
- pages/templates/: Page and section templates
- account/templates/: Authentication templates

15.4 Adminita Theme

The admin interface uses Adminita:

- Modern, clean design
- Dark mode toggle
- Better mobile responsiveness
- Collapsible navigation

https://adminita.todiane.com/

16. Docker Deployment (self-hosted version)

16.1 Container Architecture

The default docker-compose.yml runs a single web container:

- web: Django application with Gunicorn
- Volumes: Persistent storage for database, media, logs
- Port 8000: Exposed for reverse proxy

16.2 Volume Mounts

Critical distinction between baked-in and mounted content:

- Baked-in (in container): Templates, static files, code
- Mounted (persistent): Database, media uploads, logs

Template changes require container rebuild. Data persists across deployments.

16.3 Common Docker Commands

```
# Start containers
docker compose up -d

# View logs
docker compose logs -f web

# Run management commands
docker compose exec web python manage.py migrate
```

```
docker compose exec web python manage.py collectstatic

# Rebuild after code changes
docker compose build
docker compose up -d

# Stop containers
docker compose down
```

17. Maintenance & Updates

17.1 Backup Strategy

Essential backup commands:

```
# Database backup (JSON)
docker compose exec web python manage.py dumpdata > backup.json

# Media files backup
tar -czf media-backup.tar.gz data/media/

# Full data directory backup
tar -czf ebuilder-backup.tar.gz data/
```

17.2 Restore Procedures

```
# Restore database
docker compose exec web python manage.py loaddata backup.json

# Restore media
tar -xzf media-backup.tar.gz
```

17.3 Updates

To update to the latest version:

Download
Download from the dashboard area (you have up to 5 downloads).
Backup your existing site before beginning

Rebuild container
docker compose build

Apply migrations
docker compose exec web python manage.py migrate

Collect static files
docker compose exec web python manage.py collectstatic --no-input

Restart
docker compose restart

18. Security Considerations

18.1 Built-in Security

- CSRF Protection: All forms protected
- SQL Injection: Django ORM prevents injection
- XSS Prevention: Template auto-escaping
- Secure Cookies: HTTPS-only session cookies
- Password Hashing: PBKDF2 with SHA256
- HoneyPot (to stop bots signing up) and Disposable Email signup protection

18.2 File Download Security

- Files served through authenticated views only
- Direct URL access blocked
- Download limits enforced per purchase
- Files stored outside web root

18.3 Stripe Security

- Webhook signature verification
- No card data touches your server
- PaymentIntent API for proper flow

18.4 Production Checklist

- Set DEBUG=False
- Use strong SECRET_KEY
- Configure ALLOWED_HOSTS
- Enable HTTPS via reverse proxy
- Set up email for password resets
- Configure Stripe webhook endpoint

19. Troubleshooting

19.1 Common Issues

Port 8000 Already in Use

Edit docker-compose.yml and change ports from "8000:8000" to "8001:8000" or another available port.

Permission Errors on Media Files

```
sudo chown -R 1000:1000 data/
```

Static Files Not Loading

```
docker compose exec web python manage.py collectstatic --no-input
```

Admin Styling Broken

Verify Adminita is installed:

```
docker compose exec web pip list | grep adminita
```

Email Not Sending

Check EMAIL_* settings in .env and verify SMTP credentials.

19.2 Debugging

- Check logs: docker compose logs -f web
- Django shell: docker compose exec web python manage.py shell
- Database: docker compose exec web python manage.py dbshell
- Error logs: data/logs/django-error.log

20. Roadmap & Future Development

20.1 Current Version (v1.0)

- Complete digital product shop
- Blog and content management
- Docker-first distribution
- Stripe integration
- Full SEO implementation
- Multi-currency support

20.2 Planned Features

- Subscription products
- Newsletter integration
- Advanced analytics dashboard
- Coupon/discount codes
- Multiple payment gateways

Credits & Acknowledgments

eBuilder is built with these excellent open source projects:

- Django: https://www.djangoproject.com/
- Tailwind CSS: https://tailwindcss.com/
- Stripe: https://stripe.com/
- Adminita: https://github.com/djangify/adminita
- Alpine.js: https://alpinejs.dev/
- HTMX: https://htmx.org/
- TinyMCE: https://www.tiny.cloud/
- django-allauth: https://django-allauth.readthedocs.io/

Own your store. Own your data. Own your future.

https://www.djangify.com

Created by Diane Corriette | https://github.com/todiane