OneFantasticShop

Affiliate Manager

Developer Documentation

This document describes the system architecture, database schema, API routes, payment flows, and commission logic for the OneFantasticShop Affiliate Manager platform. It is intended for developers who need to maintain, extend, or integrate with the system.

Stack: Node.js 24 · TypeScript 5.9 · Express 5 · PostgreSQL · React + Vite
Auth: JWT (localStorage) with role-based middleware
Payments: Stripe (credit card) + PayPal (subscriptions)
Version Date: 13 July 2026

Table of Contents

1. System Overview

OneFantasticShop Affiliate Manager is a two-level affiliate commission platform. It allows an admin to manage a network of Super-Freelancers and Freelancers who earn commissions by recruiting paying subscribers.

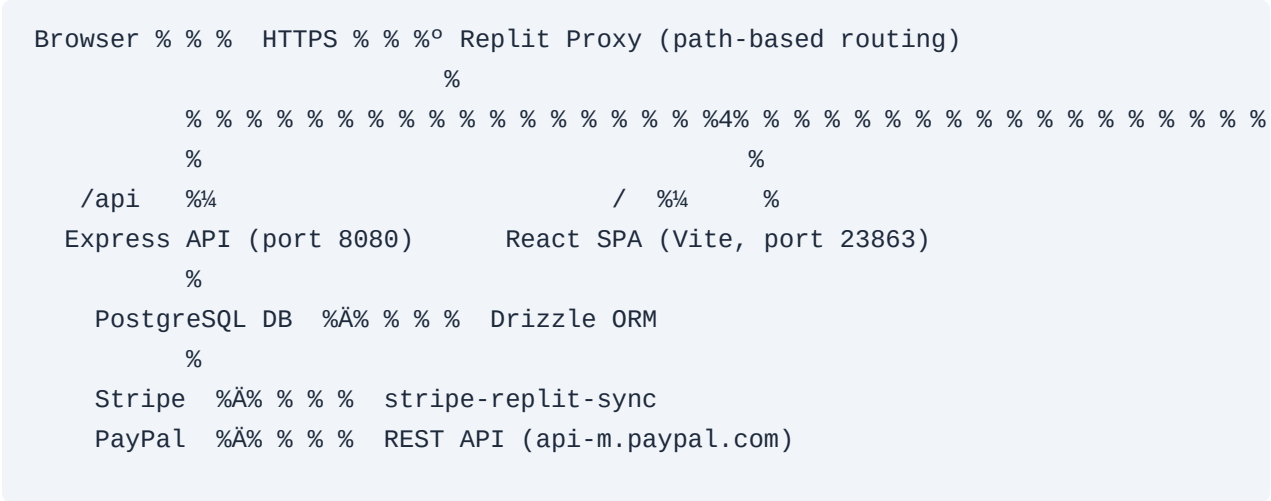
Business Model

- **Subscription price:** $4.99/month (30-day free trial on first payment)
- **Freelancer commission:** 25% of $4.99 = $1.25 per active subscriber per month
- **Super-Freelancer override:** 5% of $4.99 = $0.25 per subscriber in their team per month
- Admin creates all accounts — no self-registration for staff roles

User Journey (Subscriber)

A visitor clicks a referral link (e.g. /subscribe?ref=SUB-XXXX), fills in name/email/password, and chooses to pay via Stripe or PayPal. They enjoy a 30-day free trial, then are billed $4.99/month automatically. Commissions are generated on each successful payment.

High-Level Architecture



2. Repository Structure

The project is a pnpm workspace monorepo with the following layout:

```
artifacts/
  app/                    React + Vite frontend
  api-server/             Express 5 API
  mockup-sandbox/         Component preview server (dev only)

lib/
  api-spec/openapi.yaml   OpenAPI spec (source of truth for all contracts)
  api-client-react/       Generated React Query hooks (via Orval)
  api-zod/                Generated Zod request/response schemas
  db/                     Drizzle ORM schema + migrations
```

Key Commands

- **pnpm --filter @workspace/api-server run dev:** Start API server
- **pnpm --filter @workspace/app run dev:** Start frontend
- **pnpm run typecheck:** Full typecheck across all packages
- **pnpm --filter @workspace/api-spec run codegen:** Regenerate API hooks + Zod schemas
- **pnpm --filter @workspace/db run push:** Push DB schema changes (dev only)
- **pnpm --filter @workspace/scripts run seed-admin:** Seed default admin account

3. Roles & Permissions

There are four distinct actor types in the system, each with their own JWT token scope and dashboard.

Admin

`JWT role: admin`

- Create/manage users (freelancers, super-freelancers)
- View platform-wide stats and all subscriptions
- Approve, block, or reject user accounts
- Export commissions to CSV
- Mark commissions as paid / create payouts

Super-Freelancer

`JWT role: super_freelancer`

- View their recruited freelancers list
- See override commissions (5% per team subscriber payment)
- View their referral/recruiter code
- Check payout history

Freelancer

`JWT role: freelancer`

- View their referral link and code
- See active subscriber count and monthly earnings
- View commission history per subscriber
- Check payout history

Subscriber

`Separate subscriber JWT (not user JWT)`

- View their subscription status
- See referral history (who they referred)
- View Stripe payment history
- NOT managed via the users table — stored in subscribers table

4. Database Schema

All tables are managed via Drizzle ORM. Schema files live in lib/db/src/schema/. Run 'pnpm --filter @workspace/db run push' to apply changes in development.

users

Staff accounts: admins, freelancers, super-freelancers

- id (serial PK)
- email (text, unique)
- passwordHash (text)
- **role (enum:** admin | freelancer | super_freelancer)
- **status (enum:** pending | approved | blocked | rejected)
- referralCode (text, unique) — used by freelancers
- recruiterCode (text, unique) — used by super-freelancers
- referredByUserId (integer, FK !' users.id) — for freelancers recruited by a super-freelancer
- stripeCustomerId (text)
- name (text)

subscribers

Paying members (separate from staff users)

- id (serial PK)
- email (text, unique)
- passwordHash (text)
- name (text)
- referralCode (text, unique) — subscriber's own referral code
- referralCodeUsed (text) — code they signed up with
- **referredByType (enum:** none | subscriber)
- referredBySubscriberId (integer, FK !' subscribers.id)
- **subscriptionStatus (enum:** pending | active | cancelled | paused)
- stripeCustomerId (text)
- paypalSubscriptionId (text) — set for PayPal subscribers

subscriptions

Stripe subscription records (one per subscriber, Stripe only)

- id (serial PK)
- stripeSubscriptionId (text, unique, NOT NULL)
- subscriberId (integer, FK !' subscribers.id)
- **status (enum:** active | cancelled | past_due | trialing | ...)
- amount (numeric, default 4.99)
- stripePriceId (text)
- currentPeriodStart / currentPeriodEnd (timestamp)
- cancelledAt (timestamp)

subscriber_commissions

Commission records generated on each successful payment

• id (serial PK)
• subscriberId (integer) — who earns this commission
• sourceSubscriberId (integer) — whose payment triggered it
• subscriptionId (integer, nullable) — Stripe subscription row ID (null for PayPal)
• amount (numeric) — 1.25 for direct, 0.25 for override
• **type (enum:** direct | override)
• **status (enum:** pending | paid)
• periodStart / periodEnd (timestamp)
• paidAt (timestamp)

payouts

Batch payout records created by admin

• id (serial PK)
• userId (integer, nullable) — target staff user
• subscriberId (integer, nullable) — target subscriber (for subscriber-to-subscriber referrals)
• amount (numeric) — total payout
• month (text) — e.g. '2026-07'
• **status (enum:** pending | completed)
• paidAt (timestamp)

5. Authentication

Staff (Admin / Freelancer / Super-Freelancer)

POST /api/auth/login with { email, password } returns a JWT. The token is stored in localStorage on the frontend and sent via Authorization: Bearer <token> on every subsequent request.

```
// Token payload
{
  "sub": 1,          // user ID
  "role": "admin",   // admin | freelancer | super_freelancer
  "iat": ...,
  "exp": ...
}
```

The server validates the token with requireAuth middleware and checks the role with requireRole('admin') etc.

Subscribers

POST /api/subscribers/login returns a separate subscriber JWT. Subscriber tokens carry role: 'subscriber' and sub = subscriberId. Subscriber-protected routes use a separate requireSubscriberAuth middleware.

Password Hashing

All passwords are hashed with bcryptjs (10 rounds) before storage. The hash is stored in passwordHash and compared on login.

Default Admin Credentials

```
Email:    admin@onefantasticshop.com
Password: Admin123!

Run: pnpm --filter @workspace/scripts run seed-admin
```

6. API Routes

All routes are prefixed with /api. The API runs on port 8080 in development and is proxied through the Replit shared proxy.

Auth

POST	`/api/auth/login`	
Authenticate !' returns JWT		
POST	`/api/auth/logout`	
Invalidate session		
GET	`/api/auth/me`	Auth
Current user profile		
POST	`/api/auth/change-password`	Auth
Change password		

Admin

GET	`/api/admin/stats`	Admin
Platform-wide stats		
GET	`/api/admin/users`	Admin
List all staff users		
POST	`/api/admin/users`	Admin
Create a new user		
GET	`/api/admin/users/:id`	Admin
Get user details		
PATCH	`/api/admin/users/:id`	Admin
Update user		
PATCH	`/api/admin/users/:id/status`	Admin
Update user status		
GET	`/api/admin/subscribers`	Admin
List all subscribers		
GET	`/api/admin/subscriptions`	Admin
List all subscriptions		
GET	`/api/admin/commissions`	Admin
List all commissions		
PATCH	`/api/admin/commissions/:id/pay`	Admin
Mark commission paid		
GET	`/api/admin/commissions/export`	Admin
Export commissions CSV		
GET	`/api/admin/payouts`	Admin
List payouts		
GET	`/api/admin/payouts/unpaid`	Admin
Unpaid summary by month		
POST	`/api/admin/payouts/mark-paid`	Admin
Batch mark commissions paid		

Freelancer

GET	`/api/freelancer/dashboard`	Freelancer
	Freelancer dashboard stats	
GET	`/api/freelancer/profile`	Freelancer
	Freelancer profile	
GET	`/api/freelancer/subscribers`	Freelancer
	Referred subscribers list	
GET	`/api/freelancer/commissions`	Freelancer
	Commission history	
GET	`/api/freelancer/payouts`	Freelancer
	Payout history	

Super-Freelancer

GET	`/api/super-freelancer/dashboard`	Super-FL
	Super-freelancer dashboard	
GET	`/api/super-freelancer/profile`	Super-FL
	Profile info	
GET	`/api/super-freelancer/freelancers`	Super-FL
	Recruited freelancers	
GET	`/api/super-freelancer/commissions`	Super-FL
	Override commissions	
GET	`/api/super-freelancer/payouts`	Super-FL
	Payout history	

Subscribers

POST	`/api/subscribers/signup`	Public
	Register subscriber + referral tracking	
POST	`/api/subscribers/login`	Public
	Subscriber login !' JWT	
GET	`/api/subscribers/me`	Sub-Auth
	Subscriber profile + status	
GET	`/api/subscribers/me/referrals`	Sub-Auth
	Referral + commission history	
GET	`/api/subscribers/me/payments`	Sub-Auth
	Stripe invoice history	

Checkout

POST	`/api/checkout/session`	Public
	Create Stripe Checkout session	
POST	`/api/checkout/paypal-session`	Public
	Create PayPal subscription + return approval URL	
POST	`/api/checkout/paypal/capture`	Public
	Verify + activate PayPal sub after approval	

Webhooks

POST	`/api/stripe/webhook`	Stripe-Sig
	Stripe event receiver (raw body, before express.json)	
POST	`/api/webhooks/paypal`	Public
	PayPal event receiver (JSON)	

Other

POST	`/api/referrals/click`	Public
	Track referral link click	
GET	`/api/health`	Public
	Health check !' { ok: true }	

7. Payment Flows — Stripe

Stripe is integrated via the stripe-replit-sync Replit integration. The Stripe API key is fetched from the integration at startup — no manual STRIPE_SECRET_KEY env var is needed.

Checkout Flow

```
1. Frontend: POST /api/checkout/session  { subscriberId }
   !“
2. API: find/create Stripe Customer for subscriber
   !“
3. API: stripe.checkout.sessions.create({
     mode: "subscription",
     trial_period_days: 30,
     metadata: { subscriberId }
   })
   !“
4. Return { url } !’ frontend redirects to Stripe-hosted checkout
   !“
5. User pays !’ Stripe redirects to /subscribe/success?session_id=...
   !“
6. Stripe fires checkout.session.completed webhook
   !“
7. API: create subscriptions row, set subscriber.subscriptionStatus = 'active'
```

Recurring Payment Flow

```
Monthly billing cycle:
  Stripe fires invoice.payment_succeeded
  !’ webhookHandlers.handlePaymentSucceeded(invoice)
  !’ look up subscription by stripeSubscriptionId
  !’ look up subscriber
  !’ if subscriber.referredByType === 'subscriber':
      insert direct commission ($1.25) for referrer
      if referrer was also referred:
        insert override commission ($0.25) for referrer's referrer
```

Important: Raw Body Requirement

The Stripe webhook route is registered BEFORE express.json() in app.ts because Stripe's signature verification requires the raw request body. Adding the webhook route after express.json() will break signature verification.

```
// app.ts — ORDER MATTERS
app.post("/api/stripe/webhook", express.raw({ type: "application/json" }), handler)
app.use(express.json())  // !• must come after stripe webhook
```

8. Payment Flows — PayPal

PayPal uses their Subscriptions v1 API. Credentials (PAYPAL_CLIENT_ID, PAYPAL_CLIENT_SECRET) are stored as Replit Secrets. Set PAYPAL_MODE=sandbox to use the sandbox environment.

Checkout Flow

```
1. Frontend: POST /api/checkout/paypal-session  { subscriberId }
   !“
2. API: getPayPalAccessToken() — cached, auto-refreshed
   !“
3. API: getOrCreatePayPalProduct() — creates product once, cached
   !“
```

Recurring Payment Webhook

```
PayPal fires PAYMENT.SALE.COMPLETED on each billing:
  !’ resource.billing_agreement_id = PayPal subscription ID
  !’ look up subscriber by paypalSubscriptionId
  !’ generate commissions (same logic as Stripe)

Other events handled:
  BILLING.SUBSCRIPTION.ACTIVATED  !’ set status = active (backup for capture)
  BILLING.SUBSCRIPTION.CANCELLED
  BILLING.SUBSCRIPTION.SUSPENDED  !’ set status = cancelled
  BILLING.SUBSCRIPTION.EXPIRED
```

Webhook Setup (PayPal Dashboard)

Register the webhook URL in your PayPal Developer Dashboard !’ My Apps !’ Webhooks:

```
URL: https://<your-domain>/api/webhooks/paypal

Events to subscribe:
  BILLING.SUBSCRIPTION.ACTIVATED
  BILLING.SUBSCRIPTION.CANCELLED
  BILLING.SUBSCRIPTION.EXPIRED
  BILLING.SUBSCRIPTION.SUSPENDED
  PAYMENT.SALE.COMPLETED
```

9. Commission Logic

Commissions are generated on every successful payment event — both Stripe invoices and PayPal sales. They are written to the subscriber_commissions table.

Rates

```
Subscription amount:    $4.99 / month
Direct commission:      25% × $4.99 = $1.25  (type: 'direct')
Override commission:     5% × $4.99 = $0.25  (type: 'override')
```

Two-Level Referral Tree

```
Super-Freelancer (A)
  % % %  Freelancer (B)    !• referred by A (referredByUserId = A.id)
        % % %  Subscriber (C)  !• signed up with B's referral code

When C makes a payment:
  !’ B earns $1.25  (direct commission, type='direct')
  !’ A earns $0.25  (override commission, type='override')

Subscriber-to-Subscriber referral:
  Subscriber (D) refers Subscriber (E) using their referral code.
  When E makes a payment:
  !’ D earns $1.25  (direct commission)
  !’ D's referrer (if any) earns $0.25  (override)
```

Commission Status Lifecycle

- **pending:** generated on payment, awaiting admin payout
- **paid:** admin has marked it paid via POST /api/admin/payouts/mark-paid

10. Webhook Handlers

Stripe Events (artifacts/api-server/src/webhookHandlers.ts)

- **checkout.session.completed:** Creates subscription row, activates subscriber
- **invoice.payment_succeeded:** Generates commissions on recurring billing
- **customer.subscription.deleted:** Sets subscription status to cancelled
- **customer.subscription.updated:** Syncs subscription status changes

PayPal Events (artifacts/api-server/src/routes/paypalWebhook.ts)

- **BILLING.SUBSCRIPTION.ACTIVATED:** Sets subscriber.subscriptionStatus = active
- **PAYMENT.SALE.COMPLETED:** Generates direct + override commissions

- **BILLING.SUBSCRIPTION.CANCELLED:** Sets status = cancelled
- **BILLING.SUBSCRIPTION.SUSPENDED:** Sets status = cancelled
- **BILLING.SUBSCRIPTION.EXPIRED:** Sets status = cancelled

11. Environment Variables

Secrets are managed in Replit Secrets (not .env files). Non-secret config is stored as Replit Environment Variables.

Required

```
DATABASE_URL         PostgreSQL connection string (auto-managed by Replit)
SESSION_SECRET       JWT signing secret
PAYPAL_CLIENT_ID     PayPal app client ID
PAYPAL_CLIENT_SECRET PayPal app client secret
```

Optional

```
PAYPAL_MODE          Set to 'sandbox' to use PayPal sandbox API
                     Omit or set to anything else for live

PORT                 Port for the API server (set by Replit workflow config)
```

Stripe

Stripe credentials are fetched automatically via the stripe-replit-sync Replit integration. No manual STRIPE_SECRET_KEY is needed — just connec

t the Stripe integration from the Integrations tab.

12. Running Locally

Prerequisites

- Node.js 24+
- pnpm 10+
- PostgreSQL database (or use Replit's built-in DB)

Steps

```
# 1. Install dependencies
pnpm install

# 2. Set environment variables
export DATABASE_URL=postgresql://...
export SESSION_SECRET=your-secret
export PAYPAL_CLIENT_ID=...
export PAYPAL_CLIENT_SECRET=...
export PAYPAL_MODE=sandbox   # optional

# 3. Push schema to DB
pnpm --filter @workspace/db run push

# 4. Seed admin user
pnpm --filter @workspace/scripts run seed-admin

# 5. Start API server
PORT=8080 pnpm --filter @workspace/api-server run dev

# 6. Start frontend (separate terminal)
```

After Changing the OpenAPI Spec

```
# Regenerate React Query hooks and Zod schemas
pnpm --filter @workspace/api-spec run codegen

# Then typecheck
pnpm run typecheck:libs
```

13. Deployment

The app is deployed via Replit's built-in deployment (Publish button). Production runs the same build command as development.

Checklist Before Going Live

- Stripe integration connected in Replit Integrations tab
- Stripe webhook pointing to https://<domain>/api/stripe/webhook
- PayPal live credentials set (PAYPAL_CLIENT_ID, PAYPAL_CLIENT_SECRET)
- PAYPAL_MODE env var removed (or not set to 'sandbox')
- PayPal live webhook pointing to https://<domain>/api/webhooks/paypal
- SESSION_SECRET set to a strong random value

Production Logs

Use Replit's deployment log viewer to view production server logs. The API uses structured JSON logging (pino) — filter by level, req.url, or subscriberId for debugging.