Grimforge Admin System Guide

Table of Contents

---

1. Accessing the Admin Panel

Direct URL Access

The admin panel is accessible at: **`https://your-domain.com/admin`**

- The admin system is built with **Refine.dev** framework
- It uses **Supabase** for authentication and data storage
- The main admin route is `/admin` which redirects to `/admin/products` by default

Login Requirements

- You must have an account in Supabase Auth
- Your user must be assigned the `admin` role in the `user_roles` table
- Without admin role, you'll see: "You don't have permission to access the admin panel"

---

2. Setting Up Admin Permissions

Step 1: Create Admin Role for arg@obsidianriterecords.com

First, you need to ensure the user exists in Supabase Auth and then assign admin permissions.

Option A: If the user already exists in Supabase Auth

```sql
-- Find the user ID first
SELECT id, email FROM auth.users WHERE email = 'arg@obsidianriterecords.com';

-- Insert admin role (replace USER_ID with the actual UUID from above)
INSERT INTO public.user_roles (user_id, role)
VALUES ('USER_ID_HERE', 'admin')
ON CONFLICT (user_id, role) DO NOTHING;
```

Option B: If the user doesn't exist yet

1. Go to your Supabase dashboard
2. Navigate to Authentication > Users
3. Click "Invite a user"
4. Enter: `arg@obsidianriterecords.com`
5. The user will receive an invitation email
6. After they set their password, run the SQL above to assign admin role

Step 2: Verify Admin Access

```sql
-- Check if admin role is properly assigned
SELECT
  u.email,
  ur.role,
  ur.created_at
FROM auth.users u
JOIN public.user_roles ur ON u.id = ur.user_id
WHERE u.email = 'arg@obsidianriterecords.com';
```

Available Roles

The system supports these roles:
- `super_admin` - Full system access
- `admin` - Standard admin access
- `moderator` - Limited admin access
- `support` - Customer support access

---

3. Understanding the Database Schema

Core Tables Overview

Products & Inventory

- **`products`** - Main product catalog (vinyl records, CDs, etc.)
- **`variants`** - Product variations (different formats, colors, etc.)
- **`inventory`** - Stock levels and availability tracking

Orders & Customers

- **`customers`** - Customer information and Stripe customer IDs
- **`orders`** - Order records with payment status
- **`order_items`** - Individual items within orders
- **`addresses`** - Customer shipping and billing addresses

Admin & Security

- **`user_roles`** - Admin role assignments
- **`audit_log`** - System activity tracking
- **`stock_movements`** - Inventory change history

Key Relationships

```
products (1) → (many) variants
variants (1) → (1) inventory
customers (1) → (many) orders
orders (1) → (many) order_items
variants (1) → (many) order_items
```

---

4. User Management System

How Authentication Works

1. **Supabase Auth** handles user registration/login
2. **Row Level Security (RLS)** controls data access
3. **Admin roles** are stored in `user_roles` table
4. **JWT tokens** include role information for authorization

Managing Customer Data

```sql
-- View all customers
SELECT
  c.email,
  c.first_name,
  c.last_name,
  c.stripe_customer_id,
  COUNT(o.id) as total_orders
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, c.email, c.first_name, c.last_name, c.stripe_customer_id
ORDER BY c.created_at DESC;

-- View customer order history
SELECT
  o.order_number,
  o.status,
  o.total,
  o.created_at,
  COUNT(oi.id) as item_count
FROM orders o
JOIN order_items oi ON o.id = oi.order_id
WHERE o.customer_id = 'CUSTOMER_ID_HERE'
GROUP BY o.id, o.order_number, o.status, o.total, o.created_at
ORDER BY o.created_at DESC;
```

User Data Storage

- **Customer profiles** are automatically created when orders are placed
- **Addresses** are stored separately and linked to customers
- **Order history** is maintained indefinitely for accounting
- **Personal data** can be anonymized for GDPR compliance

---

5. Payment Processing with Stripe

How Stripe Integration Works

1. Checkout Flow

```
Customer Cart → Stripe Checkout → Payment Success → Webhook → Database Update
```

2. Webhook Processing

The system listens for these Stripe events:
- `checkout.session.completed` - Payment successful
- `payment_intent.succeeded` - Payment confirmed
- `payment_intent.payment_failed` - Payment failed
- `charge.refunded` - Refund processed

3. Order Status Flow

```
pending → paid → processing → shipped → delivered
                    ↓
              cancelled/refunded
```

Managing Payments in Admin

View Payment Status

```
-- Orders with payment information
SELECT
  o.order_number,
  o.status,
  o.payment_status,
  o.total,
  o.stripe_session_id,
  o.stripe_payment_intent_id,
  o.created_at
FROM orders o
WHERE o.payment_status = 'paid'
ORDER BY o.created_at DESC;
```

Process Refunds

```
-- Mark order as refunded (after processing refund in Stripe)
UPDATE orders
SET
  status = 'refunded',
  payment_status = 'refunded',
  updated_at = NOW()
WHERE order_number = 'ORR-001234';
```

Stripe Dashboard Access

- **Live payments**: https://dashboard.stripe.com/payments
- **Test payments**: https://dashboard.stripe.com/test/payments
- **Webhooks**: https://dashboard.stripe.com/webhooks
- **Customer data**: https://dashboard.stripe.com/customers

6. Managing Products, Orders, and Customers

Product Management

Adding New Products

1. Go to `/admin/products`
2. Click “Create Product”
3. Fill in product details:
   - Name, description, price
   - Images (upload to your CDN/storage)
   - Category and tags
   - SEO-friendly slug

Managing Inventory

```sql
-- Check low stock items
SELECT
  p.name,
  v.name as variant_name,
  i.available,
  i.reorder_point
FROM products p
JOIN variants v ON p.id = v.product_id
JOIN inventory i ON v.id = i.variant_id
WHERE i.available <= i.reorder_point
ORDER BY i.available ASC;

-- Receive new stock
SELECT receive_stock(
  'VARIANT_ID_HERE'::UUID,
  50, -- quantity
  'Received shipment from supplier',
  'ADMIN_USER_ID'::UUID
);
```

Order Management

Order Processing Workflow

1. **New Orders** - Review and validate
2. **Payment Confirmation** - Verify Stripe payment
3. **Fulfillment** - Pick, pack, ship
4. **Tracking** - Update with shipping info
5. **Completion** - Mark as delivered

Common Order Operations

```sql
-- Update order status
UPDATE orders
SET status = 'processing', updated_at = NOW()
WHERE order_number = 'ORR-001234';

-- View order details
SELECT
  o.order_number,
  o.status,
  o.total,
  c.email,
  c.first_name,
  c.last_name,
  oi.product_name,
  oi.variant_name,
  oi.quantity,
  oi.price
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN order_items oi ON o.id = oi.order_id
WHERE o.order_number = 'ORR-001234';
```

Customer Service

Common Customer Queries

```sql
-- Find customer by email
SELECT * FROM customers WHERE email ILIKE '%customer@email.com%';

-- Customer's recent orders
SELECT
  order_number,
  status,
  total,
  created_at
FROM orders
WHERE customer_id = 'CUSTOMER_ID'
ORDER BY created_at DESC
LIMIT 10;

-- Update customer information
UPDATE customers
SET
  first_name = 'New Name',
  phone = '+1234567890',
  updated_at = NOW()
WHERE email = 'customer@email.com';
```

---

7. Database Setup Scripts

Grant Admin Access Script

Save this as `grant_admin.sql`:

```sql
-- Grant admin access to a user by email
-- Usage: Replace 'user@email.com' with the actual email

DO $$
DECLARE
    user_uuid UUID;
BEGIN
    -- Get user ID from email
    SELECT id INTO user_uuid
    FROM auth.users
    WHERE email = 'arg@obsidianriterecords.com';

    IF user_uuid IS NULL THEN
        RAISE EXCEPTION 'User with email arg@obsidianriterecords.com not found';
    END IF;

    -- Insert admin role
    INSERT INTO public.user_roles (user_id, role)
    VALUES (user_uuid, 'admin')
    ON CONFLICT (user_id, role) DO NOTHING;

    RAISE NOTICE 'Admin role granted to user: %', user_uuid;
END $$;
```

Database Health Check Script

Save this as `health_check.sql` :

```sql
-- Database health check queries
SELECT 'Products' as table_name, COUNT(*) as count FROM products
UNION ALL
SELECT 'Variants', COUNT(*) FROM variants
UNION ALL
SELECT 'Orders', COUNT(*) FROM orders
UNION ALL
SELECT 'Customers', COUNT(*) FROM customers
UNION ALL
SELECT 'Admin Users', COUNT(*) FROM user_roles WHERE role = 'admin';

-- Check RLS policies
SELECT
  schemaname,
  tablename,
  policyname,
  permissive,
  roles,
  cmd,
  qual
FROM pg_policies
WHERE schemaname = 'public'
ORDER BY tablename, policyname;
```

Inventory Management Functions

```sql
-- Function to adjust inventory (for corrections)
CREATE OR REPLACE FUNCTION adjust_inventory(
  p_variant_id UUID,
  p_new_quantity INTEGER,
  p_reason TEXT DEFAULT 'Manual adjustment',
  p_user_id UUID DEFAULT NULL
) RETURNS BOOLEAN AS $$
DECLARE
  v_current_quantity INTEGER;
  v_adjustment INTEGER;
BEGIN
  -- Get current quantity
  SELECT on_hand INTO v_current_quantity
  FROM inventory
  WHERE variant_id = p_variant_id;

  IF v_current_quantity IS NULL THEN
    RAISE EXCEPTION 'Variant not found in inventory';
  END IF;

  v_adjustment := p_new_quantity - v_current_quantity;

  -- Update inventory
  UPDATE inventory
  SET
    on_hand = p_new_quantity,
    updated_at = NOW()
  WHERE variant_id = p_variant_id;

  -- Record stock movement
  INSERT INTO stock_movements (
    variant_id,
    quantity,
    movement_type,
    notes,
    created_by
  ) VALUES (
    p_variant_id,
    v_adjustment,
    'adjustment',
    p_reason,
    p_user_id
  );

  RETURN TRUE;
END;
$$ LANGUAGE plpgsql;
```

---

8. Troubleshooting

Common Issues

"You don't have permission to access the admin panel"

Solution:

1. Verify user exists in Supabase Auth

2. Check if admin role is assigned:

```sql
   SELECT * FROM user_roles WHERE user_id = (
     SELECT id FROM auth.users WHERE email = 'your@email.com'
   );
```

3. If no role found, run the grant admin script above

Admin panel shows “Loading…” indefinitely

Possible causes:
1. **Environment variables missing** - Check Netlify environment variables
2. **Database connection issues** - Verify Supabase URL and keys
3. **RLS policies blocking access** - Check if policies allow admin access

Orders not updating after Stripe payment

Check:
1. **Webhook endpoint** is configured in Stripe dashboard
2. **Webhook secret** matches environment variable
3. **Webhook events** include `checkout.session.completed`
4. **Database permissions** allow webhook to write data

Inventory not updating

Debug steps:
1. Check if `decrement_inventory` function exists
2. Verify inventory table has correct data
3. Look for errors in audit_log table
4. Test inventory functions manually

Useful Queries for Debugging

```sql
-- Recent admin activity
SELECT
  event_type,
  resource_type,
  user_id,
  metadata,
  created_at
FROM audit_log
WHERE user_id IN (
  SELECT user_id FROM user_roles WHERE role = 'admin'
)
ORDER BY created_at DESC
LIMIT 20;

-- Failed webhook events
SELECT * FROM audit_log
WHERE event_type LIKE '%stripe%'
AND metadata->>'error' IS NOT NULL
ORDER BY created_at DESC;

-- Inventory discrepancies
SELECT
  p.name,
  v.name as variant,
  i.on_hand,
  i.allocated,
  i.available,
  (SELECT SUM(quantity) FROM stock_movements sm WHERE sm.variant_id = v.id) as total_mo
vements
FROM products p
JOIN variants v ON p.id = v.product_id
JOIN inventory i ON v.id = i.variant_id
WHERE i.available < 0 OR i.on_hand < i.allocated;
```

Getting Help

1. **Database Issues**: Check Supabase dashboard logs
2. **Payment Issues**: Check Stripe dashboard events
3. **Admin Panel Issues**: Check browser console for JavaScript errors
4. **Performance Issues**: Review database query performance in Supabase

Emergency Contacts

- **Supabase Support**: https://supabase.com/support
- **Stripe Support**: https://support.stripe.com
- **Refine.dev Docs**: https://refine.dev/docs

Quick Start Checklist

- [ ] Verify Netlify environment variables are set
- [ ] Access admin panel at `/admin`
- [ ] Grant admin role to `arg@obsidianriterecords.com`
- [ ] Test login to admin panel

- [ ] Verify Stripe webhook is working
- [ ] Check product and inventory data
- [ ] Test order processing flow
- [ ] Set up monitoring and alerts

This guide covers the essential aspects of managing your Grimforge admin system. Keep this document updated as you make changes to the system.