Product Variant & View Architecture

Executive Summary

This document outlines a comprehensive architecture for handling product variants (color variations) and product views (different perspectives like front/back) in our promotional print e-commerce platform. The solution enables:

1. **Multiple color variants** of the same product with separate template images
2. **Multiple views** of products (front, back, sides) with view-specific template images
3. **Reusable print templates** that work across all color variants of a specific view
4. **Scalable storage organization** in Supabase with intuitive folder structure
5. **Efficient loading** in the designer with minimal database queries

---

Current System Analysis

Database Schema

product_templates table:

```
CREATE TABLE product_templates (
  id UUID PRIMARY KEY,
  product_key TEXT UNIQUE NOT NULL,
  name TEXT NOT NULL,
  template_url TEXT,              -- Single template URL
  colors JSONB DEFAULT '[]',
  base_price DECIMAL(10, 2),
  created_by UUID,
  created_at TIMESTAMP,
  updated_at TIMESTAMP
);
```

print_areas table:

```
CREATE TABLE print_areas (
  id UUID PRIMARY KEY,
  product_template_id UUID REFERENCES product_templates(id),
  area_key TEXT NOT NULL,
  name TEXT NOT NULL,
  x INTEGER, y INTEGER,
  width INTEGER, height INTEGER,
  max_width INTEGER, max_height INTEGER,
  shape VARCHAR(20) DEFAULT 'rectangle',
  created_at TIMESTAMP,
  updated_at TIMESTAMP,
  UNIQUE(product_template_id, area_key)
);
```

Current Storage Structure

```
/public/templates/
  ├── bag/
  │   └── template.png          (single template for all variants)
  ├── tshirt/
  │   └── template.png
  └── mug/
      └── template.png
```

Current Limitations

1. ❌ **No color variant support** - One template serves all colors
2. ❌ **No view separation** - Front/back share the same template
3. ❌ **No hierarchical organization** - Flat file structure
4. ❌ **Inflexible print areas** - Print areas tied to product, not view
5. ❌ **Manual image swapping** - No automatic loading of view-specific images

---

Proposed Architecture

Key Concepts

1. Base Products

The fundamental product type (e.g., “Tote Bag”, “T-Shirt”)
- Has a unique `product_key` (e.g., `5oz-cotton-bag`)
- Has base attributes: name, base price, available colors
- Serves as the parent for all variants and views

2. Product Views

Different perspectives or sides of the product (e.g., “front”, “back”, “sleeve”)
- Each view has its own print area configuration
- Views can have completely different print area layouts
- Example: Bag front (400x400) vs bag back (400x400) vs small logo (200x100)

3. Color Variants

Different colors of the same product view
- Each color has its own template image
- All colors of the same view share the same print area configuration
- Example: Red bag front, blue bag front, black bag front all use “bag front” print areas

4. Print Templates

Reusable print area configurations
- Defined once per product view
- Applied to all color variants of that view
- Example: One “bag-front” print template works for red, blue, and black bags

Visual Hierarchy

```
Product: "5oz Cotton Bag"
├── View: "front"
│   ├── Print Template: "bag-front" (400x400 @ 200,200)
│   ├── Variant: red → template image: /bag/front/red.png
│   ├── Variant: blue → template image: /bag/front/blue.png
│   └── Variant: black → template image: /bag/front/black.png
│
├── View: "back"
│   ├── Print Template: "bag-back" (400x400 @ 200,200)
│   ├── Variant: red → template image: /bag/back/red.png
│   ├── Variant: blue → template image: /bag/back/blue.png
│   └── Variant: black → template image: /bag/back/black.png
│
└── View: "small-logo"
    ├── Print Template: "bag-logo" (200x100 @ 300,150)
    ├── Variant: red → template image: /bag/logo/red.png
    ├── Variant: blue → template image: /bag/logo/blue.png
    └── Variant: black → template image: /bag/logo/black.png
```

Database Schema Design

Proposed Schema Changes

Option 1: Extended Current Schema (Recommended for MVP)

Keep existing tables, extend product_key convention:

```
product_key format: {base_product}#{view}
Examples:
  - "5oz-cotton-bag#front"
  - "5oz-cotton-bag#back"
  - "tshirt#front"
  - "tshirt#back"
```

template_url format:

```
Supabase Storage path: {base_product}/{view}/{color}.png
or
Supabase Storage path: {base_product}/{view}/template.png (for reference)
```

Advantages:
- ✅ Minimal schema changes
- ✅ Works with existing code
- ✅ Easy to implement immediately

Changes needed:

```sql
-- Add columns to track base product and view
ALTER TABLE product_templates
  ADD COLUMN base_product_key TEXT,
  ADD COLUMN view_name TEXT,
  ADD COLUMN is_base_template BOOLEAN DEFAULT false;

-- Add index for efficient queries
CREATE INDEX idx_base_product ON product_templates(base_product_key);
CREATE INDEX idx_view_name ON product_templates(view_name);

-- Update unique constraint to allow multiple views
ALTER TABLE product_templates
  DROP CONSTRAINT product_templates_product_key_key;

ALTER TABLE product_templates
  ADD CONSTRAINT unique_product_view
  UNIQUE(base_product_key, view_name);
```

Option 2: New Normalized Schema (Recommended for Long-term)

Create separate tables for better normalization:

```sql
-- Base products table
CREATE TABLE base_products (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  product_key TEXT UNIQUE NOT NULL,  -- e.g., "5oz-cotton-bag"
  name TEXT NOT NULL,
  description TEXT,
  base_price DECIMAL(10, 2),
  available_colors JSONB DEFAULT '[]',  -- ["#ff0000", "#0000ff"]
  category TEXT,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Product views table
CREATE TABLE product_views (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  base_product_id UUID REFERENCES base_products(id) ON DELETE CASCADE,
  view_key TEXT NOT NULL,            -- e.g., "front", "back", "sleeve"
  view_name TEXT NOT NULL,           -- e.g., "Front", "Back", "Left Sleeve"
  template_reference TEXT,           -- Reference template path (without color)
  sort_order INTEGER DEFAULT 0,      -- Display order in UI
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  UNIQUE(base_product_id, view_key)
);

-- View templates table (stores actual image URLs per color variant)
CREATE TABLE view_templates (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  product_view_id UUID REFERENCES product_views(id) ON DELETE CASCADE,
  color_hex TEXT NOT NULL,           -- e.g., "#ff0000"
  color_name TEXT,                   -- e.g., "Red", "Blue"
  template_url TEXT NOT NULL,        -- Full Supabase storage URL
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  UNIQUE(product_view_id, color_hex)
);

-- Print areas (updated to reference product_views instead)
CREATE TABLE print_areas (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  product_view_id UUID REFERENCES product_views(id) ON DELETE CASCADE,
  area_key TEXT NOT NULL,
  name TEXT NOT NULL,
  x INTEGER NOT NULL DEFAULT 0,
  y INTEGER NOT NULL DEFAULT 0,
  width INTEGER NOT NULL DEFAULT 100,
  height INTEGER NOT NULL DEFAULT 100,
  max_width INTEGER NOT NULL DEFAULT 100,
  max_height INTEGER NOT NULL DEFAULT 100,
  shape VARCHAR(20) DEFAULT 'rectangle',
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  UNIQUE(product_view_id, area_key)
);

-- Indexes
CREATE INDEX idx_base_products_key ON base_products(product_key);
CREATE INDEX idx_product_views_base ON product_views(base_product_id);
CREATE INDEX idx_view_templates_view ON view_templates(product_view_id);
CREATE INDEX idx_print_areas_view ON print_areas(product_view_id);
```

Example Data:

```
-- Base product
INSERT INTO base_products (product_key, name, base_price, available_colors)
VALUES (
  '5oz-cotton-bag',
  '5oz Cotton Bag',
  18.99,
  '["#f5deb3", "#ff0000", "#0000ff", "#000000"]'::jsonb
);

-- Product views
WITH bag_id AS (SELECT id FROM base_products WHERE product_key = '5oz-cotton-bag')
INSERT INTO product_views (base_product_id, view_key, view_name, sort_order)
VALUES
  ((SELECT id FROM bag_id), 'front', 'Front', 1),
  ((SELECT id FROM bag_id), 'back', 'Back', 2),
  ((SELECT id FROM bag_id), 'small-logo', 'Small Front Logo', 3);

-- View templates (one per color per view)
WITH front_view AS (
  SELECT pv.id
  FROM product_views pv
  JOIN base_products bp ON pv.base_product_id = bp.id
  WHERE bp.product_key = '5oz-cotton-bag' AND pv.view_key = 'front'
)
INSERT INTO view_templates (product_view_id, color_hex, color_name, template_url)
VALUES
  ((SELECT id FROM front_view), '#f5deb3', 'Beige', 'https://cdn.4imprint.com/prod/
700/520638.jpg'),
  ((SELECT id FROM front_view), '#ff0000', 'Red', 'https://lseshop.com/cdn/shop/files/
SLP19167-RD-B.jpg?v=1739275998&width=1200'),
  ((SELECT id FROM front_view), '#0000ff', 'Blue', 'https://www.printkick.com/media/
catalog/product/cache/5/image/
305x305/17f82f742ffe127f42dca9de82fb58b1/0/8/083349060da86e95724447fd8e50856cb7fb83a7_
Madras_140_gm___cotton_tote_bag_pf_royal_blue.jpg');

-- Print areas (defined once per view, applies to all colors)
WITH front_view AS (
  SELECT pv.id
  FROM product_views pv
  JOIN base_products bp ON pv.base_product_id = bp.id
  WHERE bp.product_key = '5oz-cotton-bag' AND pv.view_key = 'front'
)
INSERT INTO print_areas (product_view_id, area_key, name, x, y, width, height, max_wid
th, max_height)
VALUES
  ((SELECT id FROM front_view), 'front', 'Front', 200, 200, 400, 400, 400, 400);
```

Storage Organization

Supabase Storage Structure

Bucket: `product-templates` (already exists)

Folder Organization:

```
product-templates/
├── 5oz-cotton-bag/
│   ├── front/
│   │   ├── beige.png
│   │   ├── red.png
│   │   ├── blue.png
│   │   └── black.png
│   ├── back/
│   │   ├── beige.png
│   │   ├── red.png
│   │   ├── blue.png
│   │   └── black.png
│   └── small-logo/
│       ├── beige.png
│       ├── red.png
│       ├── blue.png
│       └── black.png
│
├── tshirt/
│   ├── front/
│   │   ├── white.png
│   │   ├── black.png
│   │   └── red.png
│   └── back/
│       ├── white.png
│       ├── black.png
│       └── red.png
│
└── mug/
    └── wrap/
        ├── white.png
        ├── black.png
        └── red.png
```

Path Convention:

```
{base_product_key}/{view_key}/{color_name_or_hex}.png

Examples:
- 5oz-cotton-bag/front/red.png
- 5oz-cotton-bag/front/beige.png
- tshirt/front/white.png
- tshirt/back/black.png
```

Naming Convention for Colors:
- Use lowercase color names: `red.png`, `blue.png`, `white.png`, `black.png`
- For non-standard colors, use hex without #: `f5deb3.png`
- Store mapping in database: `color_hex` → `color_name` → `filename`

Implementation Plan

Phase 1: Minimal Changes (Quick Win) - Option 1

Goal: Support views without major schema changes

Steps:

1. **Update product_key convention** (Code changes only)
   ```javascript
   // Current: “5oz-cotton-bag”
   // New: “5oz-cotton-bag#front”, “5oz-cotton-bag#back”

function parseProductKey(fullKey) {
const [baseKey, viewKey] = fullKey.split(‘#’);
return { baseKey, viewKey: viewKey || ‘default’ };
}
```

1. **Update template_url format** (Storage + Code)
   ```javascript
   // Current: “/templates/bag/template.png”
   // New: “5oz-cotton-bag/front/red.png”

function getTemplateUrl(baseProduct, view, color) {
const colorName = colorHexToName(color);
return `${baseProduct}/${view}/${colorName}.png`;
}
```

1. **Upload organized templates to Supabase**
   - Create folders: `5oz-cotton-bag/front/`, `5oz-cotton-bag/back/`
   - Upload color variants: `red.png`, `blue.png`, etc.

2. **Update JSON catalog**
   ```json
   {
     "5oz-cotton-bag#front": {
       "name": "5oz Cotton Bag - Front",
       "baseProduct": "5oz-cotton-bag",
       "view": "front",
       "templatePattern": "5oz-cotton-bag/front/{color}.png",
       "printAreas": { "front": {...} },
       "colors": ["#f5deb3", "#ff0000", "#0000ff"]
     },
     "5oz-cotton-bag#back": {
       "name": "5oz Cotton Bag - Back",
       "baseProduct": "5oz-cotton-bag",
       "view": "back",
       "templatePattern": "5oz-cotton-bag/back/{color}.png",
       "printAreas": { "back": {...} },
       "colors": ["#f5deb3", "#ff0000", "#0000ff"]
     }
   }
   ```

3. **Update EnhancedDesigner.jsx**
   ```javascript
   // When color changes, reload template with new color
   const loadTemplateForColor = (baseProduct, view, color) => {
   ```

```
    const colorName = getColorName(color);
    const templateUrl = `${supabaseUrl}/storage/v1/object/public/product-templates/${baseProduct}/${view}/${colorName}.png`;
    loadProductTemplate(templateUrl);
  };
```

Advantages:
- ✅ Quick to implement (1-2 days)
- ✅ No database migration needed
- ✅ Works with existing PrintAreaAdmin
- ✅ Immediate value for users

Limitations:
- ⚠️ Product keys become longer and more complex
- ⚠️ Need to maintain baseProduct + view parsing everywhere
- ⚠️ Not as clean as normalized schema

Phase 2: Normalized Schema (Long-term) - Option 2

Goal: Proper database structure for scalability

Steps:

1. **Create migration script** (Day 1)
   - Create new tables: `base_products`, `product_views`, `view_templates`
   - Migrate existing `product_templates` data
   - Create foreign key relationships

2. **Update supabaseService.js** (Day 2-3)
   - Add new CRUD functions for base_products, product_views, view_templates
   - Update `loadProductConfiguration` to query new schema
   - Add `getProductViewsForBaseProduct(baseProductKey)`
   - Add `getViewTemplateForColor(viewId, colorHex)`

3. **Update PrintAreaAdmin** (Day 3-4)
   - Add view selector dropdown
   - Allow configuring print areas per view
   - Allow uploading templates per color per view
   - Show preview of all color variants

4. **Update EnhancedDesigner** (Day 4-5)
   - Add view selector UI
   - Add color selector UI
   - Load correct template based on view + color selection
   - Load correct print areas based on view

5. **Update JSON catalog structure** (Day 5)

```json
  {
    "baseProducts": [
      {
        "key": "5oz-cotton-bag",
```

```
        "name": "5oz Cotton Bag",
        "basePrice": 18.99,
        "availableColors": [
          { "hex": "#f5deb3", "name": "Beige" },
          { "hex": "#ff0000", "name": "Red" },
          { "hex": "#0000ff", "name": "Blue" }
        ],
        "views": [
          {
            "key": "front",
            "name": "Front",
            "printAreas": { "front": {...} }
          },
          {
            "key": "back",
            "name": "Back",
            "printAreas": { "back": {...} }
          }
        ]
      }
    ]
  }
```

6. **Testing** (Day 6-7)
   - Test view switching
   - Test color switching
   - Test print area constraints per view
   - Test admin panel for managing variants

Advantages:
- ✅ Clean, normalized database structure
- ✅ Scalable to hundreds of products and variants
- ✅ Easy to query and manage
- ✅ Better performance for complex queries

Timeline: ~2 weeks for full implementation

Code Changes Required

1. Update `supabaseService.js`

Add new functions:

```
// Get base product with all views and variants
export const getBaseProductComplete = async (baseProductKey) => {
  const client = getSupabaseClient();

  const { data, error } = await client
    .from('base_products')
    .select(`
      *,
      product_views (
        *,
        view_templates (*),
        print_areas (*)
      )
    `)
    .eq('product_key', baseProductKey)
    .single();

  if (error) throw error;
  return data;
};

// Get template URL for specific view and color
export const getViewTemplateUrl = async (baseProductKey, viewKey, colorHex) => {
  const client = getSupabaseClient();

  const { data, error } = await client
    .from('view_templates')
    .select('template_url')
    .eq('product_views.base_product_id', (
      await client.from('base_products')
        .select('id')
        .eq('product_key', baseProductKey)
        .single()
    ).data.id)
    .eq('product_views.view_key', viewKey)
    .eq('color_hex', colorHex)
    .single();

  if (error) throw error;
  return data.template_url;
};

// Upload template for specific view and color
export const uploadViewTemplate = async (
  file,
  baseProductKey,
  viewKey,
  colorHex
) => {
  const client = getSupabaseClient();

  const colorName = colorHexToName(colorHex);
  const filePath = `${baseProductKey}/${viewKey}/${colorName}.png`;

  const { error: uploadError } = await client.storage
    .from('product-templates')
    .upload(filePath, file, {
      cacheControl: '3600',
      upsert: true  // Allow replacing existing
    });

  if (uploadError) throw uploadError;
```

```
  const { data: { publicUrl } } = client.storage
    .from('product-templates')
    .getPublicUrl(filePath);

  return publicUrl;
};
```

2. Update `EnhancedDesigner.jsx`

Add view and color state:

```
const [selectedView, setSelectedView] = useState('front');
const [selectedColor, setSelectedColor] = useState('#f5deb3');
const [availableViews, setAvailableViews] = useState([]);
const [availableColors, setAvailableColors] = useState([]);

// Load product with views
useEffect(() => {
  const loadProductData = async () => {
    if (!selectedProduct) return;

    const productData = await getBaseProductComplete(selectedProduct);
    setAvailableViews(productData.product_views);
    setAvailableColors(productData.available_colors);

    // Set default view and color
    if (productData.product_views.length > 0) {
      setSelectedView(productData.product_views[0].view_key);
    }
    if (productData.available_colors.length > 0) {
      setSelectedColor(productData.available_colors[0].hex);
    }
  };

  loadProductData();
}, [selectedProduct]);

// Load template when view or color changes
useEffect(() => {
  if (selectedProduct && selectedView && selectedColor) {
    loadTemplateForViewAndColor(selectedProduct, selectedView, selectedColor);
  }
}, [selectedProduct, selectedView, selectedColor]);

// Load template function
const loadTemplateForViewAndColor = async (productKey, viewKey, colorHex) => {
  const templateUrl = await getViewTemplateUrl(productKey, viewKey, colorHex);

  // Load template image to canvas
  fabric.Image.fromURL(templateUrl, (img) => {
    if (img && img._element) {
      // ... scale and position logic
      canvas.add(img);
      canvas.sendToBack(img);

      // Load print areas for this view
      loadPrintAreasForView(productKey, viewKey);
    }
  });
};
```

Add UI for view and color selection:

```
{/* View Selector */}
<div className="space-y-2">
  <label className="block text-sm font-medium">Product View</label>
  <select
    value={selectedView}
    onChange={(e) => setSelectedView(e.target.value)}
    className="w-full px-3 py-2 border rounded-md"
  >
    {availableViews.map(view => (
      <option key={view.view_key} value={view.view_key}>
        {view.view_name}
      </option>
    ))}
  </select>
</div>

{/* Color Selector */}
<div className="space-y-2">
  <label className="block text-sm font-medium">Color Variant</label>
  <div className="flex gap-2 flex-wrap">
    {availableColors.map(color => (
      <button
        key={color.hex}
        onClick={() => setSelectedColor(color.hex)}
        className={`w-10 h-10 rounded-full border-2 ${
          selectedColor === color.hex
            ? 'border-blue-500 ring-2 ring-blue-200'
            : 'border-gray-300'
        }`}
        style={{ backgroundColor: color.hex }}
        title={color.name}
      />
    ))}
  </div>
</div>
```

3. Update `PrintAreaAdmin.jsx`

Add view selector:

```
const [selectedView, setSelectedView] = useState('front');
const [viewTemplates, setViewTemplates] = useState({});

// Load all templates for all colors for this view
useEffect(() => {
  if (selectedProduct && selectedView) {
    loadViewTemplates(selectedProduct, selectedView);
  }
}, [selectedProduct, selectedView]);

const loadViewTemplates = async (productKey, viewKey) => {
  const templates = await getViewTemplates(productKey, viewKey);
  setViewTemplates(templates);
};

// Upload template for specific color
const handleTemplateUploadForColor = async (file, color) => {
  const url = await uploadViewTemplate(
    file,
    selectedProduct,
    selectedView,
    color
  );

  setViewTemplates(prev => ({
    ...prev,
    [color]: url
  }));

  // Refresh canvas if this is the currently selected color
  if (color === selectedColor) {
    loadTemplateForViewAndColor(selectedProduct, selectedView, color);
  }
};
```

Add multi-color template upload UI:

```
<div className="space-y-4">
  <h3 className="font-semibold">Upload Templates by Color</h3>

  {availableColors.map(color => (
    <div key={color.hex} className="flex items-center justify-between p-3 border roun-
ded">
      <div className="flex items-center space-x-3">
        <div
          className="w-8 h-8 rounded border"
          style={{ backgroundColor: color.hex }}
        />
        <span>{color.name}</span>
      </div>

      <div className="flex items-center space-x-2">
        {viewTemplates[color.hex] && (
          <img
            src={viewTemplates[color.hex]}
            alt={color.name}
            className="w-16 h-16 object-contain border rounded"
          />
        )}
        <button
          onClick={() => {
            const input = document.createElement('input');
            input.type = 'file';
            input.accept = 'image/*';
            input.onchange = (e) => {
              const file = e.target.files[0];
              if (file) handleTemplateUploadForColor(file, color.hex);
            };
            input.click();
          }}
          className="px-3 py-1 bg-blue-600 text-white rounded text-sm hover:bg-
blue-700"
        >
          Upload
        </button>
      </div>
    </div>
  ))}
</div>
```

Example Workflow

Admin Workflow: Setting Up a New Product

1. **Admin opens Print Area Configuration**
2. **Selects product**: “5oz Cotton Bag”
3. **Selects view**: “Front”
4. **Uploads templates for each color:**
   - Upload `beige.png` → Saved to `5oz-cotton-bag/front/beige.png`
   - Upload `red.png` → Saved to `5oz-cotton-bag/front/red.png`
   - Upload `blue.png` → Saved to `5oz-cotton-bag/front/blue.png`

5. **Configures print areas** (one configuration applies to all colors):
   - Draw rectangle: “Front” (400x400 @ position 200,200)
   - Save configuration
6. **Repeat for “Back” view:**
   - Switch to “Back” view
   - Upload `beige.png`, `red.png`, `blue.png` for back view
   - Configure print areas for back view

Customer Workflow: Designing

1. **Customer selects product**: “5oz Cotton Bag”
2. **Customer selects view**: “Front” (dropdown shows: Front, Back)
3. **Customer selects color**: Red (color swatches)
   - Designer loads: `5oz-cotton-bag/front/red.png`
   - Print areas loaded: “Front” configuration
4. **Customer adds design** (text, images, shapes)
   - Design constrained to “Front” print area
5. **Customer switches to “Back” view**
   - Designer loads: `5oz-cotton-bag/back/red.png`
   - Print areas switched to “Back” configuration
   - Customer can design on back
6. **Customer switches color to Blue**
   - Designer loads: `5oz-cotton-bag/front/blue.png` (back to front view)
   - Same design elements remain
   - Print areas unchanged (same configuration)

Migration Strategy

Data Migration Script

```
// migrate-to-views.js
import { createClient } from '@supabase/supabase-js';

const supabase = createClient(SUPABASE_URL, SUPABASE_KEY);

async function migrateToViewSchema() {
  console.log('Starting migration to view schema...');

  // 1. Get all existing product_templates
  const { data: oldTemplates } = await supabase
    .from('product_templates')
    .select('*, print_areas(*)');

  for (const oldTemplate of oldTemplates) {
    // Parse product key to determine base product and view
    const { baseKey, viewKey } = parseProductKey(oldTemplate.product_key);

    // 2. Create or get base_product
    let baseProduct = await supabase
      .from('base_products')
      .select('*')
      .eq('product_key', baseKey)
      .single();

    if (!baseProduct.data) {
      const { data } = await supabase
        .from('base_products')
        .insert({
          product_key: baseKey,
          name: oldTemplate.name,
          base_price: oldTemplate.base_price,
          available_colors: oldTemplate.colors
        })
        .select()
        .single();

      baseProduct = { data };
    }

    // 3. Create product_view
    const { data: productView } = await supabase
      .from('product_views')
      .insert({
        base_product_id: baseProduct.data.id,
        view_key: viewKey || 'default',
        view_name: capitalizeFirst(viewKey || 'default'),
        template_reference: oldTemplate.template_url
      })
      .select()
      .single();

    // 4. Create view_templates for each color
    const colors = oldTemplate.colors || [];
    for (const colorHex of colors) {
      const colorName = colorHexToName(colorHex);
      const templateUrl = oldTemplate.template_url.replace(
        'template.png',
        `${colorName}.png`
      );

      await supabase
        .from('view_templates')
```

```
        .insert({
          product_view_id: productView.id,
          color_hex: colorHex,
          color_name: colorName,
          template_url: templateUrl
        });
    }

    // 5. Migrate print_areas
    for (const printArea of oldTemplate.print_areas) {
      await supabase
        .from('print_areas')
        .insert({
          product_view_id: productView.id,
          area_key: printArea.area_key,
          name: printArea.name,
          x: printArea.x,
          y: printArea.y,
          width: printArea.width,
          height: printArea.height,
          max_width: printArea.max_width,
          max_height: printArea.max_height,
          shape: printArea.shape
        });
    }
  }

  console.log('Migration complete!');
}

// Helper functions
function parseProductKey(fullKey) {
  const parts = fullKey.split('#');
  return {
    baseKey: parts[0],
    viewKey: parts[1] || 'default'
  };
}

function colorHexToName(hex) {
  const colorMap = {
    '#ffffff': 'white',
    '#000000': 'black',
    '#ff0000': 'red',
    '#0000ff': 'blue',
    '#00ff00': 'green',
    '#f5deb3': 'beige'
  };
  return colorMap[hex.toLowerCase()] || hex.replace('#', '');
}

function capitalizeFirst(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}

// Run migration
migrateToViewSchema().catch(console.error);
```

Recommendations

For Immediate Implementation (This Week)

Use Option 1: Extended Current Schema

Why:
- ✅ Minimal code changes
- ✅ No database migration required
- ✅ Works with existing infrastructure
- ✅ Can be implemented in 2-3 days

Action Items:
1. Create folder structure in Supabase Storage: `{product}/{view}/{color}.png`
2. Upload existing templates organized by view and color
3. Update `EnhancedDesigner.jsx` to construct template URLs dynamically
4. Update JSON catalog to include view information
5. Test with a single product (e.g., "5oz Cotton Bag") before rolling out

For Long-term (Next Month)

Migrate to Option 2: Normalized Schema

Why:
- ✅ Cleaner architecture
- ✅ Better scalability
- ✅ Easier to query and manage
- ✅ Supports future features (variants, SKUs, inventory)

Action Items:
1. Create new tables in Supabase
2. Write and test migration script
3. Update all code to use new schema
4. Update admin panel for view-based management
5. Comprehensive testing before deploying

Testing Checklist

Functional Testing

- [ ] Upload template for "Bag Front - Red"
- [ ] Upload template for "Bag Front - Blue"
- [ ] Verify print areas apply to both color variants
- [ ] Switch from "Front" to "Back" view in designer
- [ ] Verify correct template loads for each view
- [ ] Verify correct print areas load for each view
- [ ] Change color while on "Front" view
- [ ] Verify template changes but print areas remain the same
- [ ] Add design elements to "Front" view
- [ ] Switch to "Back" view

- [ ] Verify design elements are cleared (or saved per view if implemented)

Edge Cases

- [ ] Product with single view (mug, poster)
- [ ] Product with single color
- [ ] Missing template image for a color (fallback behavior)
- [ ] View with no print areas configured
- [ ] Product with 10+ color variants

Performance Testing

- [ ] Load time for product with 10 color variants
- [ ] Canvas refresh time when switching views
- [ ] Database query performance for complex products
- [ ] Storage bandwidth usage

Conclusion

This architecture provides a scalable, maintainable solution for handling product variants and views. The two-phase approach allows for:

1. **Quick wins** with minimal changes (Option 1)
2. **Long-term robustness** with proper normalization (Option 2)

The key insight is that **print area configurations are reusable across color variants** of the same view, while **template images are unique** per color per view. This approach minimizes database redundancy while providing maximum flexibility.

Questions & Answers

Q: Do I need one print template per product position (e.g., one “bag front” that works for all bag colors)?

A: Yes! The print area configuration (position, size, constraints) should be defined once per product view. All color variants of that view share the same print area configuration.

Q: How do I organize product images in Supabase storage?

A: Use a hierarchical folder structure:

```
{base_product_key}/{view_key}/{color_name}.png

Example: 5oz-cotton-bag/front/red.png
```

This makes it easy to:
- Upload all variants at once

- Locate images programmatically
- Manage via Supabase Storage dashboard

Q: How should the enhanced designer pull and overlay these templates?

A: The designer should:
1. Let user select **product** → Load available views and colors
2. Let user select **view** → Load print areas for that view
3. Let user select **color** → Load template image for that color
4. Construct URL dynamically: `${baseProduct}/${view}/${color}.png`
5. Overlay print area boundaries on the template
6. Constrain user designs to the print area

Contact & Support

For questions about this architecture or implementation help, please contact the development team or create an issue in the repository.

Document Version: 1.0
Last Updated: October 16, 2025
Author: DeepAgent (Abacus.AI)