THE PETAL POUCHES - CODEBASE DOCUMENTATION

Version: 1.0

Last Updated: October 20, 2025

Stack: React + Node.js/Express + Supabase + Cloudinary

PROJECT OVERVIEW

Description: E-commerce platform for jewelry, soft toys, and lifestyle gifts targeting teenage girls and young women.

Current Status: Admin Dashboard with complete Product & Category Management implemented.

Tech Stack:

- Frontend: React 18+ with Vite
- Backend: Node.js + Express
- Database: Supabase (PostgreSQL)
- Image Storage: Cloudinary
- File Upload: Multer

ARCHITECTURE

Backend Structure

```
backend/
├── src/
│   ├── config/
│   │   ├── supabaseClient.js
│   │   ├── cloudinary.js
│   │   └── multer.js
│   ├── controllers/
│   │   ├── productController.js
│   │   └── categoryController.js
│   ├── routes/
│   │   ├── products.js
│   │   ├── categories.js
│   │   └── admin.js
```

```
│   ├── services/
│   │   └── cloudinaryService.js
│   ├── utils/
│   │   └── cloudinaryHelpers.js
│   └── index.js
├── .env
└── package.json
```

Frontend Structure

```
frontend/
├── src/
│   ├── pages/
│   │   └── Admin.jsx
│   ├── components/
│   │   └── adminComps/
│   │       ├── ProductList.jsx
│   │       ├── CreateProductForm.jsx
│   │       ├── UpdateProductForm.jsx
│   │       └── CategoriesForm.jsx
│   └── main.jsx
└── package.json
```

BACKEND CODE FILES

1. backend/src/index.js

javascript

```
const express = require('express');
const dotenv = require('dotenv');
const cors = require('cors');
const helmet = require('helmet');

dotenv.config();
const app = express();

app.use(helmet());
app.use(cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:5173',
  credentials: true
}));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Routes
app.use('/api/categories', require('./routes/categories'));
app.use('/api/admin', require('./routes/admin'));
app.use('/api/products', require('./routes/products'));

// Health check
app.get('/health', (req, res) => {
  res.status(200).json({
    success: true,
    message: 'Server is healthy',
    timestamp: new Date().toISOString()
  });
});

// Root endpoint
app.get('/', (req, res) => {
  res.json({
    success: true,
    message: 'The Petal Pouches API is running! 🌸',
    version: '1.0.0'
  });
});

// Error handler
app.use((err, req, res, next) => {
  console.error('Error:', err.stack);
  res.status(err.status || 500).json({
```

```
    success: false,
    message: err.message || 'Internal Server Error'
  });
});

// 404 handler
app.use((req, res) => {
  res.status(404).json({
    success: false,
    message: `Route not found: ${req.method} ${req.path}`
  });
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});

module.exports = app;
```

2. backend/src/config/supabaseClient.js

javascript

```
const { createClient } = require('@supabase/supabase-js');

if (!process.env.SUPABASE_URL || !process.env.SUPABASE_SERVICE_ROLE_KEY) {
  throw new Error('Missing Supabase environment variables');
}

const supabase = createClient(
  process.env.SUPABASE_URL,
  process.env.SUPABASE_SERVICE_ROLE_KEY,
  {
    auth: {
      autoRefreshToken: false,
      persistSession: false
    }
  }
);

module.exports = supabase;
```

3. backend/src/config/cloudinary.js

```javascript
const cloudinary = require('cloudinary').v2;

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET
});

module.exports = cloudinary;
```

4. backend/src/config/multer.js

```javascript
const multer = require('multer');

const storage = multer.memoryStorage();

const fileFilter = (req, file, cb) => {
  const allowedTypes = ['image/jpeg', 'image/jpg', 'image/png', 'image/webp'];

  if (allowedTypes.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(new Error('Only image files are allowed'), false);
  }
};

const upload = multer({
  storage: storage,
  fileFilter: fileFilter,
  limits: {
    fileSize: 5 * 1024 * 1024 // 5MB
  }
});

module.exports = upload;
```

5. backend/src/services/cloudinaryService.js

javascript

```
const cloudinary = require('../config/cloudinary');
const streamifier = require('streamifier');

const uploadToCloudinary = (buffer, folder = 'products') => {
  return new Promise((resolve, reject) => {
    const uploadStream = cloudinary.uploader.upload_stream(
      {
        folder: folder,
        resource_type: 'image',
        transformation: [
          { width: 1000, height: 1000, crop: 'limit' },
          { quality: 'auto' },
          { fetch_format: 'auto' }
        ]
      },
      (error, result) => {
        if (error) reject(error);
        else resolve(result);
      }
    );
    streamifier.createReadStream(buffer).pipe(uploadStream);
  });
};

const deleteFromCloudinary = async (publicId) => {
  try {
    const result = await cloudinary.uploader.destroy(publicId);
    return result;
  } catch (error) {
    console.error('Cloudinary delete error:', error);
    throw error;
  }
};

module.exports = {
  uploadToCloudinary,
  deleteFromCloudinary
};
```

6. backend/src/utils/cloudinaryHelpers.js

javascript

```
const extractPublicIdFromUrl = (cloudinaryUrl) => {
  if (!cloudinaryUrl || typeof cloudinaryUrl !== 'string') {
    return null;
  }

  try {
    const matches = cloudinaryUrl.match(/\/upload\/(?:v\d+\/)?(.+)\.\w+$/);
    if (matches && matches[1]) {
      return matches[1];
    }

    const altMatches = cloudinaryUrl.match(/\/upload\/(.+)\.\w+$/);
    if (altMatches && altMatches[1]) {
      return altMatches[1];
    }

    return null;
  } catch (error) {
    console.error('Error extracting public ID:', error);
    return null;
  }
};

module.exports = { extractPublicIdFromUrl };
```

7. backend/src/routes/products.js

javascript

```
const express = require('express');
const router = express.Router();
const { getAllProducts, getProductById } = require('../controllers/productController');

router.get('/', getAllProducts);
router.get('/:id', getProductById);

module.exports = router;
```

8. backend/src/routes/admin.js

javascript

```
const express = require('express');
const router = express.Router();
const upload = require('../config/multer');
const {
  createProduct,
  updateProduct,
  deleteProduct
} = require('../controllers/productController');

router.post('/products', upload.single('image'), createProduct);
router.put('/products/:id', upload.single('image'), updateProduct);
router.delete('/products/:id', deleteProduct);

module.exports = router;
```

9. backend/src/routes/categories.js

javascript

```
const express = require('express');
const router = express.Router();
const {
  getAllCategories,
  getCategoryById,
  createCategory,
  updateCategory,
  deleteCategory
} = require('../controllers/categoryController');

router.get('/', getAllCategories);
router.get('/:id', getCategoryById);
router.post('/admin', createCategory);
router.put('/admin/:id', updateCategory);
router.delete('/admin/:id', deleteCategory);

module.exports = router;
```

10. backend/src/controllers/productController.js

javascript

```
const supabase = require('../config/supabaseClient');
const { uploadToCloudinary, deleteFromCloudinary } = require('../services/cloudinaryService');
const { extractPublicIdFromUrl } = require('../utils/cloudinaryHelpers');

const createProduct = async (req, res) => {
  try {
    const { title, description, price, category_id, stock, sku } = req.body;

    if (!req.file) {
      return res.status(400).json({
        success: false,
        message: 'Product image is required'
      });
    }

    const cloudinaryResult = await uploadToCloudinary(req.file.buffer, 'products');

    const productData = {
      title,
      description,
      price: parseInt(price),
      stock: parseInt(stock),
      sku,
      img_url: cloudinaryResult.url,
      has_variants: false
    };

    if (category_id && category_id.trim() !== '') {
      productData.category_id = category_id;
    }

    const { data, error } = await supabase
      .from('Products')
      .insert([productData])
      .select()
      .single();

    if (error) {
      await deleteFromCloudinary(cloudinaryResult.publicId);
      throw error;
    }

    res.status(201).json({
```

```
      success: true,
      message: 'Product created successfully',
      data
    });
  } catch (error) {
    console.error('Create product error:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to create product',
      error: error.message
    });
  }
};

const getProductById = async (req, res) => {
  try {
    const { id } = req.params;

    const { data, error } = await supabase
      .from('Products')
      .select(`*, Categories (id, name, description)`)
      .eq('id', id)
      .single();

    if (error) {
      if (error.code === 'PGRST116') {
        return res.status(404).json({
          success: false,
          message: 'Product not found'
        });
      }
      throw error;
    }

    res.status(200).json({ success: true, data });
  } catch (error) {
    console.error('Get product error:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to fetch product',
      error: error.message
    });
  }
};
```

```
const updateProduct = async (req, res) => {
  try {
    const { id } = req.params;
    const { title, description, price, stock, category_id, sku } = req.body;

    const { data: existingProduct, error: fetchError } = await supabase
      .from('Products')
      .select('img_url')
      .eq('id', id)
      .single();

    if (fetchError || !existingProduct) {
      return res.status(404).json({
        success: false,
        message: 'Product not found'
      });
    }

    const updateData = {};
    if (title) updateData.title = title;
    if (description !== undefined) updateData.description = description;
    if (price) updateData.price = parseInt(price);
    if (stock !== undefined) updateData.stock = parseInt(stock);
    if (sku) updateData.sku = sku;

    if (category_id && category_id.trim() !== '') {
      updateData.category_id = category_id;
    } else if (category_id === null || category_id === '') {
      updateData.category_id = null;
    }

    if (req.file) {
      try {
        const cloudinaryResult = await uploadToCloudinary(req.file.buffer, 'products');
        updateData.img_url = cloudinaryResult.url;

        if (existingProduct.img_url) {
          const oldPublicId = extractPublicIdFromUrl(existingProduct.img_url);
          if (oldPublicId) {
            await deleteFromCloudinary(oldPublicId);
          }
        }
      } catch (uploadError) {
```

```
        return res.status(500).json({
          success: false,
          message: 'Failed to upload new image',
          error: uploadError.message
        });
      }
    }

    const { data, error } = await supabase
      .from('Products')
      .update(updateData)
      .eq('id', id)
      .select()
      .single();

    if (error) throw error;

    res.status(200).json({
      success: true,
      message: 'Product updated successfully',
      data
    });
  } catch (error) {
    console.error('Update product error:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to update product',
      error: error.message
    });
  }
};

const deleteProduct = async (req, res) => {
  try {
    const { id } = req.params;

    const { data: product, error: fetchError } = await supabase
      .from('Products')
      .select('img_url')
      .eq('id', id)
      .single();

    if (fetchError) {
      if (fetchError.code === 'PGRST116') {
```

```
        return res.status(404).json({
          success: false,
          message: 'Product not found'
        });
      }
      throw fetchError;
    }

    const { error: deleteError } = await supabase
      .from('Products')
      .delete()
      .eq('id', id);

    if (deleteError) throw deleteError;

    if (product.img_url) {
      try {
        const publicId = extractPublicIdFromUrl(product.img_url);
        if (publicId) {
          await deleteFromCloudinary(publicId);
        }
      } catch (cloudinaryError) {
        console.error('Cloudinary delete error:', cloudinaryError);
      }
    }

    res.status(200).json({
      success: true,
      message: 'Product deleted successfully'
    });
  } catch (error) {
    console.error('Delete product error:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to delete product',
      error: error.message
    });
  }
};

const getAllProducts = async (req, res) => {
  try {
    const {
      category_id,
```

```
    min_price,
    max_price,
    search,
    sort = 'created_at',
    page = 1,
    limit = 20,
    in_stock
  } = req.query;

  let query = supabase
    .from('Products')
    .select('*, Categories(id, name)', { count: 'exact' });

  if (category_id) query = query.eq('category_id', category_id);
  if (min_price) query = query.gte('price', parseInt(min_price));
  if (max_price) query = query.lte('price', parseInt(max_price));
  if (search) query = query.ilike('title', `%${search}%`);
  if (in_stock === 'true') query = query.gt('stock', 0);

  switch (sort) {
    case 'price_asc':
      query = query.order('price', { ascending: true });
      break;
    case 'price_desc':
      query = query.order('price', { ascending: false });
      break;
    default:
      query = query.order('created_at', { ascending: false });
  }

  const pageNum = parseInt(page);
  const limitNum = parseInt(limit);
  const from = (pageNum - 1) * limitNum;
  const to = from + limitNum - 1;

  query = query.range(from, to);

  const { data, error, count } = await query;

  if (error) throw error;

  const totalPages = Math.ceil(count / limitNum);

  res.status(200).json({
```

```
      success: true,
      data,
      metadata: {
        totalCount: count,
        totalPages,
        currentPage: pageNum,
        limit: limitNum,
        hasMore: pageNum < totalPages
      }
    });
  } catch (error) {
    console.error('Get all products error:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to fetch products',
      error: error.message
    });
  }
};

module.exports = {
  createProduct,
  getProductById,
  updateProduct,
  deleteProduct,
  getAllProducts
};
```

11. backend/src/controllers/categoryController.js

javascript

```
const supabase = require('../config/supabaseClient');

const createCategory = async (req, res) => {
  try {
    const { name, description } = req.body;

    if (!name || name.trim() === '') {
      return res.status(400).json({
        success: false,
        message: 'Category name is required'
      });
    }

    const { data: existingCategory } = await supabase
      .from('Categories')
      .select('id')
      .ilike('name', name.trim())
      .single();

    if (existingCategory) {
      return res.status(409).json({
        success: false,
        message: 'Category with this name already exists'
      });
    }

    const { data, error } = await supabase
      .from('Categories')
      .insert([{ name: name.trim(), description: description?.trim() || null }])
      .select()
      .single();

    if (error) throw error;

    res.status(201).json({
      success: true,
      message: 'Category created successfully',
      data
    });
  } catch (error) {
    console.error('Create category error:', error);
    res.status(500).json({
      success: false,
```

```
      message: 'Failed to create category',
      error: error.message
    });
  }
};

const getAllCategories = async (req, res) => {
  try {
    const { data, error } = await supabase
      .from('Categories')
      .select('*')
      .order('name', { ascending: true });

    if (error) throw error;

    res.status(200).json({
      success: true,
      count: data.length,
      data
    });
  } catch (error) {
    console.error('Get categories error:', error);
    res.status(500).json({
      success: false,
      message: 'Failed to fetch categories',
      error: error.message
    });
  }
};

const getCategoryById = async (req, res) => {
  try {
    const { id } = req.params;

    const { data, error } = await supabase
      .from('Categories')
      .select('*')
      .eq('id', id)
      .single();

    if (error) {
      if (error.code === 'PGRST116') {
        return res.status(404).json({
          success: false,
```

```
        message: 'Category not found'
      });
    }
    throw error;
  }

  res.status(200).json({ success: true, data });
 } catch (error) {
  console.error('Get category error:', error);
  res.status(500).json({
    success: false,
    message: 'Failed to fetch category',
    error: error.message
  });
 }
};

const updateCategory = async (req, res) => {
 try {
  const { id } = req.params;
  const { name, description } = req.body;

  if (!name && !description) {
    return res.status(400).json({
      success: false,
      message: 'At least one field is required'
    });
  }

  const { data: existingCategory } = await supabase
    .from('Categories')
    .select('id')
    .eq('id', id)
    .single();

  if (!existingCategory) {
    return res.status(404).json({
      success: false,
      message: 'Category not found'
    });
  }

  if (name && name.trim() !== '') {
    const { data: duplicateCategory } = await supabase
```

```
      .from('Categories')
      .select('id')
      .ilike('name', name.trim())
      .neq('id', id)
      .single();

    if (duplicateCategory) {
      return res.status(409).json({
        success: false,
        message: 'Another category with this name already exists'
      });
    }
  }

  const updateData = {};
  if (name && name.trim() !== '') updateData.name = name.trim();
  if (description !== undefined) updateData.description = description?.trim() || null;

  const { data, error } = await supabase
    .from('Categories')
    .update(updateData)
    .eq('id', id)
    .select()
    .single();

  if (error) throw error;

  res.status(200).json({
    success: true,
    message: 'Category updated successfully',
    data
  });
 } catch (error) {
  console.error('Update category error:', error);
  res.status(500).json({
    success: false,
    message: 'Failed to update category',
    error: error.message
  });
 }
};

const deleteCategory = async (req, res) => {
 try {
```

```
  const { id } = req.params;

  const { data: category } = await supabase
    .from('Categories')
    .select('id, name')
    .eq('id', id)
    .single();

  if (!category) {
    return res.status(404).json({
      success: false,
      message: 'Category not found'
    });
  }

  const { data: products } = await supabase
    .from('Products')
    .select('id')
    .eq('category_id', id)
    .limit(1);

  if (products && products.length > 0) {
    return res.status(400).json({
      success: false,
      message: 'Cannot delete category. Products are assigned to it.'
    });
  }

  const { error } = await supabase
    .from('Categories')
    .delete()
    .eq('id', id);

  if (error) throw error;

  res.status(200).json({
    success: true,
    message: `Category "${category.name}" deleted successfully`
  });
} catch (error) {
  console.error('Delete category error:', error);
  res.status(500).json({
    success: false,
    message: 'Failed to delete category',
```

```
    error: error.message
   });
  }
};

module.exports = {
  createCategory,
  getAllCategories,
  getCategoryById,
  updateCategory,
  deleteCategory
};
```

FRONTEND CODE FILES

1. frontend/src/pages/Admin.jsx

Key Features:

- Tab navigation (Product List, Create Product, Manage Categories)
- State management for active view and selected product
- Edit product flow with ID passing

Core Logic:

```javascript
const [activeView, setActiveView] = useState('list');
const [selectedProductId, setSelectedProductId] = useState(null);

const handleEdit = (productId) => {
  setSelectedProductId(productId);
  setActiveView('edit');
};
```

2. frontend/src/components/adminComps/ProductList.jsx

Features:

- Product table with filters (search, price range, stock)
- Pagination

- Edit/Delete actions
- Real-time product display

Key Props:

```
javascript

{ onEdit } // Callback to parent for edit action
```

3. frontend/src/components/adminComps/CreateProductForm.jsx

Features:

- Image upload with preview
- Category selection
- Quick add category inline
- Form validation

API Call:

```
javascript

POST /api/admin/products
Content-Type: multipart/form-data
Body: { image, title, description, price, stock, sku, category_id }
```

4. frontend/src/components/adminComps/UpdateProductForm.jsx

Features:

- Load existing product data
- Optional new image upload
- Quick add category inline
- Update or cancel actions

Props:

```
javascript

{ productId, onSuccess, onCancel }
```

API Call:

```javascript
PUT /api/admin/products/:id
Content-Type: multipart/form-data
Body: { image?, title, description, price, stock, sku, category_id }
```

5. frontend/src/components/adminComps/CategoriesForm.jsx

Features:

- List all categories
- Create/Update/Delete operations
- Edit mode with form population
- Validation to prevent deletion if products exist

API ENDPOINTS

Products

```
GET    /api/products            // Get all (with filters)
GET    /api/products/:id        // Get by ID
POST   /api/admin/products      // Create (with image)
PUT    /api/admin/products/:id  // Update (optional image)
DELETE /api/admin/products/:id  // Delete
```

Categories

```
GET    /api/categories          // Get all
GET    /api/categories/:id      // Get by ID
POST   /api/categories/admin    // Create
PUT    /api/categories/admin/:id  // Update
DELETE /api/categories/admin/:id  // Delete
```

DATABASE SCHEMA

Products Table

```sql
id: UUID (Primary Key)
title: VARCHAR(255)
description: TEXT
price: INTEGER
stock: INTEGER
sku: VARCHAR(100)
img_url: TEXT
category_id: UUID (Foreign Key -> Categories.id)
has_variants: BOOLEAN
created_at: TIMESTAMP
updated_at: TIMESTAMP
```

Categories Table

```sql
id: UUID (Primary Key)
name: VARCHAR(100)
description: TEXT
created_at: TIMESTAMP
updated_at: TIMESTAMP
```

ENVIRONMENT VARIABLES

Backend (.env)

```
PORT=5000
NODE_ENV=development
FRONTEND_URL=http://localhost:5173

SUPABASE_URL=your_supabase_url
SUPABASE_SERVICE_ROLE_KEY=your_service_role_key

CLOUDINARY_CLOUD_NAME=your_cloud_name
```

```
CLOUDINARY_API_KEY=your_api_key
CLOUDINARY_API_SECRET=your_api_secret
```

Frontend (.env)

```
VITE_API_BASE_URL=http://localhost:5000
```

FEATURES IMPLEMENTED ✅

1. **Product Management**
   - Create with image upload
   - Read (list with filters + single product)
   - Update with optional image change
   - Delete with Cloudinary cleanup
2. **Category Management**
   - Full CRUD operations
   - Duplicate name validation
   - Quick add from product forms
   - Protection against deletion with products
3. **Image Handling**
   - Upload to Cloudinary
   - Auto-optimization
   - Delete old images on update
   - Preview before upload
4. **Filtering & Search**
   - Search by title
   - Filter by price range
   - Filter by stock status
   - Sort by date/price
5. **Admin UI**

- Tab navigation
- Inline category creation
- Success/error messaging
- Pagination

NEXT STEPS / TODO

1. **Authentication**
   - Implement admin login
   - JWT/session management
   - Protected routes
2. **Public Frontend**
   - Product catalog page
   - Product detail page
   - Shopping cart
   - Checkout flow
3. **Gift Features**
   - Gift quiz implementation
   - Bundle creation
   - Wishlist/registry
4. **Advanced Features**
   - Order management
   - Customer accounts
   - Reviews & ratings
   - Email notifications

NOTES FOR AI CONTINUATION

Current State: Admin dashboard fully functional with product and category management.

Code Quality:

- All CRUD operations working
- Error handling in place
- Image upload/delete integrated
- Frontend-backend integration complete

Testing: Manual testing done. All endpoints verified.

Dependencies: All required npm packages installed and configured.

Ready For: Public-facing e-commerce features, authentication layer, or advanced admin features.

END OF DOCUMENTATION