\*\*Project\*\*: \*Black Styles – A culturally intelligent, AI-powered fashion assistant for men\*

\*\*Stack\*\*: React (Vite) frontend, Flask backend, OpenCV + TensorFlow for AI, Firebase/MongoDB for storage

\*\*Day 1 Progress\*\*:

\* Set up full project environment (frontend + backend)

\* Built and tested frontend photo upload component

\* Scaffolded Flask backend with image route structure

\* Resolved initial rendering bug and connected Axios to API stub

\*\*Next Steps\*\*:

\* Build `/api/image/upload` route to handle file uploads

\* Extract color palette from image with OpenCV + KMeans

\* Return top 3–5 hex colors to the frontend for display

\*\*Project\*\*: Black Styles – Culturally intelligent fashion + grooming app

\*\*Progress Day 2\*\*:

- Built and tested full backend route for image uploads

- Integrated OpenCV + scikit-learn for real-time color palette extraction

- Connected frontend to backend and validated working response pipeline

\*\*Next\*\*:

- Visually display palette on frontend

- Style with Tailwind or similar

- Begin body metric detection (MediaPipe)

# 🎨 Black Styles — Day 3 Backend Development Summary

## 🛠️ Completed Features

- Flask `/api/image/upload` route

- File type validation for secure uploads

- Request logging with Flask middleware

- Center auto-cropping with OpenCV

- Graceful error handling for image processing

## 🧪 Tested and Verified

- Image files processed → ✅ palette returned

- Bad/corrupt files → ✅ handled cleanly

- Terminal logs confirm error tracking and request metadata

## 📁 Files Updated

- `routes.py` → added upload logic

- `utils.py` → added helper for cropping + palette

- `\_\_init\_\_.py` → initialized app and routes

- `run.py` → Flask bootstrapping

## 🔄 Branch

- `feat/day3-palette-ui-backend-enhancements`

---

🟢 Backend functionality complete. Ready for Day 4 frontend UI enhancements.

# Black Styles — Day 4 Frontend Development Summary

## Features Implemented

- Added `ColorSwatches` component to render extracted color palette visually.

- Integrated palette into `UploadForm`, showing top 3–5 hex colors with swatches.

- Implemented loading state to indicate upload in progress.

- Added error UI for failed uploads or missing files.

## Stack

- Frontend: React (Vite)

- Communication: Axios POST to Flask backend

- Visualization: Dynamic swatches with hex labels

## Testing

- Verified swatches appear with correct hex values after successful upload.

- Loading indicator displays during request.

- Error messages surface with meaningful backend feedback.

## Branch

- `feat/day4-palette-ui-frontend`

## Next Goals

- Clipboard copy for hex codes.

- Persistent recent palettes.

- Styling polish (Tailwind integration).

## 🧾 Portfolio Documentation (copy into portfolio)

\*\*Project:\*\* Black Styles — culturally intelligent, AI-assisted style & palette recommender for men

\*\*Stack:\*\* React (Vite), Flask (Python), OpenCV/TensorFlow (future), MongoDB or Firebase (future)

\*\*Day 5 Highlights:\*\*

- Users can upload an image and get a palette.

- Users can copy hex colors to clipboard (quick share).

- Users can save outfits (image + palette + theme) locally for quick reference.

- Backend provides a simple theme matcher that maps palettes to seasonal themes.

\*\*Files to showcase\*\*

- `frontend/src/components/UploadForm.jsx` — upload/preview/save flow

- `frontend/src/components/ColorSwatches.jsx` — color UI + copy action

- `frontend/src/components/RecentOutfits.jsx` — saved outfits UI

- `backend/app/services/theme\_matcher.py` — initial theme mapping logic

\*\*Next steps to mention\*\*

- Persist saved outfits to a user account (MongoDB + auth)

- Improve theme matcher with ML or curated rule sets

- Add "Save to Wardrobe" and share/export features

## 📚 Project Documentation (so far)

\*\*Project goal\*\*

Black Styles is a full-stack web app that extracts color palettes from clothing images, recommends styling choices and fragrances, and helps users select outfits that align with Black American old money aesthetics fused with modern tailoring.

\*\*Architecture\*\*

- \*\*Frontend:\*\* React (Vite) — Upload UI, palette visualization, local outfit saving

- \*\*Backend:\*\* Flask — Image upload, color extraction (OpenCV + KMeans), theme matching

- \*\*Storage:\*\* localStorage for saved outfits (Day 5); plan: MongoDB/Firebase for persistence

- \*\*AI/CV stack:\*\* OpenCV + MediaPipe + TensorFlow for body metrics and advanced features (future)

\*\*Completed milestones\*\*

1. Project skeleton (frontend + backend)

2. Upload form + basic API connection

3. Image file handling with validation and cropping

4. KMeans color extraction (return hex palette)

5. Request logging & graceful error handling

6. Day 5: palette UI, copy-to-clipboard, save outfits, recent outfits, simple theme matcher

\*\*How to run locally\*\*

- Backend:

```bash

cd backend

python -m venv venv

source venv/bin/activate

pip install -r requirements.txt

python run.py

- Frontend:

cd frontend

npm install

npm run dev

API endpoints (so far):

* POST /api/image/upload — accepts multipart/form-data (image) → returns { palette: [ '#rrggbb', ... ] }
* POST /api/theme — accepts { palette: [...] } → returns { theme: 'Spring' }

# \*\*📁 Portfolio Documentation — Black Styles\*\*

```markdown

# Black Styles — Portfolio Documentation

## Project Overview

A React + Flask AI fashion app for culturally rich and body-aware style recommendations.

Features body-aware color palette extraction from clothing or full-body photos, theme-based style suggestions, and outfit persistence.

## Key Features (Implemented so far)

1. \*\*Photo Upload / Camera Input\*\* – upload images for analysis

2. \*\*Color Palette Extraction\*\* – OpenCV + KMeans for top 3–5 colors

3. \*\*Theme Matching\*\* – assign season based on palette

4. \*\*Palette Persistence\*\* – save & fetch outfits via backend (MongoDB or JSON fallback)

5. \*\*Recent Outfits Display\*\* – frontend component to visualize saved palettes with thumbnails

6. \*\*Frontend Utilities\*\* – Axios API helper functions

7. \*\*Local Storage Fallback\*\* – offline persistence

8. \*\*Request Logging & Error Handling\*\* – backend logs and graceful failures

9. \*\*OOP Design Principles\*\* – modular backend services and utilities

## Technology Stack

- \*\*Frontend:\*\* React, JavaScript

- \*\*Backend:\*\* Python, Flask

- \*\*AI/Computer Vision:\*\* OpenCV, MediaPipe, TensorFlow (for body metrics/color extraction)

- \*\*Database:\*\* MongoDB Atlas (free tier) / JSON fallback

- \*\*Storage:\*\* Firebase Storage or Cloudinary

- \*\*Version Control:\*\* GitHub (branches, PRs, commits)