D . R . I . P .

DIGITAL REGISTRY OF ITEMS & PAIRINGS

A Cloud-Connected Wardrobe Management Platform

SKYLER WATSON

ITEMS	OUTFITS	AI MODELS	CLOUD SERVICES
73	6	3	4

TECHNICAL REPORT -- MARCH 2026

React / Express.js / Supabase / OpenRouter / Gemini / BiRefNet / Railway

1. PROJECT OVERVIEW

D.R.I.P. (Digital Registry of Items & Pairings) is a cloud-connected web application for managing a personal wardrobe. The platform combines computer vision, AI metadata extraction, and real-time web search to automatically catalog, price, and organize clothing items.

Users photograph clothing items, which are then processed through a multi-stage AI pipeline: background removal via BiRefNet, metadata extraction via OpenRouter/Gemini Flash vision, and real-time value estimation via Gemini 2.5 Flash with Google Search grounding that searches actual retail and resale platforms for current pricing.

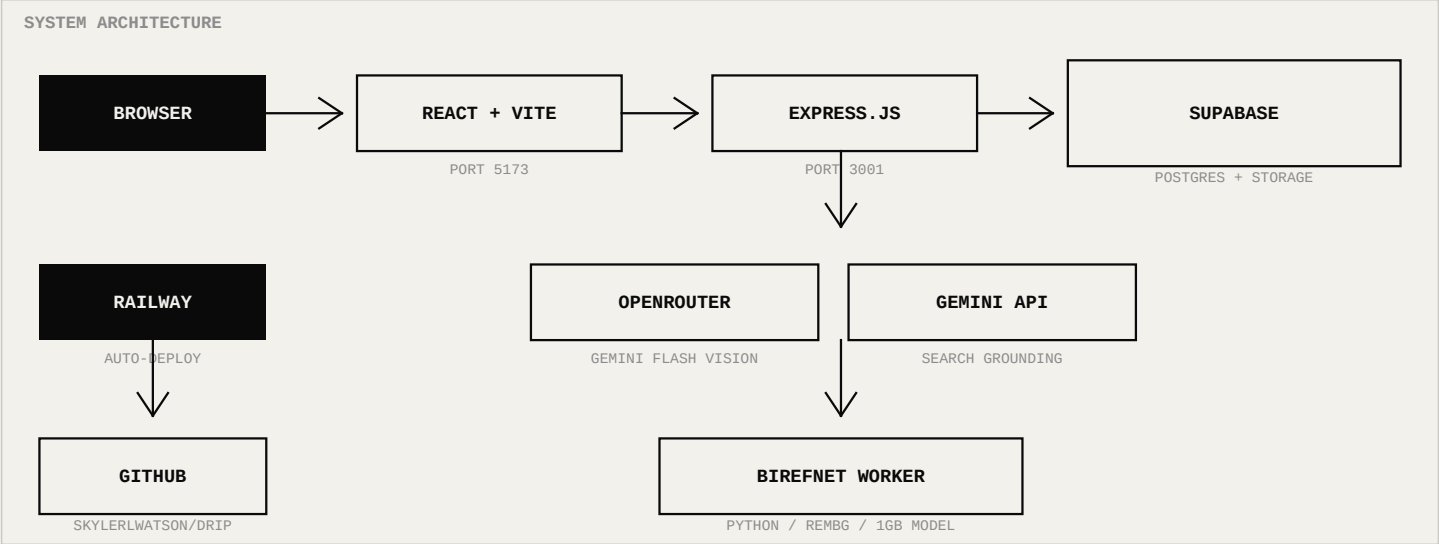
The application runs on Supabase (Postgres + Storage) for data persistence, deploys to Railway via GitHub auto-deploy, and uses a brutalist design system.

CORE CAPABILITIES

-- Photo Capture & Upload: Phone camera or desktop file upload with automatic HEIC-to-JPEG conversion for Apple devices

-- A
i

2. SYSTEM ARCHITECTURE



FRONTEND -- REACT 18 + VITE 6

Single-page app with five views: Wardrobe, Upload, Builder, Outfits, Randomize. Custom useWardrobe hook manages all API calls and state. BrandSelect component provides searchable dropdown with 40+ brands. Wardrobe view supports default category grouping and price sorting. Brutalist CSS with cream background (#f2f1ec), zero border-radius, JetBrains Mono labels.

BACKEND -- EXPRESS.JS

REST API server handling photo uploads (multer), HEIC conversion (heic-convert), Supabase CRUD operations, and AI orchestration. Spawns persistent Python BiRefNet worker for background removal. Routes to OpenRouter for photo analysis and Gemini API direct for value estimation with search grounding. Transforms between snake_case (Postgres) and camelCase (frontend) via helper functions.

DATABASE -- SUPABASE POSTGRES

Items table with 28 columns: id, name, category, subcategory, color_primary, color_secondary, color_name, brand, brand_domain, season (TEXT[]), style (TEXT[]), material, photo_original, photo_processed, estimated_value (NUMERIC), value_reason, estimated_year, notes, processing_status, location, nfc_tag_id, tags, wear_count, last_worn, date_added, and more. Outfits table stores JSONB slot configurations. Row Level Security policies written and ready for multi-user activation.

PHOTO STORAGE -- SUPABASE STORAGE

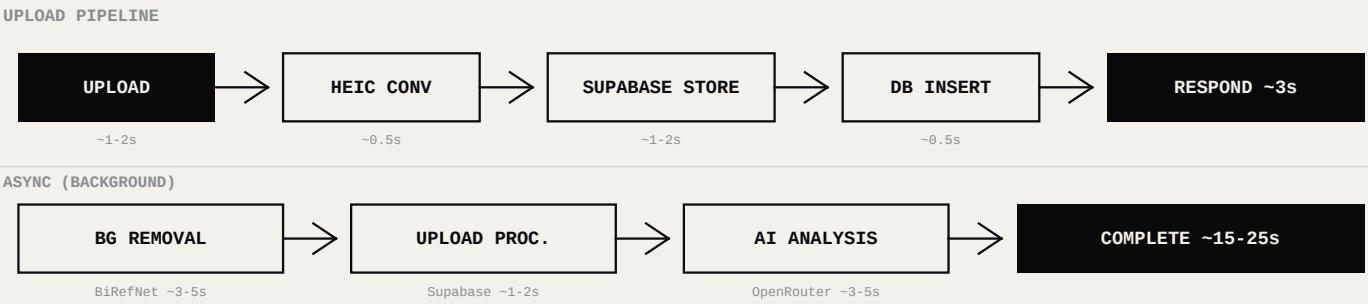
Public bucket 'photos' with originals/ and processed/ paths. Photos served via Supabase CDN URLs. Local photos/ directory is a processing cache only (gitignored). The photoUrl() utility on the frontend handles both full Supabase URLs and legacy local paths.

DEPLOYMENT -- RAILWAY

Auto-deploys from GitHub master branch. Node.js runtime with environment variables: PORT (3001), ADMIN_PASSWORD, OPENROUTER_API_KEY, GEMINI_API_KEY, SUPABASE_URL, SUPABASE_ANON_KEY.

Background removal is disabled on Railway (no Python runtime). Live at digitalregistryitemspairings.up.railway.app.

3. UPLOAD PIPELINE & PROCESSING



PER-UPLOAD TIMING BREAKDOWN

Upload + HEIC conversion	**~1-2s**	synchronous
Upload original to Supabase Storage	**~1-2s**	synchronous
Insert item to Postgres + respond	**~0.5s**	synchronous, user sees card
BiRefNet background removal	**~3-5s**	async, spinner on card
Upload processed PNG to Storage	**~1-2s**	async
OpenRouter Gemini photo analysis	**~3-5s**	async, extracts all metadata

USER SEES ITEM CARD WITH SPINNER **~3-4 seconds**

FULLY ANALYZED, SPINNER CLEARS **~15-25 seconds**

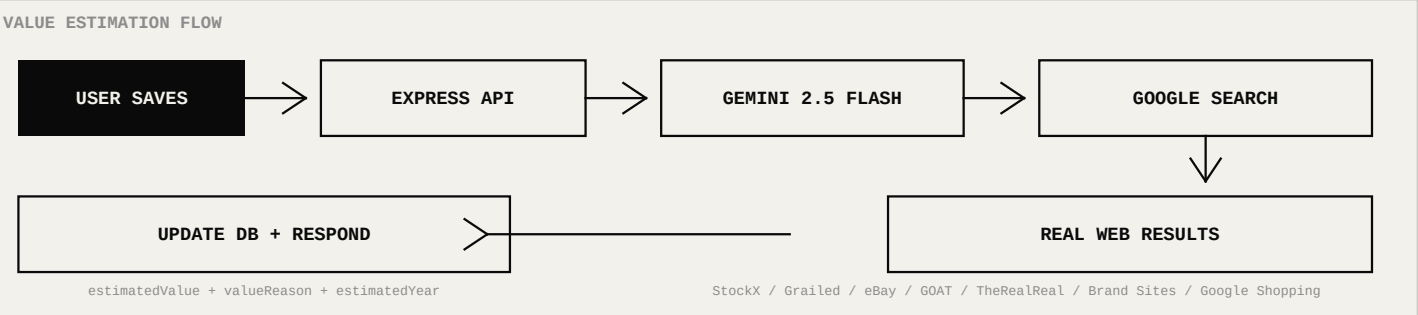
BIREFNET BACKGROUND REMOVAL

State-of-the-art BiRefNet model (~1GB) loads once into a persistent Python worker on server startup. The worker communicates via stdin/stdout JSON protocol -- the Express server writes JSON with the input/output paths, the worker processes and writes JSON back. This keeps the model in memory between requests, achieving ~3-5s per image vs ~15-20s with one-shot spawning. The worker auto-restarts on crash and falls back to one-shot mode if unavailable. Not available on Railway (no Python runtime).

AI PHOTO ANALYSIS (OPENROUTER)

Photos are sent as base64-encoded images to google/gemini-2.0-flash-001 via OpenRouter API. The prompt instructs Gemini to return structured JSON with brand-first naming (e.g. 'Nike Air Force 1' not 'White Sneakers'), actual product/model names when identifiable, and reference numbers saved to the notes field. The AI attempts to identify brands from visible logos, tags, labels, buttons, and stitching patterns. A known-brands mapping covers 40+ brands for domain resolution.

4. REAL-TIME PRICING SYSTEM



DUAL AI SYSTEM

The app uses two separate AI systems for pricing. Initial estimates come from OpenRouter's Gemini Flash vision analysis during upload -- these are rough estimates based on what the AI sees in the photo. Accurate pricing comes from the Gemini 2.5 Flash direct API with Google Search grounding enabled via tools: [{ google_search: {} }]. This allows Gemini to actually search the web in real-time before responding.

When a user edits and saves an item, the /api/estimate-value/:id endpoint sends item metadata (name, brand, category, material, color, notes/reference numbers) to Gemini with search grounding. Gemini searches actual retail and resale platforms and returns prices based on current market data -- not training data. Thinking is disabled via thinkingConfig: { thinkingBudget: 0 } to keep responses fast and reduce token usage.

PRICING LOGIC

-- If item is currently sold at retail AND still in stock: uses new retail price from brand's website or authorized retailers

-- I
p

PRICING FEATURES

-- Total closet value displayed in wardrobe header (currently $11,440)

-- F

API COST BREAKDOWN

OPENROUTER (PHOTO ANALYSIS)	**$0.11 for 334 requests / 930K tokens**
GEMINI SEARCH (VALUE EST.)	**~$0.02-0.05 for 73 items (pending)**
TOTAL PROJECT SPEND TO DATE	**~$0.13-0.16**

GEMINI MONTHLY CAP	**$5.00**
PER-ITEM VALUE ESTIMATION	**~$0.0003**

At current usage levels, the AI pricing system is effectively free. The $5/month Gemini cap supports ~15,000+ value estimations.

5. REST API ENDPOINTS

ITEMS CRUD

GET	/api/wardrobe	Full wardrobe data from Supabase
GET	/api/items	All items (?category=&search= supported)
GET	/api/items/:id	Single item by UUID
POST	/api/items	Upload photo (triggers bg-removal + AI)
PUT	/api/items/:id	Update item metadata (auth required)
DELETE	/api/items/:id	Delete item + Supabase Storage files

OUTFITS CRUD

GET	/api/outfits	All saved outfits
POST	/api/outfits	Save new outfit (JSONB slots)
PUT	/api/outfits/:id	Update existing outfit
DELETE	/api/outfits/:id	Delete outfit

AI ANALYSIS & PROCESSING

POST	/api/analyze/:id	Photo analysis via OpenRouter (vision)
POST	/api/analyze-all	Batch analyze all 'Unanalyzed' items
POST	/api/estimate-value/:id	Value estimation via Gemini + Search
POST	/api/remove-bg/:id	Background removal (BiRefNet worker)

OTHER

POST	/api/randomize	Random outfit generator with filters
POST	/api/login	Simple password auth (ADMIN_PASSWORD)

VALID CATEGORIES

top, bottom, dress, outerwear, shoes, accessory, underwear, swimwear, activewear, other

VALID OUTFIT PRESETS

casual, work, going-out, athletic, formal, date-night, travel

6. FULL TECHNOLOGY STACK

FRONTEND		
React	18.3.1	UI framework, component-based architecture
Vite	6.0.7	Build tool, dev server with HMR
react-router-dom	7.1.1	Client-side routing (5 views)
Vanilla CSS	--	Custom design system, no framework

BACKEND		
Express.js	4.21.2	REST API server
@supabase/supabase-js	2.99.3	Supabase client for DB + Storage
multer	1.4.5	Multipart file upload handling
heic-convert	2.1.0	Apple HEIC to JPEG conversion
sharp	0.34.5	Image processing (installed, HEIC via heic-convert)
dotenv	17.3.1	Environment variable loading
uuid	11.1.0	UUID generation for item IDs
cors	2.8.5	Cross-origin request handling

AI & IMAGE PROCESSING		
OpenRouter API	--	Gemini Flash vision for photo analysis
Google Gemini API	v1beta	Gemini 2.5 Flash + Search Grounding
rembg	2.0.73	Python background removal framework
BiRefNet	~1GB	State-of-the-art segmentation model
Pillow	12.1.1	Python image manipulation

CLOUD INFRASTRUCTURE		
Supabase	--	Postgres DB + Storage CDN + Auth (ready)
Railway	--	Node.js hosting, auto-deploy from GitHub
GitHub	--	Private repo: SkylerLWatson/DRIP

DEVELOPMENT TOOLS		
Claude Code	Opus 4.6	AI pair programming, batch operations
concurrently	9.1.2	Run frontend + backend simultaneously
@vitejs/plugin-react	4.3.4	React support for Vite

7. DESIGN SYSTEM

Brutalist aesthetic. Premium utilitarian -- clean, elevated, intentional. Every element serves a purpose.

COLOR PALETTE

BACKGROUND	#f2f1ec (warm cream)
SECONDARY BG	#e9e8e2 (muted cream)
TEXT PRIMARY	#0a0a0a (near black)
TEXT SECONDARY	#555555 (dark gray)
TEXT MUTED	#999999 (medium gray)
BORDER	#cccbc4 (warm gray)
DANGER	#cc0000 (red, delete actions)

TYPOGRAPHY

-- JetBrains Mono: Labels, buttons, navigation, item names. Bold, uppercase, letter-spacing: 0.12em

-- C

DESIGN RULES

-- Zero border-radius on all elements -- no rounded corners anywhere

-- N

UI COMPONENTS

-- Toggle buttons: SHOW LOGOS, SHOW PRICES, PRICE HIGH-LOW, PRICE LOW-HIGH -- active state inverts colors (dark bg, light text)

-- P

8. SETUP, COMMANDS & URLS

QUICK START

```
npm install              Install all dependencies
npm run dev:all          Start frontend + backend concurrently
npm run dev              Vite dev server only (frontend)
npm run server           Express server only (backend)
npm run build            Production build
npm run preview          Preview production build
```

ENVIRONMENT VARIABLES (.ENV)

```
SUPABASE_URL             https://psuohqoxzvtirvdwvxic.supabase.co
SUPABASE_ANON_KEY        Supabase anonymous key (eyJhbG...)
OPENROUTER_API_KEY       OpenRouter API key for photo analysis
GEMINI_API_KEY           Google Gemini API key for value estimation
ADMIN_PASSWORD           Simple auth password (drip2026)
PORT                     Server port (default: 3001)
```

LIVE URLS

-- Local Development: http://localhost:5173 (frontend) / http://localhost:3001 (API)

-- P

RAILWAY CLI

```
railway login             Authenticate with Railway
railway link              Link to DRIP project
railway service DRIP      Select DRIP service
railway variables set K=V Set environment variable
railway logs              View deployment logs
```

9. FUTURE ROADMAP

-- eBay Browse API: Show real 'Similar Items' listings on item detail page with links to actual eBay listings

-- S