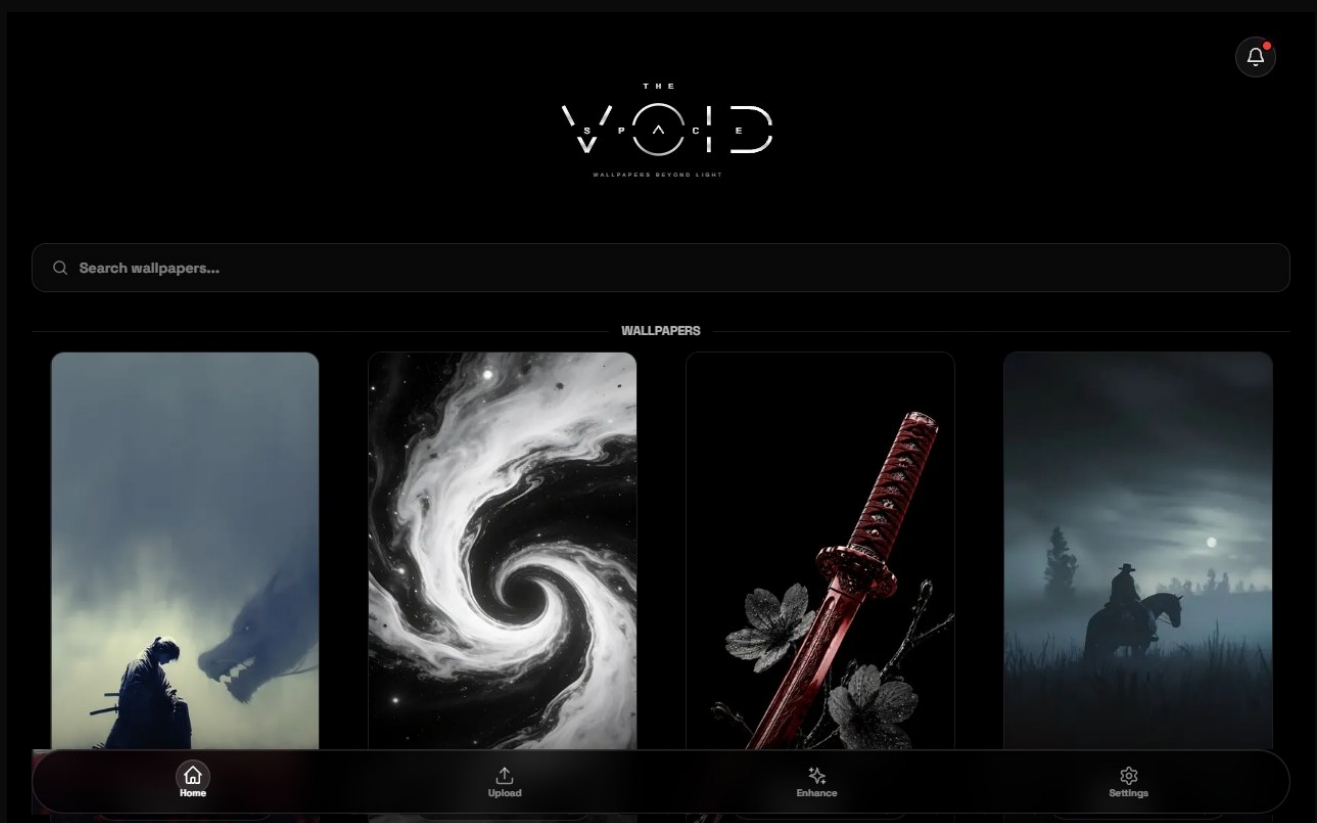


THE VOID SPACE

Syed Roshan Shareq
Project Owner and Developer

A technical reference for the wallpaper platform: architecture, accounts, image processing, moderation, storage and deployment.



Existing project gallery preview from my portfolio.

I built THE VOID SPACE around a straightforward creator workflow: submit an image, review it, and make the approved wallpaper available to the public. These notes trace the implementation behind that workflow and the constraints I need to keep in mind when maintaining it.

Contents

Project Boundary	4
Runtime Architecture	4
Files and Ownership	6
Startup and Storage Modes	6
Frontend Initialization and State	7
Authentication and Email Verification	8
Sessions and Account Security	9
Data Model and Storage Boundaries	11
Upload and Preview Workflow	12
Moderation and Media Variants	12
Gallery and Search	13
Download Pipeline	14
Full-Screen Wallpaper Preview	14
Enhancement Engine	15
Enhancement Queue and Result Lifecycle	16
Creator Profiles and Sharing	17
Feedback, Notifications, and Email Alerts	17
Administration	18
Security Boundaries and Failure Modes	19
Configuration Reference	20

Project Boundary

I built THE VOID SPACE as a mobile-first place to discover dark wallpapers and give creators a profile of their own. The main workflow is deliberately simple: browse, preview, upload, review, and publish. Accounts, image processing, feedback, and moderation sit behind that workflow rather than becoming separate products.

The project is a full-stack JavaScript application. A native Node.js HTTP server serves the frontend and API from the same origin. Supabase supplies PostgreSQL and object storage in production. Brevo sends transactional verification codes and admin alerts. Cloudinary supplies the optional AI restoration path; Sharp validates images, generates variants, and runs the built-in restoration path.

There is no Express, React, TypeScript, bundler, migration runner, or formal test suite in the archive. Node's built-in modules handle routing, files, cryptography, network interfaces, and startup. The only declared application packages are @supabase/supabase-js 2.45.4 and Sharp 0.33.5. Cloudinary and Brevo are reached through HTTP rather than provider SDKs.

These notes describe the supplied ZIP's code, not a production configuration that cannot be seen in the archive. Its README and change log contain older folder names, version labels, and descriptions. The package itself still declares version 1.0.0. I use the implementation as the source of truth wherever those labels or descriptions disagree.

Main User Journeys

Visitor	Creator	Administrator
Browse approved wallpapers	Create an account and verify email when enabled	Authenticate separately at /admin
Preview lock and home screens	Submit original images for approval	Approve, reject, rename, or remove wallpapers
Search wallpapers and public profiles	Manage avatar, password, email, and uploads	Manage creator badges and upload exemptions
Read notifications and legal pages	Enhance an image and download a temporary result	Reply to feedback and publish announcements

The browser's guest-download allowance and the server's download authorization currently disagree. That behavior is described in Download Pipeline; it is not treated as a working guest entitlement.

Runtime Architecture

The server is one large file with a manual route dispatch chain at its end. Helpers above that chain provide auth, storage abstraction, image handling, profile rendering, notifications, feedback, and administration. The frontend is split by responsibility but still uses shared global functions and variables.

Browser: HTML + CSS + shared JavaScript files
|
| same-origin fetch requests + HttpOnly cookies

```

v
Node HTTP server
|-- auth / OTP / sessions / origin checks
|-- wallpaper upload / media / moderation
|-- profiles / feedback / notifications
`-- enhancement queue / temporary results
    |
    |-- Supabase PostgreSQL + public Storage buckets
    |-- Brevo transactional email
    |-- Cloudinary signed image uploads + transforms
    `-- Sharp validation / variants / local restoration

```

Local mode substitutes JSON files and filesystem images.
Some local settings remain process-memory only.

Request Lifecycle

The HTTP callback applies security headers, parses and decodes the URL path, checks origin for state-changing API requests, and dispatches the first matching method/path. Dynamic paths extract IDs with regular expressions. Unmatched GET requests are passed to static serving; other unmatched methods return 405. Unknown API GETs consequently fall through to a plain static 404 rather than a universal JSON API error.

Protected handlers call `requireCreator()` or `requireAdmin()`. Many storage-sensitive handlers also wait for backend readiness. Image requests may return binary data, while `sendJson()` supplies content length, JSON content type, and no-store caching unless explicitly overridden.

There is no central middleware framework or transaction layer. Each handler owns its validation, authorization, persistence steps, and failure compensation. A change to a shared helper can therefore affect many routes even when the route chain itself is unchanged.

Deployment Topology

The archive's Render blueprint runs `npm install` and `npm start` on a free Node web service. The same process serves `public/`, `/admin`, API routes, media, and generated profile-sharing pages. No separate Netlify frontend is configured in this archive. A separate frontend host would need deliberate API-origin, cookie, CORS, and CSRF integration changes.

Files and Ownership

```

project/
|-- server.js          HTTP backend and service integrations
|-- package.json       Start script and runtime dependencies
|-- package-lock.json  Locked dependency graph
|-- render.yaml        Render service and environment settings
|-- start-mobile.bat   LAN-bound local startup
|-- open-firewall-port.bat Optional Windows firewall rule
|-- supabase/schema.sql Tables, constraints, indexes, triggers, RLS
|-- README.md          Handoff notes and historical change entries
|-- CHANGELOG.md       Older interface revision notes
`-- public/
    |-- index.html      Main tabbed application
    |-- admin.html / admin.js Separate moderation application
    |-- styles.css      Public/admin UI and responsive overrides
    |-- app-state.js     Shared state, account cache, tab navigation
    |-- app-search-auth.js Search, login, signup, OTP, Google flow
    |-- app-settings.js  Feedback, notifications, security screens
    |-- app-profile.js   Profile rendering, edits, sharing, init()
    |-- app-editor.js    Selection, canvas previews, uploads, cache
    |-- app-enhance.js   Enhancement requests and result lifecycle
    |-- app-gallery.js   Cards, downloads, full-screen previews
    |-- app-boot.js      Calls init()
    |-- script.js        Legacy pointer comment; not loaded by index
    |-- terms.html / privacy.html
    `-- manifest, crawl files, SVG brand and social assets

```

I keep the account cache separate from credentials. The browser's `localStorage` user object holds display information and capability flags, not a session password or the service-role key. `creatorHeaders()` now returns an empty object. Account ownership comes from the server cookie rather than a creator ID supplied by the browser.

The single-file backend and global frontend make the project easy to serve, but they create coupling. Shared helper names, script order, HTML IDs, and CSS override order form real contracts. They should be traced before making local edits.

Startup and Storage Modes

The server reads configuration once at startup. PORT defaults to 4173 and HOST to 0.0.0.0. It attempts to require Supabase and Sharp, configures Sharp memory limits if available, resolves local paths, creates session/OTP/result maps, and chooses a storage mode.

Condition	Startup Behavior
Supabase URL and service-role key both present	Create server client and bootstrap database checks
Render detected but Supabase not configured	Record a setup error; protected data operations report it
No Supabase configuration outside Render	Create local folders and JSON files
No admin password or password hash supplied	Generate a runtime password and print it in server logs

The Supabase client disables SDK session persistence and automatic token refresh. It is a server service-role client, not the client used for browser Google OAuth. Bootstrap checks app_settings, wallpapers, and users, then schedules startup maintenance.

The server does not execute schema.sql automatically. A reachable Supabase project is insufficient if the tables, columns, or buckets have not been prepared.

Local Paths and Persistence

THE_VOID_DATA_DIR overrides the data location. Otherwise Windows uses LOCALAPPDATA/THE-VOID/data; XDG systems use XDG_DATA_HOME/the-void/data; other systems use the user's home .the-void/data. This location is outside the extracted project by default, so use an explicit isolated directory for experiments.

Local users.json and wallpapers.json persist accounts, password hashes, wallpaper rows, and active user-session fields. Originals, profile pictures, and variants persist as files. writeFileAtomic() writes a temporary file, fsyncs it, and renames it. Promise queues serialize user-file and wallpaper-file mutations within this process.

General local app settings use localAppSettings, a Map. Feedback, notifications, legal-consent records, AI usage, lockout records, and most setting-based overrides do not become a persistent settings file. Upload exemptions have additional persistence on local user rows, while active sessions also use user-row fields. Local mode is therefore useful for development but is not behaviorally identical to Supabase.

Supabase Startup Maintenance

Maintenance can seed bundled wallpapers, import untracked Storage objects into rows, and backfill missing variants. The ZIP contains no bundled data directory, so the seed path normally has nothing to import. Automatic Storage synchronization is enabled by default.

Untracked originals are imported as approved wallpapers with a generic creator and no creator ownership. thumbs/ and previews/ are excluded. This is a recovery mechanism, not a moderation queue. Deleting a database row while leaving its original object can allow that file to reappear as approved at a later startup. Database and object cleanup need to be considered together.

Frontend Initialization and State

The browser loads the local Supabase UMD client first, then app-state, app-search-auth, app-settings, app-profile, app-editor, app-enhance, app-gallery, and app-boot. app-boot calls init() after the globals are available. These are classic scripts, not imported ES modules; rearranging them can break cross-file calls.

init() marks the Home tab, binds account and bottom navigation, handles an optional Google return, hydrates the current profile, builds preview presets, attaches upload/enhancement/preview handlers, loads approved wallpapers, synchronizes preview controls, and opens a profile from an /@username path when applicable.

Account Cache Versus Authority

`getUser()` reads `localStorage`. `setUser()` stores creator identity, email/display fields, avatar URLs, and capability flags returned by the server. `hydrateProfile()` calls `/api/profile` with same-origin cookies and updates or clears the cache. A network failure can retain cached identity, but a later protected API request still has to pass server authentication.

The `browserKey` is a persisted random identifier used in device-related login checks. It is not a secret credential, a hardware identifier, or an unchangeable device binding. Deleting browser storage can change it.

Tabs, Modals, and Responses

Home, Upload, Enhance, and Settings are panels in one document. Switching tabs closes relevant overlays, resets window scrolling, updates active/aria-selected state, and changes body state. Modal scroll locking is reference-counted, with a `pageshow` recovery helper to clear stale locks.

A `MutationObserver` normalizes status text, assigns success/info/warning styling, and clears completed responses after 10 seconds. Active enhancement messages are exempt while processing. Transient text and skeletons are presentation states; they are not separate backend jobs or delivery confirmations.

Authentication and Email Verification

Username are normalized with Unicode NFKC, trimmed, lowercased, stripped of a leading `@`, and converted to a restricted identifier. They must start with an alphanumeric character, contain 3-24 characters, and use letters, numbers, underscore, dot, or dash. The database stores a generated lower/trimmed `creator_name_key` with a unique index and blocks username changes through a trigger.

Emails are normalized to lowercase and checked against a configured domain allowlist. Passwords need at least eight characters with uppercase, lowercase, number, and symbol. `passwordRecord()` creates a random 16-byte salt and a 64-byte scrypt hash. Verification uses `timingSafeEqual` after deriving the supplied password with the stored salt. Hashing is synchronous and occupies the Node event loop during the derivation.

Signup Sequence

1. Collect username, email, password, browser key, and legal agreement.
2. Normalize and validate fields; check username and email uniqueness.
3. If OTP is enabled, reserve send capacity and create a six-digit cryptographic code.
4. Store the password hash/salt, code hash, consent, attempts, and expiry in a pending in-memory record.
5. Send the code through Brevo; release the send reservation if delivery fails.
6. Verify the submitted code, increment wrong attempts, and recheck uniqueness.
7. Insert an email-auth user, record legal consent, queue an admin alert, and create the application cookie session.

The code default is `THE_VOID_EMAIL_OTP_ENABLED=false`; `render.yaml` explicitly sets it true. A fresh local installation can sign up without email delivery, while the blueprint's production signup requires a configured Brevo sender. The direct signup route returns 428 when verification is required. Legacy `/api/create-user` delegates to the same signup handler.

OTP Limits

Setting	Default
Code validity	10 minutes
Resend delay	60 seconds
Wrong attempts	5
Send window	1 hour
Sends per window	5 per tracked IP/browser key
Pending records	1,000 maximum

OTP send records and pending codes live in memory, including on Supabase deployments. Restart invalidates outstanding codes and resets those rate records. The stored OTP is a scoped hash, not the plain code. Provider send success only means the API accepted the send, not that the mailbox delivered it.

Legal Consent and Google Sign-In

Signup requires an explicit legal acceptance flag. Consent versions, timestamp, and a scoped IP hash are stored under `user_legal_consents`. The application is not enforcing an old one-account-per-IP signup rule: new account rows deliberately leave legacy signup lock hashes empty.

Google OAuth is disabled by default. It requires the enable flag, Supabase backend configuration, an anon key, and Supabase's Google provider/redirect configuration. `/api/auth/config` exposes the URL and anon key only when Google is enabled. The browser completes OAuth through the Supabase client; the backend verifies the access token and provider before issuing its own application cookie. Existing eligible password profiles can link Google; new Google profiles still need a username and legal agreement.

Sessions and Account Security

User cookies use `the_void_user_session`; admin cookies use `the_void_session`. Tokens are cryptographically random and cookies are `HttpOnly`, `SameSite=Lax`, `Path=/`, with `Secure` when the request is considered HTTPS. The default session lifetime is 30 days. User token hashes are stored with creator and browser-key state; raw user tokens also have an in-memory cache.

One Active Creator Session

The active-session registry is persisted in Supabase app_settings or in local users.json. Login checks the existing session and blocks a different browser while its activity is fresh. Same-browser reauthentication can replace the token. Protected creator requests compare the current active token hash, refresh activity, and load the owner from storage.

After restart, a user cookie can be rehydrated by matching its hash to the persisted registry. Admin sessions cannot: their session Map is memory-only. Single-device checks are application rules based on a browser key and timestamps, not proof of physical-device identity.

Login Protection

Role	Wrong Attempts	Lock Duration	Scope
Creator	4	30 minutes	Browser/device-derived key
Admin	2	3 hours	Browser/device-derived key

Lockouts use settings-backed records in Supabase; local general settings are memory-only. This is targeted login protection rather than a universal rate limiter on every endpoint. Password-reset lookup returns distinguishable account-not-found errors, so it should not be described as preventing account enumeration.

Password, Email, Logout, and Deletion

Signed-in password changes require the current password and validate the replacement. That path does not explicitly clear existing sessions. Password reset sends an OTP to the stored account email and, after successful verification, clears in-memory and persisted user sessions before asking for a new login. Profile email changes require the current password and an OTP for the new, unique, allowed email.

User logout removes its token and attempts to clear the persistent active record, then expires the cookie. Admin logout only sends an expired cookie; it does not remove the old token from the server Map. A saved admin cookie can therefore still authorize until expiry or restart. This was confirmed in the isolated local check.

Account deletion requires a password for password-capable profiles or a typed username for passwordless profiles. It removes owned wallpaper rows, the user, associated images where possible, session state, and badge overrides. Database and Storage removal are sequential rather than one atomic transaction. Some unrelated setting entries keyed by creator can remain; deletion should not be described as a transactional purge of every JSON setting.

Data Model and Storage Boundaries

PostgreSQL Tables

Table	Purpose	Important Fields
users	Creator identity and credentials	creator_id, creator_name, auth_type, email, hash/salt, Google linkage, avatar paths, timestamps
wallpapers	Image ownership and moderation	id, title, creator label, creator_id, storage_path, public_url, mime, status, timestamps
app_settings	Structured operational state	key, JSONB value, updated_at

users.creator_id is a text UUID-style identity; wallpapers.id is a PostgreSQL UUID. wallpapers.creator_id references the user with ON DELETE SET NULL. Application account deletion explicitly removes uploads first; the foreign-key behavior alone would preserve rows if a user were removed directly in SQL.

The allowed auth types are password, email, guest, and google for compatibility. This does not mean public anonymous users can call creator-only upload routes. Statuses are pending, approved, and rejected. Indexes cover normalized usernames, emails, Google auth IDs, wallpaper storage paths, status/newest sorting, and creator/newest sorting.

Updated-at triggers maintain timestamps on all three tables. A username trigger rejects changes beyond case/trim-equivalent spelling. The SQL is an additive setup script with constraint/trigger recreation, not a numbered migration history.

Operational Settings

Keys hold active sessions, creator/admin login protection, upload exemptions, badge overrides, AI usage timestamps, legal consents, notifications, feedback, and seed markers. Supabase helpers read and replace whole JSON objects. Concurrent read-modify-write requests can overwrite each other's changes; this is not row-level locking or transactional accounting for each individual entry.

RLS and Public Objects

RLS is enabled on the three application tables. The script grants service_role access; the browser does not directly write those tables. Server authorization remains essential because the backend uses a service-role client rather than a per-user JWT for database operations.

The wallpapers bucket is public with a 12 MiB limit, and profile-pics is public with a 5 MiB limit. Both allow JPEG, PNG, and WebP. Storage SELECT policies permit public reads for every object in the bucket, without consulting wallpaper status. The application API filters pending images, but a known pending object's public URL can still expose the original. Public-bucket access and approved-list visibility are different security boundaries.

Custom bucket environment names require matching SQL or separately created buckets/policies; changing the env alone does not rename the defaults in schema.sql.

Upload and Preview Workflow

The Upload screen asks for an image and a title, and uses the signed-in creator name. It accepts drag/drop or file selection, creates a preview object URL, attempts browser decoding, and retains the selected original as a data URL for submission. Browser preview decoding is more permissive than the backend's accepted JPEG/PNG/WebP contract.

Canvas Simulation

The preview canvas is 1080 x 1920. Lock-screen and home-screen presets draw the image, a dark tone overlay, status details, clock or icons, and an optional guide grid. Zoom uses a cover-scale basis with a 0.05-2.2 factor; offset sliders reposition the drawing. These are presentation previews, not a guarantee of a device's real launcher layout.

`submitWallpaper()` sends `selectedUploadDataUrl`, not `drawEditedWallpaperOnly()`. The visible crop/zoom/position adjustments and simulated clock/icons are not baked into the uploaded wallpaper. The original selected image is what the current submission path transmits. The export helper exists but is not used by that submit function.

Server Validation and Ownership

1. Require the creator cookie and refresh the active session.
2. Check the rolling 24-hour upload count, normally four, unless exempt.
3. Read a bounded JSON/base64 body; parse the accepted image data URL.
4. Ask Sharp for metadata, reject unsupported formats or invalid dimensions, and enforce pixel limits.
5. Derive the owner ID/name/auth type from the session, not a caller's free-form creator label.
6. Save the original and create a pending row; queue an admin email alert.
7. On storage/row failure, attempt to remove the newly saved object or file.

The raw original limit is 12 MiB; the body ceiling includes roughly 1.4x base64 overhead plus 1 MiB. Validation limits a dimension to 12,000 pixels and total input to 50 million pixels for uploads. `preserveOriginal:true` returns the original buffer after metadata checks instead of re-encoding it. Metadata validation is not identical to fully normalizing, stripping metadata, or rewriting every original.

The four-upload check counts rows created during the rolling window across moderation states. It happens before insertion and is not an atomic quota reservation; simultaneous submissions can race. There is no server requirement that every upload itself be 9:16. The derived gallery variants are center-cropped to that ratio.

Moderation and Media Variants

New uploads are pending and absent from the public wallpaper list. The owner's profile lists their status but gives pending/rejected uploads no media URL. `/media/:id` serves non-approved images only to an admin; the owner alone does not get pending-media access through that handler.

Approval generates the small images before setting the row approved. In Supabase the original is downloaded, variants are uploaded, and then the status changes. In local mode variants are written to files before the JSON row changes. Pending-upload submission itself does not generate them.

Asset	Dimensions	Encoding	Location
Original	Source dimensions	Original bytes/format	uploads/id.ext or local uploads
Thumbnail	360 x 640	WebP quality 72	thumbs/id.webp
Preview	720 x 1280	WebP quality 78	previews/id.webp

Variants rotate for orientation and resize with fit:cover and center position. Missing variants fall back to the original at the media route. Startup maintenance can regenerate missing approved variants.

Supabase rejection changes the status then removes original and variant objects. Local rejection removes variants but retains the original. The admin Delete action is also not identical to owner deletion: it marks the row rejected and removes its files rather than deleting the row. Owner deletion removes the row and attempts object cleanup. Creator deletion has a broader cleanup path.

The Supabase approved list has a short in-memory cache, default 10 seconds. It also sends max-age=10 and stale-while-revalidate=60, and can return the last cached list with stale:true on a later database failure. Mutations invalidate the process cache. This serves the full selected set, not a paginated wallpaper API.

Gallery and Search

The gallery can first render a localStorage list cached for six hours, storing at most 96 wallpaper records. It then fetches /api/wallpapers to refresh. This is metadata caching, not a service-worker cache or a complete offline image library.

renderWallpapers() creates every card in the returned list at once. Images have explicit 360 x 640 dimensions, async decoding, eager/high-priority loading for the first four, and lazy loading thereafter. A load-more sentinel cleanup helper remains, but the active renderer is not incremental pagination or DOM virtualization.

Tapping a wallpaper reveals Preview and Download actions; creator names open public profiles. A delegated handler dismisses other open action overlays. Download state belongs to the card/preview that started it and marks that surface aria-busy.

Two Search Paths

Plain text filters title and creator text in the already loaded approvedWallpapers array. It does not issue a full-text database search. An @ prefix switches to creator search, debounced by 220ms, through /api/users/search?q=.... The server normalizes a prefix query, sorts usernames, and returns at most eight public results. Admin creator search is separate and returns up to 50 entries.

Download Pipeline

The HTML still uses real anchor elements, but the current delegated click handler prevents their native navigation and starts a streamed fetch. `wallpaperDownloadUrl()` always builds `/api/download/:id` with a timestamp; it ignores the public `downloadUrl` that the server DTO may supply.

1. Prepare the link and check the browser guest allowance.
2. Show progress and mark the initiating card busy.
3. Fetch the original from the same-origin download API.
4. Read the response stream and count actual received bytes.
5. Show a percentage when `Content-Length` is present, or real byte counts otherwise.
6. Build a Blob, create an object URL, and trigger a temporary download anchor.
7. Hold completion briefly, restore controls, and revoke the URL later.

The server responds with exact `Content-Length`, attachment disposition, octet-stream content type, and `X-Original-Content-Type`. Supabase mode downloads and buffers the original on the Node server before responding; local mode streams the file. The browser also buffers chunks before making its Blob. Thus progress is based on response transfer, not the upstream Supabase retrieval phase or an operating-system save confirmation.

Guest Quota Mismatch

The browser reserves two downloads per UTC ISO week in `localStorage`, incrementing before the network request. The download API independently requires a creator session. A guest with allowance remaining therefore reaches a 401 and is prompted to log in; the reserved local count is not refunded. Clearing storage or storage failures also bypass this client-only accounting.

The handoff's older `native/direct-public` download description is not the active behavior in this snapshot. Synthetic Blob download and same-origin server proxying are currently used. Restoring real guest downloads requires aligning authorization and client/server counting rather than only changing the displayed quota message.

Full-Screen Wallpaper Preview

The preview displays lock-screen and home-screen canvases using the same drawing helpers as Upload, but without the editable crop controls or guide grid. It loads the preview variant and falls back to the original. A load token prevents a late image from replacing a closed or newly opened preview.

The loading presentation lasts at least 1.3 seconds even if the image arrives sooner. That minimum is visual timing, not evidence of ongoing network work. Closing invalidates the token, clears canvases and download controls, and releases its scroll lock.

Viewport Sizing

visualViewport height, width, and offsets are written into CSS variables so the modal fits the screen area actually available. A requestAnimationFrame calculation uses the remaining track size to choose a 9:16 frame no wider than 340px. Resize, orientation, and visual-viewport scroll/resize events resynchronize it.

Standalone/fullscreen modes, legacy iOS installed state, and collapsed browser chrome can switch from the top close symbol to bottom Back controls. A hidden 100svh probe estimates browser-chrome changes; 24px enter and 12px exit thresholds reduce flicker, and pinch zoom does not toggle the heuristic. Some browsers do not expose chrome changes through these measurements and will retain the regular close control.

Lock and home panels use a horizontal scroll track with centered targets and explicit selector buttons. In the Back layout, Download precedes Back. This is a responsive browser preview, not native wallpaper installation or a mobile OS API integration.

Enhancement Engine

Enhancement is a creator-only workflow. The browser accepts PNG/JPEG/WebP up to 12 MiB, stores the input data URL, checks the currently available method, and submits to /api/enhance-wallpaper. Selection and request tokens plus an AbortController prevent stale results from updating a new selection.

The server returns image bytes and metadata headers, including dimensions, provider, mode label, quota state, remaining AI usage, reason, and a temporary download URL. The browser builds a preview object URL and switches controls to download mode. Status messages advance on six-second timers; they are not server-sent per-stage telemetry.

Provider Decision

Cloudinary is available only when the configured provider is cloudinary, CLOUDINARY_URL parses correctly, and the AI limit is positive. A creator normally has three successful Cloudinary uses per rolling 24 hours. Exhausted quota, missing configuration, or a provider failure selects the local Sharp path. Only a successful Cloudinary result attempts to record AI usage.

Cloudinary configuration is parsed from cloudinary://key:secret@cloud. The server signs sorted upload parameters, submits a multipart image to the image/upload API, and asks for generative restore, with upscale where appropriate. The source targets a 4096px long edge with at most 4x enlargement. Cloudinary uploads use a readyvoid/enhancements folder and unique public IDs.

Retry handling includes transient network failures and status codes 408, 420, 423, 429, and selected 5xx responses. An unsupported upscale/restore combination can fall back to restore alone before falling back to Sharp. The archive contains no Cloudinary asset-destroy call; expiry of the app's temporary result is not deletion of provider-uploaded source assets.

Local True-Tone Restoration

The active local mode uses the ultra settings: 4096px portrait or landscape long-edge target, max scale 4, quality 98. Smaller inputs can stop below 4096 when max scale is reached; very large inputs can be reduced. Natural and detail configurations exist but are not selectable in the active local processing function.

Sharp handles orientation, resize, colorspace, raw-pixel decoding, and final encoding. The custom restoration code computes luminance from RGB, generates fine/medium box-blurred luminance, estimates flat-region noise, and applies guarded denoise/deblur/detail deltas. Highlight and shadow guards limit changes near black and white. A statistics pass reduces contrast drift and clamps final RGB adjustments.

The Cloudinary result passes through a separate reference-tone lock: original pixels are resized to the enhanced dimensions, AI high-frequency detail is blended against original low-frequency tone, and bounded luminance changes are applied to the reference RGB. This attempts to preserve the dark wallpaper's color and brightness character; it is not a lossless transform or a guarantee of recovered real detail.

JPEG, PNG, and WebP outputs use Sharp encoders. JPEG transparency is flattened against black. Enhanced output is limited to 48 MiB. Increasing dimensions creates more pixels and memory pressure even when the compressed output is small.

Enhancement Queue and Result Lifecycle

`runEnhancementJob()` queues work and `pumpEnhancementQueue()` starts jobs up to the concurrency limit. Default concurrency is one. The code's waiting limit defaults to two; the Render blueprint sets it to four. A full queue returns 429 with a 15-second Retry-After. Sharp defaults are 32 MiB cache, 64 items, and one processing thread.

The queue encloses request-body reading as well as processing. It is an in-process promise queue, not a durable job system. A browser abort does not by itself cancel a provider operation already running on the server, and a restart loses all queued jobs. Base64 strings, decoded buffers, raw RGB arrays, blur arrays, output buffers, and temporary cached results can coexist; limiting Sharp cache alone does not bound total Node memory.

Temporary Download Cache

Enhanced results are stored in a process Map with a random token, creator ID, bytes, dimensions, MIME, and filename. Defaults are two cached results and a two-minute TTL. The limit is global to the process, so another creator's new results can evict an older one before its own TTL. A pruning timer also expires entries.

`/api/enhance-download/:token` requires the creator session and matching owner. Release removes the entry; expired/missing downloads return 404. The browser separately expires its preview/result after roughly two minutes, revokes object URLs, and releases server resources on reset or page exit. Provider assets are not covered by this lifecycle.

The frontend sends `requestId` and can retry once after a transient network failure, but the backend does not use that ID as an idempotency key. A response lost after successful processing can lead to duplicate work or another quota use on retry. AI usage JSON updates also have no cross-instance reservation transaction.

Creator Profiles and Sharing

The private profile API returns creator display data, capability flags, all owned upload statuses, and counts. Public profile APIs omit email and credential fields and expose only approved image entries, but counts are derived from all statuses. The distinction is content visibility, not complete concealment of pending/rejected totals.

Avatars accept a 5 MiB source, validate a 20-million-pixel ceiling, resize inside 1600 x 1600 without enlargement, and re-encode as WebP. A unique object name helps avoid stale avatar caching. Old avatar removal is attempted after replacement, with best-effort cleanup on failure.

Creators can rename their own wallpaper title and delete their own uploads. They cannot rename their immutable username. Owner queries combine wallpaper ID and creator ID, preventing ownership from being changed by a caller-supplied name. Admin edits can change a wallpaper's creator display label without changing creator_id, so the label and ownership can diverge.

Badge Rules

Approved Count	Automatic Badge
0	No earned badge
1 or more	VOID SPARK
15 or more	RIFT SHAPER
30 or more	THE VOID SPACE ARCHITECT

Admin overrides can select a tier, none, or automatic. Counts are based on approved uploads, not total submissions.

Share Pages and Images

/@username, without the space, serves the main application HTML with dynamic title, canonical, description, and Open Graph/Twitter profile metadata. The frontend opens the matching profile modal after startup. /og/profile/username.png generates a 1200 x 630 banner from SVG, current avatar, username, badge, and upload count, then rasterizes it with Sharp.

Local avatar bytes are read directly. Remote avatar retrieval has a bounded request phase and size validation; failures fall back to an initial. Shared URLs are versioned from avatar-related state to encourage refreshed previews, but social platforms control their own cache. Root metadata and structured-data content remain separate from these per-profile replacements.

Feedback, Notifications, and Email Alerts

Feedback is a signed-in creator thread, not an anonymous form or Netlify Forms integration.

/api/feedback returns only the current creator's messages. Each submit creates a new UUID item with type feedback or bug, message, identity, timestamps, new status, and optional admin reply fields.

Messages require at least 10 characters and are trimmed/capped at 1,200. A creator can send two per rolling 24 hours. Up to 300 items are retained in the global setting. Admin reply marks an item reviewed and records reply/review times; marking reviewed is also a separate action. Reviewed items expire seven days after review, cleaned on reads/writes and by an hourly timer. This is not an archival support-ticket store.

Feedback is stored under `app_settings.user_feedback` in Supabase. Local feedback disappears on restart because its setting lives in memory; the isolated restart check confirmed that difference.

Notifications

Administrators create, update, or delete global announcements. Title is capped at 80 characters and message at 600. Up to 80 normalized items are retained. Active entries appear publicly; silent entries can be listed without being selected as a new popup announcement. The browser polls every 30 seconds and tracks seen IDs in `localStorage`. Display dates are formatted DD/MM/YYYY.

Popup swipe dismissal is a browser gesture and does not acknowledge a notification to the database. The app also tracks a six-hour login-reminder interval locally. There is no WebSocket, service worker push subscription, or operating-system push delivery.

Transactional Admin Email

New account, pending upload, and feedback events can trigger Brevo messages to the configured admin address. Send failures are logged and do not roll back the completed user action. This is best-effort fire-and-forget delivery, not a persisted outbox with guaranteed retries. The same verified sender/API configuration supports OTP but the two paths have different failure semantics.

Administration

`/admin` serves a separate HTML/JavaScript application. Its login is controlled by `THE_VOID_ADMIN_PASSWORD` or a supplied SHA-256 password hash; it is not a Supabase user role. With neither value, a new random password is printed for that process run. Admin tokens live in memory and expire on restart.

The interface separates pending approvals, existing wallpapers, creator management, notifications, and feedback. Every admin action needs the server cookie even if someone loads the public admin HTML. The storage diagnostic reports configured mode, readiness, bucket names, Google enablement, and configuration booleans; it does not reveal the service-role secret.

Operations and Effects

Action	Backend Effect
Approve	Generate variants, set approved, invalidate list cache
Reject	Set rejected, clear variants; Supabase also removes original
Admin Delete	Retain rejected row, remove image objects/files where possible
Rename	Update title/creator label; not ownership
Owner Delete	Remove own row and attempt file cleanup
Delete Creator	Remove user and uploads, clean images/session/badge state
Upload Exemption	Bypass normal rolling upload count for selected creator
Badge Override	Replace or suppress automatic badge selection
Cleanup Locks	Clear selected legacy hashes and orphaned active sessions

Existing wallpaper loading uses the public approved-list route, whereas pending loading has its own admin API. Rejected rows retained for history are not a general browsable rejected-image gallery. The old lock-cleanup helper remains a maintenance tool; new signup does not impose that old one-account-per-IP rule.

Security Boundaries and Failure Modes

The server includes origin checks, HttpOnly sessions, password hashing, input limits, image metadata validation, owner-scoped mutations, admin-only controls, path confinement, and restrictive resource headers. Those controls have different scopes; the documentation should not compress them into a claim that every route is rate-limited or every stored object is private.

Origin and Browser Resource Policy

POST/PUT/PATCH/DELETE requests beneath `/api/` require an Origin or Referer origin matching expected host/protocol or an allowed configured origin. The server derives expectations from Host, forwarded headers, Render's external URL, and `THE_VOID_ALLOWED_ORIGINS`. This is origin-based CSRF protection, not a separate per-form synchronizer token.

CSP permits same-origin scripts, inline/external Google styles, broad HTTPS image/connect resources, and blocks objects and framing. Other headers include nosniff, frame denial, strict referrer policy, feature permissions, and HSTS. Forwarded host/protocol/IP headers are trusted directly, so deployment needs a proxy that controls those headers. The code itself does not define a trusted-proxy allowlist.

Constraints Visible in This Snapshot

- Public bucket reads are not moderated by wallpaper status, even though public API lists are filtered.
- Guest download UI does not match the creator-only API.

- Admin logout does not revoke the token held server-side.
- Shared settings are whole-JSON read/modify/write operations, allowing lost updates under concurrency.
- Upload quota checks precede insertion and can race; most endpoints have no general traffic limiter.
- Local settings have partial persistence and differ from production mode.
- Deletion and storage compensation are sequential, with possible orphans or partial failure.
- Startup recovery auto-approves untracked originals, affecting cleanup decisions.
- scriptSync and custom pixel loops can block the event loop; the enhancement queue is not a worker thread.
- Cloudinary source uploads have no matching deletion workflow in this code.
- Enhancement retries have no server idempotency check and provider requests have no explicit overall deadline.

These are concrete implementation properties to account for when maintaining the project. They do not require changing the supplied archive to document it accurately.

Configuration Reference

Configuration is read from `process.env`. The archive does not include `dotenv` loading or a secrets file; placing values in an unreferenced `.env` file alone will not load them. Use shell environment variables or the hosting dashboard. Keep `service-role`, `Brevo`, `Cloudinary`, `admin password`, and `hash secrets` on the server.

Core and Storage

Variable	Default / Requirement
PORT / HOST	4173 / 0.0.0.0 locally
SUPABASE_URL	Production project URL
SUPABASE_SERVICE_ROLE_KEY	Server-only production key
SUPABASE_BUCKET	wallpapers
SUPABASE_PROFILE_BUCKET	profile-pics
THE_VOID_DATA_DIR	Override local data location
THE_VOID_AUTO_SYNC_STORAGE	true unless explicitly false
THE_VOID_FORCE_BUNDLE_SEED	false; archive has no bundled data
THE_VOID_WALLPAPER_CACHE_MS	10000

Authentication and Email

Variable	Code Default / Notes
THE_VOID_ADMIN_PASSWORD	Generated if password/hash absent
THE_VOID_ADMIN_PASSWORD_HASH	Optional SHA-256 admin hash
THE_VOID_IP_HASH_SECRET	Configure a stable private scope secret
THE_VOID_SESSION_TTL_MS	30 days
THE_VOID_ACTIVE_SESSION_IDLE_MS	30 days
THE_VOID_ENABLE_GOOGLE_AUTH	false
SUPABASE_ANON_KEY	Needed if Google is enabled
THE_VOID_ALLOWED_EMAIL_DOMAINS	Common-provider allowlist
THE_VOID_EMAIL_OTP_ENABLED	false in code; true in blueprint
BREVO_API_KEY / BREVO_SENDER_EMAIL	Required for email delivery
BREVO_SENDER_NAME	THE VOID SPACE in code; THE VOID in blueprint
THE_VOID_ADMIN_EMAIL_NOTIFICATIONS	true unless explicitly false
THE_VOID_ADMIN_NOTIFY_EMAIL	Configured support/admin recipient

Quotas and Resource Settings

Variable	Code Default / Blueprint Difference
THE_VOID_UPLOAD_LIMIT_PER_24H	4
THE_VOID_USER_LOGIN_MAX_FAILED_ATTEMPTS	4
THE_VOID_USER_LOGIN_LOCK_MS	1800000
THE_VOID_ADMIN_LOGIN_MAX_FAILED_ATTEMPTS	2
THE_VOID_ADMIN_LOGIN_LOCK_MS	10800000
THE_VOID_AI_ENHANCE_PROVIDER	cloudinary; local selects built-in
CLOUDINARY_URL	Required for Cloudinary path
THE_VOID_AI_ENHANCE_LIMIT_PER_PROFILE	3 successful AI results / rolling day
THE_VOID_ENHANCE_CONCURRENCY	1
THE_VOID_ENHANCE_QUEUE_LIMIT	2 in code; 4 in blueprint
THE_VOID_ENHANCE_DOWNLOAD_TTL_MS	120000
THE_VOID_ENHANCE_DOWNLOAD_CACHE_LIMIT	2 globally
THE_VOID_SHARP_CACHE_MEMORY_MB	32
THE_VOID_SHARP_CACHE_ITEMS	64
THE_VOID_SHARP_CONCURRENCY	1
THE_VOID_ALLOWED_ORIGINS	Optional comma-separated deployment origins

OTP timing/send controls are listed in Authentication. Fixed limits such as two daily feedback messages, 300 feedback items, 80 announcements, and image dimension ceilings are constants rather than configurable entries in the blueprint.

Local Setup and Production Deployment

Local Development

Use Node.js 18 or newer; the dependency metadata declares that minimum and provider integrations rely on built-in fetch, FormData, and Blob. A currently supported Node release compatible with the locked packages is preferable for hosting.

```
# From an extracted project folder
npm ci
$env:THE_VOID_DATA_DIR = "$PWD\local-data"
$env:THE_VOID_ADMIN_PASSWORD = "your-local-admin-password"
```

```
$env:THE_VOID_ADMIN_EMAIL_NOTIFICATIONS = "false"
npm start
```

Open <http://localhost:4173> and `/admin`. If the port is occupied, set `PORT` before starting. Keep test data separate from existing user data. With no Supabase configuration and no Render detection, local fallback is selected and signup OTP defaults off. Image handling still requires Sharp; a booting server without that dependency is not a fully functioning upload/enhance installation.

`start-mobile.bat` binds all interfaces and prints LAN addresses. A phone on the same network can use that address if routing/firewall permits it. The optional firewall batch adds an inbound TCP 4173 rule and requires admin rights; it does not configure deployment, TLS, or a public tunnel. LAN HTTP also lacks some browser secure-context capabilities available on `localhost/HTTPS`.

Production Sequence

1. Prepare a Supabase project and run `supabase/schema.sql`.
2. Confirm the tables, unique indexes, triggers, RLS, and matching public buckets.
3. Create the Render web service from the blueprint or equivalent settings.
4. Add server-only Supabase, admin, and hashing secrets.
5. Verify Brevo sender/API configuration before enabling production OTP signup.
6. Configure Cloudinary only if that provider path is wanted; fallback works without it.
7. Set custom-domain origins and verify forwarded HTTPS/cookie behavior.
8. Deploy, check admin storage readiness, and exercise a full signup/upload/approval/download path.

Render installs packages at build time and starts `node server.js` through `npm start`. There is no compile step. The blueprint does not pin a Node version; package engines establishes a lower bound rather than a tested hosting release.

The public application's canonical domain is `thevoidspace.online`. Verify root/profile share URLs, robots/sitemap, and legal links after a domain change. Asset query strings are manually versioned, while static serving currently sends no-store. Hosting caches may introduce behavior beyond this source.

Restart and Backup Planning

Supabase retains database rows/settings and bucket objects. OTP records, admin sessions, enhancement queue/results, and process caches disappear on restart. User sessions can rehydrate through persistent token hashes. Back up PostgreSQL state and Storage originals together; one is not a substitute for the other. Local JSON files/images also need joint backup, while memory-only local settings cannot be recovered from those files.

Responsive UI, Accessibility, and App Metadata

The interface uses Inter/Space Grotesk from Google Fonts, dark surfaces, four-tab bottom navigation, full-screen task forms, and phone-shaped preview canvases. The stylesheet has extensive later override layers; current appearance cannot be inferred from its first declarations alone.

Native document scrolling is restored in the later rules. Safe-area insets, dynamic viewport sizes, and navigation clearance address mobile browser chrome and installed views. The manifest uses standalone mode and portrait orientation with an SVG maskable icon. Its name and Apple title use a zero-width string rather than a visible conventional app name.

There is no service-worker registration or offline fetch cache in the archive. A standalone manifest does not provide offline reliability or push notifications. Installation behavior remains browser-dependent, especially with the unusual name and SVG-only icon choice.

Buttons and many dialogs have labels and ARIA state; loading surfaces use skeletons and busy state. Some motion handlers inspect prefers-reduced-motion. However, there is no consistent modal focus trap/background inert policy, and the viewport sets user-scalable=no with maximum-scale=1. That zoom restriction is a concrete accessibility limitation; the document does not claim WCAG certification.

robots.txt permits public crawling while disallowing /admin and /api. This is a crawler instruction, not access control. The sitemap includes home and legal pages but does not enumerate creator profiles. llms.txt and ai.txt are descriptive discovery files, not model integrations. Root JSON-LD identifies the site/app/creator and does not make dynamic wallpaper data server-rendered content.

API Reference

All paths are relative to the Node server origin. State-changing API requests need the same-origin check in addition to the access listed below. Bodies are JSON unless the route returns image bytes. creator and admin denote application cookies, not client-supplied role fields.

Public and Authentication Routes

Method / Path	Access	Purpose
GET /api/wallpapers	Public	Full approved wallpaper list
GET /api/users/search?q=	Public	Username prefix search, max 8
GET /api/users/:username	Public	Public creator identity and approved uploads
GET /api/notifications	Public	Active global announcements
GET /api/auth/config	Public	OTP/Google flags; OAuth public config if enabled
POST /api/check-username	Public + origin	Normalized username availability
POST /api/auth/signup	Public + origin	Direct signup only when OTP disabled
POST /api/create-user	Public + origin	Legacy alias to signup
POST /api/auth/signup/request-otp	Public + origin	Prepare signup and send code
POST /api/auth/signup/resend-otp	Public + origin	Same request handler
POST /api/auth/signup/verify-otp	Public + origin	Verify code and create account/session
POST /api/auth/login	Public + origin	Email/username and password login
POST /api/auth/google	Public/creator + origin	Verify OAuth, create/link profile/session
POST /api/auth/password-reset/request-otp	Public + origin	Send reset code to stored email
POST /api/auth/password-reset/verify-otp	Public + origin	Change password and invalidate user sessions

Creator and Media Routes

Method / Path	Access	Purpose
GET /api/profile	Creator	Private profile and all owned upload statuses
POST /api/auth/logout	Cookie + origin	Remove user session and expire cookie
POST /api/auth/change-password	Creator	Verify current password and replace hash
POST /api/auth/email/request-otp	Creator	Validate new email and current password
POST /api/auth/email/verify-otp	Creator	Commit verified new email
POST /api/profile/avatar	Creator	Validate and store re-encoded avatar
POST /api/profile/delete	Creator	Confirm and delete account/uploads
POST /api/profile/wallpapers/:id/update	Owner	Rename own wallpaper title
POST /api/profile/wallpapers/:id/delete	Owner	Remove own wallpaper row/files
POST /api/upload	Creator	Store original and pending row
GET /media/:id?variant=thumb (or preview)	Public/admin	Approved image; admin for non-approved
GET /api/download/:id	Creator	Original attachment; guest is rejected
GET /api/enhance-method	Creator	Current provider/quota availability
POST /api/enhance-wallpaper	Creator	Queued binary image restoration
GET /api/enhance-download/:token	Result owner	Temporary enhanced image attachment
POST /api/enhance-download/:token/release	Result owner	Release cached bytes
GET /api/feedback	Creator	Own support thread
POST /api/feedback	Creator	New feedback/bug item with quota

Administrative Routes

Method / Path	Purpose
POST /api/admin/login	Admin password, device lock, cookie issuance
POST /api/admin/logout	Expire browser cookie; no token revocation
GET /api/admin/storage	Storage/config readiness diagnostic
GET /api/admin/pending	Pending moderation queue
GET /api/admin/download/:id	Admin original image attachment
POST /api/admin/wallpapers/:id/approve	Generate variants and publish
POST /api/admin/wallpapers/:id/reject	Reject and clean relevant files
POST /api/admin/wallpapers/:id/update	Title/creator display-label edit
POST /api/admin/wallpapers/:id/delete	Rejected-row retention and file cleanup
GET /api/admin/users?q=	Creator management list/search, max 50
POST /api/admin/users/:id/upload-limit	Toggle upload-count exemption
POST /api/admin/users/:id/badge	Set automatic/manual/none badge
POST /api/admin/users/:id/delete	Delete creator and owned uploads
POST /api/admin/users/cleanup-stale-locks	Selected legacy/session maintenance
GET /api/admin/notifications	All announcements
POST /api/admin/notifications	Create announcement
POST /api/admin/notifications/:id/update	Modify announcement
POST /api/admin/notifications/:id/delete	Remove announcement
GET /api/admin/feedback	All retained feedback
POST /api/admin/feedback/:id/review	Mark reviewed
POST /api/admin/feedback/:id/reply	Save reply and mark reviewed

GET /@username serves profile-aware HTML; GET /og/profile/username.png generates social imagery. /profile-pics/:filename serves local avatar files. /vendor/supabase.js exposes the installed UMD browser bundle. / and /admin serve their respective HTML files.

Response Conventions

200 indicates normal success; 201 indicates creation. Common errors include 400 validation, 401 missing/invalid auth, 403 origin rejection, 404 unknown/inaccessible media, 409 uniqueness or active-session conflicts, 423 login lock, 428 OTP requirement, 429 quotas/queue pressure, 500 storage/dependency failures, and 503 unavailable provider configuration. Not every helper emits every code, and oversized bodies can destroy the request connection rather than produce a clean JSON 413.

Maintenance and Troubleshooting

Changes That Need Coordinated Edits

- Auth fields: update browser forms/payloads, server normalization/DTOs, SQL constraints, and settings where relevant.
- New routes: add dispatch, origin-aware requests, authorization, bounded body handling, and appropriate JSON/binary response conventions.
- Upload behavior: align selected original versus canvas export, parser formats, pixel/file limits, variants, and owner DTO visibility.
- Storage cleanup: handle rows, originals, derivatives, profile files, cache invalidation, and recovery import together.
- Frontend globals: preserve load order and caller names; modules cannot be treated as isolated imports.
- Limits: distinguish code defaults, environment overrides, blueprint values, and fixed constants.
- UI changes: inspect final CSS layers and test real visual-viewport/safe-area conditions.

The source handoff requires a README change-log entry whenever application code, assets, schema, dependencies, deployment, or configuration are changed. This documentation task analyzes an extracted copy and produces a PDF; it does not modify or repack the supplied application archive.

Operational Checks

Symptom	First Checks
Render serves UI but APIs fail	Supabase env, schema columns, bucket names, bootstrap error
Local upload/enhance fails	npm ci, Sharp native support, image MIME and limits
OTP cannot send	Enabled flag, verified Brevo sender, API response, outbound restrictions
Profile blocked on another device	Browser key, active-session registry, expiry/activity, prior logout
Guest cannot download	Creator-only /api/download behavior versus frontend quota UI
Crop seems ignored after upload	submitWallpaper sends original, not canvas export
Approved tile shows original size	Missing derivative, approval failure, variant backfill
Removed image returns after restart	Remaining public object and automatic recovery import
Enhanced link expires too early	Global result eviction, two-minute TTL, server restart
AI result becomes local	Provider configuration, quota, retries, Cloudinary response
Local feedback disappears	General app settings are in-memory
Admin logout is not revocation	Cookie expiration without server Map deletion
Modal clipped on phone	visualViewport, safe-area, track bounds, later CSS overrides

Production validation needs a real OTP delivery, Supabase upload/approval/download, Cloudinary success/fallback, and backup/cleanup check. Local passing behavior alone does not certify external services or every mobile browser.

Verification Record

The documentation analysis included an isolated local runtime check using the locked dependencies, a temporary explicit data directory, local-only enhancement, disabled outbound admin email, and test accounts/images. The test server was stopped afterward. No production credentials or live user data were used.

Checks Performed

- Node syntax checks passed for server.js and all ten frontend JavaScript files.
- Home served successfully; unauthenticated admin diagnostics returned 401.
- A state-changing admin request without Origin/Referer was rejected with 403.
- Local signup, creator-cookie profile retrieval, and a pending image submission succeeded.

- Pending media remained unavailable to the owner through /media, and absent from the public list.
- Admin approval produced a 360 x 640 WebP thumbnail and exposed the approved entry.
- A creator download matched the test PNG's original bytes; the guest API returned 401.
- Public profile output omitted email, and profile HTML/social-image routes served successfully.
- The third feedback message in the test rolling window was rejected after two successful messages.
- Sharp local restoration returned an image, a creator-bound download token, and a working release path.
- Restart restored the creator session from local persistent fields while the local feedback thread was empty.
- Replaying the prior admin cookie after the logout response still authorized, confirming the revocation limitation.

Supabase, Brevo, and Cloudinary live integrations were inspected in source but not called with production credentials. Browser screenshots of every application workflow and a production security audit were not performed. The PDF's theme, typography, contents, and page layout were checked separately.

Source Navigation

Locations refer to the supplied archive and may shift after future edits. The compact files sometimes keep a complete function on one line, so references point to the relevant function rather than imply line length reflects simplicity.

Source Location	Responsibility
server.js:18	Core limits, configuration, runtime maps
server.js:198	Settings storage abstraction
server.js:347	Automatic badge thresholds
server.js:378	Admin and creator login protection
server.js:538	Security headers and origin checks
server.js:602	Local path resolution
server.js:637	Supabase bootstrap and recovery
server.js:828	Local storage preparation
server.js:897	Atomic writes and mutation queues
server.js:981	Data URL parser and image metadata validation
server.js:1063	Static serving and vendor client route
server.js:1101	Wallpaper DTO/download URL generation
server.js:1181	Signup field and legal validation
server.js:1255	Admin transactional email
server.js:1317	OTP hashing, send records and Brevo delivery
server.js:1471	script password records and cookie helpers
server.js:1527	Persistent active-session registry
server.js:1651	Session issuance and rehydration
server.js:1843	Signup OTP request and verification
server.js:1940	Password reset
server.js:2068	User login
server.js:2101	Profile email OTP
server.js:2205	User logout and Google config
server.js:2460	Private profile and account deletion
server.js:2556	Avatar validation and storage
server.js:2621	Creator search and public profile
server.js:2729	Dynamic profile HTML/social images
server.js:2925	Upload limits and creator authorization
server.js:2993	Original-preserving upload parser
server.js:3076	Natural-restoration pixel algorithm

Source Location	Responsibility
server.js:3142	AI reference-tone lock
server.js:3237	Active local ultra enhancement
server.js:3299	Cloudinary parsing, signing, retry and transforms
server.js:3527	Enhancement queue
server.js:3579	Temporary result cache and token authorization
server.js:3659	Enhancement endpoint/response headers
server.js:3719	Notification normalization and CRUD
server.js:3809	Feedback retention, quota and replies
server.js:3959	Local/Supabase upload writes
server.js:4036	Approved list and media serving
server.js:4215	Original attachment headers and variants
server.js:4283	Creator/admin downloads and admin login
server.js:4363	Admin creator operations
server.js:4712	Moderation and deletion behavior
server.js:4870	HTTP route dispatch
public/index.html:494	Frontend load order
public/app-state.js:19	Cached identity and shared UI state
public/app-search-auth.js:4	Search routing and client auth helpers
public/app-profile.js:299	init() entry point
public/app-editor.js:18	File decoding and preview controls
public/app-editor.js:75	Original upload payload and gallery cache
public/app-gallery.js:77	Download interception and guest accounting
public/app-gallery.js:208	Streamed download and Blob save
public/app-gallery.js:305	Viewport-aware preview mode
public/app-enhance.js:129	Timed progress messaging
public/app-enhance.js:631	Enhancement retries and result flow
public/app-settings.js:414	Notification polling
supabase/schema.sql:10	Creator table/constraints
supabase/schema.sql:83	Wallpaper ownership and statuses
supabase/schema.sql:122	Settings table and triggers
supabase/schema.sql:170	RLS, service-role grants and buckets
render.yaml:1	Hosted build/start and environment overrides