Project 2: Photo Editing Application

A4 - Architecture and High-Level Design

Krishhiv Mehra & Siddhant Rau

1) Client-Server Architecture

1.1 Flow

User (Browser: Chrome/Firefox/Edge/Safari) ↔ **Azure Front Door/CDN (images)** ↔ **FastAPI (Uvicorn/Gunicorn) on Azure Ubuntu VM** ↔ **Services**

- **Azure Blob Storage** (`originals/`, `variants/`, `backups/`)
- **Azure Database for PostgreSQL** (users, albums, photos, versions, tags, shares)
- **Azure Cache for Redis** (optional: sessions, rate-limit)

1.2 Component Responsibilities & Interfaces

Frontend - NextJS

- SSR/ISR for share pages; SPA for library, album, editor flows.
- Canvas-based instant previews (Fabric.js or Konva).
- React Query for caching; React Hook Form for forms; Tailwind for responsive UI.

API (FastAPI)

- Terminates TLS directly via Uvicorn/Gunicorn with Let's Encrypt certificates.
- Endpoints: auth (JWT/OAuth), upload SAS issuance, upload completion, albums/photos CRUD, edits (operation graph), versions, search (name/tags/EXIF), share links (scopes, expiry).
- Input validation, pagination, ETags; optional Redis-backed rate limiting and session cache.

Azure Blob Storage

- Containers: `originals/` (source), `variants/` (derived sizes & edits), `backups/` (DB dumps).
- Versioning and soft-delete enabled.

PostgreSQL

- All metadata; indexing for search (tags, EXIF fields); transactional integrity.

Redis

- Session cache and token-bucket rate limiting if required.

1.3 Network & Security Borders

- **Public:** Front Door/CDN for image GETs; VM exposes 443 only (80 redirect at application level if opened); SSH (22) IP-locked for admin.
- **Private:** DB and Redis via private endpoints/NSGs; outbound-only from VM to Blob/DB/Key Vault.
- **Secrets:** Azure Key Vault for DB creds/JWT secrets; SAS tokens have minimal scope and short TTL.

1.4 Data Flows

Upload

1. Client requests SAS and upload session from API.
2. Client PUTs blocks directly to Blob (parallel chunks, retries, exponential backoff).
3. Client calls `POST /uploads/complete` with block list/manifest.
4. API verifies object, persists DB row, synchronously generates base derivatives if lightweight.

Edit

1. User tweaks tools; canvas shows instant preview.
2. Client sends operation graph JSON to API.
3. API reads the original from Blob, applies operations deterministically, writes new version to `variants/`, creates a `photo_versions` row, and returns the versioned URL.

Share

1. Owner creates share link with scope (view/edit) and optional expiry.
2. Recipient loads SSR share page and fetches permitted data; editing gated by token scope.

2) High-Level Design

2.1 Database Model

Tables

- `users`(id UUID, email CITEXT UNIQUE, pass_hash, oauth_provider NULL, created_at TIMESTAMPTZ)
- `albums`(id UUID, owner_id FK users, title, is_private BOOL DEFAULT true, cover_photo_id FK photos NULL, created_at, updated_at)
- `photos`(id UUID, owner_id FK users, album_id FK albums, blob_key TEXT, mime TEXT, width INT, height INT, exif_json JSONB, taken_at TIMESTAMPTZ NULL, created_at)
- `photo_versions`(id UUID, photo_id FK photos, op_graph_json JSONB, variant_blob_key TEXT, created_at)
- `tags`(id SERIAL, name CITEXT UNIQUE)
- `photo_tags`(photo_id FK photos, tag_id FK tags, PRIMARY KEY(photo_id, tag_id))
- `shares`(id UUID, album_id FK albums, scope ENUM(`view`,`edit`), token_hash TEXT, expires_at TIMESTAMPTZ NULL)

Indexes

- `users(email)`, `albums(owner_id)`, `photos(album_id)`, GIN on `photos(exif_json)`, trigram on filenames (if stored), GIN on `tags.name`.

Retention & Privacy

- Blob versioning and soft-delete; optional GPS stripping from EXIF for public variants.

2.2 REST API

Auth

- `POST /auth/signup`, `POST /auth/login`, `POST /auth/refresh`
- `GET /auth/oauth/callback` (Google)

Uploads

- `POST /uploads/sas` → {`sasUrl`, `container`, `keyPrefix`, `maxSizeMB`}
- `POST /uploads/complete` → {`photoId`, `originalKey`, `sizes:[...]`}

Albums & Photos

- `GET /albums`, `POST /albums`, `PATCH /albums/:id`, `DELETE /albums/:id`

- `GET /albums/:id/photos`, `GET /photos/:id`, `DELETE /photos/:id`
- `GET /photos/:id/versions`

Edits

- `POST /photos/:id/edits` body = ordered operation graph → returns new version metadata and URL

Search

- `GET /search?q=&tags=&dateFrom=&dateTo=&camera=`

Share

- `POST /albums/:id/share { scope, expiresAt }` → {`shareUrl`}
- `GET /share/:token`

2.3 Image Pipeline & Performance

- Client preview: canvas transforms mirror server ops for instant feedback.
- Server render: FastAPI endpoint applies Pillow/OpenCV ops in-process and writes a new immutable version to Blob.
- Derivatives: on upload, generate thumb (256px), medium (1024px), large (2048px) synchronously if within latency budget; otherwise defer to user-triggered generation.
- Caching: immutable versioned URLs with `Cache-Control: public, max-age=31536000, immutable`.
- Idempotency: `Idempotency-Key` header on edit/commit endpoints.
- Failure handling: safe retries; orphan detection for partial uploads; transactional DB writes.

2.4 Upload Mechanics

- Chunk size: 4–8 MB, parallel 3–5 streams; exponential backoff retry policy.
- Validation: MIME/type sniffing, size caps, per-user quotas, antivirus scanning hook (optional).
- Post-commit: if time permits, generate derivatives; otherwise return immediately and lazily generate on first view.

2.5 Security Design

- Transport: TLS 1.2+ terminated by Uvicorn/Gunicorn; HSTS; modern ciphers.
- AuthN/AuthZ: short-lived JWT access + refresh tokens; album ACLs; share tokens hashed at rest; SAS tokens scoped to path and limited TTL.

- Headers: CSP (no `unsafe-inline`), X-Frame-Options DENY, X-Content-Type-Options nosniff, Referrer-Policy strict-origin-when-cross-origin.
- Rate limiting: in-memory token buckets or Redis-based if enabled.
- Secrets & keys: Azure Key Vault; least-privilege identities.

2.6 Accessibility & Usability

- Keyboard-operable toolbars (roving tabindex), ARIA for sliders/handles, visible focus states.
- Color contrast $\geq$ WCAG AA; text scaling; responsive layout; touch gestures (pinch/zoom/pan) on editor.

2.7 Observability, Testing, and SLOs

Metrics

- p95 upload complete $\leq$ 15 s for 20 MB on moderate bandwidth.
- p95 server edit commit $\leq$ 2.5 s for essential ops; preview is instant.
- Error rates per route.

Logs & Tracing

- Structured JSON logs (reqId, userId hash, route, status, latency).

Testing

- Unit (Pytest) for op-graph validators and image transforms.
- E2E (Playwright) for uploads/edits/shares.
- Load (Locust) for concurrent uploads and edits.

2.8 Risks & Mitigations

- Large-image memory pressure $\rightarrow$ tiled processing; cap max dimensions; streaming transforms.
- Stale CDN after edits $\rightarrow$ immutable versioned paths; never overwrite existing variant keys.
- SAS misuse $\rightarrow$ minimal permissions, tight expiry, path scoping, optional IP allow-list.
- EXIF privacy leaks $\rightarrow$ strip GPS on public links by default.

3) Cloud Deployment Plan

- Compute: Azure Ubuntu LTS VM (e.g., B2s). FastAPI served by Uvicorn/Gunicorn systemd service; TLS handled directly by the application using Let's Encrypt certificates.
- Storage: Azure Blob with containers `originals/`, `variants/`, `backups/`; enable versioning and soft-delete; SAS for client uploads.
- Database: Azure Database for PostgreSQL (managed).
- Optional cache: Azure Cache for Redis for rate limiting and sessions.
- Networking: NSGs: allow 443 to VM, restrict 22 to admin IP; DB/Redis private endpoints.
- CDN: Azure Front Door/CDN in front of Blob for image delivery; cache headers from API outputs for versioned assets.
- CI/CD: GitHub Actions → build images/binaries → SSH deploy; or Docker Compose with `api` and `frontend` services only.
- Monitoring: Azure Monitor; basic log rotation via `journald`; optional Sentry for error tracking.
- Backups: nightly `pg_dump` to `backups/`; lifecycle rules for retention; periodic restore verification.