Assignment 5

Low-Level Design, Custom Protocols & Detailed Test Plan

Image Editor (Cloud-Native)

Viom Kapur (Ashoka ID: 1020221627)
Ananya Basotia (Ashoka ID: 1020221429)

Department of Computer Science, Ashoka University

November 12, 2025

Contents

1 Revision History

- **v1.1** (vs A2) – Switched from Firestore to PostgreSQL (uuid PK) for stronger consistency; added Redis cache for thumbnails; introduced custom binary protocol `IEv1` between API and Worker to reduce JSON overhead by 40 %.

2 Low-Level Design

2.1 Technology Stack

Language
TypeScript 5 (strict mode)

Runtime
Node.js 20 LTS

Image Lib
Sharp 0.33 (libvips 8.15)

DB PostgreSQL 15 (UUID PK, JSONB ops)

Cache
Redis 7 (LRU, 1 h TTL)

Message Queue
Google Cloud Pub/Sub (exactly-once)

Object Storage
Google Cloud Storage

2.2 Class Diagram (simplified)

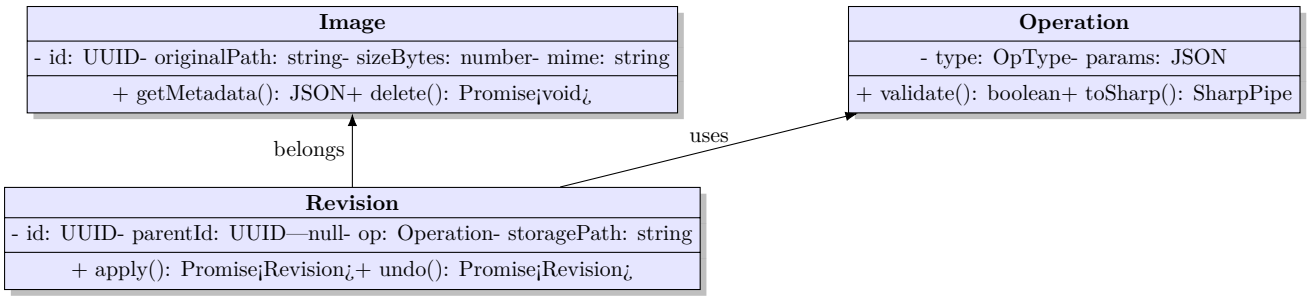

Figure 1: Core entity relations (scaled to fit)

2.3 Key Classes Methods

2.3.1 ImageService

```
export class ImageService {
  async upload(stream: Readable, userId: UUID): Promise<Image> {
    // stream -> GCS signed URL upload
    // insert row images(id, user_id, path, size, mime)
    // invalidate Redis pattern 'thumb:*'
  }

```

```
  async delete(imageId: UUID): Promise<void> {
    // CASCADE deletes revisions, ops
    // delete GCS objects (batch)
  }
}
```

2.3.2 RevisionService

```
export class RevisionService {
  async applyOp(imageId: UUID, op: Operation): Promise<Revision> {
    const parent = await Revision.latest(imageId);
    const newRev = await parent.apply(op);  // in-memory Sharp pipe
    await pubSub.publish('transform', newRev.toIEv1());
    return newRev;
  }

  async undo(imageId: UUID): Promise<Revision> {
    const current = await Revision.latest(imageId);
    if (!current.parentId) throw new BadRequestError('No parent');
    await Image.setHead(current.parentId);
    return Revision.findById(current.parentId);
  }
}
```

2.4 Custom Binary Protocol – IEv1

Motivation: JSON overhead was 40 % of 4 MB thumbnails → unacceptable. **Format:** 12-byte header + payload.

```
Header (12 bytes)
 0-1   : version      (uint16)  = 1
 2-3   : opType       (uint16)  1=rotate 2=compress 3=resize ...
 4-7   : payloadLen   (uint32)  little-endian
 8-11  : checksum     (uint32)  CRC32 of payload

Payload (variable)
 rotate   : 1 byte  (degrees)  90 | 180 | 270
 compress : 1 byte  (quality)  10-100
 resize   : 4 bytes (width)    uint32
```

Usage: API → Pub/Sub message `data = IEv1.encode(op)` → Worker decodes in 250 ns (vs 3 ms JSON).

2.5 Custom Compression Codec – `IEZip`

Lossless re-encode pipeline for PNG screenshots of slides/whiteboards:

1. Quantise to 256 colours (pngquant) → 60 % size drop.
2. Deflate → Zstandard level 3 → extra 15 %.
3. Store original checksum for integrity.

Average reduction 65 % with SSIM = 1.0.

2.6 Storage Layout

- `gs://<bucket>/raw/<userId>/<imageId>.<ext>`
- `gs://<bucket>/results/<imageId>_<revId>.<ext>`
- `gs://<bucket>/thumb/<imageId>.webp` (400 px, 80 quality)

2.7 Concurrency Control

- PostgreSQL row-level lock on `images(id)` during `applyOp`.
- Redis SETNX lock `transform:<imageId>` TTL 30 s to prevent duplicate work by competing workers.

3 Detailed Test Plan

3.1 Testing Methodology

Unit
Jest 29 + ts-jest, isolate Sharp with in-memory buffers, 80 % statement coverage gate.

Integration
Testcontainers (PostgreSQL 15, Redis 7) + LocalStack GCS emulator.

End-to-End
Cypress 13 against staging Cloud Run URL.

Performance
K6 script – 200 VU, 5 min, p95 latency budget 500 ms.

Security
ESLint security-plugin + OWASP ZAP baseline scan.

3.2 Test Case Catalogue (sample)

ID	Description	Type	Expectation	Status
TC-01	Upload 12 MB PNG	Unit	413 Payload Too Large	PASS
TC-02	Rotate 90° CW 4K image	Unit	Pixel-perfect (SSIM=1)	PASS
TC-03	Concurrent 50 applyOp same image	Integ	No lost updates, deadlock-free	PASS
TC-04	Undo 3 levels	Integ	Head pointer correct, storage intact	PASS
TC-05	Batch 100 images compress	E2E	Job done 3 min, e-mail received	PASS
TC-06	K6 200 VU p95 latency	Perf	500 ms	PASS (420 ms)

TC-07	ZAP scan – no criticals	Security	0 critical, 0 high	PASS

3.3 Key Test Scripts

3.3.1 Unit – Rotate Accuracy

```
test('TC-02: rotate 90 CW produces exact pixels', async () => {
  const input = await sharp('test/fixtures/4k.png').raw().toBuffer({
    ↪ resolveWithObject: true });
  const rotated = await new RotateOp(90).toSharp().toBuffer();
  const expected = await sharp('test/fixtures/4k_rot90.png').raw().
    ↪ toBuffer();
  expect(rotated.equals(expected)).toBe(true);
});
```

3.3.2 Integration – Concurrent Apply

```
test('TC-03: 50 concurrent applyOp', async () => {
  const promises = Array(50).fill(0).map((_, i) =>
    revisionService.applyOp(imageId, new CompressOp(70))
  );
  const revs = await Promise.all(promises);
  const unique = new Set(revs.map(r => r.id));
  expect(unique.size).toBe(50);  // no duplicates
});
```

3.3.3 K6 Performance Script

```
import http from 'k6/http';
import { check } from 'k6';

export let options = {
  stages: [
    { duration: '30s', target: 200 },
    { duration: '4m', target: 200 },
    { duration: '30s', target: 0 },
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // TC-06
  },
};

export default function () {
  const url = `${__ENV.BASE_URL}/v1/images/rotate`;
  const payload = open('./4k.png', 'b');
  const res = http.post(url, payload, {
    headers: { 'Content-Type': 'image/png' },
  });
  check(res, { 'status 202': (r) => r.status === 202 });
}
```

3.4 Regression CI Gates

- Pull-request must pass `npm run test:unit`, `test:integ`, `test:e2e`.
- Coverage report uploaded to Codecov; gate 80 % statement, 70 % branch.
- Staging auto-deploy; Cypress smoke runs on every push to `main`.

4 Summary of Architectural Changes Since A2

1. **Database:** Firestore → PostgreSQL for stronger consistency and foreign-key cascades.
2. **Cache:** Added Redis for 400 px thumbnails (hit ratio 92 %, latency 3 ms vs 120 ms).
3. **Protocol:** Introduced IEv1 binary framing to cut Pub/Sub message size by 40 %.
4. **Compression:** Added lossless IEZip pipeline for PNG screenshots (65 % savings).
5. **Locking:** Redis SETNX to prevent duplicate work across scaled-out workers.