FlexLiving — Reviews Dashboard

Complete, deep and detailed technical deliverable: scratch → code → deployment. This PDF contains architecture, data model (SQL), API contract, normalization logic, backend implementation guidance and sample code, Docker & deployment, CI/CD, frontend plan (Next.js + UI libs), observability, testing, scaling, and runbook.

1. Executive summary

Build a Reviews Dashboard for FlexLiving that aggregates reviews (Hostaway sandbox + optionally Google) and provides a Manager Dashboard to filter, approve, and publish reviews to the public property page. The backend must normalize Hostaway reviews and expose `GET /api/reviews/hostaway` as required by the assessment. This document assumes a 2-day delivery scope focused on backend excellence and a minimal polished frontend demo (Next.js).

2. Requirements (preserve original assignment)

• Implement `GET /api/reviews/hostaway` that fetches reviews from Hostaway sandbox or a provided mock and returns normalized JSON. • Build Manager Dashboard to filter, sort, approve reviews. • Build property Review Display page that only shows approved reviews. • Explore Google Reviews integration and document findings. • Deliver source code, README, brief (1-2 pages), and instructions to run locally (Docker).

3. High-level architecture

Diagram (textual):

Hostaway sandbox / Mock JSON → Backend API (Node.js + TypeScript) → Postgres (persistent) + Redis (cache) → Next.js Frontend

Optional: Google Places/Business Profile API → Backend → normalize and store as separate channel.

Key components: API service, DB, Cache, Frontend, Background workers (optional).

4. Data model (Postgres schema)

Use Postgres + JSONB for raw payload audit and normalized columns for queries.

```
-- listings
CREATE TABLE listings (
  id SERIAL PRIMARY KEY,
  hostaway_listing_id INTEGER UNIQUE,
  name TEXT NOT NULL,
  slug TEXT UNIQUE,
  created_at TIMESTAMP DEFAULT now()
);
-- reviews
CREATE TABLE reviews (
  id SERIAL PRIMARY KEY,
  hostaway_review_id INTEGER UNIQUE,
  listing_id INTEGER REFERENCES listings(id),
  review_type TEXT,
  channel TEXT,
  rating NUMERIC NULL,
  public_review TEXT,
  guest_name TEXT,
  submitted_at TIMESTAMP,
  approved BOOLEAN DEFAULT false,
  raw_json JSONB,
  created_at TIMESTAMP DEFAULT now()
);
-- review_categories
CREATE TABLE review_categories (
  id SERIAL PRIMARY KEY,
  review_id INTEGER REFERENCES reviews(id) ON DELETE CASCADE,
  category TEXT,
  rating INT
);
-- Indexes:
CREATE INDEX idx_reviews_listing ON reviews(listing_id);
CREATE INDEX idx_reviews_submitted_at ON reviews(submitted_at);
CREATE INDEX idx_reviews_approved ON reviews(approved);
```

5. Normalization rules and contract for GET /api/reviews/hostaway

Goal: ensure the frontend receives consistent shape irrespective of Hostaway or Google payload variation.

Normalized JSON shape:

```
{
  "status":"ok",
  "data":[
    {
      "id": 7453,
      "hostawayId": 7453,
      "listingName": "2B N1 A - 29 Shoreditch Heights",
      "listingId": 1234,
      "type": "host-to-guest",
      "channel": "hostaway",
      "rating": 10,
      "categories": {"cleanliness":10, "communication":10},
      "publicReview": "Shane and family are wonderful!...",
      "guestName": "Shane Finkelstein",
      "submittedAt": "2020-08-21T22:45:14Z",
      "approved": false
    }
  ]
}
```

Normalization steps:

• Convert dates to ISO 8601 UTC; validate date parsing. • Calculate `rating` from raw `rating` or average of categories if missing. • Flatten nested `reviewCategory` arrays into `categories` map and compute `averageRating` if useful. • Map review `type` to controlled enum values. • Preserve entire raw payload into `raw_json` for auditing and future re-normalization.

6. API design (contracts)

List of primary endpoints with query params and behavior:

```
GET /api/reviews/hostaway
  Query: listingId, from, to, channel, approved, page, limit
  Returns: normalized list, pagination meta
  Behavior: if HOSTAWAY configured -> call sandbox -> fallback to mocks
GET /api/reviews
  CRUD for reviews (list with filters, patch to approve/unapprove, create from webhook)
PATCH /api/reviews/:id/approve
  Body: { approved: true }
  Action: update approved flag, invalidate cache for listing
GET /api/listings
  Returns listing metadata (name, slug, hostawayListingId)
POST /admin/import/google
  Body: { placeId }
  Action: fetch google reviews via Places or Business Profile API, normalize, store as channel='google
'
```

7. Normalize function — detailed pseudocode

```
function normalizeHostawayReview(raw) {
  const id = raw.id || raw.reviewId || null;
  const categoriesArr = raw.reviewCategory || [];
  const categories = {};
  let sum = 0;
  categoriesArr.forEach(c => { categories[c.category] = c.rating; sum += c.rating; });
  const avgFromCategories = categoriesArr.length ? (sum / categoriesArr.length) : null;
  const rating = raw.rating ?? avgFromCategories ?? null;
  const submittedAt = raw.submittedAt ? new Date(raw.submittedAt).toISOString() : null;
  return {
    id,
    hostawayId: id,
    listingName: raw.listingName || raw.listing_name || null,
    listingId: raw.listingId || raw.hostawayListingId || null,
    type: normalizeType(raw.type),
    channel: raw.channel || 'hostaway',
    rating,
    categories,
    publicReview: raw.publicReview || raw.public_review || '',
    guestName: raw.guestName || raw.guest_name || 'Guest',
    submittedAt,
    approved: false,
    raw_json: raw
  };
}
```

8. Backend recommended stack & project structure

Stack: Node.js + TypeScript, Express or NestJS (NestJS recommended for structure), Prisma or TypeORM for DB, Redis, Jest for tests.

```
/backend
  /src
    /routes
      hostaway.ts
      reviews.ts
    /services
      normalize.ts
      hostawayClient.ts
    /db
      schema.prisma (or entities)
    /migrations
    /seeds
    /tests
  package.json
  tsconfig.json
  Dockerfile
  docker-compose.yml
  mocks/hostaway_reviews.json
```

9. Example Express route (handler) — full example

```
// src/routes/hostaway.ts
import express from 'express';
import fs from 'fs/promises';
import path from 'path';
import { normalizeHostawayReview } from '../services/normalize';
const router = express.Router();
router.get('/', async (req, res) => {
  try {
    // Try live Hostaway if configured (omitted for brevity) else fallback:
    const mockPath = path.join(__dirname, '..', '..', 'mocks', 'hostaway_reviews.json');
    const raw = await fs.readFile(mockPath, 'utf-8');
    const parsed = JSON.parse(raw);
    const items = parsed.result || parsed.reviews || [];
    const normalized = items.map(normalizeHostawayReview);
    // Apply filters, pagination:
    const page = parseInt(req.query.page as string) || 1;
    const limit = Math.min(parseInt(req.query.limit as string) || 25, 100);
    const start = (page - 1) * limit;
    const paged = normalized.slice(start, start + limit);
    res.json({ status: 'ok', data: paged, meta: { page, limit, total: normalized.length }});
  } catch (err) {
    console.error(err);
    res.status(500).json({ status:'error', message:'failed to load reviews' });
  }
});
export default router;
```

10. Caching & invalidation

Use Redis to cache normalized responses keyed by query params (e.g., `reviews:hostaway:listings=123:approved=false:page=1`). TTL: 2–5 minutes appropriate for near-real-time UX while protecting upstream rate limits. On review approve/unapprove events, invalidate cache keys for affected listing(s).

11. Docker and local development

Provide `docker-compose.yml` with services: app, postgres, redis. Example:

```
version: '3.8'
services:
  db:
    image: postgres:15
    env_file: .env
    environment:
      POSTGRES_DB: flex
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - db_data:/var/lib/postgresql/data
    ports:
      - '5432:5432'
  redis:
    image: redis:7
    ports:
      - '6379:6379'
  app:
    build: .
    env_file: .env
    ports:
      - '4000:4000'
    depends_on:
      - db
      - redis
volumes:
  db_data:
```

12. Sample .env (sensitive keys as placeholders)

```
DATABASE_URL=postgres://postgres:postgres@db:5432/flex
REDIS_URL=redis://redis:6379
HOSTAWAY_ACCOUNT=61148
HOSTAWAY_API_KEY=__PUT_KEY_HERE__
PORT=4000
NODE_ENV=development
```

13. Testing strategy

Unit tests: normalize function edge cases (missing rating, weird date formats, empty categories). Integration: test `GET /api/reviews/hostaway` returns expected shape, pagination, filter behavior, and cache headers. E2E: simulate approve flow and verify property page shows the approved review.

```
// jest example
describe('normalizeHostawayReview', () => {
  it('computes rating from categories if rating missing', () => {
    const raw = { id:1, reviewCategory:[{category:'cleanliness',rating:8},{category:'communication',ra
ting:10}]};
    const out = normalizeHostawayReview(raw);
    expect(out.rating).toBe(9);
  });
});
```

14. Frontend (Next.js) plan and components

Focus on delivering a polished demo using Mantine or shadcn/ui + Radix. Keep SSR for property pages and CSR for Manager Dashboard.

Pages and components:

• /manager — Manager Dashboard (Table, FiltersPanel, ApproveToggle modal) • /property/[slug] — Property page with `ReviewCard` components • Components: ReviewsTable, FiltersPanel, ReviewCard, ApproveModal, Pagination

Use Mantine for Table, DatePicker, MultiSelect. Use shadcn/ui for advanced card design and Radix for headless primitives when needed.

15. Example frontend flow (approve action)

```
// Manager approves a review:
PATCH /api/reviews/:id/approve { approved: true }
Backend updates DB, writes audit, invalidates Redis cache for listingId.
Frontend receives success -> optimistic update to table -> shows updated status.
Property page fetches approved reviews (cache gets invalidated so next fetch sees change).
```

16. Deployment options (Docker image → Cloud)

Option A — Deploy via Docker images to Google Cloud Run or AWS ECS/Fargate. Cloud Run supports concurrency and auto-scaling with minimal infra.

Option B — Deploy to Kubernetes (GKE/EKS) for large-scale needs. Use a Deployment + HorizontalPodAutoscaler, and separate StatefulSet for Postgres (managed service recommended).

Steps for Cloud Run (example):

```
# build and push
docker build -t gcr.io//flex-reviews:latest .
docker push gcr.io//flex-reviews:latest
# deploy
gcloud run deploy flex-reviews --image gcr.io//flex-reviews:latest --platform managed --region us-cent
ral1 --allow-unauthenticated --set-env-vars=DATABASE_URL=...
```

17. CI/CD (GitHub Actions example)

```
name: CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Setup Node
        uses: actions/setup-node@v4
        with: node-version: 20
      - run: npm ci
      - run: npm run lint
      - run: npm test
      - name: Build Docker image
        uses: docker/build-push-action@v5
        with:
          push: true
          tags: ghcr.io/${{ github.repository }}:latest
```

18. Observability & monitoring

• Structured logging: pino/winston with ISO timestamps and request IDs. • Metrics: expose /metrics for Prometheus (request_count, request_duration_seconds, cache_hit_ratio). • Error tracking: Sentry integration for exceptions. • Health endpoints: /health and /ready

19. Security & best practices

• Keep API keys in env, not committed. • Use helmet, cors, rate-limiter middleware. • Use parameterized queries or Prisma/ORM to avoid SQL injection. • Sanitize and validate all inputs using Zod or Joi. • Set secure headers and enforce TLS in production.

20. Scaling & long-term improvements

• Horizontal scale the API service behind a load balancer. • Offload search to ElasticSearch or Postgres full-text when queries become complex. • Use materialized views for heavy aggregation (refresh via background job/BullMQ). • Use job queue (BullMQ) for periodic ingestion of Hostaway/Google reviews.

21. Rollout & runbook (troubleshooting)

Common issues and checks:

• No reviews returned: check Hostaway sandbox credentials, fallback to mocks. • Approve not showing on property page: verify cache invalidation and that approved flag is true. • DB connection errors: verify DATABASE_URL and Postgres health. • High latency: check slow queries (enable pg_stat_statements) and Redis cache hit rates.

22. Appendix: sample mock review JSON

```
{
  "result": [
    {
      "id": 7453,
      "listingName": "2B N1 A - 29 Shoreditch Heights",
      "listingId": 1234,
      "type": "host-to-guest",
      "rating": 10,
      "reviewCategory": [
         {"category":"cleanliness","rating":10},
         {"category":"communication","rating":10}
      ],
      "publicReview": "Shane and family are wonderful!...",
      "guestName": "Shane Finkelstein",
      "submittedAt": "2020-08-21T22:45:14Z"
    }
  ]
}
```

23. Appendix: curl examples

```
# Fetch normalized hostaway reviews
curl 'http://localhost:4000/api/reviews/hostaway?page=1&limit;=25'
# Approve a review
curl -X PATCH 'http://localhost:4000/api/reviews/123/approve' -H 'Content-Type: application/json' -d '
{"approved":true}'
```

24. Deliverables checklist (what to submit)

• backend/ (source + mocks + Dockerfile + docker-compose) • frontend/ (Next.js demo with Manager and Property pages) • README.md (run and deploy instructions) • BRIEF_DOC.pdf (1-2 pages summarizing design choices) — included as part of this package • POSTMAN_collection.json or curl examples Ensure tests pass and Docker compose starts smoothly before submission.

25. Next steps I can do for you (choose one)

• Generate a runnable backend boilerplate (Express + TypeScript) with `GET /api/reviews/hostaway` implemented and mocks included. • Generate the exact Next.js frontend skeleton (pages + Mantine setup + sample components). • Produce the GitHub Actions CI file and Deployment scripts tailored to Cloud Run or ECS. Tell me which to produce now and I will generate the files for download.