FlexLiving — Reviews Dashboard

> Complete senior-backend focused deliverable, ready-to-implement in 2 days. Includes architecture, API design, DB schema, normalization logic, code snippets, Docker setup, testing, frontend plan (Next.js + recommended UI libs), Google Reviews exploration, and step-by-step local setup + deployment notes.

Important: This document preserves *every* requirement from the original assignment and adds optional, interview-grade backend features. (Original assignment file included with the submission.)

---

1) Plain-language summary (what you're asked to build)

- Build a **Reviews Dashboard** for Flex Living to help managers view and manage guest reviews per property.
- Integrate with Hostaway Reviews API — sandboxed. Use provided/mock JSON because sandbox has no reviews.
- Provide a **manager dashboard** to filter, sort, approve reviews for public display, and spot trends.
- Add a **Review Display Page** that replicates Flex Living property layout and only shows reviews approved by manager.
- Explore the possibility of integrating **Google Reviews** and document findings.
- **Mandatory:** implement `GET /api/reviews/hostaway` which fetches + normalizes Hostaway reviews and returns structured data for frontend.

(Checked against original assignment PDF to ensure nothing is removed.)

---

2) One-line senior-engineer checklist (start here)

- Implement `GET /api/reviews/hostaway` returning normalized review shape and a small mock dataset.
- Add DB model + seed for listings & reviews and a normalized reviews view.
- Provide caching, pagination, error handling, validation, and rate-limiting.
- Build Manager Dashboard pages in Next.js (use Mantine or shadcn + Radix) to filter/approve reviews.
- Create Review Display Page that reads approved reviews and matches Flex Living property layout.
- Run basic unit & integration tests and provide Docker compose + local setup.

---

3) Two-day delivery plan (block schedule)

Constraints: 2 calendar days. Prioritize backend correctness and `GET /api/reviews/hostaway` + integration with minimal frontend for demo.

Day 1 — Backend (+ core API + tests)

- 0.5h — Repo scaffolding, TypeScript + Node + NestJS or Express (I recommend NestJS for structure but Express + TypeScript is fine).
- 1.5h — Implement normalization logic and `GET /api/reviews/hostaway` route using provided/mock JSON.
- 1h — DB models (Postgres): `listings`, `reviews`, `review_categories`, seed scripts.
- 1h — Dockerfile + docker-compose (Postgres + Redis + app) + env file.`
- 1h — Caching (Redis), simple ETag/If-None-Match, rate limit middleware, input validation using Zod/Joi.
- 1h — Unit tests for normalization and integration test for API route (Jest + supertest).
- 0.5h — Write short README & 1–2 page brief documentation (tech stack, design decisions, API behavior, Google findings).

Day 2 — Frontend demo + polish + deliverables

- 1.5h — Next.js frontend skeleton with two pages: Manager Dashboard and Property Review Display.
- Use a component library (Mantine recommended for speed + aesthetics) + shadcn UI snippets if desired.
- 2h — Manager Dashboard: list reviews (paginated), filters (rating, category, channel, time), approve toggle (sends PATCH to API).
- 1h — Property Page: replicate property layout and embed approved reviews.
- 1h — QA: test flows, fix bugs, ensure `GET /api/reviews/hostaway` returns correct shape.
- 1h — Final docs, screenshots, how to run locally, pack deliverables, commit.

If time runs short, keep frontend minimal: a Next.js page using Mantine Table + Filters and a simple property page showing approved reviews.

---

4) Tech stack (recommended — interview-friendly)

- **Backend:** Node.js + TypeScript. Framework: **NestJS** (structure) or **Express + TypeScript** (faster).
- **DB:** PostgreSQL.
- **Cache/Queue:** Redis (caching + lightweight job queue for heavy normalization if needed).
- **Frontend:** Next.js (React + SSR/SSG) + **Mantine** (fast, modern components) or **shadcn/ui** + Radix + Tailwind for pixel finesse.
- **Auth/Access:** JSON Web Tokens or simple API-key for manager demo (no full auth required for assessment).
- **Testing:** Jest + supertest.
- **Lint/Style:** ESLint + Prettier + TypeScript strict.
- **Observability:** pino/winston + Prometheus metrics endpoint + Sentry optional.
- **CI/CD:** GitHub Actions that runs tests and builds Docker image.

---

5) Architecture overview (textual)

1. **Hostaway sandbox** — mocked JSON saved in `mocks/hostaway_reviews.json`.
2. **Backend API service (Node/TS)**:

3. `/api/reviews/hostaway` — reads mock or calls Hostaway, normalizes, returns structured data.
4. `/api/reviews` — CRUD operations (list, approve, filter, paginate).
5. `/api/listings` — listing metadata.
6. **Postgres** — persistent storage of listings + approved state of reviews.
7. **Redis** — cache normalized responses for 2–5 minutes, store rate-limiting counters.
8. **Next.js frontend** — Manager Dashboard + Property page.

---

6) Data model (suggested SQL schema)

```sql
-- 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, -- host-to-guest | guest-to-host etc
  channel TEXT, -- e.g. booking.com, airbnb, hostaway
  rating NUMERIC NULL,
  public_review TEXT,
  guest_name TEXT,
  submitted_at TIMESTAMP,
  approved BOOL DEFAULT false,
  raw_json JSONB,
  created_at TIMESTAMP DEFAULT now()
);

-- review categories (normalized key/value)
CREATE TABLE review_categories (
  id SERIAL PRIMARY KEY,
  review_id INTEGER REFERENCES reviews(id) ON DELETE CASCADE,
  category TEXT,
  rating INT
);
```

Indexed fields: `submitted_at`, `listing_id`, `approved`, and `rating` for efficient filters.

---

7) Normalization rules — what the `/api/reviews/hostaway` must return

Return a **consistent JSON shape** that the frontend expects. Example normalized 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: - Parse `submittedAt` -> ISO 8601 UTC. - Ensure `rating` is numeric; if missing compute average from `reviewCategory` if available. - Flatten `reviewCategory` array to `categories` map and optional `averageRating`. - Map `type` to a controlled enum. - Preserve raw JSON in `raw_json` column for audit.

---

8) Route: `GET /api/reviews/hostaway` (contract & sample code)

Contract: - **Query params:** `listingId` (optional), `from` (ISO date), `to` (ISO date), `channel`, `approved` (true/false), `page`, `limit`. - **Behavior:** If `HOSTAWAY_API` is configured, attempt to call sandbox; if not or empty results, load `mocks/hostaway_reviews.json` and return normalized list. - **Response:** 200 with normalized structure (see section 7).

Express + TypeScript sample handler (minimal):

```
// src/routes/hostaway.ts
import { Router } from 'express';
import fs from 'fs/promises';
import path from 'path';
import { normalizeHostawayReview } from '../services/normalize';
```

```
const router = Router();

router.get('/', async (req, res) => {
  try {
    // Try to call Hostaway sandbox if env provided -- else read mock
    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 || [];
    const normalized = items.map(normalizeHostawayReview);
    return res.json({ status: 'ok', data: normalized });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ status: 'error', message: 'failed to load
reviews' });
  }
});

export default router;
```

Normalization function (TypeScript):

```
// src/services/normalize.ts
export function normalizeHostawayReview(raw: any) {
  const id = raw.id ?? null;
  const categoriesArr = raw.reviewCategory ?? [];
  const categories: Record<string, number> = {};
  let avgFromCategories: number | null = null;
  if (categoriesArr.length) {
    let sum = 0;
    categoriesArr.forEach((c: any) => { categories[c.category] = c.rating;
sum += c.rating; });
    avgFromCategories = Math.round((sum / categoriesArr.length) * 10) / 10;
  }
  const rating = raw.rating ?? avgFromCategories;
  const submittedAt = raw.submittedAt ? new
Date(raw.submittedAt).toISOString() : null;
  return {
    id,
    hostawayId: id,
    listingName: raw.listingName || raw.listing_name || null,
    type: raw.type || null,
    channel: raw.channel || 'hostaway',
    rating,
    categories,
    publicReview: raw.publicReview || raw.public_review || null,
    guestName: raw.guestName || raw.guest_name || null,
    submittedAt,
    raw
```

```
  };
}
```

---

9) Caching & performance

- Cache normalized `GET /api/reviews/hostaway` responses keyed by `listingId:queryparams` for 2–5 minutes in Redis.
- Add pagination at API level (`page`, `limit`) to prevent large responses.
- Use DB indexes on `listing_id`, `approved`, and `submitted_at`.
- If reviews become a lot (100k+), add a materialized `normalized_reviews` table refreshed via a background job.

---

10) Security & best practices

- Never commit API keys. Use `.env` for `HOSTAWAY_KEY` and `HOSTAWAY_ACCOUNT` (account id given in the PDF: `61148` and API key in the PDF; for final deliverable put placeholders and list the real key only for reviewer as an env var).
- Add `helmet`, `cors`, `express-rate-limit` middleware.
- Validate inputs with Zod/Joi for query params.
- Use parameterized queries / ORM (TypeORM, Prisma) to avoid SQL injection.

---

11) Observability & ops

- Structured logging with `pino` or `winston`.
- Add metrics endpoint `/metrics` for Prometheus (requests, latency, cache hit rate).
- Use Sentry for uncaught exceptions (optional).

---

12) Testing strategy (what to include in repo)

- Unit tests for `normalizeHostawayReview` covering edge cases (missing categories, missing rating, weird date formats).
- Integration test for `GET /api/reviews/hostaway` using supertest and mocked responses (`mocks/hostaway_reviews.json`) covering pagination, filters, caching behavior, and error cases. /api/reviews/hostaway `using` supertest `and mocked` mocks/hostaway_reviews.json`.
- E2E smoke test for manager flow (approve a review -> verify property page shows it).

---

13) Frontend (Next.js) minimal structure for demo

- Pages:
- `/manager` — Manager Dashboard (table of reviews, filters: rating, category, channel, time range, approve toggle).

- `/property/[slug]` — Property details page that includes a "Guest Reviews" section showing only approved reviews.
- Components: `ReviewsTable`, `FiltersPanel`, `ApproveToggle`, `ReviewCard`.
- Use Mantine for Table, DatePicker, Modal, etc. Add `shadcn/ui` slices for advanced cards and to match look-and-feel if needed.

---

14) Google Reviews exploration (deliverable requirement)

Short findings (to include in README): - Google Places API / Place Details can return user ratings (and sometimes user reviews) for places that are listed publicly via Google Maps.
- Google My Business (now renamed to Business Profile APIs) historically required business ownership to fetch detailed reviews for a business.
- Scraping Google is against their ToS. Prefer using official Places API or Business Profile API—these are rate-limited and require API key with billing enabled.

Actionable implementation notes (if you have time): - If property has a consistent Google Place ID, call Places API `place/details` to fetch `reviews` (note limited to a few reviews only).
- Store fetched Google reviews as separate `channel = 'google'` and normalize same as Hostaway.

---

15) Local dev & run instructions (README snippet)

1. `git clone <repo>`
2. Create `.env` with:

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

3. `docker-compose up -d` (Postgres + Redis)
4. `cd backend && npm install && npm run migrate && npm run seed` (creates tables and seeds mock listings/reviews)
5. `npm run dev` (starts backend)
6. `cd frontend && npm install && npm run dev` (starts Next.js)

---

16) Deliverables checklist (what you should zip/upload for the assessment)

- `/backend` — Source code (TypeScript), `mocks/hostaway_reviews.json`, tests, Dockerfile, `docker-compose.yml`.
- `/frontend` — Next.js source with Manager Dashboard + Property page.
- `README.md` — Setup & run, tech stack, architecture, quick screenshots.

- `BRIEF_DOC.pdf` (1–2 pages) — Tech stack, design decisions, API behaviors, Google Reviews findings.
- `POSTMAN_collection.json` or curl examples — to test APIs.

---

17) Example `curl` to validate the required endpoint

```
curl 'http://localhost:4000/api/reviews/hostaway'
```

Should return normalized JSON as specified in Section 7.

---

18) Extra "senior" features (optional but strongly recommended to show seniority)

- Background job (e.g. BullMQ) to periodically fetch & re-normalize reviews from Hostaway and Google.
- Materialized `normalized_reviews` table for high throughput queries.
- Full-text search (Postgres `tsvector` or ElasticSearch) for review content.
- Role-based access control (manager vs. read-only viewer).
- Feature flag for public website toggle of reviews per listing.
- Audit table to track who approved a review and when.

---

19) Interview talking points (what to say during the interview)

- Explain normalization trade-offs (store raw JSON + normalized columns for queries).
- Talk about caching decisions and TTL choices for near-real-time UX vs. API rate limits.
- Discuss how to scale to 100k+ reviews (materialized views, batch jobs, search indices).
- Outline observability (metrics to track: review ingestion rate, cache hit rate, average approval latency).

---

20) Files to include in repo (quick tree)

```
/backend
  /src
    /routes
      hostaway.ts
      reviews.ts
    /services
      normalize.ts
    /migrations
    /seeds
    /tests
  package.json
```

```
  Dockerfile
  docker-compose.yml
  mocks/hostaway_reviews.json
/frontend
  /pages
    /manager.tsx
    /property/[slug].tsx
  /components
  package.json
README.md
BRIEF_DOC.pdf
POSTMAN_collection.json
```

---

21) Final note — what I built into this document

- I preserved every single requirement from the PDF (Hostaway integration, manager dashboard features, review display page, Google Reviews exploration, `GET /api/reviews/hostaway`, deliverables). I added prioritized backend-focused features and a realistic 2-day plan so you can ship a hardened submission quickly.

If you want, I can now: - Generate a runnable `backend` boilerplate (Express + TS) with the `GET /api/reviews/hostaway` implemented and a mocked JSON in the repo, plus `docker-compose.yml` and seed script — ready to zip.
- Or produce focused files: `mocks/hostaway_reviews.json`, `src/services/normalize.ts`, `src/routes/hostaway.ts`, `Dockerfile`, `docker-compose.yml`, and README as separate files.

Tell me which of the two you want me to generate right now and I will create the code files (TypeScript) and instructions.

Document prepared as a senior backend-focused deliverable to complete the Flex Living Reviews Dashboard assessment.