Back-end “llave en mano” (GitHub)
Sitio de oportunidades inmobiliarias en Barcelona

Febrero 2026

1. Qué entrega este documento

Este documento está escrito para que un/a programador/a pueda implementar el back-end **sin adivinar decisiones**.

Incluye:

- **Stack elegido (cerrado)**: Node.js + Express + TypeScript + PostgreSQL + PostGIS + JWT.
- **Estructura completa del repositorio**, archivos y contenidos.
- **Esquema de base de datos** (SQL) con índices geoespaciales.
- **Script de importación** desde `data_ROI.xlsx`.
- **Contrato de API** (endpoints + parámetros + formatos de salida).
- **Seguridad implementada**: JWT, roles, CORS allowlist, Helmet, rate-limit, validación.
- **Docker Compose** para levantar todo en local.
- **CI en GitHub Actions**.

2. Quickstart (en 15 minutos)

Prerequisitos

- Git
- Docker + Docker Compose
- (Opcional) Node 20+ y Python 3.11+ si querés correr fuera de Docker

Pasos

1. Crear repositorio en GitHub (vacío).
2. Clonar:

```
git clone https://github.com/TU_USUARIO/tu-repo.git
cd tu-repo
```

3. Crear los archivos exactamente como se listan en este documento (sección 7).
4. Levantar BD + API:

```
docker compose up -d --build
```

5. Correr migraciones (crea tablas e índices):

```
docker compose exec api npm run migrate
```

6. Copiar `data_ROI.xlsx` a `backend/scripts/data`$_{R}OI.xlsxyejecutarimportacin$:

7. Probar:

```
curl "http://localhost:8080/health"
```

3. Decisiones técnicas (cerradas)

3.1. Stack

- **API**: Node.js 20 + Express + TypeScript
- **DB**: PostgreSQL 16 + PostGIS 3 (geoespacial)
- **Auth**: JWT (access token), contraseñas con bcrypt, roles (user/admin)
- **Formato geográfico**: GeoJSON para el mapa

3.2. Por qué PostGIS

El front-end necesita consultas rápidas por zona visible (`bbox`) y potencial clustering. PostGIS permite:

- Guardar puntos con SRID 4326 (`geom POINT`)
- Indexar con GiST
- Filtrar por `ST_MakeEnvelope` (viewport)
- Emitir GeoJSON directo con `ST_AsGeoJSON`

4. Contrato de API (definición exacta)

4.1. Convenciones

- Base URL local: `http://localhost:8080`
- Respuestas JSON (y GeoJSON donde se indique)
- Paginación: `limit` y `offset` en listados
- Auth: header `Authorization: Bearer <token>`

4.2. Endpoints

Health

- `GET /health` → `{ ok: true }`

Propiedades (mapa)

- `GET /api/properties` → **GeoJSON FeatureCollection** (ligero)

Query params (todos opcionales salvo donde se indique):

- `bbox` (recomendado para mapa): `minLon,minLat,maxLon,maxLat`
- `neighborhood`, `district`, `postal_code`
- `price_min`, `price_max`
- `roi_min`, `roi_max`
- `comprable` (0 o 1)
- `amenities`: lista separada por comas (ej: `lift,terrace`)
- `limit` (default 2000), `offset` (default 0)

Ejemplo:

```
GET /api/properties?bbox=2.10,41.34,2.22,41.42&neighborhood=Dreta%20de%20l%E2%80%99
    Eixample&roi_min=0.05&amenities=lift,terrace
```

Propiedad (detalle)

- `GET /api/properties/:id` → detalle completo (Descripción + Clasificación)

Catálogo

- `GET /api/catalog/neighborhoods` → barrios + conteo
- `GET /api/catalog/districts` → distritos + conteo

Ranking

- `GET /api/ranking` → lista ordenada + rank

Query params:

- `scope` = `neighborhood` | `viewport` (default `neighborhood`)
- Si `scope=neighborhood`: `neighborhood` (requerido)
- Si `scope=viewport`: `bbox` (requerido)
- `mode` = `roi` | `gap` | `effective_price` | `liquidity`
- `direction` = `asc` | `desc` (default depende del modo)
- `limit` (default 50)

Definiciones:

- `ck` = probabilidad propietario 2 meses (de Excel: `prob_propietario_1`)
- `cm` = probabilidad inversor 2 meses (de Excel: `prob_investor_1`)
- `gap` = `cm/ck` usando `NULLIF(ck,0)` para evitar división por cero
- `effective_price` = `price_m2 * cm`
- `liquidity` = `cm`

Auth (seguridad)

- `POST /api/auth/register` (email, password) → crea usuario
- `POST /api/auth/login` (email, password) → devuelve JWT

Favoritos (ejemplo de endpoint protegido)

- `POST /api/user/favorites/:propertyId` (requiere JWT)
- `GET /api/user/favorites` (requiere JWT)
- `DELETE /api/user/favorites/:propertyId` (requiere JWT)

5. Seguridad (implementación concreta)

Se aplica el siguiente checklist (incluido en el código):

- HTTPS (en despliegue; en local no)
- JWT (expira en 2h), bcrypt para passwords
- Roles: `user` y `admin`
- CORS allowlist por `CORS_ORIGINS`
- Helmet (headers)
- Rate limiting en `/api`
- Validación de inputs con Zod (incluye `bbox` y rangos)
- Queries parametrizadas (evita SQL injection)
- Secrets fuera del repo: `.env` + GitHub Secrets

6. Sistema geoespacial (qué es y cómo se "descarga")

6.1. Qué es el sistema

El **sistema** para mapear con lat/lon aquí es: **PostgreSQL + PostGIS** (la base con extensión geoespacial), más una API que:

- Guarda cada propiedad como un `POINT` en SRID 4326

- Indexa con GiST (consultas rápidas por viewport)
- Devuelve GeoJSON para que el front-end renderice markers y tooltips

6.2. Dónde se obtiene

- PostgreSQL: https://www.postgresql.org/download/
- PostGIS: https://postgis.net/install/
- Para **no instalar nada**, este documento ya incluye Docker Compose usando imagen `postgis/postgis`.

7. Estructura del repositorio y archivos (llave en mano)

7.1. Árbol

```
tu-repo/
  docker-compose.yml
  .env.example
  README.md
  .github/workflows/ci.yml
  backend/
    Dockerfile
    package.json
    tsconfig.json
    src/
      server.ts
      app.ts
      config/env.ts
      db/pool.ts
      db/migrate.ts
      middlewares/auth.ts
      middlewares/validate.ts
      routes/index.ts
      routes/health.ts
      routes/properties.ts
      routes/catalog.ts
      routes/ranking.ts
      routes/auth.ts
      routes/user.ts
      schemas/common.ts
      utils/geo.ts
    db/migrations/001_init.sql
    scripts/
      import_excel.py
      requirements.txt
      data_ROI.xlsx   <-- poner aqui tu Excel
```

8. Contenido de archivos (copiar/pegar tal cual)

`docker-compose.yml`

```
services:
  db:
    image: postgis/postgis:16-3.4
    container_name: bcn_db
    environment:
      POSTGRES_USER: ${POSTGRES_USER:-bcn}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-bcn}
```

```
      POSTGRES_DB: ${POSTGRES_DB:-bcn}
    ports:
      - "5432:5432"
    volumes:
      - db_data:/var/lib/postgresql/data

  api:
    build:
      context: .
      dockerfile: backend/Dockerfile
    container_name: bcn_api
    environment:
      NODE_ENV: development
      PORT: 8080
      DATABASE_URL: postgres://${POSTGRES_USER:-bcn}:${POSTGRES_PASSWORD:-bcn}@db:5432/$
    {POSTGRES_DB:-bcn}
      JWT_SECRET: ${JWT_SECRET:-dev_secret_change_me}
      CORS_ORIGINS: ${CORS_ORIGINS:-http://localhost:3000,http://localhost:5173}
      RATE_LIMIT_WINDOW_MS: ${RATE_LIMIT_WINDOW_MS:-60000}
      RATE_LIMIT_MAX: ${RATE_LIMIT_MAX:-120}
    ports:
      - "8080:8080"
    depends_on:
      - db
    volumes:
      - ./:/app

volumes:
  db_data:
```

.env.example

```
POSTGRES_USER=bcn
POSTGRES_PASSWORD=bcn
POSTGRES_DB=bcn
JWT_SECRET=change_me_in_prod
CORS_ORIGINS=http://localhost:5173,http://localhost:3000
RATE_LIMIT_WINDOW_MS=60000
RATE_LIMIT_MAX=120
```

backend/Dockerfile

```
FROM node:20-bookworm

# Python para el import_excel.py
RUN apt-get update && apt-get install -y python3 python3-pip && rm -rf /var/lib/apt/
    lists/*

WORKDIR /app

COPY backend/package.json /app/backend/package.json
COPY backend/tsconfig.json /app/backend/tsconfig.json

WORKDIR /app/backend
RUN npm install

# deps python
```

```
COPY backend/scripts/requirements.txt /app/backend/scripts/requirements.txt
RUN pip3 install --break-system-packages -r /app/backend/scripts/requirements.txt

WORKDIR /app
COPY . /app

WORKDIR /app/backend
CMD ["npm","run","dev"]
```

backend/package.json

```
{
  "name": "bcn-roi-api",
  "version": "1.0.0",
  "private": true,
  "main": "dist/server.js",
  "scripts": {
    "dev": "ts-node-dev --respawn --transpile-only src/server.ts",
    "build": "tsc",
    "start": "node dist/server.js",
    "migrate": "ts-node src/db/migrate.ts"
  },
  "dependencies": {
    "bcrypt": "^5.1.1",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "express-rate-limit": "^7.4.0",
    "helmet": "^7.1.0",
    "jsonwebtoken": "^9.0.2",
    "pg": "^8.12.0",
    "zod": "^3.23.8"
  },
  "devDependencies": {
    "@types/bcrypt": "^5.0.2",
    "@types/cors": "^2.8.17",
    "@types/express": "^4.17.21",
    "@types/jsonwebtoken": "^9.0.6",
    "@types/node": "^20.14.10",
    "ts-node": "^10.9.2",
    "ts-node-dev": "^2.0.0",
    "typescript": "^5.5.4"
  }
}
```

backend/tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "CommonJS",
    "rootDir": "src",
    "outDir": "dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true
  }
```

```
}
```

backend/src/config/env.ts

```
import dotenv from "dotenv";
dotenv.config();

export const env = {
  NODE_ENV: process.env.NODE_ENV ?? "development",
  PORT: parseInt(process.env.PORT ?? "8080", 10),
  DATABASE_URL: process.env.DATABASE_URL ?? "",
  JWT_SECRET: process.env.JWT_SECRET ?? "",
  CORS_ORIGINS: (process.env.CORS_ORIGINS ?? "").split(",").map(s => s.trim()).filter(
    Boolean),
  RATE_LIMIT_WINDOW_MS: parseInt(process.env.RATE_LIMIT_WINDOW_MS ?? "60000", 10),
  RATE_LIMIT_MAX: parseInt(process.env.RATE_LIMIT_MAX ?? "120", 10)
};

if (!env.DATABASE_URL) throw new Error("Falta DATABASE_URL");
if (!env.JWT_SECRET) throw new Error("Falta JWT_SECRET");
```

backend/src/db/pool.ts

```
import { Pool } from "pg";
import { env } from "../config/env";

export const pool = new Pool({
  connectionString: env.DATABASE_URL
});
```

backend/src/db/migrate.ts

```
import fs from "fs";
import path from "path";
import { pool } from "./pool";

async function main() {
  await pool.query(`
    CREATE TABLE IF NOT EXISTS schema_migrations (
      id SERIAL PRIMARY KEY,
      filename TEXT UNIQUE NOT NULL,
      applied_at TIMESTAMPTZ NOT NULL DEFAULT now()
    );
  `);

  const dir = path.join(__dirname, "../../db/migrations");
  const files = fs.readdirSync(dir).filter(f => f.endsWith(".sql")).sort();

  for (const f of files) {
    const already = await pool.query(
      "SELECT 1 FROM schema_migrations WHERE filename = $1",
      [f]
    );
    if (already.rowCount) continue;
```

```
    const sql = fs.readFileSync(path.join(dir, f), "utf8");
    console.log("Applying migration:", f);
    await pool.query("BEGIN");
    try {
      await pool.query(sql);
      await pool.query("INSERT INTO schema_migrations(filename) VALUES ($1)", [f]);
      await pool.query("COMMIT");
    } catch (e) {
      await pool.query("ROLLBACK");
      throw e;
    }
  }

  console.log("Migrations OK");
  await pool.end();
}

main().catch(err => {
  console.error(err);
  process.exit(1);
});
```

backend/src/utils/geo.ts

```
export function parseBBox(bbox?: string): [number, number, number, number] | null {
  if (!bbox) return null;
  const parts = bbox.split(",").map(s => s.trim());
  if (parts.length !== 4) return null;
  const nums = parts.map(x => Number(x));
  if (nums.some(n => Number.isNaN(n))) return null;
  const [minLon, minLat, maxLon, maxLat] = nums;
  if (minLon >= maxLon || minLat >= maxLat) return null;
  return [minLon, minLat, maxLon, maxLat];
}
```

backend/src/schemas/common.ts

```
import { z } from "zod";

export const listQuerySchema = z.object({
  bbox: z.string().optional(),
  neighborhood: z.string().optional(),
  district: z.string().optional(),
  postal_code: z.string().optional(),
  price_min: z.coerce.number().optional(),
  price_max: z.coerce.number().optional(),
  roi_min: z.coerce.number().optional(),
  roi_max: z.coerce.number().optional(),
  comprable: z.coerce.number().int().optional(),
  amenities: z.string().optional(),
  limit: z.coerce.number().int().optional(),
  offset: z.coerce.number().int().optional()
});

export const rankingQuerySchema = z.object({
  scope: z.enum(["neighborhood", "viewport"]).optional(),
  neighborhood: z.string().optional(),
```

```
  bbox: z.string().optional(),
  mode: z.enum(["roi", "gap", "effective_price", "liquidity"]).optional(),
  direction: z.enum(["asc", "desc"]).optional(),
  limit: z.coerce.number().int().optional()
});

export const registerSchema = z.object({
  email: z.string().email(),
  password: z.string().min(8)
});

export const loginSchema = registerSchema;
```

backend/src/middlewares/validate.ts

```
import { Request, Response, NextFunction } from "express";
import { ZodSchema } from "zod";

export function validateQuery(schema: ZodSchema) {
  return (req: Request, res: Response, next: NextFunction) => {
    const out = schema.safeParse(req.query);
    if (!out.success) return res.status(400).json({ message: "Query invlida", issues:
    out.error.issues });
    req.query = out.data as any;
    next();
  };
}

export function validateBody(schema: ZodSchema) {
  return (req: Request, res: Response, next: NextFunction) => {
    const out = schema.safeParse(req.body);
    if (!out.success) return res.status(400).json({ message: "Body invlido", issues: out
    .error.issues });
    req.body = out.data;
    next();
  };
}
```

backend/src/middlewares/auth.ts

```
import { Request, Response, NextFunction } from "express";
import jwt from "jsonwebtoken";
import { env } from "../config/env";

export type JwtUser = { sub: string; role: "user" | "admin" };

declare global {
  namespace Express {
    interface Request { user?: JwtUser; }
  }
}

export function requireAuth(req: Request, res: Response, next: NextFunction) {
  const h = req.headers.authorization ?? "";
  const token = h.startsWith("Bearer ") ? h.slice(7) : null;
  if (!token) return res.status(401).json({ message: "No autenticado" });
  try {
```

```
    req.user = jwt.verify(token, env.JWT_SECRET) as JwtUser;
    return next();
  } catch {
    return res.status(401).json({ message: "Token invlido/expirado" });
  }
}

export function requireRole(role: "user" | "admin") {
  return (req: Request, res: Response, next: NextFunction) => {
    if (!req.user) return res.status(401).json({ message: "No autenticado" });
    if (req.user.role !== role) return res.status(403).json({ message: "Prohibido" });
    next();
  };
}
```

backend/src/routes/health.ts

```
import { Router } from "express";
export const healthRouter = Router();
healthRouter.get("/", (_req, res) => res.json({ ok: true }));
```

backend/src/routes/properties.ts

```
import { Router } from "express";
import { pool } from "../db/pool";
import { validateQuery } from "../middlewares/validate";
import { listQuerySchema } from "../schemas/common";
import { parseBBox } from "../utils/geo";

export const propertiesRouter = Router();

propertiesRouter.get("/", validateQuery(listQuerySchema), async (req, res) => {
  const q = req.query as any;

  const bbox = parseBBox(q.bbox);
  const limit = Math.min(q.limit ?? 2000, 10000);
  const offset = q.offset ?? 0;

  // filtros
  const where: string[] = [];
  const params: any[] = [];
  let i = 1;

  if (bbox) {
    where.push(`p.geom && ST_MakeEnvelope($${i++}, $${i++}, $${i++}, $${i++}, 4326)`);
    params.push(bbox[0], bbox[1], bbox[2], bbox[3]);
  }

  if (q.neighborhood) { where.push(`p.neighborhood = $${i++}`); params.push(q.
    neighborhood); }
  if (q.district)     { where.push(`p.district = $${i++}`); params.push(q.district); }
  if (q.postal_code)  { where.push(`p.postal_code = $${i++}`); params.push(q.postal_code
    ); }

  if (q.price_min !== undefined) { where.push(`p.price >= $${i++}`); params.push(q.
    price_min); }
```

```
if (q.price_max !== undefined) { where.push(`p.price <= $${i++}`); params.push(q.
  price_max); }
if (q.roi_min !== undefined)   { where.push(`p.roi >= $${i++}`); params.push(q.roi_min
  ); }
if (q.roi_max !== undefined)   { where.push(`p.roi <= $${i++}`); params.push(q.roi_max
  ); }
if (q.comprable !== undefined) { where.push(`p.comprable = $${i++}`); params.push(q.
  comprable); }

// amenities=lift,terrace,... (solo allowlist)
const allowedAmen = new Set(["lift","garage","storage","terrace","air_conditioning","
  swimming_pool","garden","sports","new_construction"]);
if (q.amenities) {
  const items = String(q.amenities).split(",").map((s: string) => s.trim()).filter(
  Boolean);
  for (const a of items) {
    if (!allowedAmen.has(a)) return res.status(400).json({ message: `Amenity invlida:
  ${a}` });
    where.push(`p.${a} = 1`);
  }
}

const whereSql = where.length ? `WHERE ${where.join(" AND ")}` : "";

// GeoJSON FeatureCollection (ligero para mapa)
const sql = `
  SELECT jsonb_build_object(
    'type','FeatureCollection',
    'features', COALESCE(jsonb_agg(
      jsonb_build_object(
        'type','Feature',
        'geometry', ST_AsGeoJSON(p.geom)::jsonb,
        'properties', jsonb_build_object(
          'id', p.id,
          'price', p.price,
          'price_m2', p.price_m2,
          'roi', p.roi,
          'bedrooms', p.bedrooms,
          'bathrooms', p.bathrooms,
          'cm', p.cm,
          'ck', p.ck,
          'gap', (p.cm / NULLIF(p.ck,0)),
          'effective_price', (p.price_m2 * p.cm),
          'neighborhood', p.neighborhood,
          'district', p.district
        )
      )
    ), '[]'::jsonb)
  ) AS geojson
  FROM (
    SELECT *
    FROM properties p
    ${whereSql}
    ORDER BY p.id
    LIMIT $${i++} OFFSET $${i++}
  ) p;
`;

params.push(limit, offset);
```

```
  const r = await pool.query(sql, params);
  res.json(r.rows[0].geojson);
});

propertiesRouter.get("/:id", async (req, res) => {
  const id = Number(req.params.id);
  if (Number.isNaN(id)) return res.status(400).json({ message: "id invlido" });

  const sql = `
    SELECT
      p.*,
      (p.cm / NULLIF(p.ck,0)) AS gap,
      (p.price_m2 * p.cm) AS effective_price
    FROM properties p
    WHERE p.id = $1
  `;
  const r = await pool.query(sql, [id]);
  if (!r.rowCount) return res.status(404).json({ message: "No encontrado" });
  res.json(r.rows[0]);
});
```

backend/src/routes/catalog.ts

```
import { Router } from "express";
import { pool } from "../db/pool";

export const catalogRouter = Router();

catalogRouter.get("/neighborhoods", async (_req, res) => {
  const r = await pool.query(`
    SELECT neighborhood, COUNT(*)::int AS count
    FROM properties
    WHERE neighborhood IS NOT NULL
    GROUP BY neighborhood
    ORDER BY count DESC, neighborhood ASC
  `);
  res.json(r.rows);
});

catalogRouter.get("/districts", async (_req, res) => {
  const r = await pool.query(`
    SELECT district, COUNT(*)::int AS count
    FROM properties
    WHERE district IS NOT NULL
    GROUP BY district
    ORDER BY count DESC, district ASC
  `);
  res.json(r.rows);
});
```

backend/src/routes/ranking.ts

```
import { Router } from "express";
import { pool } from "../db/pool";
import { validateQuery } from "../middlewares/validate";
import { rankingQuerySchema } from "../schemas/common";
```

```
import { parseBBox } from "../utils/geo";

export const rankingRouter = Router();

rankingRouter.get("/", validateQuery(rankingQuerySchema), async (req, res) => {
  const q = req.query as any;
  const scope = q.scope ?? "neighborhood";
  const mode = q.mode ?? "roi";

  // defaults por modo
  const defaultDir: Record<string,"asc"|"desc"> = {
    roi: "desc",
    gap: "desc",
    effective_price: "asc",  // "valor" por default: ms bajo mejor
    liquidity: "desc"
  };
  const direction = q.direction ?? defaultDir[mode];
  const limit = Math.min(q.limit ?? 50, 500);

  // ORDER BY seguro (whitelist)
  const orderExpr: Record<string,string> = {
    roi: "p.roi",
    gap: "(p.cm / NULLIF(p.ck,0))",
    effective_price: "(p.price_m2 * p.cm)",
    liquidity: "p.cm"
  };
  const order = orderExpr[mode];
  if (!order) return res.status(400).json({ message: "mode invlido" });

  const where: string[] = [];
  const params: any[] = [];
  let i = 1;

  if (scope === "neighborhood") {
    if (!q.neighborhood) return res.status(400).json({ message: "Falta neighborhood" });
    where.push(`p.neighborhood = $${i++}`);
    params.push(q.neighborhood);
  } else if (scope === "viewport") {
    const bbox = parseBBox(q.bbox);
    if (!bbox) return res.status(400).json({ message: "bbox requerido e invlido" });
    where.push(`p.geom && ST_MakeEnvelope($${i++},$${i++},$${i++},$${i++},4326)`);
    params.push(bbox[0], bbox[1], bbox[2], bbox[3]);
  } else {
    return res.status(400).json({ message: "scope invlido" });
  }

  const whereSql = `WHERE ${where.join(" AND ")}`;

  const sql = `
    SELECT
      ROW_NUMBER() OVER (ORDER BY ${order} ${direction})::int AS rank,
      p.id, p.title, p.neighborhood, p.district,
      p.price, p.price_m2, p.size, p.bedrooms, p.bathrooms,
      p.roi, p.comprable, p.cm, p.ck,
      (p.cm / NULLIF(p.ck,0)) AS gap,
      (p.price_m2 * p.cm) AS effective_price
    FROM properties p
    ${whereSql}
    ORDER BY ${order} ${direction}
```

```
    LIMIT $${i++}
  `;
  params.push(limit);

  const r = await pool.query(sql, params);
  res.json({ scope, mode, direction, limit, items: r.rows });
});
```

backend/src/routes/auth.ts

```
import { Router } from "express";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import { pool } from "../db/pool";
import { env } from "../config/env";
import { validateBody } from "../middlewares/validate";
import { registerSchema, loginSchema } from "../schemas/common";

export const authRouter = Router();

authRouter.post("/register", validateBody(registerSchema), async (req, res) => {
  const { email, password } = req.body as any;
  const hash = await bcrypt.hash(password, 12);

  try {
    const r = await pool.query(
      `INSERT INTO users(email, password_hash, role)
       VALUES ($1,$2,'user')
       RETURNING id, email, role`,
      [email.toLowerCase(), hash]
    );
    res.status(201).json(r.rows[0]);
  } catch (e: any) {
    if (String(e?.code) === "23505") return res.status(409).json({ message: "Email ya
    existe" });
    throw e;
  }
});

authRouter.post("/login", validateBody(loginSchema), async (req, res) => {
  const { email, password } = req.body as any;

  const r = await pool.query(
    `SELECT id, email, role, password_hash FROM users WHERE email = $1`,
    [email.toLowerCase()]
  );
  if (!r.rowCount) return res.status(401).json({ message: "Credenciales invlidas" });

  const user = r.rows[0];
  const ok = await bcrypt.compare(password, user.password_hash);
  if (!ok) return res.status(401).json({ message: "Credenciales invlidas" });

  const token = jwt.sign({ sub: String(user.id), role: user.role }, env.JWT_SECRET, {
    expiresIn: "2h" });
  res.json({ token });
});
```

backend/src/routes/user.ts

```
import { Router } from "express";
import { pool } from "../db/pool";
import { requireAuth } from "../middlewares/auth";

export const userRouter = Router();

userRouter.get("/favorites", requireAuth, async (req, res) => {
  const userId = Number(req.user!.sub);
  const r = await pool.query(
    `SELECT f.property_id, p.title, p.neighborhood, p.price, p.roi
     FROM favorites f
     JOIN properties p ON p.id = f.property_id
     WHERE f.user_id = $1
     ORDER BY f.created_at DESC`,
    [userId]
  );
  res.json(r.rows);
});

userRouter.post("/favorites/:propertyId", requireAuth, async (req, res) => {
  const userId = Number(req.user!.sub);
  const propertyId = Number(req.params.propertyId);
  if (Number.isNaN(propertyId)) return res.status(400).json({ message: "propertyId
    invlido" });

  await pool.query(
    `INSERT INTO favorites(user_id, property_id) VALUES ($1,$2)
     ON CONFLICT DO NOTHING`,
    [userId, propertyId]
  );
  res.status(204).send();
});

userRouter.delete("/favorites/:propertyId", requireAuth, async (req, res) => {
  const userId = Number(req.user!.sub);
  const propertyId = Number(req.params.propertyId);
  if (Number.isNaN(propertyId)) return res.status(400).json({ message: "propertyId
    invlido" });

  await pool.query(
    `DELETE FROM favorites WHERE user_id = $1 AND property_id = $2`,
    [userId, propertyId]
  );
  res.status(204).send();
});
```

backend/src/routes/index.ts

```
import { Router } from "express";
import { healthRouter } from "./health";
import { propertiesRouter } from "./properties";
import { catalogRouter } from "./catalog";
import { rankingRouter } from "./ranking";
import { authRouter } from "./auth";
import { userRouter } from "./user";
```

```
export const apiRouter = Router();

apiRouter.use("/health", healthRouter);
apiRouter.use("/api/properties", propertiesRouter);
apiRouter.use("/api/catalog", catalogRouter);
apiRouter.use("/api/ranking", rankingRouter);
apiRouter.use("/api/auth", authRouter);
apiRouter.use("/api/user", userRouter);
```

backend/src/app.ts

```
import express from "express";
import cors from "cors";
import helmet from "helmet";
import rateLimit from "express-rate-limit";
import { env } from "./config/env";
import { apiRouter } from "./routes/index";

export function createApp() {
  const app = express();

  app.use(express.json({ limit: "2mb" }));
  app.use(helmet());

  app.use(cors({
    origin: env.CORS_ORIGINS.length ? env.CORS_ORIGINS : false,
    credentials: true
  }));

  app.use("/api", rateLimit({
    windowMs: env.RATE_LIMIT_WINDOW_MS,
    max: env.RATE_LIMIT_MAX
  }));

  app.use(apiRouter);

  app.use((err: any, _req: any, res: any, _next: any) => {
    console.error(err);
    res.status(500).json({ message: "Error interno" });
  });

  return app;
}
```

backend/src/server.ts

```
import { env } from "./config/env";
import { createApp } from "./app";

const app = createApp();
app.listen(env.PORT, () => {
  console.log(`API listening on http://localhost:${env.PORT}`);
});
```

backend/db/migrations/001_init.sql

```
CREATE EXTENSION IF NOT EXISTS postgis;

-- Usuarios
CREATE TABLE IF NOT EXISTS users (
  id BIGSERIAL PRIMARY KEY,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role TEXT NOT NULL CHECK (role IN ('user','admin')),
  created_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Propiedades (subset "core" para la UI + ranking)
CREATE TABLE IF NOT EXISTS properties (
  id BIGINT PRIMARY KEY,
  title TEXT,
  district TEXT,
  neighborhood TEXT,
  postal_code TEXT,
  latitude DOUBLE PRECISION,
  longitude DOUBLE PRECISION,
  geom geometry(Point, 4326),

  -- Descripcin
  property_type TEXT,
  subtype TEXT,
  size NUMERIC,
  bedrooms INT,
  bathrooms INT,
  floor TEXT,
  status TEXT,
  new_construction INT,

  -- Precio
  price NUMERIC,
  price_m2 NUMERIC,

  -- Amenities (0/1)
  lift INT,
  garage INT,
  storage INT,
  terrace INT,
  air_conditioning INT,
  swimming_pool INT,
  garden INT,
  sports INT,

  -- Contexto
  ingreso NUMERIC,

  -- Clasificacin / inversin
  vi NUMERIC,
  vo NUMERIC,
  comprable INT,
  roi NUMERIC,

  -- Probabilidades (2 meses)
  ck NUMERIC, -- propietario (prob_propietario_1)
  cm NUMERIC, -- inversor (prob_investor_1)
```

```
  created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT now()
);

-- Favoritos
CREATE TABLE IF NOT EXISTS favorites (
  user_id BIGINT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  property_id BIGINT NOT NULL REFERENCES properties(id) ON DELETE CASCADE,
  created_at TIMESTAMPTZ NOT NULL DEFAULT now(),
  PRIMARY KEY (user_id, property_id)
);

-- Poblar geom desde lon/lat si existiera (por si insertan sin geom)
UPDATE properties
SET geom = ST_SetSRID(ST_MakePoint(longitude, latitude), 4326)
WHERE geom IS NULL AND longitude IS NOT NULL AND latitude IS NOT NULL;

-- ndices
CREATE INDEX IF NOT EXISTS properties_geom_gix ON properties USING GIST (geom);
CREATE INDEX IF NOT EXISTS properties_neighborhood_idx ON properties(neighborhood);
CREATE INDEX IF NOT EXISTS properties_district_idx ON properties(district);
CREATE INDEX IF NOT EXISTS properties_postal_idx ON properties(postal_code);
CREATE INDEX IF NOT EXISTS properties_price_idx ON properties(price);
CREATE INDEX IF NOT EXISTS properties_roi_idx ON properties(roi);
CREATE INDEX IF NOT EXISTS properties_comprable_idx ON properties(comprable);
```

backend/scripts/requirements.txt

```
pandas==2.2.2
openpyxl==3.1.5
psycopg2-binary==2.9.9
```

backend/scripts/import_excel.py

```
import os
import pandas as pd
import psycopg2
from psycopg2.extras import execute_values

EXCEL_PATH = os.path.join(os.path.dirname(__file__), "data_ROI.xlsx")
DATABASE_URL = os.environ.get("DATABASE_URL")

REQUIRED_COLS = [
    "id","title","district","neighborhood","postal_code","latitude","longitude",
    "property_type","subtype","size","bedrooms","bathrooms","floor","status","
    new_construction",
    "price","price_m2",
    "lift","garage","storage","terrace","air_conditioning","swimming_pool","garden","
    sports",
    "ingreso","VI","VO","comprable","ROI",
    "prob_propietario_1","prob_investor_1"
]

def main():
    if not DATABASE_URL:
        raise RuntimeError("Falta DATABASE_URL")
```

```
if not os.path.exists(EXCEL_PATH):
    raise RuntimeError(f"No existe {EXCEL_PATH}. Copi tu Excel ah.")

df = pd.read_excel(EXCEL_PATH)

missing = [c for c in REQUIRED_COLS if c not in df.columns]
if missing:
    raise RuntimeError(f"Faltan columnas en Excel: {missing}")

# Normalizacin: NaN -> None
df = df[REQUIRED_COLS].copy()
df = df.where(pd.notnull(df), None)

# ck/cm (2 meses)
df["ck"] = df["prob_propietario_1"]
df["cm"] = df["prob_investor_1"]

# renombrar a minsculas para DB
df = df.rename(columns={
    "VI":"vi",
    "VO":"vo",
    "ROI":"roi"
})

rows = []
for _, r in df.iterrows():
    rows.append((
        int(r["id"]),
        r["title"], r["district"], r["neighborhood"], str(r["postal_code"]) if r["
postal_code"] is not None else None,
        float(r["latitude"]) if r["latitude"] is not None else None,
        float(r["longitude"]) if r["longitude"] is not None else None,
        r["property_type"], r["subtype"],
        float(r["size"]) if r["size"] is not None else None,
        int(r["bedrooms"]) if r["bedrooms"] is not None else None,
        int(r["bathrooms"]) if r["bathrooms"] is not None else None,
        str(r["floor"]) if r["floor"] is not None else None,
        r["status"],
        int(r["new_construction"]) if r["new_construction"] is not None else None,
        float(r["price"]) if r["price"] is not None else None,
        float(r["price_m2"]) if r["price_m2"] is not None else None,
        int(r["lift"]) if r["lift"] is not None else None,
        int(r["garage"]) if r["garage"] is not None else None,
        int(r["storage"]) if r["storage"] is not None else None,
        int(r["terrace"]) if r["terrace"] is not None else None,
        int(r["air_conditioning"]) if r["air_conditioning"] is not None else None,
        int(r["swimming_pool"]) if r["swimming_pool"] is not None else None,
        int(r["garden"]) if r["garden"] is not None else None,
        int(r["sports"]) if r["sports"] is not None else None,
        float(r["ingreso"]) if r["ingreso"] is not None else None,
        float(r["vi"]) if r["vi"] is not None else None,
        float(r["vo"]) if r["vo"] is not None else None,
        int(r["comprable"]) if r["comprable"] is not None else None,
        float(r["roi"]) if r["roi"] is not None else None,
        float(r["ck"]) if r["ck"] is not None else None,
        float(r["cm"]) if r["cm"] is not None else None
    ))

conn = psycopg2.connect(DATABASE_URL)
```

```
conn.autocommit = False
try:
    with conn.cursor() as cur:
        sql = """
          INSERT INTO properties(
            id,title,district,neighborhood,postal_code,latitude,longitude,
            property_type,subtype,size,bedrooms,bathrooms,floor,status,
new_construction,
            price,price_m2,
            lift,garage,storage,terrace,air_conditioning,swimming_pool,garden,sports
,
            ingreso,vi,vo,comprable,roi,ck,cm,
            geom,updated_at
          )
          VALUES %s
          ON CONFLICT (id) DO UPDATE SET
            title=EXCLUDED.title,
            district=EXCLUDED.district,
            neighborhood=EXCLUDED.neighborhood,
            postal_code=EXCLUDED.postal_code,
            latitude=EXCLUDED.latitude,
            longitude=EXCLUDED.longitude,
            property_type=EXCLUDED.property_type,
            subtype=EXCLUDED.subtype,
            size=EXCLUDED.size,
            bedrooms=EXCLUDED.bedrooms,
            bathrooms=EXCLUDED.bathrooms,
            floor=EXCLUDED.floor,
            status=EXCLUDED.status,
            new_construction=EXCLUDED.new_construction,
            price=EXCLUDED.price,
            price_m2=EXCLUDED.price_m2,
            lift=EXCLUDED.lift,
            garage=EXCLUDED.garage,
            storage=EXCLUDED.storage,
            terrace=EXCLUDED.terrace,
            air_conditioning=EXCLUDED.air_conditioning,
            swimming_pool=EXCLUDED.swimming_pool,
            garden=EXCLUDED.garden,
            sports=EXCLUDED.sports,
            ingreso=EXCLUDED.ingreso,
            vi=EXCLUDED.vi,
            vo=EXCLUDED.vo,
            comprable=EXCLUDED.comprable,
            roi=EXCLUDED.roi,
            ck=EXCLUDED.ck,
            cm=EXCLUDED.cm,
            geom=EXCLUDED.geom,
            updated_at=now()
        """

        values = []
        for row in rows:
            (id_, title, district, neighborhood, postal_code, lat, lon,
             property_type, subtype, size, bedrooms, bathrooms, floor, status,
new_construction,
             price, price_m2, lift, garage, storage, terrace, air_conditioning,
swimming_pool, garden, sports,
             ingreso, vi, vo, comprable, roi, ck, cm) = row
```

```
                geom = None
                if lon is not None and lat is not None:
                    geom = f"SRID=4326;POINT({lon} {lat})"

                values.append((
                    id_, title, district, neighborhood, postal_code, lat, lon,
                    property_type, subtype, size, bedrooms, bathrooms, floor, status,
    new_construction,
                    price, price_m2, lift, garage, storage, terrace, air_conditioning,
    swimming_pool, garden, sports,
                    ingreso, vi, vo, comprable, roi, ck, cm,
                    geom
                ))

            # execute_values soporta geometra como WKT EWKT si casteamos
            template = "(" + ",".join(["%s"]*31) + ", ST_GeomFromEWKT(%s), now())"
            execute_values(cur, sql, values, template=template, page_size=500)

            # asegurar geom para filas sin seteo previo
            cur.execute("""
              UPDATE properties
              SET geom = ST_SetSRID(ST_MakePoint(longitude, latitude), 4326)
              WHERE geom IS NULL AND longitude IS NOT NULL AND latitude IS NOT NULL
            """)

        conn.commit()
        print(f"Import OK. Filas: {len(rows)}")
    except Exception:
        conn.rollback()
        raise
    finally:
        conn.close()

if __name__ == "__main__":
    main()
```

.github/workflows/ci.yml

```
name: CI
on:
  push:
  pull_request:

jobs:
  backend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: actions/setup-node@v4
        with:
          node-version: "20"

      - name: Install
        working-directory: backend
        run: npm install
```

```
      - name: Typecheck build
        working-directory: backend
        run: npm run build
```

README.md

```
# BCN ROI API (Back-end)

## Requisitos
- Docker + Docker Compose

## Setup
1) Copiar ‘.env.example‘ a ‘.env‘ (opcional, defaults OK)
2) Poner ‘data_ROI.xlsx‘ en ‘backend/scripts/data_ROI.xlsx‘

## Run
docker compose up -d --build

## Migraciones
docker compose exec api npm run migrate

## Import Excel
docker compose exec api python backend/scripts/import_excel.py

## Endpoints
- GET  /health
- GET  /api/properties?bbox=minLon,minLat,maxLon,maxLat&...
- GET  /api/properties/:id
- GET  /api/catalog/neighborhoods
- GET  /api/catalog/districts
- GET  /api/ranking?scope=neighborhood&neighborhood=...&mode=roi
- POST /api/auth/register
- POST /api/auth/login
- GET/POST/DELETE /api/user/favorites (requiere JWT)
```

9. Notas operativas (para que no haya dudas)

9.1. Qué devuelve el endpoint del mapa

`/api/properties` devuelve **GeoJSON** (FeatureCollection). Cada feature contiene:

- geometry: punto (lon/lat)
- properties: variables “ligeras” para tooltip/marker (price, ROI, bedrooms, cm, ck, gap, effective_price, etc.)

El front-end puede elegir el par ordenado (x,y) usando estas propiedades sin pedir más.

9.2. Detalle de propiedad

`/api/properties/:id` devuelve todas las columnas principales para armar la ficha con dos bloques:

- Descripción: title, barrio/distrito, postal_code, subtype, size, bedrooms, bathrooms, floor, amenities, status, ingreso.
- Clasificación: ck, cm, vi, vo, roi, comprable, gap y effective_price.

9.3. Outliers y performance

Con 7.220 puntos, la estrategia recomendada es:

- En mapa: usar `bbox` siempre que el usuario mueva/zoomee.
- En UI: si quieren recorte p95/p99, se implementa como filtros opcionales desde el front-end (pidiendo `price_max`).

10. Checklist de despliegue (producción)

Para producción, además de lo anterior:

- Usar HTTPS (reverse proxy o plataforma)
- Rotar `JWT_SECRET` y no usar valores por defecto
- Setear CORS a dominios reales
- Ajustar rate limit según tráfico
- Backups de Postgres
- GitHub Secrets para variables sensibles