Admin (Supervisor) System – Build Prompts

Below are **two copy-paste prompts** to build a simple, clear **Admin System** for supervising the whole platform. Code must be **easy to read** for non-technical reviewers: small files, plain naming, clear comments, gentle abstractions only.

- **Frontend → LevelUp** (Admin UI)
- **Backend → Replit** (REST API + DB)

  Keep parity with the previously defined Properties/Parents models. Add users/roles, leases, payments, maintenance, and audit logs.

---

1) Prompt for ** (Frontend – Admin UI)**

You are an AI that builds a **simple, readable Admin UI** for the **Supervisor** role of the Property Managers Platform. Use the spec below. Prioritize clarity over cleverness: straightforward components, plain props, comments on tricky parts.

Tech & Style

- **Framework:** Next.js 14 (App Router) + React + TypeScript.
- **UI:** Tailwind CSS + shadcn/ui, lucide-react icons.
- **Data:** Fetch via REST using a tiny wrapper `api.ts` (GET/POST/PATCH/DELETE). No global state library; use SWR or React Query simply.
- **Structure:** `/app/(admin)/admin/...` routes. Each page folder contains `page.tsx`, a small `Form.tsx` (if needed), and a `Table.tsx`.
- **Readability Rules:**
- Keep files short (<200 lines where possible).
- Use clear names (e.g., `UserForm`, `LeaseTable`).
- Add 1–2 line comments for any non-obvious logic.

Navigation

Left sidebar with sections:

- **Dashboard** (overview cards + simple charts)
- **Users & Roles**
- **Parent RealEstates**
- **Properties**
- **Leases (Contracts)**
- **Payments**
- **Maintenance**
- **Media & Files**
- **Lookups (Region/City)**
- **Notifications**
- **Audit Logs**
- **Settings**

Pages & Core Actions

Dashboard

- KPI cards: Total Parents, Properties, Occupancy %, Active Leases, Overdue Payments, Open Tickets.
- Simple charts: Occupancy by status, Payments collected this month.

Users & Roles

- List, search by name/email, filter by role (Supervisor, Manager, Viewer).
- Create/Edit user: name, email, role, status (active/disabled), optional phone.
- Reset password action (admin-triggered link/stub).
- Role permissions (checkboxes): CanManageUsers, CanManageInventory, CanViewFinance, CanEditSettings, etc.

Parent RealEstates

- CRUD like earlier spec (name, description, region, city, district, address).
- Show number of child properties; link to filtered Properties.

Properties

- CRUD with fields: parent, name, address, region, city, description, status (RENTED/RESERVED/VACANT), cost, sqm, images.
- Bulk import (CSV) and bulk status update.

Leases (Contracts)

- Fields: property, tenantName (string), tenantPhone/email (optional), startDate, endDate, rentAmount (SAR), paymentSchedule (monthly/quarterly), deposit (optional), status (Active/Expired/Cancelled), files (PDF).
- Actions: create, renew (new dates), terminate, upload contract file.

Payments

- Record payment: lease, amount, date, method (bank transfer/cash), reference.
- Auto-calc “amount due” from lease schedule; show overdue list.
- Export CSV of payments and dues.

Maintenance

- Ticket fields: property, title, description, priority (Low/Med/High), status (Open/In Progress/Resolved), assignedTo (optional), photos.
- Actions: create ticket, update status, add note, close ticket.

Media & Files

- Simple gallery/list of uploaded files with preview and delete.

Lookups (Region/City)

- Manage dropdown data: regions and their cities. CRUD with safety checks (cannot delete if in use).

Notifications

- Compose and send: email/SMS placeholders (no real integration; log to backend).
- Templates: payment due reminder, lease expiry notice, maintenance scheduled.

Audit Logs

- Table of events (actor, action, entity, entityId, timestamp, changes JSON). Search by entity or actor.

Settings

- Platform name, currency (default SAR), timezone, date format.
- Storage settings (readonly in UI; hints for backend).

UI Patterns

- Tables with search input and simple filters.
- Forms in side-drawers or modals with clear labels and helper text.
- Confirm dialogs for destructive actions.
- Toasts for success/error.
- Pagination for lists; page size 10/25/50.

API Contract (consume these routes)

Base: `/api`.

- **Users**: `GET /users`, `POST /users`, `PATCH /users/:id`, `DELETE /users/:id`.
- **Roles**: `GET /roles`, `PATCH /roles/:id` (permissions array).
- **Parents**: `GET/POST /parents`, `PATCH/DELETE /parents/:id`.
- **Properties**: `GET/POST /properties`, `PATCH/DELETE /properties/:id`.
- **Leases**: `GET/POST /leases`, `PATCH /leases/:id`, `POST /leases/:id/renew`, `POST /leases/:id/terminate`.
- **Payments**: `GET/POST /payments`.
- **Maintenance**: `GET/POST /tickets`, `PATCH /tickets/:id`.
- **Files**: `GET /files`, `POST /files` (upload), `DELETE /files/:id`.
- **Lookups**: `GET/POST /lookups/regions`, `GET/POST /lookups/cities`, `DELETE` with guard.
- **Notifications**: `POST /notifications/send`.
- **Audit**: `GET /audit`.
- **Settings**: `GET/POST /settings`.

Expect JSON `{ data, meta? }`, errors `{ fieldErrors? , message }`.

Acceptance Criteria (UI)

1. Every page supports list, search, and basic CRUD with validation and toasts.
2. All forms are simple and clearly labeled; inline errors explained in simple words.
3. Bulk import for Properties accepts CSV and shows a dry-run preview (rows to insert/update with errors highlighted).
4. Audit log visible and filterable.
5. All actions hit the routes above and handle failures gracefully.

---

2) Prompt for Replit (Backend – REST API)

You are an AI that builds a **clean and readable** Node.js backend for the Admin System. Use **Express + Prisma + PostgreSQL**. Favor simplicity: one router per resource, flat controllers with clear comments, no complex patterns. Return friendly error messages.

Tech

- Node 20, Express, Prisma (PostgreSQL). Multer for file upload. Zod for validation. Helmet, CORS, morgan. Dotenv.
- Folder structure (flat & clear):

```
src/
  index.ts            // server bootstrap
  db.ts               // Prisma client
  routes/
    users.ts
    roles.ts
    parents.ts
    properties.ts
    leases.ts
    payments.ts
    tickets.ts
    files.ts
    lookups.ts
    notifications.ts
    audit.ts
    settings.ts
  controllers/        // matching file per route with plain functions
  validators/         // zod schemas per resource
  middleware/
    errorHandler.ts
    notFound.ts
    upload.ts         // multer config
  utils/
    pagination.ts
    responses.ts      // helpers for {data, meta}
    simpleAuth.ts     // stub: attach req.user = {id:"admin",
role:"Supervisor"}
/uploads              // local files
```

Prisma Schema (entities)

```
enum Role { SUPERVISOR MANAGER VIEWER }

enum PropertyStatus { RENTED RESERVED VACANT }

enum LeaseStatus { ACTIVE EXPIRED CANCELLED }

enum TicketPriority { LOW MEDIUM HIGH }

enum TicketStatus { OPEN IN_PROGRESS RESOLVED }

model User {
  id        String @id @default(cuid())
  name      String
  email     String @unique
  role      Role   @default(MANAGER)
  phone     String?
  isActive  Boolean @default(true)
  createdAt DateTime @default(now())
}

model RolePermission {
  id        String @id @default(cuid())
  role      Role
  permissions String[] // e.g., ["CanManageUsers","CanViewFinance"]
}

model ParentRealEstate {
  id          String   @id @default(cuid())
  name        String
  description String?
  region      String
  city        String
  district    String
  address     String
  properties  Property[]
  createdAt   DateTime @default(now())
}

model Property {
  id          String   @id @default(cuid())
  parentId    String
  parent      ParentRealEstate @relation(fields: [parentId], references:
[id])
  name        String
  address     String
  region      String
  city        String
  description String?
  status      PropertyStatus @default(VACANT)
  cost        Decimal  @db.Decimal(12,2)
  sqm         Int
  images      PropertyImage[]
  leases      Lease[]
  deletedAt   DateTime?
  createdAt   DateTime @default(now())
}

model PropertyImage {
  id         String   @id @default(cuid())
  propertyId String
  property   Property @relation(fields: [propertyId], references: [id])
  url        String
  alt        String?
  createdAt  DateTime @default(now())
}

model Lease {
  id            String   @id @default(cuid())
  propertyId    String
  property      Property @relation(fields: [propertyId], references: [id])
  tenantName    String
  tenantEmail   String?
  tenantPhone   String?
  startDate     DateTime
  endDate       DateTime
  rentAmount    Decimal  @db.Decimal(12,2)
  paymentSchedule String // "monthly" | "quarterly"
  deposit       Decimal? @db.Decimal(12,2)
  status        LeaseStatus @default(ACTIVE)
  files         File[]
  payments      Payment[]
  createdAt     DateTime @default(now())
}

model Payment {
  id        String   @id @default(cuid())
  leaseId   String
  lease     Lease    @relation(fields: [leaseId], references: [id])
  amount    Decimal  @db.Decimal(12,2)
  method    String
  paidAt    DateTime
  reference String?
}

model Ticket {
  id          String   @id @default(cuid())
  propertyId  String
  property    Property @relation(fields: [propertyId], references: [id])
  title       String
  description String
  priority    TicketPriority @default(MEDIUM)
  status      TicketStatus @default(OPEN)
  assignedTo  String?
  photos      File[]
  createdAt   DateTime @default(now())
}

model File {
  id        String @id @default(cuid())
  url       String
  kind      String  // "image" | "pdf" | "other"
  note      String?
  createdAt DateTime @default(now())
  leaseId   String?
  ticketId  String?
}

model AuditLog {
  id        String @id @default(cuid())
  actorId   String
  action    String
  entity    String
  entityId  String
  changes   Json
  createdAt DateTime @default(now())
}

model LookupRegion {
  id    String @id @default(cuid())
  name  String @unique
  cities LookupCity[]
}

model LookupCity {
  id       String @id @default(cuid())
  regionId String
  region   LookupRegion @relation(fields: [regionId], references: [id])
  name     String
}

model SettingKV {
  key   String @id
  value String
}
```

Routes (match Frontend contract)

- **Users**: `GET /api/users?q=&role=&page=&pageSize=`, `POST`, `PATCH /:id`, `DELETE /:id`, `POST /:id/reset-password` (stub).
- **Roles**: `GET /api/roles`, `PATCH /api/roles/:id` (update permissions array).
- **Parents**: `GET/POST /api/parents`, `PATCH/DELETE /api/parents/:id`.
- **Properties**: `GET/POST /api/properties`, `PATCH/DELETE /api/properties/:id`, bulk: `POST /api/properties/import` (CSV dry-run with `apply=false|true`).
- **Leases**: `GET/POST /api/leases`, `PATCH /api/leases/:id`, `POST /api/leases/:id/renew`, `POST /api/leases/:id/terminate`.
- **Payments**: `GET/POST /api/payments` (filter by leaseId optional).
- **Maintenance**: `GET/POST /api/tickets`, `PATCH /api/tickets/:id`.
- **Files**: `GET /api/files`, `POST /api/files` (multer upload), `DELETE /api/files/:id`.
- **Lookups**: `GET/POST /api/lookups/regions`, `GET/POST /api/lookups/cities`, guarded deletes.
- **Notifications**: `POST /api/notifications/send` (logs the payload to AuditLog).
- **Audit**: `GET /api/audit?q=&entity=&actor=&page=`.
- **Settings**: `GET /api/settings`, `POST /api/settings`.

Validation (zod) – keep messages simple

- Friendly messages, e.g., "Name is required", "Amount must be a number $\geq$ 0".
- Reject uploads >5MB or not in allowed types: images (jpg/png/webp), pdf.

Middlewares

- `simpleAuth` (stub user in req.user), `helmet`, `cors`, `morgan`.
- `errorHandler` returns `{ message, fieldErrors? }` with proper HTTP codes.
- `audit` utility logs mutating actions (actor, action, entity, entityId, changes).

CSV Import (Properties)

- Accept CSV columns: parentName, name, address, region, city, description, status, cost, sqm.
- `apply=false` → return rows with `valid: true/false` and `errors[]`.
- `apply=true` → insert/update and return counts.

Responses

Always return `{ data, meta? }`. For lists include `{ page, pageSize, total, totalPages }`.

Setup (Replit)

1. Add secrets: `DATABASE_URL`, `PORT=3001`, `UPLOAD_DIR=./uploads`, `CORS_ORIGIN`.
2. Install: `express cors helmet morgan multer zod dotenv @prisma/client` and dev: `prisma ts-node-dev typescript @types/*`.
3. `npx prisma init` → paste schema → `npx prisma migrate dev`.
4. Create `/uploads` folder.
5. `npm run dev`.

Acceptance Criteria (Backend)

1. All endpoints above exist and pass basic validation.
2. Errors are human-readable and helpful.
3. Audit logs created for create/update/delete actions.
4. Properties CSV import supports dry-run and apply modes.
5. Pagination works and returns correct meta.

   Deliver code that is **clear, short, and well-commented** so a non-technical supervisor can open files and understand what each part does.