PHASE 0

🏢 StayWise

Smart PG & Property Management System

1️⃣ Executive Summary

NEXILRA Stay Finder is a SaaS-based property and PG management platform designed to digitize and simplify asset management for PG owners in Bengaluru.

The product provides:

- Micro-level occupancy tracking (Building → Room → Bed → Rack)
- Rent payment tracking
- Complaint management
- Tenant information management
- Role-based access control

The initial launch will focus only on PG owners in Bengaluru.

2️⃣ Market Definition

Primary Market:

PG Owners in Bengaluru

Why Bengaluru?

- Large student population

- Large IT workforce
- High density of PG clusters
- Frequent tenant turnover
- Highly fragmented management systems

3️⃣ Problem Statement

Owner Problems:

- Manual rent tracking
- No structured tenant database
- No real-time occupancy visibility
- Poor complaint tracking
- Dependence on physical registers

Tenant Problems:

- No transparent complaint channel
- No visibility of rent status
- Poor communication management

4️⃣ Product Positioning

NEXILRA is:

- A PG Management SaaS
- A Digital Occupancy System
- A Rent & Complaint Tracking Tool
- A Centralized Asset Control Platform

NEXILRA is not:

- A listing marketplace
- A brokerage service
- A public property portal

5️⃣ Target User Persona

Primary Persona – PG Owner

- Owns 1–3 PG buildings
- 30–150 beds
- Manages using Excel / Notebook
- Wants operational control
- Wants rent tracking automation

Secondary Persona – Tenant

- Student or Working Professional
- Needs complaint raise system
- Needs rent transparency

6️⃣ Product Scope – Version Strategy

Version 1 – MVP Scope

Owner Features:

- Create Building
- Create Rooms
- Create Beds
- Assign Tenant to Bed
- Mark Rent Paid/Unpaid
- Occupancy Dashboard

Tenant Features:

- View Assigned Bed
- Raise Complaint

Admin:

- Basic owner approval

Version 2 – Expansion

- Complaint history tracking
- Export reports

- Payment due notifications
- Tenant history
- Remarks tracking

Version 3 – Smart SaaS

- Online payment integration
- WhatsApp notifications
- Occupancy analytics
- AI-based pricing insight

7️⃣ Revenue Model

Subscription-based SaaS:

Starter – ₹999/month (1 building)
Growth – ₹1999/month (up to 3 buildings)
Custom – enterprise pricing

8️⃣ Strategic Focus

Launch Strategy:

- Narrow focus on PG owners only
- Do not expand scope early

- Build simple but stable foundation

Scaling Strategy:

- Build stable SaaS backend
- Add modules gradually
- Prioritize system reliability

9️⃣ Technical Direction (Initial Decision)

Frontend: React / Next.js
Backend: Node.js
Database: PostgreSQL
Authentication: Role-based access

🔟 Success Metrics (Phase 1)

- 5 PG owners onboarded
- 100+ beds managed via system
- Zero critical failures
- Positive operational feedback

PHASE 1

🏗️ *PHASE 1 – SYSTEM ARCHITECTURE & ENGINEERING BLUEPRINT*

Product: *StayWise*
Goal: *Design Scalable SaaS Foundation*

1️⃣ ARCHITECTURE DECISION

We will use:

🧩 *Layered SaaS Architecture*

Client (Frontend - Next.js)

↓

API Layer (Node.js + Express)

↓

Service Layer (Business Logic)

↓

Data Access Layer

↓

PostgreSQL Database

Why?

- *Easy to scale*
- *Easy to maintain*
- *Easy to add features later*

- *Industry standard SaaS design*

2️⃣ *TENANCY MODEL DECISION (VERY IMPORTANT)*

Since this is SaaS:

👉 *We use **Single Database, Multi-Tenant Model***

Meaning:

- *One central database*
- *Each Owner has isolated data using* `owner_id`
- *Secure row-level separation*

This is how real SaaS works.

3️⃣ *USER ROLES STRUCTURE*

We define role system from day one:

Role	***Access Level***
SUPER_ADMIN	*Platform owner (You)*
OWNER	*PG Owner*

MANAGER	*Optional sub-user under Owner*
TENANT	*Bed occupant*

Each user record will have:

id

name

email

phone

password_hash

role

owner_id (nullable for super_admin)

created_at

4️⃣ CORE SYSTEM HIERARCHY

This is your core database relationship model:

Owner

→ Building

→ Floor (optional in future)

→ Room

→ Bed

→ Tenant Assignment

No confusion.

Everything controlled via owner_id.

5️⃣ DATABASE SCHEMA DESIGN (CORE TABLES)

Now I define production-level schema.

USERS TABLE

Field	***Type***
id	*UUID*
name	*VARCHAR*
email	*VARCHAR (unique)*
phone	*VARCHAR*
password_hash	*TEXT*
role	*ENUM*
owner_id	*UUID (nullable)*

created_at	*TIMESTAMP*

OWNERS TABLE

Field	***Type***
id	*UUID*
business_name	*VARCHAR*
city	*VARCHAR*
subscription_plan	*VARCHAR*
created_at	*TIMESTAMP*

BUILDINGS TABLE

Field	***Type***
id	*UUID*
owner_id	*UUID*

name	*VARCHAR*
address	*TEXT*
created_at	*TIMESTAMP*

ROOMS TABLE

Field	***Type***
id	*UUID*
building_id	*UUID*
room_number	*VARCHAR*
capacity	*INT*
created_at	*TIMESTAMP*

BEDS TABLE

Field	***Type***

id	*UUID*
room_id	*UUID*
bed_number	*VARCHAR*
status	*ENUM (available/occupied)*
created_at	*TIMESTAMP*

TENANT_ASSIGNMENTS TABLE

Field	***Type***
id	*UUID*
bed_id	*UUID*
tenant_id	*UUID (users table)*
rent_amount	*DECIMAL*
rent_due_date	*DATE*

status	*ENUM (active/inactive)*
created_at	*TIMESTAMP*

PAYMENTS TABLE

Field	***Type***
id	*UUID*
tenant_assignment_id	*UUID*
amount	*DECIMAL*
paid_on	*DATE*
status	*ENUM (paid/pending)*

COMPLAINTS TABLE

Field	***Type***
id	*UUID*

tenant_id	*UUID*
building_id	*UUID*
description	*TEXT*
status	*ENUM (open/resolved)*
created_at	*TIMESTAMP*

6️⃣ SECURITY STRUCTURE

- *JWT Authentication*
- *Owner_id based data isolation*
- *Password hashing (bcrypt)*
- *No direct DB queries from frontend*
- *API validation layer*

7️⃣ API STRUCTURE DESIGN

RESTful structure:

POST   /auth/register

POST   /auth/login

POST   /buildings

GET   /buildings

POST   /rooms

GET   /rooms/:buildingId

POST   /beds

GET   /beds/:roomId

POST   /assign-tenant

POST   /mark-payment

POST   /complaints

GET   /complaints

8️⃣ WHAT WE BUILD FIRST

We start Backend-first.

Because:

- *System stability > UI beauty*
- *Core data logic must be strong*

🧱 *Phase 1 Execution Pack (Copy → Antigravity IDE)*

1) Create Backend Project

```
mkdir staywise && cd staywise
mkdir backend && cd backend
npm init -y
npm i express cors helmet morgan jsonwebtoken bcrypt zod dotenv
npm i prisma @prisma/client
npm i -D typescript ts-node-dev @types/express @types/cors @types/morgan
@types/jsonwebtoken @types/bcrypt
npx tsc --init
npx prisma init
```

---

2) Add Docker PostgreSQL (Local Now, Scales Later)

Create `backend/docker-compose.yml`

```
services:
 db:
   image: postgres:16
   container_name: staywise_db
   restart: unless-stopped
   environment:
     POSTGRES_USER: staywise
     POSTGRES_PASSWORD: staywise_pass
     POSTGRES_DB: staywise
   ports:
     - "5432:5432"
   volumes:
     - staywise_pg:/var/lib/postgresql/data

volumes:
 staywise_pg:
```

Run:

```
docker compose up -d
```

---

3) Environment Variables

Create `backend/.env`

```
DATABASE_URL="postgresql://staywise:staywise_pass@localhost:5432/staywise?schema=public"
PORT=4000
JWT_SECRET="change_this_to_a_long_random_string"
JWT_EXPIRES_IN="7d"
```

---

4) TypeScript config (quick usable)

Edit `backend/tsconfig.json` *(replace important parts)*

```
{
 "compilerOptions": {
   "target": "ES2021",
   "module": "CommonJS",
   "rootDir": "src",
   "outDir": "dist",
   "strict": true,
   "esModuleInterop": true,
   "skipLibCheck": true
 }
}
```

---

5) Prisma Schema (Multi-tenant SaaS-ready)

Replace `backend/prisma/schema.prisma` *with:*

```
generator client {
 provider = "prisma-client-js"
}

datasource db {
 provider = "postgresql"
 url      = env("DATABASE_URL")
```

```
}

enum Role {
 SUPER_ADMIN
 OWNER
 MANAGER
 TENANT
}

enum BedStatus {
 AVAILABLE
 OCCUPIED
}

enum AssignmentStatus {
 ACTIVE
 INACTIVE
}

enum PaymentStatus {
 PAID
 PENDING
}

enum ComplaintStatus {
 OPEN
 RESOLVED
}

model Owner {
 id               String   @id @default(uuid())
 businessName     String
 city             String   @default("Bengaluru")
 subscriptionPlan String   @default("STARTER")
 createdAt        DateTime @default(now())

 users     User[]
 buildings Building[]
}

model User {
 id          String   @id @default(uuid())
 name        String
 email       String   @unique
 phone        String?
 passwordHash String
 role         Role
 ownerId      String?
 createdAt    DateTime @default(now())

 owner Owner? @relation(fields: [ownerId], references: [id])

 tenantAssignments TenantAssignment[]
 complaints        Complaint[]
}

model Building {
 id        String   @id @default(uuid())
 ownerId   String
 name      String
 address   String
 createdAt DateTime @default(now())

 owner Owner @relation(fields: [ownerId], references: [id])
 rooms Room[]
 complaints Complaint[]
}

model Room {
 id         String   @id @default(uuid())
 buildingId String
 roomNumber String
 capacity   Int
 createdAt  DateTime @default(now())

 building Building @relation(fields: [buildingId], references: [id])
 beds Bed[]
}

model Bed {
 id        String   @id @default(uuid())
 roomId    String
 bedNumber String
 status    BedStatus @default(AVAILABLE)
 createdAt DateTime  @default(now())

 room Room @relation(fields: [roomId], references: [id])
 assignments TenantAssignment[]
```

```
}

model TenantAssignment {
 id         String   @id @default(uuid())
 bedId      String
 tenantId   String
 rentAmount Decimal  @db.Decimal(10, 2)
 rentDueDay Int      @default(5) // day of month
 status     AssignmentStatus @default(ACTIVE)
 createdAt  DateTime @default(now())

 bed    Bed  @relation(fields: [bedId], references: [id])
 tenant User @relation(fields: [tenantId], references: [id])

 payments Payment[]
}

model Payment {
 id                  String   @id @default(uuid())
 tenantAssignmentId  String
 amount              Decimal  @db.Decimal(10, 2)
 paidOn              DateTime?
 status              PaymentStatus @default(PENDING)
 createdAt           DateTime @default(now())

 assignment TenantAssignment @relation(fields: [tenantAssignmentId], references: [id])
}

model Complaint {
 id          String   @id @default(uuid())
 tenantId    String
 buildingId  String
 description String
 status      ComplaintStatus @default(OPEN)
 createdAt   DateTime @default(now())

 tenant   User     @relation(fields: [tenantId], references: [id])
 building Building @relation(fields: [buildingId], references: [id])
}
```

Run migration:

```
npx prisma migrate dev --name init
```

---

6) Backend Folder Structure

Create these folders/files:

```
mkdir -p
src/{config,lib,middlewares,modules/auth,modules/buildings,modules/rooms,modules/beds,mod
ules/tenants,modules/complaints}
touch src/server.ts src/app.ts src/config/env.ts src/lib/prisma.ts
```

`src/config/env.ts`

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

export const env = {
 PORT: process.env.PORT ? Number(process.env.PORT) : 4000,
 JWT_SECRET: process.env.JWT_SECRET || "dev_secret_change_me",
 JWT_EXPIRES_IN: process.env.JWT_EXPIRES_IN || "7d",
};
```

`src/lib/prisma.ts`

```
import { PrismaClient } from "@prisma/client";

export const prisma = new PrismaClient();
```

`src/app.ts`

```
import express from "express";
import cors from "cors";
import helmet from "helmet";
import morgan from "morgan";

import { authRouter } from "./modules/auth/routes";
import { buildingRouter } from "./modules/buildings/routes";
import { roomRouter } from "./modules/rooms/routes";
import { bedRouter } from "./modules/beds/routes";
import { tenantRouter } from "./modules/tenants/routes";
import { complaintRouter } from "./modules/complaints/routes";

export const app = express();
```

```
app.use(helmet());
app.use(cors({ origin: true, credentials: true }));
app.use(express.json({ limit: "1mb" }));
app.use(morgan("dev"));

app.get("/health", (_req, res) => res.json({ ok: true, service: "staywise-api" }));

app.use("/auth", authRouter);
app.use("/buildings", buildingRouter);
app.use("/rooms", roomRouter);
app.use("/beds", bedRouter);
app.use("/tenants", tenantRouter);
app.use("/complaints", complaintRouter);
```

`src/server.ts`

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

app.listen(env.PORT, () => {
 console.log(`✅ StayWise API running on http://localhost:${env.PORT}`);
});
```

---

7) Auth + RBAC (Core SaaS Security)

Create `src/middlewares/auth.ts`

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

export type AuthUser = {
 id: string;
 role: Role;
 ownerId?: string | null;
};

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

export function requireAuth(req: Request, res: Response, next: NextFunction) {
 const header = req.headers.authorization;
 if (!header?.startsWith("Bearer ")) return res.status(401).json({ error: "Missing token" });

 try {
   const token = header.slice(7);
   const decoded = jwt.verify(token, env.JWT_SECRET) as AuthUser;
   req.user = decoded;
   return next();
 } catch {
   return res.status(401).json({ error: "Invalid token" });
 }
}

export function requireRole(...roles: Role[]) {
 return (req: Request, res: Response, next: NextFunction) => {
   if (!req.user) return res.status(401).json({ error: "Unauthorized" });
   if (!roles.includes(req.user.role)) return res.status(403).json({ error: "Forbidden" });
   return next();
 };
}
```

Auth Module

Create `src/modules/auth/routes.ts`

```
import { Router } from "express";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import { z } from "zod";
import { prisma } from "../../lib/prisma";
import { env } from "../../config/env";
import { Role } from "@prisma/client";

export const authRouter = Router();

const ownerRegisterSchema = z.object({
 businessName: z.string().min(2),
 ownerName: z.string().min(2),
 email: z.string().email(),
 password: z.string().min(6),
 phone: z.string().optional(),
 city: z.string().optional(),
});

authRouter.post("/register-owner", async (req, res) => {
 const parsed = ownerRegisterSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

 const { businessName, ownerName, email, password, phone, city } = parsed.data;

 const existing = await prisma.user.findUnique({ where: { email } });
 if (existing) return res.status(409).json({ error: "Email already in use" });

 const owner = await prisma.owner.create({
   data: { businessName, city: city ?? "Bengaluru" },
 });

 const passwordHash = await bcrypt.hash(password, 10);

 const user = await prisma.user.create({
   data: {
     name: ownerName,
     email,
     phone,
     passwordHash,
     role: Role.OWNER,
     ownerId: owner.id,
   },
 });

 return res.json({ ok: true, ownerId: owner.id, userId: user.id });
});

const loginSchema = z.object({
 email: z.string().email(),
 password: z.string().min(1),
});

authRouter.post("/login", async (req, res) => {
 const parsed = loginSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });
 const { email, password } = parsed.data;

 const user = await prisma.user.findUnique({ where: { email } });
 if (!user) return res.status(401).json({ error: "Invalid credentials" });

 const ok = await bcrypt.compare(password, user.passwordHash);
 if (!ok) return res.status(401).json({ error: "Invalid credentials" });

 const token = jwt.sign(
   { id: user.id, role: user.role, ownerId: user.ownerId ?? null },
   env.JWT_SECRET,
   { expiresIn: env.JWT_EXPIRES_IN }
 );

 return res.json({
   ok: true,
   token,
   user: { id: user.id, name: user.name, role: user.role, ownerId: user.ownerId },
 });
});
```

---

8) Core CRUD APIs (Owner scope enforced)

Buildings

Create `src/modules/buildings/routes.ts`

```
import { Router } from "express";
import { z } from "zod";
import { prisma } from "../../lib/prisma";
import { requireAuth, requireRole } from "../../middlewares/auth";
import { Role } from "@prisma/client";

export const buildingRouter = Router();

const createSchema = z.object({
 name: z.string().min(2),
 address: z.string().min(5),
});

buildingRouter.post("/", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async (req,
res) => {
 const parsed = createSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

 const ownerId = req.user!.ownerId;
 if (!ownerId) return res.status(400).json({ error: "Owner scope missing" });

 const building = await prisma.building.create({
   data: { ownerId, ...parsed.data },
 });

 res.json({ ok: true, building });
});

buildingRouter.get("/", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async (req,
res) => {
 const ownerId = req.user!.ownerId;
 const buildings = await prisma.building.findMany({
   where: { ownerId: ownerId ?? undefined },
   orderBy: { createdAt: "desc" },
 });
 res.json({ ok: true, buildings });
});
```

Rooms

Create `src/modules/rooms/routes.ts`

```
import { Router } from "express";
import { z } from "zod";
import { prisma } from "../../lib/prisma";
import { requireAuth, requireRole } from "../../middlewares/auth";
import { Role } from "@prisma/client";

export const roomRouter = Router();

const createSchema = z.object({
 buildingId: z.string().uuid(),
 roomNumber: z.string().min(1),
 capacity: z.number().int().positive(),
});
```

```
roomRouter.post("/", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async (req, res)
=> {
 const parsed = createSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

 const ownerId = req.user!.ownerId!;
 const b = await prisma.building.findFirst({ where: { id: parsed.data.buildingId, ownerId } });
 if (!b) return res.status(404).json({ error: "Building not found" });

 const room = await prisma.room.create({ data: parsed.data });
 res.json({ ok: true, room });
});

roomRouter.get("/:buildingId", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async
(req, res) => {
 const ownerId = req.user!.ownerId!;
 const buildingId = req.params.buildingId;

 const b = await prisma.building.findFirst({ where: { id: buildingId, ownerId } });
 if (!b) return res.status(404).json({ error: "Building not found" });

 const rooms = await prisma.room.findMany({ where: { buildingId }, orderBy: { createdAt: "desc"
} });
 res.json({ ok: true, rooms });
});
```

Beds

Create `src/modules/beds/routes.ts`

```
import { Router } from "express";
import { z } from "zod";
import { prisma } from "../../lib/prisma";
import { requireAuth, requireRole } from "../../middlewares/auth";
import { Role, BedStatus } from "@prisma/client";

export const bedRouter = Router();

const createSchema = z.object({
 roomId: z.string().uuid(),
 bedNumber: z.string().min(1),
});
```

```
bedRouter.post("/", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async (req, res) => {
 const parsed = createSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

 const ownerId = req.user!.ownerId!;

 const room = await prisma.room.findUnique({ where: { id: parsed.data.roomId }, include: { building: true } });
 if (!room || room.building.ownerId !== ownerId) return res.status(404).json({ error: "Room not found" });

 const bed = await prisma.bed.create({ data: { ...parsed.data, status: BedStatus.AVAILABLE } });
 res.json({ ok: true, bed });
});

bedRouter.get("/:roomId", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async (req, res) => {
 const ownerId = req.user!.ownerId!;
 const roomId = req.params.roomId;

 const room = await prisma.room.findUnique({ where: { id: roomId }, include: { building: true } });
 if (!room || room.building.ownerId !== ownerId) return res.status(404).json({ error: "Room not found" });

 const beds = await prisma.bed.findMany({ where: { roomId }, orderBy: { createdAt: "desc" } });
 res.json({ ok: true, beds });
});
```

---

9) Tenant Assignment (Occupancy Engine)

Create `src/modules/tenants/routes.ts`

```
import { Router } from "express";
import { z } from "zod";
import bcrypt from "bcrypt";
import { prisma } from "../../lib/prisma";
import { requireAuth, requireRole } from "../../middlewares/auth";
import { Role, BedStatus, AssignmentStatus } from "@prisma/client";
```

```
export const tenantRouter = Router();

const createTenantSchema = z.object({
 name: z.string().min(2),
 email: z.string().email(),
 password: z.string().min(6),
 phone: z.string().optional(),
});

tenantRouter.post("/create", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async
(req, res) => {
 const ownerId = req.user!.ownerId!;
 const parsed = createTenantSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

 const existing = await prisma.user.findUnique({ where: { email: parsed.data.email } });
 if (existing) return res.status(409).json({ error: "Email already exists" });

 const passwordHash = await bcrypt.hash(parsed.data.password, 10);

 const tenant = await prisma.user.create({
   data: {
     name: parsed.data.name,
     email: parsed.data.email,
     phone: parsed.data.phone,
     passwordHash,
     role: Role.TENANT,
     ownerId,
   },
 });

 res.json({ ok: true, tenant: { id: tenant.id, name: tenant.name, email: tenant.email } });
});

const assignSchema = z.object({
 bedId: z.string().uuid(),
 tenantId: z.string().uuid(),
 rentAmount: z.number().positive(),
 rentDueDay: z.number().int().min(1).max(28).optional(),
});

tenantRouter.post("/assign", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async
(req, res) => {
 const ownerId = req.user!.ownerId!;
 const parsed = assignSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

 const bed = await prisma.bed.findUnique({
   where: { id: parsed.data.bedId },
   include: { room: { include: { building: true } } },
 });
 if (!bed || bed.room.building.ownerId !== ownerId) return res.status(404).json({ error: "Bed not
found" });
 if (bed.status === BedStatus.OCCUPIED) return res.status(400).json({ error: "Bed already
occupied" });

 const tenant = await prisma.user.findUnique({ where: { id: parsed.data.tenantId } });
 if (!tenant || tenant.ownerId !== ownerId || tenant.role !== Role.TENANT)
   return res.status(404).json({ error: "Tenant not found" });

 // deactivate old assignments if any
 await prisma.tenantAssignment.updateMany({
   where: { bedId: bed.id, status: AssignmentStatus.ACTIVE },
   data: { status: AssignmentStatus.INACTIVE },
 });

 const assignment = await prisma.tenantAssignment.create({
   data: {
     bedId: bed.id,
     tenantId: tenant.id,
     rentAmount: parsed.data.rentAmount,
     rentDueDay: parsed.data.rentDueDay ?? 5,
     status: AssignmentStatus.ACTIVE,
   },
 });

 await prisma.bed.update({ where: { id: bed.id }, data: { status: BedStatus.OCCUPIED } });

 res.json({ ok: true, assignment });
});
```

---

10) Complaints

Create `src/modules/complaints/routes.ts`

```
import { Router } from "express";
import { z } from "zod";
import { prisma } from "../../lib/prisma";
import { requireAuth, requireRole } from "../../middlewares/auth";
import { Role, ComplaintStatus } from "@prisma/client";

export const complaintRouter = Router();

const createSchema = z.object({
 buildingId: z.string().uuid(),
 description: z.string().min(5),
});

complaintRouter.post("/", requireAuth, requireRole(Role.TENANT), async (req, res) => {
 const parsed = createSchema.safeParse(req.body);
 if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

 const user = req.user!;
 const building = await prisma.building.findFirst({
   where: { id: parsed.data.buildingId, ownerId: user.ownerId ?? undefined },
 });
 if (!building) return res.status(404).json({ error: "Building not found" });

 const complaint = await prisma.complaint.create({
   data: {
     tenantId: user.id,
     buildingId: building.id,
     description: parsed.data.description,
     status: ComplaintStatus.OPEN,
   },
 });

 res.json({ ok: true, complaint });
});

complaintRouter.get("/", requireAuth, requireRole(Role.OWNER, Role.MANAGER), async (req,
res) => {
 const ownerId = req.user!.ownerId!;
 const complaints = await prisma.complaint.findMany({
   where: { building: { ownerId } },
   include: { tenant: { select: { id: true, name: true, email: true } }, building: { select: { id: true,
name: true } } },
   orderBy: { createdAt: "desc" },
 });
 res.json({ ok: true, complaints });
```

});

---

11) Wire Routes Exports

Create these small files:

`src/modules/auth/routes.ts` ***is already done*** ✅

`src/modules/buildings/routes.ts` ✅

`src/modules/rooms/routes.ts` ✅

`src/modules/beds/routes.ts` ✅

`src/modules/tenants/routes.ts` ✅

`src/modules/complaints/routes.ts` ✅

(You already pasted them above.)

---

12) Add Scripts to `backend/package.json`

```
{
 "scripts": {
   "dev": "ts-node-dev --respawn --transpile-only src/server.ts",
   "prisma:studio": "npx prisma studio"
 }
}
```

Run server:

```
npm run dev
```

Test:

- *`GET http://localhost:4000/health`*

1) Create Frontend App (Next.js)

From your `staywise/` folder (same level as `backend/`):

```
mkdir frontend && cd frontend
npx create-next-app@latest .
```

Choose these options when prompted:

- *TypeScript:* ***Yes***
- *ESLint:* ***Yes***
- *Tailwind:* ***Yes***
- `src/` *directory:* ***Yes***
- *App Router:* ***Yes***
- *Import alias:* ***Yes*** *(keep default `@/*`)*

---

2) Add Environment Config

Create `frontend/.env.local`

```
NEXT_PUBLIC_API_URL=http://localhost:4000
```

---

3) Add a Simple API Client + Auth Store

`frontend/src/lib/api.ts`

```
const API_URL = process.env.NEXT_PUBLIC_API_URL || "http://localhost:4000";

export function getToken() {
 if (typeof window === "undefined") return null;
 return localStorage.getItem("staywise_token");
```

```
}

export function setToken(token: string) {
 localStorage.setItem("staywise_token", token);
}

export function clearToken() {
 localStorage.removeItem("staywise_token");
 localStorage.removeItem("staywise_user");
}

export function setUser(user: any) {
 localStorage.setItem("staywise_user", JSON.stringify(user));
}

export function getUser() {
 if (typeof window === "undefined") return null;
 const raw = localStorage.getItem("staywise_user");
 return raw ? JSON.parse(raw) : null;
}

export async function apiFetch<T>(
 path: string,
 options: RequestInit = {}
): Promise<T> {
 const token = getToken();

 const res = await fetch(`${API_URL}${path}`, {
   ...options,
   headers: {
     "Content-Type": "application/json",
     ...(token ? { Authorization: `Bearer ${token}` } : {}),
     ...(options.headers || {}),
   },
   cache: "no-store",
 });

 const data = await res.json().catch(() => ({}));

 if (!res.ok) {
   const msg =
     data?.error?.message ||
     data?.error ||
     data?.message ||
    `Request failed (${res.status})`;
  throw new Error(typeof msg === "string" ? msg : "Request failed");
 }

 return data as T;
}
```

---

4) Auth Provider + Route Guard

frontend/src/components/AuthProvider.tsx

```
"use client";

import { createContext, useContext, useEffect, useMemo, useState } from "react";
import { clearToken, getToken, getUser, setToken, setUser } from "@/lib/api";

type AuthState = {
 token: string | null;
 user: any | null;
 login: (token: string, user: any) => void;
 logout: () => void;
};

const AuthContext = createContext<AuthState | null>(null);

export function AuthProvider({ children }: { children: React.ReactNode }) {
 const [token, setTok] = useState<string | null>(null);
 const [user, setUsr] = useState<any | null>(null);

 useEffect(() => {
   setTok(getToken());
   setUsr(getUser());
 }, []);

 const value = useMemo<AuthState>(
   () => ({
     token,
     user,
     login: (t, u) => {
       setToken(t);
       setUser(u);
```

```
      setTok(t);
      setUsr(u);
    },
    logout: () => {
      clearToken();
      setTok(null);
      setUsr(null);
    },
  }),
  [token, user]
 );

 return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
}

export function useAuth() {
 const ctx = useContext(AuthContext);
 if (!ctx) throw new Error("useAuth must be used inside AuthProvider");
 return ctx;
}
```

frontend/src/components/Protected.tsx

```
"use client";

import { useEffect } from "react";
import { useRouter } from "next/navigation";
import { useAuth } from "./AuthProvider";

export default function Protected({
 children,
 allowRoles,
}: {
 children: React.ReactNode;
 allowRoles?: string[];
}) {
 const router = useRouter();
 const { token, user } = useAuth();

 useEffect(() => {
  if (!token) router.replace("/login");
 }, [token, router]);

 useEffect(() => {
   if (!allowRoles || !user?.role) return;
   if (!allowRoles.includes(user.role)) router.replace("/login");
 }, [allowRoles, user, router]);

 if (!token) return null;
 if (allowRoles && user?.role && !allowRoles.includes(user.role)) return null;

 return <>{children}</>;
}
```

---

5) Wire AuthProvider into Layout

Edit: `frontend/src/app/layout.tsx`

```
import "./globals.css";
import { AuthProvider } from "@/components/AuthProvider";

export default function RootLayout({
 children,
}: {
 children: React.ReactNode;
}) {
 return (
   <html lang="en">
     <body>
       <AuthProvider>{children}</AuthProvider>
     </body>
   </html>
 );
}
```

---

6) Build Pages

A) Login Page

Create: `frontend/src/app/login/page.tsx`

```
"use client";
```

```
import { useState } from "react";
import { useRouter } from "next/navigation";
import { apiFetch } from "@/lib/api";
import { useAuth } from "@/components/AuthProvider";

export default function LoginPage() {
 const router = useRouter();
 const { login } = useAuth();
 const [email, setEmail] = useState("");
 const [password, setPassword] = useState("");
 const [err, setErr] = useState<string | null>(null);
 const [loading, setLoading] = useState(false);

 async function onSubmit(e: React.FormEvent) {
   e.preventDefault();
   setErr(null);
   setLoading(true);

   try {
     const res = await apiFetch<any>("/auth/login", {
       method: "POST",
       body: JSON.stringify({ email, password }),
     });

     login(res.token, res.user);

     // Owner/Manager → owner dashboard, Tenant → complaints later
     if (res.user.role === "TENANT") router.push("/tenant");
     else router.push("/owner");
   } catch (e: any) {
     setErr(e.message || "Login failed");
   } finally {
     setLoading(false);
   }
 }

 return (
   <div className="min-h-screen flex items-center justify-center p-6">
     <div className="w-full max-w-md border rounded-xl p-6">
       <h1 className="text-2xl font-semibold">StayWise Login</h1>
       <p className="text-sm text-gray-500 mt-1">
         Use your Owner / Manager / Tenant credentials
       </p>
```

```
    <form onSubmit={onSubmit} className="mt-6 space-y-3">
     <input
      className="w-full border rounded-lg p-3"
      placeholder="Email"
      value={email}
      onChange={(e) => setEmail(e.target.value)}
     />
     <input
      className="w-full border rounded-lg p-3"
      placeholder="Password"
      type="password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
     />

     {err && <div className="text-sm text-red-600">{err}</div>}

     <button
      className="w-full rounded-lg p-3 border font-medium hover:bg-gray-50
disabled:opacity-60"
      disabled={loading}
     >
      {loading ? "Logging in..." : "Login"}
     </button>
    </form>

    <div className="mt-5 text-xs text-gray-500">
     Backend should be running on{" "}
     <span className="font-mono">http://localhost:4000</span>
    </div>
   </div>
  </div>
 );
}
```

B) Owner Dashboard (Buildings)

Create: `frontend/src/app/owner/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { useAuth } from "@/components/AuthProvider";
import { apiFetch } from "@/lib/api";
import Link from "next/link";
import { useEffect, useState } from "react";

type Building = { id: string; name: string; address: string };

export default function OwnerDashboard() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <OwnerDashboardInner />
   </Protected>
 );
}

function OwnerDashboardInner() {
 const { user, logout } = useAuth();
 const [buildings, setBuildings] = useState<Building[]>([]);
 const [name, setName] = useState("");
 const [address, setAddress] = useState("");
 const [err, setErr] = useState<string | null>(null);

 async function load() {
   const res = await apiFetch<{ ok: boolean; buildings: Building[] }>(
     "/buildings"
   );
   setBuildings(res.buildings);
 }

 useEffect(() => {
   load().catch((e) => setErr(e.message));
 }, []);

 async function addBuilding() {
   setErr(null);
   try {
     await apiFetch("/buildings", {
       method: "POST",
       body: JSON.stringify({ name, address }),
     });
     setName("");
     setAddress("");
     await load();
   } catch (e: any) {
     setErr(e.message);
   }
 }

 return (
   <div className="p-6 max-w-5xl mx-auto">
     <div className="flex items-start justify-between gap-4">
       <div>
         <h1 className="text-2xl font-semibold">Owner Dashboard</h1>
         <p className="text-sm text-gray-600">
           Logged in as {user?.name} ({user?.role})
         </p>
       </div>

       <button
         className="border rounded-lg px-4 py-2 hover:bg-gray-50"
         onClick={logout}
       >
         Logout
       </button>
     </div>

     <div className="mt-6 border rounded-xl p-4">
       <h2 className="font-semibold">Add Building</h2>
       <div className="grid grid-cols-1 md:grid-cols-3 gap-3 mt-3">
         <input
           className="border rounded-lg p-3"
           placeholder="Building name"
           value={name}
           onChange={(e) => setName(e.target.value)}
         />
         <input
           className="border rounded-lg p-3 md:col-span-2"
           placeholder="Address"
           value={address}
           onChange={(e) => setAddress(e.target.value)}
         />
       </div>
       <button
         className="mt-3 border rounded-lg px-4 py-2 hover:bg-gray-50"
         onClick={addBuilding}
       >
         Create
       </button>
       {err && <div className="text-sm text-red-600 mt-2">{err}</div>}
      </div>

      <div className="mt-6">
       <h2 className="font-semibold">Buildings</h2>
       <div className="mt-3 grid grid-cols-1 md:grid-cols-2 gap-3">
        {buildings.map((b) => (
         <Link
          key={b.id}
          className="border rounded-xl p-4 hover:bg-gray-50"
          href={`/owner/buildings/${b.id}`}
         >
          <div className="font-semibold">{b.name}</div>
          <div className="text-sm text-gray-600 mt-1">{b.address}</div>
         </Link>
        ))}
       </div>
      </div>
     </div>
  );
 }
```

C) Building → Rooms Page

Create: `frontend/src/app/owner/buildings/[id]/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import Link from "next/link";
import { useParams } from "next/navigation";
import { useEffect, useState } from "react";

type Room = { id: string; roomNumber: string; capacity: number };

export default function BuildingRoomsPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}
```

```
function Inner() {
 const params = useParams();
 const buildingId = params.id as string;

 const [rooms, setRooms] = useState<Room[]>([]);
 const [roomNumber, setRoomNumber] = useState("");
 const [capacity, setCapacity] = useState(4);
 const [err, setErr] = useState<string | null>(null);

 async function load() {
   const res = await apiFetch<{ ok: boolean; rooms: Room[] }>(
     `/rooms/${buildingId}`
   );
   setRooms(res.rooms);
 }

 useEffect(() => {
   load().catch((e) => setErr(e.message));
 }, []);

 async function addRoom() {
   setErr(null);
   try {
     await apiFetch("/rooms", {
       method: "POST",
       body: JSON.stringify({ buildingId, roomNumber, capacity }),
     });
     setRoomNumber("");
     setCapacity(4);
     await load();
   } catch (e: any) {
     setErr(e.message);
   }
 }

 return (
   <div className="p-6 max-w-5xl mx-auto">
     <Link href="/owner" className="text-sm underline">
       ← Back to Dashboard
     </Link>

     <h1 className="text-2xl font-semibold mt-3">Rooms</h1>
     <p className="text-sm text-gray-600">Building ID: {buildingId}</p>
```

```
      <div className="mt-6 border rounded-xl p-4">
        <h2 className="font-semibold">Add Room</h2>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-3 mt-3">
          <input
            className="border rounded-lg p-3"
            placeholder="Room number (ex: 101)"
            value={roomNumber}
            onChange={(e) => setRoomNumber(e.target.value)}
          />
          <input
            className="border rounded-lg p-3"
            type="number"
            min={1}
            value={capacity}
            onChange={(e) => setCapacity(Number(e.target.value))}
          />
          <button
            className="border rounded-lg px-4 py-2 hover:bg-gray-50"
            onClick={addRoom}
          >
            Create
          </button>
        </div>
        {err && <div className="text-sm text-red-600 mt-2">{err}</div>}
      </div>

      <div className="mt-6 grid grid-cols-1 md:grid-cols-2 gap-3">
        {rooms.map((r) => (
          <Link
            key={r.id}
            className="border rounded-xl p-4 hover:bg-gray-50"
            href={`/owner/rooms/${r.id}`}
          >
            <div className="font-semibold">Room {r.roomNumber}</div>
            <div className="text-sm text-gray-600 mt-1">
              Capacity: {r.capacity}
            </div>
          </Link>
        ))}
      </div>
    </div>
  );
}
```

D) Room → Beds + Create Tenant + Assign Tenant

Create: `frontend/src/app/owner/rooms/[id]/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import Link from "next/link";
import { useParams } from "next/navigation";
import { useEffect, useState } from "react";

type Bed = { id: string; bedNumber: string; status: "AVAILABLE" | "OCCUPIED" };
type Tenant = { id: string; name: string; email: string };

export default function RoomBedsPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const params = useParams();
 const roomId = params.id as string;

 const [beds, setBeds] = useState<Bed[]>([]);
 const [bedNumber, setBedNumber] = useState("");

 const [tenantName, setTenantName] = useState("");
 const [tenantEmail, setTenantEmail] = useState("");
 const [tenantPass, setTenantPass] = useState("");
 const [tenantPhone, setTenantPhone] = useState("");

 const [selectedBedId, setSelectedBedId] = useState("");
 const [selectedTenantId, setSelectedTenantId] = useState("");
 const [rentAmount, setRentAmount] = useState(8000);
 const [rentDueDay, setRentDueDay] = useState(5);

 const [lastCreatedTenant, setLastCreatedTenant] = useState<Tenant | null>(
   null
```

```
);
const [err, setErr] = useState<string | null>(null);

async function loadBeds() {
  const res = await apiFetch<{ ok: boolean; beds: Bed[] }>(`/beds/${roomId}`);
  setBeds(res.beds);
}

useEffect(() => {
  loadBeds().catch((e) => setErr(e.message));
}, []);

async function addBed() {
  setErr(null);
  try {
    await apiFetch("/beds", {
      method: "POST",
      body: JSON.stringify({ roomId, bedNumber }),
    });
    setBedNumber("");
    await loadBeds();
  } catch (e: any) {
    setErr(e.message);
  }
}

async function createTenant() {
  setErr(null);
  try {
    const res = await apiFetch<any>("/tenants/create", {
      method: "POST",
      body: JSON.stringify({
        name: tenantName,
        email: tenantEmail,
        password: tenantPass,
        phone: tenantPhone || undefined,
      }),
    });
    setLastCreatedTenant(res.tenant);
    setSelectedTenantId(res.tenant.id);
    setTenantName("");
    setTenantEmail("");
    setTenantPass("");
    setTenantPhone("");
  } catch (e: any) {
    setErr(e.message);
  }
}

async function assignTenant() {
  setErr(null);
  try {
    await apiFetch("/tenants/assign", {
      method: "POST",
      body: JSON.stringify({
        bedId: selectedBedId,
        tenantId: selectedTenantId,
        rentAmount,
        rentDueDay,
      }),
    });
    await loadBeds();
  } catch (e: any) {
    setErr(e.message);
  }
}

return (
  <div className="p-6 max-w-5xl mx-auto">
    <Link href="/owner" className="text-sm underline">
      ← Back to Dashboard
    </Link>

    <h1 className="text-2xl font-semibold mt-3">Beds</h1>
    <p className="text-sm text-gray-600">Room ID: {roomId}</p>

    {err && <div className="text-sm text-red-600 mt-3">{err}</div>}

    <div className="mt-6 border rounded-xl p-4">
      <h2 className="font-semibold">Add Bed</h2>
      <div className="flex gap-3 mt-3">
        <input
          className="border rounded-lg p-3 flex-1"
          placeholder="Bed number (ex: A1)"
          value={bedNumber}
          onChange={(e) => setBedNumber(e.target.value)}
        />
        <button
          className="border rounded-lg px-4 py-2 hover:bg-gray-50"
          onClick={addBed}
        >
          Create
        </button>
      </div>
    </div>

    <div className="mt-6 border rounded-xl p-4">
      <h2 className="font-semibold">Beds List</h2>
      <div className="mt-3 grid grid-cols-1 md:grid-cols-2 gap-3">
        {beds.map((b) => (
          <div
            key={b.id}
            className="border rounded-xl p-4 flex items-center justify-between"
          >
            <div>
              <div className="font-semibold">Bed {b.bedNumber}</div>
              <div className="text-sm text-gray-600">{b.status}</div>
            </div>
            <button
              className="border rounded-lg px-3 py-2 hover:bg-gray-50"
              onClick={() => setSelectedBedId(b.id)}
            >
              Select
            </button>
          </div>
        ))}
      </div>

      <div className="mt-4 text-sm text-gray-700">
        Selected Bed ID:{" "}
        <span className="font-mono">{selectedBedId || "none"}</span>
      </div>
    </div>

    <div className="mt-6 border rounded-xl p-4">
      <h2 className="font-semibold">Create Tenant</h2>
      <div className="grid grid-cols-1 md:grid-cols-2 gap-3 mt-3">
        <input
          className="border rounded-lg p-3"
          placeholder="Tenant name"
          value={tenantName}
          onChange={(e) => setTenantName(e.target.value)}
        />
        <input
          className="border rounded-lg p-3"
          placeholder="Tenant email"
          value={tenantEmail}
          onChange={(e) => setTenantEmail(e.target.value)}
        />
        <input
          className="border rounded-lg p-3"
          placeholder="Tenant password"
          type="password"
          value={tenantPass}
          onChange={(e) => setTenantPass(e.target.value)}
        />
        <input
          className="border rounded-lg p-3"
          placeholder="Tenant phone (optional)"
          value={tenantPhone}
          onChange={(e) => setTenantPhone(e.target.value)}
        />
      </div>
      <button
        className="mt-3 border rounded-lg px-4 py-2 hover:bg-gray-50"
        onClick={createTenant}
      >
        Create Tenant
      </button>

      {lastCreatedTenant && (
        <div className="mt-3 text-sm text-gray-700">
          Created: <b>{lastCreatedTenant.name}</b> ({lastCreatedTenant.email})
          <div className="mt-1">
            Tenant ID: <span className="font-mono">{lastCreatedTenant.id}</span>
          </div>
        </div>
      )}
    </div>

    <div className="mt-6 border rounded-xl p-4">
      <h2 className="font-semibold">Assign Tenant to Bed</h2>

      <div className="grid grid-cols-1 md:grid-cols-2 gap-3 mt-3">
        <input
          className="border rounded-lg p-3"
```

```
          placeholder="Tenant ID"
          value={selectedTenantId}
          onChange={(e) => setSelectedTenantId(e.target.value)}
        />
        <input
          className="border rounded-lg p-3"
          placeholder="Rent amount"
          type="number"
          value={rentAmount}
          onChange={(e) => setRentAmount(Number(e.target.value))}
        />
        <input
          className="border rounded-lg p-3"
          placeholder="Rent due day (1-28)"
          type="number"
          min={1}
          max={28}
          value={rentDueDay}
          onChange={(e) => setRentDueDay(Number(e.target.value))}
        />
        <div className="border rounded-lg p-3 text-sm text-gray-700">
          Selected Bed:{" "}
          <span className="font-mono">{selectedBedId || "none"}</span>
        </div>
      </div>

      <button
        className="mt-3 border rounded-lg px-4 py-2 hover:bg-gray-50 disabled:opacity-60"
        onClick={assignTenant}
        disabled={!selectedBedId || !selectedTenantId}
      >
        Assign
      </button>
    </div>
  </div>
 );
}
```

E) Tenant Landing Page (minimal placeholder)

Create: `frontend/src/app/tenant/page.tsx`

```
"use client";
```

```
import Protected from "@/components/Protected";
import { useAuth } from "@/components/AuthProvider";

export default function TenantPage() {
 return (
   <Protected allowRoles={["TENANT"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const { user, logout } = useAuth();
 return (
   <div className="p-6 max-w-3xl mx-auto">
     <div className="flex justify-between items-start">
       <div>
         <h1 className="text-2xl font-semibold">Tenant Panel</h1>
         <p className="text-sm text-gray-600">
           Logged in as {user?.name} ({user?.role})
         </p>
         <p className="text-sm text-gray-600 mt-2">
           Next: Complaint creation UI + rent status UI.
         </p>
       </div>
       <button
         className="border rounded-lg px-4 py-2 hover:bg-gray-50"
         onClick={logout}
       >
         Logout
       </button>
     </div>
   </div>
 );
}
```

---

7) Run Frontend

```
npm run dev
```

Open:

- *Frontend:* `http://localhost:3000/login`
- *Backend:* `http://localhost:4000/health`

Quick Test Flow (Works End-to-End)

1. ***Register Owner (API)*** *using Postman/Thunder Client:*
   - `POST http://localhost:4000/auth/register-owner`
   - *body:*

     ```
      {
       "businessName": "StayWise Demo PG",
       "ownerName": "Tamil",
       "email": "owner@staywise.com",
       "password": "12345678",
       "city": "Bengaluru"
     }
     ```
2. *Login in UI with* `owner@staywise.com / 12345678`
3. *Add building → add room → add bed*
4. *Create tenant → select bed → assign tenant*
5. *Bed status becomes* ***OCCUPIED***

0) Backend small upgrade (Resolve complaints)

Update `backend/src/modules/complaints/routes.ts`

Add this at the bottom:

```
import { ComplaintStatus } from "@prisma/client";

// ...existing routes above

complaintRouter.patch(
 "/:id/resolve",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const id = req.params.id;

   const complaint = await prisma.complaint.findUnique({
     where: { id },
     include: { building: true },
   });

   if (!complaint || complaint.building.ownerId !== ownerId) {
     return res.status(404).json({ error: "Complaint not found" });
   }

   const updated = await prisma.complaint.update({
     where: { id },
     data: { status: ComplaintStatus.RESOLVED },
   });

   res.json({ ok: true, complaint: updated });
 }
);
```

Restart backend:

```
npm run dev
```

---

1) Frontend: Add a clean Top Navigation

Create: `frontend/src/components/TopNav.tsx`

```
"use client";

import Link from "next/link";
import { usePathname } from "next/navigation";
import { useAuth } from "./AuthProvider";

function NavLink({ href, label }: { href: string; label: string }) {
 const path = usePathname();
 const active = path === href || path.startsWith(href + "/");

 return (
   <Link
     href={href}
     className={`text-sm px-3 py-2 rounded-lg border hover:bg-gray-50 ${
       active ? "bg-gray-50" : ""
     }`}
   >
     {label}
   </Link>
 );
}

export default function TopNav() {
 const { user, logout } = useAuth();

 return (
   <div className="border-b">
     <div className="max-w-6xl mx-auto p-4 flex items-center justify-between gap-4">
       <Link href="/" className="font-semibold">
         StayWise
       </Link>

       <div className="flex items-center gap-2 flex-wrap">
         {user?.role === "OWNER" || user?.role === "MANAGER" ? (
           <>
             <NavLink href="/owner" label="Dashboard" />
             <NavLink href="/owner/occupancy" label="Occupancy" />
             <NavLink href="/owner/complaints" label="Complaints" />
           </>
```

```
        ) : null}

        {user?.role === "TENANT" ? (
          <>
            <NavLink href="/tenant" label="Home" />
            <NavLink href="/tenant/complaints" label="Raise Complaint" />
          </>
        ) : null}
      </div>

      <div className="flex items-center gap-3">
        {user?.name ? (
          <div className="text-sm text-gray-600">
            {user.name} ({user.role})
          </div>
        ) : null}
        {user ? (
          <button
            className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
            onClick={logout}
          >
            Logout
          </button>
        ) : (
          <Link
            className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
            href="/login"
          >
            Login
          </Link>
        )}
      </div>
    </div>
  </div>
 );
}
```

Now wire it into `frontend/src/app/layout.tsx`:

```
import "./globals.css";
import { AuthProvider } from "@/components/AuthProvider";
import TopNav from "@/components/TopNav";

export default function RootLayout({
 children,
}: {
 children: React.ReactNode;
}) {
 return (
   <html lang="en">
     <body>
       <AuthProvider>
         <TopNav />
         {children}
       </AuthProvider>
     </body>
   </html>
 );
}
```

---

2) Owner Occupancy Dashboard

Create: `frontend/src/app/owner/occupancy/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import Link from "next/link";
import { useEffect, useMemo, useState } from "react";

type Building = { id: string; name: string; address: string };
type Room = { id: string; roomNumber: string; capacity: number };
type Bed = { id: string; bedNumber: string; status: "AVAILABLE" | "OCCUPIED" };

export default function OccupancyPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}

function StatCard({ label, value }: { label: string; value: number }) {
 return (
   <div className="border rounded-xl p-4">
```

```
    <div className="text-sm text-gray-600">{label}</div>
    <div className="text-2xl font-semibold mt-1">{value}</div>
   </div>
 );
}

function Inner() {
 const [buildings, setBuildings] = useState<Building[]>([]);
 const [roomsByBuilding, setRoomsByBuilding] = useState<Record<string, Room[]>>(
   {}
 );
 const [bedsByRoom, setBedsByRoom] = useState<Record<string, Bed[]>>({});
 const [err, setErr] = useState<string | null>(null);
 const [loading, setLoading] = useState(true);

 async function loadAll() {
   setErr(null);
   setLoading(true);

   // 1) buildings
   const bRes = await apiFetch<{ buildings: Building[] }>("/buildings");
   setBuildings(bRes.buildings);

   // 2) rooms per building
   const roomMap: Record<string, Room[]> = {};
   for (const b of bRes.buildings) {
     const rRes = await apiFetch<{ rooms: Room[] }>(`/rooms/${b.id}`);
     roomMap[b.id] = rRes.rooms;
   }
   setRoomsByBuilding(roomMap);

   // 3) beds per room
   const bedMap: Record<string, Bed[]> = {};
   for (const b of bRes.buildings) {
     for (const r of roomMap[b.id] || []) {
       const bedRes = await apiFetch<{ beds: Bed[] }>(`/beds/${r.id}`);
       bedMap[r.id] = bedRes.beds;
     }
   }
   setBedsByRoom(bedMap);

   setLoading(false);
 }
```

```
useEffect(() => {
  loadAll().catch((e) => {
    setErr(e.message);
    setLoading(false);
  });
}, []);

const totals = useMemo(() => {
  let totalBeds = 0;
  let occupied = 0;

  for (const roomId of Object.keys(bedsByRoom)) {
    const beds = bedsByRoom[roomId] || [];
    totalBeds += beds.length;
    occupied += beds.filter((b) => b.status === "OCCUPIED").length;
  }

  return {
    totalBeds,
    occupied,
    available: totalBeds - occupied,
  };
}, [bedsByRoom]);

if (loading) {
  return (
    <div className="max-w-6xl mx-auto p-6">
      <div className="text-sm text-gray-600">Loading occupancy…</div>
    </div>
  );
}

return (
  <div className="max-w-6xl mx-auto p-6">
    <h1 className="text-2xl font-semibold">Occupancy Dashboard</h1>
    <p className="text-sm text-gray-600 mt-1">
      Live status across all buildings
    </p>

    {err && <div className="text-sm text-red-600 mt-3">{err}</div>}

    <div className="mt-6 grid grid-cols-1 md:grid-cols-3 gap-3">
      <StatCard label="Total Beds" value={totals.totalBeds} />
      <StatCard label="Occupied" value={totals.occupied} />
  <StatCard label="Available" value={totals.available} />
</div>

<div className="mt-8 space-y-4">
  {buildings.map((b) => {
    const rooms = roomsByBuilding[b.id] || [];
    let buildingTotal = 0;
    let buildingOccupied = 0;

    for (const r of rooms) {
      const beds = bedsByRoom[r.id] || [];
      buildingTotal += beds.length;
      buildingOccupied += beds.filter((x) => x.status === "OCCUPIED").length;
    }

    return (
      <div key={b.id} className="border rounded-xl p-4">
        <div className="flex items-start justify-between gap-4">
          <div>
            <div className="font-semibold">{b.name}</div>
            <div className="text-sm text-gray-600 mt-1">{b.address}</div>
          </div>
          <div className="text-sm text-gray-700">
            <span className="font-semibold">{buildingOccupied}</span> /{" "}
            {buildingTotal} occupied
          </div>
        </div>

        <div className="mt-3 grid grid-cols-1 md:grid-cols-2 gap-3">
          {rooms.map((r) => {
            const beds = bedsByRoom[r.id] || [];
            const occ = beds.filter((x) => x.status === "OCCUPIED").length;
            return (
              <Link
                key={r.id}
                href={`/owner/rooms/${r.id}`}
                className="border rounded-xl p-4 hover:bg-gray-50"
              >
                <div className="font-semibold">Room {r.roomNumber}</div>
                <div className="text-sm text-gray-600 mt-1">
                  {occ} / {beds.length} occupied
                </div>
              </Link>
            );
```

```
              })}
            </div>
          </div>
        );
      })}
    </div>

    <button
      className="mt-6 border rounded-lg px-4 py-2 hover:bg-gray-50"
      onClick={() => loadAll().catch((e) => setErr(e.message))}
    >
      Refresh
    </button>
  </div>
 );
}
```

---

3) Tenant Complaint UI

Create: `frontend/src/app/tenant/complaints/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useEffect, useState } from "react";

type Building = { id: string; name: string; address: string };

export default function TenantComplaintsPage() {
 return (
   <Protected allowRoles={["TENANT"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const [buildings, setBuildings] = useState<Building[]>([]);
 const [buildingId, setBuildingId] = useState("");
 const [description, setDescription] = useState("");
 const [msg, setMsg] = useState<string | null>(null);
const [err, setErr] = useState<string | null>(null);

useEffect(() => {
  // tenant can see owner's buildings to choose where complaint belongs
  apiFetch<{ buildings: Building[] }>("/buildings")
    .then((res) => {
      setBuildings(res.buildings);
      if (res.buildings[0]) setBuildingId(res.buildings[0].id);
    })
    .catch((e) => setErr(e.message));
}, []);

async function submit() {
  setErr(null);
  setMsg(null);
  try {
    await apiFetch("/complaints", {
      method: "POST",
      body: JSON.stringify({ buildingId, description }),
    });
    setDescription("");
    setMsg("Complaint submitted successfully.");
  } catch (e: any) {
    setErr(e.message);
  }
}

return (
  <div className="max-w-3xl mx-auto p-6">
    <h1 className="text-2xl font-semibold">Raise a Complaint</h1>
    <p className="text-sm text-gray-600 mt-1">
      Choose the building and explain the issue clearly.
    </p>

    <div className="mt-6 border rounded-xl p-4">
      <div className="text-sm font-medium">Building</div>
      <select
        className="mt-2 w-full border rounded-lg p-3"
        value={buildingId}
        onChange={(e) => setBuildingId(e.target.value)}
      >
        {buildings.map((b) => (
          <option key={b.id} value={b.id}>
            {b.name}
          </option>
        ))}
      </select>

      <div className="text-sm font-medium mt-4">Complaint</div>
      <textarea
        className="mt-2 w-full border rounded-lg p-3 min-h-[120px]"
        placeholder="Example: Water leakage near room 203 wash area..."
        value={description}
        onChange={(e) => setDescription(e.target.value)}
      />

      <button
        className="mt-3 border rounded-lg px-4 py-2 hover:bg-gray-50 disabled:opacity-60"
        onClick={submit}
        disabled={!buildingId || description.trim().length < 5}
      >
        Submit
      </button>

      {msg && <div className="text-sm text-green-700 mt-3">{msg}</div>}
      {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
    </div>
  </div>
);
}
```

---

4) Owner Complaint Inbox + Resolve Button

Create: `frontend/src/app/owner/complaints/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useEffect, useState } from "react";

type Complaint = {
 id: string;
 description: string;
 status: "OPEN" | "RESOLVED";
 createdAt: string;
 tenant: { id: string; name: string; email: string };
 building: { id: string; name: string };
};

export default function OwnerComplaintsPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}

function statusPill(s: string) {
 const base = "text-xs px-2 py-1 rounded-full border";
 if (s === "OPEN") return `${base} bg-gray-50`;
 return `${base} bg-white`;
}

function Inner() {
 const [complaints, setComplaints] = useState<Complaint[]>([]);
 const [err, setErr] = useState<string | null>(null);
 const [loading, setLoading] = useState(true);

 async function load() {
   setErr(null);
   setLoading(true);
   const res = await apiFetch<{ complaints: Complaint[] }>("/complaints");
   setComplaints(res.complaints);
   setLoading(false);
 }

 useEffect(() => {
   load().catch((e) => {
     setErr(e.message);
     setLoading(false);
   });
 }, []);

 async function resolve(id: string) {
   setErr(null);
   try {
     await apiFetch(`/complaints/${id}/resolve`, { method: "PATCH" });
```

```
    await load();
  } catch (e: any) {
    setErr(e.message);
  }
}

const openCount = complaints.filter((c) => c.status === "OPEN").length;

return (
  <div className="max-w-6xl mx-auto p-6">
    <div className="flex items-start justify-between gap-4">
      <div>
        <h1 className="text-2xl font-semibold">Complaints</h1>
        <p className="text-sm text-gray-600 mt-1">
          Open: {openCount} • Total: {complaints.length}
        </p>
      </div>

      <button
        className="border rounded-lg px-4 py-2 hover:bg-gray-50"
        onClick={() => load().catch((e) => setErr(e.message))}
      >
        Refresh
      </button>
    </div>

    {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
    {loading && <div className="text-sm text-gray-600 mt-3">Loading…</div>}

    <div className="mt-6 space-y-3">
      {complaints.map((c) => (
        <div key={c.id} className="border rounded-xl p-4">
          <div className="flex items-start justify-between gap-4">
            <div>
              <div className="flex items-center gap-2">
                <div className="font-semibold">{c.building.name}</div>
                <span className={statusPill(c.status)}>{c.status}</span>
              </div>
              <div className="text-sm text-gray-600 mt-1">
                Tenant: {c.tenant.name} • {c.tenant.email}
              </div>
              <div className="text-sm text-gray-600 mt-1">
                Created: {new Date(c.createdAt).toLocaleString()}
              </div>
            </div>

            {c.status === "OPEN" ? (
              <button
                className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
                onClick={() => resolve(c.id)}
              >
                Mark Resolved
              </button>
            ) : null}
          </div>

          <div className="mt-3 text-sm">{c.description}</div>
        </div>
      ))}
    </div>
  </div>
 );
}
```

---

5) Make `/` route redirect nicely

Create: `frontend/src/app/page.tsx`

```
import Link from "next/link";

export default function Home() {
 return (
   <div className="max-w-3xl mx-auto p-6">
     <h1 className="text-3xl font-semibold">StayWise</h1>
     <p className="text-sm text-gray-600 mt-2">
       Smart PG & Property Management System
     </p>

     <div className="mt-6 flex gap-3">
       <Link className="border rounded-lg px-4 py-2 hover:bg-gray-50" href="/login">
         Login
       </Link>
       <Link className="border rounded-lg px-4 py-2 hover:bg-gray-50" href="/owner">
         Owner Dashboard
       </Link>
```

```
    </div>
  </div>
 );
}
```

✅ *Test Checklist (End-to-End)*

1. *Owner login → **/owner***
2. *Create building → rooms → beds*
3. *Create tenant + assign bed → bed becomes **OCCUPIED***
4. *Tenant login → **/tenant/complaints***
5. *Submit complaint → Owner sees it in **/owner/complaints***
6. *Owner clicks **Mark Resolved** → status becomes RESOLVED*
7. *Owner occupancy dashboard shows correct counts*

✅ *BACKEND — Rent Module*

1) Update Prisma Schema (add rent period columns + unique constraint)

Edit: `backend/prisma/schema.prisma`
Update your `Payment` *model to this:*

```
model Payment {
 id                   String   @id @default(uuid())
 tenantAssignmentId  String
 amount              Decimal  @db.Decimal(10, 2)
 paidOn              DateTime?
 status              PaymentStatus @default(PENDING)

 // ✅ NEW: period tracking (prevents duplicate payments for a month)
 periodYear         Int
 periodMonth        Int

 createdAt          DateTime @default(now())

 assignment TenantAssignment @relation(fields: [tenantAssignmentId], references: [id])

 @@unique([tenantAssignmentId, periodYear, periodMonth])
}
```

Now run migration:

```
npx prisma migrate dev --name rent_period
```

---

2) Create Rent Routes

Create folder + file:

```
mkdir -p backend/src/modules/rent
touch backend/src/modules/rent/routes.ts
```

`backend/src/modules/rent/routes.ts`

```
import { Router } from "express";
import { z } from "zod";
import { prisma } from "../../lib/prisma";
import { requireAuth, requireRole } from "../../middlewares/auth";
import { Role, PaymentStatus, AssignmentStatus } from "@prisma/client";

export const rentRouter = Router();

function currentPeriod() {
 const now = new Date();
 return { year: now.getFullYear(), month: now.getMonth() + 1 };
}

// ✅ Owner/Manager: Get pending rent list for current month
rentRouter.get(
 "/pending",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const { year, month } = currentPeriod();

   // Active assignments under this owner
   const assignments = await prisma.tenantAssignment.findMany({
     where: {
       status: AssignmentStatus.ACTIVE,
       tenant: { ownerId },
       bed: { room: { building: { ownerId } } },
     },
     include: {
       tenant: { select: { id: true, name: true, email: true, phone: true } },
       bed: {
         include: { room: { include: { building: { select: { id: true, name: true } } } } },
       },
       payments: {
         where: { periodYear: year, periodMonth: month },
         select: { id: true, status: true, amount: true, paidOn: true },
       },
     },
     orderBy: { createdAt: "desc" },
   });

   // Create a pending view: paid if payment exists + status PAID
   const pending = assignments
     .map((a) => {
       const pay = a.payments[0] || null;
```

```
      const isPaid = pay?.status === PaymentStatus.PAID;

      return {
        assignmentId: a.id,
        periodYear: year,
        periodMonth: month,

        tenant: a.tenant,
        building: a.bed.room.building,
        roomNumber: a.bed.room.roomNumber,
        bedNumber: a.bed.bedNumber,

        rentAmount: a.rentAmount,
        rentDueDay: a.rentDueDay,

        payment: pay,
        isPaid,
      };
    })
    .filter((x) => !x.isPaid);

  res.json({
    ok: true,
    period: { year, month },
    count: pending.length,
    pending,
  });
 }
);

// ✅ Owner/Manager: Mark rent paid for current month (or given period)
const markPaidSchema = z.object({
 tenantAssignmentId: z.string().uuid(),
 amount: z.number().positive(),
 periodYear: z.number().int().optional(),
 periodMonth: z.number().int().min(1).max(12).optional(),
});

rentRouter.post(
 "/mark-paid",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
  const ownerId = req.user!.ownerId!;
  const parsed = markPaidSchema.safeParse(req.body);
  if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

  const { tenantAssignmentId, amount } = parsed.data;
  const { year, month } = currentPeriod();
  const periodYear = parsed.data.periodYear ?? year;
  const periodMonth = parsed.data.periodMonth ?? month;

  // Ensure assignment belongs to this owner
  const assignment = await prisma.tenantAssignment.findUnique({
    where: { id: tenantAssignmentId },
    include: {
      tenant: true,
      bed: { include: { room: { include: { building: true } } } },
    },
  });

  if (
    !assignment ||
    assignment.status !== AssignmentStatus.ACTIVE ||
    assignment.tenant.ownerId !== ownerId ||
    assignment.bed.room.building.ownerId !== ownerId
  ) {
    return res.status(404).json({ error: "Assignment not found" });
  }

  // Upsert payment for that month (unique constraint protects duplicates)
  const payment = await prisma.payment.upsert({
    where: {
      tenantAssignmentId_periodYear_periodMonth: {
        tenantAssignmentId,
        periodYear,
        periodMonth,
      },
    },
    update: {
      status: PaymentStatus.PAID,
      amount,
      paidOn: new Date(),
    },
    create: {
      tenantAssignmentId,
      periodYear,
      periodMonth,
        status: PaymentStatus.PAID,
        amount,
        paidOn: new Date(),
      },
    });

    res.json({ ok: true, payment });
  }
);

// ✅ Tenant: Check current month rent status for their ACTIVE assignment
rentRouter.get(
  "/me/status",
  requireAuth,
  requireRole(Role.TENANT),
  async (req, res) => {
    const tenantId = req.user!.id;
    const { year, month } = currentPeriod();

    const assignment = await prisma.tenantAssignment.findFirst({
      where: { tenantId, status: AssignmentStatus.ACTIVE },
      include: {
        bed: { include: { room: { include: { building: { select: { id: true, name: true } } } } } },
        payments: {
          where: { periodYear: year, periodMonth: month },
          select: { id: true, status: true, amount: true, paidOn: true },
        },
      },
    });

    if (!assignment) return res.json({ ok: true, period: { year, month }, hasAssignment: false });

    const pay = assignment.payments[0] || null;
    const isPaid = pay?.status === PaymentStatus.PAID;

    res.json({
      ok: true,
      period: { year, month },
      hasAssignment: true,
      building: assignment.bed.room.building,
      roomNumber: assignment.bed.room.roomNumber,
      bedNumber: assignment.bed.bedNumber,
      rentAmount: assignment.rentAmount,
      rentDueDay: assignment.rentDueDay,
      payment: pay,
      isPaid,
    });
  }
);

// ✅ Owner/Manager: Payment history for an assignment
rentRouter.get(
  "/history/:assignmentId",
  requireAuth,
  requireRole(Role.OWNER, Role.MANAGER),
  async (req, res) => {
    const ownerId = req.user!.ownerId!;
    const assignmentId = req.params.assignmentId;

    const assignment = await prisma.tenantAssignment.findUnique({
      where: { id: assignmentId },
      include: { tenant: true, bed: { include: { room: { include: { building: true } } } } },
    });

    if (!assignment || assignment.tenant.ownerId !== ownerId) {
      return res.status(404).json({ error: "Assignment not found" });
    }

    const payments = await prisma.payment.findMany({
      where: { tenantAssignmentId: assignmentId },
      orderBy: [{ periodYear: "desc" }, { periodMonth: "desc" }],
    });

    res.json({ ok: true, payments });
  }
);
```

---

3) Wire Rent Router into App

Edit: `backend/src/app.ts`
Add import:

```
import { rentRouter } from "./modules/rent/routes";
```

Add route:

```
app.use("/rent", rentRouter);
```

Restart backend:

```
npm run dev
```

✅ *FRONTEND — Rent Module UI*

1) Owner Rent Dashboard Page

Create: `frontend/src/app/owner/rent/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useEffect, useState } from "react";

type PendingItem = {
 assignmentId: string;
 periodYear: number;
 periodMonth: number;
 tenant: { id: string; name: string; email: string; phone?: string | null };
 building: { id: string; name: string };
 roomNumber: string;
 bedNumber: string;
 rentAmount: string; // Prisma Decimal serialized
 rentDueDay: number;
 isPaid: boolean;
};

export default function OwnerRentPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const [items, setItems] = useState<PendingItem[]>([]);
 const [period, setPeriod] = useState<{ year: number; month: number } | null>(null);
 const [err, setErr] = useState<string | null>(null);
 const [loading, setLoading] = useState(true);

 async function load() {
   setErr(null);
   setLoading(true);
   const res = await apiFetch<any>("/rent/pending");
   setItems(res.pending);
   setPeriod(res.period);
   setLoading(false);
 }

 useEffect(() => {
   load().catch((e) => {
     setErr(e.message);
     setLoading(false);
   });
 }, []);

 async function markPaid(assignmentId: string, amountStr: string) {
   setErr(null);
   try {
     await apiFetch("/rent/mark-paid", {
       method: "POST",
       body: JSON.stringify({
         tenantAssignmentId: assignmentId,
         amount: Number(amountStr),
       }),
     });
     await load();
   } catch (e: any) {
     setErr(e.message);
   }
 }

 return (
   <div className="max-w-6xl mx-auto p-6">
     <div className="flex items-start justify-between gap-4">
       <div>
         <h1 className="text-2xl font-semibold">Rent Pending</h1>
         <p className="text-sm text-gray-600 mt-1">
           {period ? `Period: ${period.month}/${period.year}` : "Period: —"}
         </p>
       </div>
       <button className="border rounded-lg px-4 py-2 hover:bg-gray-50" onClick={() => load()}>
```

```
        Refresh
      </button>
    </div>

    {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
    {loading && <div className="text-sm text-gray-600 mt-3">Loading…</div>}

    <div className="mt-6 space-y-3">
      {items.map((x) => (
        <div key={x.assignmentId} className="border rounded-xl p-4">
          <div className="flex items-start justify-between gap-4">
            <div>
              <div className="font-semibold">
                {x.tenant.name} • {x.building.name}
              </div>
              <div className="text-sm text-gray-600 mt-1">
                Room {x.roomNumber} • Bed {x.bedNumber}
              </div>
              <div className="text-sm text-gray-600 mt-1">
                Due Day: {x.rentDueDay} • Rent: ₹{x.rentAmount}
              </div>
              <div className="text-sm text-gray-600 mt-1">
                {x.tenant.email} {x.tenant.phone ? `• ${x.tenant.phone}` : ""}
              </div>
            </div>

            <button
              className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
              onClick={() => markPaid(x.assignmentId, x.rentAmount)}
            >
              Mark Paid
            </button>
          </div>
        </div>
      ))}

      {!loading && items.length === 0 ? (
        <div className="text-sm text-gray-700 border rounded-xl p-4">
          ✅ No pending rent for this period.
        </div>
      ) : null}
    </div>
  </div>
);
```

```
}
```

---

2) Add Nav link for Rent

Update `frontend/src/components/TopNav.tsx` — inside OWNER/MANAGER links add:

```
<NavLink href="/owner/rent" label="Rent" />
```

---

3) Tenant Rent Status Page

Create: `frontend/src/app/tenant/rent/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useEffect, useState } from "react";

export default function TenantRentPage() {
 return (
   <Protected allowRoles={["TENANT"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const [data, setData] = useState<any>(null);
 const [err, setErr] = useState<string | null>(null);

 async function load() {
   setErr(null);
   const res = await apiFetch("/rent/me/status");
   setData(res);
 }

 useEffect(() => {
   load().catch((e) => setErr(e.message));
 }, []);

 if (err) return <div className="max-w-3xl mx-auto p-6 text-sm text-red-600">{err}</div>;
 if (!data) return <div className="max-w-3xl mx-auto p-6 text-sm
text-gray-600">Loading…</div>;

 return (
  <div className="max-w-3xl mx-auto p-6">
   <h1 className="text-2xl font-semibold">My Rent Status</h1>
   <p className="text-sm text-gray-600 mt-1">
    Period: {data.period.month}/{data.period.year}
   </p>

   {!data.hasAssignment ? (
    <div className="mt-6 border rounded-xl p-4 text-sm text-gray-700">
     No active bed assignment found.
    </div>
   ) : (
    <div className="mt-6 border rounded-xl p-4">
     <div className="font-semibold">{data.building.name}</div>
     <div className="text-sm text-gray-600 mt-1">
      Room {data.roomNumber} • Bed {data.bedNumber}
     </div>
     <div className="text-sm text-gray-600 mt-1">
      Rent: ₹{data.rentAmount} • Due Day: {data.rentDueDay}
     </div>

     <div className="mt-4">
      <span className="text-sm font-medium">Status: </span>
      <span className="text-sm">
       {data.isPaid ? "✅ PAID" : "⚠️ PENDING"}
      </span>
     </div>

     {data.payment?.paidOn ? (
      <div className="text-sm text-gray-600 mt-1">
       Paid on: {new Date(data.payment.paidOn).toLocaleString()}
      </div>
     ) : null}
    </div>
   )}

   <button className="mt-6 border rounded-lg px-4 py-2 hover:bg-gray-50" onClick={() =>
load()}>
    Refresh
   </button>
  </div>
```

);
}

Add tenant nav link too (optional but recommended):
Update `frontend/src/components/TopNav.tsx` inside TENANT links add:

<NavLink href="/tenant/rent" label="My Rent" />

✅ *End-to-End Test*

1. *Owner creates tenant + assigns bed (already working)*
2. *Owner opens **/owner/rent** → sees pending list*
3. *Click **Mark Paid** → item disappears*
4. *Tenant opens **/tenant/rent** → sees **PAID** for the month*

✅ *BACKEND UPGRADE*

1) Add `mark-unpaid`, `summary`, `assignments` endpoints

Edit: `backend/src/modules/rent/routes.ts`
*Append the following routes **below existing ones** (keep your current routes too):*

```
import { PaymentStatus } from "@prisma/client";

// helper already exists in your file:
// function currentPeriod() { ... }

const periodSchema = z.object({
 periodYear: z.number().int().optional(),
 periodMonth: z.number().int().min(1).max(12).optional(),
});

function resolvePeriod(bodyOrQuery: any) {
 const { year, month } = currentPeriod();
 const parsed = periodSchema.safeParse(bodyOrQuery);
 const periodYear = parsed.success && parsed.data.periodYear ? parsed.data.periodYear :
year;
 const periodMonth = parsed.success && parsed.data.periodMonth ? parsed.data.periodMonth :
month;
 return { periodYear, periodMonth };
}

// ✅ Owner/Manager: List all ACTIVE assignments (for rent table & history navigation)
rentRouter.get(
 "/assignments",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const { periodYear, periodMonth } = resolvePeriod(req.query);

   const assignments = await prisma.tenantAssignment.findMany({
     where: {
       status: AssignmentStatus.ACTIVE,
       tenant: { ownerId },
       bed: { room: { building: { ownerId } } },
     },
    include: {
      tenant: { select: { id: true, name: true, email: true, phone: true } },
      bed: {
        include: {
          room: {
            include: { building: { select: { id: true, name: true } } },
          },
        },
      },
      payments: {
        where: { periodYear, periodMonth },
        select: { id: true, status: true, amount: true, paidOn: true, periodYear: true, periodMonth:
true },
      },
    },
    orderBy: { createdAt: "desc" },
  });

  const now = new Date();
  const todayDay = now.getDate();

  const rows = assignments.map((a) => {
    const pay = a.payments[0] || null;
    const isPaid = pay?.status === PaymentStatus.PAID;
    const overdue = !isPaid && todayDay > a.rentDueDay;

    return {
      assignmentId: a.id,
      periodYear,
      periodMonth,
      tenant: a.tenant,
      building: a.bed.room.building,
      roomNumber: a.bed.room.roomNumber,
      bedNumber: a.bed.bedNumber,
      rentAmount: a.rentAmount,
      rentDueDay: a.rentDueDay,
      payment: pay,
      isPaid,
      overdue,
    };
  });

  res.json({ ok: true, period: { year: periodYear, month: periodMonth }, rows });
}
```

```
);

// ✅ Owner/Manager: Summary counts (paid/pending/overdue)
rentRouter.get(
 "/summary",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const { periodYear, periodMonth } = resolvePeriod(req.query);

   const assignments = await prisma.tenantAssignment.findMany({
     where: {
       status: AssignmentStatus.ACTIVE,
       tenant: { ownerId },
       bed: { room: { building: { ownerId } } },
     },
     include: {
       payments: {
         where: { periodYear, periodMonth },
         select: { status: true },
       },
     },
   });

   const todayDay = new Date().getDate();

   let paid = 0;
   let pending = 0;
   let overdue = 0;

   for (const a of assignments) {
     const pay = a.payments[0] || null;
     const isPaid = pay?.status === PaymentStatus.PAID;

     if (isPaid) paid++;
     else {
       pending++;
       if (todayDay > a.rentDueDay) overdue++;
     }
   }

   res.json({
     ok: true,
     period: { year: periodYear, month: periodMonth },
     totalActiveAssignments: assignments.length,
     paid,
     pending,
     overdue,
   });
 }
);

// ✅ Owner/Manager: Mark Unpaid (revert). Keeps row for audit-friendly history.
const markUnpaidSchema = z.object({
 tenantAssignmentId: z.string().uuid(),
 periodYear: z.number().int().optional(),
 periodMonth: z.number().int().min(1).max(12).optional(),
});

rentRouter.post(
 "/mark-unpaid",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const parsed = markUnpaidSchema.safeParse(req.body);
   if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

   const { tenantAssignmentId } = parsed.data;
   const { periodYear, periodMonth } = resolvePeriod(parsed.data);

   const assignment = await prisma.tenantAssignment.findUnique({
     where: { id: tenantAssignmentId },
     include: {
       tenant: true,
       bed: { include: { room: { include: { building: true } } } },
     },
   });

   if (
     !assignment ||
     assignment.status !== AssignmentStatus.ACTIVE ||
     assignment.tenant.ownerId !== ownerId ||
     assignment.bed.room.building.ownerId !== ownerId
   ) {
     return res.status(404).json({ error: "Assignment not found" });
   }
   // If it doesn't exist, create a PENDING row (useful for history)
   const payment = await prisma.payment.upsert({
     where: {
       tenantAssignmentId_periodYear_periodMonth: {
         tenantAssignmentId,
         periodYear,
         periodMonth,
       },
     },
     update: {
       status: PaymentStatus.PENDING,
       paidOn: null,
     },
     create: {
       tenantAssignmentId,
       periodYear,
       periodMonth,
       status: PaymentStatus.PENDING,
       amount: assignment.rentAmount, // default expected rent
       paidOn: null,
     },
   });

   res.json({ ok: true, payment });
 }
);
```

Restart backend:

```
npm run dev
```

✅ FRONTEND UPGRADE

1) Add "Rent Overview" page with summary + table + actions + search

Create: `frontend/src/app/owner/rent/page.tsx`
(Replace your existing rent page with this upgraded version.)

```
"use client";
```

```
import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useEffect, useMemo, useState } from "react";
import Link from "next/link";

type Row = {
 assignmentId: string;
 periodYear: number;
 periodMonth: number;
 tenant: { id: string; name: string; email: string; phone?: string | null };
 building: { id: string; name: string };
 roomNumber: string;
 bedNumber: string;
 rentAmount: string;
 rentDueDay: number;
 payment: null | { id: string; status: "PAID" | "PENDING"; amount: string; paidOn: string | null };
 isPaid: boolean;
 overdue: boolean;
};

function Pill({ text }: { text: string }) {
 return <span className="text-xs px-2 py-1 rounded-full border bg-gray-50">{text}</span>;
}

function Stat({ label, value }: { label: string; value: number }) {
 return (
   <div className="border rounded-xl p-4">
     <div className="text-sm text-gray-600">{label}</div>
     <div className="text-2xl font-semibold mt-1">{value}</div>
   </div>
 );
}

export default function OwnerRentPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const [rows, setRows] = useState<Row[]>([]);
```

```
  const [summary, setSummary] = useState<any>(null);
  const [period, setPeriod] = useState<{ year: number; month: number } | null>(null);
  const [q, setQ] = useState("");
  const [err, setErr] = useState<string | null>(null);
  const [loading, setLoading] = useState(true);

  async function load() {
    setErr(null);
    setLoading(true);

    const s = await apiFetch<any>("/rent/summary");
    const r = await apiFetch<any>("/rent/assignments");

    setSummary(s);
    setRows(r.rows);
    setPeriod(r.period);

    setLoading(false);
  }

  useEffect(() => {
    load().catch((e) => {
      setErr(e.message);
      setLoading(false);
    });
  }, []);

  const filtered = useMemo(() => {
    const term = q.trim().toLowerCase();
    if (!term) return rows;
    return rows.filter((x) => {
      const hay = `${x.tenant.name} ${x.tenant.email} ${x.building.name} ${x.roomNumber}
${x.bedNumber}`.toLowerCase();
      return hay.includes(term);
    });
  }, [q, rows]);

  async function markPaid(assignmentId: string, amountStr: string) {
    setErr(null);
    try {
      await apiFetch("/rent/mark-paid", {
        method: "POST",
        body: JSON.stringify({ tenantAssignmentId: assignmentId, amount: Number(amountStr) }),
      });
      await load();
    } catch (e: any) {
      setErr(e.message);
    }
  }

  async function markUnpaid(assignmentId: string) {
    setErr(null);
    try {
      await apiFetch("/rent/mark-unpaid", {
        method: "POST",
        body: JSON.stringify({ tenantAssignmentId: assignmentId }),
      });
      await load();
    } catch (e: any) {
      setErr(e.message);
    }
  }

  return (
    <div className="max-w-6xl mx-auto p-6">
      <div className="flex items-start justify-between gap-4">
        <div>
          <h1 className="text-2xl font-semibold">Rent Overview</h1>
          <p className="text-sm text-gray-600 mt-1">
            {period ? `Period: ${period.month}/${period.year}` : "Period: —"}
          </p>
        </div>
        <button className="border rounded-lg px-4 py-2 hover:bg-gray-50" onClick={() => load()}>
          Refresh
        </button>
      </div>

      {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
      {loading && <div className="text-sm text-gray-600 mt-3">Loading…</div>}

      {summary && (
        <div className="mt-6 grid grid-cols-1 md:grid-cols-4 gap-3">
          <Stat label="Total Active" value={summary.totalActiveAssignments} />
          <Stat label="Paid" value={summary.paid} />
          <Stat label="Pending" value={summary.pending} />
          <Stat label="Overdue" value={summary.overdue} />
        </div>
      )}
```

```
<div className="mt-6 border rounded-xl p-4">
  <div className="flex flex-col md:flex-row md:items-center gap-3 justify-between">
    <div className="font-semibold">Assignments (this period)</div>
    <input
      className="border rounded-lg p-3 w-full md:w-80"
      placeholder="Search tenant / building / room / bed"
      value={q}
      onChange={(e) => setQ(e.target.value)}
    />
  </div>

  <div className="mt-4 space-y-3">
    {filtered.map((x) => (
      <div key={x.assignmentId} className="border rounded-xl p-4">
        <div className="flex items-start justify-between gap-4">
          <div>
            <div className="flex items-center gap-2 flex-wrap">
              <div className="font-semibold">{x.tenant.name}</div>
              {x.isPaid ? <Pill text="PAID" /> : <Pill text="PENDING" />}
              {x.overdue ? <Pill text="OVERDUE" /> : null}
            </div>
            <div className="text-sm text-gray-600 mt-1">
              {x.building.name} • Room {x.roomNumber} • Bed {x.bedNumber}
            </div>
            <div className="text-sm text-gray-600 mt-1">
              Rent: ₹{x.rentAmount} • Due Day: {x.rentDueDay}
            </div>
            <div className="text-sm text-gray-600 mt-1">
              {x.tenant.email} {x.tenant.phone ? `• ${x.tenant.phone}` : ""}
            </div>

            {x.payment?.paidOn ? (
              <div className="text-sm text-gray-600 mt-1">
                Paid on: {new Date(x.payment.paidOn).toLocaleString()}
              </div>
            ) : null}

            <div className="mt-2">
              <Link
                className="text-sm underline"
                href={`/owner/rent/history/${x.assignmentId}`}
              >
                View payment history
              </Link>
            </div>
          </div>

          <div className="flex flex-col gap-2">
            {!x.isPaid ? (
              <button
                className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
                onClick={() => markPaid(x.assignmentId, x.rentAmount)}
              >
                Mark Paid
              </button>
            ) : (
              <button
                className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
                onClick={() => markUnpaid(x.assignmentId)}
              >
                Mark Unpaid
              </button>
            )}
          </div>
        </div>
      </div>
    ))}

    {!loading && filtered.length === 0 ? (
      <div className="text-sm text-gray-700 border rounded-xl p-4">
        No assignments match your search.
      </div>
    ) : null}
  </div>
</div>
</div>
);
}
```

---

2) Payment History Page (per assignment)

Create: `frontend/src/app/owner/rent/history/[id]/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import Link from "next/link";
import { useParams } from "next/navigation";
import { useEffect, useState } from "react";

type Payment = {
 id: string;
 periodYear: number;
 periodMonth: number;
 status: "PAID" | "PENDING";
 amount: string;
 paidOn: string | null;
 createdAt: string;
};

export default function RentHistoryPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const params = useParams();
 const assignmentId = params.id as string;

 const [payments, setPayments] = useState<Payment[]>([]);
 const [err, setErr] = useState<string | null>(null);

 async function load() {
   setErr(null);
   const res = await apiFetch<any>(`/rent/history/${assignmentId}`);
   setPayments(res.payments);
 }

 useEffect(() => {
   load().catch((e) => setErr(e.message));
 }, []);

 return (
   <div className="max-w-3xl mx-auto p-6">
     <Link href="/owner/rent" className="text-sm underline">
       ← Back to Rent Overview
```

```
      </Link>

      <h1 className="text-2xl font-semibold mt-3">Payment History</h1>
      <p className="text-sm text-gray-600 mt-1">Assignment ID: {assignmentId}</p>

      {err && <div className="text-sm text-red-600 mt-3">{err}</div>}

      <div className="mt-6 space-y-3">
        {payments.map((p) => (
          <div key={p.id} className="border rounded-xl p-4">
            <div className="flex items-center justify-between gap-4">
              <div className="font-semibold">
                {p.periodMonth}/{p.periodYear}
              </div>
              <div className="text-sm border rounded-full px-2 py-1 bg-gray-50">
                {p.status}
              </div>
            </div>

            <div className="text-sm text-gray-600 mt-2">Amount: ₹{p.amount}</div>

            {p.paidOn ? (
              <div className="text-sm text-gray-600 mt-1">
                Paid on: {new Date(p.paidOn).toLocaleString()}
              </div>
            ) : (
              <div className="text-sm text-gray-600 mt-1">Not paid</div>
            )}

            <div className="text-xs text-gray-500 mt-2">
              Logged: {new Date(p.createdAt).toLocaleString()}
            </div>
          </div>
        ))}

        {payments.length === 0 ? (
          <div className="text-sm text-gray-700 border rounded-xl p-4">
            No payment rows yet. (Once you mark paid/unpaid, it will appear.)
          </div>
        ) : null}
      </div>
    </div>
  );
}
```

---

3) Update Nav (ensure Rent exists)

In `frontend/src/components/TopNav.tsx`, make sure OWNER links include:

<NavLink href="/owner/rent" label="Rent" />

✅ BACKEND: PDF Receipt + WhatsApp Message

1) Install PDF generator

From `backend/`:

```
npm i pdfkit
npm i -D @types/pdfkit
```

---

2) Create Receipt Service

Create: `backend/src/modules/rent/receipt.ts`

```
import PDFDocument from "pdfkit";
import { prisma } from "../../lib/prisma";

function money(v: any) {
 // Prisma Decimal comes as string in JSON sometimes; keep robust
 const n = typeof v === "string" ? Number(v) : Number(v);
 return `₹${n.toFixed(2)}`;
}

function pad2(n: number) {
 return n < 10 ? `0${n}` : `${n}`;
}

function receiptNo(paymentId: string, y: number, m: number) {
 // short but unique-ish; fine for MVP; later you can move to sequences
 return `SW-${y}${pad2(m)}-${paymentId.slice(0, 8).toUpperCase()}`;
}

export async function buildReceiptPdfBuffer(params: {
 paymentId: string;
 ownerId: string;
}) {
 const payment = await prisma.payment.findUnique({
   where: { id: params.paymentId },
   include: {
     assignment: {
      include: {
        tenant: true,
        bed: {
          include: {
            room: { include: { building: true } },
          },
        },
      },
    },
  },
});

if (!payment) return { error: "Payment not found" as const };

const assignment = payment.assignment;
const building = assignment.bed.room.building;

// ✅ multi-tenant protection
if (building.ownerId !== params.ownerId) return { error: "Forbidden" as const };

if (payment.status !== "PAID") return { error: "Receipt available only for PAID payments" as
const };

const y = payment.periodYear;
const m = payment.periodMonth;

const owner = await prisma.owner.findUnique({
  where: { id: params.ownerId },
});

const doc = new PDFDocument({ size: "A4", margin: 50 });
const chunks: Buffer[] = [];

doc.on("data", (c) => chunks.push(c));
const done = new Promise<Buffer>((resolve) => {
  doc.on("end", () => resolve(Buffer.concat(chunks)));
});

// Header
doc.fontSize(20).text("StayWise", { align: "left" });
doc.fontSize(12).fillColor("#444").text("Rent Payment Receipt", { align: "left" });
doc.moveDown(0.5);
doc.fillColor("#000");
```

```
const rNo = receiptNo(payment.id, y, m);
doc.fontSize(10).fillColor("#444").text(`Receipt No: ${rNo}`);
doc.text(`Period: ${pad2(m)}/${y}`);
doc.text(`Issued On: ${new Date().toLocaleString("en-IN")}`);
doc.moveDown(1);
doc.fillColor("#000");

// Owner Block
doc.fontSize(12).text("Billed From", { underline: true });
doc.moveDown(0.2);
doc.fontSize(10).text(owner?.businessName ?? "PG Owner");
doc.text(owner?.city ?? "Bengaluru");
doc.moveDown(0.7);

// Tenant Block
doc.fontSize(12).text("Billed To", { underline: true });
doc.moveDown(0.2);
doc.fontSize(10).text(assignment.tenant.name);
doc.text(assignment.tenant.email);
if (assignment.tenant.phone) doc.text(assignment.tenant.phone);
doc.moveDown(0.7);

// Property Block
doc.fontSize(12).text("Property Details", { underline: true });
doc.moveDown(0.2);
doc.fontSize(10).text(`Building: ${building.name}`);
doc.text(`Address: ${building.address}`);
doc.text(`Room: ${assignment.bed.room.roomNumber}    Bed:
${assignment.bed.bedNumber}`);
doc.moveDown(0.7);

// Payment Block
doc.fontSize(12).text("Payment Details", { underline: true });
doc.moveDown(0.2);

const paidOn = payment.paidOn ? new Date(payment.paidOn).toLocaleString("en-IN") : "—";

doc.fontSize(10).text(`Rent Amount: ${money(payment.amount)}`);
doc.text(`Payment Status: PAID`);
doc.text(`Paid On: ${paidOn}`);
doc.moveDown(0.7);

// Footer note
doc.fontSize(9).fillColor("#666").text(
    "Note: This is a system-generated receipt from StayWise. Keep this for your records.",
    { align: "left" }
  );

  doc.end();
  const buffer = await done;

  return {
    ok: true as const,
    buffer,
    meta: {
      receiptNo: rNo,
      filename: `StayWise_Receipt_${rNo}.pdf`,
    },
  };
}

export async function buildWhatsAppText(params: { paymentId: string; ownerId: string }) {
  const payment = await prisma.payment.findUnique({
    where: { id: params.paymentId },
    include: {
      assignment: {
        include: {
          tenant: true,
          bed: { include: { room: { include: { building: true } } } },
        },
      },
    },
  });

  if (!payment) return { error: "Payment not found" as const };

  const building = payment.assignment.bed.room.building;
  if (building.ownerId !== params.ownerId) return { error: "Forbidden" as const };
  if (payment.status !== "PAID") return { error: "Message available only for PAID payments" as
const };

  const y = payment.periodYear;
  const m = payment.periodMonth;
  const rNo = receiptNo(payment.id, y, m);

  const paidOn = payment.paidOn ? new Date(payment.paidOn).toLocaleString("en-IN") : "—";

  const text =
```

`✅ StayWise Rent Received

Receipt: ${rNo}
Tenant: ${payment.assignment.tenant.name}
Building: ${building.name}
Room/Bed: ${payment.assignment.bed.room.roomNumber} / ${payment.assignment.bed.bedNumber}
Period: ${pad2(m)}/${y}
Amount: ${money(payment.amount)}
Paid On: ${paidOn}

Thank you.`;

```
 return { ok: true as const, text };
}
```

---

3) Add routes: download PDF + WhatsApp text

Edit: `backend/src/modules/rent/routes.ts`
Add imports at top:

```
import { buildReceiptPdfBuffer, buildWhatsAppText } from "./receipt";
```

Then add these routes at bottom:

```
// ✅ Download receipt PDF (Owner/Manager)
rentRouter.get(
 "/receipt/:paymentId",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const paymentId = req.params.paymentId;

   const out = await buildReceiptPdfBuffer({ paymentId, ownerId });
   if ("error" in out) return res.status(400).json({ error: out.error });

   res.setHeader("Content-Type", "application/pdf");
   res.setHeader("Content-Disposition", `attachment; filename="${out.meta.filename}"`);
   return res.send(out.buffer);
 }
);
```

```
// ✅ WhatsApp-ready message text (Owner/Manager)
rentRouter.get(
 "/receipt/:paymentId/message",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const paymentId = req.params.paymentId;

   const out = await buildWhatsAppText({ paymentId, ownerId });
   if ("error" in out) return res.status(400).json({ error: out.error });

   return res.json({ ok: true, text: out.text });
 }
);
```

Restart backend:

```
npm run dev
```

✅ FRONTEND: Download Receipt + Copy WhatsApp Text

1) Add helper for downloading PDF with auth

Create: `frontend/src/lib/download.ts`

```
import { getToken } from "./api";

const API_URL = process.env.NEXT_PUBLIC_API_URL || "http://localhost:4000";

export async function downloadReceiptPdf(paymentId: string) {
 const token = getToken();
 if (!token) throw new Error("Not logged in");

 const res = await fetch(`${API_URL}/rent/receipt/${paymentId}`, {
   headers: { Authorization: `Bearer ${token}` },
 });

 if (!res.ok) {
    const data = await res.json().catch(() => ({}));
    throw new Error(data?.error || "Failed to download receipt");
  }

  const blob = await res.blob();
  const url = window.URL.createObjectURL(blob);

  // Try to read filename from header
  const dispo = res.headers.get("content-disposition") || "";
  const match = dispo.match(/filename="(.+)"/);
  const filename = match?.[1] || `StayWise_Receipt_${paymentId}.pdf`;

  const a = document.createElement("a");
  a.href = url;
  a.download = filename;
  document.body.appendChild(a);
  a.click();
  a.remove();
  window.URL.revokeObjectURL(url);
}

export async function fetchReceiptMessage(paymentId: string) {
  const token = getToken();
  if (!token) throw new Error("Not logged in");

  const res = await fetch(`${API_URL}/rent/receipt/${paymentId}/message`, {
    headers: { Authorization: `Bearer ${token}` },
  });

  const data = await res.json().catch(() => ({}));
  if (!res.ok) throw new Error(data?.error || "Failed to fetch message");
  return data.text as string;
}
```

---

2) Add buttons in Owner Rent Overview (paid rows)

Edit: `frontend/src/app/owner/rent/page.tsx`

1. *Import at top:*

```
import { downloadReceiptPdf, fetchReceiptMessage } from "@/lib/download";
```

2. *Add two buttons inside the PAID block (where you show “Mark Unpaid”).*
   Find this section:

```
{!x.isPaid ? (
 ...
) : (
 <button ... onClick={() => markUnpaid(x.assignmentId)}>Mark Unpaid</button>
)}
```

Replace the PAID side with this:

```
) : (
 <div className="flex flex-col gap-2">
   <button
     className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
     onClick={() => markUnpaid(x.assignmentId)}
   >
     Mark Unpaid
   </button>

   {x.payment?.id ? (
     <>
       <button
         className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
         onClick={() => downloadReceiptPdf(x.payment!.id)}
       >
         Download Receipt (PDF)
       </button>

       <button
         className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
         onClick={async () => {
           const text = await fetchReceiptMessage(x.payment!.id);
           await navigator.clipboard.writeText(text);
           alert("WhatsApp message copied!");
         }}
       >
         Copy WhatsApp Message
       </button>
     </>
   ) : null}
 </div>
)
```

✅ *Now paid rows will allow PDF download + WhatsApp text copy.*

---

3) Add receipt button in Payment History page

Edit: `frontend/src/app/owner/rent/history/[id]/page.tsx`

1. *Import:*

```
import { downloadReceiptPdf, fetchReceiptMessage } from "@/lib/download";
```

2. *Inside each payment card, add this under status:*

```
{p.status === "PAID" ? (
 <div className="mt-3 flex gap-2 flex-wrap">
   <button
     className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
     onClick={() => downloadReceiptPdf(p.id)}
   >
     Download Receipt (PDF)
   </button>
   <button
     className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
     onClick={async () => {
       const text = await fetchReceiptMessage(p.id);
       await navigator.clipboard.writeText(text);
       alert("WhatsApp message copied!");
     }}
   >
     Copy WhatsApp Message
   </button>
 </div>
) : null}
```

✅ *BACKEND POLISH*

1) Add optional payment metadata (best for real operations)

Update `backend/prisma/schema.prisma` → `Payment` model

Add these fields inside `model Payment { ... }`:

```
 method      String?  // CASH / UPI / BANK / CARD etc
 reference   String?  // UTR / Ref No
 note        String?  // optional note
```

Now migrate:

```
npx prisma migrate dev --name payment_meta
```

---

2) Upgrade `/rent/mark-paid` to accept method/reference/note

Edit: `backend/src/modules/rent/routes.ts`

Update the `markPaidSchema` to:

```
const markPaidSchema = z.object({
 tenantAssignmentId: z.string().uuid(),
 amount: z.number().positive(),
 periodYear: z.number().int().optional(),
 periodMonth: z.number().int().min(1).max(12).optional(),

 // ✅ new optional fields
 method: z.string().min(2).optional(),     // ex: CASH / UPI
 reference: z.string().min(2).optional(),  // ex: UTR
 note: z.string().max(120).optional(),
});
```

In the `upsert` for payment, include these fields:

In `update:` add

```
method: parsed.data.method ?? undefined,
reference: parsed.data.reference ?? undefined,
note: parsed.data.note ?? undefined,
```

In `create:` add

```
method: parsed.data.method ?? "CASH",
reference: parsed.data.reference ?? null,
note: parsed.data.note ?? null,
```

Restart backend:

```
npm run dev
```

---

3) Upgrade Receipt PDF styling (clean, branded, professional)

Replace `backend/src/modules/rent/receipt.ts` with this polished version:

```
import PDFDocument from "pdfkit";
import { prisma } from "../../lib/prisma";

function money(v: any) {
 const n = typeof v === "string" ? Number(v) : Number(v);
 return `₹${n.toFixed(2)}`;
}
function pad2(n: number) {
 return n < 10 ? `0${n}` : `${n}`;
}
function receiptNo(paymentId: string, y: number, m: number) {
 return `SW-${y}${pad2(m)}-${paymentId.slice(0, 8).toUpperCase()}`;
}

function line(doc: PDFDocument, y: number) {
 doc
   .moveTo(50, y)
   .lineTo(545, y)
   .lineWidth(1)
   .strokeColor("#E5E7EB")
   .stroke();
}

function labelValue(doc: PDFDocument, label: string, value: string, x: number, y: number) {
 doc.fontSize(9).fillColor("#6B7280").text(label, x, y);
 doc.fontSize(10).fillColor("#111827").text(value || "—", x, y + 12);
}

export async function buildReceiptPdfBuffer(params: { paymentId: string; ownerId: string }) {
 const payment = await prisma.payment.findUnique({
   where: { id: params.paymentId },
   include: {
     assignment: {
       include: {
         tenant: true,
         bed: { include: { room: { include: { building: true } } } },
       },
     },
   },
 });

 if (!payment) return { error: "Payment not found" as const };

 const assignment = payment.assignment;
 const building = assignment.bed.room.building;

 if (building.ownerId !== params.ownerId) return { error: "Forbidden" as const };
 if (payment.status !== "PAID") return { error: "Receipt available only for PAID payments" as
const };

 const owner = await prisma.owner.findUnique({ where: { id: params.ownerId } });

 const y = payment.periodYear;
 const m = payment.periodMonth;
 const rNo = receiptNo(payment.id, y, m);

 const issuedOn = new Date().toLocaleString("en-IN");
 const paidOn = payment.paidOn ? new Date(payment.paidOn).toLocaleString("en-IN") : "—";

 const doc = new PDFDocument({ size: "A4", margin: 50 });
 const chunks: Buffer[] = [];

 doc.on("data", (c) => chunks.push(c));
 const done = new Promise<Buffer>((resolve) => doc.on("end", () =>
resolve(Buffer.concat(chunks))));

 // ===== Header band =====
 doc.rect(0, 0, 595, 110).fill("#111827");
 doc.fillColor("#FFFFFF").fontSize(22).text("StayWise", 50, 35);
 doc.fontSize(10).fillColor("#D1D5DB").text("Rent Payment Receipt", 50, 65);

 // Right header
 doc.fillColor("#FFFFFF").fontSize(10).text(`Receipt No: ${rNo}`, 350, 35, { align: "right", width:
195 });
 doc.fillColor("#D1D5DB").text(`Period: ${pad2(m)}/${y}`, 350, 55, { align: "right", width: 195 });
 doc.fillColor("#D1D5DB").text(`Issued: ${issuedOn}`, 350, 75, { align: "right", width: 195 });

 // Reset cursor
 doc.fillColor("#111827");
 doc.y = 130;

 // ===== Parties section =====
 doc.fontSize(12).fillColor("#111827").text("Parties", 50, doc.y);
 doc.moveDown(0.6);
 line(doc, doc.y);

 const startY = doc.y + 12;

 // From
 doc.fontSize(11).fillColor("#111827").text("Billed From", 50, startY);
 doc.fontSize(10).fillColor("#111827").text(owner?.businessName ?? "PG Owner", 50, startY +
18);
 doc.fontSize(9).fillColor("#6B7280").text(owner?.city ?? "Bengaluru", 50, startY + 34);

 // To
 doc.fontSize(11).fillColor("#111827").text("Billed To", 330, startY);
 doc.fontSize(10).fillColor("#111827").text(assignment.tenant.name, 330, startY + 18);
 doc.fontSize(9).fillColor("#6B7280").text(assignment.tenant.email, 330, startY + 34);
 if (assignment.tenant.phone) doc.text(assignment.tenant.phone, 330, startY + 48);

 doc.y = startY + 75;

 // ===== Property section =====
 doc.fontSize(12).fillColor("#111827").text("Property Details", 50, doc.y);
 doc.moveDown(0.6);
 line(doc, doc.y);

 const pY = doc.y + 12;
 labelValue(doc, "Building", building.name, 50, pY);
 labelValue(doc, "Room / Bed", `${assignment.bed.room.roomNumber} /
${assignment.bed.bedNumber}`, 250, pY);
 labelValue(doc, "Address", building.address, 50, pY + 38);
```

```
doc.y = pY + 80;

// ===== Payment section =====
doc.fontSize(12).fillColor("#111827").text("Payment Details", 50, doc.y);
doc.moveDown(0.6);
line(doc, doc.y);

const payY = doc.y + 14;

// Highlight amount
doc.roundedRect(50, payY, 495, 48, 10).fill("#F3F4F6");
doc.fillColor("#111827").fontSize(12).text("Amount Paid", 70, payY + 10);
doc.fontSize(16).text(money(payment.amount), 420, payY + 12, { align: "right", width: 105 });

// Meta rows
const metaY = payY + 70;
labelValue(doc, "Status", "PAID", 50, metaY);
labelValue(doc, "Paid On", paidOn, 200, metaY);
labelValue(doc, "Method", payment.method ?? "CASH", 50, metaY + 38);
labelValue(doc, "Reference", payment.reference ?? "—", 200, metaY + 38);
labelValue(doc, "Note", payment.note ?? "—", 50, metaY + 76);

doc.y = metaY + 120;

// Footer
line(doc, doc.y);
doc.moveDown(1);
doc.fontSize(8).fillColor("#6B7280").text(
  "This is a system-generated receipt from StayWise. For support, contact your property
manager/owner.",
  50,
  doc.y,
  { width: 495 }
);

doc.end();
const buffer = await done;

return {
  ok: true as const,
  buffer,
  meta: {
    receiptNo: rNo,
    filename: `StayWise_Receipt_${rNo}.pdf`,
   },
  };
}

export async function buildWhatsAppText(params: { paymentId: string; ownerId: string }) {
  const payment = await prisma.payment.findUnique({
    where: { id: params.paymentId },
    include: {
      assignment: {
        include: {
          tenant: true,
          bed: { include: { room: { include: { building: true } } } },
        },
      },
    },
  });

  if (!payment) return { error: "Payment not found" as const };

  const building = payment.assignment.bed.room.building;
  if (building.ownerId !== params.ownerId) return { error: "Forbidden" as const };
  if (payment.status !== "PAID") return { error: "Message available only for PAID payments" as
const };

  const y = payment.periodYear;
  const m = payment.periodMonth;
  const rNo = receiptNo(payment.id, y, m);

  const paidOn = payment.paidOn ? new Date(payment.paidOn).toLocaleString("en-IN") : "—";

  const text =
`✅ StayWise Rent Received

Receipt: ${rNo}
Tenant: ${payment.assignment.tenant.name}
Building: ${building.name}
Room/Bed: ${payment.assignment.bed.room.roomNumber} /
${payment.assignment.bed.bedNumber}
Period: ${pad2(m)}/${y}
Amount: ${money(payment.amount)}
Method: ${payment.method ?? "CASH"}
Reference: ${payment.reference ?? "—"}
Paid On: ${paidOn}
```

```
Thank you.`;

 return { ok: true as const, text };
}
```

Restart backend:

```
npm run dev
```

✅ *FRONTEND POLISH*

1) Add "Send via WhatsApp Web" button (prefilled)

Edit: `frontend/src/lib/download.ts` *and add this function:*

```
export function openWhatsAppWeb(phone: string | null | undefined, text: string) {
 const clean = (phone || "").replace(/[^\d]/g, "");
 const encoded = encodeURIComponent(text);

 // If phone exists: opens chat directly, else opens WhatsApp web with text ready
 const url = clean
   ? `https://wa.me/${clean}?text=${encoded}`
   : `https://web.whatsapp.com/send?text=${encoded}`;

 window.open(url, "_blank");
}
```

---

2) Update Owner Rent Overview buttons (PDF + Copy + WhatsApp Web)

In `frontend/src/app/owner/rent/page.tsx`*:*

1. *Import:*

```
import { downloadReceiptPdf, fetchReceiptMessage, openWhatsAppWeb } from
"@/lib/download";
```

2. *In the PAID actions area (where you already have Download + Copy), add this button:*

```
<button
 className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
 onClick={async () => {
   const text = await fetchReceiptMessage(x.payment!.id);
   openWhatsAppWeb(x.tenant.phone, text);
 }}
>
 Send via WhatsApp Web
</button>
```

✅ Uses tenant phone if present; otherwise opens WhatsApp Web with message ready.

---

3) Add method/reference/note inputs when marking paid (clean + fast)

In `frontend/src/app/owner/rent/page.tsx`, inside `markPaid(...)`, change the API call body to include defaults:

```
await apiFetch("/rent/mark-paid", {
 method: "POST",
 body: JSON.stringify({
   tenantAssignmentId: assignmentId,
   amount: Number(amountStr),

   // ✅ defaults (fast)
   method: "CASH",
   reference: "",
   note: "",
 }),
});
```

1) Update Owner Rent Overview page

Edit: `frontend/src/app/owner/rent/page.tsx`
Replace your current `markPaid(...)` *+ PAID/PENDING action buttons with the following changes.*

A) Add state + helper at top (inside `Inner()`*)*

```
const [payModalOpen, setPayModalOpen] = useState(false);
const [payTarget, setPayTarget] = useState<null | { assignmentId: string; amount: string;
phone?: string | null }>(null);

const [payMethod, setPayMethod] = useState("CASH");
const [payReference, setPayReference] = useState("");
const [payNote, setPayNote] = useState("");

function openPayModal(assignmentId: string, amount: string, phone?: string | null) {
 setPayTarget({ assignmentId, amount, phone });
 setPayMethod("CASH");
 setPayReference("");
 setPayNote("");
 setPayModalOpen(true);
}

function closePayModal() {
 setPayModalOpen(false);
 setPayTarget(null);
}
```

B) Replace `markPaid` *function with this (uses modal fields)*

```
async function markPaidConfirmed() {
 if (!payTarget) return;

 setErr(null);
 try {
   await apiFetch("/rent/mark-paid", {
     method: "POST",
     body: JSON.stringify({
       tenantAssignmentId: payTarget.assignmentId,
       amount: Number(payTarget.amount),
       method: payMethod,
       reference: payReference.trim() || undefined,
       note: payNote.trim() || undefined,
     }),
   });
   closePayModal();
   await load();
 } catch (e: any) {
   setErr(e.message);
 }
}
```

C) Replace the "Mark Paid" button (Pending rows)

Find the pending action button in your rent list and replace:

```
<button
 className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
 onClick={() => markPaid(x.assignmentId, x.rentAmount)}
>
 Mark Paid
</button>
```

with:

```
<button
 className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
 onClick={() => openPayModal(x.assignmentId, x.rentAmount, x.tenant.phone)}
>
 Mark Paid
</button>
```

D) Add the Modal UI at the bottom of the JSX (just before the final closing `</div>` of the page)

```
{payModalOpen && payTarget ? (
 <div className="fixed inset-0 z-50 flex items-center justify-center p-4">
   <div
     className="absolute inset-0 bg-black/40"
     onClick={closePayModal}
   />

   <div className="relative w-full max-w-lg rounded-2xl border bg-white p-5 shadow-lg">
     <div className="flex items-start justify-between gap-4">
       <div>
         <div className="text-lg font-semibold">Mark Rent as Paid</div>
         <div className="text-sm text-gray-600 mt-1">
           Amount: ₹{payTarget.amount}
         </div>
       </div>
```

```
  <button
    className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
    onClick={closePayModal}
  >
    Close
  </button>
</div>

<div className="mt-5 space-y-4">
  <div>
    <div className="text-sm font-medium">Payment Method</div>
    <select
      className="mt-2 w-full border rounded-lg p-3"
      value={payMethod}
      onChange={(e) => setPayMethod(e.target.value)}
    >
      <option value="CASH">CASH</option>
      <option value="UPI">UPI</option>
      <option value="BANK">BANK TRANSFER</option>
      <option value="CARD">CARD</option>
    </select>
  </div>

  <div>
    <div className="text-sm font-medium">Reference / UTR (optional)</div>
    <input
      className="mt-2 w-full border rounded-lg p-3"
      placeholder="Example: 1234567890 / UTR..."
      value={payReference}
      onChange={(e) => setPayReference(e.target.value)}
    />
  </div>

  <div>
    <div className="text-sm font-medium">Note (optional)</div>
    <textarea
      className="mt-2 w-full border rounded-lg p-3 min-h-[90px]"
      placeholder="Example: Paid in advance / Partial included..."
      value={payNote}
      onChange={(e) => setPayNote(e.target.value)}
    />
  </div>
```

```
        <div className="flex gap-3 justify-end">
          <button
            className="border rounded-lg px-4 py-2 hover:bg-gray-50"
            onClick={closePayModal}
          >
            Cancel
          </button>
          <button
            className="border rounded-lg px-4 py-2 hover:bg-gray-50 disabled:opacity-60"
            onClick={markPaidConfirmed}
            disabled={!payTarget?.assignmentId}
          >
            Confirm Paid
          </button>
        </div>

        <div className="text-xs text-gray-500">
          This will update receipts + WhatsApp message automatically.
        </div>
      </div>
    </div>
  </div>
) : null}
```

✅ *BACKEND CHANGES*

1) Update Prisma Schema (Vacate metadata)

Edit: `backend/prisma/schema.prisma`
Update `TenantAssignment` *model by adding:*

```
endedAt     DateTime?
endedNote   String?
```

Run migration:

```
npx prisma migrate dev --name vacate_fields
```

---

2) Create Tenant Admin Routes (list + occupancy + vacate)

Create: `backend/src/modules/tenants/adminRoutes.ts`

```
import { Router } from "express";
import { z } from "zod";
import { prisma } from "../../lib/prisma";
import { requireAuth, requireRole } from "../../middlewares/auth";
import { AssignmentStatus, BedStatus, Role } from "@prisma/client";

export const tenantAdminRouter = Router();

// ✅ Owner/Manager: List tenants (simple)
tenantAdminRouter.get(
 "/list",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const tenants = await prisma.user.findMany({
     where: { ownerId, role: Role.TENANT },
     select: { id: true, name: true, email: true, phone: true, createdAt: true },
     orderBy: { createdAt: "desc" },
   });
   res.json({ ok: true, tenants });
 }
```

```
);

// ✅ Owner/Manager: Occupancy list (Active assignments with room/bed)
tenantAdminRouter.get(
 "/occupancy",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;

   const assignments = await prisma.tenantAssignment.findMany({
     where: {
       status: AssignmentStatus.ACTIVE,
       tenant: { ownerId },
       bed: { room: { building: { ownerId } } },
     },
     include: {
       tenant: { select: { id: true, name: true, email: true, phone: true } },
       bed: {
         select: {
           id: true,
           bedNumber: true,
           status: true,
           room: {
             select: {
               id: true,
               roomNumber: true,
               building: { select: { id: true, name: true } },
             },
           },
         },
       },
     },
     orderBy: { createdAt: "desc" },
   });

   res.json({ ok: true, assignments });
 }
);

// ✅ Owner/Manager: Vacate/Checkout tenant (most important)
const vacateSchema = z.object({
 assignmentId: z.string().uuid(),
 endedNote: z.string().max(200).optional(),
});

tenantAdminRouter.post(
 "/vacate",
 requireAuth,
 requireRole(Role.OWNER, Role.MANAGER),
 async (req, res) => {
   const ownerId = req.user!.ownerId!;
   const parsed = vacateSchema.safeParse(req.body);
   if (!parsed.success) return res.status(400).json({ error: parsed.error.flatten() });

   const assignment = await prisma.tenantAssignment.findUnique({
     where: { id: parsed.data.assignmentId },
     include: {
       tenant: true,
       bed: { include: { room: { include: { building: true } } } },
     },
   });

   if (!assignment) return res.status(404).json({ error: "Assignment not found" });

   // multi-tenant protection
   if (
     assignment.tenant.ownerId !== ownerId ||
     assignment.bed.room.building.ownerId !== ownerId
   ) {
     return res.status(403).json({ error: "Forbidden" });
   }

   if (assignment.status !== AssignmentStatus.ACTIVE) {
     return res.status(400).json({ error: "Assignment already inactive" });
   }

   // 1) set assignment inactive + end metadata
   const updated = await prisma.tenantAssignment.update({
     where: { id: assignment.id },
     data: {
       status: AssignmentStatus.INACTIVE,
       endedAt: new Date(),
       endedNote: parsed.data.endedNote ?? null,
     },
   });

   // 2) set bed available
```

```
  await prisma.bed.update({
    where: { id: assignment.bedId },
    data: { status: BedStatus.AVAILABLE },
  });

  res.json({ ok: true, assignment: updated });
 }
);
```

---

3) Wire the new router into backend app

Edit: `backend/src/app.ts`

Add import:

import { tenantAdminRouter } from "./modules/tenants/adminRoutes";

Add route:

app.use("/tenants/admin", tenantAdminRouter);

Restart backend:

npm run dev

✅ *Backend now supports:*

- `GET /tenants/admin/list`
- `GET /tenants/admin/occupancy`
- `POST /tenants/admin/vacate`

✅ *FRONTEND CHANGES*

1) Add new nav link: Tenants

Edit: `frontend/src/components/TopNav.tsx`
Inside OWNER/MANAGER links add:

```
<NavLink href="/owner/tenants" label="Tenants" />
```

---

2) Create Tenants Management Page (list + occupancy + vacate modal-lite)

Create: `frontend/src/app/owner/tenants/page.tsx`

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useEffect, useMemo, useState } from "react";

type Tenant = {
 id: string;
 name: string;
 email: string;
 phone?: string | null;
 createdAt: string;
};

type Assignment = {
 id: string;
 createdAt: string;
 tenant: { id: string; name: string; email: string; phone?: string | null };
 bed: {
   id: string;
   bedNumber: string;
   status: string;
   room: { id: string; roomNumber: string; building: { id: string; name: string } };
 };
};

export default function OwnerTenantsPage() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
```

```
}

function Pill({ text }: { text: string }) {
 return <span className="text-xs px-2 py-1 rounded-full border bg-gray-50">{text}</span>;
}

function Inner() {
 const [tenants, setTenants] = useState<Tenant[]>([]);
 const [assignments, setAssignments] = useState<Assignment[]>([]);
 const [q, setQ] = useState("");

 const [vacateOpen, setVacateOpen] = useState(false);
 const [vacateAssignmentId, setVacateAssignmentId] = useState<string>("");
 const [vacateNote, setVacateNote] = useState("");

 const [err, setErr] = useState<string | null>(null);
 const [loading, setLoading] = useState(true);

 async function load() {
   setErr(null);
   setLoading(true);
   const t = await apiFetch<any>("/tenants/admin/list");
   const o = await apiFetch<any>("/tenants/admin/occupancy");
   setTenants(t.tenants);
   setAssignments(o.assignments);
   setLoading(false);
 }

 useEffect(() => {
   load().catch((e) => {
     setErr(e.message);
     setLoading(false);
   });
 }, []);

 const occByTenant = useMemo(() => {
   const m = new Map<string, Assignment>();
   for (const a of assignments) m.set(a.tenant.id, a);
   return m;
 }, [assignments]);

 const filteredTenants = useMemo(() => {
   const term = q.trim().toLowerCase();
   if (!term) return tenants;
```

```
  return tenants.filter((t) => {
    const hay = `${t.name} ${t.email} ${t.phone || ""}`.toLowerCase();
    return hay.includes(term);
  });
}, [q, tenants]);

function openVacate(id: string) {
  setVacateAssignmentId(id);
  setVacateNote("");
  setVacateOpen(true);
}

function closeVacate() {
  setVacateOpen(false);
  setVacateAssignmentId("");
  setVacateNote("");
}

async function confirmVacate() {
  setErr(null);
  try {
    await apiFetch("/tenants/admin/vacate", {
      method: "POST",
      body: JSON.stringify({ assignmentId: vacateAssignmentId, endedNote: vacateNote.trim() ||
undefined }),
    });
    closeVacate();
    await load();
  } catch (e: any) {
    setErr(e.message);
  }
}

return (
  <div className="max-w-6xl mx-auto p-6">
    <div className="flex items-start justify-between gap-4">
      <div>
        <h1 className="text-2xl font-semibold">Tenant Management</h1>
        <p className="text-sm text-gray-600 mt-1">
          List tenants + see occupancy + vacate/checkout
        </p>
      </div>
      <button className="border rounded-lg px-4 py-2 hover:bg-gray-50" onClick={() => load()}>
        Refresh
```

```
          </button>
        </div>

        {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
        {loading && <div className="text-sm text-gray-600 mt-3">Loading…</div>}

        <div className="mt-6 border rounded-xl p-4">
          <div className="flex flex-col md:flex-row md:items-center justify-between gap-3">
            <div className="font-semibold">Tenants</div>
            <input
              className="border rounded-lg p-3 w-full md:w-80"
              placeholder="Search name / email / phone"
              value={q}
              onChange={(e) => setQ(e.target.value)}
            />
          </div>

          <div className="mt-4 space-y-3">
            {filteredTenants.map((t) => {
              const a = occByTenant.get(t.id);
              return (
                <div key={t.id} className="border rounded-xl p-4">
                  <div className="flex items-start justify-between gap-4">
                    <div>
                      <div className="flex items-center gap-2 flex-wrap">
                        <div className="font-semibold">{t.name}</div>
                        {a ? <Pill text="OCCUPYING" /> : <Pill text="NOT ASSIGNED" />}
                      </div>
                      <div className="text-sm text-gray-600 mt-1">
                        {t.email} {t.phone ? `• ${t.phone}` : ""}
                      </div>

                      {a ? (
                        <div className="text-sm text-gray-700 mt-2">
                          {a.bed.room.building.name} • Room {a.bed.room.roomNumber} • Bed
{a.bed.bedNumber}
                        </div>
                      ) : null}
                    </div>

                    {a ? (
                      <button
                        className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
                        onClick={() => openVacate(a.id)}
                      >
                        Vacate
                      </button>
                    ) : null}
                  </div>
                </div>
              );
            })}

            {!loading && filteredTenants.length === 0 ? (
              <div className="text-sm text-gray-700 border rounded-xl p-4">No tenants found.</div>
            ) : null}
          </div>
        </div>

        {/* Vacate Modal */}
        {vacateOpen ? (
          <div className="fixed inset-0 z-50 flex items-center justify-center p-4">
            <div className="absolute inset-0 bg-black/40" onClick={closeVacate} />
            <div className="relative w-full max-w-lg rounded-2xl border bg-white p-5 shadow-lg">
              <div className="flex items-start justify-between gap-4">
                <div>
                  <div className="text-lg font-semibold">Vacate / Checkout</div>
                  <div className="text-sm text-gray-600 mt-1">
                    This will free the bed and close the assignment.
                  </div>
                </div>
                <button
                  className="border rounded-lg px-3 py-2 text-sm hover:bg-gray-50"
                  onClick={closeVacate}
                >
                  Close
                </button>
              </div>

              <div className="mt-5">
                <div className="text-sm font-medium">Vacate Note (optional)</div>
                <textarea
                  className="mt-2 w-full border rounded-lg p-3 min-h-[90px]"
                  placeholder="Example: Left on good terms / Pending dues / Key returned..."
                  value={vacateNote}
                  onChange={(e) => setVacateNote(e.target.value)}
                />
              </div>
```

```
           <div className="mt-4 flex justify-end gap-3">
             <button className="border rounded-lg px-4 py-2 hover:bg-gray-50"
onClick={closeVacate}>
               Cancel
             </button>
             <button
               className="border rounded-lg px-4 py-2 hover:bg-gray-50"
               onClick={confirmVacate}
             >
               Confirm Vacate
             </button>
           </div>

           <div className="text-xs text-gray-500 mt-3">
             Tip: If you want “due pending” enforcement, we can add it next.
           </div>
         </div>
       </div>
     ) : null}
   </div>
 );
}
```

✅ *BACKEND: Add Tenant Complaint History API*

Edit: `backend/src/modules/complaints/routes.ts`
Add this route (below existing routes):

```
complaintRouter.get(
 "/me",
 requireAuth,
 requireRole(Role.TENANT),
 async (req, res) => {
   const tenantId = req.user!.id;

   const complaints = await prisma.complaint.findMany({
     where: { tenantId },
     include: { building: { select: { id: true, name: true } } },
     orderBy: { createdAt: "desc" },
   });

   res.json({ ok: true, complaints });
 }
);
```

Restart backend:

```
npm run dev
```

✅ *FRONTEND: Tenant Home Page (Assigned Stay + Rent Status)*

Replace `frontend/src/app/tenant/page.tsx` *with this:*

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useAuth } from "@/components/AuthProvider";
import { useEffect, useState } from "react";
import Link from "next/link";
```

```
export default function TenantHome() {
 return (
   <Protected allowRoles={["TENANT"]}>
     <Inner />
   </Protected>
 );
}

function Card({ title, children }: { title: string; children: React.ReactNode }) {
 return (
   <div className="border rounded-xl p-4">
     <div className="font-semibold">{title}</div>
     <div className="mt-2 text-sm text-gray-700">{children}</div>
   </div>
 );
}

function Inner() {
 const { user, logout } = useAuth();
 const [data, setData] = useState<any>(null);
 const [err, setErr] = useState<string | null>(null);

 async function load() {
   setErr(null);
   const res = await apiFetch("/rent/me/status");
   setData(res);
 }

 useEffect(() => {
   load().catch((e) => setErr(e.message));
 }, []);

 return (
   <div className="max-w-4xl mx-auto p-6">
     <div className="flex items-start justify-between gap-4">
       <div>
         <h1 className="text-2xl font-semibold">Tenant Home</h1>
         <p className="text-sm text-gray-600 mt-1">
           Welcome, {user?.name}
         </p>
       </div>
       <button
         className="border rounded-lg px-4 py-2 hover:bg-gray-50 text-sm"
```

```
      onClick={logout}
    >
      Logout
    </button>
  </div>

  {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
  {!data && !err && <div className="text-sm text-gray-600 mt-3">Loading…</div>}

  {data ? (
    <div className="mt-6 grid grid-cols-1 md:grid-cols-2 gap-3">
      <Card title="My Stay Details">
        {!data.hasAssignment ? (
          <div>No active bed assignment found.</div>
        ) : (
          <div className="space-y-1">
            <div>
              <span className="text-gray-500">Building:</span>{" "}
              <b>{data.building.name}</b>
            </div>
            <div>
              <span className="text-gray-500">Room:</span>{" "}
              <b>{data.roomNumber}</b>
            </div>
            <div>
              <span className="text-gray-500">Bed:</span>{" "}
              <b>{data.bedNumber}</b>
            </div>
          </div>
        )}
      </Card>

      <Card title="This Month Rent Status">
        <div className="space-y-1">
          <div>
            <span className="text-gray-500">Period:</span>{" "}
            <b>
              {data.period.month}/{data.period.year}
            </b>
          </div>

          {data.hasAssignment ? (
            <>
              <div>
              <span className="text-gray-500">Rent:</span>{" "}
              <b>₹{data.rentAmount}</b>
            </div>
            <div>
              <span className="text-gray-500">Due Day:</span>{" "}
              <b>{data.rentDueDay}</b>
            </div>

            <div className="mt-2">
              <span className="text-gray-500">Status:</span>{" "}
              <b>{data.isPaid ? "✅ PAID" : "⚠️ PENDING"}</b>
            </div>

            {data.payment?.paidOn ? (
              <div className="text-gray-600">
                Paid on: {new Date(data.payment.paidOn).toLocaleString()}
              </div>
            ) : null}
          </>
        ) : (
          <div>No active assignment.</div>
        )}
      </div>
    </Card>

    <div className="md:col-span-2 border rounded-xl p-4">
      <div className="font-semibold">Quick Actions</div>
      <div className="mt-3 flex gap-3 flex-wrap">
        <Link className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
href="/tenant/complaints">
          Raise / View Complaints
        </Link>
        <Link className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
href="/tenant/rent">
          Detailed Rent Page
        </Link>
        <button
          className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
          onClick={() => load().catch((e) => setErr(e.message))}
        >
          Refresh
        </button>
      </div>
    </div>
```

```
      </div>
    ) : null}
  </div>
 );
}
```

✅ FRONTEND: Tenant Complaints Page (Submit + History)

Replace `frontend/src/app/tenant/complaints/page.tsx` *with this upgraded version:*

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import { useEffect, useMemo, useState } from "react";

type Building = { id: string; name: string; address: string };
type Complaint = {
 id: string;
 description: string;
 status: "OPEN" | "RESOLVED";
 createdAt: string;
 building: { id: string; name: string };
};

function Pill({ status }: { status: "OPEN" | "RESOLVED" }) {
 const base = "text-xs px-2 py-1 rounded-full border bg-gray-50";
 return <span className={base}>{status}</span>;
}

export default function TenantComplaintsPage() {
 return (
   <Protected allowRoles={["TENANT"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const [buildings, setBuildings] = useState<Building[]>([]);
const [buildingId, setBuildingId] = useState("");
const [description, setDescription] = useState("");

const [complaints, setComplaints] = useState<Complaint[]>([]);

const [msg, setMsg] = useState<string | null>(null);
const [err, setErr] = useState<string | null>(null);

async function loadAll() {
  setErr(null);
  setMsg(null);

  // buildings (scoped by ownerId via token)
  const b = await apiFetch<{ buildings: Building[] }>("/buildings");
  setBuildings(b.buildings);
  if (!buildingId && b.buildings[0]) setBuildingId(b.buildings[0].id);

  // complaint history
  const c = await apiFetch<{ complaints: Complaint[] }>("/complaints/me");
  setComplaints(c.complaints);
}

useEffect(() => {
  loadAll().catch((e) => setErr(e.message));
}, []);

const selectedBuilding = useMemo(
  () => buildings.find((x) => x.id === buildingId),
  [buildings, buildingId]
);

async function submit() {
  setErr(null);
  setMsg(null);
  try {
    await apiFetch("/complaints", {
      method: "POST",
      body: JSON.stringify({ buildingId, description }),
    });
    setDescription("");
    setMsg("✅ Complaint submitted.");
    await loadAll();
  } catch (e: any) {
    setErr(e.message);
  }
 }

 return (
   <div className="max-w-4xl mx-auto p-6">
     <h1 className="text-2xl font-semibold">Complaints</h1>
     <p className="text-sm text-gray-600 mt-1">
       Raise a complaint and track its status.
     </p>

     <div className="mt-6 border rounded-xl p-4">
       <div className="font-semibold">Raise New Complaint</div>

       <div className="mt-3">
         <div className="text-sm font-medium">Building</div>
         <select
           className="mt-2 w-full border rounded-lg p-3"
           value={buildingId}
           onChange={(e) => setBuildingId(e.target.value)}
         >
           {buildings.map((b) => (
             <option key={b.id} value={b.id}>
               {b.name}
             </option>
           ))}
         </select>
         {selectedBuilding ? (
           <div className="text-xs text-gray-500 mt-1">
             {selectedBuilding.address}
           </div>
         ) : null}
       </div>

       <div className="mt-4">
         <div className="text-sm font-medium">Complaint</div>
         <textarea
           className="mt-2 w-full border rounded-lg p-3 min-h-[120px]"
           placeholder="Example: Water issue in wash area, please fix."
           value={description}
           onChange={(e) => setDescription(e.target.value)}
         />
       </div>

       <button
```

```
    className="mt-3 border rounded-lg px-4 py-2 hover:bg-gray-50 disabled:opacity-60"
    onClick={submit}
    disabled={!buildingId || description.trim().length < 5}
  >
    Submit
  </button>

  {msg && <div className="text-sm text-green-700 mt-3">{msg}</div>}
  {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
</div>

<div className="mt-6 border rounded-xl p-4">
  <div className="flex items-center justify-between gap-4">
    <div className="font-semibold">My Complaint History</div>
    <button
      className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
      onClick={() => loadAll().catch((e) => setErr(e.message))}
    >
      Refresh
    </button>
  </div>

  <div className="mt-4 space-y-3">
    {complaints.map((c) => (
      <div key={c.id} className="border rounded-xl p-4">
        <div className="flex items-start justify-between gap-4">
          <div>
            <div className="font-semibold">{c.building.name}</div>
            <div className="text-sm text-gray-600 mt-1">
              {new Date(c.createdAt).toLocaleString()}
            </div>
          </div>
          <Pill status={c.status} />
        </div>

        <div className="mt-3 text-sm">{c.description}</div>
      </div>
    ))}

    {complaints.length === 0 ? (
      <div className="text-sm text-gray-700">
        No complaints yet.
      </div>
    ) : null}
```

```
      </div>
    </div>
  </div>
 );
}
```

✅ NAV UPDATE (Tenant links)

Ensure `frontend/src/components/TopNav.tsx` has tenant links:

```
<NavLink href="/tenant" label="Home" />
<NavLink href="/tenant/rent" label="My Rent" />
<NavLink href="/tenant/complaints" label="Complaints" />
```

1) Replace Owner Dashboard Page

Replace file: `frontend/src/app/owner/page.tsx` *with this:*

```
"use client";

import Protected from "@/components/Protected";
import { apiFetch } from "@/lib/api";
import Link from "next/link";
import { useEffect, useMemo, useState } from "react";
import { useAuth } from "@/components/AuthProvider";

type Building = { id: string; name: string; address: string };
type Room = { id: string; roomNumber: string; capacity: number };
type Bed = { id: string; bedNumber: string; status: "AVAILABLE" | "OCCUPIED" };
type Complaint = {
 id: string;
 description: string;
 status: "OPEN" | "RESOLVED";
 createdAt: string;
 tenant: { id: string; name: string; email: string };
 building: { id: string; name: string };
};
type OccupancyAssignment = {
 id: string;
 tenant: { id: string; name: string; email: string };
 bed: { id: string; bedNumber: string; room: { roomNumber: string; building: { id: string; name:
string } } };
};

function StatCard({ label, value, hint }: { label: string; value: number | string; hint?: string }) {
 return (
   <div className="border rounded-xl p-4">
     <div className="text-sm text-gray-600">{label}</div>
     <div className="text-2xl font-semibold mt-1">{value}</div>
     {hint ? <div className="text-xs text-gray-500 mt-1">{hint}</div> : null}
   </div>
 );
}

function Section({ title, action, children }: { title: string; action?: React.ReactNode; children:
React.ReactNode }) {
 return (
   <div className="border rounded-xl p-4">
     <div className="flex items-start justify-between gap-3">
       <div className="font-semibold">{title}</div>
       {action ? action : null}
     </div>
     <div className="mt-3">{children}</div>
   </div>
 );
}

export default function OwnerHome() {
 return (
   <Protected allowRoles={["OWNER", "MANAGER"]}>
     <Inner />
   </Protected>
 );
}

function Inner() {
 const { user } = useAuth();

 const [buildings, setBuildings] = useState<Building[]>([]);
 const [roomsByBuilding, setRoomsByBuilding] = useState<Record<string, Room[]>>({});
 const [bedsByRoom, setBedsByRoom] = useState<Record<string, Bed[]>>({});

 const [rentSummary, setRentSummary] = useState<any>(null);
 const [complaints, setComplaints] = useState<Complaint[]>([]);
 const [occupancy, setOccupancy] = useState<OccupancyAssignment[]>([]);

 const [name, setName] = useState("");
 const [address, setAddress] = useState("");

 const [err, setErr] = useState<string | null>(null);
 const [loading, setLoading] = useState(true);

 async function loadAll() {
   setErr(null);
   setLoading(true);

   // 1) buildings
   const bRes = await apiFetch<{ buildings: Building[] }>("/buildings");
   setBuildings(bRes.buildings);

   // 2) rooms + beds counts (for totals)
   const roomMap: Record<string, Room[]> = {};
   const bedMap: Record<string, Bed[]> = {};

   for (const b of bRes.buildings) {
     const rRes = await apiFetch<{ rooms: Room[] }>(`/rooms/${b.id}`);
     roomMap[b.id] = rRes.rooms;

     for (const r of rRes.rooms) {
       const bedRes = await apiFetch<{ beds: Bed[] }>(`/beds/${r.id}`);
       bedMap[r.id] = bedRes.beds;
     }
   }

   setRoomsByBuilding(roomMap);
   setBedsByRoom(bedMap);

   // 3) rent summary
   const rs = await apiFetch<any>("/rent/summary");
   setRentSummary(rs);

   // 4) complaints inbox (owner)
   const cRes = await apiFetch<{ complaints: Complaint[] }>("/complaints");
   setComplaints(cRes.complaints);

   // 5) occupancy list (owner)
   const oRes = await apiFetch<{ assignments: OccupancyAssignment[]
}>("/tenants/admin/occupancy");
   setOccupancy(oRes.assignments);

   setLoading(false);
 }

 useEffect(() => {
   loadAll().catch((e) => {
     setErr(e.message);
     setLoading(false);
   });
 }, []);

 async function addBuilding() {
   setErr(null);
   try {
     await apiFetch("/buildings", {
       method: "POST",
     body: JSON.stringify({ name, address }),
   });
   setName("");
   setAddress("");
   await loadAll();
 } catch (e: any) {
   setErr(e.message);
 }
}

const totals = useMemo(() => {
 let roomCount = 0;
 let bedCount = 0;
 let occupied = 0;

 for (const bId of Object.keys(roomsByBuilding)) {
   roomCount += (roomsByBuilding[bId] || []).length;
   for (const r of roomsByBuilding[bId] || []) {
     const beds = bedsByRoom[r.id] || [];
     bedCount += beds.length;
     occupied += beds.filter((x) => x.status === "OCCUPIED").length;
   }
 }

 return {
   buildings: buildings.length,
   rooms: roomCount,
   beds: bedCount,
   occupied,
   available: bedCount - occupied,
 };
}, [buildings, roomsByBuilding, bedsByRoom]);

const openComplaints = useMemo(
 () => complaints.filter((c) => c.status === "OPEN"),
 [complaints]
);

const latestComplaints = useMemo(
 () => openComplaints.slice(0, 5),
 [openComplaints]
);

const latestOccupancy = useMemo(() => occupancy.slice(0, 6), [occupancy]);
```

```
return (
  <div className="max-w-6xl mx-auto p-6">
    <div className="flex items-start justify-between gap-4">
      <div>
        <h1 className="text-2xl font-semibold">Owner Dashboard</h1>
        <p className="text-sm text-gray-600 mt-1">
          {user?.name} ({user?.role}) • StayWise Ops Overview
        </p>
      </div>

      <button
        className="border rounded-lg px-4 py-2 hover:bg-gray-50"
        onClick={() => loadAll().catch((e) => setErr(e.message))}
      >
        Refresh
      </button>
    </div>

    {err && <div className="text-sm text-red-600 mt-3">{err}</div>}
    {loading && <div className="text-sm text-gray-600 mt-3">Loading dashboard…</div>}

    {/* Top Stats */}
    <div className="mt-6 grid grid-cols-1 md:grid-cols-5 gap-3">
      <StatCard label="Buildings" value={totals.buildings} />
      <StatCard label="Rooms" value={totals.rooms} />
      <StatCard label="Beds" value={totals.beds} />
      <StatCard label="Occupied" value={totals.occupied} />
      <StatCard label="Available" value={totals.available} />
    </div>

    {/* Rent + Complaints */}
    <div className="mt-6 grid grid-cols-1 md:grid-cols-3 gap-3">
      <Section
        title="Rent Summary (This Month)"
        action={<Link className="text-sm underline" href="/owner/rent">Go to Rent</Link>}
      >
        {rentSummary ? (
          <div className="grid grid-cols-3 gap-2">
            <div className="border rounded-lg p-3">
              <div className="text-xs text-gray-600">Paid</div>
              <div className="text-xl font-semibold">{rentSummary.paid}</div>
            </div>
            <div className="border rounded-lg p-3">
              <div className="text-xs text-gray-600">Pending</div>
              <div className="text-xl font-semibold">{rentSummary.pending}</div>
            </div>
            <div className="border rounded-lg p-3">
              <div className="text-xs text-gray-600">Overdue</div>
              <div className="text-xl font-semibold">{rentSummary.overdue}</div>
            </div>
            <div className="text-xs text-gray-500 col-span-3 mt-1">
              Period: {rentSummary.period?.month}/{rentSummary.period?.year}
            </div>
          </div>
        ) : (
          <div className="text-sm text-gray-600">Loading rent summary…</div>
        )}
      </Section>

      <Section
        title="Complaints"
        action={<Link className="text-sm underline" href="/owner/complaints">Open
  Inbox</Link>}
      >
        <div className="flex items-center justify-between">
          <div className="text-sm text-gray-700">Open complaints</div>
          <div className="text-xl font-semibold">{openComplaints.length}</div>
        </div>

        <div className="mt-3 space-y-2">
          {latestComplaints.map((c) => (
            <div key={c.id} className="border rounded-lg p-3">
              <div className="text-sm font-medium">{c.building.name}</div>
              <div className="text-xs text-gray-600 mt-1">
                {c.tenant.name} • {new Date(c.createdAt).toLocaleString()}
              </div>
              <div className="text-sm mt-2 line-clamp-2">{c.description}</div>
            </div>
          ))}

          {latestComplaints.length === 0 ? (
            <div className="text-sm text-gray-600">No open complaints ✅</div>
          ) : null}
        </div>
      </Section>

      <Section
```

```
        title="Quick Actions"
        action={<Link className="text-sm underline" href="/owner/tenants">Tenant Mgmt</Link>}
      >
        <div className="flex flex-col gap-2">
          <Link className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
href="/owner/occupancy">
            View Occupancy Dashboard
          </Link>
          <Link className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
href="/owner/rent">
            Manage Rent + Receipts
          </Link>
          <Link className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
href="/owner/complaints">
            Handle Complaints
          </Link>
          <Link className="border rounded-lg px-4 py-2 text-sm hover:bg-gray-50"
href="/owner/tenants">
            Tenants + Vacate
          </Link>
        </div>
      </Section>
    </div>

    {/* Recent Occupancy */}
    <div className="mt-6 grid grid-cols-1 md:grid-cols-2 gap-3">
      <Section title="Recent Occupancy">
        <div className="space-y-2">
          {latestOccupancy.map((a) => (
            <div key={a.id} className="border rounded-lg p-3">
              <div className="font-medium text-sm">{a.tenant.name}</div>
              <div className="text-xs text-gray-600 mt-1">
                {a.bed.room.building.name} • Room {a.bed.room.roomNumber} • Bed
{a.bed.bedNumber}
              </div>
            </div>
          ))}
          {latestOccupancy.length === 0 ? (
            <div className="text-sm text-gray-600">No active occupancy yet.</div>
          ) : null}
        </div>
      </Section>

      {/* Add Building + Buildings List */}
```

```
<Section title="Add Building">
  <div className="grid grid-cols-1 gap-3">
    <input
      className="border rounded-lg p-3"
      placeholder="Building name"
      value={name}
      onChange={(e) => setName(e.target.value)}
    />
    <input
      className="border rounded-lg p-3"
      placeholder="Address"
      value={address}
      onChange={(e) => setAddress(e.target.value)}
    />
    <button
      className="border rounded-lg px-4 py-2 hover:bg-gray-50 disabled:opacity-60"
      disabled={name.trim().length < 2 || address.trim().length < 5}
      onClick={addBuilding}
    >
      Create Building
    </button>
  </div>

  <div className="mt-5">
    <div className="font-semibold text-sm">Your Buildings</div>
    <div className="mt-2 space-y-2">
      {buildings.slice(0, 6).map((b) => (
        <Link
          key={b.id}
          href={`/owner/buildings/${b.id}`}
          className="block border rounded-lg p-3 hover:bg-gray-50"
        >
          <div className="font-medium text-sm">{b.name}</div>
          <div className="text-xs text-gray-600 mt-1">{b.address}</div>
        </Link>
      ))}
      {buildings.length === 0 ? (
        <div className="text-sm text-gray-600">No buildings yet. Add one above.</div>
      ) : null}
      {buildings.length > 6 ? (
        <div className="text-xs text-gray-500">Showing 6 of {buildings.length}</div>
      ) : null}
    </div>
  </div>
```

```
      </Section>
     </div>
   </div>
 );
}
```

---

2) Make sure TopNav includes Dashboard link

In `frontend/src/components/TopNav.tsx` (OWNER/MANAGER links), ensure you have:

```
<NavLink href="/owner" label="Dashboard" />
```

1) Final UI Polish (remove `alert()` and add lightweight toast)

No libraries. We'll create a tiny toast system.

A) Add Toast Provider

Create: `frontend/src/components/ToastProvider.tsx`

```
"use client";

import React, { createContext, useContext, useMemo, useState } from "react";

type Toast = { id: string; type: "success" | "error" | "info"; message: string };

type ToastCtx = {
 push: (t: { type?: Toast["type"]; message: string }) => void;
};

const Ctx = createContext<ToastCtx | null>(null);

export function ToastProvider({ children }: { children: React.ReactNode }) {
 const [toasts, setToasts] = useState<Toast[]>([]);

 const push = (t: { type?: Toast["type"]; message: string }) => {
   const id = crypto.randomUUID();
   const toast: Toast = { id, type: t.type ?? "info", message: t.message };
   setToasts((prev) => [toast, ...prev]);

   // auto remove
   setTimeout(() => {
     setToasts((prev) => prev.filter((x) => x.id !== id));
   }, 2500);
 };

 const value = useMemo(() => ({ push }), []);

 return (
   <Ctx.Provider value={value}>
     {children}
```

```
      <div className="fixed right-4 top-4 z-[999] flex flex-col gap-2 w-[320px] max-w-[90vw]">
        {toasts.map((t) => (
          <div
            key={t.id}
            className="border rounded-xl bg-white shadow p-3"
          >
            <div className="text-xs text-gray-500 uppercase">{t.type}</div>
            <div className="text-sm mt-1">{t.message}</div>
          </div>
        ))}
      </div>
    </Ctx.Provider>
  );
}

export function useToast() {
  const ctx = useContext(Ctx);
  if (!ctx) throw new Error("useToast must be used within ToastProvider");
  return ctx;
}
```

B) Wrap it in layout

Edit: `frontend/src/app/layout.tsx`

```
import "./globals.css";
import { AuthProvider } from "@/components/AuthProvider";
import TopNav from "@/components/TopNav";
import { ToastProvider } from "@/components/ToastProvider";

export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html lang="en">
      <body>
        <AuthProvider>
          <ToastProvider>
            <TopNav />
            {children}
          </ToastProvider>
        </AuthProvider>
```

</body>
</html>
);
}

C) Replace `alert()` calls with toast (Rent pages)

In `frontend/src/app/owner/rent/page.tsx`

At top imports add:

import { useToast } from "@/components/ToastProvider";

Inside `Inner()` *add:*

const toast = useToast();

Replace:

alert("WhatsApp message copied!");

with:

toast.push({ type: "success", message: "WhatsApp message copied." });

Also wrap errors where relevant:

Example inside catch:

toast.push({ type: "error", message: e.message || "Something went wrong" });

In `frontend/src/app/owner/rent/history/[id]/page.tsx`

Do the same replacements.

✅ *Now UX is clean and "corporate".*

---

2) Security + Hardening (Minimum for Launch)

Backend hardening (simple, essential)

A) Use a strong JWT secret

In backend `.env`:

```
JWT_SECRET="a_very_long_random_string_50+_chars"
```

B) CORS: restrict origins in production

Edit `backend/src/app.ts` CORS part:

```
const allowedOrigins =
 process.env.NODE_ENV === "production"
   ? ["https://your-frontend-domain.com"]
   : [true as any];

app.use(cors({
 origin: allowedOrigins as any,
 credentials: true,
}));
```

✅ A) Decide deployment architecture (best-practice small SaaS)

Frontend: *Vercel (Next.js)*
Backend: *Render / Railway / Fly.io (Node API)*
Database: *Managed Postgres (Railway/Render/Neon/Supabase)*

Why: fastest, scalable, minimal ops.

✅ B) Production Environment Variables

Backend (Production)

Set these in your backend hosting platform:

- `NODE_ENV=production`
- `PORT=4000` *(platform may override)*
- `DATABASE_URL=postgresql://...` *(managed DB connection string)*
- `JWT_SECRET=...` *(strong random)*
- `JWT_EXPIRES_IN=7d`

Frontend (Production)

Set in Vercel:

- `NEXT_PUBLIC_API_URL=https://your-backend-domain.com`

✅ C) Backend Build + Start Commands

Add scripts in `backend/package.json`

```
{
 "scripts": {
   "dev": "ts-node-dev --respawn --transpile-only src/server.ts",
   "build": "tsc",
   "start": "node dist/server.js",
   "prisma:deploy": "prisma migrate deploy"
 }
}
```

Deployment steps

1. ***Build backend***

   ```
   npm run build
   ```

2. ***Run DB migrations on production***

   ```
   npx prisma migrate deploy
   ```

   (or `npm run prisma:deploy`)

3. ***Start backend***

   ```
   npm start
   ```

✅ *Your API will be live.*

✅ *D) Prisma Production Notes (IMPORTANT)*

- *In production, always use:*

  ```
  prisma migrate deploy
  ```

- *In dev, use:*

  ```
  prisma migrate dev
  ```

✅ *E) Frontend Deploy (Vercel)*

1. *Push repo to GitHub*
2. *Import to Vercel*
3. *Set env:*
    - `NEXT_PUBLIC_API_URL` *→ backend URL*
4. *Deploy*

✅ *F) Post-Deploy Sanity Tests (10 minutes)*

1. `GET /health` *returns ok*
2. *Register owner → login → create building/room/bed*
3. *Create tenant → assign bed*
4. *Rent overview → mark paid → download PDF receipt*
5. *Tenant login → rent status shows paid*
6. *Tenant complaint → owner resolves*

---

4) Updated Completion Status

✅ ***V1 is now ~99% complete***

*Remaining **~1%** optional (not blocking launch):*

- *“Vacant beds” quick list on Owner Dashboard*
- *Nice logo/brand assets*
- *Rate limiting (later)*

MASTER EXECUTION ORDER

🚀 MASTER EXECUTION ORDER

Project: StayWise V1 (Full SaaS MVP)

🔵 GLOBAL RULES (Antigravity must follow)

1. Never skip steps.
2. Complete each phase fully before moving forward.
3. Do not refactor previously working modules unless breaking.
4. Maintain multi-tenant isolation using `ownerId`.
5. Use Prisma migrations only.
6. Do not introduce external libraries unless specified.
7. Keep code production-structured.
8. Use environment variables strictly.

PHASE 1 — PROJECT INITIALIZATION

1️⃣ Create Folder Structure

```
staywise/
├── backend/
└── frontend/
```

2️⃣ Backend Setup

Inside `/backend`

Initialize:

```
npm init -y
npm install express cors dotenv jsonwebtoken bcrypt zod
npm install prisma @prisma/client
npm install pdfkit
npm install -D typescript ts-node-dev @types/node @types/express @types/jsonwebtoken
@types/bcrypt @types/pdfkit
```

Initialize TypeScript:

```
npx tsc --init
```

Initialize Prisma:

```
npx prisma init
```

3️⃣ Setup PostgreSQL (Local)

Ensure local Postgres running.

Set `.env`:

```
DATABASE_URL="postgresql://postgres:password@localhost:5432/staywise"
JWT_SECRET="very_long_secure_random_string"
JWT_EXPIRES_IN=7d
PORT=4000
NODE_ENV=development
```

PHASE 2 — DATABASE SCHEMA (CRITICAL FOUNDATION)

Antigravity must generate full schema:

Required Models

- Owner

- User (roles: SUPER_ADMIN, OWNER, MANAGER, TENANT)
- Building
- Room
- Bed
- TenantAssignment
- Payment
- Complaint

Requirements:

- Multi-tenant isolation by `ownerId`
- Unique constraint on monthly rent period:

  @@unique([tenantAssignmentId, periodYear, periodMonth])

After writing schema:

```
npx prisma migrate dev --name init
```

PHASE 3 — CORE BACKEND STRUCTURE

Create:

```
src/
 server.ts
 app.ts
 lib/
   prisma.ts
 middlewares/
   auth.ts
 modules/
```

```
auth/
buildings/
rooms/
beds/
tenants/
rent/
complaints/
```

PHASE 4 — AUTH SYSTEM

Implement:

- Register OWNER
- Login
- JWT middleware
- Role-based access control
- ownerId injection from token

Must protect:

- Buildings
- Rooms
- Beds
- Assignments
- Rent
- Complaints

PHASE 5 — PROPERTY MANAGEMENT

Implement APIs:

- Create building
- List buildings
- Create room under building
- List rooms
- Create beds under room
- List beds
- Auto-set bed status AVAILABLE / OCCUPIED

PHASE 6 — TENANT MANAGEMENT

Implement:

- Create tenant (by Owner)
- Assign tenant to bed
- Set bed status OCCUPIED
- Occupancy dashboard endpoint
- Vacate endpoint:
    - Set assignment INACTIVE
    - Set bed AVAILABLE
    - Save endedAt + endedNote

PHASE 7 — RENT MODULE (Monthly System)

Implement:

- Auto current period logic
- GET rent summary
- GET rent assignments
- Mark paid
  - method
  - reference
  - note
- Mark unpaid
- Payment history
- Overdue detection logic

PHASE 8 — RECEIPT SYSTEM

Implement:

- PDFKit receipt generation
- Receipt No format:

  SW-YYYYMM-XXXXXXXX
- Download endpoint

- WhatsApp message endpoint
- Multi-tenant security check before generating

PHASE 9 — COMPLAINT SYSTEM

Implement:

- Tenant create complaint
- Owner fetch complaints
- Owner resolve complaint
- Tenant fetch own complaints

PHASE 10 — FRONTEND (Next.js App Router)

Inside `/frontend`:

npx create-next-app@latest . --typescript

Install nothing extra.

Set `.env.local`:

NEXT_PUBLIC_API_URL=http://localhost:4000

PHASE 11 — FRONTEND STRUCTURE

Create:

```
app/
 login/
 owner/
   page.tsx (Dashboard)
   rent/
   complaints/
   tenants/
   occupancy/
 tenant/
   page.tsx
   rent/
   complaints/
components/
 AuthProvider
 Protected
 TopNav
 ToastProvider
lib/
 api.ts
 download.ts
```

PHASE 12 — OWNER DASHBOARD

Must include:

- Buildings count
- Rooms count
- Beds count
- Occupied count
- Available count
- Rent summary
- Complaint summary
- Quick links

PHASE 13 — TENANT PANEL

Must include:

- Assigned building/room/bed
- This month rent status
- Raise complaint
- Complaint history

PHASE 14 — UX POLISH

Add:

- Toast system (no alert)
- Proper button disabled states
- Refresh buttons
- Error handling consistency

PHASE 15 — PRE-LAUNCH HARDENING

1. Set NODE_ENV check for CORS.
2. Strong JWT secret.
3. Use `prisma migrate deploy` for production.

4. Verify no routes exposed without auth.
5. Ensure no owner can access other owner's data.

PHASE 16 — DEPLOYMENT ORDER

Backend

1. Push to GitHub.
2. Deploy to Render/Railway.
3. Set:
    - DATABASE_URL
    - JWT_SECRET
    - NODE_ENV=production
4. Run:

```
prisma migrate deploy
```

Frontend

1. Deploy to Vercel.
2. Set:

```
NEXT_PUBLIC_API_URL=https://backend-url
```

FINAL EXECUTION CONDITION

Antigravity must not:

- Skip schema constraints
- Skip role validation
- Mix tenant data across owners
- Generate unnecessary libraries
- Break modular structure

📊 FINAL STATUS AFTER EXECUTION

When all phases complete:

- Multi-tenant PG management SaaS working
- Rent monthly system with PDF receipts
- WhatsApp integration
- Vacate system
- Complaint system
- Owner dashboard
- Tenant portal

StayWise V1 = Production Ready MVP