QuirkyRoomie MERN Stack – Full-Project Scaffold

Built as per the internship brief and UI mock-ups you shared. The repo is laid out as **/backend** (Node + Express + MongoDB, MVC pattern) and **/frontend** (React + Vite + Tailwind v3). Copy-paste the structure, then run the commands in the **Getting Started** section. All heavyweight dependencies (Tailwind, Mongoose, JWT, bcrypt) are already wired.

---

1 Directory Layout

```
quirkyroomie/
├── backend/
│   ├── config/
│   │   └── db.js                # MongoDB connection helper
│   ├── controllers/
│   │   ├── authController.js
│   │   ├── complaintController.js
│   │   ├── voteController.js
│   │   └── statsController.js
│   ├── middleware/
│   │   ├── auth.js              # JWT protect + role check
│   │   └── errorHandler.js
│   ├── models/
│   │   ├── User.js
│   │   ├── Complaint.js
│   │   ├── Vote.js
│   │   └── Flat.js
│   ├── routes/
│   │   ├── authRoutes.js
│   │   ├── complaintRoutes.js
│   │   ├── voteRoutes.js
│   │   └── statsRoutes.js
│   ├── utils/
│   │   └── punishments.js       # Quirky punishments helper
│   ├── app.js                   # Express app (exported)
│   └── server.js                # Entry-point – loads env & starts HTTP
│
├── frontend/
│   ├── public/
│   │   └── index.html
│   └── src/
│       ├── assets/              # Icons, illustrations
│       ├── components/
│       │   ├── Auth/
│       │   │   ├── LoginForm.jsx
│       │   │   └── SignupForm.jsx
│       │   ├── Layout/
│       │   │   ├── Navbar.jsx
│       │   │   └── Sidebar.jsx
│       │   ├── Complaint/
│       │   │   ├── ComplaintCard.jsx
│       │   │   ├── ComplaintForm.jsx
│       │   │   └── ComplaintList.jsx
│       │   ├── Stats/
│       │   │   ├── KarmaStats.jsx
│       │   │   ├── Leaderboard.jsx
│       │   │   └── FlatStats.jsx
│       │   └── shared/Loader.jsx
│       ├── context/
│       │   ├── AuthContext.jsx
│       │   └── ComplaintContext.jsx
│       ├── hooks/
│       │   ├── useAuth.js
│       │   └── useComplaints.js
│       ├── pages/
│       │   ├── AuthPage.jsx
│       │   └── DashboardPage.jsx
│       ├── routes/AppRouter.jsx
│       ├── App.jsx
│       ├── main.jsx
│       └── tailwind.css          # @tailwind base; @tailwind …
│
├── .env.example                  # Copy → .env local / prod
├── README.md                     # Quick-start docs
└── package.json (root script aggregator)
```

Each folder is purpose-built so you can drop in more features (Trends view, Punishment history, etc.) without breaking separation of concerns.

---

2 Getting Started

```
# clone + install deps for both workspaces
$ git clone https://github.com/your-handle/quirkyroomie.git
$ cd quirkyroomie
# ① Backend
$ cd backend && npm install && cp ../.env.example .env && npm run dev
# ② Frontend (in new terminal)
$ cd ../frontend && npm install && npm run dev
```

Env vars (backend/.env)

```
MONGO_URI=mongodb+srv://…
JWT_SECRET=superStrongSecretKey
JWT_EXPIRE=7d
CLIENT_URL=http://localhost:5173
```

---

3 Backend – Key Files

3.1 `server.js`

```
import dotenv from 'dotenv';
import http from 'http';
import app from './app.js';

dotenv.config();
const PORT = process.env.PORT || 5000;
http.createServer(app).listen(PORT, () =>
  console.log(`🚀  API running on port ${PORT}`)
);
```

3.2 `config/db.js`

```
import mongoose from 'mongoose';

export default async function connectDB() {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log('    MongoDB connected');
  } catch (err) {
    console.error('    Mongo connection failed');
    process.exit(1);
  }
}
```

Called once inside **app.js**:

```
import express from 'express';
import cors from 'cors';
import morgan from 'morgan';
import connectDB from './config/db.js';

// route imports …
connectDB();
const app = express();
app.use(cors({ origin: process.env.CLIENT_URL, credentials: true }));
app.use(express.json());
app.use(morgan('dev'));

app.use('/api/auth', authRoutes);
app.use('/api/complaints', complaintRoutes);
app.use('/api/stats', statsRoutes);

// centralised error handler
app.use(errorHandler);
export default app;
```

3.3 `models/User.js`

```
import mongoose from 'mongoose';
import bcrypt from 'bcryptjs';

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, unique: true, required: true },
  password: { type: String, required: true, select: false },
  flatCode: { type: String, required: true, index: true },
  karma: { type: Number, default: 0 },
  role: { type: String, enum: ['flatmate', 'admin'], default: 'flatmate' },
}, { timestamps: true });

userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

userSchema.methods.matchPassword = function (entered) {
  return bcrypt.compare(entered, this.password);
};

export default mongoose.model('User', userSchema);
```

3.4 `middleware/auth.js`

```
import jwt from 'jsonwebtoken';
import User from '../models/User.js';

export const protect = async (req, _res, next) => {
  const hdr = req.headers.authorization;
  if (!hdr?.startsWith('Bearer ')) return next({ status: 401, msg: 'Not
authorised' });
  try {
    const { id } = jwt.verify(hdr.split(' ')[1], process.env.JWT_SECRET);
    req.user = await User.findById(id).select('-password');
    next();
  } catch (_) {
    next({ status: 401, msg: 'Token failed' });
  }
};
```

3.5 `controllers/authController.js`

```
import jwt from 'jsonwebtoken';
import User from '../models/User.js';

const generateToken = id =>
  jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn:
process.env.JWT_EXPIRE });

export const register = async (req, res, next) => {
  try {
    const { name, email, password, flatCode } = req.body;
    if (await User.findOne({ email })) throw { status: 400, msg: 'User
exists' };
    const user = await User.create({ name, email, password, flatCode });
    res.status(201).json({
      _id: user._id,
      name: user.name,
      email: user.email,
      token: generateToken(user._id),
    });
  } catch (err) { next(err); }
};
```

> Follow the same MVC approach for `Complaint`, `Vote`, and `Stats` controllers, keeping business logic out of routes.

---

4 Frontend – React + Tailwind v3

4.1 `tailwind.config.js`

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    './index.html',
    './src/**/*.{js,jsx,ts,tsx}',
  ],
  theme: {
    extend: {
      colors: {
        'qr-orange': '#ff7a1a',
        'qr-pink': '#ff4b8d',
      },
      backgroundImage: {
        'gradient-qr': 'linear-gradient(135deg,#fffaf5 0%,#fef5ff 100%)',
      },
    },
  },
  plugins: [require('@tailwindcss/forms')],
};
```

4.2 `src/App.jsx`

```
import { BrowserRouter } from 'react-router-dom';
import AppRouter from './routes/AppRouter';
import './tailwind.css';

function App() {
  return (
    <div className="min-h-screen bg-gradient-qr text-slate-800">
      <BrowserRouter>
        <AppRouter />
      </BrowserRouter>
    </div>
  );
}
export default App;
```

4.3 Example LoginForm

```
import { useState, useContext } from 'react';
import { AuthContext } from '../../context/AuthContext';

export default function LoginForm() {
  const { login } = useContext(AuthContext);
  const [form, setForm] = useState({ email: '', password: '' });
  const handleChange = e => setForm({ ...form, [e.target.name]:
e.target.value });

  return (
    <form
      onSubmit={e => {
        e.preventDefault();
        login(form);
      }}
      className="space-y-4"
    >
      <input
        name="email"
        type="email"
        placeholder="your.email@example.com"
        className="w-full rounded-md border-slate-300 p-3 focus:border-qr-
orange focus:ring-qr-pink"
        onChange={handleChange}
      />
      <input
        name="password"
        type="password"
        placeholder="Enter your password"
        className="w-full rounded-md border-slate-300 p-3 focus:border-qr-
orange focus:ring-qr-pink"
        onChange={handleChange}
      />
      <button
        type="submit"
        className="w-full rounded-md bg-gradient-to-r from-qr-orange to-qr-
pink py-2 font-semibold text-white shadow-md hover:opacity-90"
      >
        Sign In
      </button>
    </form>
  );
}
```

4.4 AuthContext.jsx (token keep-alive)

```
import { createContext, useState, useEffect } from 'react';
import axios from 'axios';
export const AuthContext = createContext();

export function AuthProvider({ children }) {
  const [user, setUser] = useState(() =>
JSON.parse(localStorage.getItem('qrUser')));
  const login = async creds => {
    const { data } = await axios.post('/api/auth/login', creds);
    setUser(data);
    localStorage.setItem('qrUser', JSON.stringify(data));
  };
  const logout = () => {
    setUser(null);
    localStorage.removeItem('qrUser');
  };
  useEffect(() => {
    axios.defaults.headers.common.Authorization = user ? `Bearer $
{user.token}` : '';
  }, [user]);
  return <AuthContext.Provider value={{ user, login, logout }}>{children}</
AuthContext.Provider>;
}
```

Repeat the same style for **SignupForm**, **ComplaintCard**, **Leaderboard**, etc., re-using Tailwind gradients to match the provided screen-shots.

---

5 API Reference (Backend → Frontend)

Method & Route	Purpose
`POST /api/auth/register`	**Sign Up** – returns token
`POST /api/auth/login`	**Sign In** – returns token
`POST /api/complaints`	Create complaint (protected)
`GET /api/complaints`	List active complaints
`POST /api/complaints/:id/vote`	Up/Down vote
`PUT /api/complaints/:id/resolve`	Mark resolved
`GET /api/leaderboard`	Karma ranking
`GET /api/flat/stats`	Flat stats overview

These map 1-to-1 to the PDF spec.

---

6 Deployment (Optional Bonus)

- **Frontend** – `npm run build` → deploy `dist/` folder to **Vercel**.
- **Backend** – add `Procfile` (`web: node server.js`) and deploy to **Render** or **Railway**.
- Set CORS + env vars accordingly.

---

7 Next Steps

1. **Run** the scaffolds locally – you should land on the same gradient UI you mocked.
2. **Fill in** the remaining component logic (filters, charts, punishment modal).
3. **Write tests** (Jest + Supertest for API, React Testing Library for UI).
4. **Prepare Postman collection** & **README badges** before submitting.

Happy hacking & may your flat stay drama-free! 🏡