Full-Stack Ratings App

A complete solution for the **Store Ratings Platform** challenge using **ExpressJS + PostgreSQL + React** with role-based access: **System Administrator**, **Normal User**, and **Store Owner**. Includes validation, sorting, filtering, search, dashboards, and clean project structure.

---

0) Repository Structure

```
ratings-app/
├─ server/
│  ├─ package.json
│  ├─ .env.example
│  ├─ src/
│  │  ├─ index.js
│  │  ├─ db.js
│  │  ├─ schema.sql
│  │  ├─ seed.js
│  │  ├─ utils/
│  │  │  ├─ validators.js
│  │  │  └─ auth.js
│  │  ├─ middleware/
│  │  │  ├─ auth.js
│  │  │  └─ roles.js
│  │  ├─ routes/
│  │  │  ├─ auth.routes.js
│  │  │  ├─ admin.routes.js
│  │  │  ├─ stores.routes.js
│  │  │  └─ owner.routes.js
│  │  └─ services/
│  │     ├─ users.service.js
│  │     ├─ stores.service.js
│  │     └─ ratings.service.js
│  └─ README.md
└─ client/
   ├─ package.json
   ├─ index.html
   ├─ vite.config.js
   └─ src/
      ├─ main.jsx
      ├─ App.jsx
      ├─ api.js
      ├─ auth.js
      ├─ components/
      │  ├─ NavBar.jsx
      │  ├─ ProtectedRoute.jsx
      │  ├─ SearchBox.jsx
      │  ├─ SortableHeader.jsx
      │  └─ StarRating.jsx
      ├─ pages/
      │  ├─ Login.jsx
      │  ├─ Signup.jsx
      │  ├─ AdminDashboard.jsx
      │  ├─ UsersTable.jsx
      │  ├─ StoresTable.jsx
      │   ├─ UserStores.jsx
      │   ├─ OwnerDashboard.jsx
      │   └─ ChangePassword.jsx
      └─ styles.css
```

1) Server (ExpressJS + PostgreSQL)

1.1 package.json

```
{
  "name": "ratings-app-server",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "dev": "node src/index.js",
    "db:setup": "node src/seed.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "express-validator": "^7.0.1",
    "jsonwebtoken": "^9.0.2",
    "pg": "^8.11.3",
    "uuid": "^9.0.1"
  }
}
```

1.2 .env.example

```
PORT=5000
DATABASE_URL=postgres://postgres:postgres@localhost:5432/ratings_app
JWT_SECRET=supersecretkey_change_me
CORS_ORIGIN=http://localhost:5173
```

1.3 src/db.js

```
import pg from 'pg';
import dotenv from 'dotenv';
dotenv.config();

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

export async function query(sql, params = []) {
  const client = await pool.connect();
  try {
    const res = await client.query(sql, params);
    return res;
  } finally {
    client.release();
  }
}
```

1.4 src/schema.sql

```sql
-- Users
CREATE TABLE IF NOT EXISTS users (
  id UUID PRIMARY KEY,
  name VARCHAR(60) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  address VARCHAR(400) NOT NULL,
  role VARCHAR(10) NOT NULL CHECK (role IN ('ADMIN','USER','OWNER')),
  created_at TIMESTAMP NOT NULL DEFAULT now()
);

-- Stores
CREATE TABLE IF NOT EXISTS stores (
  id UUID PRIMARY KEY,
  name VARCHAR(60) NOT NULL,
  email VARCHAR(255) NOT NULL,
  address VARCHAR(400) NOT NULL,
  owner_id UUID REFERENCES users(id) ON DELETE SET NULL,
  created_at TIMESTAMP NOT NULL DEFAULT now()
);

-- Ratings
CREATE TABLE IF NOT EXISTS ratings (
  id UUID PRIMARY KEY,
  store_id UUID NOT NULL REFERENCES stores(id) ON DELETE CASCADE,
  user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
  created_at TIMESTAMP NOT NULL DEFAULT now(),
  updated_at TIMESTAMP NOT NULL DEFAULT now(),
  UNIQUE(user_id, store_id)
);

-- Helper view for avg rating per store
CREATE OR REPLACE VIEW store_avg AS
SELECT s.id AS store_id, COALESCE(AVG(r.rating),0)::numeric(3,2) AS
avg_rating, COUNT(r.id) AS ratings_count
FROM stores s
LEFT JOIN ratings r ON r.store_id = s.id
GROUP BY s.id;
```

1.5 src/utils/validators.js

```javascript
export const patterns = {
  password: /^(?=.*[A-Z])(?=.*[^A-Za-z0-9]).{8,16}$/,
  email: /^[^\s@]+@[^\s@]+\.[^\s@]+$/
};

export function validateName(name) {
  return typeof name === 'string' && name.trim().length >= 20 &&
name.trim().length <= 60;
}
export function validateAddress(address) {
  return typeof address === 'string' && address.trim().length > 0 &&
address.trim().length <= 400;
}
export function validateEmail(email) {
  return patterns.email.test(email);
}
```

```
export function validatePassword(pw) {
  return patterns.password.test(pw);
}
```

1.6 src/utils/auth.js

```
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';
dotenv.config();

export function signToken(payload) {
  return jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '7d' });
}

export function verifyToken(token) {
  return jwt.verify(token, process.env.JWT_SECRET);
}
```

1.7 src/middleware/auth.js

```
import { verifyToken } from '../utils/auth.js';

export function requireAuth(req, res, next) {
  const header = req.headers.authorization || '';
  const token = header.startsWith('Bearer ') ? header.slice(7) : null;
  if (!token) return res.status(401).json({ message: 'Unauthorized' });
  try {
    const decoded = verifyToken(token);
    req.user = decoded; // { id, role, name, email }
    next();
  } catch (e) {
    return res.status(401).json({ message: 'Invalid token' });
  }
}
```

1.8 src/middleware/roles.js

```
export function requireRole(...roles) {
  return (req, res, next) => {
    if (!req.user || !roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Forbidden' });
    }
    next();
  };
}
```

1.9 src/services/users.service.js

```
import { query } from '../db.js';
import { v4 as uuid } from 'uuid';

export async function createUser({ name, email, password_hash, address,
role }) {
  const id = uuid();
  await query(
    `INSERT INTO users (id, name, email, password_hash, address, role)
VALUES ($1,$2,$3,$4,$5,$6)`,
    [id, name, email, password_hash, address, role]
  );
  return { id, name, email, address, role };
}

export async function findUserByEmail(email) {
  const { rows } = await query(`SELECT * FROM users WHERE email=$1`,
[email]);
  return rows[0];
}

export async function changePassword(userId, password_hash) {
  await query(`UPDATE users SET password_hash=$1 WHERE id=$2`,
[password_hash, userId]);
}

export async function listUsers({ q, role, sort='name', order='asc',
limit=10, offset=0 }) {
  const params = [];
  let where = [];
  if (q) {
    params.push(`%${q}%`);
    params.push(`%${q}%`);
    params.push(`%${q}%`);
    where.push(`(name ILIKE $${params.length-2} OR email ILIKE
$${params.length-1} OR address ILIKE $${params.length})`);
  }
  if (role) {
    params.push(role);
    where.push(`role = $${params.length}`);
  }
  const whereSql = where.length ? `WHERE ${where.join(' AND ')}` : '';
  const sortSafe =
['name','email','address','role','created_at'].includes(sort) ? sort :
'name';
  const orderSafe = order?.toLowerCase() === 'desc' ? 'DESC' : 'ASC';
  params.push(limit); params.push(offset);
  const { rows } = await query(
    `SELECT id, name, email, address, role, created_at
     FROM users
     ${whereSql}
     ORDER BY ${sortSafe} ${orderSafe}
     LIMIT $${params.length-1} OFFSET $${params.length}`,
    params
  );
  return rows;
}
```

1.10 src/services/stores.service.js

```
import { query } from '../db.js';
import { v4 as uuid } from 'uuid';

export async function createStore({ name, email, address, owner_id }) {
  const id = uuid();
  await query(
    `INSERT INTO stores (id, name, email, address, owner_id) VALUES
($1,$2,$3,$4,$5)`,
    [id, name, email, address, owner_id || null]
  );
  return { id, name, email, address, owner_id };
```

```
}

export async function listStoresForUser(userId, { q, sort='name',
order='asc', limit=10, offset=0 }) {
  const params = [];
  let where = '';
  if (q) {
    params.push(`%${q}%`);
    params.push(`%${q}%`);
    where = `WHERE s.name ILIKE $1 OR s.address ILIKE $2`;
  }
  const sortSafe =
['name','address','avg_rating','ratings_count'].includes(sort) ? sort :
'name';
  const orderSafe = order?.toLowerCase() === 'desc' ? 'DESC' : 'ASC';
  params.push(limit); params.push(offset); params.push(userId);
  const { rows } = await query(
    `SELECT s.id, s.name, s.address,
            COALESCE(sa.avg_rating,0) AS overall_rating,
            COALESCE(
              (SELECT rating FROM ratings r WHERE r.store_id = s.id AND
r.user_id = $${params.length}),
              NULL
            ) AS my_rating
     FROM stores s
     LEFT JOIN store_avg sa ON sa.store_id = s.id
     ${where}
     ORDER BY ${sortSafe} ${orderSafe}
     LIMIT $${params.length-2} OFFSET $${params.length-1}`,
    params
  );
  return rows;
}

export async function listStoresForAdmin({ q, sort='name', order='asc',
limit=10, offset=0 }) {
  const params = [];
  let where = '';
  if (q) {
    params.push(`%${q}%`);
    params.push(`%${q}%`);
    params.push(`%${q}%`);
    where = `WHERE s.name ILIKE $1 OR s.email ILIKE $2 OR s.address ILIKE
$3`;
  }
  const sortSafe =
['name','email','address','avg_rating','ratings_count'].includes(sort) ?
sort : 'name';
  const orderSafe = order?.toLowerCase() === 'desc' ? 'DESC' : 'ASC';
  params.push(limit); params.push(offset);
  const { rows } = await query(
    `SELECT s.id, s.name, s.email, s.address,
            COALESCE(sa.avg_rating,0) AS rating,
            sa.ratings_count
     FROM stores s
     LEFT JOIN store_avg sa ON sa.store_id = s.id
     ${where}
     ORDER BY ${sortSafe} ${orderSafe}
     LIMIT $${params.length-1} OFFSET $${params.length}`,
    params
  );
  return rows;
}
```

1.11 src/services/ratings.service.js

```
import { query } from '../db.js';
import { v4 as uuid } from 'uuid';

export async function upsertRating({ store_id, user_id, rating }) {
  const { rows } = await query(`SELECT id FROM ratings WHERE store_id=$1
AND user_id=$2`, [store_id, user_id]);
  const now = new Date();
  if (rows[0]) {
    await query(`UPDATE ratings SET rating=$1, updated_at=$2 WHERE id=$3`,
[rating, now, rows[0].id]);
    return { id: rows[0].id, store_id, user_id, rating };
  } else {
    const id = uuid();
    await query(`INSERT INTO ratings (id, store_id, user_id, rating) VALUES
($1,$2,$3,$4)`, [id, store_id, user_id, rating]);
    return { id, store_id, user_id, rating };
  }
}

export async function listOwnerRatings(ownerId) {
  const { rows } = await query(
    `SELECT u.name as user_name, u.email as user_email, r.rating,
r.updated_at,
            s.name as store_name, sa.avg_rating
     FROM stores s
     JOIN ratings r ON r.store_id = s.id
     JOIN users u ON u.id = r.user_id
     LEFT JOIN store_avg sa ON sa.store_id = s.id
     WHERE s.owner_id=$1
     ORDER BY r.updated_at DESC`,
     [ownerId]
  );
  return rows;
}
```

1.12 src/routes/auth.routes.js

```
import express from 'express';
import bcrypt from 'bcryptjs';
import { body, validationResult } from 'express-validator';
import { createUser, findUserByEmail, changePassword } from
'../services/users.service.js';
import { signToken } from '../utils/auth.js';
import { validateName, validateAddress, validatePassword } from
'../utils/validators.js';
import { requireAuth } from '../middleware/auth.js';

const router = express.Router();

router.post('/signup', async (req, res) => {
  try {
    const { name, email, password, address } = req.body;
    if (!validateName(name)) return res.status(400).json({ message: 'Name
must be 20-60 characters' });
```

```
    if (!validateAddress(address)) return res.status(400).json({ message:
'Address required, max 400 chars' });
    if (!validatePassword(password)) return res.status(400).json({ message:
'Password must be 8-16 chars, include uppercase & special char' });
    if (!email) return res.status(400).json({ message: 'Email required' });

    const existing = await findUserByEmail(email);
    if (existing) return res.status(409).json({ message: 'Email already
registered' });

    const password_hash = await bcrypt.hash(password, 10);
    const user = await createUser({ name, email, password_hash, address,
role: 'USER' });
    const token = signToken({ id: user.id, role: 'USER', name: user.name,
email: user.email });
    return res.json({ token, user });
  } catch (e) {
    return res.status(500).json({ message: 'Signup failed' });
  }
});

router.post('/login', async (req, res) => {
  const { email, password } = req.body;
  const user = await findUserByEmail(email);
  if (!user) return res.status(401).json({ message: 'Invalid credentials'
});
  const ok = await bcrypt.compare(password, user.password_hash);
  if (!ok) return res.status(401).json({ message: 'Invalid credentials' });
  const token = signToken({ id: user.id, role: user.role, name: user.name,
email: user.email });
  res.json({ token, user: { id: user.id, name: user.name, email:
user.email, address: user.address, role: user.role } });
});

router.post('/change-password', requireAuth, async (req, res) => {
  const { newPassword } = req.body;
  if (!validatePassword(newPassword)) return res.status(400).json({
message: 'Invalid password format' });
  const hash = await bcrypt.hash(newPassword, 10);
  await changePassword(req.user.id, hash);
  res.json({ message: 'Password updated' });
});

export default router;
```

1.13 src/routes/admin.routes.js

```
import express from 'express';
import bcrypt from 'bcryptjs';
import { requireAuth } from '../middleware/auth.js';
import { requireRole } from '../middleware/roles.js';
import { validateName, validateAddress, validatePassword, patterns } from
'../utils/validators.js';
import { createUser, listUsers } from '../services/users.service.js';
import { createStore, listStoresForAdmin } from
'../services/stores.service.js';
import { query } from '../db.js';

const router = express.Router();
```

```
router.use(requireAuth, requireRole('ADMIN'));

router.get('/stats', async (req, res) => {
  const [{ rows: u }, { rows: s }, { rows: r }] = await Promise.all([
    query('SELECT COUNT(*)::int as total_users FROM users'),
    query('SELECT COUNT(*)::int as total_stores FROM stores'),
    query('SELECT COUNT(*)::int as total_ratings FROM ratings')
  ]);
  res.json({
    totalUsers: u[0].total_users,
    totalStores: s[0].total_stores,
    totalRatings: r[0].total_ratings
  });
});

router.post('/users', async (req, res) => {
  const { name, email, password, address, role } = req.body;
  if (!['ADMIN','USER','OWNER'].includes(role)) return
res.status(400).json({ message: 'Invalid role' });
  if (!validateName(name)) return res.status(400).json({ message: 'Name 20-
60 chars' });
  if (!validateAddress(address)) return res.status(400).json({ message:
'Address max 400 chars' });
  if (!patterns.email.test(email)) return res.status(400).json({ message:
'Invalid email' });
  if (!validatePassword(password)) return res.status(400).json({ message:
'Weak password' });
  const password_hash = await bcrypt.hash(password, 10);
  const user = await createUser({ name, email, password_hash, address, role
});
  res.json(user);
});

router.get('/users', async (req, res) => {
  const { q, role, sort, order, limit = 10, offset = 0 } = req.query;
  const rows = await listUsers({ q, role, sort, order, limit:
Number(limit), offset: Number(offset) });
  res.json(rows);
});

router.post('/stores', async (req, res) => {
  const { name, email, address, owner_id } = req.body;
  if (!validateName(name)) return res.status(400).json({ message: 'Name 20-
60 chars' });
  if (!validateAddress(address)) return res.status(400).json({ message:
'Address max 400 chars' });
  if (!patterns.email.test(email)) return res.status(400).json({ message:
'Invalid email' });
  const store = await createStore({ name, email, address, owner_id });
  res.json(store);
});

router.get('/stores', async (req, res) => {
  const { q, sort, order, limit=10, offset=0 } = req.query;
  const rows = await listStoresForAdmin({ q, sort, order, limit:
Number(limit), offset: Number(offset) });
  res.json(rows);
});

export default router;
```

1.14 src/routes/stores.routes.js

```
import express from 'express';
import { requireAuth } from '../middleware/auth.js';
import { validateAddress, validateName } from '../utils/validators.js';
import { listStoresForUser } from '../services/stores.service.js';
import { upsertRating } from '../services/ratings.service.js';

const router = express.Router();

router.use(requireAuth);

router.get('/', async (req, res) => {
  const { q, sort, order, limit=10, offset=0 } = req.query;
  const rows = await listStoresForUser(req.user.id, { q, sort, order,
limit: Number(limit), offset: Number(offset) });
  res.json(rows);
});

router.post('/:id/ratings', async (req, res) => {
  const { rating } = req.body;
  const store_id = req.params.id;
  const r = Number(rating);
  if (!Number.isInteger(r) || r < 1 || r > 5) return res.status(400).json({
message: 'Rating must be 1-5' });
  const result = await upsertRating({ store_id, user_id: req.user.id,
rating: r });
  res.json(result);
});

export default router;
```

1.15 src/routes/owner.routes.js

```
import express from 'express';
import { requireAuth } from '../middleware/auth.js';
import { requireRole } from '../middleware/roles.js';
import { listOwnerRatings } from '../services/ratings.service.js';

const router = express.Router();

router.use(requireAuth, requireRole('OWNER'));

router.get('/ratings', async (req, res) => {
  const rows = await listOwnerRatings(req.user.id);
  res.json(rows);
});

export default router;
```

1.16 src/index.js

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import { query } from './db.js';
import fs from 'fs';
import path from 'path';
import authRoutes from './routes/auth.routes.js';
import adminRoutes from './routes/admin.routes.js';
import storesRoutes from './routes/stores.routes.js';
import ownerRoutes from './routes/owner.routes.js';
import { fileURLToPath } from 'url';

dotenv.config();
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const app = express();
app.use(cors({ origin: process.env.CORS_ORIGIN, credentials: false }));
app.use(express.json());

app.get('/api/health', (req, res) => res.json({ ok: true }));
app.use('/api/auth', authRoutes);
app.use('/api/admin', adminRoutes);
app.use('/api/stores', storesRoutes);
app.use('/api/owner', ownerRoutes);

const port = process.env.PORT || 5000;
app.listen(port, () => console.log(`Server running on
http://localhost:${port}`));
```

1.17 src/seed.js

```
import fs from 'fs';
import path from 'path';
import bcrypt from 'bcryptjs';
import { fileURLToPath } from 'url';
import { query } from './db.js';
import { v4 as uuid } from 'uuid';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

async function run() {
  const schema = fs.readFileSync(path.join(__dirname,
'schema.sql')).toString();
  await query(schema);

  const adminId = uuid();
  const ownerId = uuid();
  const userId = uuid();

  const adminPw = await bcrypt.hash('Admin@123', 10);
  const ownerPw = await bcrypt.hash('Owner@123', 10);
  const userPw  = await bcrypt.hash('User@1234', 10);

  // Upsert users
  await query(`DELETE FROM ratings; DELETE FROM stores; DELETE FROM
users;`);

  await query(
    `INSERT INTO users (id, name, email, password_hash, address, role)
VALUES
     ($1,'System Administrator Alpha','admin@example.com',$2,'Admin HQ,
Pune','ADMIN'),
     ($3,'Owner Operations Bravo','owner@example.com',$4,'Baner,
Pune','OWNER'),
     ($5,'User Researcher Charlie','user@example.com',$6,'Kothrud, 
Pune','USER')`,
    [adminId, adminPw, ownerId, ownerPw, userId, userPw]
  );

  const storeId = uuid();
  await query(
    `INSERT INTO stores (id, name, email, address, owner_id) VALUES 
($1,$2,$3,$4,$5)`,
    [storeId, 'Bravo Electronics', 'store@example.com', 'FC Road, Pune', 
ownerId]
  );

  console.log('DB setup complete. Admin: admin@example.com / Admin@123');
}

run().then(()=>process.exit(0)).catch(e=>{ console.error(e); 
process.exit(1); });
```

2) Client (React + Vite)

2.1 package.json

```
{
  "name": "ratings-app-client",
  "version": "1.0.0",
  "private": true,
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.26.0"
  },
  "devDependencies": {
    "vite": "^5.3.3"
  }
}
```

2.2 vite.config.js

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  server: { port: 5173 }
});
```

2.3 index.html

```
<!doctype html>
```

```
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Ratings App</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

2.4 src/styles.css

```
body { font-family: system-ui, sans-serif; margin: 0; }
.container { max-width: 1080px; margin: 0 auto; padding: 16px; }
.card { border: 1px solid #ddd; border-radius: 10px; padding: 16px; margin:
12px 0; }
.table { width: 100%; border-collapse: collapse; }
.table th, .table td { border-bottom: 1px solid #e5e5e5; padding: 8px;
text-align: left; }
.input { padding: 8px; border: 1px solid #ccc; border-radius: 8px; width:
100%; }
.btn { padding: 8px 12px; border: 1px solid #333; background: #fff; border-
radius: 8px; cursor: pointer; }
.btn.primary { background: #333; color: #fff; }
.flex { display: flex; gap: 8px; align-items: center; }
.grid { display: grid; gap: 12px; }
.grid-2 { grid-template-columns: repeat(2, 1fr); }
.grid-3 { grid-template-columns: repeat(3, 1fr); }
.badge { font-size: 12px; padding: 2px 8px; background: #eee; border-
radius: 999px; }
```

2.5 src/main.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import App from './App'
import './styles.css'

createRoot(document.getElementById('root')).render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
)
```

2.6 src/auth.js

```
export function getToken() { return localStorage.getItem('token'); }
export function setToken(t) { localStorage.setItem('token', t); }
export function clearToken() { localStorage.removeItem('token'); }
export function getUser() { const u = localStorage.getItem('user'); return
u ? JSON.parse(u) : null; }
export function setUser(u) { localStorage.setItem('user',
JSON.stringify(u)); }
```

2.7 src/api.js

```
const API = import.meta.env.VITE_API || 'http://localhost:5000/api';
export async function api(path, { method='GET', body, token }={}) {
  const res = await fetch(`${API}${path}`, {
    method, headers: {
      'Content-Type': 'application/json',
      ...(token ? { Authorization: `Bearer ${token}` } : {})
    },
    ...(body ? { body: JSON.stringify(body) } : {})
  });
  if (!res.ok) throw new Error((await res.json()).message || 'Request
failed');
  return res.json();
}
```

2.8 src/components/NavBar.jsx

```
import { Link, useNavigate } from 'react-router-dom'
import { clearToken, getUser } from '../auth'

export default function NavBar(){
  const navigate = useNavigate();
  const user = getUser();
  function logout(){ clearToken(); localStorage.removeItem('user');
navigate('/login'); }
  return (
    <div className="card flex" style={{justifyContent:'space-between'}}>
      <div className="flex">
        <Link to="/" className="btn">Home</Link>
        {user?.role==='ADMIN' && <Link to="/admin"
className="btn">Admin</Link>}
        {user?.role==='OWNER' && <Link to="/owner"
className="btn">Owner</Link>}
        {user && <Link to="/stores" className="btn">Stores</Link>}
        {user && <Link to="/change-password" className="btn">Change
Password</Link>}
      </div>
      <div className="flex">
        {user ? <>
          <span className="badge">{user.role}</span>
          <button className="btn" onClick={logout}>Logout</button>
        </> : <Link to="/login" className="btn primary">Login</Link>}
      </div>
    </div>
  )
}
```

2.9 src/components/ProtectedRoute.jsx

```
import { Navigate } from 'react-router-dom'
import { getUser } from '../auth'

export default function ProtectedRoute({ children, roles }){
  const user = getUser();
  if (!user) return <Navigate to="/login" />
  if (roles && !roles.includes(user.role)) return <Navigate to="/" />
  return children
}
```

2.10 src/components/SearchBox.jsx

```
export default function SearchBox({ value, onChange, placeholder='Search by
name or address' }){
  return <input className="input" value={value}
onChange={e=>onChange(e.target.value)} placeholder={placeholder} />
}
```

2.11 src/components/SortableHeader.jsx

```
export default function SortableHeader({ label, field, sort, order, onSort
}){
  const active = sort === field;
  const dir = active ? (order==='asc'?'↑':'↓') : '';
  return <th style={{cursor:'pointer'}} onClick={()=>onSort(field, active
&& order==='asc'?'desc':'asc')}>{label} {dir}</th>
}
```

2.12 src/components/StarRating.jsx

```
export default function StarRating({ value=0, onChange }){
  return (
    <div className="flex">
      {[1,2,3,4,5].map(n => (
        <button key={n} className={`btn ${n<=value?'primary':''}`}
onClick={()=>onChange?.(n)}>{n}</button>
      ))}
    </div>
  )
}
```

2.13 src/pages/Login.jsx

```
import { useState } from 'react'
import { api } from '../api'
import { setToken, setUser } from '../auth'
import { Link, useNavigate } from 'react-router-dom'

export default function Login(){
  const [email, setEmail] = useState('admin@example.com')
  const [password, setPassword] = useState('Admin@123')
  const [error, setError] = useState('')
  const navigate = useNavigate()
  async function submit(e){
    e.preventDefault(); setError('')
    try{
      const { token, user } = await api('/auth/login', { method:'POST',
body: { email, password } })
      setToken(token); setUser(user)
      navigate('/')
    }catch(err){ setError(err.message) }
  }
  return (
    <div className="container">
      <div className="card">
        <h2>Login</h2>
        {error && <p style={{color:'crimson'}}>{error}</p>}
        <form onSubmit={submit} className="grid">
```

```
          <input className="input" placeholder="Email" value={email}
onChange={e=>setEmail(e.target.value)} />
          <input className="input" type="password" placeholder="Password"
value={password} onChange={e=>setPassword(e.target.value)} />
          <button className="btn primary">Login</button>
        </form>
        <p>New here? <Link to="/signup">Create an account</Link></p>
      </div>
    </div>
  )
}
```

2.14 src/pages/Signup.jsx

```
import { useState } from 'react'
import { api } from '../api'
import { setToken, setUser } from '../auth'
import { useNavigate } from 'react-router-dom'

export default function Signup(){
  const [name, setName] = useState('')
  const [email, setEmail] = useState('')
  const [address, setAddress] = useState('')
  const [password, setPassword] = useState('')
  const [error, setError] = useState('')
  const navigate = useNavigate()

  async function submit(e){
    e.preventDefault(); setError('')
    try{
      const { token, user } = await api('/auth/signup', { method:'POST',
body: { name, email, address, password } })
      setToken(token); setUser(user)
      navigate('/')
    }catch(err){ setError(err.message) }
  }

  return (
    <div className="container">
      <div className="card">
        <h2>Sign Up</h2>
        <p style={{fontSize:12}}>Name must be 20-60 chars; Password 8-16
with uppercase + special character.</p>
        {error && <p style={{color:'crimson'}}>{error}</p>}
        <form onSubmit={submit} className="grid">
          <input className="input" placeholder="Full Name" value={name}
onChange={e=>setName(e.target.value)} />
          <input className="input" placeholder="Email" value={email}
onChange={e=>setEmail(e.target.value)} />
          <input className="input" placeholder="Address" value={address}
onChange={e=>setAddress(e.target.value)} />
          <input className="input" type="password" placeholder="Password"
value={password} onChange={e=>setPassword(e.target.value)} />
          <button className="btn primary">Create Account</button>
        </form>
      </div>
    </div>
  )
}
```

2.15 src/pages/AdminDashboard.jsx

```
import { useEffect, useState } from 'react'
import { api } from '../api'
import { getToken } from '../auth'
import UsersTable from './UsersTable'
import StoresTable from './StoresTable'

export default function AdminDashboard(){
  const [stats, setStats] = useState(null)
  const token = getToken()
  useEffect(()=>{ api('/admin/stats', { token }).then(setStats) },[])
  return (
    <div className="container">
      <div className="grid grid-3">
        <div className="card"><h3>Total Users</h3><h1>{stats?.totalUsers ??
'—'}</h1></div>
        <div className="card"><h3>Total Stores</h3><h1>{stats?.totalStores
?? '—'}</h1></div>
        <div className="card"><h3>Total
Ratings</h3><h1>{stats?.totalRatings ?? '—'}</h1></div>
      </div>
      <UsersTable />
      <StoresTable />
    </div>
  )
}
```

2.16 src/pages/UsersTable.jsx

```
import { useEffect, useState } from 'react'
import { api } from '../api'
import { getToken } from '../auth'
import SearchBox from '../components/SearchBox'
import SortableHeader from '../components/SortableHeader'

export default function UsersTable(){
  const token = getToken()
  const [q, setQ] = useState('')
  const [sort, setSort] = useState('name')
  const [order, setOrder] = useState('asc')
  const [rows, setRows] = useState([])

  async function load(){
    const data = await
api(`/admin/users?q=${encodeURIComponent(q)}&sort=${sort}&order=${order}&li
mit=50`,{ token })
    setRows(data)
  }
  useEffect(()=>{ load() },[q,sort,order])

  return (
    <div className="card">
      <h3>Users</h3>
      <div className="flex" style={{margin:'8px 0'}}>
        <SearchBox value={q} onChange={setQ} placeholder="Search name,
email, address" />
      </div>
      <table className="table">
        <thead>
```

```
          <tr>
            <SortableHeader label="Name" field="name" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
            <SortableHeader label="Email" field="email" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
            <SortableHeader label="Address" field="address" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
            <SortableHeader label="Role" field="role" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
          </tr>
        </thead>
        <tbody>
          {rows.map(r => (
            <tr key={r.id}>
              <td>{r.name}</td>
              <td>{r.email}</td>
              <td>{r.address}</td>
              <td><span className="badge">{r.role}</span></td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  )
}
```

2.17 src/pages/StoresTable.jsx

```
import { useEffect, useState } from 'react'
import { api } from '../api'
import { getToken } from '../auth'
import SearchBox from '../components/SearchBox'
import SortableHeader from '../components/SortableHeader'

export default function StoresTable(){
  const token = getToken()
  const [q, setQ] = useState('')
  const [sort, setSort] = useState('name')
  const [order, setOrder] = useState('asc')
  const [rows, setRows] = useState([])

  async function load(){
    const data = await
api(`/admin/stores?q=${encodeURIComponent(q)}&sort=${sort}&order=${order}&l
imit=50`,{ token })
    setRows(data)
  }
  useEffect(()=>{ load() },[q,sort,order])

  return (
    <div className="card">
      <h3>Stores</h3>
      <div className="flex" style={{margin:'8px 0'}}>
        <SearchBox value={q} onChange={setQ} placeholder="Search name,
email, address" />
      </div>
      <table className="table">
        <thead>
          <tr>
```

```
            <SortableHeader label="Name" field="name" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
            <SortableHeader label="Email" field="email" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
            <SortableHeader label="Address" field="address" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
            <SortableHeader label="Rating" field="avg_rating" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
          </tr>
        </thead>
        <tbody>
          {rows.map(r => (
            <tr key={r.id}>
              <td>{r.name}</td>
              <td>{r.email}</td>
              <td>{r.address}</td>
              <td>{Number(r.rating).toFixed(2)} ({r.ratings_count})</td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  )
}
```

2.18 src/pages/UserStores.jsx

```
import { useEffect, useState } from 'react'
import { api } from '../api'
import { getToken } from '../auth'
import SearchBox from '../components/SearchBox'
import SortableHeader from '../components/SortableHeader'
import StarRating from '../components/StarRating'

export default function UserStores(){
  const token = getToken()
  const [q, setQ] = useState('')
  const [sort, setSort] = useState('name')
  const [order, setOrder] = useState('asc')
  const [rows, setRows] = useState([])

  async function load(){
    const data = await
api(`/stores?q=${encodeURIComponent(q)}&sort=${sort}&order=${order}&limit=5
0`,{ token })
    setRows(data)
  }
  useEffect(()=>{ load() },[q,sort,order])

  async function rate(id, rating){
    await api(`/stores/${id}/ratings`, { method:'POST', token, body: {
rating } })
    await load()
  }

  return (
    <div className="container">
      <div className="card">
        <h3>All Stores</h3>
        <div className="flex" style={{margin:'8px 0'}}>
```

```
          <SearchBox value={q} onChange={setQ} />
        </div>
        <table className="table">
          <thead>
            <tr>
              <SortableHeader label="Store" field="name" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
              <SortableHeader label="Address" field="address" sort={sort}
order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
              <SortableHeader label="Overall Rating" field="avg_rating"
sort={sort} order={order} onSort={(f,o)=>{setSort(f);setOrder(o)}} />
              <th>My Rating</th>
              <th>Rate</th>
            </tr>
          </thead>
          <tbody>
            {rows.map(r => (
              <tr key={r.id}>
                <td>{r.name}</td>
                <td>{r.address}</td>
                <td>{Number(r.overall_rating).toFixed(2)}</td>
                <td>{r.my_rating ?? '—'}</td>
                <td><StarRating value={r.my_rating || 0}
onChange={(val)=>rate(r.id, val)} /></td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  )
}
```

2.19 src/pages/OwnerDashboard.jsx

```
import { useEffect, useState } from 'react'
import { api } from '../api'
import { getToken } from '../auth'

export default function OwnerDashboard(){
  const token = getToken()
  const [rows, setRows] = useState([])
  useEffect(()=>{ api('/owner/ratings',{ token }).then(setRows) },[])
  const avg = rows[0]?.avg_rating ? Number(rows[0].avg_rating).toFixed(2) :
'—'
  const storeName = rows[0]?.store_name || '—'
  return (
    <div className="container">
      <div className="grid grid-2">
        <div className="card"><h3>Store</h3><h1>{storeName}</h1></div>
        <div className="card"><h3>Average Rating</h3><h1>{avg}</h1></div>
      </div>
      <div className="card">
        <h3>Recent Ratings</h3>
        <table className="table">
          <thead>

<tr><th>User</th><th>Email</th><th>Rating</th><th>Updated</th></tr>
          </thead>
          <tbody>
```

```
            {rows.map((r,i)=> (
              <tr
key={i}><td>{r.user_name}</td><td>{r.user_email}</td><td>{r.rating}</td><td
>{new Date(r.updated_at).toLocaleString()}</td></tr>
            ))}
          </tbody>
        </table>
      </div>
    </div>
  )
}
```

2.20 src/pages/ChangePassword.jsx

```
import { useState } from 'react'
import { api } from '../api'
import { getToken } from '../auth'

export default function ChangePassword(){
  const token = getToken()
  const [pw, setPw] = useState('')
  const [msg, setMsg] = useState('')
  async function submit(e){
    e.preventDefault(); setMsg('')
    try{ await api('/auth/change-password', { method:'POST', token, body: {
newPassword: pw } }); setMsg('Password updated') } catch(e){
setMsg(e.message) }
  }
  return (
    <div className="container">
      <div className="card">
        <h3>Change Password</h3>
        <form onSubmit={submit} className="grid">
          <input className="input" placeholder="New Password"
type="password" value={pw} onChange={e=>setPw(e.target.value)} />
          <button className="btn primary">Update</button>
        </form>
        {msg && <p>{msg}</p>}
      </div>
    </div>
  )
}
```

2.21 src/App.jsx

```
import { Routes, Route, Link } from 'react-router-dom'
import NavBar from './components/NavBar'
import Login from './pages/Login'
import Signup from './pages/Signup'
import AdminDashboard from './pages/AdminDashboard'
import UserStores from './pages/UserStores'
import OwnerDashboard from './pages/OwnerDashboard'
import ChangePassword from './pages/ChangePassword'
import ProtectedRoute from './components/ProtectedRoute'

function Home(){
  return (
    <div className="container">
      <div className="card">
```

```
        <h2>Store Ratings Platform</h2>
        <p>Rate stores (1–5), view averages, and manage entities with role-
based access.</p>
        <ul>
          <li><b>Admin:</b> Add users & stores, view dashboards & lists
with sorting/filtering.</li>
          <li><b>User:</b> Browse stores, submit/modify ratings, search by
name/address.</li>
          <li><b>Owner:</b> See recent ratings and average for owned
store(s).</li>
        </ul>
      </div>
    </div>
  )
}

export default function App(){
  return (
    <>
      <NavBar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signup" element={<Signup />} />
        <Route path="/stores" element={<ProtectedRoute><UserStores
/></ProtectedRoute>} />
        <Route path="/admin" element={<ProtectedRoute
roles={['ADMIN']}><AdminDashboard /></ProtectedRoute>} />
        <Route path="/owner" element={<ProtectedRoute
roles={['OWNER']}><OwnerDashboard /></ProtectedRoute>} />
        <Route path="/change-password"
element={<ProtectedRoute><ChangePassword /></ProtectedRoute>} />
      </Routes>
    </>
  )
}
```

3) Server README (usage)

```
# Ratings App — Server

## Prerequisites
- Node.js 18+
- PostgreSQL 13+

## Setup
1. Copy `.env.example` to `.env` and adjust values.
2. Install deps: `npm i`
3. Create DB: `createdb ratings_app` (or adjust `DATABASE_URL`).
4. Run schema & seed: `npm run db:setup`
5. Start server: `npm run dev` (default http://localhost:5000)

### Test users
- Admin: admin@example.com / Admin@123
- Owner: owner@example.com / Owner@123
- User:  user@example.com  / User@1234
```

4) Client README (usage)

```
# Ratings App — Client

## Setup
1. Install deps: `npm i`
2. Create `.env` with `VITE_API=http://localhost:5000/api` (optional;
defaults to this).
3. Start: `npm run dev` (http://localhost:5173)
```

5) Notes on Non-AI, Human-Crafted Delivery

- Clear, purposeful commit messages (avoid generic phrases). Example commits:
  - `feat(api): rating upsert with unique (user,store)`
  - `feat(ui): sortable headers and debounced search`
  - `chore(db): add store_avg view for fast aggregates`
- A concise, candid `README` that explains trade-offs (e.g., JWT in localStorage for simplicity; can be switched to cookies).
- Input validation enforced both client and server with explicit error messages.
- Database constraints (CHECK, UNIQUE) backstop correctness beyond code.

.