To organize your backend into a clean and scalable structure, it's best to follow a modular folder-based architecture like this:-

---

□ Recommended Project Structure

```
backend/
│
├── config/
│   └── db.js              # MongoDB connection setup
│
├── controllers/
│   ├── authController.js # Signup, login logic
│   └── recordController.js # CRUD logic for records
│
├── middleware/
│   └── auth.js            # JWT verification middleware
│
├── models/
│   ├── Admin.js           # Admin schema
│   └── Record.js          # Record schema
│
├── routes/
│   ├── authRoutes.js      # Routes for /signup and /login
│   └── recordRoutes.js    # Routes for /records
│
├── .env                   # JWT secret and DB URI
├── server.js              # Main entry point
└── package.json
```

□ 1. Initialize your backend project

```
mkdir backend
cd backend
npm init -y
```

□ 2. Install required dependencies

```
npm install express mongoose cors bcryptjs jsonwebtoken
```

□ Step-by-Step Refactoring

---

1. `config/db.js`

```javascript
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect('mongodb://localhost:27017/recordsDB', {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (err) {
    console.error('MongoDB connection failed:', err.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

2. `models/Admin.js`

```javascript
const mongoose = require('mongoose');

const adminSchema = new mongoose.Schema({
  username: String,
  password: String,
});

module.exports = mongoose.model('Admin', adminSchema);
```

---

3. models/Record.js

```
const mongoose = require('mongoose');

const recordSchema = new mongoose.Schema({
  name: String,
  email: String,
  phone: String,
  city: String,
});

module.exports = mongoose.model('Record', recordSchema);
```

4. middleware/auth.js

```
// middleware/auth.js
const jwt = require('jsonwebtoken');
const JWT_SECRET = process.env.JWT_SECRET || 'yourSecret';

function verifyToken(req, res, next) {
  const token = req.headers['authorization'];
  if (!token) return res.status(401).json({ message: 'No token provided' });

  jwt.verify(token, JWT_SECRET, (err, decoded) => {
    if (err) return res.status(403).json({ message: 'Invalid token' });
    req.adminId = decoded.id;
    next();
  });
}

module.exports = verifyToken; // ✅ Export the function directly
```

5. controllers/authController.js

```
const Admin = require('../models/Admin');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const JWT_SECRET = process.env.JWT_SECRET;

exports.signup = async (req, res) => {
  try {
    const { username, password } = req.body;
    if (!username || !password) return res.status(400).json({ message:
'Username and password are required' });

    const existing = await Admin.findOne({ username });
    if (existing) return res.status(400).json({ message: 'Username already
exists' });

    const hashed = await bcrypt.hash(password, 10);
    await Admin.create({ username, password: hashed });

    res.json({ message: 'Admin created' });
  } catch (err) {
    console.error('Signup error:', err);
    res.status(500).json({ message: 'Internal server error' });
  }
};

exports.login = async (req, res) => {
  try {
    const { username, password } = req.body;
    const admin = await Admin.findOne({ username });

    if (!admin) return res.status(404).json({ message: 'Admin not found' });

    const isMatch = await bcrypt.compare(password, admin.password);
    if (!isMatch) return res.status(401).json({ message: 'Invalid credentials'
});

    const token = jwt.sign({ id: admin._id }, JWT_SECRET, { expiresIn: '1h'
});
    res.json({ token });
  } catch (err) {
    res.status(500).json({ message: 'Login error' });
  }
};
```

6. controllers/recordController.js

```
const Record = require('../models/Record');

exports.getAllRecords = async (req, res) => {
  const records = await Record.find();
  res.json(records);
};

exports.getRecordById = async (req, res) => {
  try {
    const record = await Record.findById(req.params.id);
    if (!record) return res.status(404).json({ message: 'Record not found' });
    res.json(record);
  } catch (err) {
    res.status(400).json({ message: 'Invalid record ID' });
  }
};

exports.createRecord = async (req, res) => {
  try {
    const newRecord = new Record(req.body);
    await newRecord.save();
    res.json({ message: 'Record added', record: newRecord });
  } catch (err) {
    res.status(500).json({ message: 'Error creating record' });
  }
};

exports.updateRecord = async (req, res) => {
  try {
    const updated = await Record.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
      runValidators: true,
    });
    if (!updated) return res.status(404).json({ message: 'Record not found'
});
    res.json({ message: 'Record updated', record: updated });
  } catch (err) {
    res.status(400).json({ message: 'Invalid update request' });
  }
};

exports.deleteRecord = async (req, res) => {
  try {
    const deleted = await Record.findByIdAndDelete(req.params.id);
    if (!deleted) return res.status(404).json({ message: 'Record not found'
});
```

```
    res.json({ message: 'Record deleted' });
  } catch (err) {
    res.status(400).json({ message: 'Invalid record ID' });
  }
};
```

---

7. `routes/authRoutes.js`

```
const express = require('express');
const router = express.Router();
const { signup, login } = require('../controllers/authController');

router.post('/signup', signup);
router.post('/login', login);

module.exports = router;
```

---

8. `routes/recordRoutes.js`

```
const express = require('express');
const router = express.Router();
const verifyToken = require('../middleware/auth'); // ⍰ now a function

const {
  getAllRecords,
  getRecordById,
  createRecord,
  updateRecord,
  deleteRecord,
} = require('../controllers/recordController');

router.get('/', verifyToken, getAllRecords);
router.get('/:id', verifyToken, getRecordById);
router.post('/', verifyToken, createRecord);
router.put('/:id', verifyToken, updateRecord);
router.delete('/:id', verifyToken, deleteRecord);

module.exports = router;
```

---

9. `.env`

```
JWT_SECRET=a1b2c3d4e5f60123456789abcdef0123456789abcdef0123456789abcdef0123
```

□ Install `dotenv` in your project:

```
npm install dotenv
```

---

10. `server.js` (Main Entry Point)

```
const express = require('express');
const cors = require('cors');
const connectDB = require('./config/db');
require('dotenv').config();

const app = express();
app.use(cors());
app.use(express.json());

// Connect DB
connectDB();

// Routes
app.use('/', require('./routes/authRoutes'));
 // reuse same router
app.use('/records', require('./routes/recordRoutes'));

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

To match your **clean backend structure**, here's how to organize your **React frontend** for **maintainability**, **scalability**, and **clarity**.

□ Recommended Frontend Folder Structure

```
frontend/
├── public/
│   └── index.html
│
├── src/
│   ├── api/              # Axios API helpers
│   │   ├── auth.js
│   │   └── records.js
│   │
│   ├── components/           # Reusable components
│   │   ├── Auth/
│   │   │   ├── Login.js
│   │   │   └── Signup.js
│   │   ├── Records/
│   │   │   ├── RecordForm.js
│   │   │   └── RecordList.js
│   │
│   ├── context/              # Auth context
│   │   └── AuthContext.js
│   │
│   ├── pages/                # Main views
│   │   └── Dashboard.js
│   │
│   ├── App.js
│   ├── index.js
│   └── styles/               # (optional) CSS or SCSS
│       └── app.css
│
├── .env
└── package.json
```

---

1. Create the React App

You can use **Vite (recommended)** or **Create React App**.

Option A: Create React App (easiest)

```
npx create-react-app frontend
cd frontend
```

Option B: Vite (faster build, modern)

```
npm create vite@latest frontend
cd frontend
npm install
```

2. Install Required Dependencies

From inside the `frontend/` folder:

```
npm install axios
```

□ Step-by-Step Breakdown

1. `api/auth.js`

```
import axios from 'axios';

const API = process.env.REACT_APP_API || 'http://localhost:5000';

export const login = (data) => axios.post(`${API}/login`, data);
export const signup = (data) => axios.post(`${API}/signup`, data);
```

2. `api/records.js`

```
import axios from 'axios';

const API = process.env.REACT_APP_API || 'http://localhost:5000';

export const getRecords = (token) =>
  axios.get(`${API}/records`, {
    headers: { Authorization: token },
  });

export const createRecord = (data, token) =>
  axios.post(`${API}/records`, data, {
    headers: { Authorization: token },
  });

export const updateRecord = (id, data, token) =>
  axios.put(`${API}/records/${id}`, data, {
    headers: { Authorization: token },
  });

export const deleteRecord = (id, token) =>
  axios.delete(`${API}/records/${id}`, {
    headers: { Authorization: token },
  });
```

---

3. `context/AuthContext.js`

```
import { createContext, useState, useEffect } from 'react';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [token, setToken] = useState(localStorage.getItem('token') || '');

  const login = (newToken) => {
    setToken(newToken);
    localStorage.setItem('token', newToken);
  };

  const logout = () => {
    setToken('');
    localStorage.removeItem('token');
  };

  useEffect(() => {
    const storedToken = localStorage.getItem('token');
    if (storedToken) setToken(storedToken);
  }, []);

  return (
    <AuthContext.Provider value={{ token, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

---

4. `components/Auth/Login.js`

```
import React, { useState, useContext } from 'react';
import { login as loginAPI } from '../../api/auth';
import { AuthContext } from '../../context/AuthContext';

function Login({ switchToSignup }) {
  const [form, setForm] = useState({ username: '', password: '' });
  const { login } = useContext(AuthContext);

  const handleSubmit = async () => {
    try {
      const res = await loginAPI(form);
      login(res.data.token);
    } catch (err) {
      alert('Login failed');
    }
  };

  return (
    <div>
      <h2>Admin Login</h2>
      <input placeholder="Username" value={form.username} onChange={(e) =>
setForm({ ...form, username: e.target.value })} />
      <input type="password" placeholder="Password" value={form.password}
onChange={(e) => setForm({ ...form, password: e.target.value })} />
      <button onClick={handleSubmit}>Login</button>
      <p>
        Don't have an account? <button onClick={switchToSignup}>Sign
Up</button>
      </p>
    </div>
  );
}

export default Login;
```

---

5. **components/Auth/Signup.js**

```
import React, { useState } from 'react';
import { signup as signupAPI } from '../../api/auth';

function Signup({ onSignupSuccess }) {
  const [form, setForm] = useState({ username: '', password: '' });

  const handleSignup = async () => {
    try {
      await signupAPI(form);
      onSignupSuccess();
    } catch (err) {
      alert('Signup failed');
    }
  };

  return (
    <div>
      <h2>Sign Up</h2>
      <input placeholder="Username" value={form.username} onChange={(e) =>
setForm({ ...form, username: e.target.value })} />
      <input type="password" placeholder="Password" value={form.password}
onChange={(e) => setForm({ ...form, password: e.target.value })} />
      <button onClick={handleSignup}>Sign Up</button>
    </div>
  );
}

export default Signup;
```

---

6. **components/Records/RecordForm.js**

```
import React from 'react';

function RecordForm({ form, setForm, handleSubmit, isEditing, cancelEdit })
{
  return (
    <form onSubmit={handleSubmit}>
      <input placeholder="Name" value={form.name} onChange={(e) =>
setForm({ ...form, name: e.target.value })} />
      <input placeholder="Email" value={form.email} onChange={(e) =>
setForm({ ...form, email: e.target.value })} />
      <input placeholder="Phone" value={form.phone} onChange={(e) =>
setForm({ ...form, phone: e.target.value })} />
      <input placeholder="City" value={form.city} onChange={(e) =>
setForm({ ...form, city: e.target.value })} />
```

```
      <button type="submit">{isEditing ? 'Update' : 'Add'}</button>
      {isEditing && <button onClick={cancelEdit}>Cancel</button>}
    </form>
  );
}

export default RecordForm;
```

---

7. `components/Records/RecordList.js`

```
import React from 'react';

function RecordList({ records, onEdit, onDelete }) {
  return (
    <ul>
      {records.map((rec) => (
        <li key={rec._id}>
          {rec.name} | {rec.email} | {rec.phone} | {rec.city}
          <button onClick={() => onEdit(rec)}>Edit</button>
          <button onClick={() => onDelete(rec._id)}>Delete</button>
        </li>
      ))}
    </ul>
  );
}

export default RecordList;
```

---

8. `pages/Dashboard.js`

```
import React, { useState, useEffect, useContext } from 'react';
import RecordForm from '../components/Records/RecordForm';
import RecordList from '../components/Records/RecordList';
import { getRecords, createRecord, updateRecord, deleteRecord } from
'../api/records';
import { AuthContext } from '../context/AuthContext';

function Dashboard() {
  const { token, logout } = useContext(AuthContext);
  const [form, setForm] = useState({ name: '', email: '', phone: '', city:
'' });
  const [records, setRecords] = useState([]);
  const [editId, setEditId] = useState(null);

  const fetch = async () => {
    const res = await getRecords(token);
    setRecords(res.data);
  };

  useEffect(() => {
    fetch();
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      if (editId) {
        await updateRecord(editId, form, token);
      } else {
        await createRecord(form, token);
      }
      setForm({ name: '', email: '', phone: '', city: '' });
      setEditId(null);
      fetch();
    } catch {
      alert('Error submitting record');
    }
  };

  const handleEdit = (rec) => {
    setEditId(rec._id);
    setForm(rec);
  };

  const handleDelete = async (id) => {
    if (window.confirm('Are you sure?')) {
      await deleteRecord(id, token);
      fetch();
    }
  };

  return (
    <div style={{ padding: '20px' }}>
      <button onClick={logout}>Logout</button>
      <h2>{editId ? 'Edit' : 'Add'} Record</h2>
      <RecordForm form={form} setForm={setForm} handleSubmit={handleSubmit}
isEditing={!!editId} cancelEdit={() => { setEditId(null); setForm({ name:
'', email: '', phone: '', city: '' }); }} />
      <h2>Records</h2>
      <RecordList records={records} onEdit={handleEdit}
onDelete={handleDelete} />
    </div>
  );
}

export default Dashboard;
```

9. `App.js`

```
import React, { useContext, useState } from 'react';
import Login from './components/Auth/Login';
import Signup from './components/Auth/Signup';
import Dashboard from './pages/Dashboard';
import { AuthProvider, AuthContext } from './context/AuthContext';

function MainApp() {
  const { token } = useContext(AuthContext);
  const [showSignup, setShowSignup] = useState(false);

  const handleSignupSuccess = () => {
    setShowSignup(false);
    alert('Signup successful! You can now log in.');
  };

  if (!token) {
    return showSignup ? (
      <Signup onSignupSuccess={handleSignupSuccess} />
    ) : (
      <Login switchToSignup={() => setShowSignup(true)} />
    );
  }

  return <Dashboard />;
}


function App() {
  return (
    <AuthProvider>
      <MainApp />
    </AuthProvider>
  );
}

export default App;
```

10. `.env`

```
REACT_APP_API=http://localhost:5000
```

☐ Make sure to restart your frontend server after changing `.env`.

□ Run the App

Start Backend

```
cd backend
node server.js
```

Start Frontend

```
cd frontend
npm install
npm run dev # or npm start for Create React App
```

□ Summary

Layer	Structure
API Layer	`/api` (auth, records)
Components	Modular by purpose (Auth, Records)
Context	Auth management
Pages	Dashboard view
State	Local + Context
Styling	Optional CSS in `/styles`