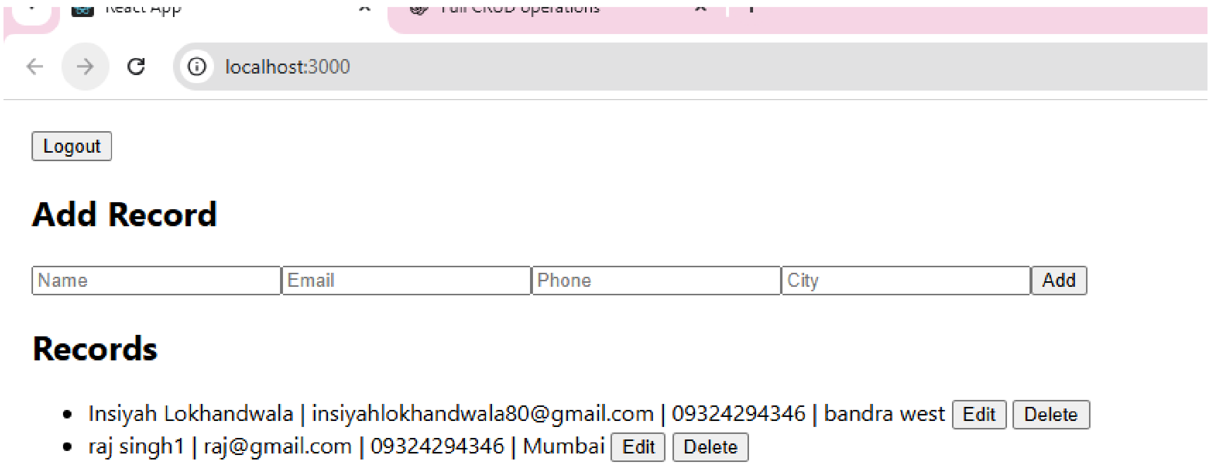
Admin singup,login,logut and manage crud operations :-



Add Record

Name | Email | Phone | City | Add

Records

- Insiyah Lokhandwala | insiyahlokhandwala80@gmail.com | 09324294346 | bandra west Edit Delete
- raj singh1 | raj@gmail.com | 09324294346 | Mumbai Edit Delete

To make it a **complete CRUD backend**, you just need to add the **Update** and **Delete** operations.

I'll expand your existing `/records` route handlers to include:

1. `GET /records` – Read all records *(Already Present)*
2. `POST /records` – Create a new record *(Already Present)*
3. `GET /records/:id` – Read a single record by ID *(Add This)*
4. `PUT /records/:id` – Update a record by ID *(Add This)*
5. `DELETE /records/:id` – Delete a record by ID *(Add This)*

☐ Updated Full CRUD Code

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

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

```
const JWT_SECRET =
'a1b2c3d4e5f60123456789abcdef0123456789abcdef0123456789abcdef0123'; // Change
this in production

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/recordsDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true
});

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

// Records Schema
const recordSchema = new mongoose.Schema({
  name: String,
  email: String,
  phone: String,
  city: String
});
const Record = mongoose.model('Record', recordSchema);

// Middleware to verify JWT token
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();
  });
}

// Signup (Create Admin Account)
app.post('/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('Error in signup:', err);
    res.status(500).json({ message: 'Internal server error' });
  }
});

// Login
app.post('/login', async (req, res) => {
  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 });
});

// -------------------- CRUD Routes for Records -------------------- //

// READ all records
app.get('/records', verifyToken, async (req, res) => {
  const records = await Record.find();
  res.json(records);
});

// READ a single record by ID
app.get('/records/:id', verifyToken, 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' });
  }
});

// CREATE a new record
app.post('/records', verifyToken, 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' });
  }
});

// UPDATE a record by ID
app.put('/records/:id', verifyToken, async (req, res) => {
  try {
    const updatedRecord = await Record.findByIdAndUpdate(req.params.id,
req.body, {
      new: true,
      runValidators: true,
    });
    if (!updatedRecord) return res.status(404).json({ message: 'Record not
found' });
    res.json({ message: 'Record updated', record: updatedRecord });
  } catch (err) {
    res.status(400).json({ message: 'Invalid update request' });
  }
});

// DELETE a record by ID
app.delete('/records/:id', verifyToken, 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' });
  }
});

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

Summary of API Endpoints

Method	Endpoint	Description
POST	`/signup`	Create admin account
POST	`/login`	Admin login, returns token
GET	`/records`	Get all records
GET	`/records/:id`	Get a specific record
POST	`/records`	Create a new record
PUT	`/records/:id`	Update an existing record
DELETE	`/records/:id`	Delete a record

Below, I'll give you the **modified `App.js`** code that includes:

- **Edit mode toggle**
- **Prefilled form on edit**
- **PUT request to update**
- **DELETE request to remove**

Src/Signup.js file code:-

```
import React, { useState } from 'react';
import axios from 'axios';

const API = 'http://localhost:5000';

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

  const handleChange = (e) => {
    setForm({ ...form, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await axios.post(`${API}/signup`, form, {
        headers: { 'Content-Type': 'application/json' }
      });
      setMessage(res.data.message || 'Signup successful');
      setForm({ username: '', password: '' });
      if (onSignupSuccess) onSignupSuccess();
    } catch (err) {
      console.error('Signup error:', err);

      if (err.response) {
        // server responded with a status outside 2xx
        setMessage(`Signup failed: ${err.response.data.message ||
err.response.statusText}`);
      } else if (err.request) {
        // request was made but no response
        setMessage('Signup failed: No response from server');
      } else {
        // other errors
        setMessage('Signup failed: ' + err.message);
      }
    }
  };

  return (
    <div style={{ padding: '20px' }}>
      <h2>Admin Signup</h2>
      <form onSubmit={handleSubmit}>
        <input
          name="username"
          placeholder="Username"
          value={form.username}
          onChange={handleChange}
          required
        />
        <input
          name="password"
          type="password"
          placeholder="Password"
          value={form.password}
          onChange={handleChange}
          required
        />
        <button type="submit">Sign Up</button>
      </form>
      {message && <p>{message}</p>}
    </div>
  );
```

```
}

export default Signup;
```

□ Updated `App.js` with Full CRUD

Replace your current `App.js` with this:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import Signup from './Signup';

const API = 'http://localhost:5000';

function App() {
  const [form, setForm] = useState({ name: '', email: '', phone: '', city: ''
});
  const [records, setRecords] = useState([]);
  const [auth, setAuth] = useState({ username: '', password: '' });
  const [token, setToken] = useState(localStorage.getItem('token') || '');
  const [showSignup, setShowSignup] = useState(false);
  const [editId, setEditId] = useState(null); // ID of record being edited

  const headers = { Authorization: token };

  const handleLogin = async () => {
    try {
      const res = await axios.post(`${API}/login`, auth);
      localStorage.setItem('token', res.data.token);
      setToken(res.data.token);
    } catch (err) {
      alert('Login failed');
    }
  };

  const handleLogout = () => {
    localStorage.removeItem('token');
    setToken('');
    setRecords([]);
  };

  const fetchRecords = async () => {
    try {
      const res = await axios.get(`${API}/records`, { headers });
      setRecords(res.data);
    } catch (err) {
      console.error(err);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (editId) {
      // UPDATE record
      try {
        await axios.put(`${API}/records/${editId}`, form, { headers });
        setEditId(null);
        setForm({ name: '', email: '', phone: '', city: '' });
        fetchRecords();
      } catch (err) {
        alert('Failed to update record');
      }
    } else {
      // CREATE record
      try {
        await axios.post(`${API}/records`, form, { headers });
        setForm({ name: '', email: '', phone: '', city: '' });
        fetchRecords();
      } catch (err) {
        alert('Failed to add record');
      }
    }
  };

  const handleEdit = (record) => {
    setEditId(record._id);
    setForm({ name: record.name, email: record.email, phone: record.phone,
city: record.city });
  };

  const handleDelete = async (id) => {
    if (!window.confirm('Are you sure you want to delete this record?'))
return;
    try {
      await axios.delete(`${API}/records/${id}`, { headers });
      fetchRecords();
    } catch (err) {
      alert('Failed to delete record');
    }
  };

  useEffect(() => {
    if (token) fetchRecords();
  }, [token]);

  // □ Not logged in
  if (!token) {
    return (
      <div style={{ padding: '20px' }}>
        {showSignup ? (
          <>
            <Signup
              onSignupSuccess={() => {
                setShowSignup(false);
                alert('Signup successful! You can now log in.');
              }}
            />
            <p>
              Already have an account?{' '}
              <button onClick={() => setShowSignup(false)}>Login</button>
            </p>
          </>
        ) : (
          <>
            <h2>Admin Login</h2>
            <input
              placeholder="Username"
              value={auth.username}
              onChange={(e) => setAuth({ ...auth, username: e.target.value })}
            />
            <input
              placeholder="Password"
              type="password"
              value={auth.password}
              onChange={(e) => setAuth({ ...auth, password: e.target.value })}
            />
            <button onClick={handleLogin}>Login</button>
            <p>
              Don't have an account?{' '}
              <button onClick={() => setShowSignup(true)}>Sign Up</button>
            </p>
          </>
        )}
      </div>
    );
  }

  // □ Logged in
  return (
    <div style={{ padding: '20px' }}>
```

```
      <button onClick={handleLogout}>Logout</button>
      <h2>{editId ? 'Edit Record' : 'Add Record'}</h2>
      <form onSubmit={handleSubmit}>
        <input
          name="name"
          placeholder="Name"
          value={form.name}
          onChange={(e) => setForm({ ...form, name: e.target.value })}
        />
        <input
          name="email"
          placeholder="Email"
          value={form.email}
          onChange={(e) => setForm({ ...form, email: e.target.value })}
        />
        <input
          name="phone"
          placeholder="Phone"
          value={form.phone}
          onChange={(e) => setForm({ ...form, phone: e.target.value })}
        />
        <input
          name="city"
          placeholder="City"
          value={form.city}
          onChange={(e) => setForm({ ...form, city: e.target.value })}
        />
        <button type="submit">{editId ? 'Update' : 'Add'}</button>
        {editId && <button onClick={() => { setEditId(null); setForm({ name:
'', email: '', phone: '', city: '' }); }}>Cancel</button>}
      </form>

      <h2>Records</h2>
      <ul>
        {records.map((rec) => (
          <li key={rec._id}>
            {rec.name} | {rec.email} | {rec.phone} | {rec.city}{' '}
            <button onClick={() => handleEdit(rec)}>Edit</button>{' '}
            <button onClick={() => handleDelete(rec._id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

□ Optional Improvements

- Sort or paginate records
- Use a table instead of a list
- Add form validation (with libraries like Formik or Yup)
- Use a UI framework (e.g., Material UI or Bootstrap)

□ Summary of Features in This Code:

Feature	□ Implemented
Admin Signup	□ via `<Signup />`
Admin Login	□ With JWT
Create Record	□
Read Records	□
Update Record	□
Delete Record	□
Logout	□
Token Storage	□ localStorage