Let’s put everything together—**frontend API routes using Axios** and the **backend steps** to support them—so you have a full working setup for CRUD operations. I’ll break it into clear steps.

□ Full CRUD Setup with Backend & Frontend API Routes

Step 1: Setup Backend

We’ll use **Express.js** as an example backend running on `http://localhost:5000`.

1.1 Install Dependencies

```
npm init -y
npm install express cors body-parser
```

1.2 Create `server.js`

```
import express from 'express';

import cors from 'cors';

import mongoose from 'mongoose';



const app = express();

const PORT = 5000;



// MongoDB connection

const MONGO_URI = 'mongodb://127.0.0.1:27017/usersdb'; // replace with your
URI

mongoose.connect(MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true
})

  .then(() => console.log('MongoDB connected'))
```

```
  .catch((err) => console.error('MongoDB connection error:', err));

// Define User schema
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
}, { timestamps: true });

const User = mongoose.model('User', userSchema);

// Middleware
app.use(cors());
app.use(express.json());

// GET all users
app.get('/api/users', async (req, res) => {
  try {
    const users = await User.find().sort({ createdAt: -1 });
    res.json(users);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch users' });
  }
});
```

```
// POST new user
app.post('/api/users', async (req, res) => {
  try {
    const newUser = new User(req.body);
    const savedUser = await newUser.save();
    res.status(201).json(savedUser);
  } catch (err) {
    res.status(500).json({ error: err.message || 'Failed to create user' });
  }
});


// PUT update user
app.put('/api/users/:id', async (req, res) => {
  try {
    const updatedUser = await User.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true, runValidators: true }
    );
    if (!updatedUser) return res.status(404).json({ error: 'User not found'
});
    res.json(updatedUser);
  } catch (err) {
    res.status(500).json({ error: err.message || 'Failed to update user' });
  }
});
```

```
// DELETE user

app.delete('/api/users/:id', async (req, res) => {

  try {

    const deletedUser = await User.findByIdAndDelete(req.params.id);

    if (!deletedUser) return res.status(404).json({ error: 'User not found'
});

    res.json(deletedUser);

  } catch (err) {

    res.status(500).json({ error: err.message || 'Failed to delete user' });

  }

});



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

 This sets up a backend API with endpoints:

- `GET /api/users` → List users
- `POST /api/users` → Add user
- `PUT /api/users/:id` → Update user
- `DELETE /api/users/:id` → Delete user

Step 2: Setup Frontend API Routes (Next.js)

We'll convert the `fetch` code to **Axios** as you requested.

2.1 Install Axios

```
npm install axios
```

---

1 API Routes

`app/api/users/route.js` (GET & POST)

```
import axios from 'axios';

// GET all users
export async function GET() {
  try {
    const res = await axios.get('http://localhost:5000/api/users');
    return new Response(JSON.stringify(res.data), { status: res.status });
  } catch (error) {
    return new Response(JSON.stringify({ error: error.message || 'Failed to
fetch users' }), { status: 500 });
  }
}

// POST new user
export async function POST(request) {
  try {
    const body = await request.json();
    const res = await axios.post('http://localhost:5000/api/users', body, {
      headers: { 'Content-Type': 'application/json' },
    });
    return new Response(JSON.stringify(res.data), { status: res.status });
  } catch (error) {
    return new Response(JSON.stringify({ error: error.message || 'Failed to
create user' }), { status: 500 });
  }
}
```

`app/api/users/[id]/route.js` (PUT & DELETE)

```
import axios from 'axios';

// PUT: Update user
export async function PUT(request, { params }) {
  const { id } = params;
  try {
    const body = await request.json();
    const res = await axios.put(`http://localhost:5000/api/users/${id}`,
body, {
      headers: { 'Content-Type': 'application/json' },
    });
    return new Response(JSON.stringify(res.data), { status: res.status });
  } catch (error) {
    return new Response(JSON.stringify({ error: error.message || 'Failed to
update user' }), { status: 500 });
  }
}

// DELETE: Remove user
export async function DELETE(request, { params }) {
  const { id } = params;
  try {
    const res = await axios.delete(`http://localhost:5000/api/users/${id}`);
    return new Response(JSON.stringify(res.data), { status: res.status });
  } catch (error) {
    return new Response(JSON.stringify({ error: error.message || 'Failed to
delete user' }), { status: 500 });
  }
}
```

2️⃣ Client-side Page (`app/users/page.jsx`)

```
'use client';

import { useState, useEffect } from 'react';
import axios from 'axios';

export default function UsersPage() {
  const [users, setUsers] = useState([]);
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [editingUser, setEditingUser] = useState(null);
  const [error, setError] = useState('');

  const fetchUsers = async () => {
    try {
      const res = await axios.get('/api/users');
      setUsers(res.data);
    } catch (err) {
```

```
      setError(err.response?.data?.error || 'Failed to fetch users');
    }
  };

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

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    try {
      if (editingUser) {
        await axios.put(`/api/users/${editingUser._id || editingUser.id}`, {
name, email });
        setEditingUser(null);
      } else {
        await axios.post('/api/users', { name, email });
      }
      setName('');
      setEmail('');
      fetchUsers();
    } catch (err) {
      setError(err.response?.data?.error || 'Failed to save user');
    }
  };

  const handleEdit = (user) => {
    setEditingUser(user);
    setName(user.name);
    setEmail(user.email);
  };

  const handleDelete = async (id) => {
    setError('');
    try {
      await axios.delete(`/api/users/${id}`);
      setUsers(users.filter((user) => (user._id || user.id) !== id));
    } catch (err) {
      setError(err.response?.data?.error || 'Failed to delete user');
    }
  };

  return (
    <div style={{ padding: '2rem', fontFamily: 'Arial' }}>
      <h1>User Management</h1>

      {error && <p style={{ color: 'red' }}>{error}</p>}

      <form onSubmit={handleSubmit} style={{ marginBottom: '2rem' }}>
        <input
          type="text"
          placeholder="Name"
          value={name}
          onChange={(e) => setName(e.target.value)}
          required
          style={{ marginRight: '1rem', padding: '0.5rem' }}
```

```
        />
        <input
          type="email"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          required
          style={{ marginRight: '1rem', padding: '0.5rem' }}
        />
        <button type="submit" style={{ padding: '0.5rem 1rem' }}>
          {editingUser ? 'Update User' : 'Add User'}
        </button>
      </form>

      <table border="1" cellPadding="10" cellSpacing="0">
        <thead>
          <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {users.length === 0 && (
            <tr>
              <td colSpan="4" style={{ textAlign: 'center' }}>
                No users found
              </td>
            </tr>
          )}
          {users.map((user) => {
            const id = user._id || user.id;
            return (
              <tr key={id}>
                <td>{id}</td>
                <td>{user.name}</td>
                <td>{user.email}</td>
                <td>
                  <button onClick={() => handleEdit(user)} style={{
marginRight: '0.5rem' }}>
                    Edit
                  </button>
                  <button onClick={() => handleDelete(id)}>Delete</button>
                </td>
              </tr>
            );
          })}
        </tbody>
      </table>
    </div>
  );
}
```

Step 3: Test the Setup

1. Start the backend:

```
node server.js
```

2. Start your Next.js frontend:

```
npm run dev
```

3. Make requests from your frontend API routes:
    - GET all users → `/api/users`
    - POST new user → `/api/users`
    - PUT update user → `/api/users/[id]`
    - DELETE user → `/api/users/[id]`

This gives you a **full stack setup with backend + frontend routes using Axios**.