Here's a Basic Crud full-stack setup using next.js, Express.js, and MongoDB to help you get started. We'll break it into parts:

1. Backend – Express + MongoDB .

a. Setup Create a new backend folder and initialize a Node.js project:

```
mkdir backend

cd backend

npm init -y
```

b. Install dependencies

```
npm install express mongoose cors
```

c. Create basic Express server (index.js):-

```
// backend/index.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const app = express();

app.use(cors());

app.use(express.json());

// MongoDB connection

mongoose.connect('mongodb://localhost:27017/mydb', {

useNewUrlParser: true,

useUnifiedTopology: true,

})
```

```
.then(() => console.log('MongoDB connected'))
.catch(err => console.error(err));
// Simple schema
const UserSchema = new mongoose.Schema({
name: String,
email: String,
});
const User = mongoose.model('User', UserSchema);
// Routes
app.get('/api/users', async (req, res) => {
const users = await User.find();
res.json(users);
});

// Add user record
app.post('/api/users', async (req, res) => {
const user = new User(req.body);
await user.save();
res.json(user);
});
// Update user by ID
app.put('/api/users/:id', async (req, res) => {
 try {
 const updatedUser = await User.findByIdAndUpdate(
 req.params.id,
```

```
 req.body,
 { new: true } // return the updated document
 );
 res.json(updatedUser);
 } catch (err) {
 res.status(500).json({ error: 'Failed to update user' });
 }
});
// Delete user by ID
app.delete('/api/users/:id', async (req, res) => {
 try {
 await User.findByIdAndDelete(req.params.id);
 res.json({ message: 'User deleted successfully' });
 } catch (err) {
 res.status(500).json({ error: 'Failed to delete user' });
 }
});

// Start server
app.listen(5000, () => {
console.log('Server running on http://localhost:5000');
});
```

✅Final Steps

- Run backend:

```
node index.js
```

For Output

Open Browser:-

http://localhost:5000/api/users

Here's a complete **Next.js (App Router) CRUD frontend in JSX format** that connects to your **Express backend** (`http://localhost:5000/api/users`) using `fetch`. This uses **client-side rendering** (with `"use client"`).

Step-by-Step Guide

✅ Step 1: Create the Next.js App with App Router

```
npx create-next-app@latest mern-crud-frontend
```

Choose Options:

- TypeScript? → **No** (since you're using `.jsx`)
- App Router → **Yes**
- Tailwind CSS → Optional
- ESLint → Yes
- src/ directory → Your choice
- Import alias → Leave default

✅ Step 2: Change Directory

```
cd mern-crud-frontend
```

✅ Step 3: Create Folder Structure

Inside `app/`, create:

```
mkdir -p app/api/users app/api/users/[id] app/users
```

📁 Final Folder Structure:

```
bash

/app
  /api
    /users
      route.js          ← GET & POST
      /[id]
        route.js        ← PUT & DELETE
  /users
    page.jsx            ← Frontend CRUD UI
```

✅ Step 4: Add API Routes

app/api/users/route.js

```
export async function GET() {
  try {
    const res = await fetch('http://localhost:5000/api/users');
    const users = await res.json();
    return new Response(JSON.stringify(users), { status: 200 });
  } catch (error) {
    return new Response(JSON.stringify({ error: 'Failed to fetch users' }),
{ status: 500 });
  }
}

export async function POST(request) {
  try {
    const body = await request.json();
    const res = await fetch('http://localhost:5000/api/users', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(body),
    });
    const user = await res.json();
    return new Response(JSON.stringify(user), { status: 201 });
  } catch (error) {
    return new Response(JSON.stringify({ error: 'Failed to create user' }),
{ status: 500 });
  }
}
```

app/api/users/[id]/route.js

```
export async function PUT(request, { params }) {
  const { id } = params;
  try {
    const body = await request.json();
    const res = await fetch(`http://localhost:5000/api/users/${id}`, {
      method: 'PUT',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(body),
    });
    const updatedUser = await res.json();
    return new Response(JSON.stringify(updatedUser), { status: 200 });
  } catch (error) {
    return new Response(JSON.stringify({ error: 'Failed to update user' }),
{ status: 500 });
  }
}

export async function DELETE(request, { params }) {
  const { id } = params;
  try {
    const res = await fetch(`http://localhost:5000/api/users/${id}`, {
      method: 'DELETE',
    });
    const result = await res.json();
    return new Response(JSON.stringify(result), { status: 200 });
  } catch (error) {
    return new Response(JSON.stringify({ error: 'Failed to delete user' }),
{ status: 500 });
  }
}
```

✅ Step 5: Create the Frontend Page

✅ *app/users/page.jsx*

```
'use client';
import { useEffect, useState } from 'react';

export default function UsersPage() {
  const [users, setUsers] = useState([]);
  const [formData, setFormData] = useState({ name: '', email: '' });
  const [editId, setEditId] = useState(null);

  const fetchUsers = async () => {
    const res = await fetch('/api/users');
    const data = await res.json();
    setUsers(data);
  };

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

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

  const handleSubmit = async (e) => {
    e.preventDefault();
```

```
    if (editId) {
      await fetch(`/api/users/${editId}`, {
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData),
      });
      setEditId(null);
    } else {
      await fetch('/api/users', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData),
      });
    }
    setFormData({ name: '', email: '' });
    fetchUsers();
  };

  const handleDelete = async (id) => {
    await fetch(`/api/users/${id}`, { method: 'DELETE' });
    fetchUsers();
  };

  const handleEdit = (user) => {
    setFormData({ name: user.name, email: user.email });
    setEditId(user._id);
  };

  return (
    <div style={{ maxWidth: '600px', margin: '40px auto' }}>
      <h1>{editId ? 'Edit User' : 'Add User'}</h1>
      <form onSubmit={handleSubmit}>
        <input type="text" name="name" placeholder="Name"
value={formData.name} onChange={handleChange} required />
        <input type="email" name="email" placeholder="Email"
value={formData.email} onChange={handleChange} required />
        <button type="submit">{editId ? 'Update' : 'Create'} User</button>
      </form>

      <ul>
        {users.map((user) => (
          <li key={user._id}>
            {user.name} - {user.email}
            <button onClick={() => handleEdit(user)}>Edit</button>
            <button onClick={() => handleDelete(user._id)} style={{ color:
'red' }}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

✅ Step 6: Run the Project

```
npm run dev
```

Go to:

```
http://localhost:3000/users
```

You now have a **full CRUD page** interacting with your **Express + MongoDB backend**

Final Folder Structure

```
/app
  /users
    page.jsx
  /api
    /users
      route.js
      /[id]
        route.js
```