Here's a **simple full-stack example** using **Express.js + MongoDB** for the backend and **React.js with Axios** for the frontend.

□ 1. Backend: Express.js + MongoDB

Install dependencies:

```
mkdir backend
Cd backend
npm init -y
npm install express mongoose cors
```

Create `server.js`:

```
// server.js
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');

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

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

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

// Model
const Record = mongoose.model('Record', recordSchema);

// Routes
app.get('/records', async (req, res) => {
```

```
  const records = await Record.find();
  res.json(records);
});

app.post('/records', async (req, res) => {
  const newRecord = new Record(req.body);
  await newRecord.save();
  res.json({ message: 'Record added successfully' });
});

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

□ 2. Frontend: React + Axios

Create React app (if not already):

```
npx create-react-app records-app
cd records-app
npm install axios
```

Replace `src/App.js` with:

```
// src/App.js
import React, { useEffect, useState } from 'react';
import axios from 'axios';

function App() {
  const [records, setRecords] = useState([]);
  const [form, setForm] = useState({ name: '', email: '', phone: '', city: ''
});

  const fetchRecords = async () => {
    const res = await axios.get('http://localhost:5000/records');
    setRecords(res.data);
  };

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

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

  const handleSubmit = async (e) => {
    e.preventDefault();
    await axios.post('http://localhost:5000/records', form);
    setForm({ name: '', email: '', phone: '', city: '' });
    fetchRecords();
  };

  return (
    <div style={{ padding: '20px' }}>
      <h2>Add Record</h2>
      <form onSubmit={handleSubmit}>
        <input name="name" placeholder="Name" value={form.name}
onChange={handleChange} />
        <input name="email" placeholder="Email" value={form.email}
onChange={handleChange} />
        <input name="phone" placeholder="Phone" value={form.phone}
onChange={handleChange} />
        <input name="city" placeholder="City" value={form.city}
onChange={handleChange} />
        <button type="submit">Add</button>
      </form>

      <h2>Records</h2>
      <ul>
        {records.map((rec, idx) => (
          <li key={idx}>
            {rec.name} | {rec.email} | {rec.phone} | {rec.city}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

□ 3. Final Steps

Start backend server:

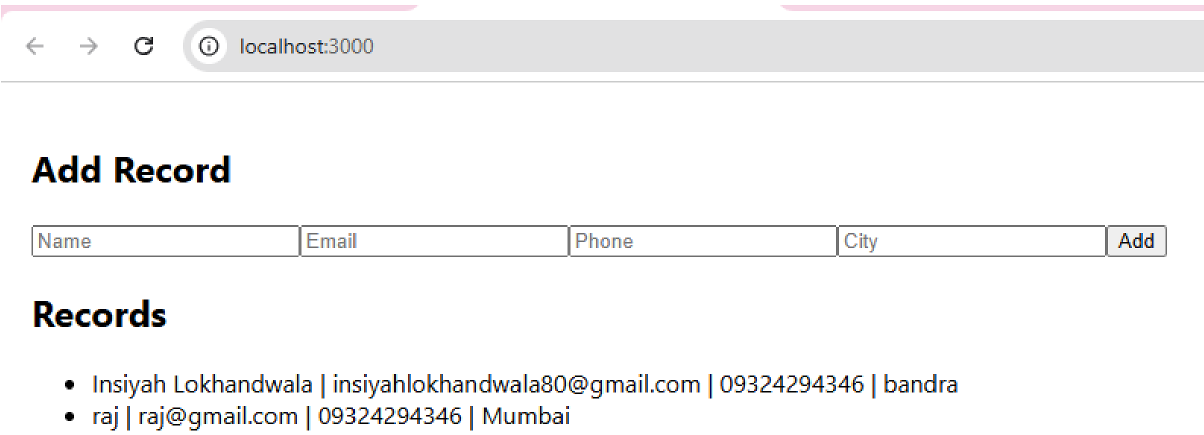
```
node server.js
```

Start frontend React app:

```
npm start
```

Expected Behavior:

- You'll see a form to enter `name`, `email`, `phone`, `city`.
- Upon submission, the record is inserted into MongoDB and displayed in the list below



Let's refactor your `server.js` into separate files to follow a clean

MVC-like Structure using:

- `models/Record.js` → Mongoose model
- `routes/recordRoutes.js` → Express routes
- `server.js` → App entry point

□ Final Structure

```
your-project/
│
├── models/
│   └── Record.js
│
├── routes/
│   └── recordRoutes.js
│
├── server.js
└── package.json
```

□ 1. `models/Record.js`

```
// 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);
```

2. routes/recordRoutes.js:-

```
// routes/

const express = require('express');
const router = express.Router();
const Record = require('../models/Record');

// GET all records
router.get('/', async (req, res) => {
  try {
    const records = await Record.find();
    res.json(records);
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch records' });
  }
});

// POST a new record
router.post('/', async (req, res) => {
  try {
    const newRecord = new Record(req.body);
    await newRecord.save();
    res.json({ message: 'Record added successfully' });
  } catch (err) {
    res.status(400).json({ error: 'Failed to add record' });
  }
});

module.exports = router;
```

□ 3. `server.js`:-

```
// server.js
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const recordRoutes = require('./routes/recordRoutes');

const app = express();

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

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/recordsDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB connection error:', err));

// Routes
app.use('/records', recordRoutes);

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

□ How to run

1. Ensure MongoDB is running (manually or as a service).
2. In your project folder, run:

```
npm install express mongoose cors
node server.js
```

You can now access:

- GET: `http://localhost:5000/records`
- POST: `http://localhost:5000/records` with body: `{ name, email, phone, city }`

Explanation of backend code :-

Here's a line-by-line explanation of your `models/Record.js` code:

File: `models/Record.js`

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

Explanation:
This line imports the **Mongoose** library, which is used to interact with a MongoDB database using JavaScript.
Mongoose provides tools to define schemas and models for organizing data.

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

Explanation:
Here you define a **Mongoose schema**, which is like a blueprint for the data in your MongoDB collection.

- `recordSchema` defines the structure of each document (record) in the MongoDB collection.
- Each field (`name`, `email`, `phone`, `city`) is set to type `String`, meaning it expects text data.

Example of a document using this schema:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "phone": "1234567890",
  "city": "New York"
}
```

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

� Explanation:

This line creates and exports a Mongoose model named 'Record'.

'Record' is the name of the model, and it will be used to create a MongoDB collection named records (Mongoose automatically pluralizes it).

The model uses the recordSchema as its structure.

By exporting it, you can use this model in other files (like your route handlers) to interact with the database (e.g., Record.find(), new Record()).

Summary

Line	Meaning
`require('mongoose')`	Import Mongoose library
`new mongoose.Schema({...})`	Define the structure of your MongoDB documents
`module.exports = mongoose.model(...)`	Create a model and export it for use in other files

Let's break down and explain each line of your `routes/recordRoutes.js` file step by step:

□ File: `routes/recordRoutes.js`

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

□ **Meaning**:
This line imports the **Express** framework.
Express provides a simple way to create APIs and handle HTTP routes (GET, POST, etc.).

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

□ **Meaning**:
Creates a **router object** using Express.
A router is like a mini version of your main app — you define specific routes here and later plug it into the main app (in `server.js`).

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

□ **Meaning**:
This imports the **Mongoose model** you defined in `models/Record.js`.
You'll use it to interact with the MongoDB collection (for finding, creating, etc.).

□ Route: GET all records

```
router.get('/', async (req, res) => {
```

□ **Meaning**:
Defines a **GET** endpoint at `/records` (in `server.js`, it's mounted as `/records`).
This means when someone makes a GET request to `http://localhost:5000/records`, this function runs.

```
  try {
    const records = await Record.find();
```

□ **Meaning**:
Uses Mongoose's `find()` to fetch **all documents** from the `records` collection.
`await` waits for the database to respond.

```
    res.json(records);
```

□ **Meaning**:
Sends the list of records back to the client as JSON.

```
  } catch (err) {
    res.status(500).json({ error: 'Failed to fetch records' });
  }
});
```

□ **Meaning**:
If something goes wrong (e.g., database error), catch it and return a **500 Internal Server Error** with a message.

□ Route: POST a new record

```
router.post('/', async (req, res) => {
```

□ **Meaning**:
Defines a **POST** endpoint at `/records`.
This handles requests to create new records (from a form or frontend using Axios).

```
  try {
    const newRecord = new Record(req.body);
```

□ **Meaning**:
Creates a new instance of the `Record` model using data from the request body.
Example body:

```
{ "name": "Alice", "email": "alice@mail.com", "phone": "1234567890",
"city": "Paris" }
```

```
    await newRecord.save();
```

□ **Meaning**:
Saves the new record to the MongoDB database.

```
    res.json({ message: 'Record added successfully' });
```

□ **Meaning**:
Returns a success message to the client as JSON.

```
  } catch (err) {
    res.status(400).json({ error: 'Failed to add record' });
```

```
  }
});
```

□ **Meaning**:
If something goes wrong during creation (e.g., invalid data), catch the error and return a **400 Bad Request** response.

```
module.exports = router;
```

□ **Meaning**:

Exports the router so it can be used in `server.js` like this:

Line:

```
app.use('/records', recordRoutes);
```

□ Meaning:

This line tells your Express app:

"For any request that starts with `/records`, use the routes defined in `recordRoutes`."

Think of it like a prefix

- `app.use('/records', ...)` is a **prefix**
- Routes in `recordRoutes` like `'/'` or `'/somePath'` will be attached **after** that

---

□ Example:

```
// recordRoutes.js
router.get('/', ...)        // handles GET /records
router.post('/', ...)       // handles POST /records
router.get('/:id', ...)     // handles GET /records/123
```

□ Summary

Code	What it does
`express.Router()`	Creates a mini route handler
`router.get()`	Handles fetching all records
`router.post()`	Handles adding a new record
`Record.find()`	Retrieves all documents from MongoDB
`new Record(req.body)`	Creates a new record object
`await newRecord.save()`	Saves it to MongoDB
`module.exports`	Makes the router usable in `server.js`

Let's go through your `server.js` file **line by line** so you understand exactly what each part does.

□ File: `server.js`

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

□ **Meaning**:
Imports the **Express** framework, which helps you build web servers and APIs easily in Node.js.

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

□ **Meaning**:
Imports **Mongoose**, a library used to connect to and interact with a **MongoDB database** using models and schemas.

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

□ **Meaning**:
Imports the **CORS** middleware, which allows your server to accept requests from other domains (like your frontend running on `localhost:3000`).

```
const recordRoutes = require('./routes/recordRoutes');
```

 Meaning:
Loads the custom routes you created for handling **record-related API endpoints** (like GET and POST for `/records`).
This comes from your `routes/recordRoutes.js` file.

```
const app = express();
```

 Meaning:
Creates an **Express application instance** (`app`) — the main object you use to configure routes, middleware, etc.

Middleware setup

```
app.use(express.json());
```

 Meaning:
Tells Express to automatically parse **JSON request bodies**, so you can access `req.body` in POST/PUT requests.

```
app.use(cors());
```

 Meaning:
Enables **CORS** (Cross-Origin Resource Sharing), which is required when your frontend and backend run on **different ports** (like React on 3000 and backend on 5000).
Without this, the browser blocks the request for security.

MongoDB connection

```
mongoose.connect('mongodb://localhost:27017/recordsDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
```

 Meaning:
Connects your app to the **MongoDB database** running locally on your machine.

- `recordsDB` is the database name.
- Options like `useNewUrlParser` and `useUnifiedTopology` avoid deprecation warnings.

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

□ **Meaning**:
If the database connection is successful, it logs a confirmation message.

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

□ **Meaning**:
If there's an error connecting to MongoDB, it prints the error in the console.

□ Route Mounting

```
app.use('/records', recordRoutes);
```

□ **Meaning**:
Mounts all routes defined in `recordRoutes.js` under the `/records` path.
So:

- `router.get('/')` becomes `GET /records`
- `router.post('/')` becomes `POST /records`

□ Start the server

```
const PORT = 5000;
```

□ **Meaning**:
Sets the port number your server will run on.
Clients can now make requests to `http://localhost:5000`.

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

□ **Meaning**:
Starts the server and begins listening for requests on port 5000.
When it starts, it logs the message to the console.

□ Summary Table

Line	What It Does
`require('express')`	Import Express
`require('mongoose')`	Import Mongoose (for MongoDB)
`require('cors')`	Import CORS middleware

Line	What It Does
`require('./routes/recordRoutes')`	Load your custom API routes
`express()`	Create an Express app
`app.use(express.json())`	Enable JSON parsing
`app.use(cors())`	Allow cross-origin requests
`mongoose.connect(...)`	Connect to MongoDB
`app.use('/records', ...)`	Attach the record routes under `/records`
`app.listen(...)`	Start the server on port 5000