**Perfect — let's build a simple Express + Mongoose backend for a collection called `Record` that stores:**
**➡ `name`, `city`, and `phone`.**

I'll show you the **complete working example**, step-by-step 

---

# Step 1: Setup Project

In your terminal:

```
mkdir record-api
cd record-api
npm init -y
npm install express mongoose
```

Then create a file:

`server.js`

---

# Step 2: Full Code (Express + MongoDB)

```
// server.js

const express = require('express');
const mongoose = require('mongoose');
const app = express();

app.use(express.json()); // to parse JSON requests

// 1 Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/recorddb')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.log(err));

// 2 Define Schema + Model
const Record = mongoose.model('Record', new mongoose.Schema({
  name: String,
  city: String,
  phone: String
}));

// 3 Routes

// ➤ GET all records
app.get('/api/records', async (req, res) => {
  const records = await Record.find();
  res.json(records);
});
```

```
// ➤ POST (Add a new record)
app.post('/api/records', async (req, res) => {
  const record = new Record({
    name: req.body.name,
    city: req.body.city,
    phone: req.body.phone
  });
  await record.save();
  res.send('Record saved successfully!');
});

// ➤ GET record by ID
app.get('/api/records/:id', async (req, res) => {
  const record = await Record.findById(req.params.id);
  if (!record) return res.status(404).send('Record not found');
  res.json(record);
});

// ➤ PUT (Update record)
app.put('/api/records/:id', async (req, res) => {
  const record = await Record.findByIdAndUpdate(
    req.params.id,
    { name: req.body.name, city: req.body.city, phone: req.body.phone },
    { new: true }
  );
  if (!record) return res.status(404).send('Record not found');
  res.json(record);
});

// ➤ DELETE record
app.delete('/api/records/:id', async (req, res) => {
  const record = await Record.findByIdAndDelete(req.params.id);
  if (!record) return res.status(404).send('Record not found');
  res.send('Record deleted successfully!');
});

// 4 Start Server
app.listen(5000, () => console.log('Server running on port 5000'));
```

---

## How It Works:-

| Route | Method | Description |
|---|---|---|
| `/api/records` | GET | Fetch all records |
| `/api/records/:id` | GET | Fetch single record by ID |
| `/api/records` | POST | Add a new record |
| `/api/records/:id` | PUT | Update an existing record |
| `/api/records/:id` | DELETE | Delete a record |

---

# □ Step 3: Test Using Postman or Thunder Client

## ➤ Add a Record (POST)

**URL:** `http://localhost:5000/api/records`
**Body (JSON):**

```
{
  "name": "John Doe",
  "city": "Mumbai",
  "phone": "9876543210"
}
```

□ Response: `"Record saved successfully!"`

---

## ➤ Get All Records

**GET:** `http://localhost:5000/api/records`

□ Response:

```
[
  {
    "_id": "674eaa4d8b...",
    "name": "John Doe",
    "city": "Mumbai",
    "phone": "9876543210"
  }
]
```

---

## ➤ Update Record

**PUT:** `http://localhost:5000/api/records/<id>`
**Body:**

```
{
  "name": "John Smith",
  "city": "Pune",
  "phone": "9999999999"
}
```

□ Response: updated record object

---

## ➢ Delete Record

**DELETE:** `http://localhost:5000/api/records/<id>`
 Response: `"Record deleted successfully!"`

---

This is a **real, full CRUD API** for a “Record” collection using:

- **MongoDB** (for storage)
- **Mongoose** (for schema + queries)
- **Express** (for routes)

Perfect — that's the **professional, interview-ready structure** most MERN developers use. It shows you understand **modular code** and **project organization** — exactly what interviewers love to see.

Let's refactor your previous "Record" CRUD API into **separate files**:

---

# □ Project Structure

Here's how your folders/files should look □

```
record-api/
│
├── server.js
├── config/
│   └── db.js
├── models/
│   └── Record.js
└── routes/
    └── recordRoutes.js
```

---

# □ Step 1: Install Dependencies

In your project folder:

```
npm init -y
npm install express mongoose
```

---

# □ Step 2: MongoDB Connection (config/db.js)

```
// config/db.js
const mongoose = require('mongoose');

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

module.exports = connectDB;
```

---

## □ Step 3: Mongoose Model (models/Record.js)

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

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

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

module.exports = Record;
```

---

## □ Step 4: Routes File (routes/recordRoutes.js)

```
// routes/recordRoutes.js
const express = require('express');
const router = express.Router();
const Record = require('../models/Record');

// ➤ GET all records
router.get('/', async (req, res) => {
  const records = await Record.find();
  res.json(records);
});

// ➤ POST (Add new record)
router.post('/', async (req, res) => {
  const record = new Record({
    name: req.body.name,
    city: req.body.city,
    phone: req.body.phone
  });
  await record.save();
  res.send('□ Record saved successfully!');
});

// ➤ GET by ID
router.get('/:id', async (req, res) => {
  const record = await Record.findById(req.params.id);
  if (!record) return res.status(404).send('Record not found');
  res.json(record);
});

// ➤ PUT (Update)
router.put('/:id', async (req, res) => {
  const record = await Record.findByIdAndUpdate(
    req.params.id,
    { name: req.body.name, city: req.body.city, phone: req.body.phone },
    { new: true }
```

```
  );
  if (!record) return res.status(404).send('Record not found');
  res.json(record);
});

// ➤ DELETE
router.delete('/:id', async (req, res) => {
  const record = await Record.findByIdAndDelete(req.params.id);
  if (!record) return res.status(404).send('Record not found');
  res.send(' Record deleted successfully!');
});

module.exports = router;
```

---

## Step 5: Main Server File (server.js)

```
// server.js
const express = require('express');
const connectDB = require('./config/db');
const recordRoutes = require('./routes/recordRoutes');

const app = express();

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

// Connect to MongoDB
connectDB();

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

// Start server
app.listen(5000, () => console.log(' Server running on port 5000'));
```

---

## Now Run It

```
node server.js
```

You'll see:

```
  MongoDB connected
 Server running on port 5000
```

---

## Test API Endpoints (same as before)

| Method | Route | Description |
|---|---|---|
| GET | /api/records | Get all records |

| Method | Route | Description |
|---|---|---|
| POST | `/api/records` | Add new record |
| GET | `/api/records/:id` | Get record by ID |
| PUT | `/api/records/:id` | Update record |
| DELETE | `/api/records/:id` | Delete record |

---

## □ Why This Separation Is Better

| File | Responsibility |
|---|---|
| **server.js** | App entry point and route setup |
| **config/db.js** | Handles MongoDB connection |
| **models/Record.js** | Defines data structure |
| **routes/recordRoutes.js** | Defines API endpoints |

This makes your code **clean, scalable, and interview-ready.**