

Let's build a **full Express.js + Mongoose REST API** using **local MongoDB** with a collection of **contacts** that have `name`, `city`, and `phone`. I'll make it modular with **models, controllers, and routes** so you can see the structure clearly.

## Step 1: Set Up Your Project

1. Make sure you have **Node.js** installed. Check with:

```
node -v
npm -v
```

2. Create a new project folder and initialize npm:

```
mkdir my-api
cd my-api
npm init -y
```

3. Install **Express.js**:

```
npm install express
```

(Optional: install **nodemon** for auto-restarting the server during development)

```
npm install --save-dev nodemon
```

In `package.json`, add a start script:

```
"scripts": {
  "start": "node index.js",
  "dev": "nodemon index.js"
}
```

`package.json` file :-

```
{
  "name": "my-api",
  "version": "1.0.0",
  "description": "",
```

```
"main": "index.js",  
"scripts": {  
  "start": "node index.js",  
  "dev": "nodemon index.js"  
},  
"keywords": [],  
"author": "",  
"license": "ISC",  
"type": "commonjs",  
"dependencies": {  
  "express": "^5.2.1"  
},  
"devDependencies": {  
  "nodemon": "^3.1.14"  
}  
}
```

Run the server:

```
npm run dev
```

---

## Folder Structure

```
my-api/  
├── controllers/  
│   └── contactController.js  
├── models/  
│   └── contactModel.js  
├── routes/  
│   └── contactRoutes.js  
├── index.js  
└── package.json
```

---

### Step 1: Model (`models/contactModel.js`)

```
const mongoose = require('mongoose');  
  
const contactSchema = new mongoose.Schema({  
  name: { type: String, required: true },  
  city: { type: String, required: true },  
  phone: { type: String, required: true }  
}, { timestamps: true });  
  
module.exports = mongoose.model('Contact', contactSchema);
```

- `timestamps: true` adds `createdAt` and `updatedAt` automatically.
- 

### Step 2: Controller (`controllers/contactController.js`)

```
const Contact = require('../models/contactModel');  
  
// Get all contacts  
const getAllContacts = async (req, res) => {  
  try {  
    const contacts = await Contact.find();  
    res.json(contacts);  
  } catch (err) {  
    res.status(500).json({ message: err.message });  
  }  
};  
  
// Get contact by ID  
const getContactById = async (req, res) => {  
  try {  
    const contact = await Contact.findById(req.params.id);  
    if (!contact) return res.status(404).json({ message: 'Contact not  
found' });  
    res.json(contact);  
  } catch (err) {  
    res.status(500).json({ message: err.message });  
  }  
};
```

```

    }
};

// Create new contact
const createContact = async (req, res) => {
  const contact = new Contact({
    name: req.body.name,
    city: req.body.city,
    phone: req.body.phone
  });

  try {
    const newContact = await contact.save();
    res.status(201).json(newContact);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};

// Update contact
const updateContact = async (req, res) => {
  try {
    const contact = await Contact.findById(req.params.id);
    if (!contact) return res.status(404).json({ message: 'Contact not
found' });

    contact.name = req.body.name || contact.name;
    contact.city = req.body.city || contact.city;
    contact.phone = req.body.phone || contact.phone;

    const updatedContact = await contact.save();
    res.json(updatedContact);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};

// Delete contact
const deleteContact = async (req, res) => {
  try {
    const contact = await Contact.findByIdAndDelete(req.params.id);
    if (!contact) return res.status(404).json({ message: 'Contact not
found' });
    res.json({ message: 'Contact deleted' });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};

module.exports = {
  getAllContacts,
  getContactById,
  createContact,

```

```
    updateContact,  
    deleteContact  
  };  
};
```

---

### Step 3: Routes (`routes/contactRoutes.js`)

```
const express = require('express');  
const router = express.Router();  
const contactController = require('../controllers/contactController');  
  
router.get('/', contactController.getAllContacts);  
router.get('/:id', contactController.getContactById);  
router.post('/', contactController.createContact);  
router.put('/:id', contactController.updateContact);  
router.delete('/:id', contactController.deleteContact);  
  
module.exports = router;
```

---

### Step 4: Main Server (`index.js`)

```
const express = require('express');  
const mongoose = require('mongoose');  
const app = express();  
const PORT = 3000;  
  
// Middleware  
app.use(express.json());  
  
// Routes  
const contactRoutes = require('./routes/contactRoutes');  
app.use('/contacts', contactRoutes);  
  
// Connect to local MongoDB  
mongoose.connect('mongodb://127.0.0.1:27017/mydb')  
  .then(() => console.log('Connected to local MongoDB'))  
  .catch(err => console.error('MongoDB connection error:', err));  
  
// Root endpoint  
app.get('/', (req, res) => {  
  res.send('Welcome to Contact REST API!');  
});  
  
// Start server  
app.listen(PORT, () => {  
  console.log(`Server running at http://localhost:${PORT}`);  
});
```

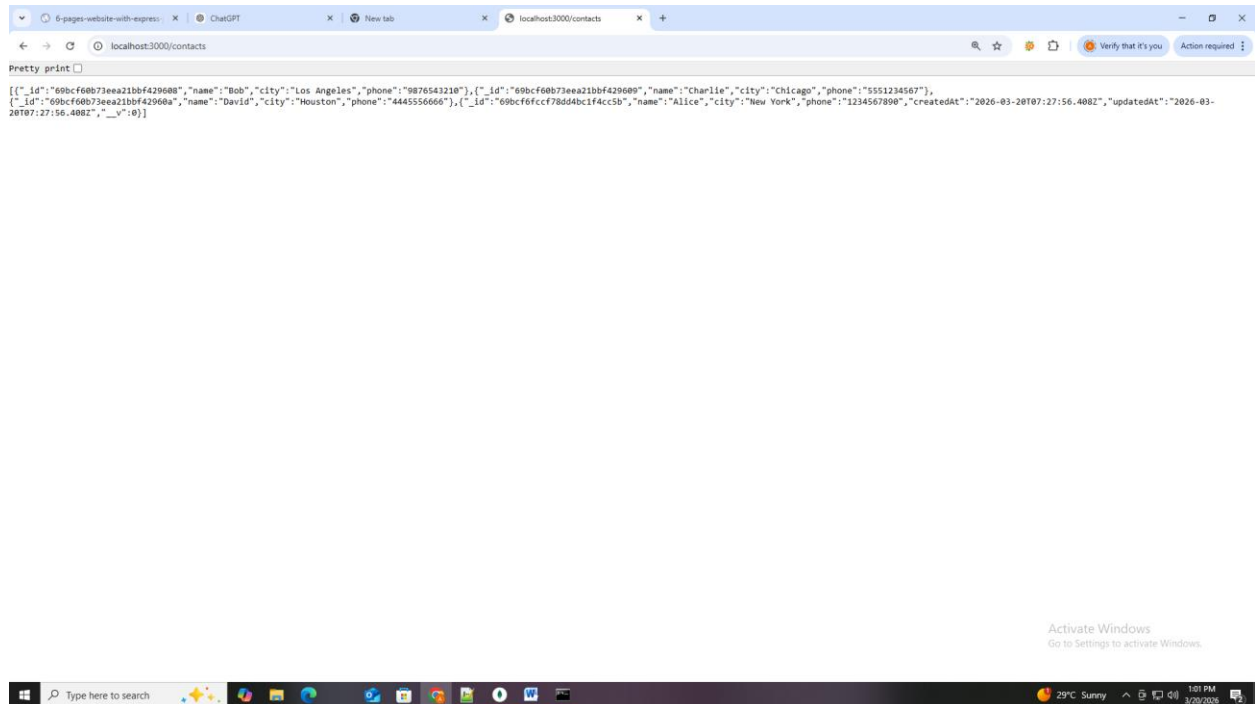
```
});
```

---

## Step 5: Test the API

### 1. GET all contacts

```
curl http://localhost:3000/contacts
```



### 2. POST a new contact

```
curl -X POST -H "Content-Type: application/json" -d '{"name": "Alice", "city": "New York", "phone": "1234567890"}' http://localhost:3000/contacts
```

### 3. GET contact by ID

```
curl http://localhost:3000/contacts/<contact_id>
```

### 4. PUT (update)

```
curl -X PUT -H "Content-Type: application/json" -d '{"city": "Los Angeles"}' http://localhost:3000/contacts/<contact_id>
```

### 5. DELETE

```
curl -X DELETE http://localhost:3000/contacts/<contact_id>
```

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.6466]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>net start MongoDB
The MongoDB Server (MongoDB) service is starting...
The MongoDB Server (MongoDB) service was started successfully.

C:\WINDOWS\system32>cd\

C:\>curl -X PUT -H "Content-Type: application/json" -d '{"city":"Boston"}' http://localhost:3000/contacts/69bcf5e773
eea21bbf429607
{"_id":"69bcf5e773eea21bbf429607","name":"Alice","city":"Boston","phone":"1234567890","updatedAt":"2026-03-20T07:26:52.3
43Z"}
C:\>curl -X DELETE http://localhost:3000/contacts/69bcf5e773eea21bbf429607
{"message":"Contact deleted"}
C:\>curl -X POST -H "Content-Type: application/json" -d '{"name":"Alice","city":"New York","phone":"123456789
0"}' http://localhost:3000/contacts
{"name":"Alice","city":"New York","phone":"1234567890","_id":"69bcf6fccf78dd4bc1f4cc5b","createdAt":"2026-03-20T07:27:56
.408Z","updatedAt":"2026-03-20T07:27:56.408Z","__v":0}
C:\>
```

Let's write the `curl` commands for Windows CMD to **update, delete, and add** the record for Alice with `_id: 69bcf5e773eea21bbf429607`.

## 1 Update Alice's city to "Boston"

```
curl -X PUT -H "Content-Type: application/json" -d '{"city":"Boston"}'
http://localhost:3000/contacts/69bcf5e773eea21bbf429607
```

- After running, her city will be updated.

## 2 Update multiple fields (name, city, phone)

```
curl -X PUT -H "Content-Type: application/json" -d '{"name":"Alice
Smith","city":"Boston","phone":"1112223333"}'
http://localhost:3000/contacts/69bcf5e773eea21bbf429607
```

- This updates all 3 fields at once.

---

### 3 Delete Alice

```
curl -X DELETE http://localhost:3000/contacts/69bcf5e773eea21bbf429607
```

- After this, the record is removed from your database.

---

### 4 Add a new Alice record

```
curl -X POST -H "Content-Type: application/json" -d
"{\"name\": \"Alice\", \"city\": \"New York\", \"phone\": \"1234567890\"}"
http://localhost:3000/contacts
```

- This creates a new record with a new `_id`.

---

### 5 Verify Alice

```
curl http://localhost:3000/contacts
```

You should see all records, including the updated or newly added Alice.

---

#### Tips for Windows CMD

- Always **escape double quotes** inside JSON with `\`
- For **PowerShell**, you can use single quotes around JSON:

```
curl -X POST -H "Content-Type: application/json" -d
'{"name": "Alice", "city": "New York", "phone": "'
```

Let's go **line by line** and explain everything in detail, so you understand exactly what each part does in your **Express + Mongoose REST API**. I'll break it down **step by step**.

---

## Step 1: Model (`models/contactModel.js`)

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

- This imports the **Mongoose library**, which is used to interact with MongoDB in an object-oriented way.
  - Mongoose allows us to define **schemas** (structure for documents) and models (classes to work with collections).
- 

```
const contactSchema = new mongoose.Schema({
  name: { type: String, required: true },
  city: { type: String, required: true },
  phone: { type: String, required: true }
}, { timestamps: true });
```

- `mongoose.Schema({...})` → defines the **structure** of documents in the `contacts` collection.
- Each field (`name`, `city`, `phone`) has:
  - `type: String` → ensures the value must be a string
  - `required: true` → MongoDB will **reject documents** without this field
- `{ timestamps: true }` → automatically adds:
  - `createdAt` → when the document was created
  - `updatedAt` → when the document was last updated

Example MongoDB document after insertion:

```
{
  "_id": "641234abcd1234",
  "name": "Alice",
  "city": "New York",
  "phone": "1234567890",
  "createdAt": "2026-03-20T12:00:00.000Z",
  "updatedAt": "2026-03-20T12:00:00.000Z"
}
```

---

```
module.exports = mongoose.model('Contact', contactSchema);
```

- `mongoose.model('Contact', contactSchema)` → creates a **Mongoose model**:
    - **Name**: `Contact`
    - **Schema**: `contactSchema`
    - This model represents the **contacts collection** in MongoDB (Mongoose automatically pluralizes the name).
  - `module.exports` → allows other files (like controllers) to **import this model**.
- 

## Step 2: Controller (`controllers/contactController.js`)

```
const Contact = require('../models/contactModel');
```

- Imports the `Contact` model so we can **query and manipulate the contacts collection**.
- 

### Get all contacts

```
const getAllContacts = async (req, res) => {
  try {
    const contacts = await Contact.find();
    res.json(contacts);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

- `async (req, res)` → asynchronous function because MongoDB operations return **Promises**
  - `Contact.find()` → fetches **all documents** from `contacts` collection
  - `res.json(contacts)` → sends JSON response to the client
  - `catch` → handles errors and sends **HTTP 500** (server error)
- 

### Get a contact by ID

```
const getContactById = async (req, res) => {
  try {
    const contact = await Contact.findById(req.params.id);
    if (!contact) return res.status(404).json({ message: 'Contact not found' });
    res.json(contact);
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

- `req.params.id` → reads the `:id` from the URL (e.g., `/contacts/641234abcd1234`)
  - `Contact.findById(id)` → finds a single contact by `_id`
  - If not found → sends **HTTP 404 Not Found**
- 

### Create a new contact

```
const createContact = async (req, res) => {
  const contact = new Contact({
    name: req.body.name,
    city: req.body.city,
    phone: req.body.phone
  });

  try {
    const newContact = await contact.save();
    res.status(201).json(newContact);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};
```

- `req.body` → contains **data sent from client** (POST request body)
  - `new Contact({...})` → creates a new Mongoose document
  - `contact.save()` → saves the document to MongoDB
  - `res.status(201)` → **HTTP 201 Created** for successful insertion
  - `catch` → sends **HTTP 400 Bad Request** if data is invalid
- 

### Update a contact

```
const updateContact = async (req, res) => {
  try {
    const contact = await Contact.findById(req.params.id);
    if (!contact) return res.status(404).json({ message: 'Contact not found' });

    contact.name = req.body.name || contact.name;
    contact.city = req.body.city || contact.city;
    contact.phone = req.body.phone || contact.phone;

    const updatedContact = await contact.save();
    res.json(updatedContact);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};
```

- Finds the contact first by ID
- Updates fields **only if they are provided**, otherwise keeps old value
- `contact.save()` → commits changes to MongoDB
- Sends updated document as JSON

---

### Delete a contact

```
const deleteContact = async (req, res) => {
  try {
    const contact = await Contact.findByIdAndDelete(req.params.id);
    if (!contact) return res.status(404).json({ message: 'Contact not
found' });
    res.json({ message: 'Contact deleted' });
  } catch (err) {
    res.status(500).json({ message: err.message });
  }
};
```

- `findByIdAndDelete` → removes document by ID
- If contact not found → HTTP 404
- Otherwise → returns a simple confirmation JSON

---

```
module.exports = {
  getAllContacts,
  getContactById,
  createContact,
  updateContact,
  deleteContact
};
```

- Exports all controller functions so they can be used in **routes**

---

## Step 3: Routes (`routes/contactRoutes.js`)

```
const express = require('express');
const router = express.Router();
const contactController = require('../controllers/contactController');
```

- Imports **Express router** → allows modular routes
- Imports the controller functions

---

### Define RESTful routes

```
router.get('/', contactController.getAllContacts);
router.get('/:id', contactController.getContactById);
router.post('/', contactController.createContact);
router.put('/:id', contactController.updateContact);
router.delete('/:id', contactController.deleteContact);
```

- GET /contacts/ → fetch all contacts
  - GET /contacts/:id → fetch single contact by ID
  - POST /contacts/ → create new contact
  - PUT /contacts/:id → update existing contact by ID
  - DELETE /contacts/:id → delete contact by ID
- 

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

- Exports router so it can be mounted in `index.js`:

```
app.use('/contacts', contactRoutes);
```

- This attaches all routes under `/contacts` URL path
- 

## □ Summary

- **Model** → defines the **structure** of documents
- **Controller** → defines **CRUD logic**
- **Routes** → maps **HTTP endpoints** to controller functions

With this structure, your Express API is **clean, modular, and fully RESTful**.

Let's go **line by line** and explain everything in your `index.js` file for your Express + MongoDB API. I'll break it down carefully so you understand what each line does.

---

## 1 Import Express

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

- Imports the **Express library**, which is a framework for building **Node.js web servers and APIs**.
  - `express` will be used to create the server, define routes, and handle requests/responses.
- 

## 2 Import Mongoose

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

- Imports **Mongoose**, which is a **MongoDB object modeling tool**.
  - Mongoose allows you to define **schemas, models**, and provides easy **CRUD operations** on MongoDB.
- 

## 3 Create Express app

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

- Calls `express()` to **create an Express application instance**.
  - This `app` object is used to **define routes, middleware, and start the server**.
- 

## 4 Define port

```
const PORT = 3000;
```

- Sets the port number for your server.

- Here, the API will listen on **port 3000**, so your endpoints will be like `http://localhost:3000`.
- 

## 5 Middleware to parse JSON

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

- Middleware tells Express to **automatically parse JSON in request bodies**.
  - Needed for `POST` and `PUT` requests where clients send JSON data.
  - Example: `req.body` will contain the JSON sent by the client.
- 

## 6 Import routes

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

- Imports your **routes module** for contacts (CRUD endpoints).
  - This connects your **controller logic** to the server.
- 

## 7 Mount routes

```
app.use('/contacts', contactRoutes);
```

- Mounts all routes from `contactRoutes` under `/contacts`.
  - Example:
    - `GET /contacts/` → get all contacts
    - `POST /contacts/` → create a contact
  - Express combines the **base path** `/contacts` with the **routes defined in the router**.
- 

## 8 Connect to local MongoDB

```
mongoose.connect('mongodb://127.0.0.1:27017/mydb')  
  .then(() => console.log('Connected to local MongoDB'))  
  .catch(err => console.error('MongoDB connection error:', err));
```

- `mongoose.connect()` → connects your Node.js app to MongoDB.
- `'mongodb://127.0.0.1:27017/mydb'`:

- 127.0.0.1 → local host (IPv4)
- 27017 → default MongoDB port
- mydb → database name
- .then() → runs if connection is successful
- .catch() → catches errors if connection fails

□ This ensures your API cannot run **CRUD operations unless MongoDB is connected.**

---

## 9 □ Root endpoint

```
app.get('/', (req, res) => {
  res.send('Welcome to Contact REST API!');
});
```

- Defines a **GET endpoint at /** (root URL).
  - req → the client request object
  - res → the response object
  - res.send() → sends a plain text response to the client
  - Example: visiting `http://localhost:3000/` shows `Welcome to Contact REST API!`.
- 

## □ Start the server

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

- Tells Express to **start listening for incoming HTTP requests** on the specified port.
  - () => console.log(...) → callback runs when the server starts successfully.
  - After this, your API is **live and ready to accept requests.**
- 

## □ Summary of flow

1. Import libraries (Express, Mongoose)
2. Create an Express app
3. Setup middleware to parse JSON
4. Import and mount routes
5. Connect to MongoDB
6. Define a root endpoint
7. Start server on a specified port

So after running `node index.js` or `npm run dev`:

- MongoDB connects
- Server listens at `http://localhost:3000`
- Routes under `/contacts` handle CRUD requests