Below is a **complete, working Contact Book REST API** exactly matching the practical question. This is **interview-ready code** with proper structure, CRUD, search, pagination, middleware, and error handling.

📁 Project Structure

```
contact-book/
│
├── controllers/
│   └── contact.controller.js
│
├── models/
│   └── contact.model.js
│
├── routes/
│   └── contact.routes.js
│
├── middlewares/
│   └── logger.middleware.js
│
├── config/
│   └── db.js
│
├── app.js
├── server.js
├── .env
└── package.json
```

📄 package.json

```json
{
  "name": "contact-book",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^8.0.0",
    "dotenv": "^16.3.1"
  }
}
```

📄.env

```
PORT=5000
MONGO_URI=mongodb://127.0.0.1:27017/contactbook
```

📄config/db.js

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

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI);
    console.log('MongoDB connected');
  } catch (error) {
    console.error('DB connection failed', error.message);
    process.exit(1);
  }
};

module.exports = connectDB;
```

📄models/contact.model.js

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

const contactSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    phone: { type: String, required: true },
    address: { type: String },
    isFavorite: { type: Boolean, default: false }
  },
  { timestamps: true }
);

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

📄middlewares/logger.middleware.js

```
const logger = (req, res, next) => {
  console.log(`${req.method} ${req.originalUrl}`);
  next();
};

module.exports = logger;
```

📄controllers/contact.controller.js

```
const mongoose = require('mongoose');
const Contact = require('../models/contact.model');

// Create Contact
exports.createContact = async (req, res) => {
  try {
    const { name, email, phone, address } = req.body;

    if (!name || !email || !phone) {
      return res.status(400).json({ message: 'Required fields missing' });
    }

    const exists = await Contact.findOne({ email });
    if (exists) {
      return res.status(409).json({ message: 'Email already exists' });
    }

    const contact = await Contact.create({ name, email, phone, address });
    res.status(201).json(contact);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

// Get All Contacts (Pagination)
exports.getContacts = async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 5;
    const skip = (page - 1) * limit;

    const contacts = await Contact.find()
      .sort({ createdAt: -1 })
      .skip(skip)
      .limit(limit);
    res.json(contacts);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

// Get Contact by ID
exports.getContactById = async (req, res) => {
  try {
    const { id } = req.params;

    if (!mongoose.Types.ObjectId.isValid(id)) {
      return res.status(400).json({ message: 'Invalid ID' });
    }

    const contact = await Contact.findById(id);
    if (!contact) {
      return res.status(404).json({ message: 'Contact not found' });
    }

    res.json(contact);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

// Update Contact
exports.updateContact = async (req, res) => {
  try {
    const { id } = req.params;

    const updated = await Contact.findByIdAndUpdate(id, req.body, {
      new: true
    });

    if (!updated) {
      return res.status(404).json({ message: 'Contact not found' });
    }

    res.json(updated);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

// Delete Contact
exports.deleteContact = async (req, res) => {
  try {
    const deleted = await Contact.findByIdAndDelete(req.params.id);

    if (!deleted) {
      return res.status(404).json({ message: 'Contact not found' });
    }

    res.json({ message: 'Contact deleted successfully' });
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

// Search Contacts
exports.searchContacts = async (req, res) => {
  try {
    const { name } = req.query;

    const results = await Contact.find({
      name: { $regex: name, $options: 'i' }
    });

    res.json(results);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};

// Get Favorites
exports.getFavorites = async (req, res) => {
  try {
    const favorites = await Contact.find({ isFavorite: true });
    res.json(favorites);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

📄routes/contact.routes.js

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

router.post('/', controller.createContact);
router.get('/', controller.getContacts);
router.get('/search', controller.searchContacts);
router.get('/favorites', controller.getFavorites);
router.get('/:id', controller.getContactById);
router.put('/:id', controller.updateContact);
router.delete('/:id', controller.deleteContact);
```

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

📄app.js

```
const express = require('express');
const logger = require('./middlewares/logger.middleware');

const app = express();

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

app.use('/api/contacts', require('./routes/contact.routes'));

module.exports = app;
```

📄server.js

```
require('dotenv').config();
const app = require('./app');
const connectDB = require('./config/db');

connectDB();

const PORT = process.env.PORT || 5000;

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

🔗APIs to Test in Postman

- POST `/api/contacts`
- GET `/api/contacts`
- GET `/api/contacts/:id`
- PUT `/api/contacts/:id`
- DELETE `/api/contacts/:id`
- GET `/api/contacts/search?name=a`
- GET `/api/contacts/favorites`

🕗Interview Tip (IMPORTANT)

Be ready to explain: - Why this folder structure - How async/await works here - How pagination is implemented - How errors are handled - What middleware does

---

If you want next: ✅Mock practical interview ✅Common mistakes interviewers point out ✅One-page revision checklist

Just tell me ♀️