1. **Setup Project Environment:**
    - **Initialize Project:** Run `npm init -y` to create a `package.json` file.
    - **Install Dependencies:** Run `npm install express mongoose cors dotenv express-async-handler`.
    - **Create Project Structure:** Create the following folders and files to organize your project:

```
/config
  - dbConnection.js
/controllers
  - contactController.js
/models
  - contactModel.js
/routes
  - contactRoutes.js
/middleware
  - errorHandler.js (for future use)
  - validateTokenHandler.js (for future use)
. env
index.js
```

2. **Configure Environment Variables:**
    - **. env File:** Create a `.env` file in the root of your project to store environment variables:

```
bash
Copy code
PORT=5001
CONNECTION_STRING=mongodb+srv://admin:1234@clus.rtibejf.mon
godb.net/mycontacts-
backend?retryWrites=true&w=majority&appName=Clus
ACCESS_TOKEN_SECRET=1234
```

3. **Database Connection:**
    - **dbConnection.js:** In the `config` folder, create `dbConnection.js` to handle MongoDB connection using Mongoose.

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

const connectDb = async () => {
  try {
    const connect = await
mongoose.connect(process.env.CONNECTION_STRING);
    console.log("Database connected: ",
connect.connection.host, connect.connection.name);
  } catch (err) {
    console.log(err);
    process.exit(1);
  }
};

module.exports = connectDb;
```

4. **Define the Data Model:**
    - **contactModel.js:** In the `models` folder, create `contactModel.js` to define the Mongoose schema and model for contacts.

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

const contactSchema = mongoose.Schema(
  {
    name: {
      type: String,
      required: [true, "Please add the contact name"],
    },
    email: {
      type: String,
      required: [true, "Please add the email address"],
    },
    phone: {
      type: String,
      required: [true, "Please add the phone number"],
    },
  },
  {
    timestamps: true,
  }
```

```
);

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

5. **Set Up Express Application:**
   - **index.js:** Create `index.js` (or `app.js`) to initialize the Express app, configure middleware, and start the server.

```
const express = require("express");
const connectDb = require("./config/dbConnection");
const cors = require("cors");
const dotenv = require("dotenv").config();

connectDb();
const app = express();

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

app.use(
  cors({
    origin: "http://localhost:5173", // Allow requests from
this origin
  })
);

app.use(express.json());

app.use("/api/contacts",
require("./routes/contactRoutes"));

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

6. **Create Routes:**
    - **contactRoutes.js:** In the `routes` folder, create `contactRoutes.js` to define the endpoints for contact operations.

```
const express = require("express");
const router = express.Router();
const {
  getContacts,
  createContact,
  getContact,
  updateContact,
  deleteContact,
} = require("../controllers/contactController");

// Define routes for CRUD operations
router.route("/").get(getContacts).post(createContact);
router.route("/:id").get(getContact).put(updateContact).del
ete(deleteContact);

module.exports = router;
```

7. **Implement Controllers:**
    - **contactController.js:** In the `controllers` folder, create `contactController.js` to implement functions for handling CRUD operations.

```
const asyncHandler = require("express-async-handler");
const Contact = require("../models/contactModel");

// Get all contacts
const getContacts = asyncHandler(async (req, res) => {
  const contacts = await Contact.find();
  res.status(200).json(contacts);
});

// Create new contact
const createContact = asyncHandler(async (req, res) => {
  const { name, email, phone } = req.body;
  if (!name || !email || !phone) {
    res.status(400);
    throw new Error("All fields are mandatory");
  }
  const contact = await Contact.create({ name, email, phone
});
```

```
  res.status(201).json(contact);
});

// Get contact by ID
const getContact = asyncHandler(async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) {
    res.status(404);
    throw new Error("Contact not found");
  }
  res.status(200).json(contact);
});

// Update contact
const updateContact = asyncHandler(async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) {
    res.status(404);
    throw new Error("Contact not found");
  }
  const updatedContact = await
Contact.findByIdAndUpdate(req.params.id, req.body, { new:
true });
  res.status(200).json(updatedContact);
});

// Delete contact
const deleteContact = asyncHandler(async (req, res) => {
  const contact = await Contact.findById(req.params.id);
  if (!contact) {
    res.status(404);
    throw new Error("Contact not found");
  }
  await Contact.deleteOne({ _id: req.params.id });
  res.status(200).json(contact);
});

module.exports = {
  getContacts,
  createContact,
  getContact,
  updateContact,
  deleteContact,
};
```

8. **Middleware (Optional for now):**
    - **errorHandler.js and validateTokenHandler.js:** Create these files in the `middleware` folder for future use if you plan to add error handling and authentication.
9. **Start the Server:**
    - **Run Server:** Use `node index.js` or `nodemon index.js` (if you have nodemon installed) to start the server. Ensure that the backend is running and connected to the database.
10. **Testing:**

- **Test Endpoints:** Use tools like Postman or Insomnia to test all routes (GET, POST, PUT, DELETE) to ensure they work as expected.

By following these detailed steps, you can sequentially set up and implement the contact management system backend, ensuring a well-organized and functional application.