Node.js and npm (or yarn) installed on your system. You can verify this by running `node -v` and `npm -v` (or `yarn -v`) in your terminal. If not installed, download them from the official Node.js website https://nodejs.org/en.

Steps:

1. **Create a Project Directory:**
   - Open your terminal and navigate to your desired workspace directory using `cd`.
   - Create a new directory for your project: `mkdir my-crud-api`
   - Change into the newly created directory: `cd my-crud-api`
2. **Initialize a Node.js Project:**
   - Use `npm init -y` (or `yarn init -y`) to create a basic `package.json` file. This file stores project metadata and dependencies. The `-y` flag fills in default values.
3. **Install Dependencies:**
   - Install the Express.js framework, a popular choice for building web applications and APIs in Node.js:

     Bash
     ```
     npm install express
     ```
   - You might also consider additional dependencies depending on your database choice and preferences:
     - For body parsing (to handle incoming request data): `body-parser`
     - For database connection (e.g., for MongoDB: `mongoose`, for MySQL: `mysql2`): Choose the appropriate package for your database.
     - For validation (to ensure data integrity): `joi`
4. **Create a Basic Server (app.js):**
   - Create a file named `app.js` (or your preferred name) in the project root.
   - Add the following code to set up a basic Express server and listen for incoming requests:

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

```
const port = process.env.PORT || 3000; // Use environment
variable or default to 3000

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

   - This code imports Express, creates an Express application instance, sets a port number (you can use an environment variable for flexibility), and starts the server, logging a message when it's ready.

5. **Define API Endpoints (routes/*.js):**
   - Create a `routes` directory to organize your API routes.
   - Inside `routes`, create files for each resource you want to manage in your API (e.g., `users.js`, `posts.js`).
   - Each route file defines CRUD operations (Create, Read, Update, Delete) using Express methods:

     JavaScript
     ```
     const express = require('express');
     const router = express.Router();

     // Example: Get all users (Read)
     router.get('/users', (req, res) => {
       // Implement logic to retrieve users from database or
     elsewhere
       res.json({ message: 'Get all users' });
     });

     // ... Add similar logic for other CRUD operations (Create,
     Update, Delete)

     module.exports = router;
     ```
   - Remember to replace the placeholder logic with code to interact with your chosen database or data source.

6. **Connect to a Database**
   - If your API requires storing data persistently, choose a database (e.g., MongoDB, MySQL) and install the appropriate Node.js package (mentioned in step 3).
   - Implement logic in your route handlers to connect to the database, perform CRUD operations, and send appropriate responses.