**Prerequisites**

1. **Node.js** and **npm** installed on your machine.
2. **MongoDB** installed or access to a MongoDB Atlas instance.

**Step 1: Set Up the Project**

Create a new directory for your project and navigate into it:

mkdir crud-app

cd crud-app

Initialize a new Node.js project:

npm init -y

Install the necessary dependencies:

npm install express mongoose body-parser

**Step 2: Create the MongoDB Schema and Model**

Create a new file called models/Item.js:

// models/Item.js

const mongoose = require('mongoose');

const ItemSchema = new mongoose.Schema({

name: {

type: String,

required: true,

},

quantity: {

type: Number,

required: true,

min: 1,

},

price: {

type: Number,

required: true,

min: 0,

},

});

module.exports = mongoose.model('Item', ItemSchema);

**Step 3: Set Up the Express Server**

Create a new file called server.js:

javascript

// server.js

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const cors = require('cors');

const Item = require('./models/Item');

const app = express();

// Middleware

app.use(bodyParser.json());

//cross-origin resource sharing

app.use(cors());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/crud-app', { useNewUrlParser: true, useUnifiedTopology: true })

.then(() => console.log('MongoDB connected'))

.catch(err => console.log(err));

// Routes

// Create an item

app.post('/additems', async (req, res) => {

try {

const newItem = new Item(req.body);

const savedItem = await newItem.save();

res.status(201).json(savedItem);

} catch (error) {

res.status(400).json({ error: error.message });

}

});

// Read all items

app.get('/getitems', async (req, res) => {

try {

const items = await Item.find();

res.status(200).json(items);

} catch (error) {

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

}

});

// Read a single item by ID

app.get('/items/:id', async (req, res) => {

try {

const item = await Item.findById(req.params.id);

if (!item) {

return res.status(404).json({ error: 'Item not found' });

}

res.status(200).json(item);

} catch (error) {

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

}

});

// Update an item by ID

app.put('/items/:id', async (req, res) => {

try {

const updatedItem = await Item.findByIdAndUpdate(req.params.id, req.body, { new: true, runValidators: true });

if (!updatedItem) {

return res.status(404).json({ error: 'Item not found' });

}

res.status(200).json(updatedItem);

} catch (error) {

res.status(400).json({ error: error.message });

}

});

// Delete an item by ID

app.delete('/items/:id', async (req, res) => {

try {

const deletedItem = await Item.findByIdAndDelete(req.params.id);

if (!deletedItem) {

return res.status(404).json({ error: 'Item not found' });

}

res.status(200).json({ message: 'Item deleted' });

} catch (error) {

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

}

});

// Start the server

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

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

**Step 4: Run the Application**

Make sure your MongoDB server is running. If you're using MongoDB locally, start the MongoDB service:

mongod

Then, run your Node.js server:

node server.js

Your server should now be running on http://localhost:5000. You can use a tool like [Postman](https://www.postman.com/) to test the CRUD operations by sending HTTP requests to the appropriate endpoints.

**CRUD Endpoints**

* **Create**: POST /items
  + Request body: { "name": "Item Name", "quantity": 10, "price": 20 }
* **Read all**: GET /items
* **Read one**: GET /items/:id
* **Update**: PUT /items/:id
  + Request body: { "name": "Updated Name", "quantity": 5, "price": 15 }
* **Delete**: DELETE /items/:id

**Step 5: Frontend (Optional)**

You can build a simple frontend using HTML, CSS, and JavaScript to interact with this backend. For simplicity, I'll provide a basic example that uses vanilla JavaScript to perform CRUD operations.

Create an index.html file:

html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CRUD App</title>

<style>

/\* Add your styling here \*/

</style>

</head>

<body>

<h1>CRUD App</h1>

<div id="item-form">

<input type="text" id="name" placeholder="Name" required>

<input type="number" id="quantity" placeholder="Quantity" required>

<input type="number" id="price" placeholder="Price" required>

<button onclick="createItem()">Add Item</button>

</div>

<ul id="items-list"></ul>

<script>

const API\_URL = 'http://localhost:5000/items';

// Fetch and display items

async function fetchItems() {

const response = await fetch(API\_URL);

const items = await response.json();

const itemsList = document.getElementById('items-list');

itemsList.innerHTML = '';

items.forEach(item => {

const li = document.createElement('li');

li.textContent = `${item.name} - Quantity: ${item.quantity} - Price: $${item.price}`;

itemsList.appendChild(li);

});

}

// Create a new item

async function createItem() {

const name = document.getElementById('name').value;

const quantity = document.getElementById('quantity').value;

const price = document.getElementById('price').value;

const response = await fetch(API\_URL, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ name, quantity, price }),

});

if (response.ok) {

fetchItems();

}

}

// Initialize the list

fetchItems();

</script>

</body>

</html>

This simple HTML page allows you to create and view items. It interacts with the Express backend you created earlier.

**Conclusion**

This is a basic CRUD application using MongoDB, Node.js, Express, and a minimal frontend with vanilla JavaScript. It covers the key aspects of the project statement:

* **CRUD Operations**: All four operations (Create, Read, Update, Delete) are implemented.
* **Data Validation**: Schema validation is done via Mongoose.
* **Document-Oriented Database**: The application follows the principles of MongoDB as a document-oriented database.
* **Error Handling**: Basic error handling is implemented for each CRUD operation.

You can further extend this application by adding more features, improving the UI, or deploying it to a cloud service like Heroku.