Build Crud API with Node.js, Express, and MongoDB

We'll be building a CRUD App with Node.js, Express, and MongoDB. We'll use Mongoose for interacting with the MongoDB instance.

Currently, most of the websites operate on an API-based backend structure, where we just send a request from the front end of the website by calling an API and obtaining the required results. In this blog, we are going to build a simple CRUD (Create, Read, Update and Delete) app with the application of Node JS, Express JS, and MongoDB from the basics. Before we jump into the application, let's look into the tools we are going to use.

Express is one of the most popular web frameworks for node.js. It is built on top of the node.js HTTP module and adds support for routing, middleware, view system, etc. It is very simple and minimal, unlike other frameworks that try to do way too much, thereby reducing the flexibility for developers to have their own design choices.

Mongoose is an ODM (Object Document Mapping) tool for Node.js and MongoDB. It helps you convert the objects in your code to documents in the database and vice versa. Mongoose provides a straightforward, schema-based solution to model your application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

CRUD is an acronym for Create, Read, Update and Delete. It is a set of operations we get servers to execute (`POST`, `GET`, `PUT` and `DELETE` requests respectively). This is what each operation does:

- **Create (POST)** - Make something
- **Read (GET)**- Get something
- **Update (PUT)** - Change something
- **Delete (DELETE)**- Remove something

we'll heavily use ES6 features like `let`, `const`, `arrow functions`, `promises` etc. It's good to familiarize yourself with these features.

we'll be building a CRUD App with Node.js, Express, and MongoDB. We'll use Mongoose for interacting with the MongoDB instance.

Step 1: Creating the Application

Fire up your terminal and create a new folder for the application.

```
mkdir crud-node-express
```

Initialize the application with a package.json file

Go to the root folder of your application and type `npm init` to initialize your app with a `package.json` file.

```
cd crud-node-express
npm init
```

Note that I've specified a file named `server.js` as the entry point of our application. We'll create `server.js` file in the next section.

Step 2: Install dependencies

We will need express, mongoose, and body-parser modules in our application. Let's install them by typing the following command:

```
npm install express body-parser mongoose --save
```

Setting up the Web Server

Let's now create the main entry point of our application. Create a new file named `server.js` in the root folder of the application with the following contents:

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

const app = express();

app.use(bodyParser.urlencoded({ extended: true }))

app.use(bodyParser.json())

app.get('/', (req, res) => {
    res.json({"message": "Hello Crud Node Express"});
});

app.listen(3000, () => {
    console.log("Server is listening on port 3000");
});
```

First, We import express and body-parser modules. Express, as you know, is a web framework that we'll be using for building the REST APIs, and body-parser is a module that parses the request (of various content types) and creates a `req.body` object that we can access in our routes.

Then, We create an express app and add two `body-parser` middlewares using express's `app.use()` method. A middleware is a function that has access to `request` and `response` objects. It can execute any code, transform the request object, or return a response.

Then, We define a simple route that returns a welcome message to the clients.

Finally, We listen on port 3000 for incoming connections.