Essay Writing Contest on "From Data to Application: A Hands-on Guide to CRUD Operations with Mongoose and Express.js"

ডেটা থেকে অ্যাপ্লিকেশন পর্যন্ত:Mongoose এবং express.js এর সাথে CRUD অপারেশনের নির্দেশনা

সূচনাঃ আজকের ওয়েব ডেভেলপমেন্টের জগতে, শক্তিশালী এবং দক্ষ অ্যাপ্লিকেশন তৈরি করা অত্যন্ত গুরুত্বপূর্ণ। যেকোনো অ্যাপ্লিকেশনের একটি মৌলিক দিক হল ডেটার উপর CRUD অপারেশন(Create, Read, Update, Delete) করার ক্ষমতা। এই প্রবন্ধে, আমরা দুটি শক্তিশালী টুল ব্যবহার করে CRUD অপারেশন বাস্তবায়নের জন্য একটি হ্যান্ডস-অন গাইড অন্বেষণ করব: Mongoose এবং Express.js। এই প্রযুক্তিগুলি, একত্রিত হলে, একটি MongoDB ডাটাবেসে ডেটা পরিচালনার জন্য একটি কার্যকর সমাধান প্রদান করে।

Express.js: Express.js হল Node.js-এর জন্য একটি জনপ্রিয় ওয়েব অ্যাপ্লিকেশন ফ্রেমওয়ার্ক যা ওয়েব অ্যাপ্লিকেশন এবং API তৈরির প্রক্রিয়াকে সহজ করে। এটি রাউটিং, মিডলওয়্যার তৈরি ইত্যাদি সহজ করে। এটি বৈশিষ্ট্যগুলির একটি শক্তিশালী সেট অফার করার সময় ওয়েব ডেভেলপমেন্টের জন্য একটি হালকা ওজনের এবং ন্যূনতম পদ্ধতি প্রদান করে।

ডেটাবেস এবং ORM(Object Relational Mappers) Node.js ব্যবহার করে ওয়েব অ্যাপ্লিকেশন তৈরিতে গুরুত্বপূর্ণ ভূমিকা পালন করে। যেমন বর্ণনা করা হয়েছে, একটি ডাটাবেস হল ডেটার একটি সংগ্রহ যা একটি নির্দিষ্ট পদ্ধতিতে সংগঠিত হয় যাতে সহজে অ্যাক্সেস, ব্যবস্থাপনা এবং তথ্য আপডেট করা যায়। একটি Node.js অ্যাপ্লিকেশনে, ডেটাবেসগুলি ডেটা সংরক্ষণ এবং পুনরুদ্ধার করতে ব্যবহৃত হয়।একটি ওআরএম হল একটি প্রোগ্রামিং কৌশল যা রিলেশনাল ডাটাবেস টেবিলে বস্তুকে ম্যাপ করে। ডাটাবেসের সাথে ইন্টারঅ্যাক্ট করার জন্য প্রয়োজনীয় কোডের পরিমাণ কমাতে এবং নিরাপত্তার একটি অতিরিক্ত স্তর প্রদান করতে ওআরএম সাহায্য করে।

Mongoose হল Node.js-এর জন্য একটি জনপ্রিয় ORM যা অ্যাপ্লিকেশন ডেটা মডেল করার জন্য একটি স্কিমা-ভিত্তিক সমাধান প্রদান করে। মঙ্গুজ ডেভেলপারদের তাদের ডেটার জন্য স্কিমা এবং মডেল নির্ধারণ করার অনুমতি দিয়ে MongoDB-এর সাথে মিথস্ক্রিয়া সহজ করে।

Mongoose : Mongoose মূলত একটি প্যাকেজ যা NodeJS অ্যাপ্লিকেশন এবং MongoDB সার্ভারের মধ্যে মধ্যস্থতাকারী হিসাবে কাজ করে। এটি একটি অবজেক্ট ডকুমেন্ট ম্যাপার (ODM) যা আমাদেরকে একটি মঙ্গোডিবি নথিতে ম্যাপ করা একটি দৃঢ়ভাবে টাইপ করা স্কিমা সহ বস্তুগুলিকে সংজ্ঞায়িত করতে দেয়। মঙ্গুজ সমস্ত CRUD ক্রিয়াকলাপকে সমর্থন করে - তৈরি করা, পুনরুদ্ধার করা, আপডেট করা এবং মুছে ফেলা।

Implementing CRUD Operations:

এই পর্যায়ে Express.js, Mongoose এবং জাভাস্ক্রিপ্টের কনসেপ্ট ব্যবহার করেই CRUD অ্যাপ্লিকেশনটি তৈরি করব।CRUD হচ্ছে Create, Read, Update, Delete এর সংক্ষিপ্ত রূপ।

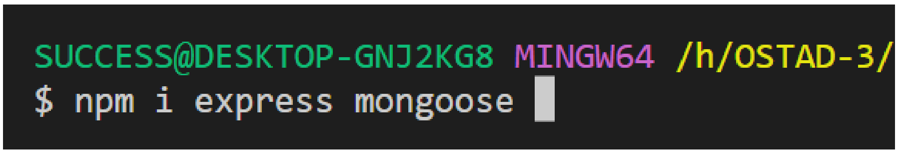
প্রজেক্টি করার জন্য আপনার কম্পিউটারে Node.js ইন্সটল থাকতে হবে। আপনার project এর জন্য নতুন ফোল্ডার তৈরি করুন এবং Visual Studio Code এ open করুন।

- **প্রথমে terminal এ (npm init –y) command দিয়ে package.json file তৈরি করতে হবে**

```
SUCCESS@DESKTOP-GNJ2KG8 MINGW64 /h/OSTAD-3/
$ npm init --y
```

```json
{
  "name": "crud",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",

  "description": ""
}

```

- **তারপর টার্মিনাল দ্বারা express এবং mongoose ইনস্টল করুন**



```
SUCCESS@DESKTOP-GNJ2KG8 MINGW64 /h/OSTAD-3/
$ npm i express mongoose
```

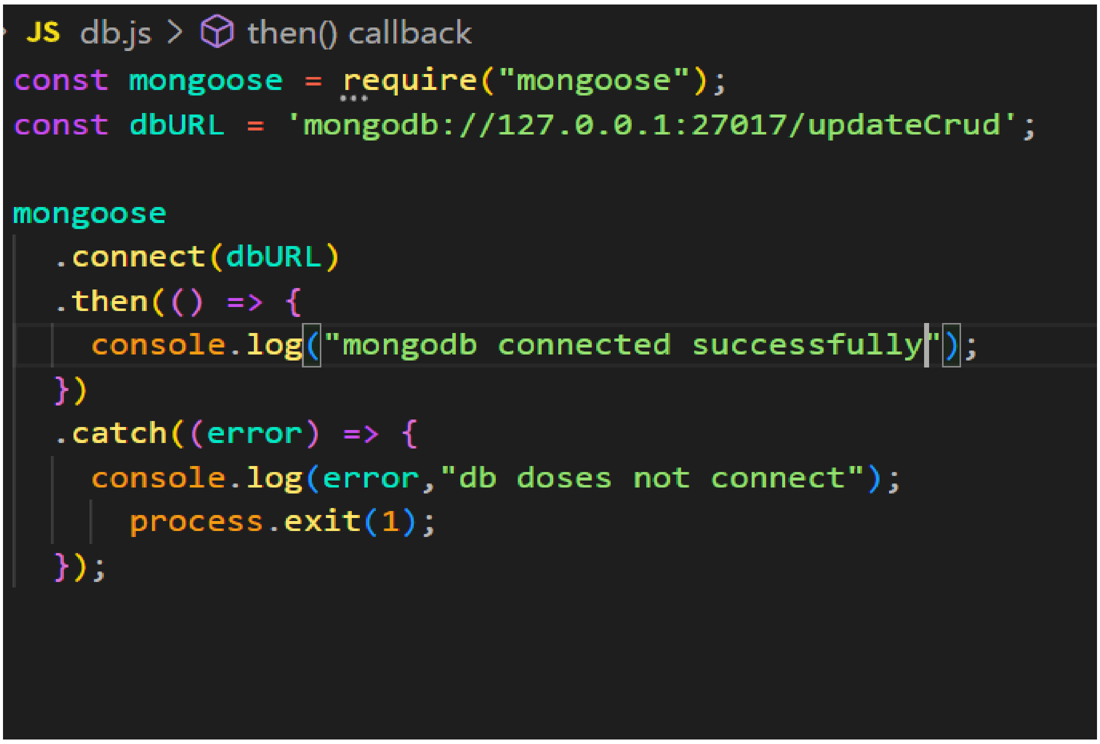
- **Project file setup and run server**

app.js নামে নতুন ফাইল তৈরি করতে হবে যা প্রধান ফাইল হিসেবে কাজ করবে এবং runserver হিসেবে কাজ করে

```javascript
const express = require('express');
const app = express();
const router = require('./route/user.route');

//middleware
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

//routing
app.use('/users', router);

//home
app.get('/', (req, res) => {
  res.send('home page');
});

const PORT = 8000;
app.listen(PORT, () => {
  console.log(`Server run at port ${PORT}`);
});

```

ফাইল রান করার জন্য terminal এ **node app.js** command দিতে হবে

ডাটাবেস কানেকশনঃ



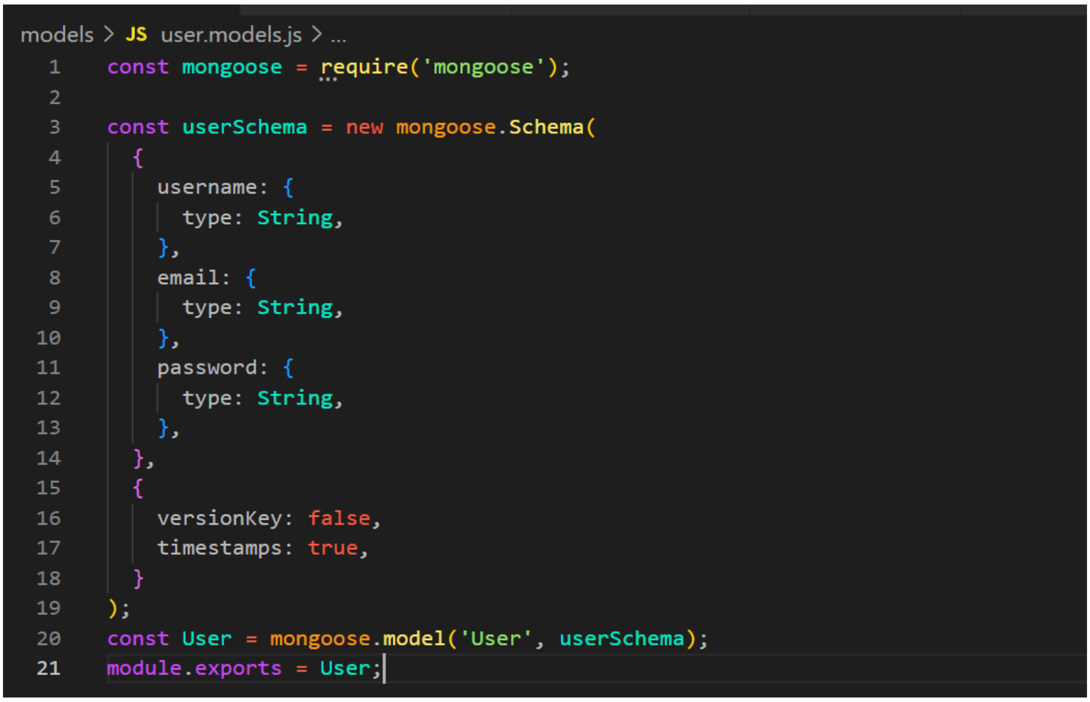
```javascript
const mongoose = require("mongoose");
const dbURL = 'mongodb://127.0.0.1:27017/updateCrud';

mongoose
  .connect(dbURL)
  .then(() => {
    console.log("mongodb connected successfully");
  })
  .catch((error) => {
    console.log(error,"db doses not connect");
      process.exit(1);
  });
```

ডাটাবেস সেট করার পর আবার terminal এ **node app.js** command দিতে হবে

ভাল্ভাবে ডাটাবেস কানেক্ট হলে mongodb connected successfully মেসেজ আসবে

Create Model & Schema:

প্রথমে আমরা ডাটাবেসের স্কীমা তৈরি করব, স্কীমা থেকে মডেল কল করতে হবে ।MongoDB এবং Node.js এর প্রসঙ্গে, Mongoose হল একটি অবজেক্ট ডেটা মডেলিং (ODM) লাইব্রেরি যা MongoDB ডাটাবেসের সাথে ইন্টারঅ্যাক্ট করার জন্য একটি উচ্চ-স্তরের বিমূর্ততা প্রদান করে। এটি আপনাকে আপনার ডেটা গঠন এবং পরিচালনা করার জন্য স্কিমা এবং মডেলগুলিকে সংজ্ঞায়িত করতে দেয়। আসুন জেনে নেওয়া যাক কিভাবে মঙ্গুজ মডেল তৈরি করা যায়।



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

const userSchema = new mongoose.Schema(
  {
    username: {
      type: String,
    },
    email: {
      type: String,
    },
    password: {
      type: String,
    },
  },
  {
    versionKey: false,
    timestamps: true,
  }
);
const User = mongoose.model('User', userSchema);
module.exports = User;
```

উপরের উদাহরণে আমরা দেখতে পাচ্ছি userSchema নামে একটি স্কীমা তৈরি করতে

```
Const userSchema  = new mongoose.Schema({

        Username: {type: String},

        email:{type:String}

});
```

স্কীমা ডিক্লেয়ার করার পর মনগোজ থেকে মডেল কল করতে হবে,

```
const User = mongoose.model('User', userSchema);
```

Define the routes:

রুটগুলি ক্লায়েন্টের অনুরোধগুলি পরিচালনা করার উপায় নির্ধারণ করে। অ্যাপ্লিকেশনের বিভিন্ন URL-এ ব্যবহারকারীর নেভিগেশন রুট দ্বারা পরিচালিত হয়। রুট নিম্নলিখিত গঠন নিতে

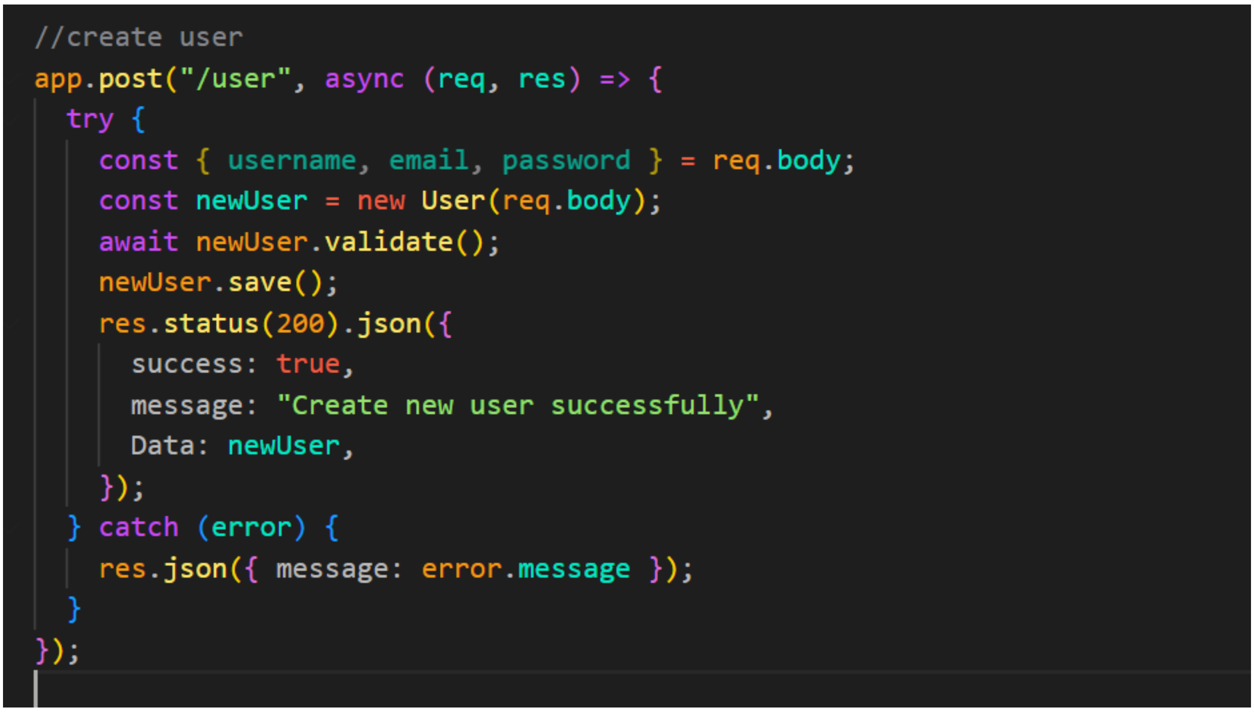
app.METHOD(PATH, HANDLER)

- app হল এক্সপ্রেসের উদাহরণ

- METHOD হল একটি HTTP অনুরোধ পদ্ধতি। বিভিন্ন ধরনের পদ্ধতি হল GET, POST, PUT, DELETE ইত্যাদি।
- PATH হল সার্ভারের পাথ বা শেষ বিন্দু।
- HANDLER হল ফাংশন যখন এন্ডপয়েন্ট মিলে যায়

Create Route:

এখানে বর্ণনা করছি কিভাবে একটি Document Create করতে পারি। বরাবরের মতোই এখানে Express.js, JavaScript এবং Mongoose এই তিনটির সমন্বয়েই API বানানো হয় ।ডাটা Create করার জন্য ব্যবহার করছি HTTP POST Method ব্যবহার করব



```
//create user
app.post("/user", async (req, res) => {
  try {
    const { username, email, password } = req.body;
    const newUser = new User(req.body);
    await newUser.validate();
    newUser.save();
    res.status(200).json({
      success: true,
      message: "Create new user successfully",
      Data: newUser,
    });
  } catch (error) {
    res.json({ message: error.message });
  }
});
```

res.body থেকে আমরা Data Destructuring করছি যা JavaScript এর কোড ।এখানে User হলো মডেল যার স্কীমা আগে ডিক্লেয়ার করা হইছে

উপরের কোডটি যা করে তা হল আমরা একটি পোস্ট অনুরোধ পাঠাই যাতে ডেটা থাকে। req.body ধারণ করে আমরা ফ্রন্টএন্ড থেকে যে ডেটা পাস করি। আমরা বডি-পার্সার ব্যবহার করে ডেটা পার্স করি এবং ভেরিয়েবলে পাঠাই। তারপরে আমরা আমাদের তৈরি ঠিকানা মডেলের একটি নতুন উদাহরণ তৈরি করি এবং collection.save পদ্ধতি ব্যবহার করে এটি ডিবিতে সংরক্ষণ করি।

Read Route:

MongoDB-তে তৈরি প্রতিটি নথির জন্য, MongoDB দ্বারা বরাদ্দ করা একটি অবজেক্ট আইডি থাকবে এবং অবজেক্ট আইডি হবে অনন্য। ডাটাবেস থেকে ব্যবহারকারীর ডেটা পুনরুদ্ধার করার জন্য আমরা অবজেক্ট আইডি ব্যবহার করি।

```
//read User
app.get('/user', async (req, res) => {
  try {
    const getAll = await User.find();
    res.status(200).json(getAll);
  } catch (error) {
    res.json({ message: error.message });
  }
});
```

এখানে async-await ব্যবহার করা হয়েছে, কারন আমি চাচ্ছি আমার কোড গুলো যেন লাইন বাই লাইন এক্সিকিউট হয় অর্থাৎ অ্যাসিঙ্ক্রোনাস ভাবে কাজ করে। যেহেতু এখানে ডাটাবেজের সাথে আমি কাজ করছি তাই একটা কাজ হতে কিছুক্ষণ সময় লাগতে পারে, User.find() ব্যবহার করা হইছে কারণ ডটাবেস থেকে ডাটা গুলো খুজার জন্য mongoose method “find” ব্যবহার করা হইছে

async/await এর ভিতরে try-catch ব্যবহার করা হয়েছে। যদি আমাদের অ্যাপ্লিকেশনে কোনরকম এরর হয় সেই ক্ষেত্রে এররটা catch ব্লকের মধ্যে চলে যাবে এবং এর ফলে এপ্লিকেশন ক্রাশ করবে না। Error handling করার জন্য try-catch ব্যবহার করা একটি বেস্ট প্রাকটিস।

Update Route:

ব্যবহারকারীকে আপডেট করার জন্য আমরা ব্যবহারকারীর অবজেক্ট আইডি ব্যবহার করব। আমরা অনুরোধের একটি প্যারামিটার হিসাবে ব্যবহারকারীর অবজেক্ট আইডি পাস করি।

ডাটা আপডেট করার জন্য PUT এবং PATCH দুইটি HTTP Method রয়েছে। PATCH হচ্ছে ডকুমেন্টের কোন নির্দিষ্ট Field এর পরিবর্তন, আর PUT হচ্ছে সম্পূর্ণ নতুন ভাবে ডকুমেন্ট কে পরিবর্তন। সাধারন দৃষ্টিতে উভয়ের কাজ একই রকম মনে হলেও Rest API এর বেস্ট প্র্যাকটিস অনুসরণ করলে যখন যেটা প্রয়োজন ঐটাই ব্যবহার করা উচিৎ। আপডেট করার সময় পাথ প্যারামিটার হিসেবে Id নিব কারণ আইডি ইউনিক মান। set হলো mongoose এর method যা ডাটাবেসের যে মান গুলো change হবে সেগুলো এর মধ্যে লিখা

```
//updateUser
app.put("/user/:id", async (req, res) => {
  try {
    const { username, email, password } = req.body;
    const { id } = req.params;
    const update = await User.findByIdAndUpdate(id,
      {
        $set: {username,email,password,},
      }, {new: true,} );
    res.status(200).json({
      success: true,
      message: "update user successfully",
      Data: update,
    });
  } catch (error) {
    res.json({ message: error.message });
  }
});
```

Delete Route:

ব্যবহারকারীর অবজেক্ট আইডি ব্যবহার করে ব্যবহারকারীকে মুছে ফেলাও করা হয়। ব্যবহারকারীকে মুছে ফেলার জন্য আমরা অনুরোধের একটি প্যারামিটার হিসাবে ব্যবহারকারীর অবজেক্ট আইডি পাস করি। ডাটা মুছে ফেলার জন্য Delete Method ব্যবহার হয়।

```
////deleteUser
app.delete('/user/:id', async (req, res) => {
  try {
    const { id } = req.params;
    const deleteUser = await User.findByIdAndDelete(id);
    res.status(200).send(deleteUser);
  } catch (error) {
    res.json({ message: error.message });
  }
});
```

উপরের কোডে, আমরা অনুরোধে যে ব্যবহারকারীকে মুছতে চাই তার আইডি পাস করি। আমাদের পাস করা অবজেক্ট আইডি সহ কোনও ব্যবহারকারী থাকলে, ব্যবহারকারীকে ডিবি(DB) থেকে মুছে ফেলা হবে এবং আপনি 'ব্যবহারকারী মুছে ফেলা' হিসাবে প্রতিক্রিয়া পাবেন। ডাটাবেজ থেকে কোন ডকুমেন্ট টি মুছে ফেলতে চাই তার id টি req.params থেকে destructuring করে নিলাম। User.findByIdAndDelete() হচ্ছে এখানে Mongoose এর অংশ, যা মূলত একটি মেথড এবং এই মেথডের ভিতরে শুধু id টা পাস করে দিলেই ডাটাবেজ থেকে উক্ত id এর সাথে যে ডকুমেন্ট ম্যাচ হবে তা Delete হয়ে যাবে।

Conclusion:

আমরা Node.js, Expess.js এবং MongoDB ব্যবহার করে একটি সাধারণ CRUD (Create, Delete, Update, Delete) অ্যাপ তৈরি করেছি। আমাদের উচিৎ Rest API ডেভেলপমেন্টের জন্য যে বেস্ট প্র্যাকটিস এবং কনভেনশন গুলো আছে তা অনুসরণ করা। ব্লগটি পড়ার জন্য ধন্যবাদ।