Episode - 6

Database, Schema & Models - Mongoose

1) How to connect MongoDB Database to your application?
==

Step1: creating a "config" folder & "database.js" file.

Step 2: Installation of Mongoose

-> Mongoose is a library that helps developers work with mongoDB by providing a Schema - based way to model data.

-> In Mongoose everything is schema based. for every MongoDB collection we create seperate schema.

-> The schema defines the structure of the document along with data type for each collection.

npm i mongoose

↳ It install the mongoose & include it in our app dependency.

Step 3: Connecting to DB cluster

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

const connectDB = async () => {
    await mongoose.connect( your MongoDB connection String + DB name )
}

connectDB().then(() => { console.log(" DB connected")
    })
    .catch((err) => { console.log(" Failed to connect")
    })
```

-> Mongoose have connect fn which accept a MongoDB connection

string to connect to the cluster and return a promise.

Step 4: Placing / Importing the DB file to app.js

```
//app.js
const DB = require("./config/database")
```

→ whenever we require the DB file all the code from DB file is read inside app.js file.

correct way:

→ 1st our app should connect to DB & then only it should start the server. So instead of calling the connectDB fn in the Database.js we just export that fn and import in app.js & connect to the server only when DB connection is successful.

eg:

```
//app.js
const {connectDB} = require("./config/database")

connectDB().then(() => {
    console.log("DB connected successfully")
    //Start the server
    app.listen(7777, () => {
        console.log("Server started")
    })
}).catch((err) => {
    console.log("error")
})
```

→ Best practice is to place connection string in .env file and access it using "dotenv" package.

2) What is Schema and How to create it?

Schema:

→ Schema is like a blueprint for the document in each collection

→ It defines how the document should have what are the fields each document should contain, what datatype it should be, whether the field is required (or) not, the length of each field & so on.

Creation of Schema

→ we create schema using mongoose.Schema() fn which accept an obj where we define each field of the document along with its datatype.

→ we create Model based on Schema. Model is what actually interacts with DB. It represents a MongoDB collection along with all the operations that can be performed on that collection.

→ we create model using mongoose.model(ModelName, SchemaName)

Model Name should always starts with Capital letter.

eg:

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

const userSchema = new mongoose.Schema({
    name: {
        type: String, required: true
    },
    email: {
        type: String, required: true
    }
})

const User = mongoose.model("User", userSchema)

module.exports = User.
```

3) How to add data manually in the database through API call?

Step:1 Creation of schema & model

Step:2 Using it to create a new document

→ we create a new document by creating a new instance of the model we defined & Pass the fields & data to it by new ModelName

→ we use Save() method to save the new instance to the DB.

→ Save() methods returns a promise so we need to await the response & send the response to the client.

eg:

```
app.post("/signup", async(req, res) => {
    const user = new User({
        name: "Shan",
        email: "test@gmail.com"
    })
    await user.save() → saving it to DB
    res.send("user created successfully")
})
```

→ New instance of User Model

→ we use Post http method since we are adding the data to DB.

⇒ when we hit the route with post method it will add this new document to user collection.

4) what are "_id" & "__v" field in the document?

→ Both fields are automatically created by MongoDB.

→ _id : id fields are used to uniquely identify the document.

→ __v : this field is used to track the version of the document whenever the doc is updated the version is incremented by default it is 0.

—— x ——