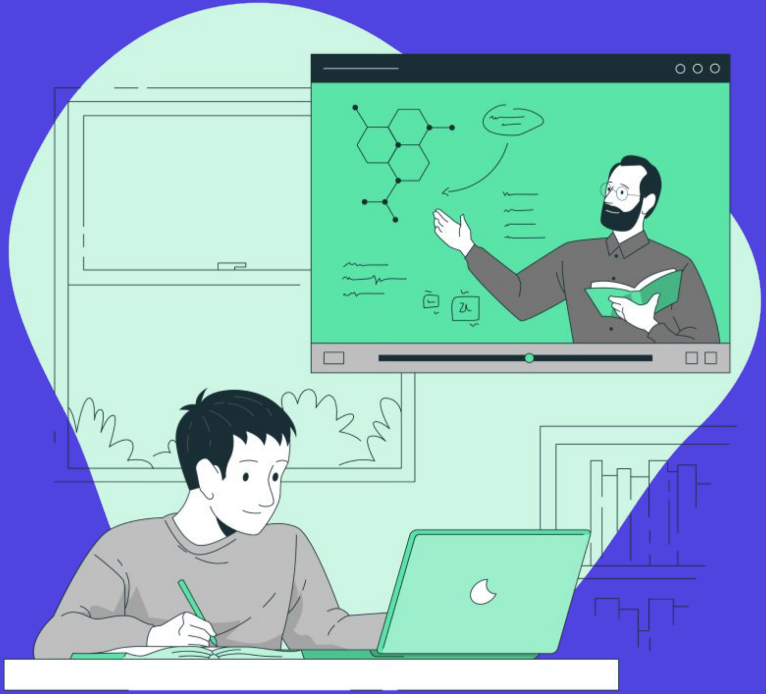
Using Mongoose

Relevel
by Unacademy



Topic to be covered

- Create Schema
- Create the database in MongoDB programmatically
- Create the collection in MongoDB programmatically
- Putting inbuilt constraints on the model schema
- Creating custom validators for the model schema
- CRUD operations with MongoDB using Mongoose

Mongoose

Mongoose is an elegant MongoDB object modeling for node.js.

Mongoose offers a straightforward, schema-based solution for modeling your application data. It includes built-in type casting, certification, question building, business logic hooks, and more.

Install Node.js and MongoDB before going further.

Install Mongoose from the command line using the next npm:

```
npm install mongoose --save
```

Init Project

Init your project

```
npm init
```

npm then asks me some questions and creates a package.json file for me. Then I start building the project.

Create index.js and require mongoose

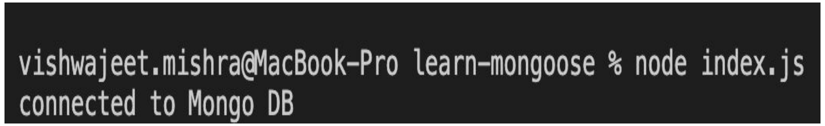
```
/**
* This file will have
the logic to play with
the mongodb
*/

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

Connect to MongoDB

Mongoose lets you start using your models right away, without having to wait for the mongoose to establish a connection to MongoDB.
This is because the mongoose buffers call the model function internally. This buffering is convenient, but a common source of confusion. If you use the model without connecting, the mongoose will not provide an error by default.

```
/**
* Making a connection to the MongoDB
*/
mongoose.connect("mongodb://localhost/demodb", () => {
   console.log("connected to Mongo DB ")
}, err => {
   console.log("Error :", err.mssage)
}
);
```

Run index.js to check if you are able to connect to MongoDB.



```
vishwajeet.mishra@MacBook-Pro learn-mongoose % node index.js
connected to Mongo DB
```

Create Schema

Create a folder name models
Create a file name students.model.js
Create a schema name studentSchema1 using mongoose

Schema maps to a MongoDB collection and defines the shape of the documents within that collection.

Here student contains the name of the typed string and the age of the type number.

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

/**
* Version 1 of the schema
*/
const studentSchema1 = new mongoose.Schema({
   name: String,
   age: Number
})


module.exports = mongoose.model("Student",
studentSchema1);
```

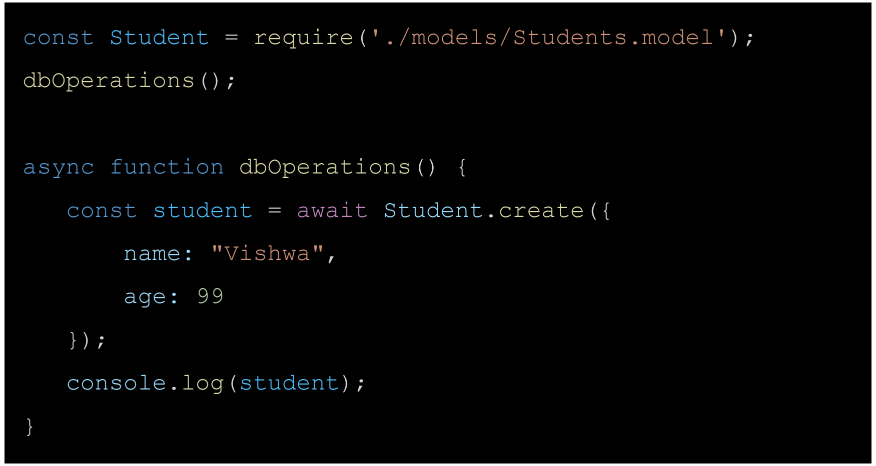
Create Schema

Add these lines in index.js

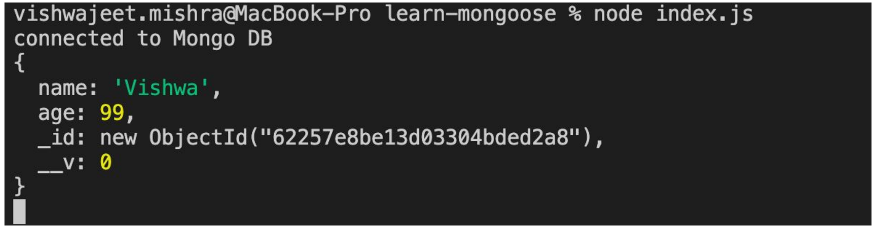
First, require the student schema in index.js. Here we are creating a function dboperations which will create an entry in the student Model with name = "Vishwa" and age = 99.

Run index.js

You will get the output of the Student Collection.



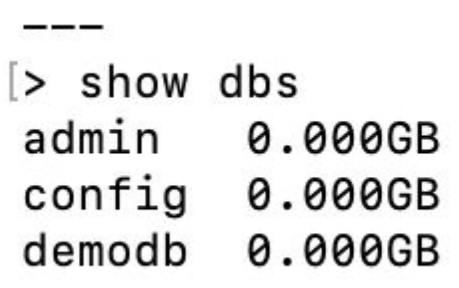
```
const Student = require('./models/Students.model');
dbOperations();


async function dbOperations() {
   const student = await Student.create({
        name: "Vishwa",
        age: 99
   });
   console.log(student);
}
```



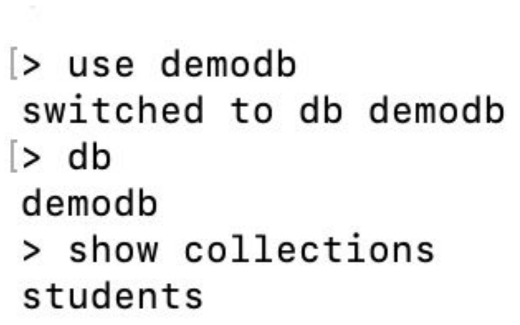
```
vishwajeet.mishra@MacBook-Pro learn-mongoose % node index.js
connected to Mongo DB
{
  name: 'Vishwa',
  age: 99,
  _id: new ObjectId("62257e8be13d03304bded2a8"),
  __v: 0
}
```

Create Schema

First, it will automatically create the DB named demoDB.

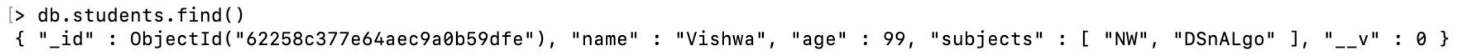


```
---
> show dbs
admin    0.000GB
config   0.000GB
demodb   0.000GB
```



```
> use demodb
switched to db demodb
> db
demodb
> show collections
students
```

It will also create a student collection.
See how data will look like in DB.



```
> db.students.find()
{ "_id" : ObjectId("62258c377e64aec9a0b59dfe"), "name" : "Vishwa", "age" : 99, "subjects" : [ "NW", "DSnALgo" ], "__v" : 0 }
```

Nested Schema

When you want to validate an array of objects/items or just object keys we need nested schema

Update the students.model.js
So, here we have created an addressSchema.
Which consist of:

- lane1:String
- lane2:String
- Street: String
- City: String
- Country: String
- Pincode: Number

Now we have to add this schema to our student schema to validate the address as well using these validations.
Also, with course as objectId, subjects, email, createdAt, UpdatedAt
Subjects will be an array of strings.

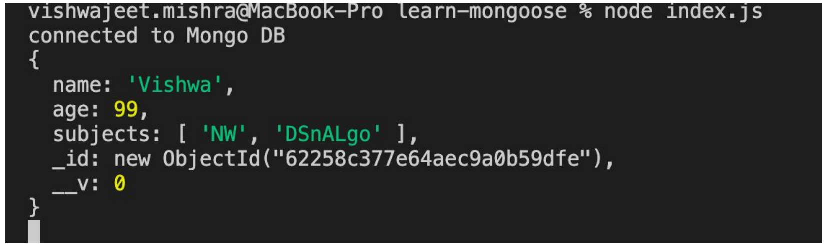
```
const mongoose = require("mongoose");


const addressSchema = new mongoose.Schema({
   lane1: String,
   lane2: String,
   street: String,
   city: String,
   country: String,
   pinCode: Number
})
const studentSchema = new mongoose.Schema({
   name: String,
   age: Number,
   email: String,
   createdAt: Date,
   updatedAt: Date,
   course: mongoose.SchemaTypes.ObjectId,
   subjects: [String],
   address: addressSchema
})



module.exports = mongoose.model("Student",
studentSchema);
```

Nested Schema and Validations

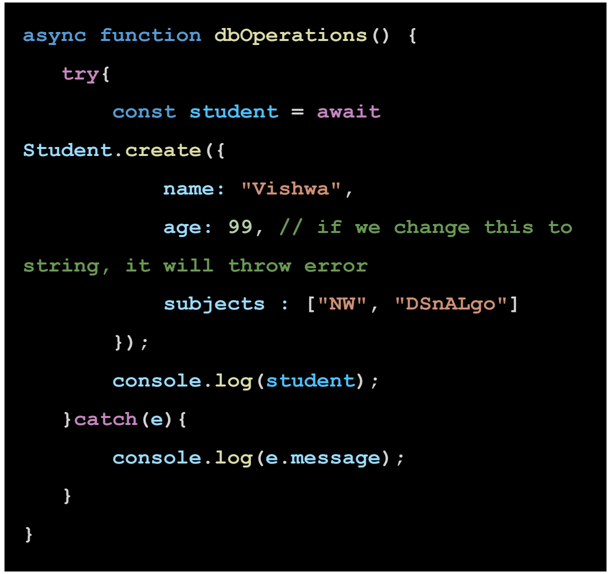
Modify Index.js

Run index.js

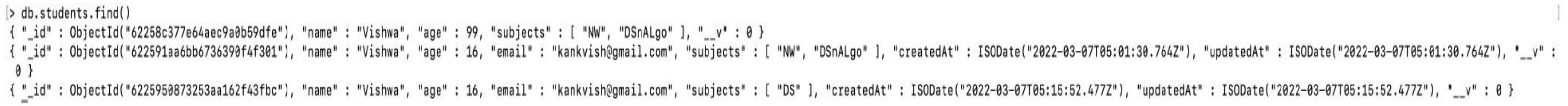
In the DB it is created as well.



```
vishwajeet.mishra@MacBook-Pro learn-mongoose % node index.js
connected to Mongo DB
{
  name: 'Vishwa',
  age: 99,
  subjects: [ 'NW', 'DSnALgo' ],
  _id: new ObjectId("62258c377e64aec9a0b59dfe"),
  __v: 0
}
```



```
async function dbOperations() {
   try{
        const student = await
Student.create({
              name: "Vishwa",
              age: 99, // if we change this to
string, it will throw error
              subjects : ["NW", "DSnALgo"]
        });
        console.log(student);
   }catch(e){
        console.log(e.message);
   }
}
```



```
> db.students.find()
{ "_id" : ObjectId("62258c377e64aec9a0b59dfe"), "name" : "Vishwa", "age" : 99, "subjects" : [ "NW", "DSnALgo" ], "__v" : 0 }
{ "_id" : ObjectId("622591aa6bb6736390f4f301"), "name" : "Vishwa", "age" : 16, "email" : "kankvish@gmail.com", "subjects" : [ "NW", "DSnALgo" ], "createdAt" : ISODate("2022-03-07T05:01:30.764Z"), "updatedAt" : ISODate("2022-03-07T05:01:30.764Z"), "__v" :
 0 }
{ "_id" : ObjectId("6225950873253aa162f43fbc"), "name" : "Vishwa", "age" : 16, "email" : "kankvish@gmail.com", "subjects" : [ "DS" ], "createdAt" : ISODate("2022-03-07T05:15:52.477Z"), "updatedAt" : ISODate("2022-03-07T05:15:52.477Z"), "__v" : 0 }
```

Add Inbuilt Validator

Data validation is important to make sure that “bad” data does not get persisted in your application.

Update students.model.js

Here we will use inbuilt validators to validate:

1. Age: Age should be greater than 16. Using min we can achieve this.
2. Email: Email has validations like.
   a. Required: It is a required field.
   b. Lowercase: It will convert all the characters to lowercase by default while storing them in the db
   c. Minimum length: We have set the minimum length to 15. If the email will be less than 15, it will through an error.

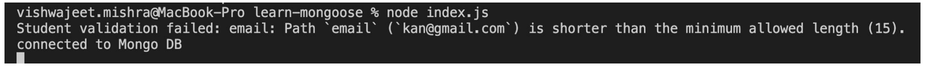
3. For Date and time, we will set a default value as well. Also, createdAt will never be changed again once created by using immutable: true.

```
const studentSchema = new mongoose.Schema({
   name: String,
   age: {
       type: Number,
       min: 16

   },
   email: {
       type: String,
       required: true,
       lowercase: true,
       minLength :15  // anything less than 10 will fail

   },
   createdAt: {
       type: Date,
       immutable: true,
       default: () => {
           return Date.now();
       }
   },
   updatedAt: {
       type: Date,
       default: () => {
           return Date.now();
       }
   },
   course: mongoose.SchemaTypes.ObjectId,
   subjects: [String],
   address: addressSchema
})
```

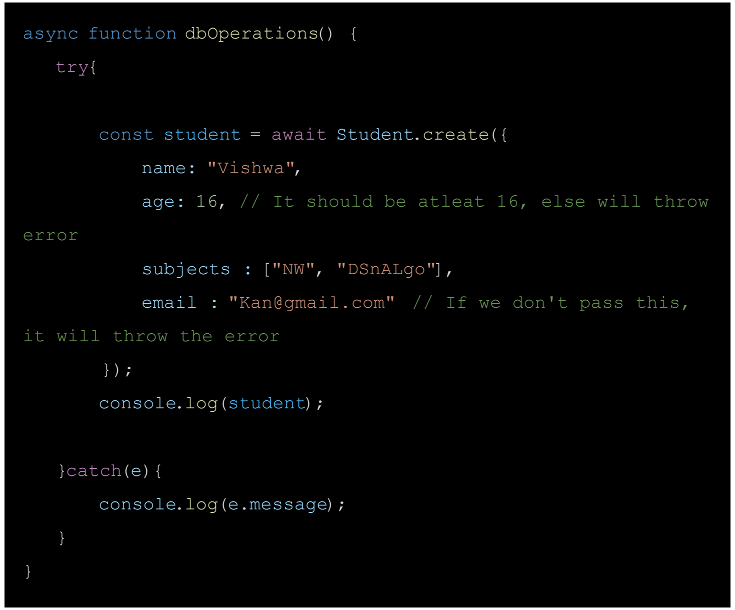
Add Inbuild Validator

Modify index.js

Now if the length of the email is less than 15 or the age is smaller than 16. Then it will through an error.

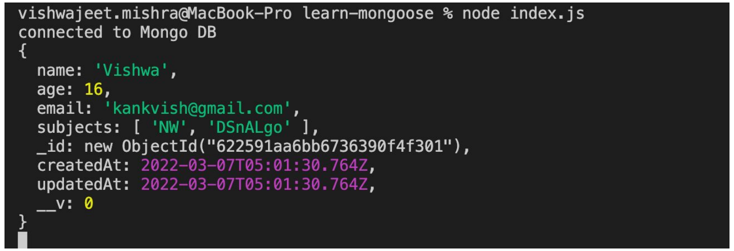


```
vishwajeet.mishra@MacBook-Pro learn-mongoose % node index.js
Student validation failed: email: Path `email` (`kan@gmail.com`) is shorter than the minimum allowed length (15).
connected to Mongo DB
```

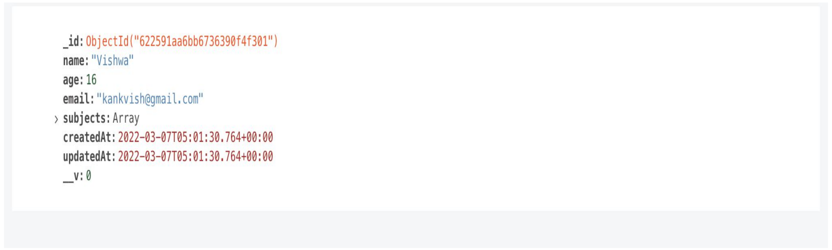


```
async function dbOperations() {
   try{

       const student = await Student.create({
           name: "Vishwa",
           age: 16, // It should be atleast 16, else will throw
error
           subjects : ["NW", "DSnALgo"],
           email : "Kan@gmail.com"  // If we don't pass this,
it will throw the error
       });
       console.log(student);

   }catch(e){
       console.log(e.message);
   }
}
```

Now change the email to something greater than 15.



```
vishwajeet.mishra@MacBook-Pro learn-mongoose % node index.js
connected to Mongo DB
{
  name: 'Vishwa',
  age: 16,
  email: 'kankvish@gmail.com',
  subjects: [ 'NW', 'DSnALgo' ],
  _id: new ObjectId("622591aa6bb6736390f4f301"),
  createdAt: 2022-03-07T05:01:30.764Z,
  updatedAt: 2022-03-07T05:01:30.764Z,
  __v: 0
}
```



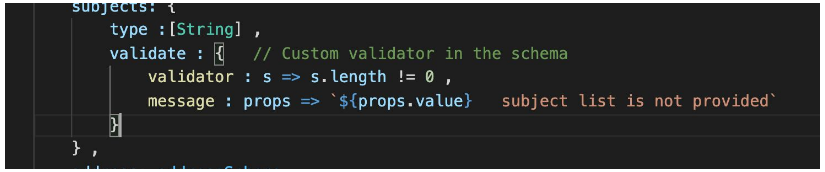
```
_id: ObjectId("622591aa6bb6736390f4f301")
name: "Vishwa"
age: 16
email: "kankvish@gmail.com"
› subjects: Array
createdAt: 2022-03-07T05:01:30.764+00:00
updatedAt: 2022-03-07T05:01:30.764+00:00
__v: 0
```

Custom Validator

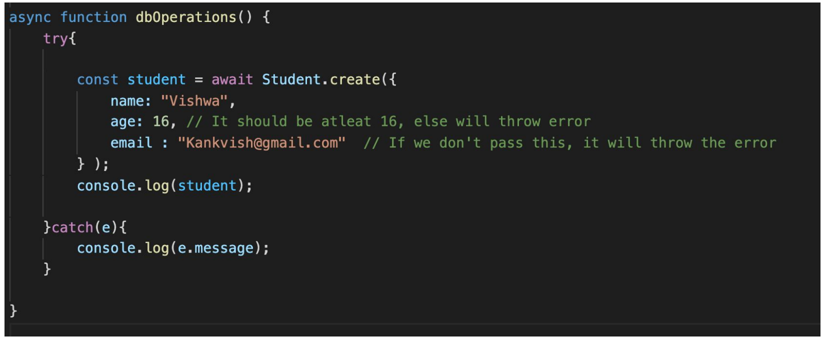
If the built-in validators aren't enough, you can define custom validators to suit your needs.

Custom validation is declared by passing a validation function.

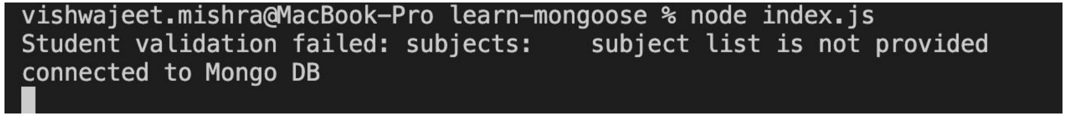
Modify students.models.js
By using validate we can make our custom validation.
Here we are implementing validation like if subjects are not present then it will through error.
Remove a subject from index.js
Run index.js



```
    subjects: {
        type :[String] ,
        validate : {   // Custom validator in the schema
            validator : s => s.length != 0 ,
            message : props => `${props.value}   subject list is not provided`
        }
    } ,
```



```
async function dbOperations() {
    try{

        const student = await Student.create({
            name: "Vishwa",
            age: 16, // It should be atleat 16, else will throw error
            email : "Kankvish@gmail.com"  // If we don't pass this, it will throw the error
        } );
        console.log(student);

    }catch(e){
        console.log(e.message);
    }

}
```



```
vishwajeet.mishra@MacBook-Pro learn-mongoose % node index.js
Student validation failed: subjects:    subject list is not provided
connected to Mongo DB
```

Queries on Database

Write queries that we will use in index.js
Add one by one all the queries and run it by replacing the id from your database.

```
//Finding by id
   try {
       const student = await Student.findById("620b26e29bea36c032f44151"); //Id of the
student we created previosuly
       console.log(student);
   } catch (err) {
       console.log(err.message)
   }
```

```
//Finding by other fields
   try{

       const student = await Student.find({name : "Mohan"}) // return the list of documents
having name as Mohan
       console.log(student);

   }catch(err){
       console.log(err.message)
   }
```

```
   //Finding just one

   try{

       const student = await Student.findOne({name : "Vishwa"}) // returns the very first
element
       console.log(student);

   }catch(err){
       console.log(err.message)
   }
```

Queries on Database

Write queries that we will use in index.js
Add one by one all the queries and run it by replacing the id from your database.

```
  //Delete one
   try{

       const student = await Student.deleteOne({name : "Vishwa"}) // delete the
very first matching entry
       console.log(student);

   }catch(err){
       console.log(err.message)
   }
```

```
   //Using where

   try {

       const student = await
Student.where("age").gt("10").where("name").equals('Vishwa').limit(1);
       console.log("Limit", student);
       //Setting the referece to other schema
       student[0].course = "620b21dd93cde00653a40c4d"

       savedStudent = await student[0].save();
       console.log("After saving ref", savedStudent);

   } catch (err) {
       console.log(err.message)
   }
```

MCQ

1. Which function is used to add a limit to your query

A. only()

B. limit()

C. till()

D. All of the above.

2. Which validator is used to set the maximum character limit

A. max

B. large

C. greater

D. All of the above

3. Which field is used to set error message in the custom validator

A. error

B. exception

C. Message

D. None of the above.

MCQ

4. How to set the custom unique id to a field.

A. mongoose.SchemaTypes.ObjectId

B. mongoose.ObjectId

C. SchemaTypes.ObjectId

D. None of the above

5. How to query for greater than with mongoose?

A. greaterThan()

B. grThan()

C. gr()

D. None of the above

Practice Problems

1. Create a schema for the ticket table which will be useful in our CRM application with custom validations.
Fields required:
title: String and required
ticketPriority: Number, required and have default value.
description: String and required
status :String, required and have default value like “OPEN”.
reporter:String
assignee :String
createdAt: Date and not able to change it. Also, have some default current time.
updatedAt: Date. Also, have some default current time.

Practice Problems

2. Create schema for User table which will be useful in our CRM application with custom validations.
Fields required:
name: String and required
userId: String, unique and required
password: String and required
email: String, lowercase, unique, with minimum length of 10 and required
createdAt: Date and not able to change it. Also, have some default current time.
updatedAt: Date. Also, have some default current time.
userType: String, required and have default value like "CUSTOMER"
userStatus: String, required and have default value like "APPROVED"
ticketsCreated : ObjectId and refer Ticket table.
ticketsAssigned : ObjectId and refer Ticket table.

Thank You!