Mongoose

কিঃ
Mongoose an elegant MongoDB object modeling for Node.js

কেনে ব্যাবহার করবো
- Provides schema-based data modeling solution
- Type casting
- Validation
- Query building
- Business logic.

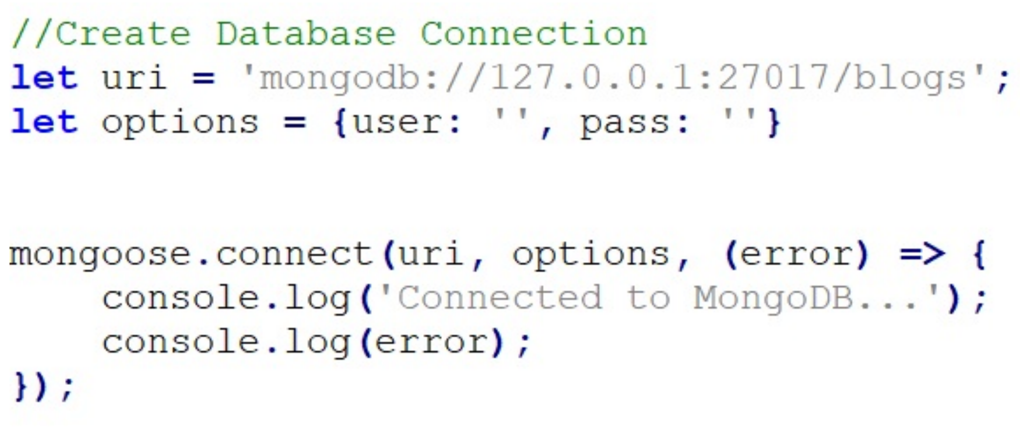
Mongoose

How to Start

1. Install mongoose
2. Create Database connection
3. Create your first mongoose model
4. Work with mongoose model

Mongoose

Create Database Connection

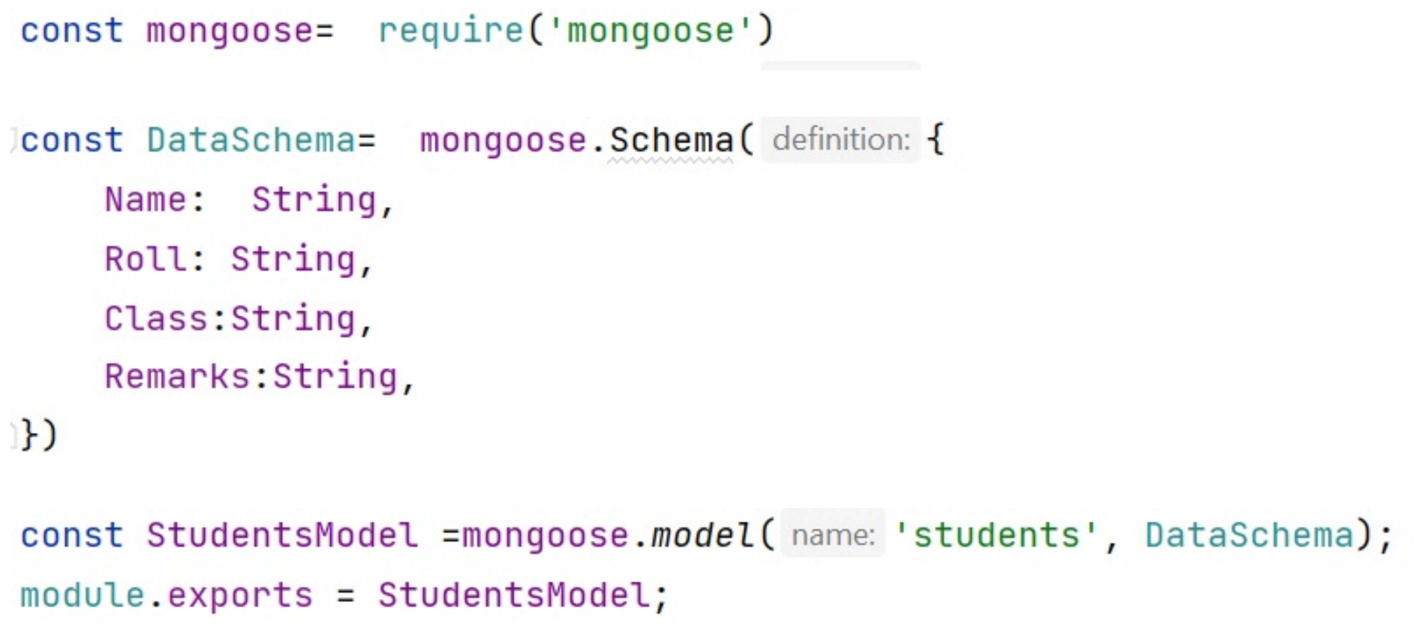


```
//Create Database Connection
let uri = 'mongodb://127.0.0.1:27017/blogs';
let options = {user: '', pass: ''}


mongoose.connect(uri, options, (error) => {
    console.log('Connected to MongoDB...');
    console.log(error);
});
```

Mongoose

Express JS
REST API DEVELOPMENT

Create your first mongoose model

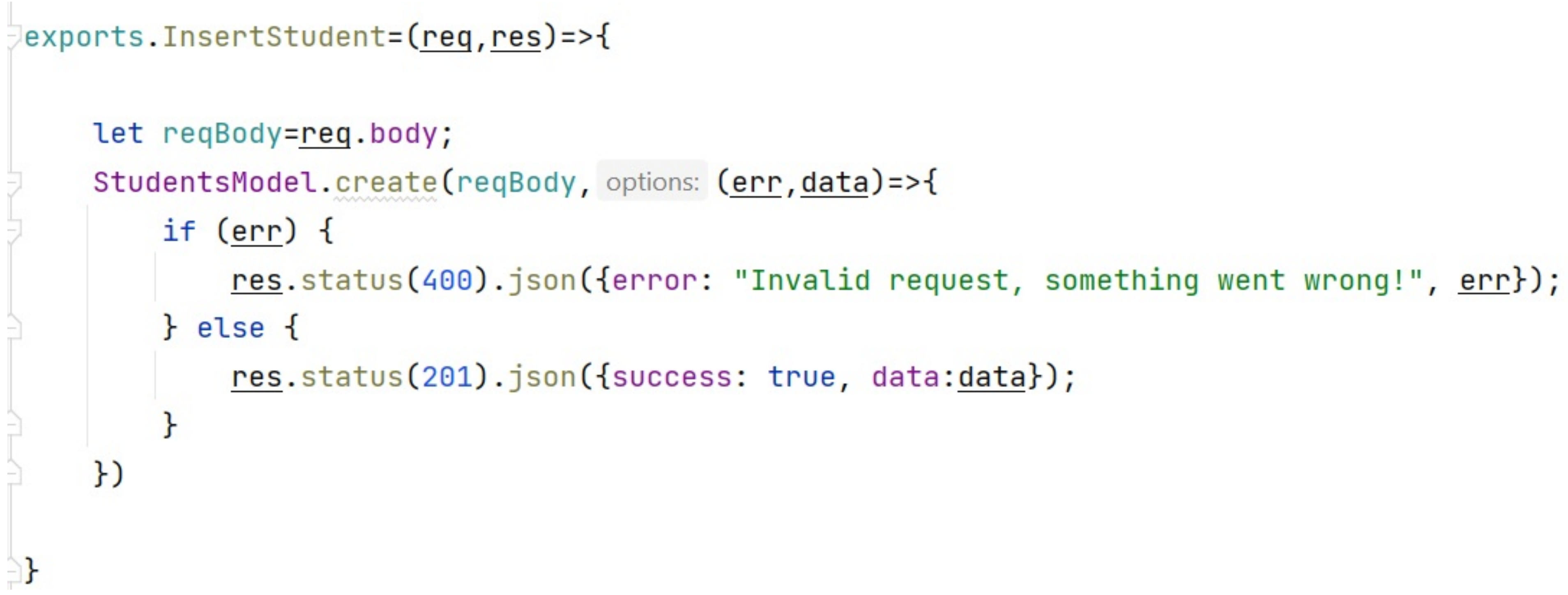


```
const mongoose=  require('mongoose')

const DataSchema=  mongoose.Schema( definition: {
    Name:  String,
    Roll: String,
    Class:String,
    Remarks:String,
})

const StudentsModel =mongoose.model( name: 'students', DataSchema);
module.exports = StudentsModel;
```

Mongoose

REST API DEVELOPMENT

Work with mongoose model (Insert)

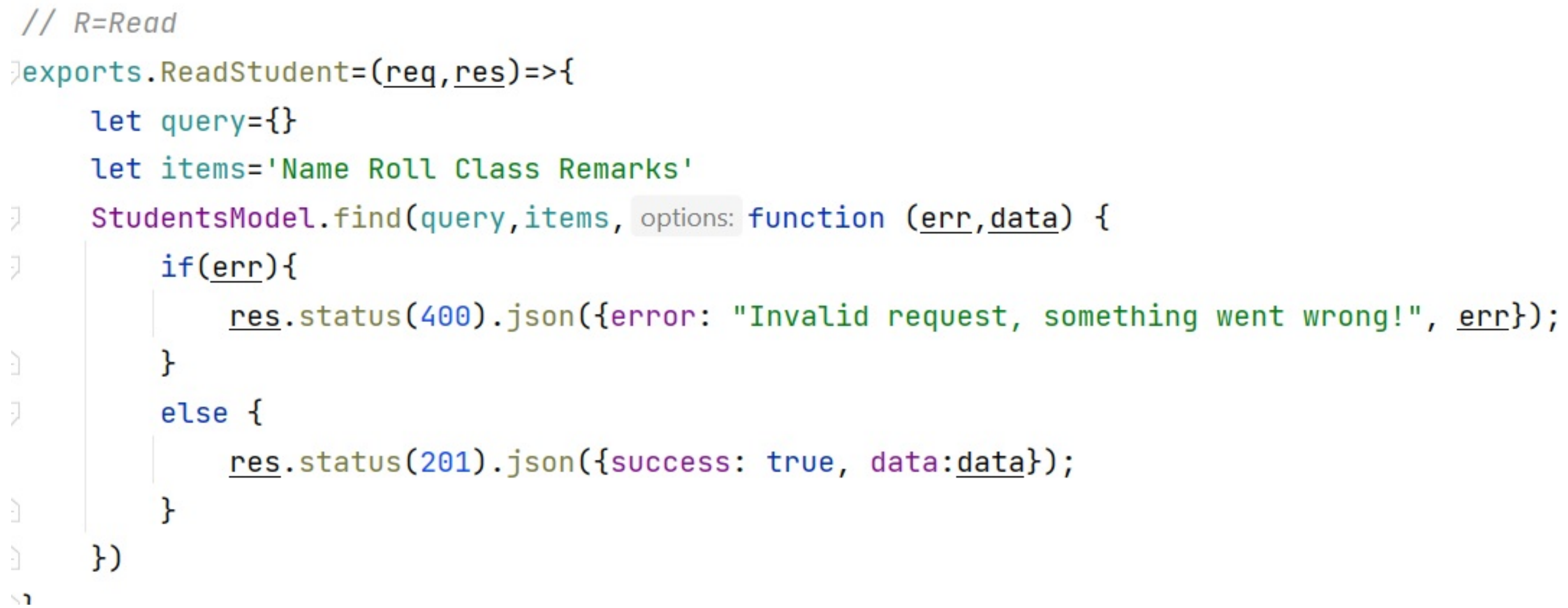


```
exports.InsertStudent=(req,res)=>{

    let reqBody=req.body;
    StudentsModel.create(reqBody, options: (err,data)=>{
        if (err) {
            res.status(400).json({error: "Invalid request, something went wrong!", err});
        } else {
            res.status(201).json({success: true, data:data});
        }
    })

}
```

Mongoose

Express JS
REST API
DEVELOPMENT

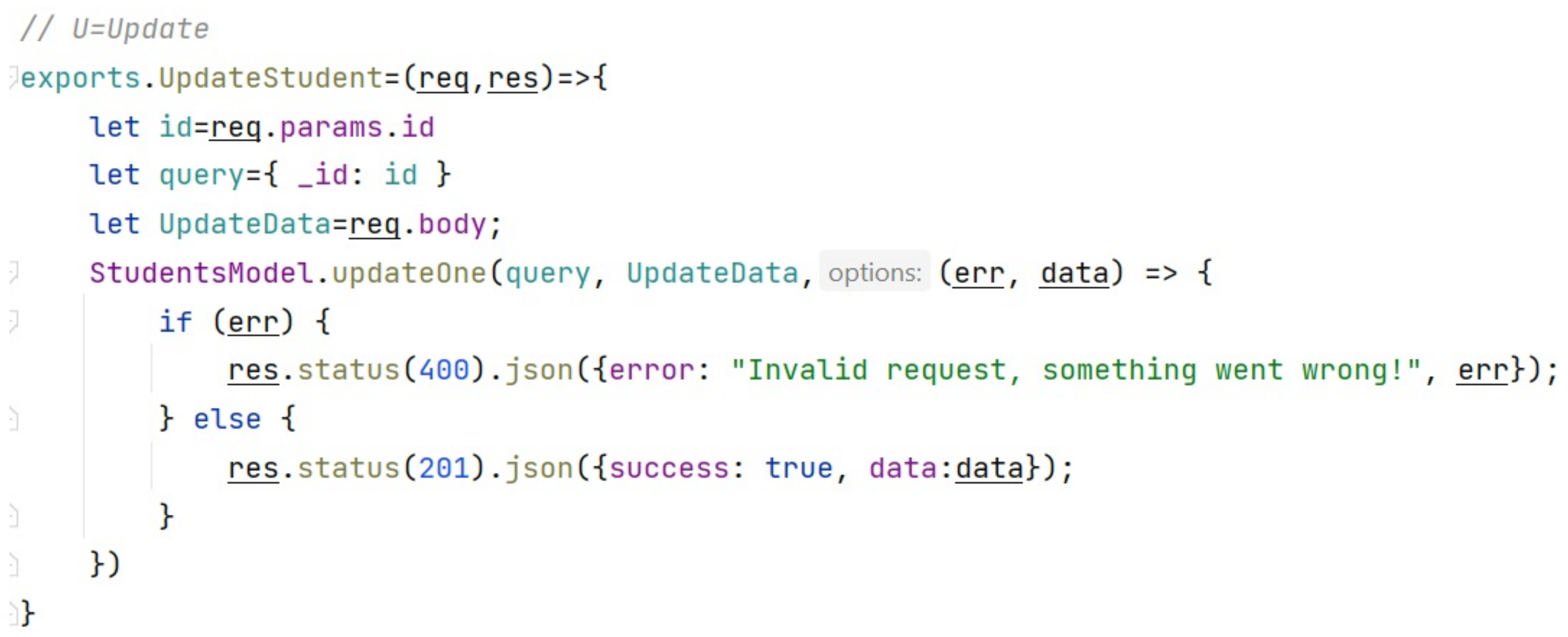
Work with mongoose model (Read)



```
// R=Read
exports.ReadStudent=(req,res)=>{
    let query={}
    let items='Name Roll Class Remarks'
    StudentsModel.find(query,items, options: function (err,data) {
        if(err){
            res.status(400).json({error: "Invalid request, something went wrong!", err});
        }
        else {
            res.status(201).json({success: true, data:data});
        }
    })
```

Mongoose

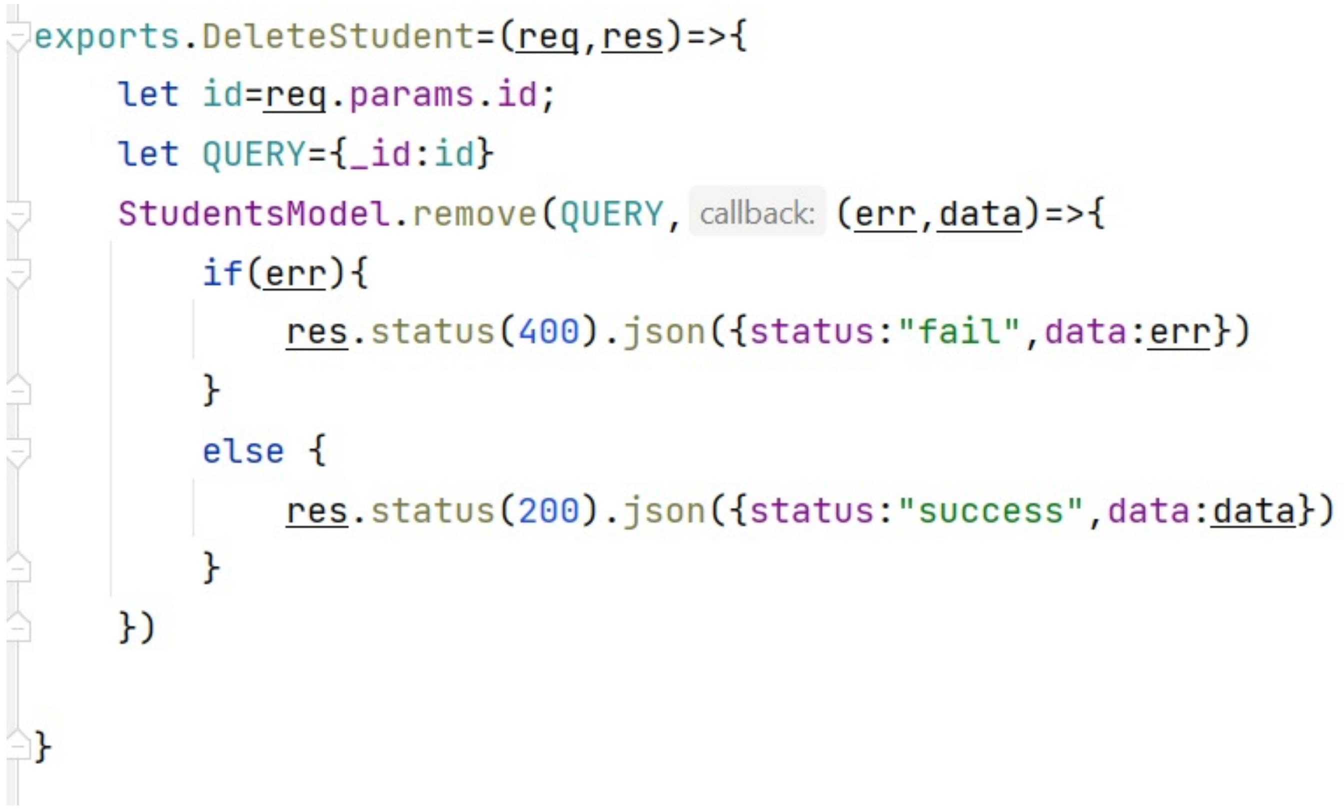
Work with mongoose model (Update)



```
// U=Update
exports.UpdateStudent=(req,res)=>{
    let id=req.params.id
    let query={ _id: id }
    let UpdateData=req.body;
    StudentsModel.updateOne(query, UpdateData, options: (err, data) => {
        if (err) {
            res.status(400).json({error: "Invalid request, something went wrong!", err});
        } else {
            res.status(201).json({success: true, data:data});
        }
    })
}
```

Mongoose

Work with mongoose model (Delete)

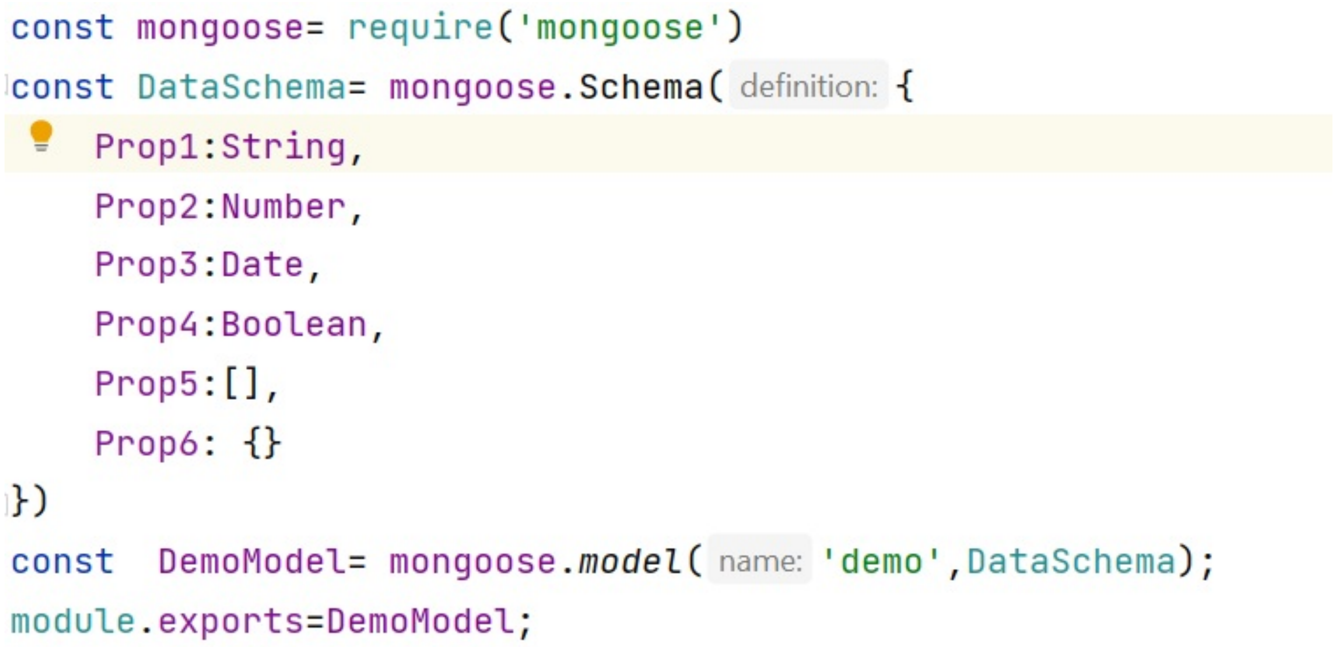


```
exports.DeleteStudent=(req,res)=>{
    let id=req.params.id;
    let QUERY={_id:id}
    StudentsModel.remove(QUERY, callback: (err,data)=>{
        if(err){
            res.status(400).json({status:"fail",data:err})
        }
        else {
            res.status(200).json({status:"success",data:data})
        }
    })

}
```

Mongoose

SchemaTypes:

1. String
2. Number
3. Date
4. Buffer
5. Boolean
6. Mixed
7. ObjectId
8. Array
9. Decimal128
10. Map



```
const mongoose= require('mongoose')
const DataSchema= mongoose.Schema( definition: {
    Prop1:String,
    Prop2:Number,
    Prop3:Date,
    Prop4:Boolean,
    Prop5:[],
    Prop6: {}
})
const  DemoModel= mongoose.model( name: 'demo',DataSchema);
module.exports=DemoModel;
```

Mongoose

Default Value & Version Key

```
date: { type: Date, default: Date.now },
```

```
{versionKey:false}
```

Mongoose

Type Casting Validation

```
Class:String,
```

Required Validation

```
Name:{type:String,required: true},
```

Unique Validation

```
Name:{type:String,unique: true},
```

Min-Max Number Validation

```
Roll: {type: Number, min: 6, max: 12},
```

Mongoose

Custom Error Message Validation

```
Roll: {
    type: Number,
    min: [6, 'Minimum Roll 6 & Maximum Roll 12, But got {VALUE}'],
    max: [12, 'Minimum Roll 6 & Maximum Roll 12, But got {VALUE}']
},
```

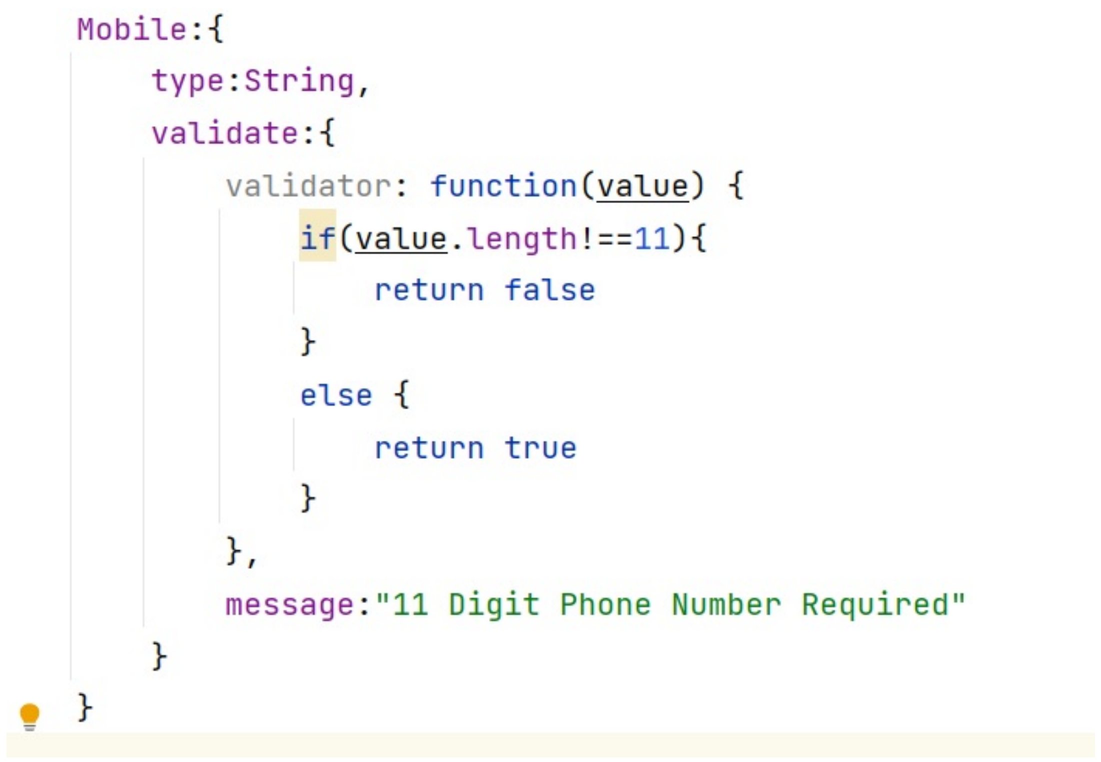
Enumerated type Validation

```
Class: {type: String,enum: { values: ['One','Two','Three','Four','Five'], message: '{VALUE} is not supported' }}
```

Mongoose

Express JS
REST API DEVELOPMENT

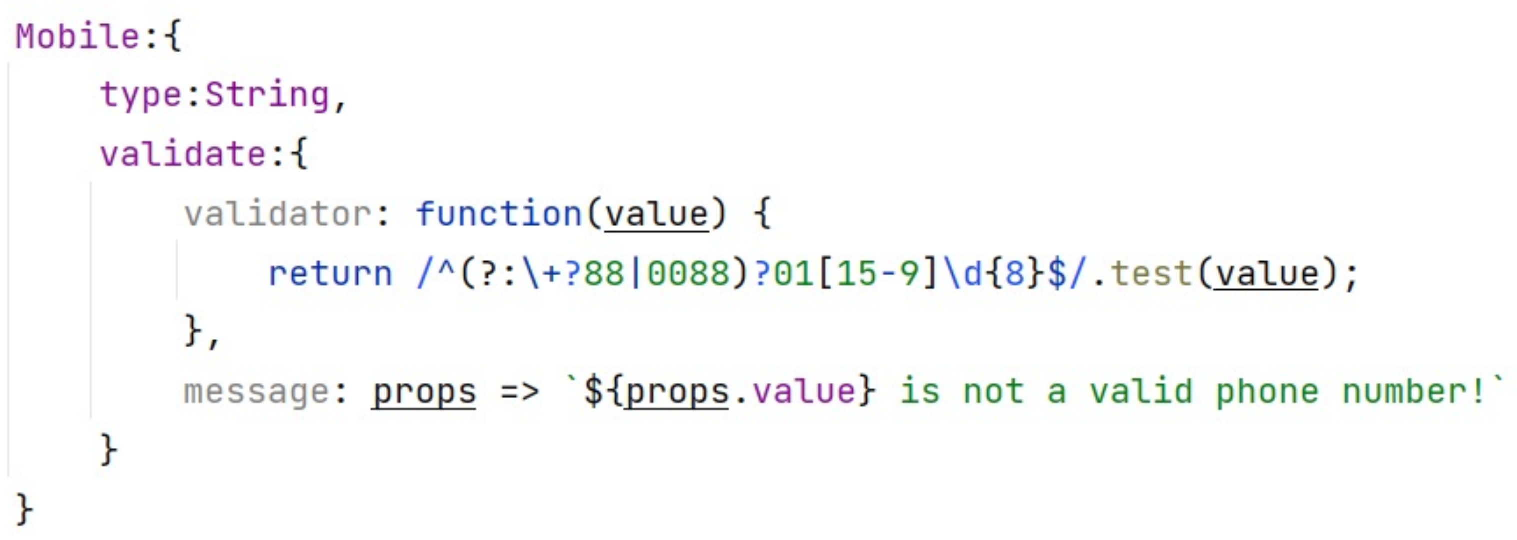
Custom Validation



```
Mobile:{
    type:String,
    validate:{
        validator: function(value) {
            if(value.length!==11){
                return false
            }
            else {
                return true
            }
        },
        message:"11 Digit Phone Number Required"
    }
}
```

Mongoose

Regex Validation



```
Mobile:{
    type:String,
    validate:{
        validator: function(value) {
            return /^(?:\+?88|0088)?01[15-9]\d{8}$/.test(value);
        },
        message: props => `${props.value} is not a valid phone number!`
    }
}
```

AUTH

Types Of Authentication – Authorization

- API Key
- Bearer Token
- Basic Auth
- Digest Auth
- Auth 1.0
- Auth 2. 0
- Hawk Authentication
- AWS Signature
- NTLM Authentication

AUTH

What About JSON Web Tokens (JWT)

AUTH

What We Will Learn

- How to encode json and create token
- How to decode token and retrieve json
- How to create auth middleware in express project
- How to apply auth middleware in route

PRACTICE PROJECT

Project Covers:

- Rest API Introduction
- JSON Best Practices
- Request response model
- Rest API Basic Best Practices
- Rest API Security Practices
- Express JS REST API Fundamental
- MongoDB Core Fundamental
- Mongoose
- Documentation
- JWT Auth

PRACTICE PROJECT

Project Features

- User Registration
- User Login
- User Auth By JWT Token
- User Profile Read
- User Profile Update
- User To–do List Create
- User To–do List Read
- User To–do List Update
- **User To–do List Delete**
- **User To–do List Complete/Cancel Mark**
- **User To–do List Complete/Cancel Filter**
- **User To–do List Date Filter**