Validating Data with Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB, and it provides built-in mechanisms for data validation. Validation ensures that only data that meets specific criteria is saved into the database. This feature is particularly useful for maintaining data integrity.

1. What is Data Validation in Mongoose?

In Mongoose, data validation is a feature that allows you to define rules directly in the schema. When you try to save or update a document, Mongoose checks the data against these rules and throws an error if the data does not meet the requirements.

How to Set Up Validation

When defining a Mongoose schema, you can include validation rules as part of the schema definition. Validation can be applied to any field.

Example Schema with Validation

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

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Name is required'], // Validation: must be present
    minlength: [3, 'Name must be at least 3 characters long'], // Validation: min length
    maxlength: [50, 'Name cannot exceed 50 characters'], // Validation: max length
  },
  email: {
    type: String,
    required: [true, 'Email is required'], // Validation: must be present
    match: [/^\S+@\S+\.\S+$/, 'Invalid email format'], // Validation: pattern
  },
  age: {
    type: Number,
    min: [18, 'Age must be at least 18'], // Validation: min value
    max: [65, 'Age must be below 65'], // Validation: max value
  },
  createdAt: {
    type: Date,
    default: Date.now, // Default value
  },
});

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

Using Middleware for Data Validation in Express.js

Middleware in Express.js can be used to validate incoming data in the request body, query parameters, or URL parameters. This is an efficient way to ensure that data is correct before processing it further in your route handler.

Below, I'll explain how to validate data using **middleware** in Express.js.

```
const validateStudent = (req, res, next) => {
   const { name, age, major, gpa, email } = req.body;

   // Validation logic
   if (!name || name.length < 3) {
     return res.status(400).json({ error: "Name is required and must be at least 3 characters
long" });
   }

   if (!age || age < 16 || age > 100) {
     return res.status(400).json({ error: "Age is required and must be between 16 and 100" });
   }

   const validMajors = ["Science", "Arts", "Commerce", "Engineering", "React", "Angular"];
   if (!major || !validMajors.includes(major)) {
     return res.status(400).json({ error: `Major is required and must be one of
${validMajors.join(", ")}` });
   }

   if (gpa < 0 || gpa > 4) {
     return res.status(400).json({ error: "GPA must be between 0 and 4" });
   }

   if(!validator.isEmail(email)){
       return res.json({error:'Please enter valid mail address'})
}
next()
  };
```

Interview Questions and answers:

What is validation in Mongoose?

A: Validation in Mongoose ensures that the data being saved to the database meets specific rules defined in the schema. It is a way to enforce data integrity by checking the fields for specific types, lengths, ranges, or formats before saving the document.

What are some built-in validators in Mongoose?

A: Mongoose provides the following built-in validators:

- required: Ensures a field is mandatory.
- min and max: Set a range for numbers.
- minLength and maxLength: Set a range for string lengths.
- enum: Ensures the value is one of the predefined options.
- match: Validates a string using a regular expression.

How can you add a custom validation rule in Mongoose?

A: You can define custom validation logic using the validate property in the schema.

```
const studentDataScheme=new mongoose.Schema({
   name:{type:String,
        required:[true,"Name is required"],
       //  lowercase:true,
       // uppercase:true,
          trim:true,
          minLength:[3,"Name must be at least 3 letter"]
       },})
```

When does Mongoose run validations?

A: Mongoose runs validations:

1. **Before saving**: On document.save() or Model.create().
2. **On updates**: If runValidators: true is explicitly set in update queries like updateOne() or findOneAndUpdate().

What is middleware in Express.js?

A: Middleware in Express.js is a function that processes incoming requests and can perform tasks like authentication, logging, validation, and error handling. Middleware functions are executed in the order they are defined in the application.

How can you use middleware for request validation in Express.js?

A: Middleware can validate request data by checking the req.body, req.params, or req.query. If the data is invalid, the middleware can terminate the request and send an error response.

```
const validateStudent = (req, res, next) => {
   const { name, age } = req.body;
   if (!name || name.length < 3) {
     return res.status(400).json({ error: "Name is required and must be at least 3 characters
long" });
   }
   if (!age || age < 16) {
     return res.status(400).json({ error: "Age must be at least 16" });
   }
   next();
 };

 app.post('/students', validateStudent, (req, res) => {
   res.send('Student data is valid');
 });
```

What are the advantages of using middleware for validation?

A:

- Centralizes validation logic, making the code cleaner.
- Reusable across multiple routes.
- Ensures consistent error handling and responses.

How do you implement custom validation logic in Express.js?

A: Custom validation logic can be written in middleware or utility functions. For example:

- Checking if a username is alphanumeric.
- Verifying the presence of required fields.
- Validating nested objects or complex data structures.

```
const validateCustom = (req, res, next) => {
   const { gpa } = req.body;
   if (gpa < 0 || gpa > 4) {
     return res.status(400).json({ error: "GPA must be between 0 and 4" });
   }
   next();
 };
```

How does custom validation in Mongoose differ from middleware-based validation?

A:

- **Mongoose Validation**: Enforces rules directly at the schema/database level. It is tied to the model and runs automatically before saving or updating data.
- **Middleware Validation**: Happens at the API level in Express.js, allowing pre-processing and client feedback before interacting with the database.

How can you handle validation errors in Mongoose?

A: Use a try-catch block or error-handling middleware to catch and handle validation errors.

Example:

```
const user = new User({ name: '' });
user.save().catch(err => {
  if (err.name === 'ValidationError') {
    console.log(err.message); // Log validation error
  }
});
```

Why is data validation important?

A: Data validation ensures:

- Data integrity and security.
- Proper functionality of the application by preventing invalid or malicious input.
- Compliance with database constraints and business rules.

How do you choose between client-side and server-side validation?

A:

- **Client-side validation**: Provides immediate feedback to users but can be bypassed.
- **Server-side validation**: Ensures robust security by enforcing rules on the server.

Both should be used together for an optimal user experience and security.