GUIDE TO HOOKS IN MONGOOSE

Imperativeness of Mongoose Middleware aka Pre/Post-hooks in API's Authentication

One of the hardest initial hurdles using Mongoose was understanding the library's middleware or "hook" system that tie particular functions to particular lifecycle and query events or into the database transaction lifecycle to perform operations.

punitkmryh
Jun 27 · 3 min read ★

. . .

Middleware is a way to customize the behaviour of your Mongoose model and it's going to allow us to do some pretty interesting things to explore this.

> ***Example:1*** *I could have a user model that has some middleware that makes sure that a username does not violate certain rules before continuing, rules like the username must begin with a letter, only contain permitted characters, and not exceed a set character length.*
>
> ***Example: 2*** *When I want to encrypt user's password hashing in an API where I can build a save hook just before saving user by hashing plain text*

Note: In order to take advantage of middleware we need to make sure that we have separate schema and separate model during implementation.

> ***Does the middleware of mongoose also refer to the middleware of express?***
>
> ***Short answer:** they're separate.*
>
> ***Longer answer:** By convention, most middleware stacks implement some kind of `next` function to call in order to proceed down the stack and call each middleware function in turn.*
>
> *It's a matter of scope. Express and Mongoose both have their own independent middleware stacks, so what the `next` function does depends on where it gets called. As a general rule of thumb, every function-- including the anonymous functions used for callbacks that accept a `next` parameter-- have their own scope.*

Mongoose Middleware :

Middleware (also called pre and post hooks) are functions which are passed control during the execution of asynchronous functions. Middleware is specified on the schema level and is useful for writing plugins. Mongoose 4.0 has 2 types of middleware:

1. **Document middleware and**
2. **Query middleware.**

Document middleware is supported for the following document functions.

- validate
- save
- remove
- updateOne
- deleteOne
- init (note: init hooks are synchronous)

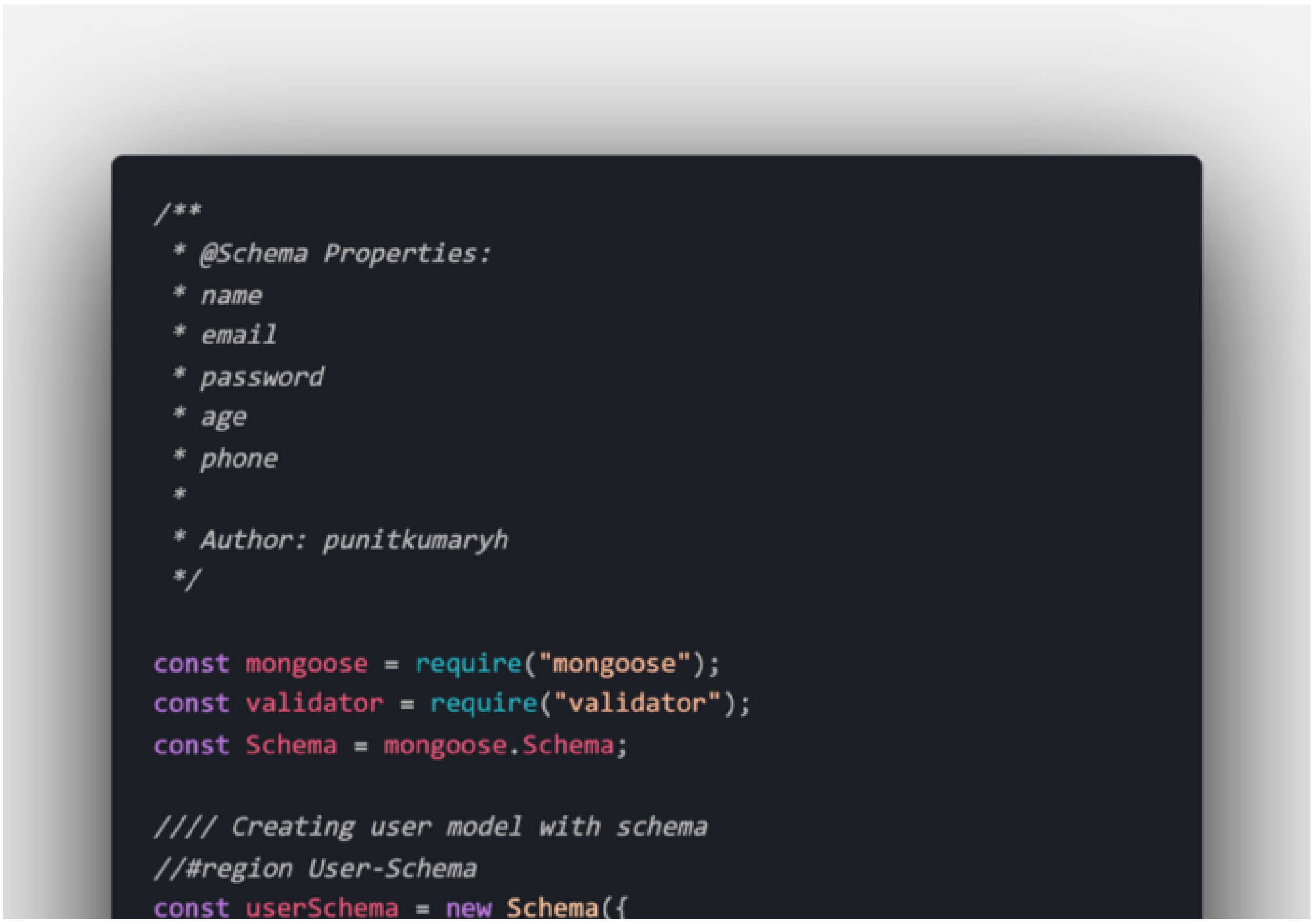
Query middleware is supported for the following Model and Query functions.

- count
- deleteMany

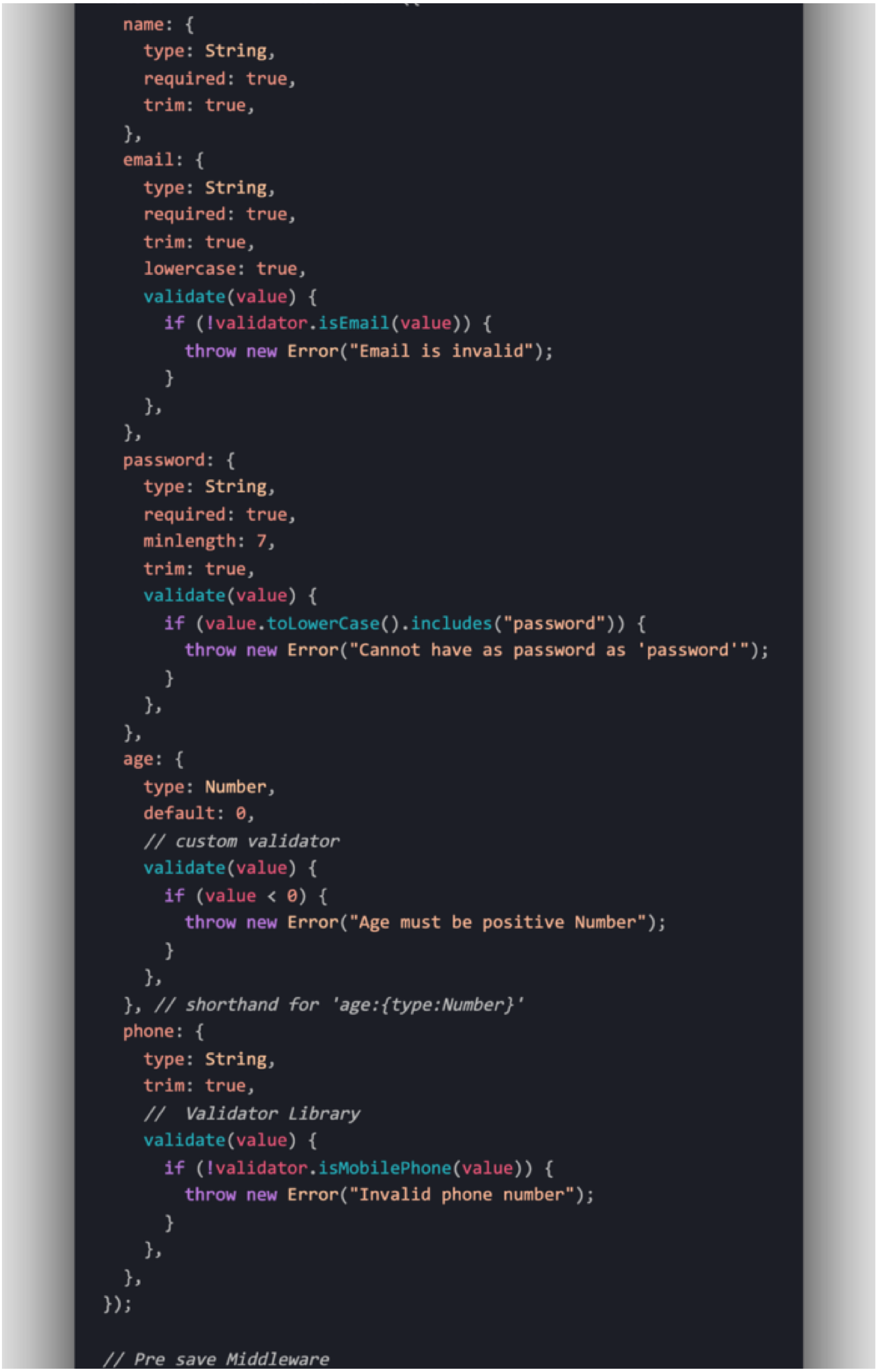
- deleteOne
- find
- findOne
- findOneAndDelete
- findOneAndRemove
- findOneAndUpdate
- remove
- update
- updateOne
- updateMany

Both document middleware and query middleware support pre and post hooks.

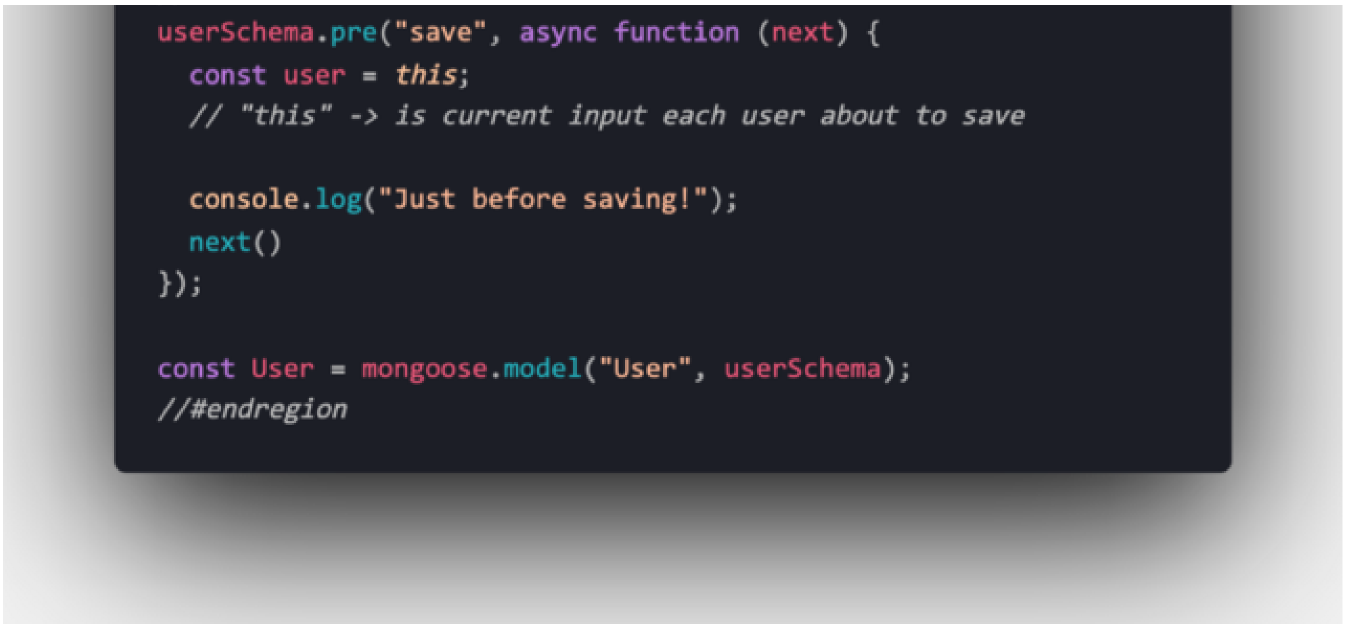
Below code shows the pre-save middleware defined for User model and will log with just before saving:



```
/**
 * @Schema Properties:
 * name
 * email
 * password
 * age
 * phone
 *
 * Author: punitkumaryh
 */

const mongoose = require("mongoose");
const validator = require("validator");
const Schema = mongoose.Schema;

//// Creating user model with schema
//#region User-Schema
const userSchema = new Schema({
```



```
  name: {
    type: String,
    required: true,
    trim: true,
  },
  email: {
    type: String,
    required: true,
    trim: true,
    lowercase: true,
    validate(value) {
      if (!validator.isEmail(value)) {
        throw new Error("Email is invalid");
      }
    },
  },
  password: {
    type: String,
    required: true,
    minlength: 7,
    trim: true,
    validate(value) {
      if (value.toLowerCase().includes("password")) {
        throw new Error("Cannot have as password as 'password'");
      }
    },
  },
  age: {
    type: Number,
    default: 0,
    // custom validator
    validate(value) {
      if (value < 0) {
        throw new Error("Age must be positive Number");
      }
    },
  }, // shorthand for 'age:{type:Number}'
  phone: {
    type: String,
    trim: true,
    //  Validator Library
    validate(value) {
      if (!validator.isMobilePhone(value)) {
        throw new Error("Invalid phone number");
      }
    },
  },
});

// Pre save Middleware
```



```
userSchema.pre("save", async function (next) {
  const user = this;
  // "this" -> is current input each user about to save

  console.log("Just before saving!");
  next()
});

const User = mongoose.model("User", userSchema);
//#endregion
```

pre-save-Hooks

Pre-hooks

Pre middleware functions are executed one after another when each middleware calls `next`. There are two types of pre hooks:

1. **Serial** and
2. **Parallel**.

Let's take one at a time and understand:

1. **Serial :**

Serial middleware is executed one after another when each middleware calls next.

Serial-Pre-hook

> *Remember we cannot use arrow function inside pre hooks as arrow function are poor binding functions.*

next argument: The whole point of this is to run some code before (a user is) saved. But how does it know when we're done running our code. Now it could just say when the function is over. But that wouldn't account for any asynchronous process which might be occurring. So that's why next is provided. We simply call next when we're done.