

Hoang Hung

Follow

Published Mar 29th, 2:44 PM

Mongoose cho MongoDB, Nodejs

Mar 29th, 2:44 PM

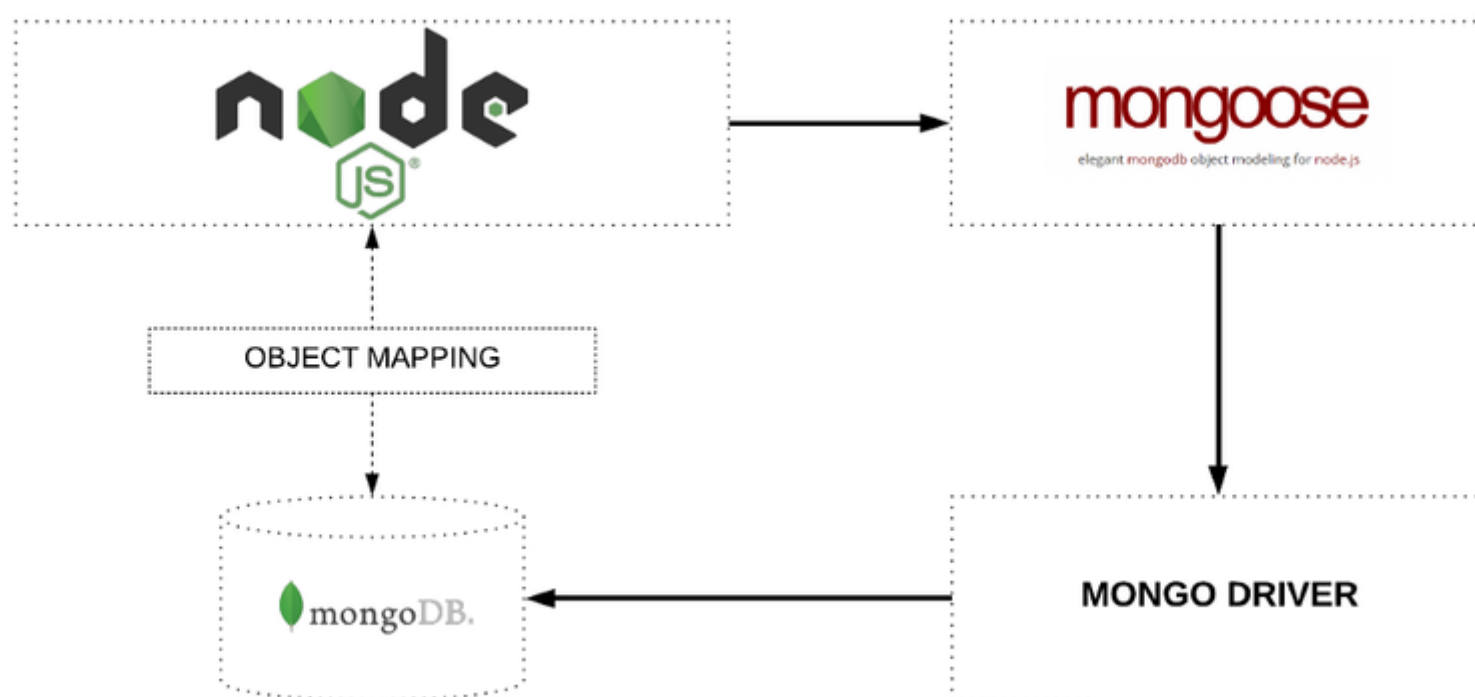
719

0

0

Report

Mongoose là một thư viện mô hình hóa đối tượng (Object Data Model - ODM) cho MongoDB và Node.js. Nó quản lý mối quan hệ giữa dữ liệu, cung cấp sự xác nhận giản đơn và được sử dụng để dịch giữa các đối tượng trong mã và biểu diễn các đối tượng trong MongoDB.



thay đổi vì nó không đạt được như các cơ sở dữ liệu SQL. Đây là một trong những lợi thế của việc sử dụng NoSQL vì nó tăng tốc độ phát triển ứng dụng và giảm sự phức tạp của việc triển khai.

Dưới đây là ví dụ về cách dữ liệu được lưu giữ trong cơ sở dữ liệu Mongo vs. SQL:

MONGO

PEOPLE

```
{
  "Id": 1,
  "FirstName": "Ada",
  "LastName": "Lovelace",
  "Email": "ada.lovelace@gmail.com",
  "Phone": [{
    "Home": "+1.123.456.7890"
  },
  {
    "Work": "+1.111.222.3333"
  }
]
}

{
  "Id": 2,
  "FirstName": "Grace",
  "LastName": "Hopper",
  "Email": "grace.hopper@gmail.com"
}

{
  "Id": 3,
  "FirstName": "Kathy",
  "LastName": "Sierra",
  "Email": "kathy.sierra@gmail.com"
}
```

SQL

PERSON

Id	FirstName	LastName	Email
1	Ada	Lovelace	ada.lovelace@gmail.com
2	Grace	Hopper	grace.hopper@gmail.com
3	Kathy	Sierra	kathy.sierra@gmail.com

PHONE_NUMBER

PersonId	PhoneId	Phone Number	Type
1	1	+1.123.456.7890	Home
1	2	+1.111.222.3333	Work

Thuật ngữ - Terminologies

Collections

Collections ở Mongo tương đương với các bảng trong các cơ sở dữ liệu quan hệ. Chúng có thể chứa nhiều tài liệu JSON.

Documents

'Documents' tương đương với các bản ghi trong SQL. Mặc dù một dòng SQL có thể tham khảo dữ liệu trong các bảng khác, các tài liệu Mongo thường kết hợp trong một tài liệu.

Fields

'Fields' thuộc tính tương tự như các cột trong một bảng SQL.

Schema

Trong khi Mongo là schema-less, SQL định nghĩa một lược đồ thông qua định nghĩa bảng. Một lược đồ Mongoose là một lớp cấu trúc dữ liệu được thi hành qua lớp ứng dụng.

Fields

'Models' là mô hình bậc cao có một lược đồ và tạo ra một thể hiện của một tài liệu tương đương với các bản ghi trong một cơ sở dữ liệu quan hệ.

Cài đặt Mongo

Schema Trong khi Mongo là schema-less, SQL định nghĩa một lược đồ thông qua định nghĩa bảng. Một lược đồ Mongoose 'là một cấu trúc dữ liệu tài liệu (hoặc hình dạng của tài liệu) được thi hành qua lớp ứng dụng.

Cài đặt MongoDB thích hợp cho Hệ điều hành của bạn từ Website MongoDB

<https://docs.mongodb.com/manual/installation/> và làm theo hướng dẫn cài đặt

- Tạo đăng ký cơ sở dữ liệu sandbox miễn phí trên mLab
- Cài đặt Mongo bằng Docker nếu bạn sử dụng docker Hãy điều hướng qua một số vấn đề cơ bản của Mongoose bằng cách triển khai mô hình dữ liệu cho một số địa chỉ đơn giản.

Cài đặt NPM Chúng ta hãy vào thư mục dự án và khởi tạo dự án của chúng ta

```
npm init -y
```

Cài đặt Mongoose và một thư viện xác nhận với lệnh sau:

```
npm install mongoose validator
```

Lệnh cài đặt trên sẽ cài đặt phiên bản mới nhất của thư viện. Cú pháp Mongoose trong bài báo này đặc trưng cho Mongoose v5 và hơn thế nữa.

Kết nối cơ sở dữ liệu Tạo tệp `./src/database.js` trong thư mục gốc của dự án.

Tiếp theo, chúng ta sẽ tạo thêm một phương thức kết nối với cơ sở dữ liệu.

Kết nối sẽ thay đổi tùy theo cài đặt của bạn.

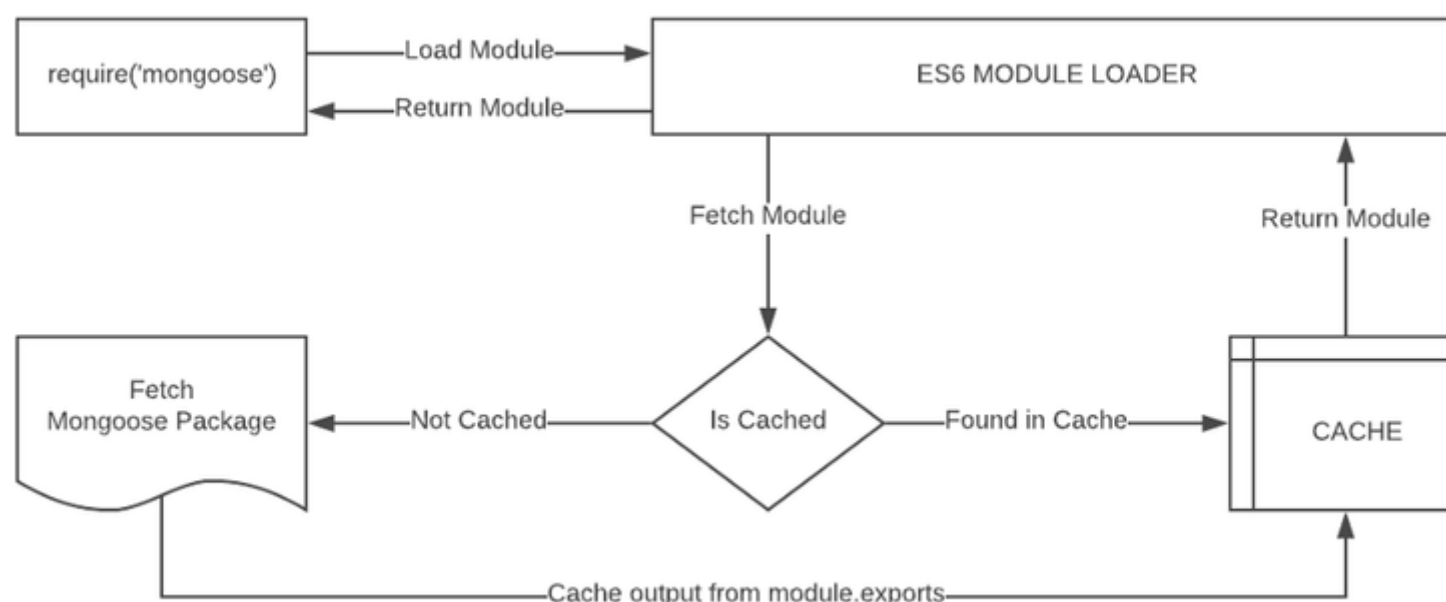
```
let mongoose = require('mongoose');

const server = '127.0.0.1:27017'; // REPLACE WITH YOUR DB SERVER
const database = 'fcc-Mail'; // REPLACE WITH YOUR DB NAME

class Database {
  constructor() {
    this._connect()
  }
  _connect() {
    mongoose.connect(`mongodb://${server}/${database}`)
      .then(() => {
        console.log('Database connection successful')
      })
      .catch(err => {
        console.error('Database connection error')
      })
  }
}

module.exports = new Database()
```

Lệnh yêu cầu ('mongoose') trả về phía trên trả về một đối tượng Singleton. Điều này có nghĩa là lần đầu tiên bạn gọi ("mongoose"), nó tạo ra một lớp Mongoose và trả lại . Trong các cuộc gọi tiếp theo, nó sẽ trả lại cùng một cá thể đã được tạo và trả lại cho bạn lần đầu tiên vì cách nhập / xuất mô-đun trong ES6.



Tương tự như vậy, chúng ta đã chuyển lớp Cơ sở dữ liệu của chúng ta thành một singleton bằng cách trả lại một thể hiện của lớp trong `module.exports` vì chúng ta chỉ cần một kết nối duy nhất với cơ sở dữ liệu.

ES6 làm cho chúng ta dễ dàng tạo mẫu singleton (ví dụ đơn) vì mô-đun hoạt động như thế nào bằng cách lưu trữ các phản hồi của một tệp tin được nhập trước đó.

Mongoose Schema vs. Model

Mongoose model bao gồm Mongoose schema. Mongoose schema xác định cấu trúc của tài liệu, các giá trị mặc định, xác nhận. trong khi Mongoose model cung cấp một giao diện cho cơ sở dữ liệu để tạo, truy vấn, cập nhật, xóa các bản ghi.

Tạo ra một mô hình Mongoose model bao gồm chủ yếu là ba phần:

1. Tham khảo Mongoose `let mongoose = require('mongoose')`

Tài liệu tham khảo này sẽ giống với tài liệu đã được trả về khi chúng tôi kết nối với cơ sở dữ liệu, có nghĩa là lược đồ và các định nghĩa mô hình sẽ không cần phải kết nối một cách rõ ràng với cơ sở dữ liệu.

2. Xác định Schema Một giản đồ định nghĩa tài sản tài sản thông qua một đối tượng mà tên khóa tương ứng với tên thuộc tính trong bộ sưu tập.

```
let emailSchema = new mongoose.Schema({
  email: String
})
```

Ở đây chúng tôi định nghĩa một thuộc tính được gọi là email với một kiểu lược đồ String ánh xạ tới một trình xác nhận nội bộ sẽ được kích hoạt khi mô hình được lưu vào cơ sở dữ liệu. Nó sẽ thất bại nếu kiểu dữ liệu của giá trị không phải là một kiểu chuỗi.

Các Loại schema sau được cho phép:

```
Array
Boolean
Buffer
Date
Mixed (A generic / flexible data type)
Number
ObjectId
String
```

Hỗn hợp và ObjectId được định nghĩa theo yêu cầu ('mongoose') Schema.Types.

3. Xuất mô hình Chúng ta gọi constructor mô hình trên Mongoose và truyền nó cho tên bộ sưu tập và một tham chiếu đến lược đồ.

```
module.exports = mongoose.model('Email', emailSchema)
```

Chúng ta hãy kết hợp đoạn mã trên thành `./src/models/email.js` để xác định nội dung của một mô hình cơ bản:

```
let mongoose = require('mongoose')
let emailSchema = new mongoose.Schema({
  email: String
})
module.exports = mongoose.model('Email', emailSchema)
```

Định nghĩa lược đồ phải đơn giản, nhưng tính phức tạp của nó thường dựa trên yêu cầu ứng dụng. Các lược đồ có thể được sử dụng lại và chúng cũng có thể chứa một số sơ đồ con. Trong ví dụ trên, giá trị của thuộc tính email là một loại giá trị đơn giản. Tuy nhiên, nó cũng có thể là một loại đối tượng với các thuộc tính bổ sung trên đó. Chúng ta có thể tạo ra một thể hiện của mô hình chúng ta đã định nghĩa ở trên và điền nó bằng cách sử dụng cú pháp sau:

```
let EmailModel = require('./email')
let msg = new EmailModel({
  email: 'ada.lovelace@gmail.com'
})
```

Hãy nâng cao lược đồ Email để làm cho thuộc tính email là một trường bắt buộc, và chuyển đổi giá trị sang chữ thường trước khi lưu nó. Chúng tôi cũng có thể thêm một chức năng xác nhận rằng sẽ đảm bảo rằng giá trị là một địa chỉ email hợp lệ. Chúng tôi sẽ tham khảo và sử dụng thư viện trình duyệt đã được cài đặt trước đó.

```
let mongoose = require('mongoose')
let validator = require('validator')
let emailSchema = new mongoose.Schema({
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    validate: (value) => {
      return validator.isEmail(value)
    }
  }
})

module.exports = mongoose.model('Email', emailSchema)
```

Hoạt động cơ bản Mongoose có một API linh hoạt và cung cấp nhiều cách để hoàn thành một nhiệm vụ. Chúng tôi sẽ không tập trung vào phần này vì nó nằm ngoài phạm vi của bài viết, nhưng hãy nhớ rằng hầu hết các hoạt động có thể được thực hiện theo nhiều cách bằng cú pháp khác nhau hoặc thông qua kiến trúc ứng dụng.

Create Record

Hãy tạo một thể hiện của mô hình email và lưu nó vào cơ sở dữ liệu:

```
let EmailModel = require('./email')

let msg = new EmailModel({
  email: 'ADA.LOVELACE@GMAIL.COM'
})
msg.save()
  .then(doc => {
    console.log(doc)
  })
  .catch(err => {
    console.error(err)
  })
```

Kết quả được trả lại khi lưu thành công:

```
{
  _id: 5a78fe3e2f44ba8f85a2409a,
  email: 'ada.lovelace@gmail.com',
  __v: 0
}
```

Các trường sau được trả về (Trường nội bộ được đặt trước bằng dấu gạch dưới "_").

Trường `_id` được Mongo tạo tự động và là một khóa chính của collection. Giá trị của nó là duy nhất.

Giá trị của trường `email` được trả lại. Lưu ý nó là chữ in thường vì thuộc tính đúng trong giản đồ

`lowercase:true`

`__v` là thuộc tính `versionKey` được đặt trên mỗi tài liệu khi được tạo bởi Mongoose. Giá trị của nó bao gồm nội dung tài liệu.

Nếu bạn cố gắng tạo tiếp với dữ liệu như trên, bạn sẽ nhận được một lỗi vì trường `email` chúng ta định nghĩa là duy nhất.

Fetch Record

Hãy thử lấy lại bản ghi chúng ta đã lưu vào cơ sở dữ liệu trước đó. Lớp mô hình cho thấy một số phương pháp tĩnh và để thực hiện trên cơ sở dữ liệu. Bây giờ chúng ta sẽ tìm kiếm hồ sơ mà chúng ta đã tạo ra trước đó bằng cách sử dụng phương pháp tìm và gửi email dưới dạng cụm từ tìm kiếm.

```
EmailModel
  .find({
    email: 'ada.lovelace@gmail.com' // search query
  })
  .then(doc => {
    console.log(doc)
  })
  .catch(err => {
    console.error(err)
  })
```

Tài liệu được trả về sẽ giống như những gì được hiển thị khi chúng tôi tạo bản ghi:

```
{
  _id: 5a78fe3e2f44ba8f85a2409a,
  email: 'ada.lovelace@gmail.com',
  __v: 0
}
```

Cập nhật Bản ghi Hãy sửa đổi bản ghi ở trên bằng cách thay đổi địa chỉ email và thêm một trường vào nó, tất cả trong một thao tác đơn lẻ. Vì lý do hiệu suất, Mongoose sẽ không trả lại tài liệu đã cập nhật nên chúng ta cần phải thông qua một tham số bổ sung để yêu cầu nó:

```
EmailModel
  .findOneAndUpdate(
    {
      email: 'ada.lovelace@gmail.com' // search query
    },
    {
      email: 'theoutlander@live.com' // field:values to update
    },
    {
      new: true, // return updated doc
      runValidators: true // validate before update
    })
  .then(doc => {
    console.log(doc)
  })
  .catch(err => {
    console.error(err)
  })
```

Tài liệu trả về sẽ chứa email đã cập nhật:

```
{
  _id: 5a78fe3e2f44ba8f85a2409a,
  email: 'theoutlander@live.com',
  __v: 0
}
```

Xóa hồ sơ Chúng tôi sẽ sử dụng lệnh `findOneAndRemove` để xóa một bản ghi. Nó trả về các tài liệu ban đầu đã được gỡ bỏ:

```
EmailModel
  .findOneAndRemove({
    email: 'theoutlander@live.com'
  })
  .then(response => {
    console.log(response)
  })
  .catch(err => {
    console.error(err)
  })
```

Người trợ giúp Chúng tôi đã xem xét một số chức năng cơ bản được biết đến như các hoạt động CRUD (Tạo, Đọc, Cập nhật, Xóa), nhưng Mongoose cũng cung cấp khả năng định cấu hình một số loại phương thức trợ giúp và thuộc tính. Chúng có thể được sử dụng để đơn giản hóa việc làm việc với dữ liệu.

Hãy tạo một giản đồ người dùng trong `/src/models/user.js` với `fieldsfirstName` và `lastName`:

```
let mongoose = require('mongoose')
let userSchema = new mongoose.Schema({
  firstName: String,
  lastName: String
})

module.exports = mongoose.model('User', userSchema)
```


Virtual Property Một số thuộc tính ảo không nhất thiết phải là tồn tại trong cơ sở dữ liệu. Chúng ta có thể thêm nó vào giản đồ của chúng ta như một người giúp đỡ để có được và thiết lập các giá trị.

Hãy tạo một thuộc tính ảo gọi là fullName có thể được sử dụng để đặt các giá trị trên firstName và lastName và lấy chúng như một giá trị kết hợp khi đọc:

```
userSchema.virtual('fullName').get(function() {
  return this.firstName + ' ' + this.lastName
})

userSchema.virtual('fullName').set(function(name) {
  let str = name.split(' ')
  this.firstName = str[0]
  this.lastName = str[1]
})
```

Gọi lại để nhận và đặt phải sử dụng từ khoá chức năng vì chúng ta cần truy cập vào mô hình thông qua từ khóa này. Sử dụng chức năng mũi tên chất béo sẽ thay đổi điều này để cập đến.

Bây giờ, chúng ta có thể đặt firstName và lastName bằng cách gán một giá trị cho fullName:

```
let model = new UserModel()
model.fullName = 'Thomas Anderson'
console.log(model.toJSON()) // Output model fields as JSON
console.log()
console.log(model.fullName) // Output the full name
```

Đoạn mã trên sẽ xuất ra những điều sau: { _id: 5a7a4248550ebb9fafd898cf, firstName: 'Thomas', lastName: 'Anderson' }

Các phương thức Chúng ta có thể tạo ra các phương thức tùy chỉnh trên lược đồ và truy cập chúng thông qua mô hình mẫu. Những phương pháp này sẽ có quyền truy cập vào đối tượng mô hình và chúng có thể được sử dụng. Ví dụ, chúng ta có thể tìm tất cả những người có cùng tên.

Trong ví dụ này, chúng ta hãy tạo một hàm để trả lại tên cho người dùng hiện tại. Hãy thêm một phương thức helper tùy chỉnh gọi là getInitials vào giản đồ:

```
userSchema.methods.getInitials = function() {
  return this.firstName[0] + this.lastName[0]
}
```

Phương pháp này sẽ có thể truy cập thông qua một mô hình ví dụ:

```
let model = new UserModel({
  firstName: 'Thomas',
  lastName: 'Anderson'
})

let initials = model.getInitials()
console.log(initials) // This will output: TA
```

Phương pháp tĩnh Tương tự như các phương thức dự, chúng ta có thể tạo các phương thức tĩnh trên lược đồ. Hãy tạo ra một phương pháp để lấy tất cả người dùng trong cơ sở dữ liệu:


```

userSchema.statics.getUsers = function() {
  return new Promise((resolve, reject) => {
    this.find((err, docs) => {
      if(err) {
        console.error(err)
        return reject(err)
      }
      resolve(docs)
    })
  })
}

```

Gọi getUsers trên lớp Model sẽ trả lại tất cả người dùng trong cơ sở dữ liệu:

```

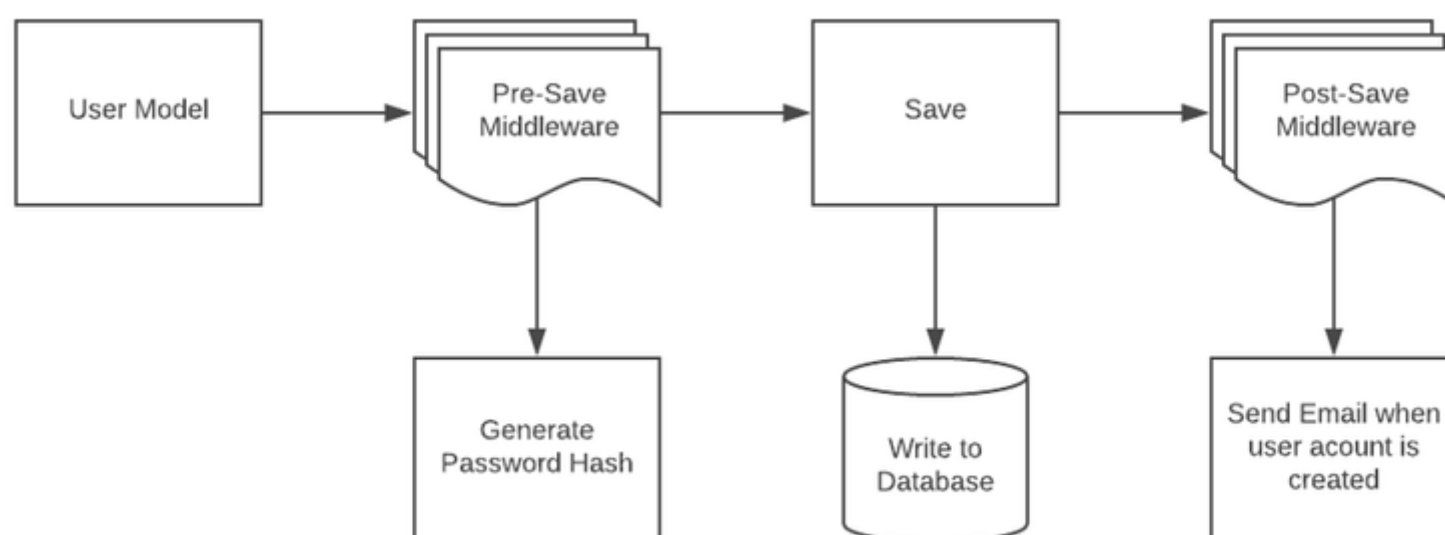
UserModel.getUsers()
  .then(docs => {
    console.log(docs)
  })
  .catch(err => {
    console.error(err)
  })

```

Thêm dự và phương pháp tĩnh là một cách tiếp cận tốt đẹp để thực hiện một giao diện tương tác cơ sở dữ liệu trên các bộ sưu tập và hồ sơ. Middleware Middleware là các chức năng chạy ở các giai đoạn cụ thể của đường ống dẫn. Mongoose hỗ trợ trung gian cho các hoạt động sau:

Tổng hợp Tài liệu Mô hình Truy vấn Ví dụ, các mô hình có chức năng trước và sau có hai tham số:

1. Loại sự kiện ('init', 'validate', 'save', 'remove')
2. Một gọi lại được thực hiện với điều này tham khảo ví dụ mô hình



Hãy thử một ví dụ bằng cách thêm hai trường được gọi là createdAt và updatedAt vào giản đồ của chúng ta:

```

let mongoose = require('mongoose')
let userSchema = new mongoose.Schema({
  firstName: String,
  lastName: String,
  createdAt: Date,
  updatedAt: Date
})

module.exports = mongoose.model('User', userSchema)

```

Khi được gọi là model.save (), có sự kiện pre('save', ...) và post('save', ...) được kích hoạt. Hãy thêm một Hook

```
userSchema.pre('save', function (next) {
  let now = Date.now()
  this.updatedAt = now
  // Set a value for createdAt only if it is null
  if (!this.createdAt) {
    this.createdAt = now
  }
  // Call the next function in the pre-save chain
  next()
})
```

Hãy tạo và lưu mô hình:

```
let UserModel = require('./user')
let model = new UserModel({
  fullName: 'Thomas Anderson'
})

msg.save()
  .then(doc => {
    console.log(doc)
  })
  .catch(err => {
    console.error(err)
  })
```

Bạn sẽ thấy các giá trị cho createdAt và updatedAt khi bản ghi được tạo ra được in:

```
{ _id: 5a7bbbeebc3b49cb919da675,
  firstName: 'Thomas',
  lastName: 'Anderson',
  updatedAt: 2018-02-08T02:54:38.888Z,
  createdAt: 2018-02-08T02:54:38.888Z,
  __v: 0 }
```

Giả sử chúng ta muốn theo dõi khi một bản ghi được tạo ra và cập nhật lần cuối trên mỗi bộ sưu tập trong cơ sở dữ liệu của chúng tôi. Thay vì lặp lại quá trình trên, chúng ta có thể tạo một plugin và áp dụng nó cho mọi schema. Hãy tạo tệp ./src/model/plugins/timestamp.js và nhân bản lại chức năng trên dưới dạng mô-đun có thể sử dụng lại được:

```
module.exports = function timestamp(schema) {
  // Add the two fields to the schema
  schema.add({
    createdAt: Date,
    updatedAt: Date
  })
  // Create a pre-save hook
  schema.pre('save', function (next) {
    let now = Date.now()
    this.updatedAt = now
    // Set a value for createdAt only if it is null
    if (!this.createdAt) {
      this.createdAt = now
    }
    // Call the next function in the pre-save chain
    next()
  })
}
```

Để sử dụng plugin này, chúng tôi chỉ cần truyền nó tới các lược đồ cần được cung cấp cho chức năng này:

```
let timestampPlugin = require('./plugins/timestamp')
emailSchema.plugin(timestampPlugin)
userSchema.plugin(timestampPlugin)
```

Query Building

Mongoose có một API rất phong phú để xử lý nhiều hoạt động phức tạp được hỗ trợ bởi MongoDB.

```
UserModel.find()           // find all users
  .skip(100)                // skip the first 100 items
  .limit(10)                // limit to 10 items
  .sort({firstName: 1})     // sort ascending by firstName
  .select({firstName: true}) // select firstName only
  .exec()                   // execute the query
  .then(docs => {
    console.log(docs)
  })
  .catch(err => {
    console.error(err)
  })
```

Kết luận

Chúng ta mới chỉ khám phá một số tính năng cơ bản của Mongoose. Đây là một thư viện phong phú đầy đủ các tính năng hữu ích và mạnh mẽ khi làm việc với các mô hình dữ liệu trong lớp ứng dụng.

Bạn có thể tương tác trực tiếp với Mongo bằng Mongo Driver, Mongoose sẽ đơn giản hóa tương tác đó bằng cách cho phép bạn mô hình các mối quan hệ giữa dữ liệu và xác nhận chúng một cách dễ dàng.

Link tham khảo

<https://www.codementor.io/theoutlander/introduction-to-mongoose-for-mongodb-gw9xw34el>

<http://mongoosejs.com/>

<https://www.mongodb.com/>

Have technical problem? [Ask on Viblo »](#)

More from Hoang Hung

[Tìm hiểu tạo Cron job trong laravel](#)

[Hoang Hung](#)

👁 58 📌 0 💬 0 ⬆ 3

[Tạo thông báo Real-Time với laravel](#)

[Hoang Hung](#)

👁 150 📌 3 💬 0 ⬆ 1

[Tìm hiểu Queue trong Laravel](#)

[Hoang Hung](#)

👁 148 📌 0 💬 0 ⬆ 0

[Triển khai ứng dụng nodejs lên heroku](#)

[Hoang Hung](#)

👁 194 📌 0 💬 0 ⬆ -1

Comments





↑ 0 ↓



RESOURCES

- [Posts](#)
- [Questions](#)
- [Videos](#)
- [Tags](#)
- [Authors](#)
- [Tools](#)

LINKS

-  [Facebook](#)
-  [GitHub](#)
-  [Browser extension](#)
-  [Atom plugin](#)

MOBILE APP

