

[Công nghệ ▾](#)[Tài liệu tham khảo & Hướng dẫn ▾](#)[Phản hồi ▾](#)[Đăng nhập](#) 

Hướng dẫn Express Phần 3: Sử dụng Database (với Mongoose)

[← Previous](#)[↑ Overview: Express Nodejs](#)[Next →](#)

Bài viết này giới thiệu tổng quan về cơ sở dữ liệu và cách dùng chúng với các ứng dụng Node/Express. Sau đó nó sẽ chỉ cho ta thấy cách sử dụng [Mongoose](#) để tạo ra kết nối đến cơ sở dữ liệu cho trang web [LocalLibrary](#). Nó giải thích cách mà schema và model của đối tượng được định nghĩa, các kiểu trường chính, và cách thức xác thực cơ bản. Nó còn trình bày một số cách chính để bạn có thể truy cập tới dữ liệu của model.

Bài viết trước: [Express Tutorial Part 2: Creating a skeleton website](#)

Mục tiêu: Có thể thiết kế và tự tạo model của riêng mình thông qua Mongoose.

Khái quát

Thủ thư sẽ dùng trang web Local Library để lưu trữ thông tin về sách và người mượn, trong khi các bạn đọc sẽ dùng nó để kiểm sách, tìm xem có bao nhiêu cuốn có sẵn, và tiếp tục như thế hoặc làm thủ tục mượn sách. Để có thể lưu trữ và truy xuất thông tin một cách hiệu quả, ta sẽ lưu trữ tất cả trong một *cơ sở dữ liệu*.

Các ứng dụng Express có thể dùng nhiều loại cơ sở dữ liệu khác nhau, và có khá là nhiều hướng tiếp cận để bạn có thể thi hành **Create, Read, Update and Delete (CRUD)**. Bài viết này sẽ cung cấp khái quát một số lựa chọn có thể áp dụng, và sẽ đi vào phân tích một phương pháp nhất định.

Tôi có thể dùng cơ sở dữ liệu nào?

Ứng dụng *Express* có thể dùng bất cứ cơ sở dữ liệu nào được hỗ trợ bởi *Node* (Chính *Express* không đưa ra bất cứ đặc tả chỉ tiết hành vi hay ràng buộc nào về hệ quản trị cơ sở dữ liệu). Thành ra có [rất nhiều](#) thứ để bạn thoải sức chọn lựa, bao gồm PostgreSQL, MySQL, Redis, SQLite, và MongoDB.

Khi chọn một cơ sở dữ liệu, bạn nên cân nhắc những thứ như là độ khó, hiệu năng, dễ dàng bảo trì, chi phí, sự hỗ trợ của cộng đồng, vân vân và mây mây. Do chưa có cơ sở dữ liệu nào đạt được danh hiệu 'tốt nhất', nên ta có thể lựa chọn hầu như mọi giải pháp vừa kể trên cho một trang cỡ nhỏ tới vừa như trang Local Library của chúng ta.

Để biết thêm chi tiết để tiện đường lựa chọn: [Database integration](#) (Tài liệu của Express).

Cách nào tốt nhất để thao tác với cơ sở dữ liệu?

Có hai hướng tiếp cận để thao tác với một cơ sở dữ liệu:

- Sử dụng ngôn ngữ truy vấn của riêng cơ sở dữ liệu đó (ví dụ như SQL)
- Sử dụng Object Data Model ("ODM") / Object Relational Model ("ORM"). ODM/ORM đại diện cho dữ liệu của trang web dưới dạng đối tượng trong JavaScript, sau đó đối chiếu với nền cơ sở dữ liệu bên dưới. Một vài ORMs được gắn với một cơ sở dữ liệu nào đó, trong khi số khác chỉ là một cầu nối giữa cơ sở dữ liệu và phần code backend.

Sử dụng ngôn ngữ truy vấn được hỗ trợ bởi cơ sở dữ liệu (như là SQL) sẽ đạt được *hiệu suất* cao nhất. ODM thường chậm hơn bởi nó phải thông dịch mã để có thể truy vấn giữa đối tượng và định dạng của cơ sở dữ liệu, mà không dùng được các truy vấn hiệu quả nhất

của cơ sở dữ liệu (điều này càng nghiêm trọng hơn khi ORM được sử dụng cho nhiều dạng cơ sở dữ liệu khác nhau, và phải tạo ra nhiều điều khoản lãng nhăng hơn đối với lượng tính năng được cơ sở dữ liệu hỗ trợ).

Điểm mạnh của ORM là lập trình viên có thể giữ tư duy như với đối tượng của JavaScript thay vì phải ngôn ngữ thuần túy viết riêng cho cơ sở dữ liệu — điều này càng đúng khi bạn phải làm việc với nhiều loại cơ sở dữ liệu (trên cùng hoặc khác trang web). ORM còn cung cấp các tính năng để xác thực và kiểm tra dữ liệu.

📌 **Mẹo:** Sử dụng ODM/ORMs thường giúp giảm thiểu chi phí phát triển và bảo trì! Trừ phi bạn đã quá thân thuộc với ngôn ngữ truy vấn thuần túy hoặc hiệu suất là trên hết, bạn nên cân nhắc đến việc sử dụng ODM.

Tôi nên dùng ORM/ODM nào?

Có nhiều giải pháp cho ODM/ORM có sẵn trên trang quản lý gói NPM (tìm theo nhãn [odm](#) và [orm](#)!).

Vào thời điểm viết bài này có một số giải pháp phổ biến như sau:

- [Mongoose](#): Mongoose là một công cụ mô hình hoá đối tượng [MongoDB](#), được thiết kế để làm việc trên môi trường bất đồng bộ.
- [Waterline](#): Một ORM trích xuất từ nền tảng web nền Express, [Sails](#). Nó cung cấp một bộ API tiêu chuẩn để truy cập vào vô số kiểu cơ sở dữ liệu khác nhau, bao gồm Redis, MySQL, LDAP, MongoDB, và Postgres.
- [Bookshelf](#): Trên nền promise và giao diện callback truyền thống, cung cấp giao tác hỗ trợ, eager/nested-eager relation loading, sự kết hợp đa hình, and hỗ trợ quan hệ một-một, một-nhiều, nhiều-nhiều. Làm việc với PostgreSQL, MySQL, và SQLite3.
- [Objection](#): Vận dụng hết sức mạnh của SQL và hạ tầng cơ sở dữ liệu bên dưới (hỗ trợ SQLite3, Postgres, và MySQL) theo cách dễ dàng nhất có thể.
- [Sequelize](#) là một ORM nền Promise dành cho Node.js và io.js. Nó hỗ trợ biên dịch PostgreSQL, MySQL, MariaDB, SQLite và MSSQL và hỗ trợ giao tác cứng, các quan hệ, read replication và nhiều hơn thế.

Như một luật ngầm định, bạn nên cân nhắc cả tính năng được công bố cũng như "hoạt động cộng đồng" (tải xuống, góp sức, báo lỗi, chất lượng của tài liệu, các thứ các thứ...) khi lựa chọn một giải pháp. Vào thời điểm viết bài thì Mongoose là một ORM phổ biến nhất, và là lựa chọn hợp lý nếu bạn dùng MongoDB làm cơ sở dữ liệu của mình.

Sử dụng Mongoose và MongoDB cho LocalLibrary

Ví dụ đối với *Local Library* (và cho cả phần còn lại của bài viết này) ta sẽ sử dụng [Mongoose ODM](#) để truy cập dữ liệu thư viện của chúng ta. Mongoose hoạt động như một frontend của [MongoDB](#), cơ sở dữ liệu mở dạng [NoSQL](#) sử dụng mô hình dữ liệu hướng document. Một "collection" và "documents", trong cơ sở dữ liệu MongoDB, [tương tự](#) với một "bảng" và "dòng" trong cơ sở dữ liệu quan hệ.

ODM và kết hợp cơ sở dữ liệu này cực kì phổ biến trong cộng đồng Node, phần lớn là bởi kho chứa document và hệ thống truy vấn khá là giống với JSON, vả dĩ nhiên rất đổi thân quen với các lập trình viên JavaScript.

📌 **Mẹo:** Bạn không cần phải biết gì về MongoDB để có thể sử dụng được Mongoose, mặc dù vài phần trong [tài liệu của Mongoose](#) sẽ dễ đọc hiểu hơn nếu bạn đã quen với MongoDB rồi.

Phần còn lại của bài viết này hướng dẫn cách để định nghĩa và truy cập schema và model của Mongoose thông qua ví dụ làm trang web cho *LocalLibrary*.

Thiết kế model LocalLibrary

Trước khi nhảy bổ vào code mô hình ầm ầm, sẽ tốt hơn nếu ta dành vài phút để nghĩ về dữ liệu ta cần phải lưu trữ và mối quan hệ giữa các đối tượng khác nhau.

Chúng ta đã biết rằng cần phải lưu trữ dữ liệu về sách (tựa đề, tóm tắt, tác giả, thể loại, mã số tiêu chuẩn quốc tế cho sách) và sẽ có khá nhiều cuốn giống nhau (với mã số quốc tế riêng biệt, tình trạng, vân vân...). Có lẽ ta sẽ cần lưu trữ nhiều thông tin về tác giả hơn chỉ là tên của người đó, và có vô số tác giả trùng hoặc có tên na ná nhau. Chúng ta muốn phân loại thông tin dựa theo tựa đề, tác giả, thể loại, và kiểu sách.

Công cuộc thiết kế mô hình yêu cầu thiết kế các mô hình riêng rẽ cho từng "đối tượng" (nhóm các thông tin có liên quan với nhau). Trong trường hợp này thì các đối tượng ấy hẳn là sách, các thuộc tính của sách, và tác giả.

Bạn chắc hẳn sẽ muốn biểu diễn mô hình dưới dạng danh sách liệt kê (ví dụ như một danh sách các lựa chọn), thay vì code cứng tất tần tật — việc này được đề nghị khi các lựa

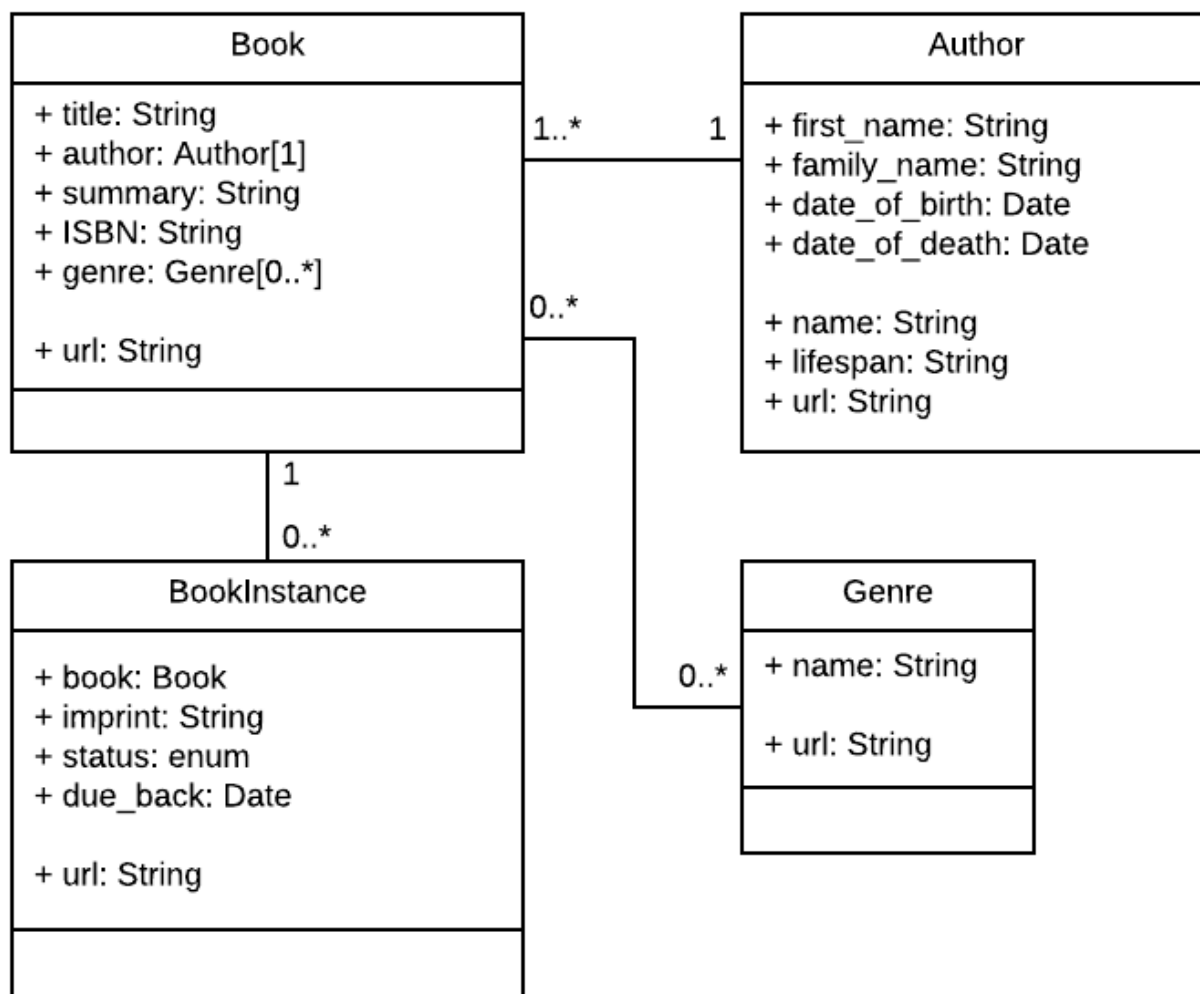
chọn vẫn chưa được liệt kê hết hoặc có thể bị thay đổi. Ứng viên sáng giá cho việc mô hình hoá này chính là thể loại sách (ví dụ như Khoa học Viễn tưởng, Ngôn tình, hoặc gì đó tương tự.)

Khi đã quyết định được các mô hình và trường dữ liệu, ta cần phải suy ngẫm về mối quan hệ giữa chúng.

Để làm tốt việc này, ta sẽ dùng biểu đồ liên hệ UML như bên dưới để biểu diễn các mô hình ta sắp định nghĩa ra (trong các hộp). Như đã nói ở trên, ta vừa tạo ra mô hình cho sách (chi tiết cơ bản nhất cho sách), phần tử của sách (lượng bản sách còn trong hệ thống), và tác giả. Chúng ta cũng vừa quyết định sẽ tạo thêm mô hình cho thể loại, để giá trị của nó thay đổi động. Ta cũng vừa quyết định sẽ không tạo mô hình cho `BookInstance:status` — chúng ta sẽ code cứng phần này đến một giá trị có thể chấp nhận được bởi ta không mong muốn giá trị của nó bị thay đổi. Bạn có thể thấy trong mỗi hộp là tên của mô hình, tên của các trường và kiểu dữ liệu tương ứng, đồng thời cả các thuộc tính và kiểu trả về nữa.

Biểu đồ còn chỉ ra mối quan hệ giữa các mô hình, bao gồm cả *bội số*. *Bội số* là các con số nhỏ nhỏ nằm trên các đường thẳng nối các hộp lại với nhau (lớn nhất và nhỏ nhất) để chỉ ra độ liên hệ trong các mối quan hệ giữa các mô hình với nhau. Lấy ví dụ như trong hình dưới, những đoạn kẻ nối giữa các hộp biểu diễn rằng Book và Genre liên quan đến nhau. Con số nằm gần với mô hình Book chỉ ra rằng Book phải có từ 0 đến nhiều Genre (bao nhiêu tùy thích), trong khi con số nằm ở đầu đoạn bên kia của Genre lại chỉ ra rằng nó có 0 hoặc nhiều liên hệ với Book.

❏ **Lưu ý:** Như đã nói trong [Mongoose primer](#) phía dưới, thường sẽ tốt hơn nếu có một trường riêng để định nghĩa mối quan hệ giữa documents/models chỉ trong *một* mô hình (bạn vẫn có thể tìm thấy mối quan hệ ngược lại bằng cách tìm kiếm `_id` liên quan trong các mô hình khác). Bên dưới biểu diễn mối quan hệ giữa Book/Genre and Book/Author trong Book schema, và mối quan hệ giữa Book/BookInstance trong BookInstance Schema. Việc lựa chọn này hơi cảm tính — ta hoàn toàn có thể định nghĩa các trường trong một schema khác.



❏ **Lưu ý:** Phần tiếp theo cung cấp kiến thức cơ bản giải thích cách mô hình được định nghĩa và sử dụng. Ta sẽ tìm cách để xây dựng đồng mô hình vừa vẽ ra trong biểu đồ trên.

Mongoose primer

Phần này giới thiệu khái quát cách để kết nối Mongoose với một cơ sở dữ liệu MongoDB, cách định nghĩa một schema và một model, và cách viết vài câu truy vấn đơn giản.

📌 **Lưu ý:** Cái primer này "bị ảnh hưởng mạnh" bởi [Mongoose quick start](#) trên *npm* và [official documentation](#).

Cài đặt Mongoose và MongoDB

Mongoose được cài đặt vào trong project của bạn (**package.json**) giống hệt như các dependency khác — dùng NPM. Để cài đặt nó, chạy câu lệnh sau trong thư mục project của bạn:

```
npm install mongoose
```

Sau khi cài xong, *Mongoose* sẽ tự động thêm mọi dependencies của nó, bao gồm cả driver cơ sở dữ liệu cho MongoDB, nhưng nó sẽ không cài đặt MongoDB đâu nhé. Nếu bạn muốn cài đặt một máy chủ MongoDB thì bạn có thể [tải xuống bộ cài tại đây](#), dành cho nhiều vô số hệ điều hành khác nhau và cài đặt nó trên hệ thống của mình. Bạn cũng có thể sử dụng MongoDB trên đám mây.

📌 **Lưu ý:** Trong bài viết này, ta sẽ sử dụng mLab, một *cơ sở dữ liệu được cung cấp dưới dạng dịch vụ* trên nền tảng điện toán đám mây và chọn [sandbox tier](#) nhé. Cái này khá hợp với khâu phát triển, và không phụ thuộc vào việc "cài đặt" hệ điều hành (cơ-sở-dữ-liệu-cung-cấp-dưới-dạng-dịch-vụ là một hướng tiếp cận nếu sử dụng trong dự án thật).

Kết nối với MongoDB

Mongoose yêu cầu một kết nối tới cơ sở dữ liệu MongoDB. Bạn có thể `require()` và kết nối cục bộ tới cơ sở dữ liệu thông qua `mongoose.connect()`, như bên dưới.

```
1 //Nhập mô-đun mongoose
2 var mongoose = require('mongoose');
3
4 //Thiết lập một kết nối mongoose mặc định
5 var mongodb = 'mongodb://127.0.0.1/my_database';
6 mongoose.connect(mongodb);
7 //Ép Mongoose sử dụng thư viện promise toàn cục
8 mongoose.Promise = global.Promise;
9 //Lấy kết nối mặc định
```

```
10 var db = mongoose.connection;  
11  
12 //Ràng buộc kết nối với sự kiện lỗi (để lấy ra thông báo khi có lỗi)  
13 db.on('error', console.error.bind(console, 'MongoDB connection error:'));
```

Bạn có thể lấy ra đối tượng mặc định Connection với `mongoose.connection`. Ngay khi đã kết nối, sự kiện sẽ nổ ra trên thuộc tính Connection.

❏ **Mẹo:** Nếu bạn muốn thêm mới các kết nối khác thì có thể dùng `mongoose.createConnection()`. Vẫn dùng chung định dạng URI (bao gồm máy chủ, cơ sở dữ liệu, cổng, lựa chọn khác.) như `connect()` và trả về một đối tượng Connection).

Định nghĩa và tạo ra mô hình

Mô hình được *định nghĩa* thông qua giao diện Schema. Schema định nghĩa các trường được lưu trong mỗi document đi kèm với điều kiện xác thực và giá trị mặc định cho chúng. Hơn nữa, bạn có thể khởi tạo các thuộc tính tĩnh và phương thức hỗ trợ để làm việc với kiểu dữ liệu của bạn dễ dàng hơn, và cả các đặc tính ảo để có thể dùng như bất cứ trường nào, mà không bị lưu vào trong cơ sở dữ liệu (ta sẽ bàn về vấn đề này sau).

Schema sau đó được "biên dịch" thành mô hình qua phương thức `mongoose.model()`. Một khi đã có mô hình thì bạn có thể dùng nó để tìm, thêm, sửa, và xoá các đối tượng cùng kiểu.

❏ **Lưu ý:** Mỗi mô hình có liên kết tới một *bộ sưu tập các tài liệu* trong cơ sở dữ liệu MongoDB. Documents sẽ chứa các trường/kiểu schema được định nghĩa trong mô hình Schema.

Định nghĩa schema

Đoạn code bên dưới trình bày cách thức tạo ra một Schema đơn giản. Đầu tiên bạn `require()` mongoose, rồi dùng phương thức khởi tạo của Schema để tạo ra một biến schema, định nghĩa một vài trường trong tham số truyền vào của phương thức khởi tạo.


```
1 //Require Mongoose
2 var mongoose = require('mongoose');
3
4 //Định nghĩa một schema
5 var Schema = mongoose.Schema;
6
7 var SomeModelSchema = new Schema({
8     a_string: String,
9     a_date: Date
10 });
```

Trong trường hợp trên ta chỉ có 2 trường, một string và một date. Trong phần tiếp theo, ta sẽ thêm một vài trường khác, xác thực, và một số phương thức khác.

Thêm mới một mô hình

Mô hình được tạo ra từ schema qua phương thức `mongoose.model()`:

```
1 // Định nghĩa schema
2 var Schema = mongoose.Schema;
3
4 var SomeModelSchema = new Schema({
5     a_string: String,
6     a_date: Date
7 });
8
9 // Biên dịch mô hình từ schema
10 var SomeModel = mongoose.model('SomeModel', SomeModelSchema );
```

Tham số thứ nhất là tên riêng cho collection sắp được tạo ra cho mô hình của bạn (Mongoose sẽ khởi tạo một collection cho mô hình *SomeModel* ở phía trên), và tham số thứ hai là schema mà bạn muốn dùng để tạo ra mô hình.

📌 **Lưu ý:** Khi đã có các lớp mô hình, bạn có thể sử dụng chúng để thêm, sửa, hoặc xóa các bản ghi, và để chạy các câu truy vấn lấy tất cả các bản ghi hoặc tạo các tập hợp con cho một số lượng bản ghi nhất định. Ta sẽ tìm hiểu việc này trong phần **Sử dụng mô hình**, và khi ta tạo khung nhìn.

Kiểu Schema (các trường)

Một schema có thể có số trường thông tin tùy ý — mỗi trường đại diện cho một document lưu trong *MongoDB*. Schema trong ví dụ bên dưới trình bày các kiểu đơn giản của các trường cũng như cách định nghĩa chúng.

```
1  var schema = new Schema(  
2  {  
3    name: String,  
4    binary: Buffer,  
5    living: Boolean,  
  
6    updated: { type: Date, default: Date.now },  
7    age: { type: Number, min: 18, max: 65, required: true },  
8    mixed: Schema.Types.Mixed,  
9    _someId: Schema.Types.ObjectId,  
10   array: [],  
11   ofString: [String], // Bạn có thể tạo mảng cho các trường khác  
12   nested: { stuff: { type: String, lowercase: true, trim: true } }  
13 })
```

Hầu hết các [SchemaTypes](#) (đồng miêu tả sau từ “type:” hoặc sau tên trường) đều tự định nghĩa chính nó. Ngoại trừ:

- **ObjectId**: Đại diện cho các thuộc tính đặc trưng của mô hình trong cơ sở dữ liệu. Chẳng hạn, một cuốn sách có thể dùng thứ này để đại diện cho tác giả của nó. Nó còn sẽ chứa cả ID đặc trưng (`_id`) cho đối tượng đặc trưng. Ta có thể dùng phương thức `populate()` để lấy các thông tin liên quan nếu cần thiết.
- [Mixed](#): Một kiểu schema chồng chập.
- `[]`: Mảng. Bạn có thể sử dụng các phương thức cho mảng riêng của JavaScript trên các mô hình này (`push`, `pop`, `unshift`, `shift`, `reduce`, vâng vâng và mây mây.). Ví dụ phía trên có một mảng đối tượng không định kiểu và một mảng đối tượng kiểu `String`, bạn vẫn có thể định nghĩa một mảng tùy ý kiểu đối tượng.

Đoạn code cũng chỉ ra hai cách để khai báo một trường:

- *Tên và kiểu* của trường là một cặp khoá-giá trị (ví dụ như với các trường `name`, `binary` và `living`).

- Tên trường chứa một đối tượng gồm có `type`, và nhiều *lựa chọn* khác. Lựa chọn bao gồm những thứ như là:
 - giá trị mặc định.
 - công cụ xác thực định sẵn (như giá trị `max/min`) và các hàm tùy chỉnh.
 - Trường ấy có bắt buộc (`required`) hay không
 - Trường kiểu `String` nên tự động ở kiểu chữ nhỏ, chữ to, hoặc tẩy gọn (ví dụ `{ type: String, lowercase: true, trim: true }`)

Để biết thêm chi tiết, mời bạn xem thêm [SchemaTypes](#) (Tài liệu của Mongoose).

Xác thực

Mongoose cung cấp một số hàm xác thực định sẵn và tùy chỉnh, và các hàm xác thực đồng bộ cũng như bất đồng bộ. Nó cho phép bạn đặc tả cả phạm vi chấp nhận hoặc giá trị và thông báo lỗi khi hàm xác thực gặp phải sự cố trong mọi trường hợp.

Các hàm xác thực định sẵn bao gồm:

- Tất cả các [SchemaTypes](#) đều có hàm xác thực là [required](#). Hàm này xác minh rằng liệu trường dữ liệu đó có bắt buộc phải được cung cấp nếu muốn lưu lại vào trong document hay không.
- [Numbers](#) có hai hàm là [min](#) và [max](#).
- [Strings](#) có:
 - [enum](#): đặc tả tập các giá trị được cho phép truyền vào trường tương ứng.
 - [match](#): đặc tả một regex để tạo luật cho xâu truyền vào.
 - [maxlength](#) và [minlength](#) cho xâu truyền vào.

Ví dụ bên dưới (hơi khác một chút so với tài liệu của Mongoose) chỉ ra cách để thêm các hàm xác minh và thông báo lỗi:

```
var breakfastSchema = new Schema({
  eggs: {
    type: Number,
    min: [6, 'Too few eggs'],
    max: 12
    required: [true, 'Why no eggs?']
  }
});
```

```
    },  
    drink: {  
      type: String,  
      enum: ['Coffee', 'Tea', 'Water', ]  
    }  
  });
```

Để biết thêm thông tin chi tiết về các hàm xác minh, hãy vào [Validation](#) (tài liệu của Mongoose) để tìm đọc.

Thuộc tính ảo

Thuộc tính ảo là các thuộc tính của document mà bạn có thể lấy ra và đặt lại mà không làm ảnh hưởng tới MongoDB. Hàm lấy ra hiệu quả cho việc định dạng lại hoặc kết hợp các trường, trong khi hàm đặt lại hữu dụng cho việc phân tách một giá trị riêng lẻ thành nhiều giá trị trong cơ sở dữ liệu. Ví dụ trong tài liệu khởi tạo (và huỷ tạo) một thuộc tính ảo tên đầy đủ từ trường họ và tên, điều đó sẽ dễ dàng và sạch sẽ hơn thay vì tạo ra một trường họ tên mỗi khi có ai đó sử dụng mẫu.

Lưu ý: Ta sẽ dùng thuộc tính ảo trong thư viện để định nghĩa một URL đặc trưng cho từng bản ghi của mô hình thông qua một đường dẫn và giá trị của mỗi bản ghi `_id` của mỗi bản ghi.

Để biết thêm chi tiết hãy vào [Virtuals](#) (tài liệu của Mongoose).

Phương thức và câu truy vấn trợ giúp

Một schema có thể còn có [phương thức biến](#), [phương thức tĩnh](#), và [hỗ trợ truy vấn](#). Phương thức biến và phương thức tĩnh gần như tương tự nhau, điểm khác biệt duy nhất là phương thức tĩnh liên kết với một bản ghi xác định và có quyền truy cập tới đối tượng hiện tại. Hỗ trợ truy vấn cho phép bạn mở rộng [chainable query builder API](#) của mongoose (ví dụ như, cho phép bạn thêm câu truy vấn "byName" sau các phương thức `find()`, `findOne()` và `findById()`).

Sử dụng mô hình

Ngay khi đã có một schema, bạn có thể dùng nó để tạo ra các mô hình. Mô hình đại diện cho một bộ sưu tập các tài liệu trong cơ sở dữ liệu mà bạn có thể tìm kiếm, trong khi các

phần tử của mô hình đại diện cho từng tài liệu mà bạn có thể lưu trữ và truy xuất.

Chúng ta chỉ có thể tìm hiểu sơ qua như trên thôi. Nếu muốn chi tiết hơn thì hãy vào xem:

🔗 [Models](#) (tài liệu của Mongoose).

Thêm mới và chỉnh sửa tài liệu

Để thêm mới một bản ghi, bạn có thể định nghĩa một phần tử của mô hình và sau đó dùng lời gọi `save()`. Ví dụ bên dưới chỉ ra rằng `SomeModel` là một đối tượng (chỉ có một trường là "name") mà ta vừa tạo ra từ schema của mình.

```
// Thêm mới một phần tử của mô hình SomeModel
var awesome_instance = new SomeModel({ name: 'awesome' });

// Lưu phần tử vừa thêm mới lại, thông qua việc truyền vào một hàm callback
awesome_instance.save(function (err) {
  if (err) return handleError(err);
  // saved!
});
```

Việc thêm bản ghi (đi kèm với sửa, xoá, và tìm kiếm) là các công việc bất đồng bộ — bạn phải truyền vào một hàm callback sau khi công việc hoàn tất. API sử dụng quy ước lỗi-trước, thế nên tham số thứ nhất trong hàm callback luôn là một giá trị lỗi (hoặc null). Nếu API trả về kết quả nào đó, nó sẽ được truyền vào qua tham số thứ hai.

Bạn cũng có thể sử dụng `create()` để vừa định nghĩa một phần tử của mô hình vừa lưu lại nó luôn. Hàm callback sẽ trả về một lỗi ứng với tham số thứ nhất và phần tử của mô hình vừa khởi tạo qua tham số thứ hai.

```
SomeModel.create({ name: 'also_awesome' }, function (err, awesome_instance) {
  if (err) return handleError(err);
  // lưu!
});
```

Mỗi mô hình đều có một kết nối liên quan (kết nối sẽ mặc định nếu dùng lệnh `mongoose.model()`). Bạn thêm mới một kết nối và gọi lệnh `.model()` để tạo thêm tài liệu trên một cơ sở dữ liệu khác.

Bạn có thể truy cập vào trường của bản ghi mới thông qua cú pháp chấm (.) , và thay đổi giá trị ở trong. Bạn sẽ phải gọi `save()` hoặc `update()` để lưu lại giá trị vừa chỉnh sửa vào cơ sở dữ liệu.

```
// Truy cập vào trường dữ liệu của bản ghi qua cú pháp (.)
console.log(awesome_instance.name); //sẽ in ra 'also_awesome'

// Thay đổi bản ghi bằng cách chỉnh sửa trường thông tin, sau đó gọi lệnh save().
awesome_instance.name="New cool name";
awesome_instance.save(function (err) {
  if (err) return handleError(err); // lưu!
});
```

Tìm kiếm các bản ghi

Bạn có thể tìm kiếm các bản ghi bằng các phương thức truy vấn, viết các câu truy vấn như đối với một tài liệu JSON. Đoạn code phía dưới trình bày cách tìm kiếm các vận động viên chơi tennis trong cơ sở dữ liệu, chỉ trả về các trường như *tên* và *tuổi* của vận động viên. Giờ ta sẽ chỉ xác định ra một trường (thể thao) bạn có thể thêm bao nhiêu tiêu chí tùy ý, xác định thêm các tiêu chí với regex, hoặc loại bỏ tất cả các điều kiện để trả về tất cả các vận động viên.

```
var Athlete = mongoose.model('Athlete', yourSchema);

// tìm tất cả các vận động viên chơi tennis, chọn hai trường 'name' và 'age'
Athlete.find({ 'sport': 'Tennis' }, 'name age', function (err, athletes) {
  if (err) return handleError(err);
  // 'athletes' chứa danh sách các vận động viên phù hợp với tiêu chí đã đề ra.
});
```

Nếu bạn ném vào một hàm callback, như ở trên, câu truy vấn sẽ được thực thi ngay lập tức. Hàm callback sẽ được gọi khi câu truy vấn hoàn tất.

📌 **Lưu ý:** Tất cả các hàm callback trong Mongoose sử dụng mẫu `callback(error, result)`. Nếu có lỗi xảy ra khi thực hiện câu truy vấn, tham số `error` sẽ chứa tất cả các lỗi, và `result` sẽ trở thành `null`. Nếu câu truy

vấn hợp lệ, tham số `error` sẽ trở thành `null`, và `result` sẽ chứa đựng kết quả của câu truy vấn.

Nếu bạn không truyền vào một hàm callback nào thì API sẽ trả về một giá trị kiểu [Query](#). Bạn có thể sử dụng đối tượng query này để kéo dài câu truy vấn trước khi thực thi nó (thông qua việc truyền vào một hàm callback) sau sử dụng phương thức `exec()`.

```
// tìm kiếm tất cả các vận động viên
var query = Athlete.find({ 'sport': 'Tennis' });

// chọn ra hai trường 'name' và 'age'
query.select('name age');

// giới hạn kết quả lại 5 bản ghi
query.limit(5);

// sắp xếp theo tên
query.sort({ age: -1 });

// thực thi câu truy vấn
query.exec(function (err, athletes) {
  if (err) return handleError(err);
  // athletes chứa một danh sách 5 vận động viên chơi tennis được xếp theo tên
})
```

Ở trên ta đưa tất cả điều kiện truy vấn vào trong phương thức `find()`. Thay vì vậy ta cũng có thể sử dụng hàm `where()`, và ta có thể xâu chuỗi các lời gọi bằng cú pháp chấm (.) thay vì phải gọi từng câu riêng rẽ. Đoạn code phía dưới làm y chang phần trên, thêm vài điều kiện cho trường tuổi.

```
Athlete.
  find().
  where('sport').equals('Tennis').
  where('age').gt(17).lt(50). //Điều kiện thêm vào sau hàm where
  limit(5).
  sort({ age: -1 }).
  select('name age').
  exec(callback); // callback ở đây là tên hàm callback của ta.
```

Phương thức `find()` lấy ra tất cả các bản ghi thoả mãn điều kiện, nhưng thường thì bạn chỉ muốn lấy ra một thôi. Các phương thức truy vấn phía dưới chỉ lấy ra một bản ghi:

- `findById()`: Tìm kiếm tài liệu theo `id` (mỗi tài liệu có một `id` duy nhất).
- `findOne()`: Tìm kiếm một tài liệu dựa theo tiêu chí đặt vào.
- `findByIdAndRemove()`, `findByIdAndUpdate()`, `findOneAndRemove()`, `findOneAndUpdate()`: Tìm kiếm một tài liệu theo `id` hoặc theo tiêu chí và sửa hoặc xoá nó. Đây là các mẫu có ích khi cần phải tìm kiếm và chỉnh sửa.

❏ **Lưu ý:** Còn có phương thức `count()` để đếm số lượng bản ghi phù hợp với điều kiện đề ra. Cái này sẽ có ích nếu bạn chỉ cần tìm ra số lượng thay vì phải kéo về tất cả các bản ghi.

Còn có nhiều thứ nữa mà bạn có thể làm với các câu truy vấn. Để biết thêm thông tin mời xem: [Queries](#) (tài liệu Mongoose).

Làm việc với tài liệu liên quan — sự cư ngụ

Bạn có thể thêm mối liên quan giữa các tài liệu/phần tử của mô hình qua trường `schema ObjectId`, hoặc từ một tài liệu đến nhiều qua một mảng `ObjectIds`. Trường này lưu trữ `id` của mô hình liên quan. Nếu bạn muốn lấy nội dung của tài liệu liên quan, bạn có thể sử dụng phương thức `populate()` trong câu truy vấn để thay thế `id` với đồng dữ liệu tương ứng.

Chẳng hạn, schema sau đây định nghĩa tác giả và tác phẩm. Mỗi tác giả có thể có nhiều tác phẩm, nên ta sử dụng một mảng đối tượng `ObjectId`. Mỗi tác phẩm có thể có một tác giả. Thuộc tính `"ref"` (được in đậm) kể cho ta biết schema nào mà model có thể gắn vào được.

```
var mongoose = require('mongoose')
, Schema = mongoose.Schema

var authorSchema = Schema({
  name      : String,
  stories   : [{ type: Schema.Types.ObjectId, ref: 'Story' }]
});
```



```
var storySchema = Schema({
  author : { type: Schema.Types.ObjectId, ref: 'Author' },
  title   : String
});

var Story = mongoose.model('Story', storySchema);
var Author = mongoose.model('Author', authorSchema);
```

Ta có thể lưu lại đồng liên quan đến tài liệu tương ứng bằng cách gán cho nó giá trị `_id`. Đoạn bên dưới ta tạo ra một tác giả, rồi một cuốn sách và gán id của tác giả vào trường tác giả của tác phẩm.

```
var bob = new Author({ name: 'Bob Smith' });

bob.save(function (err) {
  if (err) return handleError(err);

  //Bob giờ đã tồn tại, đến lúc tạo tác phẩm rồi
  var story = new Story({
    title: "Bob goes sledding",
    author: bob._id // gán _id của tác giả Bob. ID này được tạo ra mặc định!
  });

  story.save(function (err) {
    if (err) return handleError(err);
    // Bob giờ đã có tác phẩm của mình
  });
});
```

Tài liệu tác phẩm của ta giờ được nối với trường tác giả thông qua ID của trong tài liệu tác giả. Để lấy thông tin của tác giả ta dùng hàm `populate()`, như bên dưới.

```
Story
.findOne({ title: 'Bob goes sledding' })
.populate('author') //Thay thế ID của tác giả bằng thông tin của tác giả!
.exec(function (err, story) {
  if (err) return handleError(err);
  console.log('The author is %s', story.author.name);
});
```

```
// in ra "The author is Bob Smith"
});
```

- ❏ **Lưu ý:** Ta vừa thêm tác giả vào tác phẩm, nhưng lại không hề thêm tác phẩm vào mảng `stories` của tác giả. Thế thì làm thế nào để lấy ra tất cả tác phẩm của một tác giả nào đó? Có một cách là thêm tác giả vào mảng tác phẩm, nhưng như thế sẽ thành ra phân vị trí các thành phần trong khi ta cần giữ cho mối liên hệ giữa tác giả với tác phẩm được bảo toàn.

Cách tốt hơn là lấy `_id` của *tác giả*, rồi dùng `find()` để tìm trong trường tác giả xuyên suốt tác phẩm.

```
Story
.find({ author : bob._id })
.exec(function (err, stories) {
  if (err) return handleError(err);
  // trả về tất cả các tác phẩm có id của Bob.
});
```

Đến đây là đã đủ hết mọi thứ bạn cần biết trong *bài viết này rồi*. Để biết thêm thông tin chi tiết, mời bạn tham khảo [Population](#) (tài liệu của Mongoose).

Một schema/mô hình trên một tập tin

Dù bạn có thể tạo ra schema và mô hình theo bất cứ kiến trúc nào bạn thích, nhưng chúng tôi vẫn khuyến nghị nên định nghĩa chúng trên mỗi mô đun riêng rẽ (tập tin), rồi xuất mô hình ra ngoài. Làm như thế này này:

```
// Tập tin: ./models/somemodel.js

//Nhập Mongoose
var mongoose = require('mongoose');

//Định nghĩa một schema
var Schema = mongoose.Schema;

var SomeModelSchema = new Schema({
  a string      : String,
```

```
    a_date      : Date,
  });

//Xuất ra lớp mô hình "SomeModel"
module.exports = mongoose.model('SomeModel', SomeModelSchema );
```

Sau đó bạn có thể nhập và sử dụng mô hình ngay lập tức ở bất cứ đâu. Dưới đây là cách bạn lấy ra mọi phần tử của mô hình.


```
//Thê mới mô hình SomeModel thông qua lệnh require
var SomeModel = require('../models/somemodel')

// Sử dụng đối tượng SomeModel để tìm tất cả bản ghi của SomeModel
SomeModel.find(callback_function);
```

Thiết lập cơ sở dữ liệu MongoDB

Giờ khi đã hiểu những gì Mongoose có thể làm và cách để ta có thể thiết kế một cơ sở dữ liệu, đến lúc thực hành trên trang web *LocalLibrary* rồi. Bước đầu tiên trong bài thực hành là tạo mới một cơ sở dữ liệu MongoDB để lưu trữ dữ liệu cho thư viện của chúng ta.

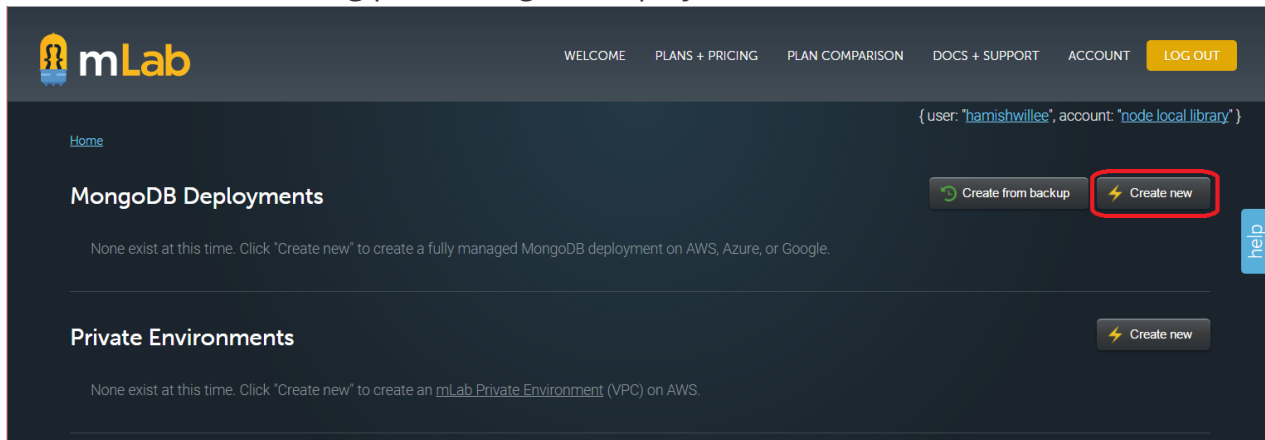
Trong bài viết này ta sẽ sử dụng cơ sở dữ liệu nền điện toán đám mây của [mLab](#) (chọn kiểu "[sandbox](#)" để dùng miễn phí). Cơ sở dữ liệu kiểu này không phù hợp dành cho các trang web thật vì nó không có dư thừa dữ liệu, nhưng lại rất hợp dành cho việc phát triển và xây dựng mẫu vật. Và bởi nó dễ dùng cũng như dễ thiết lập, và đừng quên rằng mLab là một trong những bên cung cấp khá nổi tiếng *cơ sở dữ liệu cung cấp dưới dạng dịch vụ* mà bạn có thể sẽ dùng cho dự án thật (vào thời điểm viết bài này bạn cũng có thể chọn các nhà cung cấp như [Compose](#), [ScaleGrid](#) và [MongoDB Atlas](#)).

 **Lưu ý:** Nếu bạn muốn thiết lập cơ sở dữ liệu MongoDB cục bộ thì hãy tìm và tải xuống [bản phù hợp](#) với hệ thống của mình. Phần còn lại khá là đơn giản, trừ phần URL của cơ sở dữ liệu mà bạn sẽ phải xác định nếu muốn kết nối tới.

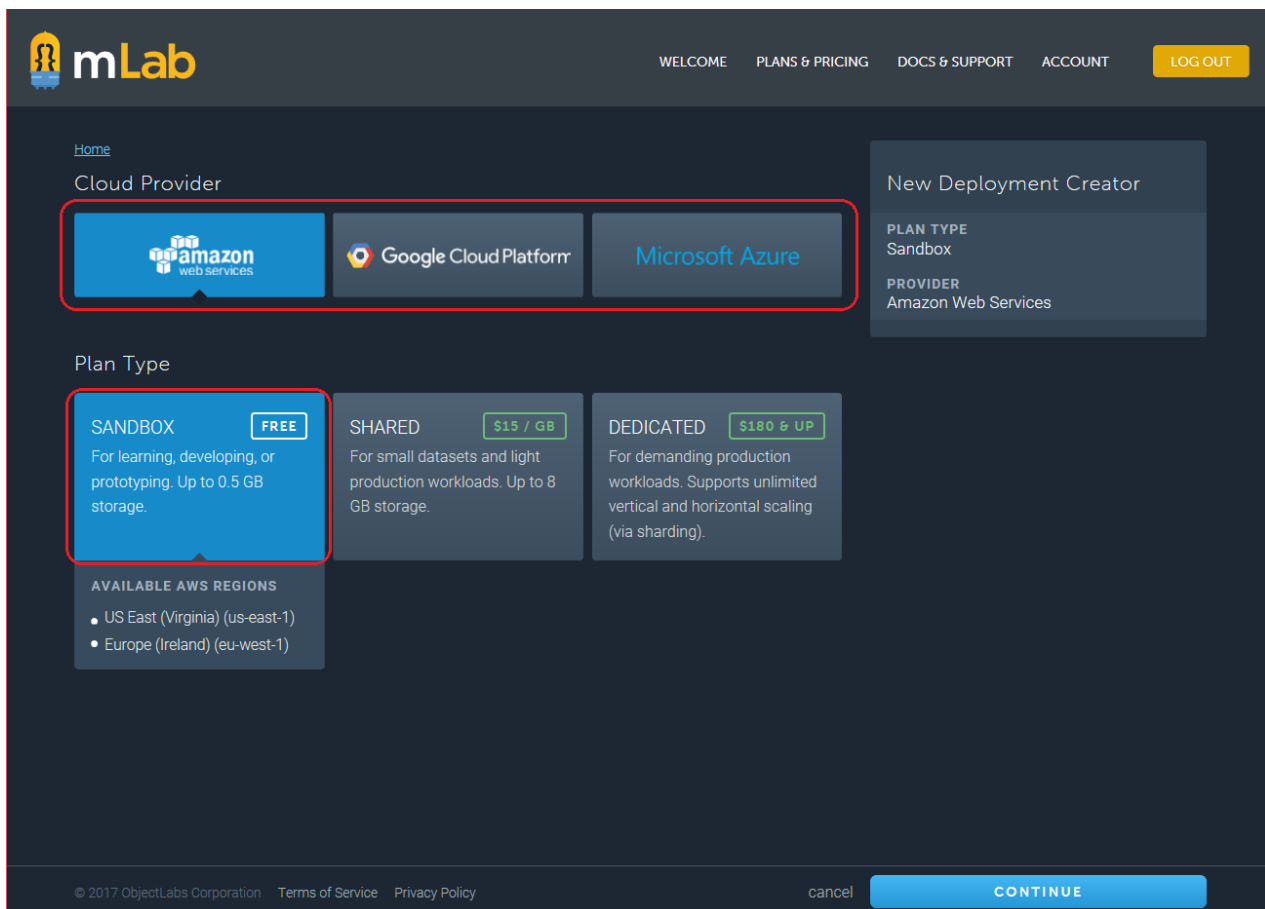
Trước hết bạn cần [tạo tài khoản mLab](#) (miễn phí, chỉ cần điền mẫu đăng ký là xong).

Sau khi đã đăng nhập vào, bạn sẽ được chuyển tới [trang chủ](#):

1. Nhấn **Create New** trong phần *MongoDB Deployments*.

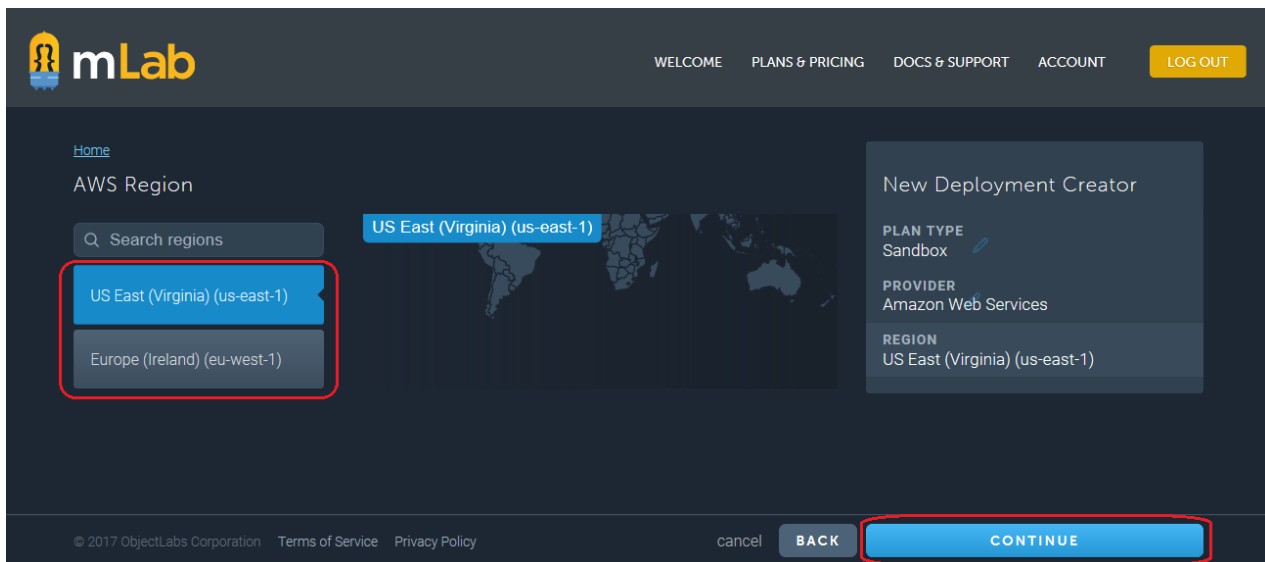


2. Nó sẽ mở ra màn hình *Cloud Provider Selection*.



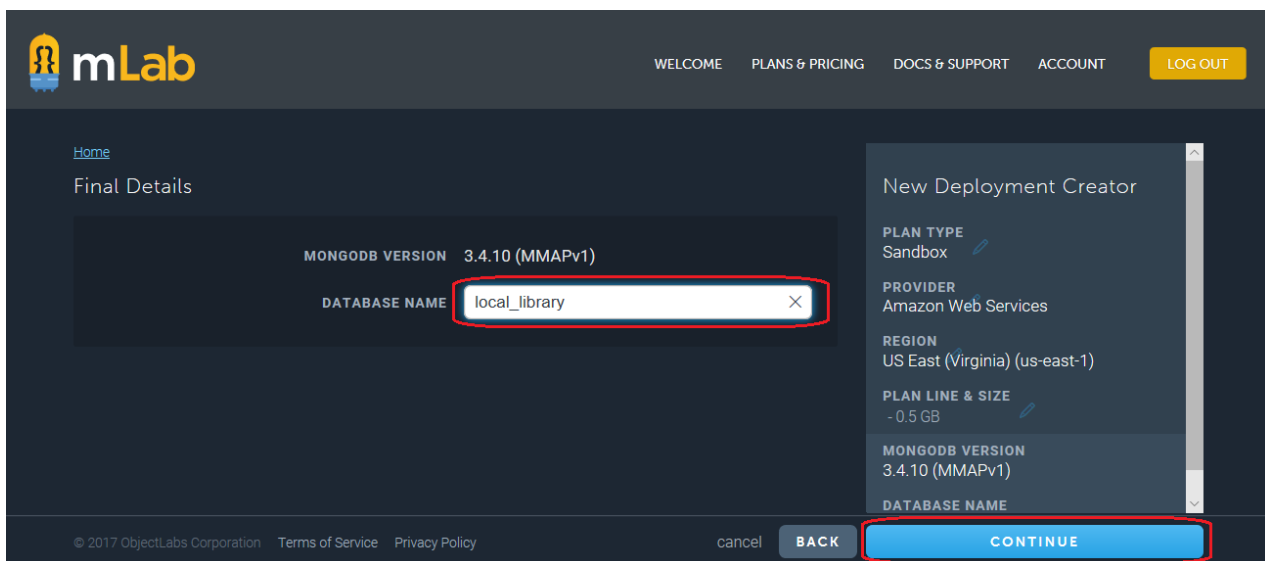
- Chọn **SANDBOX** (Free) plan trong phần *Plan Type*.
- Chọn bất cứ nhà cung cấp nào trong phần *Cloud Provider*. Mỗi nhà cung cấp khác nhau ở các vùng lãnh thổ địa lý khác nhau (ở ngay dưới phần selected plan type).
- Bấm nút **Continue**.

3. Màn hình *Select Region* mở ra.



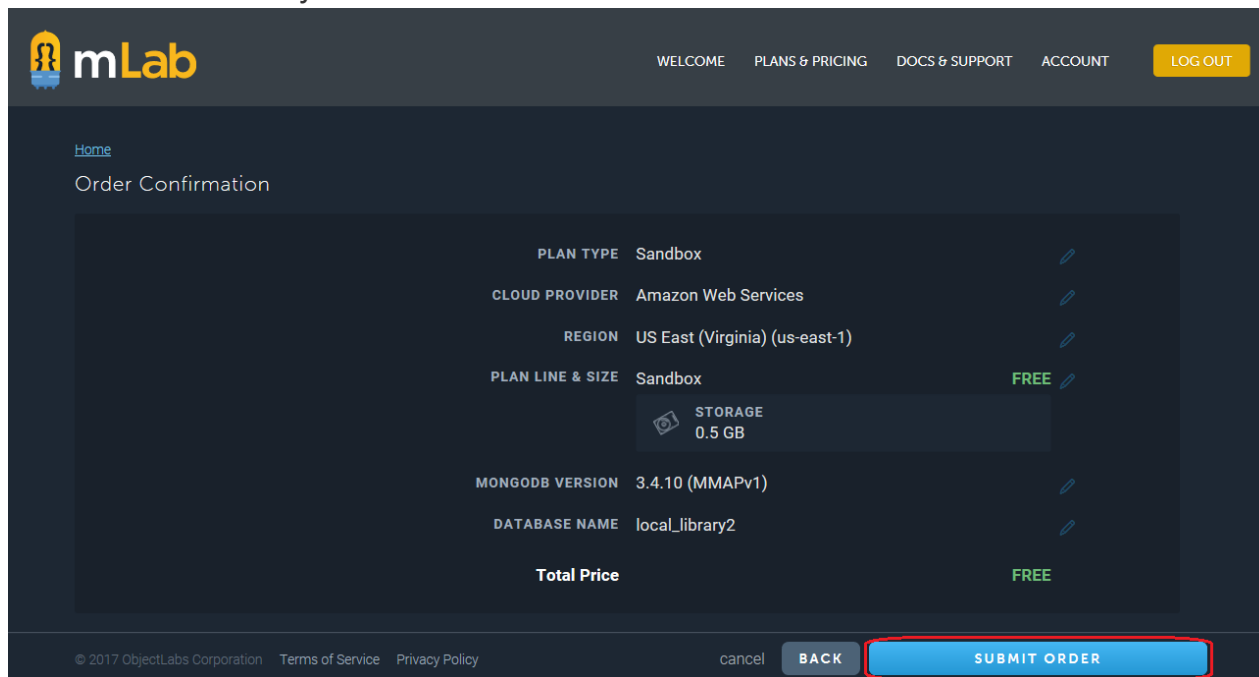
- Chọn vùng lãnh thổ gần bạn nhất rồi nhấn **Continue**.

4. Màn hình *Final Details* mở ra.



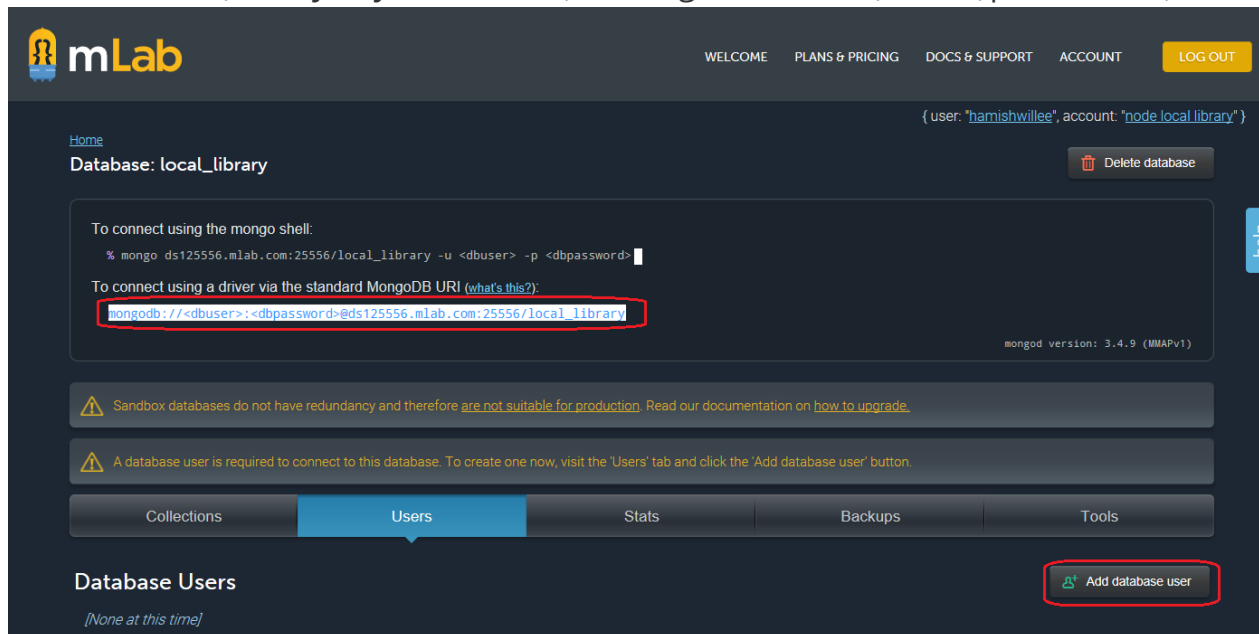
- Nhập tên cho cơ sở dữ liệu vừa tạo ra như `local_library` và chọn **Continue**.

5. Màn hình *Order Confirmation* sẽ mở lên.



- Nhấn **Submit Order** để tạo mới cơ sở dữ liệu.

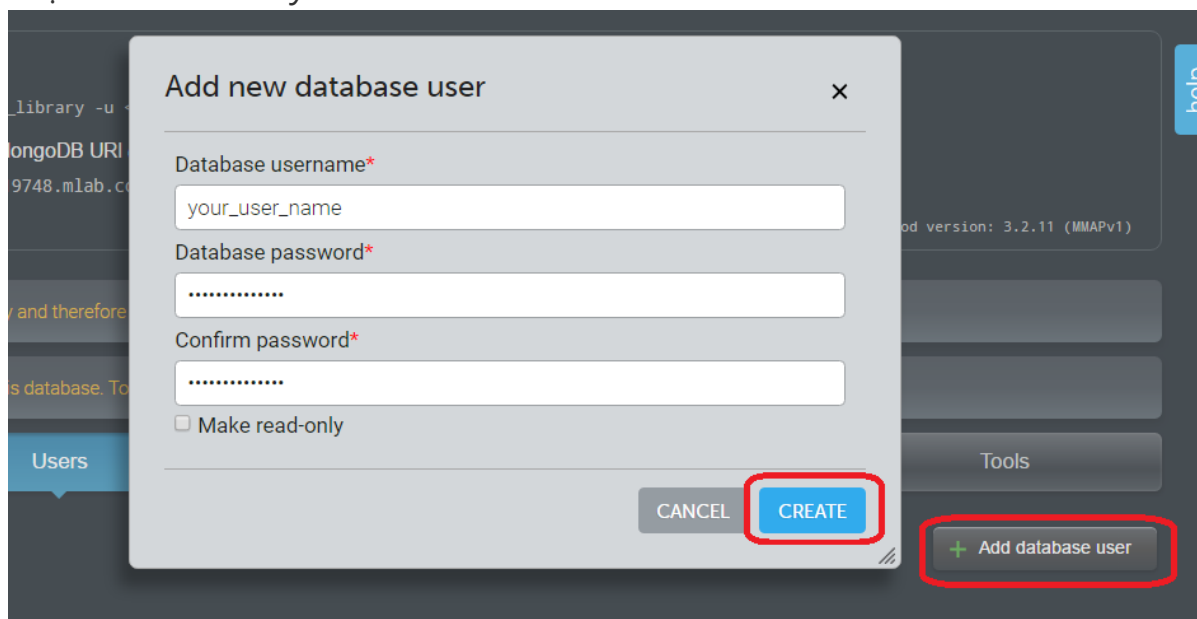
6. Bạn sẽ được điều hướng về trang chủ. Nhấn vào cơ sở dữ liệu vừa thêm mới để xem chi tiết. Như bạn thấy đấy, cơ sở dữ liệu không có bất cứ bộ sưu tập nào (dữ liệu).



URL mà bạn cần ở ngay đầu trang (trong phần khoanh đỏ). Để dùng được nó bạn phải tạo ra người dùng mới.

7. Nhấn vào tab **Users** và bấm nút **Add database user**.

8. Điền tên đăng nhập và mật khẩu (hai lần), và nhấn nút **Create**. Đừng bao giờ chọn *Make read only*.



Giờ đã có cơ sở dữ liệu rồi, và cả URL (với tên đăng nhập và mật khẩu) đã sẵn sàng để truy xuất. Trông nó sẽ như thế này:

`mongodb://your_user_namer:your_password@ds119748.mlab.com:19748/local_library.`

Cài đặt Mongoose

Mở command prompt và chuyển tới thư mục chứa trang web Local Library. Nhập lệnh dưới để cài đặt Mongoose (và đồng dependency của nó) và nó sẽ tự động thêm vào **package.json** của bạn, nếu bạn đã làm như với Mongoose Primer ở trên.

```
1 | npm install mongoose --save
```

Kết nối tới MongoDB

Mở **/app.js** (trong project của bạn) và sao chép đồng phía dưới để khai báo *đối tượng ứng dụng Express* (sau dòng `var app = express();`). Thay thế url của cơ sở dữ liệu (`'insert_your_database_url_here'`) bằng URL của mình (như cái vừa tạo ra bằng mLab).

```
1 //Thiết lập kết nối tới Mongoose
2 var mongoose = require('mongoose');
3 var mongoDB = 'insert_your_database_url_here';
4 mongoose.connect(mongoDB);
5 mongoose.Promise = global.Promise;
6 var db = mongoose.connection;
7 db.on('error', console.error.bind(console, 'MongoDB connection error:'));
```

Như đã nói trong phần Mongoose primer phía trên, đoạn code này tạo ra kết nối mặc định tới cơ sở dữ liệu và ràng buộc sự kiện lỗi (để in lỗi ra console).

Định nghĩa Schema cho LocalLibrary

Ta sẽ tạo ra mô đun cho từng mô hình, như đã đề cập phía trên. Bắt đầu bằng cách thêm mới thư mục trong thư mục gốc (**/models**) và tạo từng tập tin cho mỗi mô hình:

```
1 /express-locallibrary-tutorial //the project root
2 /models
3   author.js
4   book.js
5   bookinstance.js
6   genre.js
```

Mô hình tác giả

Sao chép đoạn code schema Author code bên dưới và dán vào tập tin **./models/author.js**. Scheme định nghĩa rằng một tác giả có kiểu `String` SchemaTypes cho hai trường họ và tên, bắt buộc và có giới hạn nhiều nhất 100 ký tự, và kiểu `Date` cho trường ngày sinh và ngày mất.


```
1 var mongoose = require('mongoose');
2
3 var Schema = mongoose.Schema;
4
5 var AuthorSchema = new Schema(
6   {
7     first_name: {type: String, required: true, max: 100},
8     family_name: {type: String, required: true, max: 100},
9     date_of_birth: {type: Date},
10    date_of_death: {type: Date},
11  }
12 );
13
14 // Tạo phương thức ảo cho tên đầy đủ
15 AuthorSchema
16   .virtual('name')
17   .get(function () {
18     return this.family_name + ', ' + this.first_name;
19   });
20
21 // Phương thức ảo cho URL của tác giả
22 AuthorSchema
23   .virtual('url')
24   .get(function () {
25     return '/catalog/author/' + this._id;
26   });
27
28 //xuất mô hình
29 module.exports = mongoose.model('Author', AuthorSchema);
```

Ta vừa khai báo phần ảo cho AuthorSchema với tên là "url" trả về URL tuyệt đối bắt buộc để lấy ra phần tử nhất định của mô hình — ta sẽ dùng thuộc tính này trong mẫu mỗi khi cần lấy ra đường dẫn tới tác giả.

❏ **Lưu ý:** Khai báo URL bằng hàm ảo trong schema là ý tưởng tốt bởi vì URL sẽ chỉ cần thay đổi tại một nơi.

Vào lúc này các URL sẽ không thể hoạt động, ta chưa đặt ra các route nào để dẫn lối cho từng phần tử của mô hình. Ta sẽ làm việc này trong các bài viết sau!

Sau khi đã xong thì ta xuất mô hình ra thôi.

Mô hình sách

Sao chép đoạn code schema Book bên dưới và dán nó vào tập tin `./models/book.js`. Làm tương tự đối như với tác giả — ta vừa khai báo một schema có nhiều trường String và một phần ảo để lấy URL của các bản ghi sách, sau đó thì xuất nó ra.

```
1  var mongoose = require('mongoose');
2
3  var Schema = mongoose.Schema;
4
5  var BookSchema = new Schema(
6    {
7      title: {type: String, required: true},
8      author: {type: Schema.ObjectId, ref: 'Author', required: true},
9      summary: {type: String, required: true},
10     isbn: {type: String, required: true},
11     genre: [{type: Schema.ObjectId, ref: 'Genre'}]
12   }
13 );
14
15 // Tạo hàm ảo lấy URL của sách
16 BookSchema
17   .virtual('url')
18   .get(function () {
19     return '/catalog/book/' + this._id;
20   });
21
22 //Xuất mô hình
23 module.exports = mongoose.model('Book', BookSchema);
```

Sự khác biệt chính là ta vừa tạo ra hai mối liên quan đến sách:

- author được trỏ tới mô hình đối tượng Author, và bắt buộc.
- genre được trỏ tới một mảng mô hình đối tượng Genre. Ta vẫn chưa định nghĩa đối tượng này!

Mô hình BookInstance

Cuối cùng sao chép đoạn code schema BookInstance bên dưới và dán nó vào tập tin `./models/bookinstance.js`. BookInstance đại diện cho số bản sách mà ai đó mượn, và bao gồm thông tin về thời điểm sách về hoặc hạn trả sách dự kiến, "đánh dấu" hoặc lấy chi tiết phiên bản.

```
1  var mongoose = require('mongoose');
2
3  var Schema = mongoose.Schema;
4
5  var BookInstanceSchema = new Schema(
6    {
7      book: { type: Schema.ObjectId, ref: 'Book', required: true }, //reference
8      imprint: {type: String, required: true},
9      status: {type: String, required: true, enum: ['Available', 'Maintenance',
10      due_back: {type: Date, default: Date.now}
11    }
12  );
13
14  // Lấy ra URL của bookinstance
15  BookInstanceSchema
16  .virtual('url')
17  .get(function () {
18    return '/catalog/bookinstance/' + this._id;
19  });
20
21  //Xuất mô hình
22  module.exports = mongoose.model('BookInstance', BookInstanceSchema);
```

Các thuộc tính mới được thêm vào trong này nằm trong phần trường dữ liệu:

- `enum`: Cho phép ta đặt giá trị chấp nhận được cho xâu truyền vào. Trong trường hợp này ta dùng nó để xác định trạng thái còn sẵn của sách (sử dụng enum sẽ giúp ta tránh khỏi các lỗi chính tả hoặc khai khổng giá trị cho trạng thái)
- `default`: Ta dùng default để đặt giá trị mặc định cho những bookinstances mới khởi tạo để bảo trì `due_back` mặc định `now` (lưu ý cách gọi hàm `Date` khi thiết lập ngày giờ!)

Những schema còn lại làm tương tự.

Mô hình thể loại - thử thách!

Mở tập tin `./models/genre.js` của bạn lên và tạo mới một schema để lưu lại thể loại sách (các kiểu sách như là truyện tiểu thuyết, tư liệu lịch sử...).

Cách định nghĩa cũng giống như các mô hình ở trên:


- Mô hình nên có một `String` `SchemaType` tên là `name` để mô tả thể loại.
- Tên này phải bắt buộc và có từ 3 đến 100 kí tự.
- Tạo một phương thức ảo cho URL của thể loại, để tên là `url`.
- Xuất mô hình.

Kiểm thử — tạo ra vài bản ghi

Xong xuôi rồi đó. Giờ ta đã có tất cả mô hình!

Để có thể kiểm thử mô hình (và để tạo ra vài sách mẫu và một số thứ khác ta sẽ dùng trong bài viết sau) ta sẽ chạy một đoạn kịch bản *independent* để tạo ra các bản ghi cho từng mô hình:

1. Tải về (hoặc tạo mới) tập tin `populatedb.js` trong thư mục `express-locallibrary-tutorial` (đồng cấp với `package.json`).

 **Lưu ý:** Bạn không cần hiểu cách thức `populatedb.js` hoạt động; nó chỉ thêm dữ liệu mẫu vào trong cơ sở dữ liệu thôi.

2. Nhập lệnh phía dưới vào trong thư mục chứa project để cài đặt mô-đun `async` để có thể chạy được đoạn kịch bản (ta sẽ bàn về việc này trong bài tiếp theo,)

```
1 | npm install async --save
```

3. Chạy đoạn kịch bản bằng `node` trong command prompt của bạn, truyền vào URL của cơ sở dữ liệu MongoDB (như cái bạn đã thay thế cho `insert_your_database_url_here`, trong `app.js` phía trên):

```
1 | node populatedb <your mongodb url>
```

4. Đoạn code sẽ chạy thành công và in ra những vật được tạo ra trên màn console.

📌 **Mẹo:** Lên cơ sở dữ liệu của bạn trên [mLab](#). Giờ bạn có thể chui vào bộ sưu tập Books, Authors, Genres và BookInstances, và kiểm tra các tài liệu vừa được tạo.

Tóm lại

Trong bài viết này ta học một chút về cơ sở dữ liệu và ORMs trên Node/Express, và cách để định nghĩa schema và mô hình của Mongoose. Sau đó ta đã thực hành thiết kế và triển khai các mô hình Book, BookInstance, Author và Genre cho trang web *LocalLibrary*.

Sau cùng ta kiểm thử những gì vừa viết ra bằng cách tạo một đồng các phần tử (bằng cách sử dụng một đoạn mã kịch bản). Trong bài tiếp theo ta sẽ học cách tạo các trang để trình bày các thứ.

Đọc thêm

- [Database integration](#) (tài liệu của Express)
- [Mongoose website](#) (tài liệu của Mongoose)
- [Mongoose Guide](#) (tài liệu của Mongoose)
- [Validation](#) (tài liệu của Mongoose)
- [Schema Types](#) (tài liệu của Mongoose)
- [Models](#) (tài liệu của Mongoose)
- [Queries](#) (tài liệu của Mongoose)
- [Population](#) (tài liệu của Mongoose)

In this module

- Express/Node introduction
 - Setting up a Node (Express) development environment
 - Express Tutorial: The Local Library website
 - Express Tutorial Part 2: Creating a skeleton website
 - Express Tutorial Part 3: Using a Database (with Mongoose)
 - Express Tutorial Part 4: Routes and controllers
 - Express Tutorial Part 5: Displaying library data
 - Express Tutorial Part 6: Working with forms
 - Express Tutorial Part 7: Deploying to production
-