Mongoose Schema vs. Model

A **Mongoose model** is a **wrapper** on the **Mongoose schema**. A Mongoose schema **defines the structure** of the document, default values, validators, etc... whereas a Mongoose model **provides an interface** to the database for creating, querying, updating, deleting records, etc.

Creating a Mongoose model comprises primarily of three parts:

1. Referencing Mongoose

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

This reference will be the same as the one that was returned when we connected to the database, which means the schema and model definitions will not need to explicitly connect to the database.

Mongoose Schema vs. Model

2. Defining the Schema

A schema defines document properties through an object where the key name corresponds to the property name in the collection.

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

Here we define a property called **email** with a schema type **String** which maps to an internal validator that will be triggered when the model is saved to the database. It will fail if the data type of the value is not a string type.

The following Schema Types are permitted:

- Array
- Boolean
- Buffer
- Date

- Mixed (A generic / flexible data type)
- Number
- ObjectId
- String

Mixed and ObjectId are defined under `require('mongoose').Schema.Types`.

Mongoose Schema vs. Model

3. Exporting a Model

We need to call the model constructor on the Mongoose instance and pass it the name of the collection and a reference to the schema definition.

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

Let's combine the code above into `./src/models/email.js` to define the contents of a basic email model:

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

Mongoose Schema vs. Model

3. Exporting a Model

A schema definition should be simple, but its complexity is usually based on application requirements. Schemas can be reused and they can contain several child-schemas too. In the example above, the value of the email property is a simple value type. However, it can also be an object type with additional properties on it.

We can also create an instance of the model we've defined above and populate it using the following syntax:

```
let EmailModel = require('./email')
let msg = new EmailModel({
```

```
  email: 'ada.lovelace@gmail.com'
})
```

Let's enhance the email schema to make the email property a unique and required field. We can then convert the value to lowercase before saving it. We can also add a validation function that will ensure that the value is a valid email address. We will reference and use the validator library installed earlier.

```
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)
```

‹ Previous next ›