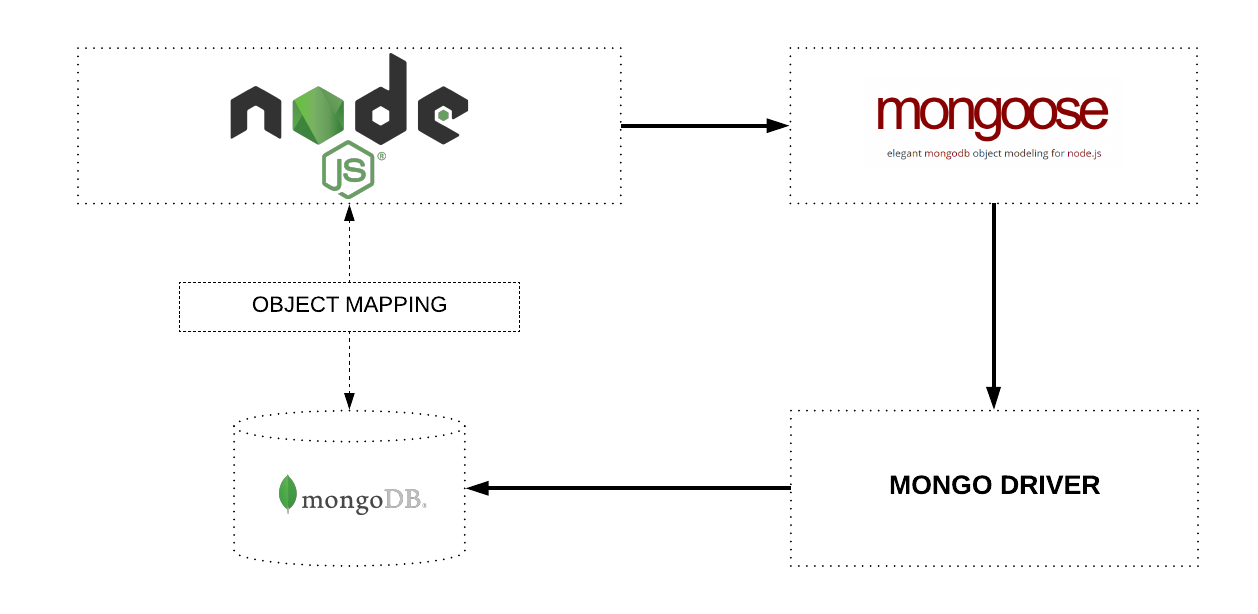
**MONGOOSE**

* Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js.
* It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.



* **Terminologies**
  + **Collections**

‘Collections’ in Mongo are equivalent to tables in relational databases. They can hold multiple JSON documents.

* + **Documents**

‘Documents’ are equivalent to records or rows of data in SQL. While a SQL row can reference data in other tables, Mongo documents usually combine that in a document.

* + **Fields**

‘Fields’ or attributes are similar to columns in a SQL table.

* + **Schema**

While Mongo is schema-less, SQL defines a schema via the table definition. A Mongoose ‘schema’ is a document data structure (or shape of the document) that is enforced via the application layer.

* + **Models**

‘Models’ are higher-order constructors that take a schema and create an instance of a document equivalent to records in a relational database.

**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.

**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.

**Mixed and ObjectId are defined under require(‘mongoose’).Schema.Types.**

**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 above code into ./src/models/email.jsto 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)

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.

Let’s enhance the Email schema to make the email property a unique, required field and 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)