**CS 624 Full-Stack Development I**

**HOS03 Building Data Model**

Developed by Amrutha

12/29/2020 Updated by Min

09/29/2021 Revised by Kim

Center for Information Assurance (CIAE) @City University of Seattle (CityU)

**Before You Start**

* The directory path shown in screenshots may be different from yours.
* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

· Connecting Express/Node applications to MongoDB using Mongoose

· Defining schemas for a data model using Mongoose

· Connecting an application to a database

· Managing databases using the MongoDB shell

· Pushing a database into a live environment

**Preparation**

1. Create a directory named "module03", copy the “module02b” directory to the “module03” one.

1. Open the “module03” folder in the VSCode.  
    Close the “Welcome” window.  
    Open the terminal (Ctrl+`).

1. Test whether the application works correctly.  
    >>>npm start

If your code did not work correctly in the previous module, clone the updated source code before continuing:

>>> git clone -b chapter-04 <https://github.com/cliveharber/gettingMean-2.git>

>> cd getting-MEAN-2

>>> npm install

**Building a Data Model with MongoDB and Mongoose**

We will be looking at the three main steps in this chapter

* Connecting the application to a database
* Define schemas and models
* Add test data to database

The diagram below shows how we are going to connect the database with the application.

A picture containing screenshot

Description automatically generated

MongoDB talks only to Mongoose, and Mongoose in turn talks to Node and Express. Angular won’t talk directly to MongoDB or Mongoose—only to the Express application.

The advantages of Mongoose are

* It enables you to define data structures and models, maintain them, and use them to interact with your database, all from the comfort of your application code.
* It includes the ability to add validation to your data definitions, meaning that you don’t have to write validation code in every place in your application where you send data back to the database.

**Installing MongoDB on macOS**

There are 2 ways to install MongoDB

* Using Homebrew
* Manually downloading

Do the installation in your mac terminal. You can choose any one of the above ways.

**Using Homebrew**

Homebrew installation:

/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"

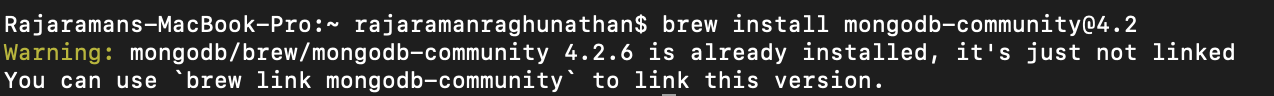
A screenshot of text

Description automatically generated

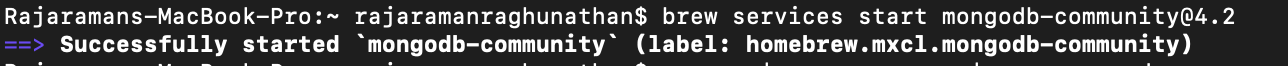
Note: wait for few minutes for the brew to update

A picture containing drawing

Description automatically generated



**Starting the MongoDB:**

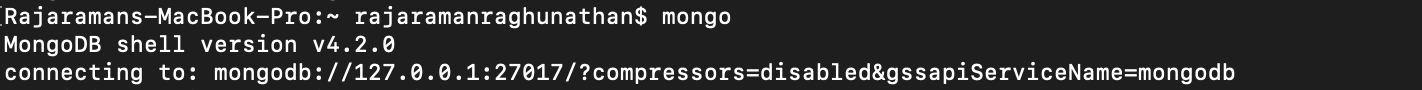


**Verifying if the MongoDB is working:**

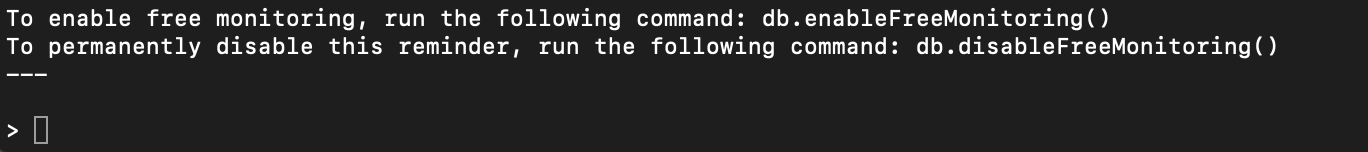


Note: Your output might be different from the above screenshot, but you should not get 0. If you get 0, try all the above steps once again.

**Use MongoDB shell:**



There will be information displayed. At the last the shell will be visible



**Close the MongoDB shell:**

**>>>** Press control+c. “Bye” message will be displayed, and the shell will be terminated.

**Loc**

Reference: Official document for installing MongoDB using Homebrew

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>

**Manually Downloading**

Follow the steps given in the below official documentation link

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x-tarball/>

**Installing MongoDB on Windows**

Follow the steps given in the below official documentation link

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>

**Installing MongoDB on Linux**

Follow the steps given in the below official documentation link

<https://docs.mongodb.com/manual/administration/install-on-linux/>

**Adding Mongoose to the application**

Mongoose is available as a npm module.

1. In your VS code terminal type the following command. When running this command, you should be in your Module3 path. (Don’t navigate to any folder within Module3)



The dependencies are added in the package.json file.

**Adding a Mongoose Connection to Your Application**

In this section you’ll connect the application to a database. MongoDB creates a database when you first try to use it

1. Go to app\_server  models and create a file “db.js”. Type the following code inside db.js.

A black and silver text on a screen

Description automatically generated

We have declared mongoose and created a connection for the application with the database. The connect() tells Mongoose to use its new internal URL parser.

Mongoose publishes events based on the status of the connection, and these events are easy to hook into so that you can see what's going on. You’ll use events to see when the connection is made, when there's an error, and when the connection is disconnected. When any one of these events occurs, you’ll log a message to the console.

1. Go to app.js and add the line highlighted in red box

A screen shot of a social media post

Description automatically generated

This line creates a connection between the application and the database, it goes to the app\_server  Modules  db.js where we have specified mongoose.

1. In the terminal check if the application is running, if not type “nodemon” to restart the application. In your browser type localhost:3000

The application should run without any error and you should be able to see the Loc8r.

In the terminal you should be able to find this line. In the place of Loc8r you should see Module6



**Model the data**

According to the MVC architecture the data must belong in the model. But right now, the data is available in the controller. Previously for the static website it was available in the views then it was moved to the controller. Now it has to be moved to the model.

A screenshot of a cell phone

Description automatically generated

**Important concept for MongoDB and Mongoose**

Mongoose was built specifically as a MongoDB Object Document Modeler (ODM) for Node applications. One key principle is that you can manage your data model from within your application. You don’t have to mess around directly with databases or external frameworks or relational mappers; you can define your data model in the comfort of your application.

First, we’ll get some naming conventions out of the way:

* In MongoDB, each entry in a database is called a document.
* In MongoDB, a group of documents is called a collection. (Think table if you’re used to relational databases.)
* In Mongoose, the definition of a document is called a schema.
* Each individual data entity defined in a schema is called a path.

A model is the compiled version of a schema. All data interactions using Mongoose go through the model.

Example of how MongoDB and Mongoose will look like:

MongoDB

A picture containing knife

Description automatically generated

Corresponding Mongoose schema

A picture containing knife

Description automatically generated

Now let’s look how to define a schema

1. Go to app\_server  models and create a new file called “locations.js” and copy paste the following code.

const mongoose = require('mongoose');

const openingTimesSchema = new mongoose.Schema({

  days: {

    type: String,

    required: true

  },

  opening: String,

  closing: String,

  closed: {

    type: Boolean,

    required: true

  }

});

const reviewSchema = new mongoose.Schema({

  author: String,

  rating: {

    type: Number,

    required: true,

    min: 0,

    max: 5

  },

  reviewText: String,

  createdOn: {

    type: Date,

    'default': Date.now

  }

});

const locationSchema = new mongoose.Schema({

  name: {

    type: String,

    required: true

  },

  address: String,

  rating: {

    type: Number,

    'default': 0,

    min: 1,

    max: 5

  },

  facilities: [String],

  coords: {

    type: { type: String},

    index: [Number]

  },

  openingTimes: [openingTimesSchema],

  reviews: [reviewSchema]

});

locationSchema.index({coords: '2dsphere'});

mongoose.model('Location', locationSchema);

The mongoose schema has been defined and the data structures have been defined with respect to locations.js under controller. The information from locations.js under the controller will be passed on the schema.

The 2dsphere here is critical part because it enables MongoDB to do the correct calculations when running queries and returning results. It allows MongoDB to calculate geometries based on a spherical object.

To build a model of the location schema, we are adding the locationSchema definition (last line)

1. In app\_server  models  db.js, add below line at the end of db.js

A screenshot of a cell phone

Description automatically generated

1. Now let’s feed the controllers with the information that has to be displayed. Go to app\_server  controllers  locations.js and copy, paste the following code.

const homelist = (req, res) => {

    res.render('locations-list', {

                pageHeader: {title: 'Loc8r', strapLine: 'Find places to work with wifi near you!'},

                sidebar: "Looking for wifi and a seat?",

                locations: [{

                  name: 'Starcups',

                  address: '125 Ward Street, Reading, RG6 1PS',

                  rating: 3,

                  facilities: ['Hot drinks', 'Food', 'Premium wifi'],

                  distance: '100m'

                },{

                  name: 'Cafe Hero',

                  address: '125 High Street, Reading, RG6 1PS',

                  rating: 4,

                  facilities: ['Hot drinks', 'Food', 'Premium wifi'],

                  distance: '200m'

                },{

                  name: 'Burger Queen',

                  address: '125 High Street, Reading, RG6 1PS',

                  rating: 2,

                  facilities: ['Food', 'Premium wifi'],

                  distance: '250m'

                }]  });

  };

  /\* GET 'Location info' page \*/

  const locationInfo = (req, res) => {

    res.render('location-info',

    {

    pageHeader: {

     title: 'Starcups',

   },

   sidebar: {

     context: 'Starcups',

     callToAction: 'please leave a review.'

   },

   location: {

     name: 'Starcups',

     address: '125 High Street, Reading, RG6 1PS',

     rating: 3,

     facilities: ['Hot drinks', 'Food', 'Premium wifi'],

     coords: {lat: 51.455041, lng: -0.9690884},

     openingTimes: [

       {

         days: 'Monday - Friday',

         opening: '7:00am',

         closing: '7:00pm',

         closed: false

       },

       {

         days: 'Saturday',

         opening: '8:00am',

         closing: '5:00pm',

         closed: false

       },

       {

         days: 'Sunday',

         closed: true

       }

     ],

     reviews: [

       {

         author: 'Simon Holmes',

         rating: 5,

         timestamp: ' 16 July 2013',

         reviewText: 'What a great place. I can\'t say enough good things about it.'

       },

       {

         author: 'Charlie Chaplin',

         rating: 3,

         timestamp: ' 16 June 2013',

         reviewText: 'It was okay. Coffee wasn\'t great, but the wifi was fast'

       }

     ]

   }

 }

);

};

  /\* GET 'Add review' page \*/

  const addReview = (req, res) => {

    res.render('location-review-form', { title: 'Review Starcups on Loc8r' ,

    pageHeader: { title: 'Review Starcups' }

  }

);

  };

  module.exports = {

    homelist,

    locationInfo,

    addReview

  };

1. Go to app\_server  controllers  others.js. In this page we will add data to the about page.

A screen shot of a social media post

Description automatically generated

The view has to consume data from the controller. In the static website we passed the information directly to the view, now it has to be changed.

1. Go to app\_server  views generic-text.pug and make the following changes. The existing code has a “p” (paragraph) which displays the information directly.

A close up of a screen

Description automatically generated

1. In the home page and location page you can see the start rating. As the same rating is used in both pages it has to be given commonly.

In the app\_server  views, create a new folder called as “\_includes”. Inside this folder create a file “sharedHTMLfunctions.pug”. Add the following code to that file.

A close up of a screen

Description automatically generated

A mixin in Pug is a function where we have to define the name of the mixin, and then nest the content of it with indentation. In the above code the outputRating is the mixin expecting the parameter rating and uses it inside for loops.

1. Go to app\_server  views  location\_info.pug and copy, paste the below code

extends layout

include \_includes/sharedHTMLfunctions

block content

  .row.banner

    .col-12

      h1= pageHeader.title

  .row

    .col-12.col-lg-9

      .row

        .col-12.col-md-6

          p.rating

            +outputRating(location.rating)

          p= location.address

          .card.card-primary

            .card-block

              h2.card-title Opening hours

              each time in location.openingTimes

                p.card-text

                  | #{time.days} :

                  if time.closed

                    | closed

                  else

                    | #{time.opening} - #{time.closing}

          .card.card-primary

            .card-block

              h2.card-title Facilities

              each facility in location.facilitiea

                  | &nbsp;#{facility}

                | &nbsp;

        .col-12.col-md-6.location-map

          .card.card-primary

            .card-block

              h2.card-title Location map

              img.img-fluid.rounded(src=`http://maps.googleapis.com/maps/api/staticmap?center=${location.coords.lat},${location.coords.lng}&zoom=17&size=400x350&sensor=false&markers=${location.coords.lat},${location.coords.lng}&scale=2&key=AIzaSyAgoFuWKCYWX1UQ51OrspggXncMw5Pea9A`)

      .row

        .col-12

          .card.card-primary.review-card

            .card-block

              a.btn.btn-primary.float-right(href='/location/review/new') Add review

              h2.card-title Customer reviews

              each review in location.reviews

                .row.review

                  .col-12.no-gutters.review-header

                    span.rating

                      +outputRating(review.rating)

                    span.reviewAuthor #{review.author}

                    small.reviewTimestamp #{review.timestamp}

                  .col-12

                    p !{(review.reviewText).replace(/\n/g, '<br/>')}

    .col-12.col-lg-3

      p.lead #{location.name} #{sidebar.context}

      p= sidebar.callToAction

In the line starting with “img.img-fluid.rounded” you have to add the Map API if you have. It has to be added after “&scale=2”.

1. Go to app\_server  views  locations-list.pug and copy, paste the below code.

extends layout

include \_includes/sharedHTMLfunctions

block content

  .row.banner

    .col-12

      h1= pageHeader.title

        small &nbsp;#{pageHeader.strapLine}

  .row

    .col-12.col-md-8

      each location in locations

        .card

          .card-block

            h4

              a(href='/location')= location.name

              +outputRating(location.rating)

              span.badge.badge-pill.badge-default.float-right= location.distance

            p.address= location.address

            .facilities

              each facility in location.facilities

                span.badge.badge-warning= facility

    .col-12.col-md-4

      p.lead= sidebar

1. In the terminal check if the application is running, if not type “nodemon” to restart the application. In your browser type localhost:3000

You should be able to see the loc8r application and click on starcups and look at the information page. Click on About to see the updated about page.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A picture containing ball, player, drawing

Description automatically generated

### Using the MongoDB Shell to Create a MongoDB Database and Add Data

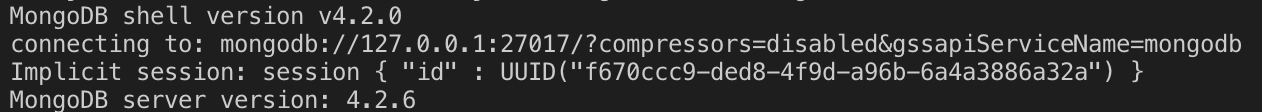
The MongoDB shell is a command-line utility that gets installed with MongoDB and allows you to interact with any MongoDB databases on your system.

### In the VS code terminal if the application is running, press control + c to stop the application.

### Type the command to start the mongo shell

### >>> mongo

Your screen should have response like this



After this the shell will be displayed where the database commands can be entered.

1. In the Mongo shell type **show dbs** to displaythe lists oflocal MongoDB databases

A close up of a sign

Description automatically generated

You should be able to see Loc8r database, even if you don’t see it no problem. Proceed with the next step.

1. Type **use Loc8r** , there are 2 reasons for using this command. The first one is, it will create a database with the name specified if it is not available and connects the shell to the database.

The second reason, if the database is already present, it connects the shell to the database.

A close up of a sign

Description automatically generated

1. Type **show collections ,** if the Loc8r database was displayed during the step 1 then you might see the output as locations. Even if the output is not displayed you don’t have to worry. There won’t be any contents inside the locations. This depicts the Model name that was used in the app\_server  models  locations.js (last line).

A close up of a sign

Description automatically generated

In the locations.js under models, the name of the model that was used was Location. The default collection name is a lowercase pluralized version of the model name. That’s the reason why you can find the collection name as locations.

1. Now we are going to pass data objects and save it under locations. In the shell type the following. After entering the first line press enter to go to the second line. In this way the command will be continued.

A screenshot of a cell phone

Description automatically generated

1. Type **db.locations.find().pretty()** in the shell to display the contents available in the locations. The pretty is for making the display readable.

A close up of text on a black background

Description automatically generated

You can find “\_id” which is automatically added by MongoDB for unique identification for this document.

Important note: Copy and save the \_id mentioned, it will be used in the next module.

1. MongoDB has an update command that accepts two arguments: a query so that it knows which document to update, and the instructions on what to do when it finds the document. Type the following in the shell.

A screenshot of a cell phone

Description automatically generated

We have specified the \_id property and assign it the value of ObjectId() as MongoDB doesn’t automatically add \_id to subdocuments as it does for documents.

After pressing enter (after the last line) you will see the following line in your shell.

A screenshot of a cell phone

Description automatically generated

This shows that the locations have been modified with the data we have entered in the above step.

**Push your work to GitHub**

Run the following commands to push your work to the GitHub repository:

Open the terminal from the VSCode by hitting the control + ~ key and type the following command:

>>> git add .

>>> git commit -m “Submission for Module 3 --yourname”

>>> git push origin master

**Push your work to Heroku**

>>> git push heroku master

If you cannot remember your branch name, run the command “git status” to check.