Setting up views for express app.

The steps towards a full stack from here are:

1. Complete the express application to query the database from a browser page.
2. Create a react application which consumes data from an api
3. Modify the express application a term to output the result in json
4. Adapt a react app to handle this

Completing the Express - Mongo application

Ensure that the express_app3 containers are stopped. With a terminal pointed to this folder:

docker-container -f stack.yml stop

```
Stopping myapp                          ...
Stopping mongo                          ...
Stopping express_app3_mongo-express_1 ... done
```

Copy the folder express_app3 to express_app4 to continue. Open a terminal shell and navigate to the new folder.

Read the next section, (don't modify code until instructed),

This section is working through the Mozilla tutorial section 5 Displaying Library data

In order to work through the tutorial code in the context of docker it is important to consider some points of detail which are required for a successful outcome.

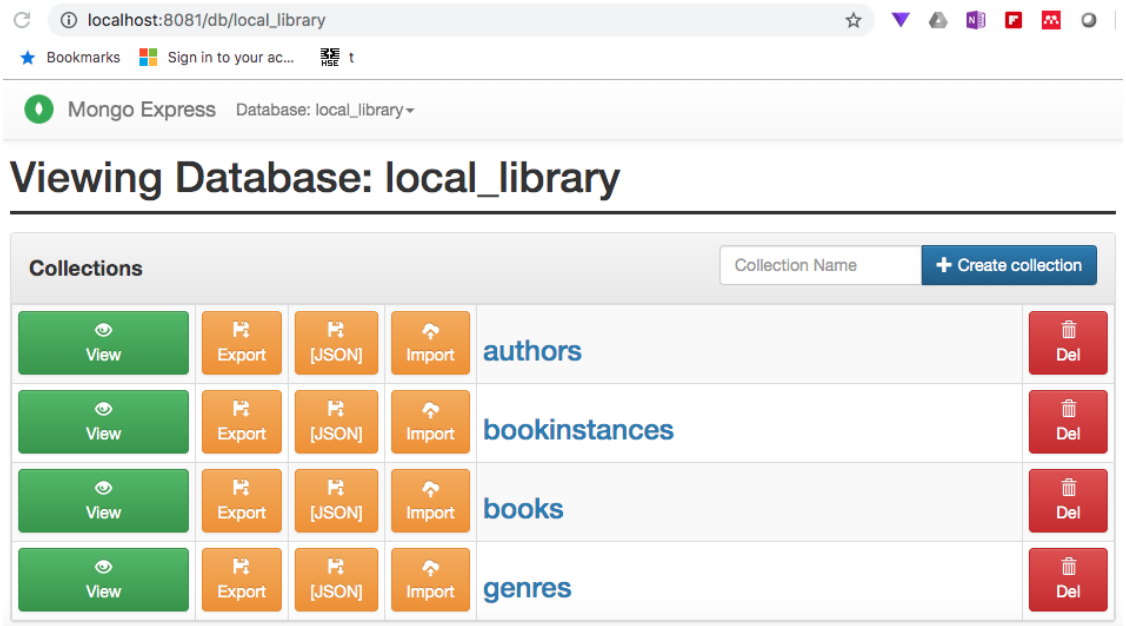
Mongoose Pluralizes

Mongoose pluralises collections. So alter mongo-init js so that collectons are authors rather than author and so on for each field. It is possible to override this pluralisation, but in practice remembering to do so on every occasion is an unnecessary burden. In the event of a mismatch in naming between collections such as author and authors the result of a query to the database would be the return of an empty dataset, no error would be raised, so the problem may be difficult to spot.

This will mean that the initialisation file must be modified to refer to plural collections.

```
db.books.drop(),
db.authors.drop(),
db.bookinstances.drop(),
db.genres.drop(),
```

This will be reflected in the collections viewed by mongo express.



plural collections

Docker needs authority

To operate on a database an application needs both a valid connection string and to be an authorised user.

The standard form of the connection string is:

```
mongodb://[username:password@]host1[:port1][,...hostN[:portN]][/[database][?options]]
```

When containers are spun up by a single docker-compose file docker creates a network connection between them. The name of the container which is runing the database service is the address to be added to the connection string rather than localhost, 127.0.0.1 or an external url.

The connection string must include the name of the database being used. When the mongo express administrative tool is used to inspect a 'database' it is found that there are actually several databases including admin, config and local. One named admin requires an admin user with a valid password to access it. The admin database contains details of the system users. The admin user must be set up to match the initial application accessing it. The mongo express application is provided with this through environment variables set in the Docker-Compose file.

```
MONGO_INITDB_ROOT_USERNAME: admin
MONGO_INITDB_ROOT_PASSWORD: password
```

Some programmers set up the values of these environment variables in a seperate file .env and then call them in as constants, but that has not been done here.

When the compose file is brought up the volume

```
    - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
```

maps the initialising javascript file to an entrypoint and runs this to populate the database. The name of the database which becomes populated is also passed through an environmental variable.

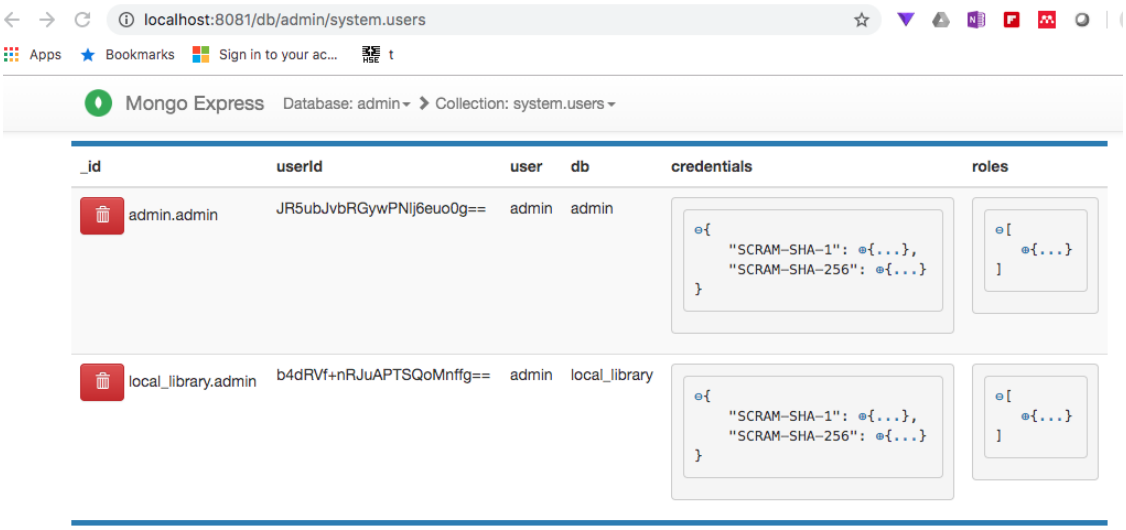
```
      MONGO_INITDB_DATABASE: local_library
```

Now the express app which is being written to interact with the database also needs to be a user listed in the system users database. The app needs to be a user with read write access to the database containing application data, in this case named local_library.

A user can be added to the system users database by addin code to create a user at the top of the mongo-init.js file.

```
db.createUser(
  {
      user: "admin",
      pwd: "password",
      roles:[
          {
              role: "readWrite",
              db:   "local_library"
          }
      ]
  }
);
```

Inspecting the system users collection shows that the user is created below the admin user.



system users

The connection string to access this is then formed in the app.js file.

```
const mongoDB = 'mongodb://mongo:27017/local_library';
```

Here the url 'mongo' is the name of the container which the database runs in whch is established in the docker-compose file.

```
services:
  mongo:
    image: mongo
    container_name: mongo
```

When the connection string is used to make a connection the auth object must match the user and password of the local_library user.

```
mongoose.connect(mongoDB, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  auth: {
  user: "admin",
  password: "password"
  }
});
```

The application can now successfully interact with the local_library database.

Some references refer to a flag –Auth used in the docker command line to invoke authorisation. There is no need to use this in the context of the docker-compose file, authorisation is in place automatically.

Normalised or Denormalised database

6 Rules of Thumb for MongoDB Schema Design discusses the trade off between *normalizing* data, so that every table is complete with embedded data from one to many relationships and *denormalizing*, so that the collection for the many contains references to the one.

Denormalizing only becomes possible in a NoSQL database, relational databases are normalized and references are made between tables.

Denormalizing means that each collection is full and complete, it has an advantage on read that the full information is read without the need to join in data from other referenced collections. This means that the data may now be held in two or more collections, which is a built in redundancy. To keep all the embedded data correct this will have a cost when creating or updating a new element.

Normalising means that in a one to many relationship, the doument for the many will include references to the one. This will have a cost on reading to populate the result from two or more collections, however the writing will be easier as the items in the one collection are isolated.

Advice is generally to prefer normalized collections. Denormalised collections are preferred in the limitted case where the scale of the one to many connection is one to few and reading is much more common than updating.

Just because you can embed does not mean you should.

Mongoose provides a convenient command *populate()* to handle normalised referenced collections so the balance of benefit moves towards normalisation.

In the original version of mongo-init.js a denormalised database was set up. The data for authors and genres were saved as arrays and then inserted into the book data.

```
bookCreate('Test Book 1', 'Summary of test book 1', 'ISBN111111', authors[4],
[genres[0],genres[1]]);
```

Provided the authors and genres arrays were pouplated before the bookCreate() this worked fine.

The Local-Library tutorial which is being dockerized is based on a normalised database, so mongo-init.js must be ajusted to match this format.

For a normalised database a reference to the .id of the authors and genres must be inserted into bookCreate. That calls for a change in flow!. The database entries for authors and genres must be made first. Then the ids are read back from the database, stored in an array and added to the bookCreate.

To write to the database after creating values we must printjson(res) immediately.

```
authorCreate('Patrick', 'Rothfuss', new Date('1973-06-06'), false);
authorCreate('Ben', 'Bova', new Date('1932-11-8'), false);
authorCreate('Isaac', 'Asimov', new Date('1920-01-02'), new Date('1992-04-
```

```
06'));
authorCreate('Bob', 'Billings', false, false);
authorCreate('Jim', 'Jones', new Date('1971-12-16'), false);

genreCreate("Fantasy");
genreCreate("Science Fiction");
genreCreate("French Poetry");

printjson(res);
```

Now before writing any more res needs to be reset to a null array so that we don't send duplicate information on the next printjson(res).

```
res = [];
```

The authors and genres databases must now be read back and an array made of the results. Note that we are using the MongoDB method [find].(https://docs.mongodb.com/manual/reference/method/db.collection.find/#find-projection) because mongoose is not invoked at this point.

```
var myAuthorCursor = db.authors.find();
var authorsArray = myAuthorCursor.toArray();


var myGenreCursor = db.genres.find();
var genresArray = myGenreCursor.toArray();
```

Now the entry for Test Book 1 relates to the object ID of the authors and genres.

```
bookCreate('Test Book 1', 'Summary of test book 1', 'ISBN111111',
authorsArray[4]._id, [genresArray[0]._id,genresArray[1]._id]);
```

Another printjson(res) is needed. finally the process is repeated to get the book object IDs for bookInstances.

Now update the full listing of **mongo-init.js** accordingly to:

```
db.createUser(
  {
      user: "admin",
      pwd: "password",
      roles:[
          {
              role: "readWrite",
              db:   "local_library"
          }
      ]
  }
);

let error = false
```

```
let genres = []
let authors = []
let books = []
let bookinstances = []


function authorCreate(first_name, family_name, date_of_birth, date_of_death)
{
    authordetail = {first_name:first_name , family_name: family_name
,date_of_birth: null, date_of_death: null}
    if (date_of_birth != false) authordetail.date_of_birth = date_of_birth;
    if (date_of_death != false) authordetail.date_of_death = date_of_death;
    authors.push(authordetail)
    res.push(db.authors.insert(authordetail))
}

function genreCreate(name) {
  genredetail = {name: name};
  genres.push(genredetail)
  res.push(db.genres.insert(genredetail))
}


function bookCreate(title, summary, isbn, author, genre) {
    bookdetail = {
      title: title,
      summary: summary,
      author: author,
      isbn: isbn,
      genre: null
    }
    if (genre != false) bookdetail.genre = genre;
    books.push(bookdetail);
    res.push(db.books.insert(bookdetail))
}

function bookInstanceCreate(book, imprint, due_back, status) {
  bookinstancedetail = {
    book: book,
    imprint: imprint,
    due_back: null,
    status: null
  }
  if (due_back != false) bookinstancedetail.due_back = due_back
  if (status != false) bookinstancedetail.status = status
    bookinstances.push(bookinstancedetail);
    res.push(db.bookinstances.insert(bookinstancedetail))
}
```

```
let res = [
  db.books.drop(),
  db.authors.drop(),
  db.bookinstances.drop(),
  db.genres.drop(),

  db.books.createIndex({ title: 1 },{ unique: true }),
  db.books.createIndex({ summary: 1 }),
  db.books.createIndex({ author: 1 }),
  db.books.createIndex({ isbn: 1 }),
  db.books.createIndex({ genre: 1 }),

  db.authors.createIndex({first_name:1},{name:"first_name"}),
  db.authors.createIndex({family_name:1},{name:"family_name"}),
  db.authors.createIndex({date_of_birth:1},{name:"date_of_birth"}),
  db.authors.createIndex({date_of_death:1},{name:"date_of_death"}),

  db.bookinstances.createIndex({book: 1}),
  db.bookinstances.createIndex({imprint: 1}),
  db.bookinstances.createIndex({due_back: 1}),
  db.bookinstances.createIndex({status: 1}),

  db.genres.createIndex({name: 1}),
]

authorCreate('Patrick', 'Rothfuss', new Date('1973-06-06'), false);
authorCreate('Ben', 'Bova', new Date('1932-11-8'), false);
authorCreate('Isaac', 'Asimov', new Date('1920-01-02'), new Date('1992-04-
06'));
authorCreate('Bob', 'Billings', false, false);
authorCreate('Jim', 'Jones', new Date('1971-12-16'), false);

genreCreate("Fantasy");
genreCreate("Science Fiction");
genreCreate("French Poetry");

printjson(res);
res = [];

var myAuthorCursor = db.authors.find();
var authorsArray = myAuthorCursor.toArray();


var myGenreCursor = db.genres.find();
var genresArray = myGenreCursor.toArray();
```

```
bookCreate('The Name of the Wind (The Kingkiller Chronicle, #1)', 'I have
stolen princesses back from sleeping barrow kings. I burned down the town of
Trebon. I have spent the night with Felurian and left with both my sanity and
my life. I was expelled from the University at a younger age than most people
are allowed in. I tread paths by moonlight that others fear to speak of
during day. I have talked to Gods, loved women, and written songs that make
the minstrels weep.', '9781473211896', authorsArray[0]._id,
[genresArray[0]._id,]);
bookCreate("The Wise Man's Fear (The Kingkiller Chronicle, #2)", 'Picking up
the tale of Kvothe Kingkiller once again, we follow him into exile, into
political intrigue, courtship, adventure, love and magic... and further along
the path that has turned Kvothe, the mightiest magician of his age, a legend
in his own time, into Kote, the unassuming pub landlord.', '9788401352836',
authorsArray[0]._id, [genresArray[0]._id,]);
bookCreate("The Slow Regard of Silent Things (Kingkiller Chronicle)", 'Deep
below the University, there is a dark place. Few people know of it: a broken
web of ancient passageways and abandoned rooms. A young woman lives there,
tucked among the sprawling tunnels of the Underthing, snug in the heart of
this forgotten place.', '9780756411336', authorsArray[0]._id,
[genresArray[0]._id,]);
bookCreate("Apes and Angels", "Humankind headed out to the stars not for
conquest, nor exploration, nor even for curiosity. Humans went to the stars
in a desperate crusade to save intelligent life wherever they found it. A
wave of death is spreading through the Milky Way galaxy, an expanding sphere
of lethal gamma ...", '9780765379528', authorsArray[1]._id,
[genresArray[1]._id,]);
bookCreate("Death Wave","In Ben Bova's previous novel New Earth, Jordan Kell
led the first human mission beyond the solar system. They discovered the
ruins of an ancient alien civilization. But one alien AI survived, and it
revealed to Jordan Kell that an explosion in the black hole at the heart of
the Milky Way galaxy has created a wave of deadly radiation, expanding out
from the core toward Earth. Unless the human race acts to save itself, all
life on Earth will be wiped out...", '9780765379504', authorsArray[1]._id,
[genresArray[1]._id,]);
bookCreate('Test Book 1', 'Summary of test book 1', 'ISBN111111',
authorsArray[4]._id, [genresArray[0]._id,genresArray[1]._id]);
bookCreate('Test Book 2', 'Summary of test book 2', 'ISBN222222',
authorsArray[4]._id, false)

printjson(res);
res = [];

var myBookCursor = db.books.find();
var booksArray = myBookCursor.toArray();

bookInstanceCreate(booksArray[0]._id, 'London Gollancz, 2014.', false,
'Available');
bookInstanceCreate(booksArray[1]._id, ' Gollancz, 2011.', '2020-06-06',
```

```
'Loaned');
bookInstanceCreate(booksArray[2]._id, ' Gollancz, 2015.', false, false);
bookInstanceCreate(booksArray[3]._id, 'New York Tom Doherty Associates,
2016.', false, 'Available');
bookInstanceCreate(booksArray[3]._id, 'New York Tom Doherty Associates,
2016.', false, 'Available');
bookInstanceCreate(booksArray[3]._id, 'New York Tom Doherty Associates,
2016.', false, 'Available');
bookInstanceCreate(booksArray[4]._id, 'New York, NY Tom Doherty Associates,
LLC, 2015.', false, 'Available');
bookInstanceCreate(booksArray[4]._id, 'New York, NY Tom Doherty Associates,
LLC, 2015.', false, 'Maintenance');
bookInstanceCreate(booksArray[4]._id, 'New York, NY Tom Doherty Associates,
LLC, 2015.', false, 'Loaned');
bookInstanceCreate(booksArray[0]._id, 'Imprint XXX2', false, false);
bookInstanceCreate(booksArray[1]._id, 'Imprint XXX3', false, false);

printjson(res)

if (error) {
  print('Error, exiting')
  quit(1)
}
```

Asynchronous flow

Information will be requested asynchonously.

If the information can be gained from a single request the format of the query is relatively straightforward.

Specify an operation to perform and a callback function which will handle the results. The callback function will have two parameters, the first is an error value, which is null if there is no error and the second is the results from the operation.

Here, count is a Mongoose Method.

```
exports.some_model_count = function(req, res, next) {

  SomeModel.count({ a_model_field:'match_value',
  function (err, result) {// callback
    if (err){console.log('error ', err)
    }else{
      res.render('the_template', { data: result });
    }
  });
}
```

However if multiple asynchronous queries are made the result can't be displayed till all callbacks are complete. to handle this use the npn *async* module.

The example below shows parallel callbacks which are passed as and object to async.parallel and the results come back in an object with the property names preserved.

```
async.parallel({
  one: function(callback) { ... },
  two: function(callback) { ... },
  //...
  something: function(callback) { ... }
  },
  // optional callback
  function(err, results) {
    // 'results' is now equal to:
    {one: 1, two: 2, ..., something: some_value}
  }
);
```

Async also allows for queries in series or waterfall ordering. The current version of async is 3.1.0 and this is compatible.

Add async to package.json

```
{
  "name": "myapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "nodemon -L start.js"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1",
    "pug": "2.0.0-beta11",
    "nodemon": "1.19.1",
    "mongoose":"^5.7.8",
    "body-parser":"^1.19.0",
    "async":"3.1.0"
  },
  "nodemonConfig": {
    "delay": "1500",
    "verbose": "true"
  }
}
```

Note that async is not required in app.js. It is only needed in files which use it and the first of these is bookController.js

```
var Book = require('../models/book');
var Author = require('../models/author');
```

```
var Genre = require('../models/genre');
var BookInstance = require('../models/bookinstance');

var async = require('async');
```

The details of this file are discussed in a later section.

Views vs API

At the moment we are rendering data to a browser not passing it back as json to react so we use views, the files for which are saved in the views folder.

It will be a later stage to edit the application to send json back to react as a rest API should.

We will use pug to render the view according to a template. The template syntax is described here.

To get a consistant look to a site a base template can be defined in layout.pug:

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    block content
```

The block tag is then used for specific layout in index.pug by extending the layout template:

```
extends layout

block content
  h1= title
  p Welcome to #{title}
```

In express_app4/myapp/views, layout.pug can be updated to the local_library base template.

```
doctype html
html(lang='en')
  head
    title= title
    meta(charset='utf-8')
    meta(name='viewport', content='width=device-width, initial-scale=1')
    link(rel='stylesheet',
href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css')

script(src='https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js
')

script(src='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.
```

```
js')
    link(rel='stylesheet', href='/stylesheets/style.css')
  body
    div(class='container-fluid')
      div(class='row')
        div(class='col-sm-2')
          block sidebar
            ul(class='sidebar-nav')
              li
                a(href='/catalog') Home
              li
                a(href='/catalog/books') All books
              li
                a(href='/catalog/authors') All authors
              li
                a(href='/catalog/genres') All genres
              li
                a(href='/catalog/bookinstances') All book-instances
              li
                hr
              li
                a(href='/catalog/author/create') Create new author
              li
                a(href='/catalog/genre/create') Create new genre
              li
                a(href='/catalog/book/create') Create new book
              li
                a(href='/catalog/bookinstance/create') Create new book
instance (copy)

        div(class='col-sm-10')
          block content
```

Note that this uses bootstrap and jquery from their content delivery networks. These could be replaced by links to the downloaded javascript libraries if required.

The view now includes a column with links to the application which match database enquiries.

Replace /public/stylesheets/style.css with:

```
.sidebar-nav {
  margin-top: 20px;
  padding: 0;
  list-style: none;
}
```

Creating a home page

The home page is called when the request is '/' and a route for this is defined within /routes/catalog.js with a callback parameter book_controller.index.

```
// GET catalog home page.
router.get('/', book_controller.index);
```

This callback index is defined in /controllers/bookController.js

```
exports.index = function(req, res) {
    res.send('NOT IMPLEMENTED: Site Home Page');
};
```

The controller index will fetch information about how many Book, BookInstance, Author and Genre records are held in the database. Render this as HTML and return it in an HTTP response.

The Mongoose countDocuments() method is used to cound the number of instances for each model.

Edit bookController.js, replacing exports.index with:

```
var Book = require('../models/book');
var Author = require('../models/author');
var Genre = require('../models/genre');
var BookInstance = require('../models/bookinstance');

var async = require('async');

exports.index = function(req, res) {

    async.parallel({
        book_count: function(callback) {
            Book.countDocuments({}, callback); // Pass an empty object as
match condition to find all documents of this collection
        },
        book_instance_count: function(callback) {
            BookInstance.countDocuments({}, callback);
        },
        book_instance_available_count: function(callback) {
            BookInstance.countDocuments({status:'Available'}, callback);
        },
        author_count: function(callback) {
            Author.countDocuments({}, callback);
        },
        genre_count: function(callback) {
            Genre.countDocuments({}, callback);
        }
    }, function(err, results) {
        res.render('index', { title: 'Local Library Home', error: err, data:
```

```
results });
    });
};
```

This renders the page even if there is an error, so would need a bit of tidying for a public interface.

Now replace the contents of /views/index.pug with:

```
extends layout

block content
  h1= title
  p Welcome to #[em LocalLibrary], a very basic Express website developed as
a tutorial example on the Mozilla Developer Network.

  h1 Dynamic content

  if error
    p Error getting dynamic content.
  else
    p The library has the following record counts:

    ul
      li #[strong Books:] !{data.book_count}
      li #[strong Copies:] !{data.book_instance_count}
      li #[strong Copies available:] !{data.book_instance_available_count}
      li #[strong Authors:] !{data.author_count}
      li #[strong Genres:] !{data.genre_count}
```

The view offers and error message if an error is found otherwise it displays the data.

Try this out. Rebuild the image to include async

    docker build -t dt/express-development .

Terminal output concludes:

```
...
Successfully built 323362fa031f
Successfully tagged dt/express-development:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-
Windows Docker host. All files and directories added to build context will
have '-rwxr-xr-x' permissions. It is recommended to double check and reset
permissions for sensitive files and directories.
```

Make sure that there are no volumes holding earlier database configuration and data.

    docker volume prune

Delete the old mongo container

docker container rm mongo

Delete old myapp container

docker container rm myapp

Check that no containers are currently running. Then start a container.

docker-compose -f stack.yml up

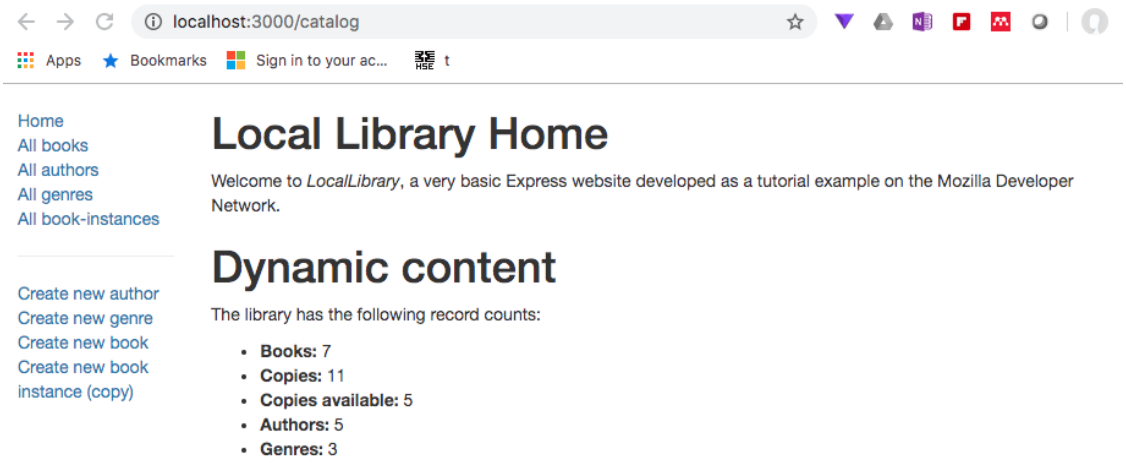
If for any reason this docker compose step does not work you will need to fix the cause (probably an old container running and using the same name for an element 'mongo') and then remove the express_app4 container with (docker-compose -f stack.yml down) before trying again.

Terminal output concludes:

```
mongo-express_1  | Welcome to mongo-express
mongo-express_1  | ------------------------
mongo-express_1  |
mongo-express_1  |
mongo-express_1  | Mongo Express server listening at http://0.0.0.0:8081
mongo            | {"t":{"$date":"2020-12-01T10:56:05.177+00:00"},"s":"I",
"c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection
accepted","attr":{"remote":"172.25.0.2:47706","connectionId":3,"connectionCou
nt":2}}
mongo-express_1  | Server is open to allow connections from anyone (0.0.0.0)
mongo-express_1  | basicAuth credentials are "admin:pass", it is recommended
you change this in your config.js!
mongo            | {"t":{"$date":"2020-12-01T10:56:05.192+00:00"},"s":"I",
"c":"NETWORK",  "id":51800,   "ctx":"conn3","msg":"client
metadata","attr":{"remote":"172.25.0.2:47706","client":"conn3","doc":{"driver
":{"name":"nodejs","version":"2.2.24"},"os":{"type":"Linux","name":"linux","a
rchitecture":"x64","version":"4.19.104-microsoft-
standard"},"platform":"Node.js v12.20.0, LE, mongodb-core: 2.1.8"}}}
mongo-express_1  | Database connected
mongo            | {"t":{"$date":"2020-12-01T10:56:05.617+00:00"},"s":"I",
"c":"ACCESS",   "id":20250,   "ctx":"conn3","msg":"Successful
authentication","attr":{"mechanism":"SCRAM-SHA-
1","principalName":"root","authenticationDatabase":"admin","client":"172.25.0
.2:47706"}}
mongo-express_1  | Admin Database connected
mongo            | {"t":{"$date":"2020-12-01T10:56:05.626+00:00"},"s":"I",
"c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection
accepted","attr":{"remote":"172.25.0.2:47708","connectionId":4,"connectionCou
nt":3}}
mongo            | {"t":{"$date":"2020-12-01T10:56:05.699+00:00"},"s":"I",
"c":"ACCESS",   "id":20250,   "ctx":"conn4","msg":"Successful
authentication","attr":{"mechanism":"SCRAM-SHA-
1","principalName":"root","authenticationDatabase":"admin","client":"172.25.0
.2:47708"}}
```

```
mongo            | {"t":{"$date":"2020-12-01T10:56:05.702+00:00"},"s":"I",
"c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection
accepted","attr":{"remote":"172.25.0.2:47710","connectionId":5,"connectionCou
nt":4}}
mongo            | {"t":{"$date":"2020-12-01T10:56:05.761+00:00"},"s":"I",
"c":"ACCESS",   "id":20250,   "ctx":"conn5","msg":"Successful
authentication","attr":{"mechanism":"SCRAM-SHA-
1","principalName":"root","authenticationDatabase":"admin","client":"172.25.0
.2:47710"}}
mongo            | {"t":{"$date":"2020-12-01T10:56:13.468+00:00"},"s":"I",
"c":"NETWORK",  "id":22943,   "ctx":"listener","msg":"Connection
accepted","attr":{"remote":"172.25.0.4:40986","connectionId":6,"connectionCou
nt":5}}
mongo            | {"t":{"$date":"2020-12-01T10:56:13.471+00:00"},"s":"I",
"c":"NETWORK",  "id":51800,   "ctx":"conn6","msg":"client
metadata","attr":{"remote":"172.25.0.4:40986","client":"conn6","doc":{"driver
":{"name":"nodejs|Mongoose","version":"3.6.3"},"os":{"type":"Linux","name":"l
inux","architecture":"x64","version":"4.19.104-microsoft-
standard"},"platform":"'Node.js v12.20.0, LE
(unified)","version":"3.6.3|5.11.0"}}}
```


http://localhost:3000/catalog

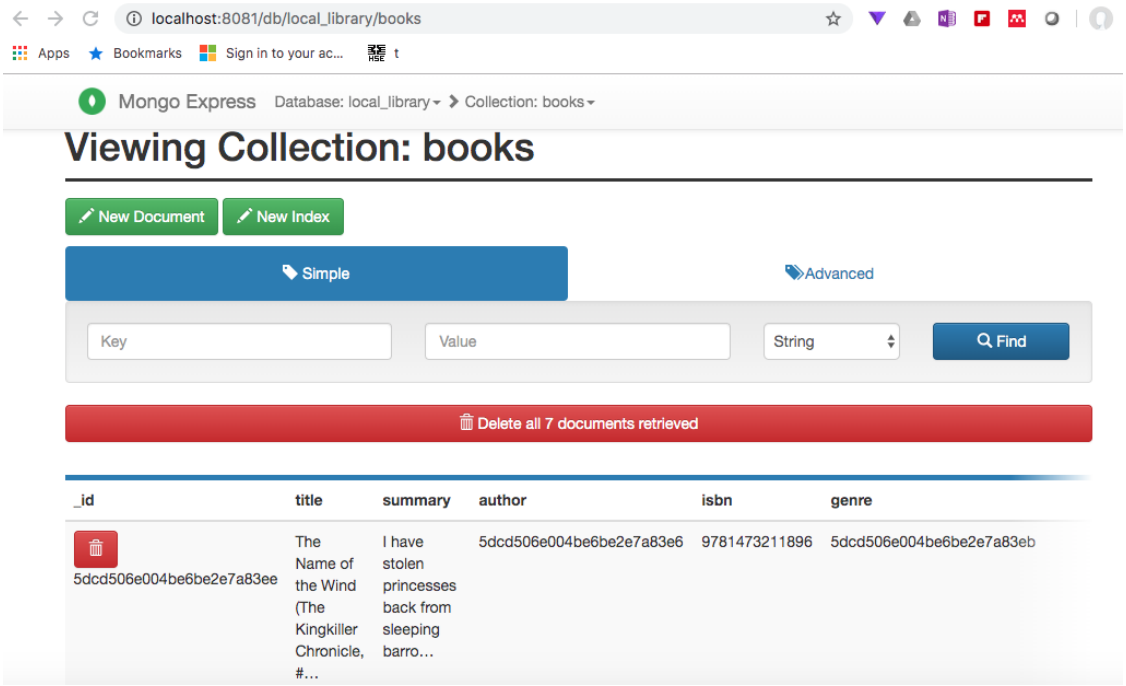


/catalog

The catalog is displayed.

Look at the book entries in the mongo express admin to see that the references to database ids are correctly inserted.

http://localhost:8081/db/local_library/books



Normalised database entries

help

If there is an error such as 'error getting dynamic content' you can still check that the database and mongo express are working by looking at

http://localhost:8081/db/local_library/books

Check the browser console for database connection errors which could be generated by a line in app.js

```
db.on('error', console.error.bind(console, 'MongoDB connection error:'));
```

Follow the data flow to look for an error. First check in routes/catalog.js

Next controller/bookController.js

Check book_controller.index

index.pug contains the line to catch errors:

```
  if error
    p Error getting dynamic content.
```

Temporarily changing the line in book_controller.index

```
 //res.render('index', { title: 'Local Library Home', error: err, data:
results });
 res.render('error', { title: 'Local Library Home', error: err, data: results
});
```

will now render the page with the view error.pug and this will display the error details.

error view

Now can aim to fix this.

Book list page

Following the tutorial for the Book list

Find a list of books and render these though a template.

In /controllers/bookController.js, replace

```
// Display list of all books.
exports.book_list = function(req, res) {
    res.send('NOT IMPLEMENTED: Book list');
};
// Display list of all books.
exports.book_list = function(req, res) {
    res.send('NOT IMPLEMENTED: Book list');
};
```

with the new exported book_list() method.

```
// Display list of all Books.
exports.book_list = function(req, res, next) {

  Book.find({}, 'title author')
    .populate('author')
    .exec(function (err, list_books) {
      if (err) { return next(err); }
      //Successful, so render
      res.render('book_list', { title: 'Book List', book_list: list_books });
    });

};
```

Create /views/book_list.pug with the content:

```
extends layout

block content
  h1= title

  ul
    - book_list.sort(function(a, b) {let textA = a.title.toUpperCase(); let
textB = b.title.toUpperCase(); return (textA < textB) ? -1 : (textA > textB)
? 1 : 0;});
    each book in book_list
      li
        a(href=book.url) #{book.title}
        |  (#{book.author.name})

    else
      li There are no books.
```

http://localhost:3000/catalog/books

Book List

Book List

This uses the models find function rather than async as there is only one question asked.

The method populate('author') uses the stored book author id to populate the result with the full author details.

Still an error here, will return to this.

Book Instances

In /controllers/bookinstanceController.js replace

```
// Display list of all BookInstances.
exports.bookinstance_list = function(req, res) {
    res.send('NOT IMPLEMENTED: BookInstance list');
};
```

With code:

```
// Display list of all BookInstances.
exports.bookinstance_list = function(req, res, next) {

  BookInstance.find()
    .populate('book')
    .exec(function (err, list_bookinstances) {
      if (err) { return next(err); }
      // Successful, so render
      res.render('bookinstance_list', { title: 'Book Instance List',
```

```
bookinstance_list: list_bookinstances });
    });

};
```

Create /views/bookinstance_list.pug with content:

```
extends layout

block content
  h1= title

  ul
    each val in bookinstance_list
      li
        a(href=val.url) #{val.book.title} : #{val.imprint} -
        if val.status=='Available'
          span.text-success #{val.status}
        else if val.status=='Maintenance'
          span.text-danger #{val.status}
        else
          span.text-warning #{val.status}
        if val.status!='Available'
          span  (Due: #{val.due_back} )

    else
      li There are no book copies in this library.
```

Date formatting

Use the npm moment module to format dates. Note current verion is 2.24.0 potentially there are changes since version 1 which may have dependency issues.

Add this module to package.json (this will require a fres build to apply changes).

```
{
  "name": "myapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "nodemon -L start.js"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1",
    "pug": "2.0.0-beta11",
    "nodemon": "1.19.1",
```

```
    "mongoose":"^5.7.8",
    "body-parser":"^1.19.0",
    "async":"1.5.2",
    "moment":"2.24.0"
  },
  "nodemonConfig": {
    "delay": "1500",
    "verbose": "true"
  }
}
```

In models/bookinstance.js require moment and add a 'due_back_formatted' virtual below the 'url' virtual

```
var mongoose = require('mongoose');
var moment = require('moment');

var Schema = mongoose.Schema;

var BookInstanceSchema = new Schema(
  {
    book: { type: Schema.Types.ObjectId, ref: 'Book', required: true },
//reference to the associated book
    imprint: {type: String, required: true},
    status: {type: String, required: true, enum: ['Available', 'Maintenance',
'Loaned', 'Reserved'], default: 'Maintenance'},
    due_back: {type: Date, default: Date.now}
  }
);

// Virtual for bookinstance's URL
BookInstanceSchema
.virtual('url')
.get(function () {
  return '/catalog/bookinstance/' + this._id;
});

BookInstanceSchema
.virtual('due_back_formatted')
.get(function () {
  return moment(this.due_back).format('MMMM Do, YYYY');
});

//Export model
module.exports = mongoose.model('BookInstance', BookInstanceSchema);
```

In /views/bookinstance_list.pug comment out and replace due_back with due_back_formatted.

```
extends layout

block content
  h1= title

  ul
    each val in bookinstance_list
      li
        a(href=val.url) #{val.book.title} : #{val.imprint} -
        if val.status=='Available'
          span.text-success #{val.status}
        else if val.status=='Maintenance'
          span.text-danger #{val.status}
        else
          span.text-warning #{val.status}
        if val.status!='Available'
          // span  (Due: #{val.due_back} )
          span  (Due: #{val.due_back_formatted} )
    else
      li There are no book copies in this library.
```

http://localhost:3000/catalog/bookinstances

Book Instance List

Book Instance List

author and genre

Still following Author list and Genre tutorial

In /controllers/authorController.js

Replace

```
// Display list of all Authors.
exports.author_list = function(req, res) {
    res.send('NOT IMPLEMENTED: Author list');
};
```

With

```
// Display list of all Authors.
exports.author_list = function(req, res, next) {

  Author.find()
    .sort([['family_name', 'ascending']])
    .exec(function (err, list_authors) {
      if (err) { return next(err); }
      //Successful, so render
```

```
      res.render('author_list', { title: 'Author List', author_list:
list_authors });
    });

};
```

The models find(), sort() and exex() functions are used to list Authors in family_name order.

Create /views/author_list.pug with content:

```
extends layout

block content
  h1= title

  ul
    each author in author_list
      li
        a(href=author.url) #{author.name}
        |  (#{author.date_of_birth} - #{author.date_of_death})

    else
      li There are no authors.
```

In controllers/genreControllers.js

replace

```
// Display list of all Genre.
exports.genre_list = function(req, res) {
    res.send('NOT IMPLEMENTED: Genre list');
};
```

With code following a similar pattern:

```
// Display list of all Genres.
exports.genre_list = function(req, res, next) {

    Genre.find()
      .sort([['name', 'ascending']])
      .exec(function (err, list_genres) {
        if (err) { return next(err); }
        //Successful, so render
        res.render('genre_list', { title: 'Genre List', genre_list:
list_genres });
      });

  };
```

Create /views/genre_list.pug with content:

```
extends layout

block content
  h1= title

  ul
    each genre in genre_list
      li
        a(href=genre.url) #{genre.name}
        |

    else
      li There are no genres
```

http://localhost:3000/catalog/authors

Author List

Author List

http://localhost:3000/catalog/genres

Genre List

Genre List

At this point the database is successfully being read.

References

6 Rules of Thumb for MongoDB Schema Design

Mozilla Express Web Framework Tutorial