

Web Information Technologies, Semester 1 2021

School of Computing and Information Systems
The University of Melbourne

Workshop 6: Heroku and MongoDB

written by Ronal Singh with contributions by
Steven Tang, Ziad Albkhetan, and Alex Wu

Objectives

1. To deploy your library app to Heroku (make it “live”)
2. To register for a MongoDB database instance on MongoDB Atlas
3. To connect the library app to the database

This workshop continues the work you did in the previous workshop. We have provided on Canvas the solution (mylibraryapp) to last week’s workshop. You can use either this solution, or the app you made last week, as the starting point for this week’s workshop.

Activity 1: Push library app to GitHub

1. In your terminal, change to your **mylibraryapp** working directory where your solution from last week is stored (or download the sample solution from Canvas).
2. Initialise a local git repository, if you haven’t already, using the command: **git init**
3. Create a **.gitignore** file at the root of your repository. Write in this file all files and folders that should not be tracked by git. This should include the **node_modules/** folder and **.env** file, and optionally editor-specific configuration files. You can copy-paste the following lines and add others depending on your work.

`node_modules/`

`.env`

`.vscode/`

gitignore tells Git which files and directories to ignore. For example, we should ignore the `node_modules` folder as others can install our dependencies using the **package.json** file.

4. Add all files to the staging area using: **git add -A**
5. Commit your files to the local repository. Use the following command (choose your own commit message):
6. **git commit -m "initial commit"**

```

/home/gregwadley/info30005/mylibraryapp/
/home/gregwadley/info30005/mylibraryapp/> git init
Initialized empty Git repository in /home/gregwadley/info30005/mylibraryapp/.git/
/home/gregwadley/info30005/mylibraryapp/> echo node_modules > .gitignore
/home/gregwadley/info30005/mylibraryapp/> ls
app.js  controllers  models  node_modules  package-lock.json  package.json  routes
/home/gregwadley/info30005/mylibraryapp/> ls -a
.  ..  .git  .gitignore  app.js  controllers  models  node_modules  package-lock.json  package.json  routes
/home/gregwadley/info30005/mylibraryapp/> cat .gitignore
node_modules
/home/gregwadley/info30005/mylibraryapp/> git add -A
/home/gregwadley/info30005/mylibraryapp/> git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   .gitignore
    new file:   app.js
    new file:   controllers/authorController.js
    new file:   models/authors.js
    new file:   package-lock.json
    new file:   package.json
    new file:   routes/authorRouter.js
/home/gregwadley/info30005/mylibraryapp/> git commit -m "first commit"

```

7. Create a remote repository on GitHub named **mylibraryapp**.
8. Link your local and remote repositories by following the instructions that appear in GitHub, or use the following commands.

(Replace username and repo_name by your account details and the name of the repository you created in (1) "mylibraryapp".)

git remote add origin

https://github.com/<username>/<repo_name>.git

push to the master branch using the command:

git push -u origin main

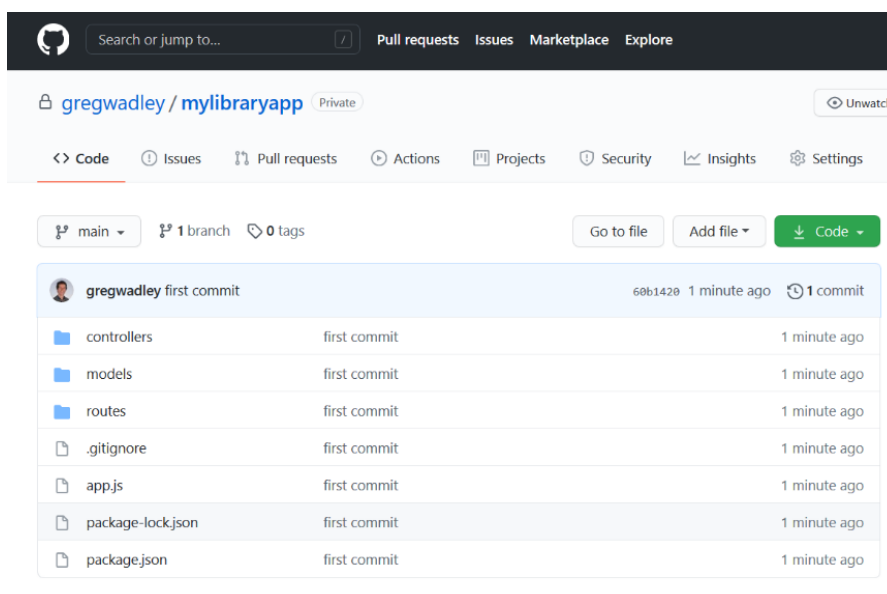


Figure 1: now your remote repo is set up

Activity 2: Setup Heroku Account and CLI

Activity 2.1: Create Heroku account:

1. Go to Heroku signup page: <https://signup.heroku.com/>
(or login at <https://id.heroku.com/login> if you already have an account)
2. Signup for a free account.
3. Fill in the form, choosing node.js as primary development language.
4. You should receive a link to the email you entered in the form to complete the registration. Follow the instructions in the email.

Activity 2.2: Heroku CLI Installation:

Heroku CLI is a command line interface to perform all tasks on Heroku through the terminal.

1. Heroku CLI requires GIT. If GIT is not installed, please install it: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
2. Install Heroku CLI for your operating system:
 - Linux: `sudo snap install --classic heroku`
 - Mac: `brew tap heroku/brew && brew install heroku`
 - Windows 64-bit: <https://cli-assets.heroku.com/heroku-x64.exe>
 - Windows 32-bit: <https://cli-assets.heroku.com/heroku-x86.exe>

More details can be found at: <https://devcenter.heroku.com/articles/heroku-cli#download-and-install>

3. Verify the installation using the command line:
`heroku --version`

Activity 3: Host your Library app on Heroku

In order to complete this activity, you need the following:

- the Library App (from activity 1 / previous workshop)
- your Heroku account (from activity 2).
- Heroku CLI installed (from activity 2).

Deployment steps:

1. Make sure that a start script is specified in the **package.json** file as follows:

```
"scripts": {
  "start": "node app.js"
},
```

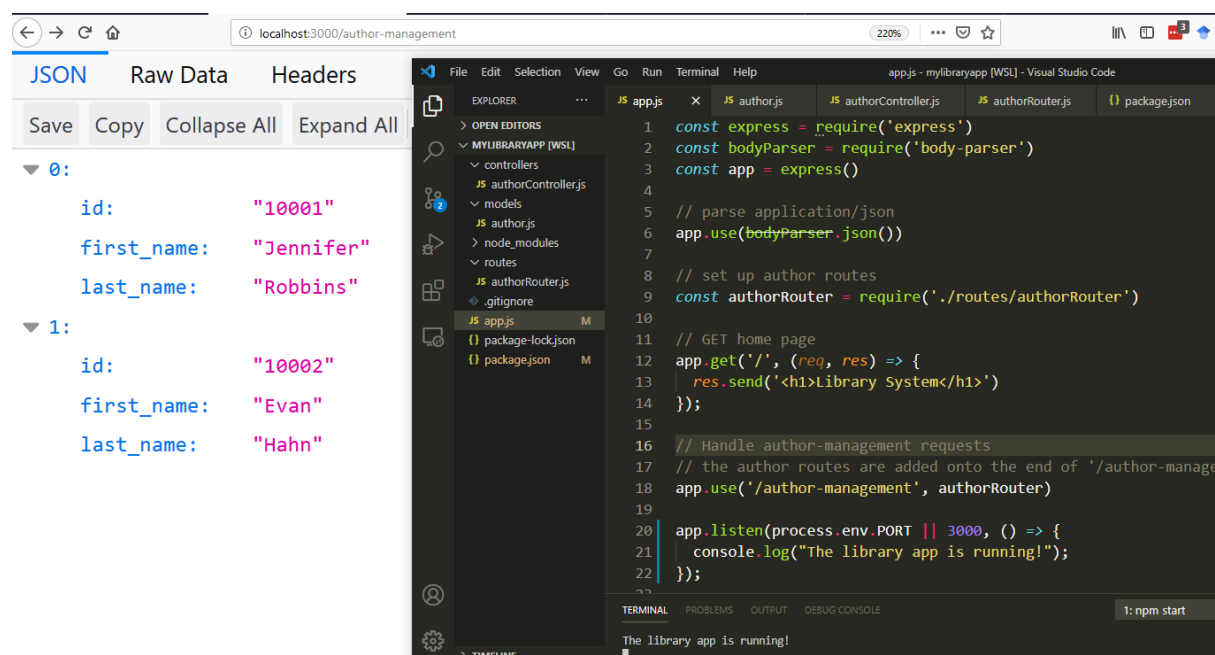
The "scripts" in a package.json file are key/value pairs where the keys are name of commands and the values are the terminal commands that are run.

2. Heroku assigns a dynamic port number to your app, therefore, there will be a problem if your app is listening to a hardcoded port number. To resolve this issue, change the `app.listen` in `app.js` to:

```
app.listen(process.env.PORT || 3000, () => {
  console.log("The library app is running!")
})
```

This will allow the app to listen to either: the port number provided by Heroku when running on Heroku, or your hard-coded port number when running locally.

3. Navigate to your app's working directory, using your terminal.
4. Make sure that the app runs locally without any problem. use **npm start** then open this link (<http://localhost:3000/author-management/>) using your browser.



5. Add and commit these new changes using the following commands:

```
git add .
git commit -m "dynamic port number for app.js"
git push
```

```
home/gregwadley/info30005/mylibraryapp/> git add .
home/gregwadley/info30005/mylibraryapp/> git commit -m "dynamic port number to app.js"
[main 7f16544] dynamic port number to app.js
2 files changed, 2 insertions(+), 2 deletions(-)
home/gregwadley/info30005/mylibraryapp/> git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 410 bytes | 205.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/gregwadley/mylibraryapp.git
60b1420..7f16544 main -> main
home/gregwadley/info30005/mylibraryapp/> git log
commit 7f16544c9006f084527f416115b01720e3f9a576 (HEAD -> main, origin/main)
author: Greg Wadley <gregwadley@gmail.com>
date: Sat Mar 27 10:44:04 2021 +1100

    dynamic port number to app.js

commit 60b14206562df1068760265595f4650fc7d0cf27
author: Greg Wadley <gregwadley@gmail.com>
date: Sat Mar 27 10:25:37 2021 +1100

    first commit
home/gregwadley/info30005/mylibraryapp/>
```

6. Login to Heroku using the command below. The terminal will ask you to press any key to complete the login through your web browser. Note if after logging in, you do not get back to the command prompt, then open a new instance of the command prompt/console/Git Bash.

```
heroku login
```

7. Complete the login to Heroku account and close the browser. You should see a message of successful login in the terminal.
8. Create a Heroku app for your library app, using the command:

```
heroku create [app-name]
```

Note that *app-name* needs to be unique across all of Heroku. You can choose a name like *mylibraryapp-<yourname>*. If you don't provide an app name, Heroku will create a random name for the app.

After the Heroku app is successfully created, you will see a link to the online app in the terminal, as well as the link for the app repository on Heroku (see screenshot).

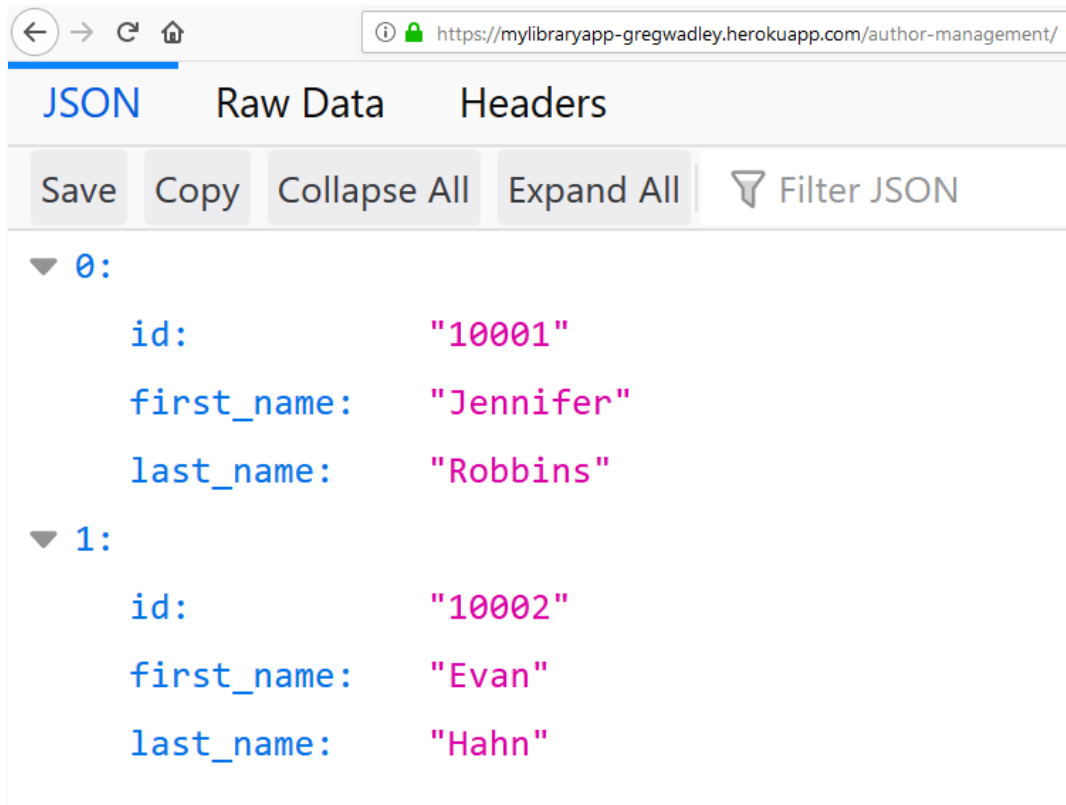
9. Deploy the app to Heroku using the command:

```
git push heroku
```

```
Logged in as gregwadley@gmail.com
/home/gregwadley/info30005/mylibraryapp/> heroku create mylibraryapp-gregwadley
Creating mylibraryapp-gregwadley... done
https://mylibraryapp-gregwadley.herokuapp.com/ | https://git.heroku.com/mylibraryapp-gregwadley.git
/home/gregwadley/info30005/mylibraryapp/> git push heroku
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 8 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (16/16), 6.93 KiB | 591.00 KiB/s, done.
Total 16 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Compressing source files... done.
remote: Building source:
```

After successful deployment, open the link you got at (8) in your web browser to see your app online or use the command line:

heroku open

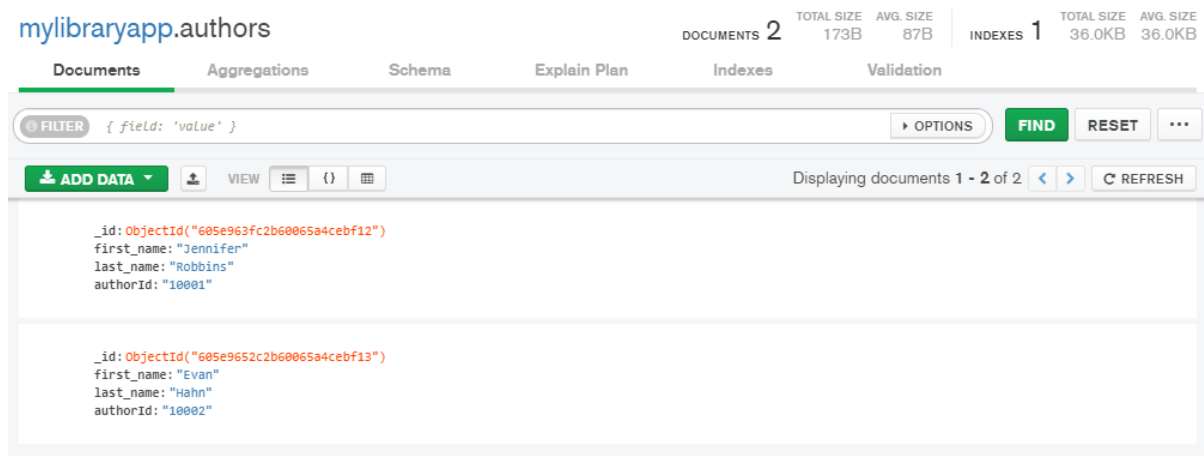


More details about deployment of Node.js apps to Heroku can found at:
<https://devcenter.heroku.com/articles/deploying-nodejs>

Also, you can link the Heroku app to your GitHub repository. You can specify a branch to deploy automatically to Heroku when changes are pushed to GitHub.

Activity 4: Set up MongoDB Atlas

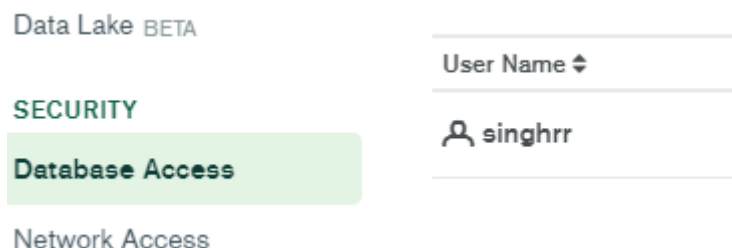
1. Follow the instructions in the "Guide for creating MongoDB on the Cloud.pdf" file to setup a MongoDB Atlas free account.
2. Set up a database named **mylibraryapp** with a collection named *authors*. Add a couple of author documents to your collection.
Note that we have changed the name of the "id" field to "authorId". This is to avoid confusion with the "_id" field that MongoDB automatically adds to each document.



NOTE: ignore the last page in *Guide for creating MongoDB on the Cloud.pdf* - you do not have to follow the instructions on that last page.

3. [OPTIONAL] You can also add different database users using the Database Access page. Then you can use this user's details to allow your app to access your MongoDB instance.

For example, I have added a user named after my username and set a password for this user. In my Node.js app, I will use this user's details (username and password) to connect to the **mylibraryapp** database.



Activity 5: Connect to Mongo using Mongoose

1. Install mongoose in your working directory. Mongoose is a MongoDB object-document modelling (ODM) tool. In your app's working directory, run the following command:












```
npm install mongoose
```

2. Environment variables allow us to manage configuration data and keep this data separate from the app source code, so that is not exposed to people reading your code.

To use environment variables locally we'll use the Node module dotenv:

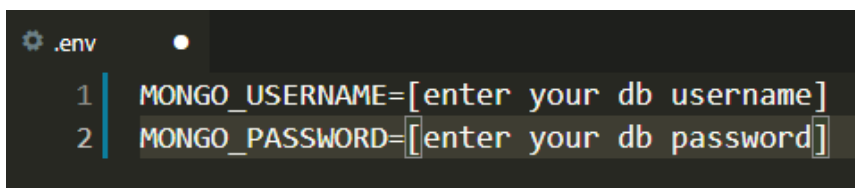
```
npm install dotenv
```

3. Create a file named `.env` inside the app folder and create two environment variables named `MONGO_USERNAME` and `MONGO_PASSWORD` inside it.

	<code>.git</code>	4/17/2020 ...	File folder	
	<code>controllers</code>	3/19/2020 ...	File folder	
	<code>models</code>	4/16/2020 ...	File folder	
	<code>New folder</code>	4/16/2020 ...	File folder	
	<code>node_modules</code>	4/16/2020 ...	File folder	
	<code>routes</code>	3/19/2020 ...	File folder	
	<code>.env</code>	4/16/2020 ...	ENV File	1 KB
	<code>.gitignore</code>	4/17/2020 ...	Text Docu...	1 KB
	<code>app.js</code>	4/17/2020 ...	JetBrains W...	1 KB
	<code>package.json</code>	4/16/2020 ...	JetBrains W...	1 KB
	<code>package-lock.json</code>	4/16/2020 ...	JetBrains W...	23 KB

`.env` should look like this

(replace the placeholders with your real MongoDB username and password)



```
.env
1 MONGO_USERNAME=[enter your db username]
2 MONGO_PASSWORD=[enter your db password]
```

4. This `.env` file won't be committed, and therefore won't be sent to Heroku. To use environment variables on Heroku we'll run two commands;


```
heroku config:set MONGO_USERNAME=[enter your db username]
heroku config:set MONGO_PASSWORD=[enter your db password]
```
5. Use the "Guide for creating MongoDB on the Cloud.pdf" (page 7) to get the cluster address from your Atlas Connection String.

6. Now we need to write additional code that makes our backend server able to talk to the cloud MongoDB server. Create a new file **models/index.js** with the following code:

```
require('dotenv').config()
const mongoose = require("mongoose")

// Connect to MongoDB - database login is retrieved from environment variables - YOU SHOULD
// USE YOUR OWN ATLAS CLUSTER
CONNECTION_STRING =
  "mongodb+srv://<username>:<password>@cluster0.srimt.mongodb.net/mylibraryapp?retryWrites=true&w=majority"
MONGO_URL =
  CONNECTION_STRING.replace("<username>", process.env.MONGO_USERNAME).replace("<password>", process.env.MONGO_PASSWORD)

mongoose.connect(MONGO_URL || "mongodb://localhost", {
  useNewUrlParser: true,
  useCreateIndex: true,
  useUnifiedTopology: true,
  useFindAndModify: false,
  dbName: "mylibraryapp"
})

const db = mongoose.connection

db.on("error", err => {
  console.error(err);
  process.exit(1)
})

db.once("open", async () => {
  console.log("Mongo connection started on " + db.host + ":" + db.port)
})

require("../author")
```

* The statement after `||` means to use the local mongoDB server as backup

* Place your Atlas cluster in the `CONNECTION_STRING` variable.

We also need to modify `app.js`, by adding: `require('../models')` .

```
1  const express = require('express')
2  const app = express();
3
4  require('../models');
5
6  // set up author routes
7  const authorRouter = require('../routes/authorRouter');
8
```

Activity 6: Set up a Mongoose schema

Now that we have all the prerequisites setup and installed. It is finally time to start using a MongoDB database instead of the static authors array which was defined earlier in `models/author.js`.

We need to declare a schema for the author documents in our `mylibraryapp` database.

The new `models/author.js` :

```
const mongoose = require("mongoose")

const authorSchema = new mongoose.Schema({
  authorId: String,
  first_name: String,
  last_name: String
})

const Author = mongoose.model("Author", authorSchema)

module.exports = Author
```

Note that we have to define our Mongoose Schema and then compile it to a Mongoose Model before we can start operating on documents (which are instances of the model).

Take a look at the other [types](#) and [constraints](#) which you can place on schemas. For example, you may want **first_name** values to be non-empty rather than null or undefined: this can be achieved with the schema definition of:

```
first_name: { type: String, required: true }
```

While your app is not yet complete, it should now be connecting successfully to Atlas. If the connection is working you'll see a message like this in the console when you run your app:

```
[nodemon] starting `node app.us app.js`
The library app is running!
Mongo connection started on cluster0-shard-00-00.sprint.mongodb.net:27017
```

Activity 7: Update controller to use database

Having defined the Mongoose model with which we can run database queries, we can now update the controller functions to interact with the database.

1. First, we need to change **controllers/authorController.js**. Change your code to the following. Two functions are supplied: you will complete the rest.

```
const mongoose = require("mongoose")

// import author model
const Author = mongoose.model("Author")

// get all authors
const getAllAuthors = async (req, res) => {
  try {
    const authors = await Author.find()
    return res.send(authors)
  } catch (err) {
    res.status(400)
    return res.send("Database query failed")
  }
}

// find one author by their id
const getOneAuthor = async (req, res) => {
  try {
    const oneAuthor = await Author.findOne( {"authorId": req.params.id})
    if (oneAuthor === null) { // no author found in database
      res.status(404)
      return res.send("Author not found")
    }
    return res.send(oneAuthor) // author was found
  } catch (err) { // error occurred
    res.status(400)
    return res.send("Database query failed")
  }
}

// remember to export the functions
module.exports = {
  getAllAuthors,
  getOneAuthor
}
```

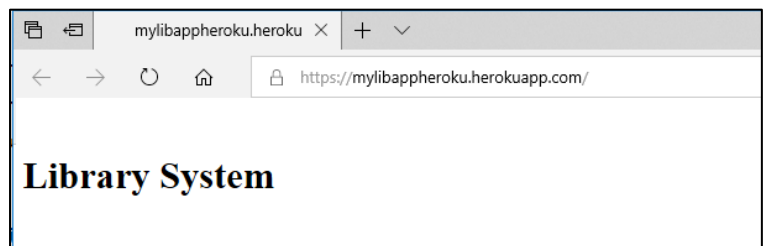
2. Edit your **authorRouter.js** file so that it refers to ":authorId" instead of ":id"
3. Update your git repository with all these changes.

```
git add .  
git commit -m "now connecting to MongoDB"  
git push
```

4. Now deploy the updated app on Heroku using the following command:

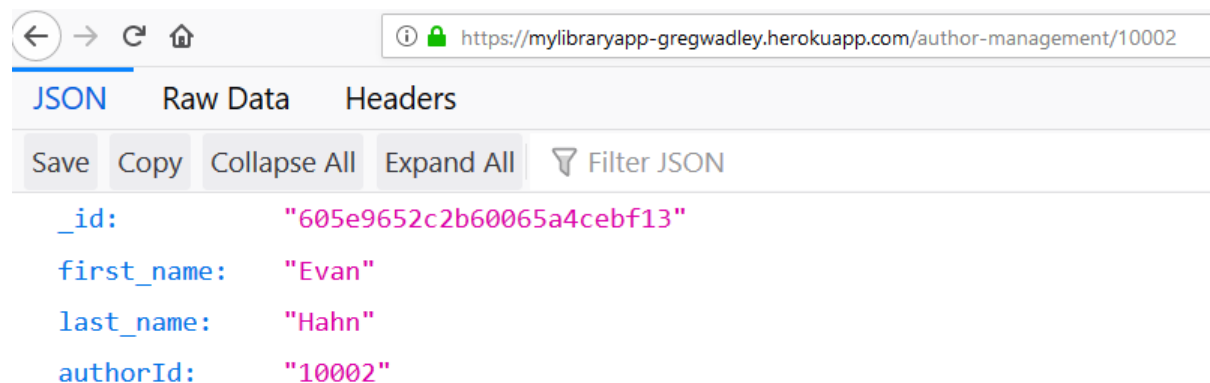
```
git push heroku
```

5. Open the new app in the browser and test it (or use an API tool to test the app).



```
heroku open
```

If your app is working ok, you should be able to query the authors database like so:



Activity 8: Other controller functions

You'll need to create some other controller functions for **authorController.js**, such as:

- update an author
- add a new author to the database.

This is left as a challenge for you!