

Web Information Technologies, Semester 1 2021

School of Computing and Information Systems
The University of Melbourne

Workshop 5: Node and Express

written by Ronal Singh

Objectives

1. install Node, npm and Express
2. start building a simple web application back-end using Node and Express

Activity 1: specifying the Library system

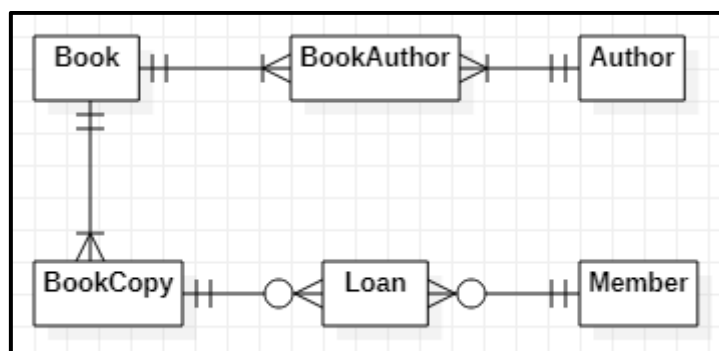
To illustrate some of the concepts that we learnt in lectures, we will progressively build a web-based Library system, for use by a small company to share books with its employees. The library system should store information about:

1. Books (and individual copies of books)
2. Authors of books
3. Members (people who borrow books)
4. Loan details

The information we need to store is illustrated using an Entity-Relationship diagram below. The ER diagram is telling us that:

- each Book has many copies
- each Book can have one or many authors
- each Author can write one or many books
- each BookCopy can be loaned to zero or many Members
- each Member can borrow zero or many BookCopies

We will start with this model of database structure, and think about how we actually implement it later.



Library System Features

We will assume that there are two types of users, librarians and members (borrowers). A member should be able to:

1. Login
2. Search for books
3. Look at their account history

A librarian should be able to:

1. Login
2. Loan out books
3. Add books, members, authors
4. Search for books
5. Look up members' profiles and loan history

That is enough for now! We want to keep things simple. Let us start!

Activity 2: Setting up Node js


1. Download and install Node from here: <https://nodejs.org/en/download/>
2. Or in Linux, type `sudo apt install nodejs`, and then `sudo apt install npm`
3. There is no need to install any additional packages at this stage.
4. Verify that you have node installed by using `node -v` command

Activity 3: Setup the app using Express

Express is a Node.js web application framework. Express is installed as a dependency of your app using *NPM*. NPM is used to fetch packages for your application.

We will setup our Library App using Express. Note that we are not using express-generator as we want to also be able to build a Node app from scratch.

1. Create a working directory named **mylibraryapp** locally (on your laptop).
2. Access that directory using the command line.
3. Use the **npm init** command to create a **package.json** file for your application. This command prompts you for a number of things, including the name (**mylibraryapp**) and version of your application and the name of the initial entry point file (set this to **app.js**). For other input, accept the defaults:

 Ubuntu


```
/home/gregwadley/info30005/> mkdir mylibraryapp
/home/gregwadley/info30005/> cd mylibraryapp/
/home/gregwadley/info30005/mylibraryapp/> ls
/home/gregwadley/info30005/mylibraryapp/> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (mylibraryapp)
version: (1.0.0)
description: my workshop demo
entry point: (index.js) app.js
```

Your `package.json` file should look like:

 Ubuntu

```
/home/gregwadley/info30005/mylibraryapp/> ls
package.json
/home/gregwadley/info30005/mylibraryapp/> cat package.json
{
  "name": "mylibraryapp",
  "version": "1.0.0",
  "description": "my workshop demo",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Greg",
  "license": "ISC"
}
/home/gregwadley/info30005/mylibraryapp/> _
```

- Now install *Express* in the **mylibraryapp** directory by entering the command **npm install express**. Your updated **package.json** should be similar to this:

```

/home/gregwadley/info30005/mylibraryapp/> npm install express
added 50 packages, and audited 51 packages in 4s

found 0 vulnerabilities
/home/gregwadley/info30005/mylibraryapp/> !cat package.json
{
  "name": "mylibraryapp",
  "version": "1.0.0",
  "description": "my workshop demo",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Greg",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
/home/gregwadley/info30005/mylibraryapp/>

```

- Now create the **app.js** file inside the **mylibraryapp** folder using the following code.

This code shows the very basic Express web application for our Library System. We use `require` to import Express, We use Express to create a server (**app**) that listens for HTTP requests on port **3000**. We can now access this server via a web browser.

The **app.get()** function responds to HTTP **GET** requests with the specified URL path (**/**) by sending back a HTML response with `<h1>Library System</h1>` in the body of the HTML document.

```

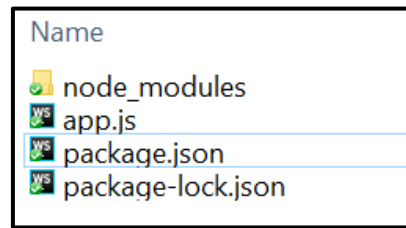
const express = require('express')
const app = express();

app.get('/', (req, res) => {
  res.send('<h1>Library System</h1>')
})

app.listen(3000, () => {
  console.log('The library app is listening on port 3000!')
})

```

6. Your folder should contain the following files and folders

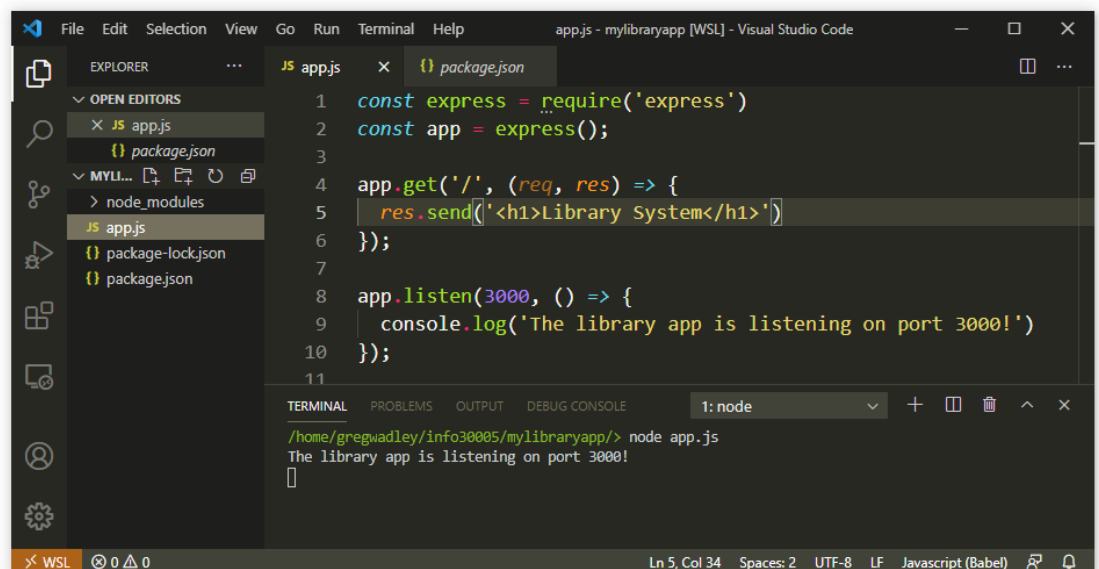


7. Next, start the library app server by entering `node app.js` at your command prompt.

Then open a web browser and browse to: <http://localhost:3000/>. You will see the **Library System** header displayed in the browser.



Library System



8. We have set up the basic library system.

What's next?

The next step is now to add features to our API server. We listed the desired features earlier.

In this workshop, we will start building the features that will allow us to manage author data. You will implement some of the other features later.

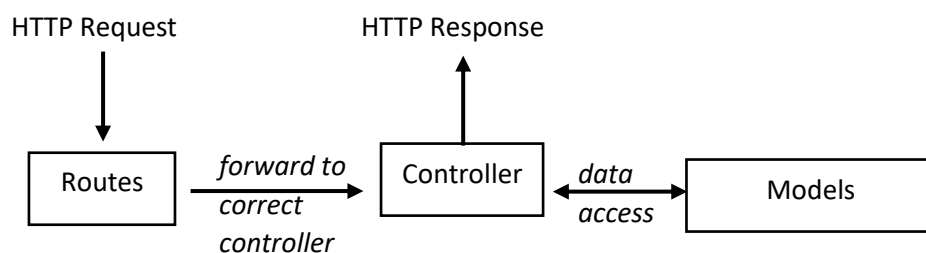
We will start with a discussion on application architecture and folder structure for our project, before implementing some routes associated with author data management.

Activity 4: Architecture and Folder Structure

A common web application design pattern is the Model-View-Controller (MVC) pattern. You can read more about this design pattern here: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>

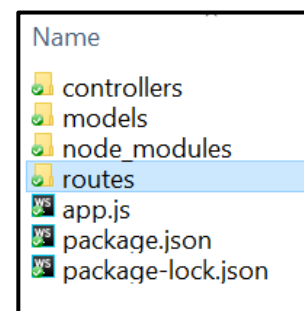
We will use the MVC pattern to create our library app. In this workshop, we will focus on Controllers. The Controller layer implements the business logic. We will also look at the Model layer, which manages the database.

In the following figure, the arrows show the flow of data and control between the various layers. Note that we are limiting ourselves to the layers relevant to this workshop. For example, we do not have a view layer. We will add the view layer in later workshops.



Create the following folders inside your working directory:

1. **models** – for database models
2. **controllers** – for business logic
3. **routes** – for server routes



Activity 5: Authors data model

We will not use a database in this workshop. Instead we will create an array of Authors and use it as we write our controllers.

1. Create a new file name **author.js** in the **models** folder.
2. Add the following array to the **author.js** file.

```

module.exports = [
  {
    "id": "10001",
    "first_name": "Jennifer",
    "last_name": "Robbins",
  },
  {
    "id": "10002",
    "first_name": "Evan",
    "last_name": "Hahn",
  }
]
  
```

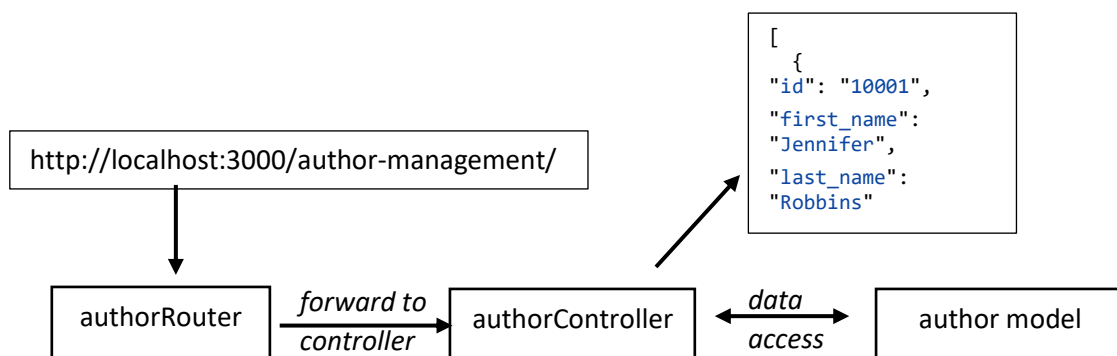
Activity 6: RESTful API to get a list of Authors

We will implement our first route that will get a list of all authors in our database (the JavaScript array in authors.js). How will this work? A user will send an HTTP request, for example, `http://localhost:3000/author-management/`.

We need to decide an appropriate and meaningful naming convention for all our routes. Naturally, the URI `author-management` means that any URI after `author-management` will be concerned with managing authors, such as viewing one or more authors, adding or removing authors, and so on.

The root URI for the `author-management` path will return the list of authors, that is, `http://localhost:3000/author-management/` ending with `'/'` should return the list of all authors. This is a design decision, and how you name your paths and how you handle the resulting functionality is application dependent.

We will use the MVC architecture, without the 'V' view layer. Our model is the author model (file created in Activity 5). The HTTP request (`http://localhost:3000/author-management/`) will first be directed to the `authorRouter`, which will then make use of the `authorController` to get and send back the list of authors.



1. First, we will write the *authorController*. The purpose of the controller is to implement the business logic around author management and manage the interaction between the router and model. Create a file called **authorController.js** inside the **controllers** folder with the following code:

```

//link to author model
const authors = require('../models/author')

// handle request to get all authors
const getAllAuthors = (req, res) => {
  res.send(authors) // send list to browser
}

module.exports = {
  getAllAuthors
}
  
```

2. The author controller will be used by the router, which will direct requests to the controller. So, add a file name **authorRouter.js** in the **routes** folder with the following code:

```
const express = require('express')

// add our router
const authorRouter = express.Router()

// require the author controller
const authorController = require('../controllers/authorController.js')

// handle the GET request to get all authors
authorRouter.get('/', authorController.getAllAuthors)

// export the router
module.exports = authorRouter
```

3. Now that we have our model, router and controller, we will let our app.js handle the requests related to author management. We need to add the author router to **app.js** and tell **app.js** that any request using **author-management** URI needs to be directed to **authorRouter**. Please replace your **app.js** with the following code:

```
const express = require('express')
const app = express()

// set up author routes
const authorRouter = require('./routes/authorRouter')

// GET home page
app.get('/', (req, res) => {
  res.send('<h1>Library System</h1>')
})

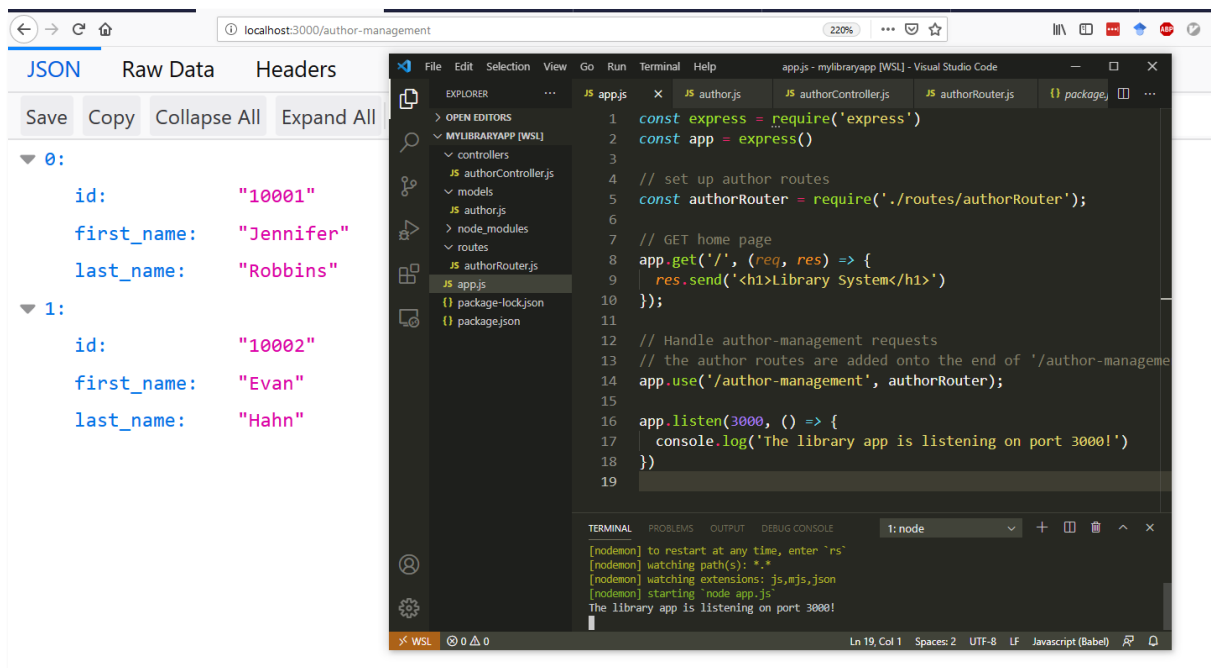
// Handle author-management requests
// the author routes are added onto the end of '/author-management'
app.use('/author-management', authorRouter)

app.listen(3000, () => {
  console.log('The library app is listening on port 3000!')
})
```


- Next, go back to your console, stop (press Ctrl-C) and then restart the library app server by re-entering `node app.js`. In your web browser, connect to <http://localhost:3000/author-management/>. You should see your list of authors in the browser.

The terminal window shows the command `node app.js` being executed, resulting in the message: "The library app is listening on port 3000!". Below the terminal, a web browser window is open at `localhost:3000/author-management`, displaying a JSON array of author objects:

```
[{"id": "10001", "first_name": "Jennifer", "last_name": "Robbins"}, {"id": "10002", "first_name": "Evan", "last_name": "Hahn"}]
```

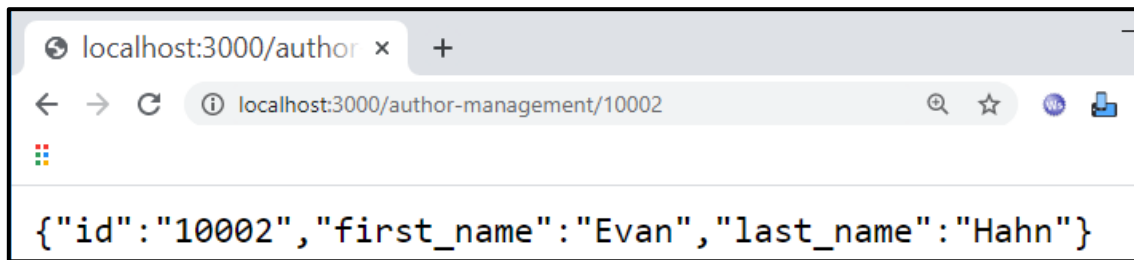


Activity 7: RESTful API to get author by ID

In this activity, we will add a feature that allows users to look up an author by their ID. As a first step, we need to decide on the route path that allows the users to supply the author ID as a parameter. We will use the following path:

`http://localhost:3000/author-management/:id`

The `:id` is a parameter that we provide with the HTTP request. These parameters are added to the `req.params` object as a key-value pair. The key is the name of parameter, `'id'` in our case. For example, the app should return author with id `'10002'` as follows:



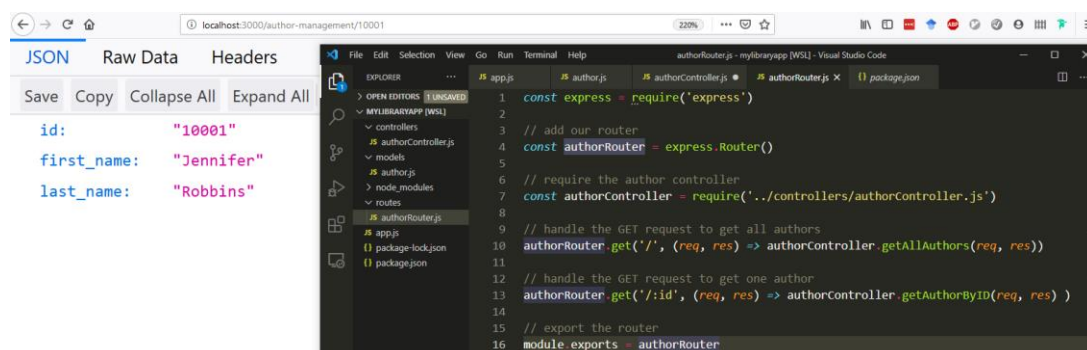
We need a controller function to handle searching for an author using the author ID attribute. You can use the following function and define it in `authorController.js`:

```
// Function to handle a request to a particular author
const getAuthorByID = (req, res) => {

  // search for author in the database via ID
  const author = authors.find(author => author.id === req.params.id)

  if (author){
    res.send(author) // send back the author details
  }
  else{
    // you can decide what to return if author is not found
    // currently, an empty list will be returned
    res.send([])
  }
}
```

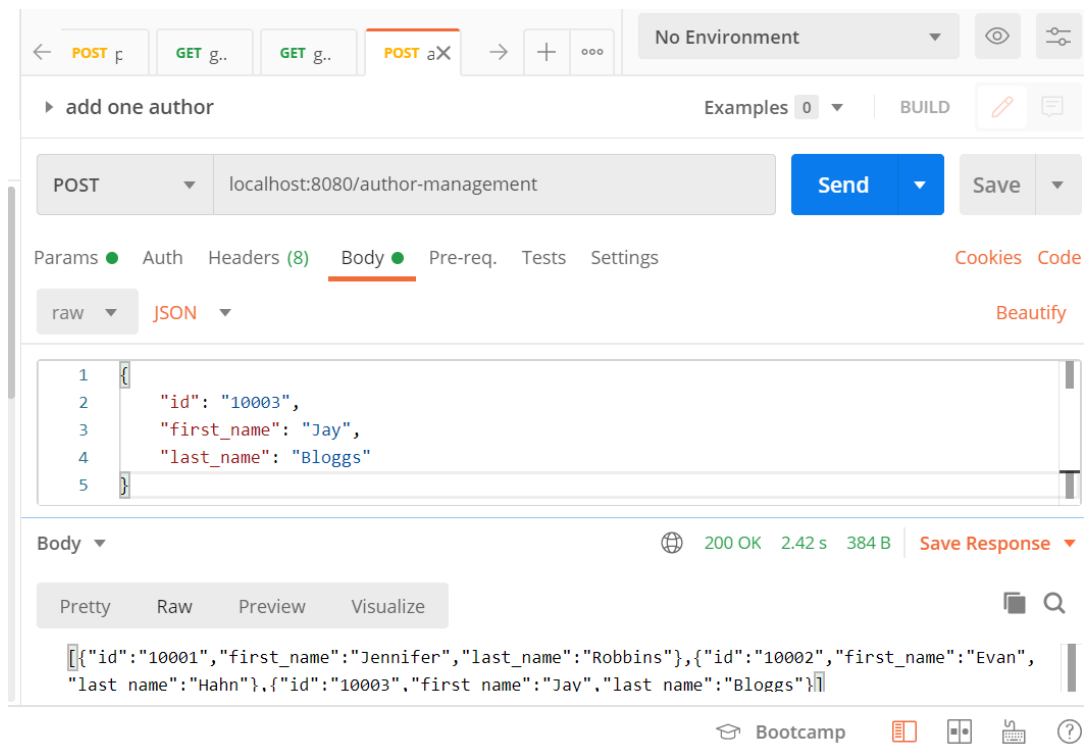
1. Export the `getAuthorByID` function in `authorController.js`.
2. Update `authorRouter.js` to add a GET method to handle the `('/:id')` path. The `'id'` is name of the URL parameter.
3. Question: Why are we not required to change `app.js`? Discuss with your group members.
4. Restart the library app and see that you are able to look up the two authors. Try with invalid author IDs. What happens?



Activity 8: [CHALLENGE] Add an author:

1. Add a new author
 - a. You need to know about the **body-parser** module
 - b. You can make use of an API tool (see below) to test your API. In the tool, you will need to specify that you are using the POST method, and specify form data in the *body*, using JSON.
2. Add the new author to the **authors** array
3. Return the new authors array to the browser to check it worked.

This screenshot from Postman shows the correct HTTP post message and response.



Online Reading

1. <https://nodejs.org/en/about/>
2. Express Routing Introduction: <https://expressjs.com/en/guide/routing.html>
3. MVC Pattern: <https://developer.mozilla.org/en-US/docs/Glossary/MVC>
4. body-parser: <https://www.npmjs.com/package/body-parser>

API Testing Tools

You can use one of the several API tools to test your server API. Some examples include:

1. Advanced REST Client: <https://install.advancedrestclient.com/install>
2. Insomnia: <https://insomnia.rest/>
3. Postman: <https://www.postman.com/>

Request

MethodGETRequest URLhttp://localhost:3000/author-management/10002

SEND

Parameters ^

HeadersVariables

Toggle source mode

+

Insert headers set

ADD HEADER

✓ Headers are valid

Headers size: 0 bytes

200 OK

11.12 ms

DETAILS ▾

```
{  "id": "10002",  "first_name": "Evan",  "last_name": "Hahn"}}
```

Selected environment: Default