CS142: Section 6

Fetching Models from Database

Section Overview

- Project Setup

- Project Implementation
  - Client Side
  - Server Side

Setup

(as usual, download project6.zip, but different because we will be using many of your project5 files)

Project 6 Files(Copy of Project 5) Before unzipping new files

- components/
- images/
- index.html (to be replaced)
- main.css
- mainController.js
- modelData/
- node_modules/*
- package.json (to be replaced)
- photo-share.html
- webServer.js (to be replaced)
- .eslintrc.json (to be replaced)
- .eslintignore

Project 6 Files After unzipping new files

- loadDatabase.js (NEW)
- nodemon.json (NEW)
- schema/ (NEW)
- test/ (NEW)
- package.json (REPLACED WITH NEW)
- index.html (REPLACED WITH NEW)
- webServer.js (REPLACED WITH NEW)
- .eslintrc.json (REPLACED WITH NEW)
- .eslintignore (REPLACED WITH NEW)
- photo-share.html
- components/
- images/
- mainController.js
- modelData/
- main.css

Using MongoDB

Start database service (before starting webserver)

```
mongod
```

Load database with photo app data

```
node loadDatabase.js
```

Mongo Client (not necessary for project, but interesting)

```
mongo
```

Inside mongo client

```
show dbs

use <db_name>

db.users.find();

db.users.find( { name: ‘Joe’ } );
```

Project Overview

Let’s look at the spec!

Project Overview

Server Side (webServer.js)

Implement Routes in webServer.js to:

1. handle incoming requests
2. find the data being requested
3. Send response
4. Handle errors

Routes

/user/list

/user/:id

/photosOfUser/:id

Client Side (component files)

delete fetchModel calls

Use axios to send requests to server

Client Side

Code Walkthrough: Client Side - React & axios

The XHR History Lesson You Never Wanted

November 19th 2017

Despite its nature as a single threaded language, one of JavaScript's greatest strengths is its ability to make asynchronous requests—AJAX.

The Greek hero **Ajax** was good friends with **Achilles** during the Trojan War until **Odysseus** made a request for Achilles' armor

https://hackernoon.com/the-xhr-history-lesson-you-never-wanted-2c892678f78d

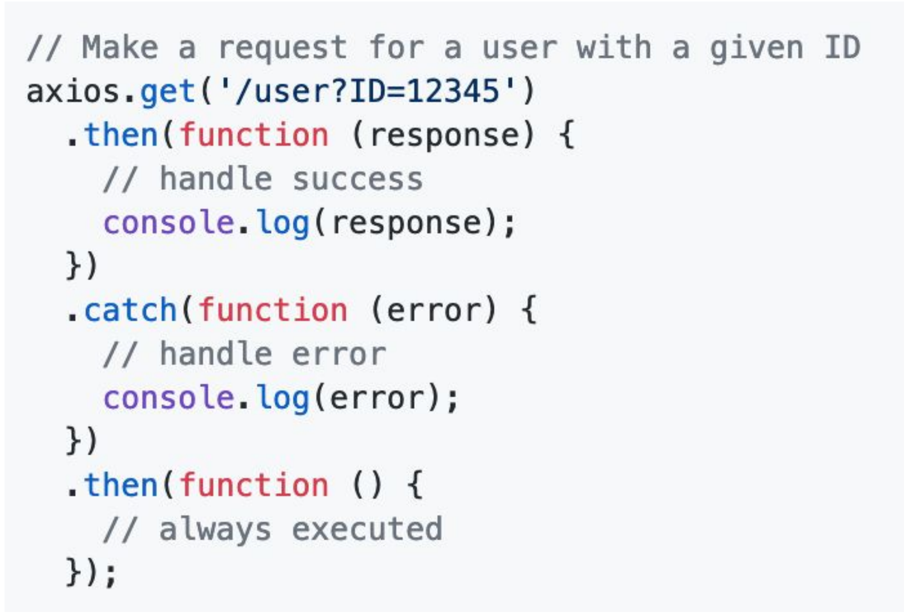
Jonathan Haines, Hackernoon

You worked really hard on getting fetchModel to work in the last assignment, but there are many packages out there that do this faster and better for us

- XMLHttpRequest is a thing of the past
- We're going to use axios
- See documentation here: https://github.com/axios/axios
- This part of the assignment is pretty straightforward -- any time you used fetchModel, replace it with axios.get

Code Walkthrough: Client Side - React & axios

- Axios is Promise-based, similar to fetchModel with slight differences
  - You might have passed .then two callbacks (one to be called on resolve, one for reject)
  - Axios expects your resolve handler in the .then method, and the reject handler in the .catch method
  - console.log your responses and errors to check what they look like!
  - Data you care about probably in response.data
  - Error info in **error.response**



```
// Make a request for a user with a given ID
axios.get('/user?ID=12345')
  .then(function (response) {
    // handle success
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  })
  .then(function () {
    // always executed
  });
```

And that’s it on the client side!

Server Side

Server Side

Implement Routes in webServer.js to:

1. handle incoming requests
2. find the data being requested
3. Send response
4. Handle errors

Routes

/user/list

/user/:id

/photosOfUser/:id

Two lectures with slides that are really worth looking at/understanding:

1. Express lecture
2. Database lecture (specifically, second half of slides about Mongoose and queries)

Server Side

Implement Routes in webServer.js to:

1. **handle incoming requests**
2. find the data being requested
3. Send response
4. Handle errors

Routes

/user/list

/user/:id

/photosOfUser/:id

Two lectures with slides that are really worth looking at/understanding:

1. Express lecture
2. Database lecture (specifically, second half of slides about Mongoose and queries)

Express Apps

Code Walkthrough: Server Side - Express

```
app.get('/some/url/route', function (request, response) {
    //request - data sent as request from client side
    //response - data to be sent back once function is done
    //TODO:
    //find data requested in database
    //send response
        //data successfully found
        //data not found or other errors
});
```

For More: See ExpressJS Lecture (do it, there's good sample code)

How? Well let’s first take a look at how webServer.js was implemented in project5.

(for those of you looking at just the slides, we’re literally going to open the project5 webServer.js file and take a look)

What’s changed in webServer.js from project5 to project6?

Server Side

Implement Routes in webServer.js to:

1. handle incoming requests
2. **find the data being requested**
3. Send response
4. Handle errors

Routes

/user/list

/user/:id

/photosOfUser/:id

Two lectures with slides that are really worth looking at/understanding:

1. Express lecture
2. Database lecture (specifically, second half of slides about Mongoose and queries)

Schemas

Mongoose: Schema define collections

Schema assign property names and their types to collections

String, Number, Date, Buffer, Boolean

Array - e.g. `comments: [ObjectId]`

ObjectId - Reference to another object

Mixed - Anything

```
var userSchema = new mongoose.Schema({
    first_name: String,
    last_name: String,
    emailAddresses: [String],
    location: String
});
```

Mongoose & MongoDB

Mongoose Example

```
var query = Band.findOne({name: "Guns N' Roses"});
assert.ok(!(query instanceof Promise));

// A query is not a fully-fledged promise, but it does have a `.then()`.
query.then(function (doc) {
  // use doc
});

// `.exec()` gives you a fully-fledged promise
var promise = query.exec();
assert.ok(promise instanceof Promise);

promise.then(function (doc) {
  // use doc
});
```

Mongoose and its quirks

Mongoose queries are **not** promises. They have a `.then()` function for co and async/await as a convenience. If you need a fully-fledged promise, use the `.exec()` function.

http://mongoosejs.com/docs/promises.html

But Model.create() and Model.prototype.save() do return Promises! More on this later

For More: See Database lecture and Mongoose Docs

Mongoose Query Operations - Query Builder

```
var query = User.find({});
```

- Projections

  ```
  query.select("first_name last_name").exec(doneCallback);
  ```

- Sorting

  ```
  query.sort("first_name").exec(doneCallback);
  ```

- Limits

  ```
  query.limit(50).exec(doneCallback);
  ```

```
query.sort("-location").select("first_name").exec(doneCallback);
```

Of course, good old Callback

```
User.findOne({login_name: login_name}, function(err, user){
    //Your code here
});
```

Code Walkthrough - Models Returned By Mongoose

`/users/list`

`[`*`User, User, User`*`]`

models returned by Mongoose are JavaScript objects but any modifications that do not match the declared schema are tossed. A simple workaround is to translate the model into JSON and back to an JavaScript objects.

```
JSON.parse(JSON.stringify(modelObject));
```

Code Walkthrough - Models Returned By MongoDB

```
/users/list
    [User, User, User]

User
    "_v": 0 (assigned by MongoDB)
    "_id": "56c7450e6719d2a7a14830c2" (assigned by MongoDB)
    "first_name": "Ian"
    "last_name": "Malcolm"
    "location": "Austin, TX"
    "description": "Should've stayed in the car."
    "occupation": "Mathematician"
```

Note: The DB returns this to you. But the assignment states that your server route should only return to the client what is needed in the list. i.e. “id”, “first_name”, “last_name”

Code Walkthrough - Models Returned By Mongoose

```
/users/:id
    “_v”: 0 (assigned by MongoDB)
   “_id”: “56c7450e6719d2a7a14830c2” (assigned by MongoDB)
    “first_name”: “Ian”
    “last_name”: “Malcolm”
    “location”: “Austin, TX”
    “description”: "Should’ve stayed in the car."
    “occupation”: “Mathematician”
```

Code Walkthrough - Models Returned By Mongoose

```
/photosOfUser/:id
    [Photo, Photo, Photo]
```

Code Walkthrough - Models Returned By Mongoose

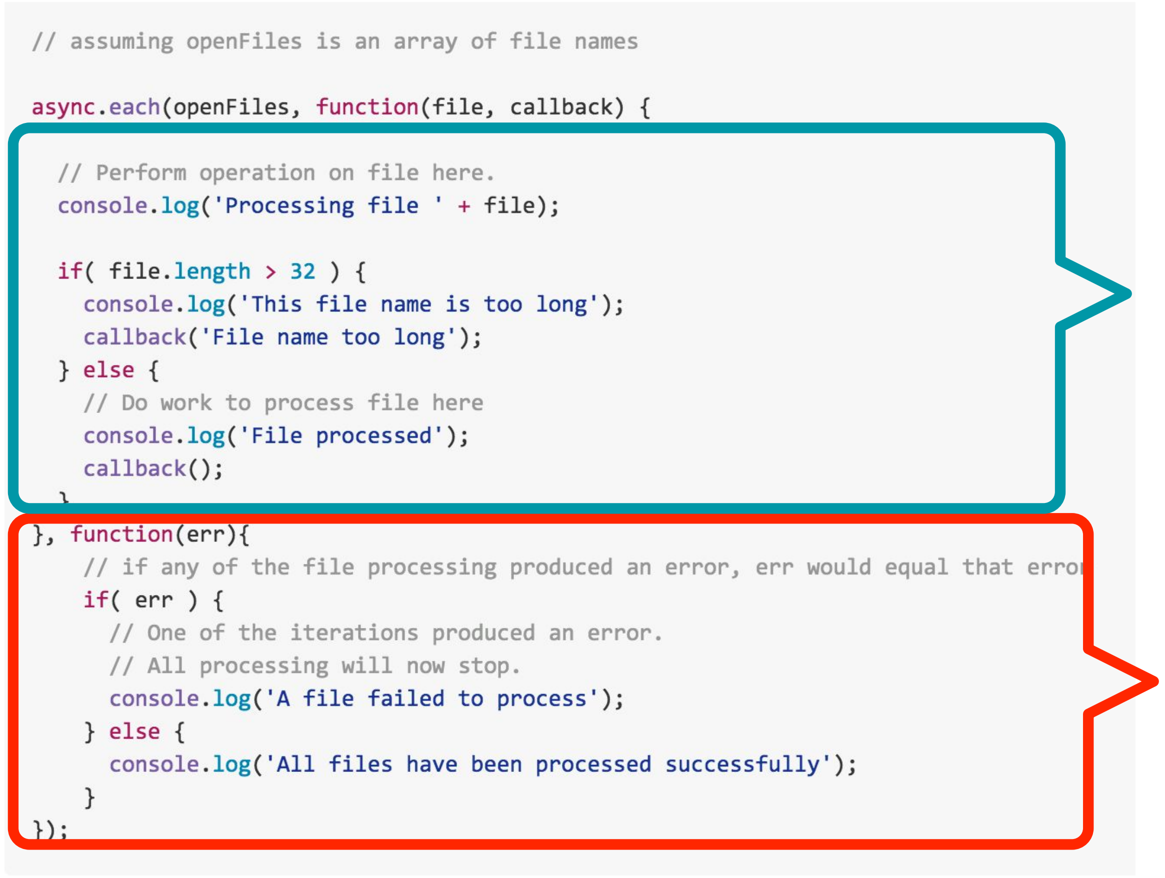
```
/photosOfUser/:id
    [Photo, Photo, Photo]

Photo
   “_v”: 1 (assigned by MongoDB)
   “_id”: “56c7450e6719d2a7a14830cb” (assigned by MongoDB)
    “date_time”: "2013-09-21T00:30:00.000Z"
    “file_name”: "ripley2.jpg"
    “user_id”: "56c7450e6719d2a7a14830c3"
    “comments”: [Comment]
```

Code Walkthrough - Models Returned By Mongoose

```
Comment
   “_id”: "56c7450e6719d2a7a14830d5" (assigned by MongoDB)
    “user_id”: "56c7450e6719d2a7a14830c3"**
   “date_time”: "2013-11-29T01:45:13.000Z"
    “comment”: "Back from my trip. Did IQs just... drop sharply while
I was away?"
```

Note： The photos model returns comment objects with the “user_id” property but it doesn’t make sense to display that value in the comments section of the app. How would you return the name of the user with this info?

async.each(array, iterator, main_callback)



```
// assuming openFiles is an array of file names

async.each(openFiles, function(file, callback) {

  // Perform operation on file here.
  console.log('Processing file ' + file);

  if( file.length > 32 ) {
    console.log('This file name is too long');
    callback('File name too long');
  } else {
    // Do work to process file here
    console.log('File processed');
    callback();
  }
}, function(err){
    // if any of the file processing produced an error, err would equal that erro
    if( err ) {
      // One of the iterations produced an error.
      // All processing will now stop.
      console.log('A file failed to process');
    } else {
      console.log('All files have been processed successfully');
    }
});
```

Calls **iterator** on every item in **array** asynchronously (in parallel).

The **iterator** is called with an item from the **array**, and a callback(err) which must be called once it has completed. If no error has occurred, the callback should be run without arguments or with an explicit null argument.
If the iterator passes an error to its callback, the **main_callback** is immediately called with the error.

main_callback(err) - A callback which is called when all iterator functions have finished, or an error occurs.

Testing:

1. [With your webserver running]
2. Open a new tab in terminal and go into the \test directory
3. npm install
4. npm test

Out of the box, you will be failing 3/11 or 5/11 tests. This is because of placeholder ModelData

This number will go down as you work on transitioning to Mongoose, and then back up!

By the end of the assignment, you should be consistently passing all tests!

Some practical but tangential information coming up, but first…

Any questions?

We’ve been running `npm install` every project,
but how would you go about downloading packages yourself?

https://www.npmjs.com/

do not install things globally for your projects
https://docs.npmjs.com/files/folders