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

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/

modelData/

main.css

Using MongoDB

Start database service (before starting webserver)

mongod <actual command depends on OS…>

Load database with photo app data

node loadDatabase.js

```
(base) qinziyuexu@DNa1ccfc6 projects %  mongod --config /usr/local/etc/mongod.conf
```

Project Overview

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

TWEET THIS

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**

Don't get fixated on this syntax; this is just an example. Use the route format described in the assignment

```
// 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);
  })
```

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 slides that are 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)

Code Walkthrough: Server Side - Express

```
app.get('/some/url/route/:thing', function (request, response) {
        //request- object with data sent as request from client side
        //response- object with 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)

See slides 5 and 6 for more about the `request` and `response` objects

Code Walkthrough: Server Side - Express

Please read through the starter example code for the `/test/:p1` route!!!

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)**

Mongoose & MongoDB

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
});
```

By default, __id property is added to schema with type ObjectId

We can create the model based on schema:
var User = mongoose.model('User', userSchema);

See `schema/` folder for more.
We define schemas and models here

Mongoose Query Operations - Query Builder

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

Just *defines* where and what to get from database

Which model should we look at?

- 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);
```

Mongoose Query Operations - Query Builder

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

This is a *query object*, **NOT** the actual data

- Projections

  From schema!

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

- Sorting

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

- Limits

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

exec() actually sends the query to the database. ***THIS IS ASYNCHRONOUS!!!***

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

Chain together filter operations to refine a query

Mongoose Query Operations - Query Builder

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

- We can define *conditions* for the `find` query

```
User.find({
        location: “Stanford, California”,
        occupation: “Web Developer”
});
```

Properties depend on schema

Translates to: “Get me only the User objects with location as “Stanford, California” and occupation as “Web Developer”.

Asynchronous programming

```
function myFunc() {
        let query = User.find({});
        query.exec((err, res) => {
                // callback function code
                ...
                console.log("The query
is done");
        }
        console.log("hello");
        return;
}
/*
hello  //myFunc returns
The query is done
*/
```

.exec is asynchronous so it is **non-blocking**

Code will continue to run after starting the query

Of course, good old callback functions

```
User.findOne({login_name: the_login_name}, function(err, user) {
    //Your code here
    console.log(user); // This would print out the Mongoose object for the user
                                                with the matching login_name
});
```

Code Walkthrough - Models Returned By Mongoose

/users/list

[User, User, User]

Objects 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 a JavaScript objects.

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

Code Walkthrough - Models Returned By MongoDB

/users/list

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

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”

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”
```

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)

There are also other functions besides `each`. Check the documentation!

The async library is different from the `async` ES6 keyword.

Arguments:

- array - an array of something, usually objects
- iterator - a function that gets called **in parallel** on each element of array
  - Should take 2 parameters: the array element and a callback function to invoke if an error occurs or when you are done processing that element
- Main_callback - a function that gets call after all elements have been processed or if an error happens
  - Takes 1 parameter: the error, if any

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 error
    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 & mongod 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!

Important things to keep in mind

- Is the code I'm writing going to be run asynchronously? How does that affect my other code?

- ***Scope. Scope. Scope.*** Where do I have access to some variable? What can I access from within some function?

Small note: look at the Mongoose documentation, not the MongoDB docs

Any questions?