­MDDN352/NWEN304 Studio Exercise #2

Today we are going to build a simple application that allows an HTML5/Javascript client running in a desktop browser to interact with a Node.js.

Note that although we are using a desktop browser that what you do will also work if you use phonegap to wrap up the HTML5/Javascript code for use on a mobile device.

Much of this studio exercise is based on tutorials from [http://cwbuecheler.com](http://cwbuecheler.com/) with modifications for our local environment.

We recommend that you use a guest login to carry out this tutorial.

You will receive a mark based on how much of the exercise you complete.

Make sure you get your name recorded at the end of the studio and show what you have done in order to receive a mark of 0/1/2% (fail, tried, did well) for each part of the exercise completed.

There are some suggestions of what to do next at the end of the session in case you have extra time.

**PART ONE. Setting up GitHub**

GitHub is a source code system and tools for GitHub are installed on both You will need to create a github account for **both of you**, we intend for you to use these later -- we have approached github for an education account supporting free private repositories and intend to use that if possible.

**1. Go here:** [**https://github.com/**](https://github.com/) **and create an account for the first team member.**

Make you choose the FREE plan, this allows the creation of repositories that are publically viewable.

**2. Repeat the step above for your other team member.**

**3. Create a repository for today’s studio exercise.**

Every time you make a commit with Git, it is stored in a repository (a.k.a. "repo"). To put your project up on GitHub, you'll need to have a GitHub repository for it to live in.

While logged in Click [New Repository](https://github.com/repositories/new).

Click "New Repository

Fill out the information on this page, I used the name **session2** and make sure you click the checkbox **Initialize this repository with a README**.

When you're done, click "Create Repository."

snapshot1.png

Congratulations! You have successfully created your first repository!

You can see the one I created here: <https://github.com/nwen304-test1/session2>

**4. Add your other team member as a collaborator so they also have access to the repository.**

<https://help.github.com/articles/how-do-i-add-a-collaborator>

**PART TWO. Creating a Server.**

We’re going to work on the School servers remotely. There are three of public [undergraduate servers](https://ecs.victoria.ac.nz/Support/TechNoteSchoolMachines" \l "Undergraduate) that we can use: barretts, embassy or regent (all AMD64/Linux).

**1. Get remote access to an undergraduate server.**

Open a **TERMINAL** and log in remotely to one of the machines use X-fowarding to allow us to run graphical programs (replace user with your ECS userid and replace machine with the name of the server):

ssh -X username@machine.ecs.vuw.ac.nz

**2. Make a local working copy of your repository.**

Have a look at the repository page, you will see a **HTTPS clone URL** in the lower right hand corner. Copy this URL.

Use the command below, use the URL you got above to do the cloning.

git clone URL

For example, my test account allowed cloning using the following use of git clone.

git clone https://github.com/nwen304-test1/session2.git

You should now see the README that was created by default.

We will now work in this directory and post changes back to the remote repository.

**3. Create the server code.**

FIRST … Make sure you have express installed.

**npm install express**

The following code is a simple server that provides quotes on request, allows new ones to be created and delete old ones.

Try and use gedit but if that doesn’t work …. use pico.

var express = require('express');

var app = express();

var quotes = [

{ author : 'Audrey Hepburn', text : "Nothing is impossible, the word itself says 'I'm possible'!"},

{ author : 'Walt Disney', text : "You may not realize it when it happens, but a kick in the teeth may be the best thing in the world for you"},

{ author : 'Unknown', text : "Even the greatest was once a beginner. Don’t be afraid to take that first step."},

{ author : 'Neale Donald Walsch', text : "You are afraid to die, and you’re afraid to live. What a way to exist."}

];

// make express handle JSON and other requests

app.use(express.bodyParser());

// serve up files from this directory

app.use(express.static(\_\_dirname));

// if not able to serve up a static file try and handle as REST invocation

app.use(app.router);

app.get('/quote/random', function(req, res) {

var id = Math.floor(Math.random() \* quotes.length);

var q = quotes[id];

res.send(q);

});

app.get('/quote/:id', function(req, res) {

if(quotes.length <= req.params.id || req.params.id < 0) {

res.statusCode = 404;

return res.send('Error 404: No quote found');

}

var q = quotes[req.params.id];

res.send(q);

});

app.post('/quote', function(req, res) {

if(!req.body.hasOwnProperty('author') || !req.body.hasOwnProperty('text')) {

res.statusCode = 400;

return res.send('Error 400: Post syntax incorrect.');

}

var newQuote = {

author : req.body.author,

text : req.body.text

};

quotes.push(newQuote);

res.send(newQuote);

});

app.delete('/quote/:id', function(req, res) {

if(quotes.length <= req.params.id) {

res.statusCode = 404;

return res.send('Error 404: No quote found');

}

quotes.splice(req.params.id, 1);

res.json(true);

});

// use PORT set as an environment variable

var server = app.listen(process.env.PORT, function() {

console.log('Listening on port %d', server.address().port);

});

Save the code into the file **quote.js**.

NWEN304 students should see that the code is very similar to the server developed during last weeks lab except for a few new features.

// serve up files from this directory

app.use(express.static(\_\_dirname));

The code above allows the serving up of files in the current directory,

app.delete('/quote/:id', function(req, res) {

…

quotes.splice(req.params.id, 1);

… });

The code above allows a client to request a particular quote by ID and extract this from the URL.

**4. Create a dummy index.html for serving up to clients.**

Also create a file called **index.html** that will (in the next step) contain your web application code.

<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">

<HTML>

<HEAD>

<TITLE>

A Small Hello

</TITLE>

</HEAD>

<BODY>

<H1>Hi</H1>

<P>This is very minimal "hello world" HTML document.</P>

</BODY>

</HTML>

**5. Test that it works.**

Start the server, set the PORT number using an environment variable (note that as multiple users are on the same machine you might get collisions so if the server doesn’t start try a different port number).

setenv PORT 8002; node quote.js &

Note that **&** makes quote.js runs in the background.

Try some tests:

curl [http://localhost:8002/](http://localhost:8002/quote/1)index.html

curl <http://localhost:8002/quote/random>

curl <http://localhost:8002/quote/2>

curl -H "Content-type:application/json" -X POST -d '{"author" : "foo", "text" : "some text"}' http://localhost:8002/quote

curl -X DELETE <http://localhost:8002/quote/2>

**6. Test it remotely.**

Try opening a browser on the Mac and navigating to <http://barretts.ecs.vuw.ac.nz:8002/index.html>

This MIGHT work but depends upon how the firewalls and other network configurations between ECS, ITS and Design are setup.

I know that I cannot access the server that way from home, at least not directly but I can if I use **ssh tunnelling**. Try this in ANOTHER INSTANCE OF TERMINAL.

You will have to change the highlighted username, servername and port to whatever is used by your server.

ssh -f **ian**@**barretts**.ecs.vuw.ac.nz -L 3001:**barretts**.ecs.vuw.ac.nz:**8002** -N

You should now be able to access the server on barretts (even from home) via:

<http://localhost:3001/index.html>

**7. Update the remote repository.**

We need to make sure your changes go into the remote repository. Use the **add** command.

git add \*

Now we mark the fact that we are going to commit these changes to the repository.

git commit -m “commit message”

Finally, make the changes permanent by pushing the changes (you will need to use your username and password you setup earlier).

git push

You can check the changes have been saved by visiting the repository in github.

**PART THREE. Creating the Web Application.**

Now we are going to switch back to working on the iMacs. We could work directly on the ECS servers but this will model how you could work together for the project with different parts of the team developing different parts of the application and using **git** to coordinate.

The focus of this part is to experiment with ways to dynamically get data from our application into a web page and also post updates back to it.

**1. Create a local working copy of the repository.**

Open a TERMINAL. Create a working directory somewhere on the Mac.

As before use:

git clone URL

All being well you should see a copy of the repository in your working directory.

**2. Test that you can make changes and commit them back.**

Make a change to index.html and commit the change back to the repository.

Use the code that we used in the last studio exercise.

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.2/jquery.mobile-1.4.2.min.css" />

<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>

<script src="http://code.jquery.com/mobile/1.4.2/jquery.mobile-1.4.2.min.js"></script>

</head>

<body>

<!-- Start of first page -->

<div data-role="page" id="foo">

<div data-role="header">

<h1>Foo</h1>

</div><!-- /header -->

<div role="main" class="ui-content">

<p>I'm first in the source order so I'm shown as the page.</p>

<p>View internal page called <a href="#bar">bar</a></p>

</div><!-- /content -->

<div data-role="footer">

<h4>Page Footer</h4>

</div><!-- /footer -->

</div><!-- /page -->

<!-- Start of second page -->

<div data-role="page" id="bar">

<div data-role="header">

<h1>Bar</h1>

</div><!-- /header -->

<div role="main" class="ui-content">

<p>I'm the second in the source order so I'm hidden when the page loads. I'm just shown if a link that references my id is beeing clicked.</p>

<p><a href="#foo">Back to foo</a></p>

</div><!-- /content -->

<div data-role="footer">

<h4>Page Footer</h4>

</div><!-- /footer -->

</div><!-- /page -->

</body>

</html>

Now in the TERMINAL, use the following commands to commit the changes to the remote repository.

git commit -a -m “updated index.html”

git push

Now switch back to your logged in TERMINAL on barretts and use the following commands to pull down the changes so that we can serve the page using your server.

git pull

Now run the quote server again.

node quote.js

Back on the Mac, try accessing <http://barretts.ecs.vuw.ac.nz:8002/index.html> OR if using ssh tunnelling still you can use <http://localhost:3001/index.html>.

All being well you should see the mobile application with multiple pages.

To find out more about using GIT, this is a nice quickstart tutorial: <http://git-scm.com/docs/gittutorial>

**3. Using AJAX to asynchronously retrieve data from the server**

.[AJAX](http://en.wikipedia.org/wiki/Ajax_(programming)) stands for asynchronous JavaScript and XML. If you see another term XHR, which is shorthand for XML HTTP request, it's the same thing. Don't be afraid of this jargon; AJAX is not rocket science.

Some example of how it is used:

* In Gmail, switch from inbox to draft. Part of the page is changed, but the page is not refreshed. You remain on the same page. Url has not changed (except for the #draft at the end of the url, but that's still the same webpage).
* In Google Reader, select a feed. The content changes, but you are not redirected to another url.
* In Google Maps, zoom in or zoom out. The map has changed, but you remain on the same page.

The key to AJAX's concept is "asynchronous". This means something happens to the page after it's loaded. Traditionally, when a page is loaded, the content remains the same until the user leaves the page. With AJAX, JavaScript grabs new content from the server and makes changes to the current page. This all happens within the lifetime of the page, no refresh or redirection is needed.

Due to browser security restrictions, most "Ajax" requests are subject to the [same origin policy](http://en.wikipedia.org/wiki/Same_origin_policy); the request can not successfully retrieve data from a different domain, subdomain, port, or protocol. **This is why we are serving the web application using the same server as the RESTful server.**

Below we will use the Jquery GET Ajax method, the documentation for this method is here: <https://api.jquery.com/jQuery.get/>

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.2/jquery.mobile-1.4.2.min.css" />

<script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>

<script src="http://code.jquery.com/mobile/1.4.2/jquery.mobile-1.4.2.min.js"></script>

</head>

<body>

<!-- Start of first page -->

<div data-role="page" id="foo">

<div data-role="header">

<h1>Foo</h1>

</div><!-- /header -->

<div role="main" class="ui-content">

<label for="text-1">Text input:</label>

<input name="text-1" id="text1" value="1" type="text">

<input type="submit" value="Submit Button" id="button1" />

<!-- put the result here -->

<div id="result" class="functions"></div>

</div><!-- /content -->

<div data-role="footer">

<h4>Page Footer</h4>

</div><!-- /footer -->

</div><!-- /page -->

<script type="text/javascript">

$(document).bind('pageinit', function() {

alert("hello");

$.ajaxSetup ({

cache: false

});

$( "#button1" ).bind( "click", function(event, ui) {

var q = $("#text1").val();

alert("Click "+q);

var jsonUrl = "http://localhost:3001/quote/"+q;

alert("Fetching result from "+jsonUrl);

$.get(jsonUrl, function(data) {

alert(data.author);

$("#result").html("<p><b>"+data.author+" said "+data.text+"</b></p>")

}, 'json');

});

});

</script>

</body>

</html>

**When you have updated index.html don’t forget to commit and push your changes to the repository, and pull the changes back.**

There is a lot going on here. A text field and button are created. The button has a function that retrieves a given quote given a number from our server and outputs it by changing the value of the results div. You could make this much more pretty by using elements from:

<http://demos.jquerymobile.com/1.4.2/>

**4. Using AJAX to add a new quote using POST**

Now try this example that sends data back to the server, in this case adding a new quote. It is bound to a button, no form elements have been used to make it general.

Documentation for POST is here: <https://api.jquery.com/jQuery.post/>

Add a new button to the page:

<label for="text-2">Add a new quote:</label>

<input type="submit" value="Submit Button" id="button2" />

Add some new handling code that uses the Jquery ajax post method to add “foo” and “some text”.

$( "#button2" ).bind( "click", function(event, ui) {

alert("Posting a new quote");

var jsonUrl = "http://localhost:3001/quote";

var newQuote = { "author" : "foo", "text" : "some text" };

$.post(jsonUrl,newQuote, function(data) {

alert("Added " + data.author + " " + data.text);

}, 'json');

});

**PART FOUR. Extending the Web Application.**

You have now completed today’s session and can get marked off by demonstrating the web application working.

If you have time you can explore the following (note I haven’t actually tried these yet myself):

1. Making **add quote** more general, get data from a form and post it to the server.

2. Add a **random quote** button.

3. Add a **delete** form to allow deletion of quotes (you will need to use the full Ajax method, see <https://api.jquery.com/jQuery.ajax/> and perhaps this <http://stackoverflow.com/questions/2153917/how-to-send-a-put-delete-request-in-jquery>).

4. Add error handling to the application.