Web and Mobile Application Development

More Nodejs

Material created by:
David Augenblick, Bill Mongan, Dan Ziegler, Samantha Bewley, and Matt Burlick

Passing Data

- Ok so now our server can:
    - Serve static pages
    - Respond to “action requests”
- How can we send addition information to the web server for it to use in generating the dynamically requested response?
- There are typically two ways

1. The GET approach - Pass the data in the URL in the form:

```
./myurl?var1=val1&var2=val2
```

2. The POST approach – Pass the data in the body of the request

Passing Data

- While the GET approach is easy/convenient, it is not very secure
    - After all, it’s clearly visible
- Plus GET parameters can get cached.
- Therefore when passing important information it is common to use POST

GET Example

- Let's say we put some information in the URL

  http://localhost:8080/getWeather?zip=19147

- Using express, on our server we can put a handler for GET type requests to the page `/getWeather` and use the passed GET information however we like.
- The GET information will be packaged in the `req.query` JSON object

```
var express = require('express');

var app = express();

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

     console.log("Got a GET request for the weather at" + req.query.zip);

});
```

POST Example

- POST data is usually sent through a form or via AJAX (where `type: "POST"`)
- Let's assume that our request was sent to `/getWeather` by one of these means.
- Then we can have a function respond to POST requests to that location:

```
var express = require('express');
var app = express();
app.post('/getWeather', function (req, res){
     console.log("Got a POST request for the weather");
});
```

POST Example

- Unfortunately getting POST data is a little bit trickier since it’s in the *body* of the request.
- A module aptly called `body-parser` can make this a bit easier for us

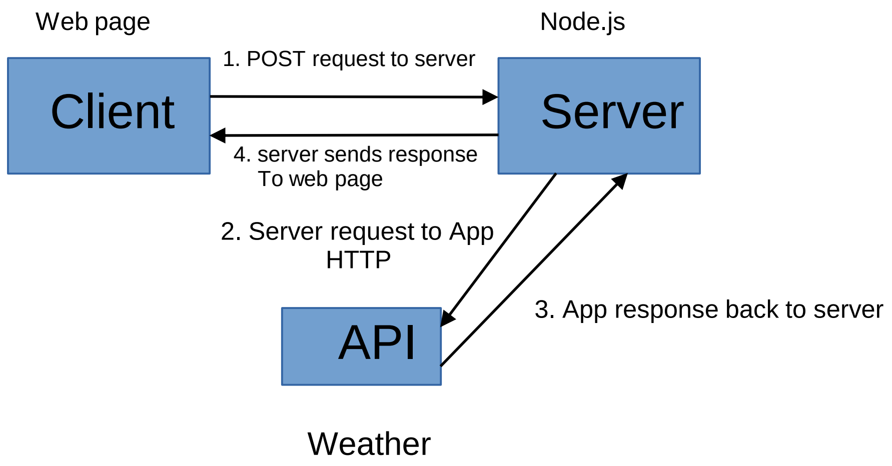
```
var express = require('express');
var app = express();
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({extended:false}));
app.use(bodyParser.json());

app.post('/weatherController', function (req, res) {
   console.log("Got a POST request for the weather at “ + req.body.city);

});
```

Full Example

- Ok let’s redo the weather example but send the request to **our** server which will make the request to the Open Map web service, process it’s results, and return the weather conditions.
    - **This way our server can store it’s key instead of having the user enter it or for it to be stored on the client side.**
- And let’s do this asynchronously!
- And for security let’s do it via POST
- Here once again is the client side’s body

```
<body>
<input type=text id="zipcode" />
<input type=button onclick="requestWeather()‟ value="Get Weather!"/>
<div id=current></div>
</body>
```

Flow Diagram for Weather Program


Web page
Node.js
1. POST request to server
Client
Server
4. server sends response To web page
2. Server request to App HTTP
3. App response back to server
API
Weather

Full Example

- In the web page's header is our AJAX request to **our** server.
- Now we request the **getWeather** action from the server and send via **POST** parameters as a JSON object.

```
<script>
function requestWeather(){
    var zip = $("#zipcode").val();
    params = {
       city: zip
    };
    var URL = "./getWeather";
    $.ajax({
        type: "POST",
        url : URL,
        dataType : "text",
        data : params,
        success : function(msg){
             $(“#current").html(msg);
        },
        error: function(jgXHR, textStatus,errorThrown){
             alert("Error: " + textStatus + " " + errorThrown);
        }
    });

}
</script>
```

Full Example

- On the server let’s allow for both static requests and a response to the **getWeather** POST request.
    - Note we’re using the `body-parser` module here.
        - If necessary `npm install body-parser`
    - And just outputting to the console some info (for now)

```
var express = require('express');
var app = express();
var bodyParser = require("body-parser");

app.use(express.static("."));
app.use(bodyParser.urlencoded({extended:false}));
app.use(bodyParser.json());

app.post('/getWeather', function (req, res) {
   console.log("Got a POST request for the weather at “ + req.body.city);
   res.send(“Coming Soon…”);
});
```

Server Code

- In general part of the server’s job will be to get data and build a response.
- As previously mentioned typically data sources include:
  - JSON data from an external web service.
  - Databases.
  - Local files.
- We’ll get to databases in a bit but for this example we need to
  - Read in from a local file the our API key
  - Make an HTTP request to the Open Weather web service, receive it’s response, and process it.
- Let’s see how we can do each of these!

File Input with Node.js

- We can read in our key from a file using the NodeJS `fs` module (built into NodeJS)(

```
var vs = require(‘fs’);
var key = fs.readFileSync('../Passwords/wkey.txt','utf8');
```

- Alternatively (and probably better since it will be easier to navigate), we could store our content in a JSON file (in json format) and require it.
  - In wukey.json
    ```
    {‘key’: val}
    ```
  - In our script:
    ```
    var wukey = require(‘../Passwords/wukey.json’);
    var key = wukey.key;
    ```
- Note: For security you should **not** store your “private” data in the directory your web server is serving from

HTTP Request from Node.js

- Next we need to be able to make an HTTP request to an external site and process it's response.
- To do the HTTP request from our server, we could use the `http` module
- However, much like how we used the `express` module instead of the `http` one for our server, we're going to use the **`request`** module instead of the `http` one for making external HTTP requests.
  - Just like `express`, it will make things easier/cleaner.
  - But it's not part of default NodeJS, so we need to get it via `npm install request`
- Then we just require it and pass it a URL and callback function
  - This callback function has three parameters
    1. Error message
    2. Full response
    3. Body of the response

HTTP Request from Node.js

- Once installed, our script can require it to get an instance of the module:

```
var request = require('request’);
```

- We can then call the `get` (or `post`) method on of this object, passing in the URL and callback function.
- This callback function has three parameters
  1. Error message
  2. Full response (including headers, etc..)
  3. Body of the response

Full Example

- Putting this together our `getWeather` handler becomes:
    - NOTE: All we did was dump the body out to the command line. Next we'll populate the response.

```
var key = fs.readFileSync('../wundergroundkey.txt','utf8');
var request = require('request');

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

      var URL = 'http://api.openweathermap.org/data/2.5/weather';
      URL += '?zip=' + req.body.city + '&appid=' + key +'&units=imperial';
      request.get(URL, function(error, response, body){
            console.log(body);
            res.send('Coming soon…');
      });
});
```

JSON Parsing in Node.js

- If you recall, the weather API is giving us a JSON response (albeit as a string in the body of the response)
- So populate our response we'll have to parse the JSON response
- Fortunately Node.js has built-in JSON handling
  - It does so through static methods of a JSON class.
- Here's some Common Operations
  - Convert a JSON object to a string

```
var obj = {'key' : 'value'};
var jsonString = JSON.stringify(obj);
```

  - Convert a JSON string to an object

```
var jsonString = "{'key' : 'value'}" ;
var obj = JSON.parse(jsonString);
```

JSON Parsing in Node.js

- Navigating through elements within a JSON object

```
var jsonMajors ={'name1' : 'CS', 'name2 : 'IT'};
for(var Key in jsonMajors) {
    console.log(“Name:” + Key + “Major:” + jsonMajor[Key];
};
```

- Parsing a Weather application response

```
var json = JSON.parse(body);
var city = json.name;
var temp_f = json.main.temp;
```

Full Example

- Finally we have everything we need to make a response to this `getWeather` request:

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

       var URL = 'http://api.openweathermap.org/data/2.5/weather‘;
       URL += '?zip=' + req.body.city + '&appid=' + key +'&units=imperial’;

       request.get(URL, function(error, response, body){
              var json = JSON.parse(body);
              var city = json.name;
              var temp = json.main.temp;
              res.write(city+" " + temp + “F”);
              res.end();
       });
});
```

Cross-Origins Error ☹

- Often you may get the annoying cross-origins request error
- There is a restriction about making general requests across different domains.
  - In particular from a browser to an external web server/service
- If you were to code up everything in the previous slides, started up the NodeJS server, double clicked to open the HTML file, then made your request, the request would be going:

  Filesystem→Browser→Localhost→External API→Localhost→Browser

- The request should be coming directly from the localhost.

  Browser→Localhost→Browser→Localhost→External API→Localhost→Browser

- To do this make sure that instead of opening the HTML file, you request it from the localhost server by typing within your browser something like:

```
http://localhost:<port>/<file>
```