Walk-through 010

EXPRESS 4 SETUP & ROUTES

Objectives:

Express is a popular node framework for building web applications
In this walk-though we will touch the following points:

- Installing Express
- Creating a web application using Express
- Basic Configuration
- Defining URL route listeners

Steps:

Installation

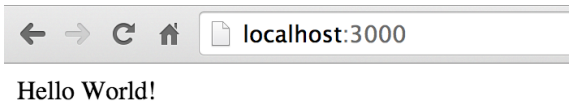
Before we'll use Web-storm's or any other editor's wizards / generators to create an express project, let's do it manually in the leanest possible way using the command line.

1. Create a new folder for this express intro project, and CD into it
2. Run **npm init**
3. Install express locally – **npm install –S express**
   (**-S** is the same as **–save**)
4. Create an **app.js** file in the root project folder with the following code taken from the express website - http://expressjs.com/starter/hello-world.html :

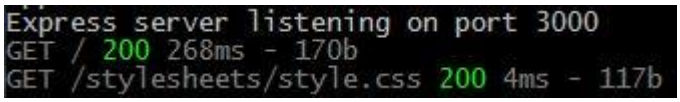
```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

var server = app.listen(3000, function () {

  var host = server.address().address;
  var port = server.address().port;

  console.log('Example app listening at http://%s:%s', host, port);

});
```

5. Run the code using **node app** or **nodemon app**
6. Go in the browser to http://localhost:3000/

   This is what you're suppose to see.



7. Let's add a simple logger to monitor our http calls in the server console.
   a. add the morgan middleware like so: **npm install -S morgan**
   b. add e require statement at the top of the app
      var *morgan* = *require*('morgan');
   c. then in the middleware section add
      app.use(*morgan*('dev'));
   d.
8. Notice the command line as it monitors all the network traffic to your app.



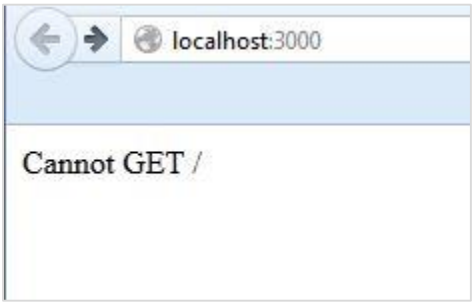
   To quit the server running, run **ctrl + c**
9. Open the app folder in Webstorm

   Express had installed a specific folder structure and some example files

   We will delete a few to make it lean and simple to begin with.
10. Delete the public, routes & views folders
11. Open app.js in the editor and delete all lines apart from these minimum:

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

var app = express();


app.set('port', process.env.PORT || 3000);


http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

12. If you'll run the app and go to the same address in the browser (localhost:3000)
    You'll get the following response:

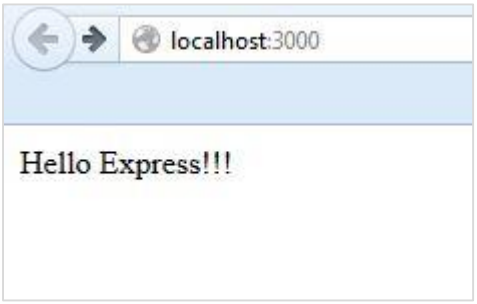


    This is due to the fact that we hadn't set up some routes for our app yet…
    We'll do that next.

13. Add your first route to listen to your root domain folder "***/***"
    We are telling express to listen to GET requests on the root route and specifying a callback that receives two parameters : **req,res** which stands for request and response
    We'll use the response object to send back a string message that would be displayed in the browser.

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

var app = express();


app.set('port', process.env.PORT || 3000);

//routes
app.get('/',function(req,res){
    res.send('Hello Express!!!');
});


http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

restart your app and refresh your browser to see the response:



14. Let's add a second route to our app and send back some basic html markup:

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

var app = express();


app.set('port', process.env.PORT || 3000);

//routes
app.get('/',function(req,res){
    res.send('Hello Express!!!');
});

app.get('/hola',function(req,res){
    var markup = '<h1>Hello Express</h1>'+
                 '<p>This is an example demonstrating some basic html markup<br/>'+
                 'being sent and rendered in the browser</p>'+
                 '<p>Prepare to be:</p>'+
                 '<ul><li>Surprised!</li>'+
                 '<li>Amazed!</li>'+
                 '<li>Inspired!</li></ul>';
    res.send(markup);
});


http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

15. Restart the server (unless you're using nodemon…) and refresh the browser:

16. Let's add another route to our app and pass in a parameter this time:

```
app.get('/shows/:showID',function(req,res){
    res.send('<h1>Up next is Show #'+req.params.showID+'</h1>');
});
```

We can specify a parameter name after a colon **:** and it will sow up in the params object of the request object. This is of course a very practical way to pass in a parameter and the use it for example to query some data from a DB to display in a given template.

17. Save – restart and test the new route in the browser.

18. Next we'll add a POST route to simulate a form sent to a path in our app.

To be able to parse the request body data we'll add a line in our app configuration:

```
var app = express();


app.set('port', process.env.PORT || 3000);
app.use(express.bodyParser());

//routes
```

19. Add the following POST route to your app

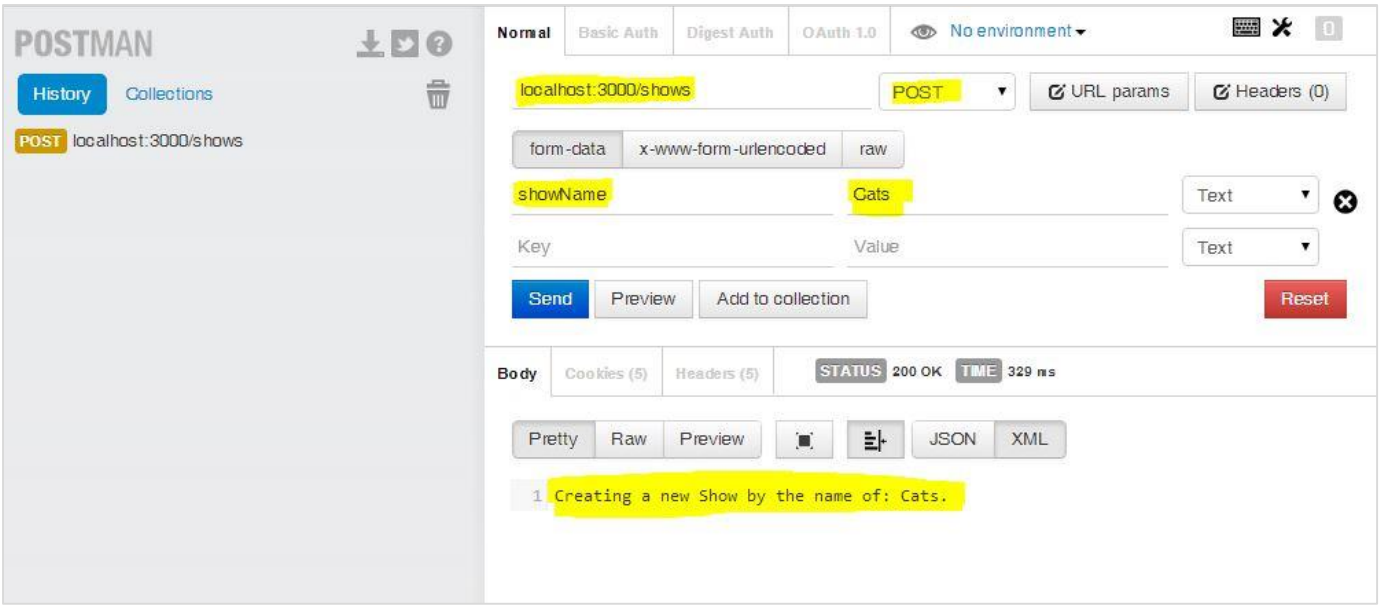
```
app.post('/shows',function(req,res){
    //instead of saving the sent data to a DB, we'll just display it
    res.send('Creating a new Show by the name of: '+req.body.showName+'.');
});
```

20. In order to simulate a POST request in the browser without building our own forms just yet, we'll need a REST client add-on.

    Best one I recommend is a Google Chrome extension called "**Postman – Rest Client**"

    Install it and send a post request to our new route – localhost:3000/shows

    passing a field we'll name **showName** with some value



You could also use **app.put** and **app.delete** to handle put and delete requests though they are not as common.

21. We can also use regular expressions in our routes like so:

    The following example accepts a number of routes, and executing differently according to what rout was passed in eventually.

```
//this route describes a regexp pattern which handles
//  /places/ +  any number of digits + optional slash + optional 'edit'
app.get(/\/places\/(\d+)\/?(edit)?/,function(req,res){
//  all of these urls will activate it:
//  /places/456
//  /places/456/
//  /places/456/edit
    var message = "place #" + req.params[0] + "'s details"

    if(req.params[1] === 'edit'){
        message = "Editing " + message;
    }else{
        message = "Viewing " + message;
    }
    res.send('<h1>' + message + '</h1>');
});
```

    Save + restart + refresh…

22. Last thing I want to mention concerning routes at this stage is handling non-existing pages.

    You can very easily add this at the bottom of your routes block to handle 404 status calls:

```
app.get('*',function(req,res){
    if(res.status(404)){
        res.send('File was not found 404');
        //or  sendfile('path/to/404.png');
    }
});
```

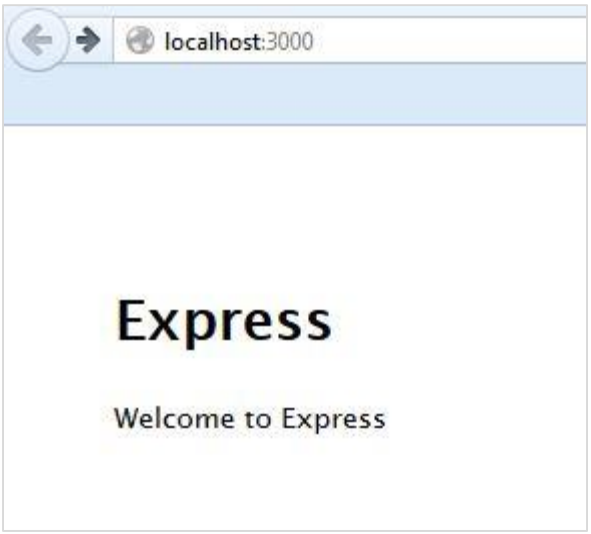
23. Install express-generator globally: **npm install -g express-generator**
24. Install express globally: **npm install -g express**
25. Run **express --help** to see the options when setting up a project with express.
26. **cd** to your root folder where you want to create a folder for your app
27. DEBUG= [YOUR APP FOLDER NAME] ./bin/www
28. Alternatively install nodemon: **npm install –g nodemon**
29. And run it – **nodemon bin/www**

npm install -S ntwitter
npm install -S sentiment

30. Write: **express** [your-app-name]
    (the app name I chose for this example is ExpressIntro…)
    This will created a folder for your app, and in it the express folder structure with some example files, an **app.js** file and a **package.json** file in which the dependencies will be noted.

```
create : ExpressIntro
create : ExpressIntro/package.json
create : ExpressIntro/app.js
create : ExpressIntro/public
create : ExpressIntro/public/images
create : ExpressIntro/public/javascripts
create : ExpressIntro/public/stylesheets
create : ExpressIntro/public/stylesheets/style.css
create : ExpressIntro/routes
create : ExpressIntro/routes/index.js
create : ExpressIntro/routes/user.js
create : ExpressIntro/views
create : ExpressIntro/views/layout.jade
create : ExpressIntro/views/index.jade

install dependencies:
  $ cd ExpressIntro && npm install

run the app:
  $ node app
```

31. **cd** into your app folder and write **npm install** to download and install the dependencies listed in package.json, or do both in one line: **cd** [your-app-name] **&& npm install**
32. now that all dependencies were installed you can run the app - **node app** which will log a notice your app is running on the default port 3000
33. In your favorite browser go to **localhost:3000** and behold!



Notes:

- Be sure to use the Express API documentation for reference - http://expressjs.com/api.html
- A good article on Express routes - http://www.packtpub.com/article/understanding-express-routes