

## Node.js Workshop 4: Web Server, Routes, Forms

After completing this workshop the student knows how to:

- Install new modules using Node Package Manager (npm)
- Create a web server using Express.js
- Implement routes when serving content
- Serve both static and dynamic content
- Form Handling & Processing in Express.js

Create a new folder called *WS4* for these assignments. Place all your code there.

### Install new modules using Node Package Manager (npm)

Node.js comes shipped with a great tool called npm (node package manager). It is a command line tool which makes it easy to download and install additional libraries to use with Node.js. These libraries are written by developers such as yourself and they contain ready-made functionality which makes coding faster and easier.

Npm is invoked using the command line terminal and typing “npm install” and then adding the name of the package at the end. For example, installing a well-known web development framework called express, would be done as described below. Try to install packages called “mysql”, “sails” and “mongodb”.

```
mjstenbe:~/workspace $ npm install express
express@4.13.4 node_modules/express
├── escape-html@1.0.3
├── array-flatten@1.1.1
├── cookie-signature@1.0.6
```

If you wish to see all the packages that are installed, type “npm list”.

```
C:\Users\miksten>npm list
C:\Users\miksten
├── express@4.13.4
├── accepts@1.2.13
│   ├── mime-types@2.1.10
│   └── mime-db@1.22.0
├── negotiator@0.5.3
├── array-flatten@1.1.1
├── content-disposition@0.5.1
├── content-type@1.0.1
├── cookie@0.1.5
├── cookie-signature@1.0.6
├── debug@2.2.0
│   └── ms@0.7.1
├── depd@1.1.0
├── escape-html@1.0.3
├── etag@1.7.0
├── finalhandler@0.4.1
│   └── unpipe@1.0.0
├── fresh@0.3.0
├── merge-descriptors@1.0.1
├── methods@1.1.2
└── on-finished@2.3.0
```

Uninstalling a package can be done in a similar fashion, by typing “npm uninstall” and finish the line with the name of the package.

```
C:\Users\miksten>npm uninstall express
- accepts@1.2.13 node_modules\accepts
- array-flatten@1.1.1 node_modules\array-flatten
- content-disposition@0.5.1 node_modules\content-disposition
- content-type@1.0.1 node_modules\content-type
- cookie@0.1.5 node_modules\cookie
- cookie-signature@1.0.6 node_modules\cookie-signature
- debug@2.2.0 node_modules\debug
- depd@1.1.0 node_modules\depd
- destroy@1.0.4 node_modules\destroy
- ee-first@1.1.1 node_modules\ee-first
- escape-html@1.0.3 node_modules\escape-html
- etag@1.7.0 node_modules\etag
- express@4.13.4 node_modules\express
- finalhandler@0.4.1 node_modules\finalhandler
```

There are lots of switches to use with npm, you can see them all by typing plain “npm” on the console.

```
C:\Users\miksten>npm
Usage: npm <command>

where <command> is one of:
  access, add-user, adduser, apihelp, author, bin, bugs, c,
  cache, completion, config, ddp, dedupe, deprecate, dist-tag,
  dist-tags, docs, edit, explore, faq, find, find-dupes, get,
  help, help-search, home, i, info, init, install, issues, la,
  link, list, ll, ln, login, logout, ls, outdated, owner,
  pack, ping, prefix, prune, publish, r, rb, rebuild, remove,
  repo, restart, rm, root, run-script, s, se, search, set,
  show, shrinkwrap, star, stars, start, stop, t, tag, team,
  test, tst, un, uninstall, unlink, unpublish, unstar, up,
  update, upgrade, v, verison, version, view, whoami
```

You can read more about npm and search for available packages on the website:

<https://www.npmjs.com/>.

## Create a web server using Express.js

Next we’ll install and take a look at Express.js framework. Express is *a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications*. Homepage and lots of information can be found at: <http://expressjs.com/>

Since Express.js is not part of Node.js core functionality, you need to install it using npm. Many other libraries are utilizing Express.js as well, so it might be that some other library already downloaded and installed Express for you as well.

```
mjstenbe:~/workspace $ npm install express
express@4.13.4 node_modules/express
├── escape-html@1.0.3
├── array-flatten@1.1.1
├── cookie-signature@1.0.6
```

## First Express.js application

After Express is installed we can start utilizing it. Type in the following codes in new file called "Task1.js". Note on code line 2, that we need to require the library in to our code to be able to use it.

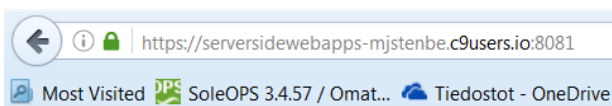
Then on lines 5-6 we are creating a route. This means we are telling Node.js how to react when a browser is asking for a specific url ("/") from our Node.js server. As you see, we are reacting by sending a text "Hello world" to the browser.

Note that using traditional programming languages, they usually require you to have a physical file from where the code is run. When using routes you can simply type in the code within the route clause in the web server code.

Finally on lines 9-10 we are engaging the web server, setting it to listen to port 8081 and logging a message to the console.

```
1
2 var express = require('express');
3 var app = express();
4
5 app.get('/', function (req, res) {
6   res.send('Hello World!');
7 });
8
9 app.listen(8081, function () {
10  console.log('Example app listening on port 8081!');
11 });|
```

Now run the program, point your browser to the Node server and try to get the following response. Make sure you remember to include the port number after the URL, for example, I'm using <https://serversidewebapps-mjstenbe.c9users.io:8081/>. If you're developing on a local machine, you would use <http://127.0.0.1:8081/> or [http://localhost:8081](http://localhost:8081/).



Hello World!

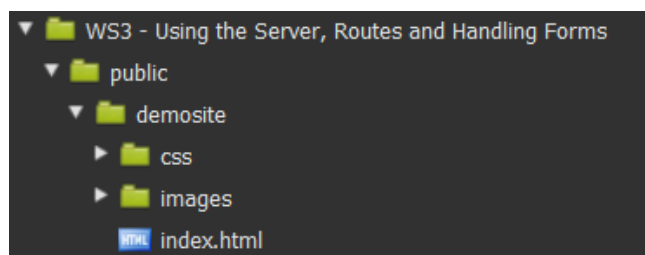
## Serving static content with routes

Previously we have served single files for the browser. Let's try another way to serve a complete directory tree to the user. This way the Node.js acts as a traditional web server serving everything from a specific directory to the browser. Note that you can still define routes within the code if needed. Now create a new file called "directoryserver.js".

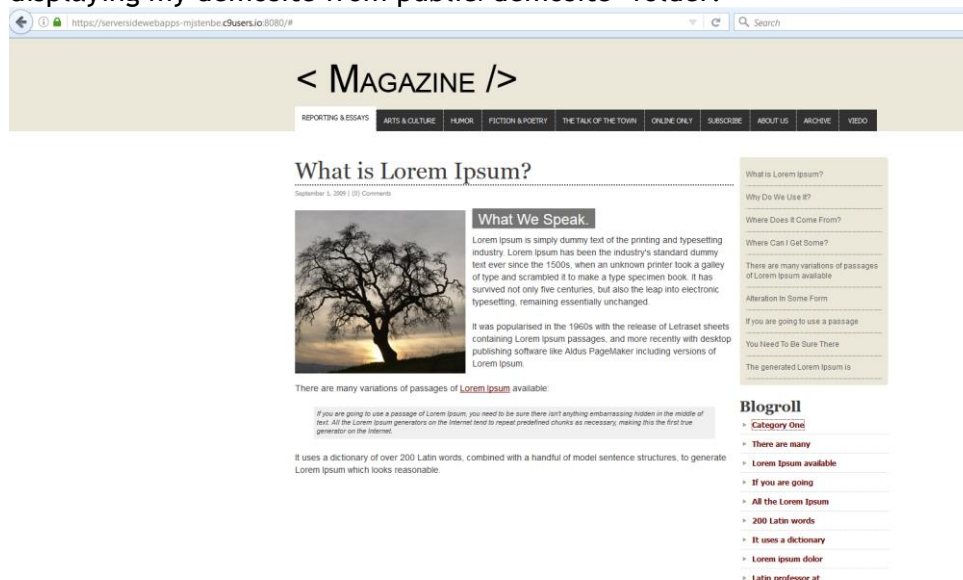
The code below would tell Node.js to serve all the static content from a subdirectory called public/demosite under the root ("/") url of your site.

```
1 var express = require('express');
2 var app = express();
3 app.use(express.static('public/demosite/'));
4
5 app.listen(8081);
6 console.log('8081 is the magic port');
7
```

Note that the directory structure should be as shown below. Your application code should be place in WS3 folder.



As for the demosite, you can upload the demosite we have used in Dynamic Web Apps -course as your demosite contents (found in Optima). Below I'm browsing to the root of my server and Node.js is displaying my demosite from public/demosite -folder.



## Implement more routes with Express.js

Routes are a powerful way of creating program logic and control flow to your web app. Earlier we wrote only one route for our application, now let's write another app which has more routes and functionality.

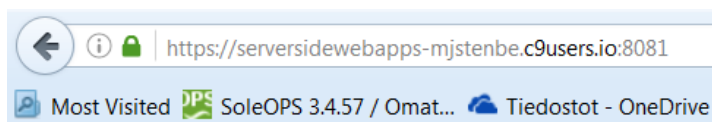
Study the code below and write it to your editor. As you see the first lines 2-3 are similar as we had before; we are just importing the express library and creating a new express.js instance as a variable called app.

Then we're creating 4 different routes in the code. One when the browser is requesting the root page ("/"), one for "/list" and another one for "/add".

The last route on lines 18-20 is a default error page. On route "\*" we're telling Node that if none of the earlier routes matched the incoming request, then we will use this one and sending out a simple error message with 404 error code. This route should always be the last one, because it blocks all the routes defined after it from executing.

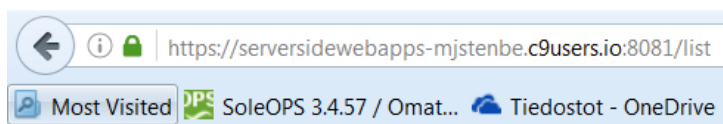
```
1
2 var express = require('express');
3 var app = express();
4
5 app.get('/', function (req, res) {
6   res.send('Hello World!');
7 });
8
9 app.get('/list', function (req, res) {
10   res.send('Listing data from a file!');
11 });
12
13 app.get('/add', function (req, res) {
14   res.send('Lets try to add some data to a file!');
15 });
16
17 //The 404 Route (ALWAYS Keep this as the last route)
18 app.get('*', function(req, res){
19   res.send('Cant find the requested page', 404);
20 });
21
22
23 app.listen(8080, function () {
24   console.log('Example app listening on port 8080!');
25 });
26
```

Now run the code and try out the URL with different page requests. As you can see, by calling different routes you can control which program segments are called. Thus the routes act almost as functions in a conventional program.



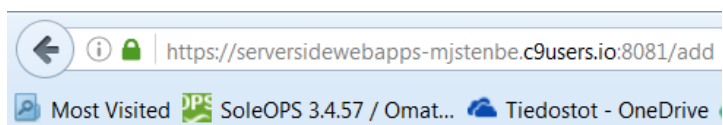
Route for: /

Hello World!



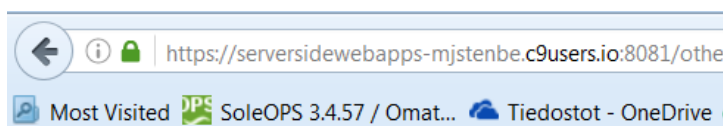
Route for: /list

Listing data from a file!



Route for: /add

Lets try to add some data to a file!



Route for: \*

Cant find the requested page

## Serving content with routes

Previously we only responded with a line of text when different routes were executed. However, now you can replace the content of the routes with whatever JavaScript functionality you can think of: filereads or writes, database requests etc. Note, that although Node.js might be serving static content on some routes, we can serve dynamic content at the other routes, such as “/list”.

Let's work on the program by adding some functionality to **all the 4 routes** we just created.

1. **The root:** Suppose we would want our program to serve a readymade HTML page when the browser requests for site root url (/). We could do this by adding the following lines to our route clause:

```
// Serving a static file instead of a written message
app.get('/', function (req, res) {
  res.sendFile(__dirname + '/public/index.html');
});
```

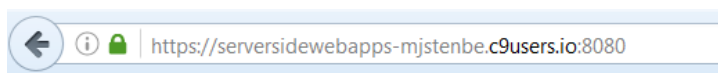
So instead of using `res.send()` -function to send plain text, we could use `res.sendFile()` -to send an entire file to the browser. The variable `__dirname` refers to the current directory my project lies in. Of course you need to create the `public` -folder as well as the `index.html` file to your project.

*NOTE: If you get error saying `res.sendFile()` is not defined, you are using an older version of Express framework. Either try to update it or use `res.sendfile('index.html')` instead.*

For this workshop you can use any HTML page you wish. My demo index page looks like this:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Homepage</title>
5   </head>
6   <body>
7     <h1>Homepage</h1>
8   </body>
9 </html>
```

And when browsing to my URL: <https://serversidewebapps-mjstenbe.c9users.io:8080/>, I would get the following output to the browser.



## Homepage

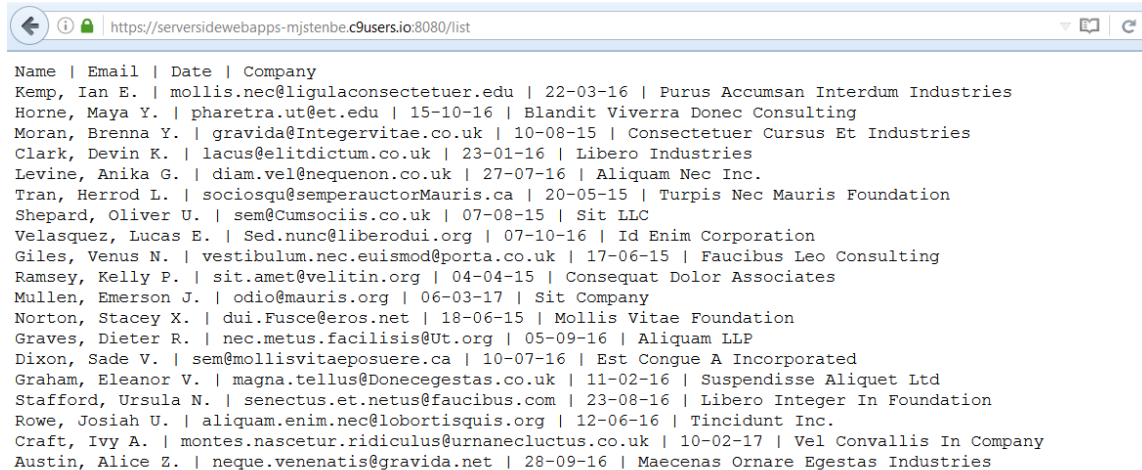
Should there be any links, images or CSS styles, they all work accordingly.

2. **The /list:** The second route we added was for `/list`. This time I would just send contents of a textfile to the user, again using the `res.sendFile()` -function. I have used a set of text data, you can find it here: <http://pastebin.com/ppJ81Y5N>.

```
// Listing user data from a file!
app.get('/list', function (req, res) {
  // Read file with some data and display to the user
  res.sendFile(__dirname+ "/exampledata.txt");
});
```



And the output to the browser would be as below:



Name	Email	Date	Company
Kemp, Ian E.	mollis.nec@ligulaconsectetuer.edu	22-03-16	Purus Accumsan Interdum Industries
Horne, Maya Y.	pharetra.ut@et.edu	15-10-16	Blandit Viverra Donec Consulting
Moran, Brenna Y.	gravida@Integervitae.co.uk	10-08-15	Consectetuer Cursus Et Industries
Clark, Devin K.	lacus@elitdictum.co.uk	23-01-16	Libero Industries
Levine, Anika G.	diam.vel@nequenon.co.uk	27-07-16	Aliquam Nec Inc.
Tran, Herrod L.	sociosqu@semperactorMauris.ca	20-05-15	Turpis Nec Mauris Foundation
Shepard, Oliver U.	sem@Cumsociis.co.uk	07-08-15	Sit LLC
Velasquez, Lucas E.	Sed.nunc@liberodui.org	07-10-16	Id Enim Corporation
Giles, Venus N.	vestibulum.nec.euismod@porta.co.uk	17-06-15	Faucibus Leo Consulting
Ramsey, Kelly P.	sit.amet@velitin.org	04-04-15	Consequat Dolor Associates
Mullen, Emerson J.	odio@mauris.org	06-03-17	Sit Company
Norton, Stacey X.	dui.Fusce@eros.net	18-06-15	Mollis Vitae Foundation
Graves, Dieter R.	nec.metus.facilisis@Ut.org	05-09-16	Aliquam LLP
Dixon, Sade V.	sem@mollisvitaeposuiere.ca	10-07-16	Est Congue A Incorporated
Graham, Eleanor V.	magna.tellus@Donecegestas.co.uk	11-02-16	Suspendisse Aliquet Ltd
Stafford, Ursula N.	senectus.et.netus@faucibus.com	23-08-16	Libero Integer In Foundation
Rowe, Josiah U.	aliquam.enim.nec@lobortisquis.org	12-06-16	Tincidunt Inc.
Craft, Ivy A.	montes.nascetur.ridiculus@urnanecluctus.co.uk	10-02-17	Vel Convallis In Company
Austin, Alice Z.	neque.venenatis@gravida.net	28-09-16	Maecenas Ornare Egestas Industries

Ofcourse the data could be in JSON format as well (and it usually is when working with Node). Try another dataset in JSON from here: <http://pastebin.com/HxbRFvjy>. Below I have created a new route called /jsondata which loads the JSON file and sends it to the browser.

```
// Send out the entire raw data
app.get('/jsondata', function (req, res) {
  var data = require('./exampledata2.json');
  res.json(data);
});
```

And yet, someone might appreciate if we would parse the data to the screen. Remember we already tried this on last weeks Workshops? There are many ways to approach this, but you could do something like this to parse the data from JSON object:

```
// Or we can parse out the details

app.get('/details', function (req, res) {
  var data = require('./exampledata2.json');

  // Parse the results into a variable
  var results = '<table border="1"> ';

  for (var i=0; i < data.length; i++){
    results +=
    '<tr>'+
    '<td>'+data[i].Name+'</td>'+
    '<td>'+data[i].Email+'</td>'+
    '</tr>';
  }

  res.send(results);
});
```

The output would be something like below. Try modifying the code above and the remaining data fields to the output. See the sample dataset to find out what data is not yet printed.



Garner, Summer K.	dictum.magna.Ut@mieleifend.com
Branch, Audra G.	amet.risus@urnaconvalliserat.com
Key, Zena H.	condimentum.Donec@afacilisisnon.com
Love, Wing Y.	sodales.nisi.magna@metus.ca
Ford, Ruby I.	eget.massa@enimcommodo.ca
Dennis, Wanda T.	imperdiet.erat@ultricessit.org
Gonzalez, Dara U.	amet.massa.Quisque@CrasinterdumNunc.ca
Phelps, Hunter L.	lorem.fringilla.ornare@scelerisque.edu
Graham, Hedwig W.	fringilla@Duisatlacus.org
Cameron, Meredith B.	semper.egestas.urna@Pellentesqueutipsum.net
Sharpe, Elmo P.	arcu.et@tempusrisusDonec.edu

3. **The /add:** Next we will implement the /add route, which will let us add data to a JSON object. Remember we did this already during last weeks session, this time we will only add the output to the browser. See the comments in the code to understand what happens.

```
// Add data
app.get('/add', function (req, res) {

  // Load the existing data from a file
  var data = require('./exampledata2.json');

  // Create a new JSON object and add it to the existing data variable
  data.push({
    "Name": "Mika Stenberg",
    "Company": "Laurea",
    "Email": "mika@laurea.fi",
    "Date": "30/3/2016 \r\n"
  });

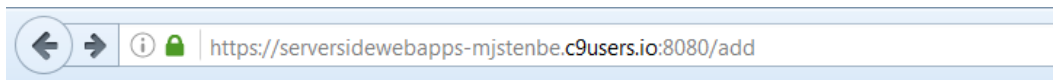
  // Convert the JSON object to a string format
  var jsonStr = JSON.stringify(data);

  // Write data to a file
  fs.writeFile('exampledata2.json', jsonStr, (err) => {
    if (err) throw err;
    console.log('It\'s saved!');
  });

  res.send("Saved the data to a file. Browse to the /details to see the contents of the file");
});
```

Hopefully most of the code looks familiar to you. Basically we just load the existing data, add a new JSON data on the bottom of it using `push()` and then saving it to the server again. This would be the easiest way to implement a simple database functionality using textfiles.

By browsing to the `/add` you should get the notification below.



Saved the data to a file. Browse to the `/details` to see the contents of the file

And looking the `/details` from the browser you can see the added data at the bottom:

Ewing, Giselle J.	nec@Cras.com
Lopez, Kermit W.	pharetra.Quisque@vitaesodalesnisi.co.uk
Byrd, Linda D.	mauris.rhonus.id@consequatnec.net
Porter, Salvador Z.	vehicula.aliquet.libero@nislsemconsequat.org
Richard, Briar P.	est.arcu@Phasellusin.org
Gonzales, Forrest Z.	dictum.magna.Ut@sapienimperdietornare.com
Mika Stenberg	mika@laurea.fi

Now you must be thinking; how can I specify the data to be saved. And this is what we will implement next using forms.

## Reading form data using Node.js

Lets first create an HTML form in a new file. Feel free to improvise. Note that the form “method” is set to POST. Also note that “action” is set to `/adduser`. These mean that the browser is sending the data using the POST method to the URL defined in action parameter.

Here’s a simple suggestion:

```

1 <h1>Add users</h1>
2 <form method="post" action="/adduser">
3   Name <input type="text" name="name">
4   Email <input type="text" name="email">
5   Company <input type="text" name="company">
6   <input type="submit" value="Submit">
7 </form>
8
9

```

To keep things simple, let's create a new JavaScript file (formdemo.js) for this form demo. You can copy and paste code from the previous file if you want to later on.

The beginning of the file contains two new lines: 6 and 7. Basically we're just importing a module called bodyParser and applying it in our Express application. Bodyparser is needed in order to get the sent data from the forms.

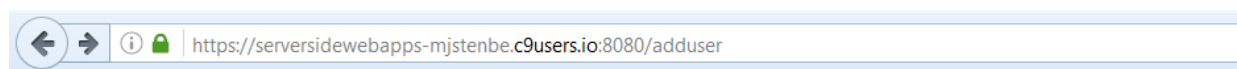
*NOTE: If you get error about body-parser, you need to install it using npm. So open a new terminal window or local Node.js terminal window and type in: "npm install body-parser".*

```
1 var express = require('express');
2 var fs = require('fs');
3 var app = express();
4 |
5 // Require the module required for using form data
6 var bodyParser = require('body-parser');
7 app.use(bodyParser.urlencoded({ extended: true })); // for parsing applicati
8
```

Next we'll create a route which will show us the form:

```
// Serve a form to the user
app.get('/adduser', function (req, res) {
  res.sendFile(__dirname + '/public/adduser.html');
});
```

Go ahead and try to see it in your web browser.



## Add users

Name  Email  Company

Then we need to set up a route, which will handle the POST request sent by the form. We defined this as /adduser in the HTML of the form. NOTE that the route is defined as app.post - not app.get as before. This is because we are expecting a POST request.

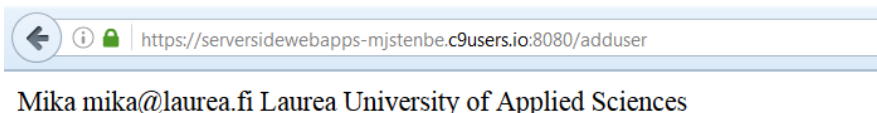
When we receive the POST request we can access the form data using the req -variable. The names of the variables are defined in the HTML of the form.

So here goes:

```
// Route for form sending the POST data

app.post('/adduser', function (req, res) {
  var data="";
  data += req.body.name + "\n";
  data += req.body.email + "\n";
  data += req.body.company + "\n";
  console.log(data);
  res.send(data);
});
```

Running the file would give us:



So now we know how to create a form, how to serve it and how to receive data. Next you can copy the details-route and the add-route from the previous demos. We will modify the add-route a bit and merge it with the adduser-route created.

The details-route reads the JSON data from a file and displays it on the browser window (see page 7 and 8 if you forgot). This is the same as before.

```
// Or we can parse out the details

app.get('/details', function (req, res) {
  var data = require('./exampledata2.json');

  // Parse the results into a variable
  var results = '<table border="1"> ';

  for (var i=0; i < data.length; i++){
    results +=
      '<tr>'+
      '<td>'+data[i].Name+'</td>'+
      '<td>'+data[i].Email+'</td>'+
      '</tr>';
  }

  res.send(results);
});
```

The add-route on the other hand is used to add data to that file. Earlier we only added data that was set in the code. Now we will modify the code so that we can add data from a form.

```
// Create a new JSON object and add it to the existing data variable
data.push({
  "Name": "Mika Stenberg",
  "Company": "Laurea",
  "Email": "mika@laurea.fi",
  "Date": "30/3/2016 \r\n"
});
```

Now we will get the data from the FORM using the req variable, like below. NOTE: The structure of the data is the same, only the contents is handled differently. See also, that the date is fetched using the JavaScripts new Date() object, to populate the field using current date.

```
data.push({
  "Name": req.body.name,
  "Company": req.body.company,
  "Email": req.body.email,
  "Date": new Date()
});
```

So the complete route for adding users would be (NOTE, we're using app.post):

```
// Route for form sending the POST data

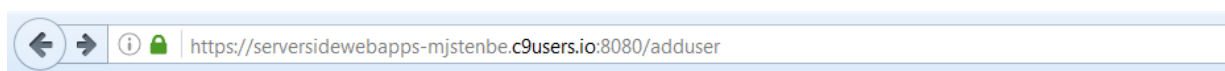
app.post('/adduser', function(req, res) {
  // Load the existing data from a file
  var data = require('./exampledata2.json');

  data.push({
    "Name": req.body.name,
    "Company": req.body.company,
    "Email": req.body.email,
    "Date": new Date()
  });

  // Convert the JSON object to a string format
  var jsonStr = JSON.stringify(data);

  // Write data to a file
  fs.writeFile('exampledata2.json', jsonStr, (err) => {
    if (err) throw err;
    console.log('It\'s saved!');
  });
  res.send("Saved the data to a file. Browse to the /details to see the contents of the file");
});
```

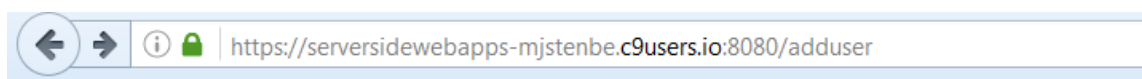
Now lets try it:



## Add users

Name  Email  Company

Clicking submit will give us:



Saved the data to a file. Browse to the /details to see the contents of the file

And browsing ahead to /details gives us the extra data I just entered (at the bottom of the page):

Ewing, Giselle J.	nec@Cras.com
Lopez, Kermit W.	pharetra.Quisque@vitaesodalesnisi.co.uk
Byrd, Linda D.	mauris.rhonus.id@consequatnec.net
Porter, Salvador Z.	vehicula.aliquet.libero@nislsemconsequat.org
Richard, Briar P.	est.arcu@Phasellusin.org
Gonzales, Forrest Z.	dictum.magna.Ut@sapienimperdietornare.com
Mika Stenberg	mika@laurea.fi
Mika Stenberg	mika@laurea.fi
Mika Stenberg	mika@laurea.fi
Mika Stenberg	mika@laurea.fi
Mika	mika@laurea.fi
Pekka	pekka@sci.fi



