Node.js Workshop 2: Working with Files and Directories

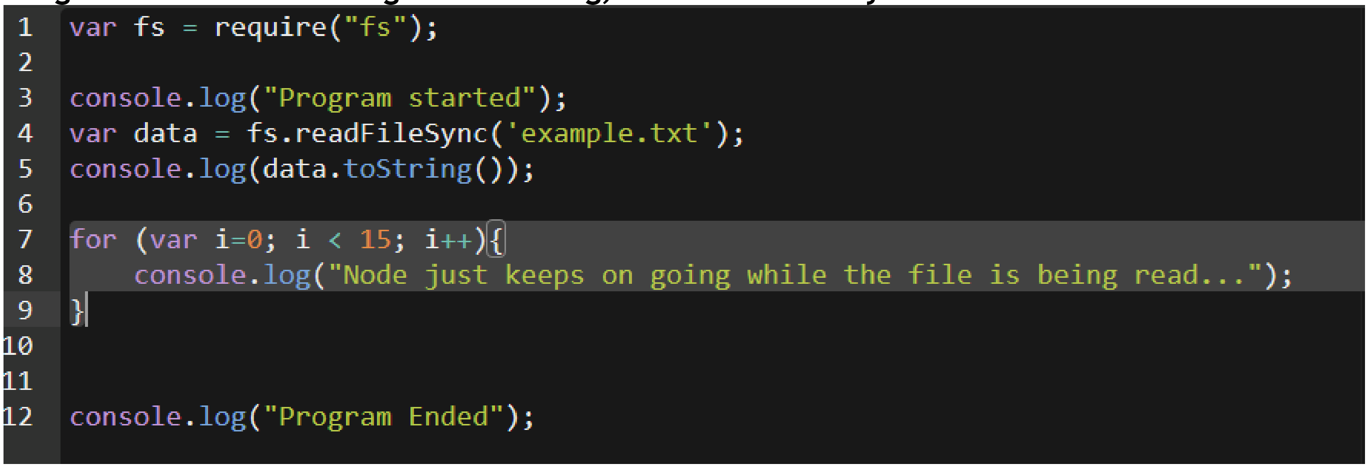
After completing this workshop the student is knows how to:

- Read Node.js API documentation in the Internet
- Understand synchronized and asynchronized code blocks
- Read, write and delete text files
- Read, write, create and remove directories
- Utilize Routes with files
- Process JSON files

Create a new folder called WS2 for these assignments. Place all your code there.

Create the apps below and see the difference. Try to understand why do they work differently? You can find the code here:

Program A: Read file using non-blocking, event driven way



```
var fs = require("fs");

console.log("Program started");
var data = fs.readFileSync('example.txt');
console.log(data.toString());

for (var i=0; i < 15; i++){
    console.log("Node just keeps on going while the file is being read...");
}


console.log("Program Ended");
```

Program B: Read file using the traditional blocking way

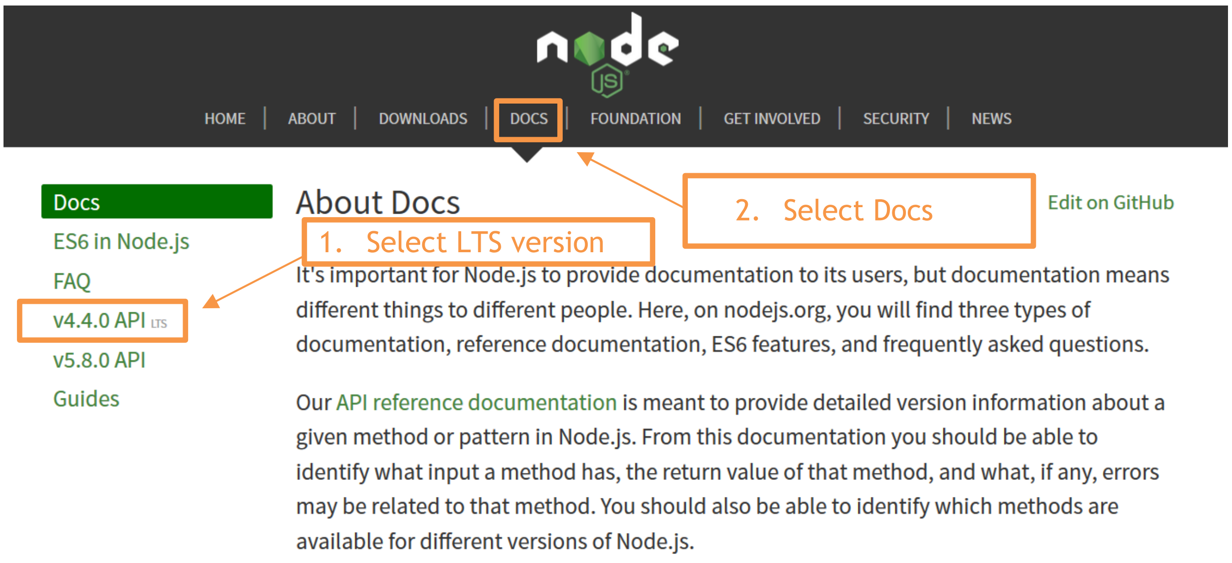
```javascript
var fs = require("fs");

console.log("Program started");

// We'll just give a name of the callback function, but define it later on
fs.readFile('example.txt', results);

for (var i=0; i < 15; i++){
    console.log("Node just keeps on going while the file is being read...");
}

// Introduce a function to deal with fileread results
function results(err, data){
    if (err) return console.error(err);
    console.log("Results of fileread:");
    console.log(data.toString());
}
```

Read Node.js API documentation in the Internet

All the modules contained in Node.js core are listed in Node.js API documentation. This should be your main source for help when trying to know how a specific function works or whether Node.js can perform something you're interested in doing.

Take a moment to browse the API and find some of the functions we've covered so far in the workshops, such as console.log and readFile / readFileSync.

In the todays tasks, you are asked to code some programs using a specific function. Use API documentation to find the function and try to figure out how it works.



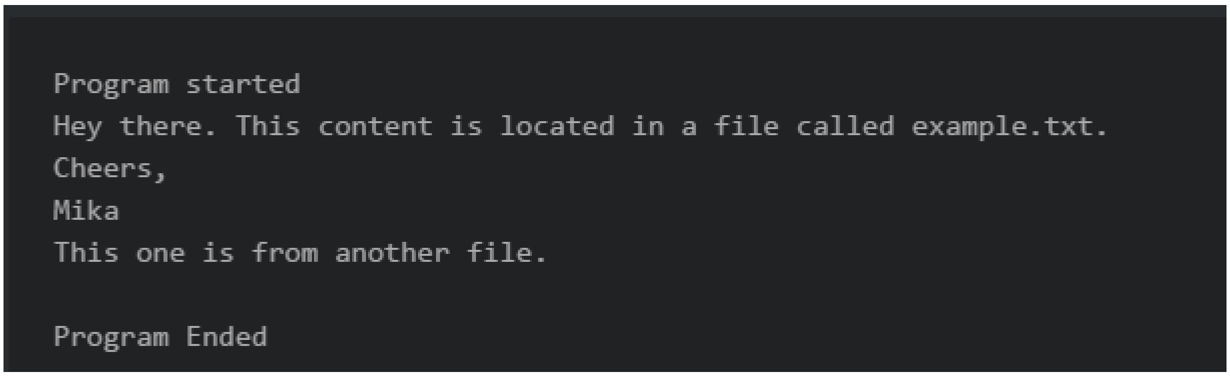
About Docs

It's important for Node.js to provide documentation to its users, but documentation means different things to different people. Here, on nodejs.org, you will find three types of documentation, reference documentation, ES6 features, and frequently asked questions.

Our API reference documentation is meant to provide detailed version information about a given method or pattern in Node.js. From this documentation you should be able to identify what input a method has, the return value of that method, and what, if any, errors may be related to that method. You should also be able to identify which methods are available for different versions of Node.js.

Read, write and delete text files

1. In the teachers presentation, you were shown how to read a text file and display the contents on console. Try this on your own, create a program “readingfiles.js” that will read one text file display the contents contents on the console.



```
Program started
Hey there. This content is located in a file called example.txt.
Cheers,
Mika
This one is from another file.

Program Ended
```

2. Modify the program so, that it will read two textfiles, and output both on the console.

3. Writing files is just as easy. It is done using the fs.writeFile –function. Create a file “combiningfiles.js” and utilize writeFile –function to it, so that it will write the text files of two files into a single new file. See the syntax and how to use writeFile from Node.js API.

4. When this works, try adding the string “I wrote this!” at the top and the bottom of the new textfile. Hint: see API for “append” related file functions.

5. Finally create a program “deletingfiles.js” which will delete the textfile you created on task 4. Use the unlink –function. See how to use it from Node.js API.

6. Create program “readingdir.js” Try to use readdir() function. Can you output a directory contents to the screen?

7. Create program “directories.js” Try mkdir and rmdir; when writing the files in step 3 first create a new folder called “newdata” and then write the file there.

Serve files to the browser

1. Last week you learned how to create a simple web server. Create a new app “routeswithactions.js” containing a simple web server outputting “hello world”.

2. When this works, modify the app a bit. Your app should respond to different routes by serving different content to the browser. This can be done by studying the request.url –parameter (remember last weeks workshop)

3. The route / should output the text “Nothing here to see” to the browser.

The route /frontpage should read a local HTML file frontpage.html and output the contents to the browser. (get sample file from https://pastebin.com/mmN3YtKK)

The route /contact should read a local HTML contact.html file and output the contents as HTML to browser (you can see sample file here: https://pastebin.com/fH6UBa4g)

The route /plaintext should read a local TXT file and output the contents as TXT to browser

The route /json should read a local JSON file and output the contents as JSON to browser (sample JSON data to a new file "sampledata.json)

Notice that your app should send different HTTP header with "Content-type" for all the different responses. How would achieve this?

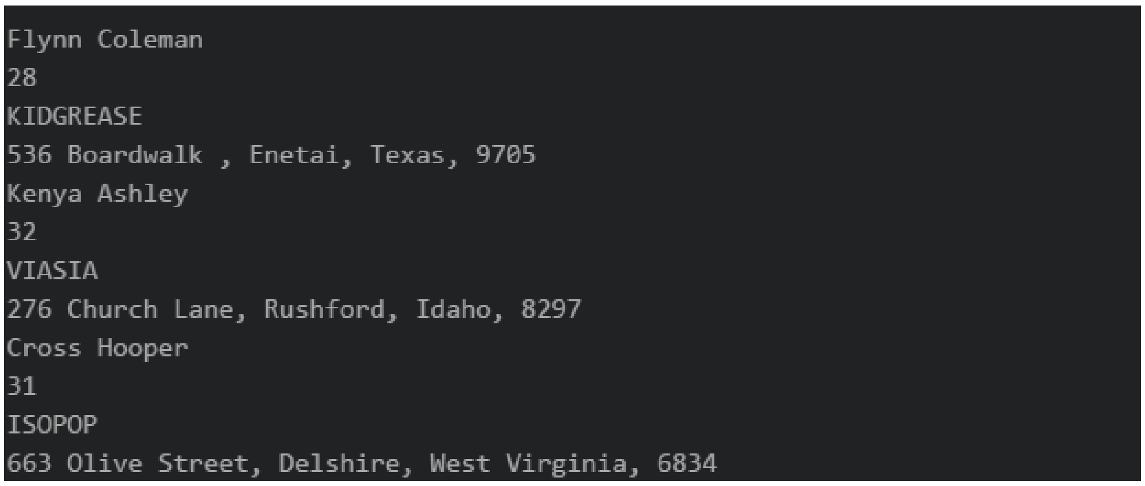
Read and process JSON files locally

Use lecture notes as a guide. There is plenty of tutorials available in the Internet.

8. Often times Online JSON formatters make reading API responses much easier. Try this: https://jsonformatter.curiousconcept.com/. If you save the data as a local file in VSCode / Atom, it will be nicely formatted as well.

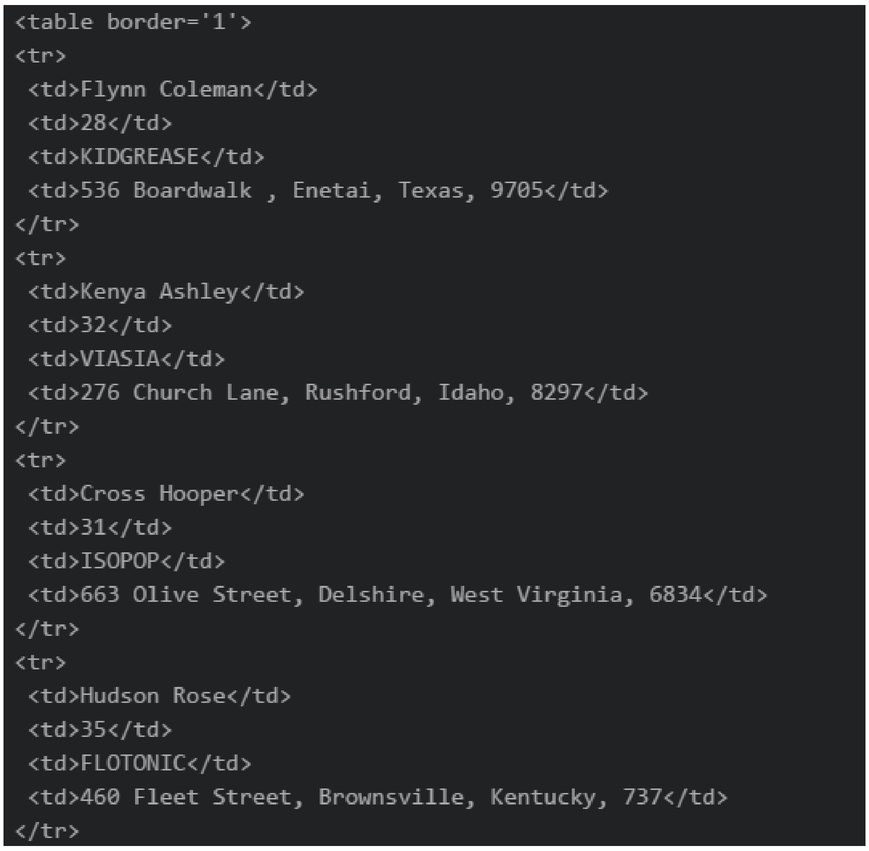
9. Then write a command line program which reads the "sampledata.json" data and does the following things:

a) iterates through the data and displays name, age, company and address data on the console.



```
Flynn Coleman
28
KIDGREASE
536 Boardwalk , Enetai, Texas, 9705
Kenya Ashley
32
VIASIA
276 Church Lane, Rushford, Idaho, 8297
Cross Hooper
31
ISOPOP
663 Olive Street, Delshire, West Virginia, 6834
```

b) same as on task a , but surround the data with HTML-tags.



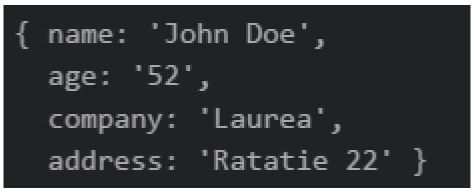
```
<table border='1'>
<tr>
 <td>Flynn Coleman</td>
 <td>28</td>
 <td>KIDGREASE</td>
 <td>536 Boardwalk , Enetai, Texas, 9705</td>
</tr>
<tr>
 <td>Kenya Ashley</td>
 <td>32</td>
 <td>VIASIA</td>
 <td>276 Church Lane, Rushford, Idaho, 8297</td>
</tr>
<tr>
 <td>Cross Hooper</td>
 <td>31</td>
 <td>ISOPOP</td>
 <td>663 Olive Street, Delshire, West Virginia, 6834</td>
</tr>
<tr>
 <td>Hudson Rose</td>
 <td>35</td>
 <td>FLOTONIC</td>
 <td>460 Fleet Street, Brownsville, Kentucky, 737</td>
</tr>
```

10. c) Create a web server and output the data as HTML to the browser.

Process JSON files

1. Create a new program “readingjson.js” which will read the sample JSON data to a variable. Output the data to the console, just to see its there.

2. Create a new variable in the code as below and add this new item to the JSON variable (See PowerPoint for push method). Output the variable to the console to check it is added. Then write the new file to the disk as “dataset.json”. Open the file to see if the added line is really there.



```
{ name: 'John Doe',
  age: '52',
  company: 'Laurea',
  address: 'Ratatie 22' }
```

3. Delete one item from the JSON file using JavaScript code. Output the data and see that it’s really gone.

4. Output the JSON data to the web browser as plain text. Notice, that your response content-type needs to be set set to “text/json”.

5. **Push (upload) all the Exercises completed here to you GitHub-account.**