**Lab 8**

Table of Contents

[Backend file manipulation 2](#_Toc97027220)

[Read a file: sync way 2](#_Toc97027221)

[Read a file: async way 2](#_Toc97027222)

[Read a file: as a module 3](#_Toc97027223)

[Error handling 4](#_Toc97027224)

[try … catch 4](#_Toc97027225)

[Variations 5](#_Toc97027226)

[Throwable in JS 5](#_Toc97027227)

[How to handle different type of exceptions 5](#_Toc97027228)

[JSON 6](#_Toc97027229)

[Write into a file 7](#_Toc97027230)

[Project settings: package.json 7](#_Toc97027231)

[To update your node\_modules folder 8](#_Toc97027232)

[To add new dependency into package.json 8](#_Toc97027233)

[To remove new dependency into package.json 9](#_Toc97027234)

[Listing dependencies 9](#_Toc97027235)

[To define your own script 9](#_Toc97027236)

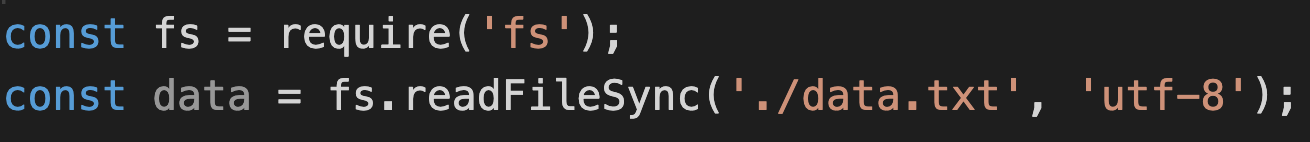
[Bonus topic: Frontend file retrieve 9](#_Toc97027237)

[Rendering Values from a JSON File 9](#_Toc97027238)

## Backend file manipulation

### Read a file: sync way

At backend, we may use fs.readFileSync() to read/load a local file (a file on the same server), like this:



Note: we should consider the case of failure to read, we will cover this in details later.

Exercise #1:

To make a node project (app.js only backend) read a file (my\_data.txt) synchronously and then print the contents to the console

my\_data.txt

|  |
| --- |
| Hello world from the .txt file. |

app.js

|  |
| --- |
| const fs = require('fs');  const data = fs.readFileSync('./my\_data.txt', 'utf-8');  console.log(data); |

Your screenshot to show it works:

|  |
| --- |
| A screenshot of a computer screen  Description automatically generated |

### Read a file: async way

At backend, we see async way to read/load a file in earlier session

fs.readFile("index.html", function (err, contents) {

if (!err) {

response.writeHead(200, { "Content-Type": "text/html" });

response.write(contents);

response.end();

}

});

Exercise #2:

To make a node project (web application: app.js server-side) read a file (index.html) asynchronously and then send the contents of .html file back to user browser.

app.js

|  |
| --- |
| var http = require('http');  var fs = require('fs');  http.createServer(function(request, response) {      fs.readFile("index.html", function(err, contents) {          response.writeHead(200, { "Content-Type": "text/html" });          response.write(contents);          response.end();      });  }).listen(8080); |

index.html (you may make your own contents, such as hello world, with HTML syntax)

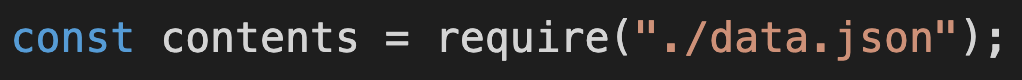
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta http-equiv="X-UA-Compatible" content="IE=edge">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <p>Execercise 2 prof!:)</p>  </body>  </html> |

Your screenshot to show it works:

|  |
| --- |
| Graphical user interface, text, application, Word  Description automatically generated |

### Read a file: as a module

At backend, we can read/load a file just simply use require as follow:



data.json

|  |
| --- |
| {  "name": "Bob",  "age": 23  } |

index.js

|  |
| --- |
| const contents = require("./data.json");  console.log(contents.name + ' is ' + contents.age); |

The json file could be more complex, such as:

data.json

|  |
| --- |
| {  "name": "Bob",  "age": 23,  "obj": {  "id": 100,  "obj2": {  "subject": "node.js"  }  }  } |

We need to update our logic to retrieve, like:

index.js

|  |
| --- |
| const contents = require("./data.json");  console.log(contents.name + ' is ' + contents.age);  console.log(contents.name + ' is learning ' + contents.obj.obj2.subject); |

We will get the listed result, when we run it with: node index

Text

Description automatically generated

Exercise #3:

To make a node project (web application: app.js server-side) read a file (my\_data.json) using require(), and then send the contents of .json file back to user browser.

app.js

|  |
| --- |
| var http = require('http');  const contents = require("./data.json");  http.createServer(function(request, response) {      response.writeHead(200, { "Content-Type": "text/html" });      response.write(contents.name + ' is ' + contents.age + " and learning " +          contents.obj1.subject + ',' + contents.obj2.subject + ',' +          contents.obj3.subject + ' this term!');      response.end();  }).listen(8080); |

my\_data.json (You may make your own contents, with JSON syntax)

|  |
| --- |
| {      "name": "Alisher",      "age": 20,      "obj1": {          "id": "comp3012",          "subject": "node.js"      },      "obj2": {          "id": "comp4669",          "subject": "Advanced web applications"      },      "obj3": {          "id": "mathh1060",          "subject": "Stats for data analysis"      }  } |

Your screenshot to show it works:

|  |
| --- |
| Graphical user interface, text, application  Description automatically generated |

## Error handling

Document :

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch>

### try … catch

The **try...catch** statement marks a block of statements to try and specifies a response should an exception be thrown.

Graphical user interface, text, application

Description automatically generated

**try\_statements**

The statements to be executed.

**catch\_statements**

Statement that is executed if an exception is thrown in the try-block.

**exception\_var**

An optional identifier to hold an exception object for the associated catch-block.

**finally\_statements**

Statements that are executed after the try statement completes. These statements execute regardless of whether an exception was thrown or caught.

### Variations

The try statement consists of a try-block, which contains one or more statements. {} must always be used, even for single statements. A catch-block, a finally-block, or both must be present. This gives us three forms for the try statement:

* try...catch
* try...finally
* try...catch...finally

A catch-block contains statements that specify what to do if an exception is thrown in the try-block. If any statement within the try-block (or in a function called from within the try-block) throws an exception, control is immediately shifted to the catch-block. If no exception is thrown in the try-block, the catch-block is skipped.

The finally-block will always execute after the try-block and catch-block(s) have finished executing. It always executes, regardless of whether an exception was thrown or caught.

You can nest one or more try statements. If an inner try statement does not have a catch-block, the enclosing try statement's catch-block is used instead.

You can also use the try statement to handle JavaScript exceptions. See the [JavaScript Guide](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling#utilizing_error_objects) for more information on JavaScript exceptions.

### Throwable in JS

We could throw a string:

|  |
| --- |
| try {  throw "a new exception"; // generates an exception  } catch (e) {  console.log(e);  } |

The catch-block specifies an identifier (e in the example above) that holds the value of the exception; this value is only available in the [scope](https://developer.mozilla.org/en-US/docs/Glossary/Scope) of the catch-block.

### How to handle different type of exceptions

We may use instanceof keyword combined with if … else statement to check the types of exceptions:

|  |
| --- |
| try {  doSomething(); // may throw three types of exceptions  } catch (e) {  if (e instanceof TypeError) {  // statements to handle TypeError exceptions  } else if (e instanceof RangeError) {  // statements to handle RangeError exceptions  } else if (e instanceof EvalError) {  // statements to handle EvalError exceptions  } else {  // statements to handle any unspecified exceptions  logMyErrors(e); // pass exception object to error handler  }  } |

## JSON

NPM uses JSON file (package.json) for configuring modules.

JSON is a standard format for transferring data over the network, and it is used in many runtimes beyond JS.

JSON is very similar to JavaScript object literals with some exceptions: one of the differences is for JSON the keys must use quotation marks. Value-wise, JSON can use string, boolean, array, null, or other valid JSON object.

Node has full support for the JSON format: **A JSON object can be loaded from the file system**. As a result Node.js relies on JSON file for configuration mechanisms

JSON global object has two useful functions:

* *JSON.stringify()* can be usedto convert a JS object to a string
* *JSON.parse()* can be used to parse a JSON string to JS object

Exercise #4:

To make a node project (app.js only backend) read the given file (my\_data.json), increase each value by 1, and then print it out as a string. [Hint: need to use stringify() and parse()]

my\_data.json

|  |
| --- |
| {  ‘a’: 11,  ‘b’: {  ‘c’: 22,  ‘d’: [33],  ‘e’: {  ‘f’: 44  }  }  } |

app.js

|  |
| --- |
| const fs = require('fs');  //var contentsD = JSON.parse(contents.b.d);  fs.readFile('./my\_data.json', 'utf-8', (error, contents) => {      if (error) {          console.log(error);      } else {          var temp = JSON.parse(contents);          var tempD = JSON.parse(temp.b.d);          temp.a += 1;          temp.b.c += 1;          temp.b.d = tempD + 1;          temp.b.e.f += 1;          console.log(temp);      }  }); |

Your screenshot to show it works:

|  |
| --- |
| A screenshot of a computer  Description automatically generated with medium confidence |

## Write into a file

Partner with fs.readFile(), we may use fs.writeFile() to write contents into a file asynchronously:

Text

Description automatically generated

Exercise #5:

To make a node project (app.js only backend) read the given file (my\_data.json), increase each value by 1, and then write it into a file called my\_new\_file.json.

my\_data.json

|  |
| --- |
| {  ‘a’: 11,  ‘b’: {  ‘c’: 22,  ‘d’: [33],  ‘e’: {  ‘f’: 44  }  }  } |

app.js

|  |
| --- |
| const fs = require('fs');  //var contentsD = JSON.parse(contents.b.d);  fs.readFile('./my\_data.json', 'utf-8', (error, contents) => {      if (error) {          console.log(error);      } else {          var temp = JSON.parse(contents);          var tempD = JSON.parse(temp.b.d);          temp.a += 1;          temp.b.c += 1;          temp.b.d = tempD + 1;          temp.b.e.f += 1;          fs.writeFile('./my\_new\_file.json', JSON.stringify(temp, null, 2), error => {              if (error) {                  console.log(error);              }          });      }  }); |

Your screenshot to show it works:

|  |
| --- |
| A screenshot of a computer  Description automatically generated with medium confidence |

We also may write into a file synchronously, using fs.writeFileSync()

## Project settings: package.json

We may set up a node project by running command “npm init” in the folder. Running this command will populate a setting file called package.json. Below is an example of package.json file.

Text

Description automatically generated

### To update your node\_modules folder

We may run the listed command to refresh our node\_modules:

npm install

This is how I run your projects on my machine.

### To add new dependency into package.json

This is how to install a package and add it into setting file as a production dependency:

npm install <package-name> --save-prod

OR:

npm install <package-name>

This is how to install a package and add it into setting file as a dev dependency:

npm install <package-name> --save-dev

This is how to install a package and add it into setting file as a dependency for both:

npm install <package-name> --save

After running any of above command, you would see this in your package.json:

* "dependencies": Packages required by your application in production.
* "devDependencies": Packages that are only needed for local development and testing.

We may also add dependencies to a package.json file, in a text editor, add an attribute called "dependencies" that references the name and [semantic version](https://docs.npmjs.com/about-semantic-versioning) of each dependency:

|  |
| --- |
| {  "name": "my\_package",  "version": "1.0.0",  "dependencies": {  "my\_dep": "^1.0.0",  "another\_dep": "~2.2.0"  }  } |

### To remove new dependency into package.json

This is how to remove a package and add it into setting file as a dependency for both:

npm uninstall <package-name> --save

**Note:** If you installed a package as a "devDependency" (i.e. with --save-dev), use --save-dev to uninstall it:

npm uninstall <package-name> --save-dev

### Listing dependencies

To check what packages have been installed in the project, we may use:

npm ls

### To define your own script

We may define customized command in the scripts section of package.json.

## Bonus topic: Frontend file retrieve

JSON is a file format widely used for static storage and app config management with any of the frameworks and data servers. Any JSON file contains the key-value pair separated by the comma operator. JavaScript objects are an integral part of the React app, so they need to get accessed from JSON files/data to be uses in components.

This guide will demonstrate how to get a JavaScript object from a JSON file or access it using a fetch() HTTP request.

### Rendering Values from a JSON File

Any JSON data can be consumed from different sources like a local JSON file by fetching the data using an API call. After getting a response from the server, you need to render its value.

You can use local JSON files to do an app config, such as API URL management based on a server environment like dev, QA, or prod.

Create one sample JSON file as given below, and save it as data.json

Text

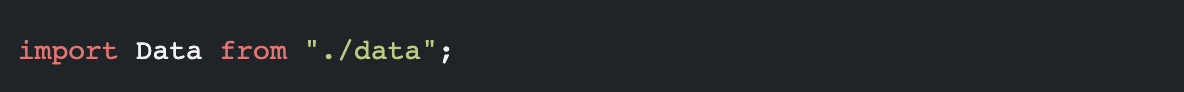
Description automatically generated

Now, if you want to render any of the key-value pairs from the JSON, the .map() function would be useful to iterate the objects; the example is below.

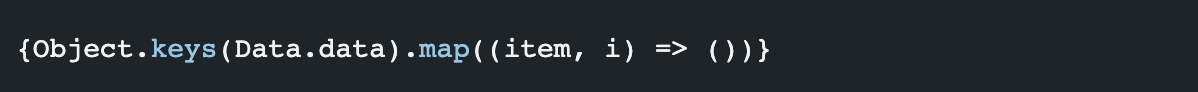
Text

Description automatically generated

In the above example, to use the local JSON file needs to be consumed using the import statement.



After that, you can access all the JSON data using Data in your component by using Object.keys() along with the .map() function.



Using a local JSON file in the React app is a common approach when you want to render some static data, maintain server config, etc.