Tutorial 6: NodeJS (ExpressJS)

Objectives

- Setting up NodeJS
- Your first ExpressJS server
- Creating API end-points with ExpressJS
    - Creating **RESTful CRUD** end-points
    - Sending data to server
        - Route params
        - Body message
        - Query params

Tutorial Exercises

In this tutorial, you will **work in pairs** to solve these exercises.

Create folder `tut05/`, and two sub-folders `/nodejs & /expressjs` and complete the tasks below.

Activity 1: Hello NodeJS (15mins)

In folder `/nodejs`, create file `server.js` and:

Create NodeJS server & run the Hello World program yourself.

For example: http://localhost:3000

```
const http = require ('http');

const server = http.createServer();

server.on('request', function(req, res) {

    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Hello World!');
```

```
})

server.on('listening', function() {
    console.log('Server running!');
});

server.listen(3000);
```

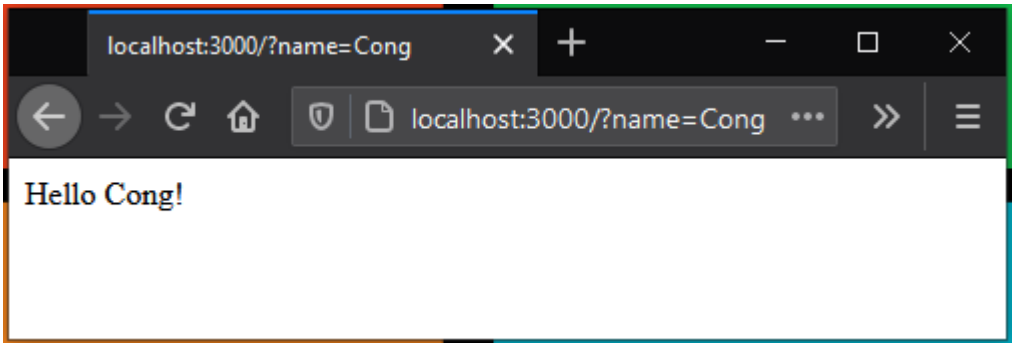
Run your server using the command:

```
node server.js
```

Activity 2: Feeling the pain of NodeJS repeating tasks (15mins)

Discussion 1: NodeJS routes problem

How to serve path `/hello` with query param `name` then print on the result page 'Hello [name]'? For example:



Hint: See URL module https://www.w3schools.com/nodejs/nodejs_url.asp

```
const url = require('url');
//…
var q = url.parse(req.url, true);
if (q.pathname === '/') {
    res.end('Hello World!');
}
if (q.pathname === '/hello') {
    res.end('Hello '+q.query.name+'!');
}
```

How about flashcards - all cards (GET)/ add card (POST) in NodeJS?

```
const url = require('url');
//…
```

```
var q = url.parse(req.url, true);
if (res.method === 'GET') {
    //…
} else if (res.method === 'POST') {
    //…
}
```

Task 1: ExpressJS routes

In folder `/expressjs`, create file `server.js` and:

Create ExpressJS server & run the Hello World program yourself.

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

const app = express();

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

app.listen(3000, function(){
    console.log('Listening on port 3000!');
});
```

Make sure to have express installed with

```
npm install express
```

Use **express** to create 4 routes (Hello World, Hello Name, GET All flashcards, POST Add new Flashcard that solve the problem from *Discussion 1*.

Note: not to complete the body of route handler functions for flashcards (later)

In this task, express need to work with query params.

Hint: In express, accessing query parameters using **req.query**.

Discussion 2: NodeJS serve static files problem

How about serving **static files** (HTML, CSS, IMAGES, etc) in NodeJS?

Hint: Section *Node.js File server* in https://www.w3schools.com/nodejs/nodejs_url.asp

Task 2: Express serve static files

Use **express** to serve static files (HTML, CSS, IMAGES, etc) in folder `/public`.

```
const app = express();

app.use(express.static('public'));
```

Exercise 1: ExpressJS Basic CRUD (30 mins)

In previous tutorial you fetch & manipulate with the WORDS data from the provided API. Now, it's your task to create your own server APIs using **ExpressJS**.

CRUD stands for:

- [C]reate create a new word
- [R]etrieve all words
- [U]pdate a specified word
- [D]elete a specified word

In `expressjs/server.js` file, declare an object of WORDS with any words that you like, for example:

```
const WORDS = {
    '네': 'yes',
    '아니요': 'no'
};
```

Note: word is unique (distinct), in this case, it is used as the primary key

Task 1: [R]etrieve all words

GET /words

Request:

- **Data**: none

Response:

- **Status**: 200 (OK)

- **JSON data**: an object of words in format of {word: definition, …}

Use-case:

- **Client**: send request to get all words
- **Server**: return object WORDS (as mentioned above) as result

Example:

Request	Response
Data: none	'{ "네": "yes", "아니요": "no" }'

Hint: in express, to return result as JSON using **res.json(JS object)**;

Task 2: [C]reate a new word

POST /words

Request:

- **Data**: JSON body contain word & definition

  eg. {"word": "네", "definition": "Yes"}

Response:

- **Status**: 201 (CREATED)
- **JSON data**: an object of just added word in format of {word: definition}

// if word already exists

- **Status**: 409 (CONFLICT)

- **Data**: none

Use-case:

- **Client**: send request with data to store a new word with given definition
- **Server**:
    - check if word already exists → return status 409 (CONFLICT)
    - save user provided word & definition into WORDS (as mentioned above) & return just created object {word: definition} as data with status 201 (CREATED)

Example:

Request	Response
Data: `'{ "word": "네", "definition": "yes" }'`	`'{ "네": "yes" }'`

Exercise 2: (OPTIONAL) ExpressJS Basic CRUD (cont)

Similarly, you can complete other routes to complete CRUD for words (support complete functions for Flashcards app)

Task 1: [U]pdate specified word

PUT /words/:word

Request:

- **Data**:
    - Route param:
        - :word: the word to update
    - JSON body contain new definition

eg. { “definition”: “Yes”}

Response:

- **Status**: 200 (OK)
- **JSON data**: an object of just updated word in format of {word: definition}

// if word does NOT exist

- **Status**: 404 (NOT FOUND)

- **Data**: none

Use-case:

- **Client**: send request with data to update the given word with new definition
- **Server**:
    - check if word does not exist → return status 404 (NOT FOUND)
    - update user provided word & definition into WORDS (as mentioned above) & return just updated object {word: definition} as data with status 200 (OK)
- *Example*:

Request	Response
PUT /words/ㄴ| Data: `'{ "definition": "yes" }'`	`'{ "ㄴ|": "yes" }'`

Task 2: [D]elete specified word

DELETE words/:word

Request:

- **Data**:
    - Route param:
        - :word: the word to update

Response:

- **Status**: 204 (NO CONTENT)
- **JSON data**: an object of just deleted word in format of {word: definition}

// if word does NOT exist

- **Status**: 404 (NOT FOUND)

- **Data**: none

Request	Response
DELETE /words/네	'{ "네": "yes" }'