Tutorial 7: NodeJS & MongoDB (1)

Objectives

In this tutorial, we will (1) continue to complete & improve our code on frontend to

- Using *async/ await* instead of *Promise .then()*
- Sending data to server using:
  - Route params
  - Post body message

& (2) get started with MongoDB

- Manipulating with MongoDB using the command-line tool

Activities

Download starter_pack, extract then run & start our activities below.

Activity 1: Form normal submission (15 mins)

```
function onUpdate(event) {
    event.preventDefault();
    // …
}

const formUpdate = document.querySelector('#form-update');
formUpdate.addEventListener('submit', onUpdate);
```

What is *event.preventDefault()* ???

Have a look on *public/index.html* then run the server then browse *localhost:8080* or *locahost:8080/index.html* to see the result on the browser. Input 'cat' & hit 'Search', you will see a form to update this word in our dictionary. The search function work normally but update does not.

- Click 'Save' and observe what happen. Explain why?
- Delete action then observe what happen. Explain why?

- Change action to */js/update.js* & observe what happen. Explain why?
- Change action to /update & observe what happen. Explain why?

Activity 2: Form GET vs POST (15 mins)

What are differences between GET & POST???

Complete Activity 1, then

- Pay attention on the url with params. Where are they from?
- How about the names? Delete one of them then re-run and observe what happen. So how to send data with a form?
- Change method to 'POST' then re-run and observe what happen. So did data send to server?

Tutorial Exercises

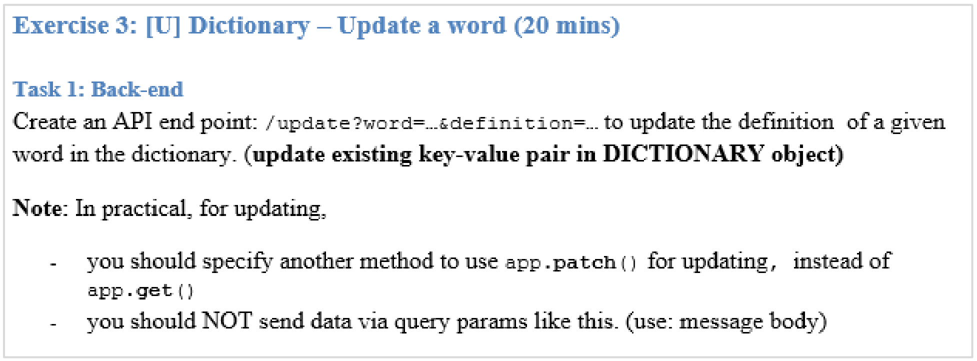
Exercise 1: Async/ Await (15 mins)

Refactor code of our two functions: lookup & update a word in dictionary to use *fetch()* with *async/ await*.

Exercise 2: Route params & Post body message (20 mins)

In the function */update*,

- Refactor to use *route param* instead of *query param*
- Note that in the previous tutorial, we highlighted:



Exercise 3: [U] Dictionary – Update a word (20 mins)

Task 1: Back-end

Create an API end point: `/update?word=…&definition=…` to update the definition of a given word in the dictionary. (**update existing key-value pair in DICTIONARY object)**

Note: In practical, for updating,

- you should specify another method to use `app.patch()` for updating, instead of `app.get()`
- you should NOT send data via query params like this. (use: message body)

Refactor to use message body instead.

Hint: you will need to use *body-parser* library.

Exercise 3: MongoDB command-line (20 mins)

In this exercise, we aim to create the database for our dictionary web application. Start the mongodb command-line then:

- Name all databases that your server has.
- Switch to use database with name: *eng-dict* (created automatically if db name not exist)
- Name all collections that this db has.
- Add a new word into *words* collection: *{word: 'dog', definition: 'friend'}*
- Add a new word into *words* collection: *{word: 'cat, definition: 'boss'}*
    - Query all *words* to check if success inserted.
- Add some more words (at least 5)
- Query for definition of the word: '*dog*'
- Update definition of the word '*dog*' from '*friend*' into '*woof woof*'
    - Query all *words* to check if success inserted.
- Set all words to have definition: '*empty: to-update*'
- Delete the word 'dog' from the words collection
    - Query all *words* to check if success inserted.
- Delete all words from collection *words*.
    - Query all *words* to check if success inserted.
- Delete the collection *words* from database
    - Name all collection that this db has to check if success.

Exercise 4: Dictionary – add or delete a word (Homework)

Base on the improved version of code for Exercise 1 and 2, complete the functions:

- Add a new word into the dictionary
- Delete given word (and its definition) from the dictionary

For further requirements, see previous tutorial: *Tutorial 06.*