Tutorial 8: NodeJS & MongoDB (2)

Objectives

In this tutorial, we will bring up our Flashcards app APIs to the next version with use of MongoDB to store words. We focus on practicing:

- Using *mongodb* driver to work with Mongo database in NodeJS
- Fetching query result for responses from server API.
- Using `package.json` to manage dependencies

Tutorial Exercises

Recall: in the previous tutorial, you completed both the backend api & frontend web app (partial) for the functions of the Flashcards application; However, all the data is now storing in a JavaScript object on the memory. This causes problems:

(1) The memory overload with the increasing number of words,
(2) All the data changes will be lost (reset) if we restart our server.

All these problems will be solved using Mongo database.

Download the **Tut 07: Solution**, extract and rename to `tut08/flashcards-mongodb/`, run application and complete the exercises below.

Exercise 1: Use `package.json` (15 mins)

Generate `package.json` file to manage dependencies of this project.

```
npm init -y
```

Open the generated file `package.json` & have a look.

Exercise 2: Use Mongo database with NodeJS (15 mins)

This exercise aims to use the `mongodb` driver to let NodeJS to work with Mongo database.

Make sure that your computer has MongoDB working. You can test with MongoDB command line (practiced in previous tutorial)

Task 1: install mongodb driver

```
npm install mongodb
```

Task 2: connect to Mongo database

Connect to MongoDB on server start. Below is example code from the lecture:

```
const DATABASE_NAME = 'eng-dict2';
const MONGO_URL = `mongodb://localhost:27017/${DATABASE_NAME}`;

let db = null;
let collection = null;

async function startServer() {
  // Set the db and collection variables before starting the server.
  const client = await mongodb.MongoClient.connect(MONGO_URL);
  db = client.db();

  collection = db.collection('words');
  // Now every route can safely use the db and collection objects.
  await app.listen(3000);
  console.log('Listening on port 3000');
}
startServer();
```

Exercise 3: [R] Flashcards – All words (10 mins)

Refactor to use MongoDB *find()* to get all words then return from database instead. Below is example code from the lecture:

```
async function printAllWords() {
  const results = await collection.find().toArray();

  for (const result of results) {
    console.log(`Word: ${result.word}, definition: ${result.definition}`);
  }
}
```

Exercise 4: [U] Flashcards – Update definition of specified word (10 mins)

Refactor to use MongoDB *updateOne()* to update the word from database instead. Below is example code from the lecture:

```
async function onSetWord(req, res) {
  const routeParams = req.params;
  const word = routeParams.word.toLowerCase();
  const definition = req.body.definition;

  const query = { word: word };
  const update = { word: word, definition: definition };
  const params = { upsert: true };
  const response =
      await collection.update(query, update, params);

  res.json({ success: true });
}
app.post('/set/:word', onSetWord);
```

Exercise 5: [C] Flashcards – Add a word with definition (10 mins)

Refactor to use MongoDB *insertOne()* to add new word into the database instead.

Exercise 6: [D] Flashcards – Delete a given word (10 mins)

Refactor to use MongoDB *deleteOne()* to delete the given word from database instead.

Exercise 7: Use ObjectID (20 mins)

It's can be seen that we are using word as the primary key for update/ delete.

Refactor the code to use `ObjectID` (primary key – auto generated value from MongoDB)