Tutorial 8: NodeJS & MongoDB (2)

Objectives

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

- Using *mongodb* library to work with Mongo database in NodeJS
- Fetching query result for responses from server API.

Tutorial Exercises

Recall: in the previous tutorial, you completed both the backend api & frontend web app for the functions of the Dictionary 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.

In the lecture, we covered lookup & update functions to work with database. You will work on these functions again for more understanding (Ex 1-3), then apply into the next functions, including create & delete (Ex 4-5). Exercise 6 is to extend with use of Handlebars template engine to render view.

Download the **starter_pack** and rename to `tut08/dictionary/`, run application and complete the exercises below.

Exercise 1: Use mongodb with NodeJS (15 mins)

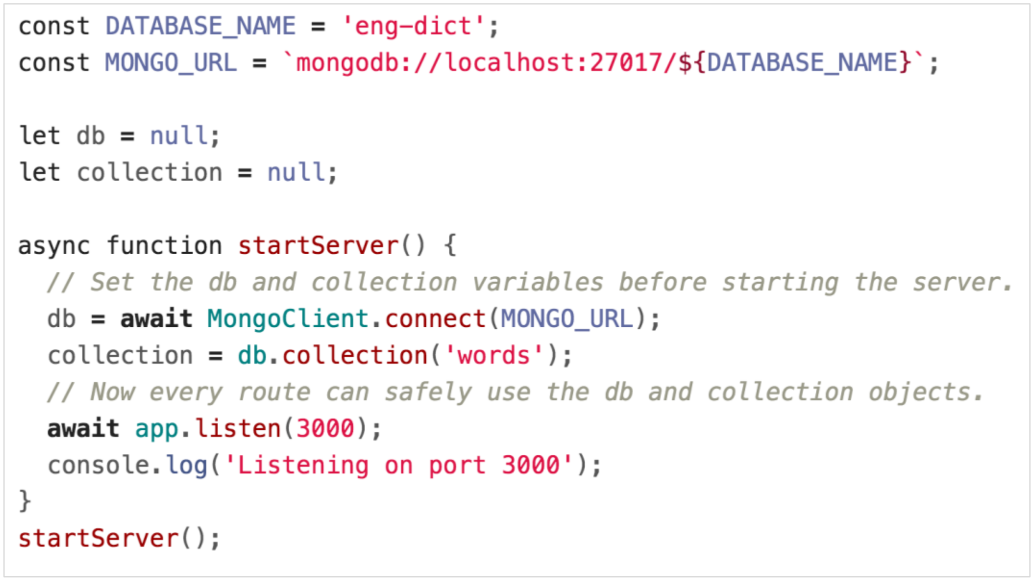
This exercise aims to use the mongodb library to let NodeJS to work with Mongo database.

Task 1: install library mongodb

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

Task 2: connect to mongodb

Make connection with mongodb on server start. Below is example code from the lecture:



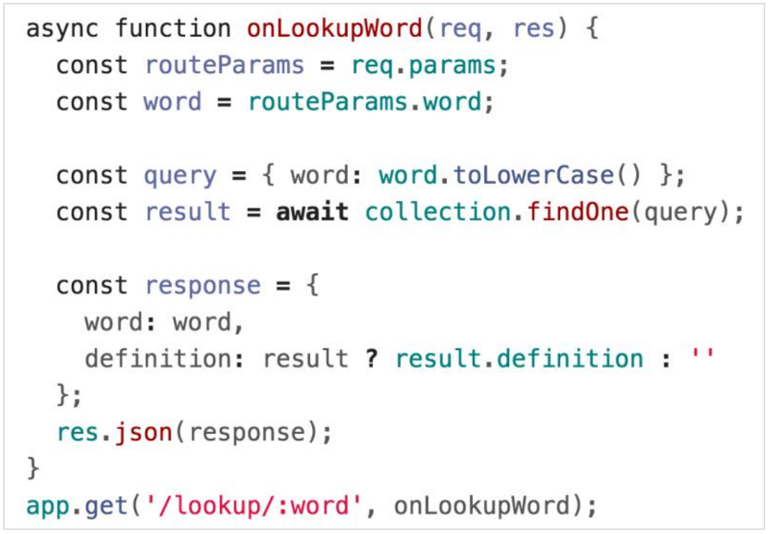
```
const DATABASE_NAME = 'eng-dict';
const MONGO_URL = `mongodb://localhost:27017/${DATABASE_NAME}`;

let db = null;
let collection = null;

async function startServer() {
  // Set the db and collection variables before starting the server.
  db = await MongoClient.connect(MONGO_URL);
  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 2: [R] Dictionary – Look up (10 mins)

Refactor to use mongodb *findOne()* to lookup the word from database instead. Below is example code from the lecture:



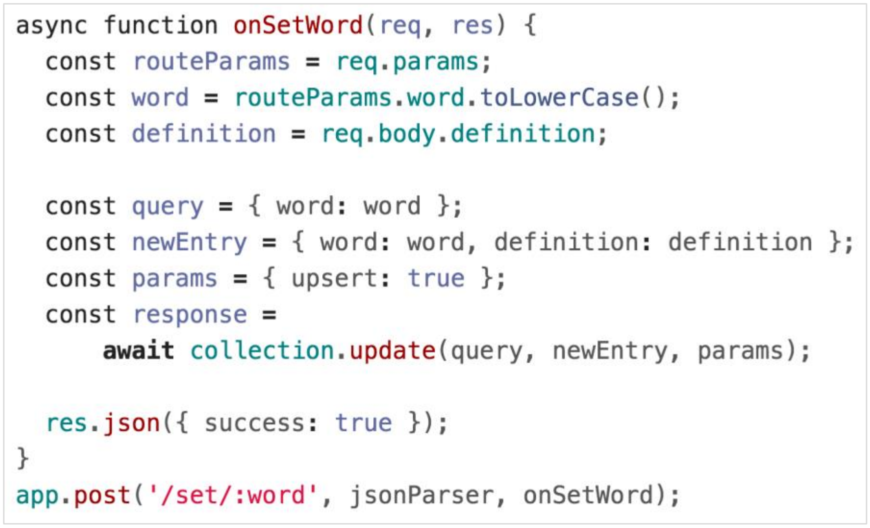
```
async function onLookupWord(req, res) {
  const routeParams = req.params;
  const word = routeParams.word;

  const query = { word: word.toLowerCase() };
  const result = await collection.findOne(query);

  const response = {
    word: word,
    definition: result ? result.definition : ''
  };
  res.json(response);
}
app.get('/lookup/:word', onLookupWord);
```

Exercise 3: [U] Dictionary – Update definition (15 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 newEntry = { word: word, definition: definition };
  const params = { upsert: true };
  const response =
      await collection.update(query, newEntry, params);

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

Exercise 4: [C] Dictionary – Add a word with definition (30 mins)

Refactor to use mongodb *insertOne()* to lookup the word from database instead.

Exercise 5: [D] Dictionary – Delete a given word (10 mins)

Refactor to use mongodb *deleteOne()* to lookup the word from database instead.

Exercise 6: [D] Dictionary Admin – List all words (20 mins)

Read the READ MORE about Handlebars template engine & use it to create an admin page, to show a table of all words in our dictionary.