Tutorial 9: Server side rendering

NodeJS (end) & ReactJS (1)

Objectives

In this tutorial, we are (1) finishing NodeJS by solving the problems with *server.js* & (2) getting started with ReactJS.

In details, we focus on practicing:

- Using Handlebars template engine
- Using Node modules, router & middleware
- Creating & running a React application

Tutorial Exercises

Recall: in the previous tutorial, you completed the Flashcards application with **server JSON APIs and MongoDB**; As you can see, it's not normal to allow user to add/ update/ delete words directly. Then, we aim to create a ***login-less*** admin space (**server-rendering** web pages) to manage the words.

On the other words, creating a module for admin the dictionary `modules/admin.js`. This module contains all the routes & handlers for admin tasks, including:

- view a table of all words,
- view word details (*presented in the lecture*),
- add a new word,
- update a word &
- delete a word.

<u>**Note**</u> that, in this admin space we use ***server-rendering multi webpages***.

Download the **tut08_solution** extract and rename to `tut09/flashcards-admin/`, run application and complete the exercises below.

Exercise 1: Creating *admin* module (30 mins)

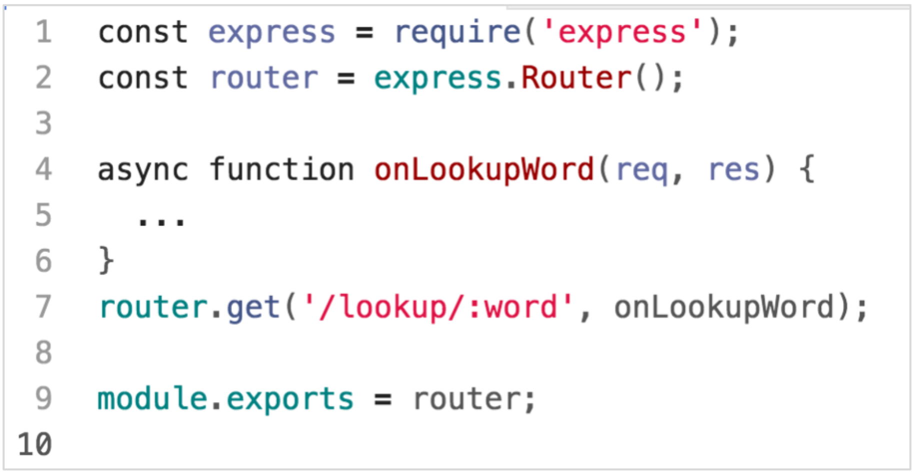
Problem: server.js now contains all the logic handling for our server. This is growing fast when we have more & mores functionalities → really hard to manage.

In this exercise, we create a separate file (module) for admin features.

Task 1: Export Routes (10 mins)

- Create the module file `modules/admin.js`.
- In this module, declare all the required routes & handler functions.
  For example, the "View a table of all words" will be named as below:
  - Route: */admin/words*
  - Handler: *index()*
- Export all routes using *express Router*.

For example, (1) to define routes for lookup function in module `api.js` & (2) to use it in express.

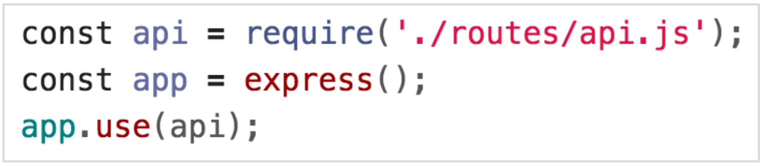


```
const express = require('express');
const router = express.Router();

async function onLookupWord(req, res) {
  ...
}
router.get('/lookup/:word', onLookupWord);

module.exports = router;

```



```
const api = require('./routes/api.js');
const app = express();
app.use(api);
```

Task 2: Using Handlebars template engine (10 mins)

- Install Handlebars & set it up to work with express.

```
const exphbs  = require('express-handlebars');
…

const app = express();
const hbs = exphbs.create();
app.engine('handlebars', hbs.engine);
app.set('view engine', 'handlebars');
```

- With all the handlers, return a ***handlebars*** view with name of the function.

Recall, **`res.render`**`(`***`viewName`***`, `***`placeholderDefs`***`)`: Returns the HTML stored in "`views/`***`viewName`***`.handlebars`" after replacing the placeholders, if they exist

```
function onGetMain(req, res) {
  res.render('index');
}
app.get('/', onGetMain);
```

Hint: You may be in error `layout main.handlebars` file not found. Layout is a very nice feature provided by the template engine where we can shared code for UI components between pages (e.g. header, navbar, footer). You need to create one OR just invoke `res.render` with `{layout: false}`

Task 3: Middleware – Access db in handlers (10 mins)

- In this case, you need to use mongo **`db`** object to manipulate with the database.
  Create a middleware to pass **`db`** to request as **`req.db`** (also for other handlers).

For example, middleware to pass the "*words*" collection for all requests:

```
async function startServer() {
  const client = mongodb.MongoClient.connect(MONGO_URL);
  const db = client.db();
  const collection = db.collection('words');
  // Adds the "words" collection to every MongoDB request.
  function setCollection(req, res, next) {
    req.collection = collection;
    next();
  }
  app.use(setCollection);
  app.use(api);

  await app.listen(3000);
  console.log('Listening on port 3000');
}
```

Note: we need to use the `modules/admin` router AFTER the middleware.

Exercise 2: View a table of all words (20 mins)

- Complete the handler function corresponding to this function, for example: *index()*
  - Query all `words` from database
  - Pass these `words` into view `views/admin/words/index.handebars`
- In the view, loop the `words` to populate as rows in the table, with 2 links to *update/ delete* at the end of each row.

  Also, a link to *add* a new word is required.

Note: you will need to use **`#each`** helper in handlebars:
http://handlebarsjs.com/builtin_helpers.html

Exercise 3: Add a new word (15 mins)

Similar to *Exercise 2*, complete the function: "*Add a new word*".

Note: after add word successfully, you should redirect user back to "View a table of all words".

https://expressjs.com/en/4x/api.html#res.redirect

Exercise 4-5: Update/ Delete a given word (15 mins)

Similar to *Exercise 3*, complete the function: "*Update a given word*" & "*Delete a given word*".

Exercise 6: Create react app (10 mins)

Create a new React application & run it is a required & good start to work with ReactJS. Follow the instructions from our lecture to create your own first React application.

- Install create-react-app by running this command in your terminal:

```
C:\Users\Your Name>npm install -g create-react-app
```

- Then you are able to create a React application, let's create one called *myfirstreact*.

```
C:\Users\Your Name>npx create-react-app myfirstreact
```

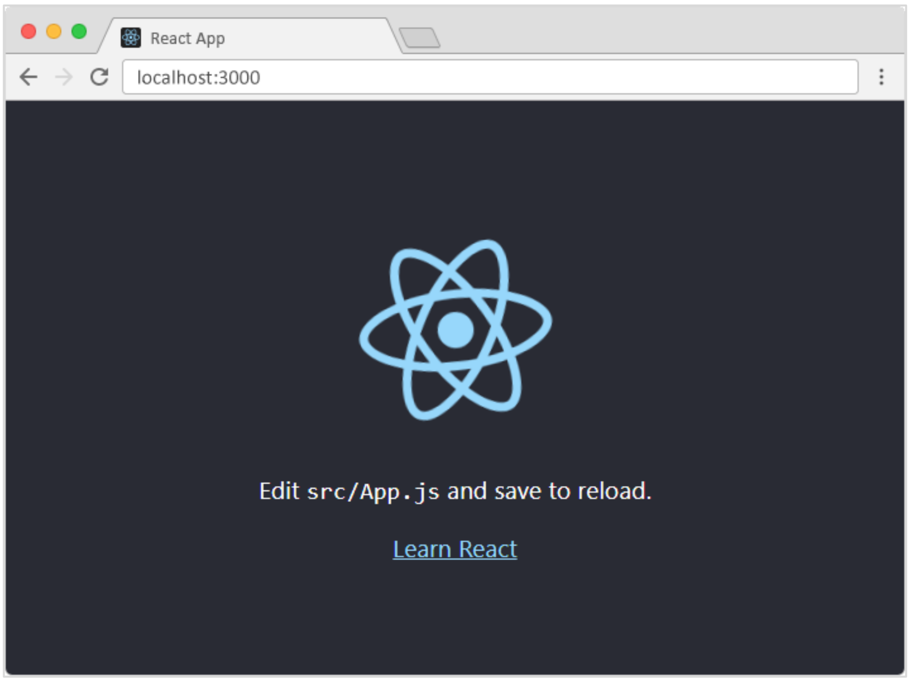
- Move to the *myfirstreact* directory

```
C:\Users\Your Name>cd myfirstreact
```

- Run application

```
C:\Users\Your Name\myfirstreact>npm start
```

You should see the result like this:



Edit src/App.js and save to reload.