Web Programming

Tutorial 7

To begin this tutorial, download `tut07-starter.zip` file and extract the `tut07` folder to a suitable location. Open the `tut07` folder in VSCode, open a terminal in VSCode and run `npm init` to create a Node.js project in this folder. When you finish writing codes for the activities, zip all your source codes (excluding the `node_modules` folder) to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `tut01`, `tut02`, `tut03`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

Quick Activity 1 – Exponents

Create the `exponents.js` file in the `tut07` folder. This file is an Express.js application which has a GET endpoint, `/math/power/:base/:exponent`. It should respond in JSON with the result of putting the `base` to the exponent `power`. If the optional query parameter `root` is set to `true`, it should also respond with the square root of the `base`.

Response with no query parameter set (http://localhost:8000/math/power/4/2);

```
{'result': 16}
```

With query parameter set (http://localhost:8000/math/power/4/2?root=true):

```
{'result': 16, 'root': 2}
```

Activity 2 – Login API

Use Express.js to build a login API in the `login-api.js` file. This API has the same functionality as the provided API in a previous tutorial. The link of the provided API:

`https://hanustartup.org/wpr/api/login.php`

Here's the API's documentation:

Service URL: `https://hanustartup.org/wpr/api/login.php`

Request method: POST

Response format: plain text

Body Parameters (2 params):

`user` : Obtained from the first input element for the username `password` : Obtained from the second input element for the password

The API you have to create should reside at: `http://localhost:PORT/login` (change PORT to your port of choice)

Requirements:

- Create a file named `users.json` to store user credentials. Put some users in there.
- In the root directory of your Node.js project, create a file named `login-api.js`.
- Set up an Express.js server to listen on a port of your choice (e.g., 3000).
- Read and parse the `users.json` file to get the user data.
- Create a POST endpoint at `/login` that accepts `application/x-www-form-urlencoded` data.
- The `API` should validate the `user` and `password` fields against the data in `users.json`.
- If the credentials match, respond with the text: `Login successful.`
- If the credentials do not match, respond with the text: `Invalid username or password.`

Note: Your client code `(HTML/CSS/JS)` is placed in the `public` folder of the project so that it will be served on localhost (by Express) when you use:

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

You don't need to worry about this folder. Use Postman to test your API in the mean time. You will complete the client side in the next activity.

Activity 3 – Login Client

The `/public/post.js` file interacts with the login API using Fetch API. Your JavaScript code should handle form submission, send a POST request to the backend, and display the server's response.

You must complete code in `post.js` file based on these requirements:

- **Form Handling**:

- Prevent the default form submission behavior using `e.preventDefault()`.

- Collect the `username` and `password` from the form inputs.

- **Send POST Request**:

- Use the Fetch API to send a `POST` request to `http://localhost:PORT/login` with the form data.

- Handle the response and display the result in a designated element on the page.

- **Error Handling**:

- Implement error handling to manage cases where the Fetch request fails or the response indicates an error.

- **Helper Functions**:

- Include helper functions to:
    - Select DOM elements (`qs`).
    - Check the response status and handle errors (`statusCheck`).

Note: Test the form in your browser to ensure it communicates correctly with the backend API and handles responses as expected.

Activity 4 – Disney Movie Tracker

Given the data in movies.json and the application's bare bone structure in the `disney.js` file under the starter folder.

`movies.json` file format:

```
{
  "little-mermaid": {
    "release-year": 1989,
    "featured-song": "Under the sea",
    "rotten-tomatoes": 93
  },
  ...
}
```

Create an endpoint `/add` which does the following:

- It is a POST endpoint which takes 4 body params: `movie`, `year`, `song` and `rating`

- If any param is not set, set the appropriate status code and send a `"Missing required parameters"` plain text message
- If all are set, read the provided `movies.json` file
- If the movie passed exists in `movies.json`, write back to the file based on the new passed parameters and send the plain text message `"updated information for designated movie"` upon success
- Otherwise, write back to the file for the movie passed and send the plain text message `"added information for designated movie"` upon success
- Make sure your endpoint handles the case of the file not existing and the possibility that an error could occur on the server.
    - If the file does not exist, send the plain text message `"file does not exist"`
    - If there is a server error, send the plain text message `"something went wrong on the server"`

Activity 5 – `jokebook` API

You are tasked with creating a web service that has two endpoints:

- Endpoint 1 (GET): `/jokebook/categories`
    - should respond with a plain text response
    - should prepend the phrase `"a possible category is "` to each possible category and each sentence should be on its own line.
- Endpoint 2 (GET): `/jokebook/joke/:category`
    - should respond with a JSON response
    - will send a random JSON response from the specified `/:category`
    - If the category is not valid, will respond with `{'error': 'no category listed for category'}`

Use the provided `jokebook-app.js` file as a starter.

Activity 6 – Remembering your users

This activity lets you practice with cookies. Create an Express.js application in a file named `cookies.js`. Create 3 pages `/page1` (GET), `/page2` (GET) and `/page2` (POST).

- `/page1` shows the text `"Welcome <user_name>"`. The user's name must be taken from a cookie named `user_name`. If the cookie is not set, show the following HTML instead:

```
You're not recognized.<br />
Please register your name <a href="/page2">here</a>.
```

- `/page2` (GET) shows a form whose method is `post` and which submits to the page `/page2`. This form lets user enter his name and has a "Save" button.
- `/page2` (POST) handles the form and sets a cookie named `user_name` which stores the user name entered in the form and will expire after 1 minute.

Use the following scenario to test this application:

1. Visit `/page1`. Expected to the the text `"You're not recognized…"`.
2. Click on the link to visit `/page2`. Expected to see a form in `/page2`.
3. Enter your name on the form, click "Save" button.
4. Re-visit `/page1`. Expected to see the text `"Welcome <user_name>"`.