HTTP API HW II

This assignment is due by the date on the dropbox.

Assignment:

Fork & Download the ***HTTP-API-Assignment-II*** starter files from our Github organization.
https://github.com/IGM-RichMedia-at-RIT/http-api-assignment-ii

In this assignment, you will use Node.js to build a simple HTTP API using the GET, HEAD & POST methods. Responses will be in json and use the appropriate status codes. You do not need to implement xml responses or client code to handle xml for this assignment. You will still need to write custom client code for sending requests, etc.

You will need to build an HTTP API that satisfies the following criteria:

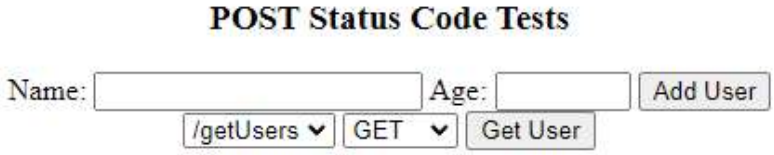
- Loads the client page provided as the home page.
- Loads the css file into the client page as text/css.
- Loads the following URLs, both with fetch() when hitting the 'send' button **and** directly to the URL (directly to URL will fire a GET request).Client fetch() requests should be made to the selected url with the selected method when the "Get User" button is pressed.
  - /getUsers with GET retrieves 200 success with results.
  - /getUsers with HEAD retrieves 200 success without results.
  - /notReal with GET retrieves a 404 not found with error message.
  - /notReal with HEAD retrieves a 404 not found without error message.
  - Any other page with a 404 status code
- POST to the following URL with fetch() when the "Add User" button is pressed.
  - /addUser: Should return a 201 status code when adding a new user, a 204 when updating the age of an existing user, and a 400 if the request is missing a name, age, or both.
- In fetch requests, client should NOT parse JSON responses for HEAD requests.
- In fetch requests, client should parse JSON responses for other responses.
- In fetch requests, status codes should be used/handled appropriately.
- JSON failures should have a human readable "message" field, as well as an unique text "id" field that is useful for developers.

HTTP API HW II

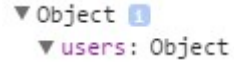
Examples of Correct Ouput:

The home page '/' (client.html and style.css)

/getUsers GET request after hitting "Get User" *BEFORE* adding a user. Should get a 200 status code in the inspector's network tab. Note that to get the "{}" output for the users object, you may need to use the JSON.stringify function, otherwise it will print out as [object Object].



Example output of the JSON response printed to the console.

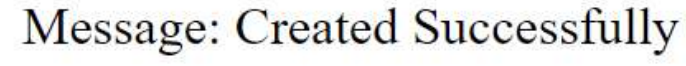


HTTP API HW II

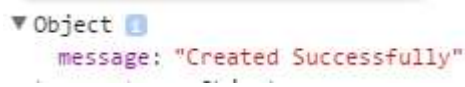
/getUsers HEAD request after hitting 'Send' with /getUsers and 'HEAD' *BEFORE* adding a user. (HEAD requests do not have a body, just status codes and meta data like content-type). Should be a 200 status code.

HEAD requests can be used to see the size or file type/format of a file before downloading. Similarly, HEAD requests can see if a file has changed since last time it was downloaded (images, CSS files, JS files, etc). If it's the same file as other places on the site, there may be no point in downloading the same file again.

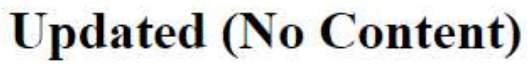
/addUser after hitting 'Add User' with an appropriate name and age. Should get a 201 status.



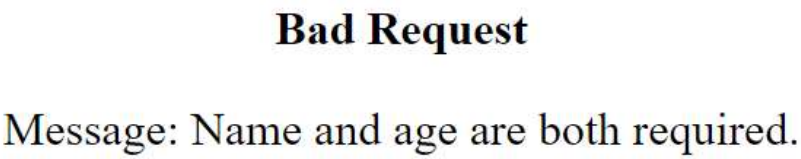
Example output of JSON response.



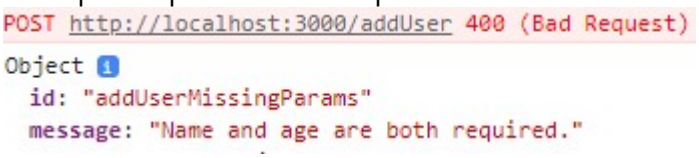
/addUser after hitting 'Add User' with an *existing* name and new age.
This will throw a 204. There is no response body in a 204 message.



/addUser after hitting 'Add User' without a name or age. Should get a 400 status.



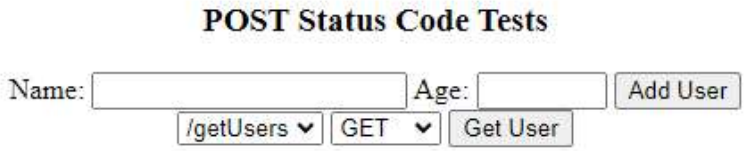
Example output of JSON response.

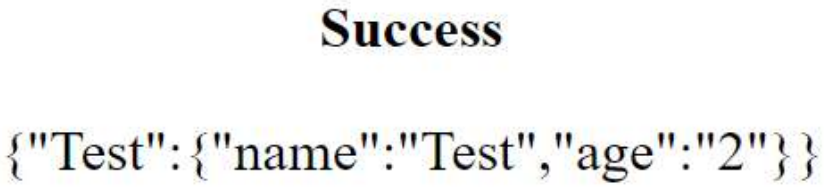


/getUsers GET request after hitting 'Get User' *AFTER* adding a user. Should be a 200 status code.

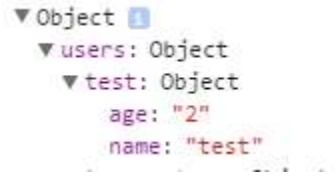
In this example, objects are indexed by the user's name, which is why the 'Test' object has the name 'Test'. If a 'test2' is added, then there would be another object called 'test2' with the name field set to 'test2'.

You do not have to structure your objects quite like that as long as you get this to work correctly. This output is achieved by calling JSON.stringify on the users element of the response object and adding it to the content section.





Example output of JSON response.

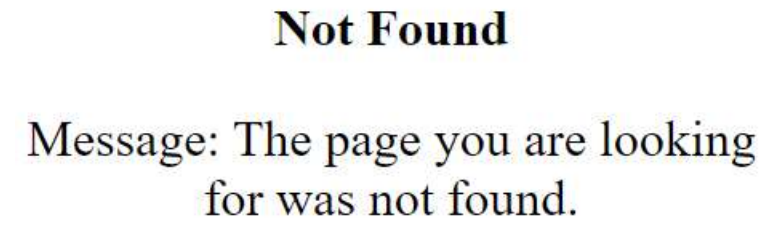


/getUsers HEAD request after hitting 'Get User' *AFTER* adding a user. Status will be a 200.

HEAD requests do not have a body, just status codes and meta data like content-type. This means that even after adding, there will not be a body. The response will contain content-type and all headers that a GET request should have.

HTTP API HW II

/notReal GET request after hitting 'Get User'. Should get a 404 status code.



Example output of JSON response.

/notReal HEAD request after hitting 'Get User'. HEAD requests do not have a body, just status codes and meta data like content-type. Should get a 404 status code.

HTTP API HW II

Example output in console

```
HEAD http://127.0.0.1:3000/notReal 404 (Not Found)
```

HTTP API HW II

Direct call to /style.css

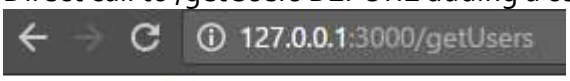

```
html, body {
  height: 100%;
  width: 100%;
  margin: 0;
  padding: 0;
}

#top {
  text-align: center;
  margin: 0 auto;
}

#top h1 {
  font-size: 2em;
}

#content {
  margin-top: 100px;
  font-size: 2em;
  text-align: center;
}
```

Direct call to /getUsers BEFORE adding a user.



{"users":{}}

Direct call to /getUsers AFTER adding a user.

{"users":{"test":{"name":"test","age":"2"}}}

Direct call to anything else

{"message":"The page you are looking for was not found.","id":"notFound"}

HTTP API HW II

ESLint & Code Quality

You are required to use ESLint with the js/recommended configuration for code quality checks. The eslint.config.js file was given to you in the starter files. Your **server** code must pass ESLint to get credit for this part. Warnings are okay, but errors must be fixed. I won't require ESLint on the client side code in this assignment. ***If you cannot fix an error or need help, please ask us!***

You are also required to use the "pretest" script in order to run ESLint. I should be able to run "*npm test*" and have ESLint scan your code for me.

Continuous Integration

You are required to use GitHub Actions for continuous integration. Your latest commit on GitHub must show a green checkmark denoting a successful run of the npm test command. Be aware that this checkmark can take a few minutes to show up, as GitHub Actions runs the test.

All starter code for this class includes the .github folder, which contains configuration for the GitHub Action that will test your code. The test should run automatically for each commit. If you don't have the .github folder, you'll have to grab it from another project.

HTTP API HW II

Rubric

DESCRIPTION	SCORE	VALUE %
Loads the client page – Loads the client.html page as the home page.		**5**
Loads the css as text/css – Loads the style.css file as text.css for the home page and directly.		**5**
/getUsers with GET works correctly – /getUsers with GET retrieves 200 success with results (even if empty). Should also work directly to URL.		**10**
/getUsers with HEAD works correctly – /getUsers with HEAD retrieves 200 success without results.		**15**
/notReal or any other page with GET works correctly – /notReal with GET retrieves a 404 not found with error message. Should also work directly to the URL.		**10**
/notReal with HEAD works correctly – /notReal with HEAD retrieves a 404 not found without error message.		**15**
/addUser with POST and new valid data works correctly – /addUser with POST and new valid data, adds a user and retrieves 201 with message.		**15**
/addUser with POST and existing data works correctly – /addUser with POST and existing valid data, updates a user and retrieves 204 without message.		**15**
/addUser with POST and invalid data works correctly – /addUser with POST and invalid data retrieves 400 with message.		**5**
JSON responses formatted correctly – JSON messages are formatted correctly. JSON errors contain a message field and error id field.		**5**
Heroku Penalty – If your app is not functional on Heroku, there will be a penalty.		**-20% (this time)**
GitHub Actions Penalty – If your build is not successful on GitHub Actions, there will be a penalty.		**-20% (this time)**
ESLint Penalties – I should be able to run 'npm test' and have ESLint scan your code without errors. Warnings are acceptable. **Check your code with ESLint often during development!** **If you cannot fix an error, please ask us!**	1 Error 2 Errors 3 Errors 4 Errors 5+ Errors	**-5%** **-10%** **-15%** **-20%** **-30%**
Additional Penalties – These are point deductions for run time errors, poorly written code or improper code. There are no set values for penalties. The more penalties you make, the more points you will lose.		
TOTAL		**100%**

Submission:

By the due date please submit a zip of your project to the dropbox. **Do not** include the node_modules folder in the zip or in your git repo.

In the submission comments include the following:

- *a link to your Heroku App (not the dashboard but the working web link from 'open app')*
- *a link to your Github repo (if private, please add the TA and myself)*