Tutorial: Consuming APIs

<https://pusher.com/tutorials/http-response-codes-part-2>

1. Build a simple Node.js backend using Express to send data.
2. Write simple HTML pages and make API calls using JavaScripts fetch API.
3. Use [Postman](https://www.getpostman.com/) for some of our tests

You will need Node 8+ and Postman installed on your machine.

# STEP 1: Create the Node.js backend

* 1. Create the Node project
     + This will create all the necessary back-end files
     + Create an index.html at your project’s root directory. All your back-end code can go here
  2. Install dependencies (= body parser)
  3. Create the server ( = specify Node server in the index.js file)
  4. Import dependencies ( = express framework with body parser middleware)
     + To create a web app using Node terminal, install Express (a minimalist framework for Node)
     + To incorporate a Node.js middleware for parsing JSON, Raw, Test and UR encoded form data, install body-parser
  5. Define some route
     + Add the routes for success status codes
     + Add the routes for redirection status codes
     + Add the routes for client error status codes
     + Add the routes for server error status codes
  6. You can start the server (run the app) by calling the node with the script (viz. index.js in your command prompt):
     + C:\Users\Ash\Documents\web-projects\http-response-codes>**node index.js**
     + You should see this response in the command line:

“Node based web application running on port 3000”

Error: ENOENT: no such file or directory, stat 'C:\Users\Ash\Documents\web-projects\http-response-codes\html\register.html'

Because you set that message at the end of your index.js:

app.listen(3000,() => {

console.info("Node based web application running on port 3000")

});

We’ll get to that error soon

* + - Next, on your browser, visit <http://localhost:3000/>
    - You should see this response in the command line:

“Error: ENOENT: no such file or directory, stat 'C:\Users\Ash\Documents\web-projects\http-response-codes\html\register.html'”

Because you do not have a register.html created yet but since you DEFINED that ROUTE in your index.js, the server tried to render it. So at least it validates that the server is trying to fetch the correct html (register.html) per your route definition:

app.get('/',(req, res) => {

res.status(200).sendFile(`${\_\_dirname}/html/register.html`)

});

1. If you simply create an empty register.html inside an html folder you will not see that error

# STEP 2: Create the HTML and JS frontend

1. Create an html directory at the root of your project
2. Inside the directory, create all the html files referenced in your backend code (code that resides in index.html)

# STEP 3: Test the end-points on the browser

1. Test that when you launch localhost://3000**/**, it brings up register.html based on your back-end code in index.js:

app.get('**/**',(req, res) => {

res.status(200).sendFile(`${\_\_dirname}/html/register.html`)

});

* register.html in turn is configured to produce a form

<form id="form">

<input type = "text" name = "name">

<input type = "email" name = "email">

<input type = "password" name = "password">

<input type = "submit" name = "submit">

</form>

* the html also has the event listener that dictates what ROUTE to FETCH and what METHOD to call

form.addEventListener('submit',(e) => {

e.preventDefault();

fetch('http://localhost:3000/register',{ //You're calling /register with POST, which in turn is configured to return 201

method: 'POST',

body: JSON.stringify(data), //convert the data object you defined into a JSON string

headers:{

'Content-Type':'application/json'

}

})

1. Test that when you **submit** the register.html form with meaningful and valid inputs:

* the html also has the event listener that dictates what ROUTE and METHOD to call
  + POST is called and the route localhost://3000/register is requested based on the register.html event listener front-end:

form.addEventListener('submit',(e) => {

e.preventDefault();

fetch('http://localhost:3000/register',{ //You're calling /register with POST, which in turn is configured to return 201

method: 'POST',

body: JSON.stringify(data), //convert the data object you defined into a JSON string

headers:{

'Content-Type':'application/json'

}

* + When the fetch is fired, it looks to the route’s configuration based on the specified route + method defined in index.js, which is:

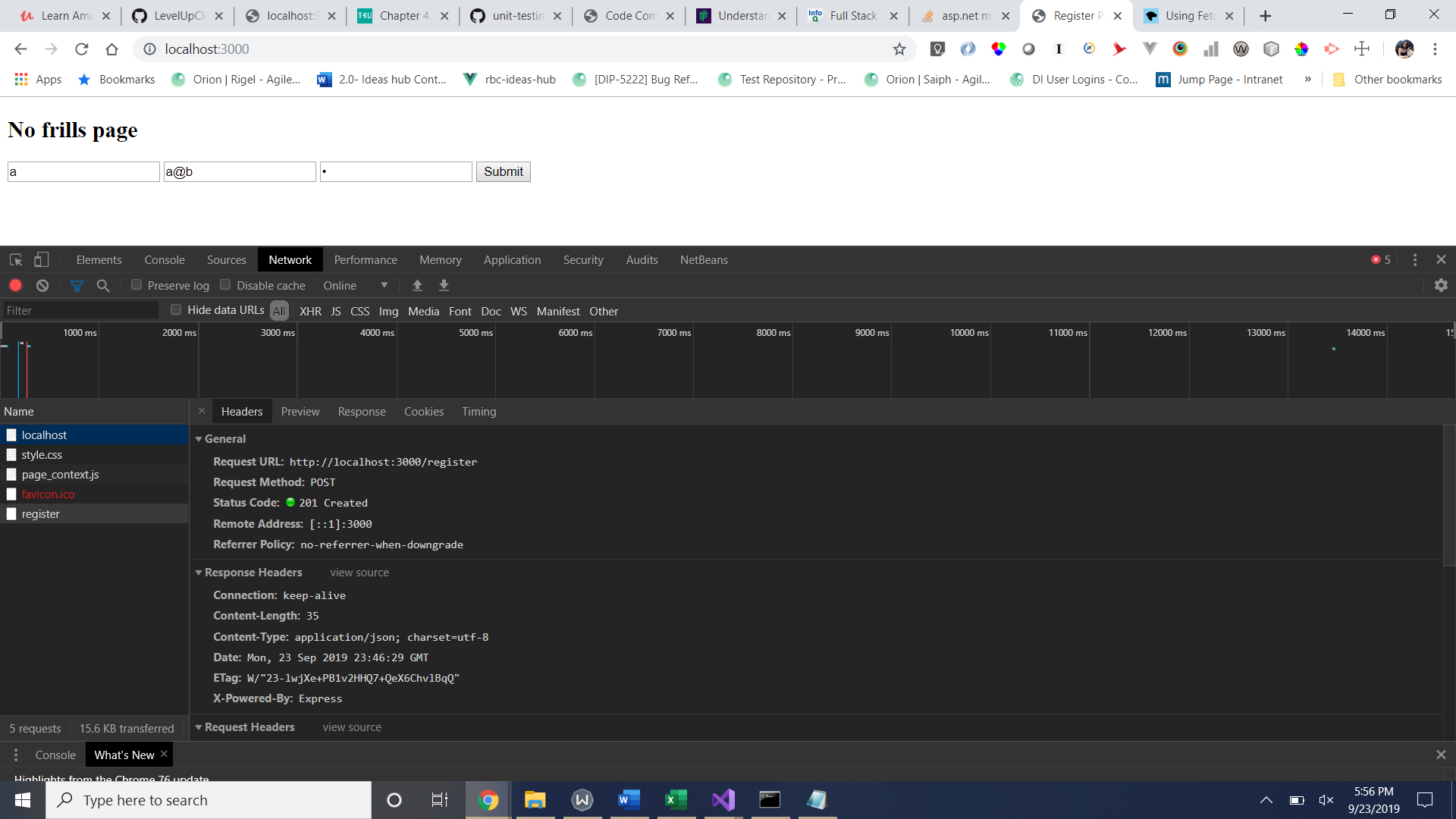
app.post('/register',(req,res) => {

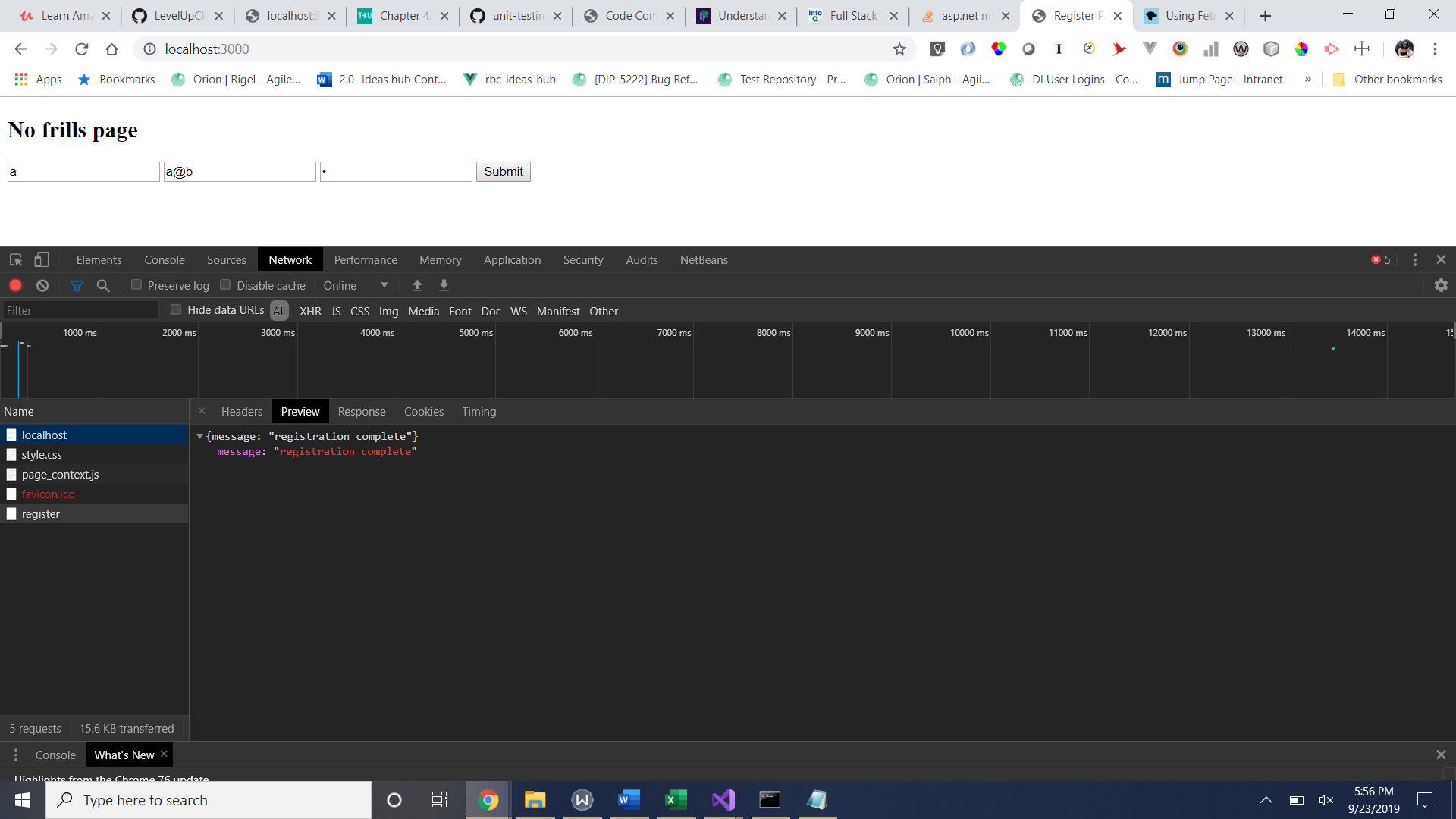
//logic to save to database should go here

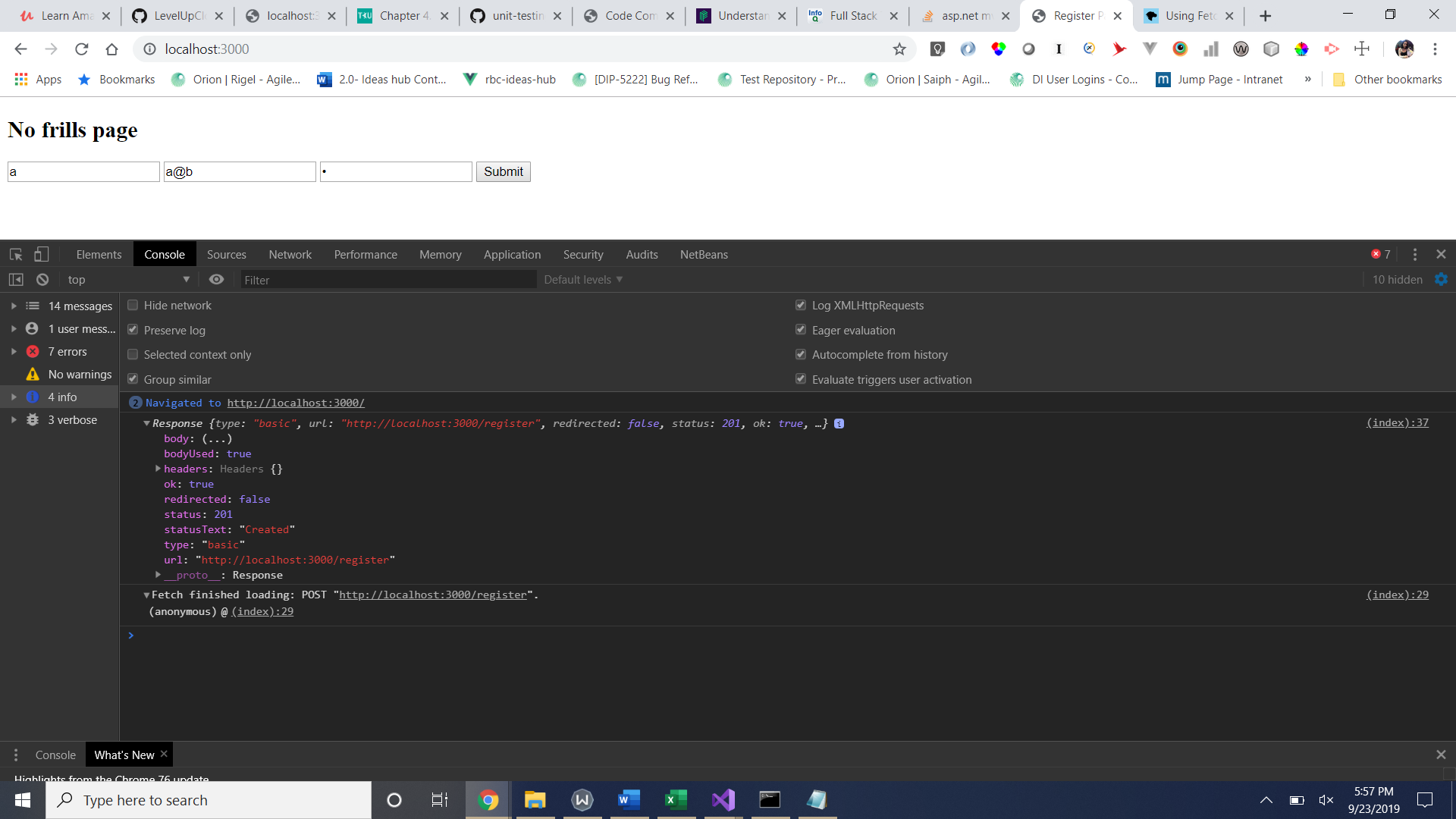
res.status(201).send({message : 'registration complete'})

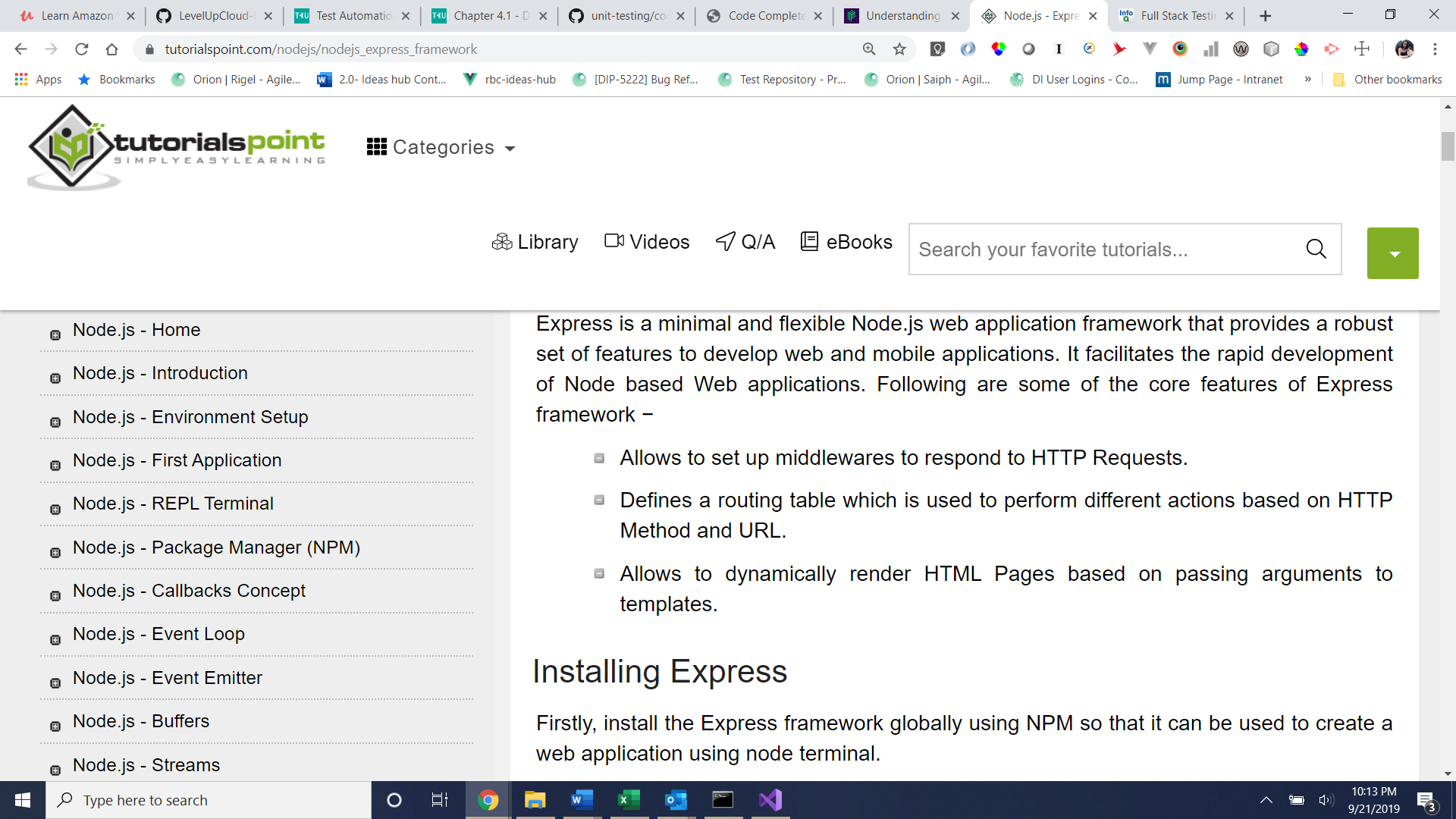
});

* + - Based on this, you’ll see a Response code of 201 on your browser
    - You’ll also see an alert message on screen and a console.log message (because you wrote them in the fetch API call inside register.html









1. Test that when you launch localhost:3000/update it brings up update.html based on your index.js back-end configuration

app.get('/update',(req,res) => {

res.status(200).sendFile(`${\_\_dirname}/html/update.html`)

});

* update.html in turn is configured to simulate an update form (with prepopulated dummy data)

<input type = "text" name = "name" value = "AceKyd" >

<input type = "email" name = "email" value = "ace@kyd.com">

<input type = "password" name = "password" value = "acekyd">

<input type = "submit" name = "submit">

* Just like register.html, the update.html also has the event listener that dictates what ROUTE to FETCH and what METHOD to call

form.addEventListener("submit",(e) => {

e.preventDefault();

fetch('http://localhost:3000/update',{

method: "PUT",

body: JSON.stringify(data), //convert the data object you defined into a JSON string

headers:{

'Content-Type':'application/json'

}

}

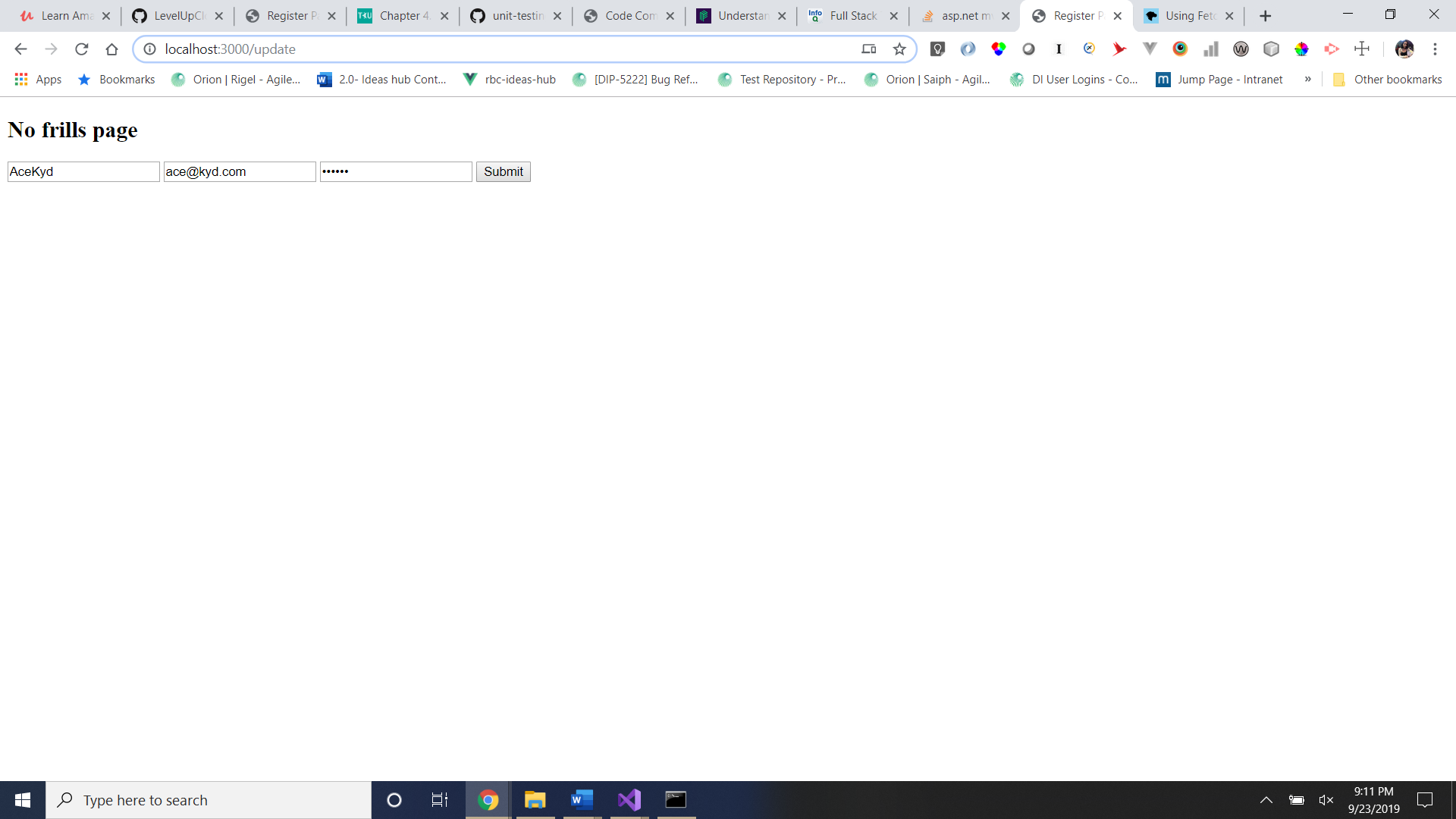
* When the fetch is fired, it looks to the route’s configuration based on the specified route + method defined in index.js, which is:

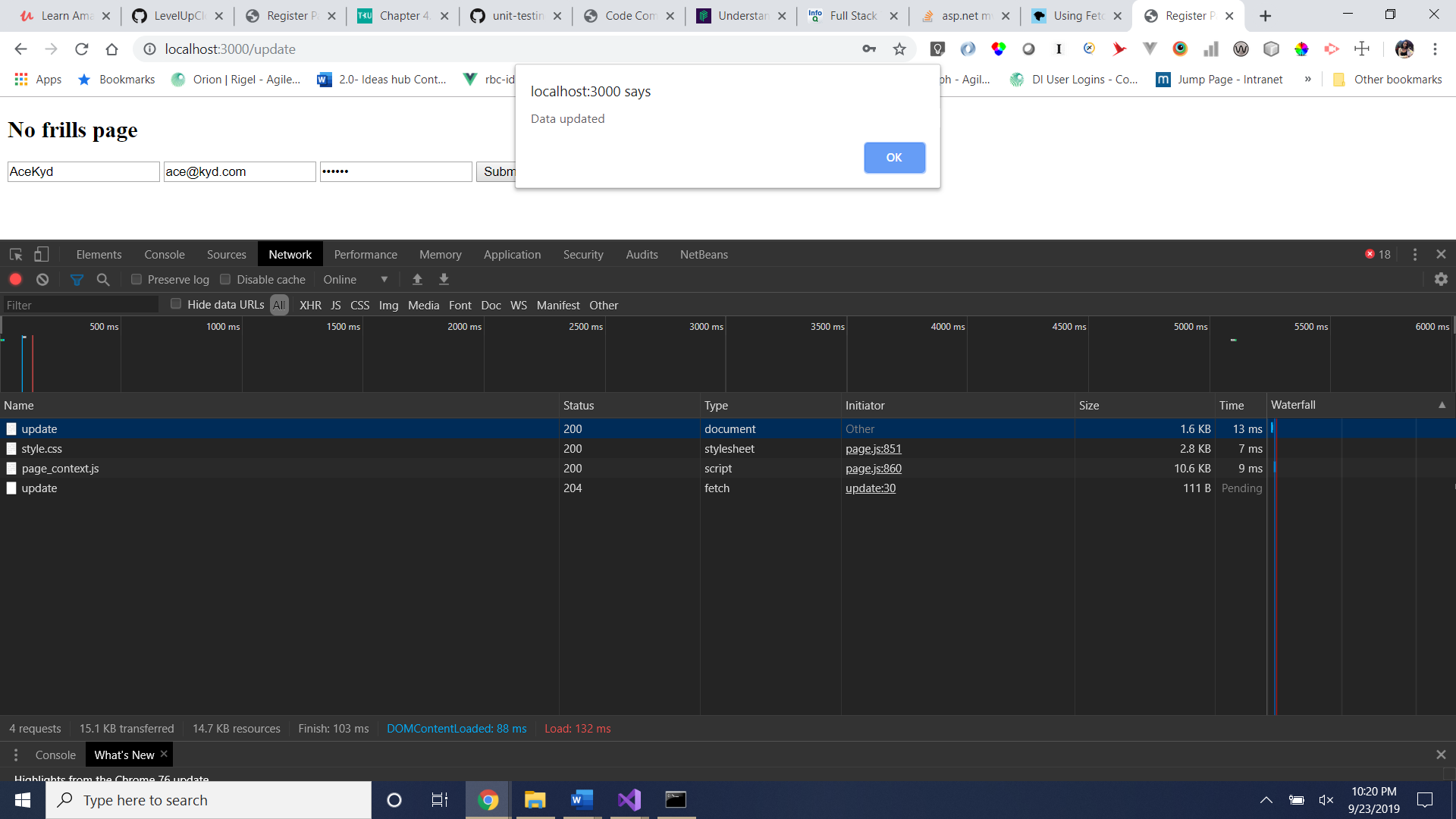
app.put('/update',() => {

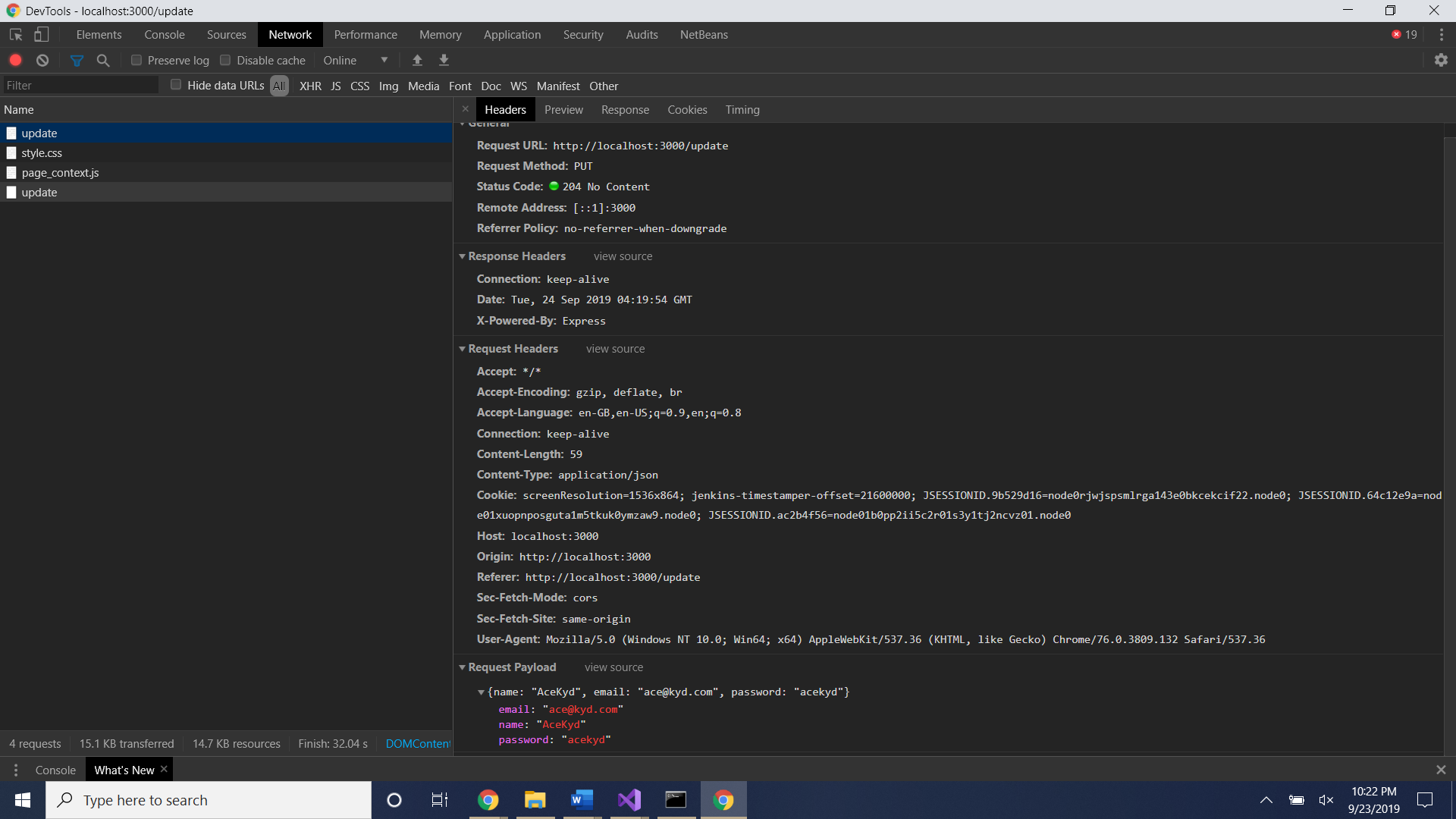
//logic to update the database record should go here.

//Since you do not have database hooked up, you'll need to simulate the update experience by creating an update form with prefilled dummy data as part of your front end [code]

res.status(204).end()







1. Test by navigating to localhost:3000/user/jane
   * Based on the routing definition in index.js (back-end), this leads to /user/jane-new

//302 FOUND

app.get('/user/jane', (req,res) => {

res.set('location','/user/jane-new')

res.status(302).send();

});

* The new location /user/jane-new then returns a message (just like an API end point would!)

app.get('/user/jane-new',(req,res) => { //This is like an API end point that returns data!

res.status(200).send({message : "This is Jane's new page"})

});

