**CS628 Full-Stack Development II – Backend**

**PE07 - Basic Node and Express - Implement a Root-Level Request Logger Middleware**

Developed by Clark Ngo on December 29, 2020

School of Technology & Computing (STC)

City University of Seattle (CityU)

**Before You Start**

* Some steps are not explained in the tutorial**.** If you are not sure what to do:
  1. Consult the resources listed below.
  2. If you cannot solve the problem after a few tries, ask a TA for help.

**Learning Outcomes**

Students will be able to:

* Learn Basics of Node and Express

**Resources**

* <https://www.freecodecamp.org/learn/apis-and-microservices/basic-node-and-express/implement-a-root-level-request-logger-middleware>

**How to Submit**

* **Upload .zip file**
* **Write a 150-word summary to explain your understandings and findings from this lab assignment.**

## **Basic Node and Express - Implement a Root-Level Request Logger Middleware**

Earlier, you were introduced to the express.static() middleware function. Now it’s time to see what middleware is, in more detail. Middleware functions are functions that take 3 arguments: the request object, the response object, and the next function in the application’s request-response cycle. These functions execute some code that can have side effects on the app, and usually add information to the request or response objects. They can also end the cycle by sending a response when some condition is met. If they don’t send the response when they are done, they start the execution of the next function in the stack. This triggers calling the 3rd argument, next().

Look at the following example:

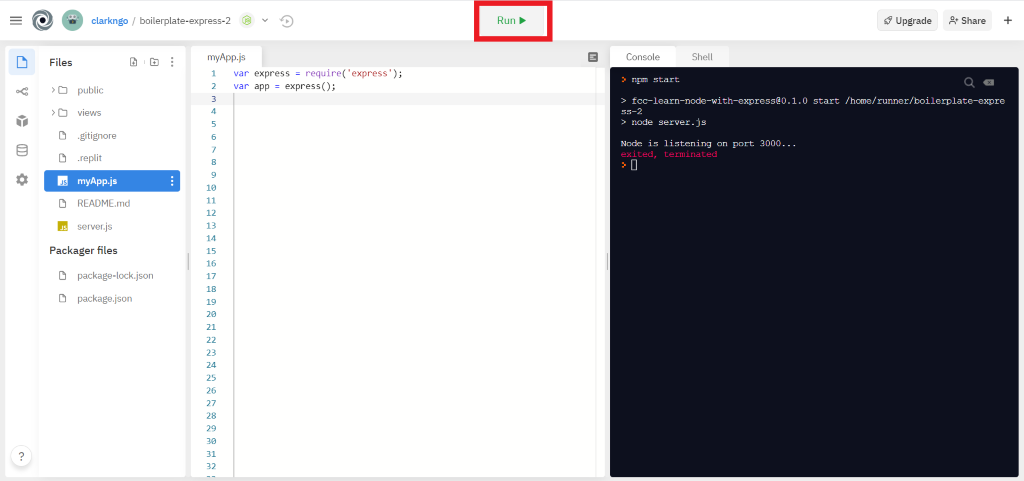
function(req, res, next) { console.log("I'm a middleware..."); next();}

Let’s suppose you mounted this function on a route. When a request matches the route, it displays the string “I’m a middleware…”, then it executes the next function in the stack. In this exercise, you are going to build root-level middleware. As you have seen in challenge 4, to mount a middleware function at root level, you can use the app.use(<mware-function>) method. In this case, the function will be executed for all the requests, but you can also set more specific conditions. For example, if you want a function to be executed only for POST requests, you could use app.post(<mware-function>). Analogous methods exist for all the HTTP verbs (GET, DELETE, PUT, …).

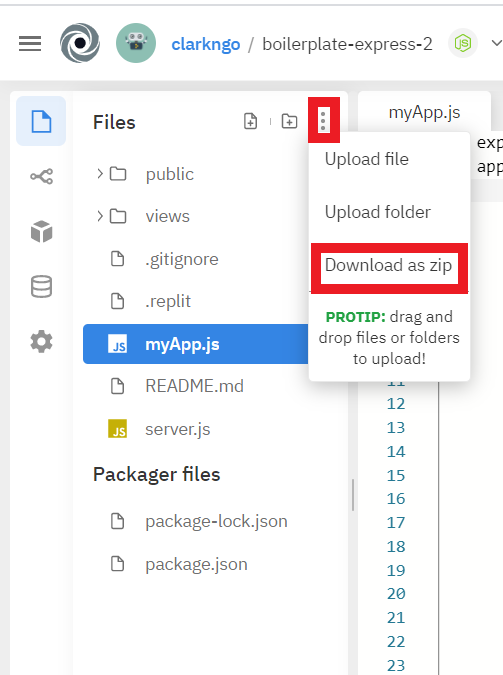
Build a simple logger. For every request, it should log to the console a string taking the following format: method path - ip. An example would look like this: GET /json - ::ffff:127.0.0.1. Note that there is a space between method and path and that the dash separating path and ip is surrounded by a space on both sides. You can get the request method (http verb), the relative route path, and the caller’s ip from the request object using req.method, req.path and req.ip. Remember to call next() when you are done, or your server will be stuck forever. Be sure to have the ‘Logs’ opened, and see what happens when some request arrives.

**Note:** Express evaluates functions in the order they appear in the code. This is true for middleware too. If you want it to work for all the routes, it should be mounted before them.

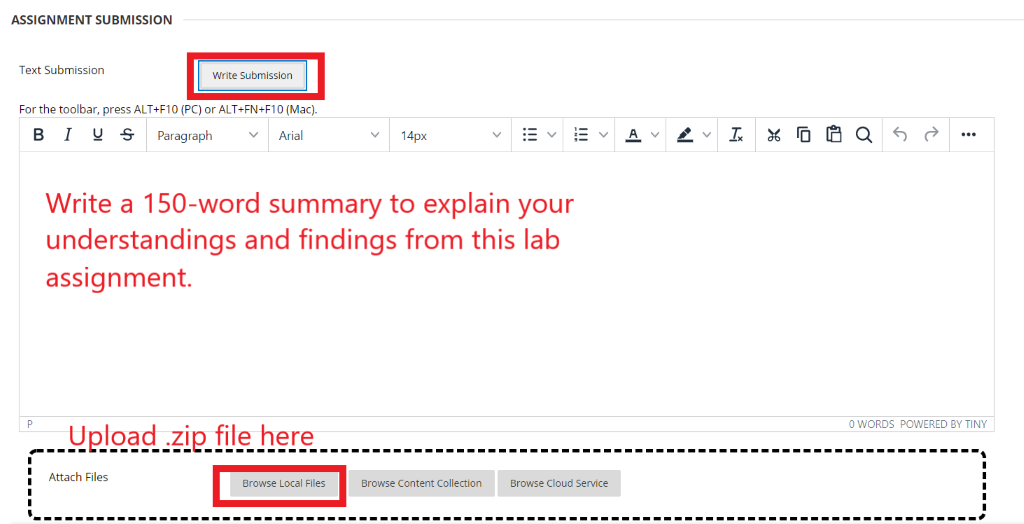
1. Start a new project on Repl.it using [this link](https://repl.it/github/freeCodeCamp/boilerplate-express).
2. Click Run to test your code

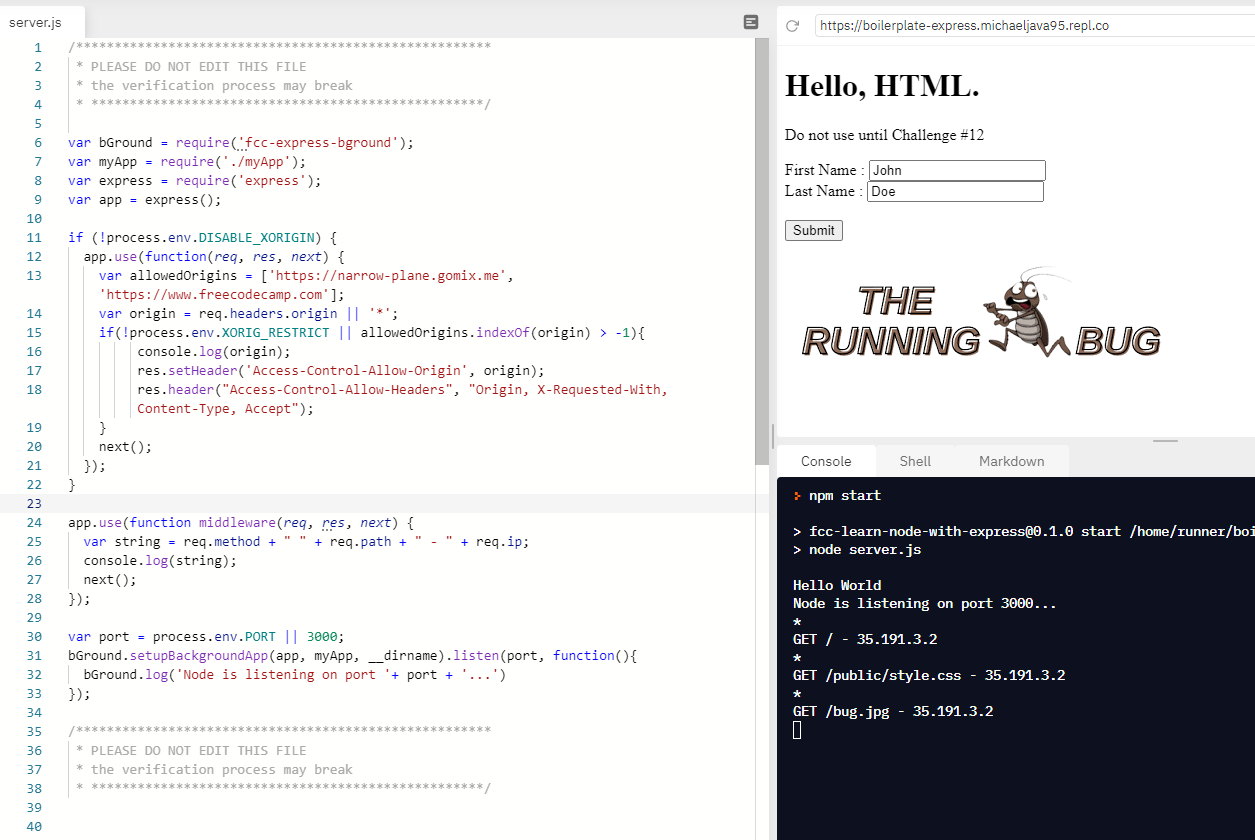


1. Download the project as .zip.



1. In BlackBoard, upload the .zip file and write a 150-word summary to explain your understandings and findings from this lab assignment.



Middleware executes before the controller executes. We can create a targeted middleware or a general middleware that executes for all requests. In the server.js file we set up our middleware and we made it a general middleware that executes for all requests. We accessed the method, path, and ip parameters of the request object and logged the mon the console. Then we let the request continue with its normal execution path.   
  


Looks like the middleware feature of javascript express environment is the same thing as the filters in the java servlet environment. Because filters are also structures that execute when certain directories are requested. And filters also allow us to do computations.

Lets say we have an AdminServlet, which is an admin menu. We login with a username and password from index.html. This form sends the data to AdminServlet with action=”AdminServlet”. But before we let the user to the admin menu page, we need to check if he is really admin or not. We can use a filter here. That filter will intercept every request and response coming in and out of its target. We set filter’s target in web.xml in filter’s url-pattern line.

In the code below, we define a servlet and a filter that has the servlet as its target.

<servlet>

<servlet-name>AdminServlet</servlet-name>

<servlet-class>AdminServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>AdminServlet</servlet-name>

<url-pattern>/AdminServlet</url-pattern>

</servlet-mapping>

<filter>

<filter-name>MyFilter</filter-name>

<filter-class>MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

<url-pattern>/AdminServlet</url-pattern>

</filter-mapping>