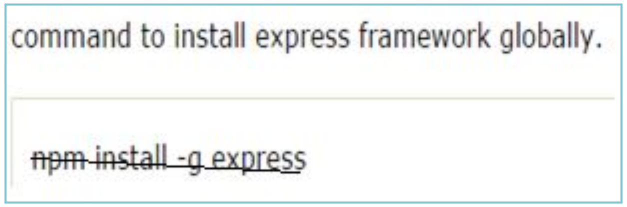
Node.js - Express Framework

Introduction

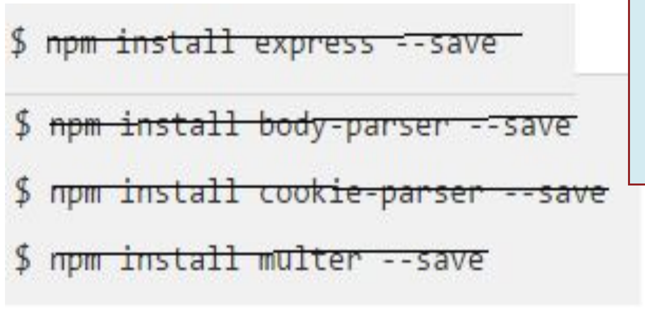
- Express is a relatively small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middleware and routing; it adds helpful utilities to Node.js's HTTP objects; it facilitates the rendering of dynamic HTML views; it defines an easily implemented extensibility standard.
- It is a fast, robust and asynchronous in nature.
- It can be used to design single-page, multi-page and hybrid web applications.
- It allows to setup middlewares to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- It allows to dynamically render HTML Pages based on passing arguments to templates.

Installing Express

- The above commands saves the installation locally in the **node_modules** directory and creates a directory express inside node_modules. You should install the following important modules along with express −
- **body-parser** − This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data.
- **cookie-parser** − Parse Cookie header and populate req.cookies with an object keyed by the cookie names.
- **multer** − This is a node.js middleware for handling multipart/form-data.


command to install express framework globally.

~~npm install -g express~~


```
~~$ npm install express --save~~
~~$ npm install body-parser --save~~
~~$ npm install cookie-parser --save~~
~~$ npm install multer --save~~
```

Note: No need to install npm packages, already available in node_modules

Writing middleware for use in Express apps

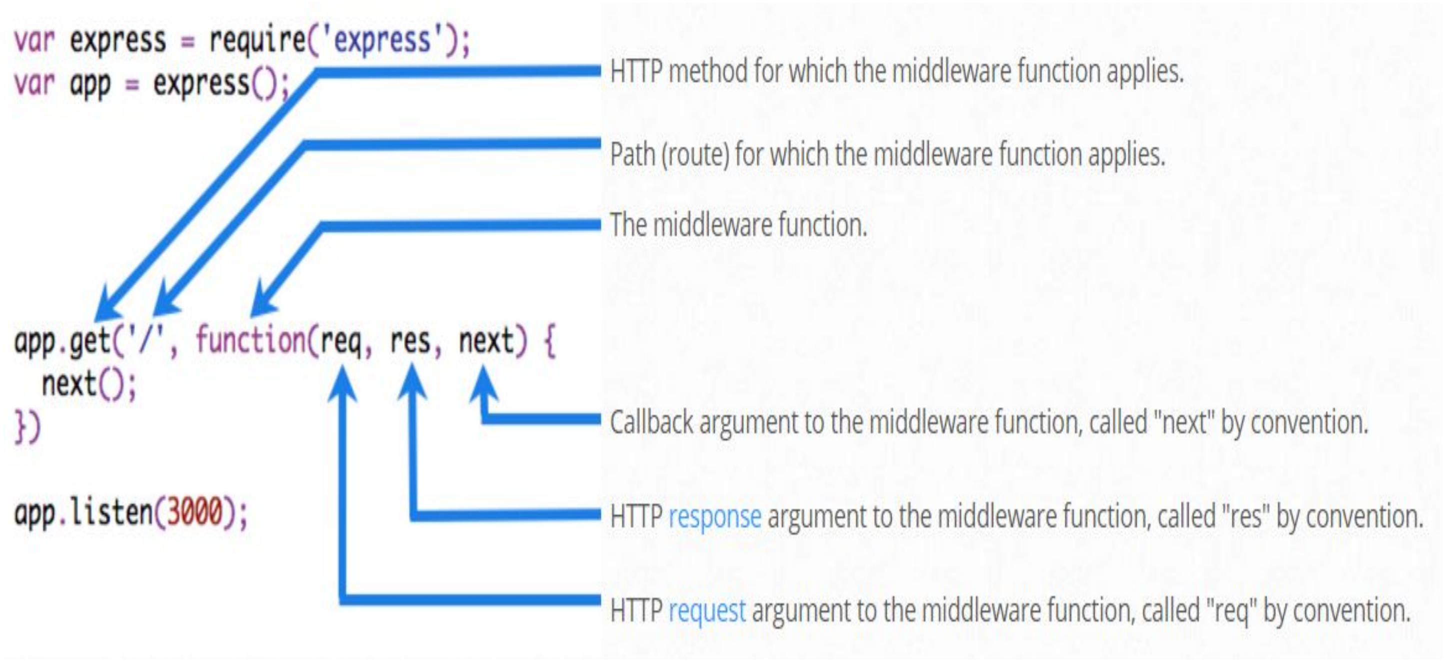
- ***Middleware*** functions are functions that have access to the request object functions are functions that have access to the request object (req), the response object (res), and the next() function in the application's request-response cycle. The next() function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.
- Middleware functions can perform the following tasks:
  - Execute any code.
  - Make changes to the request and the response objects.
  - End the request-response cycle.
  - Call the next middleware in the stack.
- If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

The basic syntax for the middleware functions are as follows –

```
app.get(path, (req, res, next) => {}, (req, res) => {})
```

The middle part **(req,res,next)=>{}** is the middleware function. Here we generally perform the actions required before the user is allowed to view the webpage or call the data and many other functions.

The following figure shows the elements of a middleware function call:

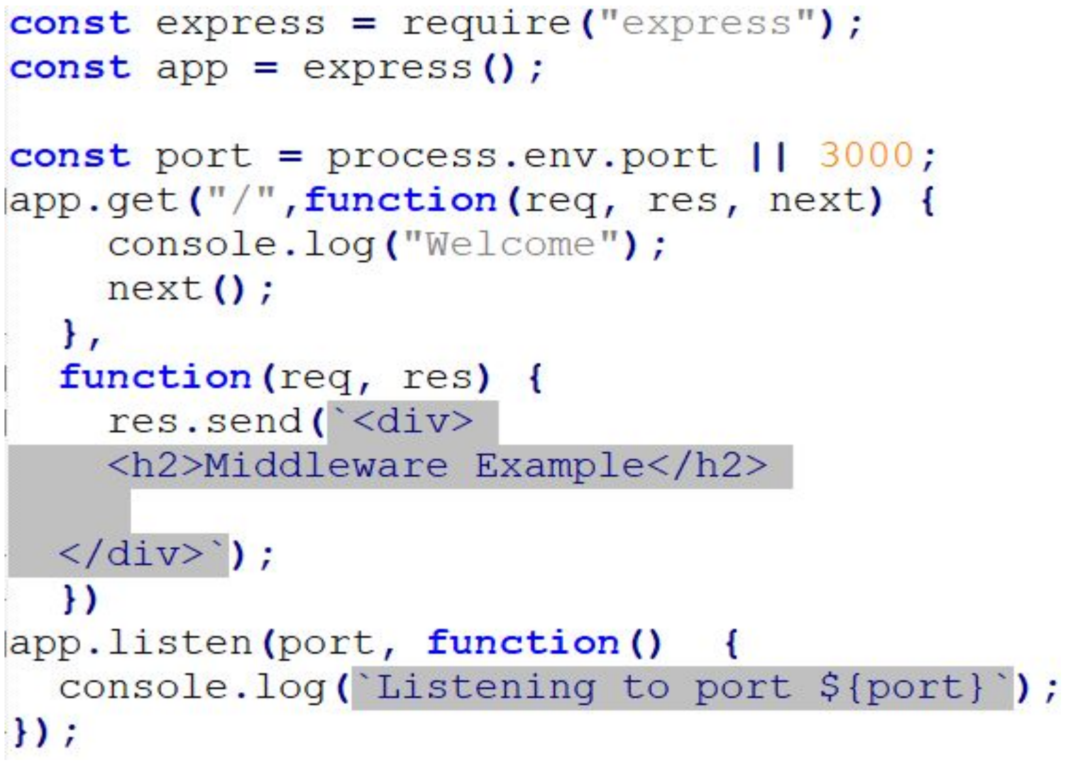


```
var express = require('express');
var app = express();

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

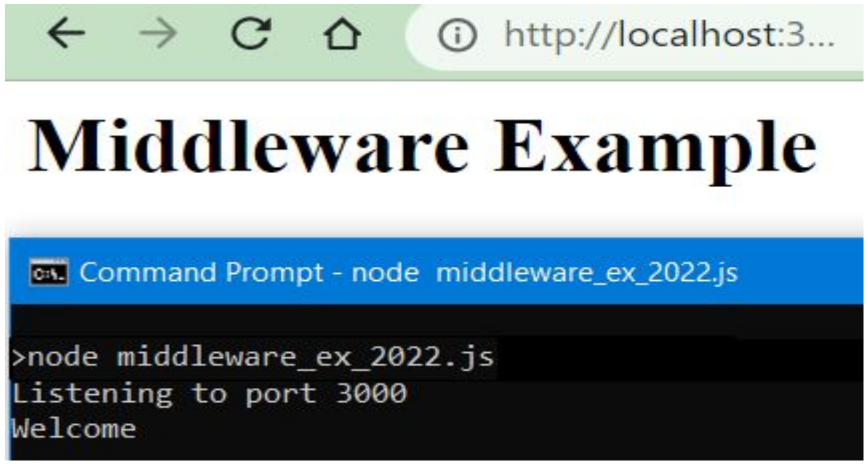
HTTP method for which the middleware function applies.
Path (route) for which the middleware function applies.
The middleware function.
Callback argument to the middleware function, called "next" by convention.
HTTP response argument to the middleware function, called "res" by convention.
HTTP request argument to the middleware function, called "req" by convention.

So let us create our own middleware and see its uses

Example : Middleware



```
const express = require("express");
const app = express();

const port = process.env.port || 3000;
app.get("/",function(req, res, next) {
    console.log("Welcome");
    next();
  },
  function(req, res) {
    res.send(`<div>
    <h2>Middleware Example</h2>

  </div>`);
  })
app.listen(port, function()  {
  console.log(`Listening to port ${port}`);
});
```



http://localhost:3...

Middleware Example

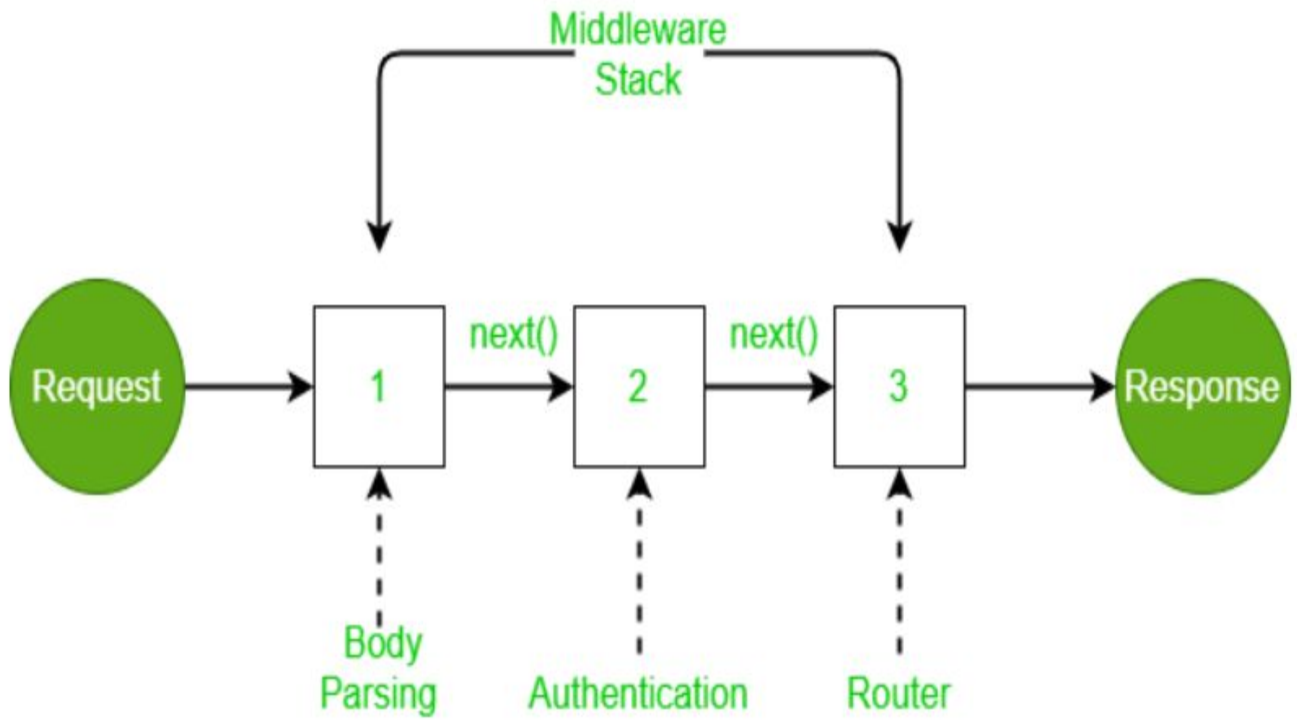
Command Prompt - node middleware_ex_2022.js

```
>node middleware_ex_2022.js
Listening to port 3000
Welcome
```

When you type http://localhost:3000 prints message “Welcome” on console from middleware function before the web page displays “Middleware Example”

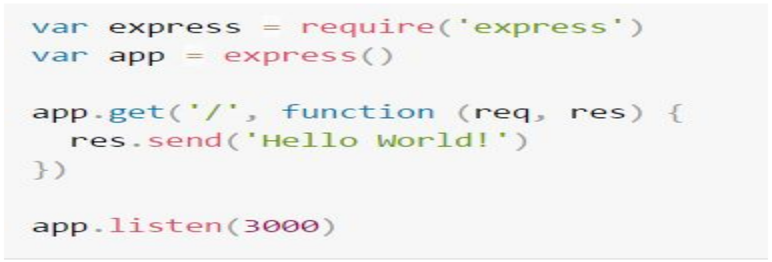
Middleware Chaining

Modified requests will be available to each middleware via the next function –



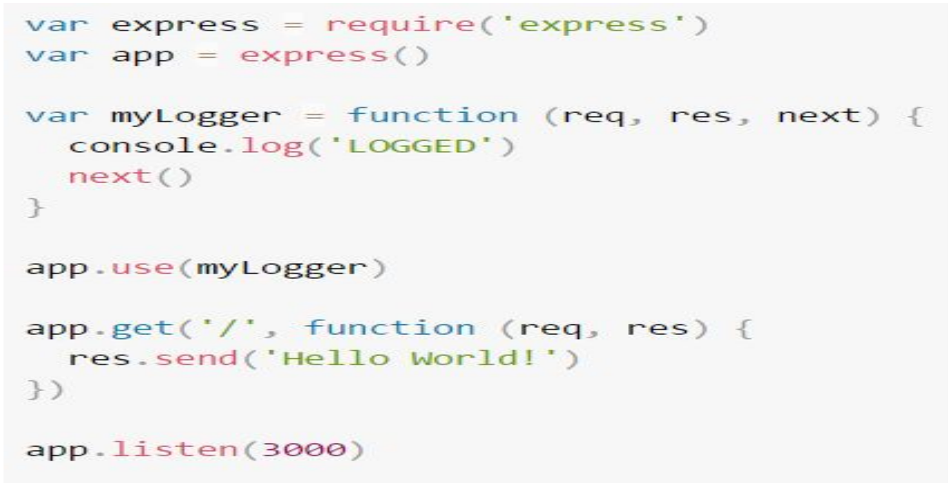
Another example with middleware

- Here is an example of a simple "Hello World" Express application.



```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000)
```

- Here is a simple example of a middleware function called "myLogger". This function just prints "LOGGED" when a request to the app passes through it. The middleware function is assigned to a variable named myLogger.



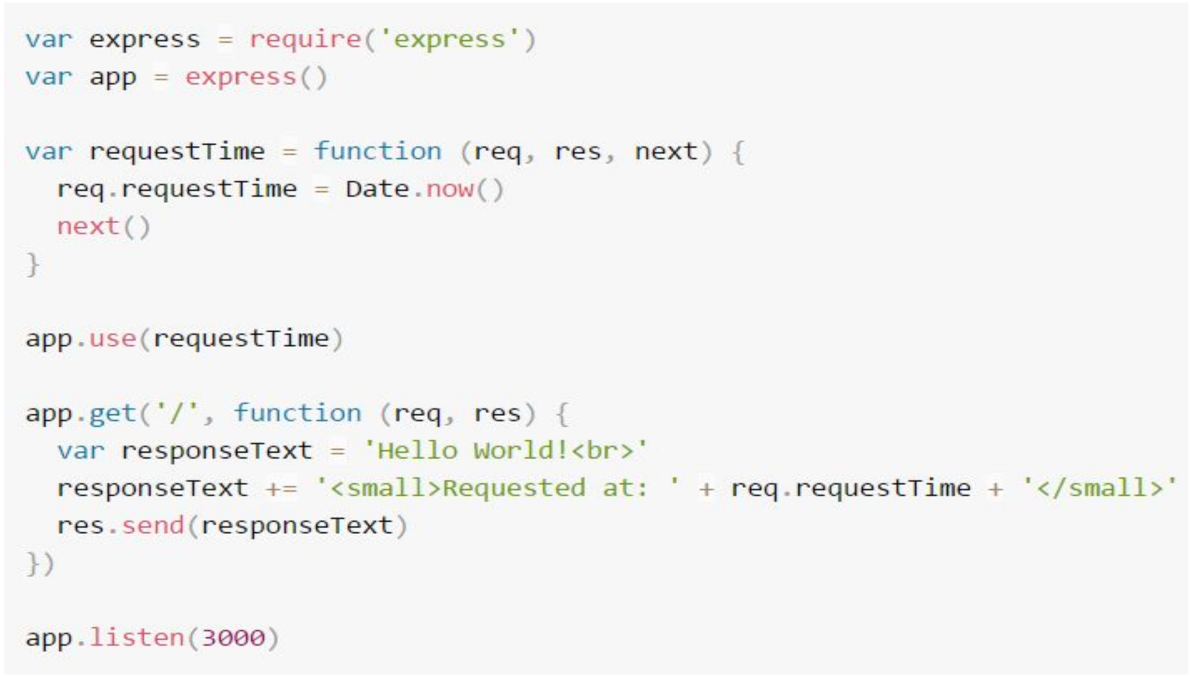
```
var express = require('express')
var app = express()

var myLogger = function (req, res, next) {
  console.log('LOGGED')
  next()
}

app.use(myLogger)

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000)
```

•Notice the call above to next(). Calling this function invokes the next middleware function in the app.

•To load the middleware function, call app.use().

- The order of middleware loading is important: middleware functions that are loaded first are also executed first.
- If myLogger is loaded after the route to the root path, the request never reaches it and the app doesn't print "LOGGED", because the route handler of the root path terminates the request-response cycle.
- The middleware function myLogger simply prints a message, then passes on the request to the next middleware function in the stack by calling the next() function.
- Next, we'll create a middleware function called "requestTime" and add it as a property called requestTime to the request object.

- When you make a request to the root of the app, the app now displays the timestamp of your request in the browser.



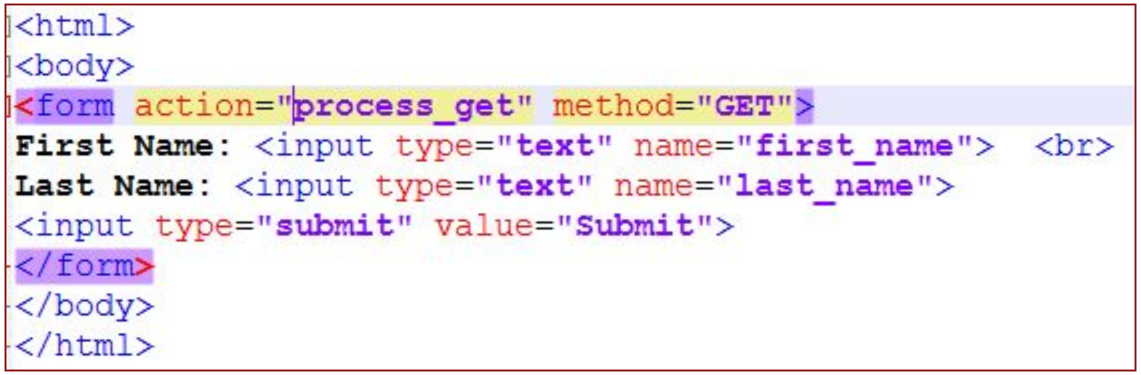
```
var express = require('express')
var app = express()

var requestTime = function (req, res, next) {
  req.requestTime = Date.now()
  next()
}

app.use(requestTime)

app.get('/', function (req, res) {
  var responseText = 'Hello World!<br>'
  responseText += '<small>Requested at: ' + req.requestTime + '</small>'
  res.send(responseText)
})

app.listen(3000)
```

- Because you have access to the request object, the response object, the next middleware function in the stack, and the whole Node.js API, the possibilities with middleware functions are endless.
- In the above middleware example a new function is used to invoke with every request via **app.use()**.

Advantages of using middleware

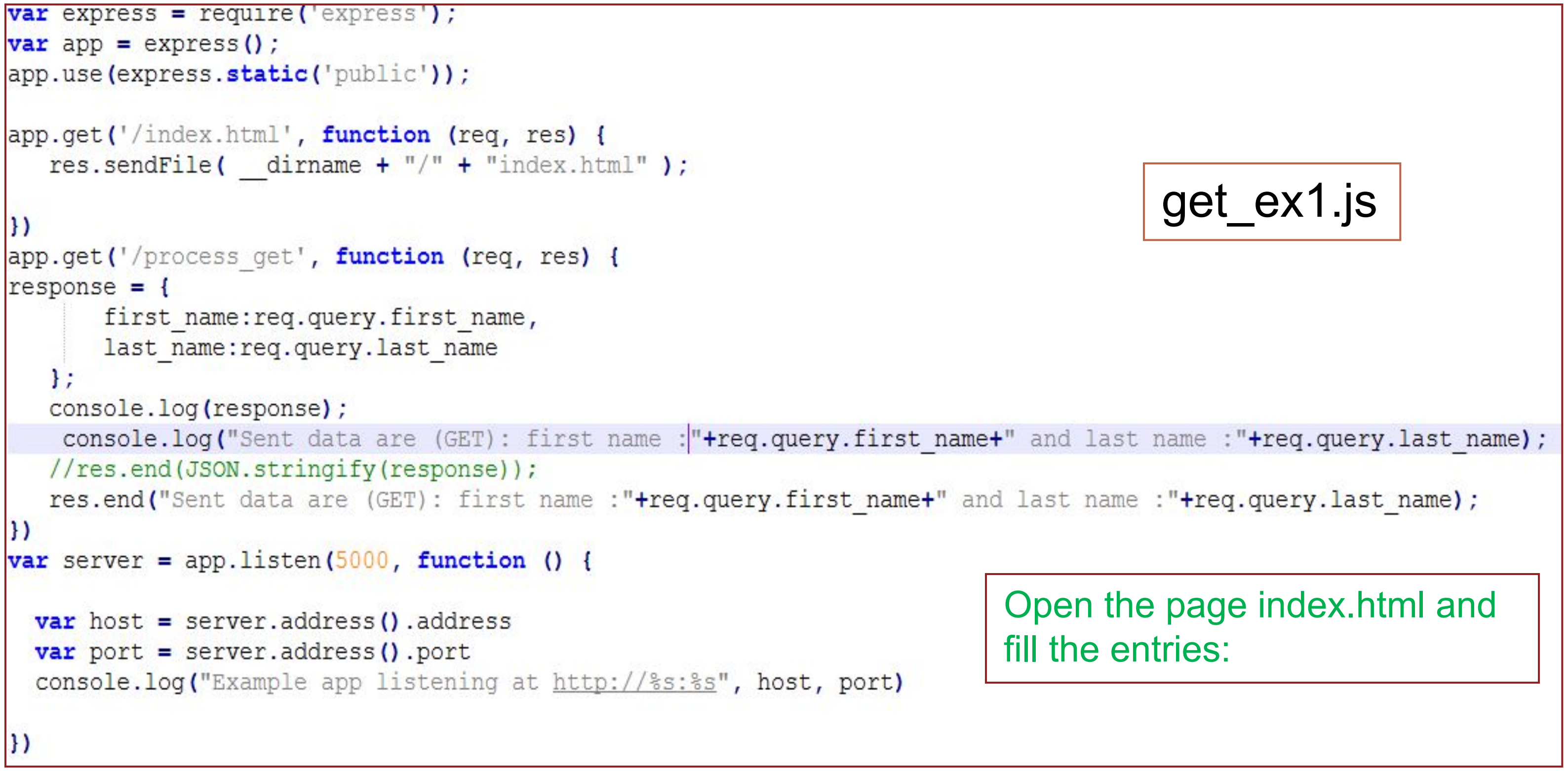
- Middleware is a function, just like route handlers. You can add more middleware's above or below using the same API.
- Middleware can process request objects multiple times before the server works for that request.
- Middleware can be used to add logging and authentication functionality.
- Middleware improves client-side rendering performance.
- Middleware is used for setting some specific HTTP headers.
- Middleware helps for Optimization and better performance.

Express.js GET Method Example 1



```html
<html>
<body>
<form action="process_get" method="GET">
First Name: <input type="text" name="first_name">  <br>
Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

index.html



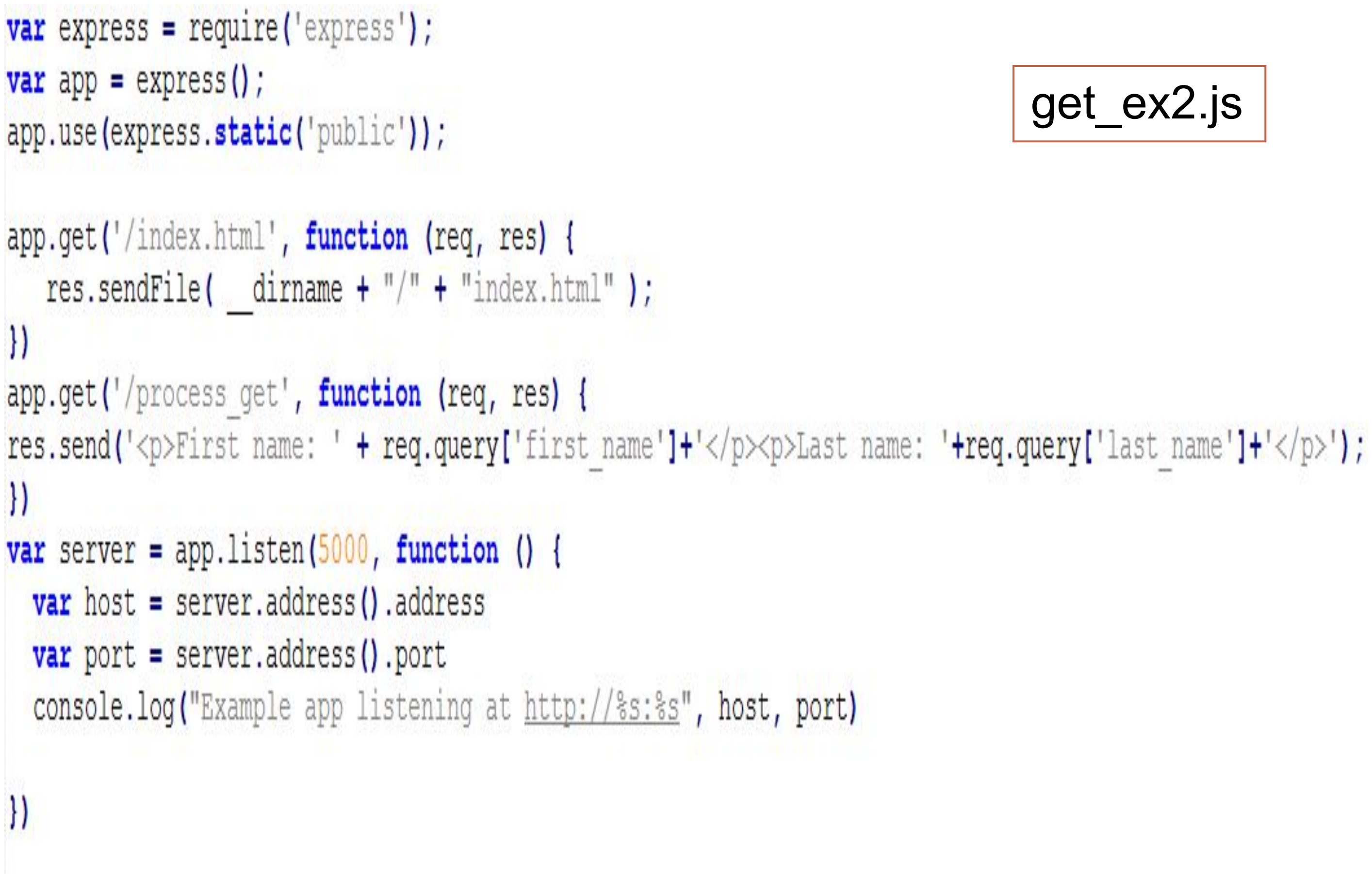
```javascript
var express = require('express');
var app = express();
app.use(express.static('public'));

app.get('/index.html', function (req, res) {
   res.sendFile( __dirname + "/" + "index.html" );

})
app.get('/process_get', function (req, res) {
response = {
       first_name:req.query.first_name,
       last_name:req.query.last_name
   };
   console.log(response);
    console.log("Sent data are (GET): first name :"+req.query.first_name+" and last name :"+req.query.last_name);
   //res.end(JSON.stringify(response));
   res.end("Sent data are (GET): first name :"+req.query.first_name+" and last name :"+req.query.last_name);
})
var server = app.listen(5000, function () {

  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)

})
```

get_ex1.js

Open the page index.html and fill the entries:

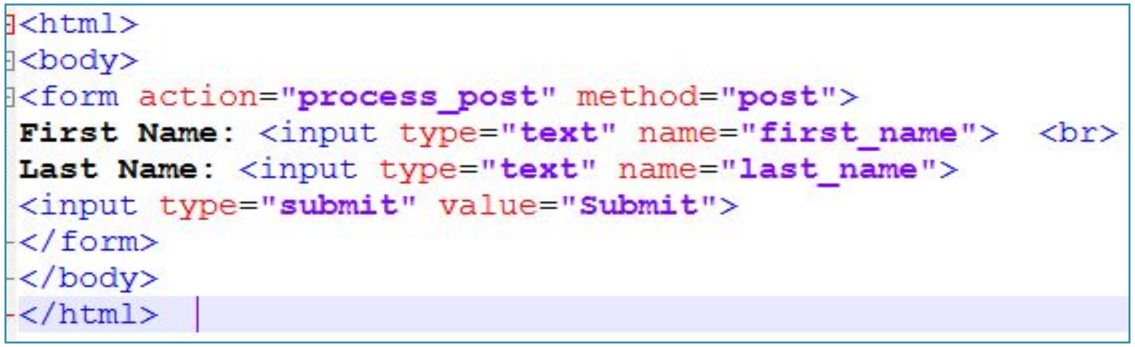
- **Fetch data in paragraph format**

get_ex2.js



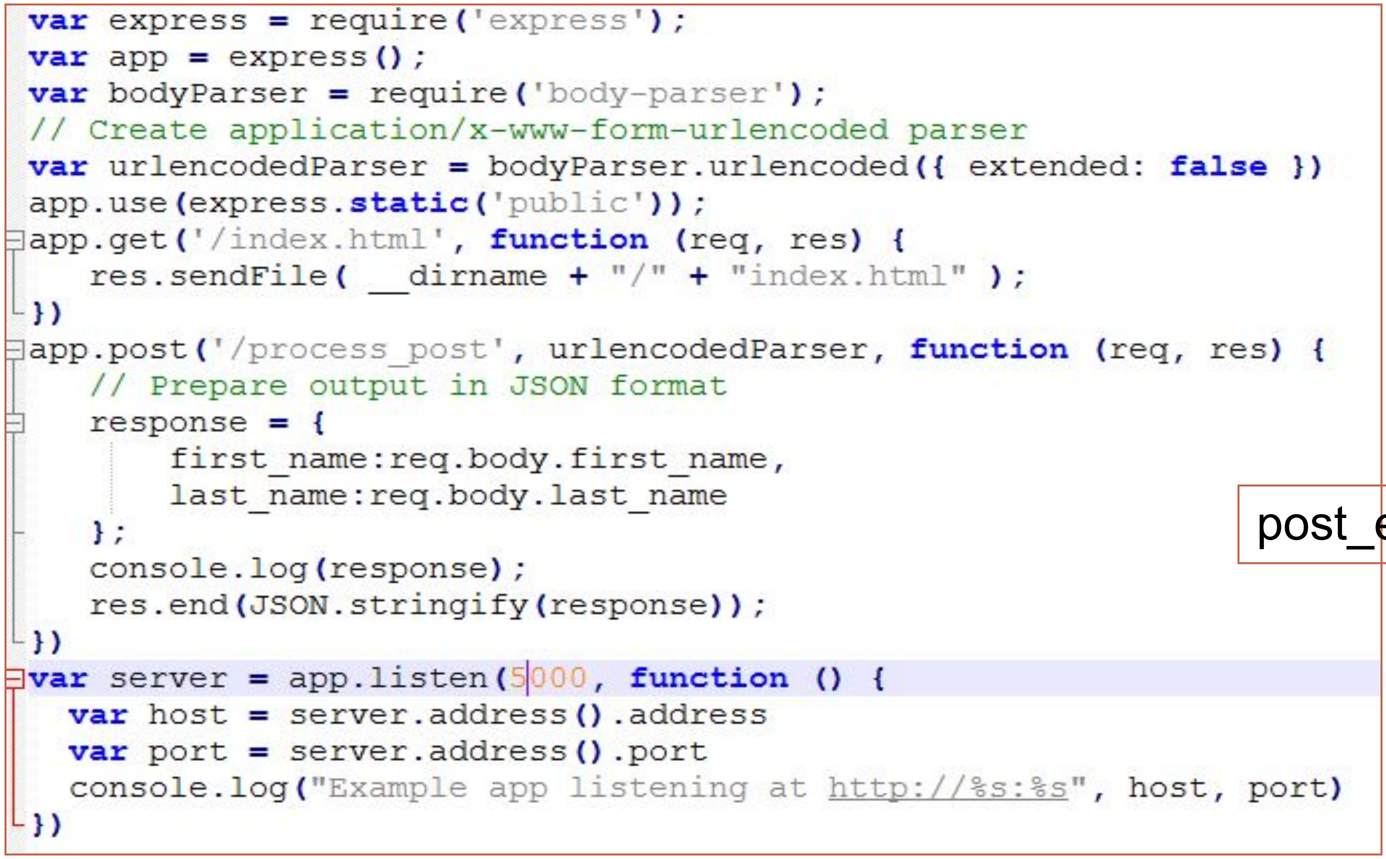
```
var express = require('express');
var app = express();
app.use(express.static('public'));

app.get('/index.html', function (req, res) {
   res.sendFile( __dirname + "/" + "index.html" );
})
app.get('/process_get', function (req, res) {
res.send('<p>First name: ' + req.query['first_name']+'</p><p>Last name: '+req.query['last_name']+'</p>');
})
var server = app.listen(5000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)

})
```

Express.js POST Method



```html
<html>
<body>
<form action="process_post" method="post">
First Name: <input type="text" name="first_name">  <br>
Last Name: <input type="text" name="last_name">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

index.html



```javascript
var express = require('express');
var app = express();
var bodyParser = require('body-parser');
// Create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })
app.use(express.static('public'));
app.get('/index.html', function (req, res) {
   res.sendFile( __dirname + "/" + "index.html" );
})
app.post('/process_post', urlencodedParser, function (req, res) {
   // Prepare output in JSON format
   response = {
       first_name:req.body.first_name,
       last_name:req.body.last_name
   };
   console.log(response);
   res.end(JSON.stringify(response));
})
var server = app.listen(5000, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

post_ex1.js