Express JS

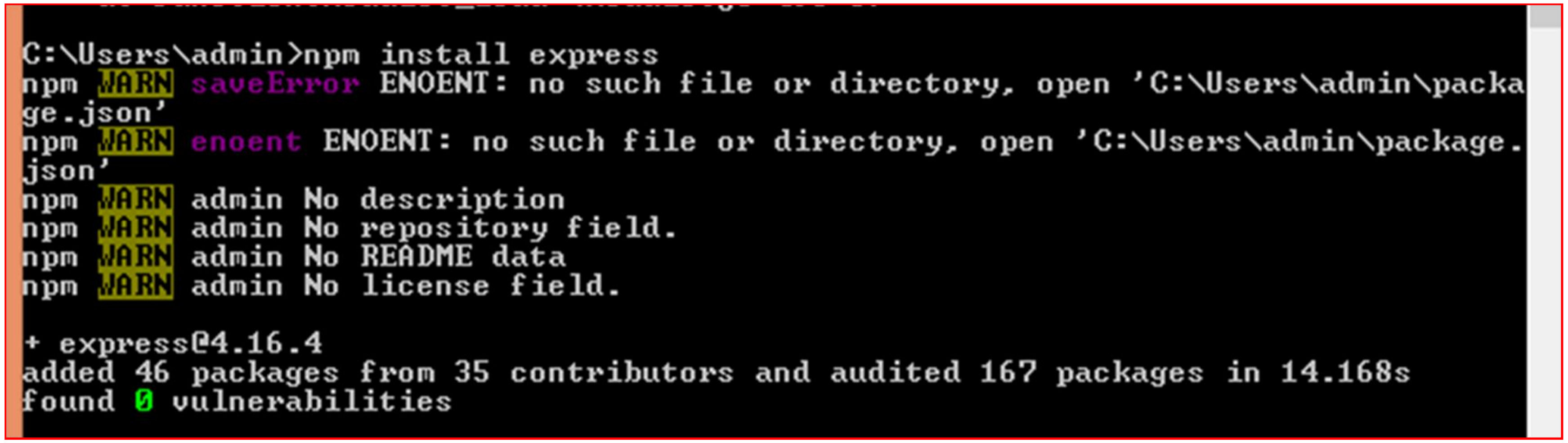
Express JS

- Node.js is an amazing tool for building networking services and applications.
- Express is a Node.js Web Framework. ExpressJS is a web application framework that provides you with a simple API to build websites, web apps and back ends.
- It is flexible as there are numerous modules available on **npm**, which can be directly plugged into Express.
- Express was developed by **TJ Holowaychuk** and is maintained by the Node.js foundation and numerous open source contributors.
- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It is an open source framework developed and maintained by the Node.js foundation.

Express framework:

- **Core features of Express framework**
  - 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.
- **Why use Express**
  - Ultra fast I/O
  - Asynchronous and single threaded
  - MVC like structure
  - Robust API makes *routing easy*

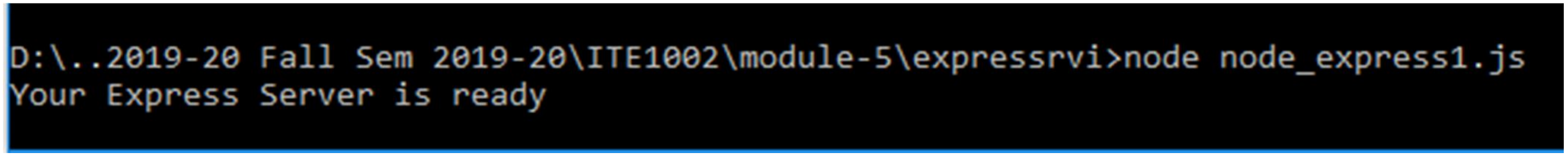
- npm install --save express
- To make our development process a lot easier, we will install a tool from npm, nodemon.
- **nodemon**→ This tool restarts our server as soon as we make a change in any of our files, otherwise we need to restart the server manually after each file modification. To install nodemon, use the following command –
- npm install -g nodemon



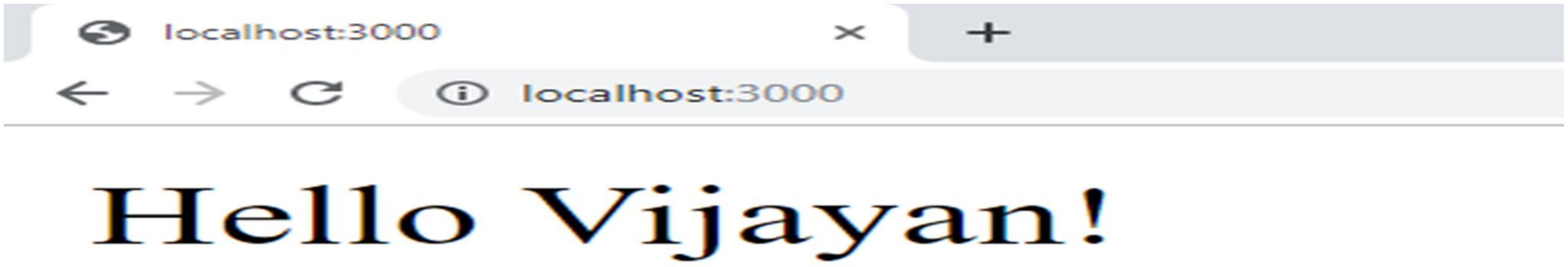
```
C:\Users\admin>npm install express
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\admin\packa
ge.json'
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\admin\package.
json'
npm WARN admin No description
npm WARN admin No repository field.
npm WARN admin No README data
npm WARN admin No license field.

+ express@4.16.4
added 46 packages from 35 contributors and audited 167 packages in 14.168s
found 0 vulnerabilities
```

C:\Users\admin\node_express1.js

```
const express = require('express')
const app = express()
app.get('/', function(req, res){ res.send('Hello Vijayan!')})
app.listen(3000, function(){ console.log(‘Your Express Server is ready')})
```



```
D:\..2019-20 Fall Sem 2019-20\ITE1002\module-5\expressrvi>node node_express1.js
Your Express Server is ready
```



Hello Vijayan!

- **app.get(route, callback)**
  - This function tells what to do when a **get** request at the given route is called.
  - The callback function has 2 parameters, ***request(req)*** and ***response(res)***.
  - The request **object(req)** represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, etc.
  - the response object represents the HTTP response that the Express app sends when it receives an HTTP request.
- **res.send()**
  - This function takes an object as input and it sends this to the requesting client. Here we are sending the string *"HelloWorld!"*.
- **app.listen(port, [host], [backlog], [callback]])**
  - This function binds and listens for connections on the specified host and port. Port is the only required parameter here
  - **Backlog→**The maximum number of queued pending connections. The default is 511.

Explanation

- First, we import the express package to the express value.
- We instantiate an application by calling its app() method. **app.get(route, callback)**
- Once we have the application object, we tell it to listen for GET requests on the / path, using the get() method.
- There is a method for every HTTP **verb**: get(), post(), put(), delete(), patch().
- Those methods accept a callback function, which is called when a request is started, and we need to handle it. An asynchronous function that is called when the server starts listening for requests.
- Express sends us two objects in this callback, which we called req and res, that represent the Request and the Response objects.
- Request is the HTTP request. It can give us all the info about that, including the request parameters, the headers, the body of the request, and more.Response is the HTTP response object that we'll send to the client.
- What we do in this callback is to send the 'Hello World!' string to the client, using the Response.send() method. This method sets that string as the body, and it closes the connection.
- The last line of the example actually starts the server, and tells it to listen on port 3000. We pass in a callback that is called when the server is ready to accept new requests.

Express routing

app.method(path, handler)

- **Method**→ get(), post() and all()
- Eg-

```
var express = require('express');
 var app = express();
app.get('/hello', function(req, res){ res.send("Hello World!"); });
app.post('/hello', function(req, res){
res.send("You just called the post method at '/hello'!\n"); });
 app.listen(3000);
```

- A special method, ***all***, is provided by Express to handle all types of http methods at a particular route using the same function
- Eg-

```
app.all('/test', function(req, res){
res.send("HTTP method doesn't have any effect on this
          route!"); });
```

Express - Routers

- To separate the routes from our main **index.js** file, we will use **Express.Router**.
- Create a new file called **things.js** and type the following in it.

```
 var express = require('express');
 var router = express.Router();
router.get('/', function(req, res){ res.send('GET route on things.');
 });
router.post('/', function(req, res){ res.send('POST route on
things.'); });
 //export this router to use in our index.js
  module.exports = router;
```

Express - Routers

- Now to use this router in our **index.js**, type in the following before the **app.listen** function call.

```
 var express = require('Express');
  var app = express();
  var things = require('./things.js');
//both index.js and things.js should be in same directory
 app.use('/things', things);
  app.listen(3000);
```

- The ***app.use*** function call on route **'/things'** attaches the **things** router with this route.
- Now whatever requests our app gets at the '/things', will be handled by our things.js router.
- The **'/'** route in things.js is actually a subroute of '/things'. Visit localhost:3000/things/

Express.js Request Object - req

- The express.js request object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

```
app.get('/', function (req, res) {
  // --
})
```

REQUEST OBJECT PROPERTIES

Property	**Description**
.app	holds a reference to the Express app object
.baseUrl	the base path on which the app responds
.body	contains the data submitted in the request body (must be parsed and populated manually before you can access it)
.cookies	contains the cookies sent by the request (needs the cookie-parser middleware)
.hostname	the server hostname
.ip	the server IP
.method	the HTTP method used
.params	the route named parameters
.path	the URL path
.protocol	the request protocol
.query	an object containing all the query strings used in the request
.secure	true if the request is secure (uses HTTPS)
.signedCookies	contains the signed cookies sent by the request (needs the cookie-parser middleware)
.xhr	true if the request is an XMLHttpRequest
.route	The currently-matched route, a string.

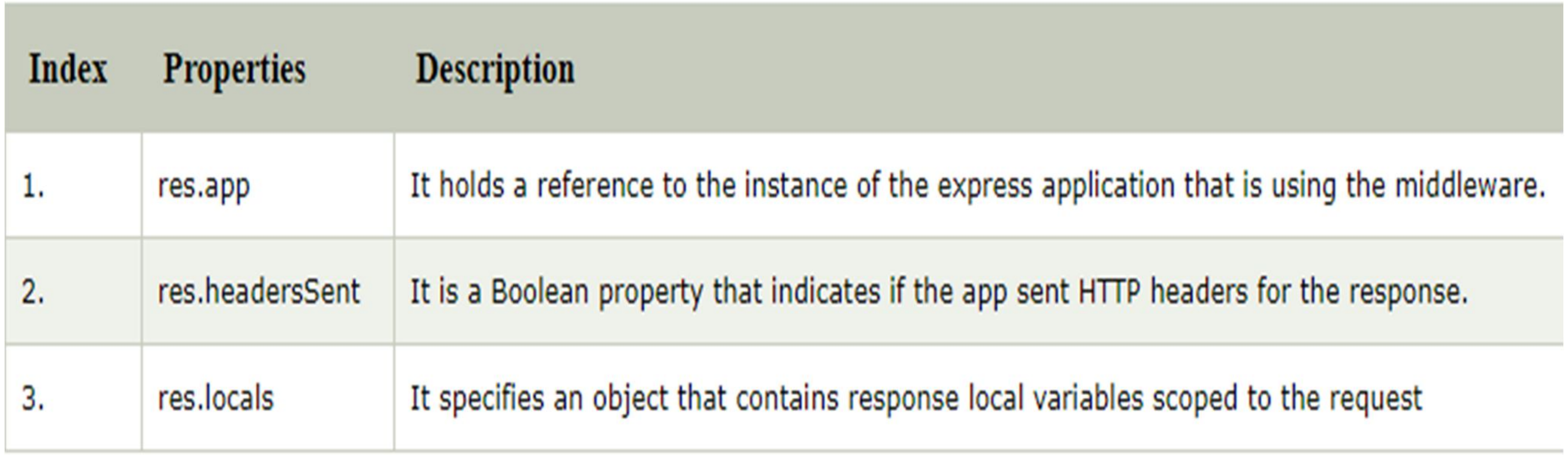
Express Request object methods

- **req.accepts (types)**
  - This method is used to check whether the specified content types are acceptable, based on the request's Accept HTTP header field.
  - req.accepts('html'); //=>?html?
  - req.accepts('text/html'); // => ?text/html?
- **req.get(field)**
  - This method returns the specified HTTP request header field.
- **req.is(type)**
  - This method returns true if the incoming request's "Content-Type" HTTP header field matches the MIME type specified by the type parameter.
  - req.is('html');
  - req.is('text/html');
  - req.is('text/*');
- **req.param(name [, defaultValue])**
  - This method is used to fetch the value of param name when present.
  - req.param('name') // ?name=sasha

Express Response object -res

- The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.
- What it does
  - It sends response back to the client browser.
  - It facilitates you to put new cookies value and that will write to the client browser (under cross domain rule).
  - Once you res.send() or res.redirect() or res.render(), you cannot do it again, otherwise, there will be uncaught error.

Response Object Properties and methods



Index	Properties	Description
1.	res.app	It holds a reference to the instance of the express application that is using the middleware.
2.	res.headersSent	It is a Boolean property that indicates if the app sent HTTP headers for the response.
3.	res.locals	It specifies an object that contains response local variables scoped to the request

- res.append(field [, value]) →This method appends the specified value to the HTTP response header field.
  - **Eg**

    res.append('Link', ['<http://localhost/>', '<http://localhost:3000/>']);
    res.append('Set-Cookie', 'foo=bar; Path=/; HttpOnly');
- res.attachment([filename]) →This method is used to send a file as an attachment in the HTTP response.
  - examples –

    res.attachment('path/to/logo.png'); res.cookie(name, value [, options])

Response Object methods

- res.cookie(name, value [, options]) →This method is used to set cookie name to value. The value parameter may be a string or object converted to JSON.
  - Eg. –

  res.cookie('name', 'tobi', { domain: '.example.com', path: '/admin', secure: true }); res.cookie('cart', { items: [1,2,3] });

  res.cookie('cart', { items: [1,2,3] }, { maxAge: 900000 });
- res.clearCookie(name [, options]) →This method is used to clear the cookie specified by name. Following are a few examples –
  - res.cookie('name', 'tobi', { path: '/admin' }); res.clearCookie('name', { path: '/admin' });

Response Object methods

- res.download(path [, filename] [, fn]) →This method is used to transfer the file at path as an "attachment". Typically, browsers will prompt the user for download. Following are a few examples –
  - res.download('/report-12345.pdf');
  - res.download('/report-12345.pdf', 'report.pdf');
- res.end([data] [, encoding])→ This method is used to end the response process. Following are a few examples –
  - res.end();   res.status(404).end();
- res.get(field) →This method is used to return the HTTP response header specified by field. Here is an examples –
  - res.get('Content-Type');
- res.json([body]) →This method is used to send a JSON response. Following are a few examples –
  - res.json(null)
  - res.json({ user: 'tobi' })
  - res.status(500).json({ error: 'message' })

Response Object methods

- res.links(links)→ This method is used to join the links provided as properties of the parameter to populate the response's Link HTTP header field. Following are a few examples –
  - res.links ({ next: 'http://api.example.com/users?page=2',
    last: 'http://api.example.com/users?page=5' });
- res.location(path) →This method is used to set the response Location HTTP header field based on the specified path parameter. Following are a few examples –
  - res.location('/foo/bar');
    res.location('foo/bar'); res.location('http://example.com');
- res.redirect([status,] path) →This method is used to redirect to the URL dervied from the specified path, with specified HTTP status code status. Following are a few examples –
  - res.redirect('/foo/bar'); res.redirect('http://example.com');
    res.redirect(301, 'http://example.com');

Response Object methods

- res.send([body]) →This method is used to send the HTTP response. Following are a few examples –
  - res.send(new Buffer('whoop')); res.send({ some: 'json' }); res.send('<p>some html</p>');res.sendFile(path [, options] [, fn])
- res.sendFile(path [, options] [, fn]) →This method is used to transfer the file at the given path. Sets the Content-Type response HTTP header field based on the filename’s extension. Here is an example –
  - res.sendFile(fileName, options, function (err) { // ... });
- res.sendStatus(statusCode)→ This method is used to set the response HTTP status code to statusCode and send its string representation as the response body. Following are a few examples –
  - res.sendStatus(200); // equivalent to res.status(200).send('OK') res.sendStatus(403); // equivalent to res.status(403).send('Forbidden')

Response Object methods

- res.set(field [, value])→ This method is used to set the response's HTTP header field to value. Following are a few examples –
  - res.set('Content-Type', 'text/plain');
  - res.set ({ 'Content-Type': 'text/plain', 'Content-Length': '123', 'ETag': '12345' })
- res.status(code)→ This method is used to set the HTTP status for the response. Following are a few examples –
  - res.status(403).end(); res.status(400).send('Bad Request'); res.status(404).sendFile('/absolute/path/to/404.png');
- res.type(type)→ This method is used to set the Content-Type HTTP header to the MIME type. Following are a few examples –
  - res.type('.html'); // => 'text/html' res.type('html'); // => 'text/html' res.type('json'); // => 'application/json' res.type('application/json'); // => 'application/json' res.type('png'); // => image/png: