Node JS

Node File System and Path

Node.js File System

- Node.js includes **fs** module to access physical file system.

- The fs module is responsible for all the asynchronous or synchronous file I/O operations.

Reading File

Use fs.readFile() method to read the physical file asynchronously.

```
fs.readFile(fileName [,options], callback)
```

- filename: Full path and name of the file as a string.
- options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
- callback: A function with two parameters err and fd. This will get called when readFile operation completes.

Reading From File

MENTORLABS℠

Writing File

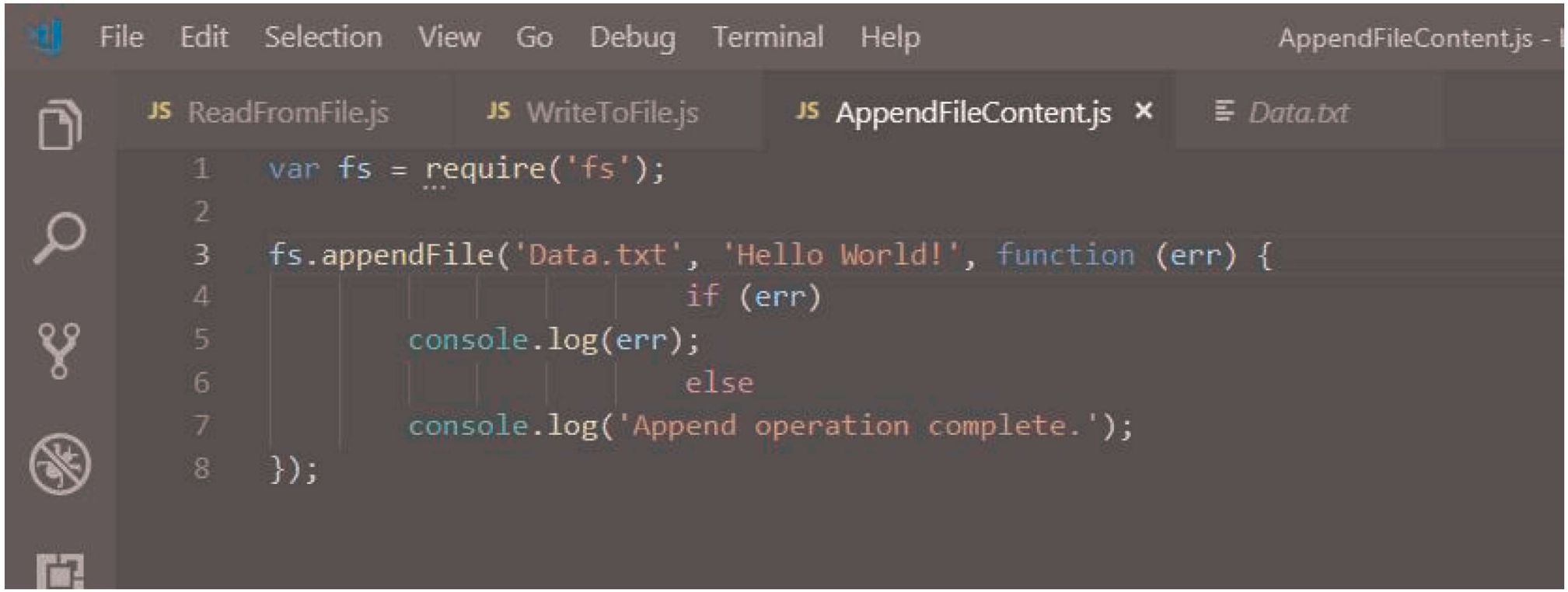
- Use fs.writeFile() method to write data to a file. ‘
- If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

```
fs.writeFile(filename, data[, options], callback)
```

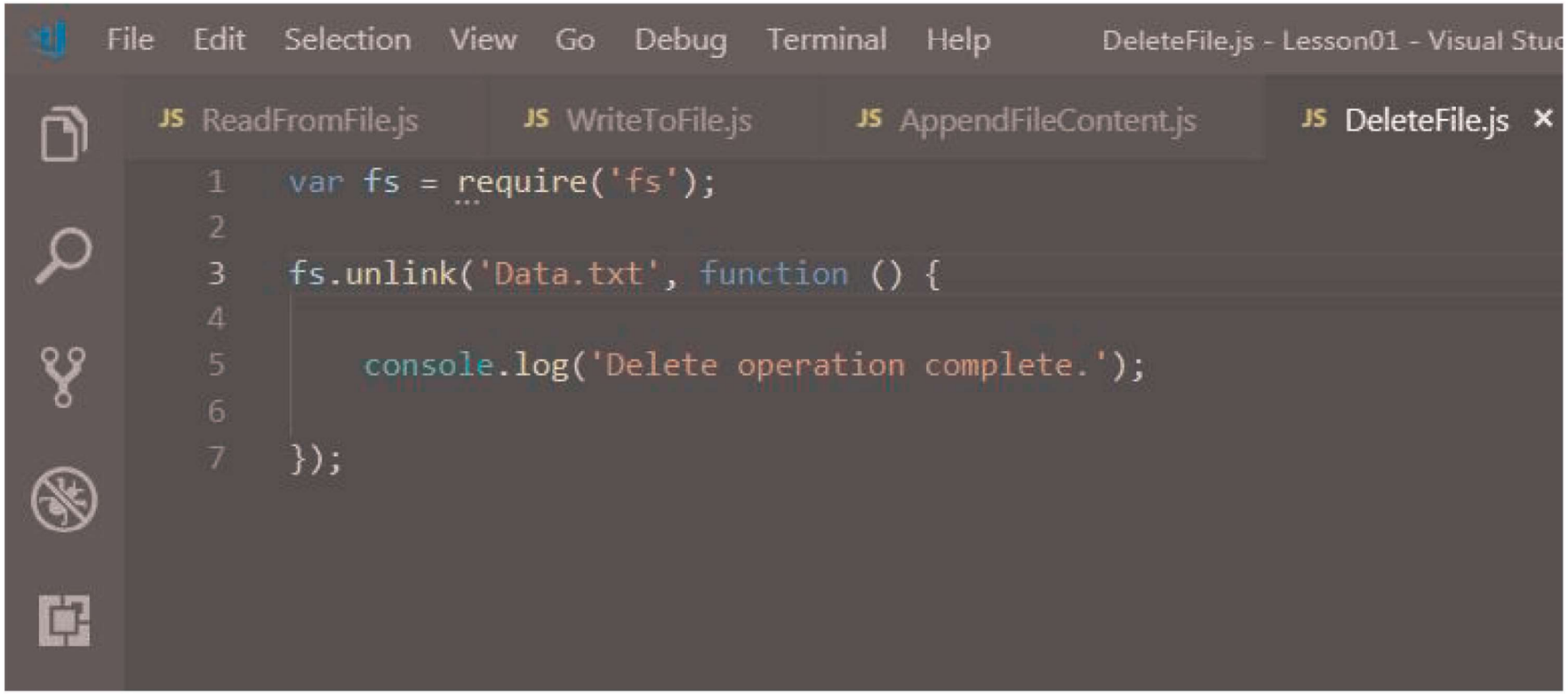
- filename: Full path and name of the file as a string.
- Data: The content to be written in a file.
- options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".
- callback: A function with two parameters err and fd. This will get called when write operation completes.

Writing to File

Append File Content



Delete File



MENTORLABS℠

Node.js Path

The Node.js path module is used to handle and transform files paths.

```
var path =  require ("path")
```

Index	Method	Description
1.	path.normalize(p)	It is used to normalize a string path, taking care of '..' and '.' parts.
2.	path.join([path1][, path2][, ...])	It is used to join all arguments together and normalize the resulting path.
3.	path.resolve([from ...], to)	It is used to resolve an absolute path.
4.	path.isabsolute(path)	It determines whether path is an absolute path. an absolute path will always resolve to the same location, regardless of the working directory.
5.	path.relative(from, to)	It is used to solve the relative path from "from" to "to".
6.	path.dirname(p)	It return the directory name of a path. It is similar to the unix dirname command

Relative Path Versus Absolute Path

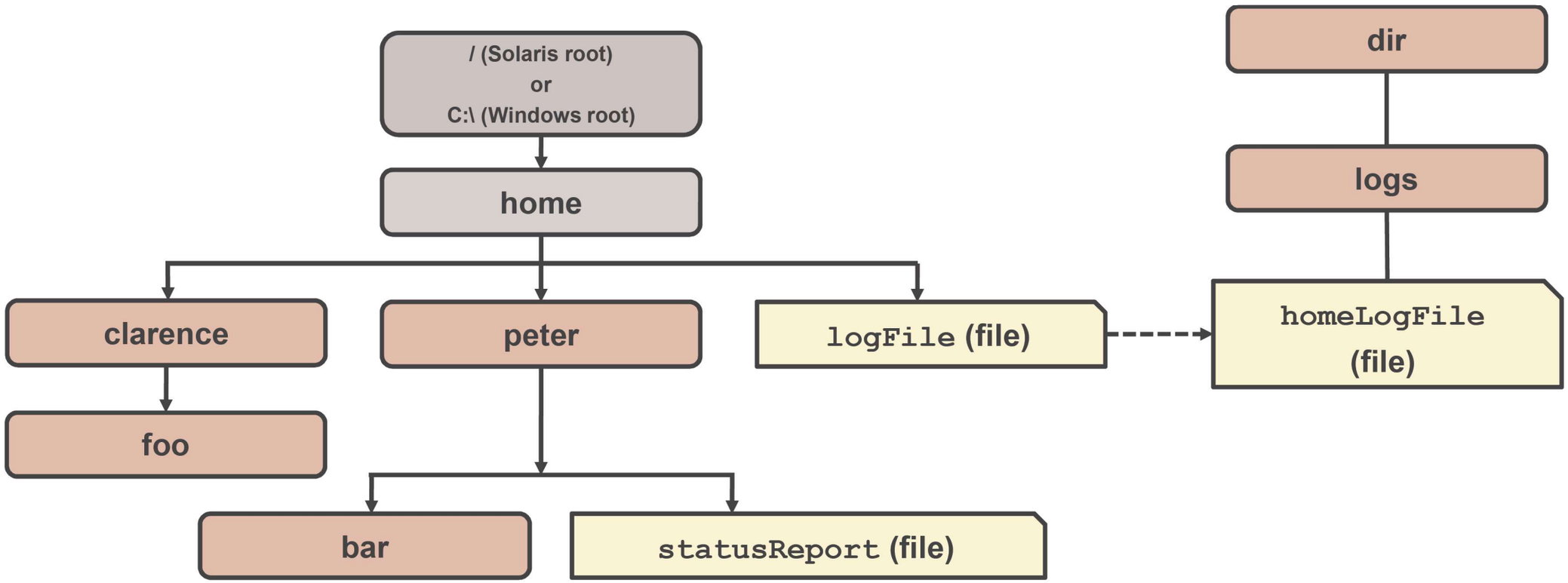
- A path is either *relative* or *absolute*.
- An absolute path always contains the root element and the complete directory list required to locate the file.
- Example:

```
...
/home/peter/statusReport
...
```

- A relative path must be combined with another path in order to access a file.
- Example:

```
...
clarence/foo
...
```

Symbolic Links



Removing Redundancies from a `Path`

- Many file systems use "`.`" notation to denote the current directory and "`..`" to denote the parent directory.
- The following examples both include redundancies:

```
/home/./clarence/foo
/home/peter/../clarence/foo
```

- The `normalize` method removes any redundant elements, which includes any "`.`" or "`directory/..`" occurrences.
- Example:

```
Path p = Paths.get("/home/peter/../clarence/foo");
Path normalizedPath = p.normalize();
```

```
/home/clarence/foo
```

Joining Two Paths

- The `resolve` method is used to combine two paths.
- Example:

```
Path p1 = Paths.get("/home/clarence/foo");
p1.resolve("bar");     // Returns /home/clarence/foo/bar
```

- Passing an absolute path to the `resolve` method returns the passed-in path.

```
Paths.get("foo").resolve("/home/clarence"); // Returns /home/clarence
```

Important method of fs module

Method	Description
fs.readFile(fileName [,options], callback)	Reads existing file.
fs.writeFile(filename, data[, options], callback)	Writes to the file. If file exists then overwrite the content otherwise creates new file.
fs.open(path, flags[, mode], callback)	Opens file for reading or writing.
fs.rename(oldPath, newPath, callback)	Renames an existing file.
fs.chown(path, uid, gid, callback)	Asynchronous chown.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.link(srcpath, dstpath, callback)	Links file asynchronously.
fs.symlink(destination, path[, type], callback)	Symlink asynchronously.

fs.rmdir(path, callback)	Renames an existing directory.
fs.mkdir(path[, mode], callback)	Creates a new directory.
fs.readdir(path, callback)	Reads the content of the specified directory.
fs.utimes(path, atime, mtime, callback)	Changes the timestamp of the file.
fs.exists(path, callback)	Determines whether the specified file exists or not.
fs.access(path[, mode], callback)	Tests a user's permissions for the specified file.
fs.appendFile(file, data[, options], callback)	Appends new content to the existing file.

Node Debugger

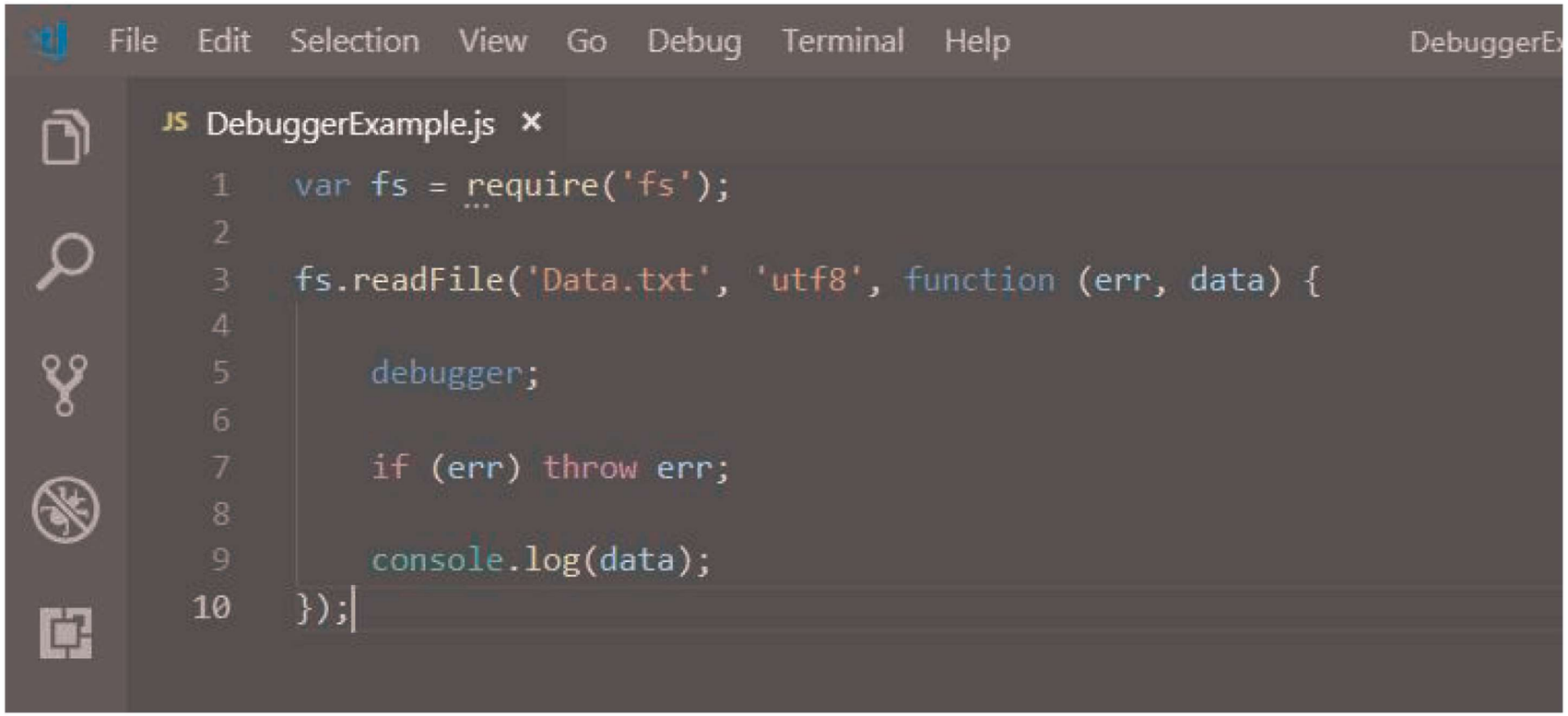
Debug Node.js Application

You can debug Node.js application using various tools including following:

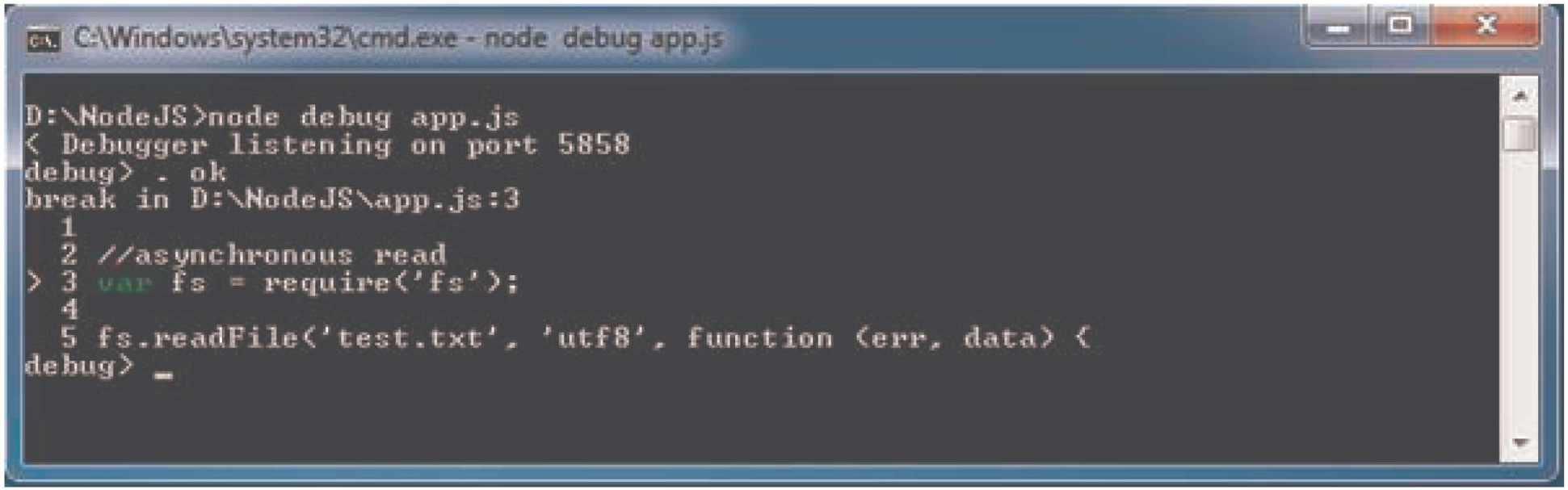
1. Core Node.js debugger
2. Node Inspector
3. Built-in debugger in IDEs

Core Node.js Debugger

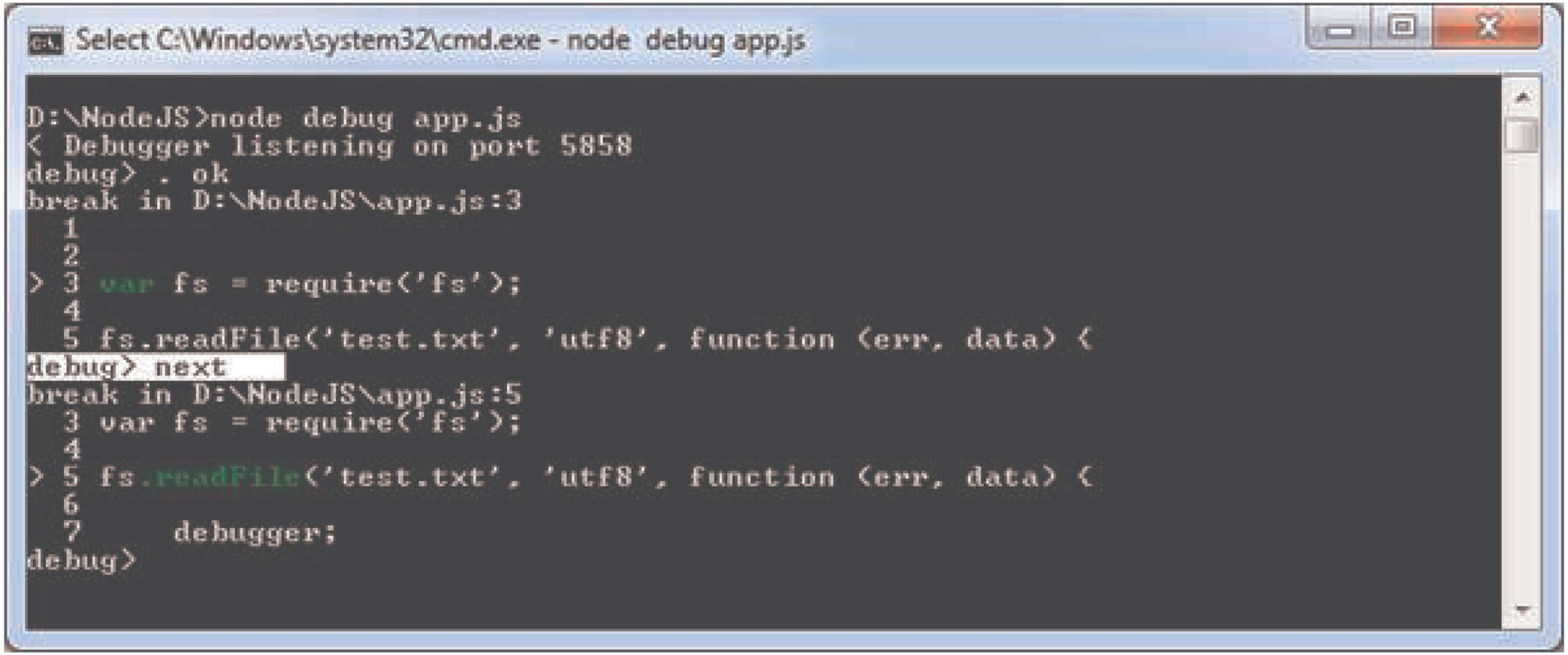
- Node.js provides built-in non-graphic debugging tool that can be used on all platforms. It provides different commands for debugging Node.js application.



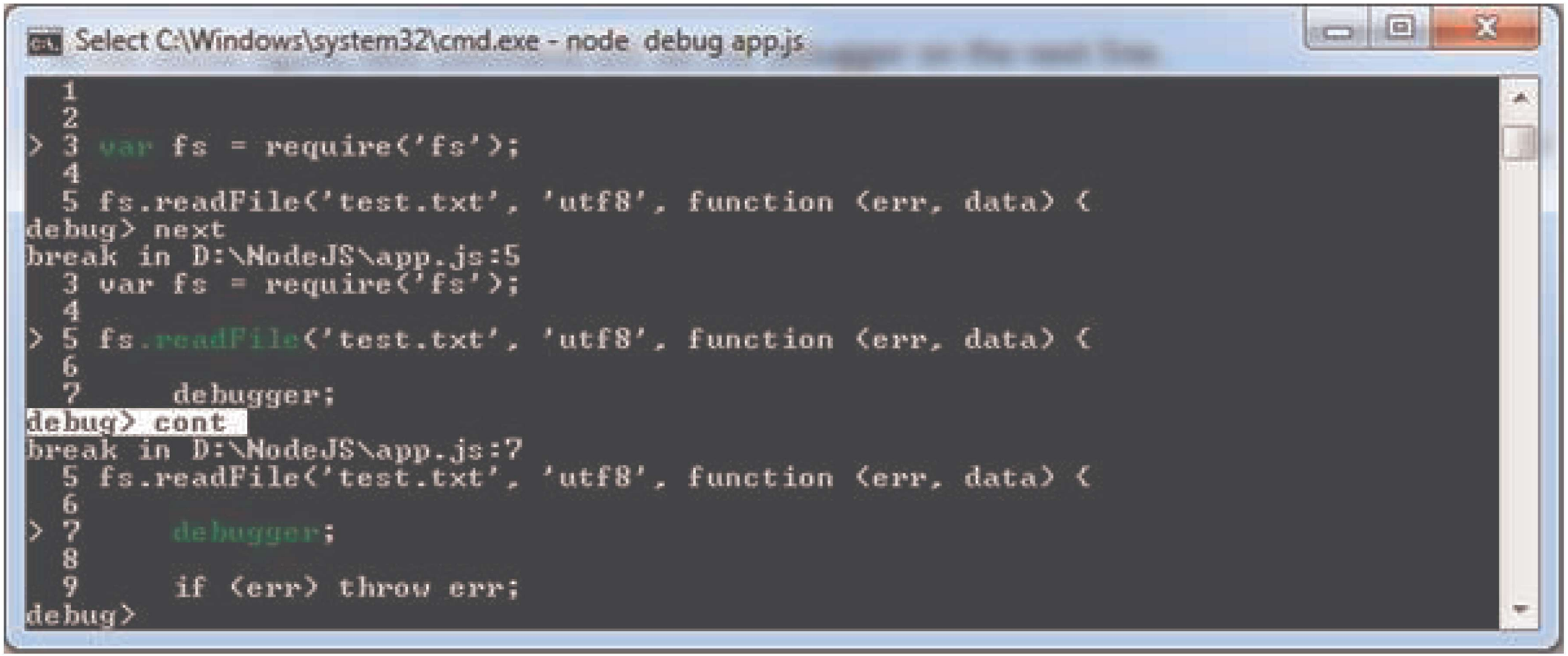
- Write debugger in your JavaScript code where you want debugger to stop.


C:\Windows\system32\cmd.exe - node debug app.js
D:\NodeJS>node debug app.js
< Debugger listening on port 5858
debug> . ok
break in D:\NodeJS\app.js:3
1
2 //asynchronous read
> 3 var fs = require('fs');
4
5 fs.readFile('test.txt', 'utf8', function (err, data) {
debug> _

Use next to move on the next statement.



Use cont to stop the execution at next "debugger", if any.



Command	Description
next	Stop at the next statement.
cont	Continue execute and stop at the debugger statement if any.
step	Step in function.
out	Step out of function.
watch	Add the expression or variable into watch.
watcher	See the value of all expressions and variables added into watch.
Pause	Pause running code.

Node Events and Event Emitter

Node.js Events

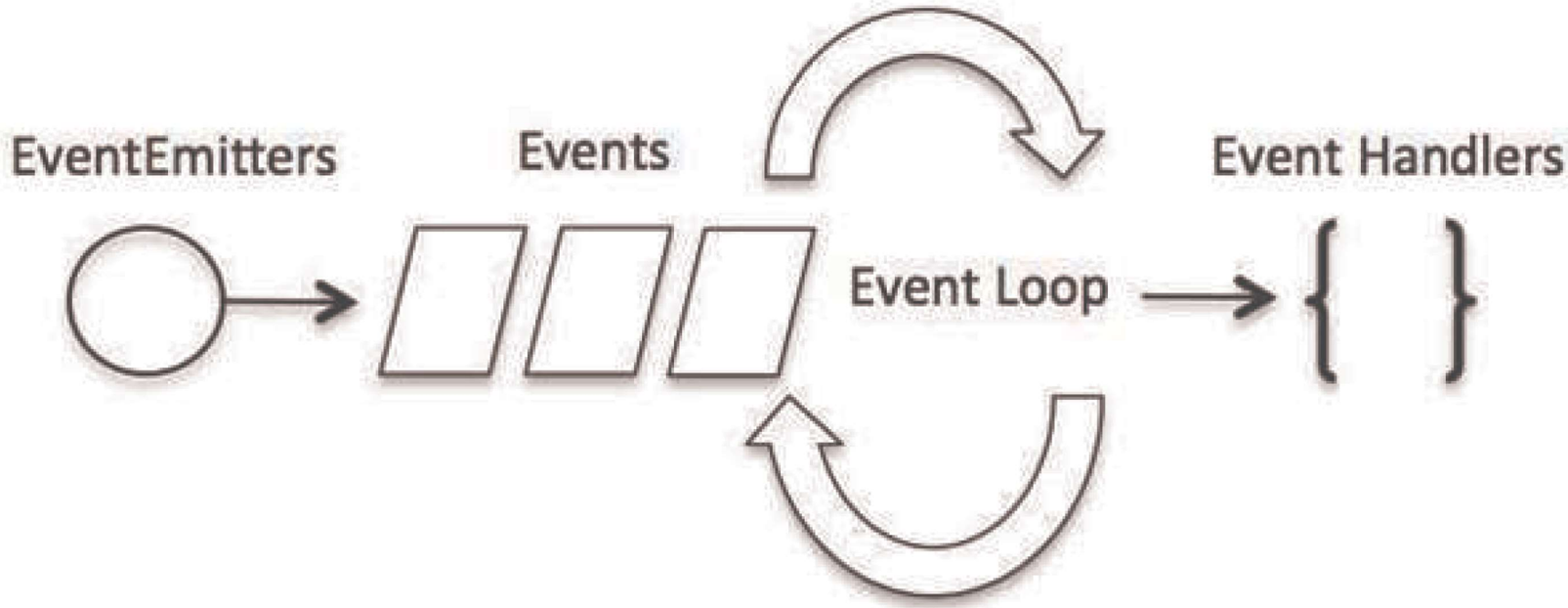
- In Node.js applications, Events and Callbacks concepts are used to provide concurrency.
- As Node.js applications are single threaded and every API of Node js are asynchronous. So it uses async function to maintain the concurrency. Node uses observer pattern.
- Node thread keeps an event loop and after the completion of any task, it fires the corresponding event which signals the event listener function to get executed.

Event Driven Programming

- Node.js uses event driven programming. It means as soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for event to occur.

- It is the one of the reason why Node.js is pretty fast compared to other similar technologies.

- There is a main loop in the event driven application that listens for events, and then triggers a callback function when one of those events is detected.


EventEmitters
Events
Event Handlers
Event Loop

Difference between Events and Callbacks:

- Events and Callbacks look similar but the differences lies in the fact that callback functions are called when an asynchronous function returns its result where as event handling works on the observer pattern.

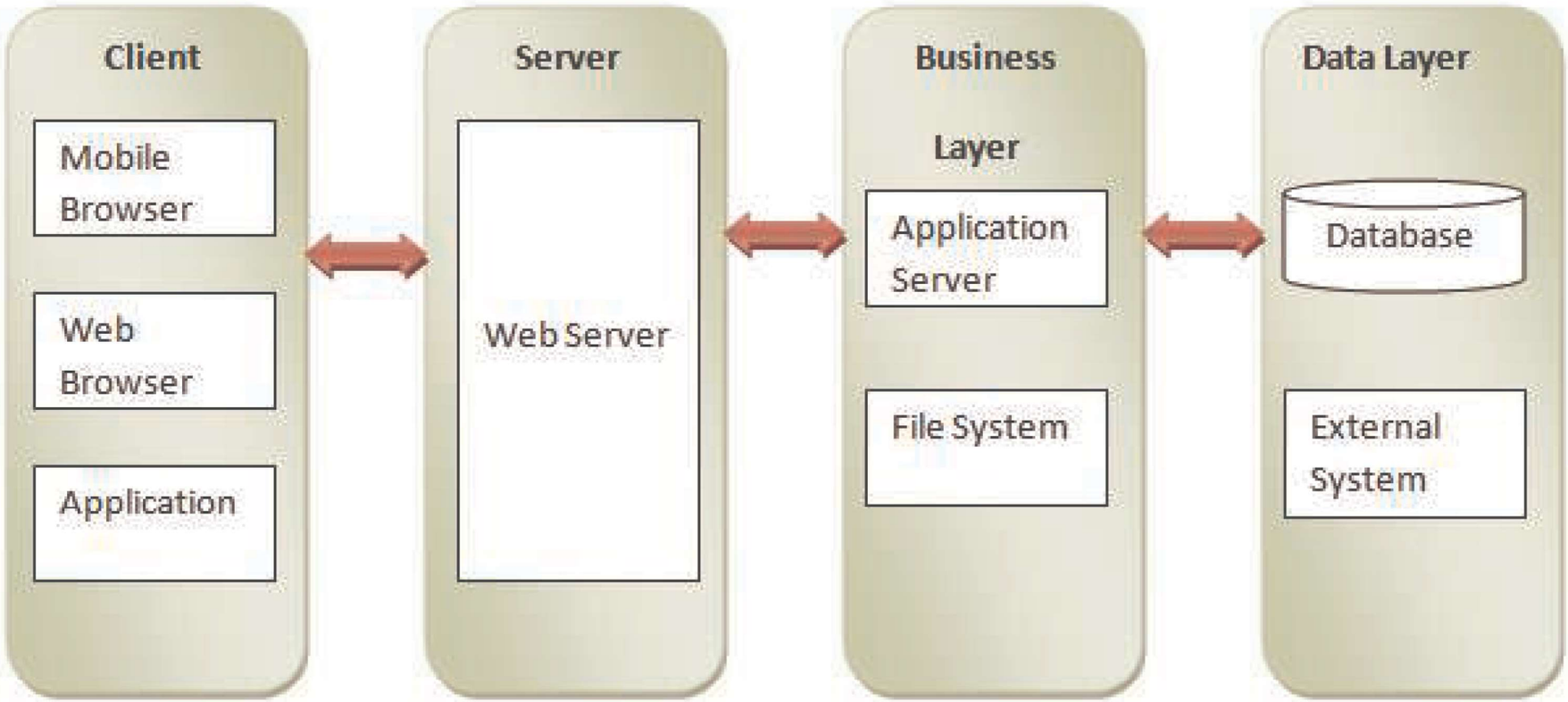
- Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through events module and EventEmitter class which is used to bind events and event listeners.

Node.js EventEmitter

- Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.

Node Web Server

Server Architecture

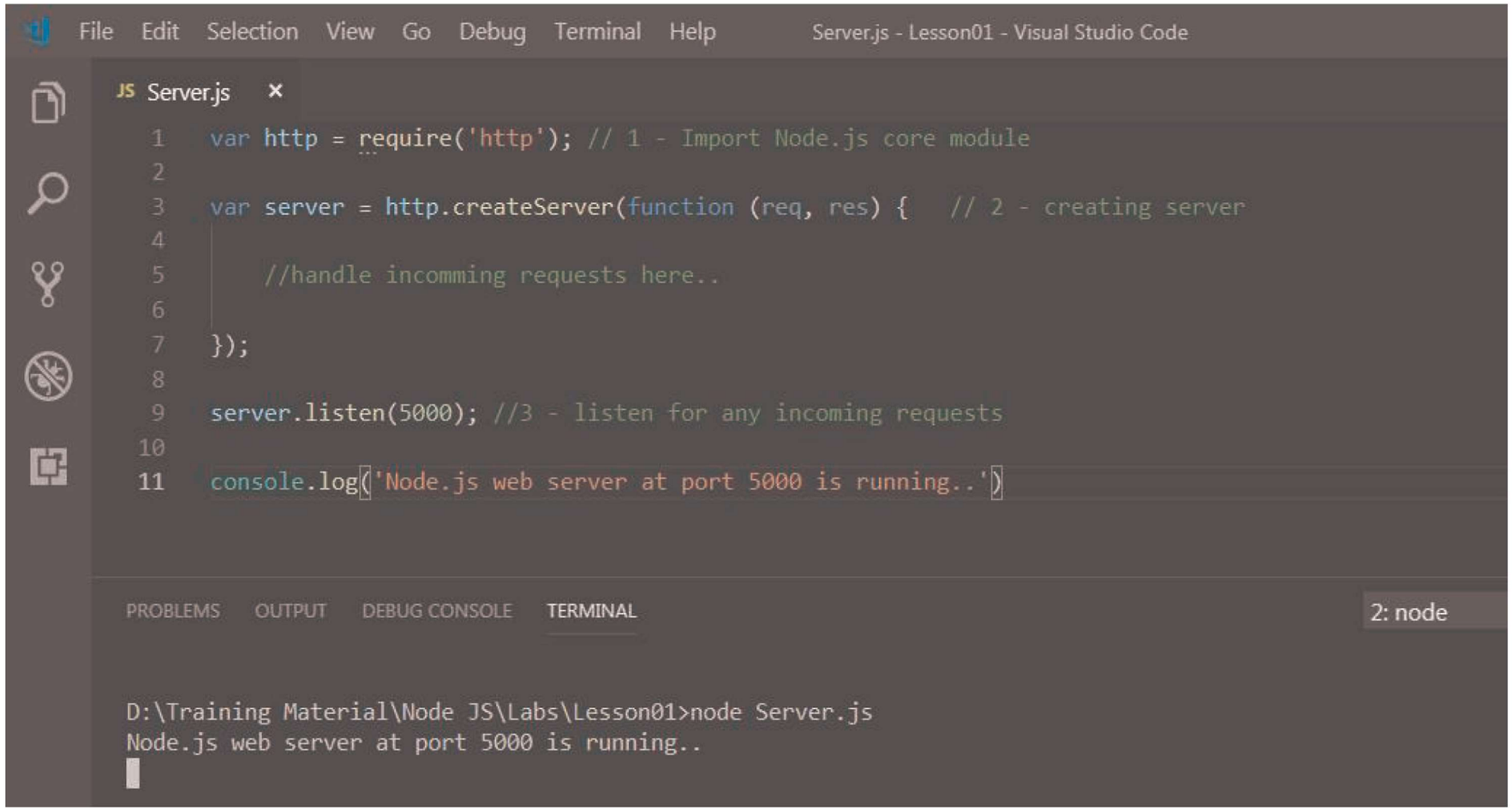


Node JS Web Server

- To access web pages of any web application, you need a web server. The web server will handle all the http requests for the web application e.g IIS is a web server for ASP.NET web applications and Apache is a web server for PHP or Java web applications.
- Node.js provides capabilities to create your own web server which will handle HTTP requests asynchronously. You can use IIS or Apache to run Node.js web application but it is recommended to use Node.js web server.

Create Node.js Web Server

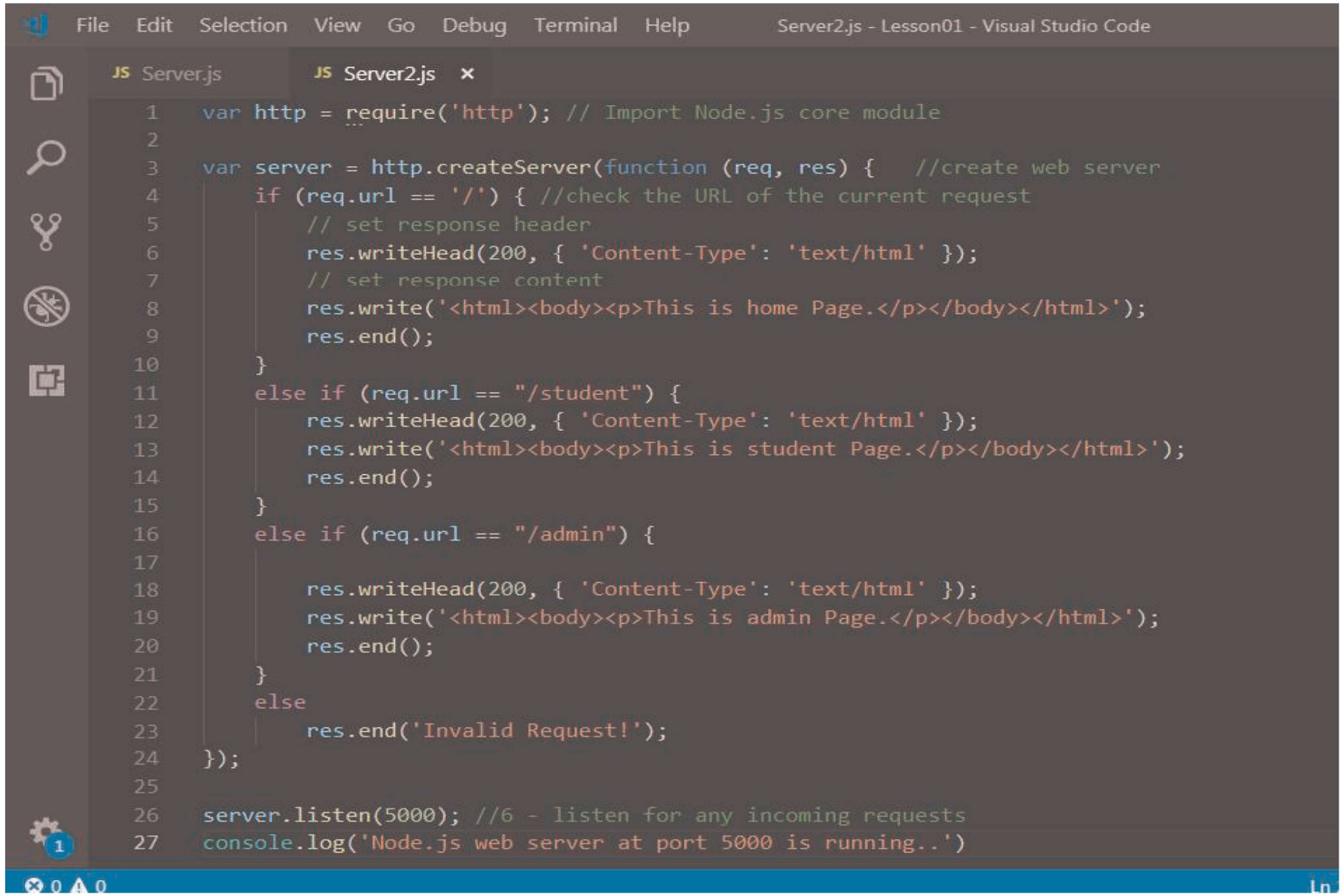
- Node.js makes it easy to create a simple web server that processes incoming requests asynchronously.



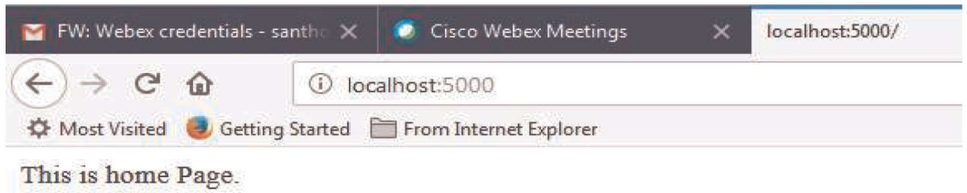
Handle HTTP Request

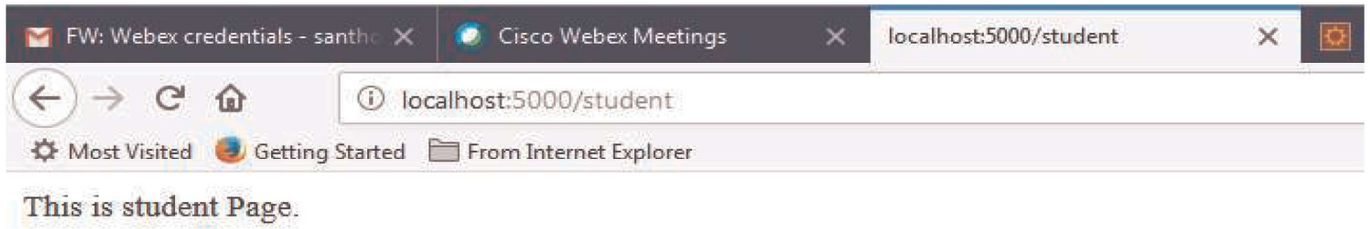
- The http.createServer() method includes request and response parameters which is supplied by Node.js.
- The request object can be used to get information about the current HTTP request e.g., url, request header, and data.
- The response object can be used to send a response for a current HTTP request.

Server2.js


File Edit Selection View Go Debug Terminal Help
Server2.js - Lesson01 - Visual Studio Code
Server.js
Server2.js
var http = require('http'); // Import Node.js core module
var server = http.createServer(function (req, res) { //create web server
if (req.url == '/') { //check the URL of the current request
// set response header
res.writeHead(200, { 'Content-Type': 'text/html' });
// set response content
res.write('<html><body> This is home Page. </body></html>');
res.end();
}
else if (req.url == "/student") {
res.writeHead(200, { 'Content-Type': 'text/html' });
res.write('<html><body> This is student Page. </body></html>');
res.end();
}
else if (req.url == "/admin") {
res.writeHead(200, { 'Content-Type': 'text/html' });
res.write('<html><body> This is admin Page. </body></html>');
res.end();
}
else
res.end('Invalid Request!');
});
server.listen(5000); //6 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')

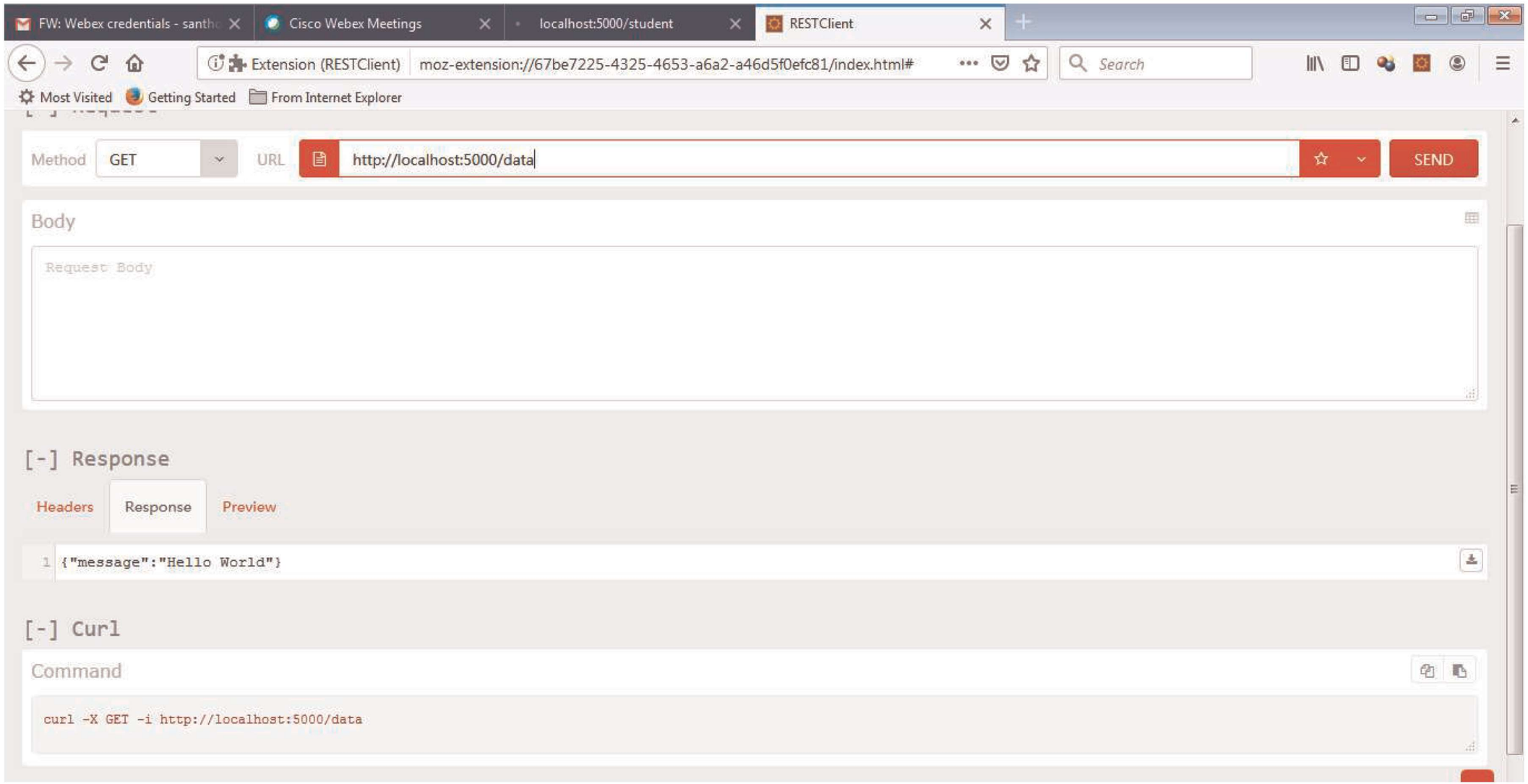
Execution


FW: Webex credentials - santh
Cisco Webex Meetings
localhost:5000/
localhost:5000
Most Visited
Getting Started
From Internet Explorer
This is home Page.


FW: Webex credentials - santh
Cisco Webex Meetings
localhost:5000/student
localhost:5000/student
Most Visited
Getting Started
From Internet Explorer
This is student Page.


[-] Response
Headers
Response
Preview
Status Code : 200 OK
connection : keep-alive
content-type : text/html
date : Thu, 24 Oct 2019 04:29:25 GMT
transfer-encoding : chunked

Sending JSON Response


FW: Webex credentials - santh
Cisco Webex Meetings
localhost:5000/student
RESTClient
Extension (RESTClient)
moz-extension://67be7225-4325-4653-a6a2-a46d5f0efc81/index.html#
Search
Most Visited
Getting Started
From Internet Explorer
Method
GET
URL
http://localhost:5000/data
SEND
Body
Request Body
[-] Response
Headers
Response
Preview
1 {"message":"Hello World"}
[-] Curl
Command
curl -X GET -i http://localhost:5000/data

MENTORLABS℠

Node JS Frameworks

Frameworks for Node.js

- There are various third party open-source frameworks available in Node Package Manager which makes Node.js application development faster and easy.
- You can choose an appropriate framework as per your application requirements.

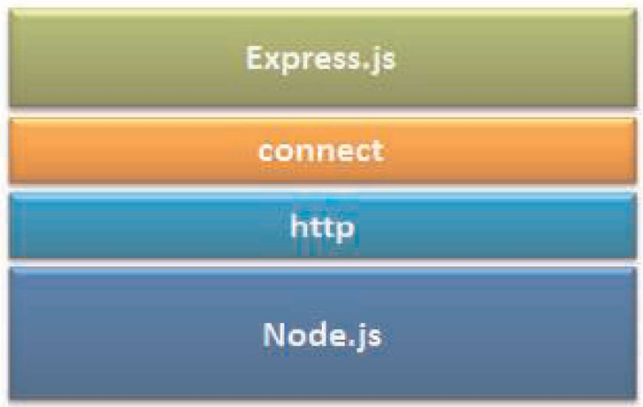
Open-Source Framework	Description
Express.js	Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. This is the most popular framework as of now for Node.js.
Geddy	Geddy is a simple, structured web application framework for Node.js based on MVC architecture.
Locomotive	Locomotive is MVC web application framework for Node.js. It supports MVC patterns, RESTful routes, and convention over configuration, while integrating seamlessly with any database and template engine. Locomotive builds on Express, preserving the power and simplicity you've come to expect from Node.
Koa	Koa is a new web framework designed by the team behind Express, which aims to be a smaller, more expressive, and more robust foundation for web applications and APIs.

Total.js	Totaljs is free web application framework for building web sites and web applications using JavaScript, HTML and CSS on Node.js
Hapi.js	Hapi is a rich Node.js framework for building applications and services.
Keystone	Keystone is the open source framework for developing database-driven websites, applications and APIs in Node.js. Built on Express and MongoDB.
Derbyjs	Derby support single-page apps that have a full MVC structure, including a model provided byÂ Racer, a template and styles based view, and controller code with application logic and routes.
Sails.js	Sails makes it easy to build custom, enterprise-grade Node.js apps. It is designed to emulate the familiar MVC pattern of frameworks like Ruby on Rails, but with support for the requirements of modern apps: data-driven APIs with a scalable, service-oriented architecture. It's especially good for building chat, realtime dashboards, or multiplayer games; but you can use it for any web application project - top to bottom.

Meteor	Meteor is a complete open source platform for building web and mobile apps in pure JavaScript.
Mojito	This HTML5 framework for the browser and server from Yahoo offers direct MVC access to the server database through the local routines. One clever feature allows the code to migrate. If the client can't run JavaScript for some reason, Mojito will run it on the server -- a convenient way to handle very thin clients.
Restify	Restify is a node.js module built specifically to enable you to build correct REST web services.
Loopback	Loopback is an open-source Node.js API framework.
ActionHero	actionhero.js is a multi-transport Node.JS API Server with integrated cluster capabilities and delayed tasks.
Frisby	Frisby is a REST API testing framework built on node.js and Jasmine that makes testing API endpoints easy, fast, and fun.
Chocolate.js	Chocolate is a simple webapp framework built on Node.js using Coffeescript.

Express.js

- "Express is a fast, unopinionated minimalist web framework for Node.js“
- Express.js is a web application framework for Node.js. It provides various features that make web application development fast and easy which otherwise takes more time using only Node.js.
- Express.js is based on the Node.js middleware module called ***connect*** which in turn uses **http** module. So, any middleware which is based on connect will also work with Express.js.



Advantages of Express.js

1. Makes Node.js web application development fast and easy.
2. Easy to configure and customize.
3. Allows you to define routes of your application based on HTTP methods and URLs.
4. Includes various middleware modules which you can use to perform additional tasks on request and response.
5. Easy to integrate with different template engines like Jade, Vash, EJS etc.
6. Allows you to define an error handling middleware.
7. Easy to serve static files and resources of your application.
8. Allows you to create REST API server.
9. Easy to connect with databases such as MongoDB, Redis, MySQL

Install Express.js

- You can install express.js using npm. The following command will install latest version of express.js globally on your machine so that every Node.js application on your machine can use it.

```
npm install -g express
```

- The following command will install latest version of express.js local to your project folder.

```
C:\MyNodeJSApp> npm install express --save
```

Express.js Web Application

Express.js provides an easy way to create web server and render HTML pages for different HTTP requests by configuring routes for your application.

Web Server

- First of all, import the Express.js module and create the web server

app.js: Express.js Web Server

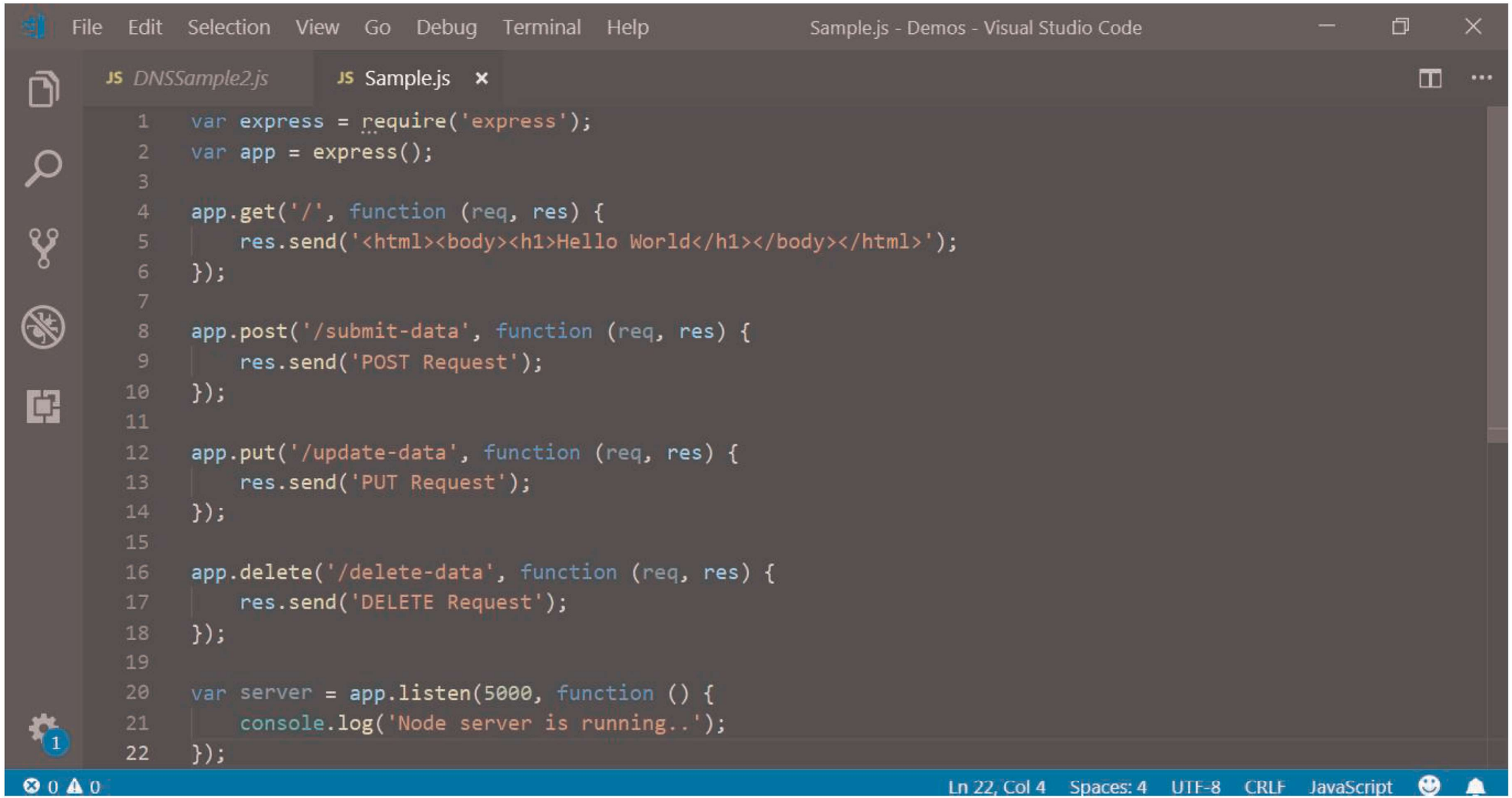
```
var express = require('express');
var app = express();

// define routes here..

var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

Details

- we imported Express.js module using require() function. The express module returns a function. This function returns an object which can be used to configure Express application
- The app object includes methods for routing HTTP requests, configuring middleware, rendering HTML views and registering a template engine.
- The app.listen() function creates the Node.js web server at the specified host and port. It is identical to Node's http.Server.listen() method.

Configure Routes

- Use app object to define different routes of your application. The app object includes get(), post(), put() and delete() methods to define routes for
  - HTTP GET
  - HTTP POST
  - HTTP PUT
  - HTTP DELETE requests respectively


File Edit Selection View Go Debug Terminal Help
Sample.js - Demos - Visual Studio Code
DNSSample2.js
Sample.js
var express = require('express');
var app = express();
app.get('/', function (req, res) {
res.send('<html><body><h1>Hello World</h1></body></html>');
});
app.post('/submit-data', function (req, res) {
res.send('POST Request');
});
app.put('/update-data', function (req, res) {
res.send('PUT Request');
});
app.delete('/delete-data', function (req, res) {
res.send('DELETE Request');
});
var server = app.listen(5000, function () {
console.log('Node server is running..');
});
0 0
Ln 22, Col 4 Spaces: 4 UTF-8 CRLF JavaScript

- app.get(), app.post(), app.put() and app.delete() methods define routes for HTTP GET, POST, PUT, DELETE respectively.

- The first parameter is a path of a route which will start after base URL.

- The callback function includes request and response object which will be executed on each request.


localhost:5000
localhost:5000
Hello World

Handle POST Request

- TO handle HTTP POST request and get data from the submitted form.
- First, create Index.html file in the root folder of your application and write the following HTML code in it.

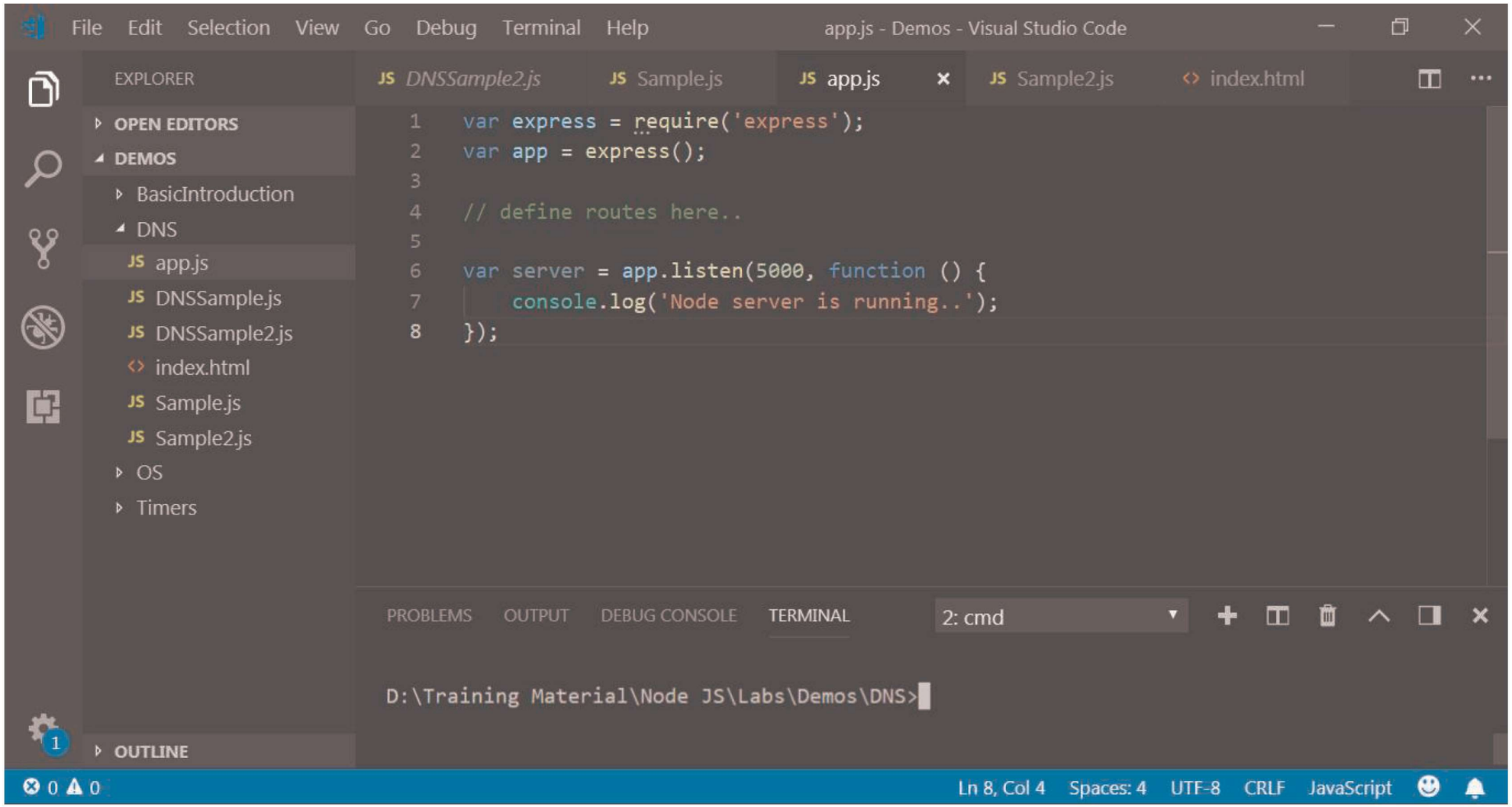
DNSSample2.js | Sample.js | Form.html

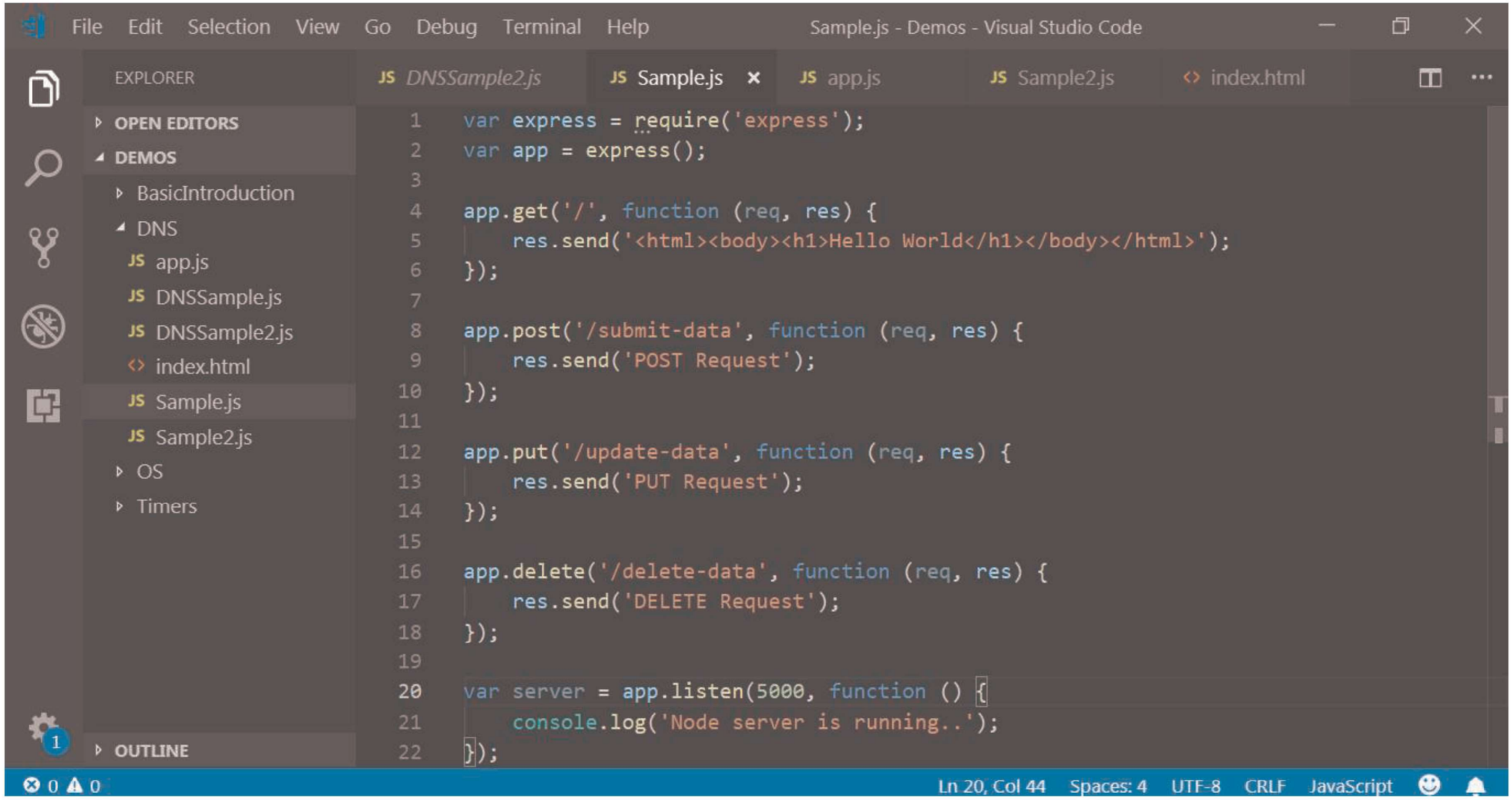
```
1   <!DOCTYPE html>
2
3   <html xmlns="http://www.w3.org/1999/xhtml">
4   <head>
5       <meta charset="utf-8" />
6       <title></title>
7   </head>
8   <body>
9       <form action="/submit-student-data" method="post">
10          First Name: <input name="firstName" type="text" /> <br />
11          Last Name: <input name="lastName" type="text" /> <br />
12          <input type="submit" />
13      </form>
14  </body>
15  </html>
```

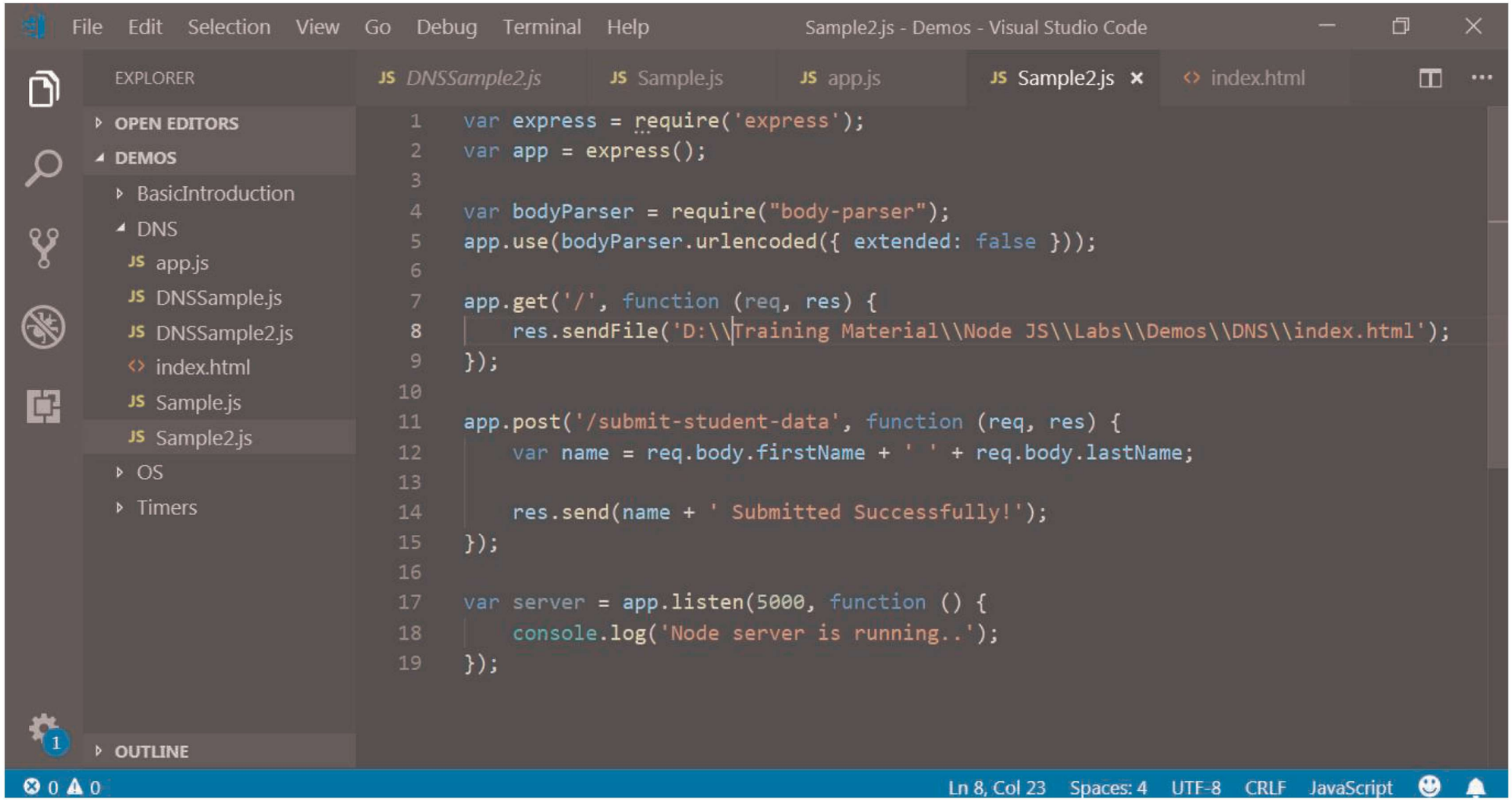
Body Parser

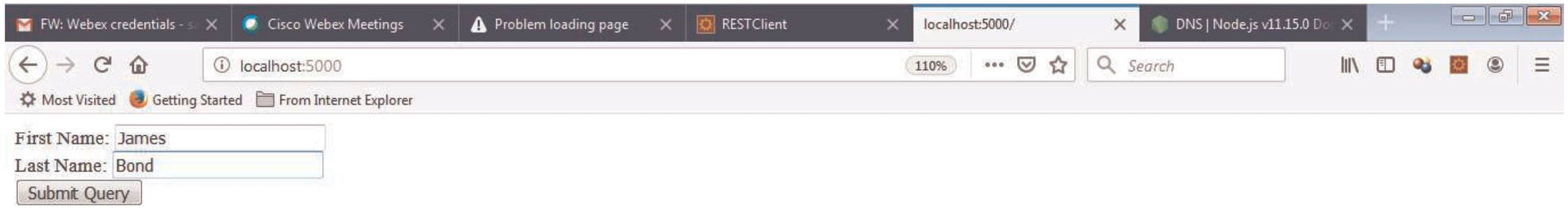
- To handle HTTP POST request in Express.js version 4 and above, you need to install middleware module called body-parser. The middleware was a part of Express.js earlier but now you have to install it separately.
- This body-parser module parses the JSON, buffer, string and url encoded data submitted using HTTP POST request. Install body-parser using NPM

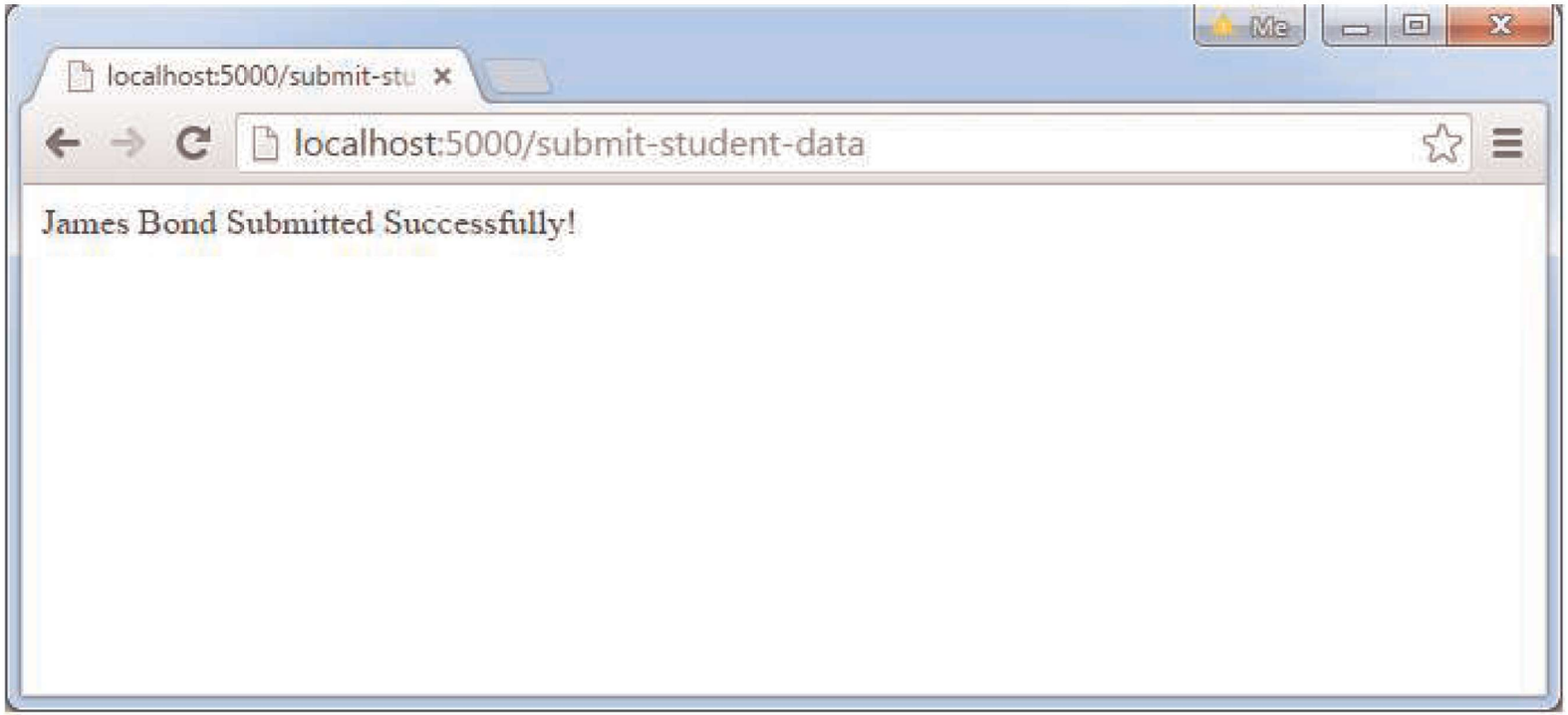
```
npm install body-parser --save
```


File Edit Selection View Go Debug Terminal Help
app.js - Demos - Visual Studio Code
EXPLORER
OPEN EDITORS
DEMOS
BasicIntroduction
DNS
app.js
DNSSample.js
DNSSample2.js
index.html
Sample.js
Sample2.js
OS
Timers
OUTLINE
DNSSample2.js
Sample.js
app.js
Sample2.js
index.html
1 var express = require('express');
2 var app = express();
3
4 // define routes here..
5
6 var server = app.listen(5000, function () {
7 console.log('Node server is running..');
8 });
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: cmd
D:\Training Material\Node JS\Labs\Demos\DNS>
0 0
Ln 8, Col 4 Spaces: 4 UTF-8 CRLF JavaScript


File Edit Selection View Go Debug Terminal Help
Sample.js - Demos - Visual Studio Code
EXPLORER
OPEN EDITORS
DEMOS
BasicIntroduction
DNS
app.js
DNSSample.js
DNSSample2.js
index.html
Sample.js
Sample2.js
OS
Timers
OUTLINE
DNSSample2.js
Sample.js
app.js
Sample2.js
index.html
var express = require('express');
var app = express();
app.get('/', function (req, res) {
res.send('<html><body><h1>Hello World</h1></body></html>');
});
app.post('/submit-data', function (req, res) {
res.send('POST Request');
});
app.put('/update-data', function (req, res) {
res.send('PUT Request');
});
app.delete('/delete-data', function (req, res) {
res.send('DELETE Request');
});
var server = app.listen(5000, function () {
console.log('Node server is running..');
});
0 0
Ln 20, Col 44
Spaces: 4
UTF-8
CRLF
JavaScript


File Edit Selection View Go Debug Terminal Help
Sample2.js - Demos - Visual Studio Code
EXPLORER
OPEN EDITORS
DEMOS
BasicIntroduction
DNS
app.js
DNSSample.js
DNSSample2.js
index.html
Sample.js
Sample2.js
OS
Timers
OUTLINE
DNSSample2.js
Sample.js
app.js
Sample2.js
index.html
1 var express = require('express');
2 var app = express();
3
4 var bodyParser = require("body-parser");
5 app.use(bodyParser.urlencoded({ extended: false }));
6
7 app.get('/', function (req, res) {
8 res.sendFile('D:\\Training Material\\Node JS\\Labs\\Demos\\DNS\\index.html');
9 });
10
11 app.post('/submit-student-data', function (req, res) {
12 var name = req.body.firstName + ' ' + req.body.lastName;
13
14 res.send(name + ' Submitted Successfully!');
15 });
16
17 var server = app.listen(5000, function () {
18 console.log('Node server is running..');
19 });
0 0
Ln 8, Col 23 Spaces: 4 UTF-8 CRLF JavaScript


FW: Webex credentials -
Cisco Webex Meetings
Problem loading page
RESTClient
localhost:5000/
DNS | Node.js v11.15.0 Do
localhost:5000
110%
Search
Most Visited
Getting Started
From Internet Explorer
First Name: James
Last Name: Bond
Submit Query


localhost:5000/submit-stu
localhost:5000/submit-student-data
James Bond Submitted Successfully!

Data Access In Node JS

Data Access in Node.js

- Node.js supports all kinds of databases no matter if it is a relational database or NoSQL database. However, NoSQL databases like MongoDb are the best fit with Node.js.
- To access the database from Node.js, you first need to install drivers for the database you want to use.

Relational databases and respective drivers.

Relational Databases	Driver	NPM Command
MS SQL Server	mssql	npm install mssql
Oracle	oracledb	npm install oracledb
MySQL	MySQL	npm install mysql
PostgreSQL	pg	npm install pg
SQLite	node-sqlite3	npm install node-sqlite

NoSQL databases and respective drives

NoSQL Databases	Driver	NPM Command
MongoDB	mongodb	npm install mongodb
Cassandra	cassandra-driver	npm install cassandra-driver
LevelDB	leveldb	npm install level levelup leveldown
RavenDB	ravendb	npm install ravendb
Neo4j	neo4j	npm install neo4j
Redis	redis	npm install redis
CouchDB	nano	npm install nano

- Install Node.js from nodejs.org.
- Install node-oracledb using the `npm` package manager, which is included in Node.js. If you are behind a firewall, you may need to set the proxy with `npm config set proxy http://myproxy.example.com:80/` .
    - Many users will be able to use a pre-built node-oracledb binary:
        - Add `oracledb` to your `package.json` dependencies or run `npm install oracledb` . This installs from the npm registry.

      Windows users will require the Visual Studio 2017 Redistributable.

Access MongoDB in Node.js

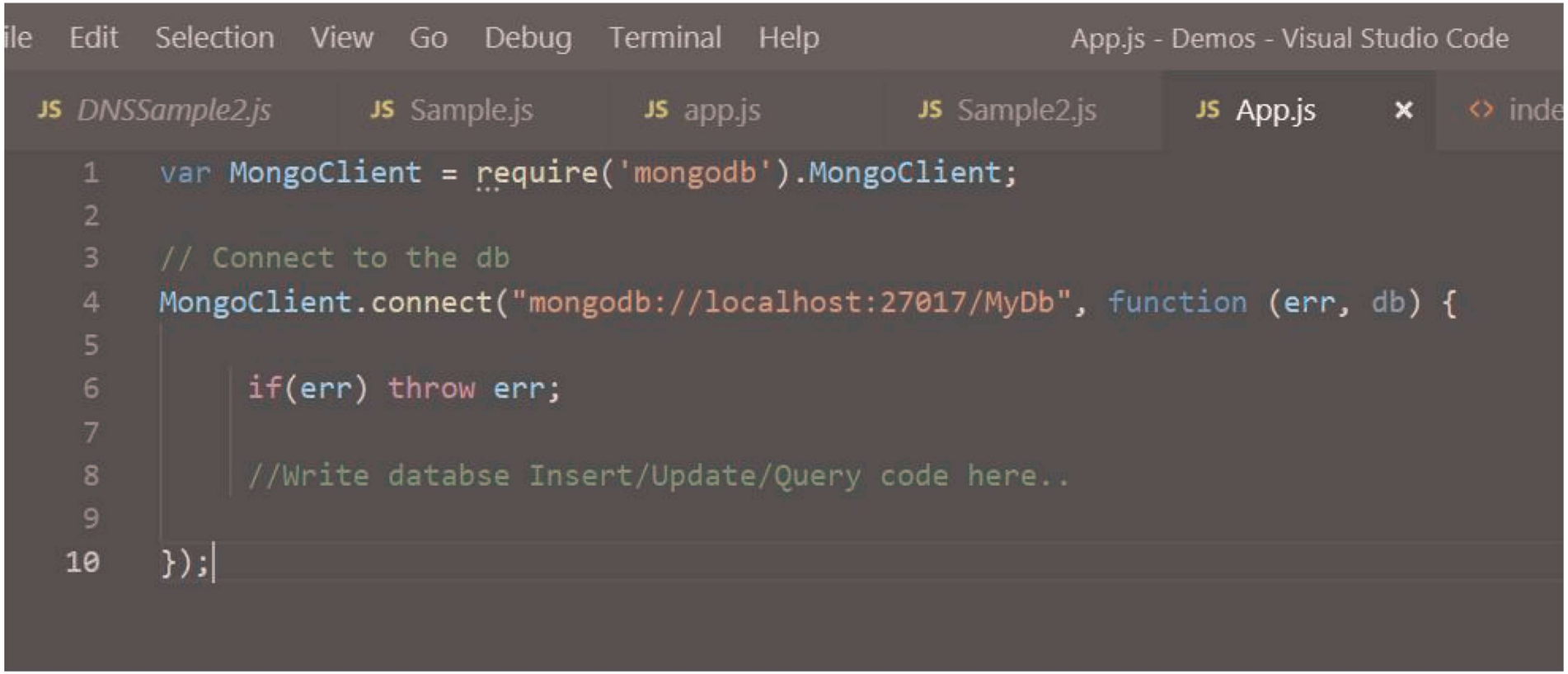
- In order to access MongoDB database, we need to install MongoDB drivers. To install native mongodb drivers using NPM, open command prompt and write the following command to install MongoDB driver in your application.

```
npm install mongodb --save
```

- This will include mongodb folder inside node_modules folder. Now, start the MongoDB server using the following command. (Assuming that your MongoDB database is at C:\MyNodeJSConsoleApp\MyMongoDB folder.)

```
mongod -dbpath C:\MyNodeJSConsoleApp\MyMongoDB
```

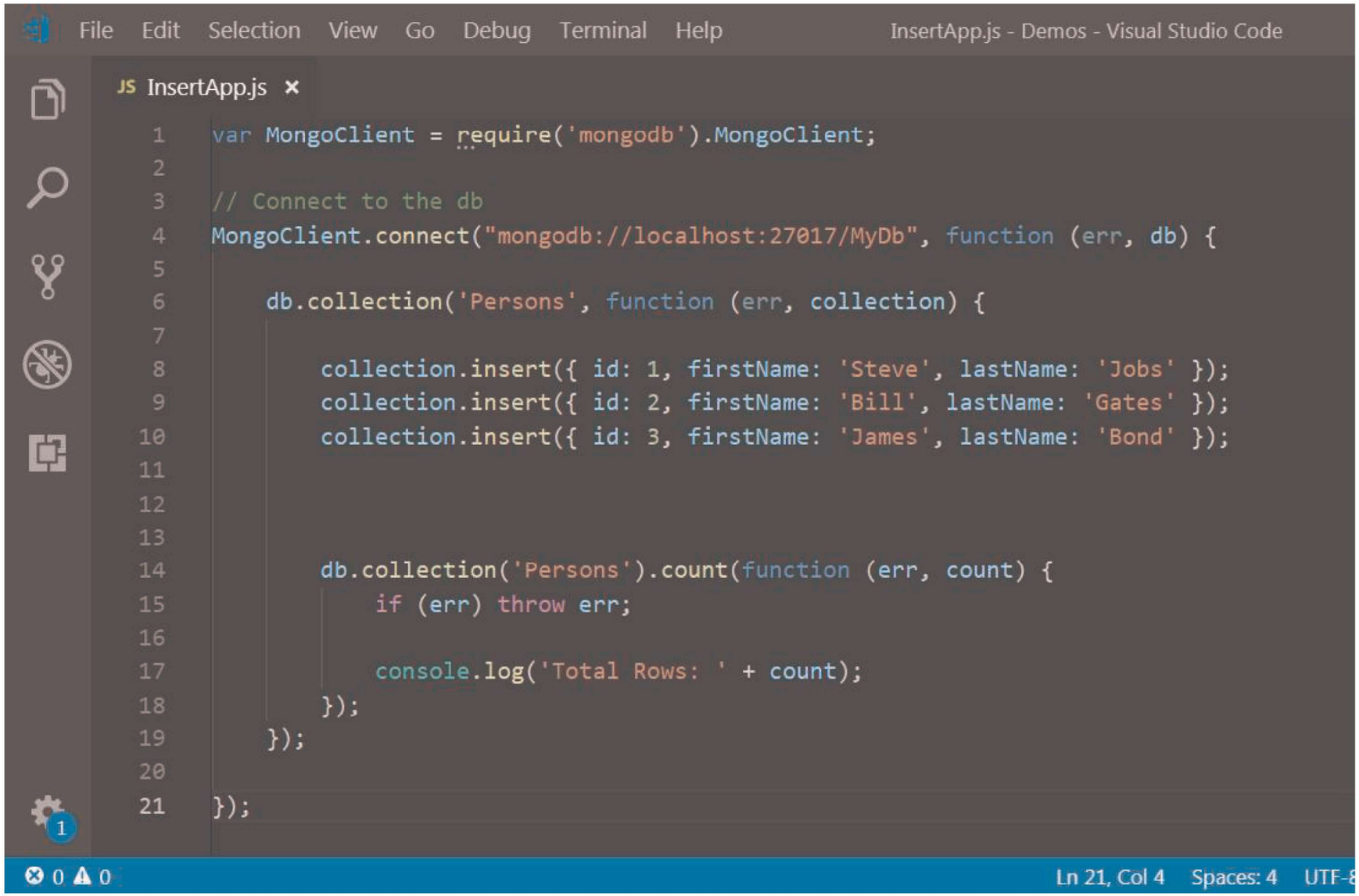
Connecting MongoDB



Details

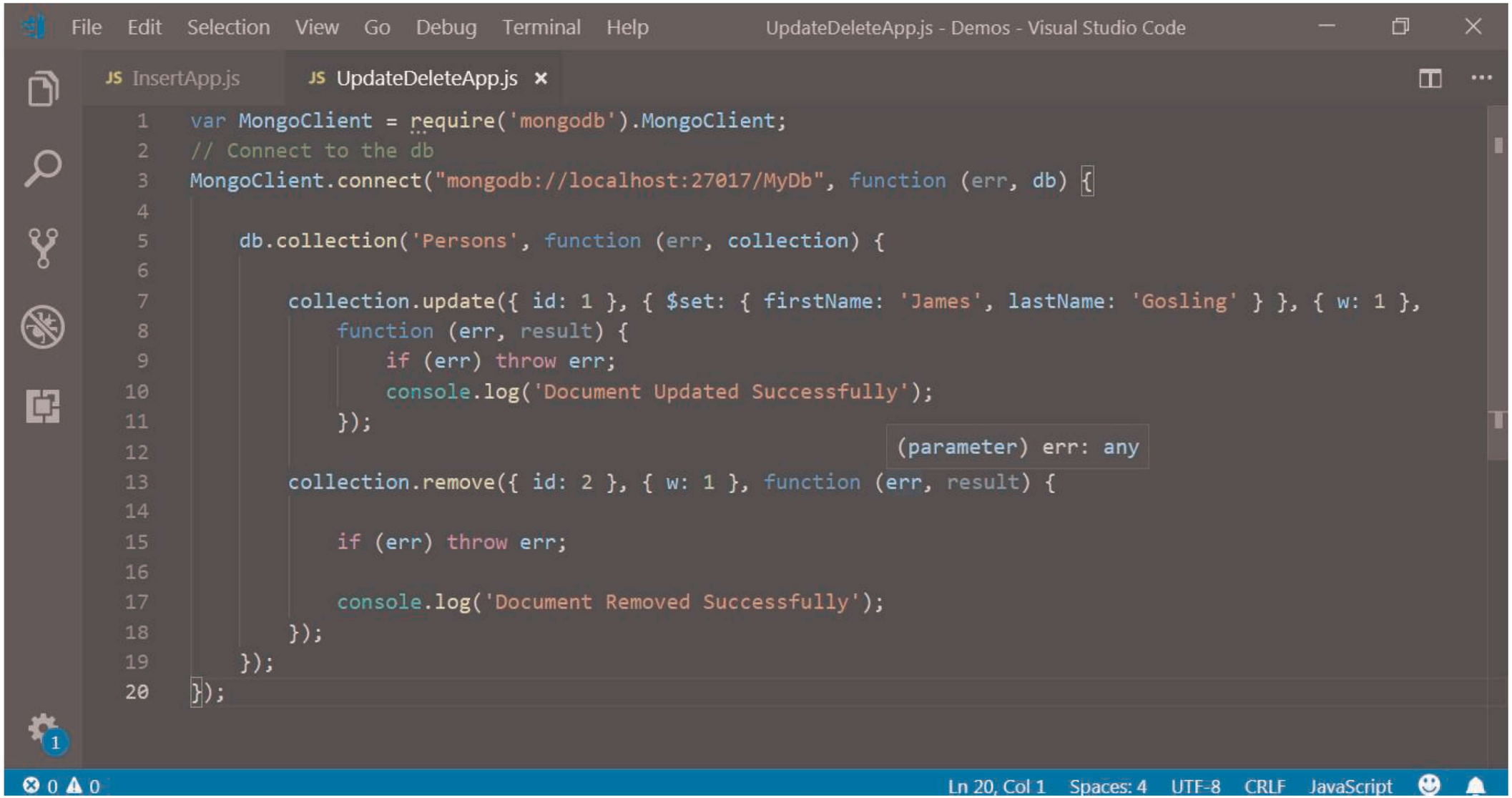
- we have imported mongodb module (native drivers) and got the reference of MongoClient object.
- Then we used MongoClient.connect() method to get the reference of specified MongoDB database.
- The specified URL "mongodb://localhost:27017/MyDb" points to your local MongoDB database created in MyMongoDB folder.
- The connect() method returns the database reference if the specified database is already exists, otherwise it creates a new database.
- Now you can write insert/update or query the MongoDB database in the callback function of the connect() method using db parameter.

Insert Documents

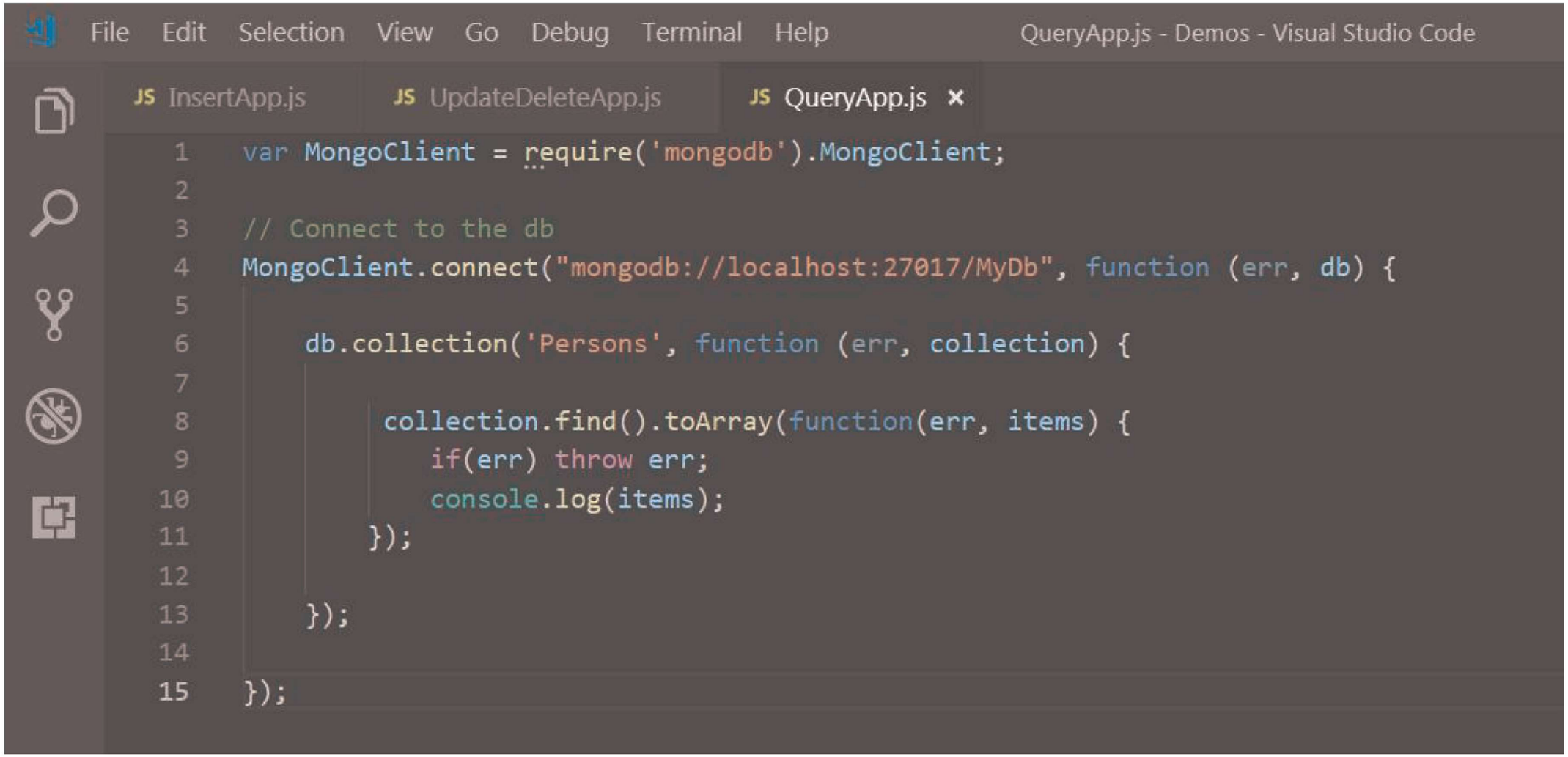


MENTORLABS℠

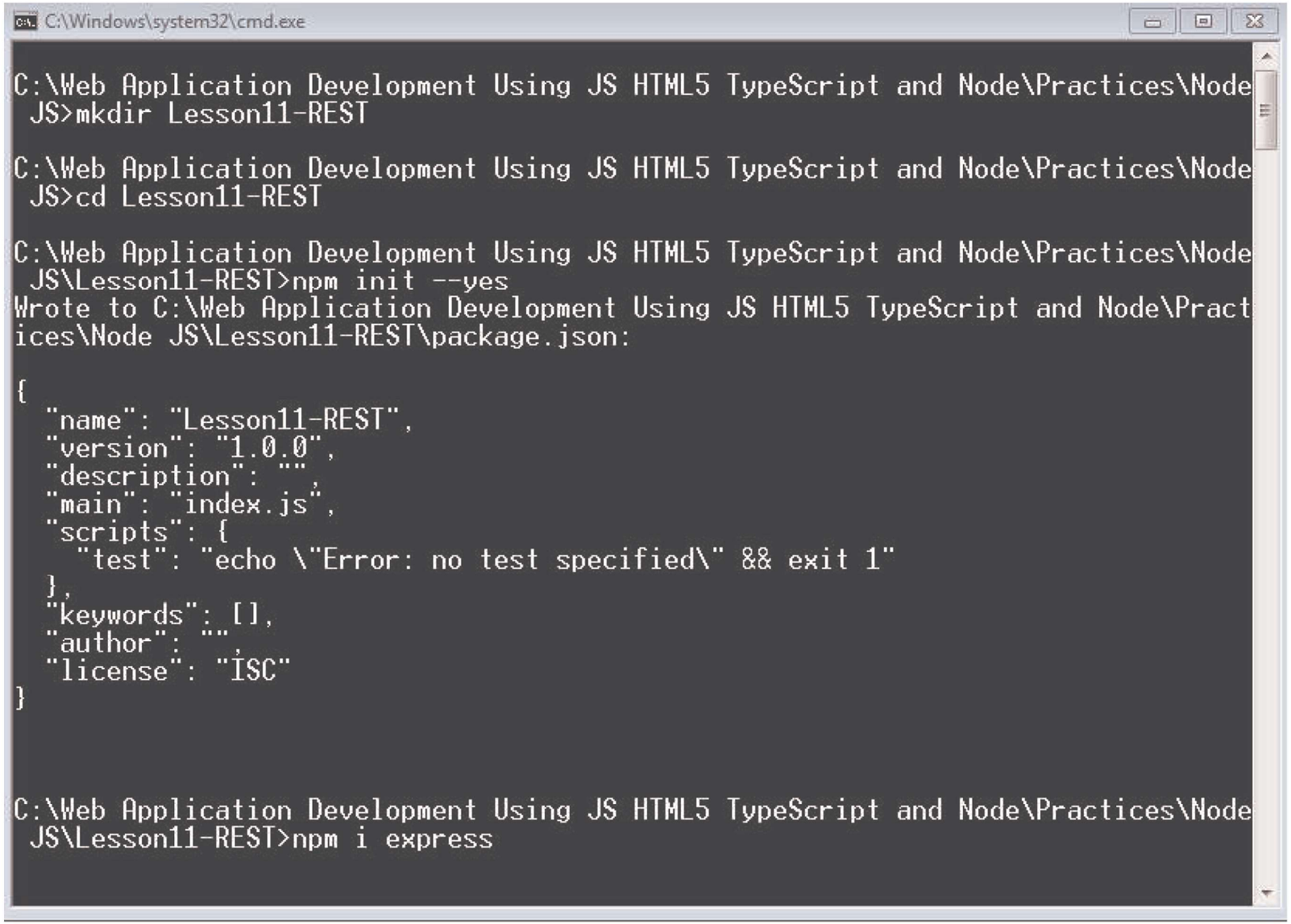
Update/Delete Documents



Query Database

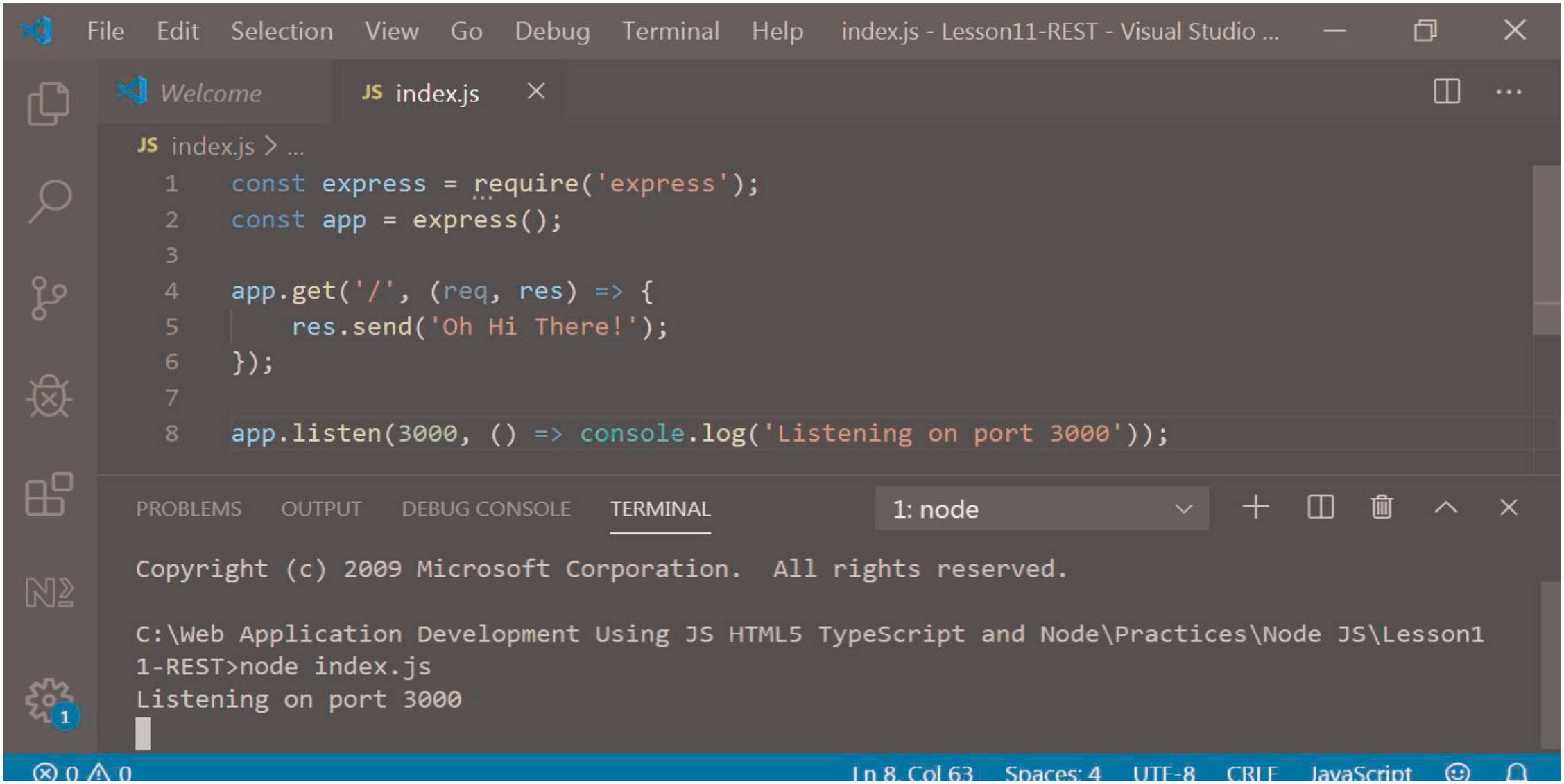


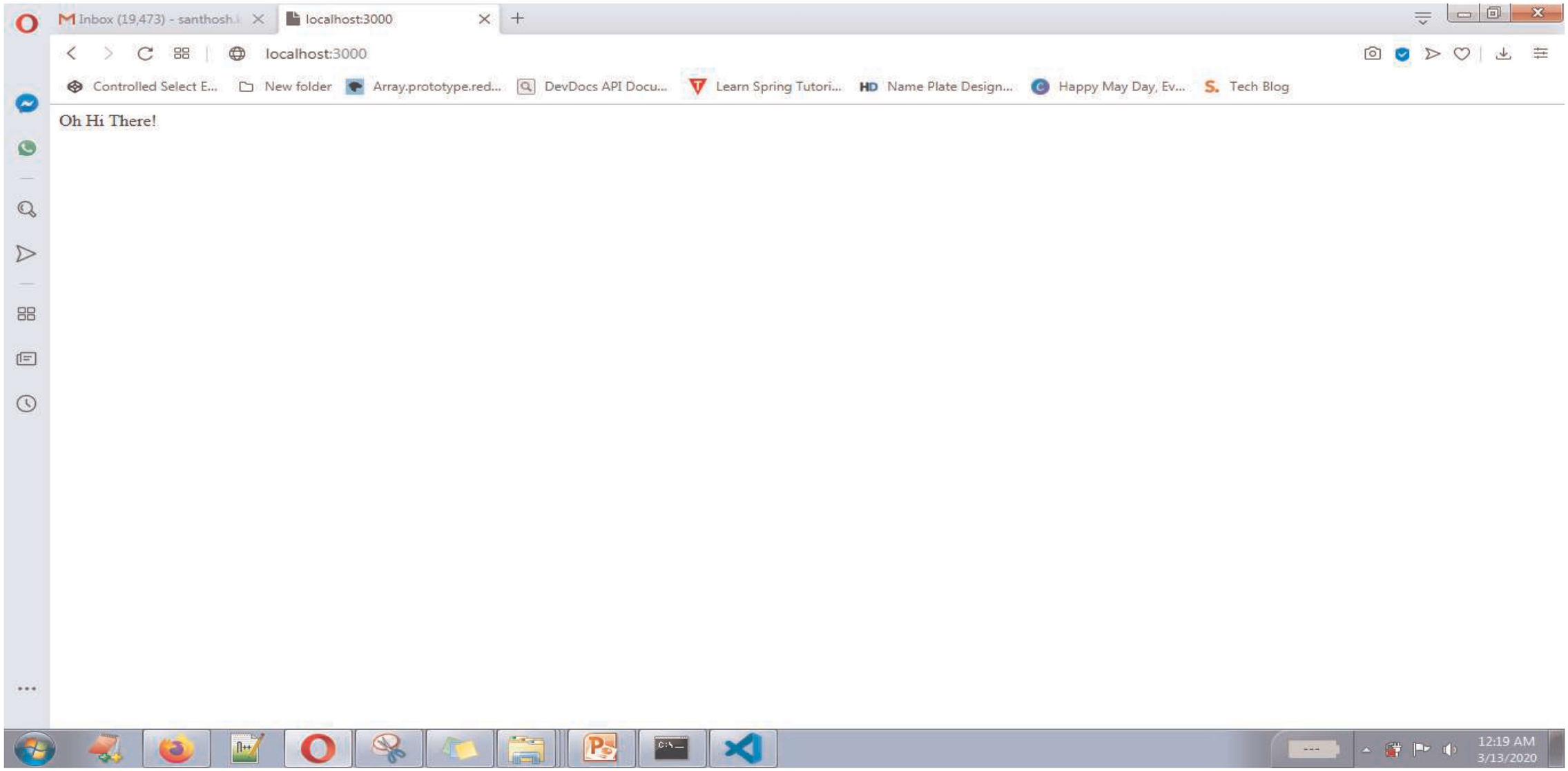
Create a Node with REST Api



- We also need an index.js file in the root of the project, so we can create that too.
- Once that is complete, add the following code.
- We are requiring the express module, and then calling the express() function and assigning the result to the app constant.
- The result is an Object, and by convention it is typically named app.
- Visual Studio Code informs us of the following about express(). “Creates an Express application. The express() function is a top-level function exported by the express module.”


File Edit Selection View Go Debug Terminal Help index.js - Lesson11-REST - Visual Studio ...
Welcome
index.js
index.js > ...
1 const express = require('express');
2 const app = express();
3
4 app.get('/', (req, res) => {
5 res.send('Oh Hi There!');
6 });
7
8 app.listen(3000, () => console.log('Listening on port 3000'));

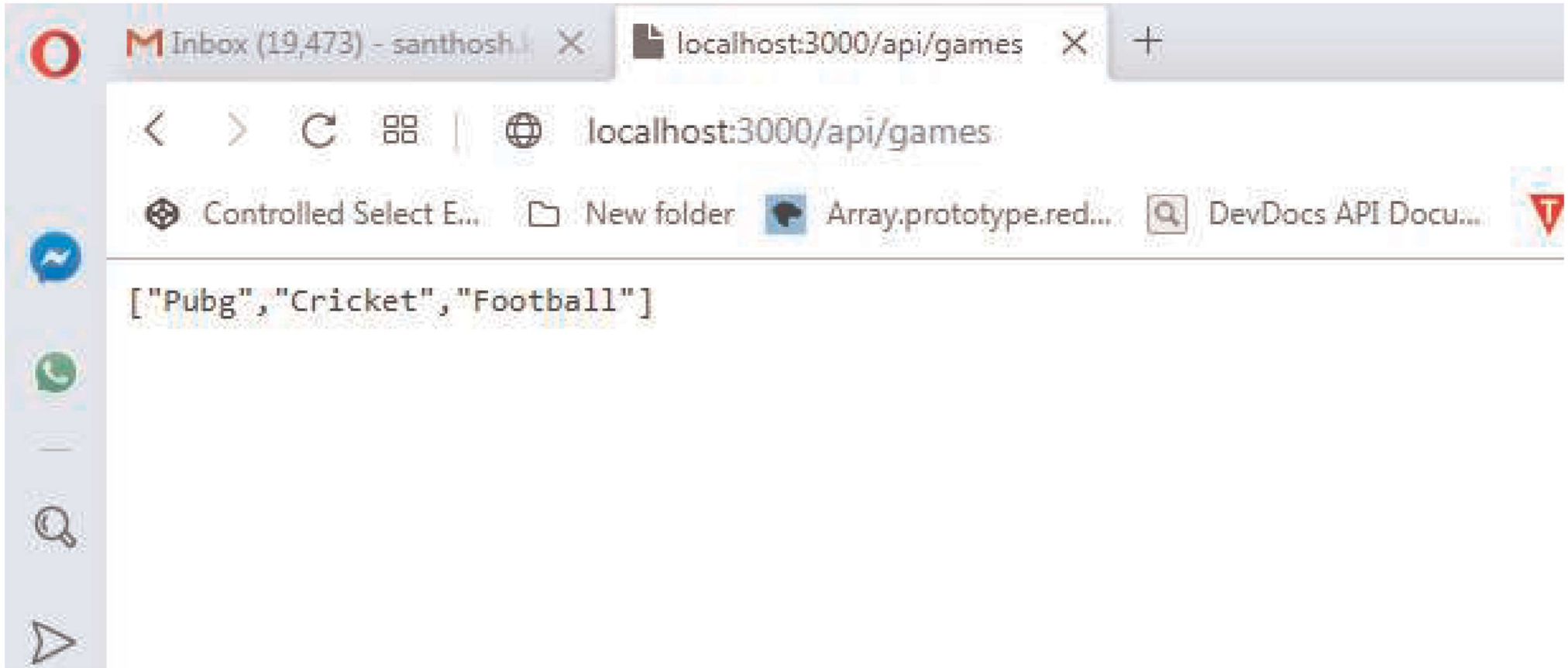

File Edit Selection View Go Debug Terminal Help index.js - Lesson11-REST - Visual Studio ...
Welcome
index.js
index.js > ...
1 const express = require('express');
2 const app = express();
3
4 app.get('/', (req, res) => {
5 res.send('Oh Hi There!');
6 });
7
8 app.listen(3000, () => console.log('Listening on port 3000'));
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: node
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Web Application Development Using JS HTML5 TypeScript and Node\Practices\Node JS\Lesson11-REST>node index.js
Listening on port 3000
0 0
Ln 8, Col 63 Spaces: 4 UTF-8 CRLF JavaScript


Inbox (19,473) - santhosh
localhost:3000
localhost:3000
Controlled Select E...
New folder
Array.prototype.red...
DevDocs API Docu...
Learn Spring Tutori...
Name Plate Design...
Happy May Day, Ev...
Tech Blog
Oh Hi There!
12:19 AM
3/13/2020

MENTORLABS℠

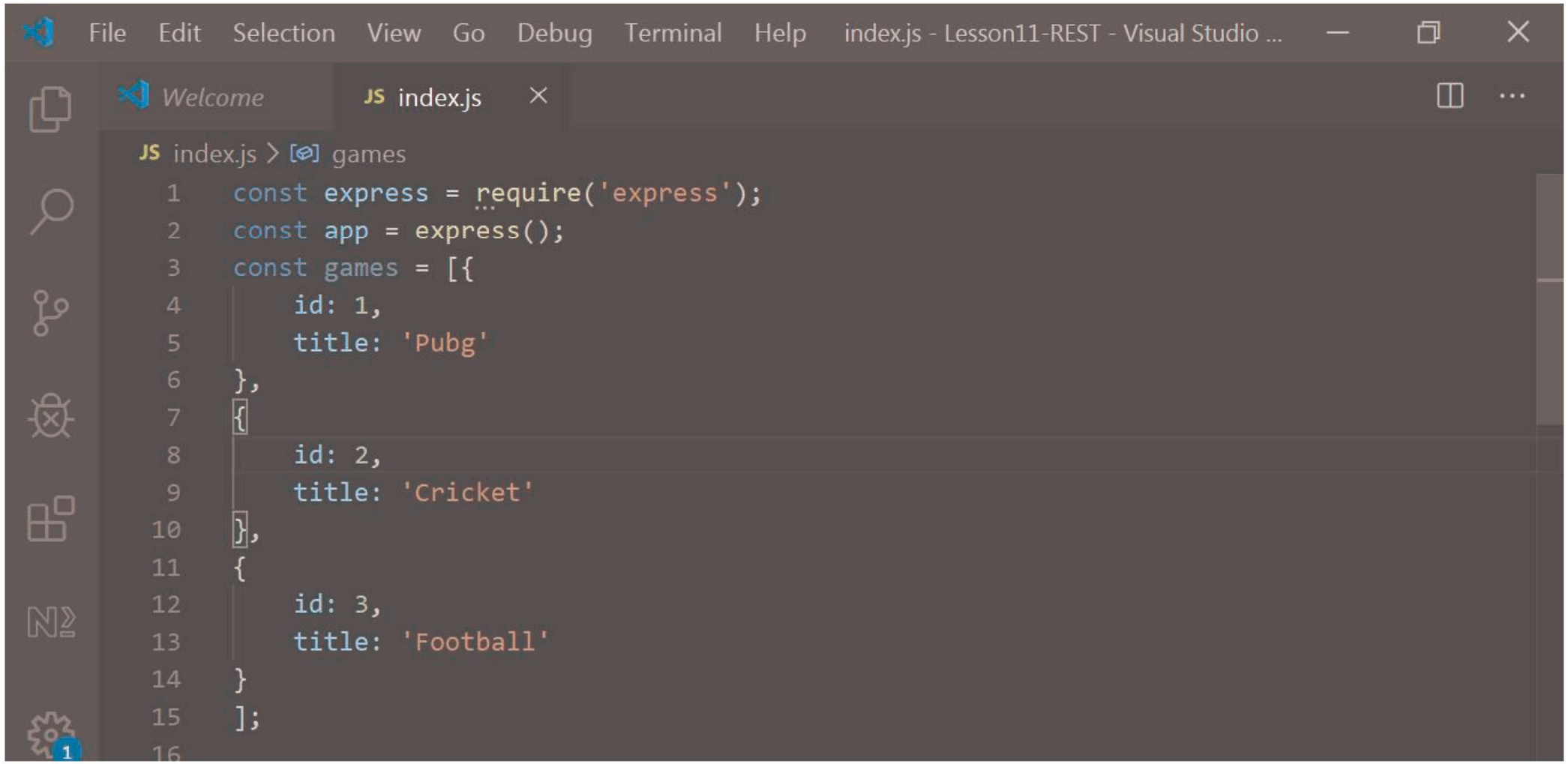
- Now we will add a new route that will simulate an API.
- We want to be able to visit the /api/games endpoint and see all the games in the system.


File Edit Selection View Go Debug Terminal Help index.js - Lesson11-REST - Visual Studio ...
Welcome
index.js
index.js > ...
1 const express = require('express');
2 const app = express();
3
4 app.get('/', (req, res) => {
5 res.send('Oh Hi There!');
6 });
7
8 app.get('/api/games', (req, res) => {
9 res.send(['Pubg', 'Cricket', 'Football']);
10 });
11
12 app.listen(3000, () => console.log('Listening on port 3000'));


Inbox (19,473) - santhosh...
localhost:3000/api/games
localhost:3000/api/games
Controlled Select E...
New folder
Array.prototype.red...
DevDocs API Docu...
["Pubg","Cricket","Football"]

HTTP GET Requests

- Now we can set up some get requests for fetching games from the server.
- This corresponds to the *Read* of crud in a rest api.
- We are simply using an array of games, as again there is no database just yet.


File Edit Selection View Go Debug Terminal Help index.js - Lesson11-REST - Visual Studio ...
Welcome
index.js
index.js > games
1 const express = require('express');
2 const app = express();
3 const games = [{
4 id: 1,
5 title: 'Pubg'
6 },
7 {
8 id: 2,
9 title: 'Cricket'
10 },
11 {
12 id: 3,
13 title: 'Football'
14 }
15];

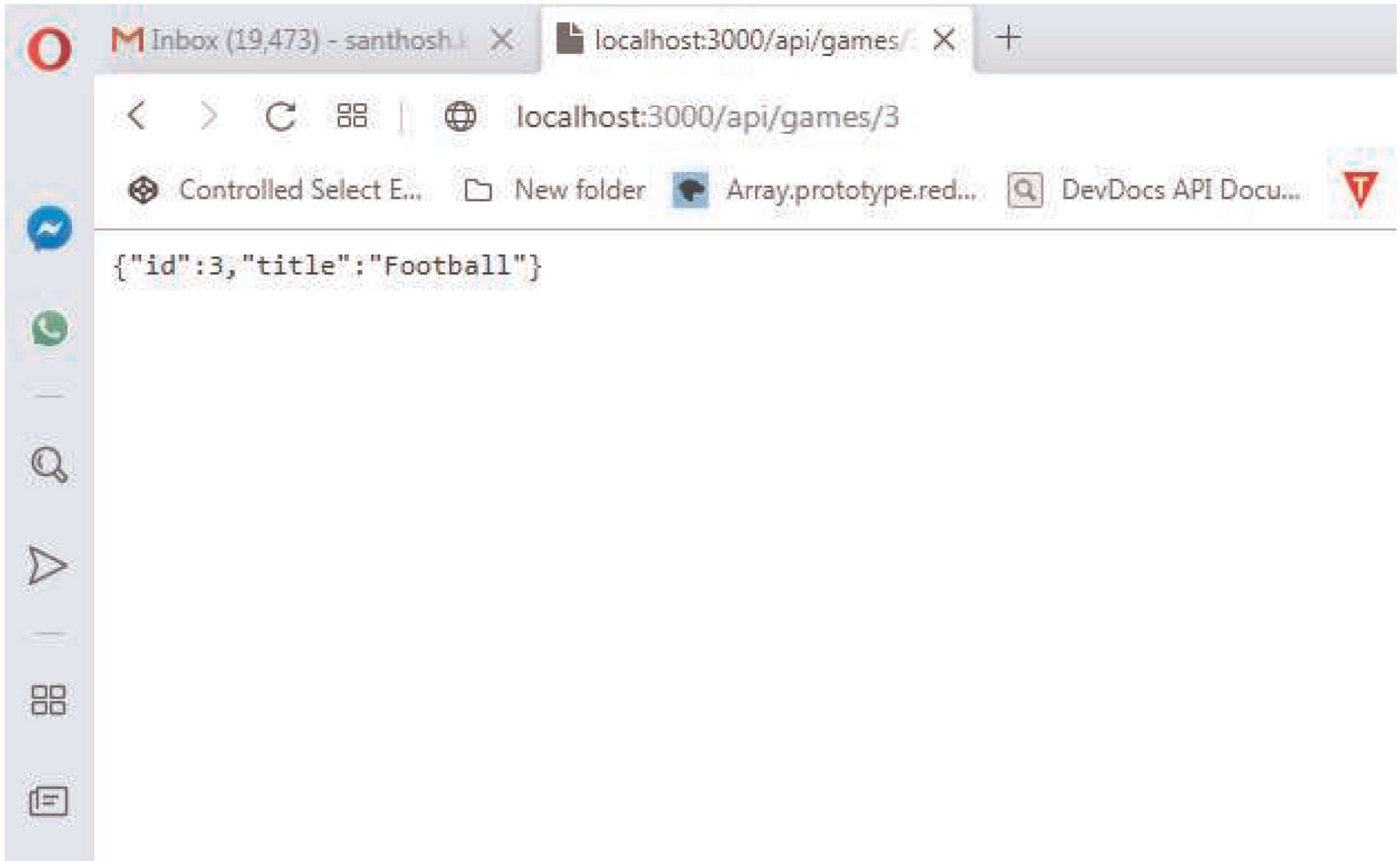

File Edit Selection View Go Debug Terminal Help index.js - Lesson11-REST - Visual Studio ...
Welcome
index.js
index.js > ...
16
17 app.get('/', (req, res) => {
18 res.send('Oh Hi There!');
19 });
20
21 app.get('/api/games', (req, res) => {
22 res.send(games);
23 });
24
25 app.listen(3000, () => console.log('Listening on port 3000'));


Inbox (19,473) - santhosh
localhost:3000/api/games
localhost:3000/api/games
Controlled Select E...
New folder
Array.prototype.red...
DevDocs API Docu...
Learn Spring Tutori...
[{"id":1,"title":"Pubg"},{"id":2,"title":"Cricket"},{"id":3,"title":"Football"}]

Finding a Game

- Now we want to be able to find a specific game only using a route parameter. For this, we can use the find function.

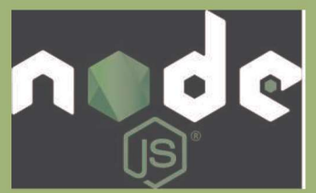
```
app.get('/api/games/:id', (req, res) => {
    const game = games.find(g => g.id === parseInt(req.params.id));
    if (!game) return res.status(404).send('The game with the given ID was not
    found.');
    res.send(game);
});

app.listen(3000, () => console.log('Listening on port 3000'));
```


Inbox (19,473) - santhosh...
localhost:3000/api/games/
localhost:3000/api/games/3
Controlled Select E...
New folder
Array.prototype.red...
DevDocs API Docu...
{"id":3,"title":"Football"}

HTTP POST Requests

- Now we need to set up the code that will allow our web server to respond to http post requests.
- This corresponds to the *Create* of crud in a rest api. We can use a post request to add a new game to the system.

```
// add a game
app.post('/api/games', (req, res) => {
    const game = {
        id: games.length + 1,
        title: req.body.title
    }
    games.push(game);
    res.send(game);
});

app.listen(3000, () => console.log('Listening on port 3000'));
```



END