Node JS

Date: 30-Mar-2020

Agenda

- Introduction to Node.js
- File System Module and Express.js
- Asynchronous Programming
- Integration with MongoDB and Email Servers
- REST APIs and GraphQL
- Building Node.js Applications using ES6
- User Authentication and Application Security
- Dynamic Client-Server Interaction using Socket.IO
- Testing Node.js Applications
- Microservices Application

1

Introduction to Node JS

What is Node.js?

- Node.js is a powerful framework developed on **Chrome’s V8 JavaScript engine** that compiles the JavaScript directly into the native machine code.
- It is a lightweight framework used for creating server-side web applications and extends JavaScript API to offer usual server-side functionalities.
- It is generally used for large-scale application development, especially for video streaming sites, single page application, and other web applications.
- Node.js *makes* use of an event-driven, non-blocking I/O model which makes it a right pick for the data-intensive real-time applications.
- Like any other programming languages, node.js makes use of packages and modules. These are the libraries that contain various functions and are imported from npm (node package manager) into our code and utilized in the programs.

Features of Node.js

- **Open Source**
  Node.js is an open source framework MIT license that is supported by a huge community. Its community is pretty much active have contributed to add new capabilities to Node.js applications.

- **Simple and Fast**
  Since Node.js is built on Google Chrome's V8 JavaScript Engine, its libraries are capable of fast code execution.

- **Asynchronous**
  All the libraries of Node.js are asynchronous which means the Node.js based servers never wait for an API to send back the response and move on to the next API.

Features of Node.js

- **High Scalability**
  Because of the event mechanism, Node.js is highly scalable and aids the server in a non-blocking response.

- **Single-Threaded**
  With the help of event looping, Node.js is able to follow the single-threaded model. This lets a single program to handle multiple requests.

- **No Buffering**
  One of the major functionalities of Node.js applications is that it never buffers any data.

- **Cross-Platform**
  Node.js can be easily built and deployed on various platforms like Windows, MAC, and Linux.

NPM (Node Package Manager)

- NPM stands for Node Package Manager which as the name suggests is a package manager for Node.js packages/modules. From Node version 0.6.0. onwards,
- npm has been added as default in the node installation.
- It saves you from the hassle of installing npm explicitly.
- **NPM basically helps in two ways:**
  - Provides and hosts Online repositories for node.js packages/modules which can be easily downloaded and used in our projects. You can find them here: npmjs.com
  - Provides the Command line utility in order to install various Node.js packages, manage Node.js versions and dependencies of the packages.

Node.js Modules

- The modules in Node.js represents various functionalities that are bundled up into single or multiple JS files.

- These modules have a unique context, thus, they never interfere nor pollute the scope of other modules.

- These modules enable the code reusability and enhance the ease of usage. Node.js basically provides three types of modules:
  - Core Modules
  - Local Modules
  - Third-Party Modules

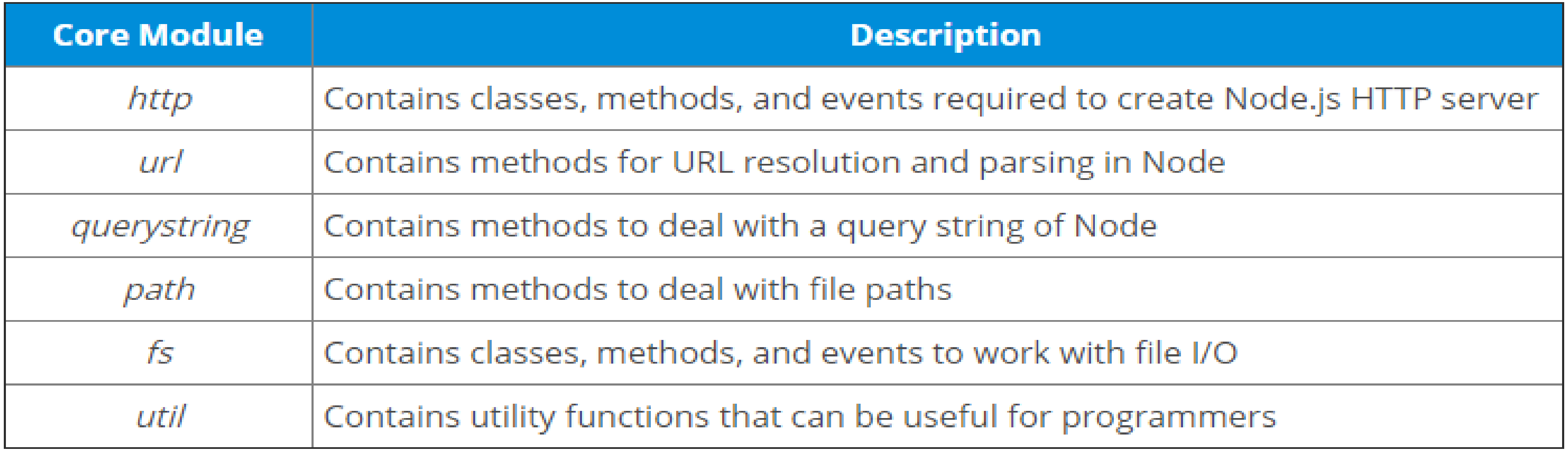
You can load module, using the below code:

```
var module = require('module_name');
```

Core Module

Since Node.js is a very ***lightweight*** framework, the core modules bundle the absolute minimum functionalities. These modules generally get loaded when the Node process starts its execution. All you need to do is, import these core modules in order to use them in your code.

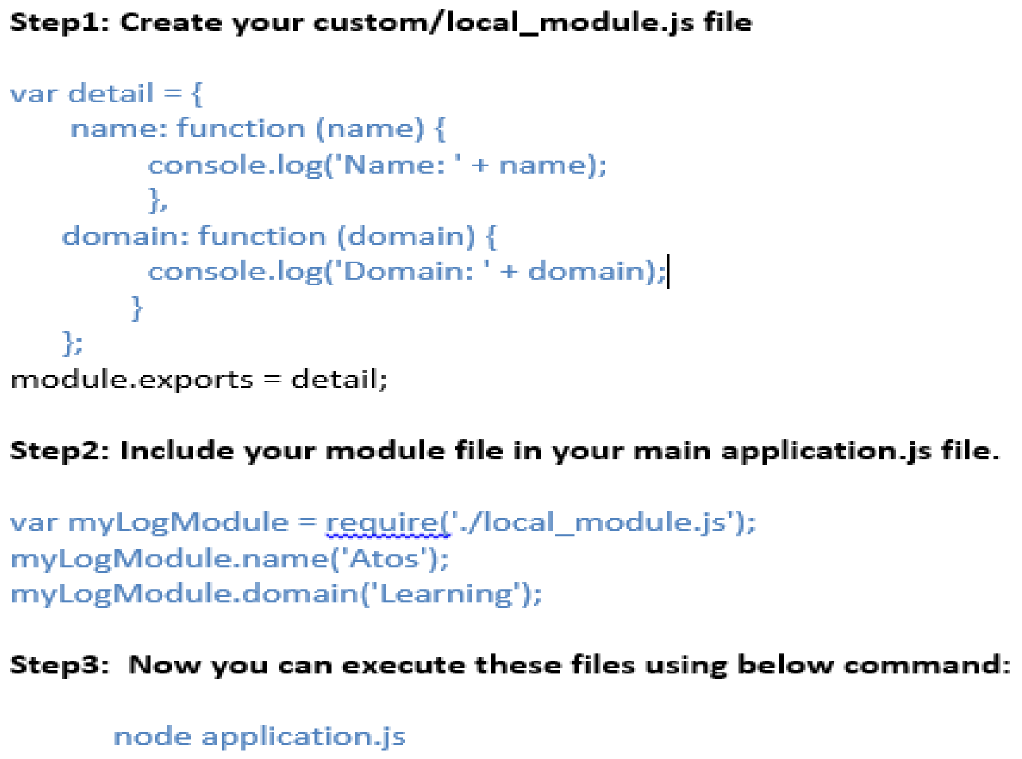
Below are few important core modules



Core Module	Description
http	Contains classes, methods, and events required to create Node.js HTTP server
url	Contains methods for URL resolution and parsing in Node
querystring	Contains methods to deal with a query string of Node
path	Contains methods to deal with file paths
fs	Contains classes, methods, and events to work with file I/O
util	Contains utility functions that can be useful for programmers

Local Module

- The local modules of Node.js are custom modules that are created locally by user/developer in the application.
- These modules can include various functionalities bundled into distinct files and folders which can be easily distributed in the Node.js community using NPM.



Step1: Create your custom/local_module.js file

```
var detail = {
      name: function (name) {
            console.log('Name: ' + name);
            },
      domain: function (domain) {
            console.log('Domain: ' + domain);
            }
      };
module.exports = detail;
```

Step2: Include your module file in your main application.js file.

```
var myLogModule = require('./local_module.js');
myLogModule.name('Atos');
myLogModule.domain('Learning');
```

Step3: Now you can execute these files using below command:

```
      node application.js
```

Third party or External Module

- You can use the external or 3rd party modules only by downloading them via NPM. These modules are generally developed by other developers and are free to use. Few of the best external modules are express, react, gulp, mongoose, mocha etc.

  - *Globally Loading the 3rd party modules:*
    - npm install -g <module_name>

  - *Include your module file in your main application file:*
    - npm install --save <module_name>

JSON File

- The **package.json file** in Node.js is the heart of the entire application. It is basically the manifest file that contains the metadata of the project. Thus, understanding and working with this file becomes very important for a successful Node project development.

- The package.json file generally consists of the metadata of the application, which is further categorized into below two categories:

  - **Identifying metadata properties:** This consists of properties like the project name, current module version, license, author of the project, project description etc.

  - **Writing directly to file:** You can directly write the necessary information into the package.json file and include it, in your project.

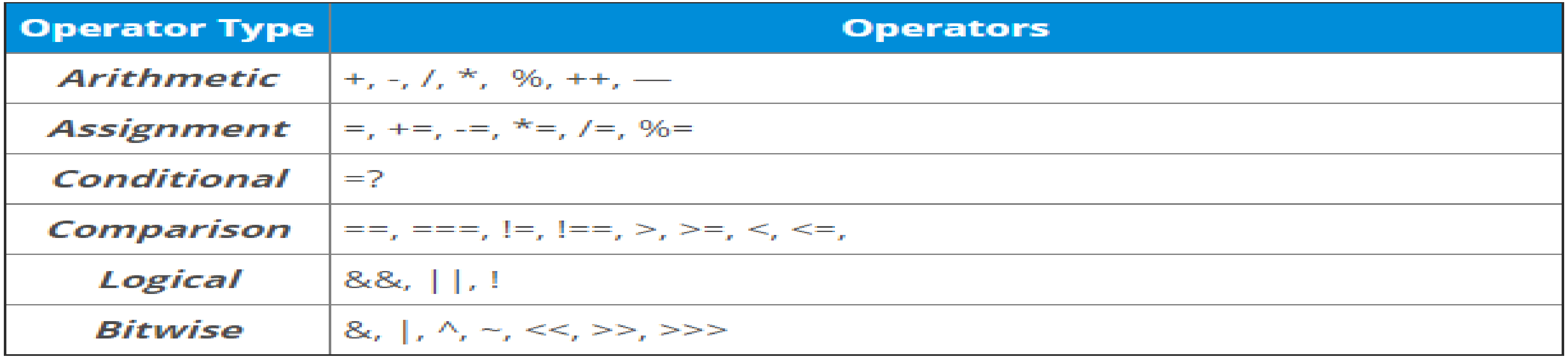
Node JS Basics

- Since Node.js is a JavaScript framework, it uses the JavaScript syntax.

- **Data Types**
  - Like any other programming language, Node.js has various datatypes, which are further categorized into Primitive and Non-Primitive datatypes.
  - ***Primitive Data Types are:***
    - String
    - Number
    - Boolean
    - Null
    - Undefined
  - ***Non-Primitive Data Types are:***
    - Object
    - Date
    - Array

Node JS Basics

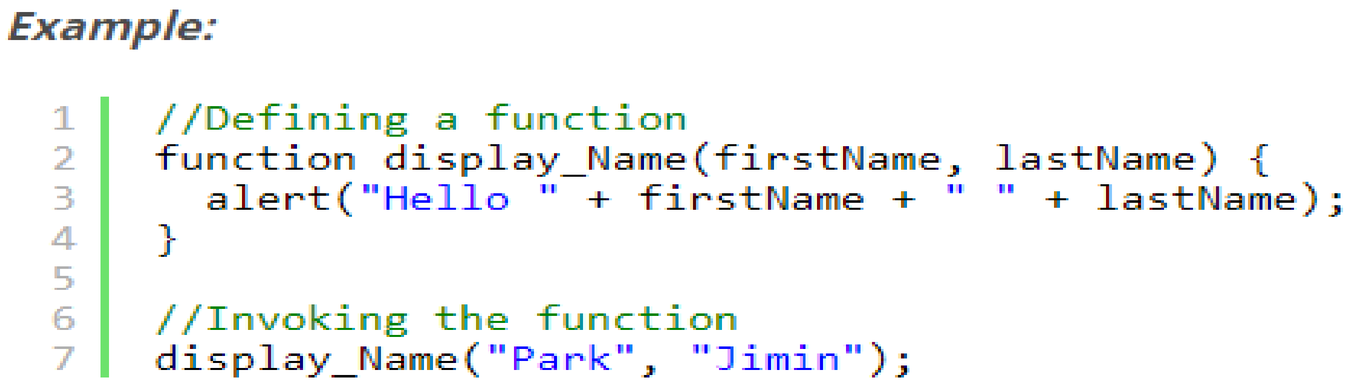
- **Variables**
  - Variable are entities that hold values which may vary during the course of a program. To create a variable in Node.js, you need to make use of a reserved keyword var. You do not have to assign a data type, as the compiler will automatically pick it.
- ***Syntax:***
  - var varName = value;
- ***Operators:*** *Node JS supports below operators*
  - var varName = value;



Operator Type	Operators
Arithmetic	+, -, /, *, %, ++, —
Assignment	=, +=, -=, *=, /=, %=
Conditional	=?
Comparison	==, ===, !=, !==, >, >=, <, <=,
Logical	&&, | |, !
Bitwise	&, |, ^, ~, <<, >>, >>>

Node JS Basics

- **Functions**
  - Functions in Node.js is a block of code that has a name and is written to achieve a particular task. You need to use the keyword function to create it. A function is generally a two-step process. First is defining the function and the second is invoking it. Below is the syntax of creating and invoking a function:



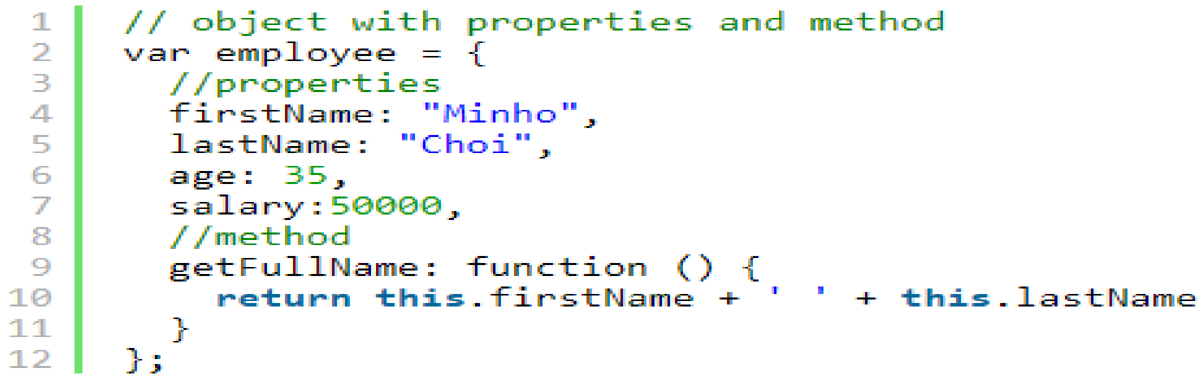
Example:

```
//Defining a function
function display_Name(firstName, lastName) {
  alert("Hello " + firstName + " " + lastName);
}

//Invoking the function
display_Name("Park", "Jimin");
```

Node JS Basics

- **Objects**
  - An object is a non-primitive data type that can hold multiple values in terms of properties and methods. Node.js objects are standalone entities as there is no concept of class. You can create an object in two ways:
    1. Using Object literal
    2. Using Object constructor

Example:



```
// object with properties and method
var employee = {
  //properties
  firstName: "Minho",
  lastName: "Choi",
  age: 35,
  salary:50000,
  //method
  getFullName: function () {
    return this.firstName + ' ' + this.lastName
  }
};
```

2

File System Module and Express.js

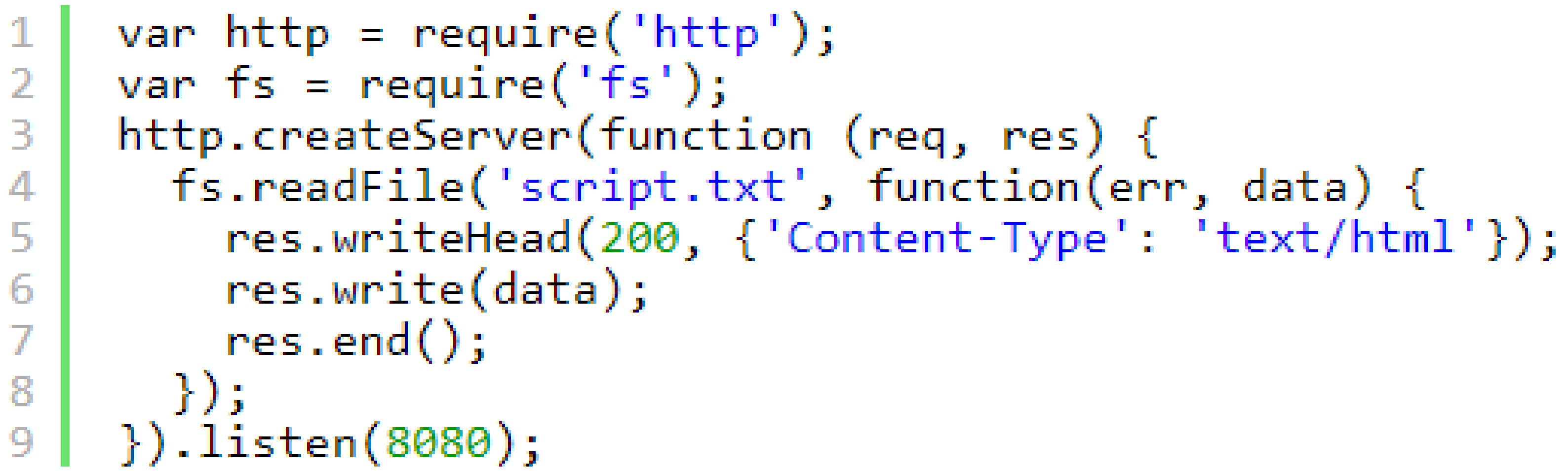
File System

- To access the physical file system, Node.js makes use of the **fs** module which basically takes care of all asynchronous and synchronous file I/O operations. This module is imported using the below command:
  - var fs = require('fs');

- Some of the general use for the File System modules are:
  1. Read files
  2. Create files
  3. Update files
  4. Rename files

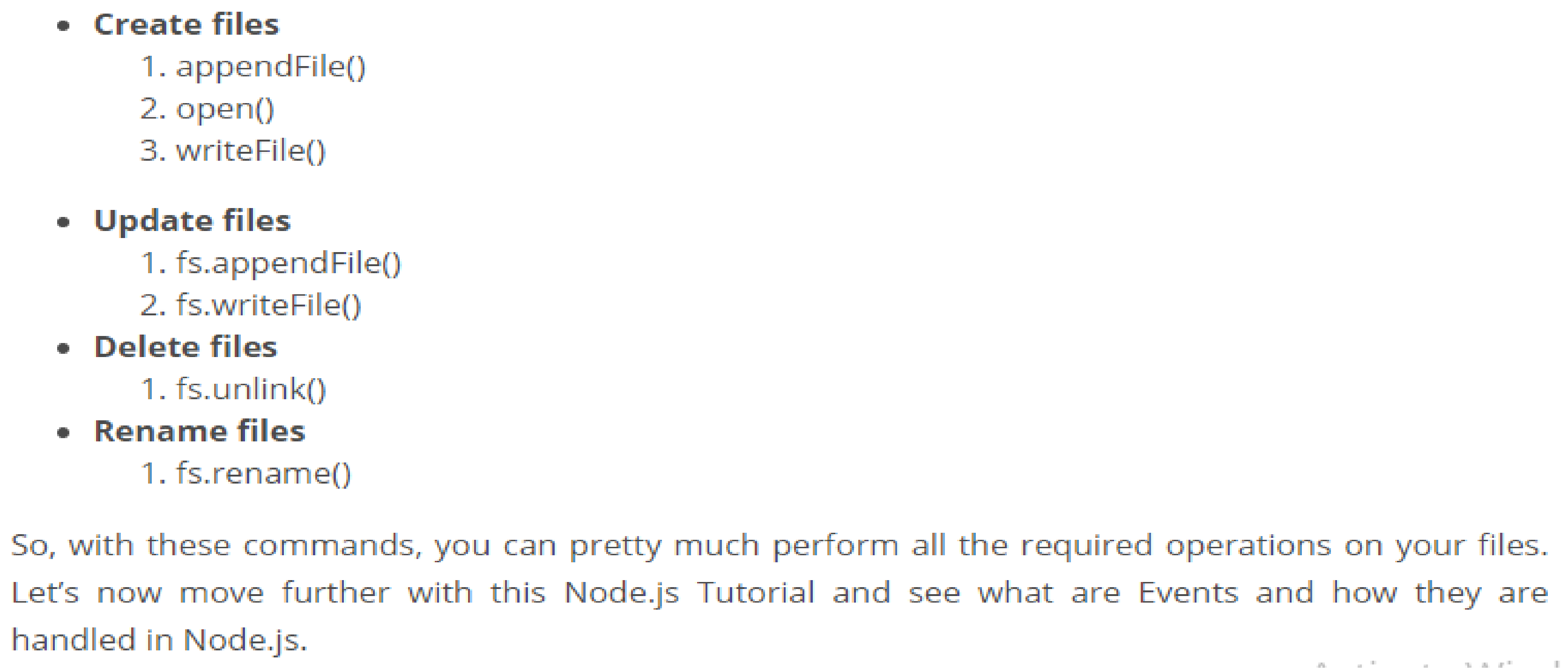
File System

- Read files
    1. fs.readFile()



```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('script.txt', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(8080);
```

File System



- **Create files**
  1. appendFile()
  2. open()
  3. writeFile()

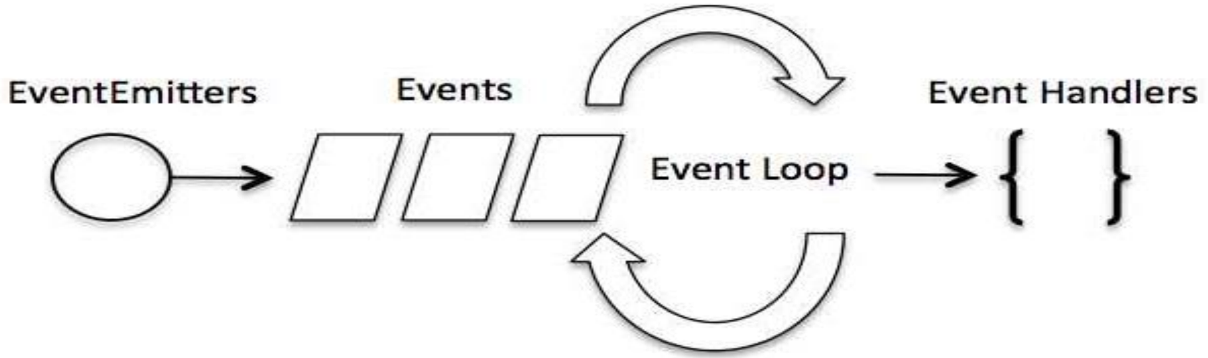
- **Update files**
  1. fs.appendFile()
  2. fs.writeFile()
- **Delete files**
  1. fs.unlink()
- **Rename files**
  1. fs.rename()

So, with these commands, you can pretty much perform all the required operations on your files. Let's now move further with this Node.js Tutorial and see what are Events and how they are handled in Node.js.

Events

- Node.js applications are single threaded and event-driven. Node.js supports concurrency as it is event-driven, and thus makes use of concepts like events and callbacks. The async function calls help Node.js in maintaining concurrency throughout the application.

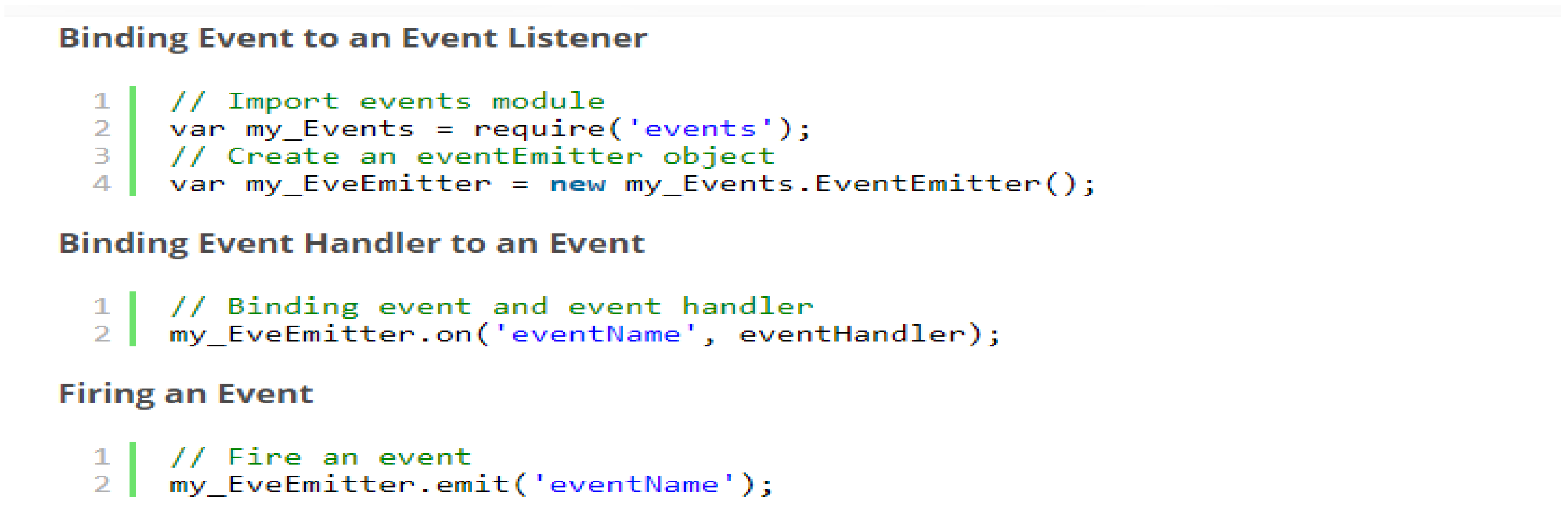
- Basically, in a Node.js application, there is a main loop which waits and listens for events, and once any event is completed, it immediately initiates a callback function.



Events

- Even though events look similar to callback functions but the difference lies in their functionalities.
- When an asynchronous function returns its results callbacks are invoked on the other hand event handling completely works on the observer pattern.
- And in Node.js, methods which listen to the events are called the observers.
- The moment, an event is triggered, its listener function automatically starts executing.
- Event modules and EventEmitter class provide multiple in-built events which are used to bind events with event listeners. Below I have written down the syntax for that.

Events



Binding Event to an Event Listener

```
// Import events module
var my_Events = require('events');
// Create an eventEmitter object
var my_EveEmitter = new my_Events.EventEmitter();
```

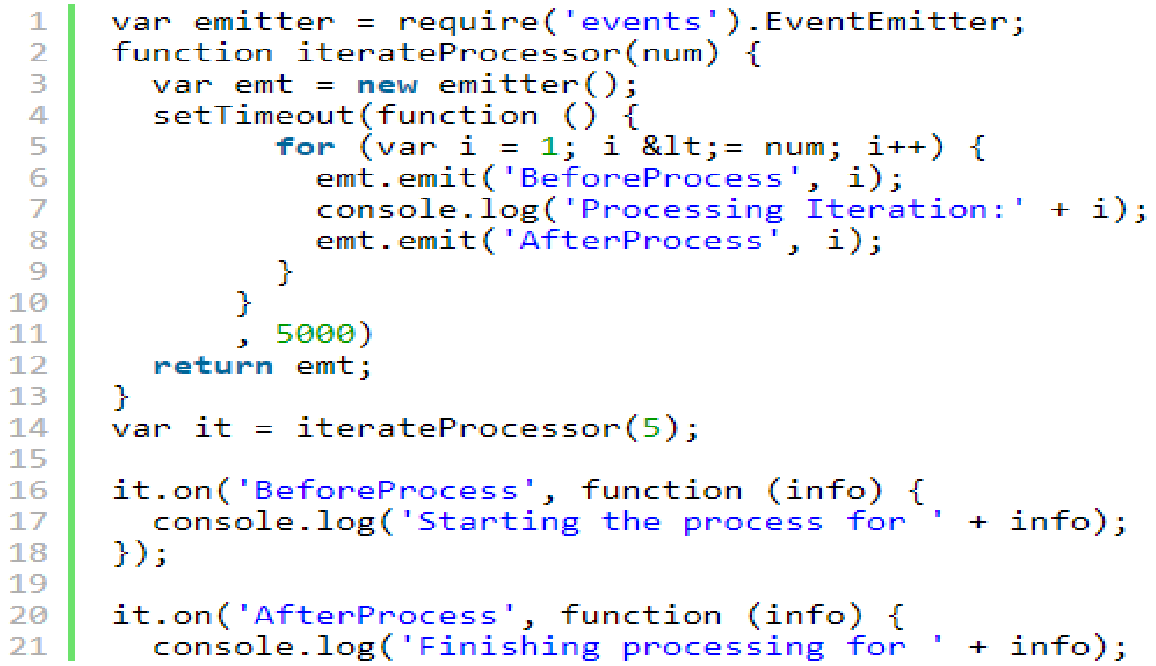
Binding Event Handler to an Event

```
// Binding event and event handler
my_EveEmitter.on('eventName', eventHandler);
```

Firing an Event

```
// Fire an event
my_EveEmitter.emit('eventName');
```

Events

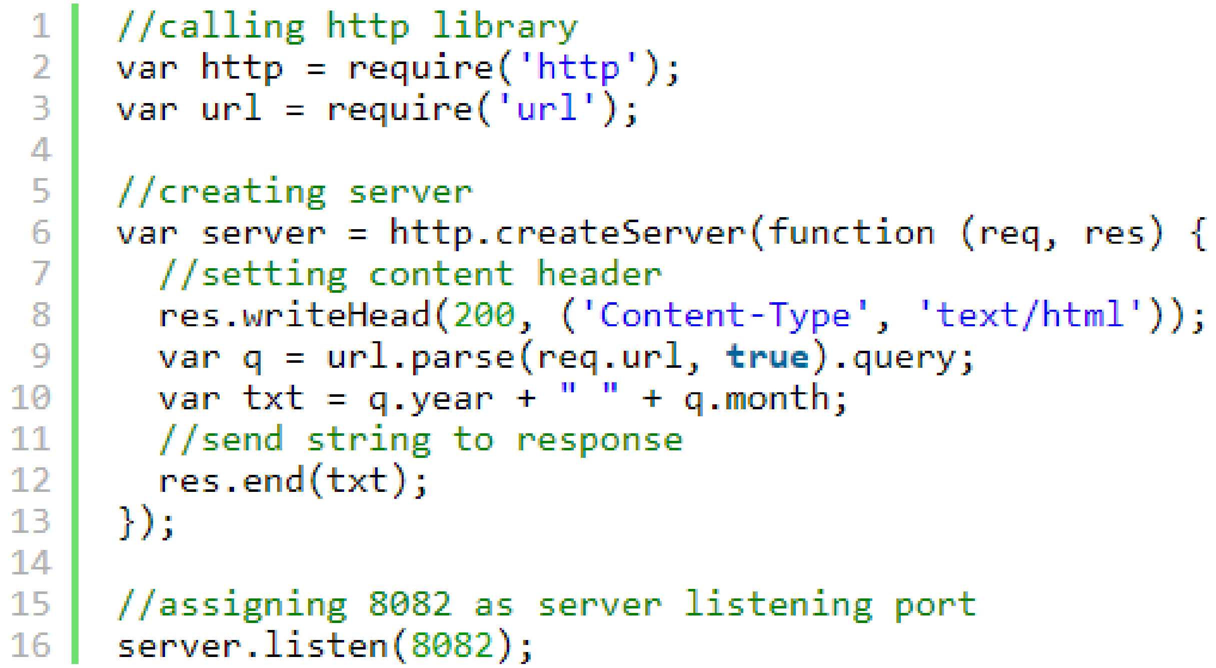


```
var emitter = require('events').EventEmitter;
function iterateProcessor(num) {
  var emt = new emitter();
  setTimeout(function () {
        for (var i = 1; i &lt;= num; i++) {
          emt.emit('BeforeProcess', i);
          console.log('Processing Iteration:' + i);
          emt.emit('AfterProcess', i);
        }
      }
      , 5000)
  return emt;
}
var it = iterateProcessor(5);

it.on('BeforeProcess', function (info) {
  console.log('Starting the process for ' + info);
});

it.on('AfterProcess', function (info) {
  console.log('Finishing processing for ' + info);
```

HTTP Module

- Generally, Node.js is used for developing server-based applications.
- But using the module, you can easily create web servers that can respond to the client requests.
- Thus it is also referred to Web Module and provides modules like HTTP and request that facilitate Node.js in processing the server requests.
- You can easily include this module in your Node.js application just by writing the below code:

HTTP Module



```javascript
//calling http library
var http = require('http');
var url = require('url');

//creating server
var server = http.createServer(function (req, res) {
  //setting content header
  res.writeHead(200, ('Content-Type', 'text/html'));
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  //send string to response
  res.end(txt);
});

//assigning 8082 as server listening port
server.listen(8082);
```

Express.js

- Express.js is a framework built on top of Node.js that facilitates the management of the flow of data between server and routes in the server-side applications. It is a lightweight and flexible framework that provides a wide range of features required for the web as well as mobile application development.

- Express.js is developed on the middleware module of Node.js called ***connect***. The connect module further makes use of **http** module to communicate with Node.js. Thus, if you are working with any of the connect based middleware modules, then you can easily integrate with Express.js.

Express.js

- Few of the major advantages of Express.js are:
  - Makes web application development faster
  - Helps in building mobile and web application of single-page, multi-page, and hybrid types
  - Express provides two templating engines namely, Jade and EJS
  - Express follows the Model-View-Controller (MVC) architecture
  - Makes integration with databases such as MongoDB, Redis, MySQL
  - Defines an error handling middleware
  - Simplifies configuration and customization easy for the application.

Express.js

- With all these features, Express takes responsibility of backend part in the MEAN stack.
- Mean Stack is the open-source JavaScript software stack that is used for building dynamic websites and web applications. Here, MEANstands for MongoDB, Express.js, AngularJS, and Node.js.
- Lets now see a simple example to understand, how Express.js works with Node.js to ease our work. But before you start working with Express.js, you need to install it in your system.
- To install Express.js globally you can use the below command:
  - npm install -g express
- Or, if you want to install it locally into your project folder, you need to execute the below command:
  - npm install express --save

Express.js Fundamentals

1. Routing and HTTP Methods

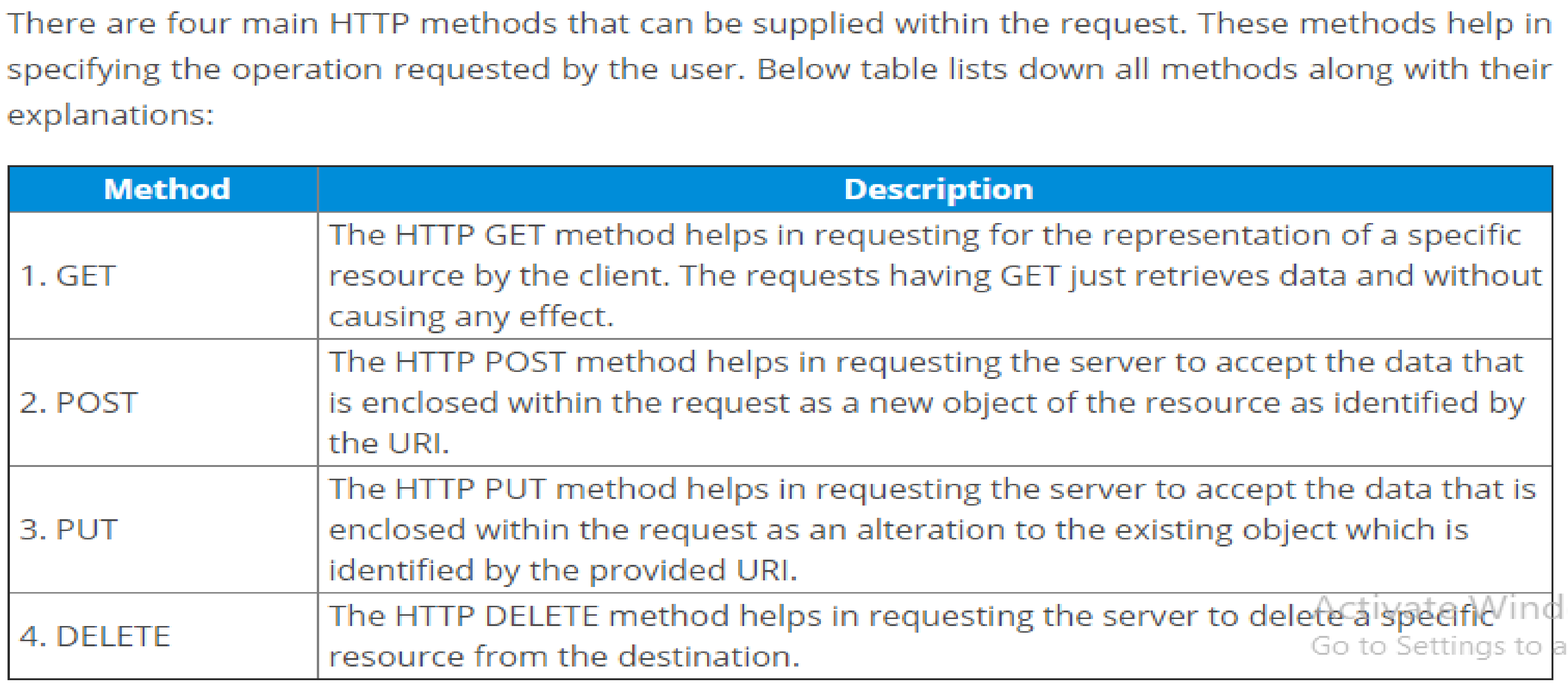
Routing refers to the process of determining a specific behavior of an application. It is used for defining how an application should respond to a client request to a particular route, path or URI along with a particular HTTP Request method. Each route can contain more than one handler functions, which is executed when the user browses for a specific route. Below is the structure of Routing in Express:

```
app.METHOD(PATH, HANDLER)
```

Where:

- app is just an instance of Express.js. You can use any variable of your choice.
- METHOD is an HTTP request method such as get, set, put, delete, etc.
- PATH is the route to the server for a specific webpage
- HANDLER is the callback function that is executed when the matching route is found.

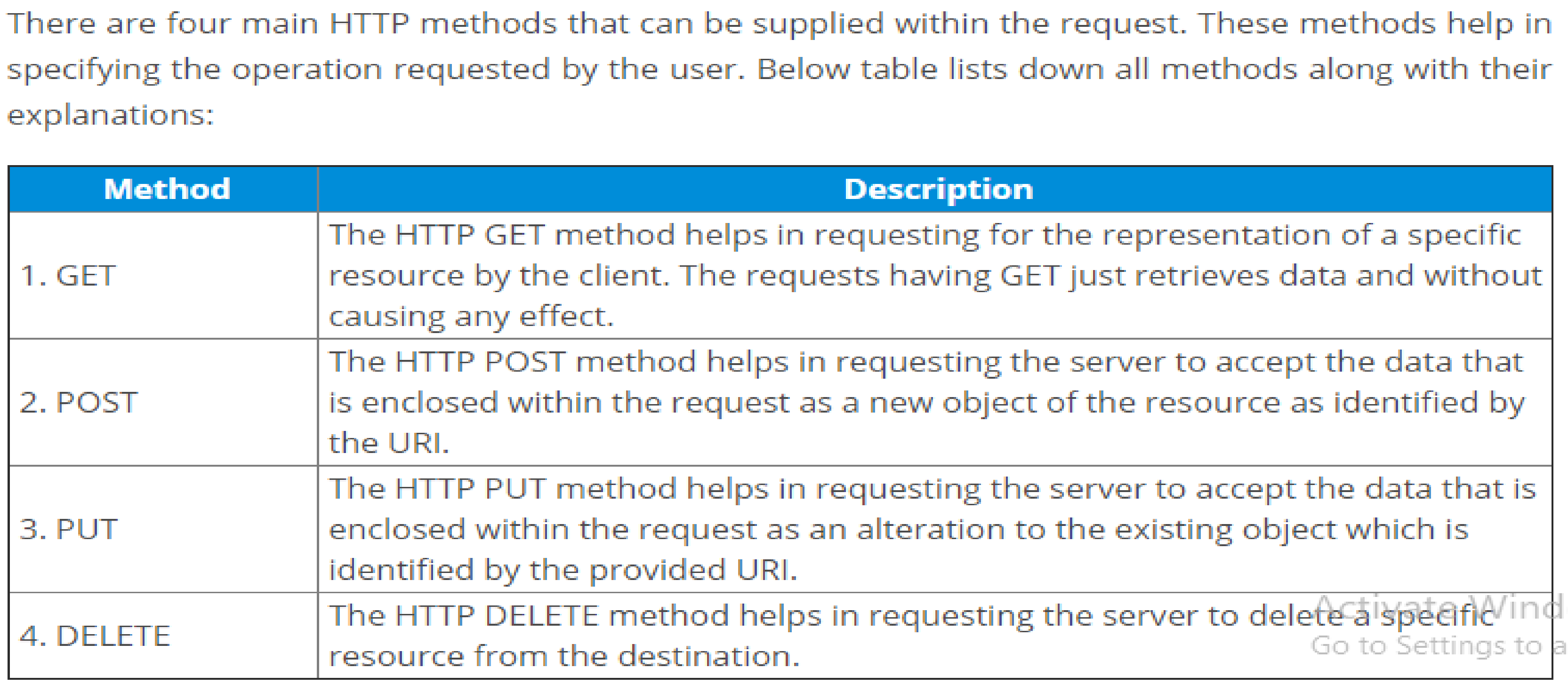
Express.js Fundamentals



There are four main HTTP methods that can be supplied within the request. These methods help in specifying the operation requested by the user. Below table lists down all methods along with their explanations:

Method	Description
1. GET	The HTTP GET method helps in requesting for the representation of a specific resource by the client. The requests having GET just retrieves data and without causing any effect.
2. POST	The HTTP POST method helps in requesting the server to accept the data that is enclosed within the request as a new object of the resource as identified by the URI.
3. PUT	The HTTP PUT method helps in requesting the server to accept the data that is enclosed within the request as an alteration to the existing object which is identified by the provided URI.
4. DELETE	The HTTP DELETE method helps in requesting the server to delete a specific resource from the destination.

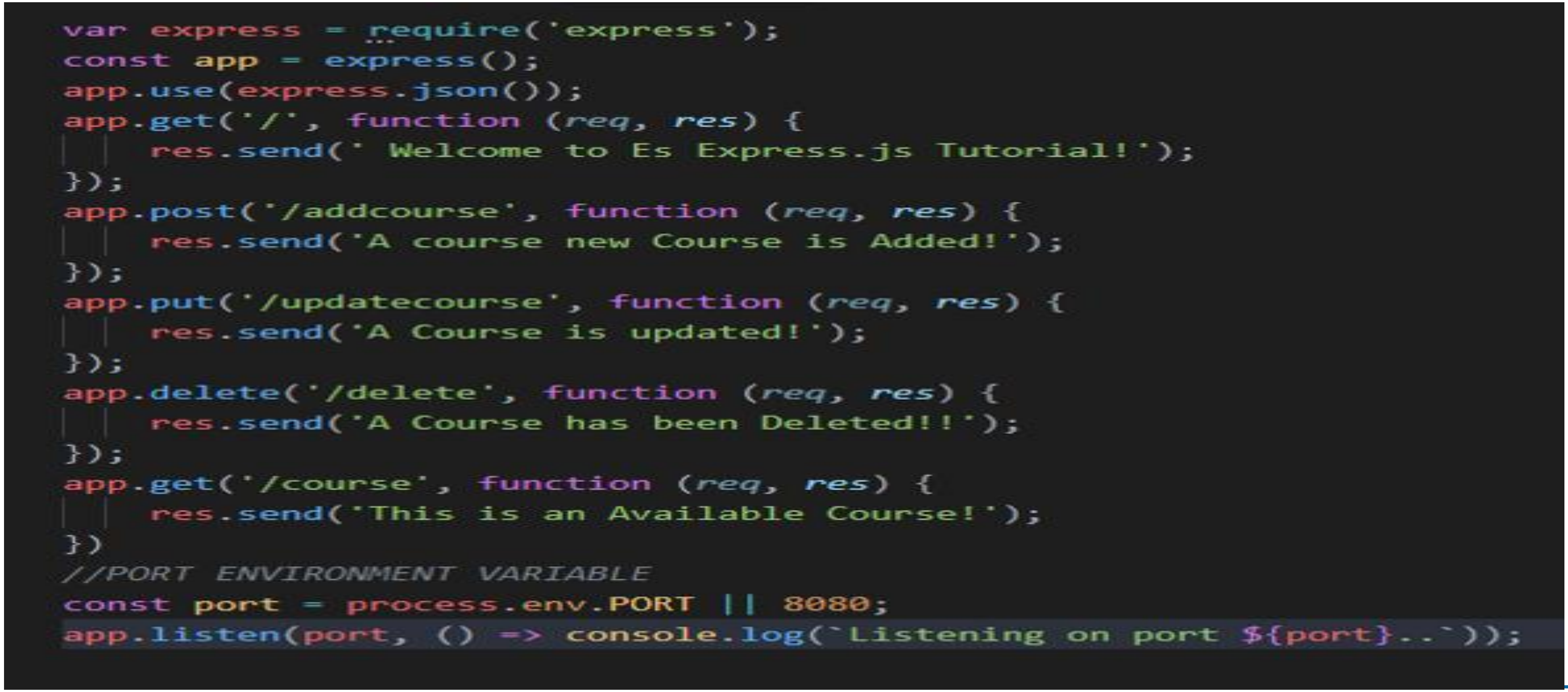
Express.js Fundamentals



There are four main HTTP methods that can be supplied within the request. These methods help in specifying the operation requested by the user. Below table lists down all methods along with their explanations:

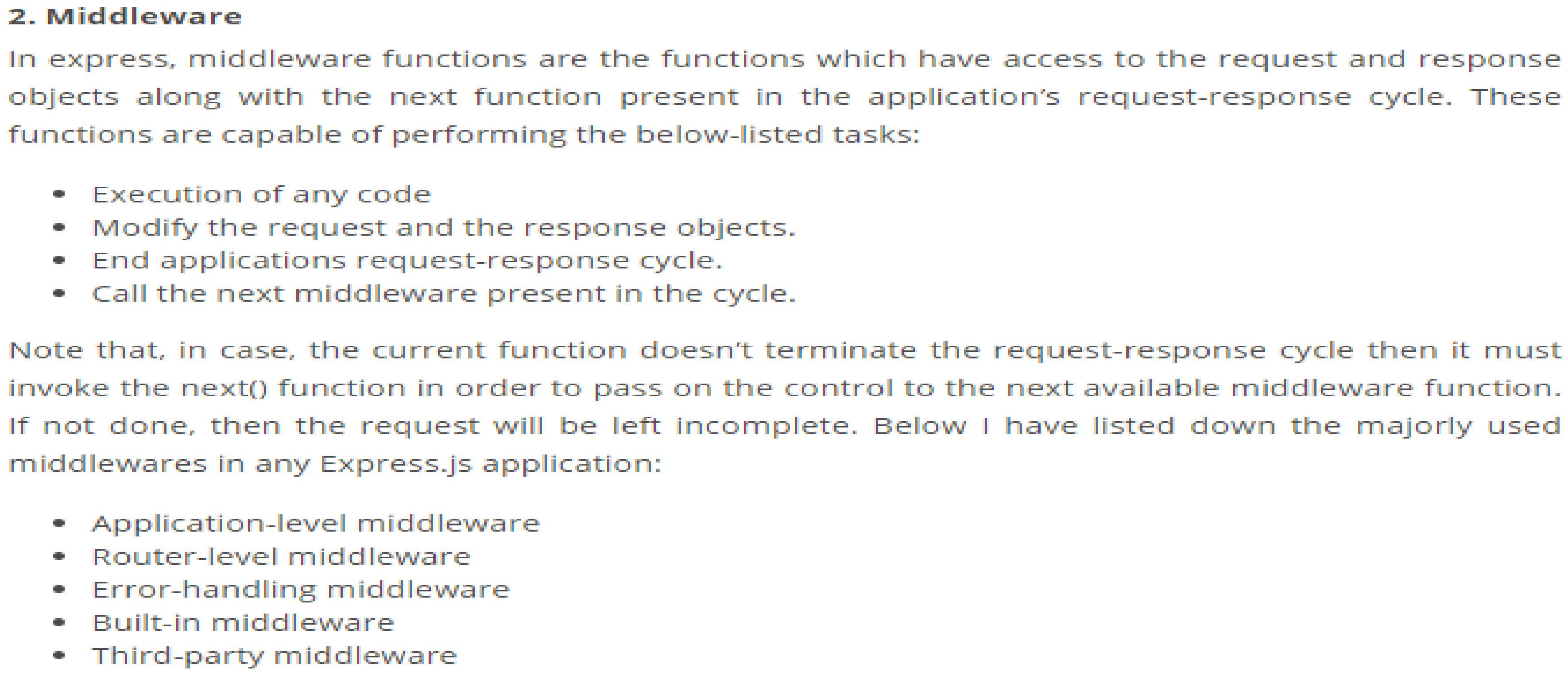
Method	Description
1. GET	The HTTP GET method helps in requesting for the representation of a specific resource by the client. The requests having GET just retrieves data and without causing any effect.
2. POST	The HTTP POST method helps in requesting the server to accept the data that is enclosed within the request as a new object of the resource as identified by the URI.
3. PUT	The HTTP PUT method helps in requesting the server to accept the data that is enclosed within the request as an alteration to the existing object which is identified by the provided URI.
4. DELETE	The HTTP DELETE method helps in requesting the server to delete a specific resource from the destination.

Application Development with Express.js



```
var express = require('express');
const app = express();
app.use(express.json());
app.get('/', function (req, res) {
    res.send(' Welcome to Es Express.js Tutorial!');
});
app.post('/addcourse', function (req, res) {
    res.send('A course new Course is Added!');
});
app.put('/updatecourse', function (req, res) {
    res.send('A Course is updated!');
});
app.delete('/delete', function (req, res) {
    res.send('A Course has been Deleted!!');
});
app.get('/course', function (req, res) {
    res.send('This is an Available Course!');
})
//PORT ENVIRONMENT VARIABLE
const port = process.env.PORT || 8080;
app.listen(port, () => console.log(`Listening on port ${port}..`));
```

Express.js Fundamentals



2. Middleware

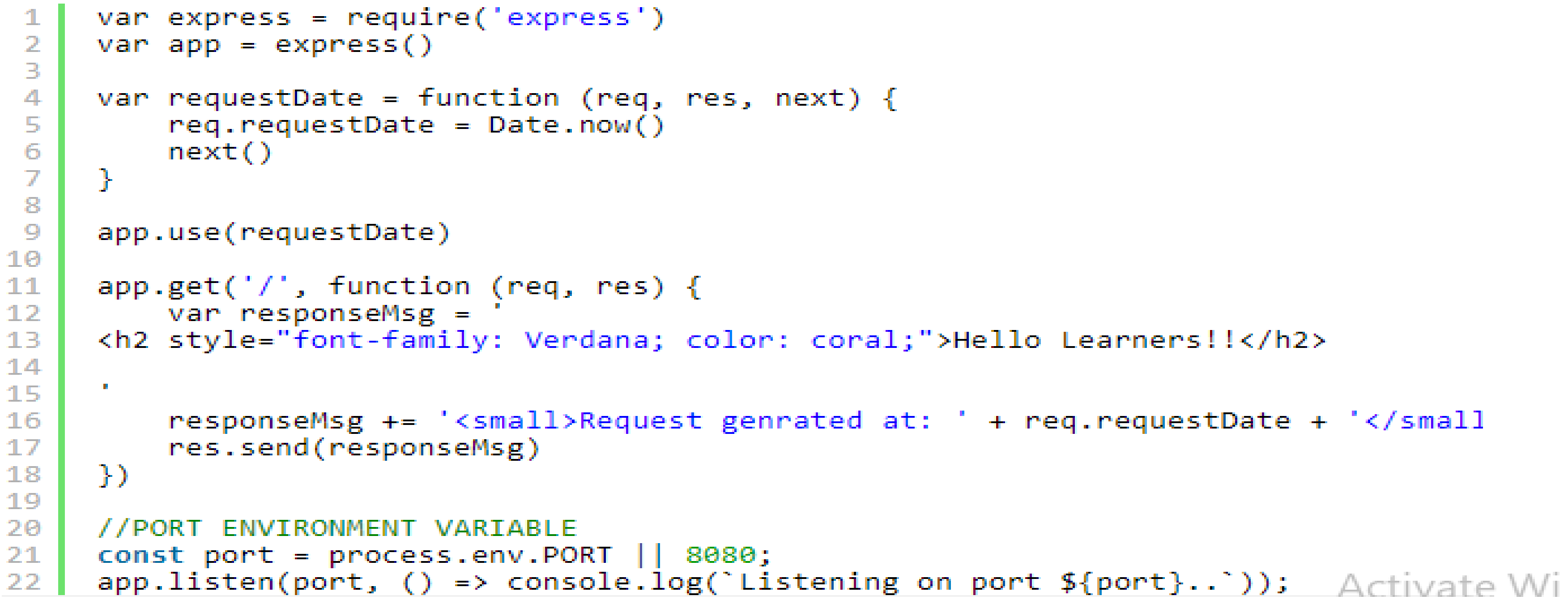
In express, middleware functions are the functions which have access to the request and response objects along with the next function present in the application's request-response cycle. These functions are capable of performing the below-listed tasks:

- Execution of any code
- Modify the request and the response objects.
- End applications request-response cycle.
- Call the next middleware present in the cycle.

Note that, in case, the current function doesn't terminate the request-response cycle then it must invoke the next() function in order to pass on the control to the next available middleware function. If not done, then the request will be left incomplete. Below I have listed down the majorly used middlewares in any Express.js application:

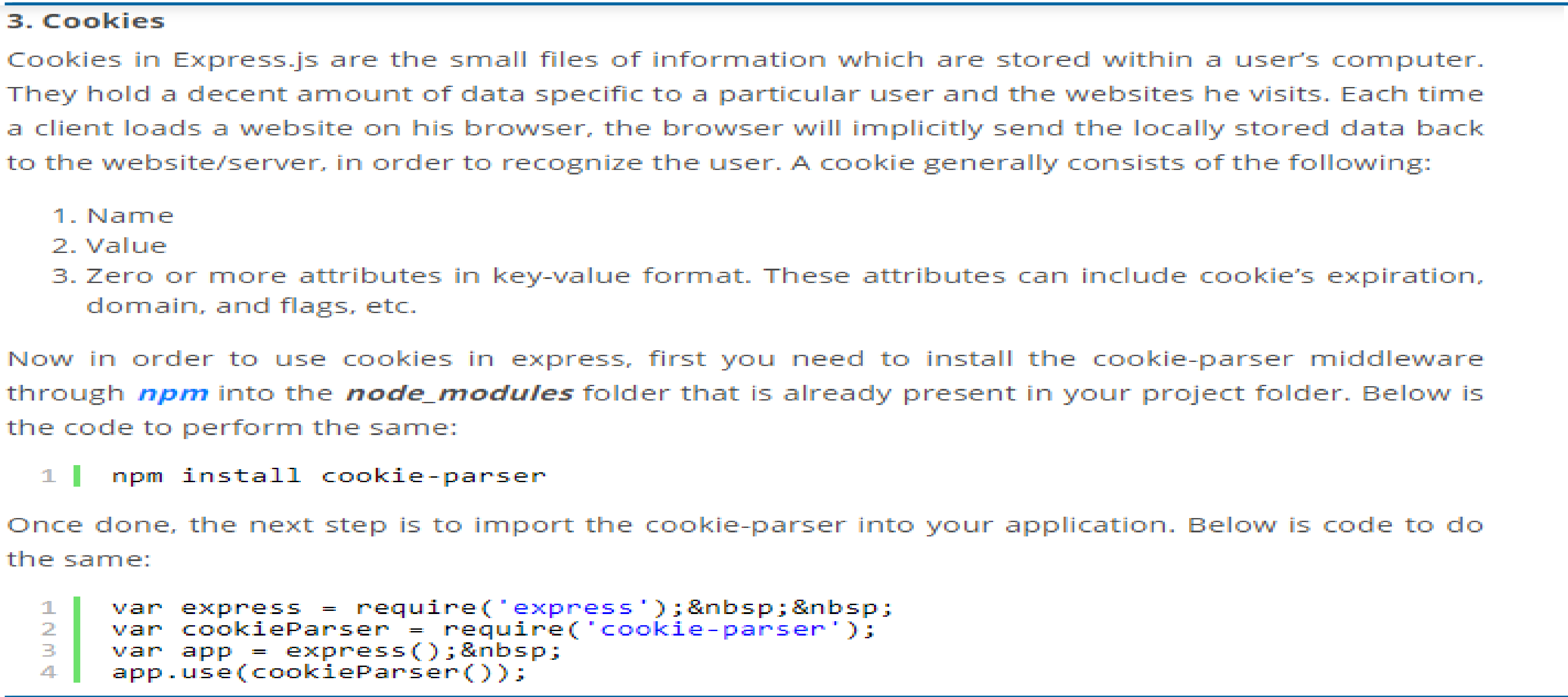
- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware

Application Development with Express.js



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

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

app.use(requestDate)

app.get('/', function (req, res) {
    var responseMsg = `
<h2 style="font-family: Verdana; color: coral;">Hello Learners!!</h2>

`
    responseMsg += '<small>Request genrated at: ' + req.requestDate + '</small
    res.send(responseMsg)
})

//PORT ENVIRONMENT VARIABLE
const port = process.env.PORT || 8080;
app.listen(port, () => console.log(`Listening on port ${port}..`));
```

Application Development with Express.js



3. Cookies

Cookies in Express.js are the small files of information which are stored within a user's computer. They hold a decent amount of data specific to a particular user and the websites he visits. Each time a client loads a website on his browser, the browser will implicitly send the locally stored data back to the website/server, in order to recognize the user. A cookie generally consists of the following:

1. Name
2. Value
3. Zero or more attributes in key-value format. These attributes can include cookie's expiration, domain, and flags, etc.

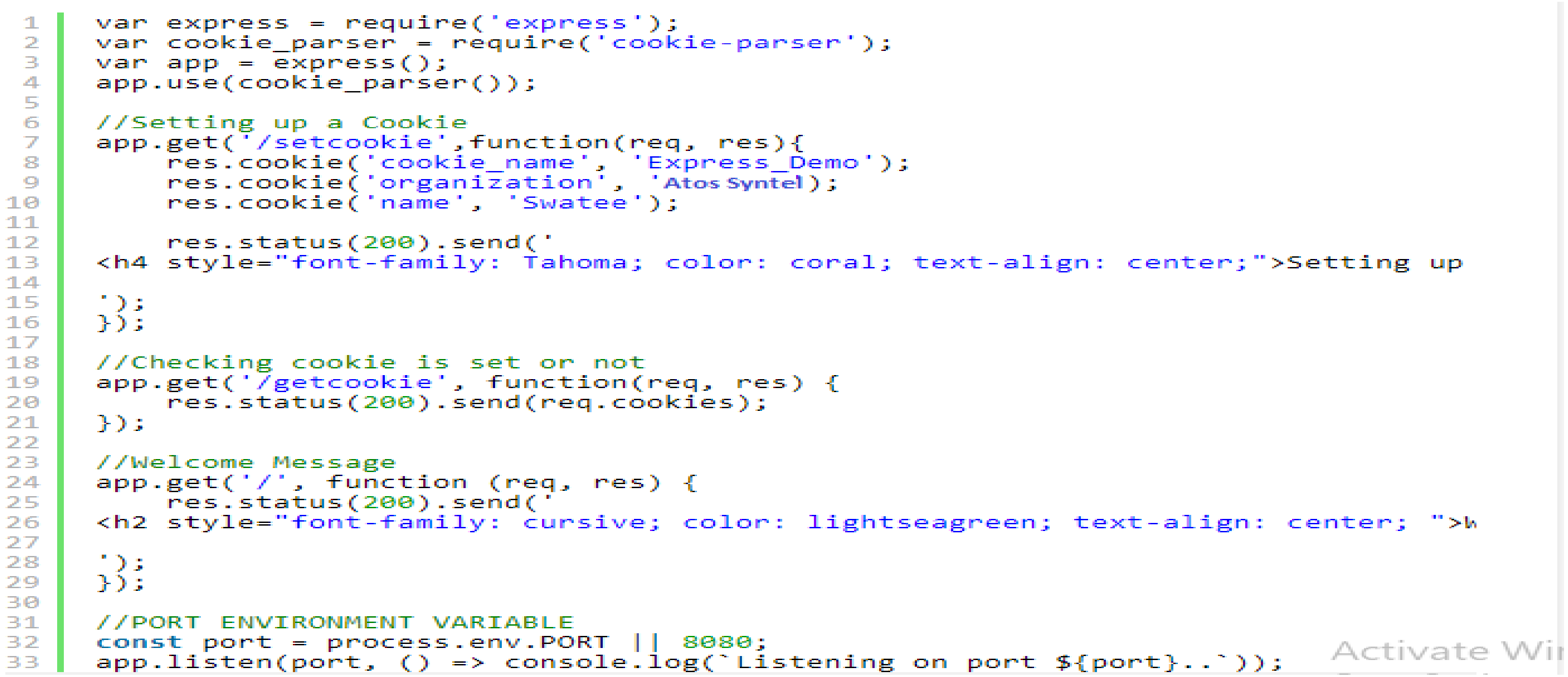
Now in order to use cookies in express, first you need to install the cookie-parser middleware through ***npm*** into the ***node_modules*** folder that is already present in your project folder. Below is the code to perform the same:

```
npm install cookie-parser
```

Once done, the next step is to import the cookie-parser into your application. Below is code to do the same:

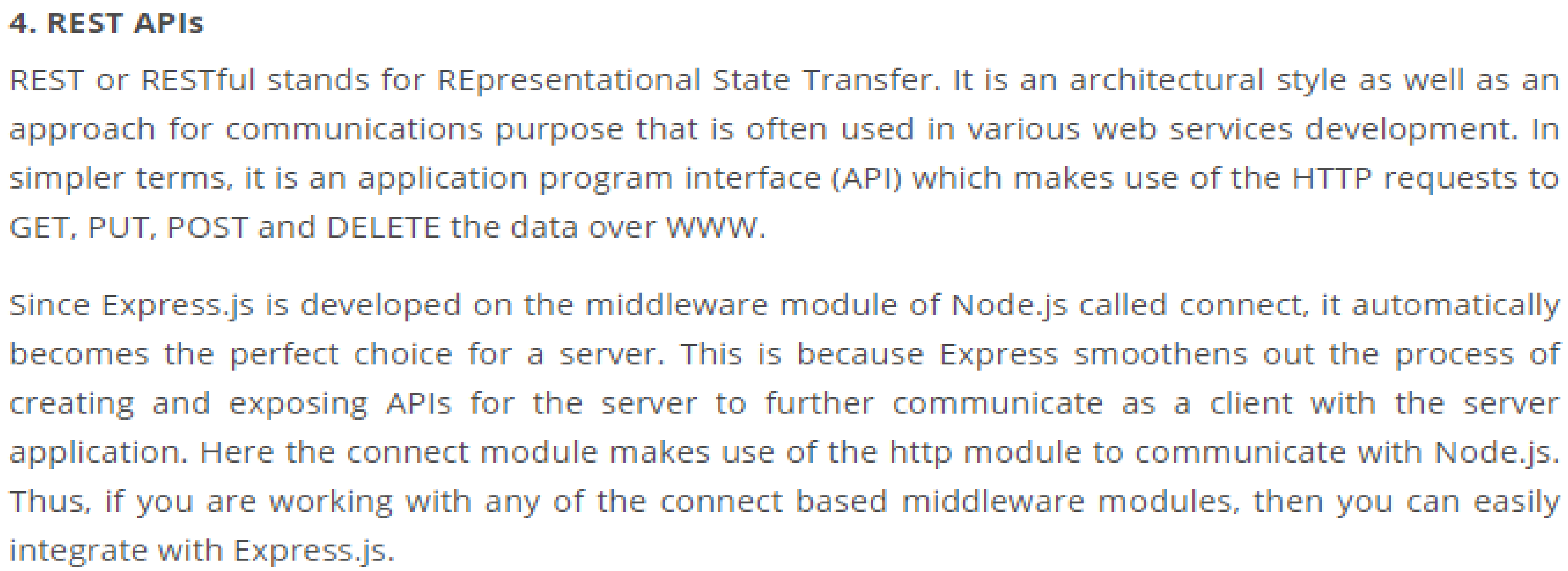
```
var express = require('express');&nbsp;&nbsp;
var cookieParser = require('cookie-parser');
var app = express();&nbsp;
app.use(cookieParser());
```

Application Development with Express.js



```
var express = require('express');
var cookie_parser = require('cookie-parser');
var app = express();
app.use(cookie_parser());

//Setting up a Cookie
app.get('/setcookie',function(req, res){
    res.cookie('cookie_name', 'Express_Demo');
    res.cookie('organization', 'Atos Syntel);
    res.cookie('name', 'Swatee');

    res.status(200).send('
<h4 style="font-family: Tahoma; color: coral; text-align: center;">Setting up

');
});

//Checking cookie is set or not
app.get('/getcookie', function(req, res) {
    res.status(200).send(req.cookies);
});

//Welcome Message
app.get('/', function (req, res) {
    res.status(200).send('
<h2 style="font-family: cursive; color: lightseagreen; text-align: center; ">W

');
});

//PORT ENVIRONMENT VARIABLE
const port = process.env.PORT || 8080;
app.listen(port, () => console.log(`Listening on port ${port}..`));
```

REST APIs

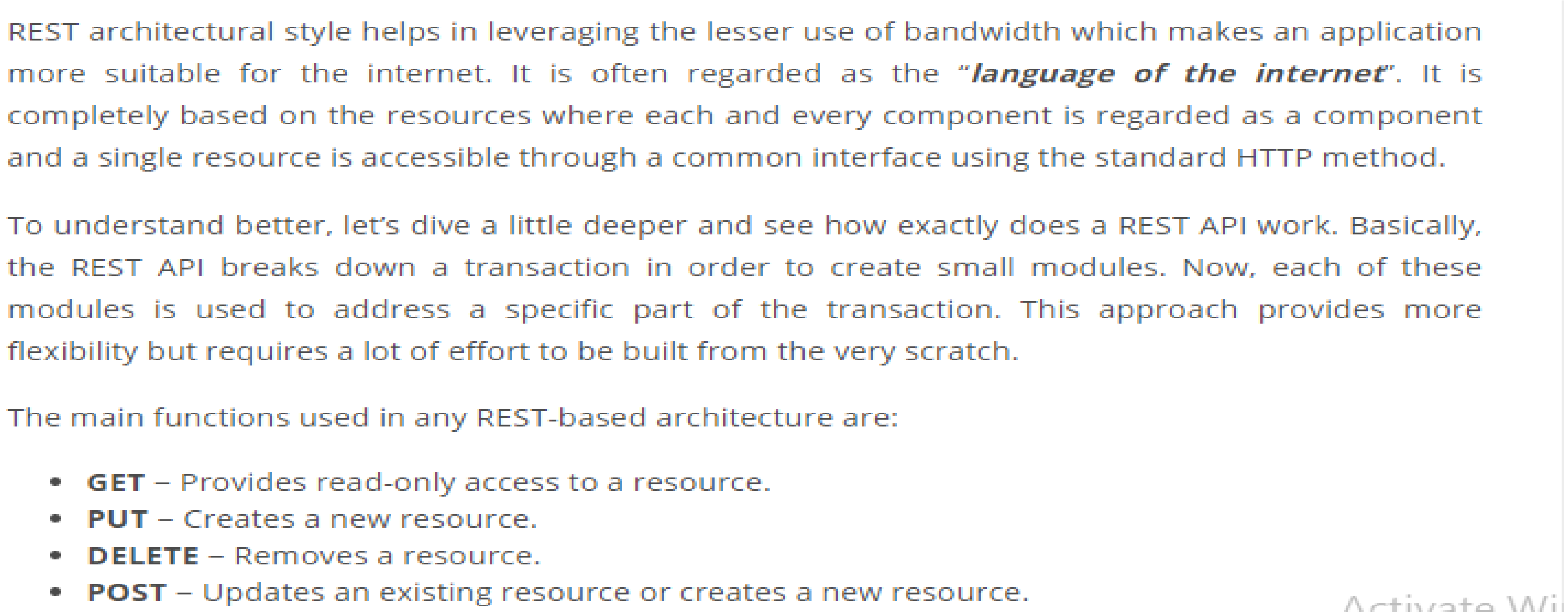


4. REST APIs

REST or RESTful stands for REpresentational State Transfer. It is an architectural style as well as an approach for communications purpose that is often used in various web services development. In simpler terms, it is an application program interface (API) which makes use of the HTTP requests to GET, PUT, POST and DELETE the data over WWW.

Since Express.js is developed on the middleware module of Node.js called connect, it automatically becomes the perfect choice for a server. This is because Express smoothens out the process of creating and exposing APIs for the server to further communicate as a client with the server application. Here the connect module makes use of the http module to communicate with Node.js. Thus, if you are working with any of the connect based middleware modules, then you can easily integrate with Express.js.

REST APIs



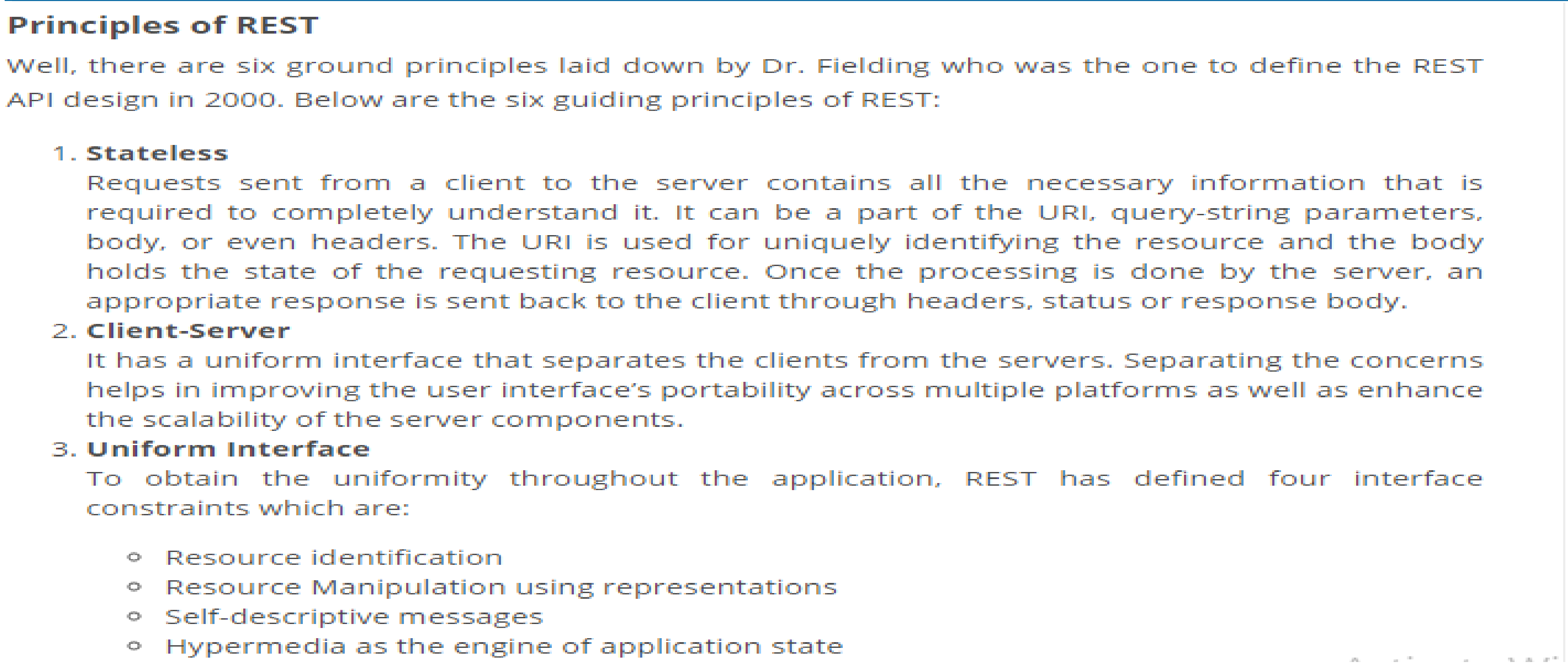
REST architectural style helps in leveraging the lesser use of bandwidth which makes an application more suitable for the internet. It is often regarded as the "***language of the internet***". It is completely based on the resources where each and every component is regarded as a component and a single resource is accessible through a common interface using the standard HTTP method.

To understand better, let's dive a little deeper and see how exactly does a REST API work. Basically, the REST API breaks down a transaction in order to create small modules. Now, each of these modules is used to address a specific part of the transaction. This approach provides more flexibility but requires a lot of effort to be built from the very scratch.

The main functions used in any REST-based architecture are:

- **GET** – Provides read-only access to a resource.
- **PUT** – Creates a new resource.
- **DELETE** – Removes a resource.
- **POST** – Updates an existing resource or creates a new resource.

REST APIs

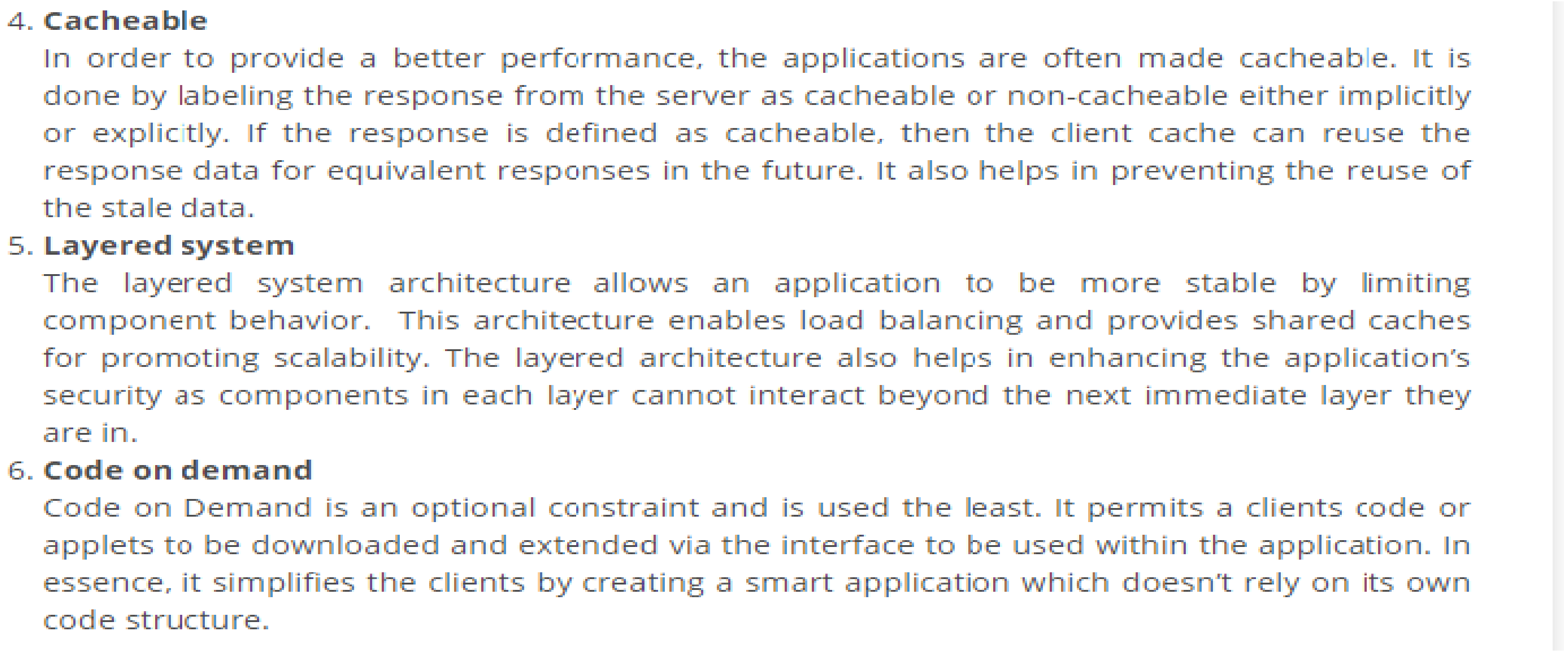


Principles of REST

Well, there are six ground principles laid down by Dr. Fielding who was the one to define the REST API design in 2000. Below are the six guiding principles of REST:

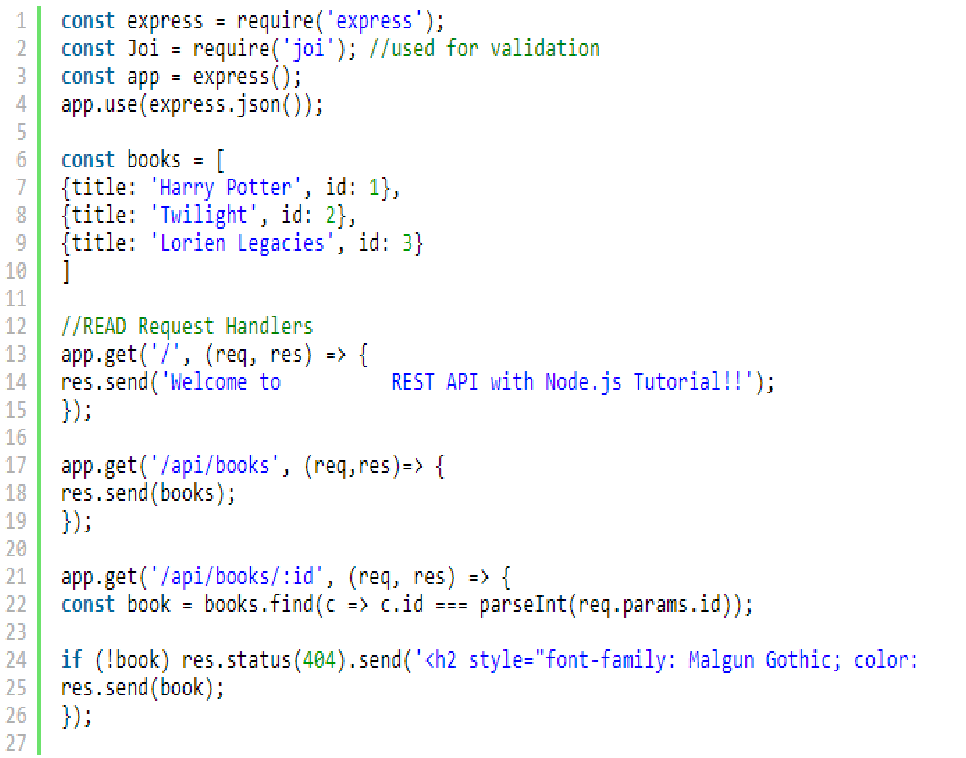
1. **Stateless**
   Requests sent from a client to the server contains all the necessary information that is required to completely understand it. It can be a part of the URI, query-string parameters, body, or even headers. The URI is used for uniquely identifying the resource and the body holds the state of the requesting resource. Once the processing is done by the server, an appropriate response is sent back to the client through headers, status or response body.
2. **Client-Server**
   It has a uniform interface that separates the clients from the servers. Separating the concerns helps in improving the user interface's portability across multiple platforms as well as enhance the scalability of the server components.
3. **Uniform Interface**
   To obtain the uniformity throughout the application, REST has defined four interface constraints which are:
   - Resource identification
   - Resource Manipulation using representations
   - Self-descriptive messages
   - Hypermedia as the engine of application state

REST APIs

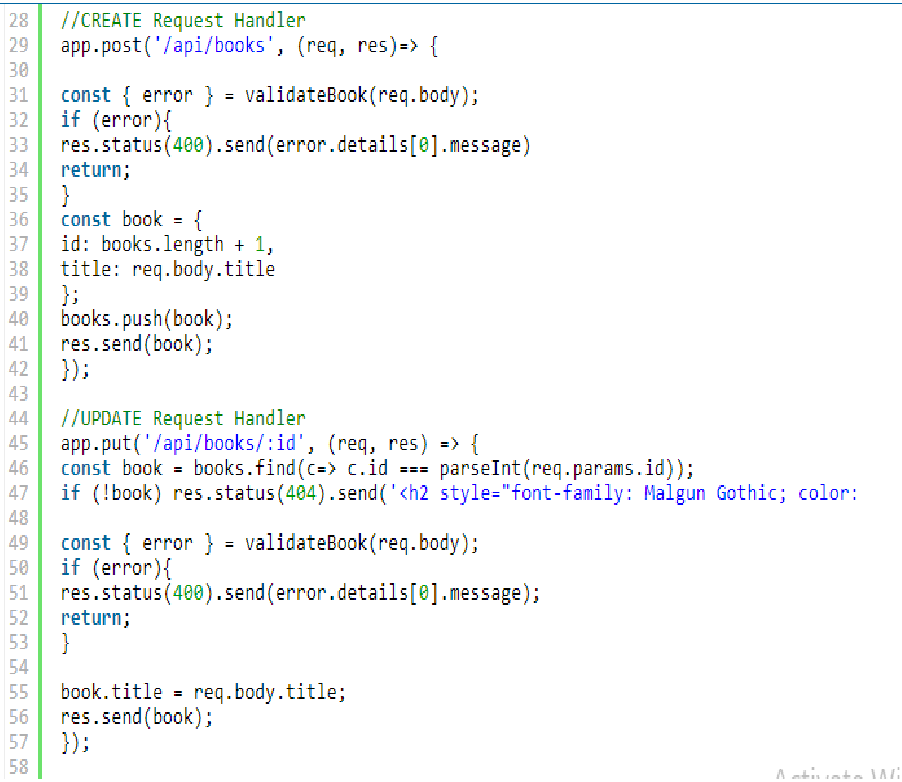


4. **Cacheable**
   In order to provide a better performance, the applications are often made cacheable. It is done by labeling the response from the server as cacheable or non-cacheable either implicitly or explicitly. If the response is defined as cacheable, then the client cache can reuse the response data for equivalent responses in the future. It also helps in preventing the reuse of the stale data.
5. **Layered system**
   The layered system architecture allows an application to be more stable by limiting component behavior. This architecture enables load balancing and provides shared caches for promoting scalability. The layered architecture also helps in enhancing the application's security as components in each layer cannot interact beyond the next immediate layer they are in.
6. **Code on demand**
   Code on Demand is an optional constraint and is used the least. It permits a clients code or applets to be downloaded and extended via the interface to be used within the application. In essence, it simplifies the clients by creating a smart application which doesn't rely on its own code structure.

REST APIs

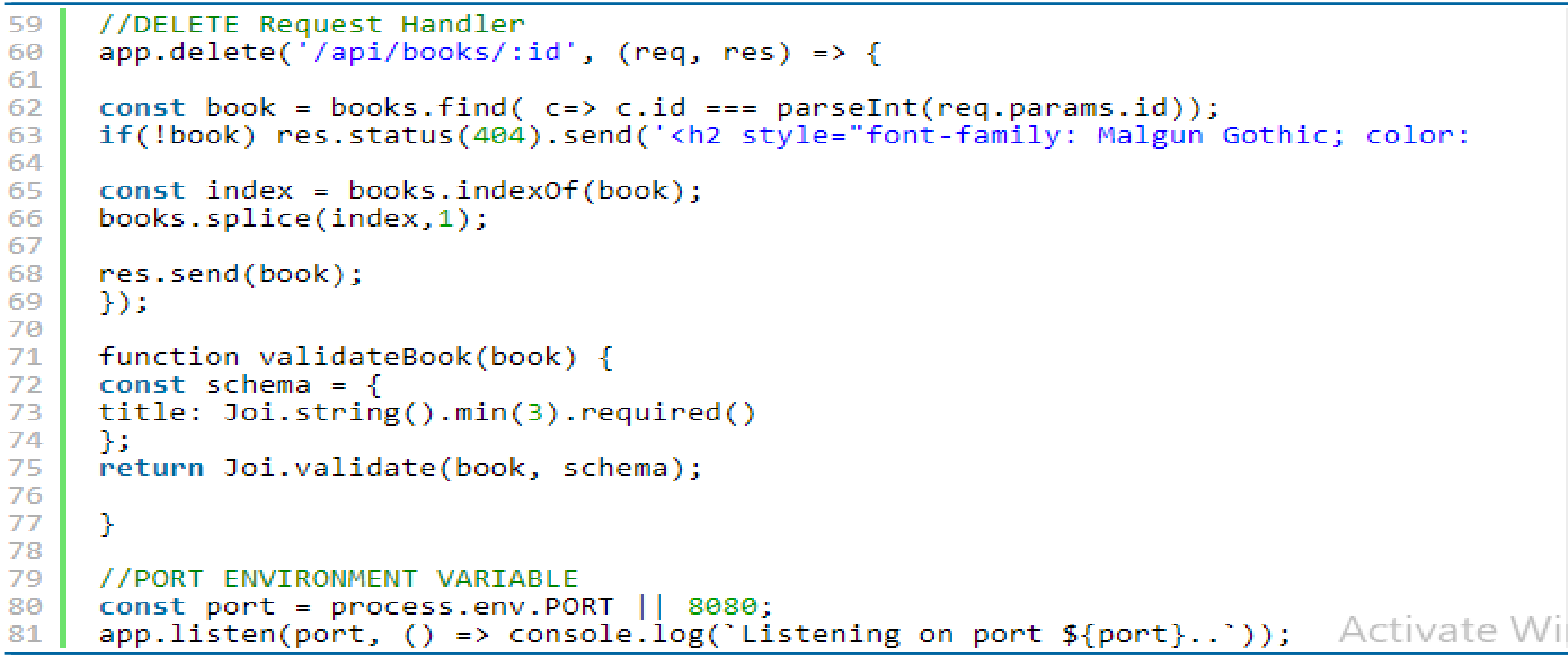


```
const express = require('express');
const Joi = require('joi'); //used for validation
const app = express();
app.use(express.json());

const books = [
{title: 'Harry Potter', id: 1},
{title: 'Twilight', id: 2},
{title: 'Lorien Legacies', id: 3}
]

//READ Request Handlers
app.get('/', (req, res) => {
res.send('Welcome to         REST API with Node.js Tutorial!!');
});

app.get('/api/books', (req,res)=> {
res.send(books);
});

app.get('/api/books/:id', (req, res) => {
const book = books.find(c => c.id === parseInt(req.params.id));

if (!book) res.status(404).send('<h2 style="font-family: Malgun Gothic; color:
res.send(book);
});

```



```
//CREATE Request Handler
app.post('/api/books', (req, res)=> {

const { error } = validateBook(req.body);
if (error){
res.status(400).send(error.details[0].message)
return;
}
const book = {
id: books.length + 1,
title: req.body.title
};
books.push(book);
res.send(book);
});

//UPDATE Request Handler
app.put('/api/books/:id', (req, res) => {
const book = books.find(c=> c.id === parseInt(req.params.id));
if (!book) res.status(404).send('<h2 style="font-family: Malgun Gothic; color:

const { error } = validateBook(req.body);
if (error){
res.status(400).send(error.details[0].message);
return;
}

book.title = req.body.title;
res.send(book);
});

```

REST APIs



```
//DELETE Request Handler
app.delete('/api/books/:id', (req, res) => {

const book = books.find( c=> c.id === parseInt(req.params.id));
if(!book) res.status(404).send('<h2 style="font-family: Malgun Gothic; color:

const index = books.indexOf(book);
books.splice(index,1);

res.send(book);
});

function validateBook(book) {
const schema = {
title: Joi.string().min(3).required()
};
return Joi.validate(book, schema);

}

//PORT ENVIRONMENT VARIABLE
const port = process.env.PORT || 8080;
app.listen(port, () => console.log(`Listening on port ${port}..`));
```

Thank You

For more information please contact:

M +91 91586 52627
pradeep.chinchole@atos.net