EXPRESS.JS

Prof. Rachana V. Modi

Node.js Frameworks

- Hapi.js
- Express.js
- Koa.js
- Sails.js
- Meteor.js
- Derby.js
- Total.js
- Adonis.js
- Nest.js
- LoopBack.js

What is Express.js?

- Express.js is a Node js web application server framework.
- Developed by **TJ Holowaychuk** and maintained by the Node.js foundation.
- Express is a fast, robust, asynchronous and open source web framework of Node.js.
- Express works as top layer of the Node.js that helps manage a server and routes. It provides a robust set of features to develop web and mobile applications.

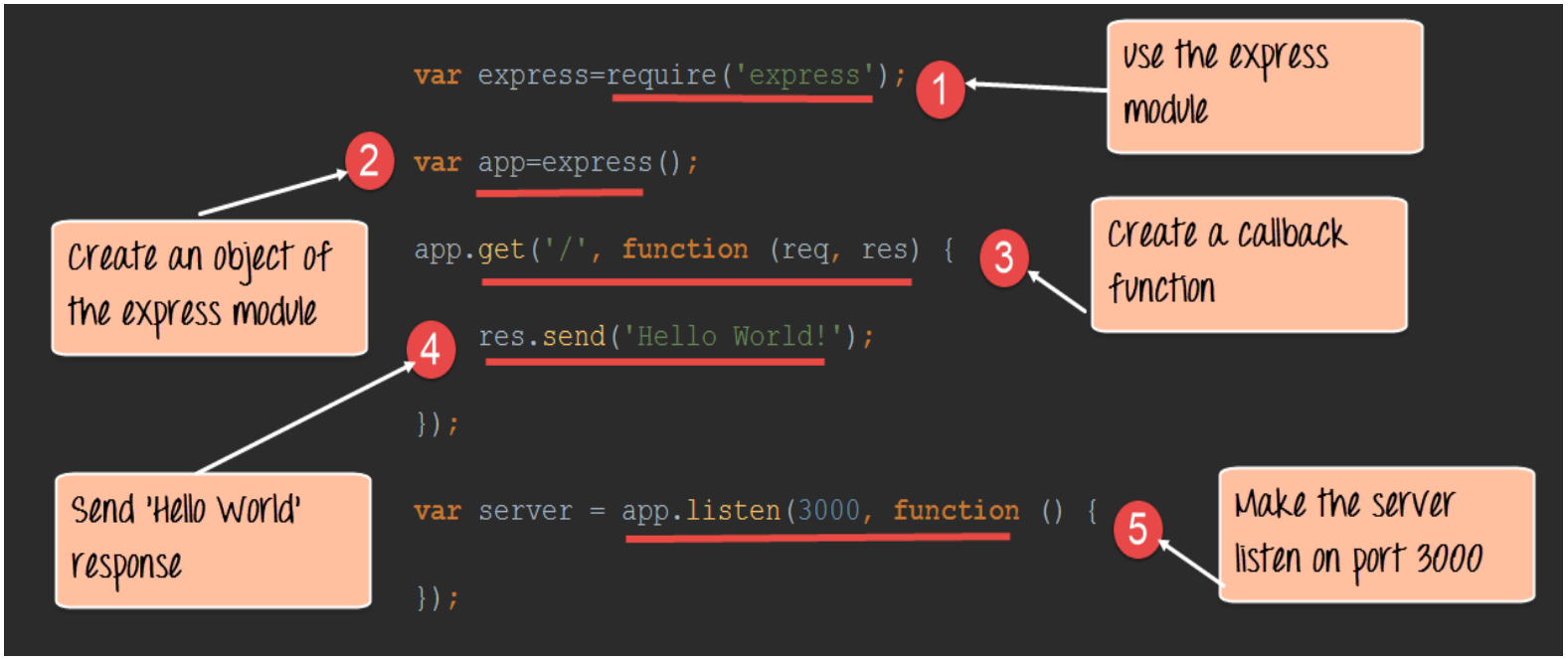
- **Installation of Express**

  npm install express

Features of Express.js

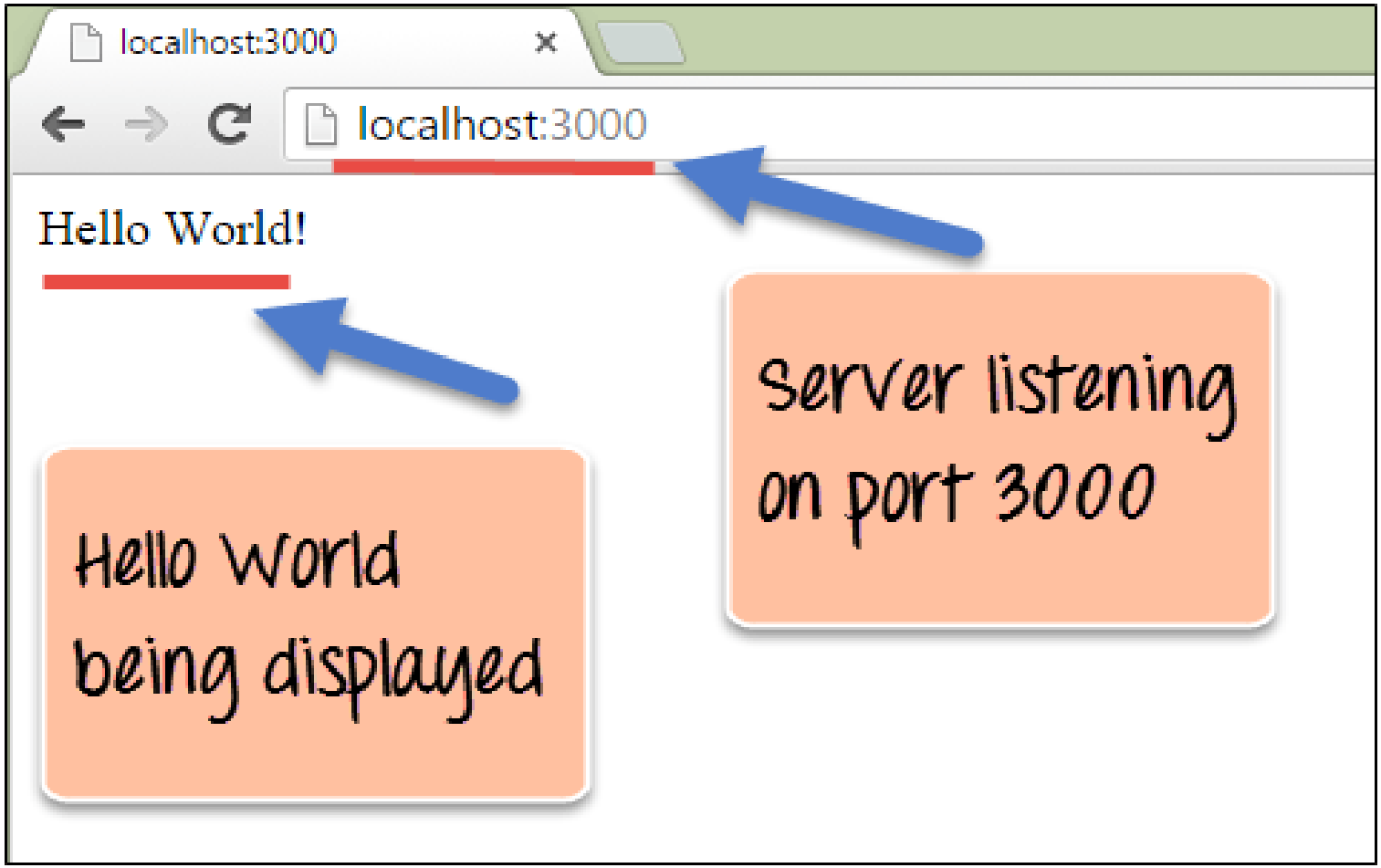
- It can be used to design single-page, multi-page and hybrid web applications.
- Easy to configure and customize.
- It allows to setup middlewares to respond to HTTP Requests.
- It defines a routing table which is used to perform different actions based on HTTP method and URL.
- It allows to dynamically render HTML Pages based on passing arguments to templates.

Sample Web Application



- The Request object (req) represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers and so on.
- The Response object (res) specifies the HTTP response which is sent by an Express app when it gets an HTTP request.
- Methods: Send(), sendFile(), redirect(), render(), json(),link(),location(),type(),status()

Output


localhost:3000
localhost:3000
Hello World!
Server listening on port 3000
Hello World being displayed

What is Route?

- Route determine the way in which an application responds to a client request to a particular endpoint.

- Example:

  http://localhost:3000/Books

  http://localhost:3000/Students

- If a GET request is made for the first URL, then the response should ideally be a list of books.
- If the GET request is made for the second URL, then the response should ideally be a list of Students.
- So based on the URL which is accessed, a different functionality on the webserver will be invoked and accordingly, the response will be sent to the client. This is the concept of routing.

Route

- Each route can have one or more handler functions, which are executed when the route is matched.

- Syntax for a Route:

  app.METHOD(PATH, HANDLER)

1) app is an instance of the express module
2) METHOD is an HTTP request method (GET, POST, PUT or DELETE)
3) PATH is a path on the server
4) HANDLER is the function executed when the route is matched

Route Example

```
const express1 = require('express')
const app = express1()
const port = 3000

app.get('/', (req, res) => {
  res.send('Welcome to GANPAT UNIVERSITY')
})
 app.route('/Node').get(function(req,res)
{
    res.send("uvpcesite on Node");
});
app.route('/Angular').get(function(req,res)
{
    res.send("uvpcesite on Angular");
});
 app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

EXAMPLES OF ROUTE PATHS BASED ON STRINGS

- **This route path will match requests to the root route, /.**

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

- **This route path will match requests to /about.**

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

- **This route path will match requests to /random.text.**

```
app.get('/random.text', function (req, res) {
  res.send('random.text')
})
```

EXAMPLES OF ROUTE PATHS BASED ON STRING PATTERNS

- **This route path will match acd and abcd. ?: 0 or 1 occurrence**

```
app.get('/ab?cd', function (req, res) {
  res.send('ab?cd')
})
```

- **This route path will match abcd, abbcd, abbbcd, and so on. + : 1 or more occurrence**

```
app.get('/ab+cd', function (req, res) {
  res.send('ab+cd')
})
```

EXAMPLES OF ROUTE PATHS BASED ON STRING PATTERNS

- **This route path will match abcd, abxcd, abRANDOMcd, ab123cd, and so on. *: 0 or more occurrence for any characters**

```
app.get('/ab*cd', function (req, res) {
  res.send('ab*cd')
})
```

- **This route path will match /abe and /abcde.**

  ( ) ?: 0 or 1 occurrence for grouping

```
app.get('/ab(cd)?e', function (req, res) {
  res.send('ab(cd)?e')
})
```

Examples of route paths based on regular expressions

- **This route path will match anything with an “a” in it.**

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

- **This route path will match butterfly and dragonfly, but not butterflyman, dragonflyman, and so on.**

```
app.get(/.*fly$/, function (req, res) {
  res.send('/.*fly$/')
})
```

Routing Parameters

- Route parameters are named URL segments that are used to capture the values specified at their position in the URL.
- req.params object is used to access all the parameters passed in the url.

- Example: http://localhost:3000/books/23

  app.get('/books/:bookId', (req, res) => {
  res.send(req.params); });
- Example:

  Route path: /users/:userId/books/:bookId

  Request URL: http://localhost:3000/users/34/books/8989
  req.params: { "userId": "34", "bookId": "8989" }

Routing Parameters

- The hyphen (-) and the dot (.) are interpreted literally, they can be used along with route parameters for useful purposes.

Example:

Route path: /flights/:from-:to

Request URL: http://localhost:3000/flights/LAX-SFO

req.params: { "from": "LAX", "to": "SFO" }

Example:

Route path: /plantae/:genus.:species

Request URL: http://localhost:3000/plantae/Prunus.persica

req.params: { "genus": "Prunus", "species": "persica" }

Routing Parameters

- The hyphen (-) and the dot (.) are interpreted literally, they can be used along with route parameters for useful purposes.

Example:

Route path: /flights/:from-:to

Request URL: http://localhost:3000/flights/LAX-SFO

req.params: { "from": "LAX", "to": "SFO" }

Example:

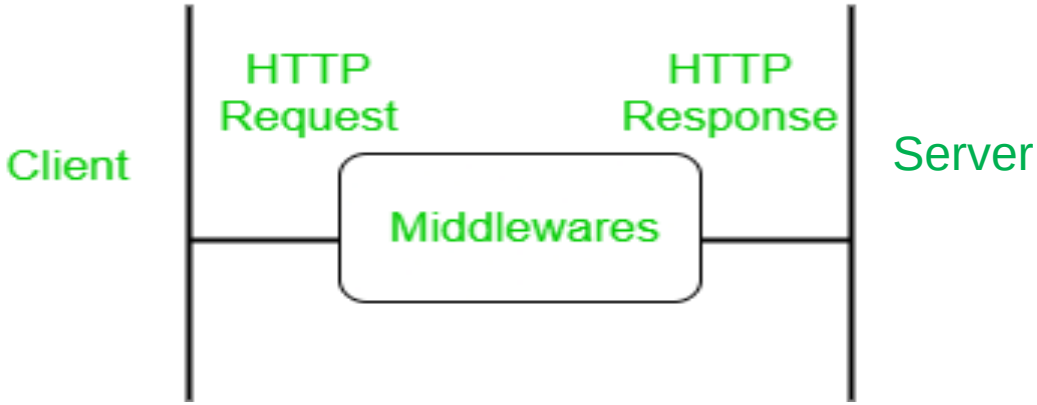
Route path: /plantae/:genus.:species

Request URL: http://localhost:3000/plantae/Prunus.persica

req.params: { "genus": "Prunus", "species": "persica" }

Express Middleware

- Middleware works between the request and response cycle.



- Middleware gets executed after the server receives the request and before the controller sends the response.
- Middleware has the access to the request object, responses object and next method. (Arguments of middleware)

Express Middleware

Advantages of Middleware:

- 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 Middleware

Syntax:

```
const middleware1 = (req, res, next)=>
{ //execute some code
next() // pass execution to the next middleware
}
const middleware2 = (req, res, next)=>
{ //execute some code
}
app.get("/", middleware1, middleware2);
```

OR

```
app.get("/", function(req, res, next){
 // first middleware
next() //Pass execution to the next middleware
},
function(){ // second middleware })
```

Express Middleware

Example of Middleware:

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

app.get("/",  (req,res,next)=>{
console.log("Hello");
next();
},
(req, res) => {
  res.send(`<div>
    <h2>Welcome to SP</h2>
    <h5>Example of Middleware</h5>   </div>`);
});
app.listen(3000);
```

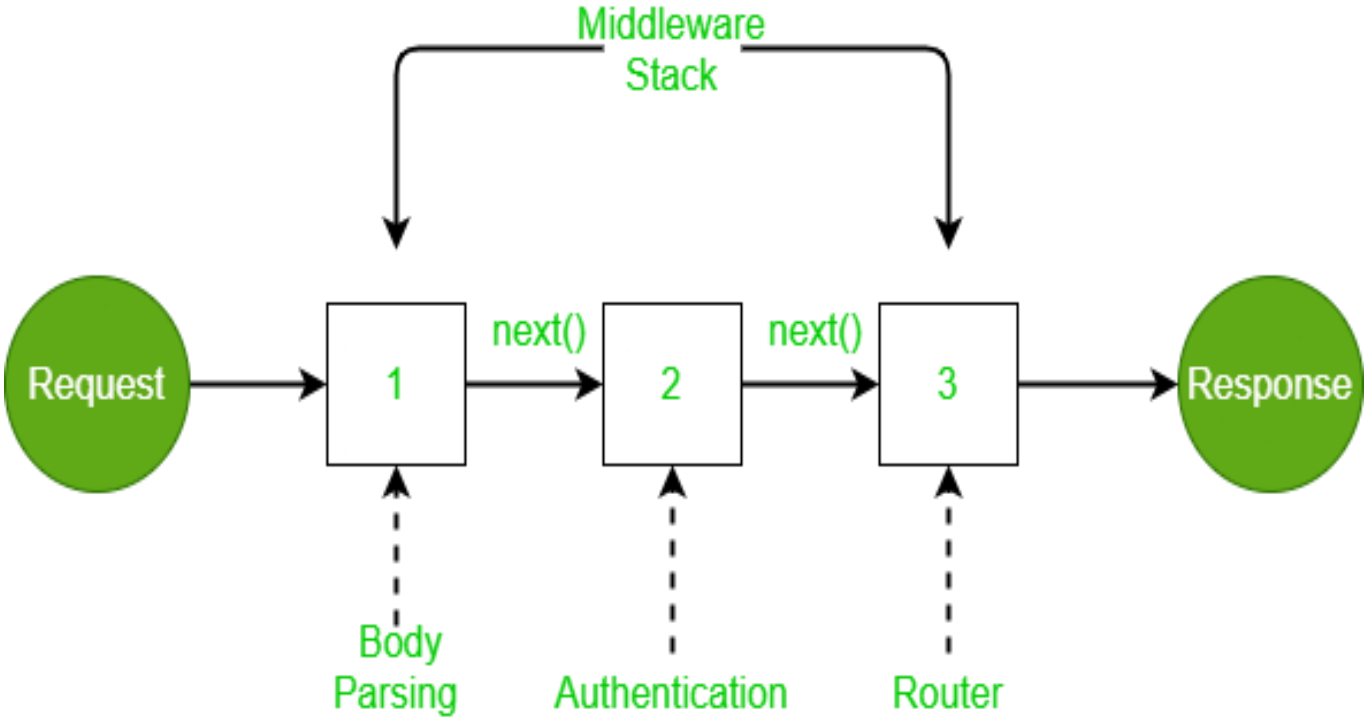
Express Middleware

Middleware Chaining:

- Middleware can be make chain between more than one middleware.
- Creation a chain of functions that are executed in order.
- The last function sends the response back to the browser. So, before sending the response back to the browser the different middleware process the request object.
- The next() function in the express is responsible for calling the next middleware function if there is one.

Express Middleware

Middleware Chaining:



Types of Middleware

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

Express Middleware

App.use():

- Middlewares can be chained.
- App.use mehod is useful for binding application-level middleware to an instance of express (app object).
- **Syntax:** app.use() or app.METHOD().
  - METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.
  - Use a comma (,) to separate more than one middleware.
- A middleware binded with app.use() will be called for every call of request of an application.

Express Middleware

Example: Middleware will be called for every call of request of an application.

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

app.use((req, res, next) => {
  console.log(req.url);
  next();
});

app.get('/', (req, res, next) => {
  res.send('Welcome Page');
});
app.get('/home', (req, res, next) => {
    res.send('Home Page');
  });
app.listen(3000);
```

HTTP METHODS WITH EXAMPLE

- GET
- POST
- DELETE
- PUT

Express Router

- Router is used to create modular and mountable route handlers for Multiple requests.
- Router() is used to create a new router object in program to handle requests.
- A router instance is a complete middleware and routing system
- **Syntax:** express.Router([options])
- **Example: Single routing**

```
var router = express.Router();
router.get('/', function (req, res, next) {
    console.log("Router Working");
    res.end();
});
app.use(router);
```

EXPRESS.ROUTER()

Example: Multiple routing

```
var router1 = express.Router();
var router2 = express.Router();
var router3 = express.Router();

router1.get('/user', function (req, res, next) {
    console.log("User Router Working");
    res.end();
});
router2.get('/admin', function (req, res, next) {
    console.log("Admin Router Working");
    res.end();
});
 router3.get('/student', function (req, res, next) {
    console.log("Student Router Working");
    res.end();
});
app.use(router1);
app.use(router2);
app.use(router3);
```

Any query??