



Introduction to Node.js

Gehad Ismail Sayed
Lab 3
Software Engineering





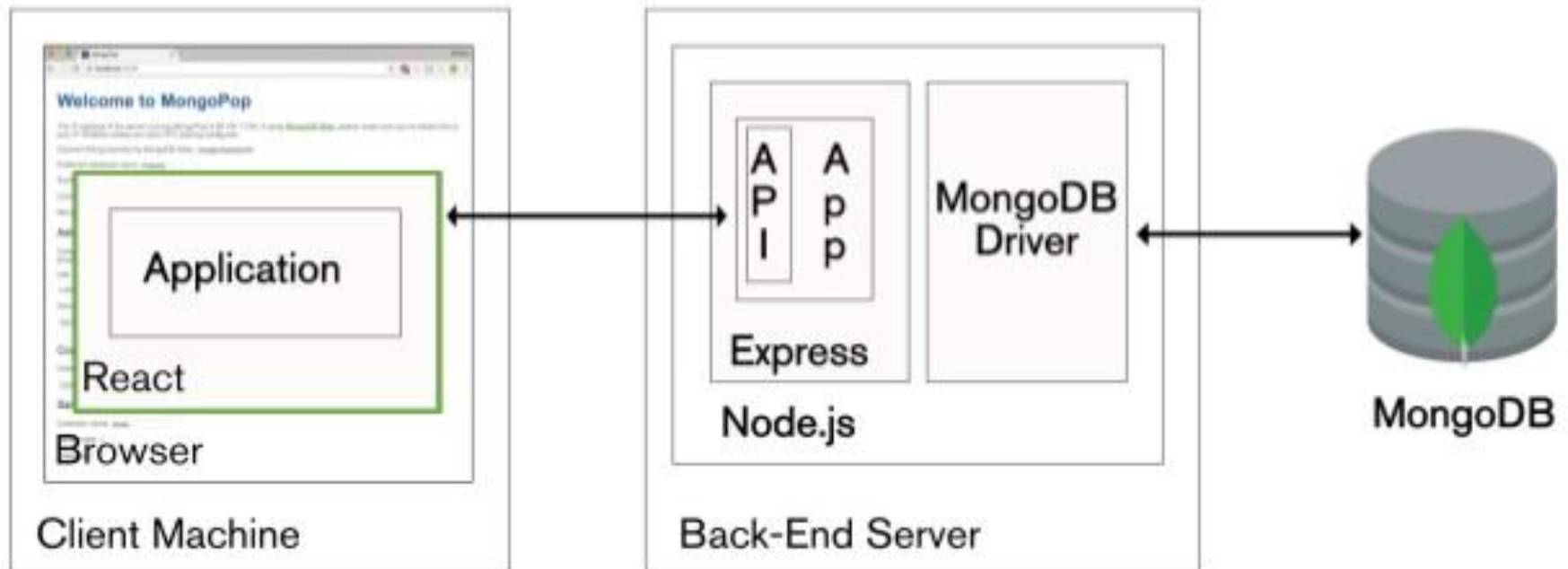
Overview

- MERN Stack
- What is Node.js?
- What can Node.js do?
- Node.js vs. PHP/ASP
- How to run Node.js?
- Node.js Built-in Modules
- Node.js as Web Server
- How the app work?
- ExpressJS
- Node Package Manager(npm)
- Postman



MERN Stack

The MERN Stack





What is Node.js?

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server



What can Node.js do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database



Node.js vs. PHP/ASP

- Here is how PHP/ASP handles a file request:
 - Sends the task to the computer's file system.
 - Waits while the file system opens and reads the file.
 - Returns the content to the client.
 - Ready to handle the next request.



Node.js vs. PHP/ASP

- Here is how Node.js handles a file request:
 - Sends the task to the computer's file system.
 - Ready to handle the next request.
 - When the file system has opened and read the file, the server returns the content to the client.



How to run Node.js?

- Install Node.js <https://nodejs.org>
- Create a Node.js file named "myfirst.js", and add the following code:
- ```
var http = require('http');
http.createServer(function (req, res)
{
 res.writeHead(200, { 'Content-
Type': 'text/html' });
 res.end('Hello World!');
}).listen(8080);
```

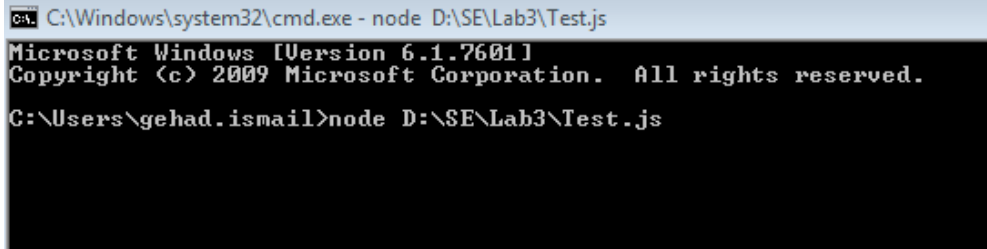




# How to run Node.js?

---

- Open Command Prompt
- Navigate to the folder that contains the file "ServerFileName.js" and initiate the Node.js File



```
C:\Windows\system32\cmd.exe - node D:\SE\Lab3\Test.js
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\gehad.ismail>node D:\SE\Lab3\Test.js
```

- If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!
- Start your internet browser, and type in the address: <http://localhost:8080>



# Node.js Built-in Modules

---

- It is the same as JavaScript libraries, it is a set of functions you want to include in your application.
- To include a module, use the `require()` function with the name of the module:
  - `var http = require('http');`
  - `var url = require('url');`
- You can create your own modules
  - Create a module that returns the current date and time:
  - `exports.myDateTime = function () {return Date();};`
  - Use the `exports` keyword to make properties and methods available outside the module file.
  - Save the code above in a file called "myfirstmodule.js"
  - Include your module using `var dt=require('./myfirstmodule.js');`



# Node.js Built-in Modules

| Module                         | Description                                            |
|--------------------------------|--------------------------------------------------------|
| <a href="#"><u>assert</u></a>  | Provides a set of assertion tests                      |
| <a href="#"><u>buffer</u></a>  | To handle binary data                                  |
| child_process                  | To run a child process                                 |
| <a href="#"><u>cluster</u></a> | To split a single Node process into multiple processes |
| <a href="#"><u>crypto</u></a>  | To handle OpenSSL cryptographic functions              |
| <a href="#"><u>dgram</u></a>   | Provides implementation of UDP datagram sockets        |
| <a href="#"><u>dns</u></a>     | To do DNS lookups and name resolution functions        |
| domain                         | Deprecated. To handle unhandled errors                 |
| <a href="#"><u>events</u></a>  | To handle events                                       |
| <a href="#"><u>fs</u></a>      | To handle the file system                              |
| <a href="#"><u>http</u></a>    | To make Node.js act as an HTTP server                  |
| <a href="#"><u>https</u></a>   | To make Node.js act as an HTTPS server.                |
| <a href="#"><u>net</u></a>     | To create servers and clients                          |
| <a href="#"><u>os</u></a>      | Provides information about the operation system        |



# Node.js Built-in Modules

| Module                                | Description                                                |
|---------------------------------------|------------------------------------------------------------|
| <a href="#"><u>path</u></a>           | To handle file paths                                       |
| <code>punycode</code>                 | Deprecated. A character encoding scheme                    |
| <a href="#"><u>querystring</u></a>    | To handle URL query strings                                |
| <a href="#"><u>readline</u></a>       | To handle readable streams one line at the time            |
| <a href="#"><u>stream</u></a>         | To handle streaming data                                   |
| <a href="#"><u>string_decoder</u></a> | To decode buffer objects into strings                      |
| <a href="#"><u>timers</u></a>         | To execute a function after a given number of milliseconds |
| <a href="#"><u>tls</u></a>            | To implement TLS and SSL protocols                         |
| <code>tty</code>                      | Provides classes used by a text terminal                   |
| <a href="#"><u>url</u></a>            | To parse URL strings                                       |
| <a href="#"><u>util</u></a>           | To access utility functions                                |
| <code>v8</code>                       | To access information about V8 (the JavaScript engine)     |
| <a href="#"><u>vm</u></a>             | To compile JavaScript code in a virtual machine            |
| <a href="#"><u>zlib</u></a>           | To compress or decompress files                            |



# Node.js as Web Server

---

- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- Use the `createServer()` method to create an HTTP server.
- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:
  - `res.writeHead(200, {'Content-Type': 'text/html'});`
  - `res.writeHead(404, {'Content-Type': 'text/html'});`



# Node.js as File Server

---

- The Node.js file system module allows you to work with the file system on your computer.
- To include the File System module, use the `require()` method:
  - `var fs = require('fs');`
- Common use for the File System module:
  - Read files
  - Create files
  - Update files
  - Delete files
  - Rename files



# Node.js as File Server

---

- Read files
  - `fs.readFile('demofile1.html', function(err, data)`
- Create files
  - `fs.appendFile()`
    - `fs.appendFile('mynewfile1.txt', 'Hello content!', function (err)`
  - `fs.open()`
    - `fs.open('mynewfile2.txt', 'w', function (err, file)`
  - `fs.writeFile()`
    - `fs.writeFile('mynewfile3.txt', 'This is my text', function (err)`



# Node.js as File Server

---

- Delete files
  - `fs.unlink('mynewfile2.txt', function (err)`
- Rename files
  - `fs.rename('mynewfile1.txt',  
'myrenamedfile.txt', function (err)`





# Node.js as File Server

---

- Upload files

- `var http = require('http');`
- `http.createServer(function (req, res) {`
- `res.writeHead(200,{'Content-Type':'text/html'});`
- `res.write('<form action="fileupload" method="post"`  
 `enctype="multipart/form-data">');`
- `res.write('<input type="file"`  
 `name="fileupload"><br>');`
- `res.write('<input type="submit">');`
- `res.write('</form>');`
- `return res.end();`
- `}).listen(8080);`



# How the app work?

---

- `app.get(route, callback)`
  - This function tells what to do when a get request at the given route is called.
- `app.post(route, callback)`
  - This function tells what to do when a post request at the given route is called.
- `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.



# ExpressJS

---

- 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
- ExpressJS is a web application framework that provides you with a simple API to build websites, web apps and back ends.
- You can simply install it
  - `npm install express --save`



# ExpressJS

---

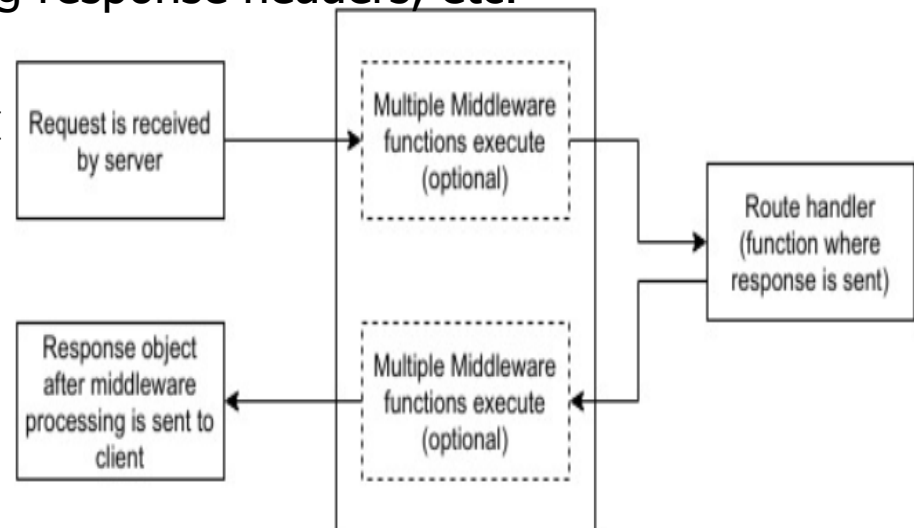
- Use `require('express')` to enable your application to use Express
  - `var express = require('express');`
  - `var app = express();`
  - `app.get('/', function(req, res){`
  - `res.send("Hello world!");`
  - `});`
- Routing
  - `var express = require('express');`
  - `var app = express();`
  - `app.get('/hello', function(req, res){`
  - `res.send("Hello World!");`
  - `});`
  - `app.post('/hello', function(req, res){`
  - `res.send("You just called the post method at '/hello'!\n");`
  - `});`

# ExpressJS

## ■ Middleware functions

- They are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.
- These functions are used to modify req and res objects for tasks like parsing request bodies, adding response headers, etc.

- `var express = require('express');`
- `var app = express();`
- `app.use('/things', function(req, res, next){`
- `console.log("A request for things`
- `received at " + Date.now());`
- `next();`
- `});`
- `// Route handler that sends the response`
- `app.get('/things', function(req, res){`
- `res.send('Things');`
- `});`
- `app.listen(3000);`





# ExpressJS

---

- One of the most important things about middleware in Express is the order in which they are written/included in your file; the order in which they are executed, given that the route matches also needs to be considered.
  - `var express = require('express');`
  - `var app = express();`
  - `//First middleware before response is sent`
  - `app.use(function(req, res, next){`
  - `console.log("Start");`
  - `next();`
  - `});`
  - `//Route handler`
  - `app.get('/', function(req, res, next){`
  - `res.send("Middle");`
  - `next();`
  - `});`
  - `app.use('/', function(req, res){`
  - `console.log('End');`
  - `});`
  - `app.listen(3000);`



# ExpressJS

---

- Third-Party Middleware Examples:

- body-parser

- This is used to parse the body of requests which have payloads attached to them. To mount body parser, we need to install it using `npm install --save body-parser`.
    - `var bodyParser = require('body-parser');`
    - `//To parse URL encoded data`
    - `app.use(bodyParser.urlencoded({ extended: false }));`
    - `//To parse json data`
    - `app.use(bodyParser.json());`

- cookie-parser

- It parses Cookie header and populate `req.cookies` with an object keyed by cookie names. To mount cookie parser, we need to install it using `npm install --save cookie-parser`.
    - `var cookieParser = require('cookie-parser');`
    - `app.use(cookieParser());`



# Node Package Manager(npm)

---

- npm is the package manager for node.
- The npm Registry is a public collection of packages of open-source code for Node.js, front-end web apps, mobile apps, robots, routers, and countless other needs of the JavaScript community.
- npm allows us to access all these packages and install them locally.
- There are two ways to install a package using npm: globally and locally.
  - Globally – This method is generally used to install development tools and CLI based packages. The installed packages are all put in a single place in your system (exactly where depends on your setup)To install a package globally, use the following code.
    - `npm install -g <package-name>`
  - Locally – This method is generally used to install frameworks and libraries. A locally installed package can be used only within the directory it is installed and they are put in the `node_modules` folder under this directory. To install a package locally, use the same command as above without the `-g` flag.
    - `npm install <package-name>`





# npm Commands

---

- `npm install` or `npm i`
  - It will install all modules listed as dependencies in `package.json`.
- `npm install <folder>`
  - Install the package in the directory as a symlink in the current project.
- `npm init`
  - It is used to create a `package.json` file which helps you manage dependencies that you install.
- `npm list -g --depth 0`
  - List of the packages, which installed globally on your system.
- `npm search`
  - Command to show everything that's available.
- `npm ls`
  - Command to show everything you've installed.
- `npm install -g 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.



# Postman

---

- What is Postman?
  - Postman is currently one of the most popular tools used in API testing.
- Why use Postman?
  - Accessibility - To use Postman, one would just need to log-in to their own accounts making it easy to access files anytime, anywhere as long as a Postman application is installed on the computer.
  - Use of Collections - Postman lets users create collections for their API calls. Each collection can create subfolders and multiple requests. This helps in organizing your test suites.
  - Collaboration - Collections and environments can be imported or exported making it easy to share files. A direct link can also be used to share collections.
  - Creating Environments - Having multiple environments aids in less repetition of tests as one can use the same collection but for a different environment. This is where parameterization will take place which we will discuss in further lessons.
  - Creation of Tests - Test checkpoints such as verifying for successful HTTP response status can be added to each API calls which help ensure test coverage.
  - Automation Testing - Through the use of the Collection Runner or Newman, tests can be run in multiple iterations saving time for repetitive tests.
  - Debugging - Postman console helps to check what data has been retrieved making it easy to debug tests.
  - Continuous Integration - With its ability to support continuous integration, development practices are maintained.



POSTMAN

# Postman

The image displays two screenshots of the Postman application interface. The top screenshot shows an 'Untitled Request' with the URL 'http://localhost:3000/'. The 'Send' button is highlighted, and a blue cloud bubble with the text 'Unsuccessful Request!' is overlaid on the right side. The bottom screenshot shows the same request, but the response is visible. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 58 ms', and 'Size: 1.37 KB'. A blue cloud bubble with the text 'Successful Request!' is overlaid on the left side. The response body is shown in the 'Test Results' tab, displaying HTML code.

GET **Untitled Request** GET <https://jsonplaceholder.typicode.com> + ... No Environment

http://localhost:3000/

GET http://localhost:3000/ Send

Params Authorization Headers Body Pre-request Script Cookies Code

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

**Could not get any response**

There was an error connecting to <http://localhost:3000/>.

**Why this might have happened:**

- **The server couldn't send a response:** Ensure that the backend is working properly
- **Self-signed SSL certificates are being blocked:** Fix this by turning off 'SSL certificate verification' in *Settings > General*
- **Proxy configured incorrectly** Ensure that proxy is configured correctly in *Settings > Proxy*
- **Request timeout:** Change request timeout in *Settings > General*

GET http://localhost:3000/ Send

Params Authorization Headers Cookies Code

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Body Cookies Headers (9) Test Results Status: 200 OK Time: 58 ms Size: 1.37 KB

Pretty Raw Preview HTML

```
1 <!DOCTYPE html>
2 <html>
3 <style>
4 input[type=text], select, #pw {
5 width: 100%;
6 padding: 10px 20px;
```

# Your Task

- Create a simple form in html and css.
- Show the entered values in server side.
- Test your url in Postman

Client Side

First Name

Last Name

Password

Submit

Server Side

```
> lab3@1.0.0 dev D:\SE\Lab3
> nodemon Test.js

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node Test.js`
Middle
Get the entered data!
{ firstname: 'Gehad', lastname: 'Ismail', pw: '123' }
Fname: Gehad
Get the entered data!
{ firstname: 'Gehad', lastname: 'Ismail', pw: '123' }
Fname: Gehad
```

Postman

GET http://localhost:3000/ Send

Params Authorization Headers Body Pre-request Script Tests Cookies Code

| KEY | VALUE | DESCRIPTION | ... |
|-----|-------|-------------|-----|
| Key | Value | Description |     |

body Cookies Headers (9) Test Results Status: 200 OK Time: 58 ms Size: 1.37 KB D

Pretty Raw Preview HTML

```
1 <!DOCTYPE html>
2 <html>
3 <style>
4 input[type=text], select,pw {
5 width: 100%;
6 margin: 10px 0px;
```