NODE JS

Prof. Vinay Joshi and Dr. Sarasvathi V
Department of Computer Science and Engineering

Acknowledgement
The slides are created from various internet resources with valuable contributions from multiple professors.

PES UNIVERSITY

NODE JS

HTTP Module

Department of Computer Science and Engineering

NODE JS

Web Server

PES UNIVERSITY

- A Web Server is a software application which handles HTTP requests sent by the HTTP client, like web browsers, and returns web pages in response to the clients.
- Web servers usually deliver html documents along with images, style sheets, and scripts.
- Most of the web servers support server-side scripts, using scripting languages or redirecting the task to an application server which retrieves data from a database and performs complex logic and then sends a result to the HTTP client through the Web server.
- Apache web server is one of the most commonly used web servers. It is an open source project.

NODE JS

Web Application Architecture

A Web application is usually divided into four layers −

Client − This layer consists of web browsers, mobile browsers or applications which can make HTTP requests to the web server.

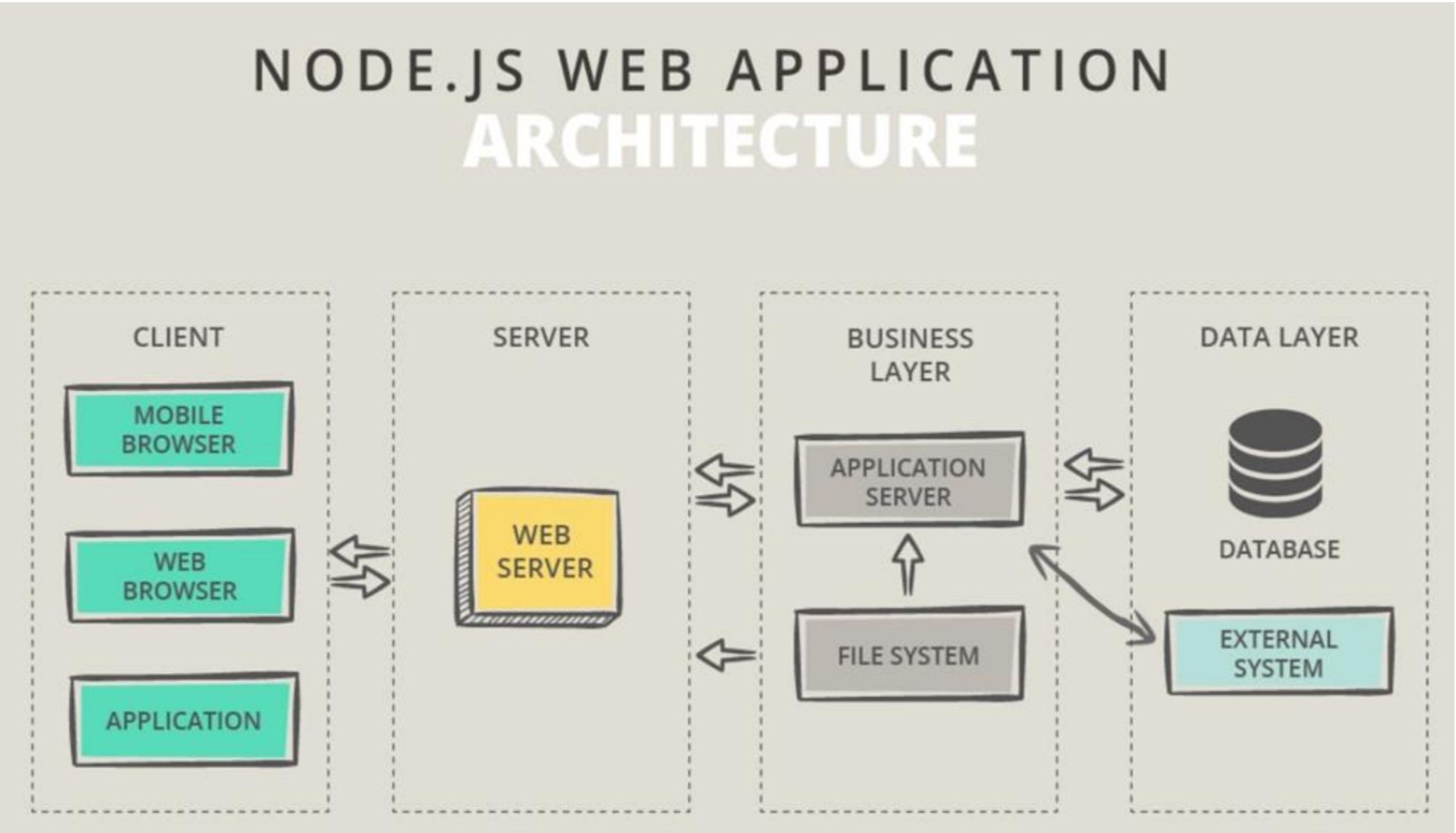
Server − This layer has the Web server which can intercept the requests made by the clients and pass them the response.

Business − This layer contains the application server which is utilized by the web server to do the required processing. This layer interacts with the data layer via the database or some external programs.

Data − This layer contains the databases or any other source of data.

NODE JS

Web Application Architecture



HTTP Module – Web Server using Node

PES UNIVERSITY

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.

HTTP Module – Web Server using Node

The components of a Node.js application.

Import required modules – We use the **require** directive to load Node.js modules.

Create server – A server which will listen to client's requests similar to Apache HTTP Server. Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

Read request and return response – The server created in an earlier step will read the HTTP request made by the client which can be a browser or a console and return the response.

Creating Node.js Application

Step 1 - Import Required Module

We use the **require** directive to load the http module and store the returned HTTP instance into an http variable as follows

```
var http = require('http');
```

PES UNIVERSITY

Step 2 - Create Server

- We use the created http instance and call **http.createServer()** method to create a server instance.
- Then we bind it at port 8088 using the **listen** method associated with the server instance.

```
http.createServer(function (request, response) {
    response.writeHead(200, {'Content-Type': 'text/plain'});
    response.end('Hello PES University\n');
 }).listen(8088);
  console.log('Server running at http://127.0.0.1:8088/')
```

Step 3 - Testing Request & Response

$node app.js

Verify the Output. Server has started.

Server running at http://127.0.0.1:8088/

HTTP Module – Web Server using Node

PES UNIVERSITY

```
var http = require('http'); // 1 - Import Node.js core module

var server = http.createServer(function (req, res) {   // 2 - creating server

    //handle incomming requests here..

});

server.listen(5000); //3 - listen for any incoming requests

console.log('Node.js web server at port 5000 is running..')
```

HTTP Module – Web Server using Node

- In the above example, we import the http module using require() function.
- The http module is a core module of Node.js, so no need to install it using NPM.
- The next step is to call createServer() method of http and specify callback function with request and response parameter.
- Finally, call listen() method of server object which was returned from createServer() method with port number, to start listening to incoming requests on port 5000.
- You can specify any unused port here.

HTTP Module – Web Server using Node

Run the above web server by writing node server.js command in command prompt or terminal window and it will display message as shown below.

C:\> node server.js
Node.js web server at port 5000 is running..

This is how you create a Node.js web server using simple steps.

Now, let's see how to handle HTTP request and send response in Node.js web server.

PES UNIVERSITY

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.

HTTP Module – Web Server using Node

```
var http=require('http');
var server=http.createServer(function(req,res){
res.write('Hello World');
res.end();
}).listen(5000);
console.log('Node.js webserver at port 5000 is running ‘)
```

To run this enter
Localhost:5000 in your browser

HTTP Module – Web Server using Node

```
var http = require('http');

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
```

HTTP Module – Web Server using Node

The function passed into the http.createServer() method, will be executed when someone tries to access the computer on port 8080.

Save the code above in a file called "demo_http.js", and initiate the file:

Initiate demo_http.js:

C:\Users\Your Name>node demo_http.js

If you have followed the same steps on your computer, you will see the same result as the example: http://localhost:8080

PES UNIVERSITY

Add an HTTP Header

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:

Example

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

HTTP Module – Web Server using Node

The first argument of the res.writeHead() method is the status code,
200 means that all is OK, the second argument is an object containing the response headers.

HTTP Module – Web Server using Node

Read the Query String

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).

This object has a property called "url" which holds the part of the url that comes after the domain name:

demo_http_url.js

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);   //requesting URL
  res.end();
}).listen(8080);
```

http://localhost:8080/summer

NODE JS

URL Module

The URL module splits up a web address into readable parts.

To include the URL module, use the require() method:

```
var url = require('url');
```

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

Node Examples > JS app.js > ...

```
1   var url = require('url');
2   var adr = 'http://localhost:8080/pesu.htm?year=2020&month=September';
3   var q = url.parse(adr, true);
4
5   console.log(q.host); //returns 'localhost:8080'
6   console.log(q.pathname); //returns '/pesu.htm'
7   console.log(q.search); //returns '?year=2020&month=September'
8
9   var qdata = q.query; //returns an object: { year: 2020, month: 'september' }
10  console.log(qdata.month); //returns 'september'
```

NODE JS

Web Client Using Node

A web client can be created using **http** module. A Screenshot of the example is below

```
var http = require('http');
var options = {
   host: 'localhost',
   port: '8081',
   path: '/index.htm'
};
var callback = function(response) {
   var body = '';
   response.on('data', function(data) {
      body += data;
   });

   response.on('end', function() {
      console.log(body);
   });
}
var req = http.request(options, callback);
req.end();
```

HTTP request using node-fetch in nodejs

node-fetch package

node-fetch is a lightweight module that enables us to use the fetch() function in NodeJS, with very similar functionality as window.fetch() in native JavaScript.

To use the module in code, use:

```
const fetch = require('node-fetch’);
```

Followed by

```
fetch(url[, options]);
```

The url parameter is simply the direct URL to the resource we wish to fetch. It has to be an absolute URL or the function will throw an error.

HTTP request using node-fetch in nodejs

The function returns a Response object that contains useful functions and information about the HTTP response, such as:

- text() - returns the response body as a string
- json() - parses the response body into a JSON object, and throws an error if the body can't be parsed
- status and statusText - contain information about the HTTP status code
- ok - equals true if status is a 2xx status code (a successful request)
- headers - an object containing response headers, a specific header can be accessed using the get() function.

Sending GET Requests Using node-fetch

```
const fetch = require('node-fetch’);
 fetch('https://google.com’)
 .then(res => res.text())
.then(text => console.log(text))
```

In the code above, we're loading the node-fetch module and then fetching the Google home page. The only parameter we've added to the fetch() function is the URL of the server we're making an HTTP request to. Because node-fetch is promise-based, we're chaining a couple of .then() functions to help us manage the response and data from our request.

NODE JS

HTTP request using node-fetch in nodejs- Client Program

PES UNIVERSITY

```
var fetch = require('node-fetch');
//import fetch from 'cross-fetch';
fetch('http://localhost:8080/sample.txt',{
    method:'POST',
    headers:{'content-type':'application/json'},
    body:JSON.stringify({"name":"Sujay1","srn":"12347"})
})
.then((res)=>res.text())
.then((res)=>console.log(res))
```

NODE JS

HTTP request using node-fetch in nodejs- Server Program

PES UNIVERSITY

```
if(request.method=='POST'){
    var myurl = url.parse(request.url)
    var pathname = myurl.pathname; // includes the '/'
    let body=[];
    request.on('data',(chunk)=>{
        body.push(chunk);
        console.log(chunk.toString())
    })
    .on('end',()=>{
        body=Buffer.concat(body).toString()
        console.log(body)
        fs.writeFile(pathname.substr(1),body,(err,res)=>{
        response.writeHead(200,{'Content-type':'text/plain'});
        response.end("Message Saved");
        })
    })
}
```

THANK YOU

Vinay Joshi and Dr.Sarasvathi V
Department of Computer Science and Engineering