UNIT-4

Introduction to Node.js

4.1 Back-end Development

Back-end development means working on server-side software, which focuses on everything you can't see on a website.

Back-end developers ensure the website performs correctly, focusing on databases, back-end logic, application programming interface (APIs), architecture, and servers.

They use code that helps browsers communicate with databases, store, understand, and delete data.

Back-end developers must be familiar with many kinds of tools and frameworks, including languages such as Python, Java, and Ruby.

They make sure the back-end performs quickly and responsively to front-end user requests. Because performance is an important aspect, when you are using lists you need to make sure they are designed for optimal performance.

Back-end developer tasks and responsibilities:

Back-end developers are required to have technical expertise, analytical thinking, and excellent collaboration skills. As a back-end web developer, you should be able to work independently to design the web infrastructure.

Build and maintain websites: A back-end developer's main responsibility is to use various tools, frameworks, and languages to determine how best to develop intuitive, user-friendly prototypes and turn them into websites. This requires an understanding of cross-platform functionality and compatibility.

Write high-quality code: To produce sustainable web applications, developers must write clean and easily maintainable code.

Perform quality assurance (QA) testing: Create and oversee testing schedules to optimize user interface and experience, ensuring optimal display on various browsers and devices.

Assess efficiency and speed: Once a website is up and running, and during updates and edits, developers need to assess its performance and scalability, adjusting code as necessary.

Troubleshoot and debug: Be able to troubleshoot issues and resolve them, while communicating them to project managers, stakeholders, and QA teams.

Train and support: Maintain workflows with client teams to ensure ongoing support, along with leading training and mentorship for junior developers.

What tools do back-end developers use?

Web developers use a variety of tools to develop, test, and maintain web applications. Some common tools for back-end developers include:

Programming languages:

- Python
- PHP
- JavaScript

- Ruby
- Java
- C#

Frameworks:

- Laravel
- Django
- Spring
- Ruby on Rails
- Meteor
- Node.js

Databases:

- MongoDB
- MySQL
- Oracle

Back-end developer technical skills:

As a back-end developer, there are certain technical skills you will need to learn to navigate developing the back-end of the web or mobile application.

Programming languages: Any back-end developer needs to be well-versed in back-end programming languages such as Python, Java, and PHP. These make the website function when used alongside databases, frameworks, and servers. Python is one of the most popular programming languages because it is compatible with artificial intelligence (AI) and machine learning, and works well for writing clear and logical code. Basic knowledge of front-end languages HTML, CSS, and JavaScript is a bonus.

Frameworks: Frameworks are the libraries of back-end programming languages that help to build the server configuration. They tend to be linked with programming languages, so if you are familiar with Python, you'll also know Flask, Django, or another Python-based framework, and so on.

Databases and servers: You'll need to understand how to stack and recover data from databases, as back-end programming controls access to this information, including storage and recovery. MongoDB and MySQL are popular database programs. The database stores and organizes the client's data so that it can be easily arranged and recovered, just like you might use cloud storage for your photos. This database then runs on a server that provides data upon request.

Application Program Interface (API): An API is a series of definitions and rules for developing application software. In addition to internet browser websites, companies often want a mobile app for iOS or Android. Knowledge of application-building languages like JavaScript will expand your job opportunities.

Accessibility and security clearance: You should develop knowledge of network protocols and web security. Knowing how to secure databases and servers will be critical to your success as a back-end developer.

4.2 Server-side JavaScript

4.2.1 Client-side VS Server-side JavaScript

Client-side JavaScript:

Client-side JavaScript refers to JavaScript code that is executed on the user's web browser. It's used to enhance the functionality and interactivity of web pages without requiring communication with the server. Common uses of client-side JavaScript include form validation, dynamic content updates, and user interface enhancements.

```
<!DOCTYPE html>
<html>
<head>
   <title>Client-side JavaScript Example</title>
   <script>
      function greet() {
         var name = prompt("Enter your name:");
         alert("Hello, " + name + "!");
      }
   </script>
</head>
<body>
   <button onclick="greet()">Click me</button>
</body>
</html>
```

In this example, when the "Click me" button is clicked, the client-side JavaScript code prompts the user for their name and displays a greeting using an alert box, all within the user's browser.

Server-side JavaScript:

Server-side JavaScript refers to JavaScript code that is executed on the server, usually in response to HTTP requests. It's used to generate dynamic content, handle data processing, and interact with databases. Server-side JavaScript is often used in conjunction with server-side frameworks and technologies like Node.js to build web applications and APIs.

Example (Node.js):

```
const http = require('http');
http.createServer(function (req, res) {
   res.writeHead(200, {'Content-Type': 'text/plain'});
   res.end('Hello from the server!');
}).listen(8080);
```

In this Node.js example, server-side JavaScript code creates a simple HTTP server that responds with "Hello from the server!" when a request is made to the specified port.

4.2.2 Server-side JavaScript

Node.js is a powerful runtime environment for executing JavaScript code outside of a web browser. It brings the JavaScript language to the server-side, enabling developers to build scalable, high-performance, and event-driven applications.

Node.js allows developers to use JavaScript both on the client-side and the server-side, providing a unified language and ecosystem. This eliminates the need for context switching and enables code reuse between the front-end and back-end. This results in improved productivity and reduced development time.

Node.js has a vast and active ecosystem of modules and libraries available through the Node Package Manager (npm). This rich ecosystem offers ready-to-use tools and packages for various functionalities, such as web frameworks, database connectors, authentication, and testing frameworks.

Developers can leverage these modules to accelerate development and enhance application functionality.

Given all that, Node.js is particularly well-suited for building:

- Web applications
- Scalable APIs
- Real-time applications requiring instant data updates and bidirectional communication like chat applications and multiplayer games
- Streaming applications like audio or video processing or real-time analytics
- Single-page applications
- Internet of Things deployments

4.3 Introduction of Node.js

Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!

Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser.

When Node.js performs an I/O operation, like reading from the network, accessing a database or the file system, instead of blocking the thread and wasting CPU cycles waiting, Node.js will resume the operations when the response comes back.

This allows Node.js to handle thousands of concurrent connections with a single server without introducing the burden of managing thread concurrency, which could be a significant source of bugs.

Node.js has a unique advantage because millions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language.

In Node.js the new ECMAScript standards can be used without problems, as you don't have to wait for all your users to update their browsers - you are in charge of deciding which ECMAScript version to use by changing the Node.js version, and you can also enable specific experimental features by running Node.js with flags.

Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

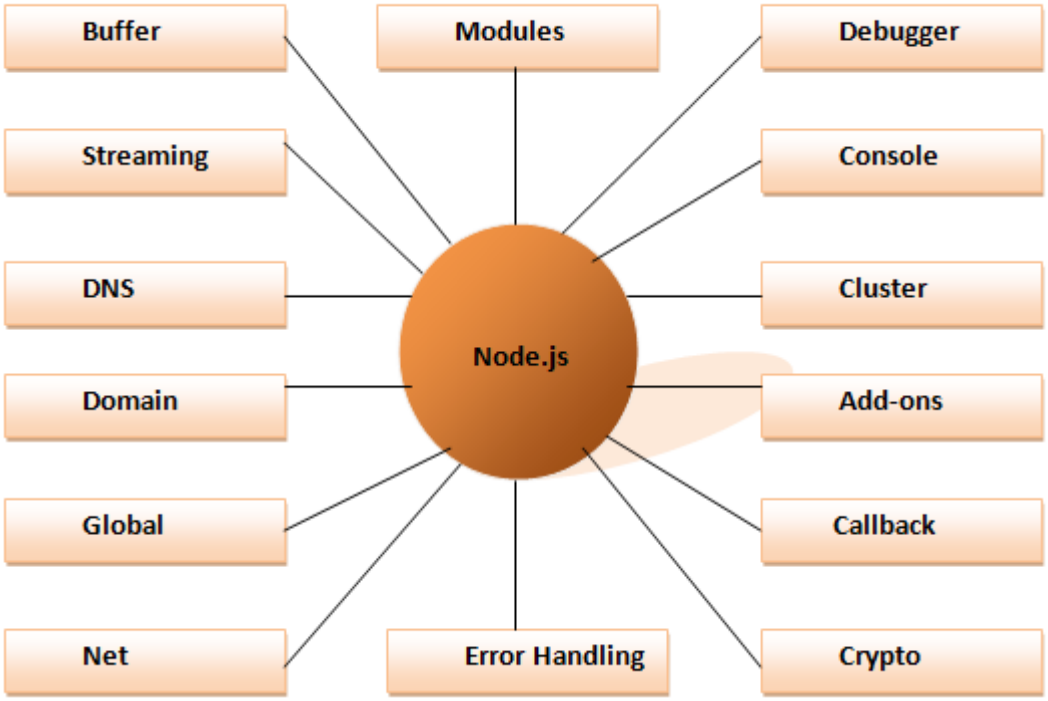
Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.

Node.js = Runtime Environment + JavaScript Library.

Following is a list of some important features of Node.js that makes it the first choice of software architects:

- Extremely fast: Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
- Single threaded: Node.js follows a single threaded model with event looping.
- Highly Scalable: Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
- No buffering: Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
- Open source: Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.



4.4 Node.js Modules

What is a Module in Node.js?

Consider modules to be the same as JavaScript libraries.

A set of functions you want to include in your application.

Built-in Modules

Node.js has a set of built-in modules which you can use without any further installation.

Include Modules

To include a module, use the **require()** function with the name of the module:

var http = require('http');

Now your application has access to the HTTP module, and is able to create a server:

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

4.4.1 Create Your Own Modules

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

Example:

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 Own Module

Now you can include and use the module in any of your Node.js files.

Example:

Use the module **"myfirstmodule"** in a Node.js file:

```
var http = require('http');
var dt = require('./myfirstmodule');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("The date and time are currently: " + dt.myDateTime());
  res.end();
}).listen(8080);
```

Notice that we use **./** to locate the module, that means that the module is located in the same folder as the Node.js file.

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

Initiate demo_module.js:

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

4.4.2 Node.js HTTP Module

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

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

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

Node.js as a 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:

Example:

```
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
```

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

4.5 Creating a web server

An Example Node.js Application

The most common example Hello World of Node.js is a web server:

```
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

To run this snippet, save it as a **server.js** file and **run node server.js** in your terminal.

This code first includes the Node.js http module.

Node.js has a fantastic standard library, including first-class support for networking.

The **createServer()** method of http creates a new HTTP server and returns it.

The server is set to listen on the specified port and host name. When the server is ready, the callback function is called, in this case informing us that the server is running.

Whenever a new request is received, the request event is called, providing two objects: a request (**an http.IncomingMessage object**) and a response (**an http.ServerResponse object**).

Those 2 objects are essential to handle the HTTP call.

The first provides the request details. In this simple example, this is not used, but you could access the request headers and request data.

The second is used to return data to the caller.

In this case with:

res.statusCode = 200; *//we set the statusCode property to 200, to indicate a successful response.*
res.setHeader('Content-Type', 'text/plain'); *//We set the Content-Type header*

and we close the response, adding the content as an argument to end():
res.end('Hello World\n');

4.6 Node.js MySQL

Node.js can be used in database applications. One of the most popular databases is MySQL.

MySQL Database:

To be able to experiment with the code examples, you should have MySQL installed on your computer.

You can download a free MySQL database at **https://www.mysql.com/downloads/.**

Install MySQL Driver

Once you have MySQL up and running on your computer, you can access it by using Node.js.

To access a MySQL database with Node.js, you need a MySQL driver. You can use the "mysql" module, downloaded from NPM.

To download and install the "mysql" module, open the Command Terminal and execute the following:

C:\Users\Your Name>npm install mysql

Now you have downloaded and installed a mysql database driver.

Node.js can use this module to manipulate the MySQL database:

var mysql = require('mysql');

- Create Connection
- Start by creating a connection to the database.
- Use the username and password from your MySQL database.
- demo_db_connection.js

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

Save the code above in a file called **"demo_db_connection.js"** and run the file:

Run "demo_db_connection.js"

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

Which will give you this result:

Output: Connected!

Now you can start querying the database using SQL statements.

Query a Database:

Use SQL statements to read from (or write to) a MySQL database. This is also called "to query" the database.

The connection object created in the example above, has a method for querying the database:

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Result: " + result);
  });
});
```

The query method takes an sql statements as a parameter and returns the result.

4.6.1 Creating a Database:

To create a database in MySQL, use the **"CREATE DATABASE"** statement:

Example:

Create a database named "mydb":

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb", function (err, result) {
    if (err) throw err;
    console.log("Database created");
  });
});
```

Save the code above in a file called **"demo_create_db.js"** and run the file:

Run "demo_create_db.js"

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

Which will give you this result:

Output: Connected!

Database created

4.6.2 Creating a Table:

Creating a Table

To create a table in MySQL, use the **"CREATE TABLE"** statement.

Make sure you define the name of the database when you create the connection:

Example:

Create a table named **"customers"**:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

Save the code above in a file called **"demo_create_table.js"** and run the file:

Run "demo_create_table.js"

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

Which will give you this result:

Output: Connected!

Table created

4.6.3 Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a column as **"INT AUTO_INCREMENT PRIMARY KEY"** which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Example:

Create primary key when creating the table:

```
var mysql = require('mysql');
var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name
VARCHAR(255), address VARCHAR(255))";
  con.query(sql, function (err, result) {
    if (err) throw err;
    console.log("Table created");
  });
});
```

Output: Connected!

Table created

4.7 Node.js Events

Node.js is perfect for event-driven applications.

Events in Node.js:

Every action on a computer is an event. Like when a connection is made or a file is opened.

Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

Example:

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
  console.log('The file is open');
});
```

Events Module

Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.

To include the built-in Events module use the require() method. In addition, all event properties and methods are an instance of an EventEmitter object. To be able to access these properties and methods, create an EventEmitter object:

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
```

The EventEmitter Object

You can assign event handlers to your own events with the EventEmitter object.

In the example below we have created a function that will be executed when a "scream" event is fired.

To fire an event, use the emit() method.

Example:

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

//Create an event handler:
var myEventHandler = function () {
  console.log('I hear a scream!');
}

//Assign the event handler to an event:
eventEmitter.on('scream', myEventHandler);

//Fire the 'scream' event:
eventEmitter.emit('scream');
```