**Index**

1. Node Introduction
2. What is Node?
3. Why Node?
4. Node installation
5. Sample program in CLI
6. JavaScript Concepts
7. Callbacks
8. Promise
9. Async - Await
10. NPM basics
11. Modules in Node
    1. Built-in modules
    2. External modules
    3. User modules
12. Debugging
13. Express Framework
    1. Introduction
    2. Installation
    3. Sample Application
    4. Routing
    5. Request Object
    6. Response Object
    7. Middleware
    8. Temple engines
14. Sails Framework
    1. Introduction
    2. Installation
    3. App Structure
15. Deployment

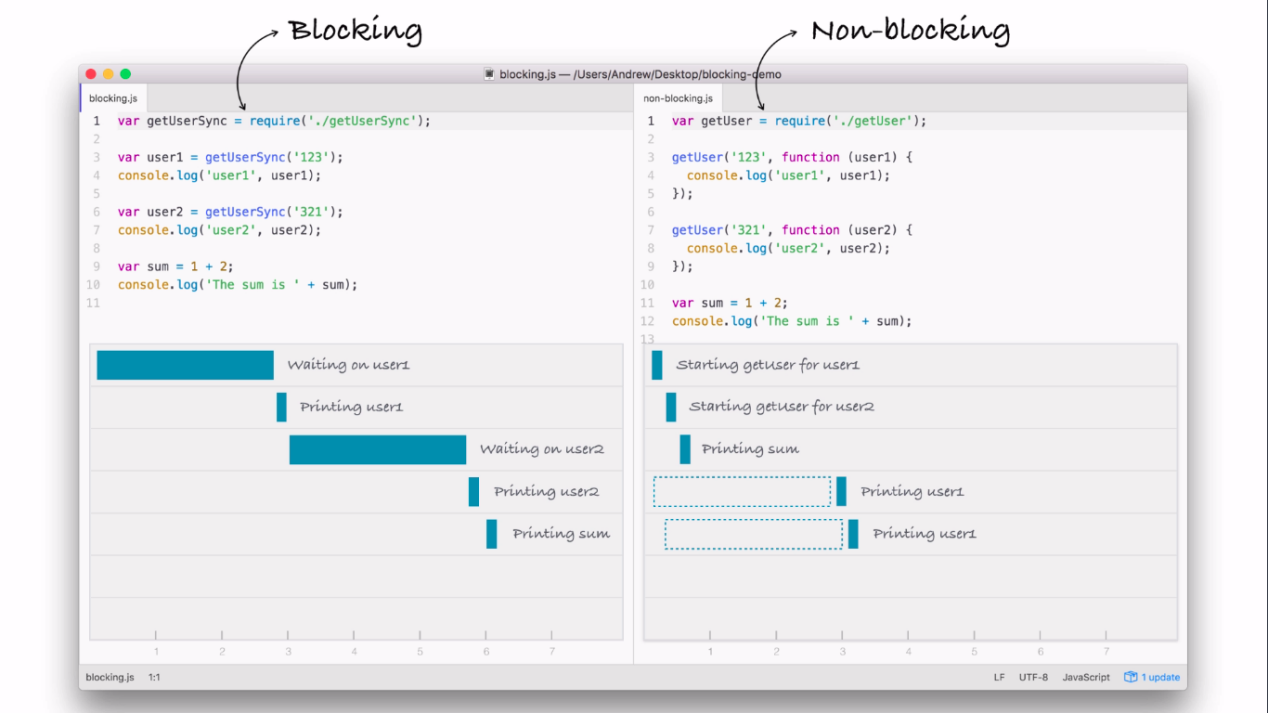
**Node Introduction**

**What is Node?**

* Node is a JavaScript runtime build on Chrome’s V8 JavaScript engine.
* Initially we were able to run JavaScript code only within the browser, but after Node came, it enabled us to run JavaScript code on the machine as a standalone process.
* In Node and Browser both, JavaScript code runs on V8 JavaScript engine, which converts JavaScript code into Machine code which will be directly interpreted by our machine.
* V8 JavaScript engine is written in C++, so if you want to extend Node language then you can write C++ extensions.
* Browser vs Node
* Window = Global
* Document = Process

**Why Node?**

* Node uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.
* Let’s understand terms event-driven and non-blocking from below scenario:



* Node’s package ecosystem, NPM, is the largest ecosystem of open source libraries in the world.

**Node Installation**

Checkout the official download [link](https://nodejs.org/en/download/) and installation page for installing Node in your system.

**Sample program in CLI**

See [Example-1](file:///E:\\Dipak\\Node%20Training\\Example-1).

**JavaScript Concepts**

* **Callbacks**

In JavaScript every function is an object, which means it can be used like any other object (String, Number etc.), so we can pass a function as an argument to another function.

See [Example-2](file:///E:\\Dipak\\Node%20Training\\Example-2).

* **Promise**

A Promise represents the eventual completion of an asynchronous operation and its resulting value.

Promise can have one if the following three states:

1. Pending - Asynchronous operation has not completed yet
2. Resolved -  Operation has completed and the Promise has a value
3. Rejected - Operation has completed with an error or failed.

See [Example-3](file:///E:\\Dipak\\Node%20Training\\Example-3).

* **Async - Await**

It is a special syntax to work with promises in a more comfortable fashion. It’s surprisingly easy to understand and use.

**Async** before a function means one simple thing, a function always returns a promise.If the code has return <non-promise> in it, then JavaScript automatically wraps it into a resolved promise with that value.

**Await** makes JavaScript wait until that promise settles and returns its result

See [Example-3](file:///E:\\Dipak\\Node%20Training\\Example-3).

**NPM basics**

NPM is the world’s largest software registry. Open source developers from every continent use NPM to share and borrow packages, and many organizations use NPM to manage private development as well.

Checkout link <https://www.youtube.com/watch?v=x03fjb2VlGY> for NPM introduction.

NPM consists of three distinct components:

1. The Website:

Use the website to discover packages, set up profiles, and manage other aspects of your NPM experience. For example, you can set up Orgs (organizations) to manage access to public or private packages.

1. The CLI:

CLI runs from a terminal and is how most developers interact with NPM.

1. The Registry:

Registry is a large public database of JavaScript software and the meta-information surrounding it.

Basic commands which we will be using mostly:

1. **npm init**

Create npm package locally.

1. **npm install**

Install all packages listed in package.json file.

1. **npm install <package\_name>**

Add package to project as dependency.

1. **npm remove <package\_name>**

Remove package from project dependency.

**Node modules**

Node modules are kind of JavaScript libraries. It is a set of functions which you want to include in your application.

We can divide modules into three categories:

1. **Built-in modules**

These are the modules which Node provides as part of its core library. Let’s see few of them as follows:

* + 1. **HTTP** – Provides functionalities to transfer data using HTTP protocol.
    2. **File System** – Provides functionalities to communicate with File System on computer.
    3. **Events** – Provides functionalities related to event creation and subscriptions.
    4. **OS** – OS modules provides number of operating system related utility methods.
    5. **Net** – Provides an asynchronous network API for creating stream-based TCP or ICP servers and clients.
    6. **Path** – Provides utility functions for working with file and directory paths.
    7. **Process** – Provides control and information about current Node process.
    8. **URL** – Provides utility functions for URL resolution and parsing.
    9. **Utilities** – It is designed to support Node’s internal modules as well as to provide useful functions to application and module developers as well.
    10. **Crypto** – Provides cryptographic functionalities that includes a set of wrappers for OpenSSL’s hash, HMAC, cipher, decipher, sign and verify functions.
    11. **Buffer** – It provides various functions to handle and manipulate binary data/streams.
    12. Others, checkout link <https://nodejs.org/dist/latest-v10.x/docs/api/> for all module reference.

See Example-4.

1. **External modules**

These are the external modules which are provided by all community developers. We can use NPM to download and use those. Few examples are as follows:

* + 1. MYSQL – To communicate with MySQL server.
    2. MONGODB – To communicate with MongoDB server.
    3. Axios – Promise based HTTP client.
    4. Formidable – To manage file upload.
    5. Nodemailer – To send mails
    6. Request – This provides the simplest way possible to make http calls. It supports https and follows redirects by default.
    7. Socket.io – Provides real-time bidirectional event-based communication.

And thousands more modules on NPM registry…

See Example-5, 7, 8.

1. **User modules**

User modules are the modules which we build during application development. We can require those modules into various other modules to reuse the functionalities provided by it.

See Example-6.

**Debugging**

Debugging is very essential for any programming language to debug any logical issues and work flows.

In Node we have two ways to debug node application:

1. **Command Line Interface:**
   1. To debug any file of Node project in CLI, we need to fire the following command:

**node inspect <filename.js>**

This will start debugging.

* 1. Debugger will first stop at the starting of the code. We can use **“n”** to go to next line and **“c”** to go to next **“debugger”** point or end of program.
  2. At any line if we want to inspect variables, we can use command **“repl”** to go into inspect mode, and we can check various variable’s values. To exit from “repl” mode we can use **“cntl+c”** command.
  3. To exit from debugging mode, we can use **“cntl+c”** command.

1. **Chrome developer tools**
   1. To debug any file of Node project in chrome developer tools, we need to fire the following command:

**node –inspect-brk <filename.js>**

* 1. Open chrome browser, and hit URL : **chrome://inspect .** we will able to see our target under “Remote target” section.
  2. Click on **“Open dedicated DevTools for Node”.**
  3. Now we will able to see the file which we have debugged.
  4. Now we can debug file using **next**, **continue** buttons, and using **breakpoints.**

**Express Framework**

**Introduction**

* Express is a fast, unopinionated, minimalist web framework for Node.js.
* It is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
* With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.
* Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

**Installation**

1. Create a directory to hold your application and make it your working directory.

**mkdir exressapp**

**cd exressapp**

1. Use the npm init command to create a package.json file for your application.

**npm init**

1. This command prompts you for a number of things, such as the name and version of your application. For now, you can simply hit RETURN to accept the defaults for most of them, with the following exception:

**entry point: (index.js)**

**Enter app.js**

1. Now install Express in the “exressapp” directory and save it in the dependencies list.

**npm install express --save**

**Sample Application**

See Example-10.

**Routing**

* Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
* Each route can have one or more handler functions, which are executed when the route is matched.
* Route definition takes the following structure:

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

Where,

* app is an instance of express.
* METHOD is an HTTP request method, in lowercase.
* PATH is a path on the server.
* HANDLER is the function executed when the route is matched.

See Example-11.

**Request**

The “**req**” object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, and so on.

* req.hostname

Contains the hostname derived from the Host HTTP header.

* req.ip

Contains the remote IP address of the request.

* req.cookies

When using **cookie-parser** middleware, this property is an object that contains cookies sent by the request. If the request contains no cookies, it defaults to {}.

* req.params

This property is an object containing properties mapped to the named route “parameters”. For example, if you have the route /user/:name, then the “name” property is available as req.params.name. This object defaults to {}.

* req.body

Contains key-value pairs of data submitted in the request body. By default, it is undefined, and is populated when you use body-parsing middleware such as body-parser and multer.

* req.query

This property is an object containing a property for each query string parameter in the route. If there is no query string, it is the empty object, {}.

* req.path

Contains the path part of the request URL.

* req.protocol

Contains the request protocol string: either http or (for TLS requests) https.

See Example-12 for each of the above options.

**Response**

The res object represents the HTTP response that an Express app sends when it gets an HTTP request.

* res.set(field [, value])

Sets the response’s HTTP header field to value. To set multiple fields at once, pass an object as the parameter.

* res.append(field [, value])

Appends the specified value to the HTTP response header field. If the header is not already set, it creates the header with the specified value. The value parameter can be a string or an array.

* res.cookie(name, value [, options])

Sets cookie name to value. The value parameter may be a string or object converted to JSON.

* res.clearCookie(name [, options])

Clears the cookie specified by name.

* res.download(path [, filename] [, options] [, fn])

Transfers the file at path as an “attachment”. Typically, browsers will prompt the user for download.

* res.redirect([status,] path)

Redirects to the URL derived from the specified path, with specified status, a positive integer that corresponds to an HTTP status code. If not specified, status defaults to “302 “Found”.

* res.status(code)

Sets the HTTP status for the response.

* res.send([body])

Sends the HTTP response. The body parameter can be a Buffer object, a String, an object, or an Array.

* res.json([body])

Sends a JSON response. This method sends a response (with the correct content-type) that is the parameter converted to a JSON string using JSON.stringify(). The parameter can be any JSON type, including object, array, string, Boolean, number, or null, and you can also use it to convert other values to JSON.

See Example-13.

**Middleware**

Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the application’s request-response cycle. The next function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.

Middleware functions can perform the following tasks:

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware in the stack.

If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

See Example-14.

**Template Engines**

A template engine enables you to use static template files in your application. At runtime, the template engine replaces variables in a template file with actual values and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page.

Some popular template engines that work with Express are Pug, Mustache, and EJS.

**Sails Framework**

**Introduction**

Sails framework is one of the popular framework of Node.js, which is build on top of Express framework. It provides a very good architecture and many useful libraries which are ready to use. It allows us flexibility to configure each and every functionality provided by it to meet our project requirements.

**Installation**

* Install Sails framework globally in our machine using the following command:

**npm install sails -g**

* Go to the directory where you want to create new project and run the following command:

**sails new sails-project**

* It will prompt you to choose your project template:

1. **Web App** : Extensible project with auth, login, & password recovery
2. **Empty App** : An empty Sails app, yours to configure

* Choose whichever option you like. It will create the project and list all of your project’s dependency.
* Go to the project directory by,

**cd sails-project**

* Start your application by,

**sails lift**

* Now you can visit **localhost:1337** to see application.

**App Structure**

Visit this [URL](https://sailsjs.com/documentation/anatomy) to understand project structure and configuration options.

**Deployment**

There are multiple options available to deploy our node application based on our requirement. We can deploy it on cloud using various technologies like Heroku, Microsoft Azure, Google Cloud Platform, Amazon Web Services and many more. We can also deploy it on local servers using tools like PM2. Deployment process varies depending on the hosting environment, hence we need to check official documents of respective hosting environment to know the exact process.

We will see an example of deploying node to local server using PM2. We can visit [PM2 Official Site](http://pm2.keymetrics.io) to understand various advanced functionality provided by PM2.