FRONTEND ADVANCED

NODEJS

SERVER SIDE SCRIPTING

TRAINING AGENDA

NodeJS Overview	Installing Node	REPL
Module System	NPM Commands	Creating Pro-style folder structure

NODEJS : INTRODUCTION

Not a programming language

V8 engine used in chrome browser as well

"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 works on event model same as JavaScript

Can create app which can work over web

Working with Async code makes node app faster

Great for creating apps which require huge amount of data transfer In real time

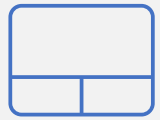
Node Runtime Environment runs JavaScript on Server

NODEJS : FEATURES

EXTREMELY FAST	I/O IS ASYNCHRONOUS	EVENT DRIVEN
SINGLE THREADED	HIGHLY SCALABLE	OPEN SOURCE

NodeJS Process Model

Node.js runs in a single process and the application code runs in a single thread.



All the user requests to web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a request.

An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes.

Internally, Node.js uses libuv for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

NODE.JS FRAMEWORKS AND TOOLS

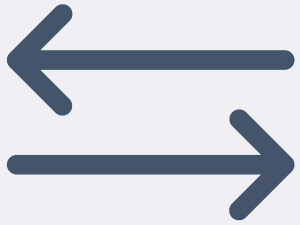
Node.js is a low-level platform. In order to make things easy and exciting for developers, thousands of libraries were built upon Node.js by the community

Framework / Tools	Description
Express	One of the most simple yet powerful ways to create a web server
Gatsby	A React-based, GraphQL powered, static site generator with a very rich ecosystem of plugins and starters
Hapi	A rich framework for building applications and services that enables developers to focus on writing reusable application logic instead of spending time building infrastructure
Loopback.io	Makes it easy to build modern applications that require complex integrations
Meteor	Integrates with frontend libs React, Vue, and Angular. Can be used to create mobile apps as well
Micro	It provides a very lightweight server to create asynchronous HTTP microservices
Nx	A toolkit for full-stack monorepo development using NestJS, Express, React, Angular, and more
Socket.io	A real-time communication engine to build network applications

DIFFERENCES BETWEEN NODE.JS AND THE BROWSER

BROWSER – CLIENT SIDE SCRIPT	NODEJS – SERVER SIDE SCRIPT
Most of the time we interact with the DOM or other Web Platform APIs like Cookies	Do not exist in Node.js. You don't have the document, window and all the other objects that are provided by the browser
NodeJS modules does not exist on browser	Node.js provides nice APIs through its modules, like the filesystem access functionality
Need to use Babel to transform your code to be ES5-compatible before shipping it to the browser	You can write all the modern ES6-7-8-9 JavaScript that your Node.js version supports
Latest browsers starting implementation of ES6 modules standard	Node.js supports both the CommonJS and ES module systems (since Node.js v12)

NODEJS : MODULE SYSTEM



Use of imports/exports for importing and exporting the modules in application.

Following are a few salient points of the module system:

Each file is its own module.

Each file has access to the current module definition using the module variable.

The export of the current module is determined by the module.exports variable.

To import a module, use the globally available require function.

NODEJS : MODULE SYSTEM

- If something is a core module, return it.
- If something is a relative path (starts with ‘./’ , ‘../’) return that file OR folder.
- If not, look for node_modules/filename or node_modules/foldername each level up until you find a file OR folder that matches something.
- If it matched a file name, return it. If it matched a folder name and it has package.json with main, return that file.
- If it matched a folder name and it has an index file, return it.

NODEJS : REQUIRE FUNCTION

The Node.js require function is the main way of importing a module.

The require function blocks further code execution until the module has been loaded.

Call require() based on some condition and therefore load the module on-demand

After the first time a require call is made to a particular file, the module.exports is cached.

Module allows you to share in-memory objects between modules.

Treats module as an object factory

NODEJS : BUILT-IN GLOBAL VARIABLES

Console

The console plays an important part in quickly showing what is happening in your application when you need to debug it.

Timers

setTimeout only executes the callback function once after the specified duration. But setInterval calls the callback repeatedly after every passing of the specified duration.

__filename and __dirname

These variables are available in each file and give you the full path to the file and directory for the current module.

Process

Use the process object to access the command line arguments. Used to put the callback into the next cycle of the Node.js event loop.

NODE PACKAGE MANAGER

NPM is the eco-system to manage the project dependencies

Commands	Description
npm install	Install the packages/ dependencies
npm uninstall	Uninstall the packages/dependencies
npm config get/set	Get /set the npm eco-system
npm update	Update the project dependencies
npm ls	List down the dependencies
npm search	Search the listed package on npm registry
npm init	Generates package.json file in local project directory
npm outdated	List down the outdated package

PACKAGE.JSON FILE

The package.json file is kind of a manifest for your project

Central repository of configuration for tools

Manages the dependencies of your project

Properties in package.json file

version indicates the current version

name sets the application/package name

description is a brief description of the app/package

main sets the entry point for the application

private if set to true prevents the app/package to be accidentally published on npm

scripts defines a set of node scripts you can run

dependencies sets a list of npm packages installed as dependencies

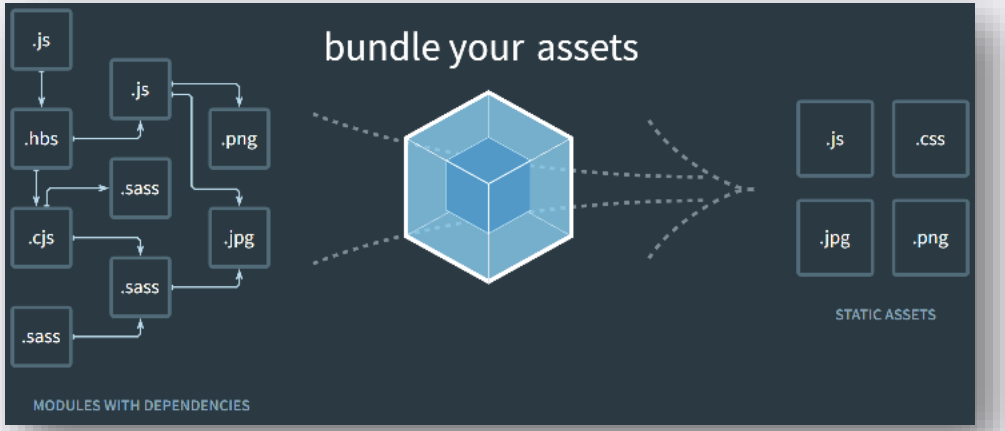
devDependencies sets a list of npm packages installed as development dependencies

FOLDER STRUCTURE

WEBPACK AND MORE

WEBPACK : INTRODUCTION

Webpack is a *static module bundler* for modern JavaScript applications



Webpack builds a dependency graph from one or more *entry points* and then combines every module your project needs into one or more *bundles*

WEBPACK : CORE CONCEPTS

Entry	An **entry point** indicates which module webpack should use to begin building out its internal dependency graph
Output	The **output** property tells webpack where to emit the *bundles* it creates and how to name these files
Loaders	**Loaders** allow webpack to process other types of files and convert them into valid modules that can be consumed by your application
Plugins	**Plugins** can be leveraged to perform a wider range of tasks like bundle optimization, asset management and injection of environment variables
Mode	By setting the **mode** parameter to either development, production or none, you can enable webpack's built-in optimizations that correspond to each environment. The default value is production

REFERENCES

READING MATERIAL

- https://nodejs.org/en/docs
- https://nodejs.dev
- https://webpack.js.org/concepts

VIDEO LINKS

- https://www.youtube.com/watch?v=zb3Qk8SG5Ms&list=PL4cUxeGkcC9jsz4LDYc6kv3ymONOKxwBU
- https://www.youtube.com/watch?v=Oe421EPjeBE&t=4708s
- https://www.youtube.com/watch?v=MpGLUVbqoYQ&t=4s