**NODE JS Cheatsheet**

What is Nodejs:-

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser, which was designed to build scalable network applications.

– Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)

– Node.js uses JavaScript on the server

– Node. js is not a programming language. Rather, it's a runtime environment that's used to run JavaScript outside the browser.

– Node. js can be used on the frontend as well as the backend.

Why Node.js:-

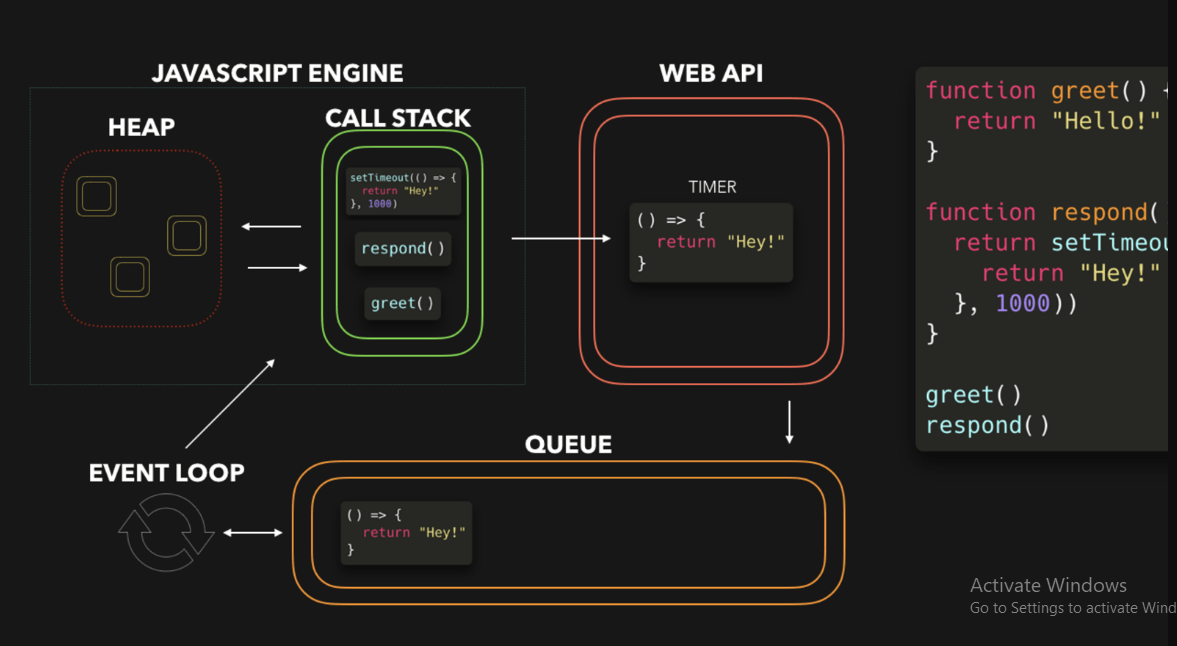
Node. js is primarily used for non-blocking, event-driven servers, due to its single-threaded nature. It's used for traditional web sites and back-end API services, but was designed with real-time, push-based architectures in mind.

Non blocking :-

It refers to the program that does not block the execution of further operations. Non-Blocking methods are executed asynchronously. Asynchronously means that the program may not necessarily execute line by line.

Event loop:-

Event loop is an endless loop, which waits for tasks, executes them and then sleeps until it receives more tasks. The event loop executes tasks from the event queue only when the call stack is empty i.e. there is no ongoing task. The event loop allows us to use callbacks and promises.



JavaScript: Client side Vs. Server side:-

Client side programming includes any coding or computation or effects or animation or any sort of interaction your website performs with the user via browser . But server side programming is that which performs all the task in the server only .

What is Client-Side:-

Client-side, on the other hand, refers to processes that are carried out on the user's device, typically in the user's web browser. These processes are executed after the website or web application has been delivered to the user's device, and they can include tasks such as rendering and displaying a web page, handling user interactions, or running JavaScript code.

What is Server-Side:-

Server-side refers to processes that are carried out on the web server, where the website or web application is hosted. These processes are typically executed by the server before the website or web application is delivered to the user's device, and they can include tasks such as retrieving data from a database, rendering a web page, or handling user input.

What is MVC (Model, View and Controller) Architecture:-

MVC is an acronym for Model-View-Controller. It is a design pattern for software projects.

In MVC pattern, application and its development are divided into three interconnected parts. The advantage of this is it helps in focusing on a specific part of the application name, the ways information is presented to and accepted from, the user. It helps in allowing for efficient code reuse and the parallel development of the application. Even if the project structure might look a little different than an ideal MVC structure, the basic program flow in and out the application remains the same.

**Explanation:-**

Model:-

Model represents the structure of data, the format and the constraints with which it is stored. It maintains the data of the application. Essentially, it is the database part of the application.

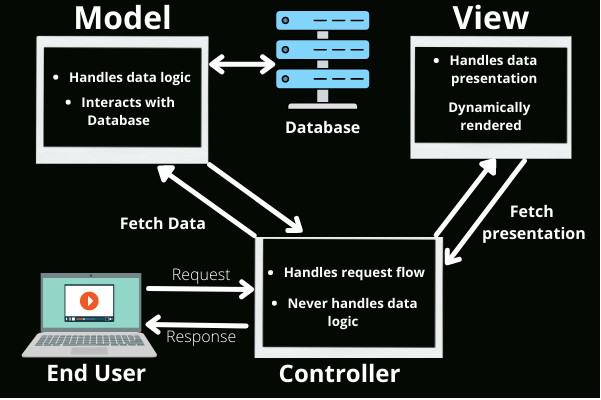
View:-

View is what is presented to the user. Views utilize the Model and present data in a form in which the user wants. A user can also be allowed to make changes to the data presented to the user. They consist of static and dynamic pages which are rendered or sent to the user when the user requests them.

Controller:-

Controller controls the requests of the user and then generates appropriate response which is fed to the viewer. Typically, the user interacts with the View, which in turn generates the appropriate request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response.

* Model is data part.
* View is User Interface part.
* Controller is request-response handler.



npm Commands

|  |  |
| --- | --- |
| **Commands** | **Decriptions** |
| npm init | Initializes a node project. Creates packages.json to track required modules/packages, as well as the node\_modules folder to track imported packages. |
| npm install | Installs any requirements for the local node package based on the contents of package.json. |
| npm install <package-name> | Installs a package from NPM’s own repository as well as any requirements specified in that package’s package.json file |

Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| **API** | Application Programming Interface. |
| **Web Service** | A type of API that supports HTTP Requests, returning data such as JSON or plain text. |
| **Express** | A module for simplifying the http-server core module in Node to implement APIs |
| **npm** | The Node Package Manager. Used to initialize package.json files and install Node project dependencies. |
| **Module** | A standalone package which can be used to extend the functionality of a Node project. |
| **Package** | Any project with a package.json file. |
| **API Documentation** | A file detailing the endpoints, usage, and functionality of various endpoints of an API. |
| **Server** | A publicly accessible machine which exchanges information with one or more clients at a time. |
| **Client** | A private or public machine which requests information from a server. |

Useful Core Modules

|  |  |
| --- | --- |
| **Module** | **Description** |
| **fs** | The “file system” module with various functions to process data in the file system. |
| **path** | Provides functions to process path strings. |

Other Most Useful Modules

The following modules must be installed for each new project using **npm install <module-name>.**

|  |  |
| --- | --- |
| **Module** | **Description** |
| **express** | A module for simplifying the http-server core module in Node to implement APIs. |
| **multer** | Used to support FormData POST requests on the server-side so we can access the req.body parameters. |
| **body-parser** | Body-parser parses is an HTTP request body that usually helps when you need to know more than just the URL being hit. Specifically in the context of a POST, PATCH, or PUT HTTP request where the information you want is contained in the body. |
| **nodemon** | It monitors your project directory and automatically restarts your node application when it detects any changes. |
| **bcrypt** | Bcrypt turns a simple password into fixed-length characters called a hash. |
| **jsonwebtoken** | JWTs are mainly used for authentication. After a user sign in to an application, the application then assigns JWT to that user. Subsequent requests by the user will include the assigned JWT. This token tells the server what routes, services, and resources the user is allowed to access. |
| **cors** | CORS or Cross-Origin Resource Sharing in Node. js is a mechanism by which a front-end client can make requests for resources to an external back-end server. The single-origin policy does not allow cross-origin requests and CORS headers are required to bypass this feature. |
| **mongoose** | Mongoose is a Node. js-based Object Data Modeling (ODM) library for MongoDB. It is akin to an Object Relational Mapper (ORM) such as SQLAlchemy for traditional SQL databases. The problem that Mongoose aims to solve is allowing developers to enforce a specific schema at the application layer. |
| **@babel/cli** | Babel comes with a built-in CLI which can be used to compile files from the command line. |
| **@babel/core** | Transform syntax. Polyfill features that are missing in your target environment (through a third-party polyfill such as core-js) Source code transformations (codemods) And more! |
| **@babel/node** | babel-node is a CLI that works exactly the same as the Node.js CLI, with the added benefit of compiling with Babel presets and plugins before running it. |
| **@babel/preset-env** | @babel/preset-env is a smart preset that allows you to use the latest JavaScript without needing to micromanage which syntax transforms (and optionally, browser polyfills) are needed by your target environment(s). |

Babel: -Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. Here are the main things Babel can do for you:

* Transform syntax
* Polyfill features that are missing in your target environment (through a third-party polyfill such as core-js)
* Source code transformations (codemods)

Example1: -

// Babel Input: ES6 arrow function

[1, 2, 3].map(n => n + 1);

// Babel Output: ES5 equivalent

[1, 2, 3].map(function(n) {

return n + 1;

});

Example2:-

Import package

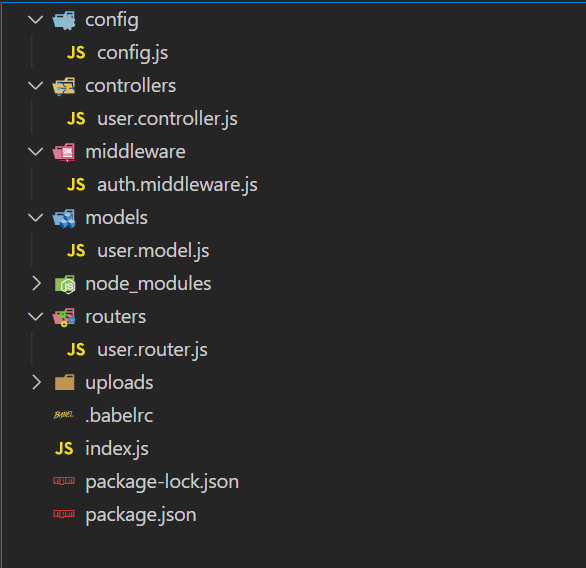
// Babel Input: ES6 arrow function

Import express from ‘express’

// Babel Output: ES5 equivalent

const express = require(‘express’)

**File structure: -**



Express.JS

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Create Node project

Create project, add package.json configuration

mkdir nodeproject # create directory

cd nodeproject # enter the directory

npm init -y # Initialize a configuration

**Install dependencies**

npm install express

**Create entry file index.js**

Basic server setup

**index.js**

*// import express and morgan*

const express = require("express")

*// create an application object*

const app = express()

*//middleware for parsing urlencoded data*

app.use(express.urlencoded({extended: true}))

*// middleware for parsing incoming json*

app.use(express.json())

*// to set a folder for static file serving*

app.use("/static", express.static("static"))

*// define a PORT variable from the environment with a default value*

const PORT = process.env.PORT || 4000

*// writing pass an anonymous function*

app.get("/endpoint", (req, res) => {

res.send("The Response")

})

*// using a named function*

function routeHandler(req, res){

res.send("the response")

}

app.get("/endpoint", routeHandler)

// Server Listener

app.listen(PORT, () => console.log(`Listening on port ${PORT}`))

**Run node app**

**$** node index.js

MongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas

MongoDB Connectivity

**Install mongoose**

$ npm install mongoose –save

*//import mongoose*

const mongoose = require('mongoose');

*//Define uri variable from the environment with a default value*

const uri = process.env.MONGO\_URI || 'mongodb://localhost/test';

*//mongo db connectivity*

mongoose.connect(uri, function(err, res) {

...

});

**Defining a schema**

const userSchema = new mongoose.Schema({

name: {

first: String,

last: { type: String, trim: true }

},

age: { type: Number, min: 0 },

posts: [ { title: String, url: String, date: Date } ],

updated: { type: Date, default: Date.now }

});

**SchemaTypes**

String

Number

Date

Buffer

Boolean

Mixed

ObjectId

Array