Install node to the windows

<https://nodejs.org/en>

node --version

v16.18.0 //My installed version

npm –version

8.15.1

**Approach:**Below are the fundamental steps to write an express app. Here we are covering the topics like setting up the environment with the installation of modules, creating an application, running the web server, and performing basic communication with the server. Must-Know how to use the node package manager for basic works, basic knowledge of the terminal for installing dependencies and modules, basic knowledge of how a web application works, and a good knowledge of ES6.

**Step by step Implementation:**

**Step 1:** Write this command in your terminal, to create a nodejs application, because our express server will work inside the node application.

**Syntax:**

npm init

This will ask you for few configurations about your project you can fill them accordingly, also you can change it later from the **package.json** file.

**Note:** Use `**npm init -y**` for default initialization

**Step 2:** Install necessary dependencies for our application.

npm install express

Something like this will be shown on successful installation,

Details: <https://www.geeksforgeeks.org/steps-to-create-an-express-js-application/>

**My Followed steps:**

**Step 1** − Start your terminal/cmd, create a new folder named hello-world and cd (create directory) into it −

npm init info

**Step 2** − Now to create the package.json file using npm, use the following code.

npm init –y

it will create auto package.json file

npm install --save express

OR npm i express

//Check the express version

> npm list express

nodewebsite1@1.0.0 F:\NODE-JS\2023\node\_projects\nodewebsite1

└── express@4.18.2

Then need to install DB

npm install --save mongoose

Or npm i mongoose

Then

We need src folder

* mkdir src
* cd src
* type nul > app.js //this will create app.js in src folder
* mkdir db
* mkdir models

# Middleware:

Functions that have access to req res object and next function

**Types:**

* Application level middleware
* Router level middleware
* Error-handling middleware
* Built-in middleware
* Third-party middleware

**Body parser**

if you used post request so you will need the body of the request, so you will need body-parser.

No need to install *body-parser* with express, but you have to use it if you will receive post request.

app.use(bodyParser.urlencoded({ extended: false }));

{ extended: false }

*false* meaning, you do not have nested data inside your *body* object. Note that: the request data embedded within the request as a body Object.\

If you don't want to use seperate npm package body-parser, latest express (4.16+) has built-in body-parser middleware and can be used like this,

const app = express();

app.use(express.json({ limit: '100mb' }));

p.s. Not all functionalities of body parse are present in the express. Refer documentation for full usage [here](http://expressjs.com/en/4x/api.html)

<https://medium.com/@mmajdanski/express-body-parser-and-why-may-not-need-it-335803cd048c>

<https://www.simplilearn.com/tutorials/nodejs-tutorial/body-parser-in-express-js>

# nodemon

nodemon is a tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected.

nodemon does **not** require any additional changes to your code or method of development. nodemon is a replacement wrapper for node. To use nodemon, replace the word node on the command line when executing your script.

>npm install nodemon

<https://www.npmjs.com/package/nodemon>

# validator.js

A library of string validators and sanitizers.

Install the library with > npm install validator

<https://www.npmjs.com/package/validator>

**Error Handling - Express JS::**

In express synchronize and asynchronies coding use different error handling system.

Synchronize - Express automatically handles error

What is an IIFE?

An IIFE (Immediately Invoked Function Expression) is a function that runs the moment it is invoked or called in the JavaScript event loop.

Having a function that behaves that way can be useful in certain situations.

IIFEs prevent pollution of the global JS scope.

Relational Database:

One to One-

Student🡪ADMITS🡪Department

<https://www.youtube.com/watch?v=wdysWKL3Qmw&ab_channel=OnlineSchool>

One to Many Relationship in DBMS-

Customer🡪GIVE🡪Order

<https://www.youtube.com/watch?v=H0U2UCahB7U&ab_channel=OnlineSchool>

<https://t4tutorials.com/php-mysqli-one-to-many-relationship-of-database/> //example

Many to Many Relationship in DBMS

Student --🡪 STUDY -🡪 Course

<https://www.youtube.com/watch?v=GoFUB-Iv38A&ab_channel=OnlineSchool>

<https://www.youtube.com/watch?v=7XHUjeU-n5E&ab_channel=TechAsif>

<https://stackoverflow.com/questions/2923809/many-to-many-relationships-examples>

//Clearly Explained

<https://www.youtube.com/watch?v=-c7PXt7i37A&ab_channel=TruthSeekers>

Foreign Key in DBMS:

<https://www.youtube.com/watch?v=hC8ETgdf6Z8&ab_channel=OnlineSchool>

Normalization in DBMS:

<https://www.youtube.com/watch?v=2e9B_c0xTDs&ab_channel=OnlineSchool>

**MongoDB**

Then we need to install mongodb in windows

>> Mongodb latest versions will not work with windows 8.1.

Download lower version setup msi file from

<https://www.mongodb.com/try/download/community>

Installed version - MongoDB server version: 4.2.24

Followed the instruction below vid::

<https://www.tutorialspoint.com/mongodb/mongodb_environment.htm>

<https://www.youtube.com/watch?v=aHEYzt8stHg&ab_channel=ThapaTechnical>

<https://www.youtube.com/watch?v=dyJFohH1DEM&list=PLHiZ4m8vCp9PHnOIT7gd30PCBoYCpGoQM&index=25&ab_channel=LearnwithSumit-LWS-Bangladesh>

For start mongodb shell go to cmd:

Just type mongo – Enter

And the mongo shell / mongo terminal will start automatically.

cls -> Enter // clear the shell

show dbs // Show all the databases

use ecommerce // Make a new db name ecommerce & select it

* Db will not show if it is empty
* Table is called collection in mongodb

Db -> Enter // Entering the selected db

For creating collection / Table we don’t need create command just insert 1 data in the table:

> db.products.insertOne({name: 'iPhone 7', price: 20000, category: 'smartphone',

active: true});

Show the tables data

ecommerce

> db.products.find();

> db.products.find().pretty(); // Showing record more nicely

For inserting multiple records:

> db.products.insertMany([{name: 'HP Pavillion', price: 8000, category: 'noteboo

k', active: true},{name: 'Dell Inspiron', price: 45000, category: 'notebook', ac

tive: false}, {name: 'Samsung Note 8', price: 72000, category: 'smartphone', act

ive: true}])

Selecting particular data:

> db.products.find({active: false, category: 'smartphone'}).pretty();

> db.products.find({active: false, category: 'notebook'}).pretty();

This show like this : --

{

"\_id" : ObjectId("6463198d9b7d3fa7f9e6681b"),

"name" : "Dell Inspiron",

"price" : 45000,

"category" : "notebook",

"active" : false

}

> db.products.find({active: false, category: 'notebook'}, {active: 0}).pretty();

{

"\_id" : ObjectId("6463198d9b7d3fa7f9e6681b"),

"name" : "Dell Inspiron",

"price" : 45000,

"category" : "notebook"

}

}

> db.products.find({category: 'smartphone'}).pretty().limit(1);

{

"\_id" : ObjectId("646316a39b7d3fa7f9e66819"),

"name" : "iPhone 7",

"price" : 20000,

"category" : "smartphone",

"active" : true

}

//Show 1 record second one

> db.products.find({category: 'smartphone'}).pretty().limit(1).skip(1);

{

"\_id" : ObjectId("6463198d9b7d3fa7f9e6681c"),

"name" : "Samsung Note 8",

"price" : 72000,

"category" : "smartphone",

"active" : true

}

//First record with findOne

//Pretty func will not work with findOne

> db.products.findOne({category: 'smartphone'});

{

"\_id" : ObjectId("646316a39b7d3fa7f9e66819"),

"name" : "iPhone 7",

"price" : 20000,

"category" : "smartphone",

"active" : true

}

// Update One record

> db.products.updateOne({name: 'iPhone 7', active: true}, {$set: {price: 25500}})

db.products.find({name: 'iPhone 7'})

// Check all and update all active set true – Multiple update

> db.products.updateMany({}, {$set: {active:true}})

// Delete one record

> db.products.deleteOne({name: 'Test'});

//Delete multiple

>db.products.deleteMany({category: ‘’notebook})

\*\* MongoDB GUI **MongoDB Compass**

We need a template engine hbs

hbs

[Express.js](https://expressjs.com/) view engine for [handlebars.js](https://handlebarsjs.com/)

<https://github.com/pillarjs/hbs>

Sort Data in Ascending & Descending Order in Node JS MongoDB

Basic syntax of MongoDB sort()

db.collection\_name.find().sort({field\_name: sort order})

<https://www.youtube.com/watch?v=krP6XMsIb8U&t=26s&ab_channel=ProgrammingExperience>

<https://www.youtube.com/watch?v=GBq3eAJziH0&ab_channel=procademy>

**Mongoose.**

elegant [mongodb](https://www.mongodb.com/) object modeling for [node.js](https://nodejs.org/en/)

Let's face it, **writing MongoDB validation, casting and business logic boilerplate is a drag**. That's why we wrote Mongoose.

<https://mongoosejs.com/>

$ npm install mongoose --save

<https://mongoosejs.com/docs/index.html>

ODM – Elegant Object Data Modeling for node js

Benefits of using mongoose

1. Abstraction from raw low level MongoDB

2. Relationship between NoSQL Data

3. Provides Schema Validation

4. Object- Data Mapping – translation of data into object that our code understands and vice versa

5. ~40-60% less code compared to raw mongodb package

In Mongo DB

Table 🡪 Collection

Record/row -> Document

**Mongoose:**

* Class
* Instance
* Instance methods
* Statics
* Query Helpers

<https://www.youtube.com/watch?v=KV4kNMxx7SY&list=PLHiZ4m8vCp9PHnOIT7gd30PCBoYCpGoQM&index=27&ab_channel=LearnwithSumit-LWS-Bangladesh>

<https://mongoosejs.com/docs/guide.html>

**node.bcrypt.js:**

<https://www.npmjs.com/package/bcrypt>

A library to help you hash passwords.

$npm install bcrypt

$ yarn add bcrypt

jsonwebtoken (JWT) nedd to install for authentication (Login/logout)

<https://www.npmjs.com/package/jsonwebtoken>

command:

$ yarn add jsonwebtoken

$ npm install jsonwebtoken

**dotenv**

For using .env file need a middlewire.

Need to install a package dotenv

$yarn add dotenv

$npm install dotenv

<https://www.npmjs.com/package/dotenv>

**colors.js package**

get color and style in your node.js console

npm install colors

<https://www.npmjs.com/package/colors>

## nanoid package

A tiny, secure, URL-friendly, unique string ID generator for JavaScript.

<https://www.npmjs.com/package/nanoid>

<https://github.com/ai/nanoid#readme>

<https://www.makeuseof.com/node-unique-ids-generate/>

# Postman : POST form-data file upload + JSON

<https://www.youtube.com/watch?v=1yqNfqfZPB8&ab_channel=ValentinDespa>

JWT.IO allows you to decode, verify and generate JWT.

<https://jwt.io/>

In NodeJS, require() is a built-in function to include external modules that exist in separate files. require() statement basically reads a JavaScript file, executes it, and then proceeds to return the export object.

var myVar = require('http'); //to use built-in modules

Var myVar2 = require('./myLocaModule') to use local modules

# JavaScript require vs import

In this article let us understand what JavaScript require() and import() statements do, how they fetch modules, and their differences. We will begin by understanding what a JavaScript module is, in the first place.

JavaScript module is a file that contains a few lines of code written in JavaScript. They are the same as JavaScript Libraries. Modules often contain a class or a library of functions that are used for a specific purpose. These can be called with the help of require and import statements. The use of modules reduces the number of lines of code in one’s program/script. A major advantage of modules is that functions of another module can be called without writing the body of the functions themselves.

Some of the common modules are:

CommonJS, AMD, RequireJS, ES(ECMAScript)6 Modules. Refer to [this medium article](https://medium.com/computed-comparisons/commonjs-vs-amd-vs-requirejs-vs-es6-modules-2e814b114a0b) for an in-depth explanation of how these modules are different.

## Syntax and explanation

### 1) require()

In NodeJS, require() is a built-in function to include external modules that exist in separate files. require() statement basically reads a JavaScript file, executes it, and then proceeds to return the export object. require() statement not only allows to add built-in core NodeJS modules but also community-based and local modules.

#### Syntax:

To include a module, the require() function is used with the name of the module:

**var** myVar = require('http'); //to use built-in modules

**Var** myVar2 = require('./myLocaModule') to use local modules

2) import()

import() & export() statements are used to refer to an ES module. Other modules with file types such as .json cannot be imported with these statements. They are permitted to be used only in ES modules and the specifier of this statement can either be a URL-style relative path or a package name. Also, the import statement cannot be used in embedded scripts unless such script has a type="module". A dynamic import can be used for scripts whose type is not “module”

Syntax:

**var** myVac = **import**("module-name");

import dotenv from 'dotenv';

Warning: To load an ES module, set "type": "module" in the package.json or use the .mjs extension.

(Use `node --trace-warnings ...` to show where the warning was created)

<https://flexiple.com/javascript/javascript-require-vs-import/>

What is express async handler for?

Simple middleware for handling exceptions inside of async express routes and passing them to your express error handlers.

# CORS কি - What is CORS policy

#### What is CORS?

CORS stands for Cross-Origin Resource Sharing. It allows us to relax the security applied to an API. This is done by bypassing the Access-Control-Allow-Origin headers, which specify which origins can access the API.

In other words, CORS is a browser security feature that restricts cross-origin HTTP requests with other servers and specifies which domains access your resources.

CORS is a node.js package for providing a [Connect](http://www.senchalabs.org/connect/)/[Express](http://expressjs.com/) middleware that can be used to enable [CORS](http://en.wikipedia.org/wiki/Cross-origin_resource_sharing) with various options.

$ npm install cors

<https://expressjs.com/en/resources/middleware/cors.html>

<https://www.youtube.com/watch?v=Ry_r8DCj3hw&ab_channel=LearnwithSumit-LWS-Bangladesh>

# Hate Try...Catch Error Handling in Node.js? Do This Instead

<https://www.youtube.com/watch?v=s5YoXms0ECs&ab_channel=Gravity>

// Global error handling in nodejs

<https://www.youtube.com/watch?v=sTYvHWEMfs0&ab_channel=ProgrammingExperience>

<https://stackoverflow.com/questions/56973265/what-does-express-async-handler-do>

<https://www.youtube.com/watch?v=abE3qFUIPGE&t=33s&ab_channel=YoussefAbbas> // concept clear

## express-async-handler

Simple middleware for handling exceptions inside of async express routes and passing them to your express error handlers.

<https://www.npmjs.com/package/express-async-handler>

npm install --save express-async-handler

# What is the lean() method in Mongoose?

lean() returns a JavaScript object instead of a Mongoose document.

When documents are queried, they are returned as **Mongoose Documents** by default. With the Mongoose **lean()** method, the documents are returned as plain objects

<https://www.educative.io/answers/what-is-the-lean-method-in-mongoose>

<https://mongoosejs.com/docs/tutorials/lean.html>

**\*\* How to run an existing node app from github?**

You have first to clone your repo:

git clone https://github.com/albinotonnina/albinotonnina.com your-destination-folder

then when the repository has been cloned:

cd your-destination-folder

npm install # installing the dependencies

npm start # or 'node .'

I've seen that maybe the build process is requested for that project: as they say in the Readme.MD you have to build the app like this:

npm run build

Or

npm run dev

for running in dev the project. Inside the package.json you can see what this commands do under the hood.

A Node.js application consists of the following three important components −

* **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.
* **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.
* Starting REPL
* REPL can be started by simply running **node** on shell/console without any arguments as follows.
* $ node

## REPL Commands

* **ctrl + c** − terminate the current command.
* **ctrl + c twice** − terminate the Node REPL.
* **ctrl + d** − terminate the Node REPL.
* **Up/Down Keys** − see command history and modify previous commands.
* **tab Keys** − list of current commands.
* **.help** − list of all commands.
* **.break** − exit from multiline expression.
* **.clear** − exit from multiline expression.
* **.save *filename*** − save the current Node REPL session to a file.
* **.load *filename*** − load file content in current Node REPL session.

## Stopping REPL

As mentioned above, you will need to use **ctrl-c twice** to come out of Node.js REPL.

$ node

>

(^C again to quit)

>