**NodeJS Introduction**

|  |  |
| --- | --- |
| Node.Js | It allows you To Run Javascript On The Server. |
| Environment | Open Source Server |
| Cost | Free |
| Run On Platforms | Windows, Linux, Unix, Mac OS X, Etc |
| Programming | Asynchronous |
| Extension | ".Js" |
| Purpose | It generates dynamic page content, does CRUD operations on server files and data base, Collects form data. |
| Nodejs Download | https://nodejs.org/ |

**Write a NodeJS Hello world Program.**

|  |  |
| --- | --- |
| Program Name | Write a NodeJS Hello world Program. |
| Program Code  File Name: helloworld.js | var httpobj = require('http');  httpobj.createServer(function (req, res)  {  res.writeHead(200,{'Content-Type': 'text/plain'});  res.end('Hello World!');  }).listen(8080); |
| Running | Compiling and Running  1. Open terminal and change the drive to code folder.  2. compiling: node helloworld.js  3. Running: open browser and type  http://localhost:8080/  4. See the output  Explanation: HTTP module used transfer data using Hyper Text Transfer Protocol (HTTP). HTTP server response is displayed as HTML and included in HTTP header with a content type. |
| Output | Hello World! |

NodeJS Program to display output in Terminal

|  |  |
| --- | --- |
| Program Name | Write a NodeJS Program to display output in Terminal. |
| Program | File Name: helloworld.js  var httpobj = require('http');  httpobj.createServer(function (req, res)  {  console.log('Display the Output in the console only');  }).listen(8080); |
| Running | 1. Open terminal and change the drive to code folder.  2. compiling: node helloworld.js  3. Running: open browser and type  http://localhost:8080/  4. See the output in Terminal. |
| Output | Display the Output in the console only |

**NodeJS Modules**

Nodejs Modules are JavaScript libraries consists of set of functions and it requires require () function.

**Creating own Modules**

|  |  |
| --- | --- |
| Program Name | Write a Program for Creating own Modules. |
| File Name: ownmodule.js | exports.DateTimeObj = function ()  {  return Date();  }; |
| File Name: Nodeexample.js | var http = require('http');  var dtobj = require('./ownmodule');  http.createServer(function (req, res)  {  res.writeHead(400, {'Content-Type': 'text/html'});  res.write("Current Date and time: " + dtobj.DateTimeObj());  res.end();  }).listen(8080); |
| Output | Current Date and time: Fri Jan 21 2022 22:39:37 GMT+0500 (West Kazakhstan Time) |

**NodeJS File System Module**

It is used to work (Read, Create, Update, Delete, Rename files) with the files in the server system.

|  |  |
| --- | --- |
| Program | NodeJS File System Module for Reading files using Filesystem object. |
| File Name: Example.html | File reading Example using nodejs |
| File Name: Nodejsexample.js | var httpobj = require('http');  var fsobj = require('fs');  httpobj.createServer(function (req, res) {  fsobj.readFile('example.html', function(err, data) {  res.writeHead(400, {'Content-Type': 'text/html'});  res.write(data);  return res.end();  });  }).listen(8080); |
| Output | File reading Example using nodejs |

**Append text to files using Filesystem object**

|  |  |
| --- | --- |
| Program Name | Write a Program for NodeJS File System Append text to files using Filesystem object. |
| Theory | Program for NodeJS File System Module for Append text to files using Filesystem object |
| Program | File Name: Example.html  File reading Example using nodejs  File Name: Nodejsexample.js  var fsobj = require('fs');  fsobj.appendFile('Example.html', 'Append text to files!', function (err) {  if (err) throw err;  console.log('File Saved!');  }); |
| Output | Example.html: File reading Example using nodejsAppend text to files! |
| Explanation | Adds text to end of the file. |

**Write text to files using Filesystem object**

|  |  |
| --- | --- |
| Program Name | Write a Program for Write text to files using Filesystem object in NodeJS. |
| Theory | Program for Write text to files using Filesystem object in NodeJS |
| Program Code | File Name: Example.html  File reading Example using nodejs  File Name: Nodejsexample.js  var fsobj = require('fs');  fsobj.writeFile('example.html', 'write content to file!', function (err) {  if (err) throw err;  console.log('File Saved!');  }); |
| Output | Example.html: write content to file! |
| Explanation | Write: creates a file if not there and writes new content to file after removing the old content. |

**Replace text in the files using Filesystem object**

|  |  |
| --- | --- |
| Program Name | Write a Program for Replace text in the files using Filesystem object in NodeJS. |
| Theory | Program for Replace text in the files using Filesystem object in NodeJS. |
| Program | File Name: Example.html  write content to file!  File Name: Nodejsexample.js  var fsobj = require('fs');  fsobj.writeFile('example.html', 'written content to file!', function (err) {  if (err) throw err;  console.log('written content to file!');  }); |
| Output | Example.html: written content to file! |

**Delete Files using Filesystem object in NodeJs**

|  |  |
| --- | --- |
| Program Name | Write a Program for Delete Files using Filesystem object in NodeJs. |
| Theory | Program for Delete Files using Filesystem object in NodeJs. |
| Program | File Name: Example.html  write content to file!  File Name: Nodejsexample.js  var fsobj = require('fs');  fsobj.writeFile('example.html', 'written content to file!', function (err) {  if (err) throw err;  console.log('written content to file!');  }); |
| Output | File deleted using File system object in nodejs Example.html: File deleted |

**Write a Program for Rename Files using Filesystem object in NodeJs.**

|  |  |
| --- | --- |
| Program Name | Program for Rename Files using Filesystem object in NodeJs |
| Program | File Name: Example.html  write content to file!  File Name: Nodejsexample.js  var fsobj = require('fs');  fsobj.rename('example.html', 'example2.html', function (err) {  if (err) throw err;  console.log('File Renamed using File system object in nodejs');  }); |
| Output | 'File Renamed using File system object in nodejs i.e 'example.html' to 'example2.html' |

**Express**

1. Express as the backend framework, to build single page, multi-page, and hybrid web and mobile applications, supports MVC (Model-View-Controller), single threaded and asynchronous model

2. Express which is a Node.js framework for web development that comes with features like routing and rendering and support for REST APIs.

**Routing**

|  |  |
| --- | --- |
| Routing | Refers to how a server side application responds to a client request to a particular endpoint. |
| endpoint | consists of a URI (a path such as / or /books) and an HTTP method such as GET, POST, PUT, DELETE, etc. |
| Routing Methods | app.get() |
| Routing path | Strings, string patterns, or regular expressions. |
| Routing Parameters | parameters passed in the url |

**Steps**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Step Name | command | Output |
| 1 | Download nodejs and install.  https://nodejs.org/en/download |  |  |
| 2 | open command prompt and type | node –version | v20.11.1 |
| 3 | Install npm (Node package manager) | npm install -g npm |  |
| 4 | Testing npm (version) | npm -v | 10.5.0 |
| 5 | Creating package.json  A JSON (JavaScript Object Notation) file is contains every information about any Express project. The number of modules installed, the name of the project, the version, and other meta information. To add Express as a module in our project, first we need to create a project directory and then create a package.json file. This will generate a package.json file in the root of the project directory. To install any module from npm we need to have package.json file exist in that directory. | npm init --yes |  |
| 6 | Installing Express conformed using package.json file named dependencies and new folder called node\_modules appeared in the root of our project directory  named dependencies  {  "name": "express-web-app", "version": "0.1.0",  "description": "", "main": "index.js",  "scripts": { "test": "echo \"Error: no test specified\" && exit 1" }, "keywords": [], "license": "MIT",  "dependencies": { "express": "4.16.0" }  } | npm install --save express |  |
| 7 | **Building /** start **a Server with Express** | node index.js |  |

**Example**

|  |  |
| --- | --- |
| index.js | var express = require('express');  var app = express();  app.get('/', function (req, res) {  console.log("Got a GET request for the homepage");  res.send('Welcome to JavaTpoint!');  })  app.post('/', function (req, res) {  console.log("Got a POST request for the homepage");  res.send('I am Impossible! ');  })  app.get('/del\_student', function (req, res) {  console.log("Got a GET request for /enrolled\_student");  res.send('I am Deleted!');  })  app.get('/enrolled\_student', function (req, res) {  console.log("Got a GET request for /enrolled\_student");  res.send('I am an enrolled student.');  })  var server = app.listen(8000, function () {  var host = server.address().address  var port = server.address().port  console.log("Example app listening at http://%s:%s", host, port)  }) |
| index.html | <a href="http://localhost:8000/">Home</a> <br><br>  <a href="http://localhost:8000/del\_student">del\_student</a> <br><br>  <a href="http://localhost:8000/enrolled\_student">enrolled\_student</a> <br><br> |
| Or | Goto browser and type and enter  <http://localhost:8000/>  <http://localhost:8000/del_student>  <http://localhost:8000/enrolled_student> |
| Output |  |

**Anatomy of an Express Application**

A typical structure of an Express server file will most likely contain the following parts:

|  |  |
| --- | --- |
| Dependencies | Importing the dependencies such as the express itself. These dependencies are installed using npm like we did in the previous example. |
| Instantiations | These are the statements to create an object. To use express, we have to instantiate the app variable from it. |
| Configurations | These statements are the custom application based settings that are defined after the instantiations or defined in a separate file (more on this when discuss the project structure) and required in our main server file. |
| Middleware | These functions determine the flow of request-response cycle. They are executred after every incoming request. We can also define custom middleware functions. We have section on them below. |
| Routes | They are the endpoints defined in our server that helps to perform operations for a particular client request. |
| Bootstrapping Server | The last that gets executed in an Express server is the app.listen() function which starts our server. |

**Template engine**

It is used to merge HTML page with the data from your program. The engine is simple and powerful.

At runtime, it replaces variables in a template file with actual values and transforms the template into an HTML file sent to the client.

Example: Directory Structure (Desktop\te\ index.js, Desktop\te\ views\ index.pug)

|  |  |
| --- | --- |
| index.js | var express = require('express');  var app = express();  app.set('view engine', 'pug')  app.get('/', function (req, res) {  console.log("Got a GET request for the homepage");  res.render('index', { title: 'Hey', message: 'Hello there!' })  })    var server = app.listen(8000, function () {  var host = server.address().address  var port = server.address().port  console.log("Example app listening at http://%s:%s", host, port)  }) |
| index.pug | html  head  title= title  body  h1= message |
| run | npm init --yes  npm install --save express  npm install pug –save  node index.js |
| output | Hello there! |

These template engines work “out-of-the-box” with Express:

|  |  |
| --- | --- |
| Pug | Haml-inspired template engine (formerly Jade). |
| Haml.js | Haml implementation. |
| EJS | Embedded JavaScript template engine. |
| hbs | Adapter for Handlebars.js, an extension of Mustache.js template engine. |
| Squirrelly | Blazing-fast template engine that supports partials, helpers, custom tags, filters, and caching. Not white-space sensitive, works with any language. |
| Eta | Super-fast lightweight embedded JS template engine. Supports custom delimiters, async, whitespace control, partials, caching, plugins. |
| combyne.js | A template engine that hopefully works the way you’d expect. |
| Nunjucks | Inspired by jinja/twig. |
| marko | A fast and lightweight HTML-based templating engine that compiles templates to CommonJS modules and supports streaming, async rendering and custom tags. (Renders directly to the HTTP response stream). |
| whiskers | Small, fast, mustachioed. |
| Blade | HTML Template Compiler, inspired by Jade & Haml. |
| Haml-Coffee | Haml templates where you can write inline CoffeeScript. |
| express-hbs | Handlebars with layouts, partials and blocks for express 3 from Barc. |
| express-handlebars | A Handlebars view engine for Express which doesn’t suck. |
| express-views-dom | A DOM view engine for Express. |
| rivets-server | Render Rivets.js templates on the server. |
| LiquidJS | A simple, expressive and safe template engine. |
| express-tl | A template-literal engine implementation for Express. |
| Twing | First-class Twig engine for Node.js. |
| Sprightly | A very light-weight JS template engine (45 lines of code), that consists of all the bare-bones features that you want to see in a template engine. |
| html-express-js | A small template engine for those that want to just serve static or dynamic HTML pages using native JavaScript. |

**MongoDB**

It is a document database and can be installed locally or hosted in the cloud.

|  |  |  |
| --- | --- | --- |
|  | **SQL Databases** | **Document Databases** |
| Considered as | Relational Databases | Document / non-relational database. It don’t store data in Document but it relational data is stored differently |
| Store data | Separate Multiple Tables | Flexible documents |
| Concept uses | Tables | collections |

**Install MongoDB Shell (mongosh)**

1. mongodb 1.10.6 download

2. mongodb compass = mongodb-windows-x86\_64-7.0.6-signed Download

3. Goto This PC properties-- > Advanced System settings-- > Environmental variables

-- > System variables -- >path-- > Add: C:\Program Files\MongoDB\Server\7.0\bin

4. Goto This PC properties-- > Advanced System settings-- > Environmental variables

-- > User variables -- >path-- > Add: C:\mongosh-1.10.6-win32-x64\bin

**Create a DB**

|  |  |
| --- | --- |
|  |  |

Example: Give data base name as “mydb” and Collection Name as “mycoll”

Add Records

|  |  |
| --- | --- |
|  | /\*\* \* Paste one or more documents here\*/  { "name":"1", "salary":"99999", "depart":"CSM" } |

**Mongodb Commands**

|  |  |  |
| --- | --- | --- |
| **Command** | **Details** | **Example** |
| Type in cmd | mongosh | mongosh |
| mongosh version | Display mongosh version | mongosh --version |
| showdbs | Shows the database of the system | showdbs |
| Use | Create / connect to a specific database | use mydb |
| Insert | Insert a document | db.emp.insert({  empname: "power",  empsalry: 99999,  dept: "csm" }) |
| createCollection | Create Collection | db.createCollection("posts") |
| insertOne | Insert single data to a collection.  db.posts.insertOne(object) | db.posts.insertOne(object)  db.posts.insertOne({"title": "wisdom"}) |
| insertMany() | Insert multiple data to a collection.  db.posts. insertMany (object) | db.posts.insertMany([  {  title: "wm1",  body: "s/w company",  category: "development",  like: 1,  tags: ["development", "maintainance"],  date: Date()  },  {  title: "wm2",  body: "s/w company",  category: "maintainance",  like: 2,  tags: ["development", "maintainance"],  date: Date()  },  ]) |
| find | db.posts.find() | db.posts.find() |
| find | findOne() | db.posts.findOne ( { category: "maintainance"}) |
| Querying Data | Display the records | db.posts.find( {category: "News"} ) |
| Projection |  | db.posts.find({}, {title: wm1, date: 1})  db.posts.find({}, {\_id: 0, title: 1, date: 1}) |
| updateOne() | Used to update a data | db.posts.updateOne( { title: "wm1" }, { $set: { like: 9} } ) |
| deleteOne | Used to delete a data | db.posts.deleteOne({ title: "wm1" }) |
| deleteMany |  | db.posts.deleteMany({ category: "Technology" }) |

**MongoDB Query Operators**

There are many query operators that can be used to compare and reference document fields.

|  |  |  |  |
| --- | --- | --- | --- |
| **Logical** **operators** | | **Comparison operators** | |
| **operator** | Returns documents where | $eq: Values are equal  $ne: Values are not equal  $gt: Value is greater than another value  $gte: Value is greater than or equal to another value | $lt: Value is less than another value  $lte: Value is less than or equal to another value  $in: Value is matched within an array |
| $and | both queries match |
| $or | either query matches |
| $nor | both queries fail to match |
| $not | the query does not match |

**Operators**

|  |  |
| --- | --- |
| **Evaluation operators** | $regex: Allows the use of regular expressions when evaluating field values  $text: Performs a text search  $where: Uses a JavaScript expression to match documents  MongoDB Update Operators |
| **Fields operators** | $currentDate: Sets the field value to the current date  $inc: Increments the field value  $rename: Renames the field  $set: Sets the value of a field  $unset: Removes the field from the document |
| **Array operators** | $addToSet: Adds distinct elements to an array  $pop: Removes the first or last element of an array  $pull: Removes all elements from an array that match the query  $push: Adds an element to an array |

**Connecting to MongoDB**

|  |  |
| --- | --- |
| code | const { MongoClient } = require(mongodb);  const uri = "mongodb://localhost:27017";  const client = new MongoClient(uri);  async function run() {  try {  await client.connect();  const db = client.db(mydb);  const collection = db.collection(mycoll);  // Find the first document in the collection  const first = await collection.findOne();  console.log(first);  } finally {  // Close the database connection when finished or an error occurs  await client.close();  }  }  run().catch(console.error); |
| Run | node index.js |
| Output | [Running] node "c:\Users\student\Desktop\index.js"  { \_id: new ObjectId('65defb558e9ad877d9ac676c'),  name: '1', salary: '99999', depart: 'CSM'}  [Done] exited with code=0 in 0.249 seconds |

db.posts.aggregate([

{ $match: { likes: { $gt: 1 } } },

{

$group: { \_id: "$category", totalLikes: { $sum: "$likes" } }

}

])