|  |
| --- |
| **NODEJS** |
| **1. NodeJS là gì ?** |
| **- NodeJS là Javascript RUN TIME**  Tạo ra một môi trường độc lập để chạy JavaScript , sử dụng lõi công nghệ JavaScript V8 Engine. ( không cần phải mở trình duyệt để chạy JavaScript nữa)  **- Sử dụng NodeJS để tạo một web Server chạy trên chính máy tính của bạn**  Giống các thằng Live Server trên Visual Studio code hoạt động  NodeJS sẽ tạo nền móng , cài đặt các thư viện vào thẳng dự án, tạo liên kết với web server  **- NodeJS là một chương trình bất đồng bộ ( asynchronous )**  NodeJS loại bỏ sự chờ đợi , sẵn sàng thực hiện yêu cầu tiếp theo  **- Chức năng của NodeJS :**   * Tạo nội dung trang động * **Creat, open - close , read, write,** delete các file trên server * Thu thập dữ liệu từ **form data** * **Add, delete, modify data** từ database   **- Một file Node.js gồm gì:**   * Node.js file chứa các yêu cầu (task) mà sẽ được thực thi dựa trên một số sự kiện nhất định ( vd: truy cập vào port trên server ) * Node.js file phải được khởi tạo trên máy chủ trước khi có bất kỳ tác động nào   Vd: Tạo file myfirst.js :  myfirst.js  var http = require('http');  http.createServer(function (req, res) {   res.writeHead(200, {'Content-Type': 'text/html'});   res.end('Hello World!'); }).listen(8080);  - Vào cmd để khai báo file trên máy tính : **node myfirst.js**  - Lúc này file được chạy và máy tính của bạn trở thành một Server sẽ lắng nghe sự kiện ở địa chỉ: [http://localhost:8080](http://localhost:8080/) và khi có yêu cầu truy cập server sẽ trả về “ Hello world “ |
| **2. Framework - ExpressJS** |
| - EpressJs là framework để xây dựng API - Application Programming Interface ( Giao diện lập trình ứng dụng - giúp các ứng dụng giao tiếp với nhau hoặc client-server giao tiếp) , quản lý các Route dễ dàng hơn |
| **3. Window Command Line** |
| Các lệnh thường dùng :   |  |  |  | | --- | --- | --- | | **Lệnh** | **Ý Nghĩa** | **Ví dụ** | | **cd** | Change directory : Truy cập vào folder | **cd workspace**  *// Truy cập vào folder có tên workspace* | | **cd..** | Quay lại thư mục gốc |  | | **dir** | Directory : Liệt kê các thư mục |  | | **mkdir** | Make directory: Tạo thư mục mới | **mkdir newfolder**  // Tạo thư mục mới tên newfolder | | **rmdir**  **rd** | Remove directory: Xoá thư mục | **rmdir newfolder** hoặc **rd newfolder**  // Xoá thư mục mới tên newfolder | | **rename** | Đổi tên thư mục | **rename newfolder myfolder**  // Đổi tên thư mục **newfolder** thành **myfolder** | | **type** | Tạo một file mới | **TYPE nul>index.js**  // Tạo một file index.js rỗng | | **del** | Xoá file | **del index.js**  // Xoá file index.js | |
| **4. Node.js Modules** |
| - Node.js Modules là tập hợp các chức năng bạn muốn có trên ứng dụng của bạn  - NodeJs có sẵn một vài modules có thể sử dụng luôn mà không cần phải cài thêm   |  |  | | --- | --- | | **Module built-in** | **Description** | | **assert** | Cung cấp tập hợp các thử nghiệm xác nhận | | **buffer** | Xử lý dữ liệu nhị phân binary data | | **child\_process** | Chạy một chương trình con | | **cluster** | Chia nhỏ một chương trình thành nhiều chương trình | | **crypto** | Xử lý các hàm mật mã OpenSSL | | **dgram** | Cung cấp triển khai các ổ cắm dữ liệu UDP | | **dns** | Tra cứu và phân giải DNS | | **events** | Xứ lý sự kiện | | **fs** | Xử lý file system | | **http** | Khiến Node.js xử lý như một HTTP server | | **https** | Khiến Node.js xử lý như một HTTPS server | | **net** | Tạo ra Servers và Clients | | **os** | Cung cấp thông tin về hệ điều hành ( Operation System ) | | **path** | xử lý các file path | | **querystring** | xử lý các URL có đính kèm thông tin | | **readline** | xử lý các luồng có thể đọc một dòng tại một thời điểm | | **stream** | xứ lý dữ liệu phát trực tuyến | | **string\_decoder** | giải mã các đối tượng buffer thành chuỗi | | **timers** | Bộ hẹn giờ để thực thi hành động tiếp theo | | **tls** | Triển khai giao thức TLS và SSL | | **url** | Phân tích đoạn URL | | **util** | Truy cập các chức năng | | **vm** | Biên dịch JavaScript trong máy ảo (virtual machine) | | **zlib** | Nén và giải nén file |   - **Để sử dụng module** trong Node.js ta sử dụng phương thức **require(‘module\_name’)**  vd: **const http = require(‘ http ’)**  **- Có thể tạo một module** bằng từ khoá **exports :**  vd: tạo file **myfirstmodule.js**  **exports.myDateTime = function( ) {**  **return console.log(‘test’)**  **}**  **- Để sử dụng module vừa tạo ta sử dụng :**  **const myfirstmodule = require(‘./myfirstmodule’)**  // đường dẫn tới file module đó , còn với những module có sẵn chỉ cần gõ tên của module không cần gõ đường dẫn |
| **4.1. HTTP module** |
| HTTP module cho phép Nodejs truyền dữ liệu qua HTTP protocol  const http = require(‘http’)   |  |  | | --- | --- | | **Method** | **Description** | | **.createServer( )** | Tạo một HTTP server | | **.get( )** | Đổi phương thức thành get( ), xử lý yêu cầu và trả về kết quả | | **.request( )** | Trả về một Object chứa tập hợp yêu cầu của người dùng |  * **.createServer( ) :**   Cú pháp: **http**.**createServer**( ***requestListener*** )  - Phương thức createServer biến máy tính của bạn thành một HTTP server  **- Tạo ra một HTTP server object có các Methods và Properties sau:**   |  |  | | --- | --- | | **Method** | **Description** | | **.close( )** | Ngắt kết nối tới server | | **.listen( )** | Khiến server lắng nghe tại Cổng ( port ) trên máy tính | | **.setTimeout( )** | Tạo thời gian chờ của Server, mặc định 2 phút, hết thời gian không kết nối được sẽ báo lỗi | | **Property** | **Description** | | **.listening** | Trả về TRUE nếu server đang kết nối | | **.maxHeadersCount** | Giới hạn số lượng Header đến | | **.timeOut** | Xác định thời gian chờ của Server |   vd: const httpServer = http.createServer( *requestListener* );  httpServer.timeOut = 2000 ;// sau 2s không kết nối sẽ báo lỗi  - ***requestListener*** : Một function được thực thi mỗi khi Server nhận được yêu cầu |
| **requestListener - Lắng nghe yêu cầu** |
| - ***requestListener*** : Một function được thực thi mỗi khi Server nhận được yêu cầu  Cú pháp: **function ( req , res ) { }**   * **req - request : Yêu cầu**   **req là một Object có các thuộc tính và phương thức sau**   |  |  | | --- | --- | |  |  | | .headers | Trả về Object chứa các key-value tên header và giá trị | | .httpVersion | Trả về phiên bản HTTP được gửi từ Client | | .method | Trả về Request Method | | .rawHeaders | Trả về Array dạng bản thô của *Header* | | .rawTrailers |  | | .setTimeout( ) | Gọi hàm sau khi một thời gian được chỉ định | | .statusCode | Trả về Status cide | | .trailers | Trả về Object chứa trailers | | .url | Trả về Url được gửi lên | | .query | Trả về Object chứa query parameter | | .body | Trả về Object chứa parameter gửi với phương thức POST |  * **res - response : Phản hồi**   **res là một Object đại diện cho luồng dữ liệu gửi lại cho phía Client**   |  |  | | --- | --- | |  |  | | .addTrailers( ) | Thêm các HTTP trailing headers | | .end( ) | Chỉ cho Server phản hồi tới đây là hoàn tất | | .getHeader( ) | Trả về giá trị của Header được xác định | | .headersSent | Trả về TRUE nếu các header đã được gửi đi | | .removerHeader( ) | xoá header được chỉ định | | .sendDate | Chuyển thành FALSE nếu DATE không được gửi đi trong header | | .setHeader( ) | Chỉnh sửa header chỉ định | | .setTimeout | Cài đặt thời gian chờ của socket | | .statusCode | Tạo status code gửi về phía Client | | .statusMessage | Tạo status message gửi về phía Client | | .write( ) | Gửi đoạn text , hoặc **html** về Client | | .writeContinue( ) | Gửi HTTP Continue message | | .writeHead( ) | Gửi một status và phản hồi với header về Client |   **vd: res.writeHead**(200, {'Content-Type': 'text/html});  - Gửi về status code: 200 , Dạng gửi về sẽ là text hoặc html |
| **4.2 fs Module - Xử lý File System** |
| - Node.js như là một File Server, cho phép bạn tương tác với các file trên máy tính.  - Những hành động thường dùng để xử lý File System:  **Read - Create - Update - Delete - Rename**  **const fs = require(‘fs’)**   * **Read Files :**   **fs.readFile( file , charser , callback )**  vd: Tạo 1 file index.html chứa nội dung HTML cùng cấp  Dùng http module tạo server:  http.createServer(function (req, res) {    **fs.readFile( ‘index.html’, function(err , data) {**  **res.writeHead( 200 , {‘Content-Type’ : ‘text/html’ } );**  **res.write(data);**  **return res.end();**  **});**  }).listen(8080);  🡺 Khi đọc file ta sẽ có 2 đối số là err - error và data - dữ liệu trong file vừa đọc  để xử lý ở **function** (err ,data ){  … code xử lý err và data …..  **return res.end()** //để báo server kết thúc phản hồi  }     * **Create File:**  |  |  | | --- | --- | |  |  | | fs.appendFile( ) | **Thêm nội dung vào cuối file, nếu file chưa có nó tạo file luôn** | | fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {   if (err) throw err;   console.log('Saved!'); }); | | | fs.open( ) | Mở file để chuẩn bị ghi nội dung vào file ,nếu file chưa có nó sẽ tạo file mới | | fs.open('mynewfile2.txt', 'w', function (err, file) {    if (err) throw err;   console.log('Saved!');  }); | | | fs.writeFile( ) | Thay thế một nội dung được chỉ định | | fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {   if (err) throw err;   console.log('Saved!'); }); | |  * **Update File:**   2 phương thức **appendFile( )** và **writeFile( )** dùng để chỉnh sửa nội dung File   * **Delete File:**  |  |  | | --- | --- | | fs.unlink( ) | Xoá file chỉ định | | fs.unlink('mynewfile2.txt', function (err) {   if (err) throw err;   console.log('File deleted!'); }); | |  * **Rename File:**  |  |  | | --- | --- | | fs.rename( ) | Thay tên cho file | | fs.rename('mynewfile2.txt', 'myrenamefile2.txt',function (err) {   if (err) throw err;   console.log('File deleted!'); }); | | |
| **4.3 URL module - Làm việc với URL** |
| **URL module giúp chia nhỏ một địa chỉ web thành các phần đọc được.**  const url = require('url'); const adr = 'http://localhost:8080/default.htm?year=2017&month=february'; const q = url**.parse**(adr, true);  console.log(q.**host**); //returns 'localhost:8080' console.log(q.**pathname**); //returns '/default.htm' console.log(q.**search**); //returns '?year=2017&month=february'  var qdata = q**.query**; //returns an object: { year: 2017, month: 'february' } console.log(qdata.month); //returns 'february'   |  |  | | --- | --- | | **Method** | **Description** | | **url.format( )** | Trả về chuỗi URL được định dạng lại | | **url.parse( )** | Trả về URL Object có thể lấy ra các phần như: **protocal , port , hostname , pathname , ……** | | **url.resolve( )** | Xử lý đoạn URL , nối 2 đoạn url với nhau | |
| **Kết hợp : http - fs - url Module** |
| - Dùng các module trên để tạo thành File Server  - Khi truyền vào URL / tên file sẽ trả lại nội dung html trong file đó  const **http** = require('http'); const **url** = require('url'); const **fs** = require('fs');  **http.createServer**(function (**req**, **res**) {   const q = **url.parse**(**req.url**, true);   const filename = "." + **q.pathname**; **fs.readFile**(filename, function(err, data) {     if (err) {       res.writeHead(404, {'Content-Type': 'text/html'});       return res.end("404 Not Found");     }      res.writeHead(200, {'Content-Type': 'text/html'});     res.write(data);     return res.end();   }); }).listen(8080); |
| **4.4 Events module** |
| - Tất cả hành động trên máy tính của bạn đều là một event vd như: 1 kết nối được tạo, một file được mở  - Module Event giúp bạn **tạo** , **chạy** và **lắng nghe** được sự kiện  - Trong NodeJS tất cả sự kiện đều thuộc về đối tượng **EventEmitter**  **const events = require( ‘events’ );**  **const eventEmitter = new events.EventEmitter( )**     |  |  | | --- | --- | | **Method & Props** | **Description** | | **.addListener()** | Thêm Listener được xác định | | **.defaultMaxListeners** | Cài đặt tối đa Listener cho 1 sự kiện , mặc định là 10 | | **.emit( )** | Gọi tất cả Listener bằng tên chỉ định | | **.eventNames( )** | Trả về một mảng chứa tên các Event được sử dụng | | **.getMaxListeners( )** | Trả về số lượng tối đa Listener cho 1 Event | | **.listenerCount( )** | Trả về số lượng Listener với tên được chỉ định | | **.listeners( )** | Trả về mảng chứa các Listener với tên được chỉ định | | **.on( )** | Thêm Listener | | **.once( )** | Thêm 1 Listener sử dụng 1 lần, sau khi sử dụng xong sẽ biến mất | | **.prependListener( )** |  | | **.prependOnceListener( )** |  | | **.removeAllListeners( )** | Xoá tất cả Listener với tên được chỉ định, hoặc xoá hết | | **.removeListener()** | Xoá 1 Listener được chỉ định | | **.setMaxListeners( )** | giống .defaultMaxListeners( ) |   **vd:**  **const events = require('events'); const eventEmitter = new events.EventEmitter();** //Create an event handler: **var myEventHandler = function () {   console.log('I hear a scream!'); }** //Assign the event handler to an event: **eventEmitter.on('scream', myEventHandler);** //Fire the 'scream' event: **eventEmitter.emit('scream');** |
| **4.5 Path Module - Xử lý các đường dẫn** |
| Thông thườngc các đường dẫn tới các file trong JS sẽ được viết bằng tay, nhưng trong NodeJS ta có PATH module sẽ hỗ trợ lấy và xử lý các đường dẫn tới file được dễ dàng hơn.  Cài đặt : **const path = require(‘ path ‘)**   |  |  | | --- | --- | | **Method & Props** | **Description** | | **path.basename( )** | Lấy tên của file đang đứng từ PATH nhập vào | | **path.dirname( )** | Lấy đường dẫn tới forder của file | | **path.extname( )** | Lấy tên đuôi của file | | **path.format( { dir: … , base: …} )** | Ghép các phần của PATH lại để trở thành PATH của file | | **path.join(** *path1***,** *path2***)** | Ghép các dữ liệu thành 1 PATH duy nhát | | **path.parse( )** | Trả lại 1 Object chứa dữ liệu của PATH | |
| **5. NodeJs npm** |
| npm - Node Packaged Manager : Quản lý các module thư viện bên ngoài |
| **6. npm - MONGO DB** |
| - Sử dụng MongoDB làm database  - Dụng module : mongodb để kết nối được với Mongodb  Cài đặt vào Nodejs: **npm i mongodb**  Sử dụng: **const mongodb = require( ‘mongodb);**  **const MongoClient = mongodb.MongoClient;**   * **CREATE DATABASE**   - Để tạo một database trong MongoDB , bắt đầu bằng việc tạo một **MongoClient** object , chỉ định một URL để kết nối với nó gồm địa chỉ ip và tên database bạn muốn tạo  vd: const url = "mongodb://localhost:27017/mydb";  mongodb://localhost:27017 : địa chỉ mặc định của mongodb  /mydb : tên của database bạn muốn kết nối    - Dùng phương thức connect( ) của MongoClient để kết nối NodeJS với database ,  nếu chưa có nó sẽ tự tạo một database  const  **MongoClient** = require('mongodb').MongoClient;  const  **url** = "mongodb://localhost:27017/mydb";   **MongoClient**.connect(**url**, function(err, db) {     if (err) throw err;   console.log("Database created!");   db.close(); });  **Khi kết nối thành công ta sẽ nhận được database - db như là một đối số để tương tác**  *--> Tuy nhiên MongoDB sẽ tạo database khi nó có nội dung bên trong, vì ở trên chúng ta chưa tạo nội dung nên khi db.close( ) sẽ chưa có database nào được tạo ra hết.*   * **CREATE COLLECTION**   - Sau khi kết nối được mongodb và nhận được database thông qua Object db  Dùng phương thức **.db( )** để tạo một Database mới mới:  **const myDatabase** = **db.db(‘ *myDatabase* ’ );**  Tạo một Collection trong myDatabase:  **const myCollection = myDatabase.createCollection(‘ *myCollection* ’);**   * **INSERT**   Dùng .insertOne( ) và .insertMany( ) cho myCollection để thêm dữ liệu vào  vd: oneObject = { name: “Binh” , age : “28” };  **myCollection.insertOne(** oneObject **);**  multiObject = [{ name: “Binh” , age : “28” } , { name: “Ha” , age : “23” } ];  **myCollection.insertMany(**multiObject**);**  *Với mỗi dữ liệu được thêm vào Collection , mongoDB sẽ tự thêm* ***\_id*** *cho nó, bạn cũng có thể thêm trước giá trị này bằng cách khai trực tiếp tại mỗi dữ liệu.*   * **FIND**   Dùng .findOne( { } ) và .find( { } ) cho myCollection để Tìm dữ liệu  Truyền đối số là **Query Object** vào trong các method trên để tìm  **Query Object** : Đối tượng truy vấn  có thể là một đối tượng cụ thể : { name: “Binh” } ;  Hoặc có thể là một regular extension: { name: /^B/ig }  vd:  **myCollection.findOne( { \_id: 102520023 } );**  Ngoài ra , có thể truyền projection như một đối số thứ 2 cho 2 phương thức ở trên để  xác định các trường dữ liệu trả về  **myCollection.findOne( { } , { projection: {**  **\_id : 0 ,** // không lấy \_id ở dữ liệu trả về  **name: 1,**  **age: 1,**  **} );**   * **SORT**   - Dùng **.sort( )** để sắp xếp các đối tượng trả về  Phương thức sort( ) nhận tham số là một Object gồm KEY và cách sắp xếp  vd: **myCollection.**sort( { name : 1 } );  // Trả về danh sách dựa vào KEY = name tăng dần  **1** hoặc **“asc”** : ascending - Tăng dần  **-1** hoặc **“des”**  : descending - Giảm dần   * **DELETE**   - Dùng **.deleteOne( {} )** và **.deleteMany( {} )** để xoá đối tượng   * **DROP COLLECTION**   - Dùng **.drop( )** để xoá Collection trong myDatabase  vd: myCollection.drop( )  \*\* Hoặc dùng phương thức **.dropCollection( )** của db để xoá Collection  **myDatabase**.dropCollection( “*myCollection*” )   * **UPDATE** * **myCollection.updateOne({ myQuery} ,{ $set: { key: “newValue” } } )** * **myCollection.updateMany( { myQuery},{ $set: { key: “newValue” } } )** * **LIMIT**   - Dùng **.limit( )** để giới hạn đối tượng trả về  vd: myCollection.find( ).limit(2) ; 🡺 Chỉ trả về 2 đối tượng   * **JOIN**   - MongoDB không phải là một cơ sở dữ liệu dạng quan hệ như SQL , nhưng bạn vẫn có thể nhóm các dữ liệu có liên quan với nhau bằng cách sau:  vd: Trong **myDatabase** có 2 Collection là **oders** và **product** như sau:  **orders:**  **[**  **{ \_id: 1, product\_id: 154, status: 1 }**  **]**  **products:**  **[   { \_id: 154, name: 'Chocolate Heaven' },   { \_id: 155, name: 'Tasty Lemons' },   { \_id: 156, name: 'Vanilla Dreams' } ]**    **Tại collection orders dựa vào product\_id tìm sản phẩm tương ứng với \_id bên collection products ta làm như sau:**  **db.collection('orders').aggregate([     { $lookup:         {          from: 'products'**, // tên collection tìm kiếm **localField: 'product\_id',** // KEY để tìm **foreignField: '\_id',** // KEY để so sánh **as: 'orderdetails'** // tạo KEY mới để trả về kết quả **}      }    ])** |
| **6.1 npm Mongoose - Driver cho MongoDB** |
| Mongoose giúp bạn làm việc với MongoDB được đơn giản hơn , không cần quá nhiều câu kệnh phức tạp. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |