NodeJS Server

- **Node.JS:**

- A server-side platform which is built on Google Chrome’s V8-JavaScript Engine which was initially built and developed by Mr. Rayn Dal in the year 2009.
- Using Node.js we could able to build faster and scalable network applications.
- Node.js internally uses event driven and non-blocking I/O mechanism (or) model which makes it light weight and efficient for data intensive real time applications which run on distributive system/devices.
- Node.js is open source and cross-platform runtime environment using which we could develop server side and networking-based applications.
- JavaScript is programming language using which we could able to add instructions within Node.js.
- Node.js comes with rich library of various JavaScript modules which simplifies the development of the web applications using Node.js.
- Node.js is a combination of Run-Time environment and JavaScript library.

- **Features of Node.JS used for developing web applications:**
- **Asynchronous and Event Driven:**

All API’s of Node.js are asynchronous and non-blocking which means it is a server which doesn’t waits for any API to return the data, the server moves to the next API after calling it, a notification mechanism of events of Node.js helps server to get response from previous API calls.

- **Faster in Processing:**

As the Node.js is built on Google Chrome’s V8-JS Engine, it is very faster in executing instructions.

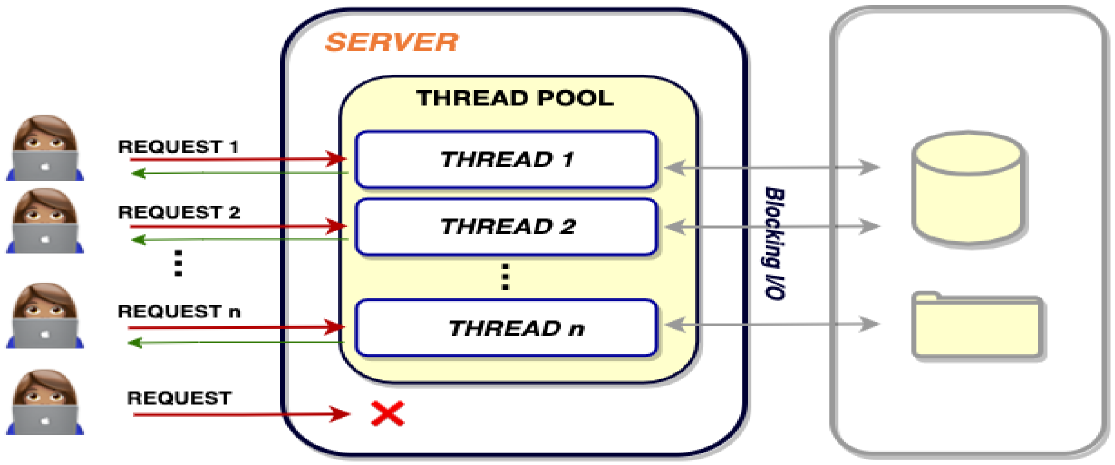
- **Single Threaded but Highly Scalable:**

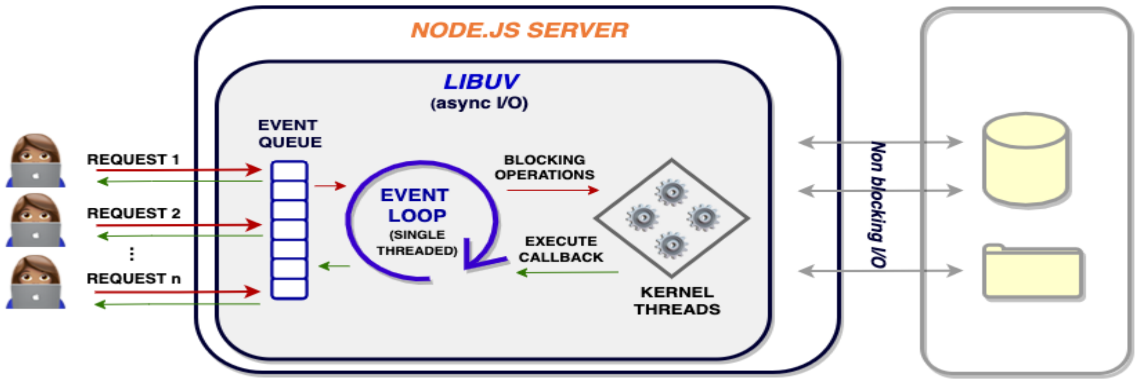
Internally Node.js uses a single threaded model with event looping, the event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable.

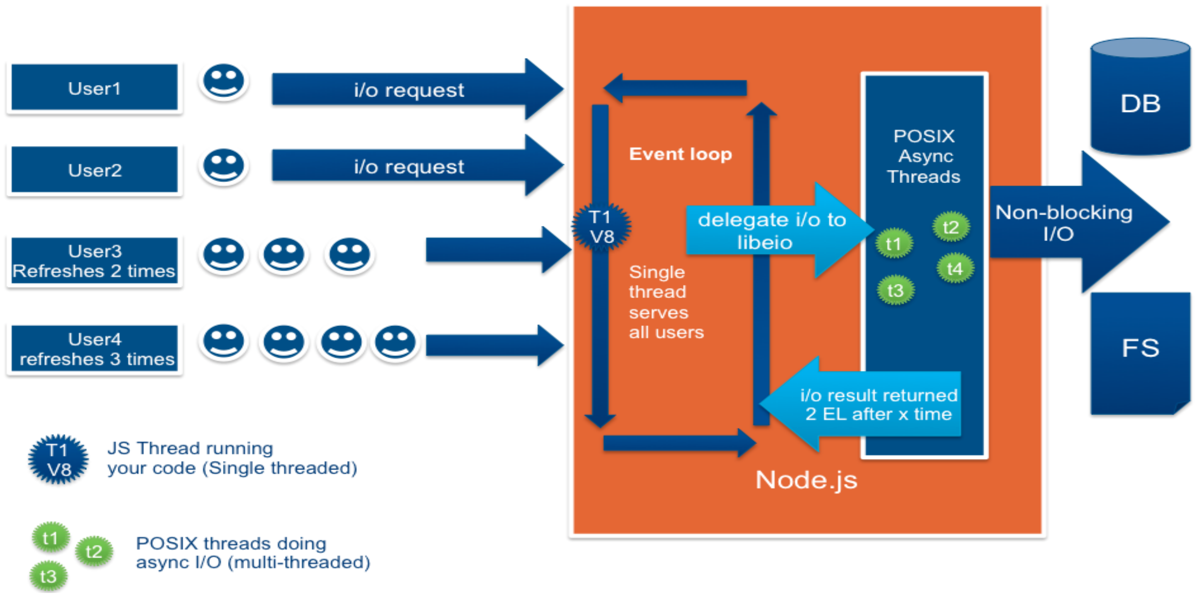
➢ **No Buffering:**

Node.js applications never buffer any data, these applications output data in chunks.

➢ **Architecture of Node.JS:**







➢ <u>Node Package Manager (NPM):</u>

- NPM is an online code repository publishing of open source Node.js code, it is also a command line utility for interacting with NPM repository for package installing and version management.
- It is through NPM we could work with installation (or) uninstallation (or) upgradation node modules. Through NPM we could even start or stop node server.

➢ <u>Node.js Module:</u>

A module in node.js is a simpler or complex functionality organized in single or multiple JavaScript files which can be reserved throughout the node.js application. Each module in node.js has its own context so that it doesn't interfere with other modules or doesn't interrupt the global scope.

It is through **NPM** we install (or) uninstall version manager of any module can be done.

Following are the basic steps which we need to create a **HTTP Node Server,**

Step-1: Download Node HTTP Module through NPM,

```
npm install http   //Downloads a HTTP Module.
```

Step-2: Create a JavaScript file and include **HTTP Module** through require method.

```
var http = require("http");    //including a Node Module.
```

Step-3: Use **"createServer"** method under HTTP, which takes a callback method as a parameter with default request and response objects.

```
http.createServer ( (req,res) => {
        …..   //Code to handle request and response
});
```

Step-4: Using response object we can specify the type of content being returned from the server.

```
res.writeHead(200,{'content-type': 'text/HTML'});
```

Using **writeHead** method we can specify the response status code along with the response text type.

Step-5: using **res.write()** method we can write the content to be responded from the server.

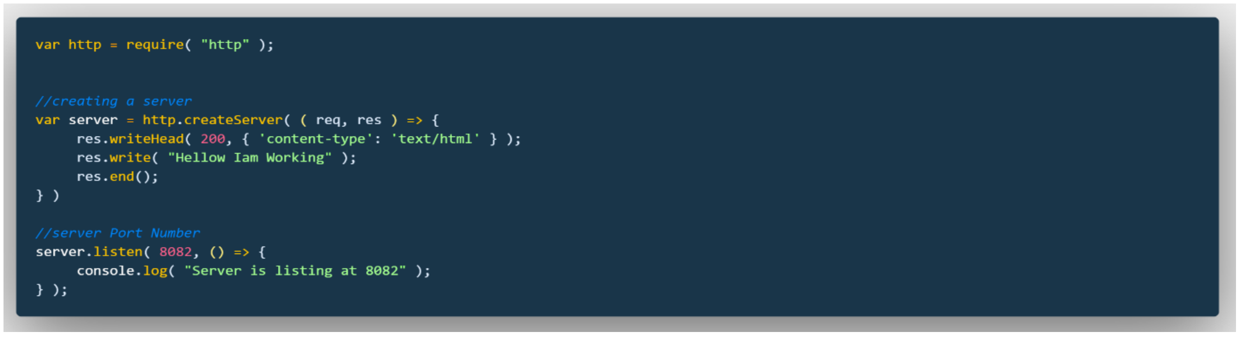
res.write("<Content>");

Step-6: res.end() is an predefined method through which we can send request using user response.

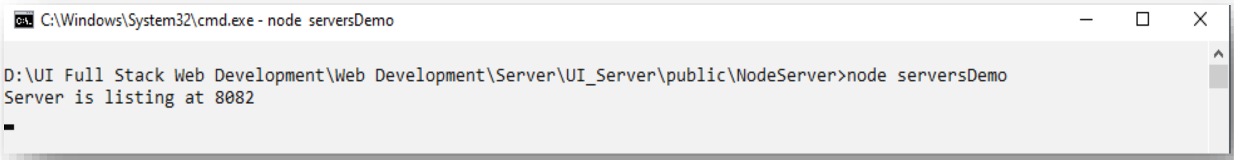
res.end();

Step-7: Make the server to listen at a particular port number

server.listen(8080);

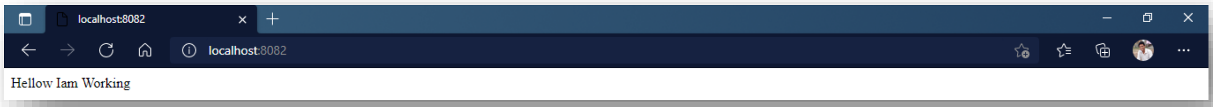


```
var http = require( "http" );


//creating a server
var server = http.createServer( ( req, res ) => {
     res.writeHead( 200, { 'content-type': 'text/html' } );
     res.write( "Hellow Iam Working" );
     res.end();
} )

//server Port Number
server.listen( 8082, () => {
     console.log( "Server is listing at 8082" );
} );
```



C:\Windows\System32\cmd.exe - node serversDemo

```
D:\UI Full Stack Web Development\Web Development\Server\UI_Server\public\NodeServer>node serversDemo
Server is listing at 8082
```



Hellow Iam Working

➢ **<u>Performing Node Operations in Node.js through fs module:</u>**

"fs" is a predefined node module through which we could able to perform read, write and append operations or any file. Following are the steps to be followed while making use of FS module,

Step-1: Download fs Module through NPM

npm i fs

Step-2: Include and create a reference for FS Module,

var fs = require(“fs”);

Step-3: Use the following predefined methods under FS Module through which we could able to do any file operations in files.

Fs.readFile(“<filename>”, (err,data) => {

//err objects holds the error if any while reading file

//Data holds the actual data of file.

});

Fs.appenFile(“<filename>”, “<Content to be written>”,(err,data) => {

//err objects holds the error if any while reading file

//Data holds the actual data of file.

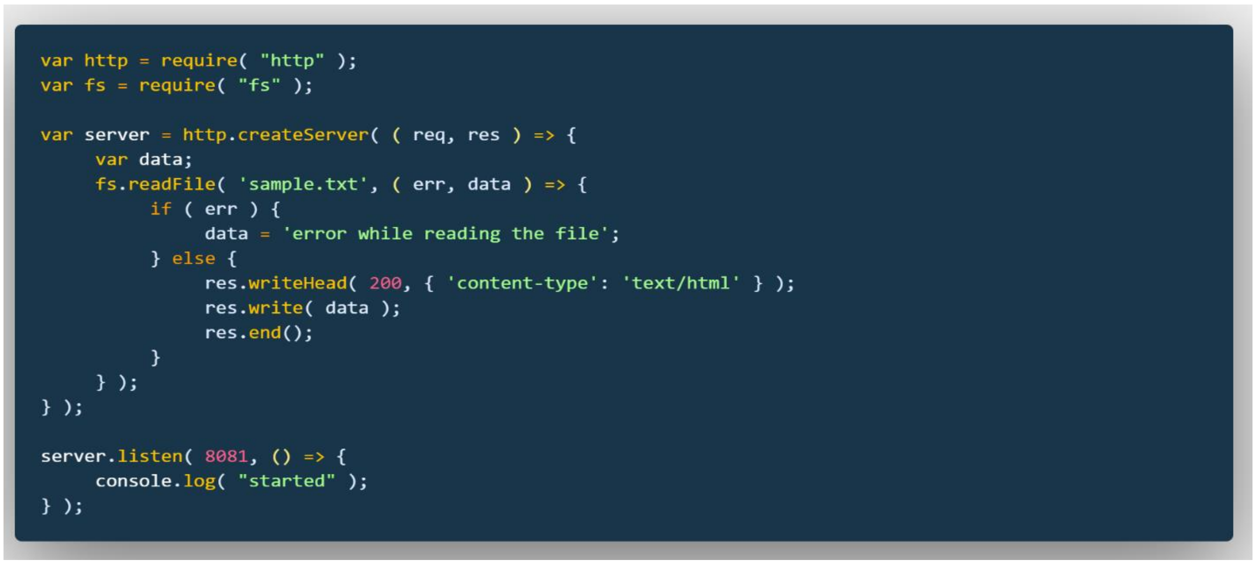
});

Fs.rename(“filename”,”newfileName”,(err) => {

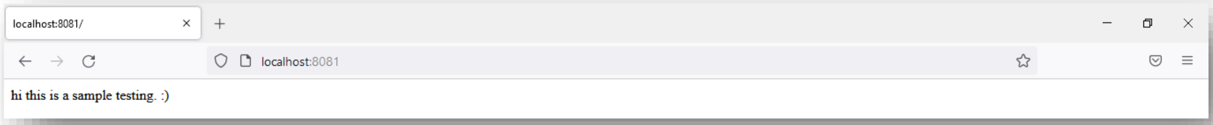
Err……

});

Ex:



```
var http = require( "http" );
var fs = require( "fs" );

var server = http.createServer( ( req, res ) => {
     var data;
     fs.readFile( 'sample.txt', ( err, data ) => {
          if ( err ) {
               data = 'error while reading the file';
          } else {
               res.writeHead( 200, { 'content-type': 'text/html' } );
               res.write( data );
               res.end();
          }
     } );
} );

server.listen( 8081, () => {
     console.log( "started" );
} );
```



hi this is a sample testing. :)

➢ Node Express.JS Framework:

Express.js is a web application framework for node.js through which we could able to create flexible node.js web applications. It is one of the most popular web frameworks provides following mechanisms,

- It writes handlers with requests with different http at different URL paths.
- It integrates with view rendering engines in order to generate response by inserting data into the template.
- It sets common web applications settings like the port to use for connecting, location of templates that are used etc.…
- Express itself is fairly minimalist developers are created compatible middleware packages to address utmost any web development problems.
- There are libraries to work with cookies, sessions, user logins, URL params, post data, security headers and so on.
- Express was released on 2010 and current latest version is 4.17.1*.

➢ Folder structure got created through express generator:

- **Node_modules:**
  Under which all the downloaded node modules of the current app will be stored. This folder name cannot be renamed.
- **Package.json file:**

  It holds a json object with all the configuration information of the current module.

  i. It holds list of dependencies of node module.
  ii. Name and Version number of the current application.
  iii. Author name and contact details.
  iv. Starting point of the current application (node start/server start).
- **Bin Folder:**
  A predefined folder holds a predefined 'www' holds the set of instructions through which we could create http server, making the server to listen a default port number handling the errors if any.

    - **Bin folder and www file** name can be renamed to a custom name, corresponding changes to be done at package.json

- **Routes Folder:**
  Holds set of JavaScript files, where each JavaScript file represents a single **web service**. Any number of web services can be added with in node server, all the **web services JS** files should be placed under the **Routes** folder.
    - Index.js, users.js are the default services available under routes folder.
    - Once the services are added under routes folder, corresponding route mapping or URL mapping has to be done under app.js file.
- **Steps to create a web service under a node express server:**
  Step-1: Create an external JS file under Routes Folder.
  Step-2: under the JS file, create an instance of express module.

  Var express = require('express');

  Step-3: create a router instance through a router class under express module.

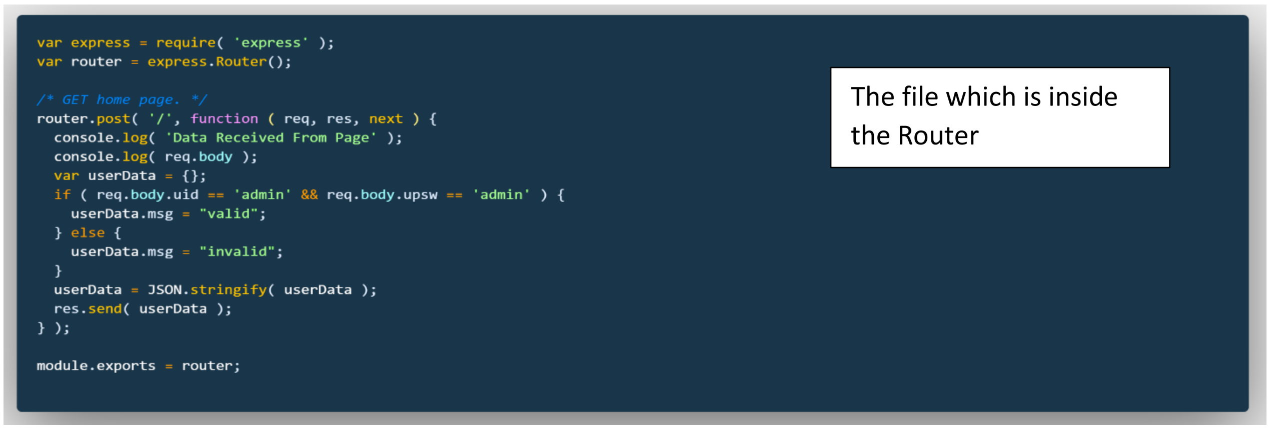
  Var router = express.Router();

  Step-4: using the router reference, through GET/POST method add the business logic of the corresponding webservice under call back method.

```
router.get("/", (req,res) => {
        .....   //Logic
        res.send();
});
        (or)
router.get("/", (req,res) => {
        .....   //Logic
        res.send();
});
```

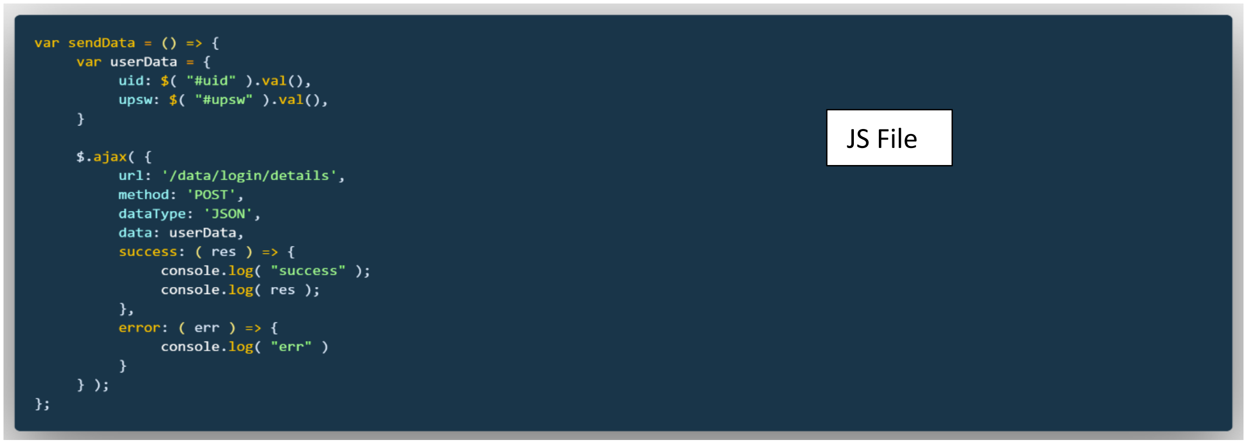
  Step-5: under the app.js file specify the router path through which web services can be accessible.

Ex:



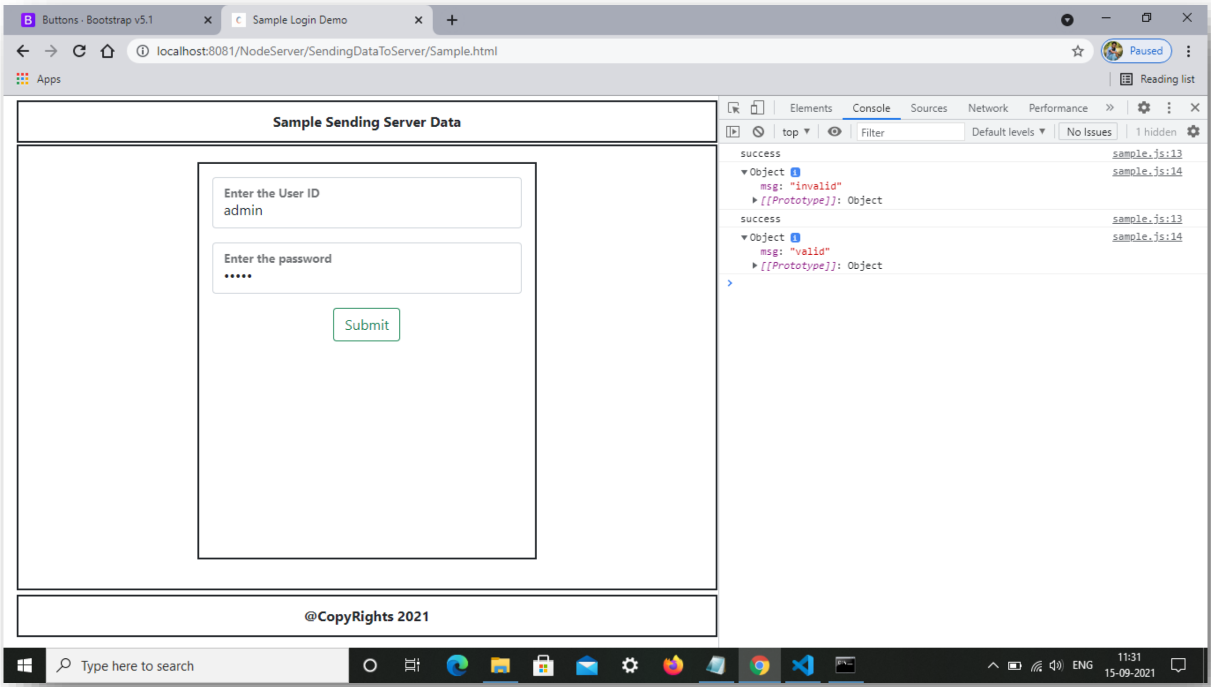
```
var express = require( 'express' );
var router = express.Router();

/* GET home page. */
router.post( '/', function ( req, res, next ) {
  console.log( 'Data Received From Page' );
  console.log( req.body );
  var userData = {};
  if ( req.body.uid == 'admin' && req.body.upsw == 'admin' ) {
    userData.msg = "valid";
  } else {
    userData.msg = "invalid";
  }
  userData = JSON.stringify( userData );
  res.send( userData );
} );

module.exports = router;
```

The file which is inside the Router



```
var sendData = () => {
     var userData = {
          uid: $( "#uid" ).val(),
          upsw: $( "#upsw" ).val(),
     }

     $.ajax( {
          url: '/data/login/details',
          method: 'POST',
          dataType: 'JSON',
          data: userData,
          success: ( res ) => {
               console.log( "success" );
               console.log( res );
          },
          error: ( err ) => {
               console.log( "err" )
          }
     } );
};
```

JS File



localhost:8081/NodeServer/SendingDataToServer/Sample.html

Sample Sending Server Data

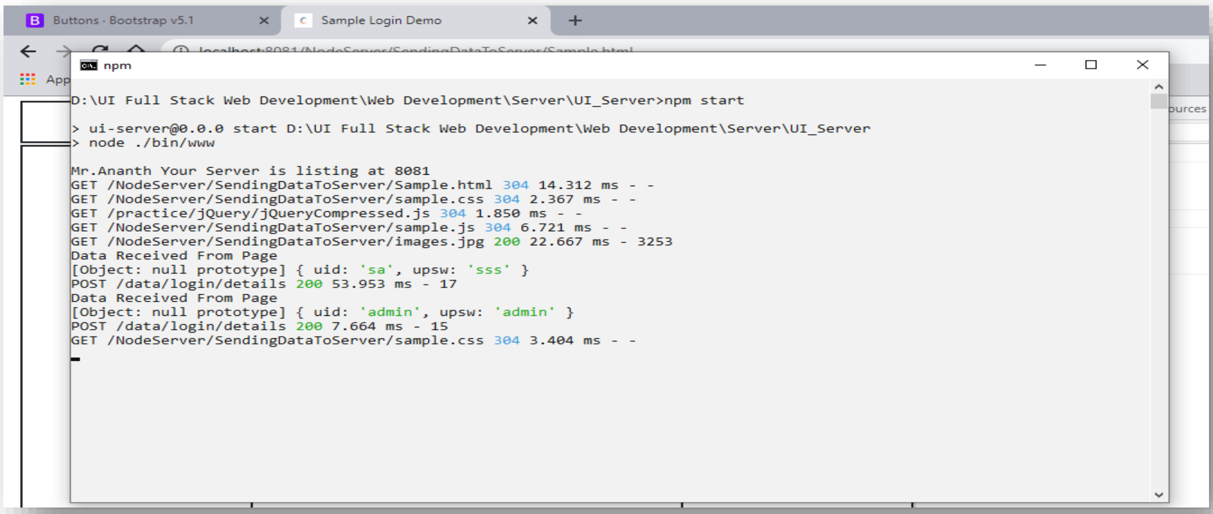
Enter the User ID: admin

Enter the password: •••••

Submit

@CopyRights 2021

```
success                                   sample.js:13
▼Object                                   sample.js:14
   msg: "invalid"
  ▶[[Prototype]]: Object
success                                   sample.js:13
▼Object                                   sample.js:14
   msg: "valid"
  ▶[[Prototype]]: Object
```



```
D:\UI Full Stack Web Development\Web Development\Server\UI_Server>npm start

> ui-server@0.0.0 start D:\UI Full Stack Web Development\Web Development\Server\UI_Server
> node ./bin/www

Mr.Ananth Your Server is listing at 8081
GET /NodeServer/SendingDataToServer/Sample.html 304 14.312 ms - -
GET /NodeServer/SendingDataToServer/sample.css 304 2.367 ms - -
GET /practice/jQuery/jQueryCompressed.js 304 1.850 ms - -
GET /NodeServer/SendingDataToServer/sample.js 304 6.721 ms - -
GET /NodeServer/SendingDataToServer/images.jpg 200 22.667 ms - 3253
Data Received From Page
[Object: null prototype] { uid: 'sa', upsw: 'sss' }
POST /data/login/details 200 53.953 ms - 17
Data Received From Page
[Object: null prototype] { uid: 'admin', upsw: 'admin' }
POST /data/login/details 200 7.664 ms - 15
GET /NodeServer/SendingDataToServer/sample.css 304 3.404 ms - -
```

Mongo DB

MongoDB:

MongoDB is a cross-platform document-oriented database program classified as a **NOSQL** database program. MongoDB uses **JSON** like documents with optional schemas. MongoDB is developed by MongoDB.INC and licensed under the server-side public license. MongoDB is a document data base with the scalability and flexibility that you want with the querying and indexing that you need.

Key Features of MongoDB:

- **Supports ad hoc queries:**

  In MongoDB you can search by field, range query and it also supports regular expression searches.

- **Indexing:**

  You can index any field in a document.

- **Replication:**

  MongoDB supports slave replication. A master can perform read and write and a slave copies data from the master and can only be used for reads or backup (not writes).

- **Duplication of data:**

  MongoDB can run over multiple servers. The data is duplicated to keep the system up and keep its running condition in case of hardware failure.

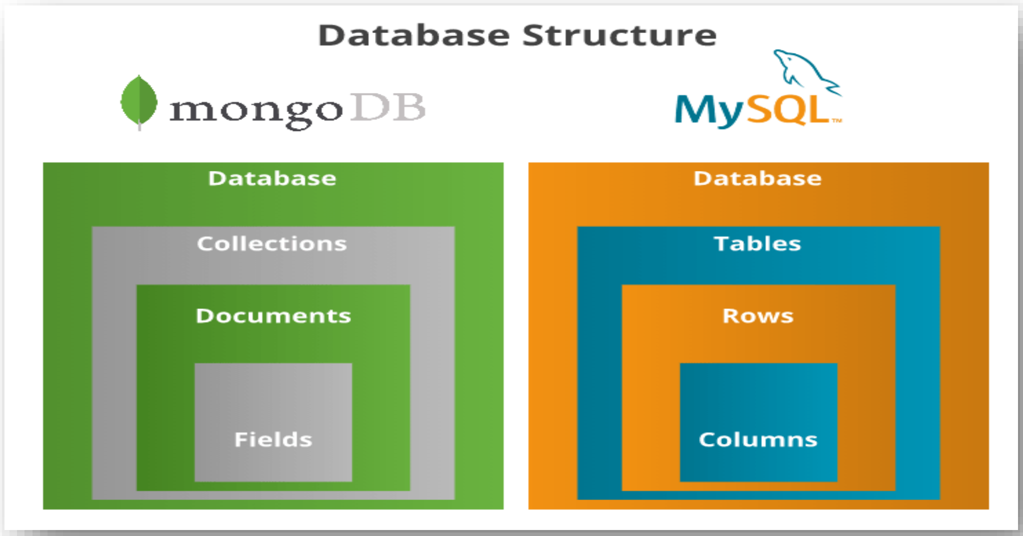
- **Load Balancing:**

  It has an automatic load balancing configuration because of the data placed in shards.

- Supports Map reduce and aggregation tools.
- Uses JavaScript instead of procedures.
- It is a schema less DB written in **C++**.
- Provides high performance.
- Stores files of any size easily without complicating your stack.

- Easy to administer in case of failures.
- It also supports:
  - JSON data model with dynamic schemas.
  - Auto-Sharing for horizontal scalability.
  - Built in replication for high availability.
  - Now a days many companies using MongoDB to create new types of applications, improve performance and availability.

➢ **Difference between MongoDB Structure and SQL Structure:**



	SQL	NoSQL
Database Type	Relational Databases	Non-relational Databases / Distributed Databases
Structure	Table-based	• Key-value pairs • Document-based • Graph databases • Wide-column stores
Scalability	Designed for scaling up vertically by upgrading one expensive custom-built hardware	Designed for scaling out horizontally by using shards to distribute load across multiple commodity (inexpensive) hardware
Strength	• Great for highly structured data and don't anticipate changes to the database structure • Working with complex queries and reports	• Pairs well with fast paced, agile development teams • Data consistency and integrity is not top priority • Expecting high transaction load

➢ <u>Commands that can be run on a Mongo Shell:</u>

- **db –** returns the name of current database been used.
- **use<db name> -** switches to specific DB.
- **db.help( ) –** throws help commands.
- **Show dbs –** throws list of all database under MongoDB.
- **Show users –** returns the list of users.
- **db.getCollectionNames( ) –** shows list of collections within a DB.
- **db.createCollection("<Collection Name>") –** used to create a collection.
  Ex: db.createCollection("Login_Details")
- **db.collection.insert({document}) –** used to insert a document to a collection.
- **db.collection.insertmany([<document1>,<document2>,…] –** used to insert many document to a single collection.

Ex:

db.Login_Details.insertMany([{_id:'adminLogin',uid:'admin',upsw:'admin'},{_id:'studentLogin',uid:'student',upsw:'student'},{_id:'parentLogin',uid:'parent',upsw:'parent'}])

- **db.collection.find({key:'value'}) –** used to find a specific document from a collection.
- **db.collection.update(<query>,<updatedDocument>).**
- **db.collectionName.remove() –** to delete a collection from database.
- **Use -** used to create a database or to switch to an existing data base.

➢ <u>Creating connection to MongoDB through Node.js:</u>

Step-1: Download and install MongoDB Node Module.

npm install mongodb - - save

Here –save is used to save in package.json

Step-2: Include MongoDB and create instance for mongo client.

Ex:var mongodb = require("mongodb").MongoClient;

Step-3: Create URL with MongoDB protocol, server name and port number it is running.

Ex: var URL = 'mongodb://localhost:27017/';

Step-4: create a connection to MongoDB through mongo client by passing the MongoDB URL.

```
mongoClient.connect( mongodbURL, ( err, client ) => {
     //Client object through which we get connected to the specific DataBase
});
```

Step-5: Through client object we could able to connect to the specified database under the MongoDB.

Ex: var db = client.db(“DataBaseName”);

Step-6: Through db object we could get reference to required collection using db.collection( ) method.

Ex: var collection = db.collection(“Collection Name”);

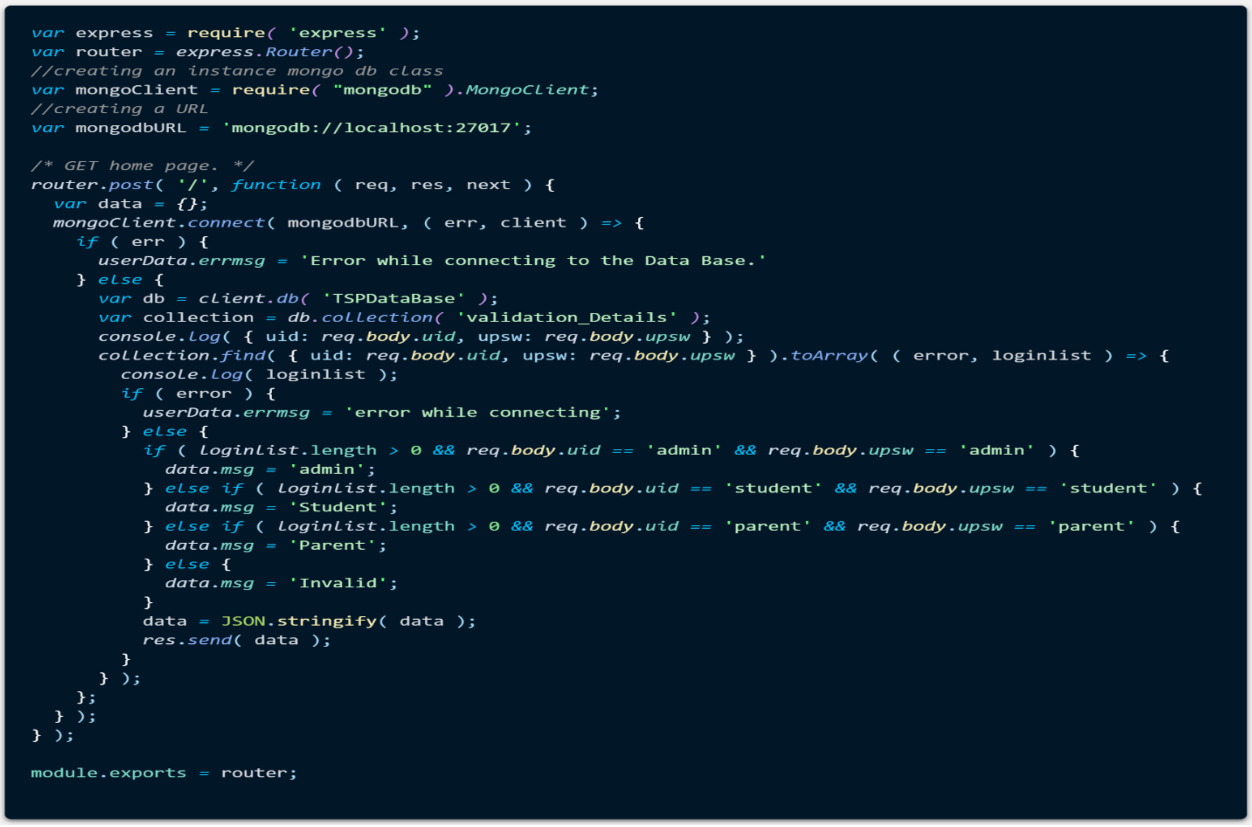
Step-7: add the required command.

Ex: collection.find(‘’).toArray((error,list)=> {

… …

});

Ex:



```
var express = require( 'express' );
var router = express.Router();
//creating an instance mongo db class
var mongoClient = require( "mongodb" ).MongoClient;
//creating a URL
var mongodbURL = 'mongodb://localhost:27017';

/* GET home page. */
router.post( '/', function ( req, res, next ) {
  var data = {};
  mongoClient.connect( mongodbURL, ( err, client ) => {
    if ( err ) {
      userData.errmsg = 'Error while connecting to the Data Base.'
    } else {
      var db = client.db( 'TSPDataBase' );
      var collection = db.collection( 'validation_Details' );
      console.log( { uid: req.body.uid, upsw: req.body.upsw } );
      collection.find( { uid: req.body.uid, upsw: req.body.upsw } ).toArray( ( error, loginlist ) => {
        console.log( loginlist );
        if ( error ) {
          userData.errmsg = 'error while connecting';
        } else {
          if ( loginList.length > 0 && req.body.uid == 'admin' && req.body.upsw == 'admin' ) {
            data.msg = 'admin';
          } else if ( loginList.length > 0 && req.body.uid == 'student' && req.body.upsw == 'student' ) {
            data.msg = 'Student';
          } else if ( loginList.length > 0 && req.body.uid == 'parent' && req.body.upsw == 'parent' ) {
            data.msg = 'Parent';
          } else {
            data.msg = 'Invalid';
          }
          data = JSON.stringify( data );
          res.send( data );
        }
      } );
    };
  } );
} );

module.exports = router;
```

➢