**Nodejs**

* Node.js is a very powerful JavaScript-based framework/platform built on Google Chrome's JavaScript V8 Engine
* Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications
* **Event loop :** one of the key concepts “Node” brings from the browser to javascript on the server.
  + In the browser the event loop is constantly listening for dom events (click, key presses etc), similarly Node event loop is constantly listening for events on server side.
  + Node does not wait or pause for any request to complete. It continues to react to events as they arrive.
  + Example for the above behavior (non blocking ) could be a web application which fetches data from database.
  + The application raises an event when an http request is received, this event generates a query to fetch data from database.
  + Once node receives data from the database, an http response is formulated and sent to the called.
  + While it is waiting for the data from the database Node is not blocked and can still handle the incoming request.
  + This non blocking approach is fundamental to Node and differentiates it from the more traditional server side programming model that requires to manage multiple threads to achieve this type of concurrency.
* Cross platform runtime environment for developing server side applications
* Provides rich library of javascript modules
* All the APIs in Node.js library are asynchronous and is non blocking.
  + It never waits for an api to return data, and moves to the next api call and it uses some mechanism **(callback functions)** to return the response of the previous call.
* Uses single threaded model with event looping which makes the server to respond in a non blocking way, which again makes the server highly scalable.
* Download and install nodejs
  + <https://nodejs.org/en/>
* Goto eclipse -> marketplace and search for node plugin and install.
* Goto project explorer, right click and create a nodejs project
* A nodejs application has three main parts
  + **require directive :** to load required modules.
  + **createServer :**  to setup a server which listens to clients request (like apache)
  + **read and return response :** server created using **createServer** reads all requests and return responses for each

**Modules & require()**

* Modules are the way by which you can bring external functionality to your Node application
* require() function loads a module and assigns it to a variable for your application to use.
* When you issue a require command, node will look for module.js file inside current directory/node\_modules directory
* A module can export specific variables, functions, objects
  + Can even import just one function or variable needed from the large module, just by specifying it after the function call.
* Common built-in Node modules include
  + **fs** : for accessing file system
  + **http** : for creating and responding to http request.
  + **os** : for accessing information about the underlying operating system where the application is running.
* Example on how to use the require function include Node modules in your application.
  + var os = require('os');  
      
    console.log(os);  
    console.log('Free mem = '+os.freemem());  
    console.log('Total mem = '+os.totalmem());  
    console.log('Os type = '+os.type());  
    console.log('CPUS = '+os.cpus());  
    console.log('homedir = '+os.homedir());  
    console.log('userinfo = '+os.userInfo());
* Another use of require function is to access functionality located in other files in your project.
* In Node module system each of your javascript file is a module and can expose functionality to be required by other files.
* This way it helps to modularize your code. (easy to develop and maintain your code base)  
  + Files within the same directory
    - var data = require(‘./data’);
  + Files from subdirectory
    - var data = require(‘./data/data2’);
  + Files from other directory
    - var data = require(‘../other/data’);
  + **Note : “ ./ “** prefix is required, even the files are from the same directory and the **“ .js “** is not included
  + Requiring just one variable from the module
    - var data = require(‘./data’).onevariable;
  + Creating a module and making values ready to export using **module.exports**
    - var cntr = 0;  
        
      var inc = function() {  
       return ++cntr;  
      };  
        
      module.exports.inc = inc;  
        
      module.exports.user = {  
       "name":"Arun",  
       "loc":"United States",  
       "display\_details": function() {  
       return this.name+", "+this.loc;  
       }   
      };
    - var data = require('./data');  
        
      console.log(data.inc());  
        
      console.log(data.user.name);  
      console.log(data.user.loc);  
      console.log(data.user.display\_details());
* **Creating callbacks**  
  + Module file : creating a function and exporting it.  
    function filter\_arr(arr, cb) {  
     for(var i=0; i<arr.length; i++) {  
     cb(arr[i]);  
     }  
    }  
      
    module.exports = { sayHello, userObj, user\_arr, filter\_arr}
  + var mod1 = require('mod1.js');  
    mod1.filter\_arr(['Arungopan', 'Nandakumar'], function(name) {  
     console.log('Name = '+name);  
    });
* **Creating promises**
  + function promise\_fn() {  
     return new Promise(function(resolve, reject) {  
     setTimeout(function() {  
     reject('Hello everyone, from promise function!!!');  
     }, 3000);  
     });  
    }
  + mod1.promise\_fn().then(function(success) {  
     console.log('Success = '+success);  
    },  
    function(err) {  
     console.log('Err = '+err);  
    });
* Installing 3rd party modules using NPM (node package manager)
  + npm install moduleName;
  + npm install request;
    - http client module
  + Before running the command in terminal, make sure you are on the right directory.
  + Then run the above command **npm install request,** which will install the request module to your directory.
  + var request = require('request');  
      
    request('http://www.google.com', function(err, response, body) {  
     if(err) {  
     console.log(err)  
     } else {  
     console.log(body);  
     }  
    });

**Events and Streams**

* Callback was the method we used to implement asynchronous code.
* Node provides one more way to implement the same using **Events.**
* var data = getData(5);
* **data** is an instance of EventEmitter class
* The result object has an **on()** function, using which each events can be accessed.
* data.on('process\_data', function(cntr) {  
   console.log('Data '+cntr+' received');  
  });
* Using the **on ()** function **process\_data** event is accessed.
* data.on('kill\_process', function(cntr) {  
   console.log('Final data '+cntr+' received and the process ended');  
  });
* kill\_process event called when there is no more data to be processed.
* **callback vs events**
  + **Callback**
    - Has one request and one reply
    - No results until all the results are received
    - Either error or result
  + **Events**
    - **.on()** can be invoked repeatedly to provide multiple function to invoke on each event.
    - Acts on result as they arrive
    - Partial result before error
* **EventEmitter class**
  + Provided by Node for building these event driven interfaces
  + The subscribers of the event call **.on()** function to subscribe to an event
  + Events are published using the **emit()** function
    - **emitter.emit(‘eventName’, arguments’);**
  + So an emitter event has two main features
    - **Emitting named events**
    - **Registering listener functions**
  + var EventEmitter = require('events').EventEmitter;  
      
    var getData = function(cnt) {  
     var e = new EventEmitter();  
     setTimeout(function() {  
     var counter = 0;  
     e.emit('event\_start');  
     var t = setInterval(function() {  
     e.emit('process\_data', ++counter);  
     if(counter==cnt) {  
     e.emit('kill\_process', counter);  
     clearInterval(t);  
     }  
     }, 1000);  
     }, 3000);  
     return e;  
    };  
      
    var data = getData(5);  
      
    data.on('event\_start', function() {  
     console.log('Request started');  
    });  
      
    data.on('process\_data', function(cntr) {  
     console.log('Data '+cntr+' received');  
    });  
      
    data.on('kill\_process', function(cntr) {  
     console.log('Final data '+cntr+' received and the process ended');  
    });

**Streams**

* Streams are objects that allows you read data from a source or write data to a destination in continuously.
* Stream extends EventEmitter class
* Each stream is either an instance of
  + Readable stream
  + Writable stream
  + Both
* A readable stream can be piped to writable stream
* Readable stream has a series of events that are emitted when new data arrives or when there is no more data etc.
  + **event:’data’**
  + **event:’end’**
  + **event:’error’**
  + **event:’close’**
  + Series of functions to pause, resume and destroy the stream, along with the pipe function
  + **pause()**
  + **resume()**
  + **destroy()**
  + **pipe()**
* Writable stream also has a series of events such as drain when it safe to write, pipe when this stream is passed to a readable stream pipe.
  + **event:’drain’**
  + **event:’error’**
  + **event:’close’**
  + **event:’pipe’**
  + Functions to write data to the stream, terminate etc.
  + **write()**
  + **end()**
  + **destroy()**
  + **destroySoon()**

**Handling files**

* Node implements file I/O using **fs module** (file system module), which can be imported using **require(‘fs’)**
  + var fs = require("fs")
* **Synchronous / Asynchronous**
  + var fs = require('fs');  
    console.log('before file read');  
    fs.readFile('files.txt', 'utf-8', function(error, data) {  
     if(error) {  
     console.log(error);  
     } else {  
     console.log(data);  
     }  
    });  
    console.log('after file read');
  + readFile() accepts three parameters
    - Filepath
    - Encoding to be used
    - Callback function which has parameters for error and data
  + Above example is the **asynchronous** implementation of file read.
  + It does not wait for the readFile operation to complete, instead the control goes to the next line and start executing it.
  + console.log('before file read sync');  
    var file\_content = fs.readFileSync('files.txt', 'utf-8');  
    console.log(file\_content);  
    console.log('after file read sync');
  + Above is a synchronous implementation of file read.
  + I makes the program execution to wait until the **readFileSync** operation has completed and then proceed to the next line for execution.
* **Write File**
  + Use writeFile to write data onto a file.
  + fs.writeFile('file\_open.txt', 'hello this is a test content','utf-8', function(error, data) {  
     if(error) {  
     console.log('Error while writing data');  
     } else {  
     console.log('Data written onto file');  
     }  
    });
  + Above example open a file file\_open.txt and write content onto the file
  + If the file is not present it will create a new file with the name specified.
* **File open**
  + Use **.open** function to create a new file.
  + fs.open('file\_open.txt', 'a+', function(error, data) {  
    });
  + .**open** accepts file path, mode in which the file has to be opened and then the callback function which has parameters for error and data
* **File open and write**
  + var fs = require('fs');  
    console.log('before file write');  
    fs.open('file\_open.txt', 'a+', function(error, data) {  
     if(error) {  
     console.log('Error occurred while trying to open file');  
     } else {   
     fs.write(data, 'hello this is a test content','utf-8', function(error, data) {  
     if(error) {  
     console.log('Error while writing data');  
     } else {  
     console.log('Data written onto file');  
     }  
     });   
     console.log('File opened successfully');  
     fs.close(data, function(error){  
     if (error){  
     console.log(err);  
     }   
     console.log("File closed successfully.");  
     });  
     }  
    });  
    console.log('After file write');
  + The above example will create a file in append mode and pass the file to write function to write data on to the file, so the previous content is overwritten.
  + Use **.close** function to close a file
* **Delete a file**
  + Use **.unlink** function to delete a file.
    - fs.unlink('file\_open.txt', function(error) {  
       if (error) {  
       console.log(err);  
       }  
       console.log("File deleted successfully!");  
      });

**Streams**

* **Read stream**  
    
  var fs = require('fs');  
  var read\_stream = fs.createReadStream('files.txt');  
  var data = '';  
  read\_stream.setEncoding('UTF8');  
    
  read\_stream.on('data', function(data\_chunk) {  
   data += data\_chunk;  
  });  
    
  read\_stream.on('end', function() {  
   console.log(data);  
  });  
    
  read\_stream.on('error', function(error){  
   console.log(error);  
  });  
  + Above example make use of readstream to read file and using the **data** event it append each chunk of data once it is received to **data variable**.
* **Write Stream**var fs = require('fs');  
  var write\_stream = fs.createWriteStream('file.txt', {'flags': 'a'});  
  var data = 'This is a data written to file using write stream.';  
    
  write\_stream.write(data, 'utf-8');  
  write\_stream.end();  
    
  write\_stream.on('finish', function() {  
   console.log('Data written to file');  
  });
* **Piping streams :** Piping in stream is a way by which to provide output of one stream as the input of another stream.  
    
  var fs = require('fs');  
  var read\_stream = fs.createReadStream('files.txt');  
  var write\_stream = fs.createWriteStream('temp.txt');  
    
  read\_stream.pipe(write\_stream);  
    
  console.log('Done...');
* Callback for handling pipe
  + var pipe\_stream = read\_stream.pipe(write\_stream);  
      
    write\_stream.on('close', function(err, data) {  
     if(err) {  
     console.log('Error streaming data');  
     } else {  
     console.log('Stream finished');  
     }  
    });

**http module**

* **Making request using http module**
  + **.request()**
  + var http = require('http');  
      
    var full\_data = '';  
    var req = http.request('http://www.google.com', function(response) {  
     response.on('data', function(chunk) {  
     full\_data += chunk;  
     console.log('data-------------------------------------------------------'+chunk);  
     });  
     response.on('end', function() {  
     console.log('full data received ======================================== '+full\_data);  
     });  
    });  
      
    req.end();
* Using Node to create web server
* var http = require('http');  
  http.createServer(function(req, res) {  
   res.writeHead(200, {'Content-Type': 'text/plain'});  
   res.end('Hello World\n');  
  }).listen(1337, '127.0.0.1');  
  console.log('Server running at <http://127.0.0.1:1337/>');
* var http = require('http'),  
   fs = require('fs');  
  http.createServer(function(req, res) {  
   if(req.url=='/index.html') {  
   fs.readFile('socket.html', 'utf-8', function(err, data) {  
   if(err) {  
    
   } else {  
   res.writeHead(200, {'Content-Type':'text/html'});  
   res.end(data);  
   }  
   });  
   }  
   res.writeHead(200, {'Content-Type': 'text/plain'});  
  }).listen(1337, '127.0.0.1');  
  console.log('Server running at http://127.0.0.1:1337/');
* In the above example we used **require** to import http modules in your nodejs application
  + var http = require("http");
* Using **http instance** we called **http.createServer()**  to create server instance and we bind it to a port using **listen method**
* **createServer** has function parameters for request and response
* http.createServer(function (request, response) {  
    
   response.writeHead(200, {'Content-Type': 'text/plain'});  
     
   response.end('Hello World\n');  
  }).listen(8081);  
  + **response.writeHead()**
    - Send the http header
    - In the above example it sets the http status code as 200 and content type as just plain text
  + **response.end()**
    - Sends the response body

**Socket.IO**

* Provides an abstraction over the various methods used to maintain an active connection between browser and server.
* In Node.js, Socket.IO provides a consistent interface for performing the socket based communication in both client (browser) and the server
* Browser will issue **io.conect()** to establish a connection to the **Node.js server.**
* Server will receives a **connection event**, and **emits a new event** if in case if you want to send an data back to the client.
* Browser will then receives the new event emitted from the server using the **.on() function.**
* **Javascript file (server)**  
  var http = require('http');  
  var socketio = require('socket.io');  
  var fs = require('fs');  
    
  var http\_handler = function(req, res) {  
   fs.readFile(\_\_dirname+'/socket.html', function(err, data) {  
   if(err) {  
   res.writeHead(500);  
   return res.end('Error loading request page..!!!');  
   } else {  
   res.writeHead(200);  
   res.end(data);  
   }  
   });  
  };  
    
  var app = http.createServer(http\_handler);  
  var io = socketio.listen(app);  
    
  // connection using express  
  var server\_listen = app.listen(3307, function() {  
   console.log('Server running @ localhost:3307');  
  });  
  var io = require('socket.io').listen(server\_listen);  
  // end connection using express  
    
  io.sockets.on('connection', function(socket) {  
   setInterval(function() {  
   var timestamp = Date.now();  
   socket.emit('msg\_to\_client', timestamp);  
   }, 2000);  
   socket.on('submit', function(data) {  
   io.sockets.emit('msg\_to\_client', data);  
   console.log('Data submitted by client = '+data);  
   });  
  });  
    
  app.listen(1337, '127.0.0.1');  
    
  console.log('Server running at <http://127.0.0.1:1337/>');
* **Html file**<html>  
   <head>  
   <title>Node.js Application</title>  
   <script src="/socket.io/socket.io.js"></script>  
   </head>  
   <body style="margin: 0 auto;width: 70%;">  
   <script type="text/javascript">  
   var socket;  
   socket = io.connect();  
   socket.on('incoming\_chat', function(data) {  
   document.write(data);  
   });  
   socket.emit('submit', 'Hello there from client');  
     
   </script>  
   </body>  
  </html>

**Mongodb**

* As per the mongodb site
* MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling.
* MongoDB obviates the need for an Object Relational Mapping (ORM) to facilitate development.
* **Documents**
  + Each record in mongodb is a document.
  + Data structure is composed of key value pair
  + Mongodb documents are very similar to json objects.
  + Values can have other documents, arrays or array of documents.
  + {  
     "name":"Arungopan",  
     "email":"gopan.arun@gmail.com",  
     "address":{  
     "country":"India",  
     "state":"Kerala",  
     "city":"Trivandrum"  
     },  
     "phone":["9446614757", "9872347239432", "988872164837"]  
       
    }
* **Collection**
  + Mongodb stores documents in collection, which acts like tables in relational databases
  + Documents stored in collection must have a unique \_id field that acts like primary key in relational database.
  + {  
     "\_id" : ObjectId("214324adsa2432434"),  
     "user1" : {  
     "name":"Arungopan",  
     "email":"gopan.arun@gmail.com",  
     "address":{  
     "country":"India",  
     "state":"Kerala",  
     "city":"Trivandrum"  
     },  
     "phone":["9446614757", "9872347239432", "988872164837"]  
       
     },   
     "user2":{  
     "name":"Arungopan",  
     "email":"gopan.arun@gmail.com",  
     "address":{  
     "country":"India",  
     "state":"Kerala",  
     "city":"Trivandrum"  
     },  
     "phone":["9446614757", "9872347239432", "988872164837"]  
       
     }  
    }

**Install mongodb**

* Goto <https://www.mongodb.com/>
* Click on download
* Goto bin folder of mongodb through command line
* Execute **mongod,** which starts the server
* Open another terminal window and execute **mongo,** provides mongo terminal
* Now we need to create a database for storage
* Commands
  + show dbs; // list all the database already present
  + If you want to create a use an existing database try **use database\_name;** command.
  + **db.dropDatabase();** // drops current database;
  + **db.createCollection(collection\_name);** // create a new collection
  + **show collections;** list all the collections available in your current database
  + **db.collection\_name.drop();** // drops collection
  + **db.collection\_name.insert(document);** // inserts a new document into the collection
  + **db.collection\_name.insertMany([document1, document2, document3]); //** pass an array of documents to insert multiple documents
  + **db.collection\_name.find();** // get all the data from your collection
  + **db.collection\_name.find().pretty();** // return data in a formatted way.
  + **db.collection\_name.findOne(); //** return just one row
  + **db.collection\_name.find().limit(5);** //return 5 rows if found
  + **Condition to be passed in find()**
  + db.users.find({“name”:”Arun”}); // checks whether value is equal to value passed
  + { "\_id" : ObjectId("4ecc05e55dd98a436ddcc47c")}
  + db.users.find({“user\_id”:{$lt:2}}).pretty(); // checks for all values less than value passed
  + **$lte** : less than or equal to
  + **$gt :** greater than
  + **$gte:** greater than or equal to
  + **$ne :** not equal to
  + **$in** : which contains the given value  
     db.details.find({rated: {$in: [“G”, “PG”]}})
  + /value/ : db.users.find({“name”:/Ar/}); // checks for all values like %Ar% in RDBMS
  + /^value/ : 'value%'
  + /value$/ : '%value'
* **Multiple conditions**
  + **AND condition** **:** db.collection\_name.find({key1:value1, key2:value2});
  + **OR condition :** db.collection\_name.find({  
     $or:[  
     {key1:value1},   
     {key2:value2}  
     ]}).pretty();
* **Update:**
  + db.collection\_name.update(condition, fields to update);
  + db.users.update({“user\_id”:”4”}, {$set:{“name”:”Arungopan”, “location”:”Trivandrum”}}); // updates the very first matched element.
  + **Multi:true :** db.users.update({“user\_id”:”4”}, {$set:{“name”:”Arungopan”, “location”:”Trivandrum”}}, {multi:true}); // updates all documents
  + **$unset :** removes field from a document
    - db.training.update({ topics:'PHP'} ,{ $unset : { 'student\_count':''} }, {multi:true})
  + **Update using \_id**
    - db.training.update({'\_id':ObjectId("58b07049cc7dd1d6b2f496fd")}, {$set:{'topics':'Angularjs 2.0'}});
* **Delete :**
  + db.collection\_name.remove({“key1”:”value1”});
  + db.users.remove({“user\_id”:4});
* **Sort :**
  + db.collection\_name.find().sort(“key”:1/-1);
    - Value 1 sorts ascending
    - Value -1 sorts descending
  + db.users.find().sort({“name”:1});
* **$elemMatch** :   
  db.details.find({  
   boxOffice: {  
   $elemMatch : {  
   country : “UK”,   
   revenue : { $gt : 15 }  
   }  
   }  
  })  
    
    
  Sample document could be something show below.  
  boxOffice : [  
   { “country” : “India”, “revenue”:2 },  
   { “country” : “UK”, “revenue”:5 },  
   { “country” : “US”, “revenue”:16 },  
   { “country” : “Canada”, “revenue”:27 },  
  ]  
  In the above query it will check documents satisfying both the conditions together.
* **$update :** 
  + db.training.update({'topics':'Angular', 'boxOffice':{$elemMatch:{'country':'India'}}}, {$set:{'boxOffice.$.revenue':45}}

**Mongodb shell multi line commands**

If you end a line with an open parenthesis ('('), an open brace ('{'), or an open bracket ('['), then the subsequent lines start with ellipsis ("...") until you enter the corresponding closing parenthesis (')'), the closing brace ('}') or the closing bracket (']')

**Mongodb - SQL comparison chart**

If you end a line with an open parenthesis ('('), an open brace ('{'), or an open bracket ('['), then the subsequent lines start with ellipsis ("...") until you enter the corresponding closing parenthesis (')'), the closing brace ('}') or the closing bracket (']')

**Mongodb Nodejs**

* Install mongodb module : npm install mongodb
* Include mongodb to your application use
  + **require('mongodb').MongoClient;**
* **.connect(url, function())** // to establish connection with mongodb
  + Url : specify connection string using url
    - mongodb://username:password@localhost:27017/users
  + var MongoClient = require('mongodb').MongoClient;  
    var url = 'mongodb://localhost:27017/users';  
      
    MongoClient.connect(url, function(err, db) {  
     console.log("Connected correctly to server.");  
     db.close();  
    });
* **Insert data**
  + Establish connection
  + Set collection to which data has to be inserted.
  + Use **insert()** function to insert data into collection
  + var MongoClient = require('mongodb').MongoClient;  
      
    var url = 'mongodb://localhost:27017/test\_db';  
      
    MongoClient.connect(url, function(err, db) {  
     console.log("Connected correctly to server.");  
     var data\_ins = {  
     "user\_id":5,  
     "name":"Test user 1",  
     "location":"Test location 1"  
     };  
     db.collection('users').insert(data\_ins, function(err, res) {  
     if(res) {  
     db.close();  
     }  
     });  
     console.log("Data inserted to users collection");  
    });
* **Query data**
  + Use **find()** function to query data from collection
  + var MongoClient = require('mongodb').MongoClient;  
      
    var url = 'mongodb://localhost:27017/test\_db';  
      
    MongoClient.connect(url, function(err, db) {  
     var data\_obj = db.collection('users').find({"name":"Arun", "location":"India"});  
     data\_obj.each(function(err, res) {  
     console.log(res);  
     });  
    });
* **Update Data**
  + Use **update()** function
  + var MongoClient = require('mongodb').MongoClient;  
      
    var url = 'mongodb://localhost:27017/test\_db';  
      
    MongoClient.connect(url, function(err, db) {  
     db.collection("users").update(  
     {"user\_id":6},  
     {$set: {  
     "location":"Trivandrum, Kerala, India"  
     }  
     }  
     );  
     var data\_obj = db.collection('users').find();  
     data\_obj.each(function(err, res) {  
     console.log(res);  
     });  
    });
* **Delete data**
  + Use **deleteOne(), deleteMany()** function
  + deleteOne() only deletes the first matching data
  + deleteMany() deletes all the matching data
  + var MongoClient = require('mongodb').MongoClient;  
      
    var url = 'mongodb://localhost:27017/test\_db';  
      
    MongoClient.connect(url, function(err, db) {  
     db.collection("users").deleteOne({"name":"Arun"});  
     var data\_obj = db.collection('users').find();  
     data\_obj.each(function(err, res) {  
     console.log(res);  
     });  
    });
  + **deleteMany** with empty condition will delete all documents from the collection
  + .drop() function to drop a collection.
    - db.collection("users").drop();

**Reusing mongodb connections**

var http = require('http'),

MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost:27017/grades',

http\_server = http.createServer(function(req, res) {

if(req.url=="/index") {

var resp = mongodb\_client.collection('users').find();

res.writeHead(200, {'Content-Type': 'text/plain'});

resp.each(function(err, data) {

res.end(JSON.stringify(data));

});

} else {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello World\n');

}

}),

mongodb\_client = '';

MongoClient.connect(url, function(err, db) {

if(err) {

console.log('Error connecting to database');

} else {

console.log("Connected correctly to server.");

mongodb\_client = db;

http\_server.listen(1337, '127.0.0.1');

console.log('Server running at http://127.0.0.1:1337/');

}

});

**Mongoose**

* Brings mongodb object modeling for node.js
* Provides schema based solution to model your application data
* A schema is a structure for a database.
* Install mongoose using npm
  + npm install mongoose
* Once you install you need to include mongoose into your application.
  + var mongoose = require('mongoose');
* To establish connection use .connect method.
  + mongoose.connect('mongodb://localhost/users');
* For us to get notified once a connection is established or an error happens, we need to setup callbacks for each.
* **Note: default promise library of mongoose is deprecated we need to use the following commands to set up the promise library for mongoose**
  + mongoose.Promise = require('q').Promise;
* var mongoose = require('mongoose');  
  mongoose.connect('mongodb://localhost/users');  
  var db = mongoose.connection;  
    
  db.on('error', function() {  
   console.log('Error happened');  
  });  
    
  db.on('open', function() {  
   console.log('Connection established');  
  });
* In Mongoose everything is derived from a Schema.
* Each schema maps to mongodb collection and defines the structure of documents within that collection.
* To create a schema
  + var user\_schema = mongoose.Schema({  
     user\_id:Number,  
     name:String,  
     location:String,  
     phone:Number,  
     active:Boolean  
     });
  + Above code creates a schema with 5 properties
  + Using schema which we just created will create **Model**.
  + A **Model** is a class using which we construct documents. In our case each document will be a user\_model with properties and behaviors as specified in schema.
  + **var user\_model = mongoose.model('user\_details', user\_schema);**
  + First param is the name of the collection the model is for and the second param is the schema for the same.
  + Once we have the model, now are good to create a document using model.
  + var user\_data = {  
     "user\_id":"2",  
     "name":"Naveen",  
     "location":"US",  
     "phone":9446614757,  
     "active":true  
    };  
    **var user = user\_model(user\_data);**
  + The above code create a document with values from **user\_data** variable.
  + Now we have created our model and document, but we have saved anything to mongodb.
  + We are now good to save data to mongodb using **.save()**
  + **student\_document.save(function(err) {  
     if(err) {  
     console.log(err);  
     } else {  
     console.log('Data saved!!!');  
     db.close();  
     }  
     });**
  + **Finding data :** use **.find()** function
  + user\_model.find({"name":"Arun", "location":"US"},function(err, res) {  
     if(err) {  
     console.log("Error occurred");  
     } else {  
     var res\_len = res.length;  
     if(res\_len==0) {  
     console.log("No matching records found");  
     } else {  
     res.forEach(function(record){  
     console.log(record);  
     });  
     }  
     }  
     });
  + In the above example if you see we can loop through our data using forEach
  + **Update document :** use **.update()**
  + user\_model.update({"user\_id":2}, {$set: {"name":"Nandakumar"}}, function(err, res) {  
     if(err) {  
     console.log('Error occurred');  
     } else {  
     console.log('Model updated');  
     }  
     });
  + **Delete document :** use **.remove()**
  + user\_model.remove({"user\_id":2}, function(err, res) {  
     if(err) {  
     console.log('Error occurred');  
     } else {  
     console.log('Document deleted');  
     }  
     });

**Express framework**

* Express in nodejs is a framework which provides features like
  + Middlewares to respond to your request
  + Routing
  + Render html web pages
* Install Express onto your application directory using npm
  + npm install express --save
* After installing require the same module in your application
  + var express = require('express');  
    var app = express();
* **Routing**
  + Way to handle multiple url context in a website, or how they respond to client requests
  + Example below show basic routing using express.
  + var express = require('express');  
    var app = express();  
      
    app.get('/', function(req, res) {  
     res.send('Hello this is default root page, displayed via express routing');  
    });  
      
    var server = app.listen(8081, function() {  
     console.log("App listening to localhost:8081")  
    });
* **Route methods**
  + A route method is derived from one of the HTTP methods, and is attached to an instance of the express class.
  + **GET method** route
    - app.get('/', function (req, res) {  
       res.send('GET request to the homepage');  
      });
  + **POST method** route
    - app.post('/', function (req, res) {  
       res.send('POST request to the homepage');  
      });
  + **All method** : not derived from any of the http methods
    - This method is used for loading middleware functions at a path for all request methods
    - the handler will be executed for requests to “/showhome” from all the methods mentioned above
    - app.all('/showhome', function (req, res, next) {  
       console.log('Loading showhome');  
      });
  + **Use method** :
    - app.use(function(req, res) {  
       res.sendStatus(404);  
      });
* Sample structure of a page with just one routing
  + var express = require('express');  
    var app = express();  
      
    app.get('/', function(req, res) {  
     res.send('Hello World');  
    });  
      
    // handles all the invalid request made to server  
    app.use(function(req, res) {  
     res.sendStatus(404);  
    });  
      
    var server = app.listen(3307, function() {  
     console.log('Server running at port 3307');  
    });
* **Route paths**
  + Route paths can be strings, string patterns, or regular expressions.
  + app.get('/', function (req, res)
    - Will match request to root “/”
  + app.get('/home', function (req, res)
    - Match request to home
  + app.get('/home.page', function (req, res)
    - Match request to home.page
  + app.get('/ho?me', function(req, res)
    - Match request to home, hme
  + app.get('/ho+me', function(req, res)
    - Match request to home, hoome, hoooome and so on
  + app.get('/ho\*me', function(req, res)
    - Match request to home, ho123me, hohhasdme and so on.
  + app.get('/h(om)?e', function(req, res)
    - Match request to home, he
  + app.get(/h/, function(req, res)
    - Match request that has “h” within the route name
* **Middlewares**
  + Middleware functions are functions that have access to the request / response objects, and the next middleware function in the application’s request-response cycle.
* **Serving Static Files**
  + Express provides a built-in middleware **express.static** to serve static files, such as images, CSS, JavaScript etc
  + app.use(express.static(‘assets’));
    - assets is the folder where i have my static files
  + Let's create a folder inside assets images and copy images onto that folder
  + Then you can browse the static files using the url.
  + var express = require('express');  
    var app = express();  
      
    app.use(express.static('assets'));  
      
    app.get('/', function(req, res) {  
     res.send('Home page');  
    })  
    var server = app.listen(8081, function() {  
     console.log("App listening to localhost:8081")  
    });
* **Using templates**
  + Require a module **consolidate**
  + **var engines = require(‘consolidate’);**
  + Consolidate is basically a set of wrappers for a number of template engines for express.
  + Once consolidate is included we need to have a few settings for the template engines
  + **Assigns nunjucks to html files (specifying the template engine)**
    - app.engine(‘html’, engines.nunjucks);
  + Statement below denotes that application uses **engines.nunjucks** to render our html files (install nunjucks module => npm install nunjucks)
    - Specifying html as the default extension
    - app.set(‘view engine’, ‘html’);
  + Specifying where the templates are located
    - app.set(‘views’, \_\_dirname+’/views’);
  + **res.render() function**
    - To render html pages use instead of res.send() use **res.render()**
    - **res.render(‘express’, {‘name’:’Marlabs’});**
  + Sample application with template engine would look like
  + var express = require('express'),  
     app = express(),  
     engines = require('consolidate');  
      
    app.engine('html', engines.nunjucks);  
    app.set('view engine', 'html');  
    app.set('views', \_\_dirname+'/views');  
      
    app.get('/', function(req, res) {  
     var companies = ['Marlabs', 'Google', 'Facebook', 'LinkedIn'];  
     res.render('express\_new', {'username':'Arungopan', 'comapnies':companies});  
    });  
      
    // handles all the invalid request made to server  
    app.use(function(req, res) {  
     res.sendStatus(404);  
    });  
      
    var server = app.listen(3307, function() {  
     console.log('Server running at port 3307');  
    });
  + Html file
  + <p>Hello {{username}}, Welcome !!!!</p>  
     <form method="POST" action="/postdata">  
     <ul>  
     {% for company in comapnies %}  
     <li><input type="radio" name="company\_val" value="{{company}}" />{{company}}</li>  
     {% endfor %}  
     </ul>  
     <input type="submit" value="Submit" />  
     </form>
* **Handling GET request**
  + Using **req.params** & **req.query** you can get values from the url
  + Sample application would look like
  + app.get('/user/:user\_id/:username', function(req, res) {  
     var user\_id = req.params.user\_id,  
     username = req.params.username,  
     getVal1 = req.query.getVal1,  
     getVal2 = req.query.getVal2;  
     res.render('users', { "name":"Marlabs",   
     "page\_name":"Users",   
     "user\_id":user\_id,   
     "username":username,  
     "getVal1":getVal1,  
     "getVal2":getVal2  
     });  
    });
* **Handling POST request**
  + Need to install body-parser module which is a middleware to handle the post request
    - npm install body-parser
  + // parse application/x-www-form-urlencoded  
    app.use(bodyParser.urlencoded({‘extended’:false}));
    - **extended : true** for using qs library (for serializing more complex datastructures)
    - **extended : false** for using querystring library (limited to flat datastructures)
  + // parse application/json  
    app.use(bodyParser.json());
  + **Sample application would look like**
  + var express = require('express'),  
     app = express(),  
     engines = require('consolidate'),  
     bodyParser = require('body-parser');  
      
    app.engine('html', engines.nunjucks);  
    app.set('view engine', 'html');  
    app.set('views', \_\_dirname+'/views');  
      
    app.use(bodyParser.urlencoded({‘extended’:false}));  
    app.use(bodyParser.json());  
      
    app.get('/companies', function(req, res) {  
     res.render('companies', {"companies":["Marlabs", "Google", "Facebook", "Microsoft"]});  
    });  
      
    app.post('/sel\_company', function(req, res) {  
     if(req.body.company\_val) {  
     res.send('The company you would like to work with is '+req.body.company\_val);  
     }  
    });  
      
      
    // handles all the invalid request made to server  
    app.use(function(req, res) {  
     res.sendStatus(404);  
    });  
      
    var server = app.listen(3307, function() {  
     console.log('Server running at port 3307');  
    });
  + Html page would look like
  + <html>  
     <head>  
     <title>Companies</title>  
     </head>  
     <body>  
     <h1>Which company do you like to work for??</h1>  
     <form method="POST" action="/sel\_company?name=Arungopan">  
     <ul>  
     {% for company in companies %}  
     <li>  
     <input type="radio" name="company\_val" value="{{company}}" />{{company}}  
     </li>  
     {% endfor %}  
     </ul>  
     <input type="submit" value="Submit" />  
     </form>  
     </body>  
    </html>
* **GET / POST Method**
  + The example below show how to pass values html form GET method
  + var express = require('express');  
    var app = express();  
      
    app.use(express.static('assets'));  
      
    app.get('/show\_get\_form', function(req, res) {  
     res.sendFile(\_\_dirname + "/" + "get.html");  
    });  
      
    app.get('/store\_get\_values', function(req, res) {  
     response = {  
     username : req.query.username,  
     password : req.query.password  
     };  
     console.log(response);  
     res.end(JSON.stringify(response));  
    });  
    app.post('/store\_get\_values', function(req, res) {  
     response = {  
     username : req.query.username,  
     password : req.query.password  
     };  
     console.log(response);  
     res.end(JSON.stringify(response));  
    });  
      
    var server = app.listen(8081, function() {  
     console.log("App listening to localhost:8081")  
    });

**RESTFul API**

* REST stands for REpresentational State Transfer.
* A web service is a collection of open protocols and standards used for exchanging data between applications or systems
* Web services based on REST Architecture are known as RESTful web services. These webservices uses HTTP methods to implement the concept of REST architecture
* Let's create a json based database for a user.
* After creating json for the users, let’s create a RESTful api for listing users “/list”
* var express = require('express');  
  var app = express();  
  var fs = require('fs');  
    
  app.get('/list', function(req, res) {  
   var json\_file\_path = \_\_dirname + "/" + "user\_data.json";  
   fs.readFile(  
   json\_file\_path,  
   'utf-8',  
   function(err, data) {  
   res.end(data);  
   }  
   );  
  });  
    
    
  var server = app.listen(8081, function() {  
   console.log("App listening to localhost:8081")  
  });
* /delete  
  var express = require('express');  
  var app = express();  
  var fs = require('fs');  
    
  app.get('/delete', function(req, res) {  
   var json\_file\_path = \_\_dirname + "/" + "user\_data.json";  
   fs.readFile(  
   json\_file\_path,  
   'utf-8',  
   function(err, data) {  
   var data\_parse = JSON.parse( data );  
   delete data\_parse["user3"];  
   fs.writeFile(  
   json\_file\_path,  
   JSON.stringify(data\_parse),  
   "utf-8",  
   function(err, data) {  
   console.log("User deleted successfully");  
   }  
   );  
   res.end(data);  
   }  
   );  
  });  
    
    
  var server = app.listen(8081, function() {  
   console.log("App listening to localhost:8081")  
  });

**Express with mongodb**

var express = require('express');  
var app = express();  
var fs = require('fs');  
var MongoClient = require('mongodb').MongoClient;  
  
app.get('/list', function(req, res) {  
  
 MongoClient.connect('mongodb://localhost:27017/users', function(err, db) {  
 if (err) {  
 throw err;  
 }  
 var data\_obj = db.collection('details').find();  
   
 var htm = '<table border="1" cellpadding="5" style="border-collapse:collapse;">';  
 htm += '<tr><th>ID</th><th>Name</th><th>Location</th><th>Phone</th></tr>'  
 data\_obj.each(function(err, res) {  
 if(res.user\_id) {  
 htm += '<tr><td>'+res.user\_id+'</td><td>'+res.name+'</td><td>'+res.location+'</td><td>'+res.phone+'</td><tr>'  
 }  
 });  
 htm += '</table>';  
 res.end(htm);  
 });  
  
});  
  
var server = app.listen(8081, function() {  
 console.log("App listening to localhost:8081")  
});

**Session handling**

* Require module **express-session,**
  + session = require('express-session'),  
    MongoDBStore = require('connect-mongodb-session')(session);
* Set the secret key for session
  + var store = new MongoDBStore({  
     uri: 'mongodb://localhost:27017/marlabs\_new',  
     collection: 'marlabs\_session'  
    });  
      
    app.use(  
     session({  
     secret:'marlabs\_sess\_secret\_key',   
     resave: true,  
     saveUninitialized: true,  
     store: store  
     })  
     );  
      
    store.on('error', function(error) {  
     console.log(error)  
    });
* var sesson\_obj = '';  
  app.get('/', function(req, res) {  
   req.session.username = 'Arungopan';  
  });
* app.post('/login', function(req, res) {  
   if(req.body.username == "admin" && req.body.password == "admin") {  
   req.session.isloggedIn = true;  
   } else {  
   req.session.isloggedIn = false;  
   }  
  });