Node.js

- Introduction
- Web Request/Response Handling (Node/Express)
- I/O Handling
- Template Engine
- Database connection with MongoDB
- Session Management

Overview

- Node.js
  - Open source, cross platform JavaScript framework
  - Server Side
  - JavaScript everywhere
  - Event driven architecture
  - Asynchronous I/O
  - Through put & Scalability improvement

Node is a single threaded event-driven execution environment

History

- Written by Ryan Dahl in 2009
- Combination of Google's v8 JavaScript Engine an event loop and low level I/O API

Installation Process

- Visit https://nodejs.org for installers specific to your OS version.
- Visit https://nodejs.org/en/download/package-manager/ for installation instructions specific to different OS.
- For windows simply click on executable file and follow the steps.
- After installing, check the version of the node using "node –version" command.
- Node.js comes with REPL (Read-Eval-Print-Loop). To launch REPL type "node" at your command prompt.

- Any Node.js expression or JavaScript code can be executed in this REPL

REPL Command List

Command	Description
.help	Display help on all the commands
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
.save filename	Save current Node REPL session to a file.
.load filename	Load the specified file in the current Node REPL session.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression

Data types (Recap)

- Primitive types
  - String, Number, Boolean, Undefined, Null, RegExp
- Reference type
  - Object (Array, Object, JSON…)
  - Buffer (for storing binary data, useful at the time of reading data from network / file)
  - Process (To get all the information about the current process of Node.js application)

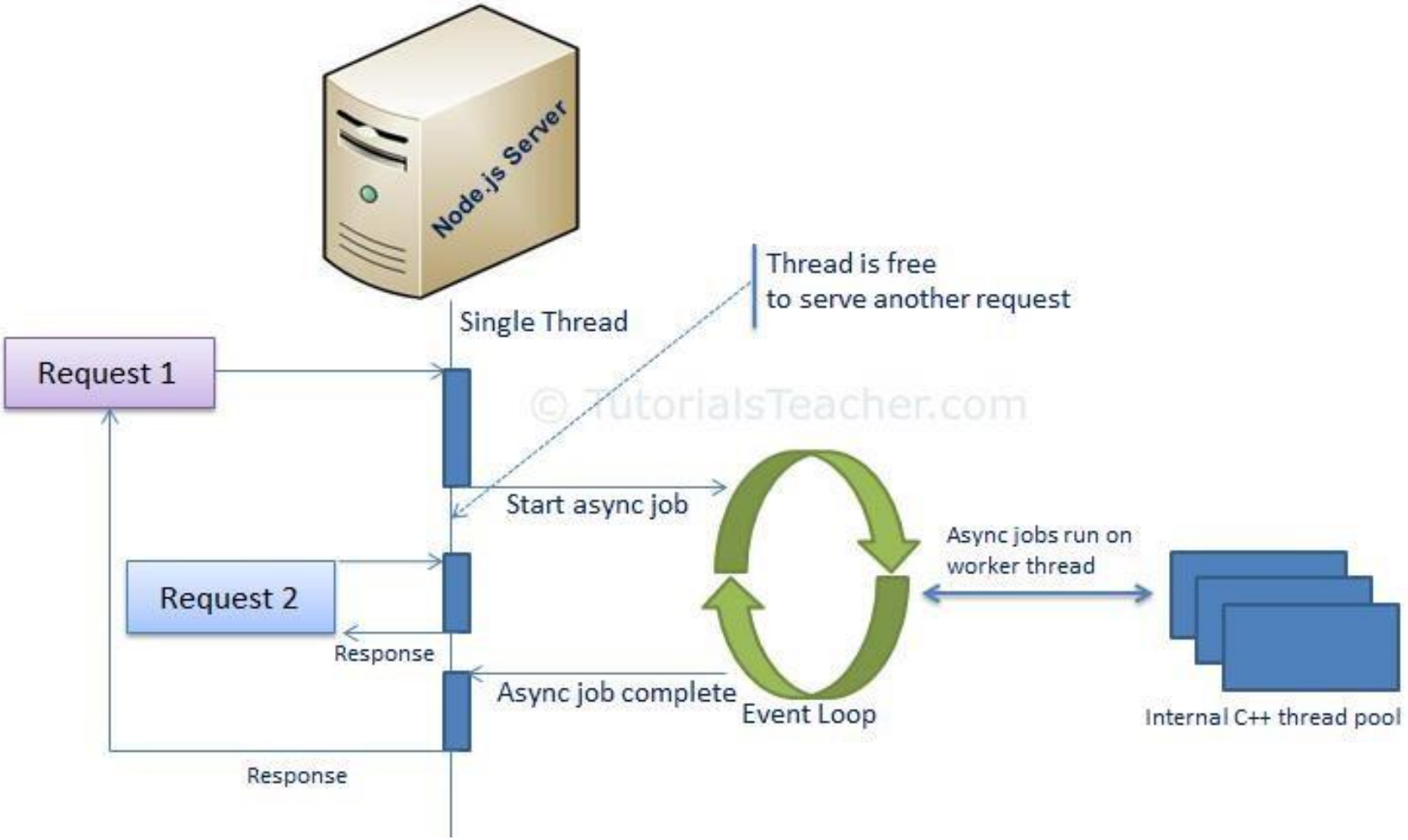
Sample: Creation of HTTP server

```
// Load HTTP module
    var http = require("http");

// Create HTTP server and listen on port 8000 for requests
http.createServer(function(request, response) {
            // Set the response HTTP header with HTTP status and Content type
            response.writeHead(200, {'Content-Type': 'text/plain'});
            // Send the response body "Hello World"
            response.end('Hello World\n');
    }).listen(8000);

// Print URL for accessing server      console.log('Server
running at http://127.0.0.1:8000/');
```

Request Response Handling Process



Creation of Node Application

- Create a directory with the name of your application. Example: mkdir firstapp
- Change to the newly created directory and type npm init.
- This will ask for number of inputs from you like name of the node application you are creating, its version number, description of the application (if any), entry point of the application (default is index.js), test command, license details etc.
- For now, you can simply accept the default values or play with it by giving new values.
- After completion of the command you will notice package.json file created in your directory.

This file lists all the dependencies that your application require in order to run smoothly

Using npm to add new modules to the project

- Npm is node package manager which can be used for installing/updating/uninstalling external packages.

- External packages can be downloaded from internet using below command
  - npm install express (to save the module in local project directory structure)
  - npm install express –g (to make the module available for all the node applications in that computer)
  - npm install express –g --save or npm install express --save will add the package dependency to the package.json file

- Use update or uninstall in place of install in the above command along with package name to be updated or uninstalled.

First Web Application

- In order to process the incoming http requests web server is must and using node.js it can be created using node.js core module called ‘http’.

- Example:

```
var http = require(‘http’);
http.createServer(function(req, res){
    //request and response handling code
```

```
}).listen(8000);
```

First Web Application

- In the request, response function callback we can handle requests to different paths as follows.

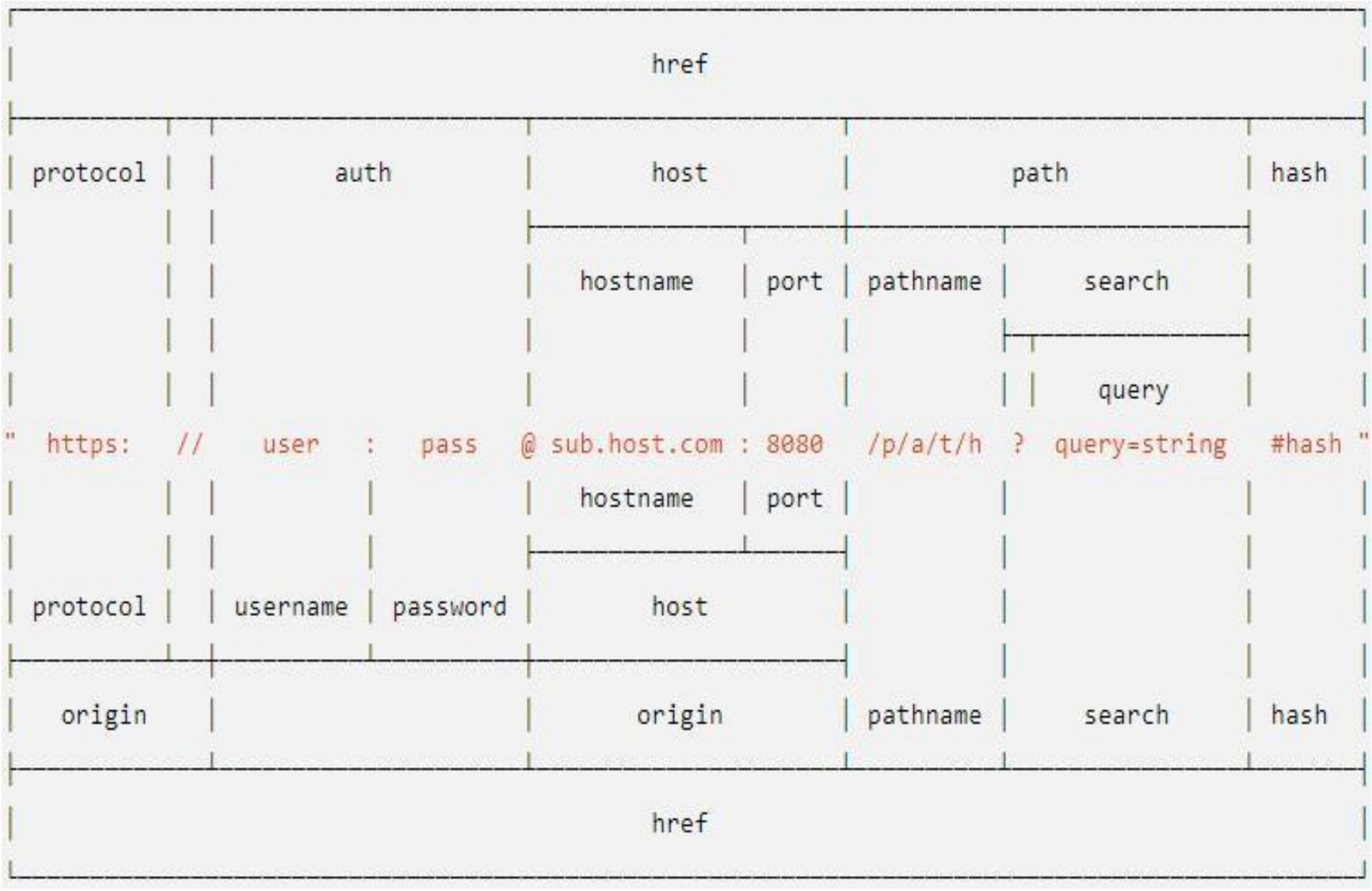
- Example:

```
if (req.url === '/') { res.writeHead('Content-
    Type':'text/html');
    res.write('<html><body><p>I am working for CDAC</p></body></html>');
    res.end();
} if (req.url === '/employee')

{…}
```

- For json response type, the Content-Type should be 'application/json'

Parsing URL


href
protocol
auth
host
path
hash
hostname
port
pathname
search
query
" https: // user : pass @ sub.host.com : 8080 /p/a/t/h ? query=string #hash "
hostname
port
protocol
username
password
host
origin
origin
pathname
search
hash
href

Parsing URL

Example: var url =

require('URL');

var address =

"http://satya:satya@www.yahoo.com";

url.parse(address,true); var host =

url.hostname; var password =

url.password; var origin = url.origin;

Scope of Variables

- By default a variable declared without var keyword is global variable in JavaScript
- In Node.js a variable declared without var keyword is a local variable
- In order to bring a variable into global scope the variable has to be exported using exports.{variable name}
  - Example: exports.log=function(){….}
- Likewise, for importing a module in Node.js we need to use require statement
  - Example: require(‘path’)

Modules

- A module is a functionality organized in single/multiple JavaScript files which can be reused in entire Node.js application anywhere

- Module can be imported using require(modulename) method. It returns the reference of the module to be loaded and using the reference the methods can be invoked corresponding to the module.

- Node.js contains following core modules
  - http for http server - url for URL resolution and parsing
  - path for working with file paths - fs for working with file I/O
  - util for working with certain utilities
  - net for working with clients and servers
  - Buffer for working with binary streams
  - crypto for working with OpenSSL cryptographic functions

- querystring for working with query string of the url

User defined modules

```
db = { getConnection:function(){ return 'get
        method is invoked';
        }
}
module.exports = db;
```

module.exports is a special object which is included in every JS file of your node.js by default.

To include / use it in another JS file of your node.js application you need to simply use the following statement.

```
var database = require('./db.js')
database.getConnection();
```

Exporting different types as modules

- exports is an object so you can attach properties or functions to it to expose them

- Exporting a string variable – Example

  module.exports = ‘Hello World’; //write this in any message.js file

  You can use this variable in another file, say for example, info.js as

  ```
  var content = require(‘./message.js’);
  console.log(content);
  ```

  OR

  module.exports.msg = ‘Hello World’; //write this in any message.js file

You can use this variable in another file, say for example, info.js as

```
var content = require('./message.js');
console.log(content.msg);
```

Example

```
exports.get = function(name,query,field) {
return new Promise(function(resolve,reject){
var table = dbase.collection(name);
        table.find(query,field).toArray(function(err,result){
            if (err) reject(err);
else
                resolve(result);
        });
  });
};
```

‘fs’ methods

- All the methods of this module contain both synchronous as well as asynchronous versions.
- fs.readFile, fs.writeFile, fs.open, fs.appendFile, fs.unlink etc.
- Do not use fs.access or fs.exists methods to check whether the file exists in the system or not before open/read/write. You may get in to race condition if two processes try to execute the methods at the same time.
- Flags such as r, r+, rs, w, w+, ws, wx, wx+, a, ax, ax+

Accessing file system from Node.js application

- The ‘fs’ (file system) module supports both synchronous and asynchronous file operations

- Example:
  - Asynchronous reading of file content

```
var fs = require('fs');
fs.readFile('Data.txt','utf8',function(err,content) {
        if (err) throw err;
        console.log(content);
});
```

  - Synchronous reading of file content

```
var fs = require('fs');
fs.readFileSync('Data.txt','utf8',function(err,content
) {
        if (err) throw err;
        console.log(content);
});
```

‘fs’ read/write example

```
fs.read(fd, buf, 0, buf.length, 0, function (err, bytes) {
        if (err) throw err;
        // Print only read bytes to avoid junk.
        if (bytes > 0) {          console.log(buf.slice(0,
        bytes).toString());
        } fs.write(fd,'Nahi nahi
        nahi',bytes,'utf8',function(err){
                if (err) throw err;
                fs.close(fd, function (err) {
                        if (err) throw err;
                });
        });
});
```

Streams

- Streams read/write data in continuous fashion.

- Emits following events during read/write operation.
  - data : when data is available for read
  - end : when no more data exists
  - finish : flush the data in to the stream
  - error : when an error occurs at the time of reading/writing

Piping the streams

- For redirecting output of one stream to as input to another stream

- Example:

```
var fs = require("fs"); var readerStream =
fs.createReadStream('input.txt'); var writerStream =
fs.createWriteStream('output.txt');
readerStream.pipe(writerStream);
```

Uploading of File Content to the

Node.js server

- Require formidable module

- Example

```
var fm = require('formidable');
http.createServer(function(req,res){ if (req.url ===
'/fileUploadHanlder') { var form = new
formidable.IncomingForm();
form.parse(req,function(err, fields,files){ res.write('File
uploaded'); res.end();
                    });
      }
}).listen(9000);
```

Database connection with MongoDB

- The official MongoDB Node.js driver provides both callback based as well as Promised based interaction with MongoDB

- Install mongodb driver using npm install mongodb --save

- In order to connect to a single MongoDB instance, example program **var** MongoClient = require('**mongodb**').MongoClient , assert = require('**assert**');

```
var url = 'mongodb://localhost:27017/dbname';
MongoClient.connect(url, function(err, db) {
        assert.equal(null, err); console.log("Connected
        successfully to server"); db.close();
});
```

Database connection with MongoDB

```
var client =
require('mongodb').MongoClient; var dbase;
var dbstr;

exports.dbconnection = function (uname,pwd,host,port,dbname,options)
{          if (!host) host = 'localhost';      if (!port) port = '27017';
    if (!options) {
options = {
db:{w:1},
        server: {maxPoolSize: 100, minPoolSize:10,socketOptions: {keepAlive:5000,
            connectTimeoutMS:30000, maxIdleTimeMS:86400000,
            reconnectTries:Number.MAX_VALUE, reconnectInterval:1000}}
      }
    }
```

Database connection with MongoDB

```
    if (!uname) dbstr = 'mongodb://'+host+':'+port+'/'+dbname;
else
       dbstr = 'mongodb://'+uname+':'+pwd+'@'+host+':'+port+'/'+dbname;

    client.connect(dbstr,options,function(err,db){
            if (err) { process.exit(0); }
dbase = db;
    });

    process.on('SIGINT', function() {
if (dbase) {
                dbase.close(function () {
                process.exit(0);
          });
    }});
```

Database connection with MongoDB

```
};
```

Document Insertion

- Insert document in to collection using callback approach

Example:

```
var insertDocuments = function(db, callback) { var
        collection = db.collection('documents');
        collection.insertMany([
                {a : 1}, {a : 2}, {a : 3}
          ], function(err, result) {
                console.log("Inserted 3 documents into the
        collection");
        callback(result);
          });
}
```

Document Insertion

- By using promise approach

- Example

```
exports.insert = function(name,query) {
return new Promise(function(resolve,reject){
        var table = dbase.collection(name);
        table.insert(query,{w:1,
        new:true},function(err,result){ if (err) {
        reject(err);
        }
        else {
                resolve(result);
        }
 });
```

```
  });
};
```

Event Handling

- Event handling require ‘events’ module.
- It can be used to raise an event and handle it •

Example:

```
var events = require('events'); var em =
new events.EventEmitter();
em.on('FirstEvent', function (data) {
console.log('First subscriber: ' + data); });
```

```
em.emit('FirstEvent', 'This is my first Node.js event
emitter example.');
```

Node.js Web Application Frameworks

- Express.js 2009 4.16.0
- Hapi.js 2011 16.6.2
- Koa 2013

Node vs Express

- Node does not support
  - specific handling for different HTTP commands (GET, PUT, DELETE etc)
  - Serve static files or use templates for creating dynamic responses

  You will have to write code on your own

- Express is a popular Node web framework (others include Feathers, ItemsAPI, LEAN-STACK, Sails etc.)
  - Write handlers for requests with different HTTP commands
  - Integrate with view rendering engines in order to generate responses through templates

- Add additional request processing “middleware” at any point within the request handling pipeline

Express

- Express application has methods for
  - Routing HTTP requests
  - Configuring middleware
  - Rendering HTML views
  - Registering a template engine
  - Modifying application settings that control how the application behaves

Express.js

- It is based on Node.js middleware module called connect which in turn uses http module.

- Install express.js using npm install express –save

- Create web server using below lines of code – Example:

```
var express = require('express');
var app = express();
var server = app.listen(9000,function(){
    console.log('Server started and listening on port 9000');
});
```

Express.js

- The express module returns a function and upon invoking it will return an object which can be used for – routing http requests
  - configuring middleware
  - rendering html views

- Listen method of the object is used for creating web server.

Basic Application using Express

```
var express = require('express');
var app = express();

var server = app.listen(5000, function () {
console.log('Node server is running..');
});
```

Handling of Requests

```
var express =
require('express'); var path =
require('path'); var app =
express();

var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({extended:false}));

app.get('/', function (req, res) {
        res.sendFile("index.html",{root:path.join(__dirname,'.')},function (err)
        { if (err) { console.log(err); res.status(err.status).end(); }
        });
});
```

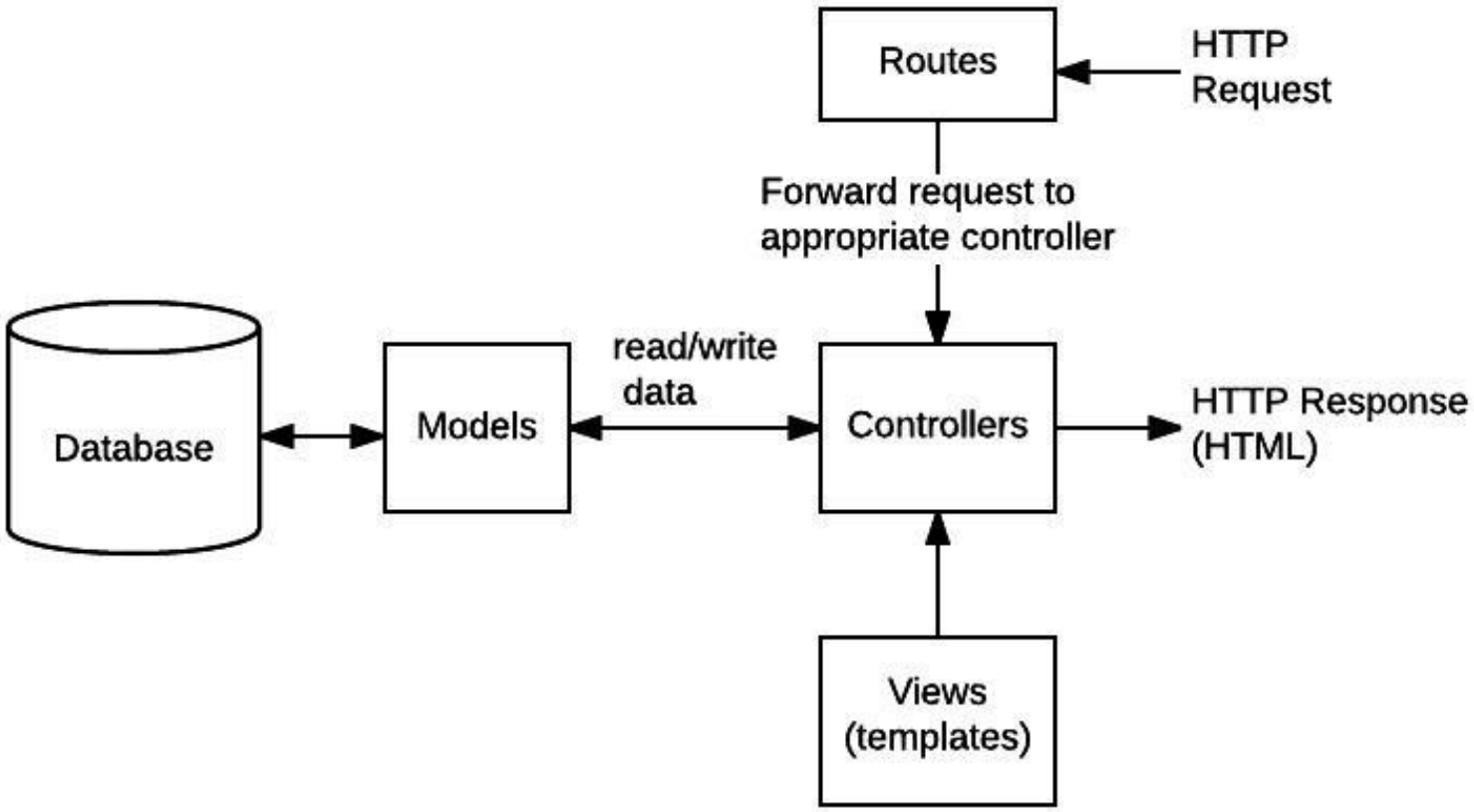
Simple Express Web Application

- Steps:
  - Execute npm install express -g
  - Execute npm install express-generator –g command
- This will be helpful in generating project skeleton structure while creating new express projects

- Create project directory using mkdir command
- CD to project directory and execute
  - express helloworld
- After express application is created execute below command
  - npm install in the same folder where the express application is created.
    This will download all the dependencies listed in the package.json file.
- To start the express application execute below command

- npm start (SET DEBUG=helloworld:* & npm start – for running in debug mode)

Web Application Execution Flow



Routing

- The app object is used for routing different types of requests
  - get
  - post
  - delete
  - put

- app.METHOD(path,handler); Syntax for handling incoming requests

- Example

```
var express = require('express');
var app = express();
app.get("/",function(req,res){.....})
app.post("/fileUpload",function(req,res){....});
```

Routing

- app.all(path,handler)
  - This method will handle all the requests irrespective of method type (GET, POST, TRACE…) – Example:

    app.all('/submit-data',function(req,res,next){…});

- Route paths can be string, string patterns or regular expressions
  - Example: app.get("/ab*de",function(req,res){…});
    app.get(/.*fly$/, function(req,res){…});

Route Handlers

- Multiple callbacks can be provided to behave like a middleware while handling a request.
- We have seen a single callback function approach to handle incoming requests in previous slides.
- Go through example routemiddleware.js and routemiddleware1.js

Chaining of Route Handlers

- For creating chainable route handlers use app.route() method.

  – Example: app.route('/student-data')
  .get(function(req,res){…})
  .post(function(req,res){…})
  .put(function(req,res){…})

Mounting of routes

- Use Router class to create modular and mountable route handlers

- Router instance is a complete middleware and routing system

- Example: (cdac.js) var express = require(‘express’); var router = express.Router();
  router.get(‘/’,function(req,res){…});
  router.get(‘/about’,function(req,res){…}); module.exports = router;

Mount the routes in the main app

```
var cdac = require('./cdac');
app.use('/cdac', cdac);
```

The above routes can be accessed as follows.

http://localhost:3000/cdac/

http://localhost:3000/cdac/about

Modularization of Routes

- Write below kind of lines in your router module
  - var router = require('express').Router();
  - var bkctrl = require('./BookController');
  - router.get('/book/create',bkctrl.createBook);
  - module.exports = router;
- Define the controller methods in separate files so that they can be imported in router file as shown above
  - exports.createBook = function(req,res){…}

Request and Response Objects

- The req object represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers and so on.
  - req.hostname, req.body, req.fresh, req.cookies, req.originalUrl, req.params, req.query, req.accepts(types)

- The res object represents the HTTP response the server sends
  - res.headersSent, res.cookie(name, value,options), res.clearCookie(name,options), res.download(path,filename,options, callback function), res.end(), res.format(object)

- The acceptable types of content can be managed using req.accepts and res.format methods

Request forwarding

- Using http methods we can forward a request from one handler page to another handler page on server side.

- Example:

```
http.get('http://localhost:5000/fetchCityArea?city='+city+'&circle='+circle+'&area='+area,function(resp){
    if (resp.statusCode === 200) {
        resp.on('data',function(buf){msg += buf;});
resp.on('end',function() {…});
    }
```

```
});
```