**DisAdvatages with NodeJS**

Managing sessions

Managing cookies

Parsing the request body for data

Organizing Routing

Extracting URL parameters

The solution for all the above enhancements is ExpressJS

We can use node packages along with express

ExpressJS is a web application framework based on nodeJS modules

* Better to Use expressJS only for the above scenarios , rest of the things can with node packages

**.JS Application**

Node Modules

**Express**

MiddleWare

* Session
* Cookies
* Routing
* URL parsing
* Express consists of lot of middle wares
* REST full interface is the main feature provided by Express
* Provides MVC Architecture and provides code Reusabilty

**Class Room Activity**

* **Discuss about REST and MVC in the class**

**REST Services**

**Database**

**Server**

GET

PUT

DELETE

POST

Mobile Apps(Android, Ios, Win)

Web Apps .Js

Win Desktop

APPs .Net

WebApps .Net

**REST Services**

Server Code for GET, POST, PUT .. are converted as URL . So the clients will call the URL in place of calling Server Code

<http://localhost:4040/GETProduct> , <http://localhost:4040/POSTProduct> , <http://localhost:4040/PUTProduct> ,

**REST**

REST access the logic through URL ,

Supports different clients apps

Support different formats , JSON, MTOM

All the server code methods were converted as URL and clients can use the URLs for data retrieval

Clients requests hits the URL , then to server and then to Database

Npm install [express@4.8.1](mailto:express@4.8.1) –save

4.8.1 is LTS version , wher as 4.15 is the latest one .

**Steps**

Open folder from VScode ->> create package .JSON ->> Install Express and save the dependency

->> add new file app.js

Var express= require(‘express’);

Var app = express();

App.get(“/”, function(req,res){

Response.end(‘Helo world….’);

});

App.get(“/home”, function(req,res){

Response.end(‘Welcome Home Page world….’);

});

App.listen(3000,function(){

Console.log(“Server Started…..”);

})

**Run – node app.js**

Express generator will install all the packages and commands require for general application structure i.e like template for project

Npm install express-generator -g

Can try to execute the application Explanation

**Configuration and settings**

By using set() fo Express app object , we set some global settings, and we can access that values easily in our application , which is recommended at the time of production

Syn : Set (“key”, “value”) – used to set some setting for the applications

Ex:

App.set(“view engine”,”jade”)

App.set(“port”,”3000”);

App.get(“key”)

App.Listen(port , callback(){});

Application(-app-) Object – denotes the express application

We can create the object by calling express function which is exported by express module

Contains methods for routing , configuring middle ware , rendering html views ,registering template engine ,

The methods are Set , Get, Listen, Put, Delete

**Structure of Express JS file**

Step 1 : import the dependencies

Step 2 : Instantiate the Express

Step 3 : Config Settings

Step 4 : using Middile ware

Step 5 : Create the Routes

Step 6 : create webserver

**Complete Structure Ex :**

// Just a demo for structure of Express

//Step 1 : import the dependencies

var express = require('express');

var path = require('path')

//Step 2 : Instantiate the Express

var app = express();

//Step 3 : Config Settings

app.set('port', process.env.PORT || 5000);

app.set('view engine','jade'); // setting the view Engine for the application

app.set('views', path.join(\_\_dirname,"views")); // setting views folder for the application

app.set('env','development'); // setting environement i.e the stage of the application

//Step 4 :

//using Middile ware

//Step 5 : Create the Routes

app.get('/', function(req, res){

res.render('index',{'userName':'SreeRama','doj':'2/2/2222'}) // render() will convert index.jade to index.html with help of Jade

})

//Step 6 : create webserver

app.listen(app.get('port'), function(req, res){

console.log("Server Started at : " + app.get('port'));

console.log("View Engine is : " + app.get('view engine'));

console.log("Views folder is : " + app.get('views'));

})

**Set() Examples**

app.set('port', process.env.PORT || 5000);

app.set('view engine','jade'); // setting the view Engine for the application

app.set('views', path.join(\_\_dirname,"views")); // setting views folder for the application

app.set('env','development'); // setting environement i.e the stage of the application

app.enable('case sensitive routing') // enable case sentive routing

app.enable('strict routing', true); // diffrntiate btwn admin url and /admin/ ie trailing ‘/’

**summary**

app.set()

app.get()

app.enable()

Header

* StatusCode
* ContentType
* Content-length

Body

* HTML tags

**Response**

**Working with Response**

Response - consists of two parts

Header - status codes , content type

Body - html Tags

Content type is the media type for the content

res.set(‘content-type’:’text/plaintext’);

res.json()

res.send()

res.sendfile()

res.local()

**Working HTML pages through view engine**

Html pages can be serverd easily from public folder by setting express.static

If we want html pages to be served by view Engine

Npm install consolidate –save

Npm install swig –save

At app.js file

var cons = require('consolidate');

app.engine('html', cons.swig)

app.set('views',path.resolve(\_\_dirname,'Views'))

app.set('view engine','html')

app.get('/',(req,res)=>{

res.render('vhome');

})

app.get('/contact',(req,res)=>{

res.render('vContact');

})

Note if We are using view Engine send the file using render() of response Object

**Routing in Express**

Inorder to implement clear separation of routing details in express we create all routes in separate file and import the file at app.js file

Use express.Router() to create routes

Ex:

**At Routes file**

var express= require('express');

var Router = express.Router();

Router.get('/',(req,res)=>{

res.end('<h2>Books Home Page </h2>')

})

Router.get('/about',(req,res)=>{

res.end('<h2>Books About Page </h2>')

})

Router.get('/contact',(req,res)=>{

res.end('<h2>Books Contact Page </h2>')

})

Router.get('/services',(req,res)=>{

res.end('<h2>Books services Page </h2>')

})

module.exports = Router;

**at App.js File**

var express = require('express');

var app = new express();

var myRoutes = require('./myRoutes');

var path =require("path");

app.set('port',4500)

app.use('/books',myRoutes)

app.get('/',(req,res)=>{

res.sendFile(path.resolve(\_\_dirname,'home.html'));

})

app.get('\*',(req,res)=>{

res.sendFile(path.resolve(\_\_dirname,'errorPage.html'));

})

app.listen(app.get('port'),(err)=>{

if(err)

console.log('Unable to start server ...')

else

console.log('server Started at : ' + 'http://localhost:'+app.get('port')+ '/')

})

**Activity**

Routing we can use as a middle ware . Routing in a web application needs to be in different files

User routing,

Product routing

Admin routing

**serving Error Pages**

We can create a custom function with req and res which need to respond for errors

app.use(function(req,res){

res.status(404);

res.render("ErrorPage",{'title':'404 Jade','message':'Page is under construction ....'})

})

If no match found for url , Automatically it will uses our custom method , error page will be displayed

**Req.params()** : used to parse the url and get the values

Practice and explain Example

Request url is /home : uname/:password

**Activity : Explain how to collect values form URL**

**Working with Middle ware**

Is a small software which we can use when reuire

By use() we can utilize middleware

App.use(express.static,’path’)

Ex:

app.use(express.static(path.join(\_\_dirname,'public')));

Now we can access the files under public folder directly through url .

static is a middle which is available by default at ExpressJS , we had may other middlewares which we need to install manually

**Middleware**

Middlw are functions that has acces to the req, res and next middle ware functions

Very useful pattern that allow developers to reuse code with in their applications and can be shared with other in the form of npm module

Is a code which can be reusable

Use() – used to use middleware

Ex : cookie , Session, errorHandling , body-parser

Requires 3 argumenst – req, res , next

Next - indicates that more middleware need to be processed, should proceed next

Syntax:

Var middlware = function(req, res, next ){

Next();

}

* For user defined middleware , programmer must call next () in the code , or else request will be time out and hang

Our request passed through different middlewares in its life cycle

Cookie

Body-Parser

Session

Error Handling

Request

Response

* Any where we can call res.end(); from there our processing end and response will send to the client

**Types of middleware**

Application level Middle ware

Is an instance of app object , can be used by using app.use() method

Routing level middleware

Builtin middleware

Error handling middle ware

Third party middle ware

* Check the middleware Documentation at **expressJS Middleware**

**Demo**

// Application middleware

var express = require('express');

var app = express();

//When any request comes ,log the reuest

app.use(function(req, res,next){

console.log('Login terminal %s : %s :%s', new Date().toString(),req.url, req.method);

next();

// if no next the server will be processing the request continously.. no end

});

app.use(function(req, res,next){ console.log('Login terminal :%s', new Date().toString());next();});

app.listen(4050, function(){

console.log('Server started...');

})

**Third Party middleware Ex:**

var express = require('express');

var app = express();

var ckParser = require('cookie-parser');

// Third Party Middleware

// Cookieparser

// install cookie parser

// import it

// use it

app.use(ckParser());

// Send a cookie

app.get('/',(req,res)=>

{

res.cookie("uName","Sreeram",{expire : 36000+Date.now()})

res.end('Hello......');

})

//remove the cookie

app.get('/removeck', function(req, res){

res.clearCookie('uName');

res.send('Cookiee Removed ..........');

});

app.listen(4060, function(){

console.log('Server started...');

})

**Body-Parser thirdparty data middleware using**

* Check the documentation in ExpressJS Site

It parses the incoming request body ,

Provides four parsers

JSON body parser

Raw body Parser

Text body parser

url – encode form body parser

**bodyParser.Json()** – returns middleware that only parse JSON and looks where the content type matches

**extended** Option : Allows for rich encoding for objects and arrays also . extended = true

Ex:

// Body-parserEx;

var express = require('express');

var bodyparser = require('body-parser');

var path = require('path');

var app = express();

app.use(express.static(path.join(\_\_dirname,"public")))

app.use(bodyparser.json());

app.use(bodyparser.urlencoded({extended:true}));

app.post('/', function(req,res){

res.send(JSON.stringify(req.body));

});

app.listen(4060, function(){

console.log('Server started...');

})

Index Page as public

<html>

<body>

<form action="/" method="POST">

Enter Movie Name : <input type="text" name="movieName" />

<input type="submit" />

</form>

</body>

</html>

**Response-time**

Creates the middle ware that records the response time for request in http server .

resTime is Elapsed time from when the req enters the middleware and to when the

headers are written to client

Ex:

var express = require('express');

var app = express();

var respTime = require('response-time');

app.use(respTime());

app.get('/', function(req,res){

setTimeout(function(){

res.status(200).end();

}, 513)

});

app.listen(4060, function(){

console.log('Server started...');

})

**Morgan middleware (Earlier is logger)**

Used to log the request

var express = require('express');

var morgan = require('morgan');

var app = express();

app.use(morgan(":method :status : url"));

//app.use(morgan("Format")) - format can be copied from site

app.get('/morgan', function(req,res){

res.send('Morgan Logging ...')

});

app.listen(4060, function(){

console.log('Server started...');

})

**Connect-timeout**

Ex: Use to set the time out for a request

var express = require('express');

var connectTimeOut = require('connect-timeout')

var app = express();

// app.use(connectTimeOut('2s')) - will sets the timeout for all the requests

app.get('/Home.html', connectTimeOut('2s'), function(req,res){

// write some thing to delay here

res.send('hello.....');

})

app.listen(4060, function(){

console.log('Server started...');

})

* **Discusss and work on other middleware – Error Handling , compression**

**Working with MongoDB Connectivity**

MongoDB

27017

Express JS

REST URI

Express Appln

Anglr / React

Appln

Mongoose

Code

Explain CRUD Operations with Mongo

Documentdb id like mongoDB , and fatser . its automatically generate the REST api also . documentDB is cloudBased

**Working with REST**

Its faster as it Uses inbuilt cache

We can create REST API using Express

It is necessary that the API can be accessed to any other clients

The REST API is available for the clients which are outside , if the client hit the URL , it will get the data

No need of views . no need of setting views

All the REST should be accessed with api in the URL

Ex: Amazon <------🡪 IDBI bank

Postman is the application developed by google for test the API uri REST . it is an extension for chrome browser

REST will work with data basing on methods

POST

PUT

DELETE

GET

For chrome we can use postman for checking REST Execution

For Mozilla we can use - RESTclient

**Postgresql REST Api**

REST services create by exprejs can be used by any front end application

Express JS

Anglr / React

Appln

postgres

5432

REST URI

Any Client Appln

pg-promise

**Third party middleware required**

Npm install bluebird -–save

npm install pg-promise -–save

npm install body-parser -–save

**Route js File**

var express = require('express');

var promise = require('bluebird');

var bodyParser = require('body-parser');

var options = {

// Initialization Options

promiseLib: promise

};

var pgp = require('pg-promise')(options);

var app = new express();

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

var connectionString = 'postgres://postgres:rama@123@localhost:5432/ossbatch1';

var db = pgp(connectionString);

app.get('/', (req, res, next) => {

db.any('select \* from employee')

.then(function (data) {

console.log(data);

res.send(data);

})

})

app.get('/:id', (req, res, next) => {

var i = parseInt(req.params.id);

db.any('select \* from employee where eno = $1' ,i)

.then(function (data) {

console.log(data);

res.send(data);

})

})

app.post('/', (req, res, next) => {

var n = req.body.eno;

var nm = req.body.ename;

var d = req.body.deptid;

var m = req.body.mngrid;

var c = req.body.city;

var s = req.body.salary;

db.none('insert into employee values($1,$2,$3,$4,$5,$6)', [n, nm, d, m, c, s]).then(function () {

console.log('Record Inserted ...')

res.status(200).send({message:"Inserted Suxces.."})

})

})

app.put('/:id', (req, res, next) => {

var i = parseInt(req.params.id);

var nm = req.body.ename;

var d = req.body.deptid;

var m = req.body.mngrid;

var c = req.body.city;

var s = req.body.salary;

db.none('update employee set ename= $1, deptid =$2, mngrid=$3, city =$4, salary =$5 where eno=$6',[ nm, d, m, c, s, i]) .then(function () {

console.log('Record UPdated Success ...')

res.status(200).send({message:"Updtaed Suxces.."})

})

})

app.delete('/:id', (req, res, next) => {

var i = parseInt(req.params.id);

db.result('delete from employee where eno=$1', i).then(function (data) {

res.status(200).send({message:"Deleted Suxces.."})

})

})

app.listen(4400, (err) => {

if (err) {

console.log('Unable to start server ...')

}

else {

console.log('server Started at : ' + 'http://localhost:4400/')

}

})