Express JS

# Introduction of Express JS

Fast , Unopinionated minimalist web framework for Node.js Backend

# Installation

* Node js app creation
* Make a directory
* Open a cmd or terminal on that directory and type

#### npm init

* Enter general information of node js app (app name, version, author etc.)
* Add Express npm install express --save
* Create index.js contains
* const express = require('express')

const app = express()

const port = 3000

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

res.send('Hello World!')

})

app.listen(port, () => {

console.log(`Example app listening at http://localhost:${port}`)

})

Explanation

require(‘express’) *for importing express*

const app = express() *for making an object of express app*

app.get(‘/’,(req,res)) *app.get, app.post, for getting a request*

res.send( ) *for sending a response*

app.listen(port) *for start listening app or defined port*

# Run app

* node index.js  *for run an app on defined port*
* nodemon index.js *for run an app and show consoles and errors on cmd or terminal*
* *Install nodemon using* npm i -g nodemon
* nodemon index.js

# Use Express as Api

const express = require('express')

const app = express()

const port = 3000

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

res.send(

[{

name: 'Jemish',

roll\_no: 04

},

{

name: 'Kittu',

roll\_no: 37

}]

)

})

app.listen(port, () => {

console.log(`Demo app listening at http://localhost:${port}`)

})

# Middleware

Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the application’s request-response cycle.

To be continue ….

# Session

There are two main ways to store the sessions with cookies

1. cookie-session npm i cookie-session
   1. Not require any Databases/Resources on the Server Side
   2. Though total sessions data is limited to browser’s max cookie size
   3. Can be used for light sessions

Example :

1. Import

var cookieSession = require('cookie-session')

1. Use

app.use(cookieSession({

name: 'session',

keys: ['enckey1', 'enckey2'],

maxAge: 24 \* 60 \* 60 \* 1000 // 24 hours

}))

1. Set Session

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

if (!req.session.user) {

req.session.user = 'Captain'

res.send('Session Not Set')

} else {

res.send('Session is Set')

}

})

1. Destroy Session

req.session = null;

1. express-session npm i express-session
   1. Session data is not saved in the cookie itself, just the session ID. Session data is stored server-side.
   2. Its Server Side Session so whenever Express App is Restated Cookies will Destroy itself

Example :

1. Import

app.use(session({

secret: 'secret\_key',

resave: true,

saveUninitialized: true,

cookie: { secure: false }

}))

1. Use

app.use(session({

secret: 'secret\_key',

resave: true,

saveUninitialized: true,

cookie: { secure: false, maxAge: 6000 }

}))

1. Set & Destroy Sessions

Same as cookie-session

# MongoDB

There are several ways to connect mongodb via nodejs

1. Mongodb official Driver
   1. Install via npm : npm i mongodb
   2. Import mongodb

var MongoClient = require('mongodb').MongoClient

* 1. Connect

MongoClient.connect('mongodb://localhost:27017/marvel', { useUnifiedTopology: true }, function (err, client) {

if (err) throw err

var db = client.db('marvel')

db.collection('avengers').find().toArray(function (err, result) {

if (err) throw err

console.log(result)

})

})

1. Mongoose
   1. Install via npm : npm install mongoose --save
   2. Import mongoose and schema

const mongoose = require("mongoose")

const { Schema } = mongoose;

* 1. Connect the Database

mongoose.connect('mongodb://Jemish:jemish@localhost:27017/library?authSource=admin', { useNewUrlParser: true, useUnifiedTopology: true });

* 1. Check the connection

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'connection error:'));

db.once('open', function () {

console.log('Database Connected')

});

* 1. Create a Schema

const schema = new Schema();

const blogSchema = new Schema({

title: String, // String is shorthand for {type: String}

author: String,

body: String,

comments: [{ body: String, date: Date }],

date: { type: Date, default: Date.now },

hidden: Boolean,

meta: {

votes: Number,

favs: Number

}

});

* 1. Create A Model of Schema

const Blog = mongoose.model('Blog', blogSchema);

* 1. Create a Record and Insert

Blog.create({name : 'Harry Potter'})

Create is used for insert a record

# Understanding CORS

By Default Express Api doesn't give Access from the Another Domain \

For that time if we want to use express apis then we need to configure cors

Without configuring cors you'll get this error

Access to XMLHttpRequest at 'http://localhost:3000/isGameOn' from origin 'http://localhost:8080' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.

For use CORS

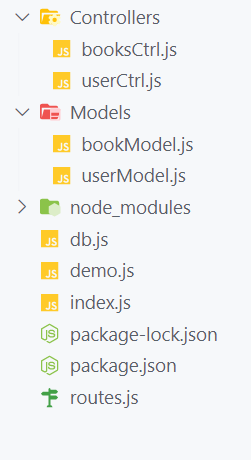
1. Install CORS lib via npm from Express App : npm install cors
2. Use and give access to these domains

const cors = require('cors');

app.use(cors({ origin: ['http://192.168.0.105:8080', 'http://localhost:8080', 'http://192.168.0.105:28499'] }));

* It will you access for the requests from the these Domains

# MVC Structure



Here we use NodeJS express framework for only Rest Api so I’m not going to import any views

1. **Index.js => Create Index file on the Root Directory**

const express = require('express')

const app = express()

const routes = require("./routes")

const db = require("./db")

app.use("", routes)

const port = 9000

app.listen(port, () => {

console.log(`\n MVC app listening at http://localhost:${port}`)

})

1. **Db.js => Create Database Connection File in Root Directory**
2. **Routes.js => Create Route file on the Root Directory**

var express = require('express');

var router = express.Router();

var books = require('./Controllers/booksCtrl');

var users = require('./Controllers/userCtrl');

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

res.send('Hello World');

})

// Books Routes

router.get('/add-books', books.AddBooks)

router.get('/books', books.AllBooks);

// Users Routes

router.get('/users', users.AllUsers);

router.get('/add-users', users.AddUsers);

module.exports = router;

**Note : Here I’m importing Controllers for use their functions**

1. **BooksCtrl.js => Create a Controllers and functions**

const booksModel = require('../Models/bookModel')

exports.AllBooks = async (req, res) => {

let resp = { status: false, data: [], msg: 'Invalid Request' };

let data = await booksModel.find({});

resp.data = data;

resp.status = true;

resp.msg = 'Book Data'

res.send(resp)

}

exports.AddBooks = async (req, res) => {

const BookData = {

name: "And Then There Were None",

author: "Agatha Christie",

price: 1350,

releaseYear: 2010,

}

let add = await booksModel.create(BookData)

res.send(add)

}

**Note : Here I’m importing Model for use them for Crud Operations**

1. **BookModel.js => Create Database Model**

const mongoose = require("mongoose")

const schema = mongoose.Schema({

name: {

type: String, alias: 'Name'

},

author: {

type: String, alias: 'Author',

default: Date.now

},

price: {

type: Number, alias: 'Price'

},

releaseYear: {

type: Number, alias: 'Release Year'

},

isAvailable: {

type: Boolean, alias: 'Available',

default: true

},

}, { strict: false })

module.exports = mongoose.model("book", schema)

**Note :Here {strict : false} make sense, after retrieving query result it gives facility to change query results, add new fields to query results**