|  |  |  |  |
| --- | --- | --- | --- |
| **1. Welcome to Our Course** | |  | |
| [1. Introduction to Course FREE PREVIEW](https://twinklecats.thinkific.com/enroll/512417?et=free_trial) | |  | |
| [2. What is NodeJS FREE PREVIEW](https://twinklecats.thinkific.com/enroll/512417?et=free_trial) | |  | |
| [3. Requirements of This Course FREE PREVIEW](https://twinklecats.thinkific.com/enroll/512417?et=free_trial) | |  | |
| [4. Installing NodeJS FREE PREVIEW](https://twinklecats.thinkific.com/enroll/512417?et=free_trial) | |  | |
| **2. NodeJS Module System** | |  | |
| 1. Introduction to NodeJS | | Nodejs এর মাধ্যমে জাভাস্ক্রিপ্টের কোড পিসিতে রান করা যায়।নোডজেএস এর ফাংশনালিটি কিভাভে এপ্লাই করতে হয়্ তাই শিখতে হবে।নোডজেএস এর অনেক ফিচার আছে যা নোডজেএস ছাড়া ব্রাউজারে করা যায় না। | |
| 2. What is Global Object | | নোডজেএস এ কোন উইন্ডো অবজেক্ট নেই।nodejs এ উইন্ডো অবজেক্ট নেই। এখানে আছে গ্লোবাল অবজেক্ট। এখানে ভেরিয়েবল বা মেথড যা কিছুই ব্যবহার করা হউক না কেন তা গ্লোবাল এর সাথে এটাচ হয়ে যাবে ।  //console.log(\_\_dirname) //কোথায় আছি সে লোকেশন প্রিন্ট করবে।  // console.log(\_\_filename) //ফাইলের নামসহ কোথায় আছি সে লোকেশন প্রিন্ট করবে।  ডকুমেন্টেশেন দেখতেই হবে। ডকুমেন্টেশন না দেখলে ভাল মানের ডেভলপার হওয়া যায় না।  <https://nodejs.org/api/>  <https://nodejs.org/api/globals.html>  উইন্ডো অবজেক্ট ও গ্লোবাল অবজেক্টের এর মধ্যে কিছু পার্থক্য আছে।আমরা কোন ফাইলের মধ্যে কোন ভেরিয়েবল বা মেথড ডিক্লার করলে তা গ্লোবাল অবজেক্টের সাথে এটাচট হবে না। | |
| 3. What is Module | | নোডজেএস এ প্রতিটি ফাইলই হলো এক একটি মডিউল বা ছোট লাইব্রেরী। যেমন math.js ১টি মডিউল যেখানে মেথ সংক্রান্ত সব ফাংশন আছে।http.js ফাইলে এইচটিটিপি সংক্রান্ত সকল ফাংশন আছে। নোডজেএস ১টি ফাইলকে কল্পনা করে ১টি নির্দিষ্ট মডিউল হিসেবে।  console.log(module) মডিউল একটি অবজেক্ট। | |
| 4. Creating Our First Module | | Math.js ফাইলে নিচের কোড গুলো আছে। exports করার 4টি উপায় আছে।  exports.add = (a, b) => a + b // add মেথডের আছে exports কিওয়ার্ড টি ব্যবহার করে অন্য ফাইলে এক্সপোর্ট করা হয়েছে।এটি ১টি উপায়  const sub = (a, b) => a - b  const div = (a, b) => a / b  module.exports.test = (a, b) => add(a, b) / sub(a, b) // সিংগল ভেরিয়েব বা মেথড এক্সপোর্ট করার জন্য  // module.exports = {  // test,  // add  // } // ডেসটাক্সার করে  // module.exports.add = add // এভাবে প্রতিটি পৃথকভাবে  // module.exports.test = test | |
| 5. Import Our Created Module | | const math = require('./math') // math.js ফাইল হতে ইমপোর্ট করা হয়েছে।  const { add } = require('./math') //ডেসটাক্চার করেও নেওয়া যায়।  // console.log(math)  console.log(math.add(1, 2)) | |
| 6. NodeJS Path Module | | const path = require('path')  const myCurrentPath = \_\_filename  console.log(\_\_filename) //অবস্থানসহ বর্তমান ফাইলের নাম দেখাবে  console.log(myCurrentPath)  console.log(path.basename(\_\_dirname)) //বর্তমান ডারেক্টরী দেখাবে  console.log(path.extname(myCurrentPath)) //বর্তমান ফাইলের এক্সটেনশন দেখাবে  let pathObj = {  dir: 'usr/local',  name: 'testFile',  ext: '.js'  }  console.log(path.format(pathObj)) // অবজেক্ট কে পাথে রুপান্তর করে  console.log(path.isAbsolute(myCurrentPath)) // এবসুলেট পাথ কি না চেক করে  console.log(path.isAbsolute('./math.js')) / ফলস রিটার্ণ করবে  console.log(path.join('usr', 'local', 'hmnayem', 'testfile.js')) //প্লাটফর্ম অনুযায়ী পাথ রিটার্ণ  করবে (উইন্ডোজ/লিনাক্স/মেক অনুযায়ী)  console.log(path.resolve(\_\_dirname, 'scripts', 'test.js')) //এটি অবজেক্ট রিটার্ণ করবে।  console.log(path.parse(myCurrentPath)) | |
| 7. NodeJS OS Module | | const os = require('os')  console.log(os.arch())  console.log(os.cpus())  console.log(os.freemem())  console.log(os.networkInterfaces())  console.log(os.hostname())  console.log(os.homedir())  console.log(os.type())  console.log(os.userInfo()) | |
| 8. Write Javascript Object to File | | const fs = require('fs')  const testObj = { //জাভাস্ক্রিপট অবজেক্ট  name: 'HM Nayem',  email: 'hmnayem@gmail.com',  address: {  city: 'Dhaka',  country: 'BD'  }  }  const data = JSON.stringify(testObj) // JSON.stringify ফাংশনের মাধ্যমে অবজেক্টকে জেসন  অবজেক্ট তৈরী করা হয়।  fs.writeFile('./test.json', data, (err) => { // fs.writeFile(’ফাইল পাথ’, জেসন অবজেক্ট , কলব্যাক ফাংশন)  if (err) {  console.log(err)  } else {  console.log('File Write Successful')  }  }) | |
| 9. Read File As Javascript Object | | const fs = require('fs')  fs.readFile('./test.json', (err, data) => {  if (err) {  return console.log(err)  }  let obj = JSON.parse(data) // JSON.parse() মেথড জেসন ফাইল কে জাভাস্ক্রিপ্ট অবজেক্টে রুপান্তর  করে।  console.log(obj.address)  }) | |
| 10. NodeJS Http Module | | const http = require('http') //এইচটিটিপি মডিউল ইমপোর্ট করা হয়েছে।  const server = http.createServer((req, res) => { // createServer(কলব্যাংক ফাংশন)  console.log(req.url) // req মানে রিকুয়েষ্ট, res মানে রেসপন্স  res.end('<h1>Hello NodeJS, You Are Beauty</h1>') // req.url মানে কোথায় থেকে রিকুয়েষ্ট পাঠানো হয়েছে।  }) // res.end('html code পাঠানো যায়')  server.listen(4141, () => { // http.createServer(callback).listen(port, callback)  console.log('Server is Running on PORT 4141')  }) // পূণরায় দেখবো\*\*\*\*\*\*\*\*\* | |
| 11. Fun Project with File And HTTP Module | | const fs = require('fs')  const http = require('http')  const server = http.createServer((req, res) => {  fs.readFile('./index.html', (err, data) => { // fs.readFile('file path', callback)  res.write(data)  res.end()  })  })  server.listen(4141, () => console.log('Server is Running')) | |
| Source Code | |  | |
| **3. Node Package Manager (NPM)** | |  | |
| 1. What is Node Package Manager (NPM) | | **// আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 2. How to Enable NPM in Our Project | | Npm init -y // -y এর মাধ্যমে সবকিছুকে ওকে করা হয়েছে। এর মাধ্যমে package.json ফাইল তৈরী হবে যাতে সকল তথ্য থাকে। | |
| 3. What is Package.JSON | | package.json এর বিবরণ **// আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 4. How to Install Node Packages | | Npm i chalk (npm install chalk), 1টি পেকেজ ইনস্টল করলে এটি যদি অন্য কোন পেকেজের উপর ডিপেন্ডেট হয় তাহলে সকল ডিপেন্ডেট পেকেজগুলো ইন্সটল হয়ে যাবে। প্যাকেজের মধ্যেও package.json ফাইল থাকে। এনপিএম এর মাধ্যমে কোন পেকেজ ইনেবল করলে package-lock.json ফাইলও তৈরী হয়। package-lock.json ফাইলে ভার্সন সংক্রান্ত তথ্য থাকে। এ ফাইলে কোন কাজ করার প্রয়োজন নেই। এটি অটোমেটিক আপডেট হয়। | |
| 5. What is Dependecy | | "dependencies": {  "chalk": "^2.4.2"  }, **// আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 6. What is Dev Dependency | | "devDependencies": {  "nodemon": "^1.19.1"  } // devDependencies হলো এমন পেকেজ যেগুলো প্রজেক্ট ডেভলপ করার সময়ে প্রয়োজন হয় অর্থাত ডেভলপ করার কাজ কে সহজ করে দেয়।কিন্ত প্রজেক্ট শেষ হওয়ার পর রান করার জন্য আর প্রয়োজন নেই।যেমন nodemon প্রজেক্ট ডেভলপ করা কালীন সময়ে বার বার রান করা হতে রক্ষা করে।কিন্ত প্রজেক্টের কাজ শেষ হলে এর আর প্রয়োজন নেই। **devDependencies ইন্সটল করার উপায় হলো npm install --save-dev packageName (এটি আরও সহজে করা যায় এভাবে npm i -D** nodemon **)// 3:00** এর পর দেখা যেতে পারে। **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 7. How to Install Global Packages | | ৩ ধরনের ডিপেনডেন্সী আছে। নরমাল ডিপেনডেন্সী (যা প্রজেক্টের জন্য দরকার), ডেভডিপেনডেন্সী (প্রজেক্ট ডেভলপ করার সময়ে দরকার, প্রজেক্ট কমপ্লিট হওয়ার পর আর দরকার নেই), গ্লোবাল ডিপেনডেন্সী (যা প্রতিটি প্রজেক্টের জন্য দরকার হবে তা প্রতিটি প্রজেক্টের জন্য আলাদা আলাদা ইন্সটল না করে গ্লোবালী ইন্সটল করা যায়, গ্লোবালি ইন্সটল করলে package.json ফাইলে এটি এড হবে না, গ্লোবাল নামের একটি জায়গায় এটি এড হবে ) **npm i -g nodemon** **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 8. Install Package Via Package.JSON | | **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 9. How to Write Custom Scripts | | package.json ফাইলের স্ক্রিপ্ট টেগ ব্যবহার করে আমরা অনেক কিছু করতে পারি। যেমন চেন্জ ডিরেক্টরী, প্রজেক্ট বিল্ড করা, যেকোন অপারেটিং সিস্টেম এর কমান্ড রান করতে পারি, সার্ভার রান করা ইত্যাদি  "scripts": {  "start": "nodemon index.js", // index.js ফাইল টি nodemon দ্বারা শুরু হবে। এ কমান্ড টি npm  start অথবা npm run start এভাবে রান করা যায়। ২টি কমান্ডই  সেম কাজ করে।  "prod": "node index.js" // এ কমান্ড টি প্রডাকশন ভার্সনের জন্য। এখানে prod না দিয়ে অন্য  কোন শব্দও ব্যবহার করা যায়। npm run prod  } //**আর দেখার খুব একটা প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 10. How to Uninstall a Package | | **Npm uninstall packageName** যেমন npm **uninstall faker** কমান্ড দিলে পেকেজটি আনইনস্টল  হয়ে যাবে। **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| Source Code | |  | |
| **4. Introduction to ExpressJS** | |  | |
| 1. Intro | | **ExpressJS আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 2. What is Web Framework | | Framework পরিচিতি **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 3. NodeJS Frameworks | | **Nodejs Framework list** : AdonisJs, **expressjs, Meteor.js** (full stalk framework), Nest.js, Sails.js, Koa.js, LoopBack.js, Hapi.js, Derby.js, Total.js ইত্যাদি।**nodejs ও expressjs ভালভাবে শিখার আগে দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 4. What is ExpressJS | | ExpressJS পরিচিতি **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 5. Installing ExpressJS | | Npm init -y  Npm i express  Touch app.js // Touch app.js কমান্ড এর মাধ্যমে ফাইল তৈরী করা হয়েছে।  Code . // বিএস কোড রান করা  package.json ফাইল কাস্টমাইজ করা  "main": "app.js",  "main": "index.js" এর পরিবর্তে "main": "app.js"  "start" : "node app.js"  // code in app.js  const express = require('express')  console.log(express)  সব শেষে npm start কমান্ড দিয়ে রান করা  Cat package.json  //কমান্ড দিলে দেখা যাবে ডিপেন্ডডেন্সীর ভিতরে express পেকেজটি আছে। cat filename.extension কমান্ডের দিয়ে ফাইল পড়া হয়। **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 6. First Express Server | | এইচটিটিপি সার্ভার মডিউল দিয়ে সার্ভার বানো যায়। কিন্তু এক্সপেস এ এভাবে সার্ভার বানাতে হবে না। | |
| 7. Handle Our First Route | | const express = require('express')  const app = express()  app.get('/', (req, res)=> { // **app.method('route url', callback)**  res.send('<h1>I am listening</h1>')  } )  const PORT = process.env.PORT || 8080  app.listen(PORT, ()=> {  console.log('Sever is Running on PORT ${PORT}')  })  **// app.listen(8080, callback) এভাবে দেওয়া যায়। দেখার প্রয়োজন নেই।** | |
| 8. Create Multiple Routes | | const express = require('express')  const app = express()  app.get('/about', (req, res)=> {  res.send('<h1>I am about page</h1>') } )  app.get('/help', (req, res)=> {  res.send('<h1>I am help page</h1>') } )  app.get('/', (req, res)=> {  res.send('<h1>I am listening</h1>') } )  **app.get('\*', (req, res)=> {**  **res.send('<h1>404 Not Founding</h1>') } )**  const PORT = process.env.PORT || 8080  app.listen(PORT, ()=> {  console.log('Sever is Running on PORT ${PORT}')  })  **সব সময় root route এর উপরে অন্যান্য route গুলো ডিকলার করতে, হবে 404 পেজের জন্য root route এর নিচে '\*' ডিকলার করতে হবে।আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 9. What is Request Object | | <http://expressjs.com/en/5x/api.html#req> প্রয়োজনে কয়েকবার দেখবো এবং সংশ্লিষ্ট পেজে গিয়ে দেখবো | |
| 10. What is Response Object | | app.get('/json', (req, res)=> {  res.json({ message: 'I am a response from your route handler' }) } )  res.send(Buffer.from('whoop'))  res.send({ some: 'json' })  res.send('<p>some html</p>')  res.status(404).send('Sorry, we cannot find that!')  res.status(500).send({ error: 'something blew up' })  res.json() এভাবে আমরা জেসন অবজেক্ট সেন্ট করতে পারি। এইচটিএমএল ফাইল সেন্ট করতে পারি।১টি ইমেজ সেন্ট করতে পারি।কাস্টম হেডার সেন্ট করতে পারি।  প্রয়োজনে কয়েকবার দেখবো এবং সংশ্লিষ্ট পেজে গিয়ে দেখবো ।হবে।  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 11. HTTP Methods | | <https://www.restapitutorial.com/lessons/httpmethods.html> ভাল করে পড়ে দেখবো। | |
| 12. Installing Nodemon | | হবে।**আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| Source Code | | হবে।**আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| **5. Working with Express Middleware** | |  | |
| 1. What is Middleware | | **Middleware হচ্ছে এক প্রকার ফাংশন । Middleware ফাংশনের মাধ্যমে ইউজারের পাঠানো** Request কে চেক করা হয়।হবে।বিভিন্ন **Middleware বিভিন্ন কাজ করে থাকে। যেমন কোন Middleware শুধু ভেলিডেশনের কাজ করে।কোন Middleware ইউজার লগিন আছে কিনা চেক করে।**  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 2. Using Third Party Middleware | | Morgan ছোট ১টি Middleware । এটি ইনকামিং Request গুলোকে চেক করে।  const express = require('express')  const morgan = require('morgan')  const app = express()  app.use(morgan('dev')) // app.use(এখানে মিডলয়্যার ব্যবহার করতে হয়)  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 3. Using middleware to specific routes | | আমরা প্রতিটি রাউট এর ভিতরে মিডলয়্যার ব্যবহার করতে পারি।  app.use(morgan('dev')) // এভাবে ব্যবহার করলে সকল রাউট মিডলয়্যার টি পাবে।  app.get('/about', morgan('dev'), (req, res)=> {  res.send('<h1>I am about page. ..</h1>') } )    app.get('রাউট', মিডলয়্যার, কন্ট্রলার কলব্যাক ফাংশন ) //এখানে মিডলয়্যার এর জায়গায় কমা দিয়ে একাধিক মিডলয়্যার ব্যবহার করা যায়।  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 4. Our First Custom Middleware | | function customMiddleware(req, res, next){  if(req.url === '/help'){  res.send('<h1>Sorry, This page is blocked by admin</h1>') }  // res.send(এখানে এইচটিএমএল ফাইল, এইচটিএমএল কোড, জেসন কোড পাঠানো যায়)  next() // এ ফাংশনটি না দিয়ে রাউট লোড হবে না।  }  app.use(customMiddleware)  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 5. Tiny Logger Middleware | | প্রতিটি রাউট হেন্ডেলারই এক একটি মিডলয়্যার। রাউটের কন্ট্রলারেও next পাঠানো যায়।  //**এ অংশটি আবার দেখবো। কয়েক বার দেখবো।** | |
| 6. Express Middlewares | | <http://expressjs.com/en/resources/middleware.html> Express এর সকল মিডলয়্যার সম্পর্কে বিস্তারিত এখানে আছে। | |
| Source Code | | **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| **6. Working with Express Router** | |  | |
| 1. Introduction to Express Route | | আমরা যখন বড় কোন প্রজেক্ট করবো তথন app.js ফাইলের মধ্যে ‍app.get() এভাবে রাউট ব্যবহার করবো না। আমরা app.js ফাইল হতে রাউটগুলোকে আলাদা করার জন্য এক্সপ্রেস জেএস এর রাউটার ফাংশনটি ব্যবহার করবো।  http://expressjs.com > Resources > utility modules > router এখানে বিস্তারিত বিবরণ আছে।  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 2. How to Create Routes | | ইউজারের জন্য রাউটার থাকতে পারে, লাইকের জন্য রা্উটার থাকতে পারে, লগইনের জন্য রাউটার থাকতে পারে, কমেন্টের জন্য রাউটার থাকতে পারে, পেমেন্টের জন্য রাউটার থাকতে পারে, প্রতিটি কাজের জন্য রাউটার তৈরী করা যেতে পারে। রাউট নিয়ে গ্রোপ করা যেতে পারে,  const express = require('express')  const userRouter = express.Router() // এভাবে ইউজারের জন্য ১টি গ্রোপ করে আলাদা  const postRouter = express.Router() এভাবে পোস্টের জন্য ১টি গ্রোপ  const morgan = require('morgan')  const app = express()  //user router  const userRouter = express.Router() //এটি এক্সপ্রেসের ১টি ফাংশন  userRouter.get('/login', (req, res) => {  res.send('<h1>I am login route</h1>') })  userRouter.get('/logout', (req, res) => {  res.send('<h1>I am logout route</h1>') })  userRouter.get('/signup', (req, res) => {  res.send('<h1>I am signup route</h1>') })  // end user router  app.use('/user', userRouter) //এখানে user রাউট গ্রোপ করা হয়েছে।  (/user/login, /user/logout, /user/signup), app.use('/user', userRouter) এই অংশটি app.js ফাইলে থাকে। **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 3. Separate Route Files | | // userRoute.js file  const userRouter = require('express').Router()  userRouter.get('/login', (req, res) => {  res.send('<h1>I am login route</h1>') })  userRouter.get('/logout', (req, res) => {  res.send('<h1>I am logout route</h1>') })  userRouter.get('/signup', (req, res) => {  res.send('<h1>I am signup route</h1>')  })  //app.js file  const userRouter = require('./userRoute')  const postRouter =require('./postRoute')  app.use('/user', userRouter) //এখানে user রাউট গ্রোপ করা হয়েছে।  app.use('/posts', postRouter) //এখানে posts রাউট গ্রোপ করা হয়েছে।  // exports.router = userRouter //এইভাবেও এক্সপোর্ট করা যায়।  module.exports = userRouter //এইভাবেও এক্সপোর্ট করা যায়।  Router.js ফাইল তৈরী করে সেখানে ইউজার গ্রোপ রাউট অংশ নিয়ে যাওয়া হয়েছে।  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 4. Route And HTTP Methods | | const Router = require('express').Router()  Router.get('/', (req, res) => {  res.send('<h1>Render All Posts</h1>') })  Router.post('/', (req, res) => {  res.send('<h1>Create New Post</h1>') })  Router.put('/', (req, res) => {  res.send('<h1>update Post</h1>') })  Router.delete('/', (req, res) => {  res.send('<h1>Delete Post</h1>') })  module.exports = Router  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 5. Route Parameters | | app.get('/products/:prodId', (req, res) => {  res.send('I am listening') }) **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 6. Extract Route Parameters | | **app.get('/products/:prodId/reviews/:reviewId', (req, res) => {**  **console.log(req.params)**  **res.send('I am listening prodId :' + req.params.prodId + ' </br>review parameter:' + req.params.reviewId) })**  //প্রতিটি রিকুয়েস্ট অবজেক্ট এর সাথে params নামে ১টি প্রপার্টি আছে। রাউট আকারে যত পেরামিটার পাস করা হয় সব **পেরামিটার params এর মধ্যে থাকে।**  **Req.params এর মাধ্যমে** রিকুয়েস্ট অবজেক্ট এর তথ্য ধরা যায়।  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 7. Route Parameter for Posts | | const Router = require('express').Router()  Router.get('/', (req, res) => {  res.send('<h1>Render All Posts</h1>') })  Router.get('/:postId', (req, res) => {  res.send('I am POST = ' + **req.params.postId**) })  Router.post('/', (req, res) => {  res.send('<h1>Create New Post</h1>') })  Router.put('/:postId', (req, res) => {  res.send('<h1>update Post</h1> =' + req.params.postId) })  Router.delete('/:postId', (req, res) => {  res.send('<h1>Delete Post</h1> = ' + req.params.postId) })  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*** | |
| 8. Extracting Query String | | localhost:8080/posts?category=tech&page=108&filter=latest এভাবে রিকুয়েস্ট পাঠালে নিচের মতো করে রিকুয়েস্ট ধরা যায়।  let { category, page, filter } = req.query  //প্রতিটি রিকুয়েস্ট অবজেক্ট এর সাথে Query নামে আরও ১টি প্রপার্টি থাকে যেটি একটি অবজেক্ট। এটি কি ভেলূ পেয়ার হিসেবে থাকে। এটি কে ডেসটাক্চারিং করেও ব্যবহার করা যায়। localhost:8080/posts?category=tech&page=108&filter=latest | |
| 9. Seperate Controllers | | প্রতিটি রাউট এর সাথে একটি করে কন্ট্রলার রয়েছে।  যেমন userRouter.get('/login', (req, res) => {  res.send('<h1>I am login route 5</h1>') })  এখানে লাল লেখা অংশটি কন্ট্রলার। Route হতে Controller গুলো আলাদা ফাইলে সংরক্ষণ করা হয়েছে।  //postRoute.js  const Router = require('express').Router()  const { getAllPost, getSinglePost, createNewPost, updatePost, deletePost } =  require('./postController')  Router.get('/', getAllPost)  Router.get('/:postId', getSinglePost)  Router.post('/', createNewPost)  Router.put('/:postId', updatePost)  Router.delete('/:postId', deletePost)  // exports.router = userRouter //এইভাবেও এক্সপোর্ট করা যায়।  module.exports = Router //এইভাবেও এক্সপোর্ট করা যায়।  //postController.js  exports.getAllPost = (req, res) => {  let { category, page, filter } = req.query  res.send('Render All Posts') }  exports.getSinglePost = (req, res) => {  res.send('I am POST = ' + req.params.postId) }  exports.createNewPost = (req, res) => {  res.send('Create New Post') }  exports.updatePost = (req, res) => {  res.send('Update Your Existing Post = ' + req.params.postId) }  exports.deletePost = (req, res) => {  res.send('Delete Your Existing Post = ' + req.params.postId) }  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| Source Code | |  | |
| **7. Test Your Knowledge - Contact List APP** | |  | |
| 1. Intro | |  | |
| 2. Setup Our Project | | Npm init -y  Npm i express morgan  Npm i -D nodemon  Create app.js change index.js to app.js  ‍start nodemon app.js  const express = require('express')  const app = express()  const PORT = process.env.PORT || 8180  app.listen(PORT, () => {  console.log(`SERVER IS RUNNING ON PORT ${PORT}`) })  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 3. Create Contact Model | | class Contacts {  constructor(){  this.contacts = [] }  getAllContacts(){  return this.contacts }  getContactsById(id){  return this.contacts.find(contact => contact.id === id) }  createContact(contact){  contact.id = this.contacts.length + 1  this.contacts.push(contact)  return contact }  updateContactById(id, updatedContact){  let index = this.contacts.findIndex(contact => contact.id == id)  this.contacts[index].name = updatedContact.name || this.contacts[index].name  this.contacts[index].phone = updatedContact.phone || this.contacts[index].phone  this.contacts[index].email = updatedContact.email || this.contacts[index].email  return this.contacts[index]  }  deleteContactById(id) {  let index = this.contacts.findIndex(contact => contact.id === id)  let deletedObj = this.contacts[index]  this.contacts = this.contacts.filter(contact => contact.id !== id)  return deletedObj  } }  module.exports = new Contacts() | |
| 4. Create Contact Routes | |  | |
| 5. Create and Get Contacts | |  | |
| 6. Update And Delete Contact | |  | |
| 7. Test Our Project | |  | |
| Source Code | |  | |
| **8. Working with Template Engine in Express** | |  | |
| 1. What is Template Engine | | Done **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 2. Template Engines Supported By Express | | [http://expressjs.com](http://expressjs.com/) -> Resources -> Temmplate engines popular(pug, hbs, ejs) https://ejs.co | |
| 3. Embedded Javascript (EJS)First Look | | EJS **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 4. How to Setup EJS in Express APP | | Npm i ejs  Express js ১টি ভিউ ইঞ্জিন প্রোবাইড করে থাকে। app.set('view engine', 'ejs') ‍এপ.জেএস ফাইলে app.set('view engine', 'ejs') এই লাইনের মাধ্যমে বলে দেওয়া যায় 'view engine' প্রতিস্থাপিত হবে 'ejs' দিয়ে। এক্সপ্রেস জেএস ডিফল্টভাবে সকল ভিউজকে ভিউজ নামক ডাইরেক্টরীতে রাখা হবে কল্পনা করে।আপনি যদি মনে করেন যে না সকল ভিউজকে অন্য কোন স্থানে রাখা হবে তাও আপনি করতে পারেন।(যেমন app.set('views', 'template/views') এখানে সকল ভিউজগুলোর লোকেশন টেম্লেট ডাইরেক্টরীর ভিউজ। (1:50মিনিট)) কিন্তু আমরা ডিফল্টভাবে সকল পেজগুলোকে views নামক ডিরেক্টরীর মধ্যে রাখবো। views ডাইরেক্টরীর মধ্যে সকল ফাইলগুলোই হবে ejs এক্সটেনশন ফাইল।  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 5. Render Some Dynamic Data | |  | |
| 6. Condition in EJS | | <% if (post.published) { %>  <h3> Title: <%= post.title %> </h3>  <p> Body: <%= post.body %> </p>  <% } else { %>  <p> Post is not yet published </p>  <% } %>  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 7. Loops in EJS | | <ul>  <% for (let p of posts) { %>  <li>  <%= p.title %> By <%= p.author %>  </li>  <% } %>  </ul>  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 8. Create Multiple Pages | |  | |
| 9. Working with Partials in EJS | | <%- include('../partials/head.ejs') %>  <%- include('../partials/nav.ejs') %>  <h1> <%= title %> </h1>  <p>this is a dummy text. this is a dummy text. this is a dummy text. this is a dummy text. this is a dummy text. this is a dummy text.</p>  <%= new Date().toString() %>  <% if (post.published) { %>  <h3> Title: <%= post.title %> </h3>  <p> Body: <%= post.body %> </p>  <% } else { %>  <p> Post is not yet published </p>  <% } %>  <ul>  <% for (let p of posts) { %>  <li>  <%= p.title %> By <%= p.author %>  </li>  <% } %>  </ul>  <%- include('../partials/footer.ejs') %>  // nav.ejs  <nav>  <ul>  <li> <a href="/"> Home </a> </li>  <li> <a href="/about"> About </a> </li>  <li> <a href="/help"> Help </a> </li>  <li> <a href="/test"> Test </a> </li>  </ul>  </nav>  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 10. Create Header Footer Challenge | | Same of 09 **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| Source Code | |  | |
| **9. How to Store Data in Express APP** | |  | |
| 1. Introduction to Databse | |  | |
| 2. What is MongoDB | |  | |
| 3. Download and Install MongoDB | | Mongodb নিয়ে কাজ করতে চাইলে আমাদের মনগোডিবির এক্সেস দরকার।মনগোডিবির এক্সেস আমরা দুইভাবে পেতে পারি।আমরা আমাদের লোকাল পিসিতে মনগোডিবি ইনন্সল করতে পারি অথবা আমরা মনগোডিবির ক্লাউড সার্ভার ব্যবহার করতে পারি।মনগোডিবি একটা রিয়েল টাইম সার্ভিস প্রোভাইড করে থাকে mongodb.com।সেখানে আমরা যদি একাউন্ট ক্রিয়েট করি আমরা মনগোডিবির ডাটাবেস ব্যবহার করতে পারি।আমাদের লোকাল মেশিনে কোন রকম ঝামেলা পোহাতে হবে না। ডিরেকলি তারা আমাদের একটি ইউআরএল প্রোভাইড করবে। সে ইউআরএল ব্যবহার করে মনগোডিবি কানেক্ট করবো।মনগোডিবির ক্লাউড সার্ভার ব্যবহার করতে চাইলে MongoDB Atlas এ যেতে হবে। | |
| 4. Introduction to MLAB | |  | |
| 5. Introduction to Mongoose | | **মংগুডিবি ডাটাবেজের সাথে অনেকগুলো উপায়ে কানেক্ট করা যায়।মংগুডিবি ড্রাইভার ইনস্টল করে খুব সহজেই নোডজেএসএ বা এক্সপেসজেএস এ মংগুডিবির কানেক্ট করা যায়। এক্ষেত্রে সমস্যা হচ্ছে যদি কোন ORM Tools ব্যবহার না করলে সবগুলো কোয়েরীগুলোকে নিজে থেকে লিখতে হবে।অর্থাত একই টাইপের কোড বার বার লিখতে হবে।এই সমস্যা সমাধান করার জন্য বিভিন্ন ORM tools থাকে।ORM মানে হচ্ছে অবজেক্ট রিলেশনাল মেপিং। Mongoose হলো সেরকম একটি টুলস যা ব্যবহার করে মংগুডিবি ডাটাবেজের সাথে সকল কাজ যেমন ডাটা ডিলিট করা, ডাটা মুছে ফেলা, ডাটা কুয়েরী করা ও ডাটা আপডেট করার কাজগুলো খুব সহজে করা যায়।**  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 6. Connect Database Using Mongoose | | **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 7. First Data Saved to MongoDB | | **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 8. Design Contact Schema | |  | |
| 9. Create Contact Model | |  | |
| 10. Router and Controller for Contacts | |  | |
| 11. Create New Contact | |  | |
| 12. Query Contacts From Database | |  | |
| 13. Update And Delete Contact | |  | |
| 14. Finishing Touch | |  | |
| Source Code | |  | |
| **10. Test Your Knowledge - Contact List APP with Frontend** | | | |
| 1. Intro | | Npm i ejs express mongoose morgan কমান্ড দিয়ে ইনস্টল করে নিতে হবে। npm i -D nodemon ইনস্টল দিতে হবে। app.js এর মধ্যে ‍app.se('view engine', 'ejs') এই লাইনটির মাধ্যমে ইজেএস সেট করে নিতে হবে।তারপর ভিউজ ফাইল তৈরী করতে হবে।  <form action="/contacts" class="form-inline" method="POST" id="my-form">  <div class="form-group mr-3">  <input type="text" name="name" placeholder="Name" class="form-control <%= error.name ? 'is-invalid': '' %>">  <div class="invalid-feedback">  <%= error.name %>  </div>  </div>  <div class="form-group mr-3">  <input type="text" name="phone" placeholder="Phone" class="form-control <%= error.phone ? 'is-invalid': '' %>">  <div class="invalid-feedback">  <%= error.phone %>  </div>  </div>  <div class="form-group mr-3">  <input type="email" name="email" placeholder="Email" class="form-control <%= error.email ? 'is-invalid': '' %>">  <div class="invalid-feedback">  <%= error.email %>  </div>  </div>  <div class="form-group mr-3">  <input type="hidden" name="id">  </div>  <input type="submit" class="btn btn-primary" value="Submit">  </form>  <table class="table">  <thead>  <tr>  <th scope="col">Name</th>  <th scope="col">Phone</th>  <th scope="col">Email</th>  <th scope="col">Action</th>  </tr>  </thead>  <tbody>  <% for (let contact of contacts) { %>  <tr>  <td><%= contact.name %></td>  <td><%= contact.phone %></td>  <td><%= contact.email %></td>  <td>  <button  class="btn btn-warning btn-sm edit-btn"  data-name="<%= contact.name %>"  data-phone="<%= contact.phone %>"  data-email="<%= contact.email %>"  data-id="<%= contact.\_id %>"  >  Edit  </button>  <a href="/contacts/delete/<%= contact.\_id %>" class="btn btn-danger btn-sm">  Delete  </a>  </td>  </tr>  <% } %>  </tbody>  </table>  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 2. Validate Form Data | | Form এর মাধ্যমে যে ডাটা সাবমিট করা হয় সে ডাটাগুলো req.body এর মাধ্যমে ধরা যায়, console.log(req.body) করে দেখা যায়। req.body এর তথ্যগুলো ১টি এ্যারের মধ্যে থাকে। এই এ্যারেকে ডেসটাক্সরিং করা যায়। app.js ফাইলের মধ্যে app.use(express.urlencoded({ extended: true })) টি ব্যবহার করা হয়েছে। এর মাধ্যমে req.body তে আমরা ফর্মে সাবমিট করা ডাটাগুলো পেয়ে থাকি।  exports.createContact = (req, res) => { এই কন্ট্রলারের মধ্যে নিচের কোডের মাধ্যমে ফর্ম এ সাবমিটকৃত name, phone, email গুলো ভেলিডেট করা হয়েছে }  let error = {}  if (!name) {  error.name = 'Please Provide A Name' }  if (!phone) {  error.phone = 'Please Provide A Phone Number' }  if (!email) {  error.email = 'Please Provide an Email' }  let isError = Object.keys(error).length > 0  if (isError) {  Contact.find()  .then(contacts => {  return res.render('index', { contacts, error })  })  .catch(e => {  console.log(e)  return res.json({ message: 'Error Occurred' })  })  }  **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | |
| 3. Create New Contact | | exports.createContact = (req, res) => {  let { name, phone, email, id } = req.body  let error = {}  if (!name) { error.name = 'Please Provide A Name' }  if (!phone) { error.phone = 'Please Provide A Phone Number' }  if (!email) { error.email = 'Please Provide an Email' }  let isError = Object.keys(error).length > 0  if (isError) {  Contact.find()  .then(contacts => {return res.render('index', { contacts, error }) })  .catch(e => {  console.log(e)  return res.json({message: 'Error Occurred' })  }) }  if (id) {  Contact.findOneAndUpdate({ \_id: id },  { $set: { name, email, phone } })  .then(() => {  Contact.find()  .then(contacts => { res.render('index', {contacts, error: {}}) })  })  .catch(e => {  console.log(e)  return res.json({ message: 'Error Occurred' })  })  } else {  let contact = new Contact({ name, email, phone })  contact.save()  .then(c => {  Contact.find()  .then(contacts => { return res.render('index', { contacts, error: {} }) })  })  .catch(e => {  console.log(e)  return res.json({ message: 'Error Occurred' })  })  }  } | |
| 4. Delete Contact By ID | | exports.deleteContact = (req, res) => {  let { id } = req.params  Contact.findOneAndDelete({ \_id: id })  .then(() => {  Contact.find()  .then(contacts => { res.render('index', { contacts, error: {} }) })  })  .catch(e => {  console.log(e)  res.json({ message: 'Error Occurred' })  })  } | |
| 5. Edit Contact By ID (Front End) | | if (id) {  Contact.findOneAndUpdate({\_id: id },  { $set: { name, email, phone } })  .then(() => {  Contact.find()  .then(contacts => { res.render('index', {contacts, error: {}})  })  })  .catch(e => {  console.log(e)  return res.json({ message: 'Error Occurred' })  })  } | |
| 6. Edit Contact By ID (Backend) | |  | |
| Source Code | |  | |
| **11. Planning Our Final Project According to MVC** | | | |
| 1. What is MVC |  | | |
| 2. Why MVC is important |  | | |
| 3. What we are going to build | We are going to Build a blog with multiple user and user dashboard | | |
| 4. Create Our Project Structures | Npm init -y, create directory controllers, models, routers, views, create file app.js, (যে ডাইরেক্টরী তৈরী করা হল তাতে প্রজেক্ট নিয়ে কাজ করা যাবে কোন সমস্যা হবে না।, তবে কিছু কাস্টম মিডলয়্যার তৈরী করার জন্য আরও কিছু ডাইরেক্টরী তৈরী করা হবে) create directory middleware, আমাদের কিছু কনফিগার করা লাগতে পারে। যেমন প্রডাক্টশন লেভেলের কনফিগারেশন, ডেভেলপমেন্ট লেভেলের কনফিগারেশন তার জন্য ডাইরেক্টরী করা হবে config, create directory config, ইউটিলিটি টুলস রাখার জন্য একটি ডাইরেক্টরী থাকতে পারে।তার নাম হতে পারে utils। create directory utils। আমাদের অনেক রকমের ভেলিডেশন করার প্রয়োজন হতে পারে। ইউজার রেজিষ্ট্রেশন ফর্ম সাবমিট করবে। প্রোফাইল আপডেট করতে পারে। লগইন ফর্ম সাবমিট করতে পারে। এগুলো ভেলিডেশন করার জন্য একটি ডাইরেক্টরী রাখতে পারি। তার নাম হতে পারে validator । প্রজেক্টের স্টেটেক্‌স অনেক ফাইল থাকতে পারে। যেমন সিএসএস ফাইল, জাভাস্ক্রিফট ফাইল, ইমেজ ফাইল, আপলোডকৃত ফাইল ইত্যাদি রাখার জন্য একটি ডাইরেক্টরী তৈরী করতে হবে যার নাম হতে পারে public । public ডাইরেক্টরীর মধ্যে আরও ডাইরেক্টরী থাকতে পারে যেমন : styles, scripts, images, uploads ।দটগদ | | |
| **12. Find Models From Requirements** |  | | |
| 1. Why Finding Model at Beginning is Important | Model অত্যান্ত গুরুত্বপূর্ণ অংশ। মডেলের কাজটি অত্যান্ত নিখুঁতভাবে করতে হবে। প্রয়োজনে আবার দেখুন। | | |
| 2. Find Models in Our Project | আমাদের প্রজেক্টে USER, PROFILE, POST, COMMENT এ ৪টি মডেল থাকতে পারে। এছাড়া BOOKMARK নামেও ১টি মডেল রাখা যেতে পারে। LIKE এবং DISLIKE কে আমরা POST মডেল এর সাথে রাখতে পারি। | | |
| 3. Relationship Between Our Models | USER,  PROFILE,  POST,  COMMENT | | |
| 4. Create User Model | // Name, Email, Password and Profile  const { Schema, model } = require('mongoose')  const Profile = require('./Profile')  const userSchema = new Schema({  name: { type: String, trim: true, maxlength: 30, required: true },  email: { type: String, trim: true, required: true },  password: { type: String, required: true },  profile: { type: Schema.Types.ObjectId, ref: Profile }  }, {  timestamps: true  })  const User = model('User', userSchema)  module.exports = User | | |
| 5. Create Profile Model | // user, title, bio, profilePics, links: {fb, twi, }, posts, bookmarks  const { Schema, model } = require('mongoose')  const User = require('./User')  const Post = require('./Post')  const profileSchema = new Schema({  user: { type: Schema.Types.ObjectId, ref: User, required: true },  title: { type: String, trim: true, maxlength: 100 },  bio: { type: String, trim: true, maxlength: 500 },  profilePic: String,  links: { website: String, facebook: String, twitter: String, github: String },  posts: [ { type: Schema.Types.ObjectId, ref: Post } ],  bookmarks: [ { type: Schema.Types.ObjectId, ref: Post } ]  }, {timestamps: true})  const Profile = model('Profile', profileSchema)  module.exports = Profile | | |
| 6. Create Post Model | **// title, body, author, tags, thumbnail, readTime, likes, dislikes, comments**  **const { Schema, model } = require('mongoose')**  **const User = require('./User')**  **const Comment = require('./Comment')**  **const postSchema = new Schema({**  **title: { type: String, required: true, trim: true, maxlength: 100 },**  **body: { type: String, required: true },**  **author: { type: Schema.Types.ObjectId, ref: User, required: true },**  **tags: { type: [String], required: true },**  **thumbnail: String,**  **readTime: String,**  **likes: [{ type: Schema.Types.ObjectId, ref: User }],**  **dislikes: [ { type: Schema.Types.ObjectId, ref: User } ],**  **comments: [ { type: Schema.Types.ObjectId, ref: Comment } ]**  **},**  **{ timestamps: true })**  **const Post = model('Post', postSchema)**  **module.exports = Post** | | |
| 7. Create Comment Model | // post, user, body, replies  const { Schema, model } = require('mongoose')  const User = require('./User')  const Post = require('./Post')  const commentSchema = new Schema({  post: { type: Schema.Types.ObjectId, ref: Post, required: true },  user: { type: Schema.Types.ObjectId, ref: User, required: true },  body: { type: String, trim: true, required: true },  replies: [ { body: { type: String, required: true },  user: { type: Schema.Types.ObjectId, ref: User, required: true },  createAt: { type: Date, default: new Date()  }  }  ]  }, { timestamps: true })  const Comment = model('Comment', commentSchema)  module.exports = Comment | | |
| 8. Tweak Our Models A Little Bit |  | | |
| Source Code |  | | |
| **13. Handle Authentication in Our Project** |  | | |
| 1. Intro | //authRoute.js  const router = require('express').Router()  const {  signupGetController,  signupPostController,  loginGetController,  loginPostController,  logoutController  } = require('../controllers/authController')  router.get('/signup', signupGetController)  router.post('/signup', signupPostController)  router.get('/login', loginGetController)  router.post('/login', loginPostController)  router.get('/logout', logoutController)  module.exports = router  **সকল রাউটেই সংশ্লিষ্ট রা্উটের সাথে মিল রেখে নাম দেওয়া ভাল। রাউটের নামের সাথে Route শব্দটি রাখাও ভাল।কন্ট্রলারও রাউটের সাথে মিল রেখে নাম দেওয়া ভাল। যেমন রাউট authRoute.js এবং কন্ট্রলার authController.js ।**  const bcrypt = require('bcrypt')  const User = require('../models/User')  exports.signupGetController = (req, res, next) =>  {res.render('pages/auth/signup', {title: 'Create A New Account'})  }  exports.signupPostController = async (req, res, next) => {  let { username, email, password } = req.body  try {  let hashedPassword = await bcrypt.hash(password, 11)  let user = new User({ username, email, password: hashedPassword })  let createdUser = await user.save()  console.log('User Created Successfully', createdUser)  res.render('pages/auth/signup', {title: 'Create A New Account'})  } catch (e) {  console.log(e)  next(e)  }  }  exports.loginGetController = (req, res, next) => {  res.render('pages/auth/login', {title: 'Login to Your Account'})  }  exports.loginPostController = async (req, res, next) => {  let { email, password } = req.body    try {  let user = await User.findOne({ email })  if (!user) { return res.json({ message: 'Invalid Credential' }) }  let match = await bcrypt.compare(password, user.password)  if (!match) { return res.json({ message: 'Invalid Credential' })  }  console.log('Successfully Logged In', user)  res.render('pages/auth/login', {title: 'Login to Your Account'})  } catch (e) {  console.log(e)  next(e)  }  }  exports.logoutController = (req, res, next) => { } | | |
| 2. Setup App For Handling EJS | Npm i morgan ejs  // app.js  const express = require('express')  const morgan = require('morgan')  const mongoose = require('mongoose')  // Import Routes  const authRoutes = require('./routes/authRoute')  const app = express()  // Setup View Engine  app.set('view engine', 'ejs')  app.set('views', 'views')  // **app.set('views', 'এখানে যে নাম দেওয়া হবে তা হবে ভিউজ ফোল্ডার')**  // Middleware Array  const **middleware** = [  morgan('dev'),  express.static('public'),  express.urlencoded({ extended: true }),  express.json()  ]  **app.use(middleware)**  **// এখানে middleware এ্যারে ব্যবহার করে morgan('dev'), express.static('public'), express.urlencoded({ extended: true }), express.json() এগুলোকে ব্যবহার করা হয়েছে (13.2. Setup App For Handling EJS )।**    app.use('/auth', authRoutes)  **//সকল রাউট রুট রাউটের উপরে ব্যবহার করতে হবে।**  app.get('/', (req, res) => {  res.json({ message: 'Hello World' })  })  const PORT = process.env.PORT || 8080  mongoose.connect('mongodb://admin:pass123@ds343217.mlab.com:43217/exp-blog',  { useNewUrlParser: true })  .then(() => {  console.log('Database Connected')  app.listen(PORT, () => { console.log(`Server is running on PORT ${PORT}`) })  })  .catch(e => { return console.log(e) }) | | |
| 3. Create Our Partials | **views\partials\footer.ejs, head.ejs, bavigation.ejs**  **views\pages\auth\login.ejs, signup.ejs নিয়ে কাজ করা হয়েছে।** | | |
| 4. Create Signup Template | // views\pages\auth\Signup.ejs  <%- include('../../partials/head.ejs') %>  <div class="container my-container my-5">  <div class="row">  <div class="col-md-6 offset-md-3">  <div class="card card-body">  <h4 class="mb-3">Create A New Account</h4>  <form action="/auth/signup" method="POST">  <div class="form-group">  <label for="username">Enter Your User Name (Must Be Uniq)</label>  <input type="text" name="username" id="username" class="form-control">  </div>  <div class="form-group">  <label for="email">Enter Your Valid Email</label>  <input type="email" name="email" id="email" class="form-control">  </div>  <div class="form-group">  <label for="password">Enter A Strong Password</label>  <input type="password" name="password" id="password" class="form-control">  </div>  <div class="form-group">  <label for="confirmPassword">Confirm Your Password</label><input type="password" name="confirmPassword" id="confirmPassword" class="form-control">  </div>  <input type="submit" class="btn btn-primary" value="Signup">  </form>  <a href="/auth/login" class="my-3 text-center"> Already have An Account? Login Herte </a>  </div>  </div>  </div>  </div>  <%- include('../../partials/fotter.ejs') %>  </body>  </html> | | |
| 5. Create Signup Controller | **const bcrypt = require('bcrypt')**  const User = require('../models/User')  exports.signupGetController = (req, res, next) => {  res.render('pages/auth/signup', {title: 'Create A New Account'}) }  exports.signupPostController = async (req, res, next) => {  let { username, email, password } = req.body  try {  let hashedPassword = await bcrypt.hash(password, 11)  let user = new User({ username, email, password: hashedPassword })  let createdUser = await user.save()  console.log('User Created Successfully', createdUser)  res.render('pages/auth/signup', {title: 'Create A New Account'})  } catch (e) {  console.log(e)  next(e)  }  } | | |
| 6. Connect Database and Create First User | const PORT = process.env.PORT || 8080  mongoose.connect('mongodb://admin:pass123@ds343217.mlab.com:43217/exp-blog',  { useNewUrlParser: true })  .then(() => {  console.log('Database Connected')  app.listen(PORT, () => {  console.log(`Server is running on PORT ${PORT}`)  })  })  .catch(e => {  return console.log(e)  }) | | |
| 7. Secure Password By Hashing | Npm i bcrypt | | |
| 8. Create Login Template | //login.ejs  <%- include('../../partials/head.ejs') %>  <div class="container my-container my-5">  <div class="row">  <div class="col-md-6 offset-md-3">  <div class="card card-body">  <h4 class="mb-3">Login to Your Account</h4>  <form action="/auth/login" method="POST">  <div class="form-group">  <label for="email">Enter Your Valid Email</label>  <input type="email" name="email" id="email" class="form-control">  </div>  <div class="form-group">  <label for="password">Enter A Strong Password</label>  <input type="password" name="password" id="password" class="form-control">  </div>  <input type="submit" class="btn btn-primary" value="Signup">  </form>  <a href="/auth/signup" class="my-3 text-center"> Don't have an Account? Create One </a>  </div>  </div>  </div>  </div>  <%- include('../../partials/fotter.ejs') %>  </body>  </html>  //controllers\authController.js  exports.loginGetController = (req, res, next) => {  res.render('pages/auth/login', {title: 'Login to Your Account'})  }  // 14-exp-blog-auth\routes\authRoute.js  router.get('/login', loginGetController) | | |
| 9. Create Login Controller | // 14-exp-blog-auth\routes\authRoute.js  router.post('/login', loginPostController)  //controllers\authController.js  exports.loginPostController = async (req, res, next) => {  let { email, password } = req.body    try {  let user = await User.findOne({ email })  if (!user) {  return res.json({  message: 'Invalid Credential'  })  }  let match = await bcrypt.compare(password, user.password)  if (!match) {  return res.json({  message: 'Invalid Credential'  })  }  console.log('Successfully Logged In', user)  res.render('pages/auth/login', {title: 'Login to Your Account'})  } catch (e) {  console.log(e)  next(e)  }  } | | |
| 10. Conclusion | Conclusion **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | | |
| Source Code |  | | |
| **14. Validate Your Requested Data** |  | | |
| 1. What is Validation | **Validation ৩টি উপায়ে করা যায়। যথা front End Validation, Backend Validation ও Database Validation. front End Validation ক্লায়েন্ট সাইটে জাভাস্ক্রিপট বা জেকুয়ারী ব্যবহার করে করা হয়। front End Validation কে ইচ্ছা করলে ইউজার অভারটেক করতে পারেন। এ জন্য front End Validation গুত্বপূর্ণ নয়। Backend Validation ই হলো সবচেয়ে গুরুত্বপূর্ণ ভেলিডেশন। আর একটি হলো Database Validation। Database Validationও front End Validation মতো অপশনাল। আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | | |
| 2. Validation Techniques | অনেক উপায়ে Validation করা যায়। নিজে নিজে কোড লিখে কাস্টম ভেলিডেশন করা যায়। এর সুবিধা ও অসুবিধা ২টিই আছে। Validation যেহেতু প্রায় একই রকম হয়, তাই Validation এর জন্য অনেক থার্ড পার্টি লাইব্রেরী পাওয়া যায়। এগুলোর মধ্যে ১টি মজার লাইব্রেরী হলো validator.js । এটি npm i validator এভাবে ইনস্টল করা যায়। আর ১টি মজার লাইব্রেরী হলো joi । এটিও npm i joi এভাবে ইনস্টল করা যায়। express js এ আরও সুন্দর উপায় হলো **express-validator** ভেলিডেটর ব্যবহার করা। express-validator হলো express js এর ১টি মিডলওয়্যার। | | |
| 3. Express Validator |  | | |
| 4. Create Playground for Validator |  | | |
| 5. Validation Check with Error Message | router.post('/validator',  [  check('username')  .not()  .isEmpty()  .withMessage('Username Can not be empty')  .isLength({  max: 15  })  .withMessage(`Username can not be greater than 15 Character`)  .trim(),  check('email')  .isEmail()  .withMessage('Please Provide A Valid Email')  .normalizeEmail(),  check('password').custom(value => {  if (value.length < 5) {  throw new Error('Password Must be greater than 5 characters')  }  return true  }),  check('confirmPassword').custom((value, {  req  }) => {  if (value !== req.body.password) {  throw new Error('Password Does Not Match')  }  return true  })  ],  (req, res, next) => {  let errors = validationResult(req)  const formatter = (error) => error.msg  console.log(errors.isEmpty())  console.log(errors.array())  console.log(errors.mapped())  console.log(errors.formatWith(formatter).mapped())  console.log(req.body.username, req.body.email)  res.render('playground/signup', {  title: 'Validator Playground'  })  })  module.exports = router | | |
| 6. Error Message with Formatter |  | | |
| 7. Create Custom validation |  | | |
| 8. What is Sanitizer |  | | |
| 9. Create Validator for Signup |  | | |
| 10. Get Error Message from Signup Form |  | | |
| 11. Show Error Message to User |  | | |
| 12. Render Submitted Data Back |  | | |
| 13. Write Validation Logic to Separate File |  | | |
| 14. Create Login Validator [Challenge] |  | | |
| Source Code |  | | |
| **15. How to Manage Session and Cookies** |  | | |
| 1. How does Authentication work | **আর দেখার প্রয়োজন নেই।\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*** | | |
| 2. What is Cookie |  | | |
| 3. How to Create and Use Cookie |  | | |
| 4. How to Configure Cookies |  | | |
| 5. What is Session |  | | |
| 6. How to Create And Configure Session |  | | |
| 7. Connect MLAB Database with Compass |  | | |
| 8. How to Create Session Store |  | | |
| 9. Create bindUserWithRequest Middleware |  | | |
| 10. Create setLocals Middleware |  | | |
| 11. Create Dashboard Page |  | | |
| 12. Create isAuthenticated Middleware |  | | |
| 13. Create Logout Functionalities |  | | |
| 14. Create isUnauthenticated Middleware |  | | |
| Source Code |  | | |
| **16. Alert Message, Environment Variables and Configurations** | | | |
| 1. Intro | | |  |
| 2. Setup Connect Flash | | |  |
| 3. Create Flash Class | | |  |
| 4. Using Flash in Our Project | | |  |
| 5. Test Flash Messages | | |  |
| 6. How to Check Current Environment | | |  |
| 7. Using Environment Variables | | |  |
| 8. How to Use DotEnv | | |  |
| 9. Setup Configuration File | | |  |
| 10. Using Config Module | | |  |
| 11. Custom Environment Variables | | |  |
| 12. Conclusion | | |  |
| Source Code | | |  |
| **17. Error Handling and Debugging** | | |  |
| 1. Intro | | |  |
| 2. Different Type of Errors | | |  |
| 3. Debugging vs Error Handling | | |  |
| 4. Debuggin and Error Handling Action | | |  |
| 5. Debug Module | | |  |
| 6. Chalk Module | | |  |
| 7. Node Debugger | | |  |
| 8. Chrome Debug Tolls | | |  |
| 9. Separate Route and Middleware File | | |  |
| 10. 404 Not Found | | |  |
| 11. 500 Internal Server Error | | |  |
| Source Code | | |  |
| **18. Handling File Uploads** | | |  |
| 1. Intro | | |  |
| 2. What is Multer | | |  |
| 3. Setup Playground | | |  |
| 4. Setup Storage for Multer | | |  |
| 5. Create Upload Middleware | | |  |
| 6. Handle Uploads | | |  |
| Source Code | | |  |
| **19. Create and Manage Profile for Our Project** | | |  |
| 1. Conditionally Render Dashboard or Profile | | |  |
| 2. Create Profile Template | | |  |
| 3. Create Upload Controller and Routes | | |  |
| 4. Setup Croppie JS for Cropping Image | | |  |
| 5. Upload Profile Pics | | |  |
| 6. Remove Profile Pics | | |  |
| 7. Validate Profile | | |  |
| 8. Handle Error to Create Profile Page | | |  |
| 9. Save Profile Data | | |  |
| 10. Edit Profile Template | | |  |
| 11. Edit Profile Controller | | |  |
| 12. Bug Fixing | | |  |
| Source Code | | |  |
| **20. Create And Mange Posts for Our Project** | | |  |
| 1. Intro | | |  |
| 2. Create Post Template | | |  |
| 3. Tiny MCE Front End Setup | | |  |
| 4. Tiny MCE Bachend | | |  |
| 5. Validate Create Post Form | | |  |
| 6. Error Handling | | |  |
| 7. Create Post Controller | | |  |
| 8. Post Create Testing | | |  |
| 9. Edit Post Template | | |  |
| 10. Update Post | | |  |
| 11. Test Our Edit Functionality | | |  |
| 12. Delete Post | | |  |
| 13. Get All My Posts | | |  |
| 14. Creating Side Bar | | |  |
| Source Code | | |  |
| **21. Create Rest API for Reactions** | | |  |
| 1. Intro | | |  |
| 2. Create Comment API | | |  |
| 3. Create Reply Controller API | | |  |
| 4. Create Like API | | |  |
| 5. Create Dislike Controller API | | |  |
| 6. Create Bookmarks Controller API | | |  |
| Source Code | | |  |
| **22. Explorer, Single Post Detailed View and Searching** | | | |
| 1. Setup Explorer Files | | |  |
| 2. Explorer Template | | |  |
| 3. Create Filter Functionalities | | |  |
| 4. Create Pagination Functionalities | | |  |
| 5. Add Bookmarks | | |  |
| 7. Single Post Page Template | | |  |
| 8. Single Page Controller | | |  |
| 9. Like Functionalities Frontend | | |  |
| 10. Handle Comments From Frontend | | |  |
| 11. Reply Comment from Frontend | | |  |
| 12. Create Search Backend | | |  |
| 14. Debugg And Conclusion | | |  |
| 13. Create Search Result Page | | |  |
| 16. Author Controller | | |  |
| 15. author page template | | |  |
| Source Code | | |  |
| **23. Dashboard Creation and Finishing Touch** | | |  |
| 1. Intro | | |  |
| 2. Bookmarks Dashboard Page | | |  |
| 3. Comment Dashboard Page | | |  |
| 4. Change Password Page | | |  |
| 5. Dashboard Final Page | | |  |
| 6. Navigation and Finishing Touch | | |  |
| Source Code | | |  |
| **24. Deployment and Conclusion** | | |  |
| Deployment and Conclusion [Uploading] | | |  |