Express Creating Environment

First we need to install NVM into an environment

**Step #1: Install NVM (Node Version Manager)**

Use the following curl command to kick-off the install script:

curl https://raw.githubusercontent.com/creationix/nvm/v0.25.0/install.sh | bash

At the time of publication, **NVM v0.25.0** was the most recent version available. You should check the [GitHub project page for the latest release of NVM](https://github.com/creationix/nvm/releases" \t "_blank), and adjust the above command to include the newest version. For example, if the most recent version was 0.30.2, then your command would be similar to:

Shell :-

Using username "opc".

Authenticating with public key "rsa-key-20180718"

[opc@mhealthvm1 ~]$ sudo -i

[root@mhealthvm1 ~]# nvm --version

0.25.0

**Step #2: Install Latest Nodejs (Node Version Manager)**

[root@mhealthvm1 ~]# nvm install 12.16.1

######################################################################## 100.0%

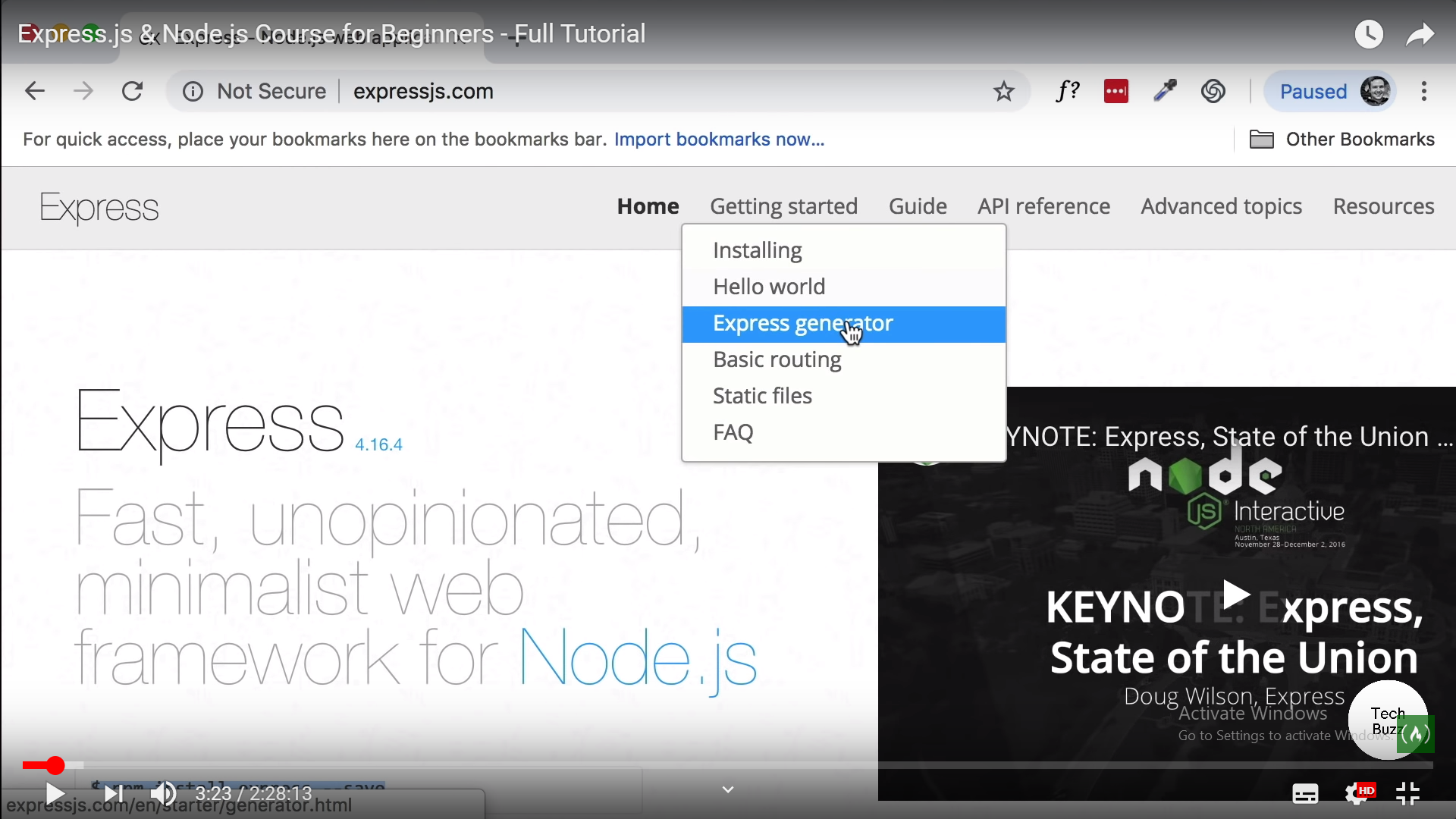
Checksums empty

Now using node v12.16.1 (npm v6.13.4)

[root@mhealthvm1 ~]#

**Step #3: Install Express Framework for Nodejs**

In this we will go to expressJS.com and then scroll down to the Express generator



[root@mhealthvm1 ~]# cd /home/

[root@mhealthvm1 home]# mkdir project

[root@mhealthvm1 home]# cd project

[root@mhealthvm1 project]# mkdir express

[root@mhealthvm1 project]# cd express/

[root@mhealthvm1 express]# npm install express-generator -g

npm WARN deprecated mkdirp@0.5.1: Legacy versions of mkdirp are no longer suppor ted. Please update to mkdirp 1.x. (Note that the API surface has changed to use Promises in 1.x.)

/root/.nvm/versions/node/v12.16.1/bin/express -> /root/.nvm/versions/node/v12.16 .1/lib/node\_modules/express-generator/bin/express-cli.js

+ express-generator@4.16.1

added 10 packages from 13 contributors in 1.173s

[root@mhealthvm1 express]# ls

[root@mhealthvm1 express]# express –h

The above commands installs the express framework .

**Step #4: Install Express Directory and basic files**

[root@mhealthvm1 express]# express --view=pug myapp

create : myapp/

create : myapp/public/

create : myapp/public/javascripts/

create : myapp/public/images/

create : myapp/public/stylesheets/

create : myapp/public/stylesheets/style.css

create : myapp/routes/

create : myapp/routes/index.js

create : myapp/routes/users.js

create : myapp/views/

create : myapp/views/error.pug

create : myapp/views/index.pug

create : myapp/views/layout.pug

create : myapp/app.js

create : myapp/package.json

create : myapp/bin/

create : myapp/bin/www

change directory:

$ cd myapp

install dependencies:

$ npm install

run the app:

$ DEBUG=myapp:\* npm start

**Step #5: Install Web based Editor**

# Start Node

[root@mhealthvm1 .nvm]# nvm ls

v12.16.1

node -> stable (-> v12.16.1) (default)

stable -> 12.16 (-> v12.16.1) (default)

iojs -> iojs- (-> N/A) (default)

[root@mhealthvm1 .nvm]# nvm use 12.16

Now using node v12.16.1 (npm v6.13.4)

[root@mhealthvm1 .nvm]# npm --version

6.13.4

[root@mhealthvm1 .nvm]#

**Moving to cloud9**

Check if nodejs is installed ?

#adduser userc9

#usermod –aG userc9 userc9

#vi /etc/sudoers

Change the line to

%userc9 ALL=(ALL) ALL

Now login to the userc9 account

#sudo userc9

$sudo yum install git

[userc9@mhealthvm1 root]$ pwd

/root

[userc9@mhealthvm1 root]$ cd /home/userc9/

[userc9@mhealthvm1 ~]$ git clone http://github.com/c9/core.git c9

Cloning into 'c9'...

remote: Enumerating objects: 2, done.

remote: Counting objects: 100% (2/2), done.

remote: Compressing objects: 100% (2/2), done.

remote: Total 52830 (delta 0), reused 0 (delta 0), pack-reused 52828

Receiving objects: 100% (52830/52830), 35.34 MiB | 7.74 MiB/s, done.

Resolving deltas: 100% (32447/32447), done.

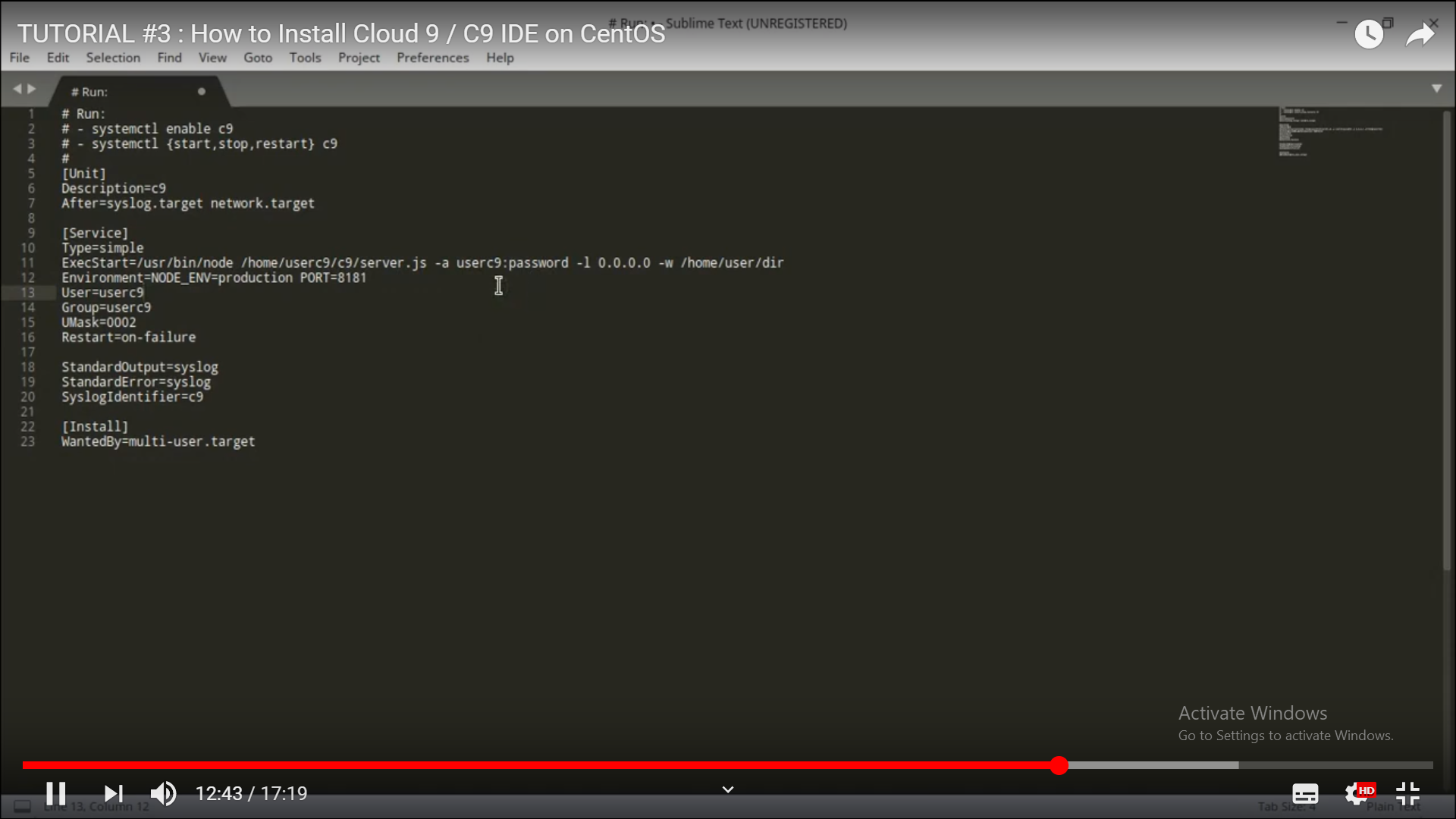
[userc9@mhealthvm1 ~]$ cd c9

[userc9@mhealthvm1 ~]$ scripts/install-sdk.sh

--------------------------------------------------------------------

Success!

run 'node server.js -p 8080 –l 0.0.0.0 –a userc9:password :' to launch Cloud9

[userc9@mhealthvm1 c9]$ sudo node server.js -p 8080 -l 0.0.0.0 -a userc9:userc9 -w /home/project/express/myapp

**Step #6: Running the basic App**

[opc@mhealthvm1 ~]$ sudo -i

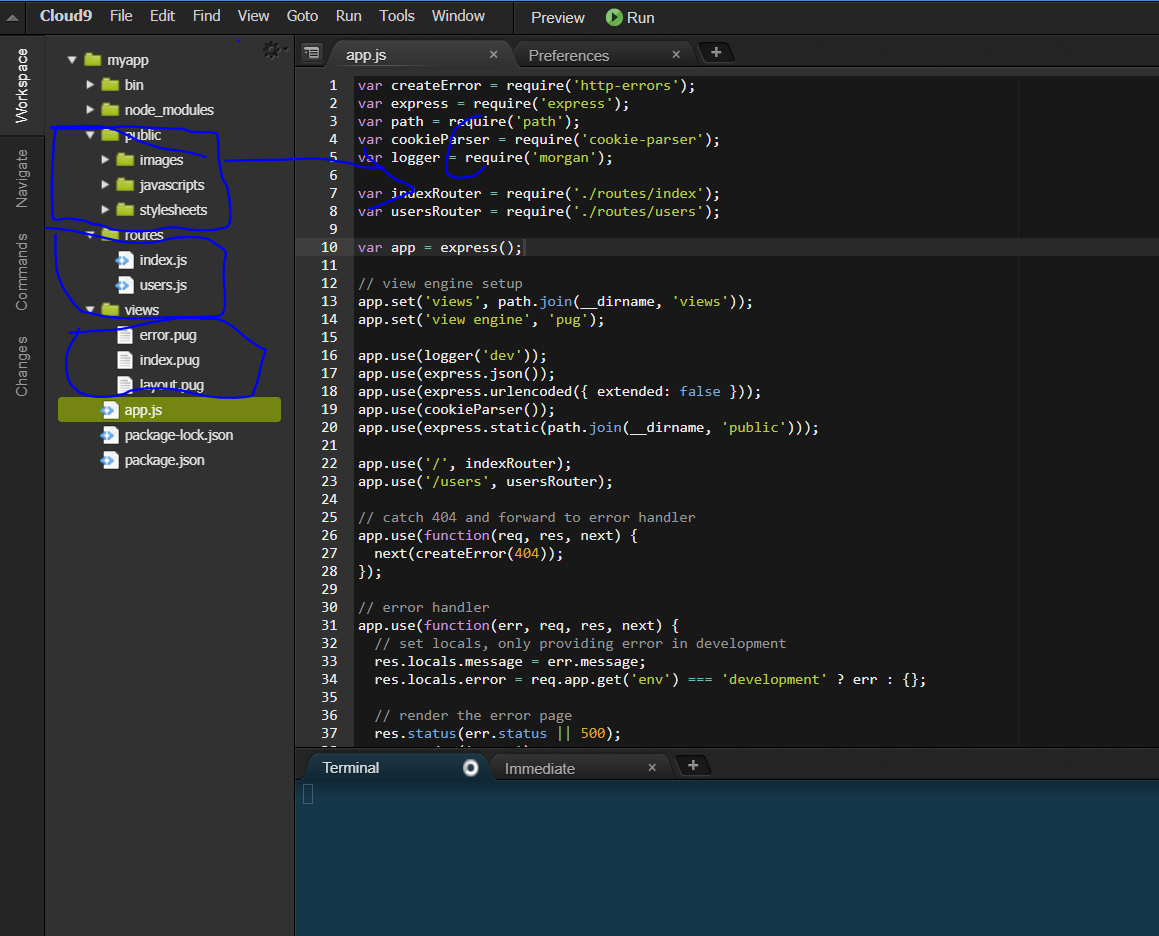
[root@mhealthvm1 ~]# cd /home/project/express/

[root@mhealthvm1 express]# cd myapp

[root@mhealthvm1 myapp]# DEBUG=myapp/\* npm start

**Step #7: Looking at Basic App Files**

The Basic Module in the File components is the following



So It has got

1. Public Folders 🡪 Just contains the public files as Images ,JS and CSS files
2. Routes 🡪 They are basically contains the routing structure for the files
3. Views 🡪 They contain the template files
4. App.js 🡪 This is the base file which invokes the basics of the App

Last but not the least , In the end we need to export the files .

Import the library by using the

Var express = require(‘express’);

Then we initialize the basic constructor

Var app=express();

Then the properties for the app are changed

View Engine setup

// view engine setup

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'pug');

//Basic libraries are being set

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

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

//Router files are being set

app.use('/', indexRouter);

app.use('/users', usersRouter);

// catch 404 and forward to error handler

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

next(createError(404));

});

// error handler

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

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get('env') === 'development' ? err : {};

// render the error page

res.status(err.status || 500);

res.render('error');

});

module.exports = app;

**Step #8: Looking at Routing Files**

Routing file works for the function to be invoked when some route is called

Routes/index.js

var express = require('express');

var router = express.Router();

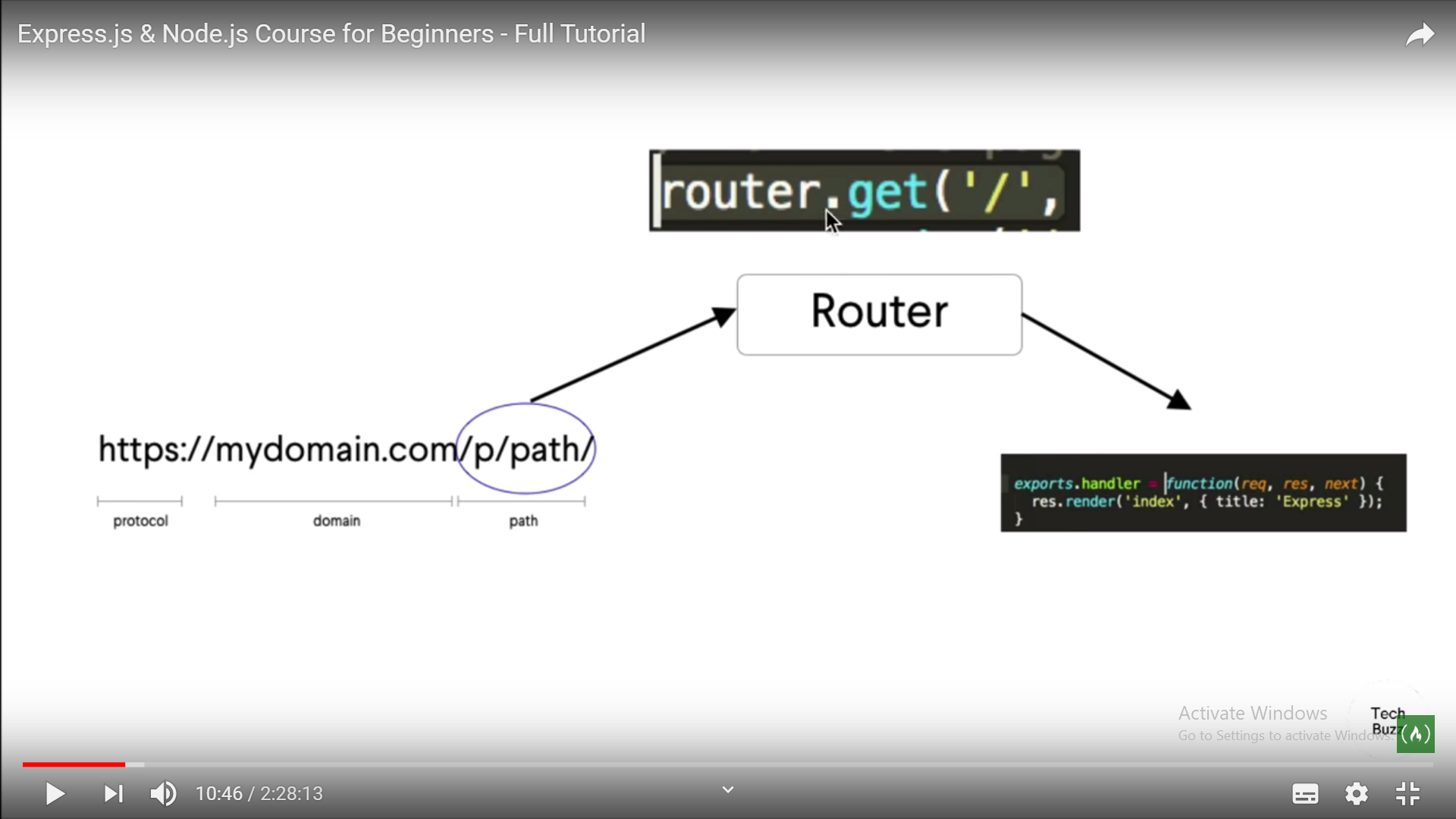
/\* GET home page. \*/

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

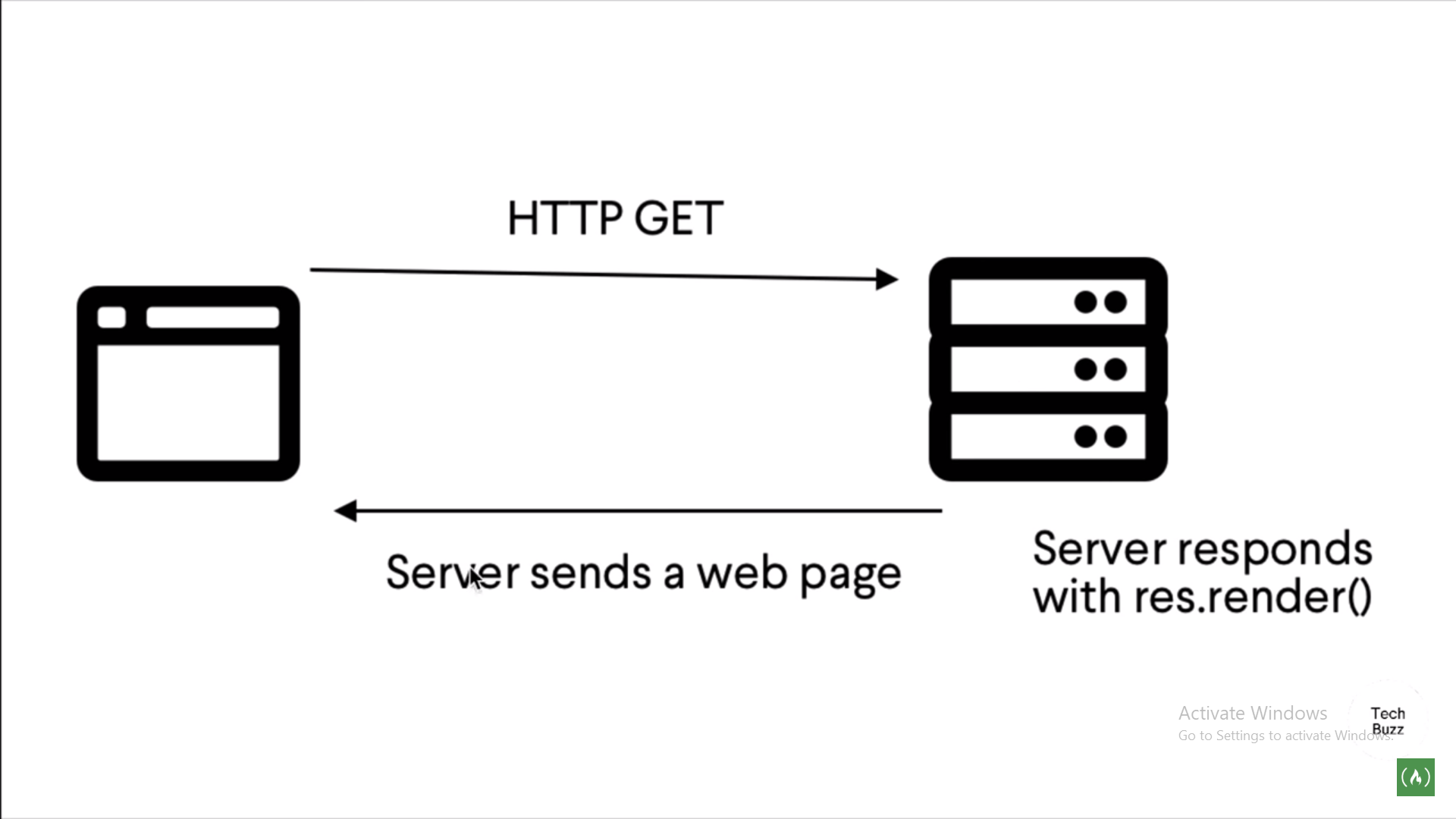
res.render('index', { title: 'Express' });

});

module.exports = router;



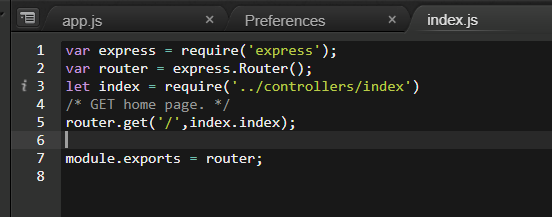
**Step #9: Understanding how does Client server**



When the bowser hits the url <http://10.189.10.116:3000/>

It hits the file APP.js

It then goes to the routes directory index.js

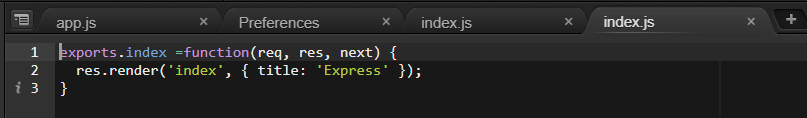


It contains the line

Router.get(‘/’,index.index);

It means that it will call the ../controllers/index file and there

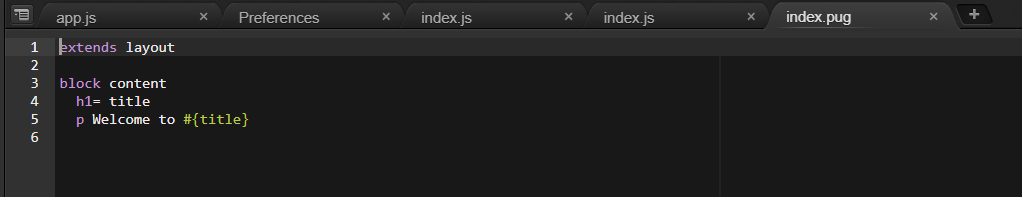
Exports.index method is called .



Here the res.render contains two parameters

1. Index->name of the **views** to be find in the views folder
2. {title :’Express’} Parameter to be passed as variable

Now take the Views index.pug file



Here the #{title} is passed via the res.render method .

**Step #10: Adding changed files to github and commit**

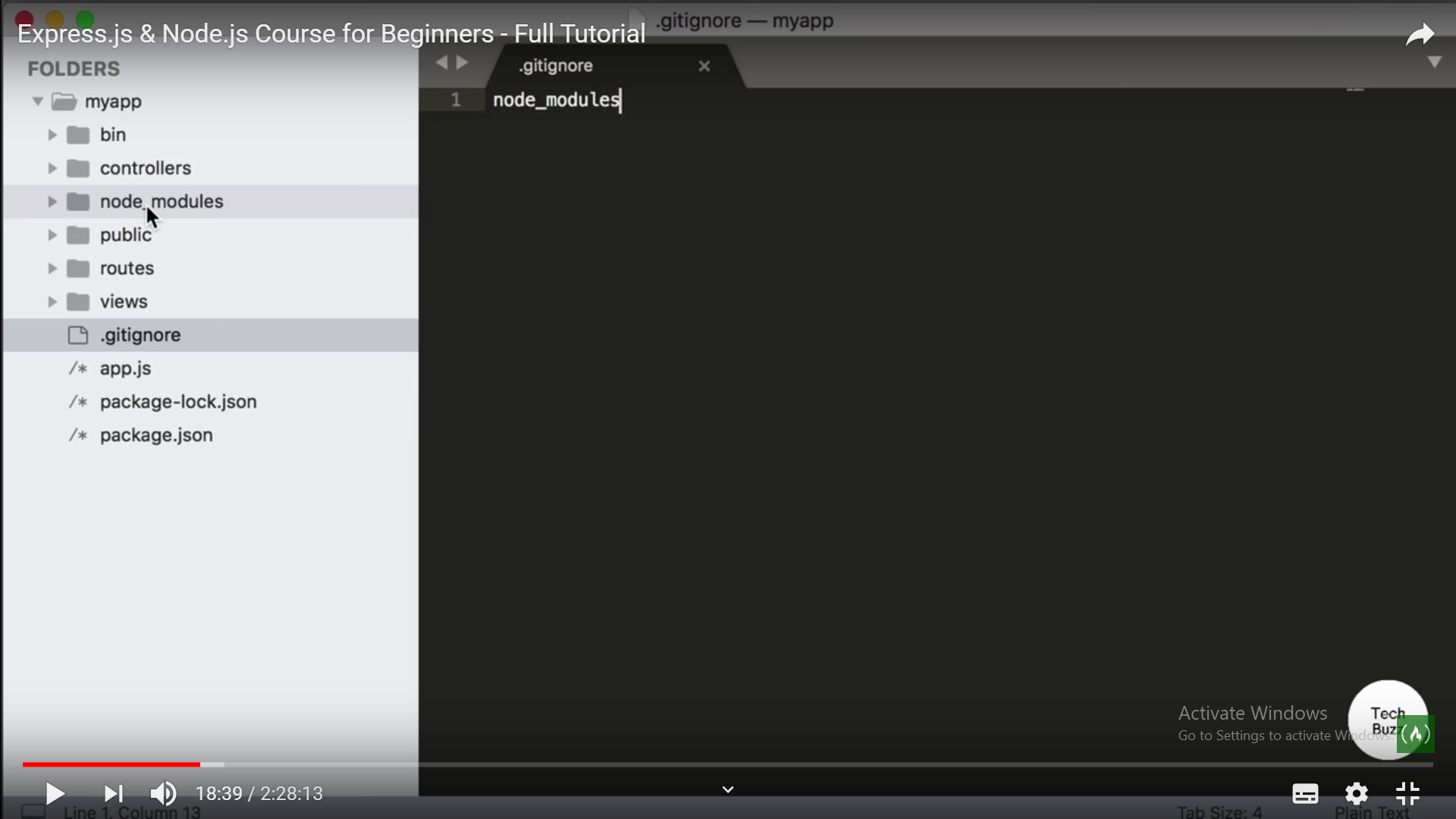
[root@mhealthvm1 ~]# cd /home/project/express/myapp

[root@mhealthvm1 myapp]# git init-db

Initialized empty Git repository in /home/project/express/myapp/.git/

[root@mhealthvm1 myapp]#

Add Gitignore file ..



**Adding files to Git**

[root@mhealthvm1 myapp]# git add .

**Committing files to Github**

[root@mhealthvm1 myapp]# git commit -a

# Please enter the commit message for your changes. Lines starting

# with '#' will be ignored, and an empty message aborts the commit.

#

# Committer: root <root@mhealthvm1.compute-500016327.in1.internal>

#

# On branch master

#

# Initial commit

#

# Changes to be committed:

# (use "git rm --cached <file>..." to unstage)

#

# new file: .c9/.nakignore

# new file: .c9/metadata/tab0

# new file: .c9/metadata/tab3

# new file: .c9/metadata/workspace/.gitignore

# new file: .c9/metadata/workspace/app.js

# new file: .c9/metadata/workspace/controllers/index.js

# new file: .c9/metadata/workspace/routes/index.js

# new file: .c9/metadata/workspace/views/index.pug

# new file: .c9/project.settings

# new file: .c9/state.settings

".git/COMMIT\_EDITMSG" 36L, 1100C

[root@mhealthvm1 myapp]# git log

commit d9f2e29bfc0801e48dec600c612c67eacf238b2c

Author: root <root@mhealthvm1.compute-500016327.in1.internal>

Date: Mon Mar 30 08:06:45 2020 -0400

Initial commit

**Step #10: Challenging of restarting the Node browser so we use Nodemon**

[root@mhealthvm1 myapp]# npm install nodemon --save-dev

> nodemon@2.0.2 postinstall /home/project/express/myapp/node\_modules/nodemon

> node bin/postinstall || exit 0

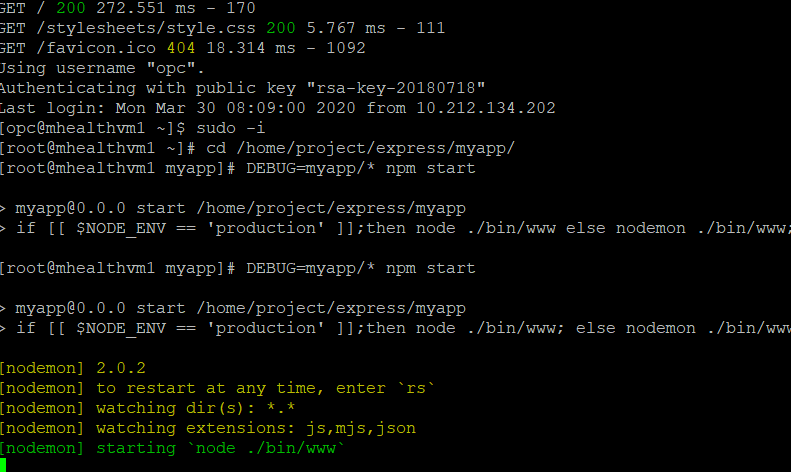
Love nodemon? You can now support the project via the open collective:

> https://opencollective.com/nodemon/donate

Make changes in package.json file

"start": "if [[ $NODE\_ENV == 'production' ]];then node ./bin/www; else nodemon ./bin/www;fi"

Then restart the server



**Step #10: Steps to connect to Database through Sequelize**

195 sequelize init

196 sequelize db:migrate

198 sequelize db:migrate

201 sequelize create --name Leads --attributes email:string

202 sequelize model:create --name Leads --attributes email:string

204 sequelize db:migrate

**Step #10: Steps to create new Table User through Sequelize**

[root@mhealthvm1 myapp]# sequelize model:create --name Leads\_Users --attributes "firstname:string, lastname:string,status:boolean,email:string,password:string"

Sequelize CLI [Node: 12.16.1, CLI: 5.5.1, ORM: 5.21.5]

New model was created at /home/project/express/myapp/models/leads\_users.js .

New migration was created at /home/project/express/myapp/migrations/20200408083517-Leads\_Users.js .

[root@mhealthvm1 myapp]#

**Step #11: Steps to now Install additional Libraries for the following**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | **Passport** | **Used for User authentication** |
| 2 | **Passport-local** | **For local user authentication** |
| 3 | **Bcrypt** | **For encryption of passwords** |
| 4 | **Validator** | **For validating the input fields for the Login /Signup** |
| 5 | **Express-session** | **Package for managing the session in Express** |
| 6 | **Connect-flash** | **For displaying the Error messages .** |
|  |  |  |

**Command required is as follows:-**

**[root@mhealthvm1 myapp]# sudo npm install passport passport-local bcrypt express-session validator connect-flash --save**

**This worked !!!!!(given error without SUDO**

Step #11: Steps to now Install additional validator and Lodash Libraries for the following

**[opc@mhealthvm1 ~]$ sudo -i**

**[root@mhealthvm1 ~]# cd /home/project/express/myapp/**

**[root@mhealthvm1 myapp]# npm install validator lodash --save**

**npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.2 (node\_modules/fsevents):**

**npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})**

**+ validator@13.0.0**

**+ lodash@4.17.15**

**updated 2 packages and audited 563 packages in 4.358s**

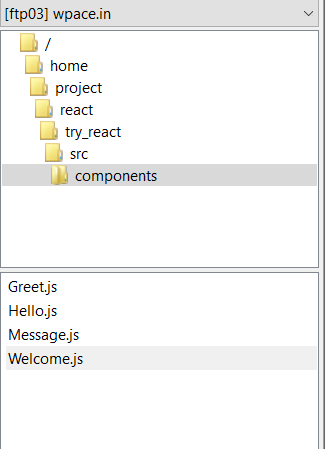
**4 packages are looking for funding**

**run `npm fund` for details**

**found 1 low severity vulnerability**

**run `npm audit fix` to fix them, or `npm audit` for details**

**Create a Folder components in the SRC folder**

1. 
2. **Now create a file name named *Message.js***

Import React from ‘react’

function Message(){

return <h1> HEllo Anuj</h1>

}

export default Message

1. **Now this File is being called by the *App.js***

Import React from ‘react’

Import Abc from ‘./components/Message’

function App() {

return (

<div className="App">

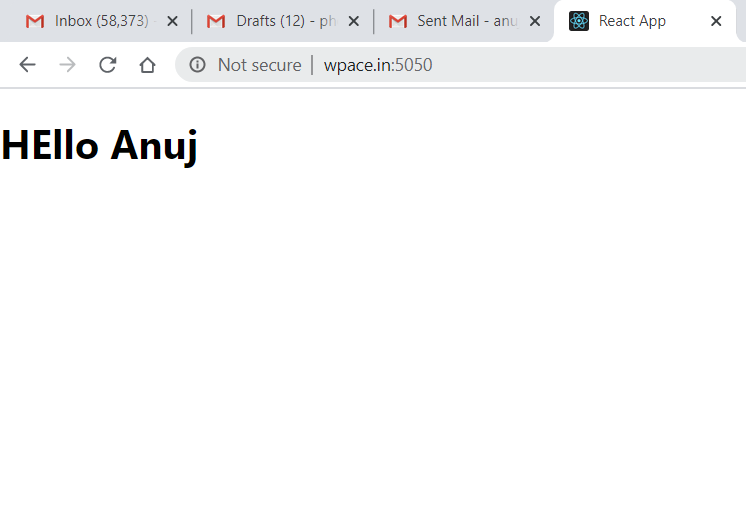
<Abc />

</div>

} ;}

Export default App;

1. The Output on the Browser



Express Creating Environment

Rest steps remain as such

1. **Now create a file name named *Message.js***

Import React from ‘react’

function Message(props){

return <h1> HEllo ~~Anuj~~ {props.name} your Gender is {props.gender}</h1>

}

export default Message

1. **Now this File is being called by the *App.js***

Import React from ‘react’

Import Abc from ‘./components/Message’

function App() {

return (

<div className="App">

<Abc name="Anuj" gender="male" />

<Abc name="Ayaan" gender="male" />

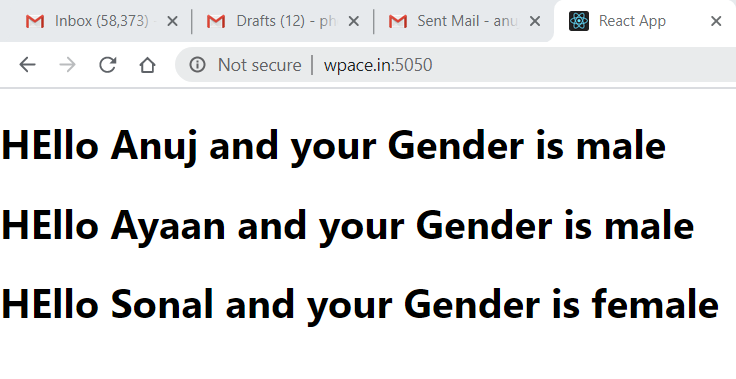
<Abc name="Sonal" gender="female" />

</div>

} ;}

Export default App;

1. The Output is



Point to note that Since we are using the default in export syntax that’s why we can use Abc as name of our component .

Else other option is

Import React from ‘react’

export function Message(props){

return <h1> HEllo {props.name} your Gender is {props.gender}</h1>

}

//export default Message

Here the Error comes

Failed to compile

./src/App.js  
Attempted import error: './components/Greet' does not contain a default export (imported as 'Abc').

This error occurred during the build time and cannot be dismissed.

Because it is NAMED EXPORT since it is exporting only the function with name “Message”

Replace ‘Abc’ with ‘Message’

Import React from ‘react’

Import {Message} from ‘./components/Message’

function App() {

return (

<div className="App">

<Message name="Anuj" gender="male" />

<Message name="Ayaan" gender="male" />

<Message name="Sonal" gender="female" />

</div>

} ;}

Export default App;

Now with JSX Element

return(

React.CreateElement(

“div”,

{className:”App”},

React.CreateElement(

“Message”,

{name:”Anuj”,gender:”male”});

React.CreateElement(

“Message”,

{name:”Anuj”,gender:”male”});

React.CreateElement(

“Message”,

{name:”Anuj”,gender:”male”});

Express Creating Environment

Like the function they can also accept properties and generate the JSX (HTML)

Class Components

They can also maintain **State locally.**

Generates JSX (HTML)

Props as Input

Lets create a basic component

1. **It should first import the Component class from the ‘react’**

Import React,{Component} from ‘react’

1. **Next it should have class with name of component**

class CMessage extends Component {

}

1. **Then it should have a render function which can return NULL or HTML**

class CMessage extends Component {

render(){

return <h1> Hello Anuj </h1>

}

}

1. **Now create a file name named C*Message.js***

Import React,{Component} from ‘react’

class CMessage extends Component {

render(){

return <h1> Hello Anuj </h1>

}

}

export default Message

1. **Now this File is being called by the *App.js***

Import React from ‘react’

Import CMessage from ‘./components/CMessage’

function App() {

return (

<div className="App">

< CMessage name="Anuj" gender="male" />

< CMessage name="Ayaan" gender="male" />

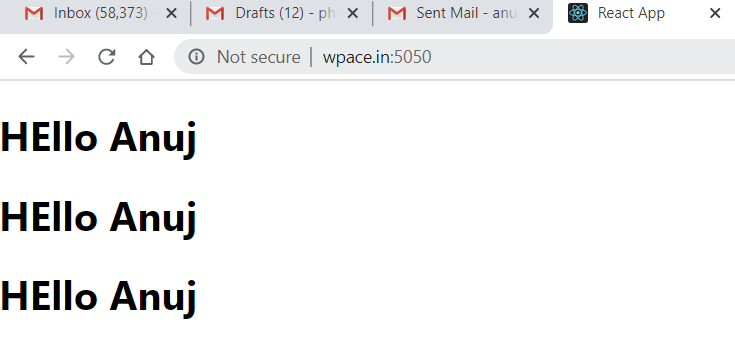
< CMessage name="Sonal" gender="female" />

</div>

} ;}

Export default App;

1. The Output is



1. Passing Variables in the CLASS COMPONENT ***CMessage.js***

Import React,{Component} from ‘react’

class CMessage extends Component {

render(){

const {name,gender} = this.props  
<div>

<h1>HEllo {this.props.name } , your gender is {this.props.gender}</h1>

<h1>Hello {name}, your gender is {gender}</h1>

<p>{props.children}</p>

</div>

}

}

export default Message

Few things to note :-

1. The props property is IMMUTABLE ie its values cant be changed
2. It returns only one HTML element . So they have to be wrapped in single <div> element.

**De Structuring props**

render(){

const {name,gender} = this.props  
<div>

<h1>HEllo {this.props.name } , your gender is {this.props.gender}</h1>

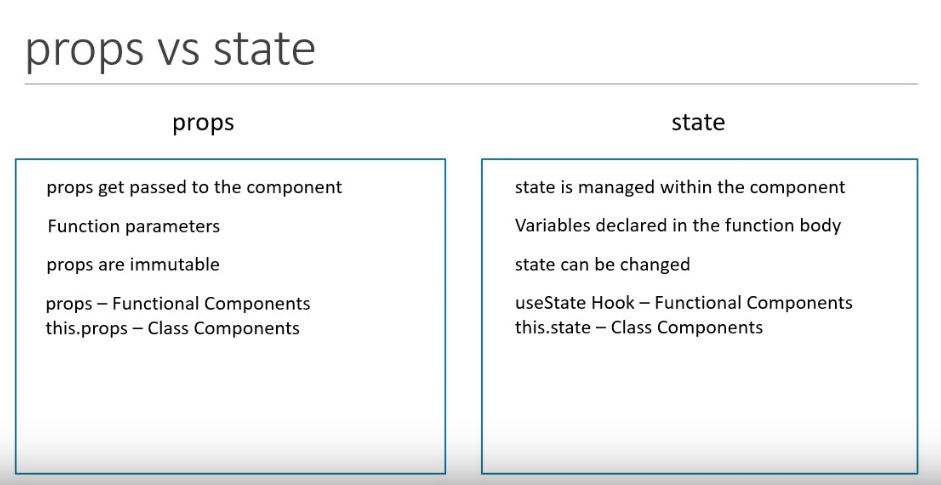
<h1>Hello {name}, your gender is {gender}</h1>

<p>{props.children}</p>

</div>

Express Creating Environment

Another way of passing the values in the JSX which renders the HTML by way of children component .



For using the STATE , it is like session variables of PHP which stores the values and changes on runtime .

Lets work around this using an example where the div displays the message

“Welcome Visitor”

And once the Subscribe button is clicked it says

“Thank you for subscribing “

1. We need to constructor to define the original value in the STATE

constructor() {

super()

this.state = {

message: "Welcome Visitor"

}

}

1. We need to call the function inside the JSX by calling the EVENT method

<button onClick={() => this.changeMessage()}>Subscribe</button>

1. Lets Define the event function by changing the value of the state in the Event Function

changeMessage(){

this.setState({

message:"Thank you for subscribing"

})

}

1. The complete Program CMessage.js

import React , {Component} from 'react'

class CMessage extends Component

{

constructor(){

super()

this.state ={

message: "Welcome Visitor"

}

}

changeMessage(){

this.setState({

message:"Thank you for subscribing"

})

}

render(){

return (

<div>

<h1>

{this.state.message}

</h1>

<button onClick={() => this.changeMessage()}>Subscribe</button>

</div>

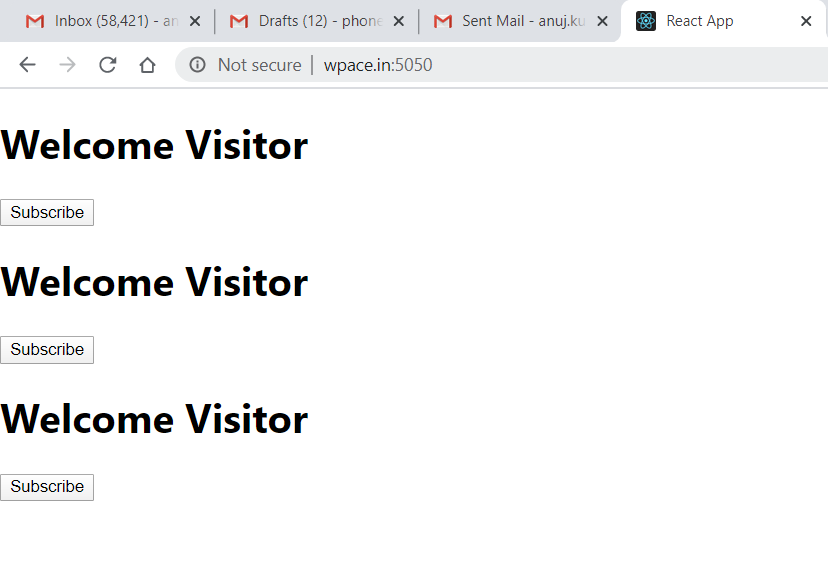
)

}

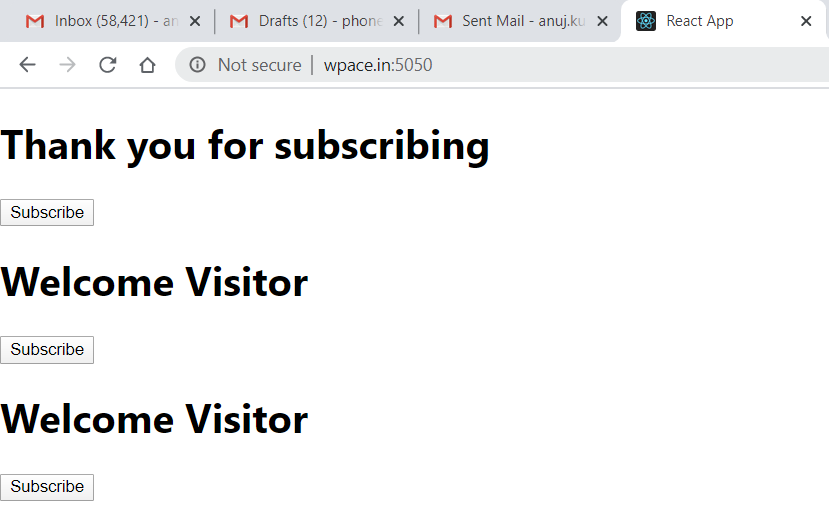
}

export default CMessage

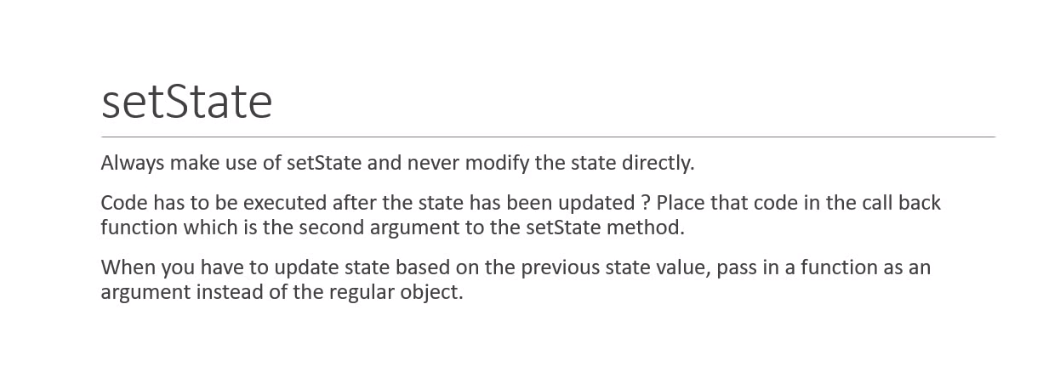
Before :-



After :-



Express Creating Environment



1. Now we should do it by incrementing the counter on click of the button

***Counter.js***

import React,{Component} from 'react'

class Counter extends Component

{

constructor(){

super()

this.state={

count:0

}

}

increment(){

this.setState({

count:this.state.count + 1

})

console.log(this.state.count)

}

render(){

return(

<div>

<h1>{this.state.count}</h1>

<button onClick={() => this.increment()}>increment</button>

</div>)

}

}

export default Counter

**Synchronous vs Asynchronous CALLS**

Note two sentences

<button onClick={() => this.increment()}>increment</button>

Whenever the Event is clicked , it makes an asychrounous call to the setState function

Whereas the console.log function is called synchronously .

Increment (){

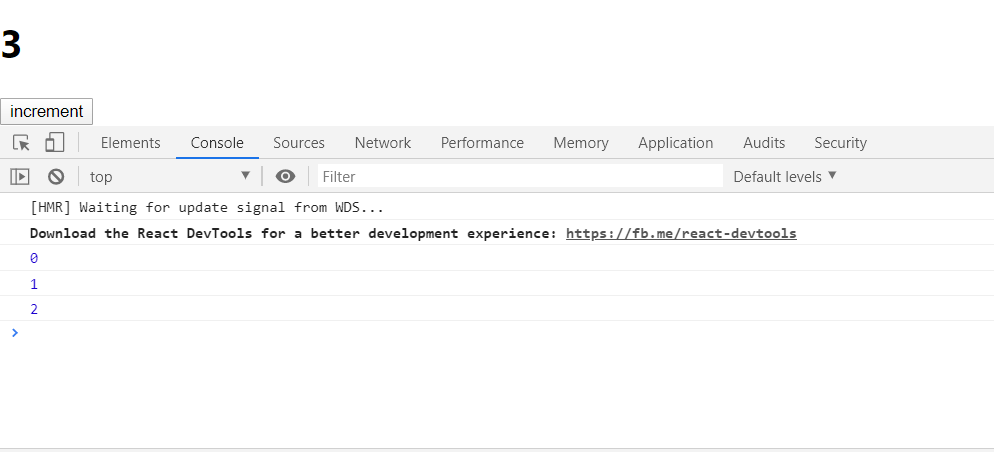
:

:

console.log(this.state.count)

}

In output



To Rectify this we call this as an CALLBACK function to setState Method

increment(){

this.setState({

count:this.state.count + 1

},() => {console.log(this.state.count)})

}

**CALLING PREVSTATE**

To implement this lets do the chaining of the event by calling the increment function 5 times .

import React,{Component} from 'react'

class Counter extends Component

{

constructor(){

super()

this.state={

count:0

}

}

increment(){

this.setState({

count:this.state.count + 1

},() => { console.log(this.state.count)

})

}

incrementFive(){

this.increment()

this.increment()

this.increment()

this.increment()

this.increment()

}

render(){

return(

<div>

<h1>{this.state.count}</h1>

<button onClick={() => this.incrementFive()}>increment</button>

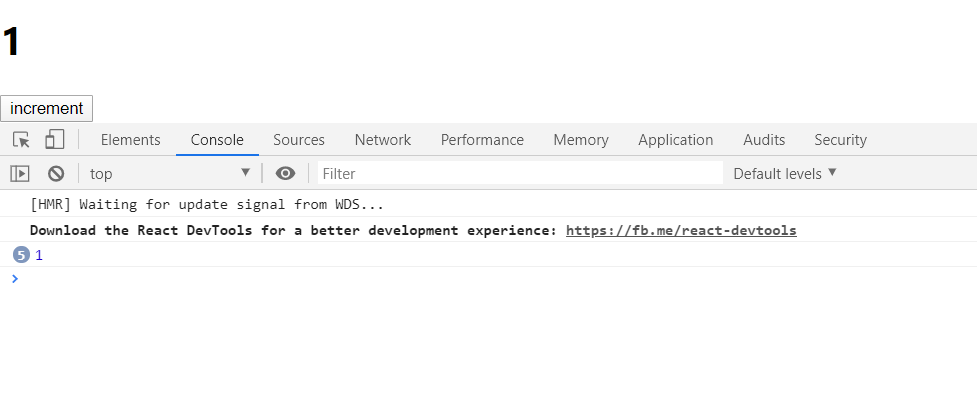
</div>)

}

}

export default Counter

Output



This means that on clicking only 1 time , count is increment by 1 as for all the count is grouped into 1 only.

Thereby we need to CHAIN it with previous state .

*this.setState({*

*count:this.state.count + 1*

*},() => {console.log(this.state.count)})*

Is changed to

increment(){

this.setState(prevState => ({

count:prevState.count + 1

}),() => {console.log(this.state.count)})

}

Express Creating Environment

Why We need to Bind ?

In Event , ***this*** is a Javascript keyword and thereby it remains undefined

Lets try this with an example

**EventBind.js**

import React,{Component} from 'react'

class EventBind extends Component

{

constructor(){

super()

this.state={

message:"Hello"

}

this.clickHandler = this.clickHandler.bind(this)

}

clickHandler(){

this.setState=({

message:"GoodBye"

})

console.log(this)

}

render(){

return(

<div>

<h1>{this.state.message}</h1>

<button onClick={this.clickHandler}>Click Me</button>

{/\*Option 1\*/}

<button onClick={this.clickHandler.bind(this)}>Click Me</button>

{/\*Option 2\*/}

<button onClick={() => this.clickHandler ()}>Click Me</button>

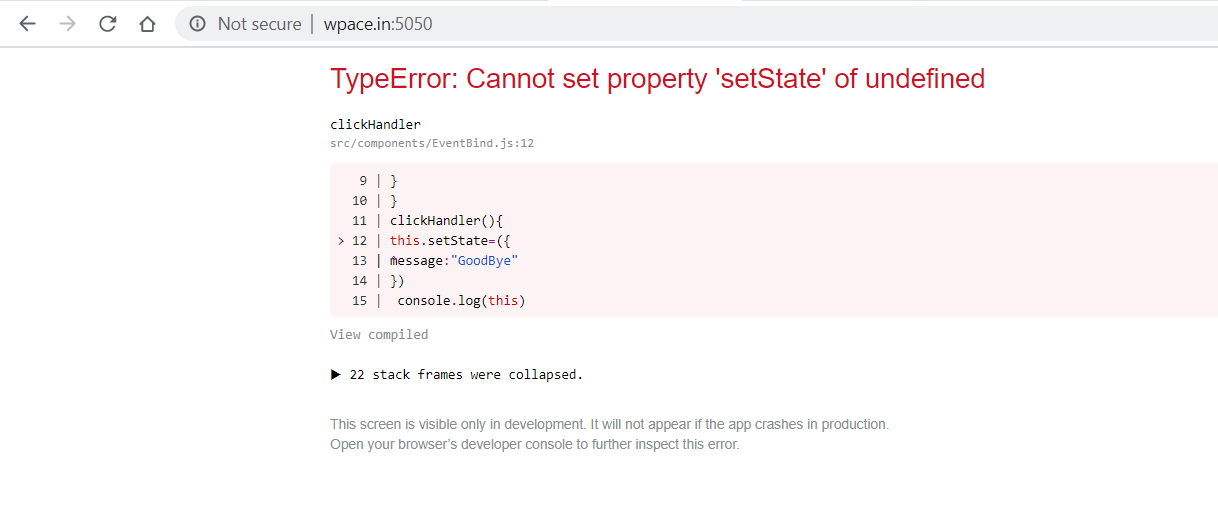
</div>)

}

}

export default EventBind

Output



It has two Options

1. Inside the render method with
   1. **<button onClick={this.clickHandler.bind(this)}>Click Me</button>**
2. Inside the render Method
   1. **<button onClick={() => this.clickHandler ()}>Click Me</button>**

Both of the Above methods are in render function and have issues with the performance of the application

**So the best option has to be Bind Inside the Constructor**

Putting the below in the constructor

this.clickHandler = this.clickHandler.bind(this)

Render method will have simply

<button onClick={this.clickHandler}>Click Me</button>

Final the Most efficient way is to change the ***clickHandler method to the Arrow Function***

import React,{Component} from 'react'

class EventBind extends Component

{

constructor(){

super()

this.state={

message:"Hello"

}

// this.clickHandler = this.clickHandler.bind(this)

}

//clickHandler(){

//this.setState({

// message:"GoodBye"

// })

// console.log(this)

// }

clickHandler = () =>{

this.setState({

message:"GoodBye"

})

console.log(this)

}

render(){

return(

<div>

<h1>{this.state.message}</h1>

<button onClick={this.clickHandler}>Click Me</button>

{/\*<button onClick={this.clickHandler.bind(this)}>Click Me</button> \*/}

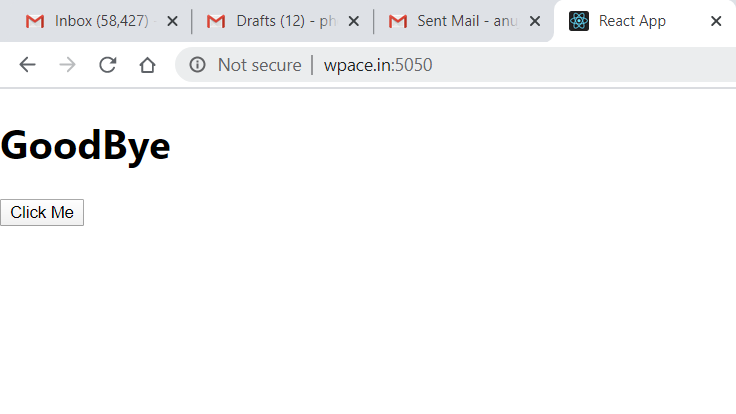
{/\*<button onClick={() => this.clickHandler()}>Click Me</button> \*/}

</div>)

}

}

export default EventBind



Express Creating Environment

Here the List is pretty simple by using the Array.map method

Lets do this by building a new functional component

**NameList.js**

import React from 'react'

import NameItem from './NameItem'

function NameList(){

const nameList = [{'Name':'Anuj','Age':'43','Gender':'Male'},{'Name':'Sonal','Age':'41','Gender':'Female'},{'Name':'Ayaan','Age':'11','Gender':'Male'}];

//const nameMap=nameList.map(name => <h1>My name is {name.Name} and my age is {name.Age}</h1>)

const nameMap=nameList.map(name => <NameItem item={name} />)

return(

<div>

{nameMap}

</div>

)

}

export default NameList

Now we have made the NameItem -> which describes the layout of each Item.

**NameItem.js**

import React from 'react'

function NameItem(props)

{

return(

<div>

Hi, your name is <h1>{props.item.Name}</h1>

Age :{props.item.Age}

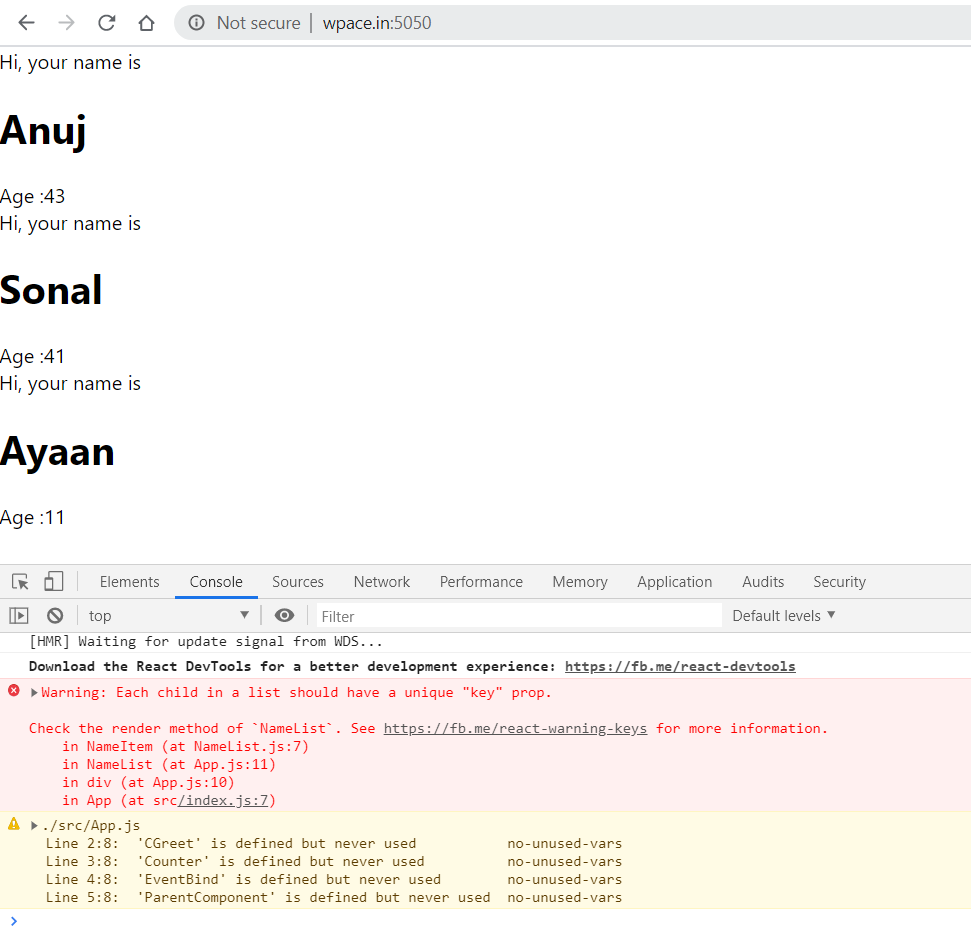
</div>

)

}

export default NameItem

Output :



But in the End there is an error which shows the Key property is not unique

So for this add ‘id’ to an array and change the method to following :-

import React from 'react'

import NameItem from './NameItem'

function NameList(){

const nameList = [{'id':'1','Name':'Anuj','Age':'43','Gender':'Male'},{'id':'2','Name':'Sonal','Age':'41','Gender':'Female'},{'id':'3','Name':'Ayaan','Age':'11','Gender':'Male'}];

//const nameMap=nameList.map(name => <h1>My name is {name.Name} and my age is {name.Age}</h1>)

const nameMap=nameList.map(name => <NameItem key={name.id} item={name} />)

return(

<div>

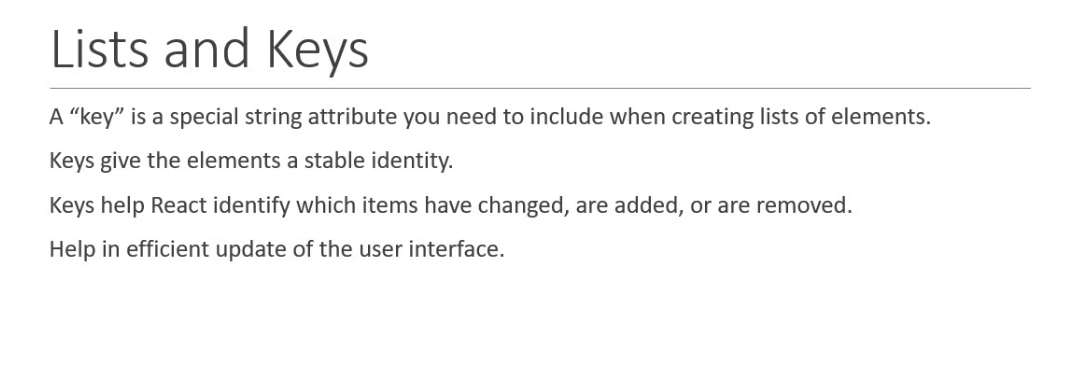
{nameMap}

</div>

)

}

Export default NameList



Express Creating Environment

Example of using the Form handling methods .

import React,{Component} from 'react'

class Form extends Component

{

constructor(){

super();

this.state={

username:'',

topic:'',

comments:''

}

}

handleUsernameChange = event => {

this.setState({

username:event.target.value

})

}

handleCommentsChange = event => {

this.setState({

comments:event.target.value

})

}

handleTopicChange = event => {

this.setState({

topic:event.target.value

})

}

handleSubmitEvent = event =>{

alert(`${this.state.username} ${this.state.comments} ${this.state.topic}`)

event.preventDefault()

}

render(){

return(

<form onSubmit={this.handleSubmitEvent}>

<div>Username

<input type="text" value={this.state.username} onChange={this.handleUsernameChange} />

</div>

<div>Comments

<textarea value={this.state.comments} onChange={this.handleCommentsChange} />

</div>

<div> Select the Topic

<select value={this.state.topic} onChange={this.handleTopicChange} >

<option value="react">React</option>

<option value="angular">Angular</option>

<option value="vue">Vue</option>

</select>

</div>

<div>

<input type="submit" value="Submit" />

</div>

</form>

)

}

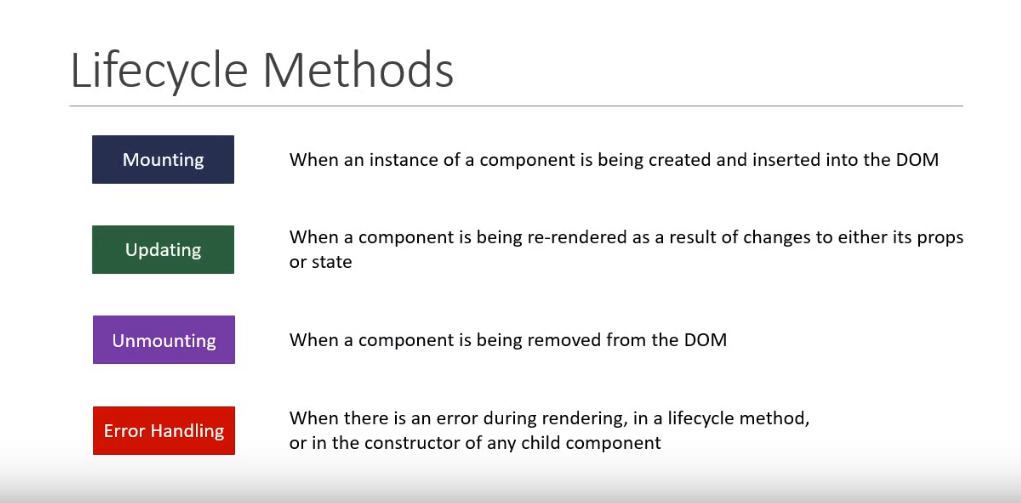
}

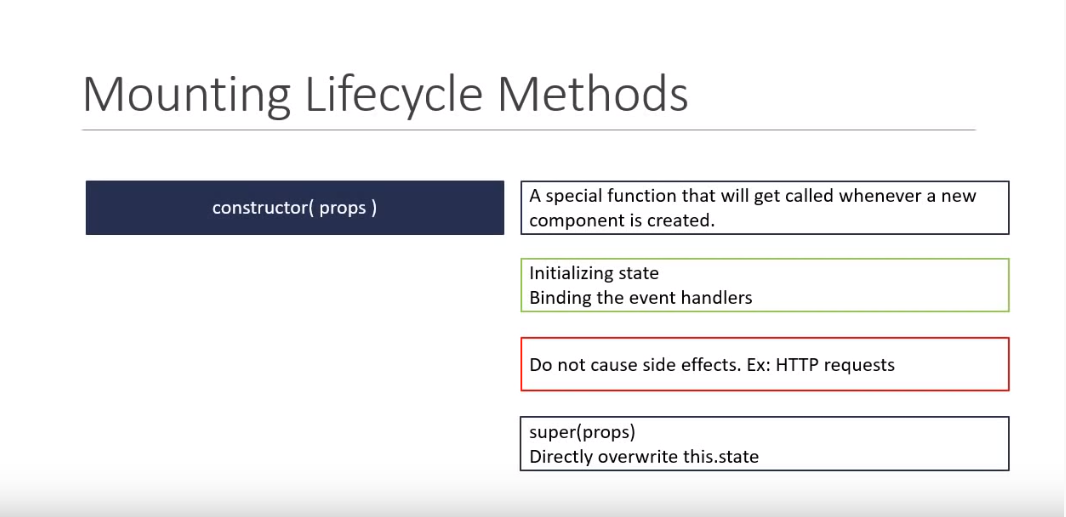
export default Form

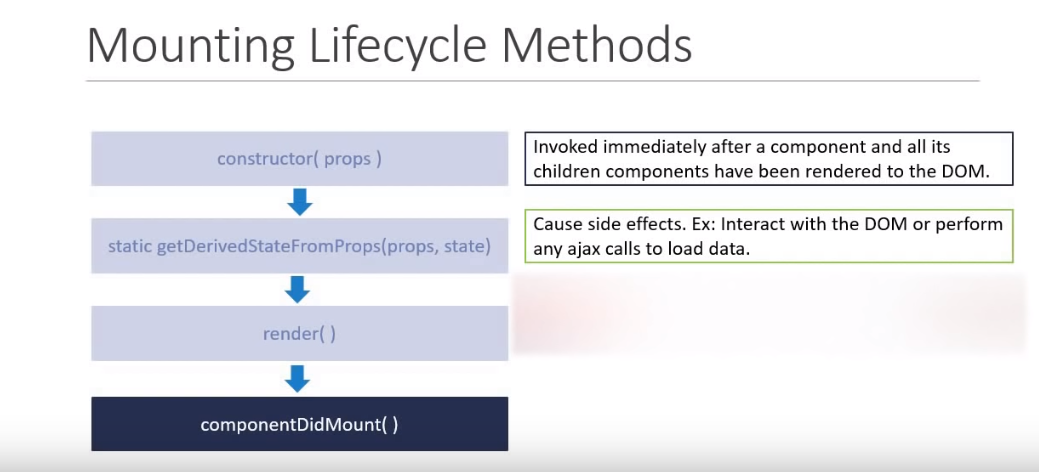
Express Creating Environment

Lifecycle method are the important part of the react which changes with the component being insetrted into the DOM(ie Mounting) Having the change of Data or the state via the State change methods and Unmounting means removal of the component from the DOM .

Then there is an error handling for it to Error Handling method.







Lets look at the code for different Methods

import React,{Component} from 'react'

class LifeCycleA extends Component

{

constructor(props){

super(props)

this.state={

name:'Anuj'

}

console.log("LifeCycleA Constructor called")

}

static getDerivedStateFromProps(props,state)

{

console.log("LifeCycleA getDerivedStateFromProps called")

return null

}

componentDidMount(){

console.log("LifeCycleA component Did Mount called")

}

render(){

console.log("LifeCycleA render called")

return(

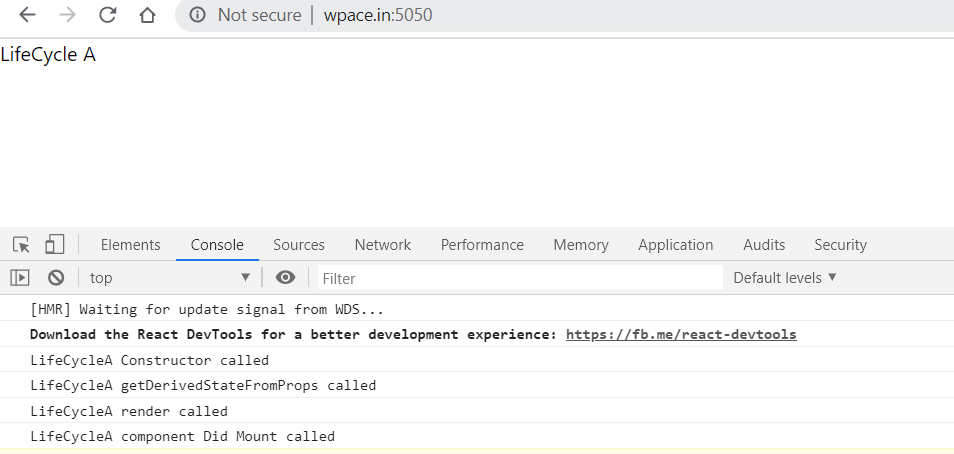
<div>LifeCycle A</div>

)

}

}

export default LifeCycleA



Now When we include another child component by duplicating the same file and including it in the LifeCycleA file

Render method is changed to

render(){

console.log("LifeCycleA render called")

return(

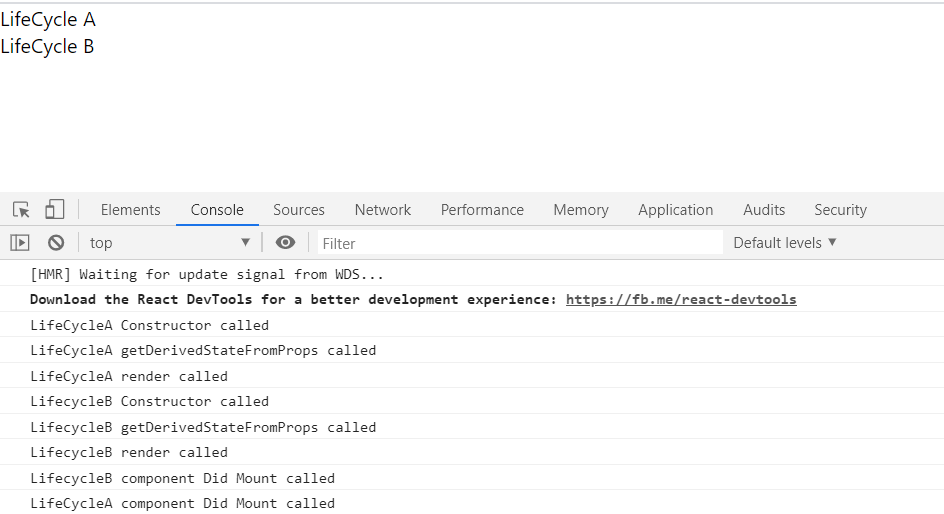
<div>LifeCycle A

<LifeCycleB />

</div>

)

}



Now Try Updating the Update Method

import React,{Component} from 'react'

import LifeCycleB from './LifeCycleB'

class LifeCycleA extends Component

{

constructor(props){

super(props)

this.state={

name:'Anuj'

}

console.log("LifeCycleA Constructor called")

}

static getDerivedStateFromProps(props,state)

{

console.log("LifeCycleA getDerivedStateFromProps called")

return null

}

componentDidMount(){

console.log("LifeCycleA component Did Mount called")

}

shouldComponentUpdate()

{

console.log("LifeCycleA Should component Update called")

return true

}

getSnapshotBeforeUpdate()

{

console.log("LifeCycleA get Snapshot Before Update called")

return null

}

componentDidUpdate(){

console.log("LifeCycleA component Did Update called")

return true

}

changeState = () => {

this.setState({

name:'CodeVolution'

})

}

render(){

console.log("LifeCycleA render called")

return(

<div>LifeCycle A

<LifeCycleB />

<button onClick={this.changeState}>Change State</button>

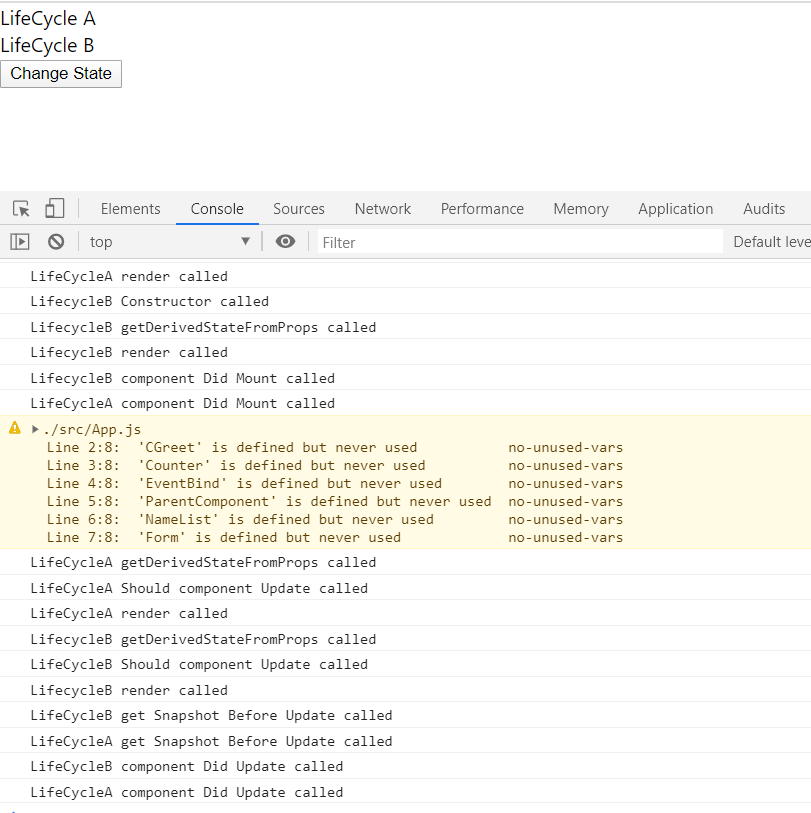
</div>

)

}

}

export default LifeCycleA



Express Creating Environment

Why Fragments ??

import React from 'react'

function FragmentDemo(){

return(

<div>

<h1>Fragment Demo</h1>

<p> This is example of the Fragment </p>

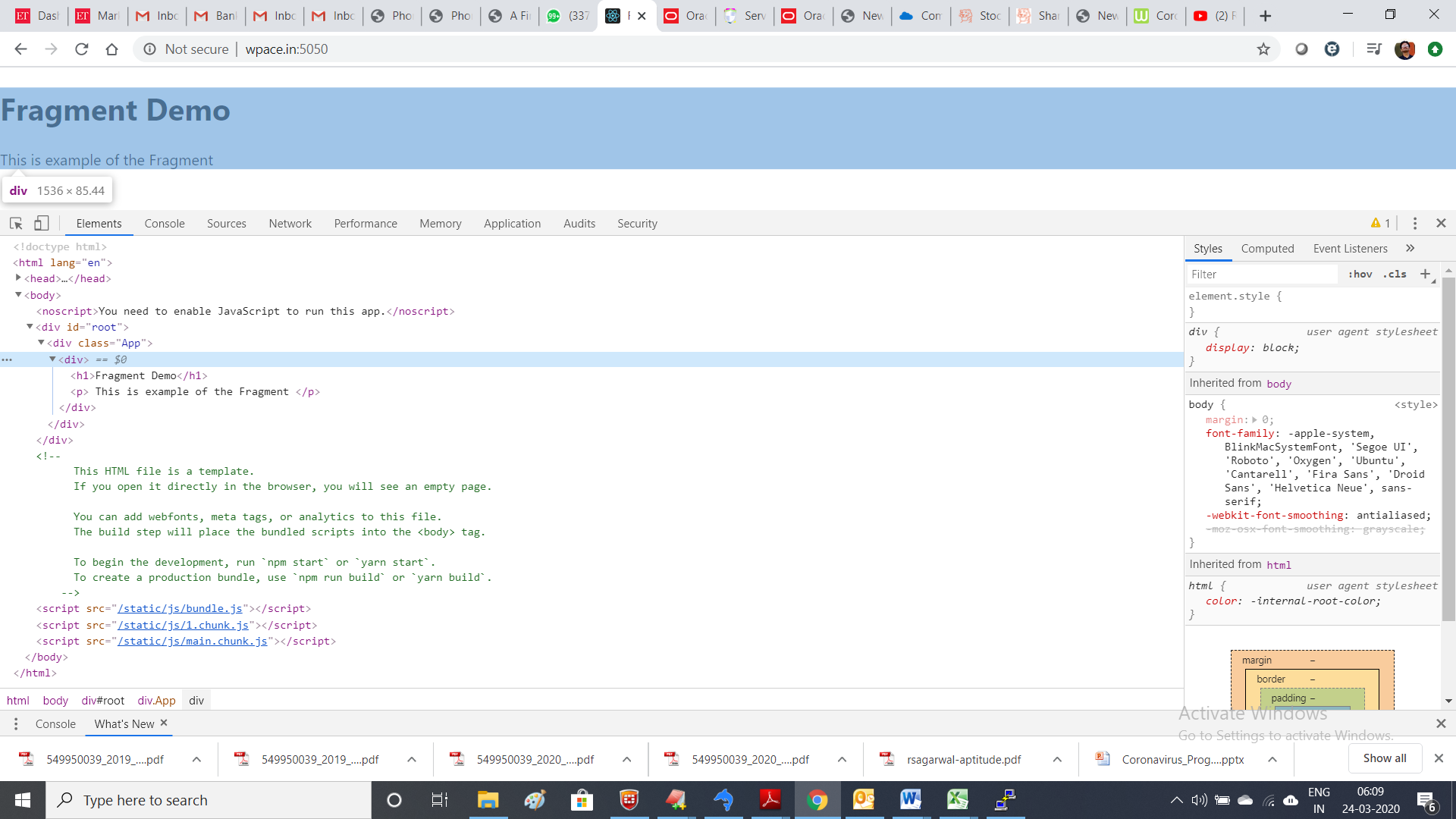
</div>

)

}

export default FragmentDemo

In normal scenario , we get the extra div tag in the DOM component .



So to remove it we use the following syntax instead of div tag.

import React from 'react'

function FragmentDemo(){

return(

<React.Fragment>

<h1>Fragment Demo</h1>

<p> This is example of the Fragment </p>

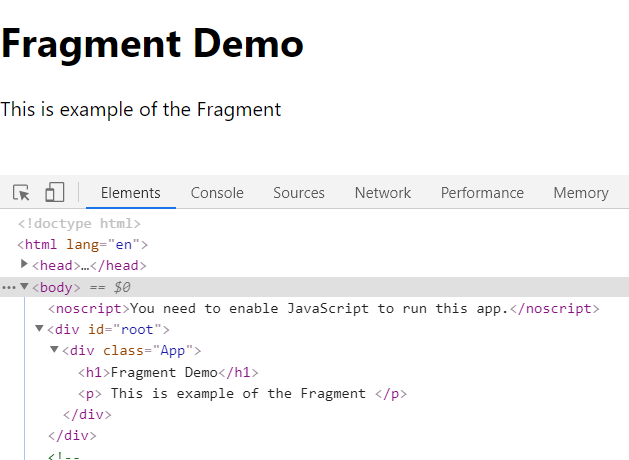
</ React.Fragment>>

)

}

export default FragmentDemo

Now with this the output is as follows :-



NO DIC Tag with this .