Preventing Rendering

* Preventing component from rendering

To prevent a component from rendering then based upon condition, return “**NULL**”

**Example:**

import React from 'react'

import { useState } from 'react';

import Card from '@mui/material/Card';

import CardContent from '@mui/material/CardContent';

import CardMedia from '@mui/material/CardMedia';

import Typography from '@mui/material/Typography';

import { CardActionArea } from '@mui/material';

import Divider from '@mui/material/Divider';

import Stack from '@mui/material/Stack';

function Rendering(props) {

if (props.value === 'notDisplay') {

return null;

}

if (props.value === 'display') {

return (

<h1>Hello World </h1>

)

}

}

function App() {

return (

<div className="App">

<div>

<h1 style={{ color: 'green' }}>

ReactJS Example

</h1>

<h3>

Preventing Rendering of

Components in React

</h3>

<Rendering value="display" />

<Rendering value="notDisplay" />

</div>

</div>

);

}

export default App;

**Output:**

ReactJS Example

Preventing Rendering of Components in React

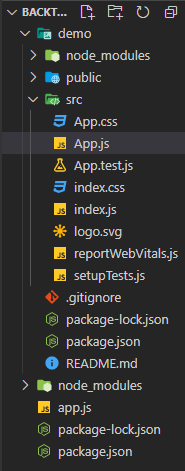
Hello World

* How to connect ReactJS with Node JS

**Prerequisites:**

* Basic knowledge of React and Node.
* Node.js installed (version 12+).
* npm installed (version 6+).

**Project Structure:**This is the structure when all the modules and required files are ready.



**Backend setup:** Firstly we will work on our backend(NodeJS) portion. In your working folder make a file named app.js for NodeJS and package.json file to run all the modules we required.

**Installing Module:**

* Installing ExpressJS is a nodeJS framework:

npm install express

* Installing nodemon:

npm install nodemon

**Configuration of package .json file:**Add the start and dev script, which are important for running and dynamically running the code after changes made in your Node.js app respectively in**package.json** file as shown below.

{

"name": "demoapp",

"version": "1.0.0",

"description": "",

"main": "app.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1",

"start": "node app.js",

"dev": "nodemon app.js"

},

"author": "",

"license": "ISC",

"dependencies": {

"express": "^4.17.1"

}

}

**Filename- app.js:**Here is the simple JavaScript code that should be written in app.js which is for NodeJS.

const express = require("express");

const app = express();

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

res.send("Hello World!");

});

const PORT = process.env.PORT || 8080;

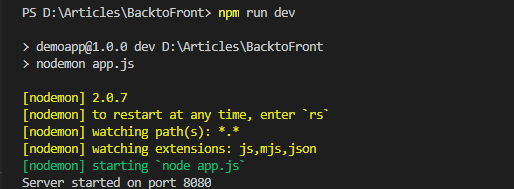
app.listen(PORT, console.log(`Server started on port ${PORT}`));

Run the application using the following command:

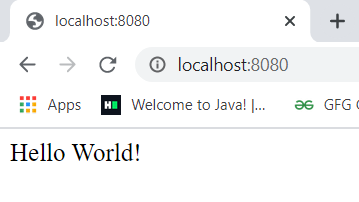
npm run dev

**Output:**

* This will be a console output.



Now go to **http://localhost:8080/** in your browser, you will see the following output.



**Filename- app.js:**Now for connecting the React part we have to make some changes in the app.js of NodeJS. We have completed the backend part.

const express = require("express");

const app = express();

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

console.log("Connected to React");

res.redirect("/");

});

const PORT = process.env.PORT || 8080;

app.listen(PORT, console.log(`Server started on port ${PORT}`));

**Frontend setup:**First, we have to create React app and run your app by writing the below command.

npx create-react-app demo

cd demo

npm start

**Output:**Now go to**http://localhost:3000/** in your browser, you will see the following output.



**Connecting:** We have completed both the frontend and backend parts, now we have to connect both. Now for connecting Reactjs with Nodejs we have added this line in package.json of react app folder:

"proxy": "http://localhost:8080

**Filename- package.json:** The package.json file is in your React app folder. This tells React to proxy API requests to the Node.js server built with Express in our project.

{

"name": "demo",

"version": "0.1.0",

"private": true,

"proxy": "http://localhost:8080",

"dependencies": {

"@testing-library/jest-dom": "^5.11.4",

"@testing-library/react": "^11.1.0",

"@testing-library/user-event": "^12.1.10",

"react": "^17.0.1",

"react-dom": "^17.0.1",

"react-scripts": "4.0.2",

"web-vitals": "^1.0.1"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test",

"eject": "react-scripts eject"

},

"eslintConfig": {

"extends": [

"react-app",

"react-app/jest"

]

},

"browserslist": {

"production": [

">0.2%",

"not dead",

"not op\_mini all"

],

"development": [

"last 1 chrome version",

"last 1 firefox version",

"last 1 safari version"

]}}

**Filename- App.js:**Made some changes in app.js of React to show that they are connected.

import logo from "./logo.svg";

import "./App.css";

function App() {

return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo"

alt="logo" />

<p>A simple React app.....</p>

<a

className="App-link"

href="https://reactjs.org"

target="\_blank"

rel="noopener noreferrer"

>

Learn React

</a>

<form action="../../post" method="post"

className="form">

<button type="submit">Connected?</button>

</form>

</header>

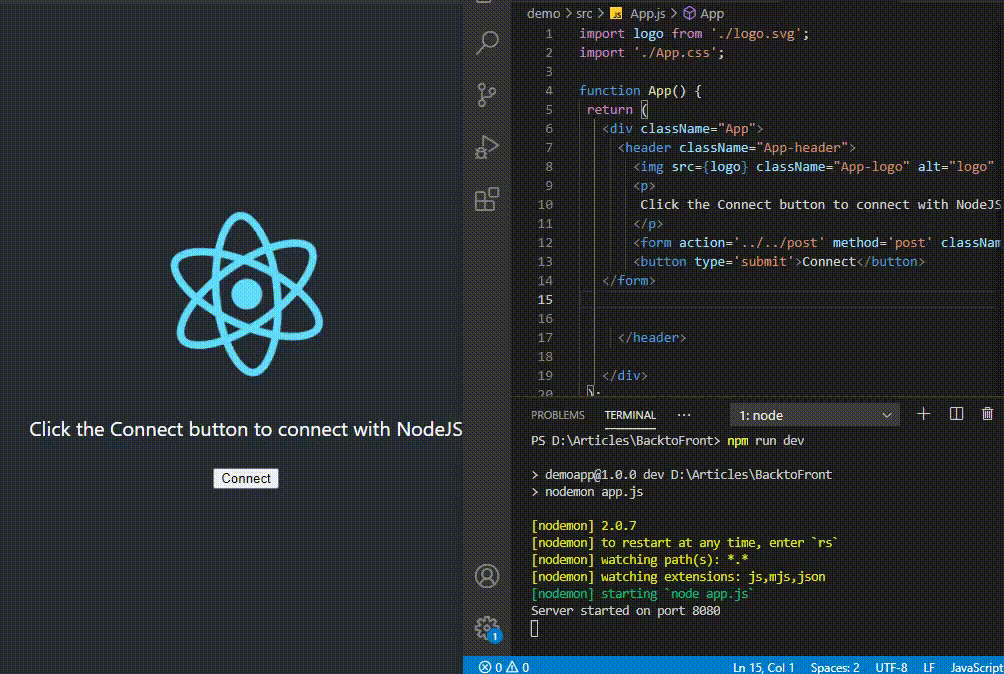
</div>

);}

export default App;

Now run the Nodejs process **npm run** **dev**in one terminal and in another terminal start Reactjs using **npm start** simultaneously**.**

**Output:**We see react output we see a button “Connect” we have to click it. Now when we see the console server-side we see that the ReactJS is connected with NodeJS.



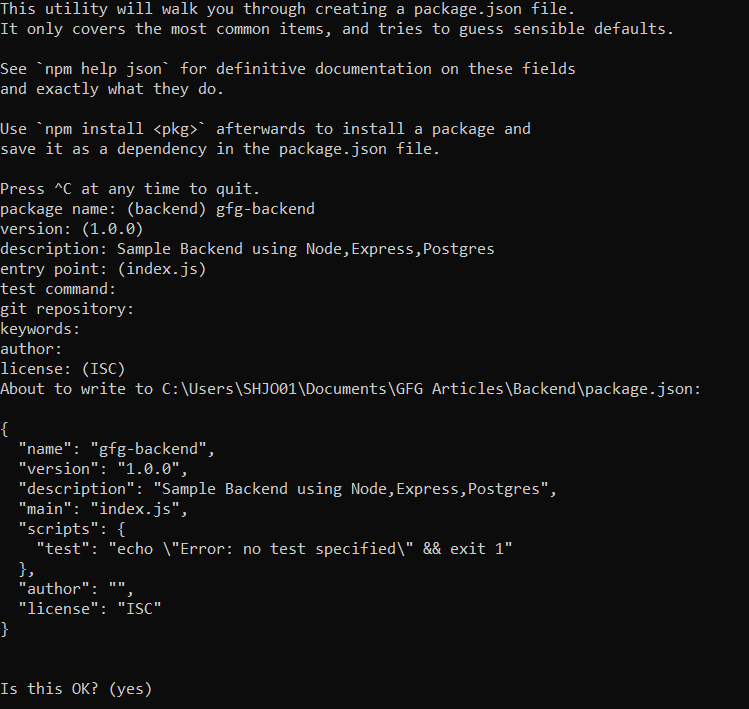
* Backend Setup

Go to the Directory where you want to create project

Initialize the Node Project using:

npm init

Type the name of Project and Other Details or Press Enter if you want to go with Default.



* Install express using npm

npm install --save express

* Install the node-postgres Client using npm

npm install --save pg

* Install the postgres module for serializing and de-serializing JSON data in to hstore format using npm.

npm install --save pg-hstore

* Create a file index.js as entry point to the backend.
* Now Install body-parser using npm

npm install --save body-parser

Now add the below code to index.js file which initiates the express server, creates a pool connection and also creates a REST API ‘/testdata’. Don’t forget to add your Password while pool creation in the below code.

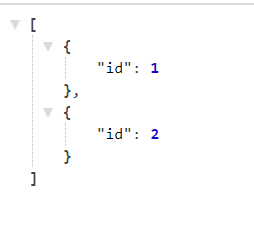
|  |
| --- |
| // Entry Point of the API Server  const express = require('express');  /\* Creates an Express application.     The express() function is a top-level     function exported by the express module.  \*/  const app = express();  const Pool = require('pg').Pool;  const pool = new Pool({      user: 'postgres',      host: 'localhost',      database: 'gfgbackend',      password: 'postgres',      dialect: 'postgres',      port: 5432  });  /\* To handle the HTTP Methods Body Parser     is used, Generally used to extract the     entire body portion of an incoming     request stream and exposes it on req.body  \*/  const bodyParser = require('body-parser');  app.use(bodyParser.json())  app.use(bodyParser.urlencoded({ extended: false }));  pool.connect((err, client, release) => {      if (err) {          return console.error(              'Error acquiring client', err.stack)      }      client.query('SELECT NOW()', (err, result) => {          release()          if (err) {              return console.error(                  'Error executing query', err.stack)          }          console.log("Connected to Database !")      })  })  app.get('/testdata', (req, res, next) => {      console.log("TEST DATA :");      pool.query('Select \* from test')          .then(testData => {              console.log(testData);              res.send(testData.rows);          })  })  // Require the Routes API  // Create a Server and run it on the port 3000  const server = app.listen(3000, function () {      let host = server.address().address      let port = server.address().port      // Starting the Server at the port 3000  }) |

* Now, start the backend server using:

node index.js

* Open Browser and try to router to:

http://localhost:3000/testdata

* Now, you can see the data from test table as follows:  
  
* propTypes

Before React 15.5.0, proptypes are available in the React package, but in later versions of React, you need to add a dependency to your project. You can add a dependency to your project using the below command:

npm install prop-types --save

We can use propType to validate any data we get from the prop. But before using it, we have to import it. Add the following line to the top of your index.js file:

import PropTypes from 'prop-types';

Once we have imported propTypes, we are ready to work with them. Like default props, propTypes are also objects where keys are the prop names and values are their types.

Below syntax shows how to use propTypes:

ComponentClassName.propTypes{

    propName1 : PropTypes.string,

    propName2 : PropTypes.bool,

    propName3 : PropTypes.array,

    .

    .

    .

    .

    propNamen : PropTypes.anyOtherType

}

In the above syntax, componentClassName is the name of the component class; Any other type can be any type we are allowed to pass as prop.

A warning will appear on the console for props that do not validate the type of data specified by propTypes. Let's look at a complete program that uses proptypes for validation for a better understanding:

import PropTypes from 'prop-types';

import React from 'react';

import ReactDOM from 'react-dom';

// Component

class ComponentExample extends React.Component{

    render(){

        return(

                <div>

                    {/\* printing all props \*/}

                    <h1>

                        {this.props.arrayProp}

                        <br />

                        {this.props.stringProp}

                        <br />

                        {this.props.numberProp}

                        <br />

                        {this.props.boolProp}

                        <br />

                    </h1>

                </div>

            );

    }

}

// Validating prop types

ComponentExample.propTypes = {

    arrayProp: PropTypes.array,

    stringProp: PropTypes.string,

    numberProp: PropTypes.number,

    boolProp: PropTypes.bool,

}

// Creating default props

ComponentExample.defaultProps = {

    arrayProp: ['Denail', 'Sairin', 'Rompol'],

    stringProp: "JavaTpoint",

    numberProp: "10",

    boolProp: true,

}

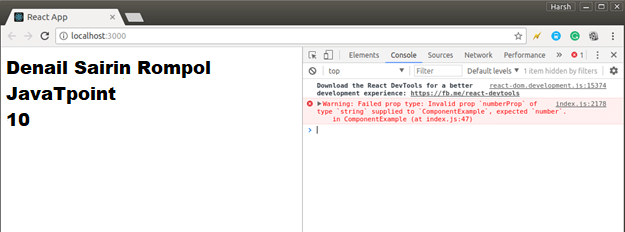
ReactDOM.render(

    <ComponentExample  />,

    document.getElementById("root")

);

**Output:**



* getDefaultProps

The **getDefaultProps()** method is called once and cached before any instance of the component are created. This method returns an object which properties values will be set on **this.props** if that prop is not specified by the parent component.

The **getInitialState()**method is also invoked once, right before the mounting phase. The return value of this method will be used as initial value of **this.state** and should be an object.

**Example:**

//getDefaultProps

var Person = React.createClass({getDefaultProps: function() {  
  
return ({ age: ‘unknown’ });  
  
},  
  
//…..  
  
});

//getInitialState

var MessageBox = React.createClass({  
  
getInitialState: function() {  
  
return {nameWithQualifier: ‘Mr. ‘ + this.props.name};  
  
},  
  
//…  
  
});

* Binding the context( Inline)

an inline function is a function that is defined and passed down inside the render method of a React component.

**Alternatives to inline function**

**Bind in constructor:** One of the most common patterns is to define the function within the class component and then bind context to the function in constructor. We only need to bind the current context if we want to use this keyword inside the handler function.

export default class CounterApp extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

this.increaseCount = this.increaseCount.bind(this);

}

increaseCount() {

this.setState({ count: this.state.count + 1 });

}

render() {

return (

<div className="App">

<button onClick={this.increaseCount}>COUNT ({this.state.count})</button>

</div>

);

}

}

**Bind in render:** Another common pattern is to bind the context inline when the function is passed down. Eventually, this gets repetitive and hence the first approach is more popular.

render() {

return (

<div className="App">

<button onClick={this.increaseCount.bind(this)}>COUNT ({this.state.count})</button>

</div>

);

}

**Define as public field:**

increaseCount = () => {

this.setState({ count: this.state.count + 1 });

};

render() {

return (

<div className="App">

<button onClick={this.increaseCount}>

COUNT ({this.state.count})

</button>

</div>

);

}

* UI without ReactJS & Problem
* How the ReactJS solving the problem

The main problem solved by react is the development of large applications that have temporal data changes. The data in such applications is not static so it changes over time. We can look at React as the V layer in model-view-controller (MVC) pattern of software development.

**React has several advantages that make it an excellent choice for developing user interfaces.**

* It is simple. You just need to express the appearance of your app and React takes care of your user interface updates when there are changes in the data
* It is declarative. When changes in data happen React is aware of how to update the changes
* React is component based. This makes development of reusable code that is easy to test.