Amrita School of Computing

Department of Computer Science Engineering

22AIE457

Full Stack Development

Lab Manual

VII SEM CSE(AIE)

ODD SEM 2025-2026

Amrita Vishwa Vidyapeetham

Chennai

Ex 1

Git and GitHub Tutorial – Version Control for Beginners

Prerequisites

To complete this tutorial, you'll need the following:

- A command line interface.
- A text editor of your choice (I will be using VS Code).
- A GitHub account

What is Git?

- Git is a version control system which lets you track changes you make to your files over time. With Git, you can revert to various states of your files (like a time traveling machine). You can also make a copy of your file, make changes to that copy, and then merge these changes to the original copy.
- For example, you could be working on a website's landing page and discover that you do not like the navigation bar. But at the same time, you might not want to start altering its components because it might get worse.
- With Git, you can create an identical copy of that file and play around with the navigation bar. Then, when you are satisfied with your changes, you can merge the copy to the original file.
- You are not limited to using Git just for source code files – you can also use it to keep track of text files or even images. This means that Git is not just for developers – anyone can find it helpful.

How to install Git

- To use Git, you have to install it on your computer. To do this, you can download the latest version on the official website. You can download for your operating system from the options given.
- You can also install Git using the command line, but since the commands vary with each operating system, we'll focus on the more general approach.

How to configure Git

- I will assume that at this point you have installed Git. To verify this, you can run this command on the command line: git --version. This shows you the current version installed on your PC.
- The next thing you'll need to do is to set your username and email address. Git will use this information to identify who made specific changes to files.
- To set your username, type and execute these commands: git config --global user.name "YOUR_USERNAME" and git config --global user.email "YOUR_EMAIL". Just make sure to replace "YOUR_USERNAME" and "YOUR_EMAIL" with the values you choose.

How to Create and Initialize a Project in Git

- We are finally done with installing and setting up Git. It is now time to create our project.
- I have created a folder on my desktop called Git and GitHub tutorial. Using the command line, navigate to your new project's location. For me, I would run the following commands:
- cd desktop
- cd Git and GitHub tutorial

- If you are new to the command line and are still learning how to use it to navigate around your PC, then I would suggest using Microsoft's Visual Studio Code. It is a code editor which has an inbuilt terminal for executing commands. You can download it here.
- After installing VS Code, open your project in the editor and open a new terminal for your project. This automatically points the terminal/command line to your project's path.
- Now to initialize your project, simply run git init. This will tell Git to get ready to start watching your files for every change that occurs. It looks like this:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial
$ git init
Initialized empty Git repository in C:/Users/IHECHIKARA/Desktop/Git and GitHub tutorial/.git/
```

git init

- The first line has information about my PC and the path to where the folder exists. The second line is the command git init, and the third line is the response sent back telling me that my repository (repo) has been initialized. It is considered empty because we have not told Git what files to track.
- A repository is just another way to define a project being watched/tracked by Git.

Git project files

- I have created only one file called todo.txt. This is what the file looks like:
- MY TO-DO LIST

- 1. Write an article.
- 2. Code.
- 3. Study books.
- 4. Attend classes on time.
- 5. Visit aunt.
- 6. Apply for remote jobs.
- Before we proceed with learning other Git commands, let's talk about GitHub.

What is GitHub?

- GitHub is an online hosting service for Git repositories. Imagine working on a project at home and while you are away, maybe at a friend's place, you suddenly remember the solution to a code error that has kept you restless for days.
- You cannot make these changes because your PC is not with you. But if you have your project hosted on GitHub, you can access and download that project with a command on whatever computer you have access to. Then you can make your changes and push the latest version back to GitHub.
- In summary, GitHub lets you store your repo on their platform. Another awesome feature that comes with GitHub is the ability to collaborate with other developers from any location.
- Now that we have created and initialized our project locally, let's push it to GitHub.
- If you are a beginner, you will come across some new terms like push, commit, add, and so on – but do not be overwhelmed by them. With some practice you will be able to remember these terms and what they do.

How to push a repository to GitHub

- I will divide this section into steps to help you understand the process more clearly.

Step 1 – Create a GitHub account

- To be able to use GitHub, you will have to create an account first. You can do that on their website.

Step 2 – Create a repository

- You can click on the + symbol on the top right corner of the page then choose "New repository". Give your repo a name then scroll down and click on "Create repository".

Step 3 – Add and commit file(s)

- Before we "add" and "commit" our files, you need to understand the stages of a file being tracked by Git.

Committed state

- A file is in the **committed** state when all the changes made to the file have been saved in the local repo. Files in the committed stage are files ready to be pushed to the remote repo (on GitHub).

Modified state

- A file in the **modified** state has some changes made to it but it's not yet saved. This means that the state of the file has been altered from its previous state in the committed state.

Staged state

- A file in the **staged** state means it is ready to be committed. In this state, all necessary changes have been made so the next step is to move the file to the commit state.
- You can understand this better by imagining Git as a camera. The camera will only take a snapshot when the file reaches the commit state. After this state, the camera starts comparing changes being made to the same file with the last snapshot (this is the modified state). And when the required changes have been made, the file is staged and moved to the commit state for a new snapshot.
- This might be a lot of information to take in at the moment, but do not be discouraged – it gets easier with practice.

How to add files in Git

- When we first initialized our project, the file was not being tracked by Git. To do that, we use this command git add. The period or dot that comes after add means all the files that exist in the repository. If you want to add a specific file, maybe one named about.txt, you use git add about.txt.
- Now our file is in the staged state. You will not get a response after this command, but to know what state your file is in, you can run the git status command.

How to commit files in Git

- The next state for a file after the staged state is the committed state. To commit our file, we use the git commit -m "first commit" command.
- The first part of the command git commit tells Git that all the files staged are ready to be committed so it is time to take a snapshot. The second part -m "first commit" is the commit message. -m is shorthand for message while the text inside the parenthesis is the commit message.
- After executing this command, you should get a response similar to this:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git commit -m "first commit"
[main (root-commit) 48195f0] first commit
 1 file changed, 8 insertions(+)
 create mode 100644 todo.txt
```

git commit

- Now our file is in the committed state.

Step 4 – Push the repository to GitHub

- After you create the repo, you should be redirected to a page that tells you how to create a repo locally or push an existing one.
- In our case, the project already exists locally so we will use commands in the "…or push an existing repository from the command line" section. These are the commands:
- git remote add origin https://github.com/ihechikara/git-and-github-tutorial.git
- git branch -M main
- git push -u origin main
- The first command git remote add origin https://github.com/ihechikara/git-and-github-tutorial.git creates a connection between your local repo and the remote repo on Github.
- The URL for your remote project should be entirely different from the one above. So to follow along, make sure you are following the steps and working with your own remote repo. You won't usually get a response after executing this command but make sure you have an internet connection.
- The second command git branch -M main changes your main branch's name to "main". The default branch might be created as "master", but "main" is the standard name for this repo now. There is usually no response here.
- The last command git push -u origin main pushes your repo from your local device to GitHub. You should get a response similar to this:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 325 bytes | 325.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ihechikara/git-and-github-tutorial.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

git push

- To help you deepen your understanding of file stages, I will make changes to the file and then push the new version to GitHub.
- Recall that our file is now in the committed state. Let's make changes to the file and take note of the states.
- I am going to add a new task to the to-do list:
- MY TO-DO LIST

- 1. Write an article.

- 2. Code.
- 3. Study books.
- 4. Attend classes on time.
- 5. Visit aunt.
- 6. Apply for remote jobs.
- 7. Practice code
- After adding the new task, run the git status command. This is what you should see:

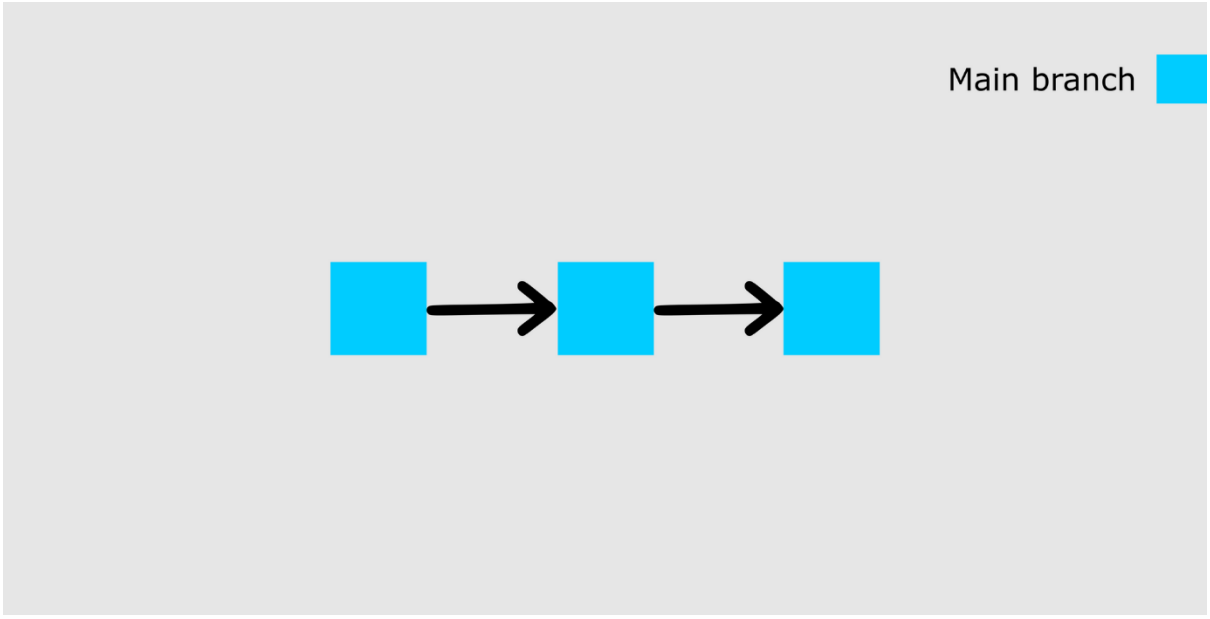
```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   todo.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

git status

- After making changes to the file, it moved to the modified state – but it's not yet staged for commit, so you can't push it to GitHub yet. Git has not taken a final snapshot of this current state as it's only comparing the changes we have made now with the last snapshot.
- Now we are going to add (stage) this file and then commit and push it. This is the same as in the last section.
- We first add the file by using git add . which adds all the files in the folder (one file in our case). Then we commit the file by running git commit -m "added new task" followed by git push -u origin main.
- Those are the three steps to pushing your modified files to GitHub. You add, commit, and then push. I hope you now understand file stages and the commands associated with them.

  ****

  How to Use Branches in Git
- With branches, you can create a copy of a file you would like to work on without messing up the original copy. You can either merge these changes to the original copy or just let the branch remain independent.
- Before we go into using branches, I want to show you a visual representation of our repo which looks like this:



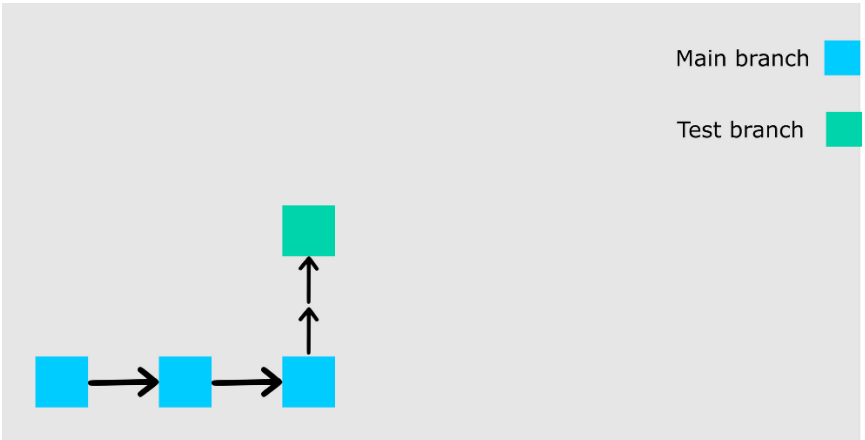
main banch

- The image above shows our main branch with the last two commits (the first commit and the added new task commit).
- At this point, I want to add more tasks to the list but I am not yet sure whether I want them on my main list. So I will create a new branch called test to see what my list would look like with more tasks included.
- To create a new branch, run this command: git checkout -b test. I will break it down.
- checkout tells Git it is supposed to switch to a new branch. -b tells Git to create a new branch. test is the name of the branch to be created and switched to. Here is the response you should get:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git checkout -b test
Switched to a new branch 'test'
```

git checkout -b

- Now that we have a new branch created, this is what our repo will look like:



git branch

- We created the new branch from the state of our last commit. Let's now add more tasks to this new branch.
- MY TO-DO LIST

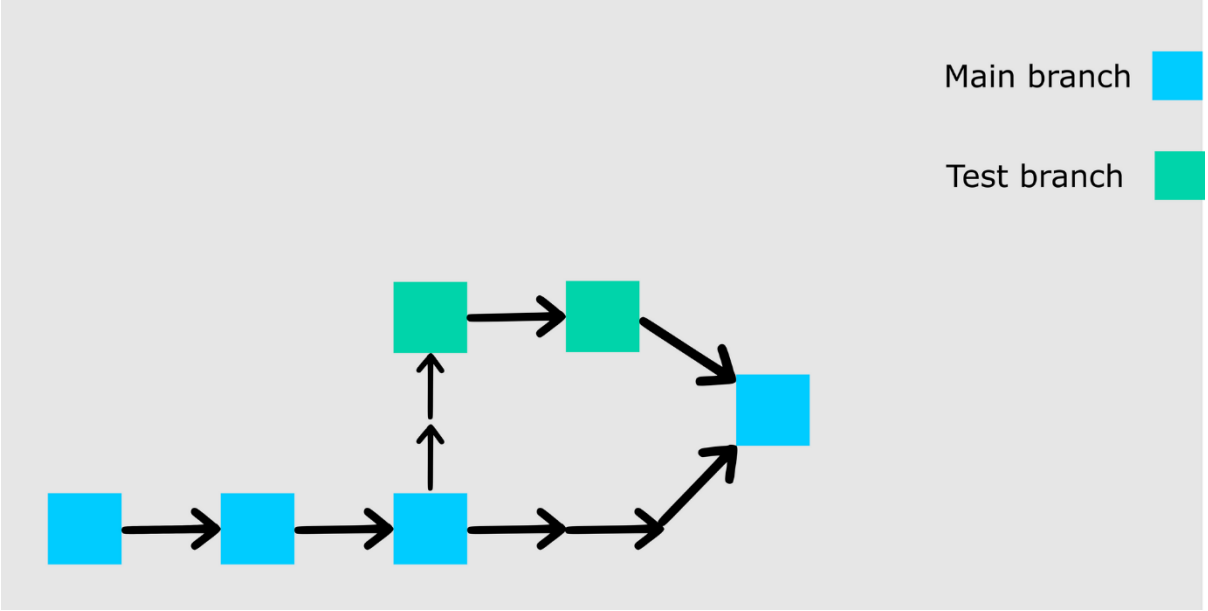
- 1. Write an article.
- 2. Code.
- 3. Study books.
- 4. Attend classes on time.
- 5. Visit aunt.
- 6. Apply for remote jobs.
- 7. Practice code
- 8. Complete internship task.
- 9. Practice chess openings.
- 10. Solve chess puzzles.
- 11. Check exam schedule.

- I have added four new tasks. To merge the new state to the main branch, you have to first stage and commit this branch. I will not go into details about how to do this as we did it twice in the last section.
- You should try doing it yourself so you understand how it works. As a hint, add the file and then commit with a message (refer to the previous section for details showing you how to do that).
- After committing your test branch, switch back to the main branch by running this command: git checkout main.
- Did you notice that we did not add -b ? This is because we are not creating a new branch but rather switching to an existing one. You can check all the branches that exist in your repo by running the git branch command.
- Now we can merge the changes we made in the test branch into the main branch by running git merge test. At this point, you will see all the changes made in the test branch reflected on the main branch. You should also receive a response similar to this:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git merge test
Updating 61dad8f..33a2410
Fast-forward
 todo.txt | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)
```

git merge

- Here is a visual representation of our repo:



git merge

- If you go on to push your repo to GitHub, you'll see that the test branch will not be pushed. It will only remain in your local repo. If you would like to push your test branch, switch to the branch using git checkout test and then run git push -u origin test.

- **How to Pull a Repository in Git**
- To pull in Git means to clone a remote repository's current state into your computer/repository. This comes in handy when you want to work on your repo from a different computer or when you are contributing to an open source project online.
- To test this, don't worry about switching to a new computer. Just run cd .. to leave the current directory and go back one step. In my own case, I have navigated back to my desktop.
- Go to GitHub, and on your repository's main page you should see a green button that says "Code". When you click on the button, you should see some options in a dropdown menu. Go on and copy the HTTPS URL.
- After that, run git clone YOUR_HTTPS_URL. This command pulls the remote repository into your local computer in a folder called git-and-git-tutorial. That is:

```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop
$ git clone https://github.com/ihechikara/git-and-github-tutorial.git
Cloning into 'git-and-github-tutorial'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 14 (delta 2), reused 10 (delta 1), pack-reused 0
Receiving objects: 100% (14/14), done.
Resolving deltas: 100% (2/2), done.
```

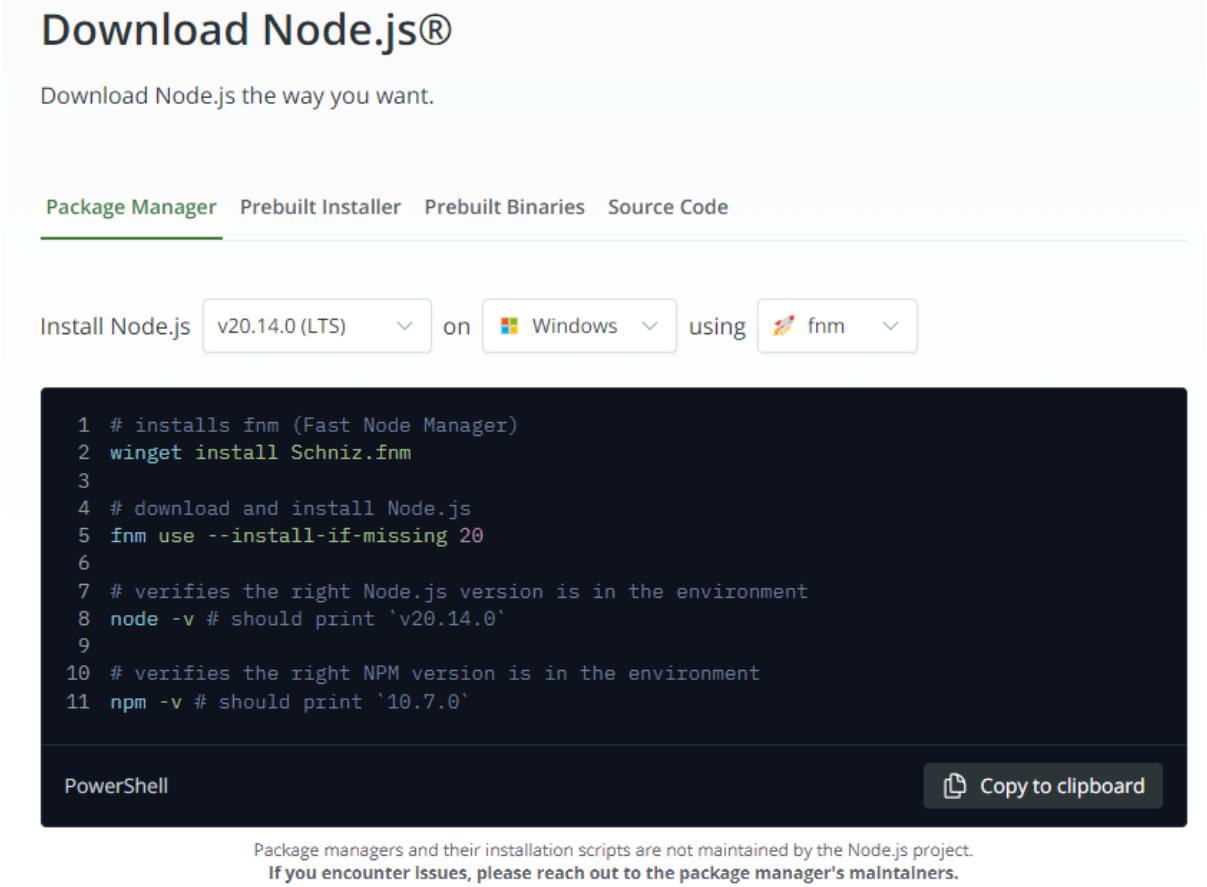
git clone

Ex 2:

Installation of Node JS on Windows

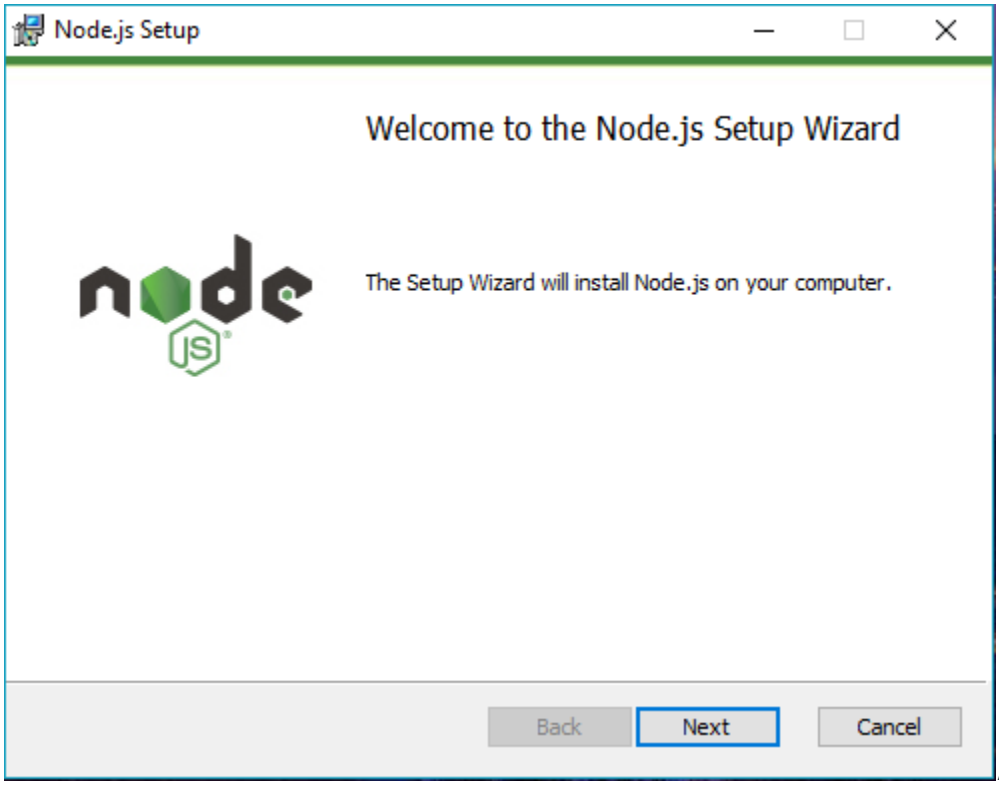
Node.js can be installed in multiple ways on a computer. The approach used by you depends on the existing Node.js development environment in the system. There are different package installers for different environments. You can install Node.js by grabbing a copy of the source code and compiling the application. Another way of installing Node.js is by cloning the Node.js GIT repository in all three environments and then installing it on the system.

Installing Node On Windows (WINDOWS 10)
You have to follow the following steps to install the Node.js on your Windows :
Step 1: Download the NodeJS
Downloading the Node.js '.msi' installer the first step to install Node.js on Windows is to download the installer. Visit the official Node.js website i.e) https://nodejs.org/en/download/



Download Node.js

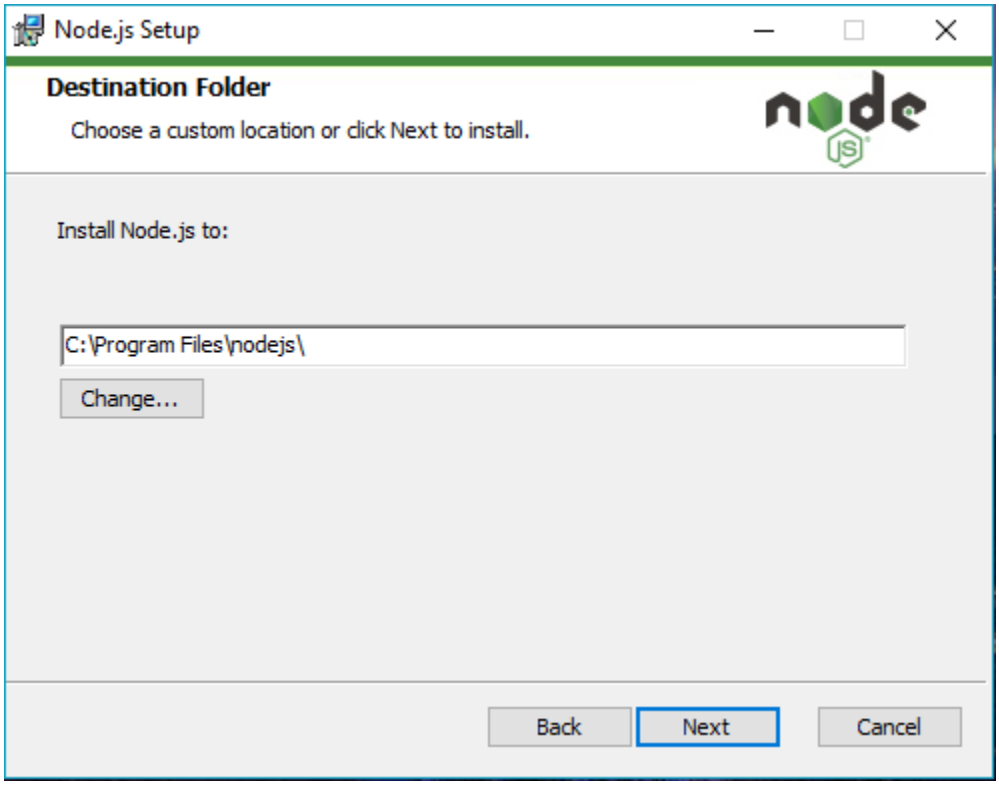
Step 2: Running the Node.js installer
Now you need to install the node.js installer on your PC. You need to follow the following steps for the Node.js to be installed:
Double-click on the .msi installer
The Node.js Setup wizard will open.
Welcome To Node.js Setup Wizard
Select "Next"



After clicking "Next", the End-User License Agreement (EULA) will open.

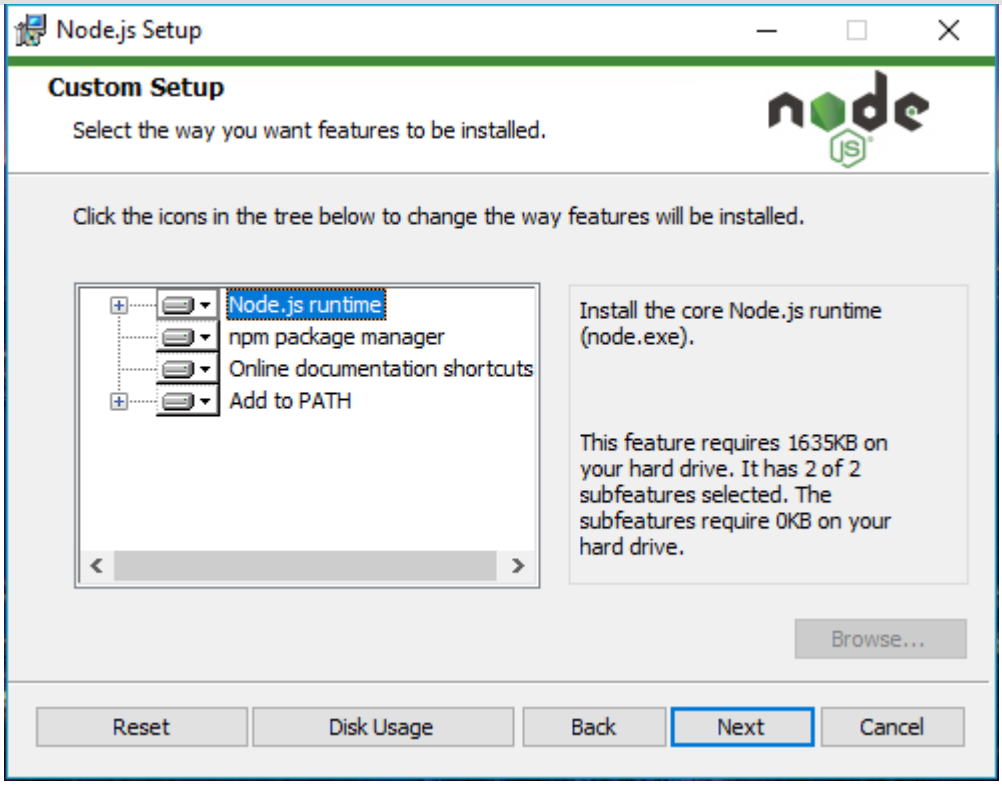
Check "I accept the terms in the License Agreement"
Select "Next"

Destination Folder Set the Destination Folder where you want to install Node.js & Select "Next".



Custom Setup

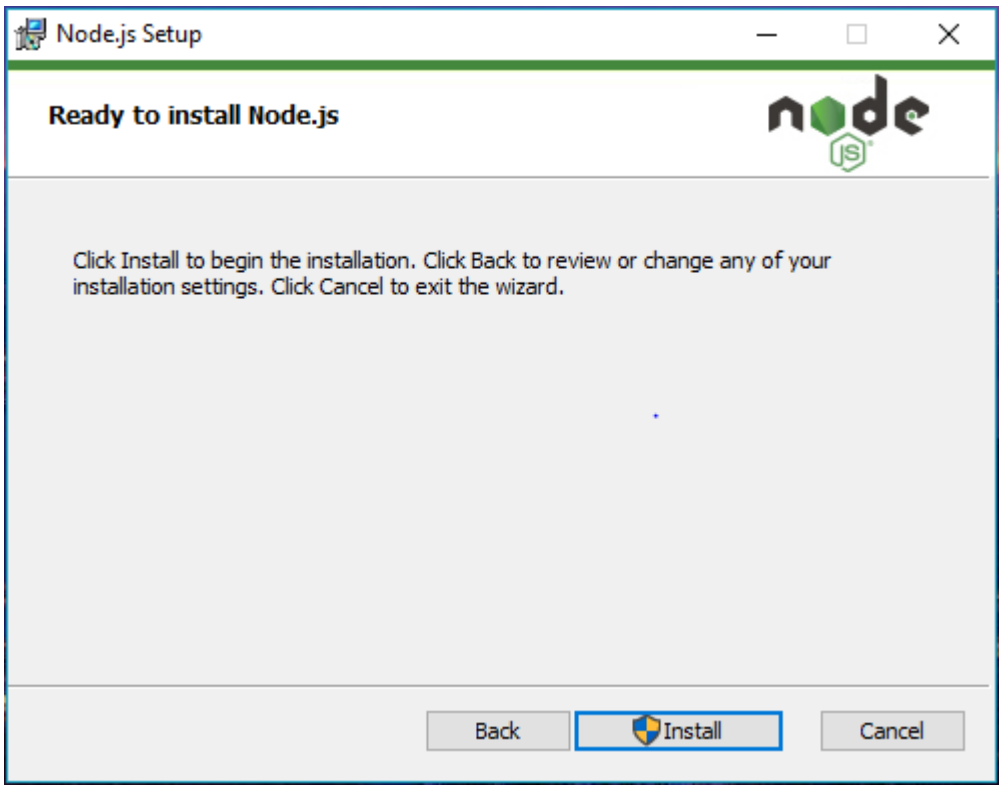
Select “Next”



Ready to Install Node.js.

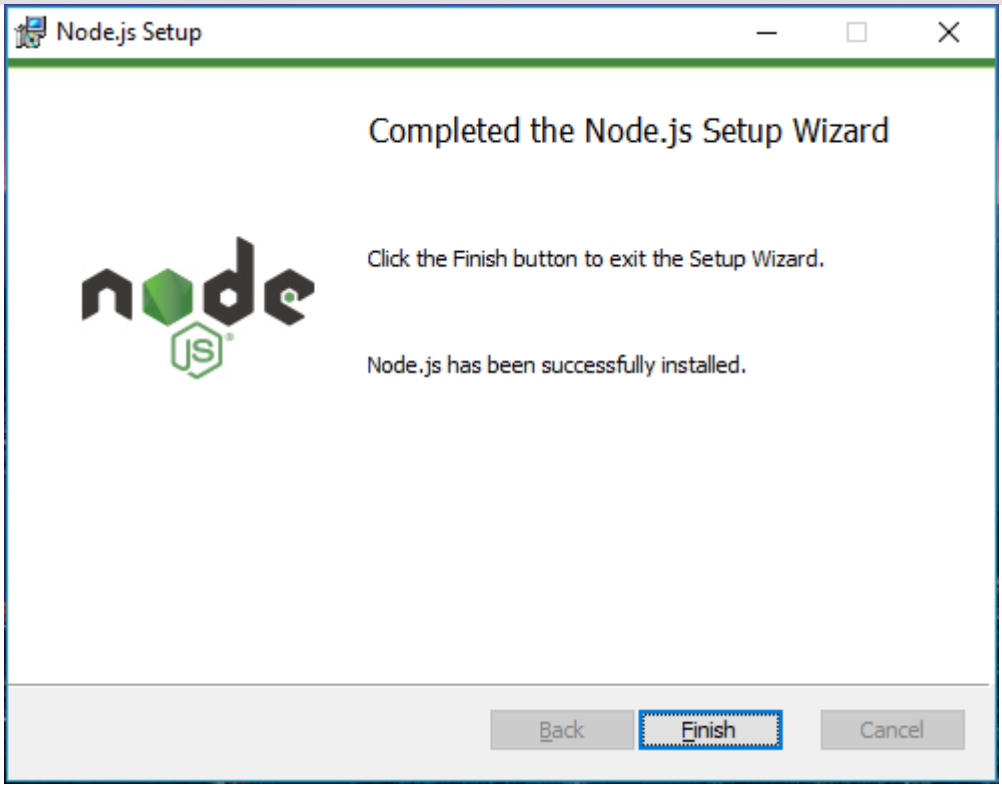
The installer may prompt you to “install tools for native modules”.

Select “Install”



Do not close or cancel the installer until the installation is complete. Complete the Node.js Setup Wizard.

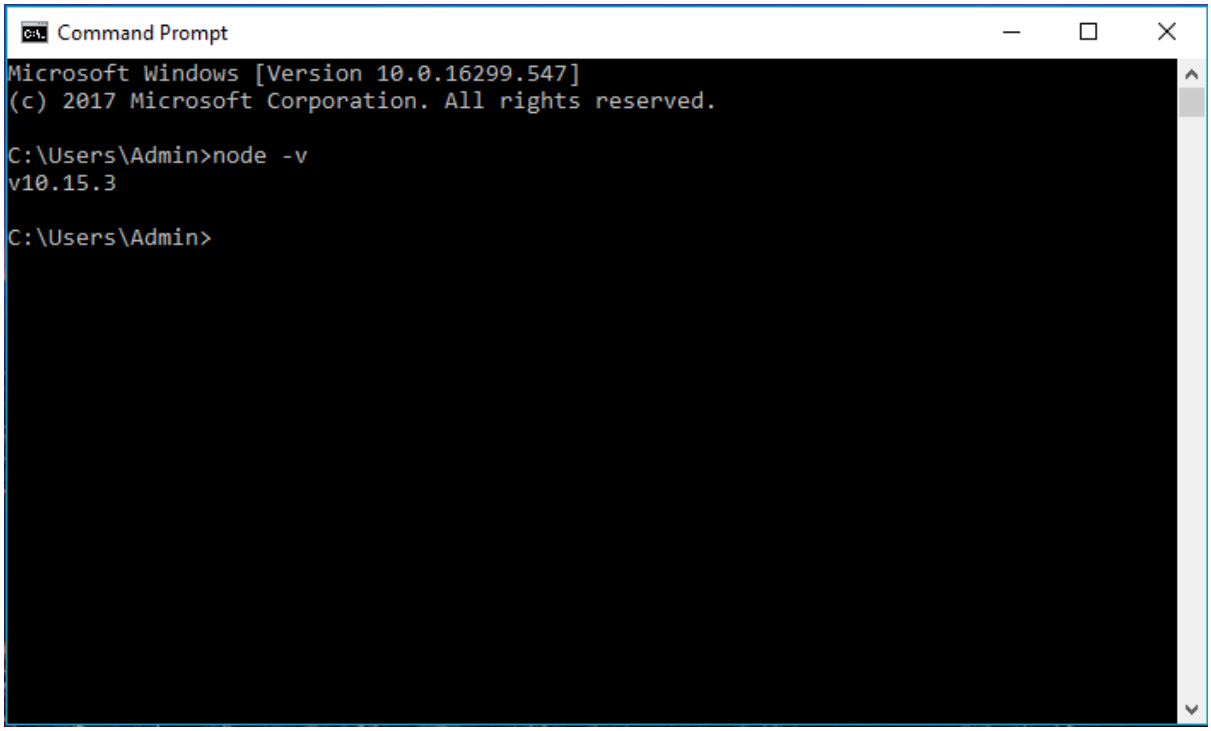
Click "Finish"



Step 3: Verify that Node.js was properly installed or not

To check that node.js was completely installed on your system or not, you can run the following command in your command prompt or Windows Powershell and test it:-

```
C:\Users\Admin> node -v
```



If node.js was completely installed on your system, the command prompt will print the version of the Node JS installed.

Step 4: Updating the Local npm version

You can run the following command, to quickly update the npm

```
npm install npm --global // Updates the 'CLI' client
```

Step 5 : Running Sample application

An Example Node.js Application

The most common example Hello World of Node.js is a web server:

```
const { createServer } = require('node:http');

const hostname = '127.0.0.1';

const port = 3000;

const server = createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader('Content-Type', 'text/plain');

  res.end('Hello World');

});

server.listen(port, hostname, () => {
```

```
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

To run this snippet, save it as a server.js file and run node server.js in your terminal.

This code first includes the Node.js http module.

Node.js has a fantastic standard library, including first-class support for networking.

The createServer() method of http creates a new HTTP server and returns it.

The server is set to listen on the specified port and host name. When the server is ready, the callback function is called, in this case informing us that the server is running.

Whenever a new request is received, the request event is called, providing two objects: a request (an http.IncomingMessage object) and a response (an http.ServerResponse object).

Those 2 objects are essential to handle the HTTP call.

The first provides the request details. In this simple example, this is not used, but you could access the request headers and request data.

The second is used to return data to the caller.

In this case with:

```
res.statusCode = 200;
```

we set the statusCode property to 200, to indicate a successful response.

We set the Content-Type header:

```
res.setHeader('Content-Type', 'text/plain');
```

and we close the response, adding the content as an argument to end():

```
res.end('Hello World\n');
```

STEP 6: Open browser and type address 127.0.0.1:3000

Output: helloworld

Step 7 : VScode nodejs

Create a folder

Create file First.js with following content

```
const x=" My first web page"
```

```
Console.log(x)
```

Go to terminal and run

node .\First.js

Ex 3

3 a) Basic Structure of an HTML Document

An HTML document typically starts with a <!DOCTYPE html> declaration, followed by the <html>, <head>, and <body> tags.

```
html
<!DOCTYPE html>
<html>
<head>
    <title>My Webpage</title>
</head>
<body>
    <!-- Content goes here -->
</body>
</html>
```

Common HTML Tags and Their Usage

Headers

Header tags range from <h1> to <h6>, with <h1> being the largest and most important heading, and <h6> the smallest.

```
html
<h1>Main Heading</h1>
<h2>Subheading</h2>
<h3>Sub-subheading</h3>
```

Paragraphs (<p>)

Paragraphs are used to define blocks of text.

```
html
Copy code
<p>This is a paragraph of text.</p>
```

Comments

Comments are not displayed in the browser and are used to leave notes within the code.

```
html
<!-- This is a comment -->
```

Divisions ( )

The tag is used to group block-level content or elements.

```
html
Copy code
```

```
<div>
   <h2>Section Title</h2>
   <p>This is a paragraph within a div.</p>
</div>
```

Lists

There are two main types of lists: ordered (<ol>) and unordered (<ul>).

- **Ordered List**:

```
html
<ol>
   <li>First item</li>
   <li>Second item</li>
   <li>Third item</li>
</ol>
```

- **Unordered List**:

```
html
<ul>
   <li>First item</li>
   <li>Second item</li>
   <li>Third item</li>
</ul>
```

Definition List

Definition lists are used for terms and their definitions.

```
html
<dl>
   <dt>HTML</dt>
   <dd>Hypertext Markup Language</dd>
   <dt>CSS</dt>
   <dd>Cascading Style Sheets</dd>
</dl>
```

Links (<a>)

Links are created using the <a> tag.

```
html
<a href="https://www.example.com">Visit Example</a>
```

To create a mailto link, which opens the user's email client:

```
html
<a href="mailto:someone@example.com">Send Email</a>
```

To create a call link, which initiates a phone call on mobile devices:

```
html
<a href="tel:+1234567890">Call Us</a>
```

Self-Closing Tags

Some HTML tags are self-closing, meaning they don't need an explicit closing tag. Examples include <img>,
, and <hr>.

```
html
<img src="image.jpg" alt="Description">
<br> <!-- Line break -->
<hr> <!-- Horizontal rule -->
```

Input Tags

Input tags are used within forms to collect user data.

```
html
<form>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name">

    <label for="email">Email:</label>
    <input type="email" id="email" name="email">

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

Additional Elements

Download Links

To create a link that allows users to download a file, use the download attribute in the <a> tag.

```
html
<a href="file.zip" download>Download File</a>
```

```
 </html>
```

3 b) Design the following static web pages required for an online book store web site.
i. Home Page ii. Login Page iii. Catalogue Page

1) **HOME PAGE:**

The static home page must contain three **frames**.

Top frame : Logo and the college name and links to Home page, Login page, Registration page,

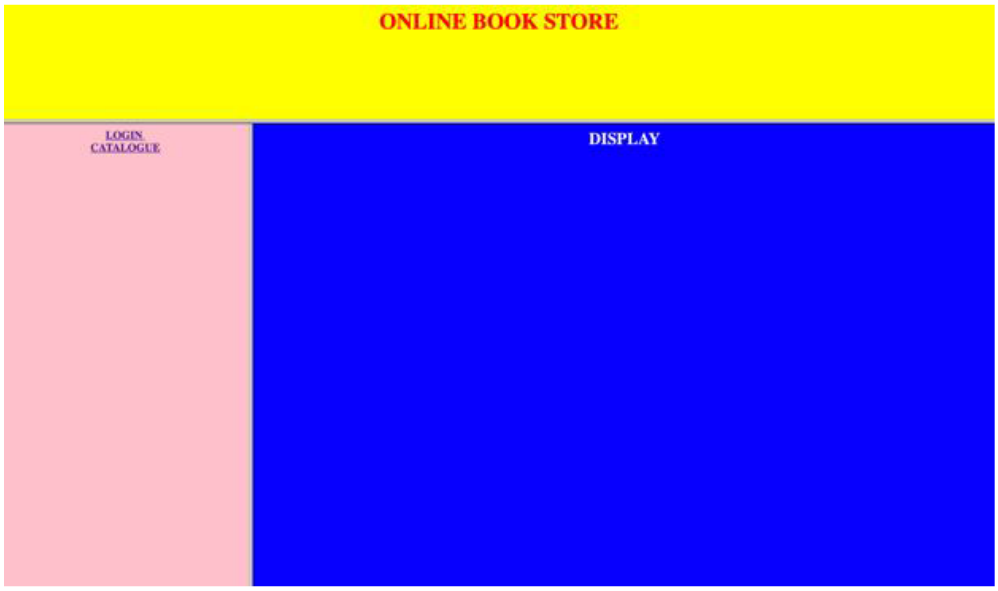
Catalogue page and Cart page (the description of these pages will be given below).

Left frame : At least four links for navigation, which will display the catalogue of respective links.

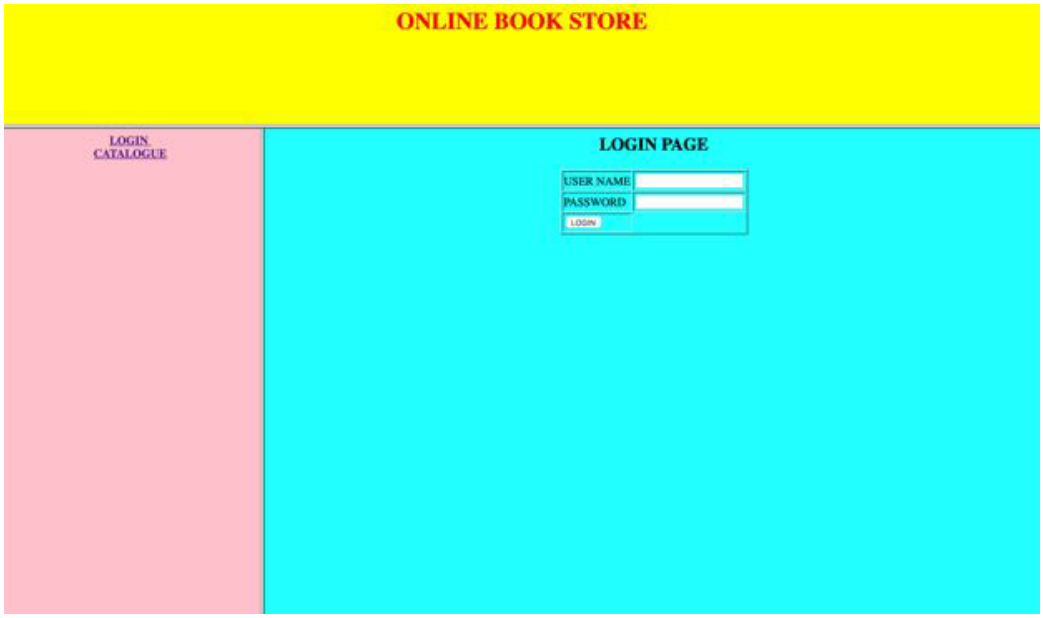
For e.g.: When you click the link **"CSE"** the catalogue for **CSE** Books should be displayed in the

Right frame.

Right frame: The pages to the links in the left frame must be loaded here. Initially this page contains description of the web site.

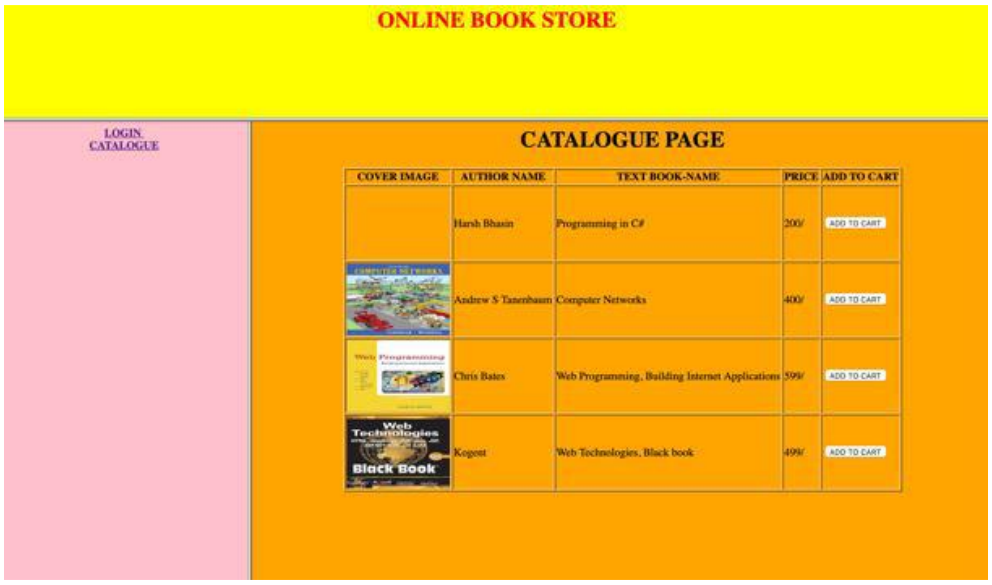


LOGIN PAGE: This page looks like below:



CATOLOGUE PAGE:
The catalogue page should contain the details of all the books available in the web site in a table.
The details should contain the following:
1. Snap shot of Cover Page.
2. Author Name.
3. Publisher.

4. Price.
5. Add to cart button.



Source Code

home.html

```
<html>
<head>
<frameset rows="20%,80%">
<frame name="title" src="titlepage.html">
<frameset cols="25%,75%">
<frame name="list" src="list.html">
<frame name="display" src="display.html">
</frameset>
</frameset>
</head>
</html>
```

titlepage.html

```
<html>
<head>
<title>title page</title>
</head>
<body bgcolor="yellow" text="red">
<h1 align="center"> ONLINE BOOK STORE </h1>
</body>
</html>
```

list.html

```
<html>
<head>
<title> List</title>
</head>
<body bgcolor="pink">
```

```
<center>
<a href="login.html" target="display"><b> LOGIN </b></a><br>
<a href="catalogue.html" target="display"><b> CATALOGUE </b></a>
</center>
</body>
</html>
```

Left.html

```
<html>
<head>
<title>Top Page</title>
</head>
<body>
<a href="cse.html" target="body">CSE</a><br />
<a href="ece.html" target="body">ECE</a><br />
<a href="eee.html" target="body">EEE</a><br />
<a href="civil.html" target="body">CIVIL</a><br />

</body>
</html>
```

Login.html

```
<html>
<body bgcolor="cyan">
<h2 align="center">LOGIN PAGE</h2>
<center>
<form>
<table border="1">
<tr>
<td><label>USER NAME</label></td>
<td><input type="text" name="uname"></td>
</tr>
<tr>
<td><label>PASSWORD</label></td>
<td><input type="password" name="pwd"></td>
</tr>
<tr>
<td><input type="submit" value="LOGIN"></td>
</tr>
</table>
</form>
</center>
</body>
</html>
```

Catalogue.html

```
<html>
<body bgcolor="orange" >
<h1 align="center">CATALOGUE PAGE</h1>
<center>
<table border="1">
<tr>
<th> COVER IMAGE</th>
```

```
<th> AUTHOR NAME</th>
<th> TEXT BOOK-NAME</th>
<th> PRICE</th>
<th> ADD TO CART</th>
</tr>
<tr>
<td><imgsrc="c#.jpg" height="105" width="150"></td>
<td>Harsh Bhasin</td>
<td> Programming in C#</td>
<td>200/</td>
<td><input type="button" value="ADD TO CART"</td>
<tr>
<td><imgsrc="cn.jpg" height="105" width="150"></td>

<td>Andrew S Tanenbaum</td>
<td>Computer Networks</td>
<td>400/</td>
<td><input type="button" value="ADD TO CART"</td>
<tr>
<td><imgsrc="wt.jpg" height="105" width="150"></td>
<td>Chris Bates</td>
<td>Web Programming, Building Internet Applications</td>
<td>599/</td>
<td><input type="button" value="ADD TO CART"</td>
<tr>
<td><imgsrc="wt1.jpg" height="105" width="150"></td>
<td>Kogent</td>
<td>Web Technologies, Black book</td>
<td>499/</td>
<td><input type="button" value="ADD TO CART"</td>
</table>
</center>
</bdoy>
</html>
```

display.html

```
<html>
<head>
<title>display</title>
</head>
<body bgcolor="blue" text="white">
<h2 align="center">DISPLAY</h2>
</body>
</html>
```

3 c)

Design a web page using CSS which includes the following:
i.Use different font and text styles
ii. Set a background image for both the page and single element on the page.
iii. Define styles for links

iv. Working with layers
v. Adding a Customized cursor

1) Use different font, styles:
In the style definition you define how each selector should work (font, color etc.). Then, in the body
of your pages, you refer to these selectors to activate the styles.
For example:

```
<HTML>
<HEAD>
<style type="text/css">
B.headline {color:red, font-size:22px, font-family:arial, text-decoration:underline}
</style>
</HEAD>
<BODY>
<b>This is normal bold</b><br>
```

Selector {cursor:value}
For example:

```
<html>
<head>
<style type="text/css">
.xlink {cursor:crosshair}
.hlink{cursor:help}
</style>
</head>
<body>
<b>
<a href="mypage.htm" class="xlink">CROSS LINK</a>
<br>
<a href="mypage.htm" class="hlink">HELP LINK</a>
</b>
</body>
</html>
<b class="headline">This is headline style bold</b>
</BODY>
</HTML>
```

2) Set a background image for both the page and single elements on the page. You can define the
background image for the page like this:
BODY {background-image:url(myimage.gif),}
3) Control the repetition of the image with the background-repeat property. As background-repeat:
repeat
Tiles the image until the entire page is filled, just like an ordinary background image in plain HTML.
4) Define styles for links as
A:link
A:visited
A:active

A:hover
Example:
<style type="text/css">
A:link {text-decoration: none}
A:visited {text-decoration: none}
A:active {text-decoration: none}
A:hover {text-decoration: underline, color: red,}
</style>
5) Work with layers:
For example:
LAYER 1 ON TOP:
LAYER 1 LAYER 2
LAYER 2 ON TOP:
LAYER 1 LAYER 2
6) Add a customized cursor:
Selector {cursor:value}
For example:
<html>
<head>
<style type="text/css">
.xlink {cursor:crosshair}
.hlink{cursor:help}
</style>
</head>
<body>

CROSS LINK

HELP LINK

</body>
</html>

Description

- **CSS** stands for **C**ascading **S**tyle **S**heets
- Styles define **how to display** HTML elements
- Styles are normally stored in **Style Sheets**
- Styles were added to HTML 4.0 **to solve a problem**
- **External Style Sheets** can save you a lot of work
- External Style Sheets are stored in **CSS files**
- Multiple style definitions will **cascade** into one

HTML tags were originally designed to define the content of a document.
As the two major browsers - Netscape and Internet Explorer - continued to add new HTML tags

and attributes (like the <font> tag and the color attribute) to the original HTML specification, it became more and more difficult to create Web sites where the content of HTML documents was clearly separated from the document's presentation layout.
To solve this problem, the World Wide Web Consortium (W3C) - the non profit, standard setting consortium, responsible for standardizing HTML - created STYLES in addition to HTML 4.0.
1) Use different font, styles:
Fontstyle.html

```
<html>
<head>
<link rel="stylesheet" href="font.css">
</head>
<body>
<p class="center">My text with font-arial</p>
<p class="right">My text with font-calibre</p>
<p class="left">My text with font-cooper</p>
</body>
</html>
```

font.css

```
p.center
{
color:pink;
text-align:center;
letter-spacing:3;
font-family:arial;
font-variant:small;
font-weight:bold;

}
p.right
{
color:orange;
text-align:right;
letter-spacing:10;
font-family:calibre;
font-variant:uppercase;
font-weight:bold;
}
p.left
{
color:green;
text-align:left;
letter-spacing:15;
font-family:cooper;
font-variant:uppercase;
font-weight:bold;
}
```

2) Set a background image
background.html

```
<html>
```

```
<head>
<style type="text/css">
body
{
background-image:url(wt.jpg);
background-repeat:repeat;
}
</style>
</head>
<body>
<h1 align="center">WebTechnology</h1>
</body>
</html>
```

3. Control the repetition of the image with the background-repeat property.

background-image.html

```
<html>
<head>
<style type="text/css">
body
{
background-image:url(wt.jpg);
background-color:pink;
background-repeat:no-repeat;
background-position:center;

background-attachment:fixed;
}
h1
{
background-image:url(wt1.jpg);
background-position:right;
color:red;
background-repeat:no-repeat;
}
</style>
</head>
<body>
<h1 align="center">WEB TECHNOLOGY<br></h1>
</body>
</html>
```

4) Define styles for links

Link.html

```
<html>
<head>
<style type="text/css">
a:link
{
color:red;
text-decoration:overline;
font-family:arial;
```

```
font-size:spot;
}
a:visited
{
color:yellow;
text-decoration:line-through;
font-family:arial;
font-size:spot;
}
a:hover
{
color:blue;
text-decoration:overline;
font-family:arial;
font-size:spot;
}
a:active
{
color:green;
text-decoration:overline;
font-family:arial;
font-size:spot;
}

</style>
</head>
<center>
<body bgcolor="pink">
<a href="wt.jpg" tagrget="_blank";>Link</a><br><br>
<a href="wt1.jpg" tagrget="_blank";>Link1</a><br><br>
</body>
</center>
</html>
```

5) Work with layers

layers.html

```
<html>
<body bgcolor="white" align="left">
<h1 style="background-color:blue; position:relative; top:0; left:0; z-index:2; height:100;">
This is
heading-1.</h1>
<h2 style="background-color:green; position:relative; top:30; left:50; z-index:3;
height:100;"> This
is heading-2.</h2>
<h3 style="background-color:red; position:relative; top:55 left:100; z-index:1; height:100;">
This is
heading-3.</h3>
</body>
</html>
```

6) Add a customized cursor

cursor.html

```
<html>
<body>
<p style="cursor:not-allowed;">NOT-ALLOWED CURSOR</p>
<p style="cursor:progress;">PROGRESS CURSOR</p>
<p style="cursor:wait;">WAIT CURSOR</p>
<p style="cursor:zoom-in;">ZOOM-IN CURSOR</p>
<p style="cursor:zoom-out;">ZOOM-OUT CURSOR</p>
<p style="cursor:no-drop;">NO-DROP CURSOR</p>
<p style="cursor:move;">MOVE CURSOR</p>
<p style="cursor:default;">DEFAULT CURSOR</p>
<p style="cursor:scroll;">SCROLL CURSOR</p>
<p style="cursor:crosshair;">CROSSHAIR CURSOR</p>
<p style="cursor:help;">HELP CURSOR</p>
<body>
</html>
```

Ex: 4

HTML Responsive Web Design

HTML Responsive Web Design is a smart way of designing web pages to look great on any device. It's all about making HTML elements adjust themselves, like resizing or hiding, based on the device's screen size. This ensures that no matter if someone's using a phone, tablet, or computer, the web pages will always look good.

Examples of HTML Responsive Web Design

- HTML Viewport meta tag for Responsive Web Design
- Responsive Images
- Responsive Texts
- CSS media Queries
- Responsive Layouts

1. HTML Viewport meta tag for Responsive Web Design

The HTML viewport is the visible area of the screen that users see. It changes depending on the device being used. With this approach, we set the width of web pages to match the available screen width, making it 100%. This helps content adapt and look good on any device, ensuring a responsive layout.

Syntax

```
<meta name="viewport" content= "width=device-width, initial-scale=1.0">
```

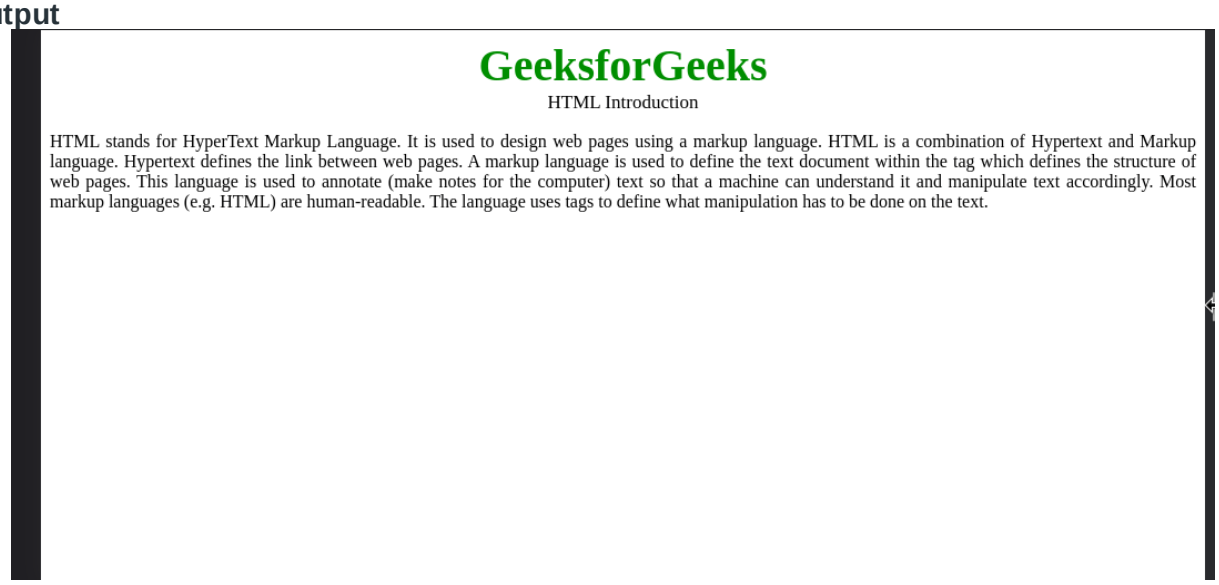
Example: Use of the HTML viewport meta tag for responsive pages.

HTML

```
<!DOCTYPE html>
<html>

<head>
    <title>GeeksforGeeks</title>
    <meta charset="utf-8"
          name="viewport"
          content="width=device-width,
                   initial-scale=1.0" />
    <style>
        .gfg {
            font-size: 40px;
            font-weight: bold;
            color: green;
            text-align: center;
        }

        .geeks {
            font-size: 17px;
            text-align: center;
        }

        p {
            text-align: justify;
        }
    </style>
```

```
</head>

<body>
   <div class="gfg">GeeksforGeeks</div>
   <div class="geeks">HTML Introduction</div>

   <p>
      HTML stands for HyperText Markup Language. It is
      used to design web pages using a markup
      language. HTML is a combination of Hypertext and
      Markup language. Hypertext defines the link
      between web pages. A markup language is used to
      define the text document within the tag which
      defines the structure of web pages. This
      language is used to annotate (make notes for the
      computer) text so that a machine can understand
      it and manipulate text accordingly. Most markup
      languages (e.g. HTML) are human-readable. The
      language uses tags to define what manipulation
      has to be done on the text.
   </p>
</body>

</html>
```

Output



2. Responsive Images

Responsive images play a key role in responsive websites. These are images that can adjust their size, getting bigger or smaller, based on the width of the browser. By being responsive, images enhance user experience across different devices with varying screen sizes. The following are the techniques to use the responsive images:

2.1 Using width Property

The image can be responsive & scale up & down with the help of CSS width property by setting its value as 100%.

Syntax

```
<img src="..." style="width:100%;">
```

Example: In this example, we will use the image width property to occupy 100% of the screen width.

HTML

```
<!DOCTYPE html>
<html>

<head>
    <meta name="viewport" content=
        "width=device-width, initial-scale=1.0" />
</head>

<body>
    <img class=".img-fluid" src=
"https://media.geeksforgeeks.org/wp-content/uploads/20220201191443/logo-200x32.png"
         style="width: 100%" />

    <h2>Responsive Images</h2>

    <p>
        Responsive images are just a part of Responsive
        websites. Images that can change their
        dimensions, scaling them up or down, according
        to the browser width are responsive images. The
        above image is responsive as it is adjusting
        itself according to the width of the browser.
    </p>
</body>

</html>
```

Output

Responsive Images

Responsive images are just a part of Responsive websites. Images that can change their dimensions, scaling them up or down, according to the browser width are responsive images. The above image is responsive as it is adjusting itself according to the width of the browser.

2.2 Using the max-width Property

The max-width property sets the maximum width of an element. It restricts the element's width from exceeding a certain value. If the content is larger than this maximum width, it wraps to the next line. However, if the content is smaller, the property has no impact. Also, if the content is larger than its original size, it won't scale up beyond the specified max-width.

Syntax:

```
<img src="..." style="max-width:100%; height:auto;">
```

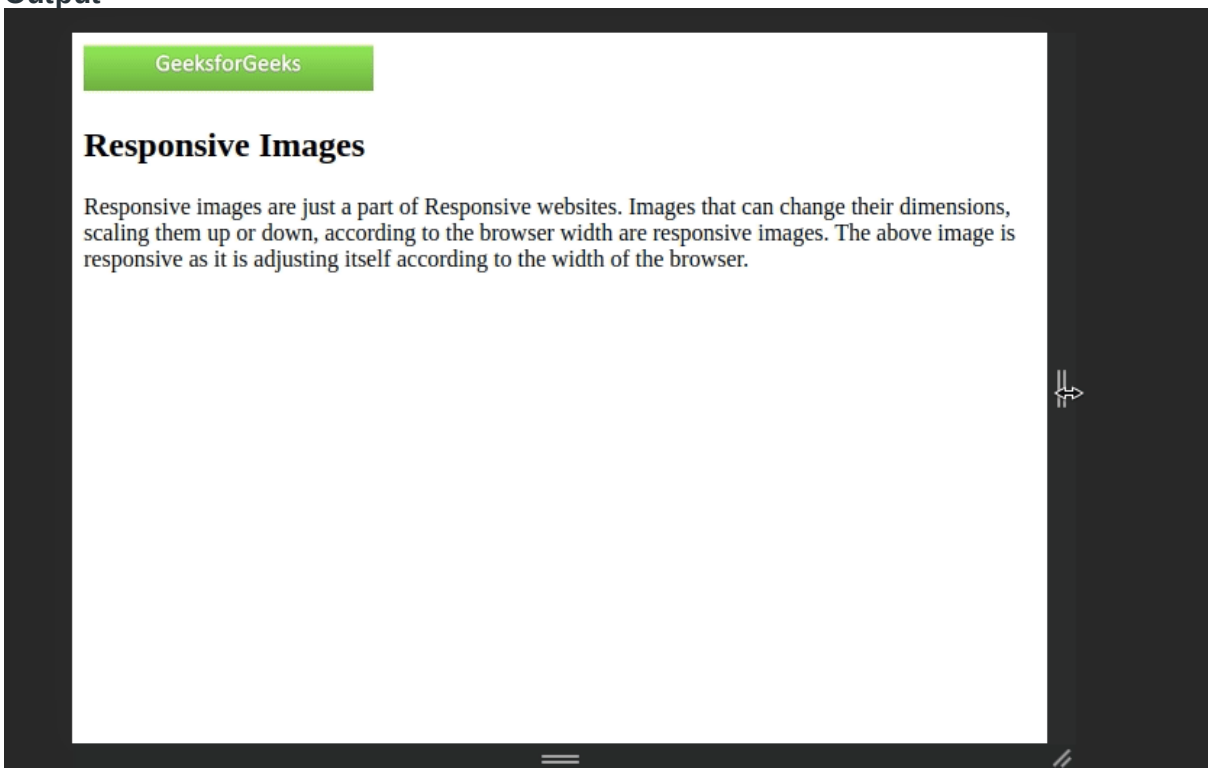
Example: Implementation of the max-width Property for making the Responsive Images

HTML

```
<!DOCTYPE html>
```

```
<html>

<head>
   <meta name="viewport"
         content="width=device-width,
                  initial-scale=1.0" />
</head>

<body>
   <img class=".img-fluid"
        src=
"https://media.geeksforgeeks.org/wp-content/uploads/20220201191443/logo-200x32.png"
        style="max-width:100%;
               height:auto;" />
   <h2>Responsive Images</h2>
   <p>
      Responsive images are just a part of Responsive
      websites. Images that can change their
      dimensions, scaling them up or down, according
      to the browser width are responsive images. The
      above image is responsive as it is adjusting
      itself according to the width of the browser.
   </p>
</body>

</html>
```

Output



2.3 Responsive Image for different Browser Width

To make images responsive, HTML provides the <picture> element. It gives web developers the flexibility to specify different image resources depending on the size of the browser window

Example

HTML

```
<!DOCTYPE html>
<html>

<head>
    <title>HTML Responsive Web Design</title>
</head>

<body style="text-align: center;">
    <h1 style="color: green;">GeeksforGeeks</h1>

    <h2>HTML picture Tag</h2>

    <picture>
        <source media="(min-width: 700px)"
                srcset=
"https://media.geeksforgeeks.org/wp-content/uploads/20190825000042/geeks-221.png">

        <source media="(min-width: 450px)"
                srcset=
"https://media.geeksforgeeks.org/wp-content/uploads/20190802021607/geeks14.png">

        <img src=
"https://media.geeksforgeeks.org/wp-content/uploads/20190808102629/geeks15.png"
             alt="GFG">
    </picture>
</body>

</html>
```

Output

3. Responsive Texts

In this method, we set font sizes using %, vw, vh, etc. This ensures that text sizes are responsive, adjusting automatically until reaching a certain limit. Once the limit is reached, the content is justified to fit within the available width.

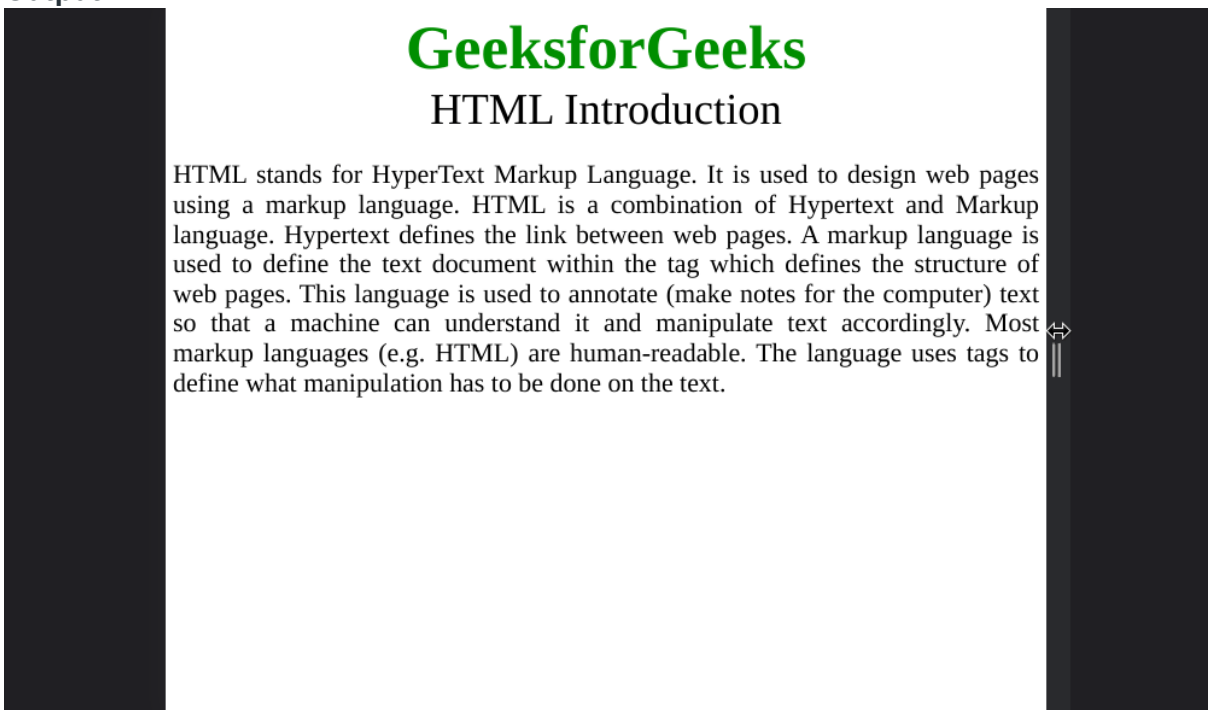
Example: This example demonstrates Responsive Web Design by making Responsive Texts.

HTML

```
<!DOCTYPE html>
<html>

<head>
    <style>
        body {
            max-width: 100%;
        }

        .gfg {
            font-size: 7vw;
            font-weight: bold;
            color: green;
            text-align: center;
        }

        .geeks {
            font-size: 5vw;
            text-align: center;
        }

        p {
            font-size: 3vw;
            text-align: justify;
        }
    </style>
</head>

<body>
    <div class="gfg">GeeksforGeeks</div>
    <div class="geeks">HTML Introduction</div>

    <p>
        HTML stands for HyperText Markup Language. It is
        used to design web pages using a markup
        language. HTML is a combination of Hypertext and
        Markup language. Hypertext defines the link
        between web pages. A markup language is used to
        define the text document within the tag which
        defines the structure of web pages. This
        language is used to annotate (make notes for the
        computer) text so that a machine can understand
        it and manipulate text accordingly. Most markup
        languages (e.g. HTML) are human-readable. The
        language uses tags to define what manipulation
        has to be done on the text.
    </p>
</body>
```

```
</html>
```

Output:



4. CSS media Queries

The Media query in CSS is essential for crafting responsive web designs. It ensures that web pages adapt to various screen sizes and device types. Breakpoints are set to define when the content starts to adjust or change layout based on the device's width.

Media queries can be used to check many things:

- width and height of the viewport
- width and height of the device
- Orientation
- Resolution

Syntax

```
@media not | only mediatype and (expression) {
  // Code content
}
```

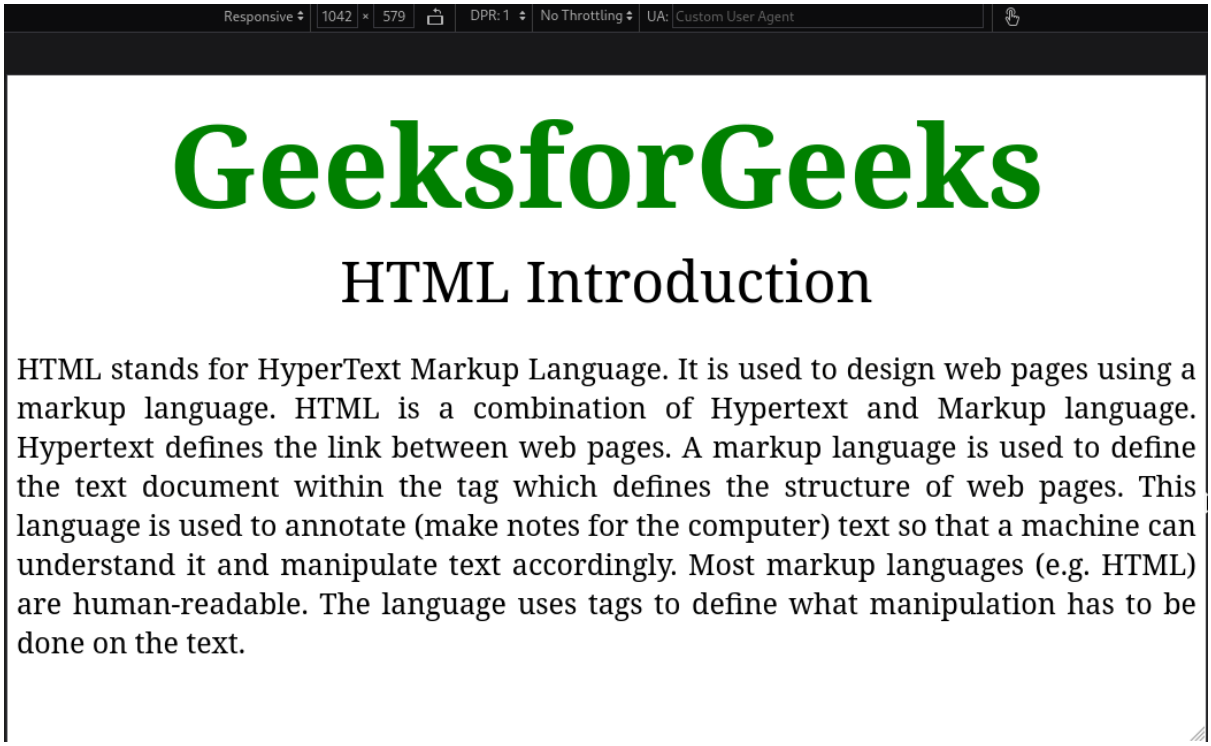
Example: In this example, we will use screen size in a media query to make the webpage responsive according to different breakpoints.

HTML

```
<!DOCTYPE html>
<html>

<head>
  <title>GeeksforGeeks</title>
  <style>
    .gfg {
      font-size: 100px;
      font-weight: bold;
      color: green;
      text-align: center;
    }

    .geeks {
      font-size: 50px;
      text-align: center;
```

```
        }

        p {
            font-size: 25px;
            text-align: justify;
        }

        /* styling for screen width less than 800 */
        @media screen and (max-width: 800px) {
            body {
                background-color: aqua;
            }

            .gfg {
                font-size: 50px;
            }

            .geeks {
                font-size: 25px;
            }

            p {
                font-size: 12px;
            }
        }
    </style>
</head>

<body>
    <div class="gfg">GeeksforGeeks</div>
    <div class="geeks">HTML Introduction</div>

    <p>
        HTML stands for HyperText Markup Language. It is
        used to design web pages using a markup
        language. HTML is a combination of Hypertext and
        Markup language. Hypertext defines the link
        between web pages. A markup language is used to
        define the text document within the tag which
        defines the structure of web pages. This
        language is used to annotate (make notes for the
        computer) text so that a machine can understand
        it and manipulate text accordingly. Most markup
        languages (e.g. HTML) are human-readable. The
        language uses tags to define what manipulation
        has to be done on the text.
    </p>
</body>

</html>
```

Output: Background color and font size transition for width less than 800px.



Note: Sometimes, this method doesn't show the correct output on Google Chrome.

5. Responsive Layouts

The responsive layout module of CSS includes the following properties.

5.1 Using flexbox property

In this approach, we will use CSS display property to make the page responsive. Display layouts like flexbox, inline, blocks, and grids can be used to make the design responsive. CSS flexbox property auto adjusts the content (no. of columns in a row) according to the screen width as shown in the output gif.

Syntax

```
.container{
    display:                                                    flexbox;
}
```

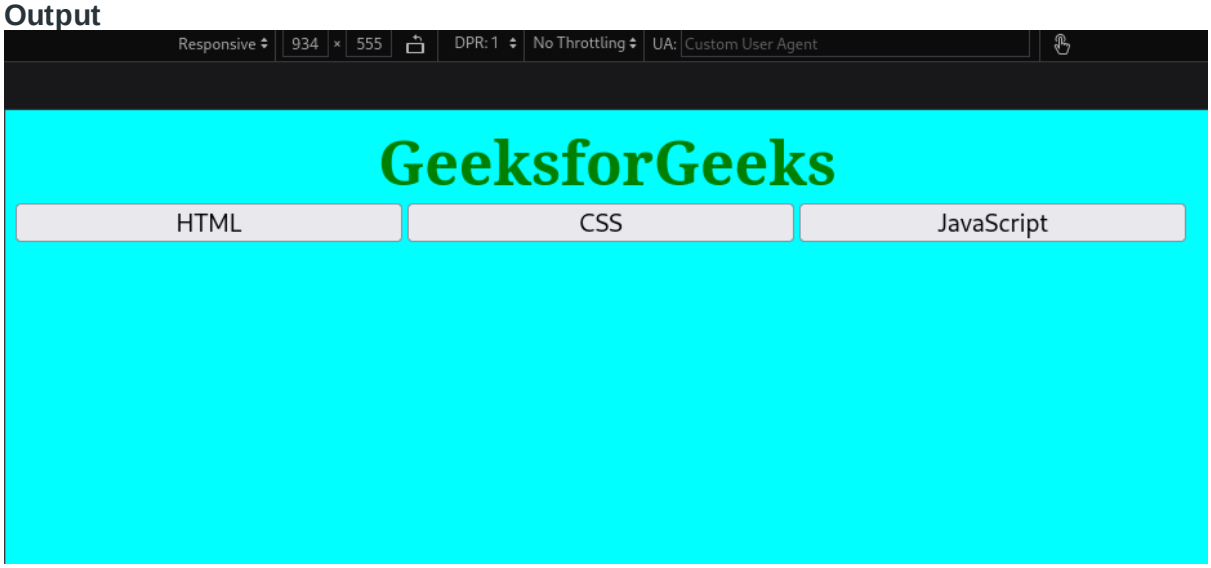
Example: In this example, we will use display type flexbox to show adjust items automatically according to the screen size.

HTML

```
<!DOCTYPE html>
<html>

<head>
    <title>GeeksforGeeks</title>
    <style>
        body {
            background-color: aqua;
        }
        .gfg {
            font-size: 5vw;
            font-weight: bold;
            color: green;
            text-align: center;
        }
        button {
            width: 300px;
            font-size: larger;
```

```
        }
        .container {
            display: flexbox;
        }
    </style>
</head>

<body>
    <div class="gfg">GeeksforGeeks</div>
    <div class="container">
        <button>HTML</button>
        <button>CSS</button>
        <button>JavaScript</button>
    </div>
</body>

</html>
```

Output



Note: Sometimes, this method doesn't show the correct output on Google Chrome.

5.2 Using CSS Grids

This approach uses a CSS display grid to create a 2D layout along with other grid options. It allows us to decide the number of columns we want to keep and instead of rearranging the columns like Flexbox, it adjusts the content within individual column elements.

Syntax

```
.container{
    display: grid;
    /* To define colums*/
    grid-template-columns: 1fr 1fr;
}
```

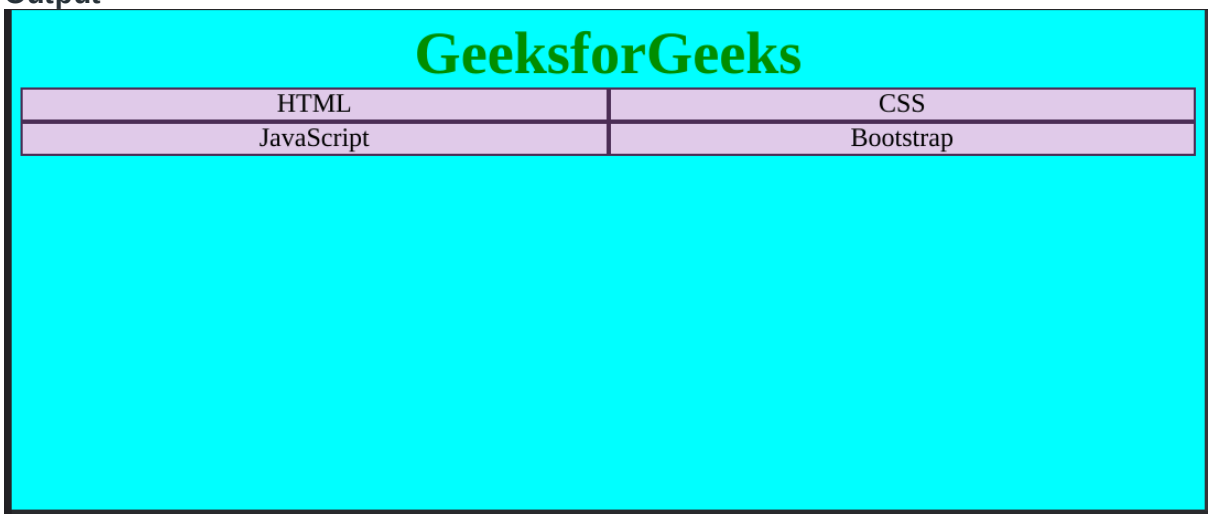
Example: In this example, CSS Grid layout is used to arrange contents in 2D form, i.e., rows and columns.

HTML

```
<!DOCTYPE html>
<html>

<head>
    <title>GeeksforGeeks</title>
    <style>
        body {
```

```
            background-color: aqua;
        }
        .gfg {
            font-size: 5vw;
            font-weight: bold;
            color: green;
            text-align: center;
        }

        .container {
            font-size: x-large;
            text-align: center;
            display: grid;
            grid-template-columns: 1fr 1fr;
        }
        .grid-item {
            background-color: rgb(220, 208, 232);
            border: 2px solid rgb(70, 54, 84);
        }
    </style>
</head>

<body>
    <div class="gfg">GeeksforGeeks</div>
    <div class="container">
        <div class="grid-item">HTML</div>
        <div class="grid-item">CSS</div>
        <div class="grid-item">JavaScript</div>
        <div class="grid-item">Bootstrap</div>
    </div>
</body>

</html>
```

Output



Note: Sometimes, this method doesn't show the correct output on Google Chrome.

5.3 Using CSS MultiColumn

It is similar to grids. CSS MulotiColumn allows developers to choose the properties like no. of columns, width, gap, etc. for each column. These values remain unchanged but the content inside the columns adjusts.

Syntax

```
.container{
   column-count:  3;                                    /* Number of columns*/
   column-gap:  20px;                                   /* Gap between columns*/
   column-width:  200px;                                /* Width of each column*/
   /*                  Other                  column                  properties*/
}
```

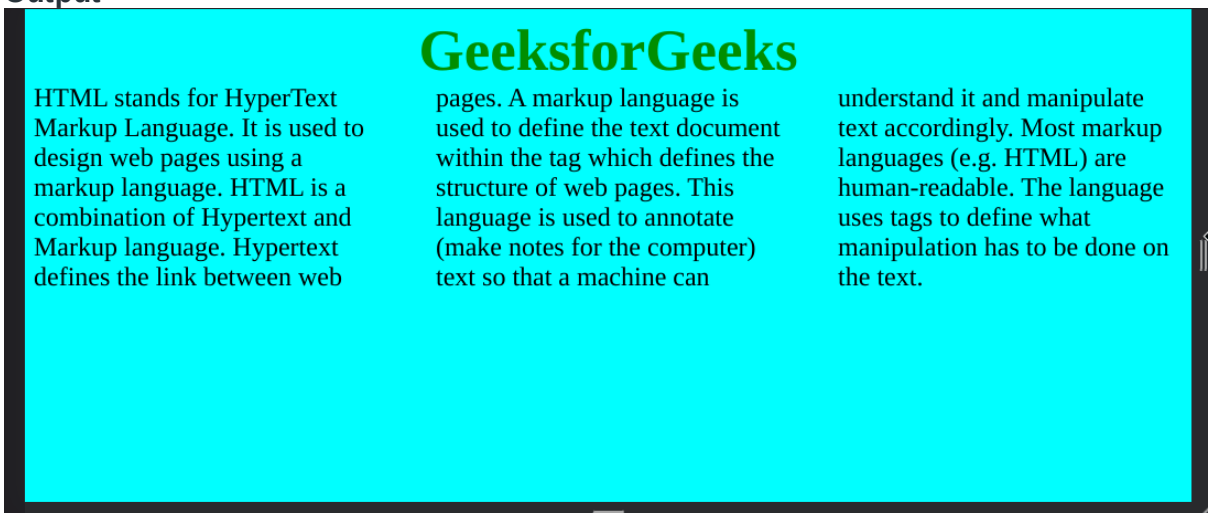
Example: This example uses CSS MultiColumn for content division into a specific number of columns.

HTML

```
<!DOCTYPE html>
<html>

<head>
   <title>GeeksforGeeks</title>
   <style>
      body {
         background-color: aqua;
      }
      .gfg {
         font-size: 5vw;
         font-weight: bold;
         color: green;
         text-align: center;
      }

      .container {
         font-size: x-large;
         text-align: left;
         column-count: 3;
         column-gap: 5%;
      }
   </style>
</head>

<body>
   <div class="gfg">GeeksforGeeks</div>
   <div class="container">
      <div>
         HTML stands for HyperText Markup Language.
         It is used to design web pages using a
         markup language. HTML is a combination of
         Hypertext and Markup language. Hypertext
         defines the link between web pages. A markup
         language is used to define the text document
         within the tag which defines the structure
         of web pages. This language is used to
         annotate (make notes for the computer) text
         so that a machine can understand it and
         manipulate text accordingly. Most markup
         languages (e.g. HTML) are human-readable.
         The language uses tags to define what
         manipulation has to be done on the text.
      </div>
   </div>
</body>
```

```
</html>
```

Output



HTML Responsive Web Design allows websites to adapt to various screen sizes, ensuring a seamless user experience across devices. By using HTML and responsive techniques, designers and developers can create websites that look great whether viewed on a desktop or a smartphone. This flexibility enhances accessibility and usability, making browsing a breeze for all users. In essence, HTML Responsive Web Design is crucial for ensuring that websites remain functional and visually appealing regardless of the device being used.

Ex: 5

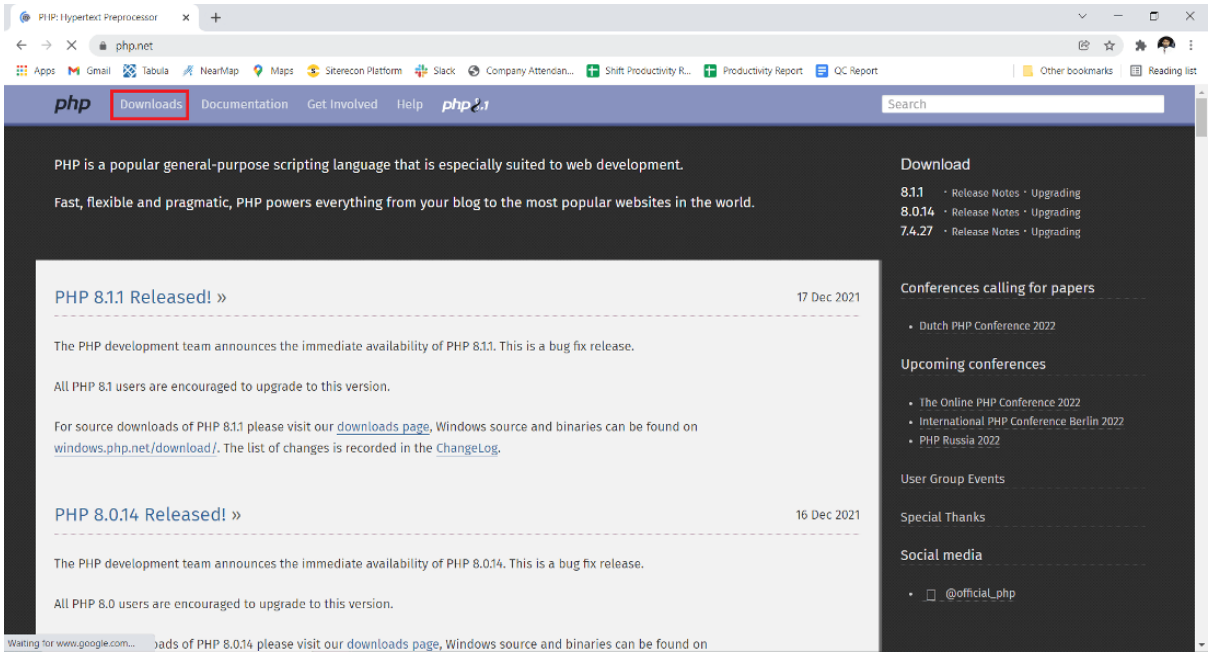
a) How to install PHP in windows 10

PHP is a general-purpose scripting language geared towards web development. PHP is open-source software (OSS). PHP is free to use and download. PHP stands for PHP Hypertext Preprocessor. PHP files have an extension called .php. PHP supports many databases MySQL, Oracle, etc. PHP is free to download and use. It was created by a Danish Canadian and designed by Rasmus Lerdorf in 1994. The common websites that are used are eCommerce, Social Platforms, Email Hosting, etc.

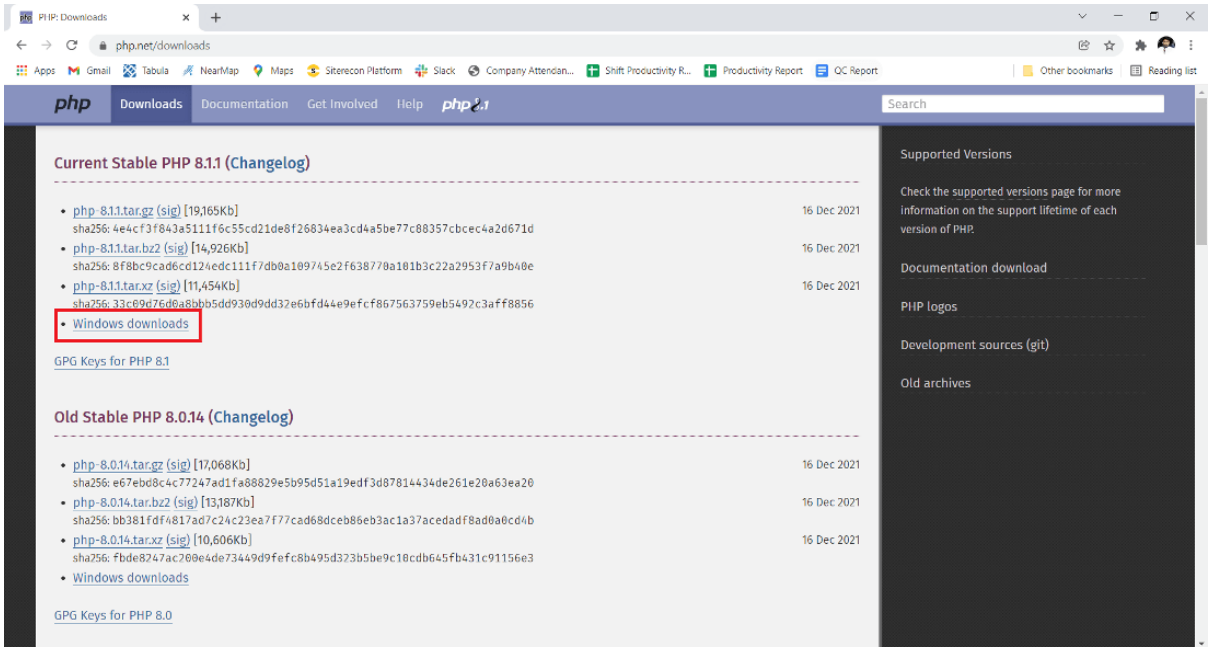
Installing PHP on Windows

Follow the below steps to install PHP on Windows:

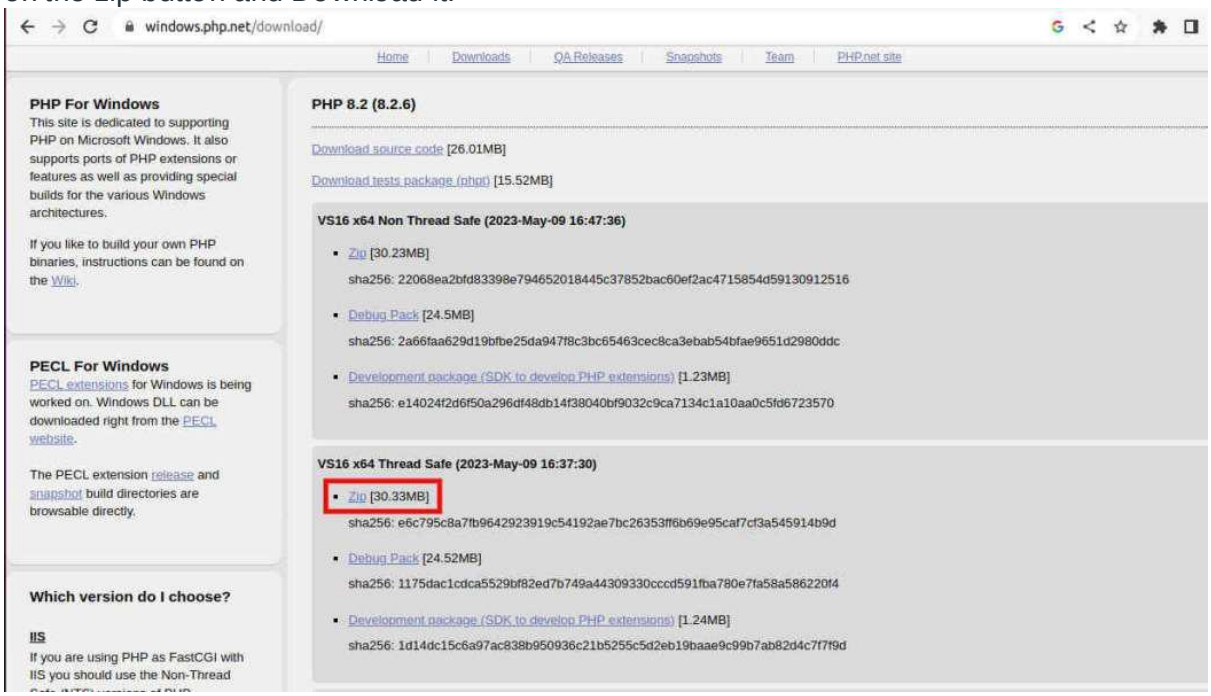
Step 1: Visit *https://www.php.net/downloads* website using any web browser and click on Downloads.



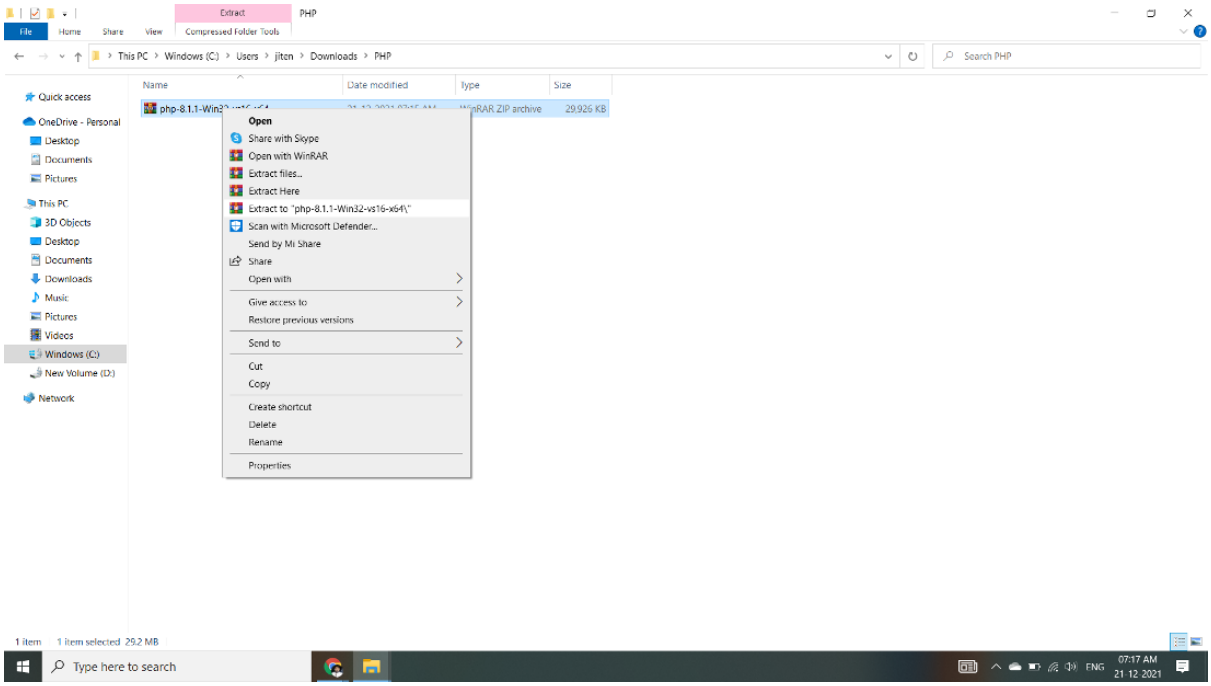
Step 2: Click on the Windows "**Downloads**" button.



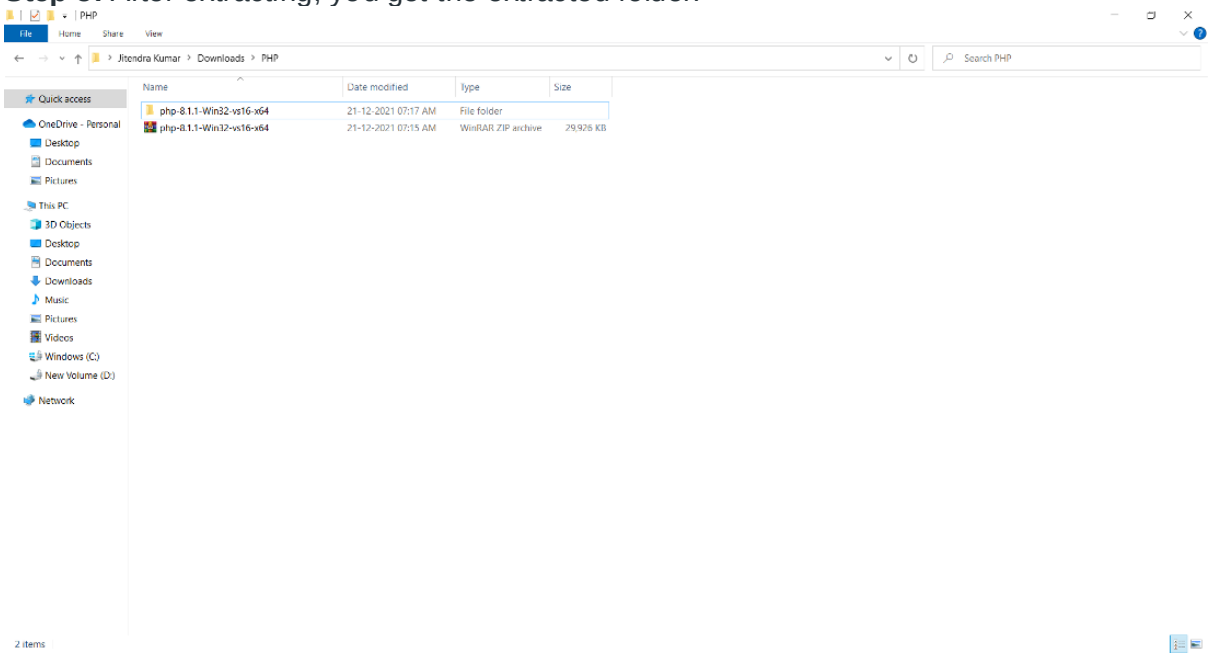
Step 3: The new webpage has different options, choose the Thread safe version, and click on the zip button and Download it.



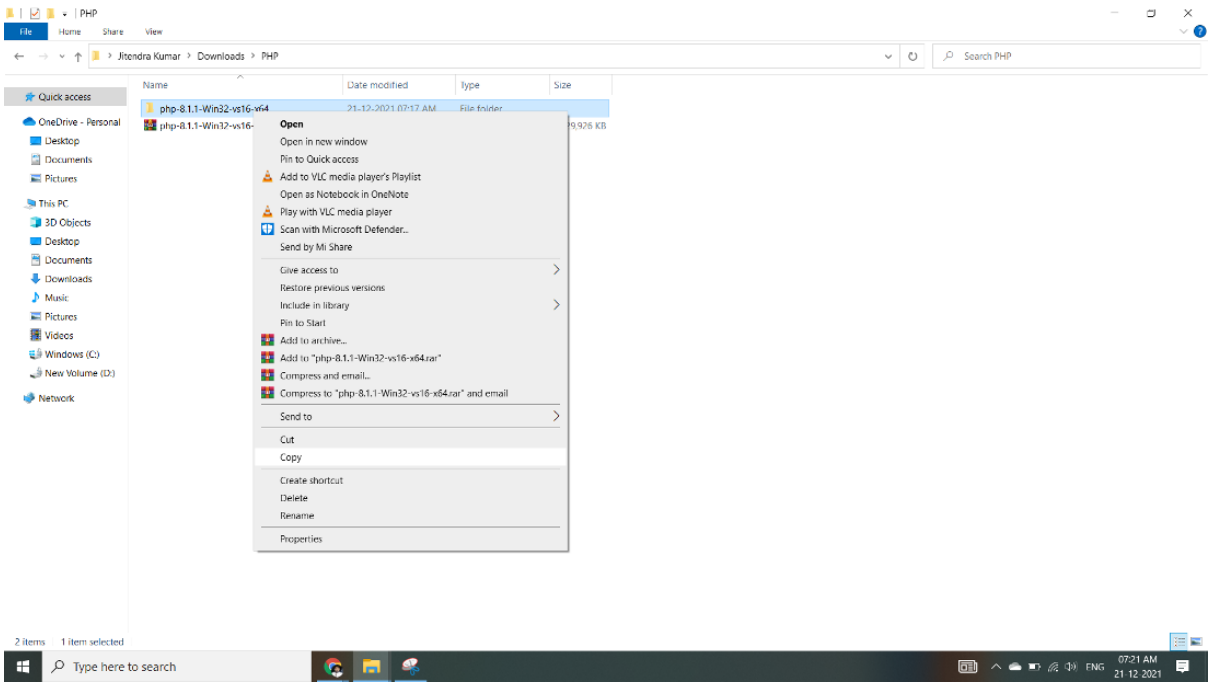
Step 4: Now check for the zip file in downloads in your system and extract it.



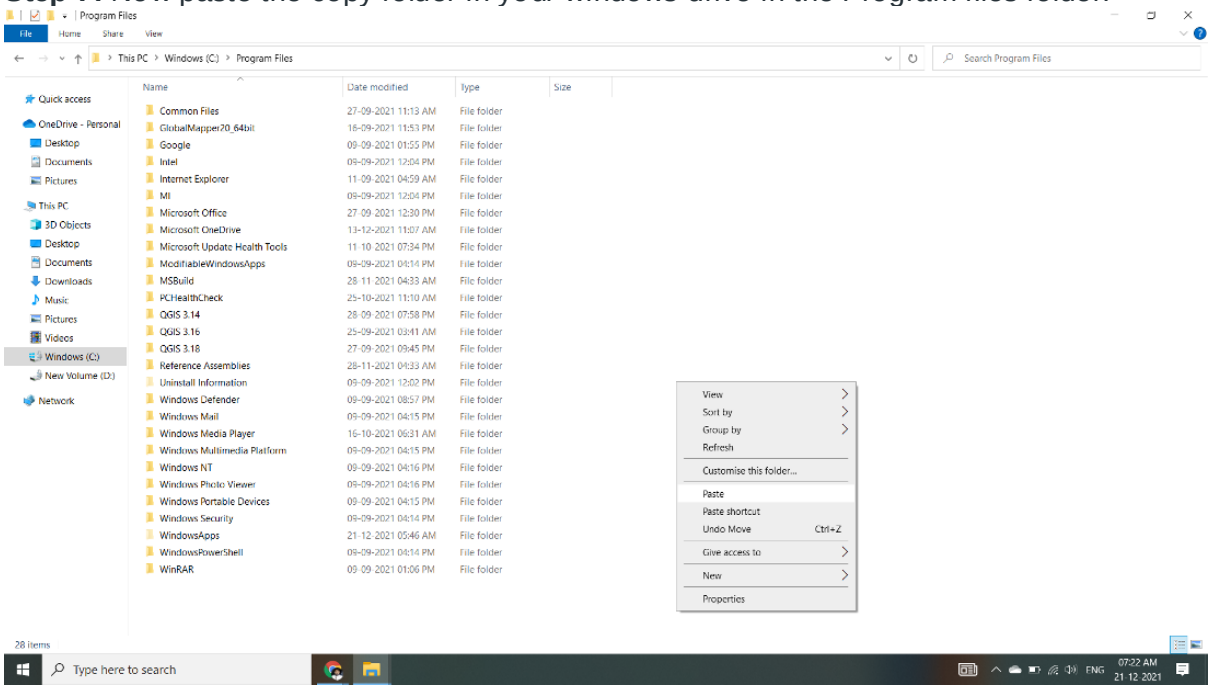
Step 5: After extracting, you get the extracted folder.



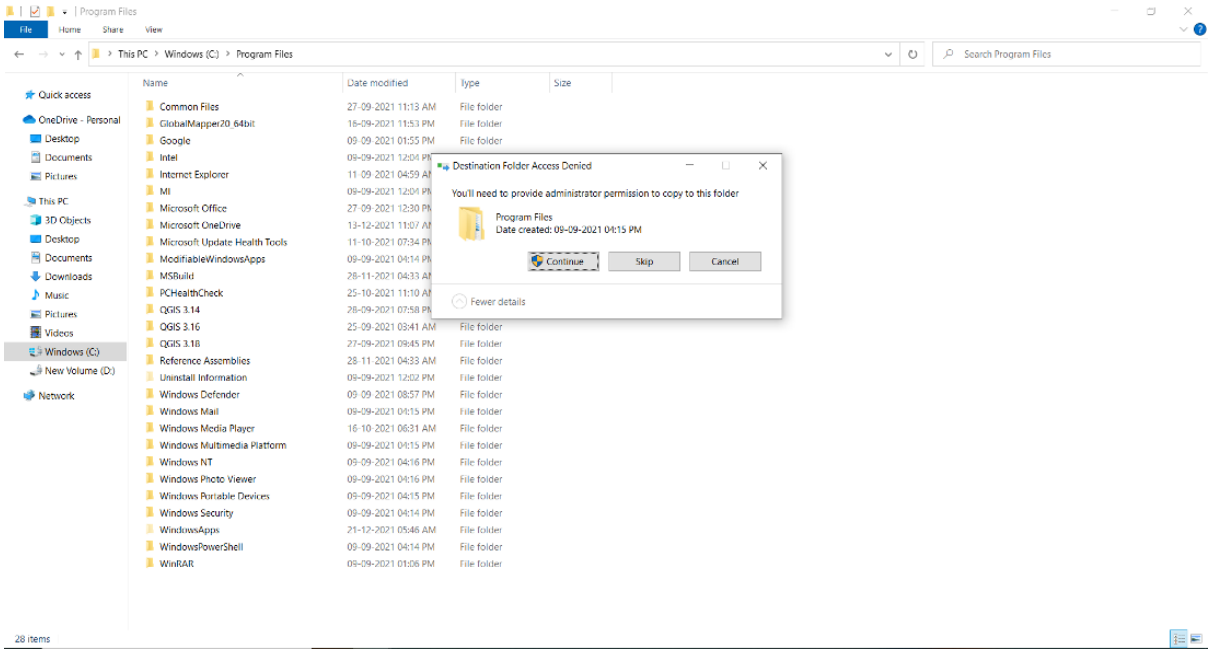
Step 6: Now copy the extracted folder.



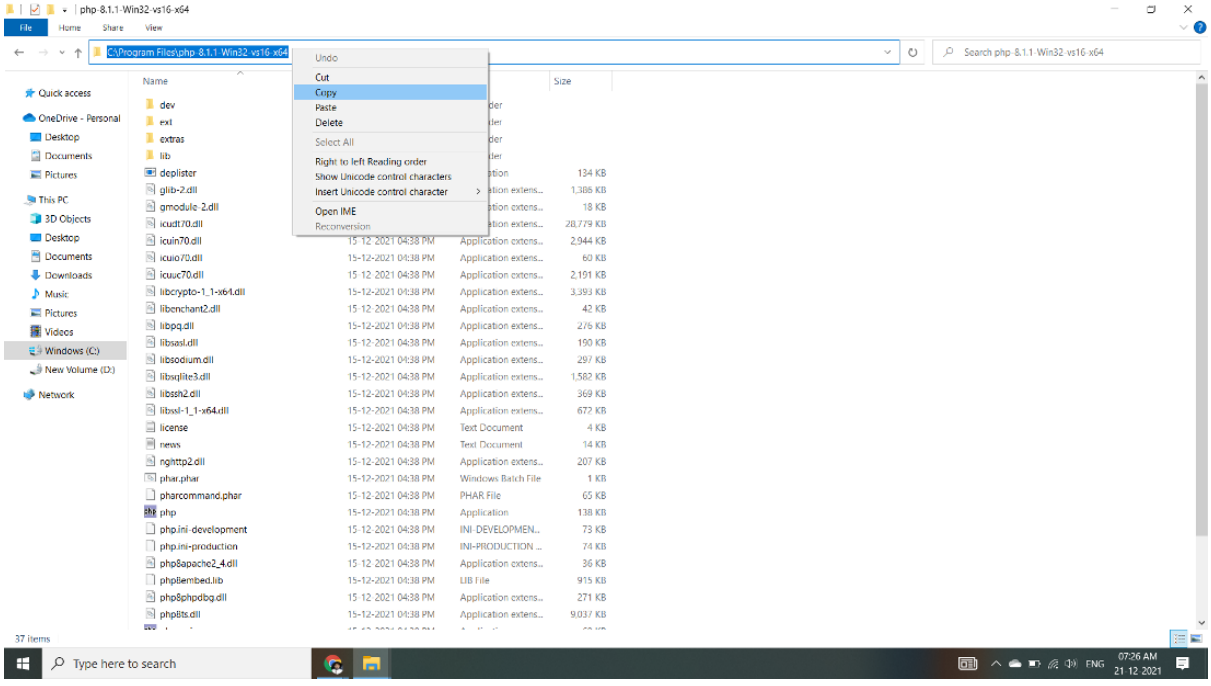
Step 7: Now paste the copy folder in your windows drive in the Program files folder.



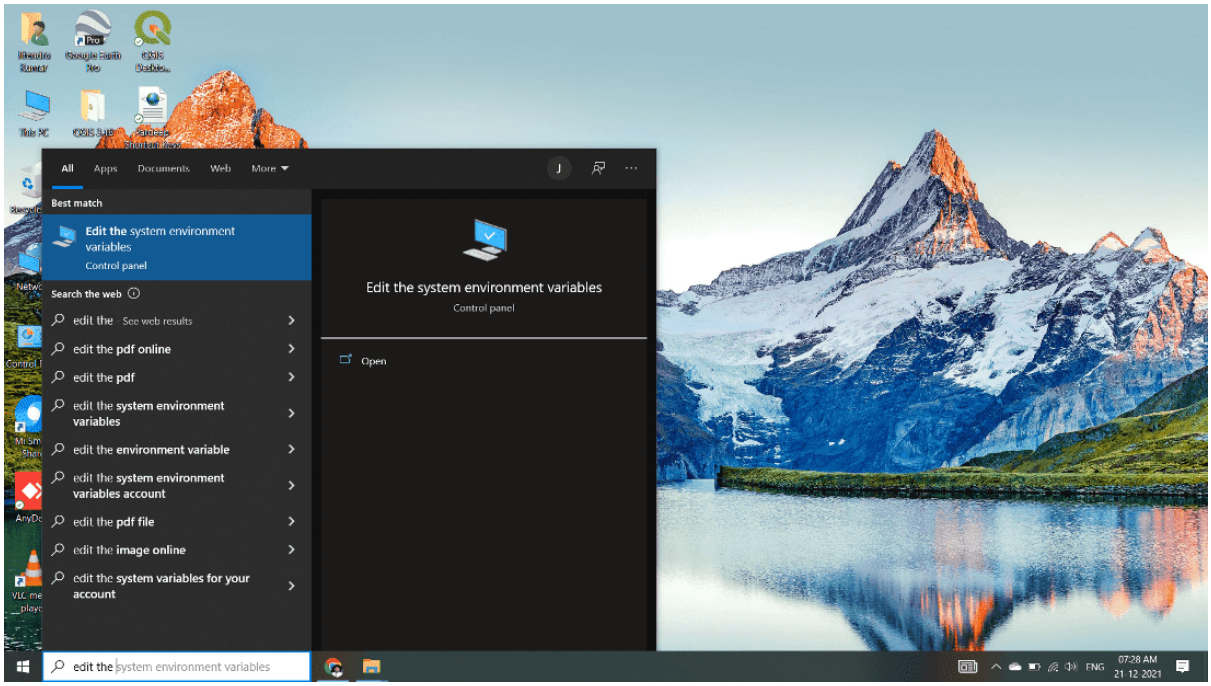
Step 8: Now the Permission Windows appears to paste the folder in program files then click on “Continue”.



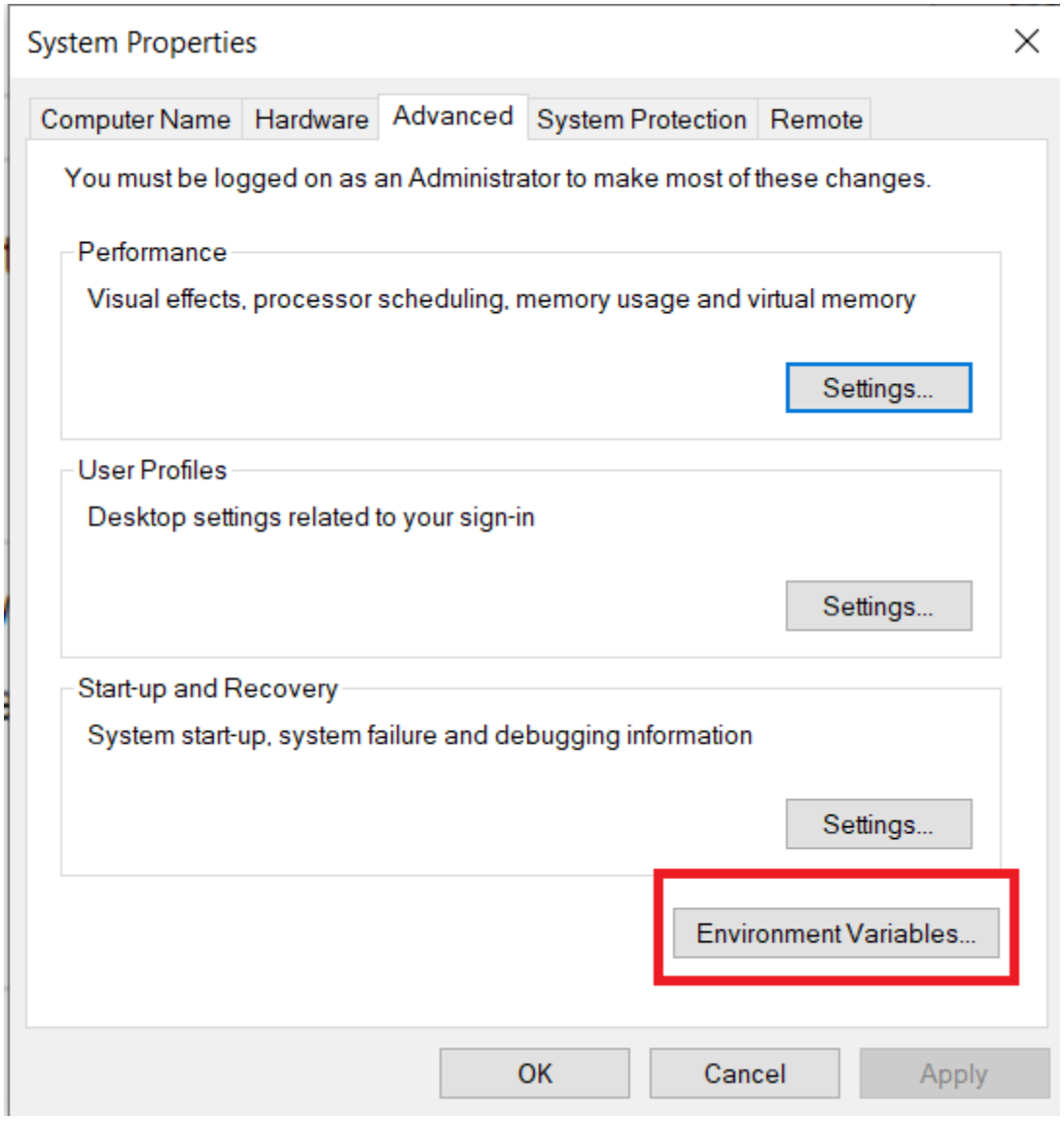
Step 9: Now after pasting the folder then copy the address of the folder in program files.



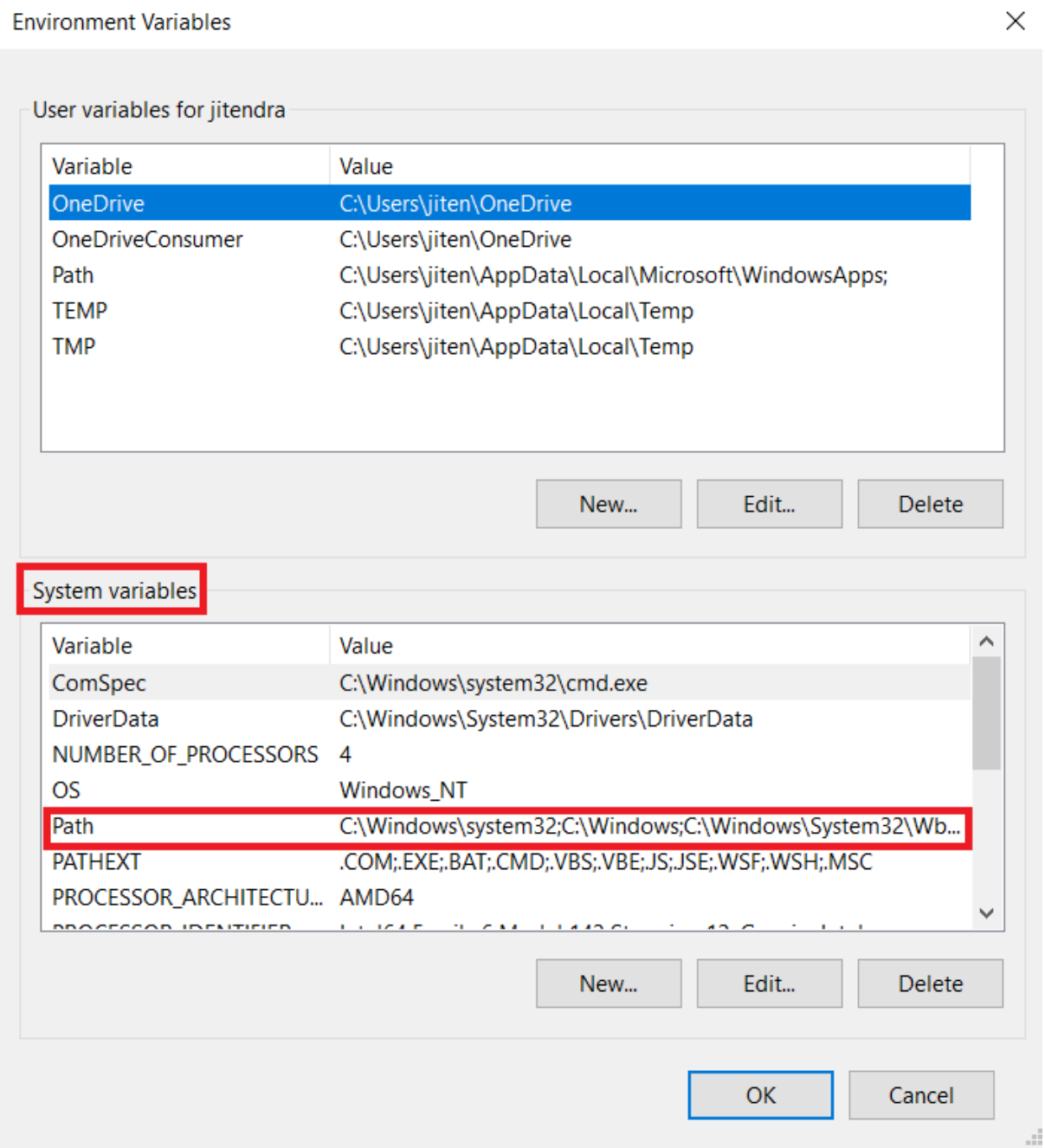
Step 10: Now click on Start Menu and search “Edit the system environment variables” and open it.



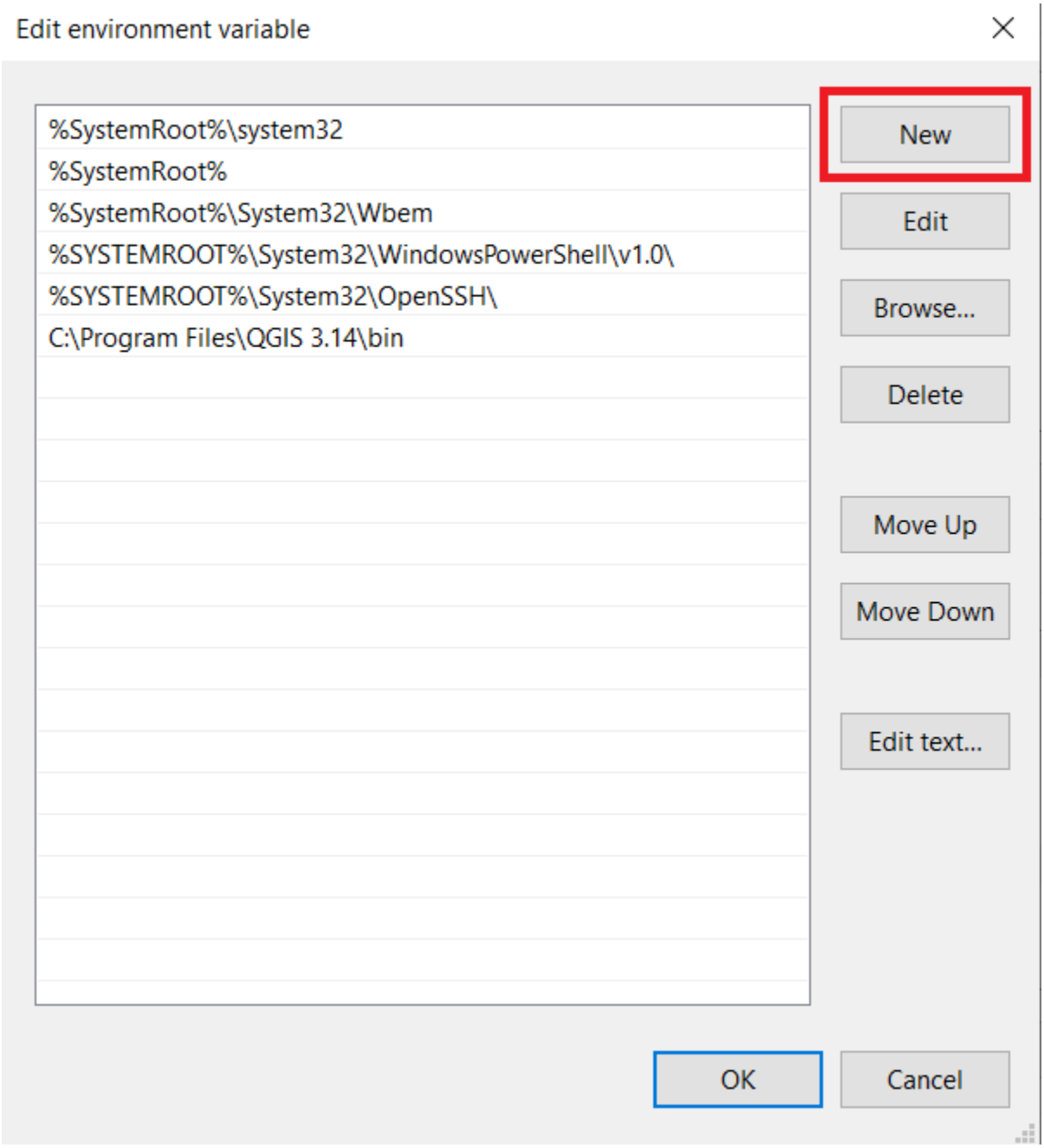
Step 11: After opening System, Variable New window appears, and click on “Environment Variables…”



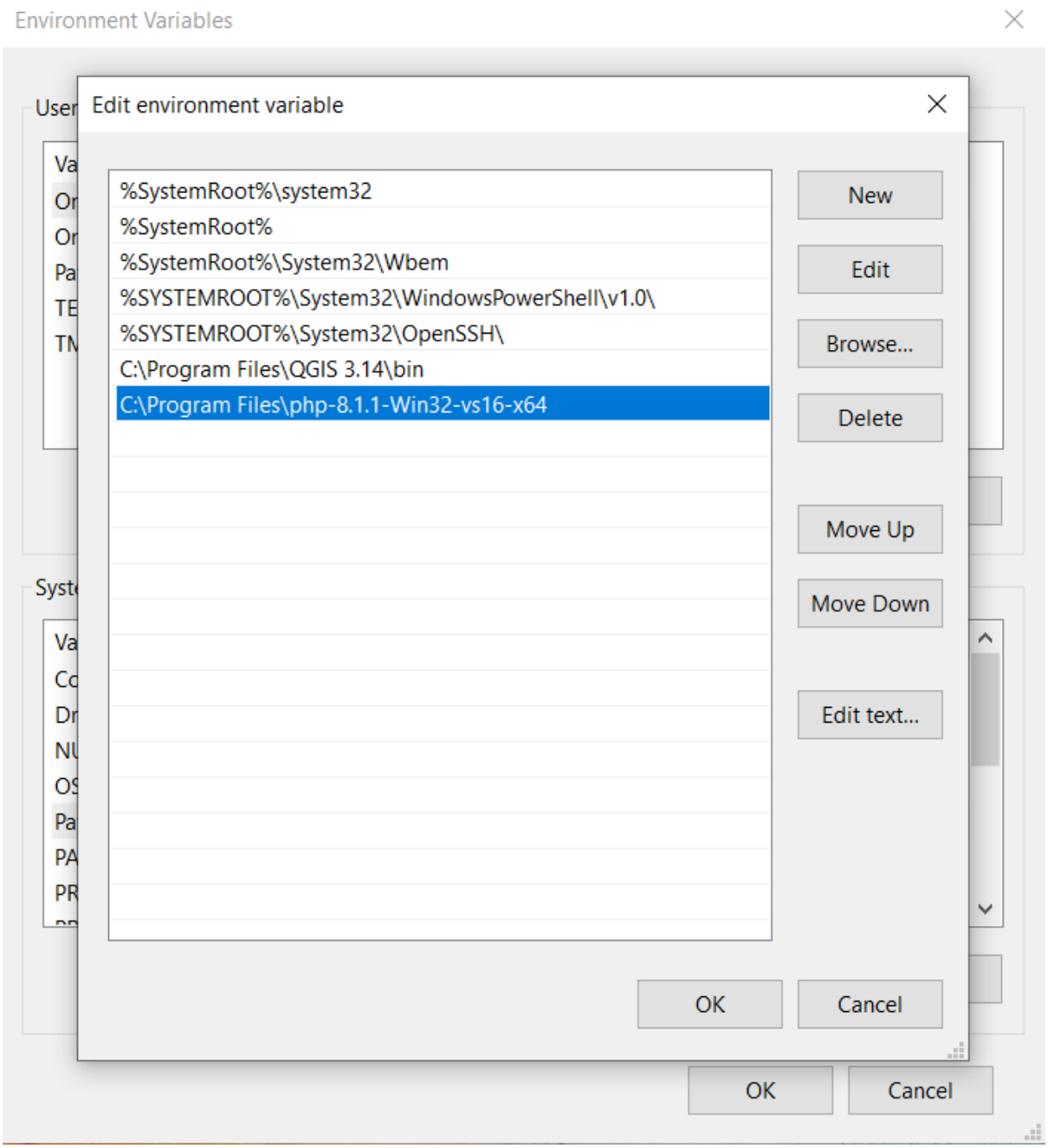
Step 12: Now go to the "System variables" *Path* option and double click on *Path*.



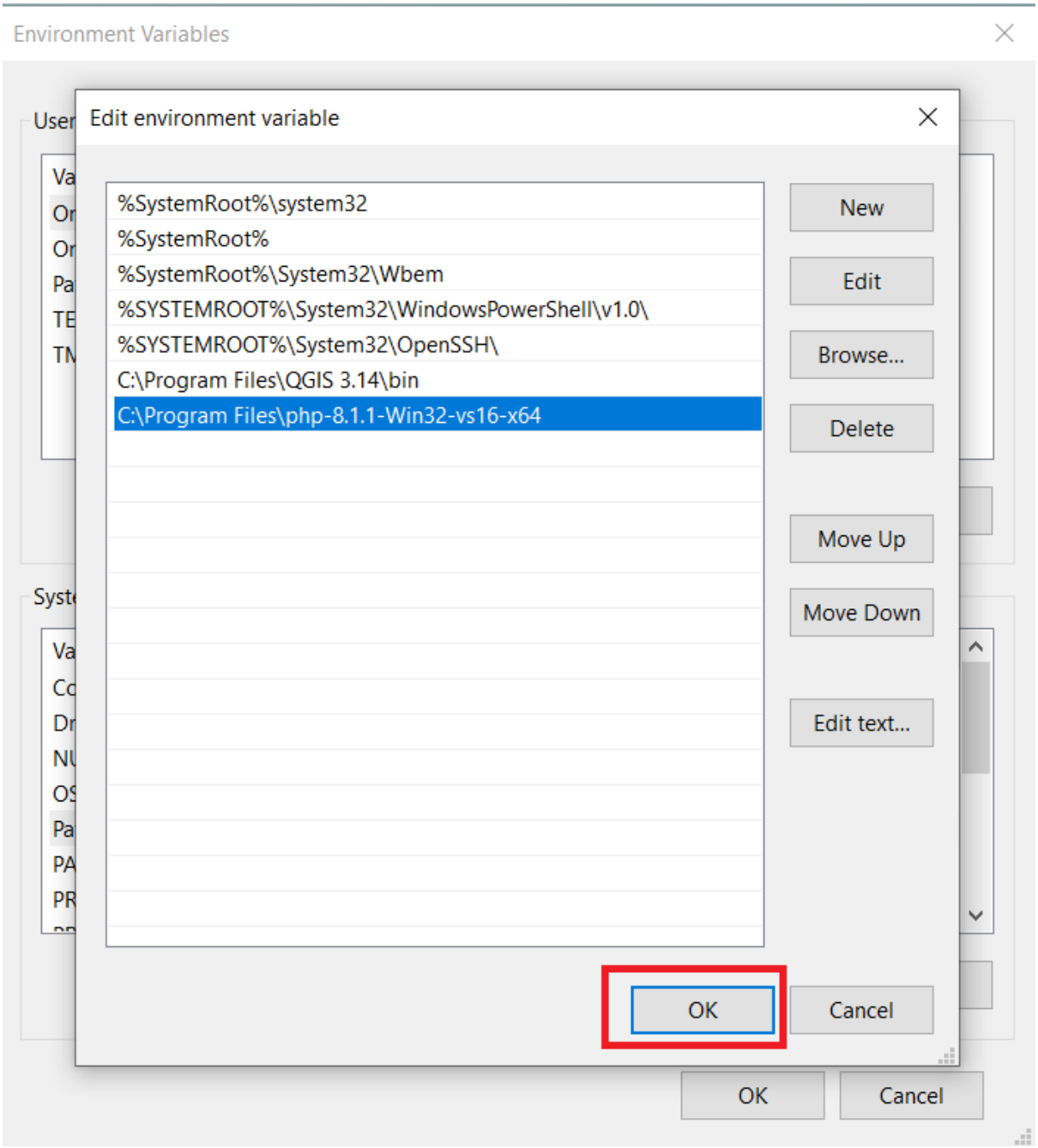
Step 13: Next screen will open and click on the “New” button.



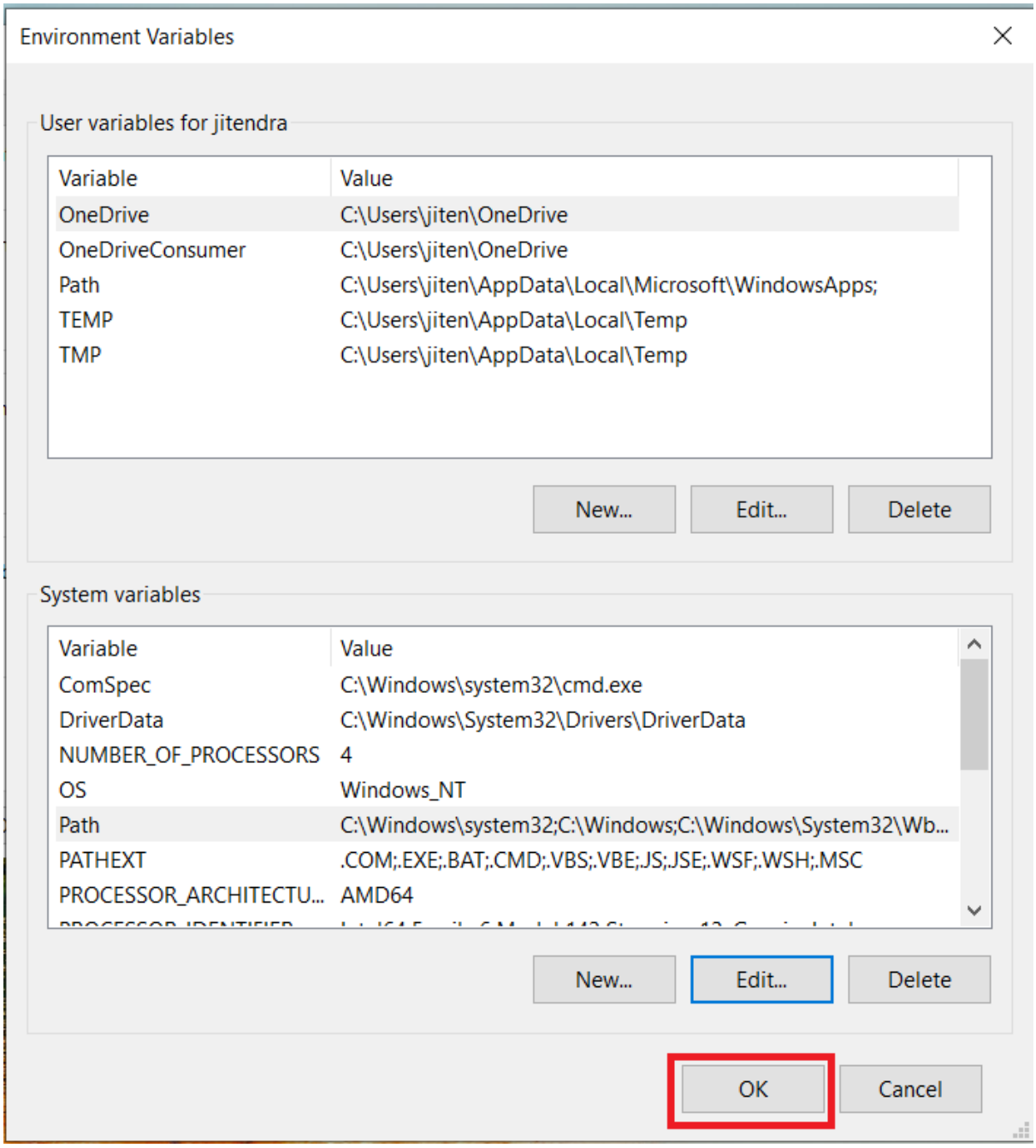
Step 14: After New Paste the address we copy from program files to new and click on *Enter* button.



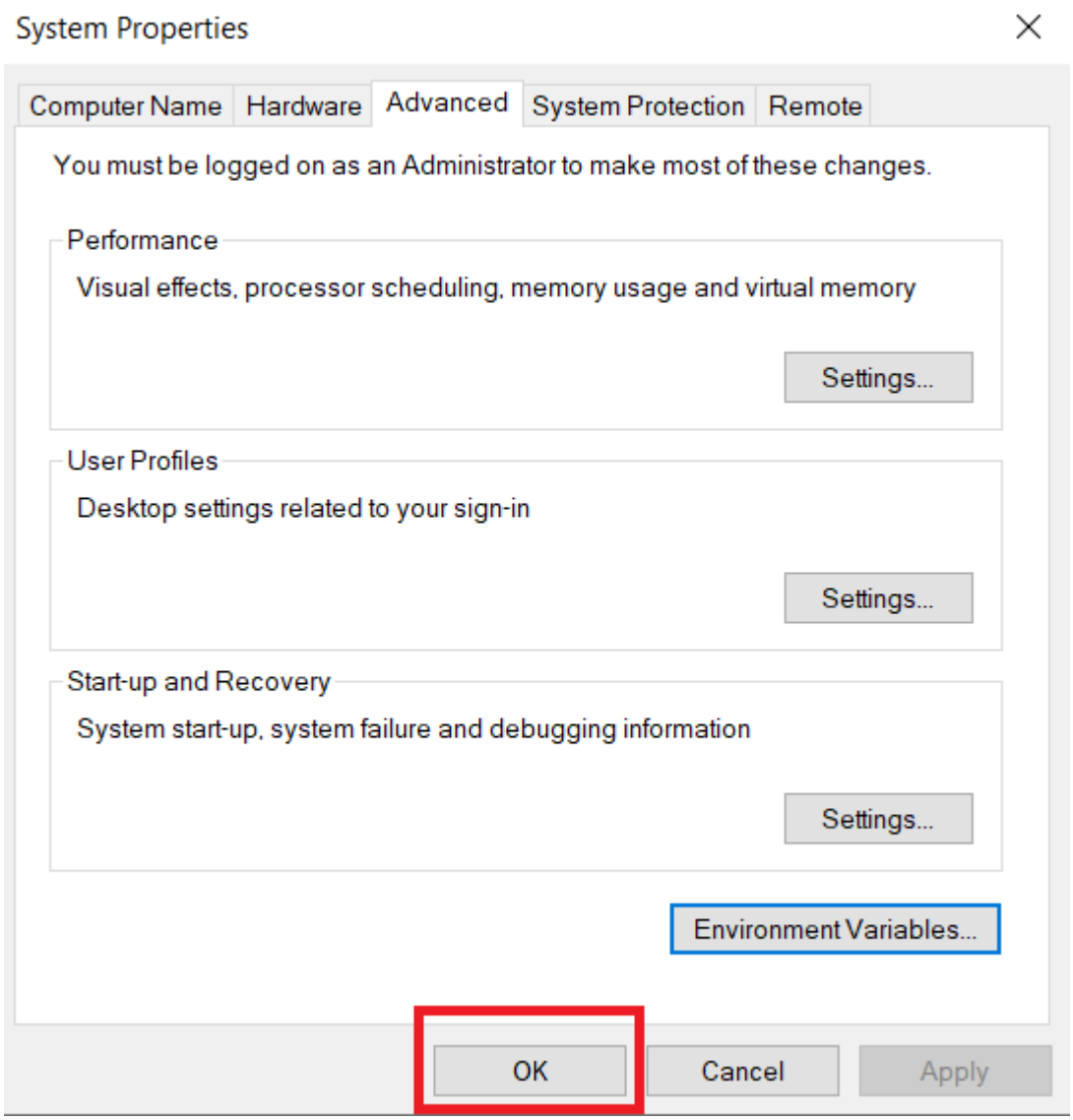
Step 15: Now Click on the **OK** button.



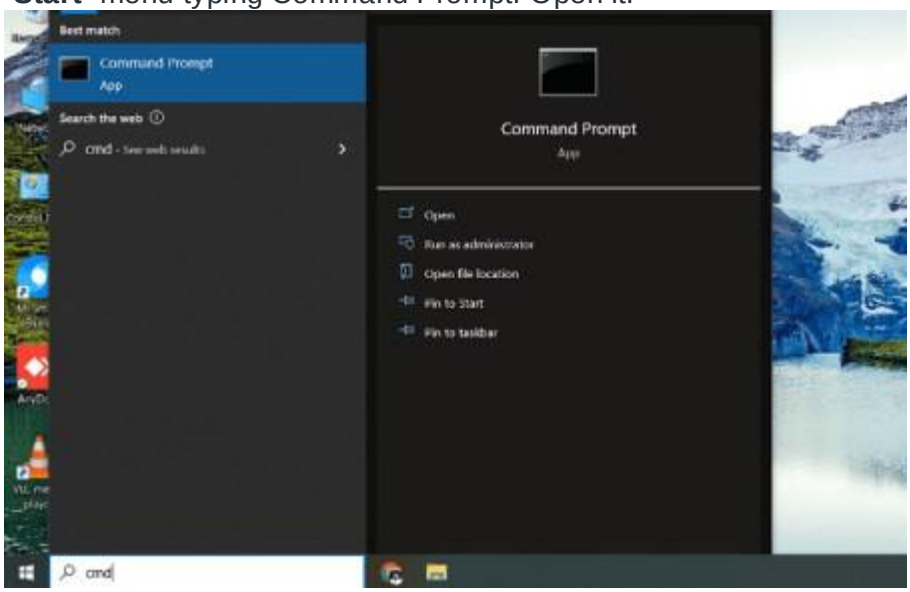
Step 16: Click on the **OK** button.



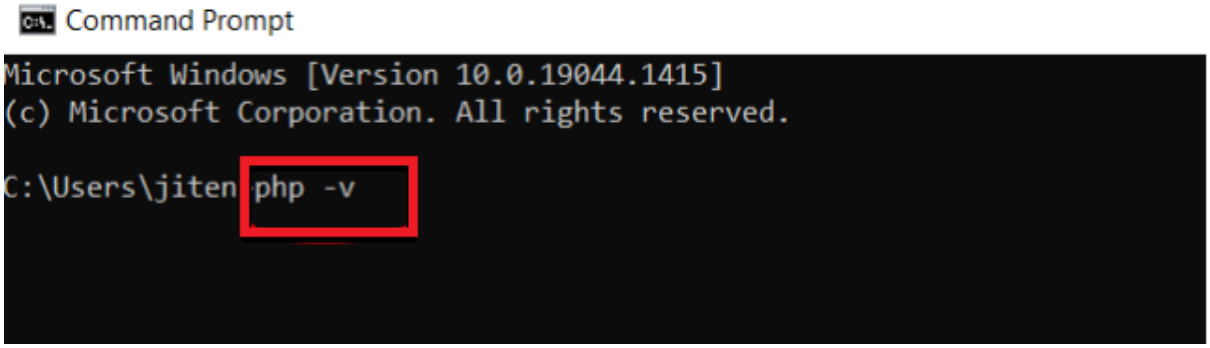
Step 17: Click on **OK** for saving changes.



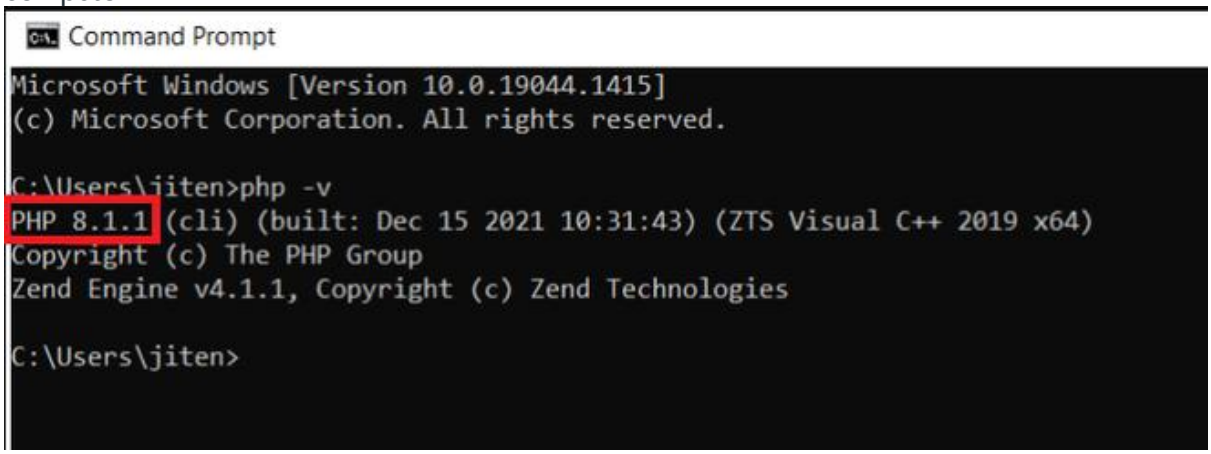
Step 18: Now your PHP is installed on your computer. You may check by going to the "**Start**" menu typing Command Prompt. Open it.



Step 19: When the Command Prompt opens, type **php -v**



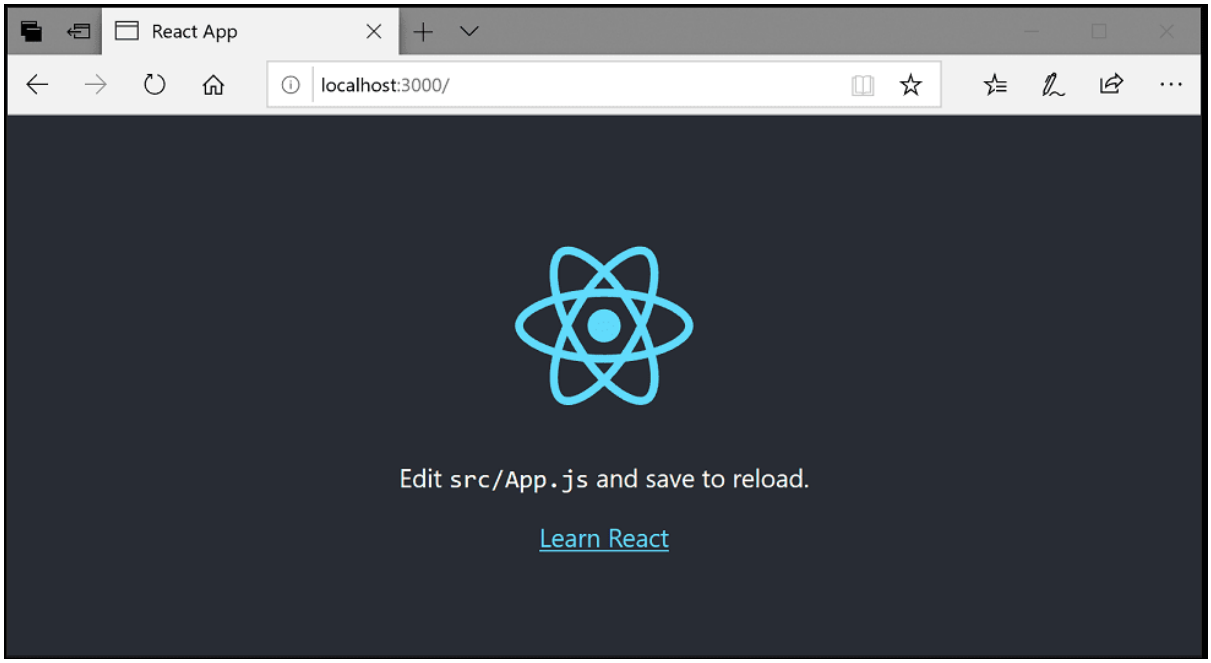
Step 20: Now enter the command prompt to show the version of PHP installed on your computer.



You have successfully installed PHP on your Windows 10 system.

5 b) Using React in Visual Studio Code

React is a popular JavaScript library developed by Facebook for building user interfaces. The Visual Studio Code editor supports React.js IntelliSense and code navigation out of the box.



Welcome to React

We'll be using the create-react-app generator for this tutorial. To use the generator as well as run the React application server, you'll need Node.js JavaScript runtime and npm (Node.js package manager) installed. npm is included with Node.js which you can download and install from Node.js downloads.

Tip: To test that you have Node.js and npm correctly installed on your machine, you can type node --version and npm --version in a terminal or command prompt.
You can now create a new React application by typing:

```
npx create-react-app my-app
```

where my-app is the name of the folder for your application. This may take a few minutes to create the React application and install its dependencies.

Note: If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app to ensure that npx always uses the latest version.
Let's quickly run our React application by navigating to the new folder and typing npm start to start the web server and open the application in a browser:

```
cd my-app

npm start
```

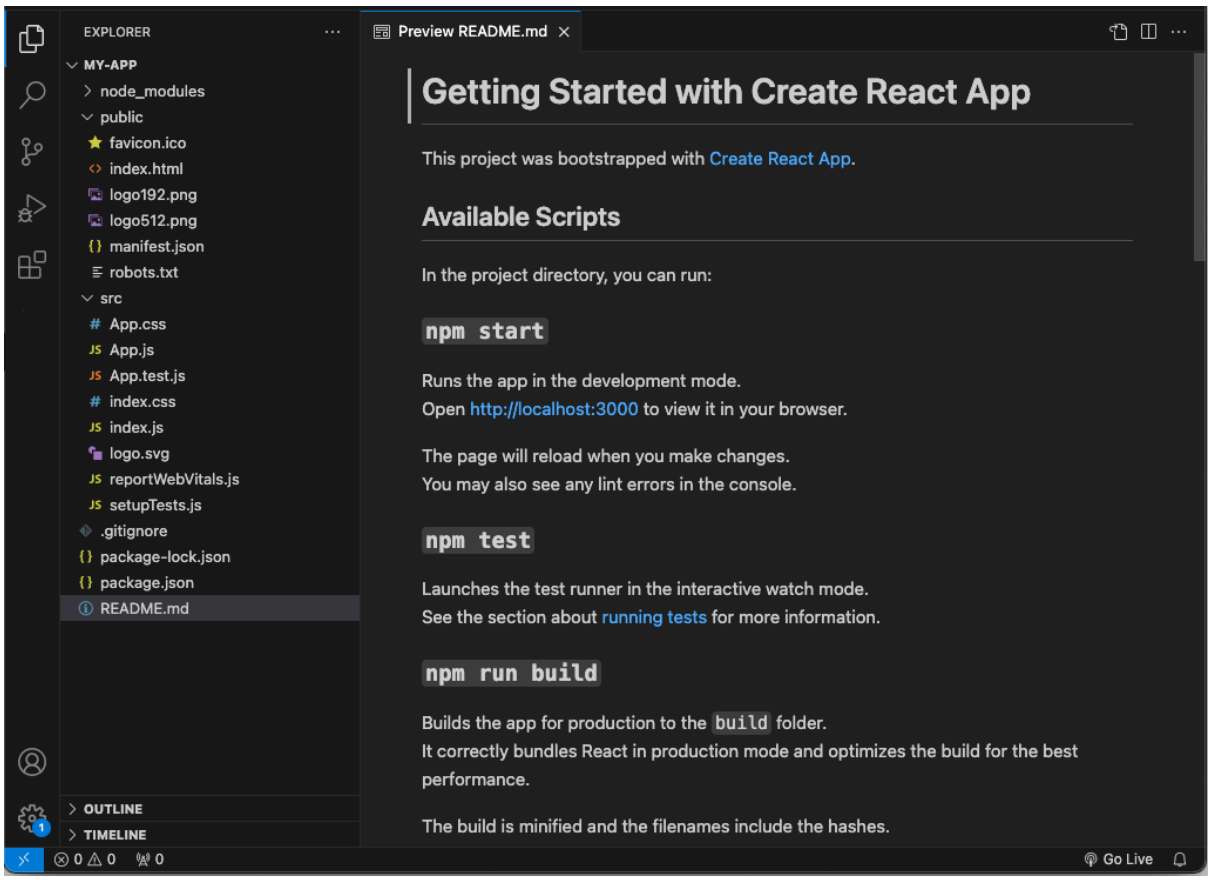
You should see the React logo and a link to "Learn React" on http://localhost:3000 in your browser. We'll leave the web server running while we look at the application with VS Code.

To open your React application in VS Code, open another terminal or command prompt window, navigate to the my-app folder and type code .:

```
cd my-app

code .
```

Markdown preview

In the File Explorer, one file you'll see is the application README.md Markdown file. This has lots of great information about the application and React in general. A nice way to review the README is by using the VS Code Markdown Preview. You can open the preview in either the current editor group (**Markdown: Open Preview** Ctrl+Shift+V) or in a new editor group to the side (**Markdown: Open Preview to the Side** Ctrl+K V). You'll get nice formatting, hyperlink navigation to headers, and syntax highlighting in code blocks.



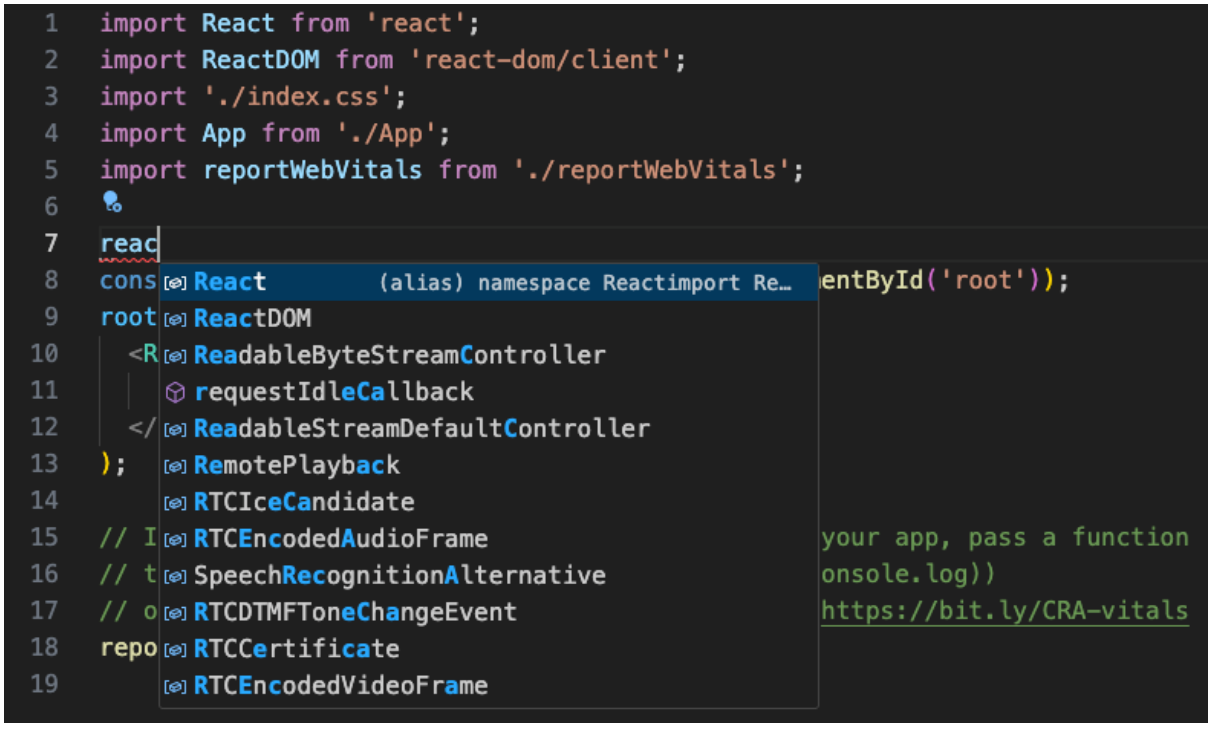
Syntax highlighting and bracket matching

Now expand the src folder and select the index.js file. You'll notice that VS Code has syntax highlighting for the various source code elements and, if you put the cursor on a parenthesis, the matching bracket is also selected.

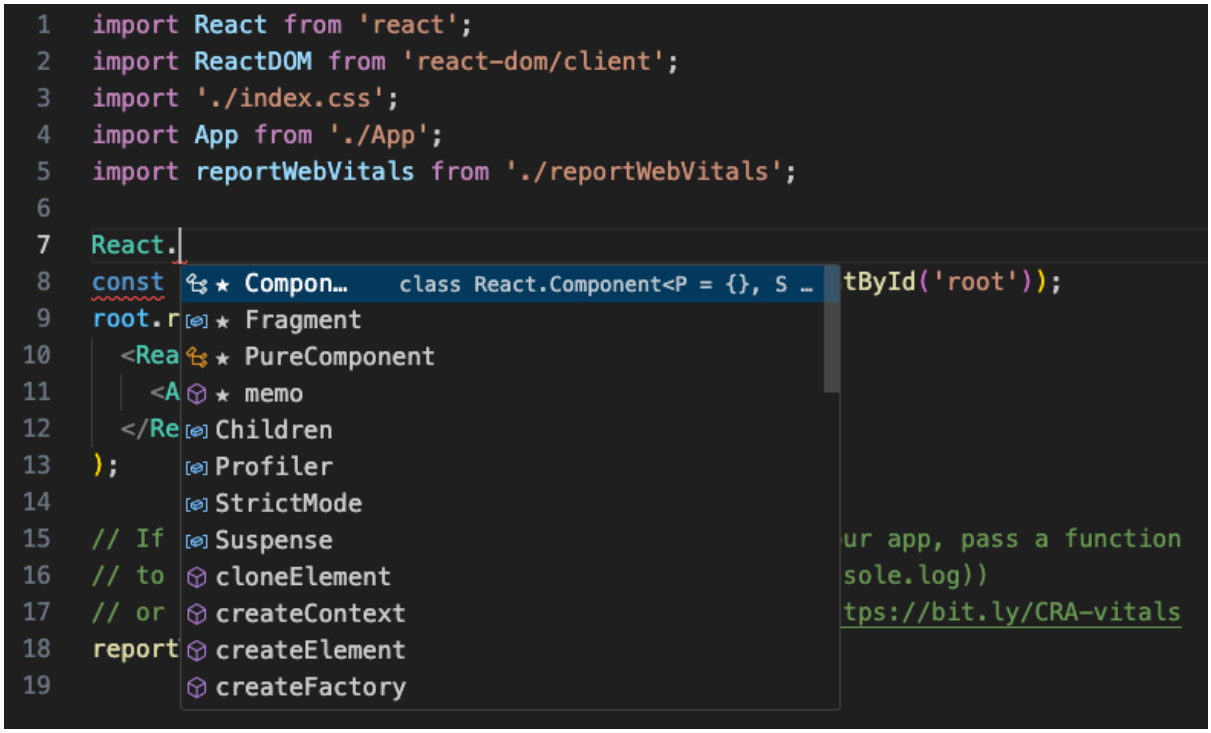
```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

IntelliSense

As you start typing in index.js, you'll see smart suggestions or completions.

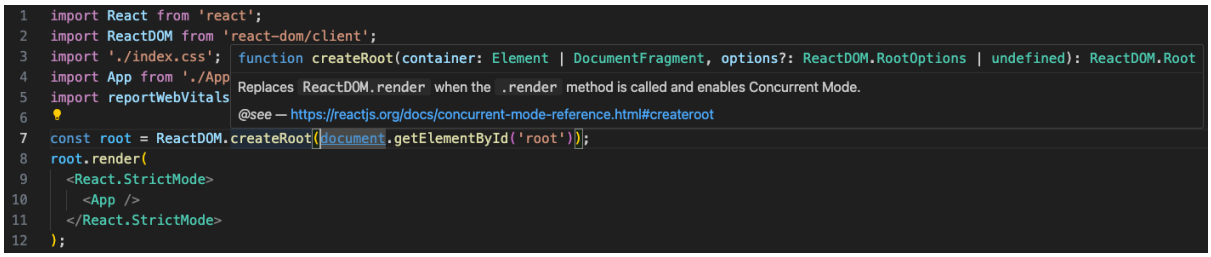


After you select a suggestion and type ., you see the types and methods on the object through IntelliSense.



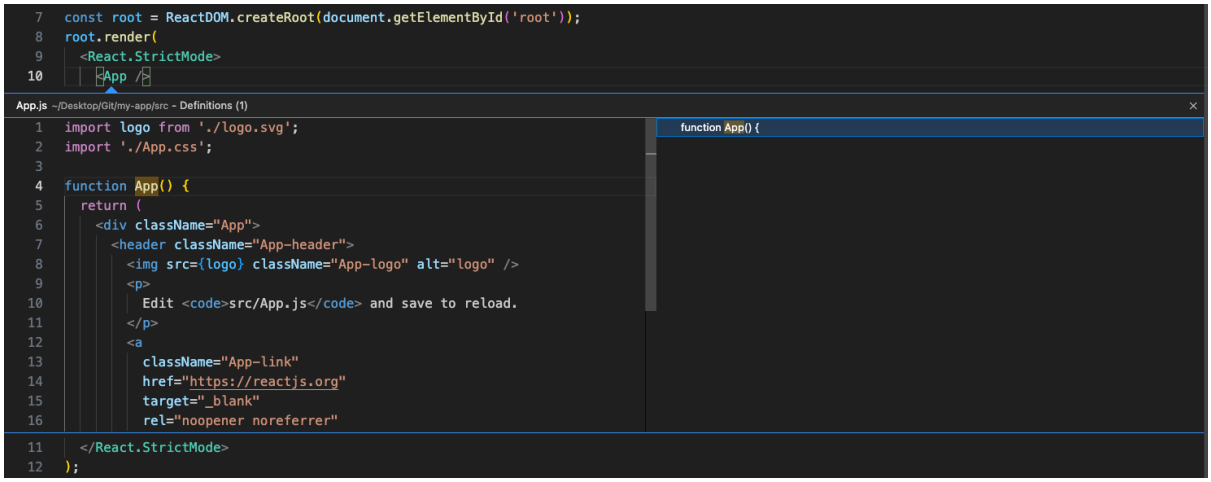
VS Code uses the TypeScript language service for its JavaScript code intelligence and it has a feature called Automatic Type Acquisition (ATA). ATA pulls down the npm Type Declaration files (*.d.ts) for the npm modules referenced in the package.json.

If you select a method, you'll also get parameter help:



Go to Definition, Peek definition

Through the TypeScript language service, VS Code can also provide type definition information in the editor through **Go to Definition** (F12) or **Peek Definition** (Alt+F12). Put the cursor over the App, right click and select **Peek Definition**. A Peek window will open showing the App definition from App.js.



Press Escape to close the Peek window.

Hello World

Let's update the sample application to "Hello World!". Create a component inside index.js called HelloWorld that contains a H1 header with "Hello, world!" and replace the <App /> tag in root.render with <HelloWorld />.

```
import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';


function HelloWorld() {

  return <h1 className="greeting">Hello, world!</h1>;

}


const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <HelloWorld />

  </React.StrictMode>

);
```

```
// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Once you save the index.js file, the running instance of the server will update the web page and you'll see "Hello World!" when you refresh your browser.

Tip: VS Code supports Auto Save, which by default saves your files after a delay. Check the **Auto Save** option in the **File** menu to turn on Auto Save or directly configure the files.autoSave user setting.

Debugging React

To debug the client side React code, we'll use the built-in JavaScript debugger.

Note: This tutorial assumes you have the Edge browser installed. If you want to debug using Chrome, replace the launch type with chrome. There is also a debugger for the Firefox browser.

Set a breakpoint

To set a breakpoint in index.js, click on the gutter to the left of the line numbers. This will set a breakpoint which will be visible as a red circle.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

function HelloWorld() {
  return <h1 className="greeting">Hello, world!</h1>
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <HelloWorld />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Configure the debugger

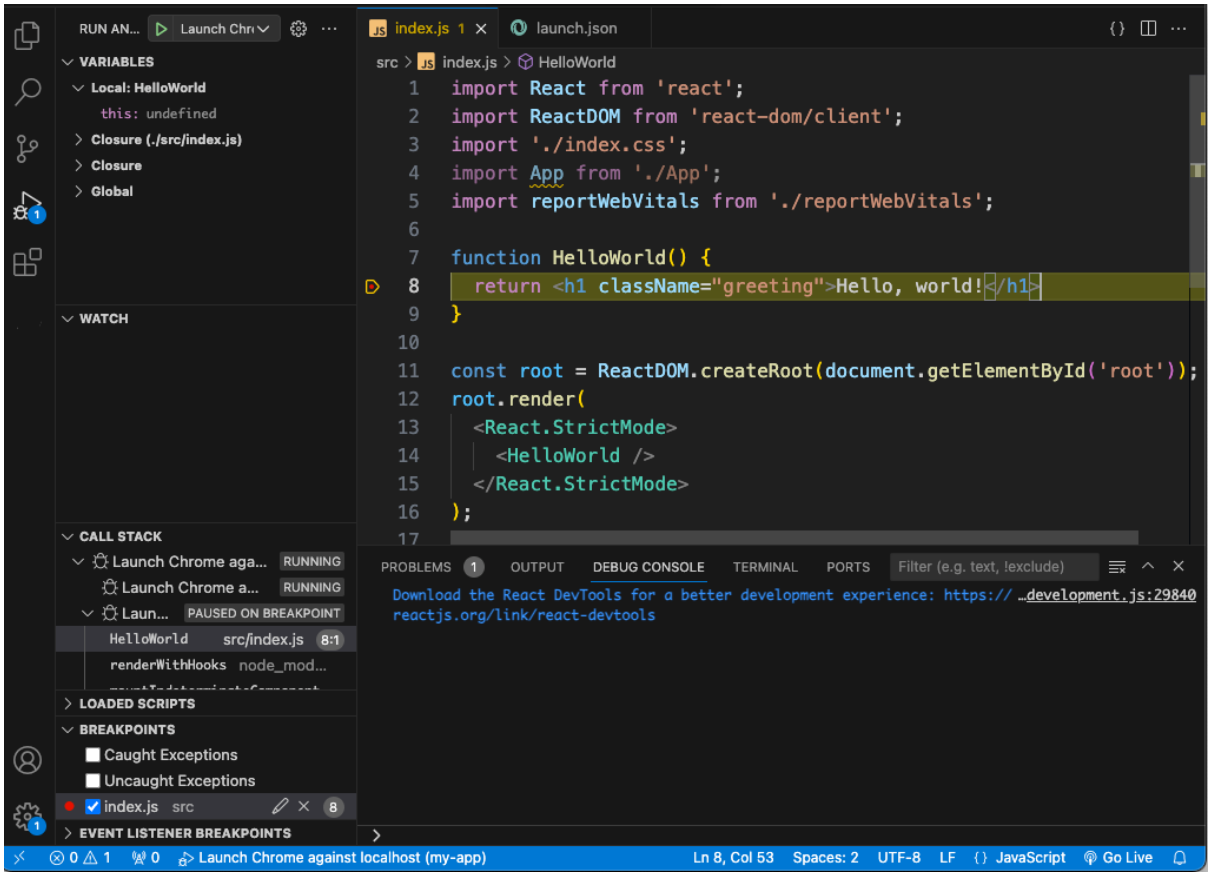
We need to initially configure the debugger. To do so, go to the **Run and Debug** view (Ctrl+Shift+D) and select the **create a launch.json file** link to create a launch.json debugger configuration file. Choose **Web App (Edge)** from the **Select debugger** dropdown list. This will create a launch.json file in a new .vscode folder in your project which includes a configuration to launch the website.

We need to make one change for our example: change the port of the url from 8080 to 3000. Your launch.json should look like this:

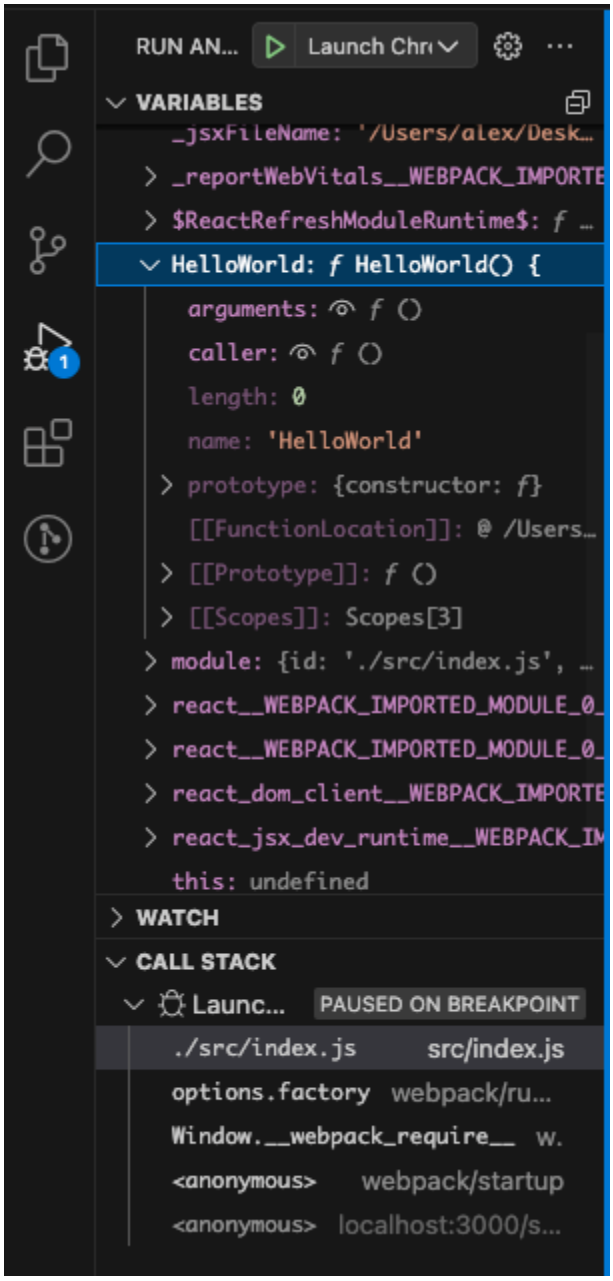
```
{
 "version": "0.2.0",
 "configurations": [
  {
   "type": "msedge",
   "request": "launch",
   "name": "Launch Edge against localhost",
   "url": "http://localhost:3000",
   "webRoot": "${workspaceFolder}"
  }
 ]
```

```
}
```

Ensure that your development server is running (npm start). Then press F5 or the green arrow to launch the debugger and open a new browser instance. The source code where the breakpoint is set runs on startup before the debugger was attached, so we won't hit the breakpoint until we refresh the web page. Refresh the page and you should hit your breakpoint.



You can step through your source code (F10), inspect variables such as HelloWorld, and see the call stack of the client side React application.



For more information about the debugger and its available options, check out our documentation on browser debugging.

Live editing and debugging

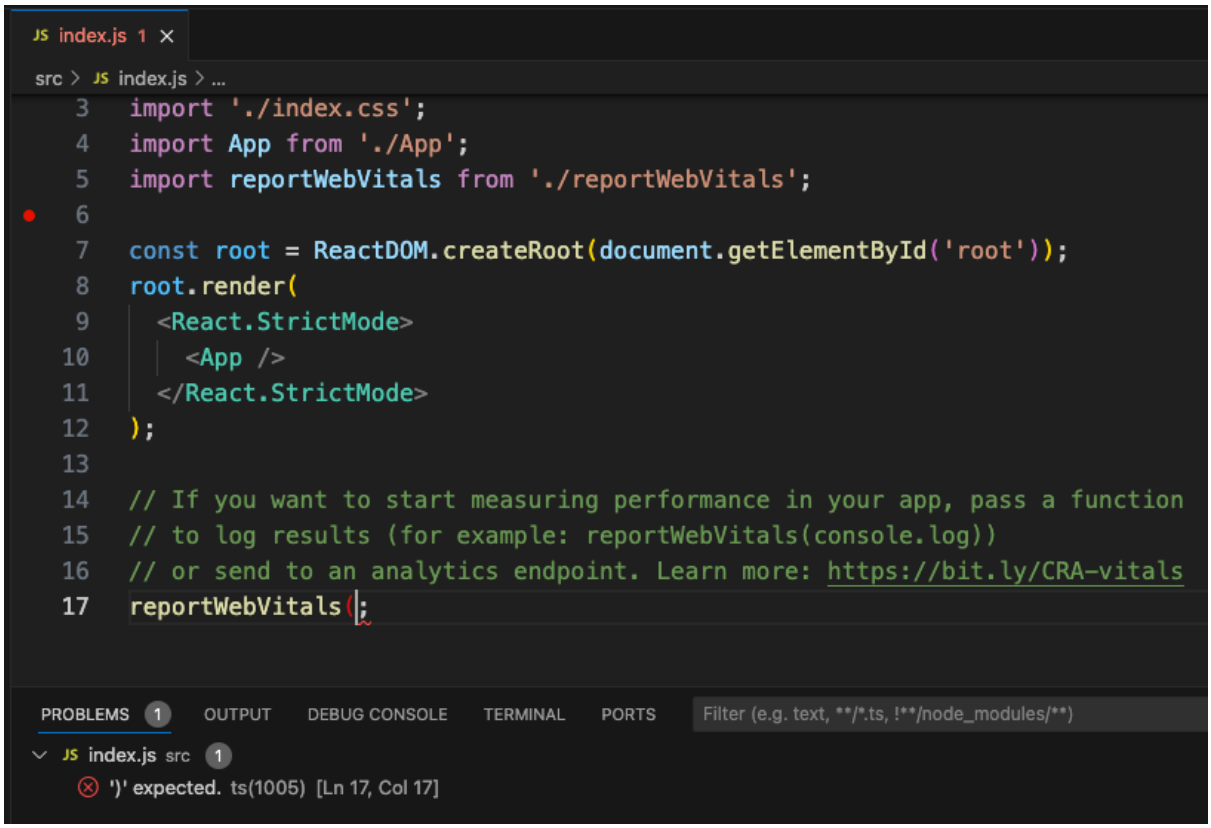
If you are using webpack together with your React app, you can have a more efficient workflow by taking advantage of webpack's HMR mechanism which enables you to have live editing and debugging directly from VS Code. You can learn more in this Live edit and debug your React apps directly from VS Code blog post and the webpack Hot Module Replacement documentation.

Linting

Linters analyze your source code and can warn you about potential problems before you run your application. The JavaScript language services included with VS Code has syntax error

checking support by default, which you can see in action in the **Problems** panel (**View** > **Problems** Ctrl+Shift+M).

Try making a small error in your React source code and you'll see a red squiggle and an error in the **Problems** panel.

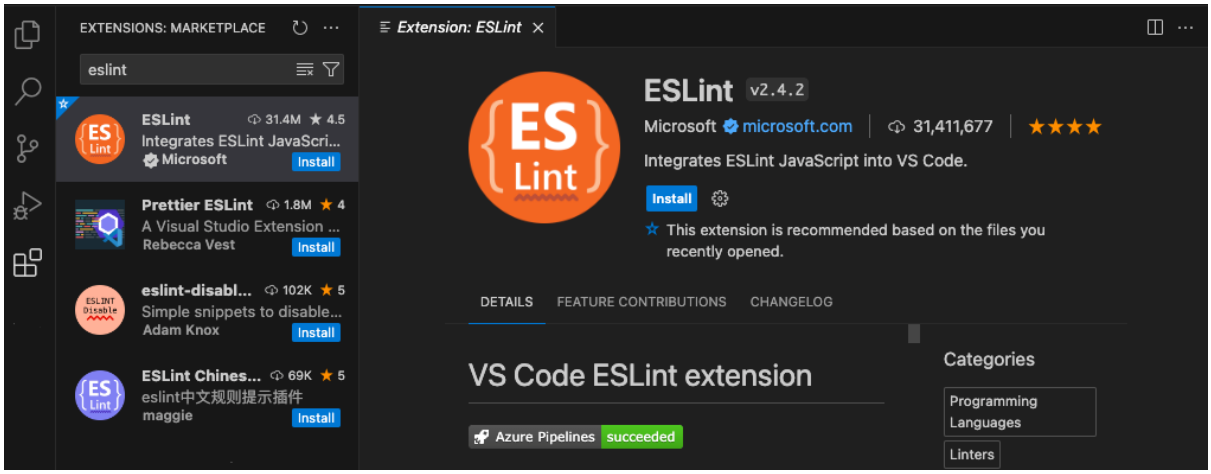


Linters can provide more sophisticated analysis, enforcing coding conventions and detecting anti-patterns. A popular JavaScript linter is ESLint. ESLint, when combined with the ESLint VS Code extension, provides a great in-product linting experience.

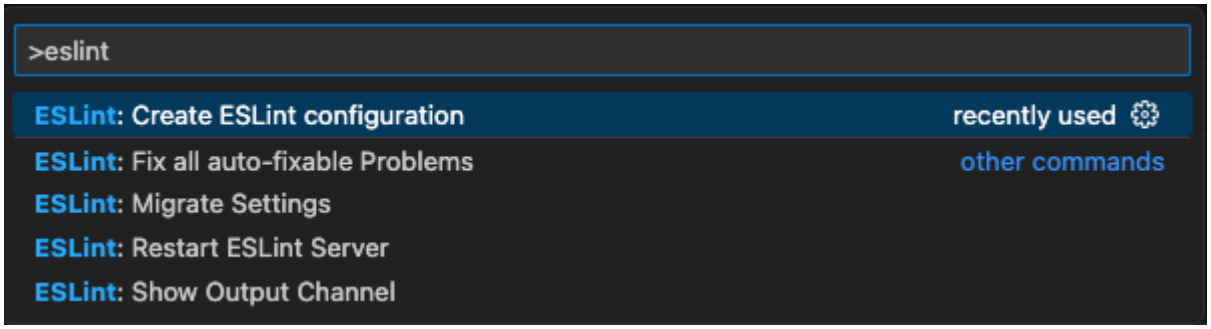
First, install the ESLint command-line tool:

```
npm install -g eslint
```

Then install the ESLint extension by going to the **Extensions** view and typing 'eslint'.



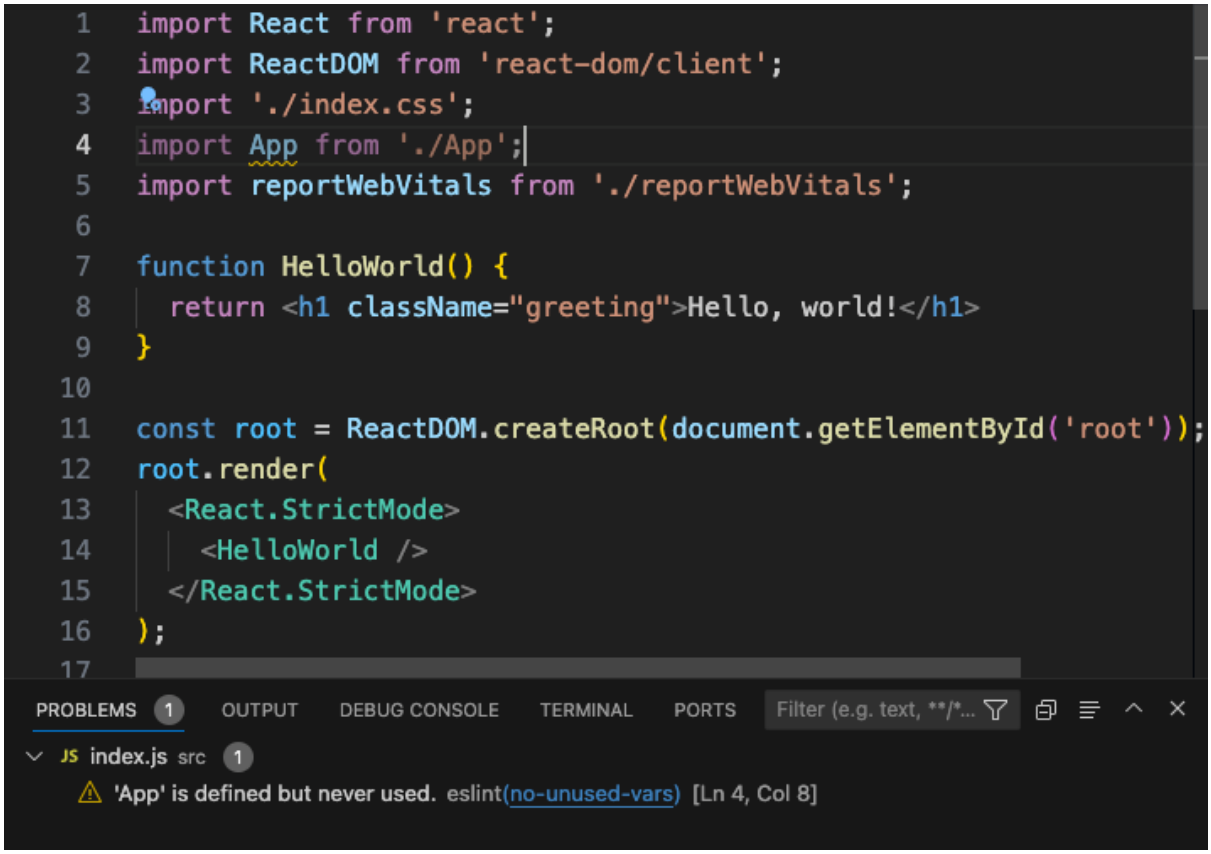
Once the ESLint extension is installed and VS Code reloaded, you'll want to create an ESLint configuration file, .eslintrc.js. You can create one using the extension's **ESLint: Create ESLint configuration** command from the **Command Palette** (Ctrl+Shift+P).



The command will prompt you to answer a series of questions in the **Terminal** panel. Take the defaults, and it will create a .eslintrc.js file in your project root that looks something like this:

```
module.exports = {
  env: {
    browser: true,
    es2020: true
  },
  extends: ['eslint:recommended', 'plugin:react/recommended'],
  parserOptions: {
    ecmaFeatures: {
      jsx: true
    },
    ecmaVersion: 11,
    sourceType: 'module'
  },
  plugins: ['react'],
  rules: {}
};
```

ESLint will now analyze open files and shows a warning in index.js about 'App' being defined but never used.

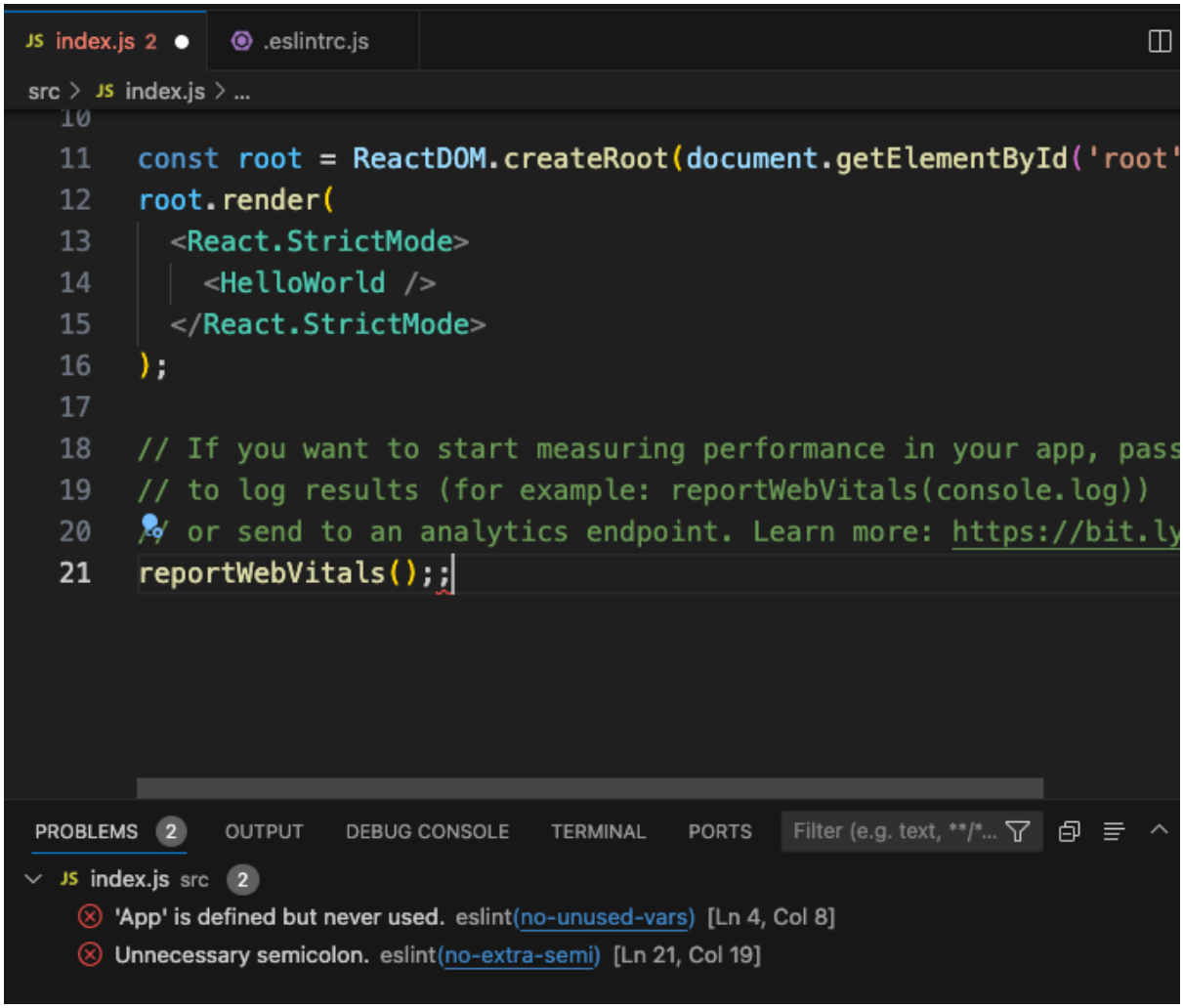


You can modify the ESLint rules in the .eslintrc.js file.

Let's add an error rule for extra semi-colons:

```
"rules": {
    "no-extra-semi":"error"
 }
```

Now when you mistakenly have multiple semicolons on a line, you'll see an error (red squiggle) in the editor and error entry in the **Problems** panel.


index.js 2
.eslintrc.js
src > index.js > ...
10
11 const root = ReactDOM.createRoot(document.getElementById('root'
12 root.render(
13 <React.StrictMode>
14 <HelloWorld />
15 </React.StrictMode>
16);
17
18 // If you want to start measuring performance in your app, pass
19 // to log results (for example: reportWebVitals(console.log))
20 // or send to an analytics endpoint. Learn more: https://bit.ly
21 reportWebVitals();;
PROBLEMS 2
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Filter (e.g. text, **/*...
index.js src 2
'App' is defined but never used. eslint(no-unused-vars) [Ln 4, Col 8]
Unnecessary semicolon. eslint(no-extra-semi) [Ln 21, Col 19]

Ex: 6

Javascript

1. Write a JavaScript program to compute the sum of an array of integers

```
function sumArray(numbers) {
    let sum = 0;
    for (let i = 0; i < numbers.length; i++) {
        sum += numbers[i];
    }
    return sum;
}

// Sample usage
let integers = [5, 10, 15, 20];
console.log("The sum of the array is:", sumArray(integers));
```

2. Write a JavaScript program to determine whether a given year is a leap year in the Gregorian calendar

```
function isLeapYear(year) {
    if ((year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0)) {
        return true;
    } else {
        return false;
    }
}

// Sample usage
let year = 2024;
if (isLeapYear(year)) {
    console.log(year + " is a leap year.");
} else {
    console.log(year + " is not a leap year.");
}
```

3. Write a JavaScript function that checks whether a passed string is palindrome or not

```
function isPalindrome(str) {
```

```
    // Convert the string to lowercase and remove non-alphanumeric characters
    let cleanedStr = str.toLowerCase().replace(/[^a-z0-9]/g, '');

    // Check if the cleaned string reads the same backward and forward
    let reversedStr = cleanedStr.split('').reverse().join('');
    return cleanedStr === reversedStr;
}

// Sample usage
let testString = "A man, a plan, a canal: Panama";
if (isPalindrome(testString)) {
    console.log(`"${testString}" is a palindrome.`);
} else {
    console.log(`"${testString}" is not a palindrome.`);
}
```

4. Write a JavaScript program to test the first character of a string is uppercase or not

```
function isFirstCharUppercase(str) {
    // Check if the string is non-empty and if the first character is uppercase
    return str.length > 0 && str[0] === str[0].toUpperCase();
}

// Sample usage
let testString = "Hello, world!";
if (isFirstCharUppercase(testString)) {
    console.log(`The first character of "${testString}" is uppercase.`);
} else {
    console.log(`The first character of "${testString}" is not uppercase.`);



}
```

5. Write a JavaScript program to set the background colour of a paragraph

```
function isFirstCharUppercase(str) {
    // Check if the string is non-empty and if the first character is uppercase
    return str.length > 0 && str[0] === str[0].toUpperCase();
```

```
}

// Sample usage
let testString = "Hello, world!";
if (isFirstCharUppercase(testString)) {
    console.log(`The first character of "${testString}" is uppercase.`);
} else {
    console.log(`The first character of "${testString}" is not uppercase.`);
}
```

6. Write a JavaScript program to check the given number is mobile number or not using form

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mobile Number Validation</title>
</head>
<body>
    <h2>Mobile Number Validation Form</h2>
    <form id="mobileForm">
        <label for="mobileNumber">Enter Mobile Number:</label>
        <input type="text" id="mobileNumber" placeholder="Enter 10-digit mobile number"
required>
        <button type="button" onclick="validateMobileNumber()">Submit</button>
    </form>
    <p id="result"></p>

    <script>
        function validateMobileNumber() {
            // Get the input value
            const mobileNumber = document.getElementById("mobileNumber").value;
```

```
        // Regular expression to check if it is a 10-digit number
        const mobilePattern = /^[0-9]{10}$/;

        // Validate and display result
        if (mobilePattern.test(mobileNumber)) {
            document.getElementById("result").innerText = "Valid mobile number.";
        } else {
            document.getElementById("result").innerText = "Invalid mobile number. Please enter a 10-digit number.";
        }
    }
  </script>
</body>
</html>
```

7. Design a student registration form and apply validation on it by using external javascript.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Registration Form</title>
  <link rel="stylesheet" href="style.css"> <!-- Optional for styling -->
</head>
<body>
  <h2>Student Registration Form</h2>
  <form id="registrationForm" onsubmit="return validateForm()">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
```

```
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>

        <label for="age">Age:</label>
        <input type="number" id="age" name="age" min="1" required>

        <label for="mobile">Mobile Number:</label>
        <input type="text" id="mobile" name="mobile" required placeholder="Enter 10-digit
number">

        <label for="address">Address:</label>
        <textarea id="address" name="address" required></textarea>

        <button type="submit">Register</button>
    </form>

    <p id="errorMessages"></p>
    <script src="validation.js"></script> <!-- Link to external JavaScript file -->
</body>
</html>

function validateForm() {
    // Get form field values
    const name = document.getElementById("name").value;
    const email = document.getElementById("email").value;
    const age = document.getElementById("age").value;
    const mobile = document.getElementById("mobile").value;
    const address = document.getElementById("address").value;

    let errorMessages = "";

    // Name validation: Ensure it's non-empty and only contains letters
    const namePattern = /^[A-Za-z\s]+$/;
```

```
    if (!namePattern.test(name)) {
        errorMessages += "Name must contain only letters and spaces.\n";
    }

    // Email validation: Checked by input type=email
    if (email === "") {
        errorMessages += "Email is required.\n";
    }

    // Age validation: Ensure it’s a positive integer
    if (isNaN(age) || age < 1) {
        errorMessages += "Please enter a valid age.\n";
    }

    // Mobile number validation: Should be exactly 10 digits
    const mobilePattern = /^[0-9]{10}$/;
    if (!mobilePattern.test(mobile)) {
        errorMessages += "Mobile number must be exactly 10 digits.\n";
    }

    // Address validation: Ensure it’s non-empty
    if (address.trim() === "") {
        errorMessages += "Address is required.\n";
    }

    // Display error messages or submit form
    if (errorMessages) {
        document.getElementById("errorMessages").innerText = errorMessages;
        return false; // Prevent form submission
    } else {
        alert("Registration Successful!");
        return true; // Allow form submission
```

```
    }
}

body {
    font-family: Arial, sans-serif;
}

h2 {
    text-align: center;
}

form {
    max-width: 400px;
    margin: auto;
    padding: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

label {
    display: block;
    margin-top: 10px;
}

input, textarea, button {
    width: 100%;
    padding: 8px;
    margin-top: 5px;
}

button {
    background-color: #4CAF50;
```

```
    color: white;
    border: none;
    cursor: pointer;
    margin-top: 15px;
}

button:hover {
    background-color: #45a049;
}

#errorMessages {
    color: red;
    text-align: center;
}
```

8. Design a simple Calculator by using html, css, and javascript

Step 1: HTML Structure (save as index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Calculator</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="calculator">
        <input type="text" id="display" disabled>

        <div class="buttons">
            <button onclick="clearDisplay()">C</button>
            <button onclick="deleteLast()">DEL</button>
            <button onclick="appendToDisplay('%')">%</button>
            <button onclick="appendToDisplay('/')">/</button>

            <button onclick="appendToDisplay('7')">7</button>
            <button onclick="appendToDisplay('8')">8</button>
            <button onclick="appendToDisplay('9')">9</button>
            <button onclick="appendToDisplay('*')">*</button>
```

```
            <button onclick="appendToDisplay('4')">4</button>
            <button onclick="appendToDisplay('5')">5</button>
            <button onclick="appendToDisplay('6')">6</button>
            <button onclick="appendToDisplay('-')">-</button>

            <button onclick="appendToDisplay('1')">1</button>
            <button onclick="appendToDisplay('2')">2</button>
            <button onclick="appendToDisplay('3')">3</button>
            <button onclick="appendToDisplay('+')">+</button>

            <button onclick="appendToDisplay('0')">0</button>
            <button onclick="appendToDisplay('.')">.</button>
            <button onclick="calculateResult()">=</button>
        </div>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

Step 2: CSS Styling (save as style.css)

```
body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    margin: 0;
}

.calculator {
    width: 250px;
    background-color: #222;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.2);
}

#display {
    width: 100%;
    height: 40px;
    font-size: 1.5em;
    text-align: right;
    margin-bottom: 10px;
    padding: 10px;
    border: none;
    background-color: #333;
    color: #fff;
```

```
    border-radius: 5px;
}

.buttons {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    gap: 10px;
}

button {
    width: 100%;
    padding: 15px;
    font-size: 1.2em;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    background-color: #444;
    color: #fff;
}

button:hover {
    background-color: #666;
}

button:nth-child(16), /* Equal button */
button:nth-child(4), /* Division */
button:nth-child(8), /* Multiplication */
button:nth-child(12), /* Subtraction */
button:nth-child(16) { /* Addition */
    background-color: #ff9500;
}

button:nth-child(16):hover {
    background-color: #ffa733;
}
```

Step 3: JavaScript Functionality (save as script.js)

```
// Function to append value to the display
function appendToDisplay(value) {
    document.getElementById("display").value += value;
}

// Function to clear the display
function clearDisplay() {
    document.getElementById("display").value = "";
}

// Function to delete the last character from the display
function deleteLast() {
```

```
    let display = document.getElementById("display").value;
    document.getElementById("display").value = display.slice(0, -1);
}

// Function to calculate and display the result
function calculateResult() {
    try {
        let display = document.getElementById("display").value;
        let result = eval(display); // Note: Use caution with eval for security
        document.getElementById("display").value = result;
    } catch (error) {
        document.getElementById("display").value = "Error";
    }
}
```

Ex : 7

Login form using Node JS and MongoDB

Prerequisite: Install MongoDB in windows

- Follow these simple steps to learn how to create a login form using Node.js and MongoDB. Node JS login form allows users to log in to the website after they have created their account using the signup form.

Express:

It is a small framework that sits on top of Node.js's web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middleware and routing; it adds helpful utilities to Node.js's HTTP objects; it facilitates the rendering of dynamic HTTP objects.

MongoDB:

the most popular NoSQL database is an open-source document-oriented database. The term 'NoSQL' means 'non-relational'. It means that MongoDB isn't based on the table-like relational database structure but provides an altogether different mechanism for the storage and retrieval of data. This format of storage is called BSON (similar to JSON format).

Passport:

It is authentication middleware for Node. js. Extremely flexible and modular, Passport can be unobtrusively dropped in to any Express-based web application. A comprehensive set of strategies supports authentication using a username and password, Facebook, Twitter, and more.

Steps to Create Project and Install Modules:

Step 1: Start the project using the following command in your project folder

npm init

Step 2: Install the required modules using the following command

npm i express ejs mongoose body-parser express-session
npm i passport passport-local
npm i passport-local-mongoose

Step 3: Create two folders inside the project directory using the following command

mkdir model
mkdir views

Step 4: Create another file named app.js inside project directory

touch app.js{ New-item or copy}

Step 5: Navigate inside model folder and create a file User.js which will contain our Schema

cd model
touch User.js {copy in windows}

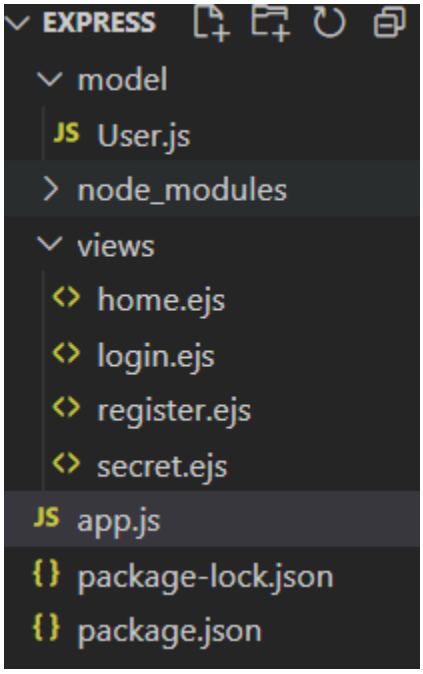
Step 6: Navigate inside views folder and create the following ejs files

cd views
touch home.ejs
touch login.ejs
touch secret.ejs
touch register.ejs

Step 7: Run the following command to ensure all modules are loaded

npm i

Project Structure:



The updated dependencies in **package.json** file.

```
"dependencies": {
    "body": "^5.1.0",
    "ejs": "^3.1.8",
    "express": "^4.18.2",
    "express-session": "^1.17.3",
    "mongoose": "^6.9.1",
    "parser": "^0.1.4",
    "passport": "^0.6.0",
    "passport-local": "^1.0.0",
    "passport-local-mongoose": "^7.1.2"
}
```

Example: Add the following code in **App.js** and **User.js** file

JavaScriptJavaScript

// Filename - App.js

```
const express = require("express"),
    mongoose = require("mongoose"),
    passport = require("passport"),
    bodyParser = require("body-parser"),
    LocalStrategy = require("passport-local"),
    passportLocalMongoose =
        require("passport-local-mongoose")
const User = require("./model/User");
let app = express();

mongoose.connect("mongodb://localhost/27017");

app.set("view engine", "ejs");
app.use(bodyParser.urlencoded({ extended: true }));
app.use(require("express-session")({
    secret: "Rusty is a dog",
    resave: false,
    saveUninitialized: false
}));

app.use(passport.initialize());
app.use(passport.session());

passport.use(new LocalStrategy(User.authenticate()));
passport.serializeUser(User.serializeUser());
passport.deserializeUser(User.deserializeUser());

//=====================
// ROUTES
//=====================
```

```
// Showing home page
app.get("/", function (req, res) {
    res.render("home");
});

// Showing secret page
app.get("/secret", isLoggedIn, function (req, res) {
    res.render("secret");
});

// Showing register form
app.get("/register", function (req, res) {
    res.render("register");
});

// Handling user signup
app.post("/register", async (req, res) => {
    const user = await User.create({
      username: req.body.username,
      password: req.body.password
    });

    return res.status(200).json(user);
  });

//Showing login form
app.get("/login", function (req, res) {
    res.render("login");
});

//Handling user login
app.post("/login", async function(req, res){
```

```
    try {
       // check if the user exists
       const user = await User.findOne({ username: req.body.username });
       if (user) {
         //check if password matches
         const result = req.body.password === user.password;
         if (result) {
           res.render("secret");
         } else {
           res.status(400).json({ error: "password doesn't match" });
         }
       } else {
         res.status(400).json({ error: "User doesn't exist" });
       }
     } catch (error) {
       res.status(400).json({ error });
     }
});

//Handling user logout
app.get("/logout", function (req, res) {
    req.logout(function(err) {
       if (err) { return next(err); }
       res.redirect('/');
     });
});




function isLoggedIn(req, res, next) {
    if (req.isAuthenticated()) return next();
    res.redirect("/login");
```

```
}

let port = process.env.PORT || 3000;
app.listen(port, function () {
    console.log("Server Has Started!");
});

// Filename - model/User.js

const mongoose = require('mongoose')
const Schema = mongoose.Schema
const passportLocalMongoose = require('passport-local-mongoose');
var User = new Schema({
    username: {
        type: String
    },
    password: {
        type: String
    }
})

User.plugin(passportLocalMongoose);

module.exports = mongoose.model('User', User)
```

Add the following codes in the folder of views

1.// Filename - views/home.ejs

```
<h1>This is home page</h1>

<li><a href="/register">Sign up!!</a></li>
```

```
<li><a href="/login">Login</a></li>
<li><a href="/logout">Logout</a></li>
```

2. // Filename - views/login.ejs

```
<h1>login</h1>

<form action="/login" method="POST">
    <input type="text" name="username"
        placeholder="username">
    <input type="password" name="password"
        placeholder="password">
    <button>login</button>
</form>

<h1>This is home page</h1>

<li><a href="/register">Sign up!!</a></li>
<li><a href="/login">Login</a></li>
<li><a href="/logout">Logout</a></li>
```

3. // Filename - views/register.ejs

```
<h1> Sign up form </h1>

<form action="/register" method="POST">
    <input type="text" name="username"
        placeholder="username">
    <input type="password" name="password"
        placeholder="password">
    <button>Submit</button>
</form>
```

```
<h1>This is home page</h1>

<li><a href="/register">Sign up!!</a></li>
<li><a href="/login">Login</a></li>
<li><a href="/logout">Logout</a></li>
```

4. // Filename - views/secret.ejs

```
<h1>This is secret page</h1>

<img src=
"https://media.geeksforgeeks.org/wp-content/cdn-uploads/20210420155809/gfg-new-logo.png">

<h1>This is home page</h1>

<li><a href="/register">Sign up!!</a></li>
<li><a href="/login">Login</a></li>
<li><a href="/logout">Logout</a></li>
```

Steps to run the application:

Step 1:To run the above code you should first have the mongoose server running

If you do not have the mongoose folder setup follow this article

After setting up mongoDB start the server using following command

```
mongod
```

Step 2: Type the following command in terminal of your project directory

```
node app.js
```

Step 3: Open your web browser and type the following address in the URL bar

http://localhost:3000/

Output:

Ex:8

Basic Login System with Node.js, Express, and MySQL

1. Why create a login system with Node.js as opposed to PHP?

Node.js is a powerful open-source server environment that leverages JavaScript as its core scripting language.

As more people become aware of Node.js, it is becoming increasingly popular in the development of web applications. Therefore, if you plan to develop applications for the future web, I highly suggest you enhance your knowledge with Node.js.

Node.js's package manager (NPM) already has over 450,000 packages available for you to download. Those numbers alone indicate how fast it's growing.

2.Setup & File Structure

- **Create a new directory called nodelogin, which can be created anywhere on your environment.**
- **Open the command line as administrator, and navigate to your new directory with the following command: cd c:\nodeprojects\nodelogin**
- **Run the command: npm init - it will prompt us to enter a package name, enter: login.**
- **When it prompts to enter the entry point, enter login.js.**

. Requirements

- **MySQL Server >= 5.6**
- **Node.js**

Install following

- **Express - Install with command: npm install express --save.**
- **Express Sessions - Install with command: npm install express-session --save.**
- **MySQL for Node.js - Install with command: npm install mysql –save**

File Structure

```
\-- nodelogin
    |-- login.html
    |-- login.js
    \-- static
        |-- style.css
```

3. Styling the Login Form with CSS

Cascading style sheets will enable us to structure the login form and make it look more appealing. The stylesheet file consists of properties that are associated with HTML elements.

Edit the style.css file and add:

CSS

```
* {
    box-sizing: border-box;
    font-family: -apple-system, BlinkMacSystemFont, "segoe ui", roboto, oxygen, ubuntu, cantarell, "fira sans", "droid sans", "helvetica neue", Arial, sans-serif;
    font-size: 16px;
}
body {
    background-color: #435165;
}
.login {
    width: 400px;
    background-color: #ffffff;
    box-shadow: 0 0 9px 0 rgba(0, 0, 0, 0.3);
    margin: 100px auto;
}
.login h1 {
    text-align: center;
    color: #5b6574;
    font-size: 24px;
    padding: 20px 0 20px 0;
    border-bottom: 1px solid #dee0e4;
}
.login form {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    padding-top: 20px;
}
```

```
.login form label {
    display: flex;
    justify-content: center;
    align-items: center;
    width: 50px;
    height: 50px;
    background-color: #3274d6;
    color: #ffffff;
}
.login form input[type="password"], .login form input[type="text"] {
    width: 310px;
    height: 50px;
    border: 1px solid #dee0e4;
    margin-bottom: 20px;
    padding: 0 15px;
}
.login form input[type="submit"] {
    width: 100%;
    padding: 15px;
   margin-top: 20px;
    background-color: #3274d6;
    border: 0;
    cursor: pointer;
    font-weight: bold;
    color: #ffffff;
    transition: background-color 0.2s;
}
.login form input[type="submit"]:hover {
  background-color: #2868c7;
    transition: background-color 0.2s;
}
```

That's all we need to add to our CSS file.

4. Creating the Login Template with HTML

The login form will consist of an HTML form element and input elements, enabling the user to enter and submit their details. There is no need to include Node.js code in the template file.

Edit the login.html file and add:

HTML

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,minimum-scale=1">
        <title>Login</title>
  <!-- the form awesome library is used to add icons to our form -->
        <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.1/css/all.css">
  <!-- include the stylesheet file -->
  <link href="/style.css" rel="stylesheet" type="text/css">
    </head>
    <body>
        <div class="login">
            <h1>Login</h1>
            <form action="/auth" method="post">
                <label for="username">
                    <!-- font awesome icon -->
                    <i class="fas fa-user"></i>
                </label>
                <input type="text" name="username" placeholder="Username" id="username" required>
                <label for="password">
                    <i class="fas fa-lock"></i>
                </label>
                <input type="password" name="password" placeholder="Password" id="password" required>
```

```
                <input type="submit" value="Login">
            </form>
        </div>
    </body>
</html>
```

Let's narrow down what each element will do.

- **Form — we need to use a form element to submit data to our Node.js app. The action attribute will point to our auth route (POST request), which we will create later on.**
    - **Input (username) — will capture the user's username. In addition, the required attribute is declared to ensure the field is mandatory.**
    - **Input (password) — will capture the user's password.**
    - **Input (submit) — button that will be used to submit the form.**

The login template will enable users to submit their details, and we'll use Node.js to validate the details. We'll be using a POST request to capture the details, which we can then handle in our Node.js auth route.

5. Creating the Login App with Node.js

Now that we have all our basics finished, we can finally start developing our app with Node.js.

To use a module in a Node.js app, we need to include it. Therefore, we need to add the following variables to our login.js file:

JS

```
const mysql = require('mysql');

const express = require('express');

const session = require('express-session');

const path = require('path');
```

The above code will include the MySQL, Express, Express-session, and Path modules that are associated with the variables we have declared.

Before we implement the database connection code, we need a database to connect to. Therefore, we must execute the below SQL statement either with the command line or your preferred MySQL Editor. Make sure the MySQL server is running on port 3306, otherwise the script will fail to connect to the database.

SQL

```
CREATE DATABASE IF NOT EXISTS `nodelogin` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;

USE `nodelogin`;
```

```
CREATE TABLE IF NOT EXISTS `accounts` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(50) NOT NULL,
  `password` varchar(255) NOT NULL,
  `email` varchar(100) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;


INSERT INTO `accounts` (`id`, `username`, `password`, `email`) VALUES (1, 'test', 'test', 'test@test.com');
```

The above SQL statement will create the database (nodelogin) and create the accounts table. In addition, it will insert a test account that we can use for testing purposes.

We can now connect to our database with the following code:

JS

```
const connection = mysql.createConnection({
        host     : 'localhost',
        user     : 'root',
        password : '',
        database : 'nodelogin'
});
```

The connection details must reflect your database credentials. In most local environments, the default username is root, so you might not have to change anything, but in production mode, we highly suggest you change the default username for MySQL and set a strong password.

Express is what we'll use for our web application, which includes packages that are essential for server-side web development, such as sessions and handling HTTP requests.

Add the following code to initialize express:

JS

```
const app = express();
```

After we need to associate the modules we'll be using with Express:

JS

```
app.use(session({
        secret: 'secret',
```

```
        resave: true,
        saveUninitialized: true
}));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'static')));
```

Make sure to update the secret code variable when declaring the session function, as it will used to secure the session data. We'll be using sessions to determine whether the user is logged in or not. The json and urlencoded methods will extract the form data from our login.html file.

Security TipWhen deploying your project to a production server, ensure you're leveraging SSL, as it will help secure the browser cookies that are associated with sessions.

After, we need to declare the login route that will output our login.html file to the client using a GET request.

Add the following:

JS

```
// http://localhost:3000/
app.get('/', function(request, response) {
        // Render login template
        response.sendFile(path.join(__dirname + '/login.html'));
});
```

When the client establishes a new connection to our Node.js server, it will output the login.html file.

Next, we need to add a new route that will authenticate the user.

Add the following:

JS

```
// http://localhost:3000/auth
app.post('/auth', function(request, response) {
        // Capture the input fields
        let username = request.body.username;
        let password = request.body.password;
        // Ensure the input fields exists and are not empty
        if (username && password) {
```

```
		// Execute SQL query that'll select the account from the database based on the specified username and password
		connection.query('SELECT * FROM accounts WHERE username = ? AND password = ?', [username, password], function(error, results, fields) {
			// If there is an issue with the query, output the error
			if (error) throw error;
			// If the account exists
			if (results.length > 0) {
				// Authenticate the user
				request.session.loggedin = true;
				request.session.username = username;
				// Redirect to home page
				response.redirect('/home');
			} else {
				response.send('Incorrect Username and/or Password!');
			}
			response.end();
		});
	} else {
		response.send('Please enter Username and Password!');
		response.end();
	}
});
```

The above code will create our authentication route using the POST method, capturing and validating our input fields when the user submits the login form.

Remember the action we declared for our form in the login template? We are using the same value for the path in our new route, so when the user submits the form, the /auth will be appended to the URL.

When the user submits the form, the code will check if both input fields are not empty and will subsequently select the account from our accounts table in our MySQL database. The user is successfully authenticated and redirected to the home page if the account exists. If not, they will encounter an error message.

The loggedin session variable will be used to determine whether the user is logged in or not, and the username variable we can use to output on the home page.

We can finally create the home route that will output the user's username.

Add the following:

JS

```
// http://localhost:3000/home
app.get('/home', function(request, response) {
	// If the user is loggedin
	if (request.session.loggedin) {
		// Output username
		response.send('Welcome back, ' + request.session.username + '!');
	} else {
		// Not logged in
		response.send('Please login to view this page!');
	}
	response.end();
});
```

Finally, our Node.js server needs to listen on a port, so for testing purposes, we can use port 3000.

Add the following:

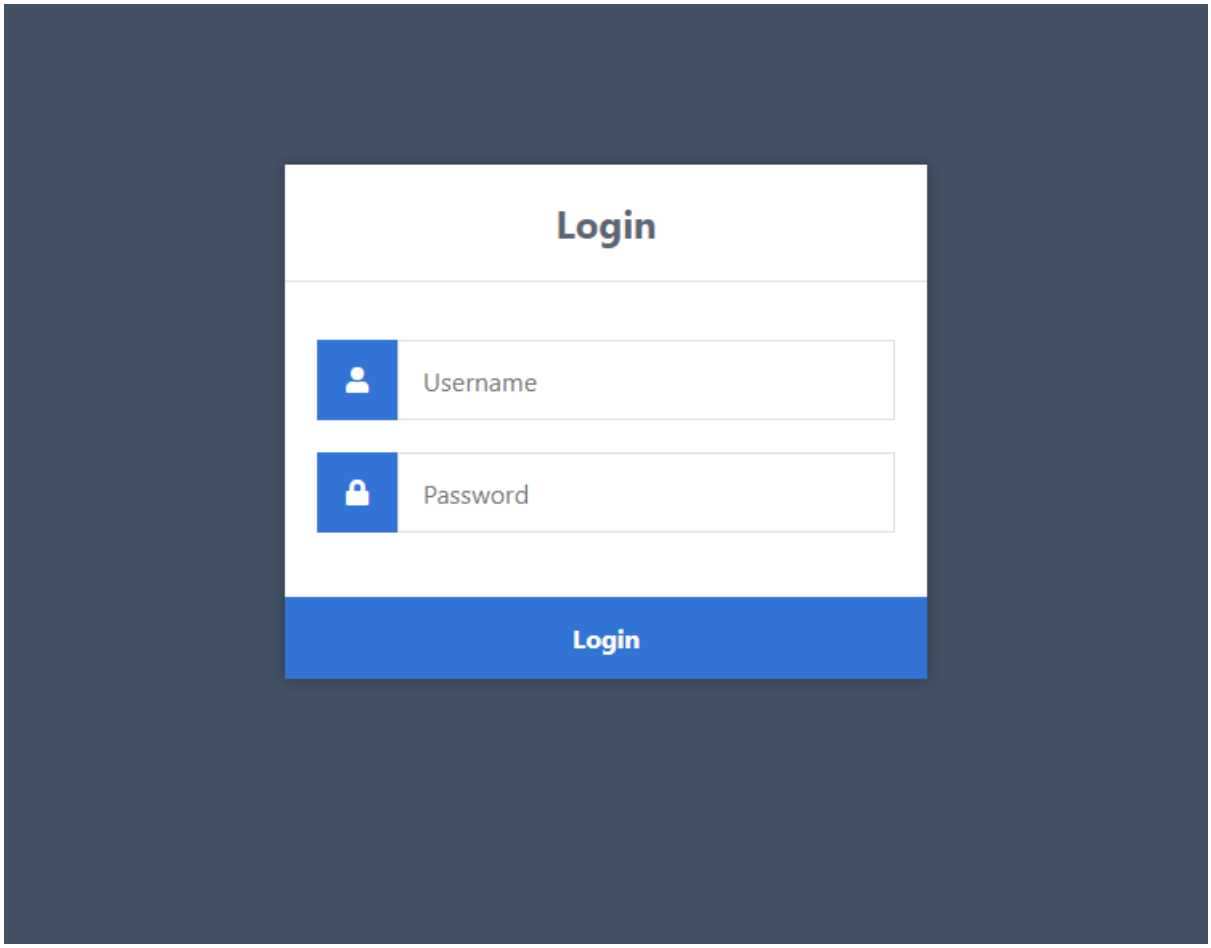
JS

```
app.listen(3000);
```

Ideally, when you deploy your login system to a production server, you would want your server to listen on port 80, so you don't have to specify the port number in the URL.

To start our Node.js app, we can execute the command node login.js in the command line. If we navigate to http://localhost:3000/ in our browser, we should see the login form, which should look like the following:

http://localhost:3000/



You can proceed to log in with the test account (username: test, password: test). If successful, you will see the username displayed on the screen.

Ex:9

Create Live Editable Table with jQuery, PHP and MySQL

Live HTML table edit or inline table edit is a very user friendly feature that enable users to edit HTML table value directly by clicking on table cells.

HTML table with jQuery and PHP, so the file structure for this example is following.

- **index.php**
- **custom_table_edit.js**
- **live_edit.php**

Steps1: Create MySQL Database Table
As we will display data record in HTML Table from MySQL database and implement live HTML table edit, so first we will create MySQL table **developers** and then insert few records to display.

```
CREATE TABLE `developers` (
`id` int(11) NOT NULL,
`name` varchar(255) NOT NULL,
`skills` varchar(255) NOT NULL,
`address` varchar(255) NOT NULL,
`gender` varchar(255) NOT NULL,
`designation` varchar(255) NOT NULL,
`age` int(11) NOT NULL
```

Steps2: Include jQuery and Tabledit plugin
As we will handle HTML Table data export using jQuery plguin **Tabledit**, so we will include jQuery and plugin files in **index.php**.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
<script type="text/javascript" src="dist/jquery.tabledit.js"></script>
<script type="text/javascript" src="custom_table_edit.js"></script>
```

Steps3: Create HTML Table with Data from MySQL
Now in **index.php**, we will create HTML table with dynamic data from MySQL database.

```
<table id="data_table" class="table table-striped">
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Gender</th>
<th>Age</th>
<th>Designation</th>
<th>Address</th>
```

```
</tr>
</thead>
<tbody>
<?php
$sql_query = "SELECT id, name, gender, address, designation, age FROM developers
LIMIT 10";
$resultset = mysqli_query($conn, $sql_query) or die("database error:".
mysqli_error($conn));
while( $developer = mysqli_fetch_assoc($resultset) ) {
?>
<tr id="<?php echo $developer ['id']; ?>">
<td><?php echo $developer ['id']; ?></td>
<td><?php echo $developer ['name']; ?></td>
<td><?php echo $developer ['gender']; ?></td>
<td><?php echo $developer ['age']; ?></td>
<td><?php echo $developer ['designation']; ?></td>
<td><?php echo $developer ['address']; ?></td>
</tr>
<?php } ?>
```

Steps4: Make HTML table Editable with Tabledit Plugin

In **custom_table_edit.js**, we will call **Tabledit()** function with HTML table id to make table cells editable with required configuration. We will also use **url** property to make Ajax call to **live_edit.php** on edit save to save edit changes into MySQL database table.

```
$(document).ready(function(){
$('#data_table').Tabledit({
deleteButton: false,
editButton: false,
columns: {
identifier: [0, 'id'],
editable: [[1, 'name'], [2, 'gender'], [3, 'age'], [4, 'designation'], [5, 'address']]
},
hideIdentifier: true,
```

Steps5: Save Live HTML Table Edit into MySQL Database

Now finally in **live_edit.php**, we will handle functionality to update edit changes into MySQL database table.

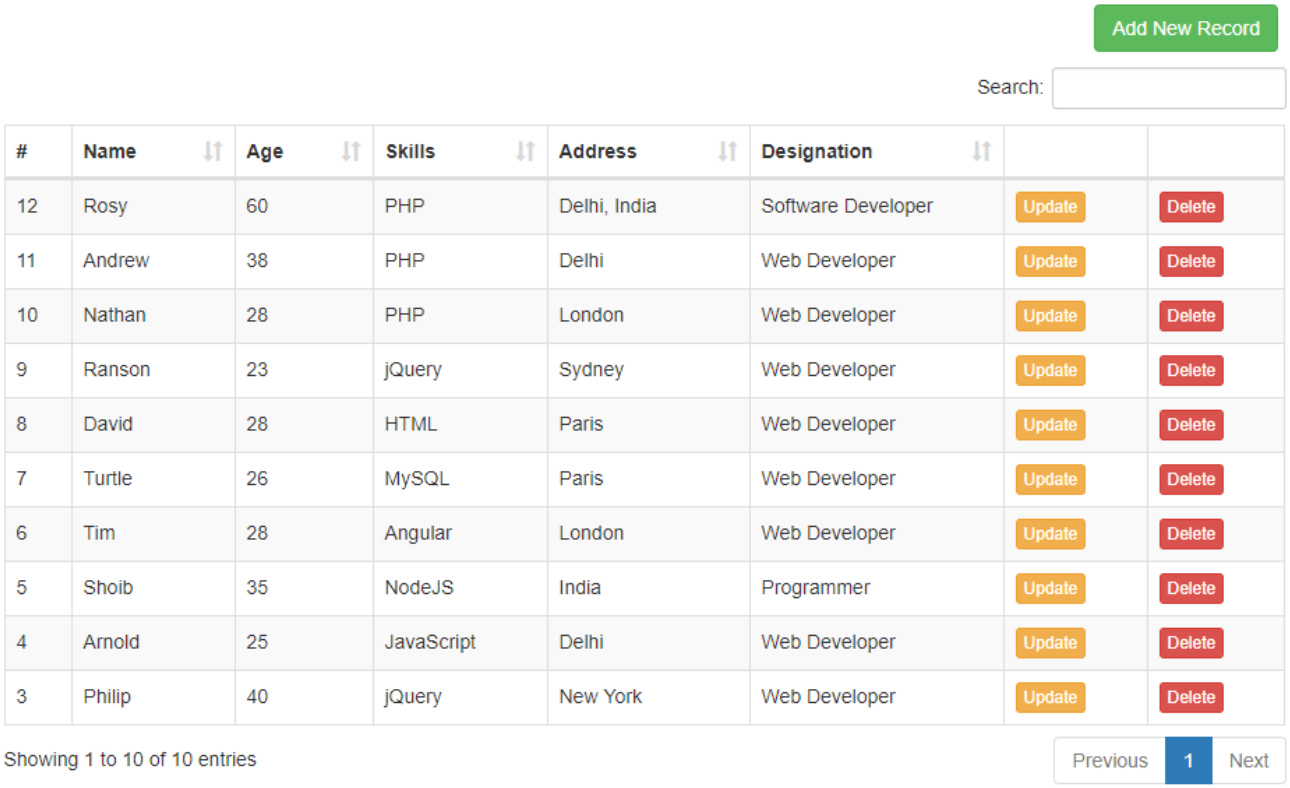
```
<?php
include_once("db_connect.php");
$input = filter_input_array(INPUT_POST);
if ($input['action'] == 'edit') {
```

```
$update_field='';
if(isset($input['name'])) {
$update_field.= "name='".$input['name']."'";
} else if(isset($input['gender'])) {
$update_field.= "gender='".$input['gender']."'";
} else if(isset($input['address'])) {
$update_field.= "address='".$input['address']."'";
} else if(isset($input['age'])) {
$update_field.= "age='".$input['age']."'";
} else if(isset($input['designation'])) {
$update_field.= "designation='".$input['designation']."'";
}
if($update_field && $input['id']) {
$sql_query = "UPDATE developers SET $update_field WHERE id='" . $input['id'] . "'";
mysqli_query($conn, $sql_query) or die("database error:". mysqli_error($conn));
}
```

Live Add Edit Delete Datatables Records with Ajax, PHP & MySQL

DataTables is a jQuery JavaScript library to convert simple HTML table to dynamic feature rich table.

The jQuery DataTables are very user friendly to list records with live add, edit, delete records without page refresh. Due to this, DataTables used widely in web application to list records.



As we will cover this tutorial with live example to Live Add Edit Delete DataTables Records with Ajax, PHP & MySQL, so the major files for this example is following.

- live-add-edit-delete-datatables-php-mysql-demo
  - config
    - Database.php
  - Class
    - Records.php
  - js
    - ajax.js
  - index.php
  - ajax_action.php

Step1: Create MySQL Database Tables

First we will create MySQL database tables **live_records** to add, edit and delete records.

```
CREATE TABLE `live_records` (
 `id` int(11) NOT NULL,
 `name` varchar(255) NOT NULL,
 `skills` varchar(255) NOT NULL,
 `address` varchar(255) NOT NULL,
 `designation` varchar(255) NOT NULL,
```

```
  `age` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

ALTER TABLE `live_records`
  ADD PRIMARY KEY (`id`);

ALTER TABLE `live_records`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

Step2: List Records in DataTables

In **index.php** file, we will create Table to list records using jQuery DataTables.

```
<table id="recordListing" class="table table-bordered table-striped">
        <thead>
                <tr>
                        <th>#</th>
                        <th>Name</th>
                        <th>Age</th>
                        <th>Skills</th>
                        <th>Address</th>
                        <th>Designation</th>

                        <th></th>
                        <th></th>
                </tr>
        </thead>
</table>
```

We will initialize jQuery DataTables in **ajax.js** file and make ajax request to action **listRecords** to make server side request to fetch records to load in DataTables.

```
var dataRecords = $('#recordListing').DataTable({
                        "lengthChange": false,
                        "processing":true,
                        "serverSide":true,
                        'serverMethod': 'post',
```

```
                "order":[],
                "ajax":{
                        url:"ajax_action.php",
                        type:"POST",
                        data:{action:'listRecords'},
                        dataType:"json"
                },
                "columnDefs":[
                        {
                                "targets":[0, 6, 7],
                                "orderable":false,
                        },
                ],
                "pageLength": 10
        });
```

We will call method **listRecords()** on action **listRecords** to list records.

```
$record = new Records();
if(!empty($_POST['action']) && $_POST['action'] == 'listRecords') {
        $record->listRecords();
}
```

We will create method **listRecords()** in class **Records.php** to fetch records from MySQL database and return as JSON data.

```
public function listRecords(){

        $sqlQuery = "SELECT * FROM ".$this->recordsTable." ";
        if(!empty($_POST["search"]["value"])){
                $sqlQuery .= 'where(id LIKE "%'.$_POST["search"]["value"].'%" ';
                $sqlQuery .= ' OR name LIKE "%'.$_POST["search"]["value"].'%" ';

                $sqlQuery .= ' OR designation LIKE "%'.$_POST["search"]["value"].'%" ';
                $sqlQuery .= ' OR address LIKE "%'.$_POST["search"]["value"].'%" ';
                $sqlQuery .= ' OR skills LIKE "%'.$_POST["search"]["value"].'%") ';

        }
```

```
        if(!empty($_POST["order"])){
                $sqlQuery .= 'ORDER BY '.$_POST['order']['0']['column'].' '.$_POST['orde
r']['0']['dir'].' ';
        } else {
                $sqlQuery .= 'ORDER BY id DESC ';
        }

        if($_POST["length"] != -1){
                $sqlQuery .= 'LIMIT ' . $_POST['start'] . ', ' . $_POST['length'];
        }

        $stmt = $this->conn->prepare($sqlQuery);
        $stmt->execute();
        $result = $stmt->get_result();

        $stmtTotal = $this->conn->prepare("SELECT * FROM ".$this->recordsTable);
        $stmtTotal->execute();
        $allResult = $stmtTotal->get_result();
        $allRecords = $allResult->num_rows;

        $displayRecords = $result->num_rows;
        $records = array();
        while ($record = $result->fetch_assoc()) {
                $rows = array();
                $rows[] = $record['id'];
                $rows[] = ucfirst($record['name']);
                $rows[] = $record['age'];
                $rows[] = $record['skills'];
                $rows[] = $record['address'];
                $rows[] = $record['designation'];

                $rows[] = '<button type="button" name="update" id="'.$record["id"].'"
 class="btn btn-warning btn-xs update">Update</button>';
                $rows[] = '<button type="button" name="delete" id="'.$record["id"].'"
```

```
 class="btn btn-danger btn-xs delete" >Delete</button>';
                $records[] = $rows;
        }

        $output = array(
                "draw"     =>         intval($_POST["draw"]),
                "iTotalRecords"     =>         $displayRecords,
                "iTotalDisplayRecords"        =>  $allRecords,
                "data"     =>         $records
        );

        echo json_encode($output);
}
```

Step3: Handle Add New Record

In **index.php** file, we will create Bootstrap modal to add new records to jQuery DataTables.

```
<div id="recordModal" class="modal fade">
        <div class="modal-dialog">
                <form method="post" id="recordForm">
                        <div class="modal-content">
                                <div class="modal-header">
                                        <button type="button" class="close"
data-dismiss="modal">×</button>
                                        <h4 class="modal-title"><i
class="fa fa-plus"></i> Add Record</h4>
                                </div>
                                <div class="modal-body">
                                        <div class="form-group"
                                                <label for="name" class="control-la
bel">Name</label>
                                                <input type="text" class="form-contr
ol"
id="name" name="name" placeholder="Name" required>
```

```
                                        </div>
                                        <div class="form-group">
                                                <label for="age" class="control-label">Age</label>
                                                <input type="number" class="form-control"
id="age" name="age" placeholder="Age">
                                        </div>
                                        <div class="form-group">
                                                <label for="lastname" class="control-label">Skills</label>
                                                <input type="text" class="form-control"
id="skills" name="skills" placeholder="Skills" required>
                                        </div>
                                        <div class="form-group">
                                                <label for="address" class="control-label">Address</label>
                                                <textarea class="form-control"
rows="5" id="address" name="address"></textarea>
                                        </div>
                                        <div class="form-group">
                                                <label for="lastname" class="control-label">Designation</label>
                                                <input type="text" class="form-control"
id="designation" name="designation" placeholder="Designation">
                                        </div>

                                </div>
                                <div class="modal-footer">
                                        <input type="hidden" name="id" id="id" />
                                        <input type="hidden" name="action" id="action" value="" />
                                        <input type="submit" name="save" id="save"
class="btn btn-info" value="Save" />
```

```
                                                <button type="button" class="btn btn-default"
data-dismiss="modal">Close</button>
                                        </div>
                                </div>
                        </form>
                </div>
</div>
```

We will handle modal form submit using jQuery and make Ajax request with action **addRecord** to add new records.

```
$("#recordModal").on('submit','#recordForm', function(event){
        event.preventDefault();
        $('#save').attr('disabled','disabled');
        var formData = $(this).serialize();
        $.ajax({
                url:"ajax_action.php",
                method:"POST",
                data:formData,
                success:function(data){
                        $('#recordForm')[0].reset();
                        $('#recordModal').modal('hide');

                        $('#save').attr('disabled', false);
                        dataRecords.ajax.reload();
                }
        })
});
```

We will call method **addRecord()** on action **addRecord** to add new records.

```
$database = new Database();
$db = $database->getConnection();


$record = new Records($db);


if(!empty($_POST['action']) && $_POST['action'] == 'addRecord') {
```

```
        $record->name = $_POST["name"];
    $record->age = $_POST["age"];
    $record->skills = $_POST["skills"];
        $record->address = $_POST["address"];
        $record->designation = $_POST["designation"];
        $record->addRecord();
}
```

We will create method **addRecord()** in class **Records.php** to add new records into MySQL database.

```
public function addRecord(){

                if($this->name) {

                        $stmt = $this->conn->prepare("
                        INSERT INTO ".$this->recordsTable."(`name`, `age`, `skills`, `address`, `designation`)
                        VALUES(?,?,?,?,?)");

                        $this->name = htmlspecialchars(strip_tags($this->name));
                        $this->age = htmlspecialchars(strip_tags($this->age));
                        $this->skills = htmlspecialchars(strip_tags($this->skills));
                        $this->address = htmlspecialchars(strip_tags($this->address));
                        $this->designation = htmlspecialchars(strip_tags($this->designation));


                        $stmt->bind_param("sisss", $this->name, $this->age, $this->skills,
$this->address, $this->designation);

                        if($stmt->execute()){
                                return true;
                        }
                }
```

```
        }
```

Step4: Handle Update Record

We will handle records update functionality by populating records values to update modal form by make Ajax request to action **getRecord** to load values to modal form input.

```
$("#recordListing").on('click', '.update', function(){
        var id = $(this).attr("id");
        var action = 'getRecord';
        $.ajax({
                url:'ajax_action.php',
                method:"POST",
                data:{id:id, action:action},
                dataType:"json",
                success:function(data){
                        $('#recordModal').modal('show');
                        $('#id').val(data.id);
                        $('#name').val(data.name);
                        $('#age').val(data.age);
                        $('#skills').val(data.skills);
                        $('#address').val(data.address);
                        $('#designation').val(data.designation);
                        $('.modal-title').html(" Edit Records");
                        $('#action').val('updateRecord');
                        $('#save').val('Save');
                }
        })
});
```

We will call method **updateRecord()** from class **Records.php** to update records.

```
$database = new Database();
$db = $database->getConnection();


$record = new Records($db);
```

```
if(!empty($_POST['action']) && $_POST['action'] == 'updateRecord') {
        $record->id = $_POST["id"];
        $record->name = $_POST["name"];
    $record->age = $_POST["age"];
    $record->skills = $_POST["skills"];
        $record->address = $_POST["address"];
        $record->designation = $_POST["designation"];
        $record->updateRecord();
}
```

We will create method **updateRecord()** in class **Records.php** to update records into MySQL database table.

```
public function updateRecord(){

            if($this->id) {

                $stmt = $this->conn->prepare("
                UPDATE ".$this->recordsTable."
                SET name= ?, age = ?, skills = ?, address = ?, designation = ?
                WHERE id = ?");

                $this->id = htmlspecialchars(strip_tags($this->id));
                $this->name = htmlspecialchars(strip_tags($this->name));
                $this->age = htmlspecialchars(strip_tags($this->age));
                $this->skills = htmlspecialchars(strip_tags($this->skills));
                $this->address = htmlspecialchars(strip_tags($this->address));
                $this->designation = htmlspecialchars(strip_tags($this->designa
tion));


                $stmt->bind_param("sisssi", $this->name, $this->age, $this->skil
ls, $this->address, $this->designation, $this->id);

                if($stmt->execute()){
```

```
                                        return true;
                                }


                        }
                }
```

Step5: Handle Delete Records

We will handle records delete functionality by making ajax request with action **deleteRecord** to delete record from MySQL database table.

```
$("#recordListing").on('click', '.delete', function(){
        var id = $(this).attr("id");
        var action = "deleteRecord";
        if(confirm("Are you sure you want to delete this record?")) {
                $.ajax({
                        url:"ajax_action.php",
                        method:"POST",
                        data:{id:id, action:action},
                        success:function(data) {

                                dataRecords.ajax.reload();
                        }
                })
        } else {
                return false;
        }
});
```

We will call method **deleteRecord()** from class **Records.php** on action **deleteRecord** to delete records.

```
$database = new Database();
$db = $database->getConnection();


$record = new Records($db);


if(!empty($_POST['action']) && $_POST['action'] == 'deleteRecord') {
```

```
        $record->id = $_POST["id"];
        $record->deleteRecord();
}
```

We will create method **deleteRecord()** in class **Records.php** to delete records into MySQL database table.

```
public function deleteRecord(){
        if($this->id) {

                $stmt = $this->conn->prepare("
                        DELETE FROM ".$this->recordsTable."
                        WHERE id = ?");

                $this->id = htmlspecialchars(strip_tags($this->id));

                $stmt->bind_param("i", $this->id);

                if($stmt->execute()){
                        return true;
                }
        }
    }
```

Ex: 10

How to Submit AJAX Forms with JQuery

Introduction

jQuery can be paired with form submission to handle validation. This has the benefit of providing users with feedback on any errors in their input.

Prerequisites

- assumes you have PHP installed locally and are able to run the built-in web server. You may be able to consult one of our tutorials for installing PHP in your environment.
- Some familiarity with selectors and methods from the jQuery library.
- Some familiarity with classes from the Bootstrap library.
- A code editor.
- A modern web browser.

Note: This tutorial does not specify the latest versions of jQuery (currently 3.5.1) or Bootstrap (currently 5.0.0-beta1). However, many of the lessons in this tutorial still pertain to the latest versions.

This tutorial was verified with PHP v7.3.24, jQuery v2.0.3, and Bootstrap v3.0.3.

Step 1 — Building the Backend with PHP

For the purposes of this tutorial, the backend will be written in PHP.

First, open a terminal window and create a new project directory:

```
1. mkdir jquery-form-validation
2.
```

Copy

Navigate to this new project directory:

```
1. cd jquery-form-validation
2.
```

Copy

Then, use your code editor to create a new process.php file:

process.php

```
<?php

$errors = [];
$data = [];

if (empty($_POST['name'])) {
    $errors['name'] = 'Name is required.';
}

if (empty($_POST['email'])) {
    $errors['email'] = 'Email is required.';
}
```

```
if (empty($_POST['superheroAlias'])) {
    $errors['superheroAlias'] = 'Superhero alias is required.';
}

if (!empty($errors)) {
    $data['success'] = false;
    $data['errors'] = $errors;
} else {
    $data['success'] = true;
    $data['message'] = 'Success!';
}

echo json_encode($data);
```

Copy

This file will take values for name, email, and superheroAlias. If any of these values are not provided, an error message will be sent back. Many other validations could be performed at this point, but for the purposes of this tutorial, you will only be ensuring these required inputs have been provided. Otherwise, if a value for name, email, and superheroAlias are present, a success message will be sent back.

Note: In a real-world scenario, the backend would also be responsible for other tasks such as taking the data and saving changes to a database, creating a session, or sending an email.

Now that you have the form processing completed, you can create the form.

Step 2 — Building the Frontend with HTML and CSS

For the purposes of this tutorial, Bootstrap will be used to build out the views.

In your project directory, use your code editor to create an index.html file:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Form Example</title>
    <link
      rel="stylesheet"
      href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css"
    />
    <script src="//ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script>
</head>
  <body>
    <div class="col-sm-6 col-sm-offset-3">
      <h1>Processing an AJAX Form</h1>

      <form action="process.php" method="POST">
        <div id="name-group" class="form-group">
          <label for="name">Name</label>
          <input
            type="text"
            class="form-control"
            id="name"
```

```
          name="name"
          placeholder="Full Name"
        />
      </div>

      <div id="email-group" class="form-group">
        <label for="email">Email</label>
        <input
          type="text"
          class="form-control"
          id="email"
          name="email"
          placeholder="email@example.com"
        />
      </div>

      <div id="superhero-group" class="form-group">
        <label for="superheroAlias">Superhero Alias</label>
        <input
          type="text"
          class="form-control"
          id="superheroAlias"
          name="superheroAlias"
          placeholder="Ant Man, Wonder Woman, Black Panther, Superman, Black Widow"
        />
      </div>

      <button type="submit" class="btn btn-success">
        Submit
      </button>
    </form>
  </div>
 </body>
</html>
```

Copy

The CDN (content delivery network) version of Bootstrap and jQuery will be referenced. The form's action will be set to the PHP file that was created earlier. The form will consist of fields for name, email, and superheroAlias. The form will also need a **Submit** button.

Open a terminal window and navigate to the project directory. And run the PHP server:

```
1. php -S localhost:8000
2.
```

Copy

Visit localhost:8000 in your web browser and observe the following:

Now that you have the form completed, you can create the script to handle form submission.

Step 3 — Handling Form Submit Logic in JavaScript and jQuery

To submit a form via AJAX, your script will need to handle four tasks:

- Capture the form submit button so that the default action does not take place.
- Get all of the data from the form using jQuery.
- Submit the form data using AJAX.
- Display errors if there are any.

In your project directory, use your code editor to create a new form.js file:

form.js

```
$(document).ready(function () {
  $("form").submit(function (event) {
    var formData = {
      name: $("#name").val(),
      email: $("#email").val(),
      superheroAlias: $("#superheroAlias").val(),
    };

    $.ajax({
      type: "POST",
      url: "process.php",
      data: formData,
      dataType: "json",
      encode: true,
    }).done(function (data) {
      console.log(data);
    });

    event.preventDefault();
  });
});
```

Copy

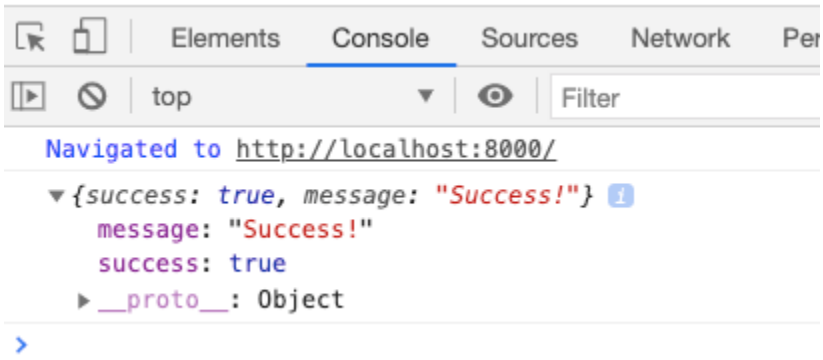
This code retrieves the data from the name, email, and superheroAlias fields. It then performs an AJAX request to process.php with a payload of form data. After a successful connection, the console will log the response data. event.preventDefault() is used to prevent the form from behaving by default by reloading the page on submission.
After saving your changes to form.js, revisit the index.html file with your code editor. Add a reference to the new JavaScript file:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Form Example</title>
    <!-- ... -->
    <script src="form.js"></script>
</head>
<!-- ... -->
```

Copy

Now when a user presses the **Submit** button on the form, the JavaScript code will retrieve all the input values and send a POST request to process.php.
Note: You will be using the .done callback to handle a successful AJAX request. This used to be called .success, but that has since been deprecated in jQuery 1.8+.
Another alternative is to use serialize instead of pulling the form information individually.
The PHP script will process the inputs that the AJAX call sent and return the $data[] array that was created. You can observe this in your browser's console after you submit your form:



Now that you have the form logic completed, you can create the script to handle form errors.

Step 4 — Displaying Form Validation Errors

In the PHP script, the code checks to ensure that all the fields are required. If a field is not present, an error is sent back.

Revisit form.js and add the following highlighted lines of code:

form.js

```
// ...

  $.ajax({
    type: "POST",
    url: "process.php",
    data: formData,
    dataType: "json",
    encode: true,
```

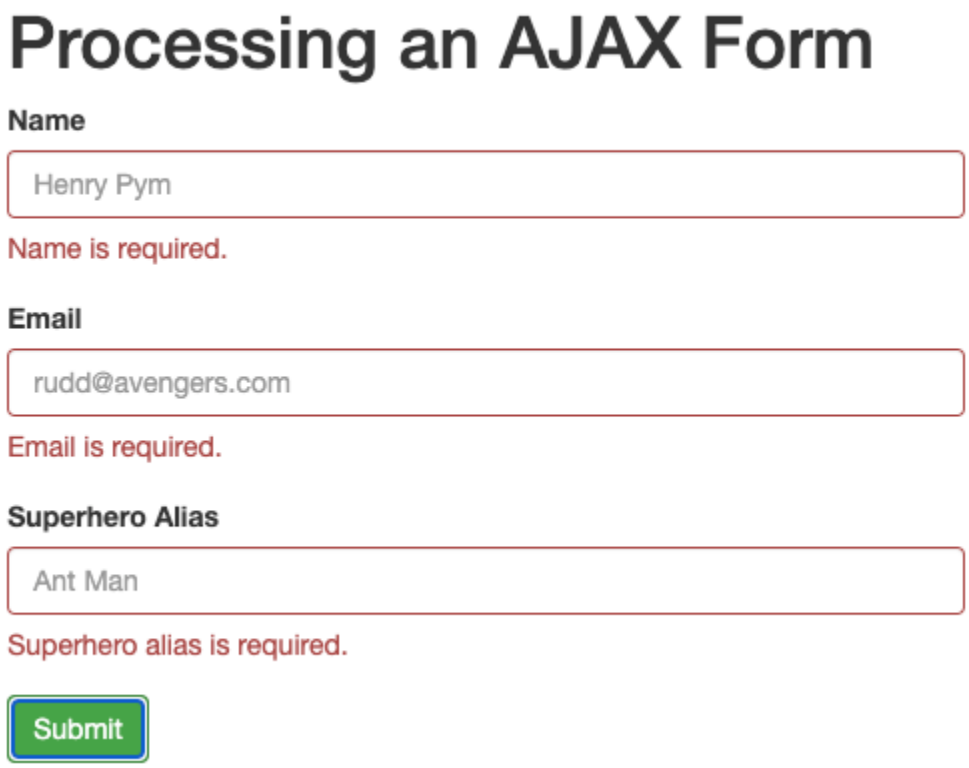
```
  }).done(function (data) {
    console.log(data);

    if (!data.success) {
      if (data.errors.name) {
        $("#name-group").addClass("has-error");
        $("#name-group").append(
          '<div class="help-block">' + data.errors.name + "</div>"
        );
      }

      if (data.errors.email) {
        $("#email-group").addClass("has-error");
        $("#email-group").append(
          '<div class="help-block">' + data.errors.email + "</div>"
        );
      }

      if (data.errors.superheroAlias) {
        $("#superhero-group").addClass("has-error");
        $("#superhero-group").append(
          '<div class="help-block">' + data.errors.superheroAlias + "</div>"
        );
      }
    } else {
      $("form").html(
        '<div class="alert alert-success">' + data.message + "</div>"
      );
    }

  });

  event.preventDefault();
});
// ...
```

Copy

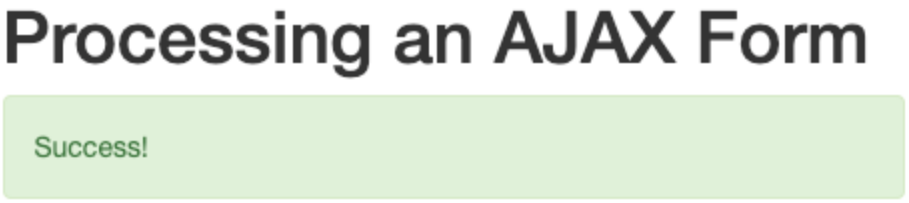
This code checks to see if the response contains an error for each field. If an error is present, it adds a has-error class and appends the error message.

Now, revisit your form in a web browser and experiment with submitting data with the form.

If there are any errors that come back from the server, the form will provide feedback on any required fields:



And if there are no errors that come back from the server, the form will provide feedback for a successful submission:



Every time we submit the form, our errors from our previous submission are still there. You will need to clear them by removing them as soon as the form is submitted again.

Revisit form.js and add the following highlighted lines of code:

form.js

```
// ...

  $("form").submit(function (event) {
    $(".form-group").removeClass("has-error");
    $(".help-block").remove();

    // ...
  });

// ...
```

Copy

This code will remove the has-error class from all .form-group elements. It will also remove all .help-block elements with error messages.

Step 5 — Displaying Server Connection Errors

If there is an error connecting to the server, there will be no JSON response from the AJAX call. To prevent users from waiting for a response that will never arrive, you can provide an error message for connection failures.

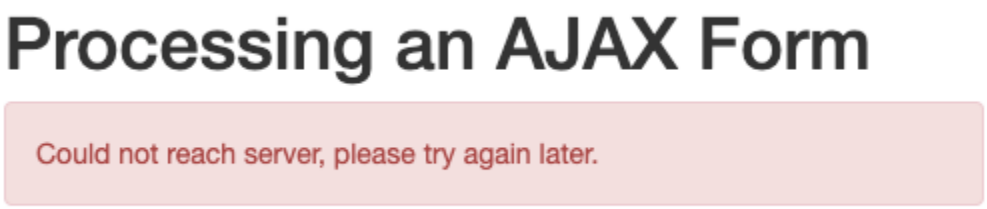
Revisit form.js and add the following highlighted lines of code:

form.js

```
// ...

    $.ajax({
      type: "POST",
      url: "process.php",
      data: formData,
      dataType: "json",
      encode: true,
    })
      .done(function(data) {
        // ...
      })
      .fail(function (data) {
        $("form").html(
          '<div class="alert alert-danger">Could not reach server, please try again later.</div>'
        );
      });

// ...
```

Copy

If the server is broken or down for any reason, a user who attempts to submit a form will get an error message:



Now that you have the server error message complete, you have completed the example form.

Using $.post instead of $.ajax

jQuery also provides a $.post shorthand method as an alternative to $.ajax.
The $.ajax code in form.js could be rewritten with $.post:

```
$.post('process.php', function(formData) {
  // place success code here
})
  .fail(function(data) {
```

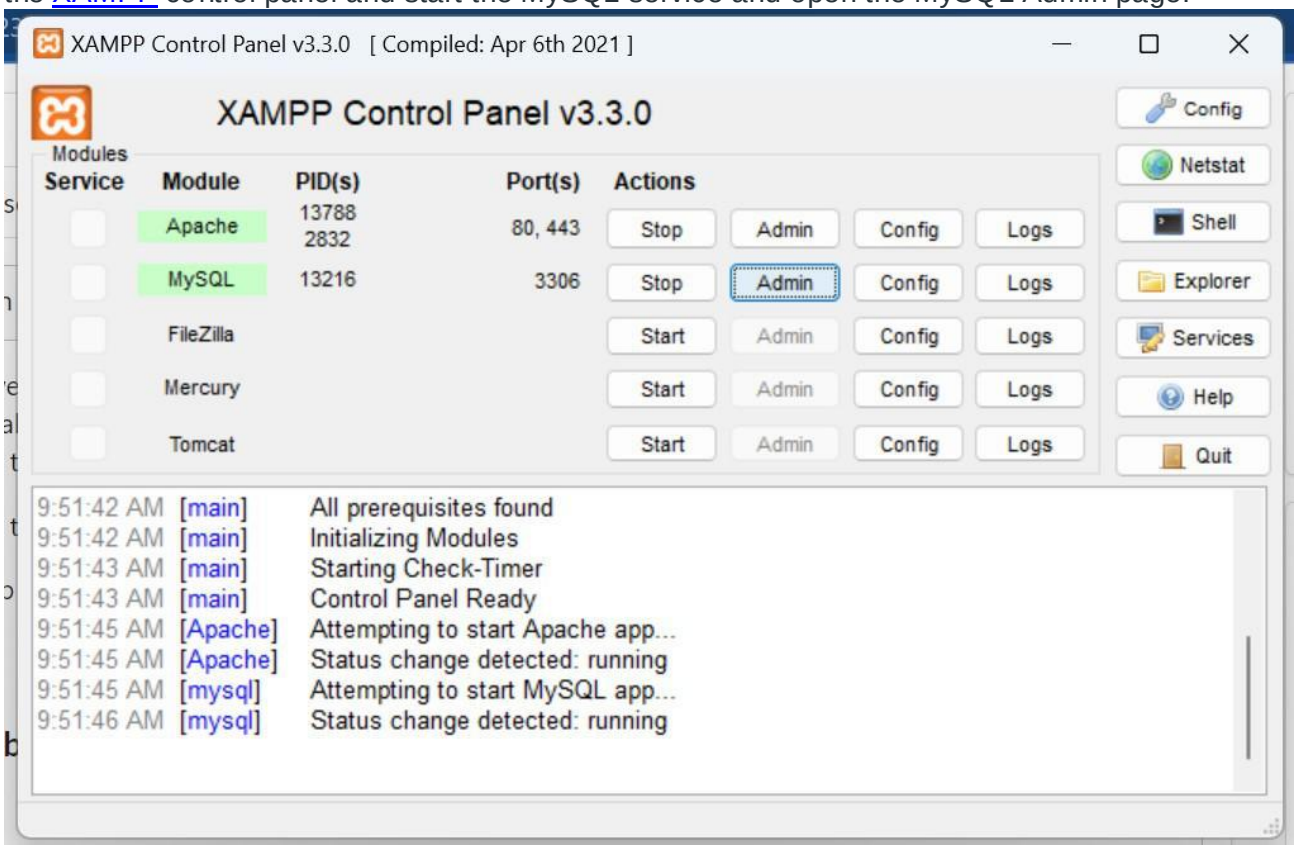
```
    // place error code here
  });
```

The advantage of $.post is it does not require as much connection configuration to be declared.

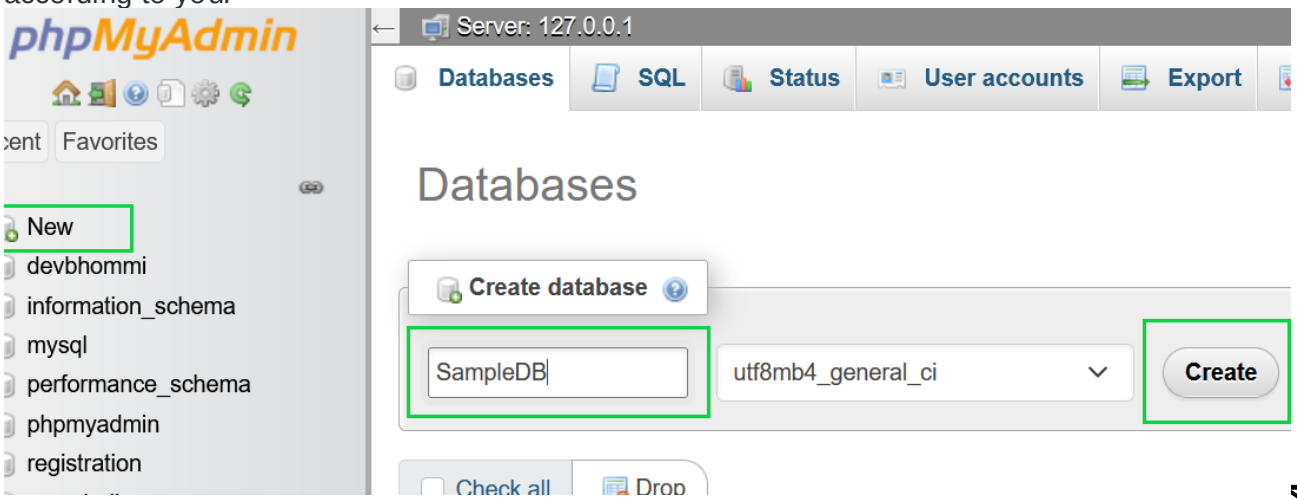
Ex: 11

AJAX Database Operations

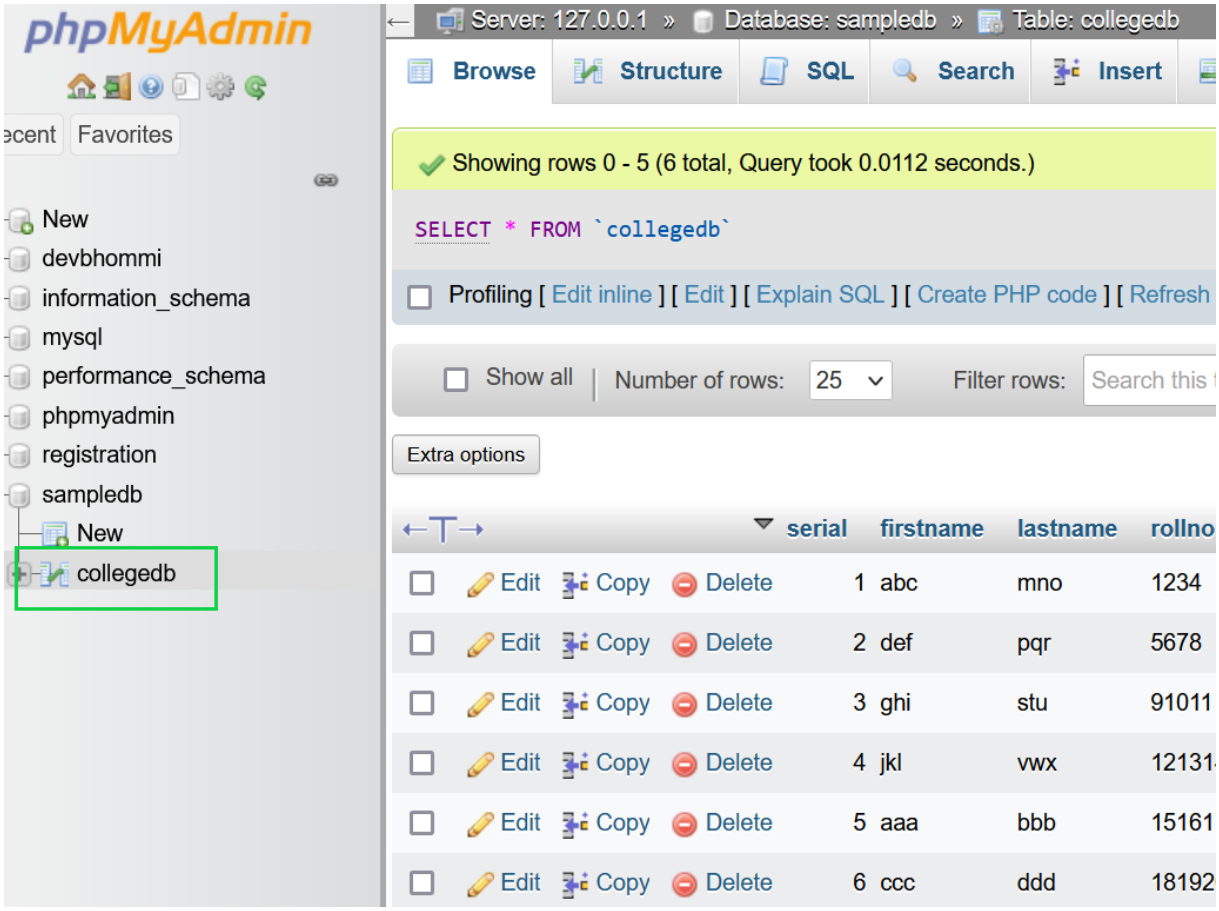
To perform this task we will create some sample entries in our MySQL database. Start the XAMPP control panel and start the MySQL service and open the MySQL Admin page.



MySQL Database: Click on new, create a new database, and name the database according to you.



Create a new table within the database.



Insert the following entries into the table.

Create the client-side PHP file which contains the AJAX request object and displays the result.

Client Side:

Index.html : We have created a Student Database that contains the details of the student. We want to display this data based on the condition that the student's CGPA in DB should be equal to the user-entered CGPA input.

- In this file, we have used basic HTML tags and created a form that accepts the user inputs and one submit button.
- When the user click submit button the JavaScript function **ajax_fun()** will be called. This function contains the required AJAX functions.
- The first line **const ajax_Request = new XMLHttpRequest();** will create an AJAX HTTP request object.
- If AJAX requests status will be OK or 200. we are just changing the HTML content of the HTML div element with id “result-box”

Index.html

```
<!DOCTYPE html>
<html>
<head>
   <style>
      input {
         padding: 0.5rem 2rem;
         margin-top: 10px; /* Added margin for better spacing */
      }
      label {
         font-size: 1rem;
         color: black;
      }
      .form-group {
         margin-bottom: 15px;
      }
   </style>
</head>
<body>
   <h1 style="color:green;">AjAX example</h1>
   <h3>Ajax Database Operations</h3>
   <hr><br>

   <!-- Form Section -->
   <div class="form-group">
      <label for="cgpa_val">Enter the CGPA: </label>
      <input type="text" id="cgpa_val" name="stu_cgpa">
      <input type="button" onclick="ajax_fun()" value="Submit">
   </div>

   <br>
   <hr>

   <!-- Result Section -->
   <div id="result-box">Result will display here</div>

   <script language="javascript" type="text/javascript">
      function ajax_fun() {
         var str = document.getElementById('cgpa_val').value;
```

```
        // Input validation to check if CGPA is provided
        if (str === "") {
            document.getElementById("result-box").innerHTML = "Please enter a valid
CGPA.";
            return;  // Exit if input is empty
        }

        const ajax_Request = new XMLHttpRequest();
        ajax_Request.onreadystatechange = function() {
            if (ajax_Request.readyState == 4 && this.status == 200) {
                document.getElementById("result-box").innerHTML =
ajax_Request.responseText;
            }
        };

        console.log("CGPA passed to server: " + str);  // Debugging the CGPA value

        // Send the request to the server with the CGPA value
        ajax_Request.open("GET", "index.php?q=" + str, true);
        ajax_Request.send();
    }
  </script>
</body>
</html>
```

Server Side:

index.php: We will create the server-side PHP file that will make a connection with the database to get the responses and will send the responses to the AJAX request object.

- We are creating the Database Connection using the following PHP function.
  - <?php
  - $dbhost = "localhost";
  - $dbuser = "root";
  - $dbpass = "root";
  - $dbname = "sample";
  - // Establish the connection
  - $con = mysqli_connect($dbhost, $dbuser, $dbpass, $dbname);
  - // Check if the connection is successful
  - if ($con->connect_error) {
  - exit('Could not connect to the database: ' . $con->connect_error);
  - }
  - // Check if the 'q' parameter is set and not empty
  - if (isset($_GET['q']) && !empty($_GET['q'])) {
  - $cgpa = $_GET['q']; // Getting the 'q' parameter from the URL

```
    echo "Received CGPA: " . htmlspecialchars($cgpa) . "<br>";  // Debugging: check the received value

    // Prepare a SQL query to avoid SQL injection
    $stmt = $con->prepare("SELECT * FROM student WHERE cgpa = ?");
    $stmt->bind_param("s", $cgpa);  // Binding the parameter (as string type)
    $stmt->execute();
    $result = $stmt->get_result();

    // Check if any rows are returned
    if ($result && $result->num_rows > 0) {
        echo "<table border='1' style='border-collapse:collapse;'>
                <tr>
                  <th style='padding:10px;'>FirstName</th>
                  <th style='padding:10px;'>LastName</th>
                  <th style='padding:10px;'>Rollno</th>
                  <th style='padding:10px;'>CGPA</th>
                </tr>";

        while ($row = $result->fetch_assoc()) {
            echo "<tr>
                  <td style='padding:10px;'>" . $row['firstname'] . "</td>
                  <td style='padding:10px;'>" . $row['lastname'] . "</td>
                  <td style='padding:10px;'>" . $row['rollno'] . "</td>
                  <td style='padding:10px;'>" . $row['cgpa'] . "</td>
                 </tr>";
        }

        echo "</table>";
    } else {
        echo "No records found for the CGPA: " . htmlspecialchars($cgpa);
    }

    // Close the prepared statement
    $stmt->close();
} else {
    echo "Please provide a valid CGPA.";
}

// Close the database connection
mysqli_close($con);
?>
```

Ouput

AjAX example

Ajax Database Operations

Enter the CGPA:

Received CGPA: 8

FirstName	LastName	Rollno	CGPA
aa	bb	2	8