The Unix Command Line

Select the directory you want to upload.

* To create a folder in command line –> mkdir(make directory) Music(folder name).
* To back from the folder –> /cd ..
* To quickly open the terminal in the vs code use –> ctrl `
* If to enter the root directory --> cd ~
* To clear the full terminal use --> clear
* To create a new file in a folder use --> for example we have to create a text file --- (touch Text2.txt).
* To directly open the file in the use 🡪> open image.txt
* To delete files in the folder use 🡪 rm filename.rtf
* To delete all files in the folder at once use 🡪 (rm \*)

# Link for the more command line keyewords <https://www.learnenough.com/command-line-tutorial>

Node.JS

Node provides a JS runtime environment on any computer. It is not a framework. By the help of Node.js we can run our JS file.

It allows us to create our website back-end.

* To check version of node.js – node -v
* To run command written in the JS file using the Node.js we first enter in the directory using the cd.

Node Module-

<https://nodejs.org/docs/latest-v18.x/api/index.html>

* In the 2.2 Native module folder we created a JS folder in which we write a command in Node.js for creating a file and Also the message we write in the message.txt file in the same folder.
* To run the command written in the JS file we first enter into the file directory and write the -- node filename into the terminal.

Example—node index.js

# **npm(Node Package manager)**

* Open the folder directory in the terminal and type –> (npm init -y)command.

It will initialise command for making of the package.json file.

To install Packages of npm - <https://www.npmjs.com/>

* To install npm package in the folder/directory – npm install package\_name
* Then from the documentation write the code EX- 2.3+NPM-folder name.
* Require function is used in the commonJS(CJS)
* import function is used in the ECMAScript(EJS), It provides the same functionality as the commonJS function.
* import xxx from 'superheroes'; --- The string name should be same as the name in the package.json file. Ex- superheroes

# Express.js with Node.js

Express.js is a framework of JS. It allows us to create our website back-end.

Creating an Express server

* Create Directory
* Create index.js file
* Initialise NPM using (npm init -y).
* Install the express package using (npm I express)
* Write Server application in index.js
* Start server.

Then change the “type” in the package.json file to module

Using “type”:”module”.

* We **can not run** our local server just by node index.js file , by this we can only check that our file is executed successfully or not.
* netstat -ano |findstr "LISTENING". By using this command we can find which ports(server) are open in our computer at that time.
* ***We can stop our server / clear our output terminal that is generated after the server request by – (ctrl + c).***

HTTP requests

The purpose of our http is to communicate our client computer to the server side.

* When we make these http request , there are five main words come –

These are:-

* GET – (Getting something from the computer or send request of taking getting resource to the server)
* POST – (Server sending resource )
* PUT –(Replace resource)
* PATCH – (Patch up a resource means for example we can replace the resource just by ourself).
* DELETE –(Delete resource ).

Note: There is a package called (nodemon) by which we can restart our server automatically. By this when we change something in our index.js file then it automatically restart our server.

* For this we can use (nodemon index.js) instead of node index.js.

# **PoSTMAN**

HTTP response status codes includes whether a specific HTTP request has been successfully completed.   
Some grouped responses are:

* Informational responses(100 - 199) –> Hole on
* Successful responses (200-299) 🡪 Here you go
* Redirection messages(300- 399) 🡪 Go away
* Client error responses(400 -499) 🡪 You fucked up
* Server error responses(500-599) 🡪 I fucked up

MDN Docs fro Server INFO—

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

After this we done a exercise in the POSTMAN folder for this we need a POSTMAN app in which we can run our localhost’s and there we can directly get our results. GET, POST, PUT, DELETE,PATCH.

# **Middelwares**

Middle wares are basically acts as the middle man between the server and the user. It uses in ***Authentication , accepting cookies , Pre- Processing, Error, Logging -in***.

1. We have to install the npm package called body-parser. It helps in submitting the details of the form which is created. Like the email id and the password.
2. https://www.npmjs.com/package/body-parser   
   npm I body-parser.

* We have to use POST request b/c it sends the request/ resource from the server to the client.

Example – see it in the Middleware named folder saved in the PC.

Some Middlewares Explanation 🡪

1. import express from 'express';

Imports the Express framework, which simplifies the process of building a web server in Node.js.

1. import morgan from 'morgan';

Imports Morgan, a middleware for logging HTTP requests. This is useful for debugging and monitoring your app.

1. import bodyParser from 'body-parser';

Imports the Body-Parser middleware, which parses incoming request bodies (e.g., form submissions).

1. import { dirname } from "path";

Imports the dirname function from the Path module, which helps in getting the directory name of the current file.

1. import { fileURLToPath } from "url";

Imports fileURLToPath, which is used to convert a file:// URL into a path string (useful in ES Modules).

# **EJS(Embedded JS)**

We can write ejs file using the (**.ejs**) extension.

First we have to install npm package of ejs using 🡪 npm I ejs.

In ejs file we code the same html and then integrate the JS using the **<%= %>.** In JS file instead of app.send 🡪 app.sender(index.ejs,{ }) ;

This sends commands to ejs file and then prints the code written into it.

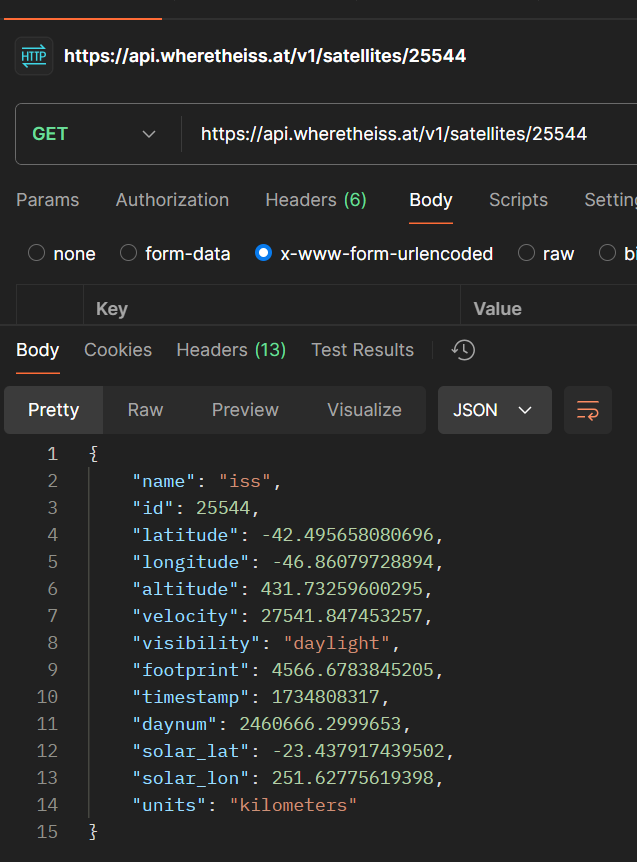
***🡪 Ejs Tags*** -- <https://github.com/mde/ejs/blob/main/docs/syntax.md>

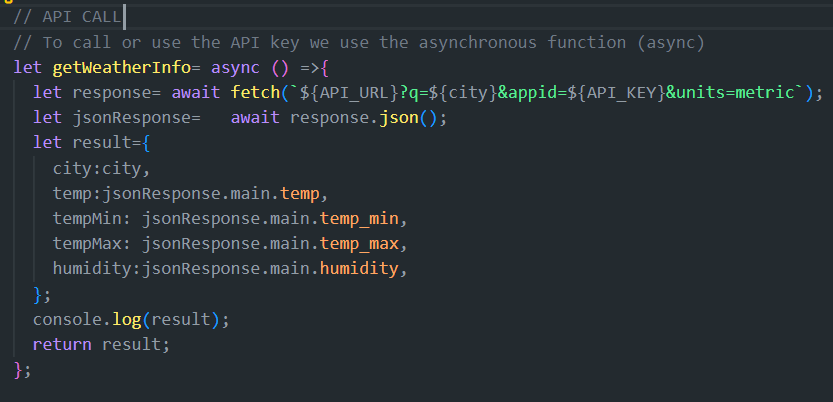
Code Example refer – 4.1Ejs

# **API(Application Programming Interface)**

# 

# REST API --

WE use GET, POST, PUT, PATCH , DELETE services for using the rest api’s.  
  
example of the –(where is the location of the iss space station is now) <https://wheretheiss.at/w/developer>  
You can simply write the GET http request and get the coordinates.

**API CALL**

# **Git And Github**

1. git init 🡪 it initialises a git file in the file directory.
2. git add file\_name🡪 It add the file into the git.  
    a. To add multiple files at a single time we can use 🡪 (git add .)
3. To remove the added file we use 🡪 git rm –cached -r file\_name
4. To commit the file into the git 🡪 git commit -m “Commit message”
5. git log 🡪 to see the details of the commit made.
6. To revert the changes made in the file we can use a command 🡪

( git checkout file\_name) .

7.We can also check that what is the difference between the last commit and the initial change in the file 🡪 git diff file\_name

8. To hide some file for being committing to github we simply add these file\_name into the (**.gitignore**) file and then push gitignore file into github .

9. <https://github.com/github/gitignore/blob/main/Node.gitignore>  
from this link paste the code into the .gitignore file and these lines of code ignore all the user related data to push into the github so that our credentials become safe.

10. To clone a repository we use🡪 git clone link  
 or we can simply download that file.  
11. Some basic open source beginner friendly projects 🡪 <https://github.com/MunGell/awesome-for-beginners>

**Branching And Merging**

1. To add a branch in the git file🡪 git branch new\_branch\_name
2. To check for the branches available 🡪 git branch  
   it will show all the available branches present
3. To switch the branch from main branch 🡪 git checkout branch\_name

**To merge the branches** 🡪 git merge branch\_name

After that on the CLI type (q!)

# **REACT**

**Component**- Individual reusable pieces of code, which we can reuse in our development further whenever we want to build the same thing as the component.  
Example - we build a button using React then we can insert the same button in our page when we want.

**JSX – (JavaScript Extension Syntax)**It Lets us write HTML directly inside JS.

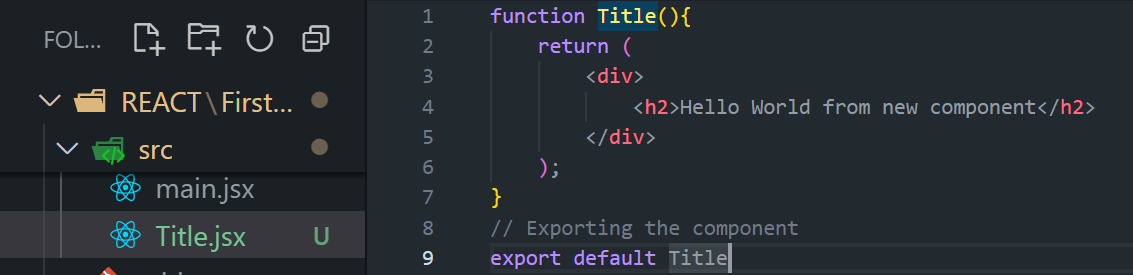
**Setup React.js –**

First Install Vite in CLI --- npm create vite@latest  
Then choose the framework—like   
-> Vanilla  
->Vue  
->React etc.

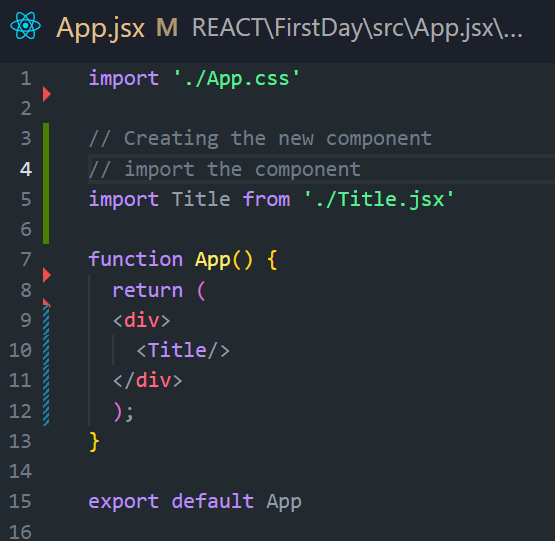
Then Enter in that folder and type (npm install) it will automatically install all the packages inside the vite.

**(npm run dev )** To start the server.

**Creating the Component:**

To make a new component we first creating the new file with that component name . for example 🡪 Title component   


* Then we have to import that new component in the App.jsx file



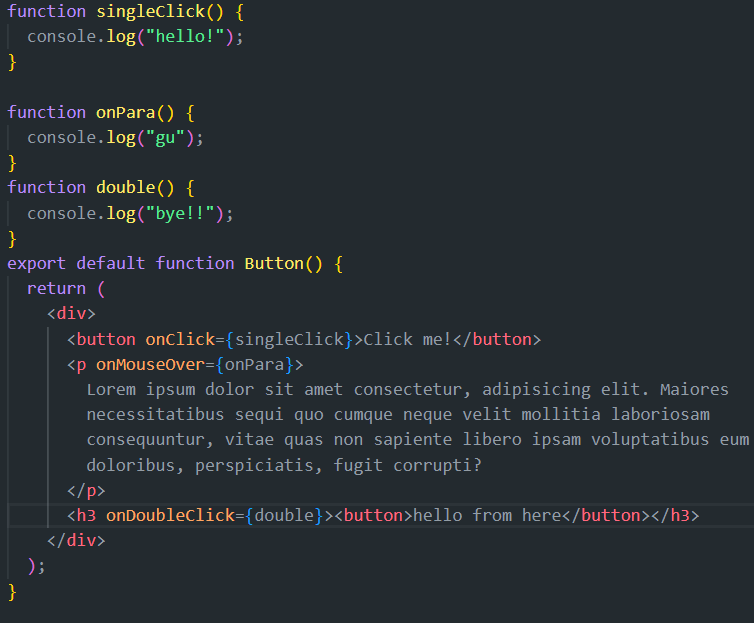
* We can return only one root element at a time   
  if we have to return multiple tags then put them into the <div></div> tag .

**Writing Markup in JSX**

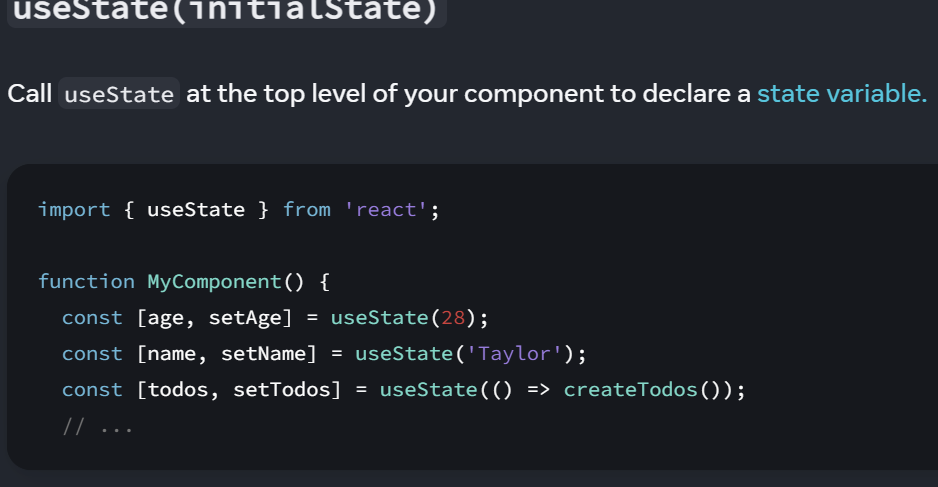
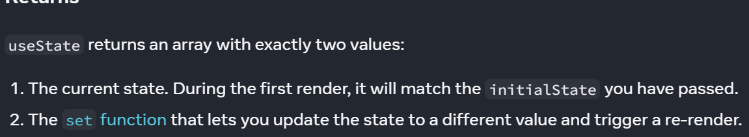
1. Return a single root element.
2. Close all the tags.
3. camelCase most of the things.

* **We have to import the file for their use**

**Click Events**

onClick{},onMouseOver{}, onDoubleClick{} etc..

Use the “on” keyword in stating of the events.

 **USESTATE**<https://react.dev/reference/react/useState#usage>

In the to do project we have to define the unique key for each new data we insert so that it can access all the inserted values . so for this we have to use (uuid)🡪

<https://www.npmjs.com/package/uuid>

**ARRAYS**

We can use the arrays in the

[**https://react.dev/learn/updating-arrays-in-state**](https://react.dev/learn/updating-arrays-in-state)

**Material UI**

Basically it is the CSS for the react which we used ..

INSTALLATION🡪<https://mui.com/material-ui/getting-started/installation/>

MINI Project 🡪 Weather Forecast widget.