**HTTP**

Hyper Text Transfer protocol

Common language for communication between client and server

Communicate to outside world through request and response

Browser – HTTP client sending request

GET , POST ,PUT ,DELETE – just four words in this language

GET – get html file from server

POST – add data in to server

PUT – update data into server

DELETE – delete data in the backend

Can also be used to fetch documents, photos, videos etc. (Communicate with servers)

Response -> messages and documents // look for different kinds of messages in w3schools

Sending data to the server:

through

1. query strings/**GET** request(key and values)

[file:///F:/gitrepo/http/RegistrationForm.html**?firstname=&lastname=&psw=&**](file:///F:/gitrepo/http/RegistrationForm.html?firstname=&lastname=&psw=&)**email=&submit=register**

1. body / **POST** request

Information attached to form body

HTTPS:Hyper Text Transfer Protocol Secured: data/information in some other form

**JSON**

When exchanging data between browser and server, the data can only be text.

Server can be of any language

JSON -> JavaScript Object Notation

Syntax for storing and exchanging data (also xml can be used but outdated)

JSON is text, written with JavaScript object notation

Json can be read by any language

Working:

JSON.stringify(user); // user is a js object

Converted into json string and it is sent to server where it uses JSON.parse(user);

JSON.parse(money) // money – data from server

Server returns data that is parsed to get the data.

**AJAX**

Ajax is a technology that allows you to read from web server after the webpage is loaded and update a webpage without reloading the page and finally send the data in the background while the user is interacting with the website

In react : using ***fetch***

fetch(‘/my/ url) .then(response => {

console.log(response);

});

*Code returns promise. if it returns promise , use ‘then’ function*

Ajax – change content dynamically

fetch('https://jsonplaceholder.typicode.com/users').then(response =>response.json()).then(data => console.log(data));

ajax call – fetch here

Ajax – make single page application using ajax .

ajax is a combination of tools – using fetch api , using http , using json to communicate with servers

**Promise**

A promise is an object that may produce a single value sometime in the future, either a resolved value or a reason that it’s not resolved(rejected)

Promise states: 1) fulfilled or 2) rejected or 3) pending

const promise = new Promise((resolve,reject) =>{

if(true){

resolve("Stuff Works");

}

else{

reject("not working !");

}

});

//promise.then(result => console.log(result));

promise

.then(resp => resp + "!")

.then(resp => resp + "?")

.then(resp => resp +"#")

.then(resp => console.log(resp));

promise

.then(response=>response +"!")

.then(response=> response +"?")

.then(response=>{

throw Error;

console.log("response")

})

.catch(()=>console.log("error"));

Promise are like event listeners except promise can only succeed or fail once, it cannot succeed or fail twice.

Useful for Asynchronous success or failure ----like api call

Reacting to something that happens asynchronously

Reacting to something that happens asynchronously

promise.then(response=>response + "!!!!" )

.then(response=>{

throw Error;

response + "is this ";

})

.then(response=>response +"?")

.then(response=>console.log(response))

.catch(()=>console.log("error"));

const promise2= new Promise((resolve,reject)=>{

*setTimeout(resolve,100,"hii");*

})

const promise3 = new Promise((resolve,reject)=>{

setTimeout(resolve,1000,"after some time");

})

const promise4 = new Promise((resolve,reject)=>{

setTimeout(resolve,3000,"phoooo, waited too long");

})

Promise.all([promise,promise2,promise3,promise4]).then(values=>console.log(values));

const urls = [

'https://jsonplaceholder.typicode.com/users',

'https://jsonplaceholder.typicode.com/albums',

'https://jsonplaceholder.typicode.com/posts',

]

Promise.all(urls.map(url=>{

return fetch(url).then(response=>response.json());

})).then(results=>{

console.log(results[0]);

console.log(results[1]);

console.log(results[2]);

}).catch(()=>console.log(‘error’));

Extra;

promise.then(console.log);// promise.then(resp=>console.log(resp));

const promise = new Promise((resolve, reject) => {

setTimeout(() => {

resolve("success");

}, 4000)

});

const promise = Promise.resolve(

setTimeout(() => {

console.log("success");

}, 4000)

);

Promise.reject('failed')

.catch(console.log('Ooops something went wrong'))

Async Await

Instead of promise, async await can be used. It looks better. ES8

Using promise:

fetch()

.then(response => response.json())

.then(console.log)

Using async function :

const asyncTest =async function () {

const fetchdata =await fetch('https://swapi.co/api/starships/9/');

const data = await fetchdata.json();

console.log(data);

}

syncTest();

const urls = [

'https://jsonplaceholder.typicode.com/users',

'https://jsonplaceholder.typicode.com/posts',

'https://jsonplaceholder.typicode.com/albums'

]

const getData = async function() {

const [ users, posts, albums ] = await Promise.all(urls.map(async function (url){

const userurl=await fetch(url);

const data =await userurl.json(); // or return userurl.json();

return data; //

}

));

console.log('users', users);

console.log('posta', posts);

console.log('albums', albums);

}

Try catch:

const getData = async function() {

try {

const [ users, posts, albums ] = await Promise.all(urls.map(async function(url) {

const response = await fetch(url);

return response.json();

}));

console.log('users', users);

console.log('posta', posts);

console.log('albums', albums);

} catch (err) {

console.log('ooooooops', err);

}

}