**ساخت متغییر**

در جاوا اسکریپت سه روش برای ساخت متغییر ها وجود دارد:

* روش اول:

استفاده از کلمه کلیدی let در ابتدا و سپس نام گذاری آن و در انتها مقدار دهی آن. در این روش امکان تغییر مقدار وجود دارد.

let username = "reza"

* روش دوم:

استفاده از کلمه کلیدی const به جای let. در این روش امکان تغییر مقدار وجود ندارد و مقدار ثابت است.

const username = "reza"

* روش سوم:

استفاده از کلمه کلیدی var. در این روش مقدار ثابت است ولی متغییر گلوبال است.

var username = "reza"

**انواع متغییر ها در js**

string – number – Boolean – undefined – null

let username = "reza" string

let age = 17 number

let is\_admin = true Boolean

let country; undefined

**دستور alert ,prompt ,confirm**

* با استفاده از دستور alert میتوانیم باکس هشداری را به کاربر نمایش دهیم.

alert("hello world")

* با استفاده از دستور prompt میتوانیم یک مقدار را از کاربر دریافت کنیم.

prompt("enter your name:")

* دستور confirm یک سوال دو وضعیتی از کاربر میپرسد که میتواند خروجی آن ok یا cancel باشد.

confirm("are you ok?")

**Js operator**

اپراتور ها در جاوا اسکریپت به سه دسته تقسیم میشوند:

* Arthemic : + , - , \* , \*\* , / , % , ++ , --
* Assigment : = , += , -= , \*= , /= , %=
* Compersion : == , === , != , !== , < , > , <= , >=

**شرط ها در js**

شرط ها برای چک کردن یک یا چند متغییر مورد استفاده قرار میگیرد که خروجی آن چیزی جز true,false نیست.

برای مثال:

const username = "reza"

if(username == "reza")

{

    console.log("welcome reza")

}

else if(username == "ali")

{

    console.log("welcome ali")

}

else

{

    console.log("access denied")

}

در مثال بالا گفته شده است اگر مقدار متغییر username برابر با reza بود پیغامی با محتوای خوش آمد گویی به رضا نمایش داده شود و در غیر اینصورت اگر مقدار username برابر با ali بود پیامی با محتوای خوش آمد گویی به علی چاپ شود و در غیر این صورت پیغامی با محتوای عدم دسترسی نمایش داده شود.

**حلقه while**

برای تکرار یک کار خاص مورد استفاده قرار میگیرد و با کمک یک متغییر بیرونی و یک

شرط داخلی تعداد گام های این حلقه را کنترل میکنیم:

let index = 0

While(index >= 10)

{

Console.log(index)

Index++

}

کد بالا اعداد 0 تا 10 را برای ما به نمایش میگذارد.

**حلقه for**

حلقه ای همانند حلقه while است با این تفاوت که از یک متغییر داخلی برای کنترل گام شمار استفاده میشود:

for (let index = 0; index <= 10; index++) {

    console.log(index)

}

کد بالا اعداد 0 تا 10 را نمایش میدهد.

**Objects**

متغییر هایی هستند که از دو بخش key , value تشکیل شده اند.

let user = {

    name : "reza",

    family : "derakhshan",

    country : "iran",

    age : 17

}

**Format strings**

یکی از روش های text formatting در جاوا اسکریپت است. برای استفاده از این روش به صورت زیر عمل میکنیم:

let username = "reza"

let lastname = "derakhshan"

console.log(`welcome ${username} ${lastname}`)

**حلقه زدن رو یک object**

در این روش از حلقه ی for و کلمه کلیدی in استفاده میکنیم:

let user = {

    name : "reza",

    family : "derakhshan",

    country : "iran",

    age : 17

}

for (let i in user) {

    console.log(`${i} : ${user[i]}`)}

**آرایه ها در js**

آرایه ها لیستی از مقدار ها هستند . این میتواند شامل هر نوع متغییری باشید از string گرفته تا Boolean.

let user = ['reza','ali','mohsen']

حلقه زدن رو یک آرایه

در این روش همانند حلقه زدن بر روی object عمل میکنیم با این تفاوت که این بار از کلمه ی کلیدی of استفاده میکنیم:

let user = ['reza','ali','mohsen']

for (let i of user) {

    console.log(i)

}

**Js switch**

این دستور یکی از دستورات شرطی js است و همچنین راه حلی مناسب برای شرط های تو درتو است.برای نوشتن switch دو روش وجود دارد:

روش اول:

let username = "reza"

switch(username) {

  case "ali":

    console.log("welcome ali")

    break;

  case "reza":

    console.log("welcome reza")

    break;

  default:

    console.log("access denied")

}

Exected output : welcome reza

روش دوم:

switch (true) {

  case username == "ali":

    console.log("welcome ali")

    break;

  case username == "reza":

    console.log("welocme reza")

break;

  default:

    console.log("access denied")

    break;

}

Extected output : welcome reza

**Js functions**

تابع ها مجموعه ای از کد هایی هستند که شما هر زمان که خواستید میتوانید به آن دسترسی داشته باشید.تابع ها به صورت function hoisting هستند یعنی هر جا که صدا بزنیمشون کار میکنند.تابع ها زمانی اجرا میشواند که آن را صدا بزنیم.3 نوع فانکشن داریم:

* Function declration:

function myFunction(carname) {

  console.log(carname)

}

myFunction("Volvo")

* Function expression:

var myFunction = function(carname) {

  console.log(carname)

}

myFunction("Volvo")

* Arrow functions

var myFunction = (carname) => {

console.log(carname)

}

myFunction("Volvo")

**Parameter and Argument in function**

در جاوا اسکریپت پارامتر ها همان ورودی های تابع ها هستند و آرگومانت ها مقادیری هستند که هنگام صدا کردن تابع میدهیم.

**Object methods**

به فانکشن هایی که درون آبجکت ها به عنوان key و value تعریف میشوند متود گفته میشود.

const person = {

    firstName: "John",

    lastName: "Doe",

    id: 5566,

    fullName: function() {

      return *this*.firstName + " " + *this*.lastName;

    }

};

**Dom**

مخفف (documents object model) که معنای دسترسی به فایل html و جست و جو در آن است.در Dom تمام عناصر html به عنوان یک آبجکت تعریف میشوند و برای دسترسی به آن ها میتوانیم از روش های زیر استفاده بکنیم:

* جست و جو با کمک id

let find = document.getElementById("id")

* جست و جو با کمک tag name

let find = document.getElementsByTagName("name")

* جست و جو با کمک نام class

let find = document.getElementsByClassName("name")

* جست و جوی کلی

let find = document.querySelectorAll("p.intro");

*یک لیست شامل تمام تگ ها <p> که کلاس intro دارند رو برمیگردونه*

**Class list**

در مبحث قبلی یاد گرفتیم که چجوری به tag های html دسترسی پیدا کنیم حالا در این مبحث یاد میگیرم چجوری بر روی آنها تغییراتی اعمال بکنیم.

با استفاده از classlist میتوانیم به لیست کلاس های تگ مورد نظر دسترسی داشته باشیم و تغییراتی مثل add,remove,toggle,contains بر روی آنها اعمال کنیم.

* Add

یک کلاس جدید به تگ اضافه میکند.

find.*classList*.add("test")

* Remove

کلاس مد نظر را پاک میکند.

find.*classList*.remove("test")

* Toggle

چک میکند اگر کلاس وجود داشته باشد حذفش میکند و اگر وجود نداشته باشد اضافه میکند.

find.*classList*.toggle("test")

* Contains

یک دستوری شرطی است و چک میکند که ایا کلاسی با این نام وجود دارد یا خیر.

find.*classList*.contains("test")

* Replace

دو مقدار دریافت میکند . مقدار اول نام کلاس و مقدار دوم نام کلاس جایگزین.

find.*classList*.replace("test","blah")

**JavaScript Events**

این مبحث به شما کمک میکند هنگامی که یک رویداد خاص مانند کلیک کردن اتفاق میفتد بتوانید بر روی tag مورد نظر تغییراتی اعمال بکنید.

let btn = document.getElementsByTagName("button")[0]

btn.addEventListener("click", () =>{

    alert("hello world")

})

در این کد هنگامی که بر روی دکمه کلیک شود پیغام hello world به کاربر نمایش داده میشود.

**Form Events in JavaScript**

قبلا به رویداد click اشاره کرده بودیم حالا میخوایم چهار رویداد submit,change,input,blur رو با هم بررسی کنیم.این چهار رویداد برای form,input ها هستند.

const formElement = document.getElementById('myForm');

برای مثال ما میخواهیم رویداد های form بالا رو کنترل کنیم:

* Change

این رویداد زمانی رخ میدهد که مقدار input های داخل form تغییر کنید:

formElement.addEventListener('change', function(event) {

    event.preventDefault()

    console.log('new value: ', event.target.name.value);

});

نکته: رویداد های مربوط به form حتما یک ورودی به arrow function انها داده میشود که با استفاده از اون میتوان به مقدار ها دسترسی پیدا کرد.

نکته: برای دسترسی راحت تر, بهتر است که به input های داخل form یک name بدهیم برای مثال:

<form action="" method="post">

    <input type="text" name="username">

     <input type="submit" value="GO">

</form>

نکته: برای ارسال داده ها و چک کردن آن حتما باید از یک input یا button از نوع submit استفاده بکنید.

* Submit

برای وقتی است که کاربر یک form را ارسال میکند :

formElement.addEventListener('submit', function(event) {

    event.preventDefault();

    console.log('form sent');

});

نکته: اگر دقت کنید در کد بالا از متودی به نام preventDefault استفاده شده است.حالا این چیه و چیکار میکنه; اگر دقت کرده باشید بعد از کلیک کردن بر رو روی دکمه و ارسال فرم صفحه رفرش میشه و این موضوع باعث میشه که ما به مقدار داخل فرم دسترسی نداشته باشیم.این متود میاد از رفرش شدن صفحه جلوگیری میکنه.

* Input

رویدادی شبیه به change است ولی با یک تفاوت: در رویداد change بعد از اینکه کاربر از input خارج شد رویداد اجرا میشود ولی در رویداد input بلافاصله بعد از تغییر مقدار رویداد اجرا میشود:

formElement.addEventListener('input', function(event) {

    console.log('the value is change:', event.*target*.name.value);

});

* Blur

این رویداد زمانی رخ میدهد که input از حالت فوکوس خارج شود معمولا برای اعتبار سنجی فرم ها مورد استفاده قرار میگیرد:

inputElement.addEventListener('blur', function(event) {

    console.log('The field is out of focus');

});

نکته:در مثال بالا از یک input استفاده شده است نه یک form .

**Array Methods**

در این مبحث قراره چند تا از متود های آرایه رو با هم دیگه بررسی کنیم.

const plants = ['broccoli', 'cauliflower', 'cabbage', 'kale', 'tomato'];

این لیست رو ذر نظر داشته باشید.

* Pop

اخرین عضو آرایه را پاک میکند

plants.pop()

* Push

یک عضو جدید به آخر آرایه اضافه میکند:

plants.push(kale);

* Shift

بدونه از اول پاک میکنه

plants.shift();

* Unshift

یک عضو جدید به اول آرایه اضافه میکند:

plants.unshift("Kale");

* Length

طول آرایه رو برمیگردونه

plants.length();

* Contact

دوتا آرایه رو با هم ترکیب میکنه

const array1 = ['a', 'b', 'c'];

const array2 = ['d', 'e', 'f'];

const array3 = array1.concat(array2);

* Splice

سه تا ورودی میگیره ورودی اول index ورودی دوم گام شمار و در آخر مقدار جدید.

حالا چیکار میکنه میاد ایندکسی که بهش دادیو پیدا میکنه به اندازه تعداد گام هایی که بهش گفتی حرکت میکنه و مقدار های جدید رو حذف یا جایگزین میکنه.

plants.splice(4, 1, 'some plant');

* Slice

دو تا ایندکس بهش میدی و اون میاد از شروع تا یکی قبل از پایان رو بهت برمیگردونه.

plants.slice(2, 4)

Extected output : cabbage , kale

* Find

این متود به صورت arrow function مورد استفاده قرار میگیرد که یک شرط میگیرد و در داخل لیست شرط رو بر روی تک تک مقدار ها انجام میده و اولین مقداری که شرط رو true کنه رو برمیگردونه

const array1 = [5, 12, 8, 130, 44];

const found = array1.find((element) => element > 10);

console.log(found);

Expected output: 12

* Findindex

دقیقا همون بالایی است اما با این تفاوت که بجای مقدار index به ما برمیگردونه

const found = array1.findIndex((element) => element > 10);

Expected output: 1

* Includes

یک متودی هست که خروجیش یا true است یا false و میاد چک مکینه ببینه که مقداری که بهش دادی وجود دارد یا خیر.

const found = array1.includes(5)

Expected output: true

* Map

به صورت arrow function نوشته میشود به به ما کمک میکند تا یک عملیات ریاضی بر روی تمام مقدار ها انجام دهیم. برای مثال در کد زیر تمام اعضای ارایه رو دو برابر کرده ایم.

const found = array1.map((x) => x \* 2)

Expected output: 10 , 24 , 16 , 260 , 88

* Filter

دقیقا همون find است ولی با یک فرق!! اگر یادتون باشه find فقط اولین مقداری که شرط رو true میکرد رو برمیگردند ولی filter میاد تمامی مقادیری که شرط رو true کرده اند رو برمگیردونه.

const array1 = [5, 12, 8, 130, 44];

const found = array1.find((element) => element > 10);

console.log(found);

Expected output: 12 , 130 , 44

* Reverse

آرایه رو برعکس میکنه

array1.reverse()

* Foreach

میاد رو یه لیست یک حلقه for میزنه مثلا تو کد زیر میاد تک تک اعضای آرایه رو لاگ میگیره.

const array1 = ['a', 'b', 'c'];

array1.forEach((element) => console.log(element));

**import-export**

فرض کنید که دو فایل با نام های app.js,main.js دارید میخواید از محتوای داخل فایل اول در فایل دوم استفاده کنید; برای انجام این کار باید از export و import استفاده کنید.

نکته در فایل html لینکدهی جاوا اسکریپت باید به صورت زیر باشد:

<script type="module" src="js/app.js"></script>

main.js:

export function myFunction(name) {

    console.log(`Welcome ${name}`)

}

app.js:

import { myFunction } from "./main.js";

myFunction("reza")

Expected output : Welcome reza

**String Methods**

* Charat

یک ایندکس بهش میدی و اون میاد مقداری که در اون ایندکس وجود داره رو برمیگردونه

const sentence = 'The quick brown fox jumps over the lazy dog.';

const index = 4;

console.log(`The character at index ${index} is ${sentence.charAt(index)}`);

Expected output: "The character at index 4 is q"

* Charcodeat

کد اسکی اون ایندکسی که انتخاب کردی رو میده

const sentence = 'The quick brown fox jumps over the lazy dog.';

const index = 4;

console.log(

  `Character code ${sentence.charCodeAt(index)} is equal to ${sentence.charAt(

    index,

  )}`,

);

Expected output: "Character code 113 is equal to q"

* Trim

Whitespce هارو از متغییر حذف میکنه

const greeting = '   Hello world!   ';

console.log(greeting);

Expected output: "   Hello world!   ";

console.log(greeting.trim());

Expected output: "Hello world!";

* Length

طول یک رشته رو بهت برمیگردونه

const str = 'Life, the universe and everything. Answer:';

console.log(`${str} ${str.*length*}`);

Expected output: "Life, the universe and everything. Answer: 42"

* Concat

دوتا مقدار رو با هم یکی میکنه

const str1 = 'Hello';

const str2 = 'World';

console.log(str1.concat(' ', str2));

Expected output: "Hello World"

* To lower case

همانطور که از اسمش معلومه کله مقدار رو به حروف کوچک تقسیم میکنه

const dotted = 'ISTANBUL';

console.log(dotted.toLowerCase());

Expected output: "i̇stanbul"

* To upper case

دقیقا بر عکس متود بالایی است و بعید میدونم نیازی به مثال باشه .

* Slice

کارش جدا کردن متن است . بهش یک ایندکس میدی و از اونجا به بعد رو بهت برمیگردونه میتونی براش شروع و پایان هم در نظر بگیری.

const str = 'The quick brown fox jumps over the lazy dog.';

console.log(str.slice(31));

Expected output: "the lazy dog."

console.log(str.slice(4, 19));

Expected output: "quick brown fox"

console.log(str.slice(-4));

Expected output: "dog."

console.log(str.slice(-9, -5));

Expected output: "lazy"

* Split

با یک مثال براتون توضیح میدم :

فرض کنید یک رشته دارید که داخلش چندین کلمه هست و میخواهید تک تک این کلمات رو جدا کنید و داخل یک لیست ذخیره کنید و به کلمه چهارم داخل این لیست دسترسی پیدا کنید.

const str = 'The quick brown fox jumps over the lazy dog.';

const words = str.split(' ');

console.log(words[3]);

Expected output: "fox"

این کار رو به راحتی میتونید با split انجام بدید.

حالا فرض کنید میخواید تک تک حروف رو داخل یک لیست بریزید و در اخر به حرف نهم لیست دسترسی داشته باشید :

const str = 'The quick brown fox jumps over the lazy dog.';

const chars = str.split('');

console.log(chars[8]);

Expected output: "k"

**Number Methods**

* toString

همانطور که از اسمش مشخصه عدد رو داخل یک رشته برمیگردونه

console.log((10 \*\* 21.5).toString());

Exected output : "3.1622776601683794e+21"

* toFixed

کارش فیکس کردن عدده; میتونی براش مشخص کنی که تا چند عدد اعشار برات فیکس بکنه

نکته:حروجیش رشته است.

const numObj = 12345.6789;

numObj.toFixed();

Expected output : ‘12346’

numObj.toFixed(1);

Expected output : ‘12345.7'

numObj.toFixed(6);

Expected output : ‘*12345.678900'*

* isInteger

چک میکنه ببینه عدد یک عدد صحیح است یا نه.

Number.isInteger(1);

Expected output : *true*

Number.isInteger(0.1);

Expected output : *false*

* Math

همانطور که از اسمش مشخص محاسبات ریاضیات را برای ما انجام میدهد

در ادامه چند مثال ازش میبینیم:

* + Math.floor

رو به پایین رند میکنه

console.log(Math.floor(5.95));

Expected output: 5

* + Math.ceil

رو به بالا رند میکنه

console.log(Math.ceil(0.95));

Expected output: 1

* + Math.round

عددو رند میکنه; بعید میدونم نیازی به مثال داشته باشه

* + Math.trunc

اعشار رو حذف میکنه

console.log(Math.trunc(13.37));

Expected output: 13

**LocalStorage**

به کاربر ها این امکان را میدهد که داده های خود را در فضای ذخیره مرورگر خود نگه دارند.

همانند cookies و session ها ولی روش مطمئن تری نسبت به این دو است.

* setItem

برای ذخیره کردن داده ها استفاده میشود;شباهت زیادی به object دارد به صورتی که یک key و یک value میگیرد.

localStorage.setItem(key, value);

* getItem

وقتی میخوایم داده هارو بگیریم از این متود استفاده میکنیم.

let user = localStorage.getItem(key);

console.log(user)

* removeItem

از اسمش معلومه دیگه!! برای پاک کردن یک داده بکار میره

localStorage.removeItem(key);

* clear

هرچی هست و نیستو پاک میکنه

localStorage.clear();

**OOP in js**

Oop یا object oriented programming یا به زبان فارسی برنامه نویسی شی گرا.

برنامه نویسی شی گرا در جاوااسکریت به دو قسمت ES5 و ES6 تقسیم میشود .

* ES5

در این نسخه ما چیزی تحت عنوان class نداریم و باید از Constracture functions و الگوی های پروتوتایپی استفاده بکنیم, به مثال زیر توجه کنید:

function Person(name, age) {

*this*.name = name;

*this*.age = age;

}

Person.prototype.greet = function () {

  console.log(

    "Hello, my name is " + *this*.name + " and I am " + *this*.age + " years old."

  );

};

var person1 = new Person("Ali", 30);

person1.greet();

Expected Output : Hello, my name is Ali and I am 30 years old.

در این کد ما ابتدا یک Constracture Function ایجاد کرده ایم و پارامتر های name,age رو به عنوان ورودی بهش دادیم.

در ادامه یک الگوی پروتوتایپ به نام greet به پروتوتایپ person اضافه کردیم.

و در اخر یک نمونه از Person ساختیم و متد greet رو صدا زدیم.

* ES6

در این نسخه ما به class ها دسترسی داریم و میتوانیم از انها برای برنامه نویسی شئ گرا استفاده بکنیم به مثال زیر توجه کنید :

class Person {

    constructor(name, age) {

*this*.name = name;

*this*.age = age;

    }

    greet() {

      console.log(`Hello, my name is ${*this*.name} and I am ${*this*.age} years old.`);

    }

  }

  const person1 = new Person('Ali', 30);

  person1.greet();

  Expected Output : Hello, my name is Ali and I am 30 years old.

این کد همان مفهموم کد قبلی رو داره ولی با ساختاری جدید و منطقی .

ابتدا اومدیم یک class با نام Person ایجاد کردیم و به عنوان پارامتر در داخل Constracture Function دو تا ورودی name,age رو بهش دادیم.

و در ادامه یک متود با نام greet ساختیم.

و در اخر یک نمونه از این class ساختیم و متود greet رو صدا زدیم به همین راحتی!

**AJAX**

AJAX مخفف Asynchronous JavaScript and XML است که یک تکنولوژی برای مدیریت داده ها از طریق api است.

برای این کار 3 روش وجود دارد: Callback Functions – Promise – Async/Await

بیاید دونه دونه همشون رو بررسی کنیم.

* Callback Functions

توابعی هستند که به عنوان یک ارگومان به تابعی دیگر ارسال میشوند به مثال زیر توجه کنید:

function greet(name, callback) {

console.log(`Hi ${name}`);

callback();

}

function callMe() {

console.log('I am the callback function');

}

greet('Peter', callMe);

در این مثال greet تابعی است که name,callback را به عنوان ارگومان دریافت میکند و پس از نمایش پیامی تابع callMe را صدا میزند.

در callback functions ما چیزی داریم به نام callback hell حالا اینی که میگیم چی هست؟

به وضعیتی گفته میشود که به مقدار خیلی زیاد توابع تو در تو یا به اصطلاح nested ایجاد کرده ایم و کد به طرز عیجیبی غیر قابل خوندن شده است.

* Promise

روشی مدرن تر برای مدیریت عملیات های ناهمزمان است که میتواند در یکی از سه حالت : pending – fullfield – rejected باشد.

در Promise با کمک متود های then و catch ارور ها و عملیات های دیگر را کنترل کنید

بیاید با یک مثال براتون توضیح بدم :

function fetchData(success) {

  return new Promise((resolve, reject) => {

    setTimeout(() => {

      if (success) {

        resolve("Data loaded successfully!");

      } else {

        reject("Error: Failed to load data.");

      }

    }, 2000);

  });

}

fetchData(true)

  .then((response) => {

    console.log(response);

  })

  .catch((error) => {

    console.error(error);

  });

در این کد ما یک promise داریم که یک ورودی دریافت میکند و چک میکند ببیند اگر ورودی true بود یک ریسپانس resolve برمیگرداند در غیر این صورت reject را برمیگرداند.

* Async / Await

روشی دیگر برای مدیریت عملیات های نا همزمان است. این روش به صورت خودکار یک promise به ما برمیگرداند که این به این معنا است که ما میتونیم از متد then در این روش نیز استفاده بکنیم به مثال زیر توجه کنید:

async function myFunction() {

  return "Hello";

}

myFunction().then(

  (data) => console.log(data)

);

Expected Output : Hello

این کد به ما یک promise برمیگردانه که خروجیش کلمه Hello است.

حالا Await چیه هرگاه در Async Function با عملیاتی داشتیم که تا اتمام ان نباید بقیه کد اجرا بشه از Await استفاده میکنیم به مثال زیر توجه کنید :

function helloWorld() {

  return new Promise((resolve) => {

      setTimeout(() => {

          console.log("Hello World");

          resolve()

      }, 2000);

  });

}

async function myFunction() {

  await helloWorld();

  console.log("welcome");

}

myFunction();

در این کد اگر از Await استفاده نکنیم ابتدا Welcome را نمایش میدهد و سپس Hello World را نمایش میدهد ولی از اونجایی که ما از Await استفاده کرده ایم میاد صبر میکنه هر وقت کار فانکشن helloWolrd به اتمام رسید سراغ ادامه کد میره.

حالا این کجا بکار میاد ؟در مواقعی که ما قراره از api ها برای دریافت اطلاعات و نمایش آن ها در وبسایت استفاده کنیم مجبوریم از توابع Async / Await و یا Promise استفاده کنیم. در غیر این صورت پروژه به مشکل میخوره و خروجی مد نظر را به نمایش نمیزارد.

**Api**

در ادامه قراره راجب نحوه دریافت داده از api ها صحبت کنیم .

قبل از این بخش پیشنهاد میکنم حتما برید راجب api ها یکم تحقیق کنید ولی به صورت کلی api رابطی هست بین frontend developerو backend developer که برنامه نویس فرانت میاد به این api ریکوئست میزنه و داده هارو از دیتابیس دریافت میکنه.

Api ها متد های متفاوتی دارند مانند get,post,delete,put که هرکدوم یک سری کارهارو انجام میدن.

برای دریافت داده از api 2 روش وجود داره : Fetch – axios

اول از هرچیزی یه api لازم داریم تا بهش ریکوئست بزنیم.  
<https://jsonplaceholder.typicode.com/todos>

این api به ما لیستی از تسک ها میده و از نوع get هست حالا بیاید با هر دو روش fetch,axios بهش ریکوئست بزنیم;

* Fetch

fetch("https://jsonplaceholder.typicode.com/todos")

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

  .then((data) => console.log(data))

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

در وحله اول به ریکوئست میزنیم داده ای که به ما برمیگردونه را باید فرمت json تغییر بدیم و در نهایت log میگیریمش و در کنسول نمایشش میدیم یک catch هم براش ست کردیم که اگه اروری داد به ما نمایش بده خروجیش یک ارایه با طول 200 است.

حالا بریم با axios بزنیم

* Axios

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

ابتدا باید لینک cdn را به فایل html خودمون اضافه کنیم.

async function getData() {

  const data = await axios.get("https://jsonplaceholder.typicode.com/todos")

  console.log(data.data)

}

getData()

از اونجایی که قراره یک عملیات ناهمزمان انجام بدیم پس باید یک async function نیز داشته باشیم. اول اومدیم با استفاده از axios به api ریکوئست زدیم. خروجی که به ما میده یک آبجکت هست که داخلش یک key به نام دیتا داره و حالا اگه این key رو از ابجکت log بگیریم دقیقا همون خروجی رو به ما میده که fetch داد.حالا برای کنترل خطا باید try/catch استفاده کنیم. حالا اینی که میگیم چی هست میاد تلاش میکنه کاری که ازش خواستیم رو انجام بده و اگر به هر دلیلی نتونس این کارو انجام بده عملیات رو لغو میکنه و میره کاری که داخل catch بهش گفتیم رو انجام میده به این صورت:

async function getData() {

  try {

    const data = await axios.get("https://jsonplaceholder.typicode.com/todos")

    console.log(data.data)

  } catch (error) {

    console.log("Api Error")

  }

}

getData()

اگر دقت کنید error رو به عنوان پارامتر به catch دادیم که میاد با اروری که باهاش مواجه شدید پر میشه.

حالا قراره به یک api از نوع post ریکوئست بزنیم این نوع api ها برای وقنیه که قراره همراه با ریکوئست ما یک سری داده به api ارسال بشه مثل عملیات authentication.  
بیاید با یک مثال ساده با هم بررسی کنیم . مثلا همین عملیات authentication .

واسه شروع به یک api لاگین نیاز داریم که من از این api استفاده میکنم : <https://reqres.in/api/login>

حالا بیاید با دو روش fetch,axios با هم تستش کنیم.

* Fetch

اول نیاز به مقادیری داریم که باید به api ارسال بشه

خوده وبسایت این api به ما به صورت پیش فرض یک سری مقادیر داده که بتونیم باهاش api رو تست کنیم:

const data = {

  "email": "eve.holt@reqres.in",

  "password": "cityslicka"

}

ما اینجا یک آبجکت با نام ذیتا دایم که داخلش دو مقدار email,password رو ذخیره

میکنه طبق چیزی که خود داکیومنت این api به ما گفته اگر عملیات با موفقیت انجام بشه باید به ما یک توکن برگردونه.

fetch("https://reqres.in/api/login", {

  method: "POST",

  headers: {

    "Content-Type": "application/json",

  },

  body: JSON.stringify(data),

})

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

  .then((data) => console.log(data))

  .catch((err) => console.log(console.log(err.message)));

این کد الان خروجیش همون توکنی که گقته بودم رو میده بیاید خط به خط بررسیش کنیم.اول از همه fetch رو صدا میزنیم و مقدار اولی که بهش میدیم api ما هست و مقدار دوک یک ابجکت هست حالا بیاید ابجکترو بررسی کنیم.در وحله اول نوع متد این api رو مشخص کردیم که از نوع post است header رو برای این قرار میدیم که برنامه ما بفهمه که چه داده هایی باید برگردانده شوند و چگونه باید با این داده ها کار میکند و در اخر هم یک body براش تعریف میکنیم و data را به یک رشته تبدیل میکنیم و تحویل api میدهیم بقیه کد هم کارهای تکراری است که قبلا توضیح داده ام.

حالا بیاید همینو با axios بزنیم.

* Axios

async function authentication() {

  const response = await axios.post("https://reqres.in/api/login", {

    email: "eve.holt@reqres.in",

    password: "citysli",

  });

  const data = await response.data;

  console.log(data);

}

authentication();

بعید میدونم دیگه نیازی به توضیح داشته باشه دقیقا مثل get با این تفاوت که یک

ورودی دوم که data ما هست هم بهش میدیم.

نوع api دیگه هم داریم : put – patch – delete که هر سه تاشون دقیقا مثل پستن فقط یک مثال از هرکدوم میزنم و رد میشم ازشون.

* Put
  + Fetch

const data = {

    "name": "morpheus",

    "job": "zion resident"

};

fetch("https://reqres.in/api/users/2", {

  method: "PUT",

  headers: {

    "Content-Type": "application/json",

  },

  body: JSON.stringify(data),

})

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

  .then((data) => console.log(data))

  .catch((err) => console.log(console.log(err.message)));

* + Axios

async function update() {

  const response = await axios.put("https://reqres.in/api/users/2", {

    name: "morpheus",

    job: "zion resident",

  });

  const data = await response.data

  console.log(data)

}

update()

* Patch
  + Fetch

const data = {

    "name": "morpheus",

    "job": "zion resident"

};

fetch("https://reqres.in/api/users/2", {

  method: "PATCH",

  headers: {

    "Content-Type": "application/json",

  },

  body: JSON.stringify(data),

})

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

  .then((data) => console.log(data))

.catch((err) => console.log(console.log(err.message)));

* + Axios

async function update() {

  const response = await axios.patch("https://reqres.in/api/users/2", {

    name: "morpheus",

    job: "zion resident",

  });

  const data = await response.data

  console.log(data)

}

update()

* Delete
  + Fetch

fetch("https://reqres.in/api/users/2", {

  method: "Delete",

  headers: {

    "Content-Type": "application/json",

  },

})

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

  .catch((err) => console.log(console.log(err.message)));

* + Axios

async function deleteUser() {

  const response = await axios.delete("https://reqres.in/api/users/2");

  console.log(response)

}

deleteUser()

این وسط دوتا نکته هم بگم :

نکته اول اینه که api چه از نوع put باشه چه patch هیچ فرقی نمیکنه و نکته دوم اینه که api هایی که متدشون delete است هیچ دیتایی به ما برنمیگردن و همین که status کدئ 204 به ما برگردونن یعنی کد داره درست کار میکنه.

**Babel and Webpack**

هر پروژه ای میتونه تو دو حالت باشه production,development یا به صورت خلاصه prod,dev هم بهش میگن.

Dev به حالتی گفته میشود که پروژه همچنان در حال توسعه است و به ورژن نهایی خودش نرسیده.

Prod به حالتی میگن که پروژه به اتمام رسیده و قراره هاست بشه تا کاربران ازش استفاده بکنند.

در حالت prod یک سری کار ها باید انجام بشه تا پروژه اماده بشه

برای مثال باید حجم پروژه رو به کمترین حالت ممکن برسونیم حالا چطوری باید این کارو بکنیم؟ باید فایل های css,js رو یکی کنیم چجوری؟ با کمک webpack,babel میتونیم این کارو بکنیم البته به یک نکته ی دیگه هم باید توجه کنیم اونم اینه که ممکن برخی از مرورگر ها از کد های ما پشتیبانی نکنند مثلا فرض کنید یک کاربر با اولین نسخه ی گوگل میاد تا از وبسایت ما استفاده کنه و قاعدتا وبسایت ما نمیتونه بهش خدمات ارائه کنه برای حل این مشکل بازم webpack,babel به کمک ما میاد و کد مارو به اولین نسخه ی جاوااسکریپت تبدیل میکنه.

حالا بیاید بریم با هم تستش کنیم.

برای شروع باید npm رو سیستمتون نصب داشته باشید.

دایرکتوری پروژه ای که قراره باهاش babel,webpack رو تست کنیم به این صورته:

Src

* Js
  + App.js
* Style
  + Style.css
* Index.html

در روت پروژه یه ترمینال باز کنید و این دستور رو بنویسید:

npm init -y

بعدش این دستور رو میزنیم تا babel رو نصب کنه:

npm install --save-dev @babel/core @babel/cli

حالا package.json را باز میکنیم و در قسمت scripts این کد را وارد میکنیم:

"scripts": {

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

    "babel":"bable ./src/js/app.js -o ./src/js/boundle.js"

  },

قسمت babel رو اضافه کنید.

بعدش در روت پروژه یه فایل با نام babel.config.json میسازید و کد زیر رو داخلش قرار میدید.

{

    "presets": ["@babel/preset-env"]

}

حالا اگر npm run babel رو بزنیم میاد یه فایل جاوا اسکرپیت جدید داخل همون ادرسی که بهش دادیم برامون میسازه و محتواش کدیه که به اولین ورژن جاوا اسکریپت تبدیل شده .

کد همون کدی هست که ما نوشتیم ولی در ورژن های قدیمی تر.

حالا بریم سراغ webpack.

npm install --save-dev webpack-cli

این کدو تو ترمینال بزنید تا webpack نصب بشه.

دوباره میریم سراغ scripts تو package.json و کد زیر رو بهش اضافه میکنیم:

 "scripts": {

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

    "babel": "bable ./src/js/app.js -o ./src/js/boundle.js",

    "build":"webpack"

  },

بعدش یک فایل با نام webpack.config.js در روت پروژه بسازید .

const path = require('path');

module.exports = {

  entry: './src/js/app.js',

  output: {

    path: path.resolve(\_\_dirname, 'src/js/'),

    filename: 'bundle.js',

  },

};

این کد رو داخلش مینویسیم.

حالا باید یک سری loader ها نصب کنیم.

npm install --save-dev css-loader

npm install --save-dev style-loader

با این دوتا کد css-loader,style-loader رو نصب میکنیم این دوتا به ما کمک میکنند تا بتونیم فایل های css و js رو با هم یکی کنیم.

حالا دوباره برمیگردیم به webpack.config.js و این دفعه کد های زیر رو بهش اضافه میکنیم.

module: {

    rules: [

      {

        test: /\.css$/i,

        use: ["style-loader", "css-loader"],

      },

    ],

  },

در نهایت باید فایلمون اینجوری باشه:

const path = require('path');

module.exports = {

  entry: './src/js/app.js',

  output: {

    path: path.resolve(\_\_dirname, 'src/js/'),

    filename: 'bundle.js',

  },

  module: {

    rules: [

      {

        test: /\.css$/i,

        use: ["style-loader", "css-loader"],

      },

    ],

  },

};

حالا همه چی امادس تا پروژمون رو به حالت prod ست کنیم و فقط کافیه دستور npm run build رو بزنیم تا فایل boundle.js رو برامون بسازه.