<https://github.com/public-apis/public-apis>

თემების მიხედვით არის მერე ცამონათვალი ცრილში თუ არ ჭირდება authentication - უწერია no da http-yes შეგჩიძლია წამოვიღოთ ეეს ინფრორმაცია ჩვენთან დაგამოვიყენოთ ჩვენს აპლიკაციაში ajax call-it

ჯავასკრიპტი გვაქვს სინქრონული და ასინქრონული, სინქრონულია როდესაც კოდი არის ბლოქინგ და კოდი შესრულებუს დროს როგორც დავწერთ მიყვება ხაზების თანმიმდევრობით როგორც გვიწერია, ხოლო ასინქრონული არის non-blocking დრაც ნიშნავს რომ თანმდიმდევრობა შეიძლება ხაზების მიხედვით და დროით სიზუსტით არ იყოს დაცული, ასინქრონულია settimer მაგალითად და image-src როცა დრო ჭირდება ჩატვირთვას და ბრაუზერში გაშცვრებას ეს ყველაფერი კოდს ხდის ასინქრონულს

გვაქვს ძალიან მნიშვნელოვანი რამ API რაც არის application programming inferface ერას არის სერვერსა და უზერს შორის, მას ეძახიან ასევე online-apis - ის იღებს მონაცემებს სერვერიდან და პასუხს აპრუნებს მოთხოვნის შესაბამისად. ამისათვის ვიყენებთ ajax call-s ამ სახელით ვიყენებთ თუმცა ძველი სახელი შეიძლება შეგვხვდეს. XLM

ერთ-ერთი ძალიან მნიშვნელოვანი API data froamt aris JSON DATA FORMAT.

როდესაც ავირჩევთ თემის მიხედვით გითზე ჩვენ სასურველ აპის ვაწვებით და გავდავდივართ სხვა პანჯარაზე, სადაც გვხვდება api andpoints- რომელც ასევე url. Iqve dabla aris all და ამ ლინკს წამოვიღებთ.

ანუ ეს ნიშნავს ajax call, მაგრამ როდესაც ჩვენ მივიღებტ ეიპიადან ინფორმაციას ის იქნება ძალიან დიდი და მერე უნდა გამოვიყენოთ JSON.parse(responsiveText) რომ ინფორმაცია ობიექტში მოგვცეს ცალცალკე ჩაშლილდად და გასაგებდ.

ჩვენ შეგვიძლია გაოვიყენოთ ajac call ერთდროულად რამდენჯერმე callback functionit მაგრამ როცა ძალიან ბევრი ინფორრმაცა გვინდა ერთმანეთის თანმიმდევრობით მაშინ უკვე გვჭირდებებოდა ES5 ში function hell რომელიც კოდს ძალიან რთულად წასაკიტხს და შესანარცუნებებელს ხდიდა მაგრამ საბედნიეროდ es6 ში შემოვიდა უკვე promises.

ES6 ში შემოდის fetch function, რომელიც Apiდან იღებს უკვე მოთხოვნას და ეს არის promise

For example

Const request = fetch(url)

Console.log(request) და კოსონლში უკვე შედეგად ვიღებთ დაპირებას.

ფრომისის განმარტება შეიძლება იყოს ასეთი - კონტეინერი რომელმაც უნდა მიიღოს მომავალი value და ეს დაპირება მოდის AJAX CALL-IDAN

Ajax call -dan შეიძლება დაპირებაზე მივიღოთ fullfiled ან rejection, ანუ ან შესრულება ან უარყოფა. Fetch ფუნქციას რომელსაც გადავცემთ api-დან წამოღებულ urls ყოველთვის არის promise და საჭიროა მეორე მოთხოვნაზე return fetch დავუწეროთ.

იმ შემთხვევაში თუ დაპირების დროს მივიღეთ ინტერნეტ კავშირის შემთხვევაში rejecteდ და ერორი კონსოლში ამის თავიდან ასაცილებლად ჩვენ გვაქვს მეთოდი .catch(arr =>alert(arr)).

და სასურველია რომ ეს ერორის დაჭერის მეთოდი გამოვიყენოთ ამ promises ჯაჭვის სულ ბოლოში.

.then მეთოდი ფრომისში ყოველთვის ასრულებს fuflfilled

.catch მეთოდი ასრულებს როცა უარყოფილია პირობა, rejected, მაგრამ ეს მხოლოდ ის ერორია როცა დაიკარგება კავშირი ინტერნეტთან, არის სხვა ერორი როდესაც მაგალითად რამე ინფორმაციის პოვნას გადავცემთ და არ მოიძებნება ამ დროს აბრუნებს 404 ერორს, რომ ინფორმაცია არ მოიძებნა, ასეთი ერორის დაჭერა არ ხერხდება .catch მეთოდით.

.finally ხოლო გვაქვს ეს მეთოდიც რომელიც არ აქვს მნიშვნელობა ერორს დააბრუნებს თუ შესრულებას, ყველა ვარიანში მუშაობს. ასეთ დროს ხელოვნურად ვქმნით პირობით ასეთ რამეს დაპირების ჯაჭვში. If(!response.ok) Throw new Error(~country not found ${response.status}`);

ამ შექმნილი ერორების გამოყენება ძალიან მნიშვნელოვანია რაიმე პრექტის აპლიკაციის გაკეთების დროს რადგან ასეთი ერორების დროს არ ვართ და ცული და საჭიროა მომხმარებლისთვის.

როდესაც ვმუშაობთ promisebze და გვაქვს ერთდროულად then. მეთოდი რომელიც callback ფუნციით აბრუნებს APIდან წამოღებულ რესპონს ამ დროს ამ კოდში თაიმერის დაყენებას აზრი ეკარგება და გარანტია ის დრო არასდროს არ არის სულ თავში რომ იყოს კოდის, რადგან მიღებული რესპონსივი ეშვება microtask queue-ში რომელიც პრიპრიტეტულია კოდის შესრულების დროს და ამ მიაიკროტასკიდან პირველი უშვებს ამ კოდს, ასინქრონულ ჯავასკრიპში პოტოს ჩართვირთვა download პროცესის დროს ეს არ ეშვება call-stack-ში ეს იცდის და მუშაობს web API environment-ში. ჩვენ გვაქვს ასევე callback queue სადაც შედის callback functionebi და ეს callback functionebi გადადის თანმიმდევრულად call-stackshi, მაგრამ microtask queue აქ არის პრიპრიტეტული ეს არის მთავარი.

2017 წლიდან ჯავასკიპტში გაჩნდა await promise რომელიც არის გაცილებით მარტივი და ვიზუალურად დინამიური then. Callback functionebisa da returnebis ნაცვლად. ანუ გვაქვს ასეთი სინტქაქსი

Const something = Async function(){

Const res = Await fetch(url)

Const pos = await getPosition ();

Const {latitude: lat, longtitude: lng} = pos.cords (when we must take coordinates for Geologetion APi

ანუ ეს await გვაქვს ორი ახალი პრომისი რომელიც გვიბრუნდება და ვინახავტ ცვლადში იმიტო გვაქვს ყველაზე განსაზღვრული const რაღაცა;

ამ ახალ ვერსიაშუც აუცილებელია ერორების დაჭერა და აქაც როცა კავშირია დაკარგული ინტეტნეტთან ვიყენებთ catchს. მაგრამ სხვა პროპბლემის დროს მაგალიტად არ იძებნება რაიმე მონაცემბი ამ დროს ვიყენებთ სხვა ერორის აღმოსაჩენ პირობას.

If(!response.ok) throw ner Error (‘problem getting ragaca’); ეს არის ასეთი პირობის დაშვება