Урок №3

(четверть 2, семестр 2,

Продвинутый курс JS)

**АСИНХРОННЫЕ ЗАПРОСЫ В JS**

**Основы асинхронного**

**JavaScript.AJAX, JSON и Promises**

Список рассмотренных вопросов на уроке №3

(тема Асинхронные запросы)

|  |  |
| --- | --- |
| **1. Синхронный JS** |  |
| 1.1. Строка JSON |  |
| **1.2 . Как строку с сервера преобразовать в объект ( JSON.parse )** |  |
| 1.3. JSON. Создать / изменить объект для передачи на сервер |  |
| 1.4. Как объект в виде строки послать обратно на сервер (JSON.stringify), 2 варианта |  |
| 1.5. JSON. Вытаскиваю из « сырых значений» в объекте нужное значение (нормалайзер) |  |
| 1.5.1 Дополнительный разбор по CALLBACK из функции в п.1.5 (с табличкой: что и когда) |  |
|  |  |
| **2. АСИНХРОННЫЙ КОД JS** |  |
| Еще раз разъяснения по callback |  |
| 2.1. Promise, then, catch  Задача 1: вернуть ПРОМИС, что бы потом его обработать  Задача 2: описать **Promice**, и выловить результат**.then**  Задача 3: обработать ошибки Promise через **.catch** |  |
| 2.2. Сахар (вместо Promise): ASYNC / AWAIT |  |
|  |  |
| **3. Практика: XMLHttpRequest vs FETCH** |  |
| 3.1. ES5. Класс XMLHttpRequest: запросить и получить данные, вывести на экран (на примере локального файла tel.json) |  |
| 3.2. ES 6. Fetch API: запрос данных и вывод их на экран  Пример № 1. ЗАПРОС НА «СЕРВЕР» (локальный файл)  Пример № 2. ЗАПРОС НА СЕРВЕР и ИЗВЛЕЧЬ ДАННЫЕ (на реальном сервере с json-файлами)  \*Пример №3. ЗАПРОС НА СЕРВЕР, ИЗВЛЕЧЕНИЕ И ВЫВОД ДАННЫХ НА ЭКРАН БРАУЗЕРА (**на реальном сервере с json-файлом в 100 записей**) |  |
|  |  |
| **4. Продолжаем делать проект МАГАЗИНА (на уроке)** |  |
| 4.1.1. ООП: Код магазина с прошлого урока №2 (исходник для работы на уроке №3), который надо переделать под запросы с сервера |  |
| 4.1.2. ООП и ES\_5: магазин с XHR-запросом на сервер (на уроке) |  |
| 4.2. Задача: запрос данных в json-файле из каталога в GitHub-репозитории ( ES 6: FETCH API) |  |
|  |  |
| **Решение д/задания из урока №2**. Надо было написать пустые классы. Просчитать сумму товаров (не записывал) |  |
| **Домашняя работа для этого урока№3.**  Написать запрос на Promise(устар. метод!).  Создать Методы для Корзины: добавить/удалить /Получить список товаров Корзины (на основании запросов к серверу!) |  |
|  |  |

1.1. Строка JSON

<body>

<div id="ajax-block"></div>

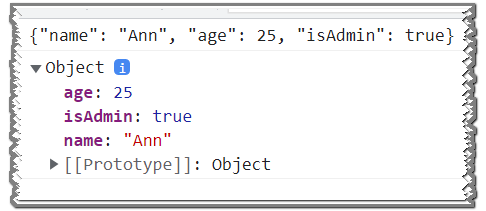
<button id="ajax-get">Async get data</button>

<script> <script>

Пишем данные в формате JSON для переменной

**let userJson = '{"name": "Ann", "age": 25, "isAdmin": true}';**

**console.log (userJson);**



***(что в консоли браузера: строка на входе)***

Правила форматаJSON:

1. JSON - ‘строка’, оформленная по определенным правилам, чтоб потом можно было обратно *распарсить*
2. Это {объект } Это строка !
3. С этими данными надо работать как с объектом (через распарсивание)
4. Все строки и ключи в “двойных кавычках” (“key”: “string” , “key”: number)
5. Ключи разделяются запятой
6. Все сущности – без двойных кавычек:undefined, true, false
7. JSON оформляются *парсерами и стрингифаерами*
8. Такие строки прилетают с сервера (после запроса на него)

Новое для меня: JSON *parse (из строки – в сущности) stringify*

JSON (**JavaScript** ObjectNotation) – это общий формат для представления значений и объектов. Вместо XML (неудобство в тегах). **Объектная нотация javaScripta**

Методы: parse, stringify

1.2 . Как строку с сервера преобразовать в объект ( JSON.parse )

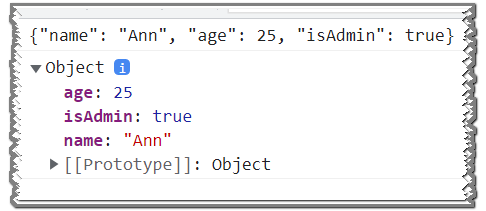
Запрос на сервер -> в ответ строка - > ее надо преобразовать (распарсть) в объект (для удобства обработки сущности / объекта)

Timecode:00:31+

**let userJson = '{"name": "Ann", "age": 25, "isAdmin": true}';**

**let user = JSON.parse (userJson);**

**console.log (user);**



***(что в консоли браузера: строка преобразована в объект )***

*Распарсинг строки с сервера в объект – позволит дальше работать с объектом, извлекая из него нужные точечные данные:*

**console.log (user.name, user.age)** // Ann 25

Пример ключ:значенние

|  |  |
| --- | --- |
| Key (ключ) | value (значение) |
| **"name":** | **"Ann"** |
| **"age":** | **25** |
| **"isAdmin":** | **true** |

*Получена с сервера строка -->* ***JSON.parse*** *-> стал объект.*

*С объектом (с его свойствами) дальше делаем все, что надо фронтендеру.*

*Parse , stringify – это методы (функции JS)*

1.3. JSON. Создать / изменить объект для передачи на сервер;

Получили строку с сервера –> преобразовали ее в объект:

**let userJson = '{"name": "Ann", "age": 25, "isAdmin": true}';**

**let user = JSON.parse (userJson);**

В уже существующую переменную USER (созданный в п.2, как let user) – добавилось еще пару строк (к примеру, юзер заполнил Контактную форму):

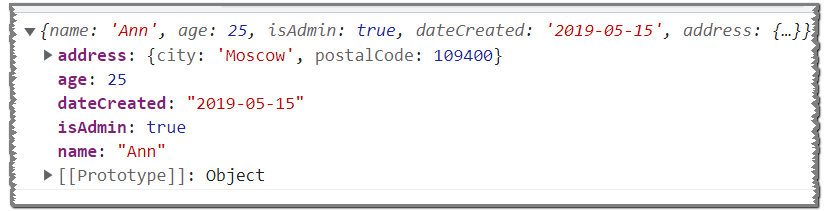
**user.dateCreated = '2019-05-15';**

**user.address = {**

**city: 'Moscow',**

**postalCode: 109400,**

**}; console.log ( user );**



***(в консоли Хрома: объект “user” дополнен новыми данными )***

Требуется выслать данные (информация в виде объекта) на сервер ( для этого надо объект преобразовать обратно в строку):

(см далее – п.4 )

1.4. Как объект в виде строки послать обратно на сервер (JSON.stringify).

**let userJson = '{"name": "Ann", "age": 25, "isAdmin": true}';**

**let user = JSON.parse (userJson);**

**user.dateCreated = '2019-05-15';user.address = {**

**city: 'Moscow', postalCode: 109400, };**

Как передать на сервер в строчном формате? **JSON.stringify :**

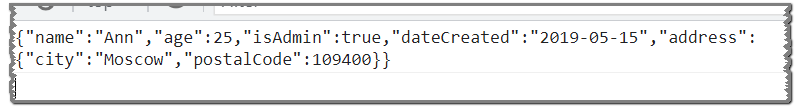
Вариант 1 (все в одной строке: с одним параметром):

Требуется выслать данные (информация в виде объекта в ранее созданной/измененной переменной **user**  см. п.3) на сервер.

Для этого надо объект преобразовать обратно в строку:

**const editedUser = JSON.stringify (user)**

**console.log ( editedUser );**



***( консоль: строка “editedUser” так готова уйти на сервер)***

Фронтендер (получил…отправил): JSON.parse( )…JSON.stringify ( ). А во фреймворках это все будет зашито на уровне библиотек, нужно-то только скармливать данные :)

Эту строку ( переменную **editedUser**) уже можно парсить (перевод обратно в объект). К примеру, как получить отдельно значение года из всей этой каши – см. пример из части 3

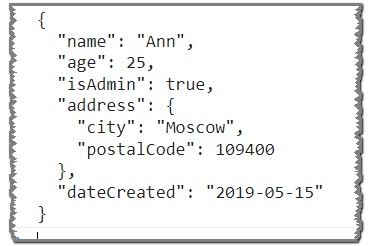
См. далее - (вариант 2, редкий): Передача на сервер в строчном формате

Вариант 2, редкий(!)

(сортировка удобочитаемая: с параметром массив, содержащим ключ / ключи, 2-кол-во пробелов):

|  |  |
| --- | --- |
| **let userJson = '**  **{"name": "Ann", "age": 25, "isAdmin": true}';**  **let user = JSON.parse (userJson);**  **user.dateCreated = '2019-05-15';**  **user.address = {**  **city: 'Moscow',**  **postalCode: 109400,**  **};** | получено с сервера  **JSON.parse** преобразовано в объект  объект дописан |
| **const editedUser = JSON.stringify (**  **user,**  **['name', 'age', 'isAdmin', 'address', 'city', 'postalCode', 'dateCreated'],**  **2);** | **JSON.stringify**  вариант передачи (объекта в виде строки ) на сервер.  Нюанс при заполнении address{ }! |

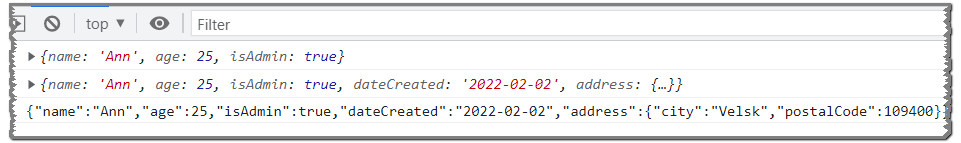
**console.log (editedUser);**

****

***( консоль: так переменная “editedUser” готова уйти на сервер)***

Итог по получению строки с сервера -> преобразование в объект для работы -> отправка обратно на сервер:

|  |  |
| --- | --- |
| **let userJson = '**  **{"name": "Ann", "age": 25, "isAdmin": true}';**  **let user = JSON.parse (userJson);**  **user.dateCreated = '2019-05-15'; user.address = { city: 'Moscow', postalCode: 109400, };** | Получено с сервера  **JSON.parse** преобразовано в объект для обработки  объект дописан / изменен |
| **const editedUser = JSON.stringify (user)** | **JSON.stringify**  вариант передачи (объекта в виде строки ) на сервер. |



***( консоль: (в 3 строки ) принять-изменить-выслать обратно )***

Как из **editedUser**  при помощи нормалайзера вытащить к-либо значение (на примере new Data () ) – см. далее

\_\_\_\_\_\_\_\_\_\_

**Новые знания:**

Как создать новый объект (как создать копию объекта )

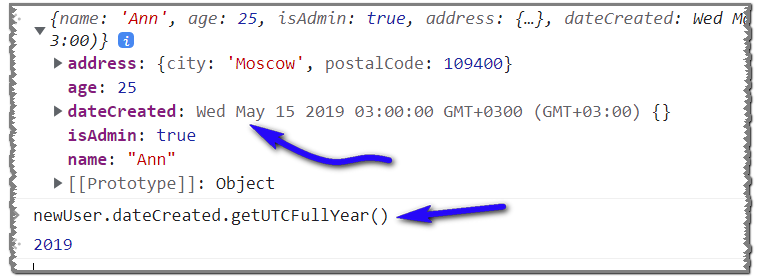
JSON.stringify и JSON.parse используют для глубокого копирования массива / объекта. Это инициализирует новые сущности – в новых ячейках памяти. Но методы не будут скопированы!

1.5. JSON. Вытаскиваю из « сырых значений» в объекте нужное значение (нормалайзер);

При помощи **JSON.parse** извлекаю **Дату 2022 год**

Пишем НОРМАЛАЙЗЕР. Он вклинивается в **JSON.parse** ираскладывает по полочкам извлечённые данные из очень сырых данных с сервера (массив и словарик – что какому индексу соответствует)

|  |  |
| --- | --- |
| **const newUser = JSON.parse (*editedUser*, function (key, value) {**  **if (key === '*dateCreated*'){**  **value = new Date(value);**  **} return value;**  **});**  **console.log(newUser);** | Переменная **newUser** = парсю  ***editedUser*** это парамс , **function** это колбэк (параметры ключ/значение **key, value** для передачи в callback) {  найти строку ***dateCreated*** *и преобразовать в экземпляр класса* **new Date**  }  Сам callback вот он:  **function (key, value) {**  **if (key === '*dateCreated*'){**  **value = new Date(value);**  **} return value; }** |



Работа с датой (как экземпляром класса **new Date**) после нормалайзинга:

**newUser.dateCreated.getFullYear( )** // 2022

**1.5.1 Дополнительный разбор по CALLBACK из функции в п.1.5**

timecode: 00:54:55

**const newUser = JSON.parse (*editedUser*, function (key, value) {**

**if (key === ' *dateCreated* ' ){**

**value = new Date(value);**

**} return value;**

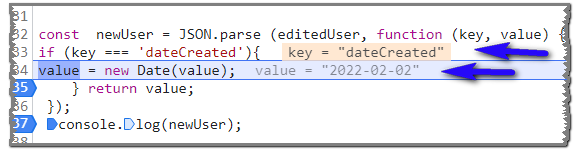
**}**

**);**

1. фрагмент всего, что относим к callback – выделено серым

2. При обработке параметров **(key, value)** у callback происходит следующее.

|  |  |  |  |
| --- | --- | --- | --- |
| круг | Что подставляется в параметры function (**key, value**) | Совпало ли с **if** **(key === '*dateCreated* ' )** | **return value** |
| 1 | function (“name”, “Ann”) | нет | х |
| 2 | function (“age”, 25) | нет | х |
| … | function ... … … | нет | х |
| 5 | function (“dateCreated", 2022) | ДА | "2022-02-02" |
| … | function ... … … | нет | х |



Пример ключ: значение

|  |  |
| --- | --- |
| Key (ключ) | value (значение) |
| **"name":** | **"Ann"** |
| **"age":** | **25** |
| **"isAdmin":** | **true** |

про JSON – закончили. Далее – асинхрон

Часть 2. АСИНХРОННЫЙ КОД

таймкод 00:59

Асинхронность – это то, что возникает не здесь и сейчас, а через промежуток времени. В работе с асинхроном помогут:

SetInterval, setTimeout

Пример кода, отрабатывающего синхронно

|  |  |
| --- | --- |
| **const num = ( ) => {**  **let b;**  **setTimeout ( ( ) => {**  **b = 20;**  **}, 500 );**  **return b;**  **};** | Объявление  **setTimeout ( )**  эта ф-ция просто вызовется внутри стрелочной ф-ции.  Увидит let b.  Вернет пустую let b == undefined  Увидит и отработает setTimeout ( ) ! |
| **console.log (num ( ) );** | // undefined |

Это все тот же синхронный код, вызывающий ф-цию setTimeout (), которая ч/з …0,5 сек. присваивает 20 для b. НО число 20 уже не успевает вывестись на экран, т.к.уже произошёл возврат return b.

А что делать, если вместо setTimeout будет запрос на сервер, который надо отработать? И нельзя получить ответ undefined ! Ответ далее.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PS.: см. п.3.2. (через **fetch** , ES6) научился отправлять на Сервер запрос, получать ответ и выводить итоги на экран

Задача: а если это ответ с сервера, который мне надо обработать (не должен быть ответ undefined)? Ответ: внутрь setTimeout вставить функцию, применив концепцию callBack

таймкод 1ч : 08+

Применение концепции коллбеков *(NB: не ясен пример)*

|  |  |
| --- | --- |
| **const num = (cb ) => {**  **setTimeout ( ( ) => {**  **cb (** 20**);**  **}, 1500 );**  **};**  **num ( (data) => {**  **console.log (data);**  **} )** | cb = это параметр ф-ции num, т.е.  num = (функция callBack)    через 1,5 сек. в консоль выводится число 20 |

Прим.: cb = это от слова callback

Как сработает callback (cb) - см.ранее в п.1.5.1

(NB: не всем был ясен пример – препод его расшифровывал – см. далее)

**Еще раз разъяснения по callback**

таймкод 1ч : 13+

|  |  |
| --- | --- |
| **const num = (cb ) => {**  **setTimeout ( ( ) => {**  **cb (** 20**);**  **}, 1500 );**  **};**  **num ( (data) => {**  **console.log (data);**  **} )** | Серым цветом – объявленная функция |

**const num = (ТутФунцияКоторуюПередаютВМеня) => {**

**setTimeout ( ( ) => {**

**cb (** 20**);**

**}, 1500 );**

**};**

**num ( (ЛюбоеИмяПараметра) => {**

**console.log (ЛюбоеИмяПараметра);**

**} )**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(ТутФунцияКоторуюПередаютВМеня)** – это параметр функции (тут может быть функция, любое число, массив, etc…)

**…num = (cb ) …**– то есть вызываю функцию **setTimeout**

**num ( (ЛюбоеИмяПараметра) …** - это просто вызов функции **cb (** 20**)**

|  |  |  |
| --- | --- | --- |
| 1 | **const num** | **num**: undefined |
| 2 | num ( (data) | **num**: *(cb ) => { setTimeout ( ( ) => { cb ( 20); }, 1500 ); }* |
| 3 | setTimeout | **cb**: *(data) => { console.log (data); }*  **num**: *(cb ) => { setTimeout ( ( ) => { cb ( 20); }, 1500 ); }* |
| 4 | cb ( 20) | **data**: 20 |
| 5 | console.log (data); | **data**: 20 |
| 6 | Вывод: | 20 |

Эволюция: **callback\_Hell** ->  **Promise** -> **async / await**

2.1. Promise, then, catch

**Promise**  - это класс, который опишет , вызовет ф-цию и обработать её (указать правила, как в любом Class). Эта упорядоченность избавляет от callback’ ов. Но лучше сейчас не Promise, а его сахар: **async / await** (о них – см. дальше)

Задача 1: вернуть ПРОМИС, что бы потом его обработать

|  |  |
| --- | --- |
| **let num = (a) => {**  **return new Promise ( ( ) => {**  **} );**  **}**  **console.log ( num ( ) )** | 1. Функция для возврата экземпляра Promise  Образец рабочего ПРОМИСА  **( ) => { }** это callback функция, чтоб **console.log** показала не undefine  Консоль покажет состояние  [PromiseState]: **pending**  (подвисшее состояние, в ожидании) |

Задача 2: описать **Promice**, и выловить результат**.then**

|  |  |  |
| --- | --- | --- |
| **let num = (a) => {**  **return new Promise (**  **(resolve, reject ) => {**  **setTimeout (**  **( ) => { resolve( 20); }, 1500);**  **} );**  **}** | | 1. На вход обозначу два параметра – они же функции / они же параметры ( **resolve, reject** ):  Ф-ция **RESOLVE** вызывается Promise ,когда все отлично отработается;  Ф-ция **REJECT** вызывается, когда запросы падают с ошибками (ошибка 500 на серваке и т.п.). REJECT лучше всегда описывать, чтоб знать проблему  2. **setTimeOut** ( с callBack - фунцией) это логика, которую надо выслать на сервак (если запрос успешно отработается)  3. вызываем **resolve (…)** (если удачно сработает, то сделать то-то) |
| Как выловить полученный результат в **resolve (20)**?!!! | | |
| **num ( ) .then ( ( data ) => {**  **console.log(data);**  **})** | **num ( )**  вызов Promise  Всё, что опишу в **then**– это параметр resolve  В консоли через 1,5 сек. будет число 20 | |

Задача 3: обработать ошибки Promise через **.catch**

|  |  |
| --- | --- |
| **let num = (a) => {**  **return new Promise( (resolve, reject) => {**  **setTimeout ( ( ) => {**  **if (a) resolve (a + 20);**  **else reject ('no param a');**  **}, 1500);**  **});**  **}** | Указал параметр а  На вход обозначу два параметра – они же функции / они же параметры ( **resolve, reject** )  Есть «а», тогда вызовется ф-ция  **resolve (a + 20)**  Иначе вызовется ф-ция  **reject ('no param a')** |
| **num ( )**  **.then( ( data ) => {**  **console.log(data);**  **})**  **.catch ( ( err ) => {**  **console.log (err)**  **});** | если num (20)  **num ( )**  вызов Promise  Всё, что опишу в **then**– это параметр resolve  В консоли через 1,5 сек. будет число 40  если num ( )  Т.к. нет «а», то в Консоли отработает ОШИБКА (catch). Выдаст: **'no param a**  Так ловлю ошибку и обрабатываю ее (и при этом код продолжает работу) |

1. Описать ПРОМИС
2. Указать**resolve, reject** (хорошо, плохо) RES / REJ
3. **resolve** описывается в блоке THEN (может идти по порядку много then’ов)
4. **reject** описывается в блокеCATCH
5. Есть «синтаксический сахар»: ASYNC / AWAIT

**Promise - это объект, в котором есть свойства resolve и reject и эти свойства назначаются с учетом исполнения асинхронности**

**Callback функция - это функция, которую запускает другая функция в определенный момент времени**

2.2. Сахар: ASYNC / AWAIT

Эволюция: **callback\_Hell** ->  **Promise** -> **async / await**

|  |  |
| --- | --- |
| **let num = (a) => {**  **return new Promise( (resolve, reject) => {**  **setTimeout ( ( ) => {**  **if ( a) resolve(a + 20);**  **else reject ('no param a');**  **}, 1500);**  **} );**  **}** | Указал параметр а  На вход обозначу два параметра – они же функции / они же параметры ( **resolve, reject** )  Есть «а», тогда вызовется ф-ция  **resolve (a + 20)**  Иначе вызовется ф-ция  **reject ('no param a')** |
| **async function myNum ( ) {**  **const result = await num(10);**  **console.log(result );**  **}**  **myNum ( );** | Вариант №1  Создать функцию **myNum**  Магия: **await**  Печать результата  Вызов функции |

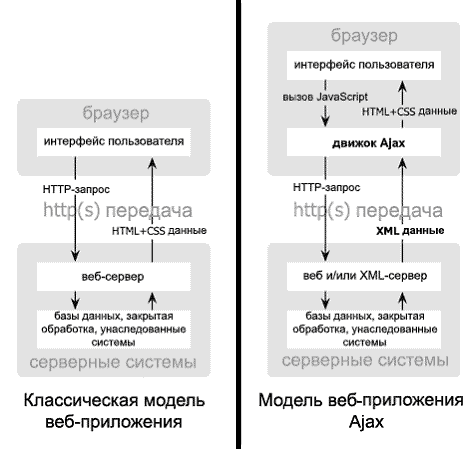
|  |  |
| --- | --- |
| **async function myNum ( ) {**  **try{**  **const result = await num(10);**  **const result2 = await num(20);**  **console.log(result, result2);**  **} catch(e) {**  **console.log(e)**  **}**  **}**  **myNum ( );** | Вариант №2 (для двух значений)  num это экземпляр Promise  На выполнение каждого result будет тратиться по 1,5 сек.  **catch(e)**  отлов ошибок |

**ASYNC / AWAIT** – (сахар над промисами) вместо .then / catch

Совет препода – сра зу работать на ASYNC / AWAIT

3. Практика: XMLHttpRequest vs FETCH

timecode 01:54+



Исходные данные

1. Есть файл 'tel.json' . Который содержит данные {object }:

**{ "name": "Ann", "tel": "+93223435234"}**

2. Или есть json-данные на сайтеhttps://jsonplaceholder.typicode.com/posts

(Для этого можно создать переменную ‘nameURL’):

**let nameURL = ‘https://jsonplaceholder.typicode.com/posts’**

Задача:

1. данные из файла или из сайта надо забрать

2. вывести их на странице.

3.1. ES5. Класс XMLHttpRequest: сделай запрос->

данные получить -> вывести на экран

Создать запрос к локальному файлу 'tel.json'

Можно было и к url-адресу сервера ( *jsonplaceholder.typicode.com/posts* )

// клик кнопки (c id = 'ajax-get')

**var xhr = new XMLHttpRequest ( );** // экз-р класса XMLHttpRequest

**xhr.open ('GET', 'tel.json', true);** // **Open** – просто настройка, где

( метод **GET** , “**ПУТЬ** (к файлу json) / адрес URL / API ”, true = обозначение **асинхронного** запроса)

**xhr.onreadystatechange = function ( ) {**

// **onreadystatechange** – это при СМЕНЕ\_СТАТУСА\_ЗАПРОСА – вызываем функцию ( ):

**console.log (xhr.readyState);** // **readyState** - проверка статуса

// 0 - запрос не инициализирован

// 1 - загрузка данных

// 2 - запрос принят сервером

// 3 - идет обмен данным

// 4 - запрос выполнен (не гарант успешности)

//обработка запроса в зависимости от статуса –

**if ( xhr.readyState ! == 4) return;** // обработка запроса в зависимости от статуса (можно ч/з IF или switch / case)

// Если статус !== 4, тогда return

**if (xhr.status !== 200) {**

**console.log ( ` Error ${xhr.status} ${ xhr.statusText } ` );**

// Если статус !==200 ( НЕ успешно обработано), то вывод инфы

( **${код ошибки} ${текст ошибки}**  )

(пример: ‘ 404: Page not FOUND’)

**} else {**

**console.log ( xhr.responseText ) ;** // Если статус == 200, то выводи текст (т. к. запрос – это «строка»)

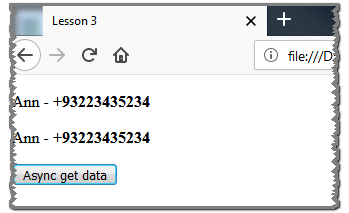
**let data = JSON.parse (xhr.responseText) ;**

**let block = document.getElementById ('ajax-block') ;**

**block.insertAdjacentHTML ( 'beforeend',**

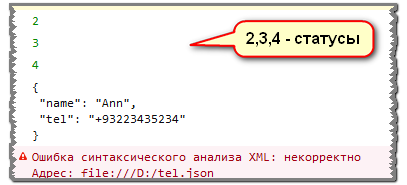
**`** <p> **${data.name} - ${data.tel}** </p> **` );**

**} };**

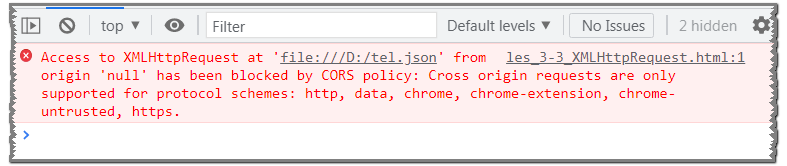
**xhr.send ( );** // фактическая отправка запроса (идет работа с потоками - ?)

**} ) ;**

**В консоли FireFox (старая ver 57.0): напечатало на экран**



**В консоли FireFox (старая ver 57.0): выдало: статусы, name, tel**



**В консоли Хрома: внутр.защита не пропускает такие запросы. Решение проблемы – см. стр. 44 (установка** расширения в Хром CORS Unblock**)**

**Итого по ES 5 XMLHttpRequest ( ) :**

1. Куда подключаться (путь к файлу / к URL) **xhr.open ( )**
2. Как будем обрабатывать **xhr.onreadystatechange = function ( ) {..}**
3. Распарсить **JSON.parse ( )**
4. И вывести на экран **insertAdjacentHTML (…)**
5. Послать запрос **xhr.send ( )**

ToDo: отработать видео с Владилен Минин

Урок 5. JavaScript. Promise. Что это, как работает (+ пример)

Урок 14. JavaScript. Запрос на сервер. Fetch, XMLHttpRequest (XHR), Ajax

3.2. ES 6. Fetch API:

запрос данных и вывод их на экран

в FETCH по умолчанию считается метод GET + URL адрес

FETCH это обычный ПРОМИС ( можно применить asyn/await , либо then и т.д.)

timecode 02:13 +

Пример № 1. ЗАПРОС НА «СЕРВЕР»

(этот пример на локальном файле. Запрос имеет смысл слать на сервер, или на localhost, а не к файлу на диске С)

HTML-код:

<div id="ajax-block"></div>

<button id="ajax-get">Async get data</button>

JS-код:

**document.getElementById ('ajax-get'). addEventListener ('click', ( ) => {**

**fetch ('tel.json')**  *// вариант пути: fetch (http://... )*

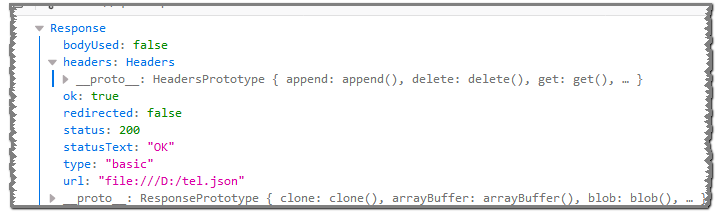
**.then ( (result) => {** *// обработчик then в случае успеха*

**console.log (result) ;**

**} ) ;**

**.catch ( (err) => {** // обработчик catch в случае ошибки

**console.log(err); } ); } );**



**В консоли FireFox (старая ver 57.0): видим Response {…. }**

**Но пока НЕ ВИДИМ : ИМЯ и ТЕЛЕФОН**

Как увидеть из FETCH данные (Имя и Телефон)?

Ответ: пропишем еще один THEN

(см .далее )

Пример № 2. ЗАПРОС НА СЕРВЕР и ИЗВЛЕЧЬ ДАННЫЕ

(на примере данных с сервера *jsonplaceholder.typicode.com/posts*)

HTML-код:

<div id="ajax-block"></div>

<button id="ajax-get">Async get data</button>

SCRIPT - код:

**let nameURL = 'https://jsonplaceholder.typicode.com/posts';**

**document.getElementById ('ajax-get'). addEventListener ('click', ( ) => {**

**fetch ( nameURL )** // в FITCH обязательны два THEN

**.then ((result) => {** //1-й обязательный then (инф-я о рез-те запроса)

**return result.json ( );**

**} )**

**.then ( (data) => {** //2-й обязательный then (пойманные итоговые данные)

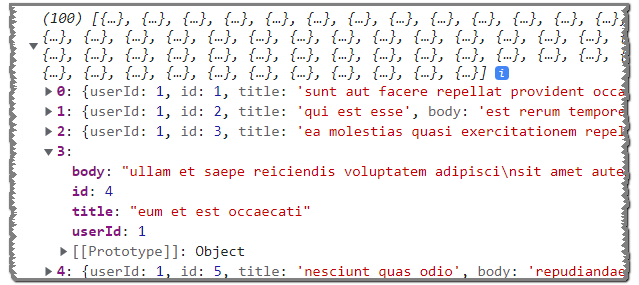
**console.log ( data ) ;**

**} )**

**.catch ( ( err ) => {**

**console.log (err);**

**}); });**

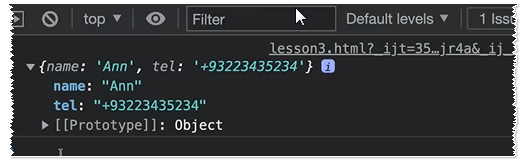


**Консоль Хрома: Ответ сервера - массив данных (готов к обработке)**

Как обработать массив (через forEach) – см. Пример№3

В дополнение к примеру №2:

Если вместо ссылки на url сайта подставить ссылку на локальный файл (как в примере №1 – “ tel.json ” ), то выдаст в консоли такой результат *распарсинга*:

****

**В консоли (скрин из видео, т.к. срабатывает локально через NodeJS)**

Имей в виду при работе с FITCH, что надо 2 .then:

1-й – информация о рез-те запроса (return result.json )

2-й – отлов итоговых данных (data)

**//fetch - это функция, которая выполняет запрос к серверу и возращает промис**

Для чего нужно знать о FETCH? Чтоб слать запросы на сервер (для динамичных сайтов, для минимального трафика запросов в 1000 раз). Позволяет не перезагружать при каждом запросе страницу. Это же верно и для Promise, async/await

Значение ***data***. Именно из неё извлекаются данные для дальнейшей обработки (циклом или проч.) и вывода полученного текста на экран !!

(этот вывод смог сделать только при самостоятельной работе – см п.5)

Пример №3. ЗАПРОС НА СЕРВЕР и ВЫВОД ДАННЫХ НА ЭКРАН

(на примере данных с сервера **https://jsonplaceholder.typicode.com/posts**)

**Html-код**

<div id="ajax-block"></div>

<button id="ajax-get">Async get data</button>

**Script -код**

**let nameURL = 'https://jsonplaceholder.typicode.com/posts';**

**document.getElementById ('ajax-get'). addEventListener ('click', ( ) => {**

**fetch (nameURL)** // в FETCH по дефолту встроен (GET, Путь, true)

**.then ((result) => { return result.json ( ); })** //1-й then (инф-я о рез-те запроса)

**.then ((data) => {** //2-й обязат. then ( итоги). Из ***data*** – извлекаются данные для обраб-ки!

**let block = document.getElementById ('ajax-block');** // куда вставлять

**data.forEach ((ev) => {** // Цикл FOREACH для перебора 100 записей на сайте

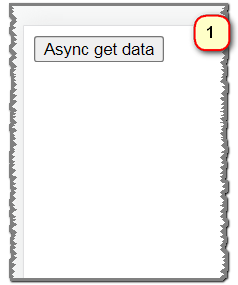
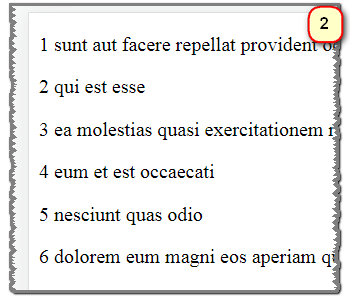
**block.insertAdjacentHTML ('beforeend', `<p>${ev.id} ${ev.title}</p>`);**

**});**

**})**

**.catch((err) => {console.log (err); });**

**});**

**В браузере Chrome: Клик по кнопке выдает: 100 id + 100 заголовков (title)**

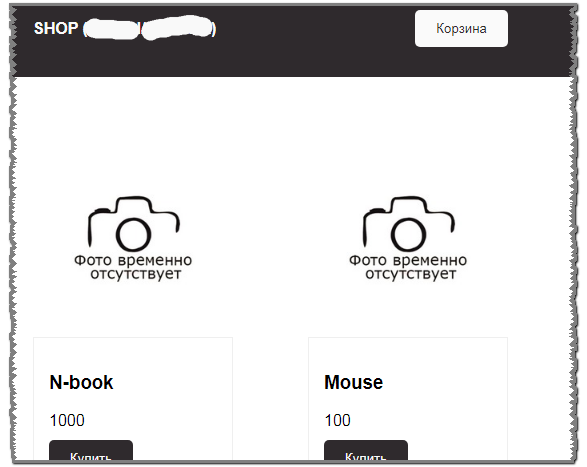
4. Продолжаем делать проект МАГАЗИНА (на уроке)

4.1. Задача: получить данные о товарах с сервера (на ES 5)

Time code: 02час 19мин +

**4.1.1. ООП: Код магазина с прошлого урока №2 (исходник для работы на уроке №3), который надо переделать под запросы с сервера**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Тут будет вписана ф-ция из ES5: XMLHttpRequest() | | | |
| **class ProductList** {  constructor (container = '.products') {  this.container = container;  this.goods = [ ];  this.productObjects = [ ];  this. fetchGoodsData ( )  this. render ( )  } | | | |  |
| **fetchGoodsData ( )** {  this.goods = [ {  id: 1, title: 'N-book', price: 1000 },  { id: 2, title: 'Mouse', price: 100 },  { id: 3, title: 'Keyboard', price: 250 },  { id: 4, title: 'Gamepad', price: 150 },  ];  } | | Удалить это.  Вместо него будет – запрос на сервер | | |
| **render ( )** {  const catalogBlock = document.querySelector ( this.container );  for (let product of this.goods ) {  const productObject = new ProductItem ( product );  console.log ( productObject )  this.productObjects.push ( productObject );  catalogBlock.insertAdjacentHTML ( 'beforeend',  productObject.getHTMLString ( ) );  } } } // class ProductList: end | | | Отрисовщик карточки товара | |
| **class ProductItem** {  constructor ( product, img = 'no\_foto.jpg' ) {  this.title = product.title;  this.price = product.price;  this.id = product.id;  this.img = img; //вар-т: … = product.img || 'no\_foto.jpg'  } | | Инициализация для карточки товара | | |
| **getHTMLString ( )** {return ` <div class= "product-item" data-id= "$ { this.id }">  <img src="${ this.img }" alt="Some img">  <div class="desc">  <h3>${ this.title }</h3>  <p>${ this.price } </p>  <button class="buy-btn">Купить</button>  </div> </div>`; } } | | | | Из чего состоит (html-разметка) карточка товара |
| const catalog = new ProductList ( ); | | | | Старт |



**4.1.2. ООП и ES\_5: магазин с XHR-запросом на сервер (на уроке)**

файл **les\_3-6\_XHR.js**

|  |  |  |  |
| --- | --- | --- | --- |
| const API = 'https://raw.githubusercontent.com/GeekBrainsTutorial/online-store-api/master/responses' | | | Указал ссылку на учебный сервер (без /каталогов) |
| function **getRequest** (url, cb) {  var xhr = new **XMLHttpRequest()**;  xhr.**open**('GET', url, true);  xhr.**onreadystatechange** = function () {  if (xhr.readyState === 4) {  if (xhr.status !== 200) {  console.log('Error!');  } else {  cb (xhr.responseText);  } } }  xhr.**send()**;  } | Смысл ф-ции: сделай запрос к URL и выполни действие (активируй callback)  Под ES\_5:  Функция (на входе ожидаю url, callback)  Класс XHR()  Если ГотовностьСтатус ===4  Если Статус НЕ 200  то это Ошибка  Иначе (т.е. если статус ==200)  передаю responseText | | |
| **class ProductList** {  constructor(cart, container = '.products') {    this. container = container;  this. goods = [ ];  this. productObjects = [ ];    } );  this.\_fetchGoodsData( );  } | | **this. goods** – сюда сохраняются распарсенные значения от запроса **getRequest** () на сервер | |

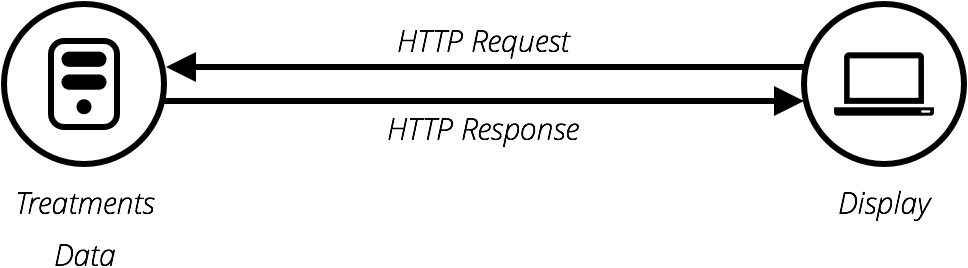
|  |  |  |
| --- | --- | --- |
| **fetchGoodsData( )**  {  **getRequest**(` ${API} / catalogData.json `, (response) => {  console.log(response);  **this. goods** = JSON.parse(response);  console.log(this goods);  **this. render( )**;  });  } | Шлем асинхронный запрос  ( ${url каталога-заглушки на GitHub} , функция-обработчик )  Полученные данные от запроса - надо сохранить (распарсить) в **this. goods**  **this. render( )** отрисовка полученных / сохраненных данных в html-разметке | |
| **render()** {  const catalogBlock = document.**querySelector** (this. container);  for (let product of this. goods) {  const productObject = **new** ProductItem (product);  this. productObjects.push (productObject);  catalogBlock.i**nsertAdjacentHTML** ('beforeend', productObject.**getHTMLString**( ) );  }  }  } | |  |
| **class ProductItem** {  constructor ( product, img = 'no\_foto.jpg' ) {  this.product\_name = product.product\_name;  this.price = product.price;  this.id = product.id;  this.img = product.img || 'no\_foto.jpg'  } | |  |
| **getHTMLString()** {  return `<div class="product-item" data-id = "${this.id}">  <img src="${ this.img }" alt="Some img">  <div class="desc">  <h3>${ this.product\_name }</h3>  <p>${ this.price } </p>  <button class="buy-btn">Купить</button>  </div> </div>`;  }  } | |  |
| const catalog = new ProductList( ); |  | |



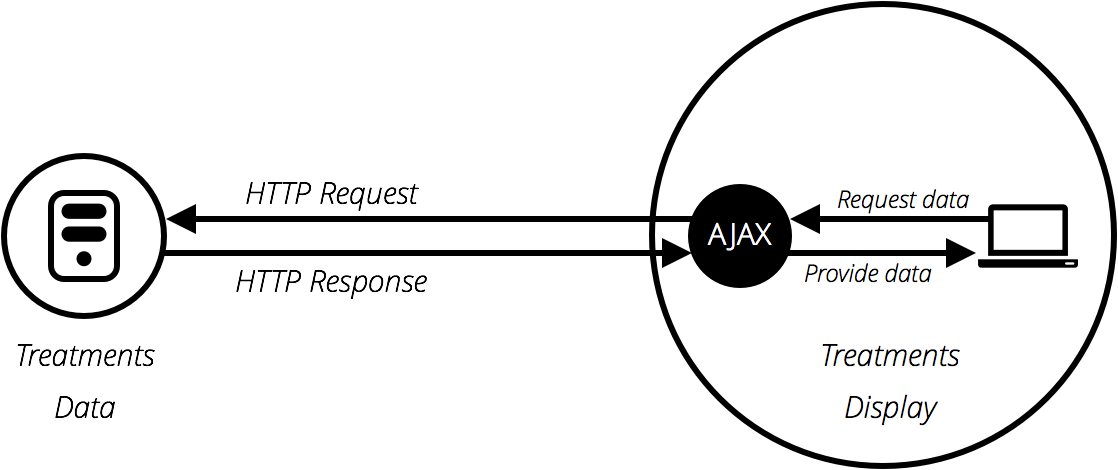
**Отрисуется 2 товара (Ноут 45600 и Мышка 1000)**

Готовый файл файл **les\_3-6\_XHR.js**

https://developer.mozilla.org/ru/docs/Learn/JavaScript/Client-side\_web\_APIs/Fetching\_data



Проблема с этой моделью заключается в том, что всякий раз, когда вы хотите обновить любую часть страницы, например, чтобы отобразить новый набор продуктов или загрузить новую страницу, вам нужно снова загрузить всю страницу. Это очень расточительно



AJAX привел к созданию технологий, позволяющих веб-страницам запрашивать небольшие фрагменты данных ( [HTML](https://developer.mozilla.org/en-US/docs/Web/HTML), [XML](https://developer.mozilla.org/ru/docs/Glossary/XML), [JSON](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON) или обычный текст) и отображать их только при необходимости .

Это достигается с помощью таких API, как [**XMLHttpRequest**](https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest) или - более новой - **[Fetch API](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)**. Эти технологии позволяют веб-страницам напрямую обрабатывать JSON-запросы данных, доступных на сервере; и форматировать результирующие данные по мере необходимости перед их отображением.

XHR существует уже давно и имеет отличную кросс-браузерную поддержку. Fetch станет более популярным.

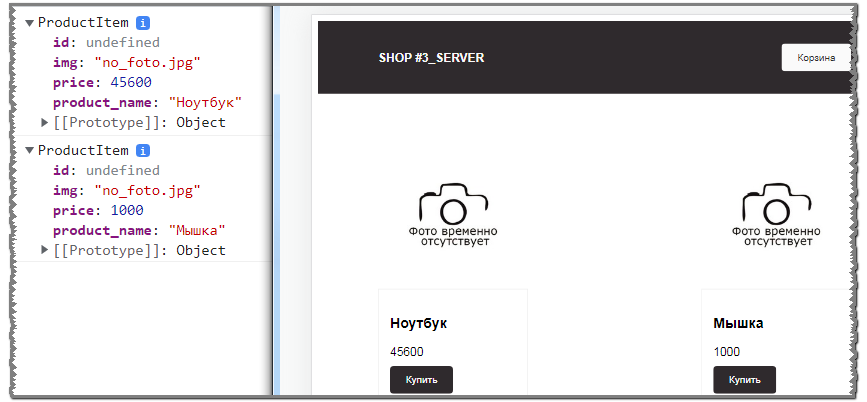
**4.2. Задача: запрос данных в json-файле из каталога в GitHub-репозитории ( ES 6: FETCH API)**

Файл les\_3-7\_FETCH.js Time code: 02час 32мин 30’’

|  |  |  |
| --- | --- | --- |
| <**div class="logo">Shop #3\_Server</div>**  **<button class="btn-cart" type="button">Корзина</button>** | | HTML-код |
|  |  | |
| **const API** = 'https://raw.githubusercontent.com /GeekBrainsTutorial/ online-store-api/master/ responses' | Ссылка на каталог в репозитории (учебный от GB) на GitHub  В нем json-файл с данными о товаре | |
|  |  | |
| **class ProductList** {  constructor (container = '.products') {  this.container = container;  this.goods = [ ];  this.productObjects = [ ];  this.getProducts( ) **.then** ( (data) => {  this.goods = data;  this.render ( );  } );  } | Класс Каталога: **ProductList**  Конструктор  инициализация через THIS  this.getProducts( ) **.then**  это второй then из FETCH-метода | |
| **getProducts ( )** {  return fetch (`${API} /catalogData.json`)  .then (response => response.json( ) )  .catch(err => console.log(err) );  } | // FETCH-метод  Запрос на каталог на Сервере  **.then** - 1й then ; 2й then см.в constructor  если будут ошибки в запросе | |

В этом коде ничего не менялось:

|  |  |  |
| --- | --- | --- |
| **render ( )** {  const catalogBlock = document.querySelector ( this.container );  for (let product of this.goods ) {  const productObject = new ProductItem ( product );  console.log ( productObject )  this.productObjects.push ( productObject );  catalogBlock.insertAdjacentHTML ( 'beforeend',  productObject.getHTMLString ( ) );  } }  } // class ProductList: end | | Отрисовщик  карточки товара |
| **class ProductItem** {  constructor ( product, img = 'no\_foto.jpg' ) {  this.product\_name = product.product\_name;  this.price = product.price;  this.id = product.id;  this.img = img; // как вар-т: … = product.img || 'no\_foto.jpg'  } | | Инициализация для карточки товара |
| **getHTMLString** ( ) {return ` <div class="product-item"  data-id ="${ this.id }">  <img src="${ this.img }" alt="Some img">  <div class="desc">  <h3>${ this.product\_name }</h3>  <p>${ this.price } </p>  <button class="buy-btn">Купить</button>  </div>  </div>`;  }  } // class ProductItem: end | | Из чего состоит (html-разметка) карточка товара |
| const catalog = **new** ProductList ( ); | Старт | |
|  |  | |

 **les\_3-7\_FETCH.js**

5. Поиск решений д/задания урока №3

Создать Методы для Корзины: добавить/удалить /Получить список товаров Корзины (на основании запросов к серверу!)

**5.1. Пояснения преподавателя:**

1. наладить в/действие между карточками товара и корзиной

2. Класс Каталога (**ProductList -**?) готов, его можно брать за основу

2.1. В нем методы добавление, удаление,

2.2. …

|  |  |
| --- | --- |
| constructor(cart, container = '.products') {  … this.cart = cart;  ….  } |  |
| …. addToCart(id) {  // some code get good by id  // this.cart.add(goodData);  }  …. | Как-то ищу нужный товар  Копировать его данные  Передать эти данные у Корзину |
| const cart = new Cart();  const catalog = new ProductList(cart); | ( ProductList(cart) – не ясен механизм передачи ссылки в Каталог на экземпляр Корзины) |

3 По идее, будет работать как в Каталоге, плюс-минус свои методы

3.1. Первичная отрисовка происходит как в Каталоге

3.2. Запрашивать текущее состояние Корзины с Сервера с API на GitHub (**getBasket.json**)

3.3. При добавлении Товара в Корзину, буду обращаться на Сервер (**addToBasket.json**). Текст в файле “result”: 1

Надо проверять … **if result ==1** .., то делаю добавление Товара в Корзину

Это сделано для эмуляции работы с API (т.к. на каждое «Добавить», «Удалить», «Изменить кол-во» надо слать запрос на сервер; если все хорошо – приходит статус ОК / 1 …). Это надо для синхронизации фронта и бэкенда

3.4. Для удаления Товара из Корзины – запрос на **deletFromBasket.json**

3.5. Получение Товара – запрос на **catalogData.json** (сделано уже в коде). Это просто – инфа о товаре. А вот его кол-во – получать только если с сервера придет result ==1.

4. Правила использования статических JSON-файлов

Для получения статического JSON-ответа можно обращаться по URL-адресу

**https://raw.githubusercontent.com/GeekBrainsTutorial/online-store-api/master/responses/имяФайла.json**

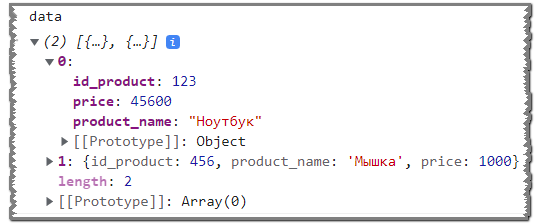
Это вернет чистый JSON-ответ .

**5.2. Словарик :)**

|  |  |  |
| --- | --- | --- |
| слово | перевод | Пример в коде (в консоли браузера) |
| **fetch** | принести | return fetch (`${API}/catalogData.json`) |
| **then** | тогда | then (response => response.json( ) ) |
| **response** | Отклик | Response {type: 'cors', url: 'https:// ИмяСайта/catalogData.json', redirected: false, status: 200, ok: true, …} |
| **pending** | В ожидании |  |
| **rejected** | отклоненный |  |
| **promise** | обещать |  |
|  |  |  |

Значение ***data***. Именно из неё извлекаются данные для дальнейшей обработки (циклом или проч.) и вывода полученного текста на экран !!

(этот вывод смог сделать только при самостоятельной работе – см п.5)



**В консоли на втором THEN можно уже увидеть данные о товарах, взятых с адреса `${API}/catalogData.json`**

**Строка кода:** then ((data) => { this.array = data; this.goodsFSPerebor( ); } )

См. код (запрос на сервер и получение + отрисовка Карточек Товара)

на след. странице:

**5.3. запрос на сервер, получение + отрисовка Карточек Товара (пока без Корзины)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **const API** = 'https://raw.githubusercontent.com/ GeekBrainsTutorial/online-store-api/master/responses'  class Identif{  constructor ( foto = 'noFoto.jpg'){    this.array = **[ ]**;  this.foto = foto;  this.goodsFromServer();  this.goodsFSPerebor() ;  } | | | **const API** ссылка на Каталог с json-файлами (url д.б. БезПробелов!)  **class Identif**  Идентифицирую (что?):  **array** массив (для 2-го then – он как накопитель?)  **[]**-убирает ошибку при forEach: "Cannot read properties.."  **Foto** фото товара («нетФото»)  **goodsFromServer** метод ПолученияДанныхИзСервера  **goodsFSPerebor** метод ПеребораПолученныхДанных | |
| goodsFromServer( ){  //this.array = [{id: 1, title: 'Note', price: 100},{id: 2, title: 'Mouse', price: 200},];  return **fetch** (`${API}/catalogData.json`)  **.then** (response => response.json( ) )    **.then** ( (**data**) => { this.array = data; this.goodsFSPerebor( ); } )  **.catch** (err => console.log(err) );  } | | **goodsFromServer** метод Получения НАСТОЯЩИХ данных с сервера!  //это были ручные данные (якобы с сервера)  **fetch** (принести из каталога сервера)  **1-й then** (инф-я о рез-те запроса)  **2-й then** ( итоговые данные: data); Данные из **data** (через стрелочную функ-ю) данные отдаем в функцию ***goodsFSPerebor*** (для обраб-ки: цикл + отрисовка в html –коде)  **.catch**  Например, при ошибке в url выдало ошибку 404 | | |
| **goodsFSPerebor( )**{  (this.array).forEach( (perebor)=> {  console.log(perebor)  console.log(perebor.id\_product, perebor.product\_name, perebor.price)  let value = perebor;  this.renderCard (value)  });  } | **goodsFSPerebor( )**  Перебирает массив (значения с сервера)+ каждое значение передать-вставить в РендерКарточки  //{id\_product: 123, product\_name: 'Ноутбук', price: 45600} //{...Мышка}  //123 'Ноутбук' 45600 // 456 'Мышка' 1000  Получилось пробросить полученный **let value** в ф-ию **renderCard(value)** т.е. РендерКарточки | | | |
| **renderCard(value)**{  console.log ('проброска успешно: '+ value.product\_name )  let **divProducts**= document.querySelector ('.products');    divProducts.insertAdjacentHTML ('beforeend' ,  `  <div class="product-item" data-id= "${value.id\_product}">  <img src="${this.foto}" alt="foto">  <div class="desc">  <h3>${value.product\_name}</h3>  <p>${value.price}</p>  <button class="buy-btn">Купить</button>  </div> </div> `) }  } //class Identif{..}: end | | | | **renderCard(value)**  Проброшенная ф-ция (где получаю в цикле id, price, title)  Значения value, кот.поступают по очереди из цикла - буду отрисовывать в виде отдельных Карточек Товара  **divProducts** - селектор, куда вставлю КарточкуТовара  в тег*'.products'* вставить вот такой html-код с ${шаблонными строками} |
| let identif = new Identif () | | | | вызов/активация класса Identif{..}  (экземпляр Класса Identif) |

**5.4. Задача: создать методы для КОРЗИНЫ:**

**запрос на сервер, получение данных о состоянии, отрисовка**

Схема работы Корзины (мой взгляд):

1. Слушаю клик кнопки «Купить»

2. При клике: запрос на сервер:

2.1. запрос состояния (result: 1)

2.2. получение data (ИмяТовара, ЦенаТовара, ID\_Товара, ФотоПоУмолчанию)

2.3. Передача данных в отрисовку Корзины

3. Корзина (отрисовать заготовку)

3.1. Строка ФОТО | ИМЯ | ЦЕНА | КОЛ-ВО | +и -

3.2. «+» добавление, «-»убрать

3.3. Если кол-во Товара <1, то убрать строку из Корзины

3.4. При кликах на кнопку «Корзина»: она появляется / убирается с экрана

**Прим.:**

Для получения статического JSON-ответа можно обращаться по URL-адресу.

Это вернет чистый JSON-ответ согласно прилагающейся документации.

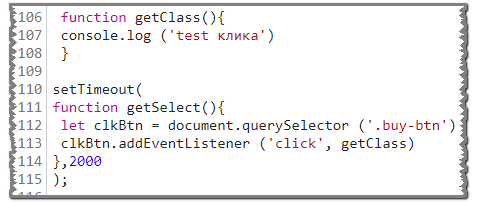
**https://raw.githubusercontent.com/GeekBrainsTutorial/online-store-api/master/responses/имяФайла.json**

|  |  |  |
| --- | --- | --- |
| **Добавить товар в корзину** | addToBasket.json | { result: 1 } |
| **Удалить товар из корзины** | deleteFromBasket.json | { result: 1 } |
| **Получить список товаров** | catalogData.json | …"id\_product": 123,  "product\_name": "Ноутбук",  "price": 45600  },  {  "id\_product": 456,  "product\_name": "Мышка",  "price": 1000 … |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Запрос** | **Mock-файл ответа** | **Данные запроса** | **Данные ОК ответа** | **Данные ответа с ошибкой** | **Данные ОК ответа JSON** | **Данные ответа JSON с ошибкой** | **Комментарий** |
| **Корзина и каталог** |  |  |  |  |  |  |  |
| Добавить товар в корзину | addToBasket.json | {  "id\_product" : 123,  "quantity" : 1 } | (string) 1 | (string) 0 | { result: 1 } | { result: 0, errorMessage : "Сообщение об ошибке" } | Подразумевается, что целевая корзина пользователя идентифицируется на стороне сервера (неважно, залогинен он или нет) |
| Удалить товар из корзины | deleteFromBasket.json | {  "id\_product" : 123 } | (string) 1 | (string) 0 | { result: 1 } | { result: 0, errorMessage : "Сообщение об ошибке" } |  |
| Получить список товаров | catalogData.json | {  "page\_number": 1,   "id\_category": 1 } | (string) 1 | (string) 0 | {  "page\_number": 1,  "products":   [  {  "id\_product": 123,  "product\_name": "Ноутбук",  "price": 45600  },  {  "id\_product": 456,  "product\_name": "Мышка",  "price": 1000  }  ]} | { result: 0, errorMessage : "Сообщение об ошибке" } | Возвращается список продуктов для выбранной категории. Сразу же заложена возможность пагинации. |

Удалил графу «Получить корзину пользователя» / getBasket.json / "id\_user": 1 / …

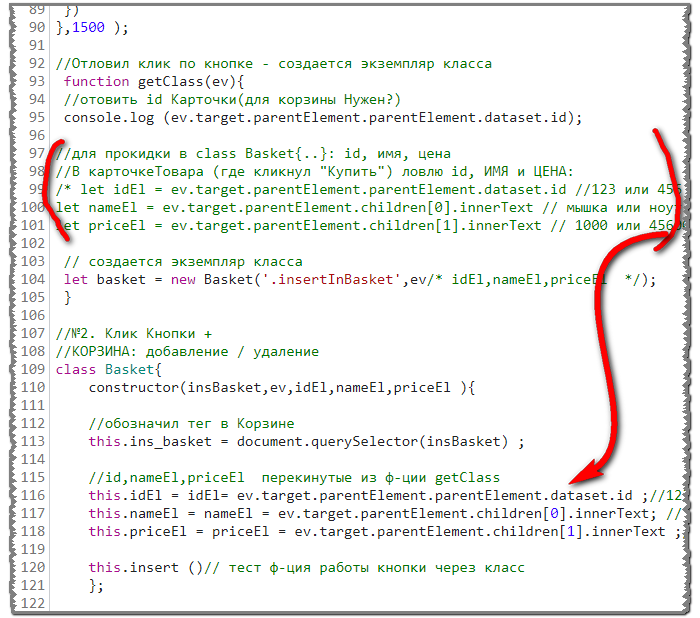
5.5. Каротчки с сервера загружаются не быстро, поэтому пришлось SetTimeout для кнопки ставить



**Сначала карточки загрузилось, а с ними и html код для кнопок «Купить».**

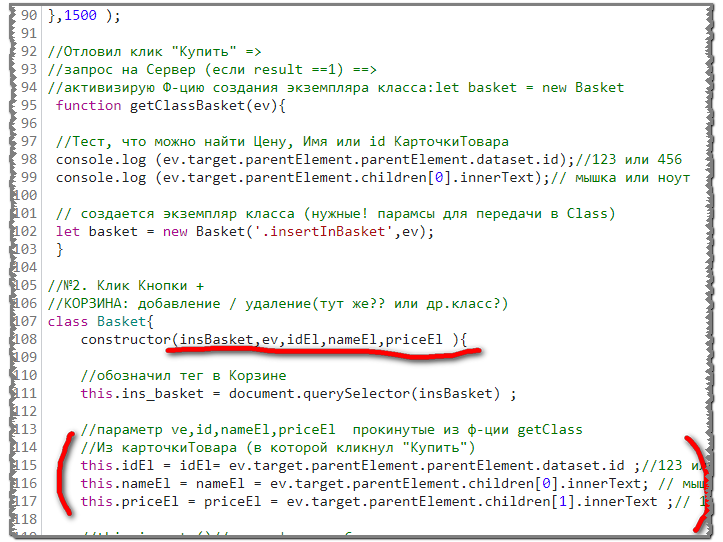
**Потом можно и код document.querySelector ('.buy-btn') применить !**

5.6. Решил все переменные, которым можно инициализировать данные из Карточки, на которую кликнул «Купить» перетащить в сам Class



**Пример прокидки (1й вариант: закомментировано, 2й вариант – без комментов) из Функции getClass(ev) данных Имя,Цена,id сразу в class Basket{…}. Внутри Функции getClass(ev) создаю экземпляр класса let basket = new Basket**

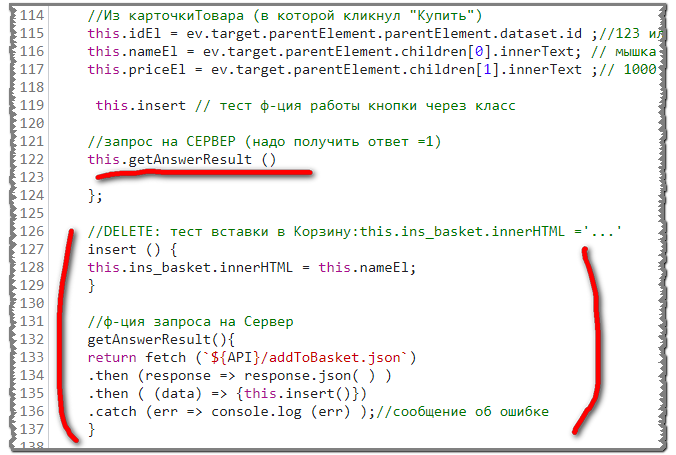
Как стало = см. на след. странице

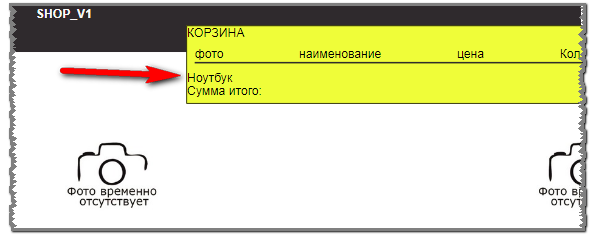


**Из п.5.6. : как стало после прокидки из Функции getClass(ev) в экземпляр класса let basket = new Basket**

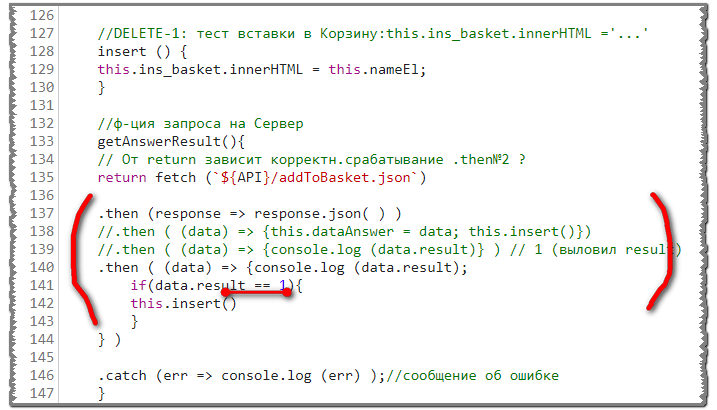
|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

5.7. тестирую вставку (пока от ответа Сервера не зависит)





5.8. Вставка (пробная) на основании того, что ответ от запроса на СЕРВЕР будет result ==1.



5.9. **ToDO**: как понять, что в Корзине уже есть html\_код с добавленным товаром (после первого клика по кнопке «Купить»)?

Мое решение (пересмотрено в п 5.11 – **ориентир на ID**):

1. querySelector для data-тега «кол-во» (quantity) в Корзине

2. Если НОЛЬ , то ренедерим строки добавленного товара

3. Если >0, то просто увеличь quantity +1

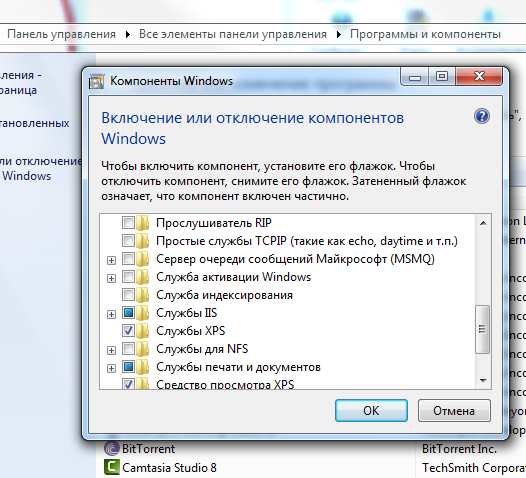
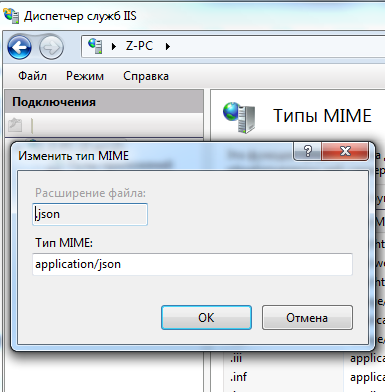
4. При удалении товара: quantity -1. Как только quantity=0, то строка должна быть удалена из Корзины

* **До этого хотел заморочиться:**
* с добавлением id в [ ], причем на каждый клик «Купить». Накопительный массив.
* Каждый клик «купить» - искал бы в [ ] наличие такого id, если его не было – то ренедер строки добавленного товара. Если был бы – то просто quantity +1.
* При удалении 1 товара : удалить такой один id из накопленных в [ ]. Как только опустеет [], то строку вообще удалить из Корзины.
* Как-то сложно в реализации

**5.10 Есть необходимость вызывать локальный json-файл (в Chrome)**

Решение:

1. дал доступ в IIS + добавил в MIME Types значение .json

2. изменил hosts 127.0.0.1

3. Сменил путь внутри js файла на локальный



4. установил расширение в Хром CORS Unblock (чекнув там все галки). Может прокатит и без первых трех пунктов?

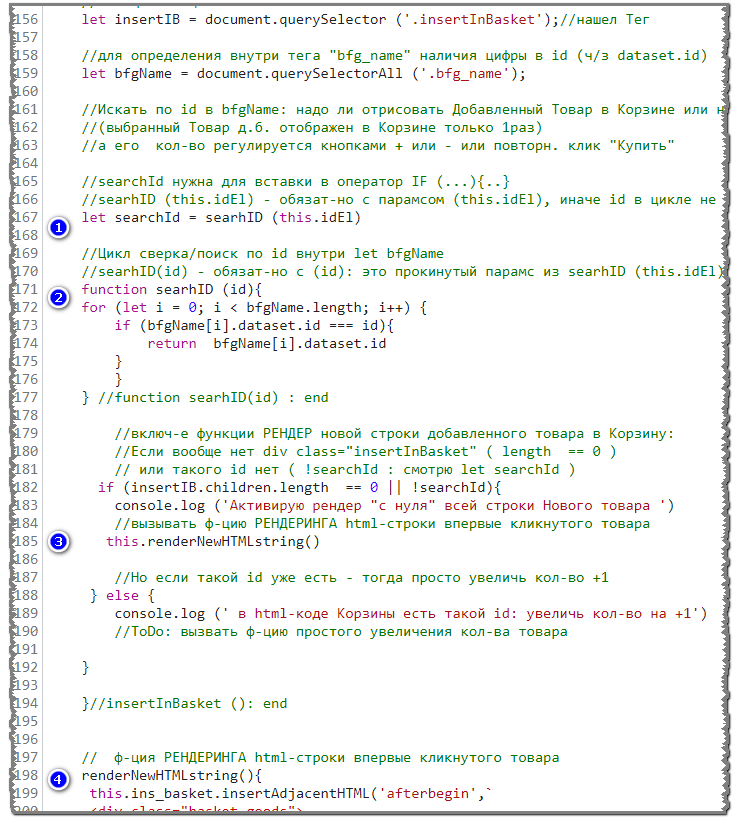
Далее (см. п. 5.11):

Сделать , чтоб при клике на кнопку «Купить» - в корзину прилетали только те товары, которые в Корзине не показаны.

В случае повторного клика «Купить» (т.е. товар уже отображен в корзине) – увеличивается только значение (цифра) количества товара на +1

**ПЕРЕСМОТР КОНЦЕПЦИИ ДОБАВЛЕНИЯ ТОВАРА В КОРЗИНУ (ПО ID)**

5.11. Сразу сопоставляю наличие одинаковых id в кликнутой карточке товара и в Корзине.



Если такого id нет, то смело отрисовать строку в корзине добавленного товара. Если такой ID есть, тогда – просто надо уведичивать цифру кол-ва товара на +1

В зависимости от того есть такой id в Корзине или нет такого id – развиваю два сценария событий (после перебора циклом всех тегов '.bfg\_name' ):

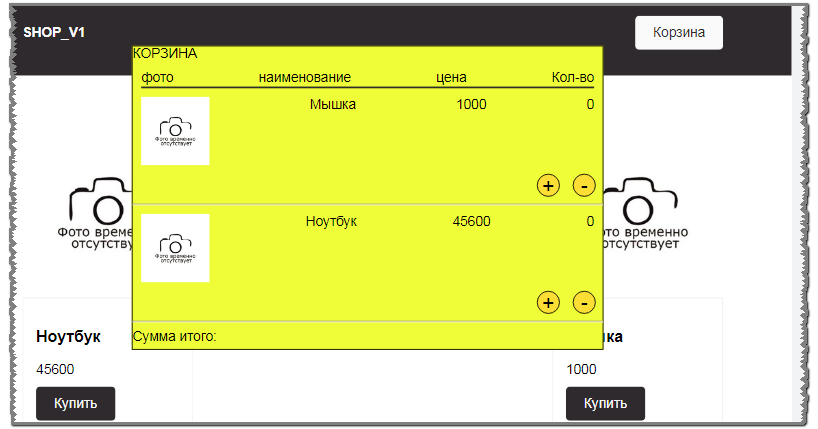
**А). ID совпали (true)**

Значит, в Корзине уже есть / было сделано добавление / была кликунта кнопка «Купить».

Действие: просто увеличь кол-во товара +1

**Б). ID нет такого (false)**

Т.о., рендерить строку товара с «нуля», чтоб появилась в Корзине!



Далее ( )

Сделать ф-цию по увеличению цифры кол-ва добавляемого товара:

Клик по кнопке «купить»: кол-во +1

Клик по кнопке «+»:кол-во +1

Клик по кнопке «-»:кол-во -1

Если товаров становится в Корзине = НОЛЬ, то эту строку удалить

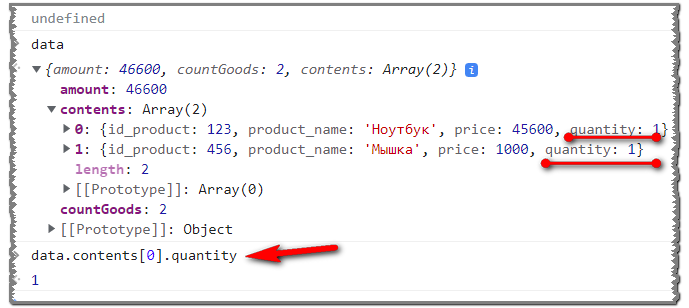
**5.12. Увеличить кол-во товара в Корзине после клика на кнопку (повторного клика, т.е. когда Товар уже единожды отрисовался в Корзине).**

СамСебеЗадачуДал.

Запрос на сервер – к удаленному каталогу ( ${API}/getBasket.json ).

Цель: найти quantity (колич-во) = 1

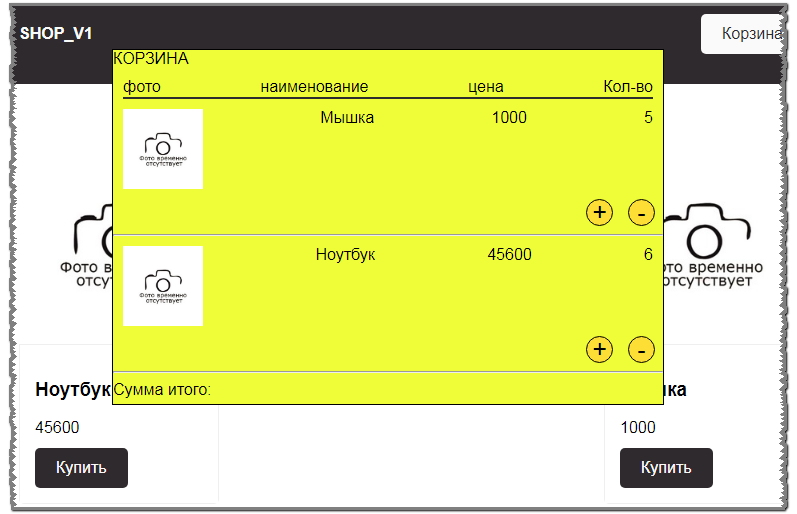
Разбор запроса в консоли выглядит так:



**data.contents[0].quantity – это команда, чтоб вытащить «1»**

Т.о. придется применять цикл для вытаскивания из массива нужной «1».

13.02.2022 г.: На 280 строк кода (+1/2 комментарии):



Сделай:

увеличить / уменьшить через + и – (Пока не буду делать, но сделал).

Тотальная сумма

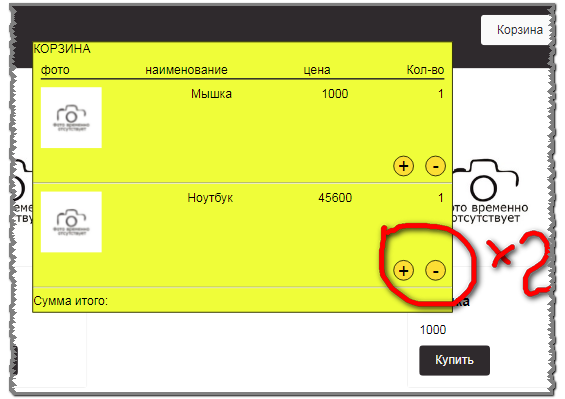
Строка товара удаляется, когда кол-во = 0

По клику на «Корзина»: появляется / исчезает окошко Корзины

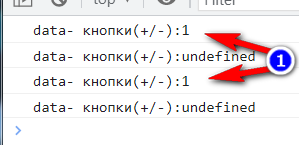
**5.13 Поиск – как избежать задвоение событий на одном клике**

15.02.22

Сложность: если кликать в нижнем поле (в примере – это поле с ноутбуком), то «+»или «-» автоматом ЗАДВАИВАЛИ КЛИКИ (всплытие событий).

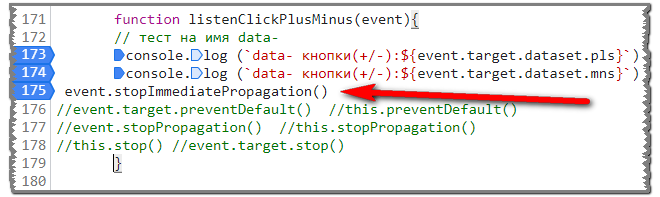


То есть КЛИК один = но событий ДВА (идентичных), а нужно ОДНО событие.



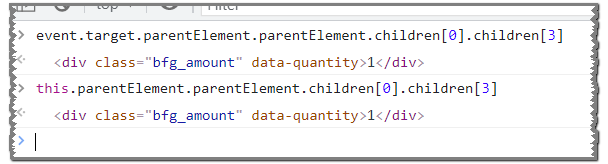
**Один клик – ждал одно событие, а всплывает ДВА**

Решение: **event.stopImmediatePropagation()**



**5.14 Поиск – как от кнопки +/- прокинуть путь до тега «Количество »**

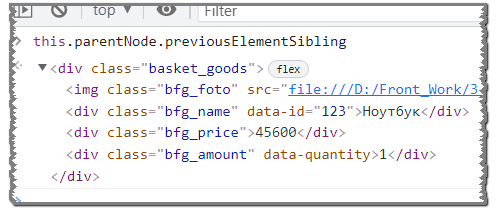
Через event / и можно через this (?)

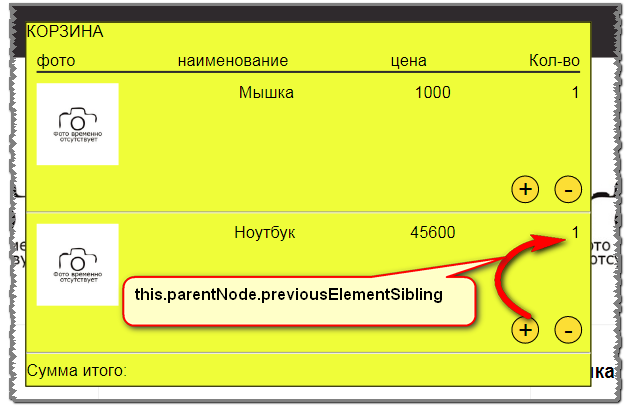


**2 варианта прокидки от кнопки + / - до тега «Количество»:**

**через event, через this**

а вот так – выход просто: с НИЖНЕГО тега на тег ВЫШЕ !



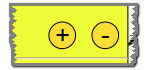


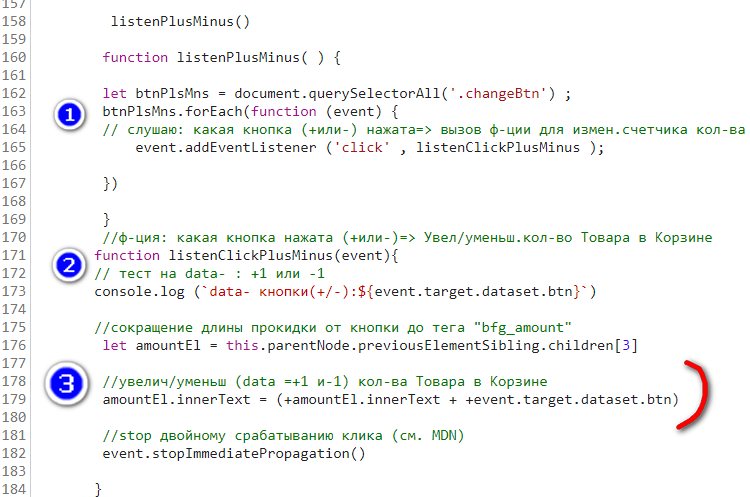
**this.parentNode.previousElementSibling**

**5.14.1 Увеличение / уменьшение кол-ва Товара в корзине**

***(общий принцип)***

this.parentNode.previousElementSibling.children[3].innerText += +event.target.dataset.btn

Этот код увеличивает/уменьшает ( от нажатия клавиши +/- ) кол-во Товара в Корзине

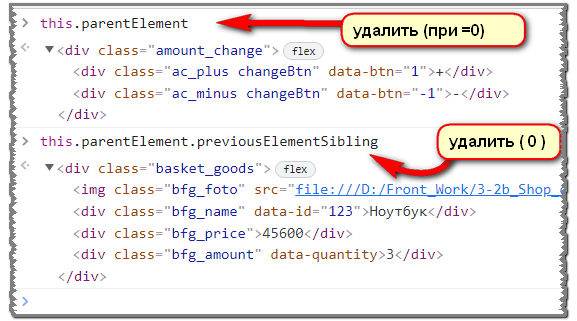


Задача:

При кол-ве товара в Корзине = 0 = > Строка (где кол-во = 0) должна исчезнуть

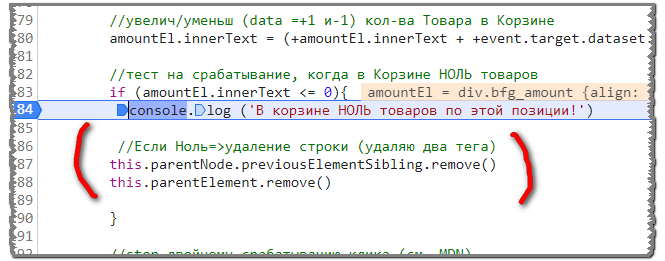
**5.15 Удаление .REMOVE() при кол-ве товара в Корзине = 0**

А). ЧТО удалить два тега в строке, где будет НОЛЬ товаров



Б). Удаление строго по такому порядку (иначе не найдет ):

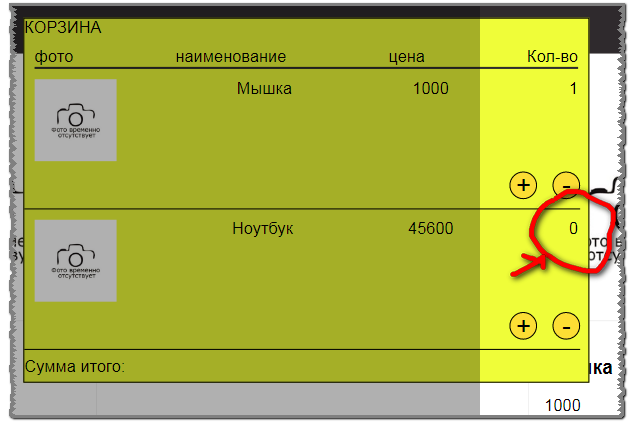
1. this.parentNode.previousElementSibling.remove()
2. this.parentElement.remove()



**Проверка и код удаления двух тегов**

Картинка – на след.странице

Как только стало НОЛЬ товаров в Корзине:



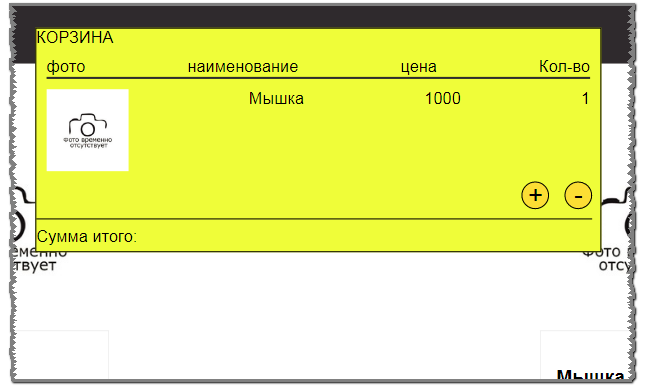
Сработал код удаления двух тегов:

//Если Ноль=>удаление строки (удаляю два тега)

this.parentNode.previousElementSibling.remove()

this.parentElement.remove()

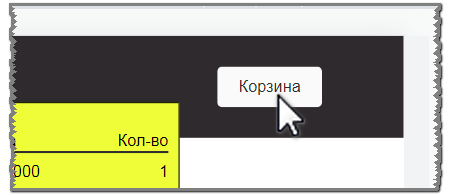
Стало:



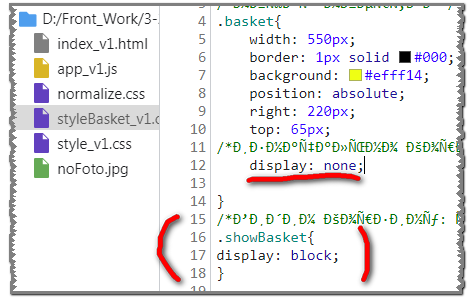
Нижний блок с товаром исчез

**5.16. Задача:**

по клику на кнопку «Корзина» - она должна появляться или исчезать **(toggle)**



Стили CSS заданы так:



По умолчанию (при открытии страницы – Корзину не видим)

Клик по кнопке «Корзина»: активирует видимость корзины через стиль:

**.showBasket{**

**display: block;**

**}**

Полный код ниже:

Код с class, **toggle**

|  |  |
| --- | --- |
| document.querySelector ('.btn-cart'). addEventListener ('click', changeShowBasket)  function changeShowBasket( ){  const toggleBasket = new ToggleBasket ('.basket')  }  class ToggleBasket {  constructor ( basketEl) {  this.basket = document.querySelector (basketEl);  this.listenBtnBasket ( )  }  listenBtnBasket ( ){    **this.basket.classList.toggle ('showBasket')**    }  } *//class ToggleBasket{..}: end* | //делаю видимость/невидимость Корзины с товарами.  //По умолчанию (при открытии страницы – Корзину не видим)  //Клик по кнопке «Корзина»: активирует видимость корзины через стиль в CSS  //Кнопка "Корзина"- слушаю клик=> вызов ф-ции, кот.активир. экземп-р класса  //вызов ф-ции, кот.активир. экз-р класса ToggleBasket  //экз-р класса ToggleBasket (тег самой Корзины)  //Исполнение toggle смог вставить в class(!)  //тег самой корзины (значение basketEl = '.basket')  //автовызов ф-ции видим/не видим Корзину  //видим или не видим (toggle: вкл/выкл класса "видимости") |
|  |  |

Корзину делал 2.02.22 … 16.02.22 (вместе с Ковид, тренировками, основной работой, ведением этого рабочего лога)