Więcej JavaScriptu :)

Hello

Maciej Kucharski

Software Developer @SentiOne

Trainer @InfoShare Academy

Czego się dzisiaj nauczymy?

- Jak reagować na czynności użytkownika na stronie
- Jak obsługiwać formularze

REPO: https://github.com/infoshareacademy/jfdzr5-workshops-js

Events 

Zadzwonimy do Pana...

Events

Do tej pory, w programowaniu w JavaScript, kod robił to co programista napisał, od góry do dołu. Krok po kroku realizował czynności wykonując jedno polecenie po drugim.

Dzisiejsze aplikacje i systemy nie mogą działać wyłącznie w taki sposób, ponieważ dużą rolę w życiu aplikacji odgrywa użytkownik. To użytkownik decyduje kiedy i co ma się wykonać (o ile system to umożliwia). Takie założenie powoduje, że nie wiemy kiedy mamy wywołać odpowiednie funkcje i czy w ogóle one powinny się wykonać.

Events
Zdarzenia w życiu codziennym

Event - zdarzenie, po wystąpieniu, którego ma wykonać się odpowiednia czynność funkcja.

Dla przykładu takim zdarzeniem z życia codziennego może być:

- Pobudka - zapalenie światła w pokoju
- Wyjście z domu - zamknięcie na klucz drzwi
- Spadek temperatury na zewnątrz - zamknięcie okien w domu
- …

Events
Zdarzenia w przeglądarce

Event - zdarzenie, po wystąpieniu, którego ma wykonać się odpowiednia czynność funkcja.

Zdarzeń w przeglądarce jest bardzo dużo, jednak część z nim będziemy stosować najczęściej. Są nimi na przykład:

- onClick
- onChange
- onMouseOver
- onMouseLeave
- onKeyUp / onKeyDown
- onLoad

https://developer.mozilla.org/en-US/docs/Web/API/Document#event_handlers

Events

Podpięcie się pod zdarzenie w HTML

```
<div onClick='alert("bum")'>Czesc, kliknij mnie a zacznie się magia! </div>
```

Events

Podpięcie się pod zdarzenie w JS

```html
<button id='welcome'>Kliknij mnie!</button>
```

```js
const welcomeButton = document.querySelector("#welcome");

welcomeButton.addEventListener("click", () => {
  console.log("Button został kliknięty");
});
```

Events

Odpinanie się od zdarzenia

```
<button id='welcome'>Kliknij mnie!</button>
```

```
const sayHello = () => {
  alert("HELLO");
};

const welcomeButton = document.querySelector("#welcome");

welcomeButton.addEventListener("click", sayHello);

welcomeButton.removeEventListener("click", sayHello);
```

DEMO 

Folder: events

Events
Szczegóły zdarzenia

Po wystąpieniu zdarzenia, do funkcji, która jest podpięta pod dane zdarzenie przekazywany jest obiekt ze szczegółami danego zdarzenia.

Znajdziemy tam wiele informacji, np. Takich jak, na jakim elemencie wystąpiło zdarzenie, kiedy, itp…

```
const welcomeButton = document.querySelector("#welcome");

welcomeButton.addEventListener("click", (event) => {
  console.log(event);
});
```

Zadanie 

Folder: funnyMouse

Events

Szczegóły zdarzenia - target

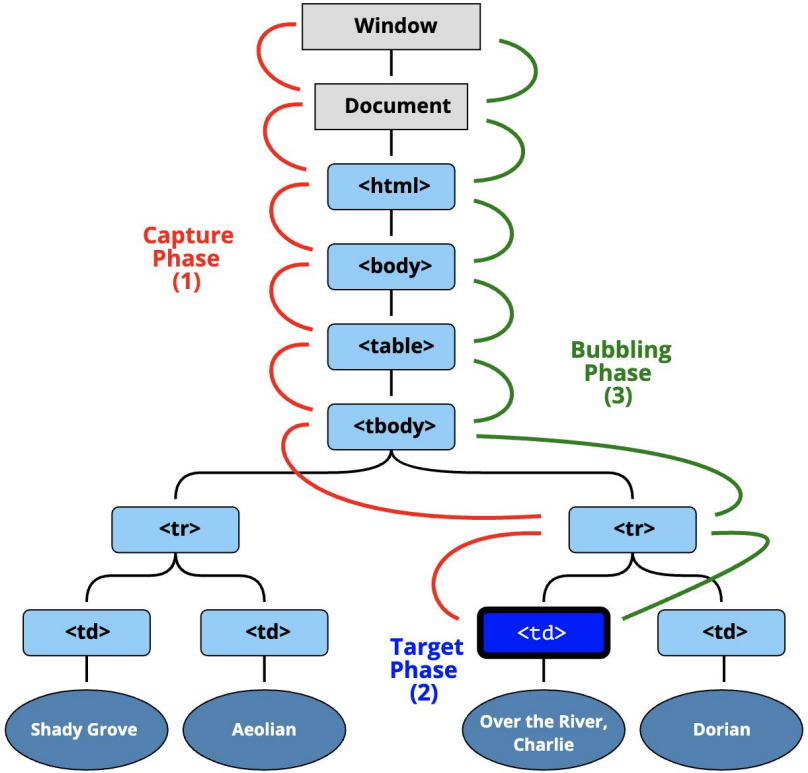
W obiekcie ze szczegółami *eventu* możemy znaleźć takie pola jak:

- .target - element na którym został wywołany event (np. click)
- .currentTarget - element, na którym podpięty jest event Listener

Nie zawsze te dwa pola wskazują na ten sam element!

Events

Jak działa *event*? Fazy *eventów*



https://javascript.info/bubbling-and-capturing

Events
Jak działa *event*? Fazy *eventów*

Standard opisuje 3 fazy propagacji eventów

1. Capturing phase – event jest propagowany z góry do dołu
2. Target phase – event dotarł do elementu, który go zainicjował
3. Bubbling phase – event jest propagowany ponownie do góry

https://javascript.info/bubbling-and-capturing

Events
Jak działa *event*? Fazy *eventów*

Standard opisuje 3 fazy propagacji eventów

1. Capturing phase – event jest propagowany z góry do dołu
2. Target phase – event dotarł do elementu, który go zainicjował
3. Bubbling phase – event jest propagowany ponownie do góry

https://javascript.info/bubbling-and-capturing

DEMO 

Folder: propagation

Events

Zatrzymywanie propagowania eventów

Nie zawsze propagacja eventów jest nam potrzebna. Można zatrzymać ten mechanizm wykorzystując metodę:

```
document.querySelector("#outer").addEventListener("click", () => {
  alert("click on outer");
});

const element = document.querySelector("#inner");

element.addEventListener("click", event => {
  event.stopPropagation();
});

element.addEventListener("click", event => {
  event.stopImmediatePropagation();
});
```

DEMO/zadanie 

Folder: multiSelect

Czego się dzisiaj nauczymy?

- Jak pisać funkcje i wywoływać funkcję w JS
- Jak reagować na czynności użytkownika na stronie
- Jak obsługiwać formularze
- Jak zapisywać dane w przeglądarce

Formularze 

Podaj mi swój numer, a oddzwonię

Formularze

Formularze są nieodłączną częścią praktycznie każdej aplikacji internetowej. Poprzez formularze, inputy, przyciski możemy komunikować się przez przeglądarkę z użytkownikiem.

JS daje nam możliwości oprogramowania tych elementów, tak, żebyśmy byli w stanie pobierać lub wyświetlać użytkownikowi informacje.

Formularze

Rodzaje kontrolek


```
<input type="text" name='username' placeholder="User name"/>

<input type="number" name='age' placeholder="User age"/>

<input type="date" name='birthday' placeholder="User birthday"/>
<div>
    <textarea name="message" rows="6"></textarea>
</div>

<div>
    <input type="radio" name="sex" value='F'/>F
    <input type="radio" name="sex" value='M'/>M
    <input type="radio" name="sex" value='Other'/>Other
</div>

<div>
  <select name="languages" id="languages">
    <option value="eng">English</option>
    <option value="pl">Polish</option>
  </select>
</div>

<div>
  <input type="checkbox" id="isAdult" name="isAdult" checked> Is
Adult?
</div>
```

Formularze

Reagowanie na zmianę

Jest co najmniej kilka sposób na reagowanie na zmianę, wpisanie, kliknięcie w obrębie danego elementu (np. *inputa*)

```
document.querySelector("#username").addEventListener("keydown", e => {
  console.log("event type: KEYDOWN");
  console.log("value: ", event.target.value);
  console.log("field name: ", event.target.name);
});
```

Formularze

Reagowanie na zmianę

W przypadku checkboxa:

```
document.querySelector("#isAdult").addEventListener("change", e => {
  console.log("event type: CHANGE");
  console.log("value: ", event.target.checked);
  console.log("field name: ", event.target.name);
});
```

Formularze
Reagowanie na zmianę

W przypadku option:

```
document.querySelector("#languages").addEventListener("change", e => {
  console.log("event type: CHANGE");
  console.log("value: ", event.target.value);
  console.log("field name: ", event.target.name);
});
```

Formularze

Wypełnianie danymi

```
const username = document.querySelector("#username");
username.value = "Maciej";

const isAdult = document.querySelector("#isAdult");
isAdult.checked = true;
```

DEMO 

Folder: inputsManipulation

Formularze

<Form>

Formularze stanowią logiczną całość i pozwalają opakować wiele kontrolek w logiczną całość.

```
<form name='loginForm'>
    <input type="text" placeholder='username'>

    <input type="password" placeholder="password">

    <button type="submit">Submit</button>
</form>
```

Formularze

<Form>

Formularze dają też nieco wygodniejszą formę na pobieranie danych wpisanych do kontrolek. Nadanie atrybutu *name* elementowi *<form>* spowoduje, że w obiekcie *document* będziemy mieć referencje do formularza i kontrolek, które są w formularzu:

```
const loginForm = document.forms.loginForm;

loginForm.addEventListener("submit", () => {

  console.log("onsubmit");
  console.log(loginForm.elements.username.value);
});
```

Formularze

<Form>

Natywnym zachowaniem formularza, jest wysyłanie danych do serwera na adres podany w atrybucie:

```
<form action="/action_page" method="post">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname"><br><br>
  <label for="lname">Last name:</label>
  <input type="text" id="lname" name="lname"><br><br>
  <input type="submit" value="Submit">
</form>
```

Zauważ, że nawet jeśli nie podamy atrybutów action i method, przeglądarka i tak się wyśle request, nie zawsze jest to oczekiwane zachowanie. Spróbujmy więc powstrzymać przed tym przeglądarkę.

Formularze

<Form> preventDefault();

```
const loginForm = document.forms.loginForm;

loginForm.addEventListener("submit", event => {

  event.preventDefault();

  console.log("onsubmit");
  console.log(loginForm.elements.username.value);
});
```

Dzięki temu, możesz zaprogramować swoje zachowanie na kliknięcie przez użytkownika przycisku submit

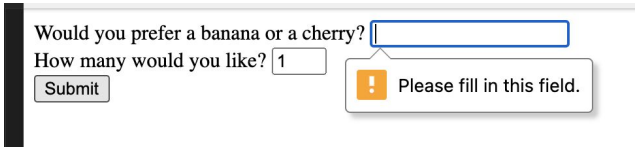
Formularze
Blur i focus

Rodzajów eventów jest całkiem sporo, ciężko je wszystkie zapamiętać, ale z tej długiej listy warto jeszcze znać te dwa eventy:

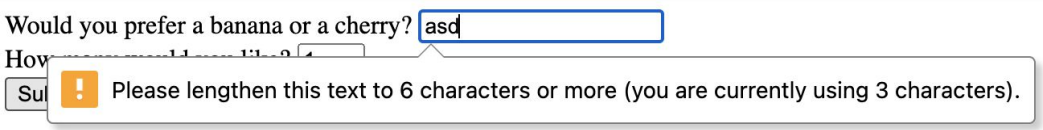
- Focus - kiedy użytkownik kliknie w input i input będzie ‘aktywny’ na wpisywanie
- Blur - kiedy użytkownik odkliknie focus z danego elementu

Formularze
Wbudowana walidacja

Użytkownicy mają to do siebie, że nie zawsze podają dokładnie to czego oczekuje programista. Po to jest walidacja, aby pokazać użytkownikowi, że źle wypełnij nasz formularz:




```html
<form>
    <div>
        <label for="choose">Would you prefer a banana or a cherry?</label>
        <input type="text" id="choose" name="i_like" required minlength="6" maxlength="6">
    </div>
    <div>
        <label for="number">How many would you like?</label>
        <input type="number" id="number" name="amount" value="1" min="1" max="10">
    </div>
    <div>
        <button>Submit</button>
    </div>
</form>
```



https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation

Zadanie 

Folder: simplyForm

Zadanie 

Folder: toDoApp

Zapamiętywanie danych

Zapamiętywanie danych w przeglądarce

Przeglądarka to sprytne oprogramowanie. Poza takimi oczywistymi funkcjonalnościami jakie do tej pory poznaliśmy, potrafi też wiele wiele więcej.

Jedną z tych umiejętności jest zapamiętywanie danych w swojej przestrzeni, z której aplikacje webowe mogą korzystać pomiędzy wejściami użytkownika na stronę.

Przeglądarka potrafi zapisać dane bez użycia backendu.

Zapamiętywanie danych w przeglądarce

Jest kilka rodzajów wbudowanych “pamięci” przeglądarki. Różnią się one pojemnością, okresem żywotności zapisanych w nich danych i funkcją, jaką realizują dodatkowo.

Zapamiętywanie danych w przeglądarce

Cookies 🍪

Ciasteczka, jedna z najbardziej klasycznych form zapisywania danych.

Wiele starszych przeglądarek wspiera tylko tą formę przechowywania danych.

Mały rozmiar (4KB pamięci) często wyklucza użycie tej formy w dużych aplikacjach.

Najważniejsza cecha to, fakt, że cookiesy może ustawiać backend poprzez odpowiednie nagłówki w request nawet bez użycia JS.

Dodatkowo, ciasteczka można zabezpieczyć przed odczytaniem/manipulacją ze strony Javascriptu.

Zapamiętywanie danych w przeglądarce

Web Storage API 💿

Wraz z nadejściem HTML5 doszedł nowy mechanizm na przechowywanie danych: Web Storage API.

Wprowadził on dwie przestrzenie do zapisu danych:

- window.localStorage
- window.sessionStorage

Zapamiętywanie danych w przeglądarce

Porównanie

LocalStorage	5MB/10MB storage It's not session based, need to be deleted via JS or manually Client side reading only Less older browsers support
SessionStorage	5MB storage It's session based and working per window or tab Client side reading only Less older browsers support
Cookie	4KB storage Expiry depends on the setting and working per window or tab Server and client side reading More older browsers support

Zapamiętywanie danych w przeglądarce

Zapisywanie z JS do cookiesów

Zapisywanie danych z JS do cookiesów jest średnio wygodne, ponieważ wszystko opiera się na jednym stringu, który musimy sobie sami tak przetworzyć jak to uważamy za słuszne.

Zapamiętywanie danych w przeglądarce
Zapisywanie z JS do cookiesów

Zapisywanie danych z JS do cookiesów jest średnio wygodne, ponieważ wszystko opiera się na jednym stringu, który musimy sobie sami tak przetworzyć jak to uważamy za słuszne.

```
document.cookie = 'token="test"; max-age=3600;'

console.log(document.cookie)
```

Zapamiętywanie danych w przeglądarce
Zapisywanie z JS do cookiesów

Usuwanie cookiesa polega na nadpisaniu jego wartosci:

```
document.cookie = 'token=test;'

console.log(document.cookie);

document.cookie = "token=tt; expires=Thu, 01 Jan 1970 00:00:00 UTC;";

console.log(document.cookie);
```

Zapamiętywanie danych w przeglądarce
LocalStorage i SessionStorage

LocalStorage i SessionStorage jest zdecydowanie wygodniejsze w użytkowaniu. Mamy dostępne metody, które realizują łatwo manipulacje zapisanymi danymi.

SessionStorage i LocalStorage mają takie samo API.

```
localStorage.setItem("username", "Maciej")

console.log(localStorage.getItem("username"))


localStorage.removeItem("username")

console.log(localStorage.getItem("username"))
```

Zadanie 7

Zapisz swoje imię do localStorage.

Zadanie 8

Stwórz obiekt:

{ userName: ‘Jan’, age: 21 }

I zapisz go do localStorage

Zapamiętywanie danych w przeglądarce
JSON

Czy poprzednie zadanie się powiodło? Czy może zapisało się do LS:

Key	Value
user	[object Object]

Czy wiesz czemu tak się stało?

Zapamiętywanie danych w przeglądarce

JSON - stringify()

```
const user = {
  name: "Jan",
  age: 21
}

localStorage.setItem("user", JSON.stringify(user))

console.log(localStorage.getItem("user"))
```

Zapamiętywanie danych w przeglądarce

JSON - parse()

```
const userFromLS = localStorage.getItem('user')


const userObject = JSON.parse(userFromLS);
console.log(userObject)
```

Zadanie 9

Zaimplementuj aplikacje z formularzem do logowania.

Formularz ma mieć dwa pola: login i email. Kliknięcie *submit* powinno spowodować zapisanie loginu i email do localStorage.

Jeżeli użytkownik wejdzie na stronę i będzie coś zapisane w LS to wyświetl użytkownikowi zapisane dane. Jeżeli jest pusto (nie ma takiego klucza) - pokaż formularz.