11 EVENTS, udalosti

najčastejšie chceš spustiť kód, keď používateľ **klikne** na element. alebo keď **odošle formulár**. tu hore ti dám kód, ktorý na to potrebuješ. potom dole nižšie ti ho podrobne vysvetlím. možno až príliš podrobne 😄

tebe reálne budú stačiť tieto 2 ukážky kódu tu hore. aby si vedel/a reagovať na klik a odoslanie formuláru. viac nepotrebuješ. neboj sa, toto neni maturita na strednej, **nebudem ťa nútiť nabiflovať sa knihu textu na skúšku**, neboj! tieto 2 kusy kódu ti stačia.

ale o eventoch sa dá povedať veľa a *ak ty chceš*, detailne ti dole všetko vysvetlím. napríklad ti poviem aj to, ako písať HTML kód tak, aby tvoju appku mohli používať aj ľudia, ktorí nevidia.

CLICK NA BUTTON alebo ODKAZ

povedzme, že v HTML kóde máš takýto `<button>` element a odkaz, `<a>` element:

```
<button>GO!</button>
<a href="https://www.vasopatejdl.com">klik sem</a>
```

ak chceš spustiť javascript kód po kliknutí na ne:

```
const button = document.querySelector('button');
const link = document.querySelector('a');

button.addEventListener('click', function () {
  // kod tu sa spusti po kliknuti button
});

// pri odkazoch a formularoch potrebujes zabranit tomu
// aby ta prehliadac hodil na inu adresu, takto:
link.addEventListener('click', function (event) {
  event.preventDefault();
});
```

ako klik sa berie aj dotyk prstom na mobile.

ODOSLANIE FORMULÁRU

povedzme, že máš v HTML takýto formulár:

```
<form action="">
  <input type="search" name="q" placeholder="čo hľadáš?">
```

```
  <button type="submit">GO!</button>
</form>
```

keď ho user odošle (či už enterom alebo kliknutím na GO!):

```
const form = document.querySelector('form');
const searchInput = form.querySelector('input[type=search]');

form.addEventListener('submit', function (event) {
  event.preventDefault();

  if (searchInput.value.trim()) {
    // teraz v searchInput.value máš text, čo user napísal
  }
});
```

detailné vysvetlovačky teraz:

EVENTY

počas kurzu sa postupne posúvame ku reálnej, iteraktívnej appke. už vieme napríklad získať dáta cez `fetch`. ale teraz ich potrebujeme získať *až vtedy*, keď si ich používateľ nejakou akciou vyžiada. potrebujeme reagovať na používateľa. alebo všebecnejšie: **potrebujeme reagovať na udalosti**. na takzvané **eventy**.

predstav si **SPOTIFY**: keď používateľ do spotify napíše "vašo patejdl", spustí sa `fetch` na údaje o majstrovskom umelcovi. *spotify reaguje na udalosť písania po klávesnici*.

predstav si **YOUTUBE**: like pre videoklip vaša patejdla sa uloží až po tom, čo používateľ klikne na like button. *youtube reaguje na udalosť kliknutia na button*.

my vieme, že funkcia je séria príkazov, ktoré majú spoločnú misiu. a vieme napísať funkciu, ktorá sa spustí po tom, čo nastane udalosť. cez napríklad:

CLICK EVENT

ešte raz a detailnejšie: povedzme, že chceš spustiť kód, keď používateľ klikne na `<header>` element. a povedzme, že taký element vôbec máš v HTML kóde! ak v HTML máš:

```
<header>
  <img src="./images/playdate.png" alt="playdate image">
</header>
```

tak v javascripte:

```
// naprv z html kodu ziskas odkaz na tento element
// kedze chcem element <header>, cez query selector hladam "header"
// ak chces vediet viac o selektoroch, kurz Webrebel, sekcia o CSS
selektoroch
const header = document.querySelector('header');
console.log(header); // skontrolujem, ci ho mam

// header elementu teraz pridam tzv. EVENT LISTENER!
// chce 2 argumenty:
// - nazov eventu
// - a funkciu, ktora sa spusti ako reakcia na event
header.addEventListener('click', function() {
  console.log('klik na header');
});
```

teraz **keď** klikneš na `<header>`, konzole vypíša *"klik na header"*. v youtube príklade ten header môže byť like button a namiesto výpisu do konzole sa nový like uloží do databázy.

ten `click` je jeden preddefinovaných názvov. je ich kopa, nižšie ti vymenujem také, ktoré sa používajú často.

skrátená, arrow function verzia zápisu:

```
// v javascripte mozes namiesto function() napisat () =>
header.addEventListener('click', () => {
  // kod, ktory sa spusti po kliknuti na header element
});
```

INTERAKTÍVNE ELEMENTY

niektoré HTML elementy budú v tvojej appke iba tak pasívne existovať. zobrazovať sa na obrazovke. **iné elementy budú interaktívne**. ty sa rozhoduješ, ktoré elementy budú interaktívne. ja som v príklade hore ukázal prstom na `header` element a povedal *"ja chcem, aby tento header bol interaktívny. a konkrétne chcem, aby vedel reagovať na to, keď naň používateľ klikne myšou."*

a teraz vieš napísať funkciu, ktorá sa spustí, keď naň používateľ klikne.

spotify má v HTML kóde element `<input type="search">`. spotify vývojári naň ukázali prstom a povedali *"tento input element bude interaktívny. konkrétne chceme, aby reagoval na to, keď doň používateľ bude písať text*.

a teraz vedia napísať funkciu, ktorá sa spustí, keď používateľ píše text do inputu.

KEĎ CHCEŠ REAGOVAŤ NA UDALOSTI

1. určíš, ktorý element má byť interaktívny
2. vyberieš, na ktorú udalosť má reagovať
3. napíšeš funkciu, ktorá sa vtedy má spustiť

konkrétne v kóde:

1. cez `querySelector` vyberieš HTML element, ktorý má byť interaktívny
2. cez `addEventListener` mu pridáš schopnosť reagovať
3. do `addEventListener` pošleš 2 veci: názov udalosti, na ktorú chceš reagovať a funkciu, ktorá sa vtedy má spustiť

niekoľko často používaných eventov: `click`, `keyup`, `submit`, `change`, `input`, `load`, `scroll`... toto sú v javascripte preddefinované názvy. google existuje, MDN existuje, ChatGPT ti vie pomôcť nájsť najpoužívanejšie eventy a na čo slúžia 🙂

je toho kopa, ale najčastejšie asi budeme reagovať na `click`, `submit`, `keyup` a možno `input`? každopádne neboj sa nič, platí to, čo vždy: áno, existuje *množstvo* eventov, ale reálne budeš používať zopár. takže nauč sa najprv používať `click` a `submit` a ostatné príde časom.

mini bonus: keby napríklad chceš **reagovať na pohyb myšou**:

```
const header = document.querySelector('header');

// ked mysou vojdes do header elementu
header.addEventListener('mouseenter', () => {
  console.log('myš DNU', header);
});

// ked mysou vyjdes von z header elementu
header.addEventListener('mouseleave', () => {
  console.log('myš VON', header);
});

// pre kontinuálny pohyb môžeš skúsiť 'mousemove'
```

EVENT LISTENER (ploštice, štěnice, načúvač, odposlech)

všimni si ten `addEventListener` hore. v doslovnom preklade to znamená *"pridaj odposlech na udalosti"*. **ja si to vždy predstavujem ako štěnice (ploštice) na odposluch**. jak vo filme! špionážnom. proste predstav si, že nájdeš ten element pripneš naň plošticu. a ona počúva, naslouchá, či nenastala udalosť. napríklad `click`. a teraz tá štenice nonstop monitoruje *"nastal klik? nastal klik? nastal klik? nastal klik??"* a v momente, keď nastane, sa spustí kód v priloženej funkcii.

na každý element ty môžeš pripnúť mnoho štenic. každá z nich potrebuje 2 informácie: názov eventu a funkciu, ktorá sa má spustiť. a každá z nich nonstop naslouchá.

AK MÁŠ VIAC ELEMENTOV

v našej žltej playdate appke máme od prvej hodiny 3 `<li>` elementy. povedzme, že chceš, aby každý z nich reagoval na klik:

```
// v HTML kode mame 3 li elementy
// takze toto vrati kolekciu 3 elementov
// vsimni si querySelectorAll namiesto querySelector (bez All)
const cardElements = document.querySelectorAll('li');

// kedze mame kolekciu, cylkom prejdeme cez kazdy element
cardElements.forEach(card => {
  // na kazdy z nich nalepime plosticu / stenici
  // ktora bude naslouchat, ci nastal click element
  card.addEventListener('click', () => {
    // kod, ktory sa ma spustit po kliku na li
    // ...
  });
});
```

jo? ak mám 10 elementov a chcem, aby každý z nich reagovat na udalosť, viem si v HTML kóde nájsť všetkých 10 elementov pomocou `querySelectorAll`. keďže teraz pracujem s kolekciou (je ich viac, je to pole) tak každú položku individuálne spracujem cez `forEach`. je to furt to isté dokola! ak mám viac vecí: `forEach`. a teraz v cykle po jednom každý jeden z tej kolekcie 10 elemetov dostane svoj vlastný listener (vlastnú plošticu) ktorá bude načúvať, či nenastal klik. ak áno, spustí sa funkcia.

POKROČILÝ SPÔSOB: ak to chceš vylepšiť, vieš listener pridať rodičovskému elementu. otvor si HTML kód. vlastne celý čas maj kód otvorený. veci, ktoré v týchto textoch vysvetľujem, si skúšaj v kóde. v HTML kóde vidíš, že `li` elementy patria do `ul` zoznamu. prípadne do `.cards` elementu, ten je ďalší z rodičov. ty vieš prilepiť jeden listener na rodiča a vo funkcii potom kontrolovať, či klik konkrétne nastal na element typu `li`. **na domácu úlohu sa pokús zistiť, ako na to.** budeš chcieť použiť tzv. `target`. na to rozhodne budeš potrebovať:

EVENT OBJECT

takže ty počúvaš, či nastala udalosť a vtedy spustíš funkciu. prehliadač ti do tej funkcie prepošle tzv. `event object`. ak počúvaš, či nastala udalosť ťuknutia do klávesnice, často ťa bude zaujímať, ktorá konkrétne klávesa to bola. možno chceš iný kód spustiť pri stlačení klávesy ENTER a iný, pri stlačení klávesy ESCAPE. o každom evente vieš získať bonusové informácie. cez `event object`.

všetko zostáva rovnaké, len do zátvoriek funkcie napíšeš `event`:

```
header.addEventListener('click', (event) => {
  console.log(event); // bonusove info o evente, ak by ta zaujimalo
```

```
});
```

každá udalosť má k sebe pribalený `event` object, ktorý nesie bonusové info o udalosti. povedzme ak ťukneš do klávesnice, v `event` objekte nájdeš napríklad konkrétne ktorá klávesa to bola. ale získaš vďaka nemu aj ďalšie schopnosti. napríklad:

PREVENT DEFAULT

prehliadač má svoje vstavané správanie. keď máš v HTML kóde `<a>` element, je to odkaz. úloha prehliadača je po kliknutí na odkaz ťa zobrať na stránku, na ktorú odkaz smeruje. predstav si ale, že ty chceš napísať javascript funkciu, ktorá sa spustí po kliknutí na odkaz. **máš problém!** pretože úloha prehliadača je v prvom rade otvoriť stránku, na ktorú odkaz smeruje! takže tvoja funkcia sa možno spustí, možno sa nespustí, ty to nikdy neuvidíš, pretože už dávno sa tvoje oči pozerajú na inú stránku 😃

čiže občas - **hlavne pri práci s odkazmi a formulármi** - potrebuješ **zabrániť vstavanému správaniu prehliadača**. potrebuješ mu povedať: *"počuj, ja chápem, že ty máš nejakú robotu. ale nerob ju! namiesto toho spusti môj kód."*

cez **event.preventDefault()**:

```
const link = document.querySelector('a');

link.addEventListener('click', (event) => {
  event.preventDefault(); // zabrani prehliadaci otvorit inu stranku

  // teraz mozes pisat kod, ktory sa spusti po kliknuti na odkaz
});
```

tvoja úloha je odteraz si pozorne všímať všetky stránky a mobilné appky, ktoré používaš. **sleduj, ako na teba reagujú.** sleduj, ako sa veci dejú *po tom*, čo niekam klikneš, napíšeš text, potiahneš palcom dole po mobile, stlačíš button, nakloníš telefón, čo sa stane po ľavom a pravom kliku, pri scrolle, po pohybe myšou... dejú sa veci. tie veci sa nedejú samé od seba. niekto to musel naprogramovať. **napísať funkcie, ktoré sa spustia ako reakcia na udalosti**, ktoré používateľ vyvoláva. vďaka eventom vieš spustiť funkciu ako reakcia na používateľa. vďaka tomu sú tvoje aplikácie interaktívne. ty robíš veci pre človeka. a vďaka eventom človek vie tvoje veci používať.

FORM ELEMENT

predstav si registračný formulár. **je to formulár**. predstav si, že sa prihlasuješ do appky: meno a heslo zádavaš cez prihlasovací formulár. **je to formulár**. predstav si eshop: meno a adresu doručenia zadávaš cez objednávkový formulár. **je to formulár.**

opakujem slovo formulár. **formulár**.

opakujem slovo formulár, pretože keď získavaš údaje od používateľa, používaš HTML elementy ako `input`, `textarea`, `select`, `button`, `label`, ... a tie elementy dokopy tvoria **formulár** a formulárové elementy **patria do HTML `<form>` elementu.**:

```
<form action="/spracuj-data.php" method="get">
  <input type="text" name="title" placeholder="new title">
  <input type="text" name="content" placeholder="new text">
  <button type="submit">add new thing</button>
</form>
```

ak chceš userovi dať priestor písať text, použiješ `input` alebo `textarea`. ale `input` a `textarea` sú formulárové elementy a to znamená, že ich musíš obaliť do `form` elementu.

my tu robíme tzv. **frontend**. to, s čim požívateľ interaguje. napríklad formulár, do ktorého používateľ píše údaje. tie údaje sa odošlú na server, kde ich spracuje tzv. **backend**. môžeš to byť ty, môže to byť tvoj kolega. ale pointa je, že ten HTML zápis tam hore jasne hovorí, že textové údaje `title` a `content` (pretože `name="title"` a `name="content"`) sa majú odoslať na adresu `/spracuj-data.php`, kde ich tzv. server spracuje a napríklad uloží do databázy.

bez týchto údajov je tvoj HTML kód nesprávny a nekompletný. nerob také veci, že iba náhodne do HTML šľahneš jeden `input` element. formulárové elementy potrebujú `name=""` atribút a patria do `form` elementu. pretože potom prehliadač vie, ktoré údaje patria k sebe a vie s nimi správne pracovať.

NEVIEŠ, KTO POUŽÍVA TVOJU APPKU

na hodine som ukazoval, že niektorí ľudia sú zvyknutí odoslať formlár tak, že vyplnia údaje do inputov a stlačia ENTER. iní ľudia sú zvyknutí odoslať formár tak, že myšou stlačia button. **ak ty napíšeš správny HTML kód, prehliadač zaručí, že oba spôsoby automaticky budú fungovať.**

a teraz si zober, že existujú nevidiaci ľudia. a oni samozrejme tiež chcú používať internet. oni používajú špecializovaný softvér, ktoré im dokáže napríklad prečítať obsah stránky. a nevidiaci človek pravdepodobne formulár nebude odosielať stlačením tlačidla na myši.

a toto je dôvod, prečo ty nesmieš len tak náhodne šlahnúť jeden `input` a jeden `button` hocikde do HTML kódu. ty v javascripte síce vieš napísať kód, že *"ak user myšou stlačí button, tak nájdi text v inpute a spracuj ho"* ale to bude človeku, ktorý myš používať nemôže, veľký prd platné.

toto je dôvod, prečo sa musíš snažiť písať správny HTML kód. naša žltá playdate stránka by vizuálne vyzerala rovnako, keby `<form>` element z HTML kódu vynechám. a často, keď sa ľudia učia javascript (alebo react, ...) `<form>` elementy z kódu vynechávajú. **nerob to**.

ak používaš formulárové elementy ako `input`, tvoja úloha je umiestniť ich do elementu `form`. pretože ty ani netušíš, akí rôzni ľudia s akými rôznymi typmi akých rôznych softvérov tvoju appku budú používať. ale ak kód formuláru napíšeš správne, tak všetok ten softvér bude vedieť údaje spracovať. a bude schopný tie údaje odoslať.

v javascripte existuje event, ktorý vie odchytiť odoslanie formuláru bez ohľadu na to, akým spôsobom nastalo:

SUBMIT EVENT (a `input` a `change` eventy)

```
// najdes formular
const form = document.querySelector('form');

// submit event nastane pri odoslani formularu
form.addEventListener('submit', (event) => {
  // podobne ako pri odkazoch, uloha prehliadaca
  // je odoslat udaje na inu adresu
  // a prehliadac ta na tu adresu vezme spolu s nimi
  // takze ak chces udaje spracovat javascriptom ty, musis:
  event.preventDefault();

  // a teraz klasicky napises, co potrebujes
});
```

keď chceš javascriptom spracovať údaje z formulára, používaš `submit` event.

v predošlom texte som spomínal spotify: keď tam začneš písať do inputu, spustí sa vyhľadávanie. ak chceš reagovať na *akt písania* do inputu, môžeš na input cez `addEventListener` naviazať `input` event. ak chceš reagovať na *zmenu hodnoty* vo formulárovom elemente, môžeš použiť `change` event.

pri formulároch budeš asi najčastejšie používať `submit` event. ak chceš reagovať na zmeny hodnôt v inputoch samotných, často budeš používať `input` alebo `change` eventy. znova opakujem: existuje mdn, existuje google, chatgpt... a tvoja robota je ich používať.

ROBOTA PROGRAMÁTORA

znova opakujem, že úloha programátora je neustále čítať. neustále si dohľadávať nové informácie a neustále skúšať písať kód. programátor viac kódu prečíta ako napíše.

preto ti tu teraz nebudem podrobne na príkladoch vysvetľovať `input` a `change` eventy. **namiesto toho budem očakávať, že si informácie o nich vyhľadáš, a že sa ich pokúsiš použiť v tvojom kóde :slight_smile:** pretože práca programátora neni písať kód jak robot. práca programátora je hľadať informácie, pokúšať sa ich použiť a pokúšať sa ich pochopiť.

nefunguje to tak, že ty sa naučíš, čo máš písať a potom to píšeš. ak by tá robota bola takto chladná a presná, tak by ju fakt mohol robiť počítač sám a človek by nebol potrebný. funguje to tak, že ty dostaneš úlohu a netušíš, ako ju spraviť a dohľadávaš si informácie a skúšaš ich použiť tak dlho, až to začne fungovať ✌️

EVENTY V PREHLIADAČI

ja som sa tu zameral na eventy ako `click`, ktoré "spôsobuje" používateľ svojím klikaním a plieskaním po klávesnici a tak. ale eventy neustále nastávajú aj v prehliadači. eventy ako `offline` a `online`: prehliadač vyšle signál, keď príde o sieť alebo keď sa naopak pripojí na internet. prípadne `DOMContentLoaded` event sa odpáli, keď prehliadač zanalyzuje všetky HTML elementy v tvojom kóde a vzťahy medzi nimi. ak napríklad na obrázok pripneš `load` event, vieš spustiť funkciu v momente, keď je obrázok kompletne stiahnutý. ak prehrávaš video alebo audio, po konci prehrávania sa odpáli `ended` event... vieš spustiť javascript funkciu v momente, keď napríklad skončí animácia. istotne nájdeš príklady aj v https://animate.style/ a https://gsap.com dokumentáciách.

POZOR NA ELEMENTY Z BUDÚCNOSTI !!

v minulom texte som ukázal príklad, kde v HTML kóde existujú 3 `li` a na každý z nich som pripol listener na click event. to znamená, že existujú 3 elementy, ktoré počúvajú, či na ne niekto klikol.

čo ak neskôr pribudne štvrtý `li` element? napríklad vytvorený na základe údajov, ktoré `fetch`neme z externého zdroja? bude aj tento nový, štvrtý element reagovať na kliknutie? 😏

často sa ti stane, že napíšeš kód, elementy reagujú na udalosti, všetko ok. a potom si všimneš, že niektoré z nich nereagujú. a nechápeš prečo. a trháš si vlasy. pozeráš sa na ne očami a hovoríš *"všetky z nich vypadajú rovnako, prečo niektoré reagujú a iné nie?"* **vtedy si daj otázku: existovali tieto elementy v momente, keď sa rozdávali štěnice?**

možno teraz nevieš, o čom hovorím. ako vždy: ty si neučíš, keď ma počúvaš. ty sa neučíš, keď čítaš moje články. ty sa učíš vtedy, keď skúšaš písať kód. a keď budeš skúšať písať kód a strestneš sa so situáciou, že niektoré elementy reagujú a iné nie, vtedy si spomeň na tento text 😌