Django

Curs 10,11

pentru intrebari: irina.ciocan@gmail.com

Cuprins

Observații. Dacă deschideți cursul în Google Slides, faceți click pe "Present" (dreapta-sus) pentru a parcurge cursul cum a fost intenționat (cu animații, linkuri și alte efecte). Linkurile de mai jos sunt doar către începutul unei secțiuni de curs (uneori, o secțiune se întinde pe mai multe slide-uri pe care trebuie să le parcurgeți). Fiecare slide are link înapoi către cuprins.

1.

irina.ciocan@gmail.com

Coș virtual

Cuprins

Un coș virtual este un spațiu unde clienții își adună produsele pe care doresc să le cumpere, înainte de a finaliza comanda. Funcționează similar unui coș de cumpărături dintr-un magazin fizic, doar că în format digital.

Implementarea unui coș virtual presupune o interfață intuitivă și ușor de utilizat, unde clienții pot adăuga produse în coș, modifica cantitățile, elimina produse și vedea un rezumat al comenzii.

Avantajele unui coș virtual:

- Îmbunătățește experiența utilizatorului: Un coș ușor de utilizat îi încurajează pe clienți să finalizeze cumpărăturile.
- Reduce abandonul cumpărăturilor: Un coș bine optimizat poate reduce numărul de clienți care abandonează cumpărăturile spre deosebire de site-urile în care utilizatorul trebuie să listeze printr-un mesaj ce dorește să comande.
- Crește vânzările: Un coș funcțional și intuitiv, eventual care oferă facilități pe care alte site-uri nu le oferă, sau prin care pot fi accesate sugestii către produse similare, poate duce la creșterea vânzărilor.

irina.ciocan@gmail.com

Coș virtual - funcționalități

Cuprins

Funcționalități esențiale:

- **Adăugare produse**: Clienții pot adăuga produse în coș printr-un buton sau link.
- **Modificare cantități**: Clienții pot modifica numărul de produse pentru fiecare articol din coș.
- **Eliminare produse**: Clienții pot elimina produse din coș dacă își schimbă decizia.
- **Calculul automat al prețurilor**: Coșul trebuie să calculeze automat prețul total al comenzii, inclusiv taxe și reduceri.
- **Aplicarea de cupoane**: Dacă există cupoane sau coduri promoționale, coșul trebuie să le poată aplica și să recalculeze prețul.
- **Integrare cu alte sisteme**: Coșul virtual trebuie să fie integrat cu alte sisteme ale magazinului online, precum:
  - Catalogul de produse: Pentru a afișa informații corecte despre produse.
  - Sistemul de plăți: Pentru a procesa plăți online.
  - Sistemul de inventar: Pentru a verifica disponibilitatea produselor.
  - Securitate: Este esențial ca coșul virtual să fie securizat pentru a proteja datele clienților și a preveni fraudele.

irina.ciocan@gmail.com

JS client - localStorage

Cuprins

Datele stocate în localStorage rămân disponibile chiar și după ce utilizatorul închide browserul și îl redeschide. Ele sunt eliminate doar dacă utilizatorul șterge manual cache-ul browserului sau dacă sunt eliminate explicit prin codul aplicației.

- Limita localStorage-ului este în general de aproximativ 5-10 MB pe domeniu, în funcție de browser.
- Datele sunt accesibile de pe toate filele și ferestrele care au același protocol (http/https) și domeniu.

- Este organizat sub formă de chei unice cu valori asociate.
- Atât cheile cât și valorile sunt stringuri

- Numărul de perechi cheie-valoare stocate poate fi obținut prin proprietatea localStorage.length

Observație: localStorage este pe mașina clientului; deci serverul nu are acces la el

Metode localStorage - setItem, getItem

Cuprins

setItem(key, value) - Salvează o valoare asociată cu o cheie specifică în localStorage. Dacă cheia există deja, valoarea este suprascrisă.
Argumente:

- key (string): Numele cheii.
- value (string): Valoarea asociată cheii. Variabila value trebuie să fie de tip string; dacă nu este, va fi convertită implicit.

```
localStorage.setItem('cos_virtual', '1,4,5');
localStorage.setItem('username', 'Alex');
```

getItem(key) - Returnează valoarea asociată cu o cheie specifică. Argumentul key este de tip string și reprezintă numele cheii. Returnează valoarea asociată cheii sau null dacă nu există cheia.

```
let id_uri = localStorage.getItem('cos_virtual');
```

Metode localStorage - key

Cuprins

key(index) - returnează numele (cheia) unei intrări pe baza indexului său. Argumentul *index* este un număr întreg și reprezintă poziția intrării în localStorage. Indexul începe de la 0 și este bazat pe ordinea în care au fost adăugate datele.

Returnează *null* dacă indexul nu este valid.

```
localStorage.setItem('cheie1', 'valoare1');
localStorage.setItem('cheie2', 'valoare2');

console.log(localStorage.key(0)); // 'cheie1'
console.log(localStorage.key(1)); // 'cheie2'
```

Metode localStorage - removeItem, clear

Cuprins

removeItem(key) - elimină o pereche cheie-valoare din localStorage pe baza cheii. Argumentul key este de tip string și reprezintă numele cheii care trebuie eliminată.

```
localStorage.removeItem('cos_virtual');
```

clear() - șterge toate datele din localStorage pentru domeniul curent.

```
localStorage.clear();
```

După apelul metodei clear() lungimea localStorage-ului devine 0.

localStorage - observații

Cuprins

- Dacă trebuie să stocați obiecte sau alte tipuri de date, utilizați metode precum JSON.stringify() și JSON.parse()

```
let obiect = { nume: 'Irina', varsta: 38 };
localStorage.setItem('utilizator', JSON.stringify(obiect));

let utilizator = JSON.parse(localStorage.getItem('utilizator'));
console.log(utilizator.nume);
```

- localStorage este accesibil de pe orice script care rulează în aceeași origine (domeniu), deci evitați stocarea informațiilor sensibile, cum ar fi parolele sau token-urile de autentificare.
- localStorage sincronizează automat modificările între paginile ale aceluiași domeniu. Dacă o valoare este actualizată într-o pagină, alte pagini deschise pot primi evenimentul storage pentru a detecta modificarea.

```
window.addEventListener('storage', function(event) {
    console.log(`Cheia ${event.key} a fost actualizata la ${event.newValue}`);
});
```

Metoda fetch

Cuprins

Metoda fetch este folosită pentru a trimite cereri HTTP către un server și pentru a primi răspunsuri. Implicit, aceasta face o cerere de tip GET (dacă nu este specificat altceva). Este preferată față de metodele mai vechi precum XMLHttpRequest datorită interfeței sale moderne bazate pe obiecte de tip Promise.

Sintaxă:

```
fetch(url, [optiuni])
```

Argumente:

- url: URL-ul (relativ sau absolut) către care se face cererea.
- opțiuni: un obiect care conține opțiuni pentru configurarea cererii, cum ar fi metoda, anteturile, corpul cererii, etc.

fetch - exemplu de tip GET

Cuprins

Considerăm că în exemplul de mai jos primim niste date în format JSON:

```
fetch('http://localhost:8000/aplicatie_exemplu/proceseaza_date')
  .then(raspuns => raspuns.json())
  .then(date => console.log(date))
  .catch(eroare => console.error('Eroare:', eroare));
```

Variabila raspuns este de tip Response, un obiect care conține datele returnate de server. Pentru a extrage datele, de obicei folosim metode precum response.json(), response.text() sau response.blob().

Metoda fetch returnează un obiect de tip Promise. Aceasta este fie rezolvată (dacă cererea are succes), fie respinsă (în caz de eroare de rețea sau altă problemă).

În caz de succes, cu raspuns.json() convertim datele în obiect pe care îl putem apoi accesa în variabila date (care, în exemplu este afișată în consolă).

Putem trata și erorile prin metoda catch. Eventuala eroare este stocată în acest exemplu în variabila *eroare*.

fetch - exemplu de tip GET cu parametri de query

Cuprins

Pentru a adăuga parametri de query la URL:

```
const url = new URL('http://localhost:8000/aplicatie_exemplu/proceseaza_date');
url.searchParams.append('a', '10');
url.searchParams.append('b', '1');
url.searchParams.append('text', 'ceva');

fetch(url)
  .then(raspuns => raspuns.json())
  .then(date => console.log(date))
  .catch(eroare => console.error('Eroare:', eroare));
```

irina.ciocan@gmail.com

Obiectul Response

Cuprins

Proprietăți și metode ale obiectului Response:

- response.ok: true dacă cererea a fost reușită (status 200-299).
- response.status: Codul de status HTTP (ex. 200, 404, 500).
- response.statusText: Text asociat cu codul de status (ex. "OK", "Not Found").
- response.headers: Obiect ce conține anteturile răspunsului.
- response.json(): Metodă pentru a obține conținutul răspunsului în format json.
- response.text(): Metodă pentru a obține conținutul răspunsului în format text.
- response.blob(): Metodă pentru a obține conținutul răspunsului în format blob.

Exemplu fetch cu await

Cuprins

Uneori ne este mai ușor să folosim fetch într-o funcție asincronă și să primim răspunsul returnat de fetch, apelând funcția cu cuvântul cheie await:

```
async function obtineDate() {
    try {
        const raspuns = await fetch('http://localhost:8000/aplicatie_exemplu/proceseaza_date');
            if (!raspuns.ok) {
                throw new Error(`Eroare HTTP. Status: ${raspuns.status}`);
            }
        const date = await raspuns.json();
        console.log('Date primite:', date);
        }
    catch (eroare) {
        console.error('A apărut o eroare:', eroare);
    }
}
obtineDate();
```

Fetch cu POST

Cuprins

Funcția fetch pentru cererile HTTP de tip POST este utilizată pentru a trimite date către un server. Spre deosebire de cererile GET, care doar obțin informații, cererile POST sunt folosite, de obicei, pentru a crea resurse sau pentru a trimite date ce necesită procesare, cum ar fi date din formulare, date JSON etc.

De obicei, apelul fetch cu metoda POST are formatul:

```
fetch(url, {
    method: 'POST',
    headers: {},
    body: date
});
```

Argumente:

- url: adresa către care se face cererea.
- headers: anteturile cererii (de exemplu, Content-Type). Mai des setate pentru cereri de tip POST decât GET
- body: Datele care urmează să fie trimise către server. Acestea pot fi stringuri, JSON, sau alte tipuri de date.

Exemplu fetch cu POST

Cuprins

```
fetch('http://localhost:8000/aplicatie_exemplu/proceseaza_date', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        nume: 'Irina',
        varsta: 38
    })
})
    .then(raspuns => {
        if (!raspuns.ok) {
            throw new Error(`Eroare HTTP. Status: ${raspuns.status}`);
        }
        return raspuns.json();
    })
    .then(date => {
        console.log('Raspuns:', date);
    })
    .catch(eroare => {
        console.error('Eroare:', eroare);
    });
```

Se vor mai adauga

Cuprins

- Se vor mai adauga slide-uri pentru saptamana 11

Bibliografie și alte resurse

Cuprins

- https://developer.mozilla.org/en-US/docs/Web/API/Storage_API/Storage_quotas_and_eviction_criteria
- https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API