

Персистиращи локални данни

Методи и добри практики за използване на локални
данни в браузъра

Въведение

В тази презентация ще разгледаме различните методи за съхраняване на данни в браузъра, включително `LocalStorage`, `SessionStorage`, `IndexedDB` и `Cache Storage`. Ще обсъдим тяхната функционалност, ограниченията им и добри практики при тяхното използване.

Какво означава

"персистиране на данни"?

Персистиране на данни = съхраняване на информация в браузъра, така че да остане дори след презареждане или затваряне на страницата.

Използва се за:

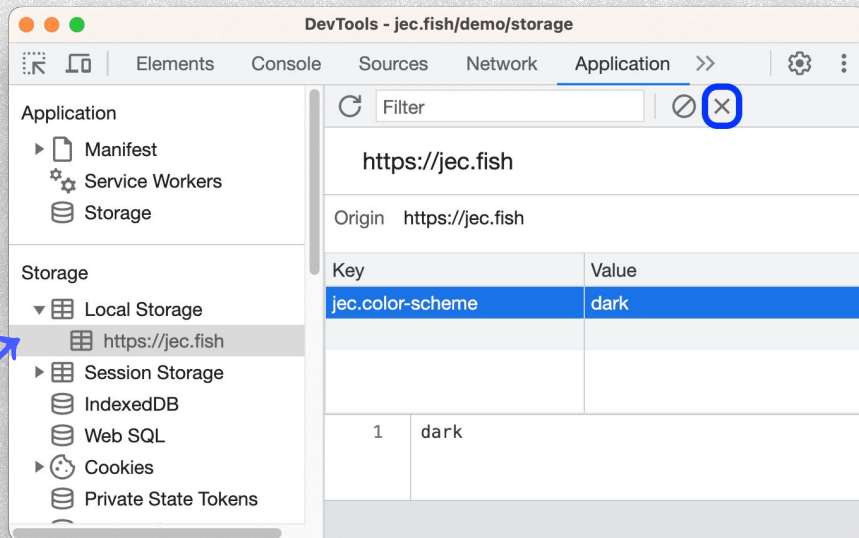
- Запазване на потребителски настройки (език, тема)
- Данни на клиенти при SPA (Single Page Applications)
 - Офлайн достъп

01

Методи за съхранение

01.1. Какво е LocalStorage ?

- Част от Web Storage API
- Съхранява ключ-стойност двойки
- Данните остават, докато не ги изтрием
- Ограничение: ~5MB
- Работи синхронно (blocking)



Как се използва Local Storage ?

```
// Записване
localStorage.setItem('theme', 'dark');

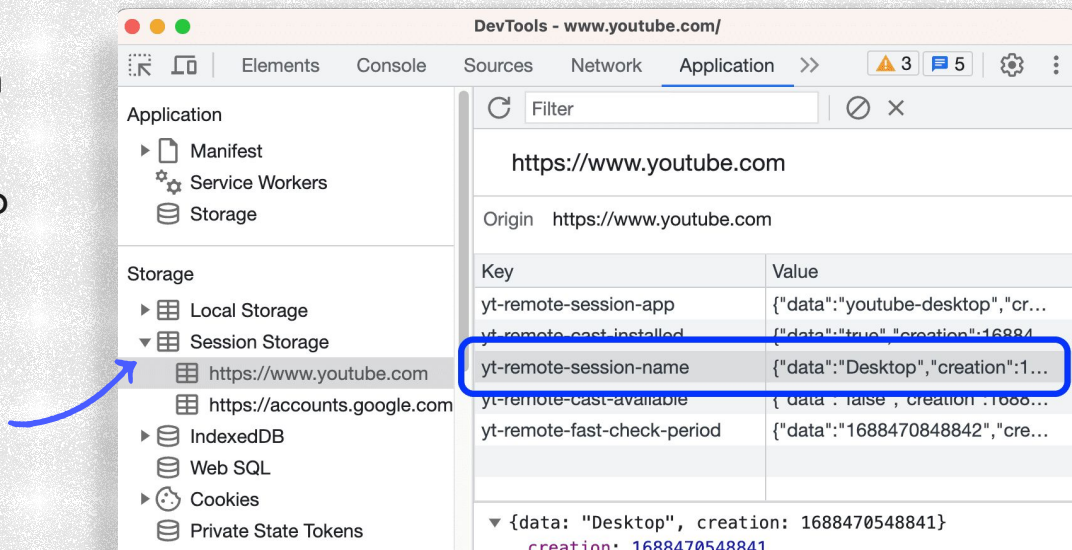
// Прочитане
const theme = localStorage.getItem('theme');

// Изтриване
localStorage.removeItem('theme');

// Изчистване на всички ключове
localStorage.clear();
```


01.2. Какво е SessionStorage ?

- Част от Web Storage API
- Съхранява ключ-стойност двойки (винаги низове)
- Данните остават, докато прозорецът е отворен, след затваряне се изтриват
- Ограничение: ~5MB
- Няма споделяне между табове
- Работи синхронно



ОСНОВНИ МЕТОДИ

```
// Записване на стойност под ключ
sessionStorage.setItem('username', 'fmiStudent123')

// Четене на стойността, записана под даден ключ
let username = sessionStorage.getItem('username');
console.log(username); // "fmiStudent123"

// Съхраняване и четене на обект (с конвертиране)
let user = { name: "Ivan", age: 30 };
sessionStorage.setItem('user', JSON.stringify(user));

let storedUser = JSON.parse(sessionStorage.getItem('user'));
console.log(storedUser.name); // "Ivan"
```


Основни методи

```
// Изтриване на ключ и стойност
sessionStorage.removeItem('user');

// Изтриване на всички записи в sessionStorage
sessionStorage.clear();

// Проверяваме дали има запис
if (sessionStorage.getItem('user')) {
    console.log('Потребителят е в сесията.');
```



```
} else {
    console.log('Няма потребител в сесията.');
```



```
}
```


✓ Кога е подходящо да използваме SessionStorage?

- Когато данните трябва да се запазват само докато табът е отворен
- За формуляри, филтри, търсения и други временни действия.
- При многостъпкови процеси (напр. попълване на анкета в няколко стъпки)

! Ограничения

- Данните изчезват при затваряне на таба
- Не е подходящ за голям обем информация
- Уязвим е при XSS атаки — данните могат да бъдат откраднати

01.3.

IndexedDB

Какво е IndexedDB?

- Вградена в браузъра база данни за съхранение на големи количества данни от клиентската страна
- Поддържа сложни структури и работи офлайн
- Асинхронна – не блокира потребителския интерфейс
- Използва се в приложения, които изискват локална устойчивост и бърз достъп до данни

Основни концепции

- database – съдържа обекти и техните данни
- object store – еквивалент на „таблица“, но по-гъвкава
- key – уникален идентификатор за всеки обект
- index – позволява търсене по не-ключови свойства
- transaction – всички операции минават през транзакции
- request – всички действия са асинхронни и връщат IDBRequest

Предимства

- Поддържа голям обем данни
- Може да съхранява сложни обекти (включително файлове, изображения, JSON)
- Работи офлайн
- Може да се комбинира със Service Workers за PWA приложения

Недостатъци

- Сложен API – използва събития вместо `async/await`
- Изисква повече код в сравнение с `LocalStorage` или `SessionStorage`
- Не е напълно еднакво реализирана във всички браузъри
- Може да е трудна за начинаещи без помощни библиотеки

Пример (съхранение на задачи в IndexedDB)

```
1  const request = indexedDB.open("MyDatabase", 1);
2
3  request.onupgradeneeded = function (event) {
4    const db = event.target.result;
5    const store = db.createObjectStore("tasks", { keyPath: "id" });
6    store.createIndex("by_status", "status");
7  };
8
9  request.onsuccess = function (event) {
10   const db = event.target.result;
11   const tx = db.transaction("tasks", "readwrite");
12   const store = tx.objectStore("tasks");
13   store.add({ id: 1, title: "Презентация", status: "готово" });
14  };
```


Библиотеки за улеснение

Dexie.js – популярен wrapper, който опростява работата с IndexedDB

Пример с Dexie.js:

```
1  const db = new Dexie("TasksDB");  
2  db.version(1).stores({ tasks: "++id,title,status" });  
3  await db.tasks.add({ title: "IndexedDB част", status: "в процес" });
```


Кога и как да използваме IndexedDB

Кога е подходяща:

- Когато данните са големи или сложни
- Когато е нужен офлайн достъп
- За бележници, задачи, чатове, PWA

Добри практики:

- Използвай Dexie.js за опростен синтаксис
- Обработвай грешки чрез `onerror`
- Проверявай транзакциите
- Поддържай версия и миграции на базата

01.4.

Cache Storage

Какво е браузърски cache?

- Временна памет на локалния диск, в която браузърът запазва копия на вече изтеглени ресурси като HTML, CSS, изображения и дори HTTP заявки с цел даденият ресурс да се зареди директно от диска при повторно посещение, като това ускорява страницата.
- Браузърът сам решава какво да запази в кеша и за колко време, като това решение зависи от инструкции, които уеб сървърът изпраща съдържайки се в HTTP заглавки (Cache-Control, Expires, ETag, Last Modified).

Какво е Cache API ?

- Cache API е JavaScript интерфейс, който позволява на уеб приложенията да съхраняват и извличат HTTP заявки и отговори. Полезно е за **офлайн достъп**. Cache API е част от спецификацията на **Service Worker**.
- Основните методи са:
 - `caches.open(cacheName)` - отваря (или създава) кеш с дадено име.
 - `cache.add(request)` - изтегля ресурс от мрежата и го добавя в кеша.
 - `cache.put(request, response)` - добавя конкретна заявка и отговор в кеша.
 - `cache.match(request)` - търси и връща отговора от кеша за дадена заявка.
 - `cache.delete(request)` - премахва конкретна заявка от кеша.

Пример за използване на Cache API?

```
1  const staticResources = [  
2    '/',  
3    '/index.html',  
4    '/styles.css',  
5    'script.js',  
6    '/offline.html',  
7    '/images/logo.png'  
8  ];  
9  
10 window.addEventListener('load', () => {  
11   if('caches' in window){  
12     caches.open('site-static-v1').then(cache => {  
13       return cache.addAll(staticResources);  
14     }).catch(error => {  
15       console.error(error);  
16     });  
17   }  
18 });
```


Какво е Service Worker?

- JavaScript, който браузърът изпълнява, отделно от основната уеб страница. Той действа като посредник между уеб приложението и мрежата, позволявайки:
 - Прихващане на мрежови заявки и да ги обработва по собствен начин;
 - Кеширане на ресурси за офлайн достъп;
 - Синхронизиране на данни във фонов режим.
- Изисква HTTPS за сигурност: Работи само на защитени сайтове (https://), защото прихваща мрежови заявки и може да контролира съдържание.

02

Сравнение и добри практики

Сравнение на LocalStorage, SessionStorage, IndexedDB и Cache API

Метод	Капацитет	Достъпност	Времетраене	Тип данни	Асинхронност
LocalStorage	~5MB	Всички браузъри	Персистиращ (завинаги)	Ключ-стойност (string)	Синхронна
SessionStorage	~5MB	Всички браузъри	Само за сесията	Ключ-стойност (string)	Синхронна
IndexedDB	Десетки/стотици MB	Всички браузъри (но с разлики)	Персистиращ	Сложни обекти, файлове	Асинхронна
Cache API	Зависи от браузър	Само с Service Worker	Зависи от стратегията	HTTP заявки/отговори	Асинхронна

Оптимални случаи на употреба

LocalStorage:

Потребителски настройки (тема, език).

Малки, прости данни (напр. конфигурации).

SessionStorage:

Временни данни (напр. форма за многостъпково попълване).

IndexedDB:

Големи обеми данни (офлайн PWA, чатове, бележници).

Сложни заявки и транзакции.

Cache API:

Офлайн ресурси (CSS, JS, изображения).

Динамични заявки с Service Worker.

Потенциални рискове и как да ги избегнем

XSS (Cross-Site Scripting):

LocalStorage/SessionStorage са уязвими към XSS атаки → не съхранявай чувствителни данни (токени, пароли).

Достъп до данни:

Всички локални данни са достъпни чрез DevTools → криптирай важна информация.

Изчистване на кеша:

Данните могат да се изгубят, ако потребителят изтрие историята → винаги имай fallback към сървър.

Примери за ефективно комбиниране

LocalStorage + IndexedDB: Приложение за бележки (Notes App)

Съхранява потребителски настройки (тема, език) (LocalStorage) +
Съхранява бележките (голям обем данни) (IndexedDB).

```
// LocalStorage: Запазване на темата
localStorage.setItem('theme', 'dark');

// IndexedDB: Създаване на база данни за бележки (с Dexie.js за опростяване)
const db = new Dexie('NotesDB');
db.version(1).stores({ notes: '++id, title, content' });

// Добавяне на бележка
await db.notes.add({ title: 'Купи храна', content: 'Мляко, яйца, плодове' });
```


Примери за ефективно комбиниране

SessionStorage + HTTP кеш: Многостъпкова форма за покупка

Временно запазва данни между стъпките (SessionStorage).+
бързо зареждане на ресурси (HTTP кеш).

```
// SessionStorage: Запазване на данни от стъпка 1
sessionStorage.setItem('userDetails', JSON.stringify({ name:
'Иван', email: 'ivan@example.com' }));
```

```
// Вземане на данни при следваща стъпка
const userDetails =
JSON.parse(sessionStorage.getItem('userDetails'));
```


Ключови изводи

1. Изборът зависи от нуждите: простота vs. мощност.
2. Бъдете наясно с рисковете откъм сигурността (XSS, криптиране).
3. В повечето случаи ще ви се налага да комбинирате няколко технологии за съхранение на данни.

Благодарим!

Изготвено от екип 6:

Ани Божкова

Марина Кръстева

Деян Цветанов

Стефан Иванов

Енис Моллаахмед

CREDITS: This presentation template was created by [Slidesgo](#), and includes icons by [Flaticon](#), and infographics & images by [Freepik](#)