Тема 17. Хранение и загрузка данных .

хекслет колледж

Цель занятия:

Сформировать представление о клиентском хранилище веб-приложений.

Учебные вопросы:

1. Введение: Проблема потери данных.
2. Основы localStorage
3. Работа со сложными данными: JSON
4. Типовой алгоритм интерактивного приложения

1. Введение: Проблема потери данных.

Жизненный цикл данных в JavaScript.

Когда мы создаем переменные, массивы или объекты в коде JS, они живут в оперативной памяти браузера.

- Как это работает: Пока вкладка открыта, данные доступны.
- Проблема: Как только пользователь нажимает кнопку «Обновить» (F5), закрывает вкладку или браузер — оперативная память очищается. Все состояния приложения сбрасывается к исходному коду.

Зачем нам «длинная память»?

Для разработки современных интерфейсов важно, чтобы сайт «узнавал» пользователя и помнил его действия.
Примеры из практики:

- Настройки интерфейса: Пользователь переключил сайт на «Темную тему». Без сохранения данных, при переходе на следующую страницу его снова «ослепит» белым фоном.
- Длинные формы: Пользователь заполняет анкету из 20 полей, случайно обновил страницу — все введенные данные исчезли. Это плохой UX (пользовательский опыт).
- Корзина товаров: В интернет-магазине товары в корзине не должны пропадать, если пользователь закроет сайт и вернется через час.
- Список задач (To-Do List): Классическая учебная задача. Пользователь записывает планы на день, и они должны храниться в браузере «навечно», пока он сам их не удалит.

Краткоживущие vs Долговременные данные

Тип данных	Где храним	Сколько живут	Пример
Временные	Переменные (let, const)	До первой перезагрузки страницы	Текущий текст в строке поиска
Сессионные	sessionStorage	Пока открыта вкладка	Результат фильтрации в текущем поиске
Долговременные	localStorage	Годами (пока не очистят кэш)	ID пользователя, выбранный язык, товары в корзине

Пример для демонстрации

Откройте консоль и введите:
let userName = "Алексей";
console.log("Меня зовут: " + userName);

Затем нажмите F5 и снова введите:
console.log(userName); // Ошибка: userName is not defined

Вывод: Обычные переменные не подходят для сохранения состояния приложения между визитами.
Для решения этой проблемы нам нужны специальные хранилища браузера, к изучению которых мы переходим.

Место хранения в браузере (DevTools)

Где физически можно увидеть эти данные?

- Открываем Инструменты разработчика (F12).
- Переходим на вкладку Application (в Chrome/Edge) или Storage (в Firefox).
- В левом меню находим раздел Local Storage.

Важно: Данные в localStorage привязаны к домену (протокол + имя сайта + порт). Данные с google.com никогда не будут доступны на yandex.ru

Вывод по вопросу:

- **Проблема**: Обычные переменные JavaScript живут только в оперативной памяти. Любая перезагрузка страницы или закрытие вкладки полностью уничтожает текущее состояние приложения.
- **Решение**: Для создания качественных веб-приложений (настроек темы, корзин товаров, списков дел) нам необходимо использовать клиентские хранилища, которые позволяют записывать данные на «жесткий диск» браузера.
- **Инструмент**: В рамках нашего курса основным инструментом станет **localStorage** — самое простое и мощное хранилище для долговременных данных в формате «ключ — значение».

2. Основы localStorage

localStorage - Это встроенное в браузер хранилище данных, которое позволяет сайтам сохранять информацию на устройстве пользователя.
Ключевые особенности:

- Объем: Можно сохранить около 5–10 МБ данных (это очень много для текста).
- Долговечность: Данные не имеют срока годности. Они останутся там, даже если вы выключите компьютер.
- Формат: Данные хранятся в виде пар «Ключ — Значение» (как обычный объект JS, но с нюансами).
- Только строки: Это самое важное ограничение. И ключ, и значение обязательно должны быть строками.

Основные методы (API)

Для управления данными нам нужно всего 4 метода глобального объекта localStorage.

А) Запись данных: setItem()
Принимает два аргумента: имя ключа и строку, которую нужно сохранить. Если ключ уже существует, значение перезапишется.

```
localStorage.setItem('theme', 'dark');
localStorage.setItem('user_name', 'Anton');
```

Б) Чтение данных: getItem()
Принимает один аргумент — имя ключа.
Возвращает строку.

```
const currentTheme = localStorage.getItem('theme');
console.log(currentTheme); // Выведет: "dark"
```

Важно: Если ключа не существует, метод вернет null. Это стандартный способ проверить, сохранял ли пользователь что-то ранее.

В) Удаление данных: removeItem()

Удаляет конкретную пару ключ-значение.

```
localStorage.removeItem('theme'); // Больше темы нет
```

Г) Полная очистка: clear()

Удаляет вообще всё, что ваш сайт сохранил в localStorage. Используйте осторожно!

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

Нюанс: Приведение к строке.

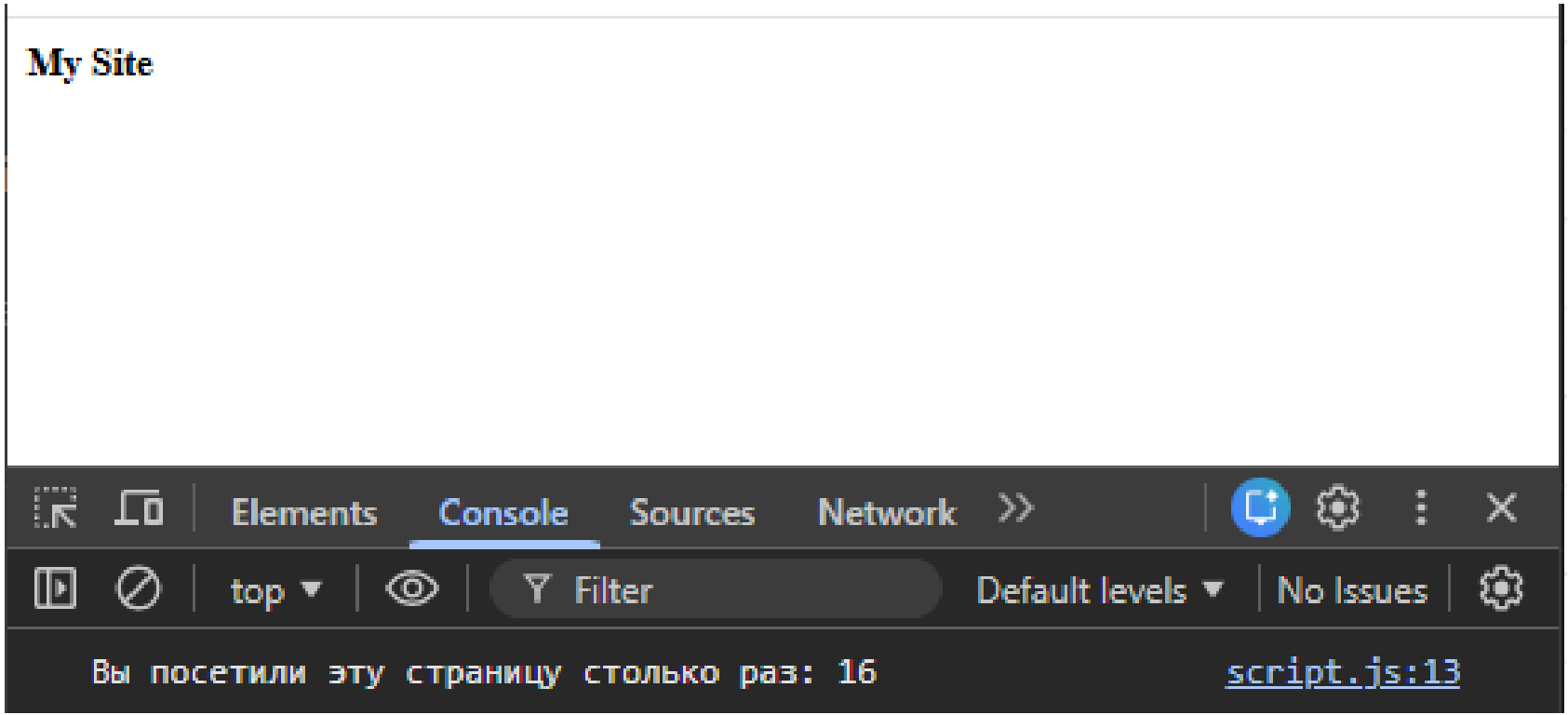
Если вы попытаетесь сохранить в localStorage число или булево значение, браузер автоматически превратит их в строку.

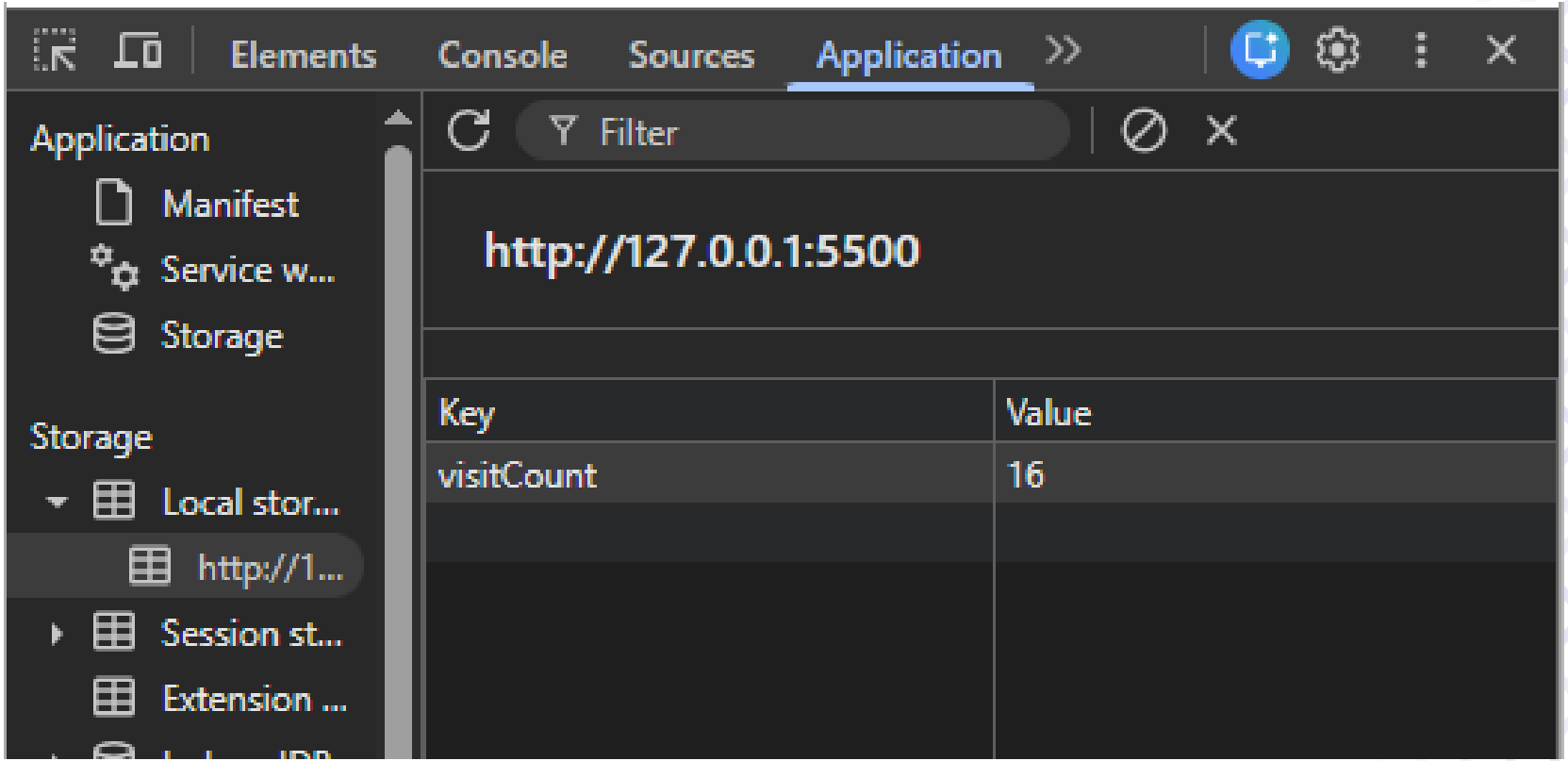
```
localStorage.setItem('isLogged', true);
let res = localStorage.getItem('isLogged');

console.log(typeof res); // "string", а не "boolean"!
console.log(res === true); // false, потому что "true" !== true
```

Эту проблему мы решим в блоке про JSON, но сейчас запомните: на выходе всегда строка.

Пример: "Счетчик визитов"

```
// 1. Пытаемся достать старое значение
let visits = localStorage.getItem('visitCount');

// 2. Если данных нет (первый раз на сайте), ставим 0
if (visits === null) {
    visits = 0;
}

// 3. Увеличиваем и сохраняем обратно
visits = Number(visits) + 1;
localStorage.setItem('visitCount', visits);

console.log(`Вы посетили эту страницу столько раз: ${visits}`);
```





Вывод:

- localStorage работает по принципу простого словаря: даем уникальное имя (ключ) и кладем туда данные (значение).
- API предельно простое: используем setItem для сохранения и getItem для получения данных.
- Главная ловушка: всегда нужно помнить, что хранилище работает только со строками. Если вы положили туда число 5, достанете вы оттуда строку "5".

3. Работа со сложными данными: JSON

Проблема: Объект в localStorage.

Как мы уже знаем, localStorage превращает всё в строку. Если мы попытаемся сохранить массив или объект «напрямую», произойдет следующее:

```
const user = { name: "Ivan", age: 25 };
localStorage.setItem('user_data', user);

const savedUser = localStorage.getItem('user_data');
console.log(savedUser); // Выведет: "[object Object]"
```

Строка "[object Object]" — это просто текстовое описание типа данных. Сами данные (имя и возраст) потеряны навсегда, их нельзя превратить обратно в объект.

Что такое JSON?

JSON (JavaScript Object Notation) — это формат данных, который представляет собой строку, написанную по правилам синтаксиса объектов JavaScript.

- Это «универсальный язык» для передачи данных.
- Нам он нужен как «контейнер», чтобы упаковать сложный объект в строку.

Инструменты: JSON.stringify и JSON.parse

Для работы с хранилищем нам нужны два метода глобального объекта JSON:

А) Упаковка: JSON.stringify(value)
Превращает любой объект или массив в JSON-строку.

```
const cart = ['Apple', 'Banana', 'Orange'];
const jsonString = JSON.stringify(cart);

console.log(jsonString); // '["Apple","Banana","Orange"]' — теперь это строка!
localStorage.setItem('myCart', jsonString);
```

Б) Распаковка: JSON.parse(string)

Берет JSON-строку и превращает её обратно в полноценный объект или массив JavaScript.

```
const dataFromStorage = localStorage.getItem('myCart');
const originalArray = JSON.parse(dataFromStorage);

console.log(originalArray[1]); // "Banana" — снова работает как массив!
```

Стандартный паттерн (Алгоритм работы):

- При чтении: Получаем строку из getItem и сразу пропускаем через JSON.parse.
- При записи: Берем объект/массив и сразу пропускаем через JSON.stringify перед setItem.

Важный совет: Всегда проверяйте данные на null. Если в хранилище еще ничего нет, JSON.parse(null) не выдаст ошибку, но вернет null, что может сломать дальнейшую логику.

Пример: Сохранение профиля.

```
const profile = {
    theme: 'dark',
    fontSize: 16,
    notifications: true
};

// Сохраняем
localStorage.setItem('settings', JSON.stringify(profile));

// Читаем (с подстраховкой)
const raw = localStorage.getItem('settings');
const settings = raw ? JSON.parse(raw) : { theme: 'light' }; // Значение по умолчанию

console.log(settings.theme);
```

Вывод:

- JSON — это мост между сложными данными (объектами/массивами) и строковым хранилищем localStorage.
- Метод stringify используется «на входе» в хранилище (упаковываем данные).
- Метод parse используется «на выходе» (распаковываем данные для работы в коде).
- Без этой связки невозможно реализовать корзину покупок или список задач.

4. Типовой алгоритм интерактивного приложения

1. Фаза Инициализации (Старт страницы)
Этот код должен стоять в самом начале скрипта. Его задача — восстановить то, что было сохранено ранее.
Алгоритм:

- Пытаемся получить данные из localStorage по ключу (например, 'tasks').
- Проверяем: если данные есть — парсим их (JSON.parse). Если данных нет — создаем пустую заготовку (например, пустой массив []).
- Отрисовываем данные на странице (вызываем функцию рендеринга).

```
// Код в начале файла
const rawData = localStorage.getItem('myAppTasks');
const tasks = rawData ? JSON.parse(rawData) : [];

// Сразу отображаем то, что достали
renderTasks(tasks);
```

2. Фаза Сохранения (Действие пользователя)

Данные в хранилище нужно обновлять каждый раз, когда меняется состояние приложения (добавили задачу, удалили товар, нажали чекбокс).

Алгоритм:

- Пользователь совершает действие (событие click, submit).
- Мы обновляем наш массив/объект в памяти JS.
- Вызываем команду сохранения: **localStorage.setItem('key', JSON.stringify(data))**
- Обновляем интерфейс (вызываем render).

Пример: список покупок.

```
<h1>Список покупок</h1>

<form id="shop-form">
    <input type="text" id="item-input" placeholder="Что купить?" required>
    <button type="submit">Добавить</button>
</form>

<ul id="list-container"></ul>
<button id="clear-btn">Очистить список</button>
```

```
// 1. ИНИЦИАЛИЗАЦИЯ ДАННЫХ
// Пытаемся загрузить данные, если их нет — используем пустой массив
const rawData = localStorage.getItem('shopping_list');
let items = rawData ? JSON.parse(rawData) : [];

// Ссылки на элементы DOM
const form = document.querySelector('#shop-form');
const input = document.querySelector('#item-input');
const container = document.querySelector('#list-container');
const clearBtn = document.querySelector('#clear-btn');
```

```
// Функция сохранения данных в localStorage
function save() {
    // Превращаем массив объектов в JSON-строку
    localStorage.setItem('shopping_list', JSON.stringify(items));
}

// Функция отрисовки интерфейса
function render() {
    container.innerHTML = ''; // Очищаем список перед перерисовкой
    items.forEach((product) => {
        const li = document.createElement('li');
        li.className = 'item';
        // Добавляем текст в элемент списка
        li.innerHTML = `<span>${product.title}</span>`;
        container.appendChild(li);
    });
}
```

```
// Добавление нового товара
form.addEventListener('submit', (e) => {
    e.preventDefault(); // Отмена перезагрузки страницы
    const newItem = {
        title: input.value,
    };
    items.push(newItem); // Добавляем в массив
    save();              // Сохраняем в localStorage
    render();            // Обновляем экран
    input.value = '';    // Очищаем поле ввода
});
```

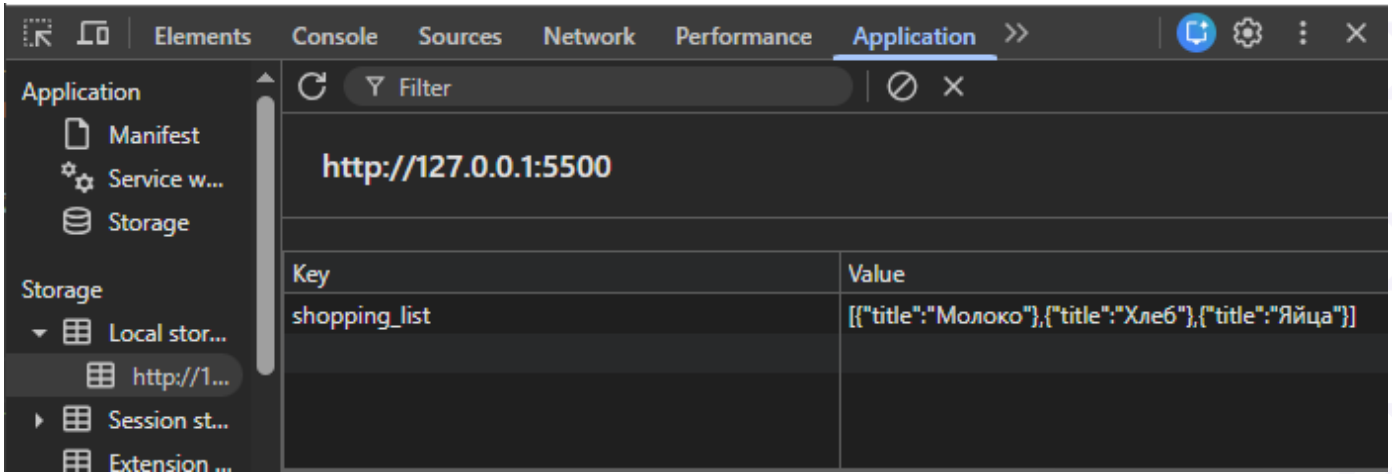
```
// Полная очистка
clearBtn.addEventListener('click', () => {
    if (confirm('Очистить весь список?')) {
        items = [];
        localStorage.removeItem('shopping_list'); // Или localStorage.clear()
        render();
    }
});

// 3. ЗАПУСК ПРИ СТАРТЕ
// Вызываем рендер сразу, чтобы показать сохраненные данные
render();
```

Список покупок

Что купить? Добавить

Молоко
Хлеб
Яйца

Очистить список



Вопрос: «А когда именно сохранять данные?».

- Плохо: Сохранять только при нажатии на кнопку «Сохранить» (пользователь может забыть её нажать).
- Хорошо: Автосохранение. Любое изменение в данных (добавление, удаление, редактирование) должно моментально дублироваться в localStorage. Это гарантирует сохранность данных даже при внезапном выключении компьютера.

Вывод:

- Алгоритм всегда цикличный: Прочитали при старте → Изменили при действии → Перезаписали в хранилище.
- Функция рендеринга должна быть отделена от логики данных. Она просто берет текущий массив (неважно, откуда он взялся — из хранилища или только что создан) и рисует его на экране.

Вывод по теме лекции:

- Обычные переменные очищаются при перезагрузке страницы. Хранилища позволяют сайту «помнить» пользователя и его данные (корзину, настройки, списки).
- localStorage — Хранит данные бессрочно. Работает по принципу «Ключ — Значение». Хранить можно только строки.
- Чтобы сохранить массив или объект, используем JSON.stringify().
- Чтобы вернуть данные в рабочее состояние (в объект/массив), используем JSON.parse().
- При загрузке страницы: Проверить localStorage -> Распаковать (parse) -> Отрисовать.
- При изменении: Обновить массив в коде -> Упаковать (stringify) -> Сохранить.

Контрольные вопросы:

- Что произойдет с данными в localStorage, если пользователь закроет браузер и выключит компьютер?
- Можно ли из JS-скрипта на сайте mysite.ru прочитать данные, которые сохранил сайт google.com? Почему?
- В чем главное отличие sessionStorage от localStorage?
- Какого типа данные возвращает метод localStorage.getItem()?
- Что вернет localStorage.getItem('some_key'), если такого ключа в хранилище не существует?
- Почему код localStorage.setItem('user', {name: 'Ivan'}) приведет к ошибке в логике приложения? Что мы увидим в хранилище вместо объекта?
- Какую задачу выполняет метод JSON.stringify()? В какой момент (до или после setItem) его нужно вызывать?
- Зачем нам нужен метод JSON.parse() при получении данных из хранилища?
- Почему не стоит хранить пароль пользователя в localStorage, даже если это очень удобно для автоматического входа?
- Если место в localStorage закончится (вы превысите 5-10 МБ), что произойдет при попытке записать новые данные?

Домашнее задание:

https://ru.hexlet.io/courses/js-basics

хекслет колледж

@HEXLY.KZ