Лабораторная работа №10
Отладка веб-приложений

1 Цель работы

1.1 Изучить процесс отладки веб-приложений средствами разработчика в браузере.
1.2 Изучить процесс средств тестирования API.

2 Литература

2.1 https://www.unisender.com/ru/blog/gid-po-devtools-chrome-i-drugih-brauzerov/ - Гид по инструментам разработчика
2.2 https://skillbox.ru/media/code/chto-mozhno-delat-v-chrome-devtools-5-poleznykh-funktsiy-dlya-nachinayushchikh/ - Chrome DevTools: основные инструменты и полезные функции
2.3 https://testengineer.ru/gajd-po-testirovaniyu-v-postman/ - Большой гайд по тестированию с Postman для начинающих

3 Подготовка к работе

3.1 Повторить теоретический материал (см.п.2).
3.2 Изучить описание лабораторной работы.

4 Основное оборудование

4.1 Персональный компьютер.

5 Задание

Исходный код для отладки и тестирования в файлах LabWork10.html и script.js.

5.1 Основы работы с DOM: исследование, изменения и событийные слушатели

Цель: Освоить работу с DOM-деревом, изменение стилей и текста, а также добавление событийных слушателей для отслеживания взаимодействия с элементами.

1. Исследование структуры DOM:
- Открыть DevTools на тестируемой странице.
- Перейти на вкладку Elements и изучить структуру HTML.
- Найти элементы, такие как заголовок, кнопка, изображение, форма.

2. Изменение элементов через DevTools и консоль:
- Применить панель Styles для изменения стиля заголовка и кнопки.
- В Console выполнить код для изменения текста заголовка:

```
document.querySelector('h1').textContent = 'Тестовый заголовок';
```

Для разрешения выполнения js в Console написать allow pasting перед выполнением js
- Найти изображение и измените его src на другой URL через Console.

3. Добавление и отслеживание событий:
- В Console добавить обработчик клика для кнопки, который будет выводить сообщение:

```
const button2 = document.querySelector('.button-class');
button2.addEventListener('click', () => {
    console.log("Кнопка была нажата (2)");
```

```
});
```

- Перейти на вкладку Event Listeners в Elements, Найти кнопку и проверить, что событие клика добавлено.

4. Отслеживание определенных типов событий

- В секции Event Listener Breakpoints развернуть список событий Mouse и установить брейкпоинт на событии click.
- Нажать на кнопку на странице и посмотреть, где в коде произошло событие click.
- Проанализировать стек вызовов, чтобы увидеть, какие функции выполняются при этом событии.

5. Поиск элементов на странице

- Выполнить следующий код в консоли и описать, что он делает:

```
const header = document.querySelector('#header');
console.log(header);
```

- Определить, есть ли элемент с указанным селектором на странице:

```
const element = document.querySelector('...'); // Замените на реальный селектор
if (element) {
    console.log('Элемент найден!');
} else {
    console.error('Элемент не найден');
}
```

6. Имитация ввода в поля форм и отправка формы

- Использовать следующий код для заполнения поля ввода:

```
const inputField = document.querySelector('#searchText');
inputField.value = 'test_data';
```

- Использовать следующий код для отправки формы:

```
const form = document.querySelector('form');
form.submit();
```

5.2 Отладка JavaScript: точки останова, условия и пошаговое выполнение кода

Цель: Научиться устанавливать и использовать точки останова, как обычные, так и условные, а также понимать, как пошаговая отладка помогает разбираться в логике кода.

1. Установка точки останова:

- Перейти на вкладку Sources и найти строку в коде, где выводится лог (или добавить его):

```
console.log("Точка останова на этом логе");
```

- Установить точку останова на этой строке, кликнув рядом с номером строки.

2. Пошаговая отладка:

- Выполнить действие, которое приводит к выполнению кода с точкой останова, и начните пошаговую отладку, используя Step Over (F10), Step Into (F11) и Step Out (Shift+F11) для анализа пошагового выполнения кода.

- Изучить текущие значения переменных во время выполнения кода, используя вкладку Scope.

3. Использование условной точки останова:

- Добавить счетчик нажатий для кнопки:

```
let clickCount = 0;
document.querySelector('.button-class').addEventListener('click', () => {
    clickCount++;
    console.log("Кнопка нажата", clickCount, "раз");
});
```

- Поставить точку останова с условием clickCount >= 3.
- Нажать кнопку и изучить, что точка останова срабатывает только на третьем клике.
- Добавить в JavaScript-файл цикл for, который выводит в консоль числа от 1 до 100.
- Установить условный брейкпоинт на строке с выводом, чтобы он срабатывал только когда число делится на 10.
- Запустить скрипт и проверить, что выполнение кода останавливается только на числах, кратных 10.
- Использовать консоль для анализа значения переменной на каждой итерации, когда срабатывает брейкпоинт.

4. Настройка вывода разных типов сообщений:

- Реализовать в консоли вывод сообщений, используя console.log для вывода информации, console.error для вывода ошибки и console.warn для предупреждения.
- Добавить функцию, вычисляющую результат деления двух чисел.
- Реализовать в консоли вывод сообщений, используя console.log для вывода информации о параметре, console.error для вывода ошибки.

5.3. **Работа с сетевыми запросами и производительностью**

Цель: Научиться мониторить сетевые запросы, анализировать их параметры, тестировать производительность страницы при низкой скорости соединения и оценивать задержки в производительности.

1. Мониторинг сетевых запросов:

- Перейти на вкладку Network и обновите страницу.
- Отфильтруйте запросы по типу (например, XHR), чтобы увидеть только AJAX-запросы.
- Найти запрос к API, изображению или другому ресурсу, открыть его и изучить Headers, Response, Timing для анализа.

2. Эмуляция медленного соединения:

- Изменить параметры сети на Offline / 4G / 3G и обновить страницу.
- Изучить, как это влияет на загрузку ресурсов, отметить, какие элементы загружаются дольше всего.

3. Измерение производительности:

- Перейти на вкладку Performance и начать запись.
- Выполнить несколько действий на странице (например, прокрутка, клики).

- Остановить запись и изучить, где возникают задержки, какие функции занимают больше всего времени.

4 Анализ утечек памяти:

- Перейти на вкладку Memory.
- Нажать Take snapshot, затем нажать на кнопку на странице, реализующу загрузку данных, несколько раз и создать еще один snapshot.
- Сравнить снимки, чтобы увидеть, как изменилась память и найти элементы, которые продолжают занимать память.
- Сделать выводы о возможных утечках памяти.

5.4. Тестирование адаптивности и отладка элементов, загружаемых динамически

Цель: Научиться проверять адаптивность страницы и разбираться в работе асинхронных функций и динамически загружаемых элементов на странице.

1. Эмуляция устройств и проверка адаптивности:

- Перейти в Device Toolbar (иконка телефона в DevTools или Ctrl+Shift+M).
- Выбрать устройства с разными разрешениями экрана (например, iPhone, iPad, Desktop) и убедитесь, что интерфейс страницы отображается корректно.
- Проверить, как изменяется навигация и положение элементов на разных экранах.
- Проверить адаптивность разных сайтов при разных разрешениях сайта и ориентации устройства.

2. Асинхронное обновление элементов:

- На странице должен быть элемент, который загружается или обновляется с задержкой (например, элемент dynamicElement, у которого настроить загрузку на 10 секунде). В Console выполнить код, чтобы проверять его появление через каждые 2 секунды:

```
const intervalId = setInterval(() => {
    const dynamicElement = document.querySelector('.dynamic-element');
    if (dynamicElement) {
        console.log("Динамический элемент загружен:", dynamicElement);
        clearInterval(intervalId); // Остановка проверки
    } else {
        console.log("Элемент еще не загружен");
    }
}, 2000);
```

3. Отладка асинхронных функций с использованием точек останова:

- Применить точку останова для асинхронного кода, например, добавить задержку для появления элемента через setTimeout, установить точку останова внутри этой функции, и проанализировать, когда элемент загружается.

5.5. Логирование, отладка и устранение ошибок

Цель: Освоить создание логов для отслеживания состояния приложения, работу с сообщениями об ошибках и методологию поиска и устранения ошибок в DevTools.

1. Логирование данных и состояния:

- Создать глобальную переменную для хранения состояния и данных, доступных для тестирования в Console:

```
window.testState = { clickCount: 0, lastMessage: "Никаких сообщений" };
```

2. Логирование действий и ошибок:

- добавить код для отслеживания кликов по кнопке, увеличения счетчика и записи сообщения:

```
const button = document.querySelector('.button-class');
button.addEventListener('click', () => {
    window.testState.clickCount++;
    window.testState.lastMessage = Кнопка нажата ${window.testState.clickCount} раз;
    console.log(window.testState.lastMessage);
});
```

- добавить в код ошибку (например, вызов несуществующей переменной) и проверить, как она отображается в Console:

```
console.log(undefinedVariable); // Ошибка
```

3. Устранение ошибок с помощью сообщений Console:

- Внимательно изучить сообщения в Console, Применить вкладку Sources и Call Stack для анализа причин ошибок и быстрого перехода к месту их возникновения в коде.
- Исправить код или добавить проверку существования переменных перед их использованием.

5.6 Отправка GET-запроса и анализ ответа

Цель: Изучить, как отправлять GET-запросы через Swagger UI и анализировать ответы.

1. Открыть Postman и Swagger UI (например, на https://petstore.swagger.io/ или https://dummyjson.com/docs/products).
2. В разделе "GET /pets" отправить запрос для получения списка всех домашних животных:
- Нажать на кнопку "Try it out".
- Нажать "Execute".
3. Проанализировать ответ:
- Проверить структуру ответа (например, статус 200, данные в JSON-формате).
- Обратить внимание на различные поля данных, такие как имя, идентификатор и статус животного.
4. Использовать Postman и Swagger для фильтрации и сортировки данных (если доступно).

5.7 Отправка POST-запроса с созданием нового ресурса

Цель: Изучить, как отправлять POST-запросы для создания нового ресурса.

1. Открыть Postman и Swagger UI (например, на https://petstore.swagger.io/ или https://dummyjson.com/docs/products).
2. В разделе "POST /pets" отправить запрос для создания нового животного, указав все необходимые поля в теле запроса (например, имя и статус):

- Нажать на кнопку "Try it out".
- Заполнить необходимые поля в теле запроса. Например:

```
{
  "name": "Барсик",
  "status": "available"
}
```

- Нажать "Execute".
3. Проанализировать ответ:
- Проверить, что запрос прошел успешно (статус 201 Created).
- Посмотреть, был ли создан новый ресурс и какие данные были возвращены.
Ожидаемый результат:
- Ответ должен содержать статус 201 и новый объект ресурса с данными, которые вы отправили.
- Тело ответа должно включать возвращенные данные о новом животном, такие как `id`, `name`, `status`.

5.8 Отправка PUT-запроса и анализ обновления ресурса
Цель: Изучить, как отправлять PUT-запросы для обновления существующего ресурса.

1. Открыть Postman и Swagger UI (например, на https://petstore.swagger.io/ или https://dummyjson.com/docs/products).
2. В разделе "PUT /pets/{petId}" отправить запрос для обновления данных существующего животного:
- Нажать на кнопку "Try it out".
- Ввести существующий `petId` (например, выбрать животное из результатов запроса GET).
- Заполнить необходимые поля для обновления. Например, изменить статус животного на "sold":

```
{
  "id": 1,
  "name": "Барсик",
  "status": "sold"
}
```

- Нажать "Execute".
3. Проанализировать ответ:
- Проверить, что запрос был выполнен успешно (статус 200).
- Изучить обновленные данные о животном в теле ответа.
Ожидаемый результат:
- Ответ должен содержать статус 200 OK и обновленные данные о животном.
- Тело ответа должно содержать измененные данные с новым значением статуса `sold`.

6 Порядок выполнения работы
6.1 Выполнить задание из п.5.
6.2 Ответить на контрольные вопросы.

7 Содержание отчета
7.1 Титульный лист
7.2 Цель работы

7.3 Скриншоты выполнения отладки с пояснениями
7.4 Ответы на контрольные вопросы
7.5 Вывод

8 Контрольные вопросы

8.1 Что такое «Postman»?
8.2 Что такое «Swagger»?
8.3 Как можно получить доступ к элементу, используя document.querySelector()?
8.4 Как получить информацию о производительности и памяти, используя средства разработчика в браузере?
8.5 Как выполняется логирование средствами JavaScript?
8.6 Как можно протестировать интерфейс веб-приложения, используя средства разработчика в браузере?