Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Кафедра информационных систем и программной инженерии

Лабораторная работа №2

по дисциплине "Графический и веб-дизайн"

ТЕМА РАБОТЫ:

Асинхронное программирование

Выполнил:

студент гр. ПРИ-120

Парахин К.В.

Принял:

Шамышев А.

Владимир 2023 г.

Цель работы:

Изучить принципы работы цикла событий в JavaScript; получить навыки написания асинхронного кода с помощью колбэков, промисов и асинхронных функций; освоить механизм вещания и обработки событий.

Задание на лабораторную работу:

1. Реализовать условную конструкцию с использованием колбэков и промисов;

2. Реализовать цикл с использованием колбэков и промисов;

3. На примере Fetch API реализовать запросы с обработкой с помощью .then() и с помощью async await

3.1. Получить список постов с сайта <https://jsonplaceholder.typicode.com/>

3.2. Взять случайный пост (сгенерировать его порядковый номер с помощью Math.random()) и получить информацию о пользователе, написавшего его.

3.3. Добавить обработчики ошибок

3.4. Ответы сервера вывести на экран

4. Создать событие и подписаться на него.

Выполнение работы:

1. Реализовать условную конструкцию с использованием колбэков и промисов;

function checkCondition(condition1, condition2, callback) {

if (condition1) {

callback(null, 'В первом условии получено true, ошибка не получена');

} else {

checkConditionWithPromise(condition2)

.then(function(result) {

callback(null, result);

})

.catch(function(err) {

callback(err, null);

});

}

}

function checkConditionWithPromise(condition) {

return new Promise(function(resolve, reject) {

if (condition) {

resolve('В условии 2 получено true, вызван resolve');

} else {

reject('В условии 2 получено false, вызван reject');

}

});

}

checkCondition(false, true, function(err, result) {

if (err) {

console.log(err);

} else {

console.log(result);

}

});

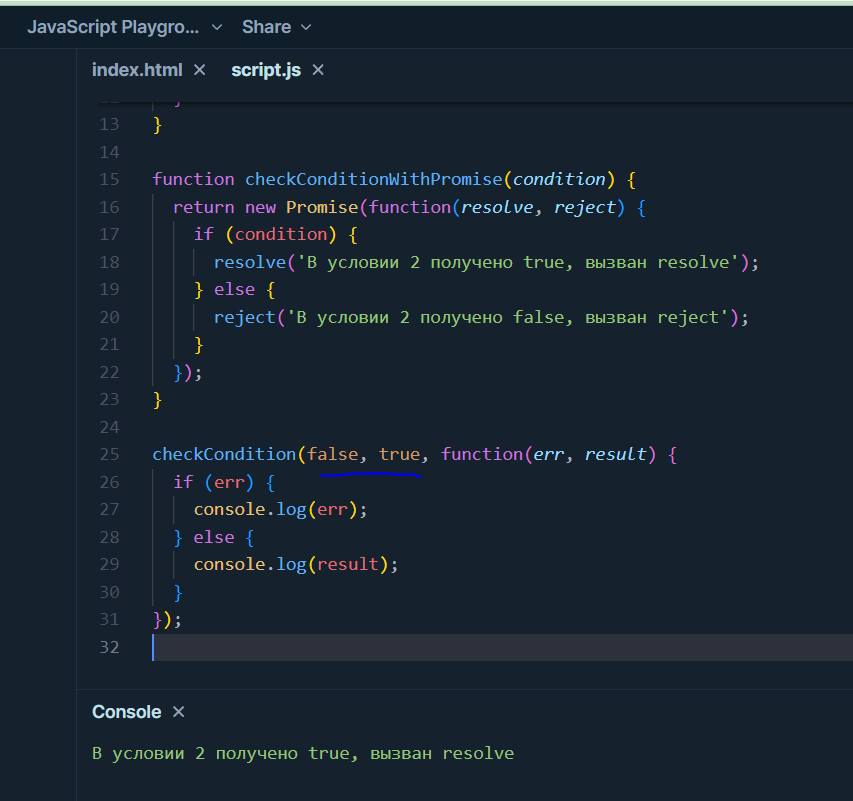


Рисунок 1. Тестирование скрипта с колбэками и прмисами в условии

2. Реализовать цикл с использованием колбэков и промисов

function loop(count, callback) {

return new Promise(function(resolve, reject) {

let results = [];

for (let i = 0; i < count; i++) {

let result = callback(i);

if (result instanceof Promise) {

console.log('Промис для задачи был получен');

result.then(function(res) {

results.push(res);

console.log('Добавление очередного элемента по промису');

if (results.length === count) {

resolve(results);

console.log('В полученном массиве ровно 5 элементов по получению промиса о завершении задачи');

}

}).catch(function(err) {

reject(err);

console.log('В результате выполнения задачи была получена ошибка')

});

} else {

results.push(result);

console.log('Добавление очередного элемента по колбэку');

if (results.length === count) {

resolve(results);

console.log('В полученном массиве ровно 5 элементов по получению колбэка о завершении задачи');

}

}

}

});

}

loop(5, function(i) {

return new Promise(function(resolve, reject) {

setTimeout(function() {

console.log(i);

resolve(i);

}, 1000);

});

})

.then(function(results) {

console.log(results);

})

.catch(function(err) {

console.log(err);

});

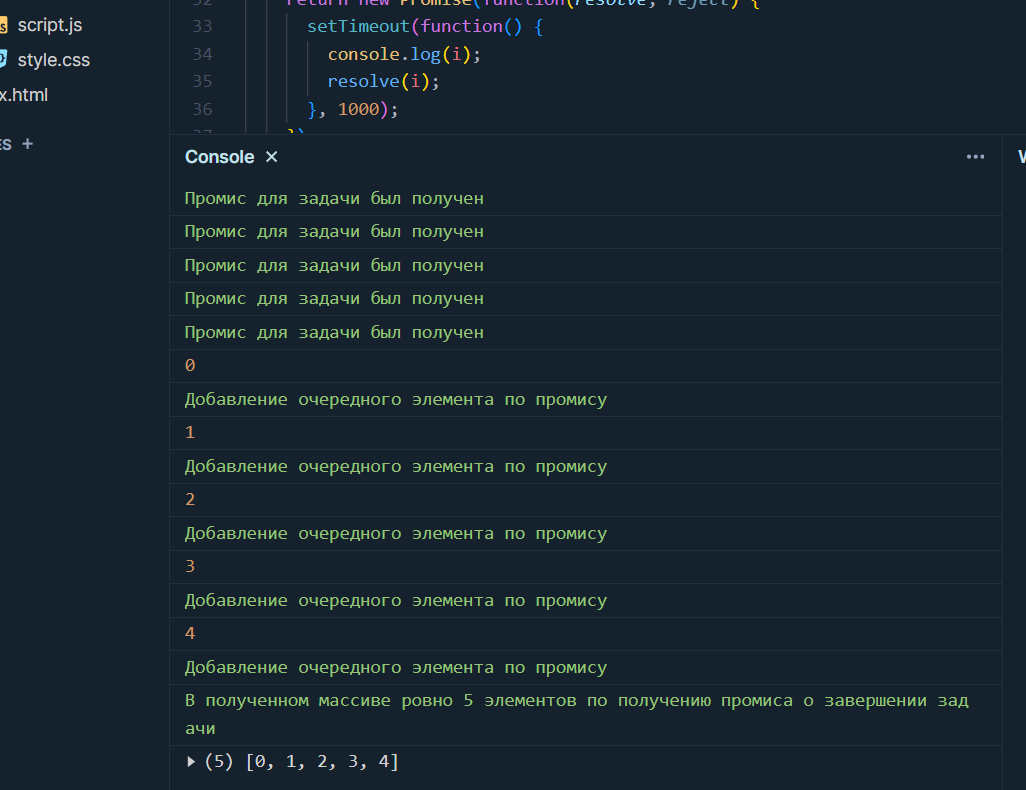


Рисунок 2. Тестирование скрипта с колбэком и промисами внутри цикла

3. На примере Fetch API реализовать запросы с обработкой с помощью .then() и с помощью async await

3.1. Получить список постов с сайта <https://jsonplaceholder.typicode.com/>

3.2. Взять случайный пост (сгенерировать его порядковый номер с помощью Math.random()) и получить информацию о пользователе, написавшего его.

3.3. Добавить обработчики ошибок

3.4. Ответы сервера вывести на экран

1. Получение всех постов через продолжения основной задачи ( с помощью конструкции .then) и в асинхронном режиме через конструкцию async await.

function getAllPostsWithWithThenConstruction(){

   console.log('Получение ответа в синхронном режиме через конструкцию then:');

   fetch(`https://jsonplaceholder.typicode.com/posts`)

     .then(function(*response*) {

    return response.json();

   })

    .then(function(*posts*) {

      let postsArray = Array.from(posts);

      console.log(`Было получено ${postsArray.length} постов с сайта`);

      posts.forEach(*post* => console.log(`Пост с номером ${post.id}: ${post.title}`));

      })

    .catch(function(*err*) {

        console.log(err);

      });

}

  async function getAllPostsAsync(){

    try{

      console.log('Получение ответа в асинхронном режиме через конструкцию async:');

      let response = await fetch(`https://jsonplaceholder.typicode.com/posts`);

      let posts = await response.json();

      let postsArray = Array.from(posts);

      console.log(`Было получено ${postsArray.length} постов с сайта`);

      posts.forEach(*post* => console.log(`Пост с номером ${post.id}: ${post.title}`));

    }

    catch(err){

      console.log(err);

    }

  }

getAllPostsWithWithThenConstruction();

*//getAllPostsAsync();*

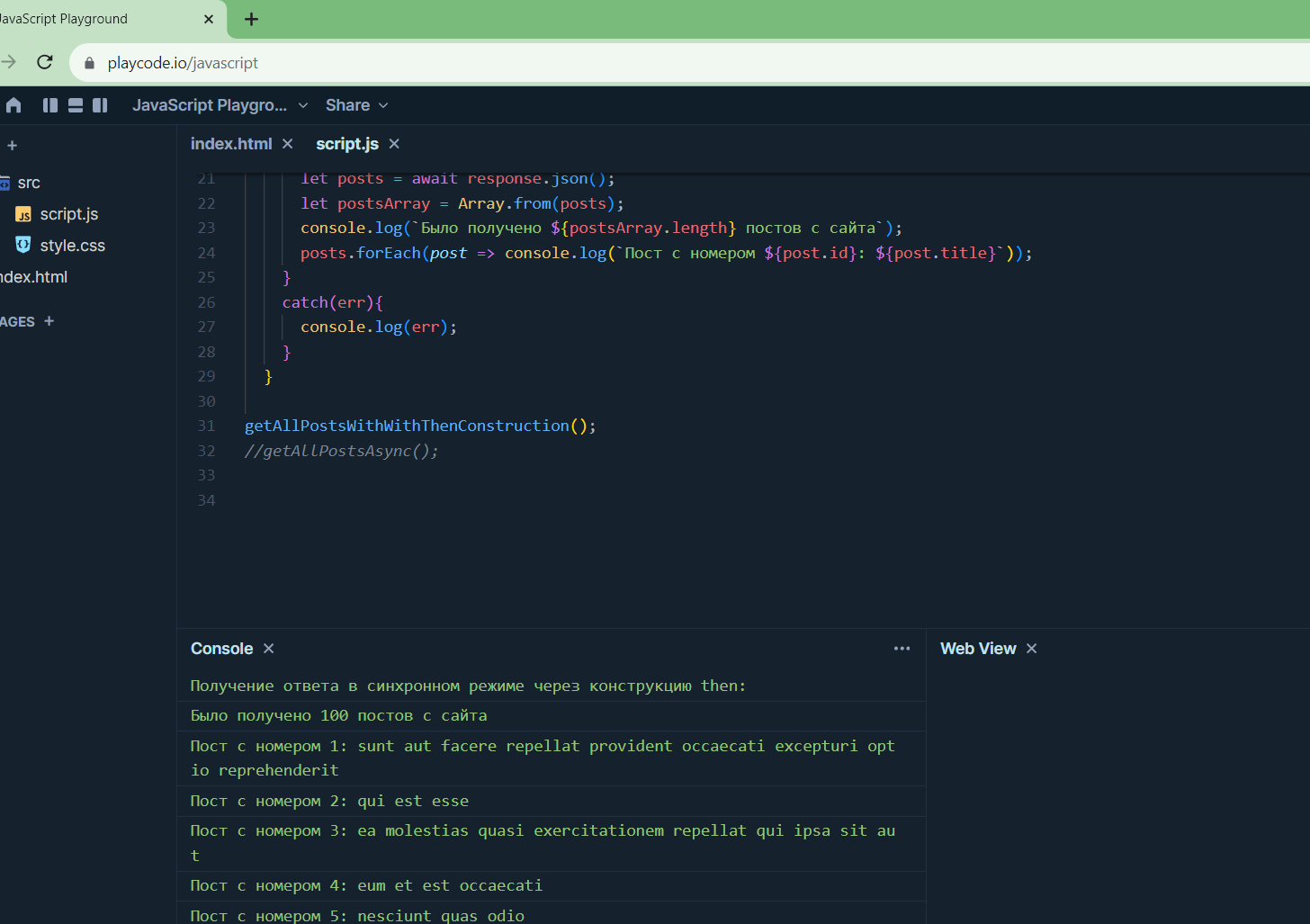


Рисунок 3.1.1. Получение всех постов с помощью продолжения задачи

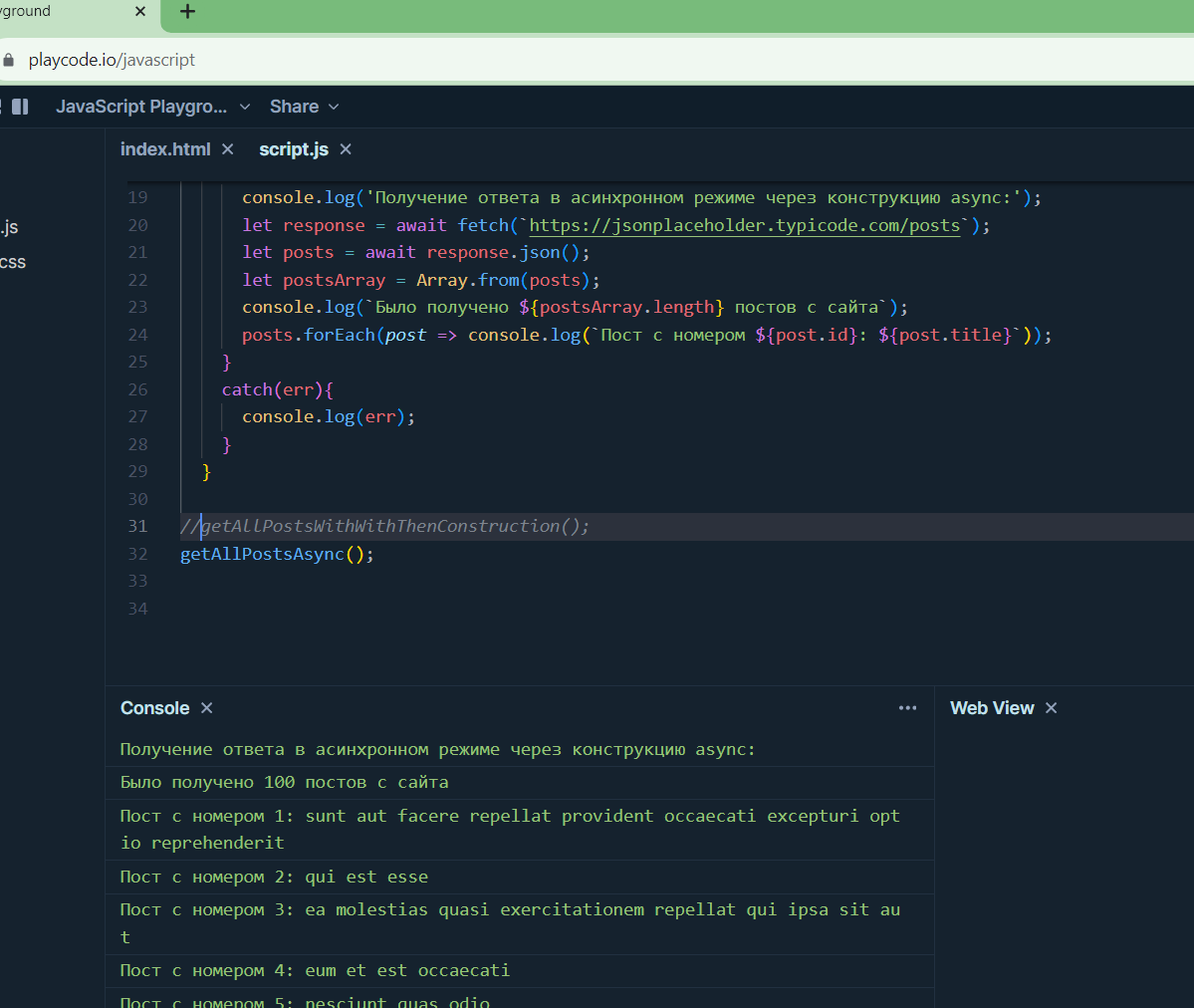


Рисунок 3.1.2. Получение всех постов с помощью конструкции async await

1. Получение информации о посте со сгенерированным номером и пользователе, отправившем его (с помощью конструкции .then и асинхронно через async await)

function getPostAndRelatedUserWithThenConstruction(){

let postId = Math.floor(Math.random() \* 100) + 1;

console.log(`Сгенерирован номер поста ${postId}`);

console.log('Получение ответа в синхронном режиме через конструкцию then:')

fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`)

  .then(function(*response*) {

    return response.json();

  })

  .then(function(*post*) {

    let userId = post.userId;

    fetch(`https://jsonplaceholder.typicode.com/users/${userId}`)

      .then(function(*response*) {

        return response.json();

      })

      .then(function(*user*) {

        console.log(`Пост: ${post.title}, выложенный пользователем ${user.name}`);

      })

      .catch(function(*err*) {

        console.log(err);

      });

  })

  .catch(function(*err*) {

    console.log(err);

  });

}

    async function getPostAndRelatedUserAsync() {

    try {

      let postId = Math.floor(Math.random() \* 100) + 1;

      console.log(`Сгенерирован номер поста ${postId}`);

      let response = await fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`);

      let post = await response.json();

      let userId = post.userId;

      response = await fetch(`https://jsonplaceholder.typicode.com/users/${userId}`);

      let user = await response.json();

      console.log(`Пост с номером ${postId}: ${post.title}, выложенный пользователем ${user.name}`);

    } catch (err) {

      console.log(err);

    }

  }

getPostAndRelatedUserWithThenConstruction();

*//getPostAndRelatedUserAsync();*

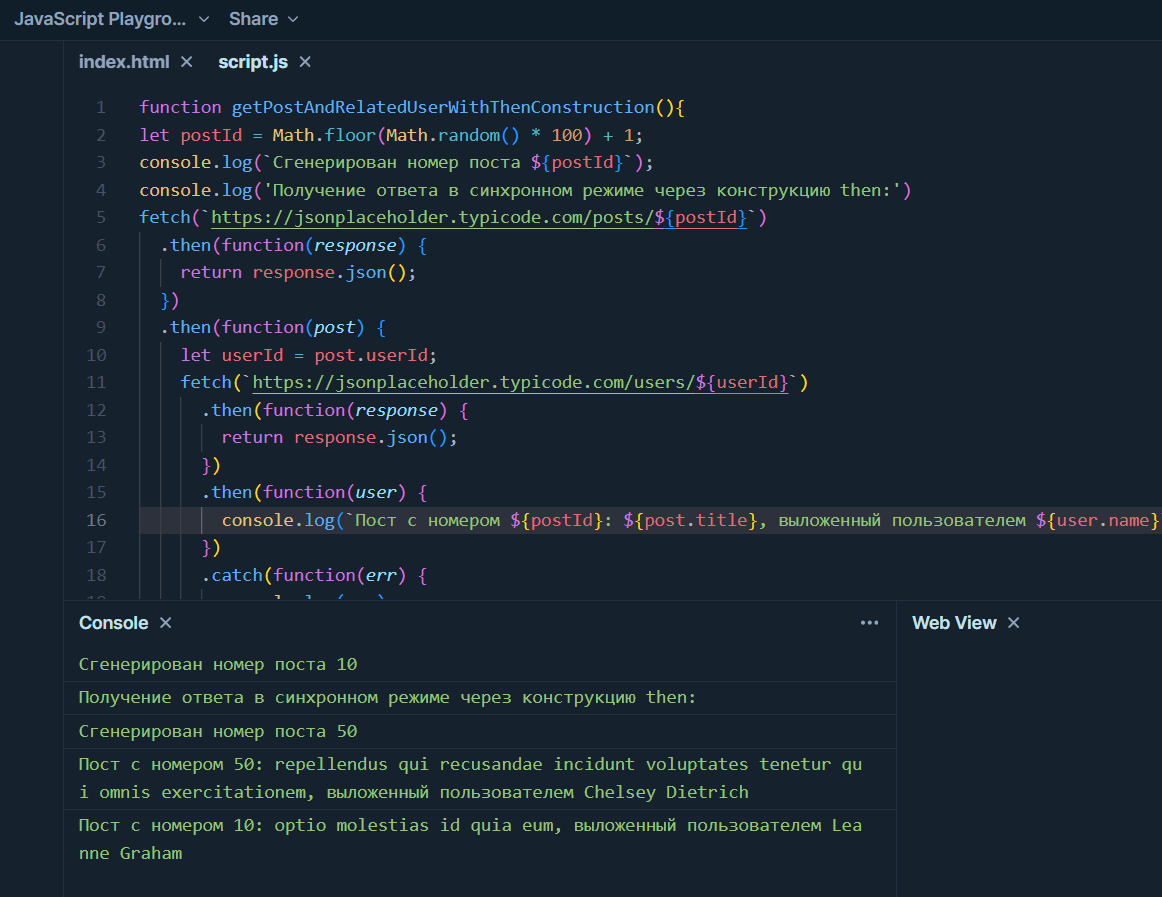


Рисунок 3.2. Тестирование получения поста со сгенерированным номером обоими способами

4. Создать событие и подписаться на него.

<!DOCTYPE html>

<html>

    <script src="number4.js">

    </script>

    <body>

        <h4>Нажмите на кнопку ниже: </h4>

        <br>

        <button id="button">

            Кнопка

        </button>

    </body>

</html>

window.addEventListener("load", function(){

    let button = document.querySelector('button');

    button.addEventListener('click', function() {

        alert("Кнопку нажали");

    });

});

// another way to dispatch this event

const event = new Event('click');

button.dispatchEvent(event);

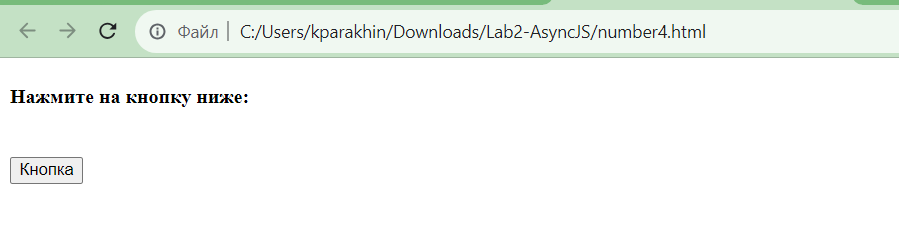


Рисунок 4.1 Запуск формы и ожидание события клика

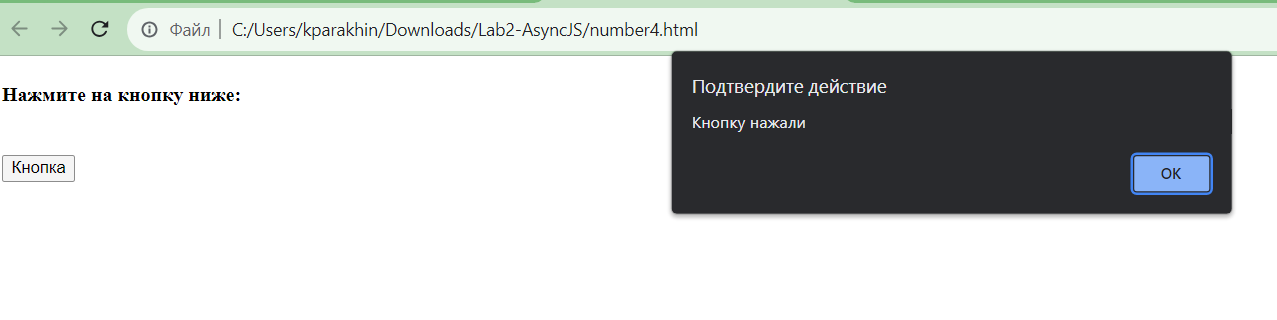


Рисунок 4.2. Действие, выполненное после отправки триггера для события (вызов обработчиков функции alert)

Вывод

В результате выполнения работы я изучил принципы работы цикла событий в JavaScript, получил навыки написания асинхронного кода с помощью колбэков, промисов и асинхронных функций, а также освоил механизм вещания и обработки событий.