# Лабораторная работа №8

Тема: Изучение технологии AJAX

**Задание**

На основе лабораторной работы №7 реализовать добавление постов/статей, лайков и комментариев без перезагрузки страницы.

**Требования**

* Использование методов библиотеки JQuery.

# Справочная информация

Ajax – это аббревиатура, означающая "Асинхронный JavaScript и XML" (Asynchronous JavaScript and XML). Это набор методов веб-разработки, которые позволяют веб-приложениям работать асинхронно — обрабатывать любые запросы к серверу в фоновом режиме. В результате любое веб-приложение, использующее AJAX, может отправлять и извлекать данные с сервера без необходимости перезагрузки всей страницы. Благодаря этому пользователь может продолжать работу со страницей.

Взаимодействия Ajax могут использоваться для выполнения таких задач, как проверка правильности ввода, отправка комментария, динамическое обновление данных на странице, передача элементов форм, автозаполнение и т.д.

## Преимущества

Экономия трафика

Использование AJAX позволяет сократить трафик при работе с веб-приложением благодаря тому, что вместо загрузки всей страницы достаточно загрузить только изменившуюся часть или вообще только получить/передать набор данных в формате JSON или XML, а затем изменить содержимое страницы с помощью JavaScript.

Уменьшение нагрузки на сервер

При правильной реализации AJAX позволяет снизить нагрузку на сервер в несколько раз.

В частности, все страницы сайта чаще всего генерируются по одному шаблону, включая неизменные элементы («шапка», «навигационная панель», «подвал» и т. д.), для генерации которых требуются обращения к разным файлам, время на обработку скриптов (а иногда и запросы к БД) — всё это можно опустить, если заменить полную загрузку страницы генерацией и передачей лишь содержательной части.

Ускорение реакции интерфейса

Поскольку загрузка изменившейся части значительно быстрее, то пользователь видит результат своих действий быстрее и без мерцания страницы (возникающего при полной перезагрузке).

Возможности для интерактивной обработки

Например, при вводе поискового запроса в Google выводится подсказка с возможными вариантами запроса. На многих сайтах при регистрации пользователь вводит имя, и сразу же видит, доступно это имя или нет. AJAX удобен для программирования чатов, административных панелей и других инструментов, которые выводят меняющиеся со временем данные.

Мультимедиа не останавливается

Страница не перезагружается, плеер продолжает работать. Потому AJAX ценен на аудио- и видеохостингах.

## JSON

Хотя Ajax — это стандарт для XML, в реальности многие приложения отправляют данные в формате JSON. В большинстве случаев это удобнее, чем использовать XML.

JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript.

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

1. Набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистрозависимая: имена с буквами в разных регистрах считаются разными[4]), значением — любая форма.
2. Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Структуры данных, используемые JSON, поддерживаются любым современным языком программирования, что и позволяет применять JSON для обмена данными между различными языками программирования и программными системами.

Пример

{

"firstName": "Иван",

"lastName": "Иванов",

"address": {

"streetAddress": "Московское ш., 101, кв.101",

"city": "Ленинград",

"postalCode": 101101

},

"phoneNumbers": [

"812 123-1234",

"916 123-4567"

]

}

## AJAX с использованием jQuery

jQuery имеет ряд функций, позволяющих обмениваться данными с сервером без перезагрузки страницы (технология ajax). Расммотрим основные из них.

**$.ajax()**

Осуществляет запрос к серверу без перезагрузки страницы. Это низкоуровневый метод, обладающий большим количеством настроек. Он лежит в основе работы всех остальных методов ajax.

Настройки beforeSend, error, dataFilter, success и complete позволяют установить обработчики событий, которые происходят в определенные моменты выполнения каждого ajax-запроса.

beforeSend происходит непосредственно перед отправкой запроса на сервер.

error происходит в случае неудачного выполнения запроса.

dataFilter происходит в момент прибытия данных с сервера. Позволяет обработать "сырые" данные, присланные сервером.

success происходит в случае удачного завершения запроса.

complete происходит в случае любого завершения запроса.

Функция $.ajax() узнает о типе присланных сервером данных от самого сервера (средствами MIME). Кроме этого, существует возможность лично указать (уточнить), как следует интерпретировать эти данные. Это делается с помощью параметра dataType. Возможные значения этого параметра:

"xml" — полученный xml-документ будет доступен в текстовом виде.

"html" — полученный html будет доступен в текстовом виде. Если он содержит скрипты в тегах <script>, то они будут автоматически выполнены, только когда html-текст будет помещен в DOM.

"script" — полученные данные будут исполнены как javascript.

"json" — полученные данные будут предварительно преобразованы в javascript-объект.

"text" — полученные данные окажутся доступными в виде обычного текста, без предварительной обработки.

Пример использования:

$.ajax({

url: '/data/people.json',

dataType: 'json',

success: function( resp ) {

$( '#target').html( resp.people[0].name );

},

error: function( req, status, err ) {

console.log( 'что-то пошло не так', status, err );

}

});

**$.get()** - производит запрос к серверу методом GET

**$.post()** - производит запрос к серверу методом POST

Если мы делаем простой запрос и при этом не заботимся об обработке ошибок — jQuery предлагает несколько более простых методов, которые позволяют нам использовать сокращённый синтаксис. Каждый из этих методов принимает адрес, необязательный объект данных и функцию для обработки успешного запроса.

$.get( '/data/people.html', function( html ){

$( '#target' ).html( html );

});

$.post( '/data/save', { name: 'Ребекка' }, function( resp ) {

console.log( resp );

});

Передача данных и работа с формами

Мы можем отправить данные с нашим запросом, установив свойство data в объекте конфигурации, либо передав объект в качестве второго аргумента в один из удобных методов.

Для запроса GET эти данные будут прикреплены к строке адреса; для запроса POST они будут отправлены в качестве данных формы. jQuery предлагает полезный метод .serialize() для получения данных формы и преобразования их в формат строки запроса (field1name=field1value&field2name=field2value...):

$( 'form' ).submit(function( event ) {

event.preventDefault();

var form = $( this );

$.ajax({

type: 'POST',

url: '/data/save',

data: form.serialize(),

dataType: 'json',

success: function( resp ) {

console.log( resp );

}

});

});

## Пример использования

Рассмотрим пример, который извлекает контент JSON из файла PHP на стороне сервера, используя AJAX.

Для начала создадим файл index.php, как показано в следующем фрагменте, который отображает базовую форму входа.

<!doctype html>

<html>

<head>

<script src="https://code.jquery.com/jquery-3.3.1.js" integrity="sha256-2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60=" crossorigin="anonymous"></script>

</head>

<body>

<form id="loginform" method="post">

<div>

Username:

<input type="text" name="username" id="username" />

Password:

<input type="password" name="password" id="password" />

<input type="submit" name="loginBtn" id="loginBtn" value="Login" />

</div>

</form>

<script type="text/javascript">

$(document).ready(function() {

$('#loginform').submit(function(e) {

e.preventDefault();

$.ajax({

type: "POST",

url: 'login.php',

data: $(this).serialize(),

success: function(response)

{

var jsonData = JSON.parse(response);

// user is logged in successfully in the back-end

// let's redirect

if (jsonData.success == "1")

{

location.href = 'my\_profile.php';

}

else

{

alert('Invalid Credentials!');

}

}

});

});

});

</script>

</body>

</html>

Файл index.php представляет собой стандартную HTML-форму, которая содержит поля имени пользователя и пароля. Он также содержит фрагмент JavaScript jQuery, который следует схеме, которую мы видели выше.

Мы использовали событие submit элемента формы, которое будет срабатывать, когда пользователь нажимает кнопку отправки. В этом обработчике событий мы инициировали вызов AJAX, который отправляет данные формы в файл login.php, используя метод POST асинхронно. Получив ответ от сервера, мы анализируем его, используя метод parse объекта JSON. И наконец, основываясь на успехе или неудаче, мы предпринимаем соответствующие действия.

Давайте также посмотрим, как выглядит login.php.

<?php

if (isset($\_POST['username']) && $\_POST['username'] && isset($\_POST['password']) && $\_POST['password']) {

// do user authentication as per your requirements

// ...

// ...

// based on successful authentication

echo json\_encode(array('success' => 1));

} else {

echo json\_encode(array('success' => 0));

}

Файл login.php содержит логику аутентификации пользователей и возвращает ответ JSON, основанный на успешном или неудачном входе в систему.