 [Translated from Russian to Ukrainian - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=docx&utm_campaign=attribution)

*Лабораторна робота №5*

Web – Технологія AJAX:  
використання JavaScript, DOM, XMLHttpRequest та XML

Мета роботи:знайомство з новим підходом до організації веб-застосунків – Asynchronous JavaScript + XML (AJAX).

# Матеріал, що вивчається: клієнтська частина програми - мова розмітки (X)HTML, стилі CSS, динамічне відображення та взаємодія з користувачем за допомогою Document Object Model (DOM), асинхронні запити за допомогою XMLHttpRequest, мова JavaScript.

1. Постановка задачі

Розробити та реалізувати на мові JavaScript (використовуючи об'єкт XMLHttpRequest і DOM) клієнтську частину веб-додатка, а також, використовуючи PHP, реалізувати серверний сценарій (скрипт) для обробки запитів користувача та подання результатів у вигляді генерованих фрагментів даних, що динамічно оновлюються в браузері користувача.

2. Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання для виконання роботи.
2. Вивчити теоретичний матеріал.
3. Визначити завдання, які має вирішувати сценарій, що розробляється.
4. Розробити текстовий інтерфейс користувача відповідно до завдання, а також кінцевий вид документа, що генерується, з областями для оновлення даних.
5. Реалізувати мовою PHP спроектований серверний сценарій.
6. Реалізувати на мові JavaScript клієнтську частину програми.
7. Протестувати локально розроблений сценарій.
8. Зробити висновки щодо роботи.

**3. Зміст звіту**

1. Постановка задачі. Опис завдань, які вирішуються серверним сценарієм.
2. Короткий опис алгоритму сценарію.
3. Опис використовуваних вхідних даних та функцій.
4. Лістинг вихідного коду сценарію з коментарями, а також результати його роботи (скриншот або текстове подання).
5. Висновки щодо роботи.

4. Основні теоретичні відомості

**4.1. Огляд технології AJAX.**

Донедавна веб-програми значно поступалися звичайному програмному забезпеченню практично за всіма параметрами: починаючи від зовнішнього вигляду і закінчуючи інтерактивністю. Але зараз відмінності між цими видами ПЗ стають все менш помітними.

Щоб зрозуміти, про що йдеться, достатньо подивитися на такі веб-програми як Google Suggest або, наприклад, Google Maps. Це інтерактивні функціональні програми, які повною мірою взаємодіють із користувачем, інформація на екрані оновлюється практично миттєво, не змушуючи чекати на перезавантаження сторінки. Це лише два приклади нового підходу до розробки веб-додатків, який називається «Ajax», що означає «асинхронний JavaScript плюс XML», або, якщо бути точнішим, «асинхронний JavaScript+CSS+DOM+XMLHttpRequest». Нова парадигма веб-програмування несе фундаментальні зміни для всього інтернету.

Парадигма Ajax передбачає використання кількох технологій, які при спільному і правильному застосуванні здатні забезпечити нову функціональність веб-додатків.

\* дизайн на основі XHTML та CSS, що відповідає рекомендаціям W3C;

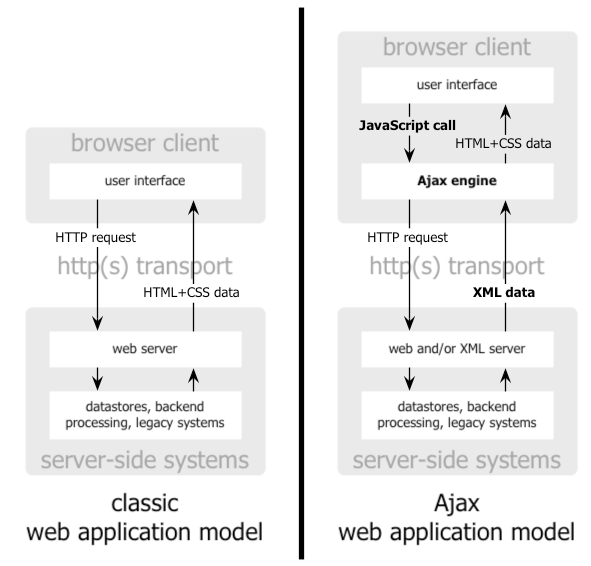
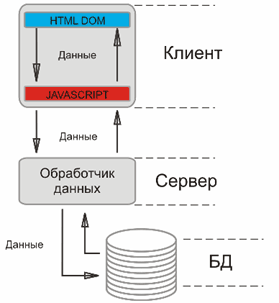
\* динамічне відображення інформації та інтерактивність на основі DOM;

\* обробка даних з використанням XML та XSLT;

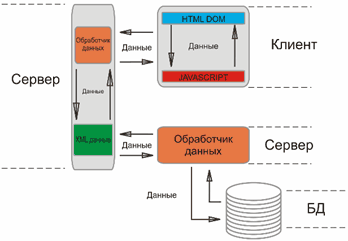
\* асинхронне отримання даних з допомогою XMLHttpRequest;

\* та JavaScript, який об'єднує все це разом.

Кожна з цих технологій давно відома, але всі разом вони дають кумулятивний ефект – це Ajax. Нова парадигма повністю виключає методи розробки сайтів, у яких застосовувалася класична модель: клієнт відправляє HTTP-запит, сервер його обробляє та повертає у відповідь HTML-сторінку. Ця модель, яка органічно вписується в гіпертекстове середовище, не дуже добре підходить для веб-додатків. Парадигма Ajax передбачає набагато грамотніший алгоритм обміну даними, як це наочно показано на схемі.

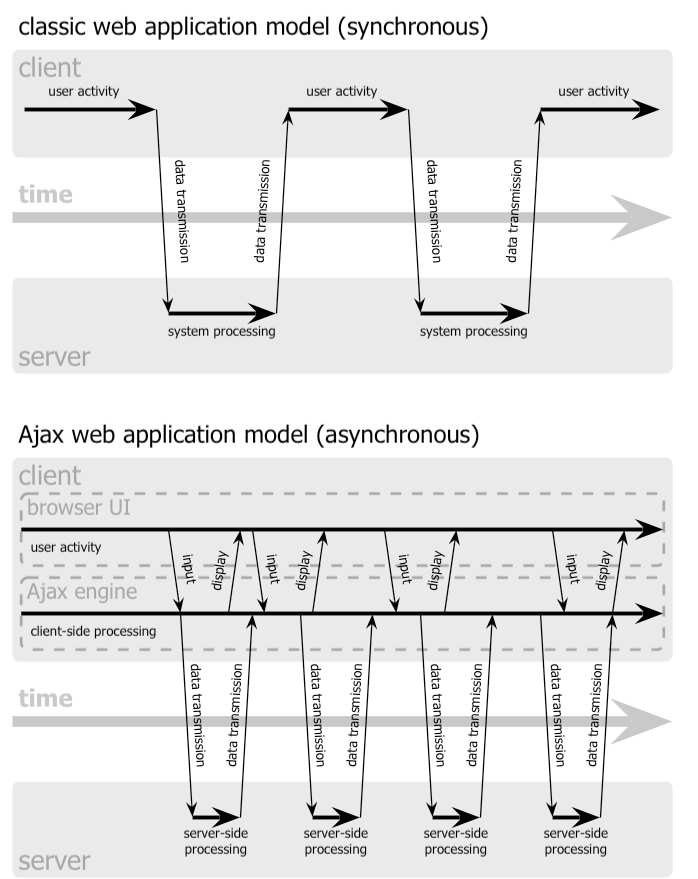


*Класична модель*



*AJAX*

Асинхронний обмін даними є більш прийнятним з точки зору юзабіліті (зручності використання), тому що користувачу не доведеться дивитися в порожній екран, чекаючи перезавантаження сторінки. Інтерфейс веб-застосунку, одного разу завантажившись, вже не зникне з дисплея. В ідеалі користувач взагалі не повинен помічати, коли програма звернулася до сервера, тому що дані підвантажуються у фоновому режимі дрібними порціями.

(Пояснення до схеми див. у конспекті лекцій).

Перерахуємо інтернет-компанії, які застосовують цей інноваційний підхід до веб-додатків. По-перше, це компанія Google, яка розробила відповідно до принципів Ajax програми Orkut, Gmail, останню бета-версію Google Groups, Google Suggest та Google Maps. Багато підходів Ajax реалізовано у сервісі для обміну фотографіями Flickr та пошуковику A9.

4.2. Асинхронні запити та об'єкт XMLHttpRequest

Одна з головних труднощів, з якими стикаються розробники інтерфейсів веб-застосунків, полягає в тому, що після того, як сторінка опинилася в браузері клієнта, зв'язок браузера із сервером закінчується. Будь-яка дія з елементом інтерфейсу вимагає повторного звернення до сервера із повторним завантаженням нової сторінки. Через це веб-додаток втрачає свою елегантність та повільно працює. Цю проблему можна вирішити за допомогою JavaScript та об'єкта XMLHttpRequest.

Завдяки JavaScript певну кількість програмної логіки можна перенести до HTML-сторінки, що дозволить швидко реагувати на дії користувача.

Об'єкт XMLHttpRequest вперше був реалізований компанією Microsoft як об'єкт ActiveX, але зараз він доступний як вбудований об'єкт у всіх браузерах Mozilla і Safari. Цей об'єкт дозволяє JavaScript здійснювати HTTP-запити до віддаленого сервера без необхідності перезавантажувати сторінку. По суті HTTP-запити відправляються та виходять повністю за "лаштунками" сторінки, а користувач їх навіть не помічає.

Перерахуємо методи та властивості об'єкта, які будуть використані далі:

\* XMLHTTPRequest.open("method", "URL", async, "uname", "pswd") - створює запит до сервера.

- method - тип запиту, наприклад, GET

- URL - URL запиту, наприклад http://localhost/file.xml

- async - якщо True, то використовуватиметься асинхронний запит, тобто виконання скрипту продовжиться після надсилання запиту. В іншому випадку скрипт чекатиме на відповідь від сервера, заморозивши інтерфейс браузера.

- uname, pswd - логін та пароль для простої веб-авторизації.

\* XMLHTTPRequest.send("content") - відправляє запит на сервер. Значенням content можуть бути дані для запиту POST або порожній рядок.

\* XMLHTTPRequest.onreadystatechange - обробник подій, що спрацьовує на кожну зміну стану об'єкта. Стан об'єкта може бути таким:

- 0 - до того, як запит відправлений (uninitialized)

- 1 - об'єкт ініціалізований (loading)

- 2 - отримано відповідь від сервера (loaded)

- 3 - з'єднання з сервером активно (interactive)

- 4 - об'єкт завершив роботу (complete)

\* XMLHTTPRequest.responseText - повертає отримані від сервера дані у вигляді рядка.

\* XMLHTTPRequest.responseXML - якщо відповідь сервера прийшла у вигляді правильного XML, повертає XML DOM об'єкт.

\* XMLHTTPRequest.status - повертає статус HTTP-відповіді у вигляді числа. Наприклад, 404, якщо запитувана сторінка не була знайдена на сервері.

Через свою суперечливу історію об'єкт XMLHttpRequest ще не є частиною будь-якого стандарту. Тому існує два відмінні один від одного методи виклику цього об'єкта в коді скрипта. В Internet Explorer об'єкт ActiveX викликається так:

var req = new ActiveXObject("Microsoft.XMLHTTP");

У Mozilla і Safari це робиться простіше (оскільки там це об'єкт, вбудований у JavaScript):

var req = новий XMLHttpRequest();

Зрозуміло, через такі відмінності вам необхідно створювати в коді гілки, кожна з яких виконуватиметься залежно від того, в якому браузері завантажено скрипт:

var req;

function loadXMLDoc(url) {

// branch for native XMLHttpRequest object

if (window.XMLHttpRequest) {

req = новий XMLHttpRequest();

req.onreadystatechange = processReqChange;

req.open("GET", url, true);

req.send(null);

// branch for IE/Windows ActiveX version

} else if (window.ActiveXObject) {

req = new ActiveXObject("Microsoft.XMLHTTP");

if (req) {

req.onreadystatechange = processReqChange;

req.open("GET", url, true);

req.send();

}

}

}

У цьому коді особливо важливо звернути увагу на властивість навитратідержав. Подивіться, як йому надається значення функції processReqChange. Ця властивість - хендлер події, яка запускається щоразу, коли змінюється стан об'єкта req. Стани позначаються номерами з 0 (об'єкт неініціалізований) по 4 (запит виконаний). Важливо це тому, що наш скрипт не чекатиме відповіді від сервера, щоб продовжити свою роботу. HTTP-запит буде сформовано та надіслано на сервер, але скрипт буде виконуватися далі. Через те, що ми вибрали такий варіант поведінки, нам не можна просто наприкінці функції повернути результат запиту, оскільки нам невідомо, чи ми його отримали до цього часу чи ні. Для цього ми і передбачили функцію processReqChange, яка відстежуватиме стан об'єкта req, і повідомить нам у потрібний час, що процес отримання документа закінчено, і ми можемо йти далі.

Для цього функції processReqChange потрібно перевіряти дві речі. Перша - чекати, коли стан об'єкта req зміниться на 4 (що означає, що процес отримання документа із сервера закінчено). Друге, це перевірити HTTP-статус відповіді. Ви знаєте, що код 404 означає "файл не знайдено" і 500 - "відбулася помилка на сервері". Але нам потрібний старий добрий код 200 ("все ОК"), який означає, що на сервері наш запит успішно виконано. Якщо ми отримали стан 4 і код 200, ми можемо продовжувати виконання нашого скрипту і обробляти результати, отримані від сервера. Зрозуміло, інакше ми повинні обробити всі помилки, наприклад, якщо код відповіді відрізняється від 200.

function processReqChange()

{

// only if req shows "complete"

if (req.readyState == 4) {

// only if "OK"

if (req.status == 200) {

// ...processing statements go here...

} else {

alert("There was a problem retrieving

the XML data:\n" + req.statusText);

}

}

}

4.3. Динамічні зміни веб-сторінки на стороні користувача за допомогою JavaScript та DOM

Як ви знаєте, DOM описує структуру веб-документа і дозволяє звернутися до будь-якої частини (вузла) і змінити його. Основна функція, яка використовується для доступу до елементів на сторінці – getElementById. Застосовується вона так:

var pTag1 = document.getElementById('myID');

У цьому випадку змінна pTag1 стає посиланням елемент з id='myID'. Зазвичай такими елементами є блоки, що позначаються div або span тегами, або поля введення форми, наприклад:

<div id=”myRegion”>Error</div>

Способи зміни відображення такого блоку будуть показані на прикладі.

4.4. Приклад (відповідь у форматі XML). Перевірка унікальності ніку (імені) при реєстрації нового користувача

У багатьох веб-програмах є процедура, коли новий користувач реєструється на сервері і потрібно вибрати "нік" для реєстрації. Дуже часто цей "нік" має бути унікальним, і тому після того, як користувач вибрав собі "нік", на сервері здійснюється перевірка бази даних користувачів, є вже такий "нік" чи ні. Якщо ви коли-небудь реєструвалися на якомусь поштовому веб-сервері, ви пам'ятаєте, як стомливо підшукувати "нік", яким ще ніхто не користується. Було б дуже добре, якби перевірка здійснювалася без потреби щоразу оновлювати сторінку.

Для вирішення ми скористаємося чотирма ключовими елементами: XHTML-формою, функцією JavaScript, спеціально написаною для цієї ситуації, двома нашими спільними функціями, про які ми говорили вище, і нарешті серверним скриптом, який буде звертатися до бази даних.

*Форма*

Це найлегша частина роботи - найпростіша форма з полем для введення "ніка". На подію onblur ми прив'язуємо наш скрипт перевірки. Для того, щоб вивести користувачеві повідомлення про результати перевірки, вставляємо його у форму та ховаємо за допомогою CSS. Це більш елегантний та ввічливий спосіб, ніж стандартне діалогове вікно функції alert().

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

onblur="checkName(this.value,'')" />

<span class="hidden" id="nameCheckFailed">

Це ім'я є в використанні, please try another.

</span>

У CSS оголошено властивості класу hidden, а також ще одного класу error, який служить для виведення повідомлень про помилку.

span.hidden {

display: none;

}

span.error{

display: inline;

color: black;

background-color: pink;

}

*Обробка введених даних*

Функція checkName використовується для перевірки даних, введених користувачем форми. Завдання функції - взяти дані, вирішити, якому серверному скрипту ці дані віддати, викликати іншу функцію, яка зробить всю роботу з HTTP-запитами та відповідями, а потім обробити відповідь. Ця наша функція складатиметься із двох частин. Одна частина бере дані з форми, а інша обробляє відповідь від сервера. Пояснення, навіщо це зроблено трохи пізніше.

function checkName(input, response)

{

if (response != ''){

// Response mode

message = document.getElementById('nameCheckFailed');

if (response == '1'){

message.className = 'error';

}else{

message.className = 'hidden';

}

}else{

// Input mode

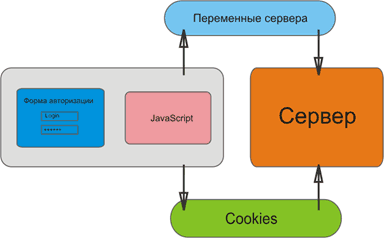
url = 'http://localhost/xml/checkUserName.php?q=' + input;

loadXMLDoc(url);

}

}

Відповідь обробляється просто – відповідь, яку ми отримаємо від серверного скрипту, буде або 1, або 0. 1 означає, що такий "нік" вже кимось використовується. Залежно від відповіді, наша функція змінює назву класу повідомлення про помилку - воно або показується, або ховається. Як відомо з коду, роботу на сервері виконує скрипт checkUserName.php.



*Схема роботи*

*Формування HTTP-запиту та відповіді*

Як ви бачили вище, робота з HTTP виконується двома функціями: loadXMLDoc та processReqChange. У першому практично нічого міняти не треба, а в другому потрібно щось змінити для роботи з DOM-ом.

Як ви пам'ятаєте, доки успішний результат виконання запиту не буде передано функції processReqChange, ми не можемо з цієї функції повернути будь-яке значення. Через це нам потрібно виконувати явний виклик функції з іншого місця коду, щоб скористатися результатом. Ось чому функція checkName розбита на дві частини. Таким чином, головне завдання функції processReqChange полягає в тому, щоб обробити XML-відповідь, отриману від сервера, і передати певне значення цієї відповіді функції checkName.

Ми не хочемо вносити до цих функцій будь-який специфічний код, оскільки ці функції можуть використовуватися й іншими елементами сторінки для звернення до сервера. Тому ми не вписували в функцію processReqChange ім'я функції checkName. Натомість ми вирішили, що сервер сам у своїй відповіді даватиме назву функції, яка до нього зверталася.

<?xml version="1.0" encoding="UTF-8"

standalone="yes"?>

<response>

<method>checkName</method>

<result>1</result>

</response>

Розбір цієї простої відповіді виконується без будь-яких проблем.

function processReqChange()

{

// only if req shows "complete"

if (req.readyState == 4) {

// only if "OK"

if (req.status == 200) {

// ...processing statements go here...

response = req.responseXML.documentElement;

method = response.getElementsByTagName('method')[0].firstChild.data;

result = response.getElementsByTagName('result')[0].firstChild.data;

eval(method + '(\'\', result)');

} else {

alert("There was a problem retrieving the XML data:\n" + req.statusText);

}

}

}

Звернувшись до властивості responseXML об'єкта XMLHttpRequest, ми отримуємо XML-відповідь, що надійшла від сервера, який потім розбираємо за допомогою DOM. Взявши з відповіді назву функції, яка вимагала цю відповідь, ми знаємо, яку функцію треба передати отримане значення. Після того, як ви закінчите тестування, заберіть умову else, щоб скрипт не турбував користувачів зайвим повідомленням про помилку.

*Серверний скрипт*

Останній елемент - це серверний скрипт, який отримуватиме запит, оброблятиме його і повертатиме відповідь у вигляді XML. У нашому прикладі скрипт звертатиметься до бази даних користувачів, щоб визначити, чи використовується вже даний "нік" чи ні. Для стислості в моєму прикладі скрипт не використовує базу даних, а просто перевіряє два імені "Viktor" та "Fairy".

<?php

header('Content-Type: text/xml');

function nameInUse($q)

{

if (isset($q)){

switch(strtolower($q))

{

case 'Viktor' :

return '1';

break;

case 'Fairy' :

return '1';

break;

default:

return '0';

}

}else{

return '0';

}

}

?>

<?php echo '<?xml version="1.0" encoding="UTF-8"

standalone="yes"?>'; ?>

<response>

<method>checkName</method>

<result><?php

echo nameInUse($\_GET['q']) ?>

</result>

</response>

Зрозуміло, ту саму процедуру перевірки треба передбачити і в серверному скрипті, який буде виконуватися, коли користувач натисне кнопку "Submit" на формі реєстрації. Це на той випадок, якщо у нього в браузері було відключено JavaScript, або користувач навмисно ввів нік, який вже використовується. Крім того, на завантажених сайтах може так статися, що під час вибору "ніка" він був вільний, а на момент подачі його вже хтось зайняв.

Як наступний крок розширте самі функціональність скрипта. Наприклад, сервер може повертати варіанти альтернативних "ніків" на основі "ніка", обраного користувачем, але вже зайнятого.

4.5. Приклад (відповідь у текстовому форматі). Завантаження фрагмента тексту із сервера та його відображення на сторінці

Спочатку Javascript:

function createRequestObject() {

var ro;

var browser = navigator.appName;

if(browser == 'Microsoft Internet Explorer'){

ro = new ActiveXObject('Microsoft.XMLHTTP');

}else{

ro = новий XMLHttpRequest();

}

return ro;

}

var http = createRequestObject(); //Створили об'єкт

//ф-ція надсилає запит

function sndReq(action) {

http.open('get', 'rpc.php?action='+action);

http.onreadystatechange = handleResponse;

http.send(null);

}

//ф-ція обробляє відповідь

function handleResponse() {

if(http.readyState == 4){

var response = http.responseText;

var update = новий Array();

if(response.indexOf('|' != -1)) {

//Розбиваємо рядок відповіді за символом '|'

update = response.split('|');

document.getElementById(update[0]).innerHTML = update[1];

}

}

}

Цей код створює об'єкт запиту, сам запит та функцію для його прийняття та розбору (за назвами, власне, неважко здогадатися). Для використання потрібно включити цей скрипт до вашої сторінки. (<script type="text/javascript" src="js.js"></script>)

Тепер для того, щоб надіслати запит, потрібно його прикріпити до якоїсь події. Наприклад onclick або помістивши прямо в href ось так:

<a href=”javascript:sndReq('foo')”>[foo]</a>

Це означає, що при натисканні на це посилання буде надіслано такий запит rpc.php?action=foo.

У rpc.php у вас може бути приблизно такий код:

switch($\_REQUEST['action']) {

case 'foo': /\*…action=foo…\*/

/\*тут наприклад запит до бази\*/

echo "foo|foo done";

break;

…

}

Тепер звернемо увагу на handleResponse. Воно розбирає рядок “foo|foo done” та поділяє його за символом '|'. Причому використовує те, що було до '|' - як id блоку в HTML, а те, що після - як новий innerHTML для цього блоку (тобто код HTML вставляється всередину блоку). В даному випадку, якщо у вас на сторінці є такий HTML:

<divid=”foo”></div>

То як тільки ви натисніть на посилання, він динамічно зміниться на: <div id=”foo”>foo done</div>

Ось, власне, і все. Решта лише надбудова над цим простим скриптом. Замінюйте мою просту відповідь від сервера “id|text” на більш багатий формат XML, та робіть запит складніше, ніж просто з однією змінною.

Узагальнити цей підхід, наприклад, для відсилання кількох змінних, було б дуже просто, приблизно так:

function sndReqArg(action,arg) {

http.open('get', 'rpc.php?action='+action+'&arg='+arg);

http.onreadystatechange = handleResponse;

http.send(null);

}

А функцію handleResponse можна легко розширити для цікавіших цілей, ніж просто заміщення вмісту DIV.

4.6. приклад. Поле SELECT з динамічним пошуком

У цьому прикладі буде показано ще один спосіб (універсальніший) отримання об'єкта XMLHTTPRequest у браузері.

Припустимо, у нас є таблиця, в якій близько мільйона записів. Користувачеві необхідно вибрати лише один запис з таблиці (реалізація відношення "один до багатьох"). Вибір користувача є лише одним із етапів заповнення великої веб-форми.

Звичайно, для того, щоб користувач міг вибрати потрібний запис з мільйона, потрібні якісь засоби пошуку цього запису. Наприклад, простий текстовий пошук за назвою.

У традиційному веб-додатку для цієї мети довелося б використовувати окрему сторінку і зберігати інші дані форми в сесії користувача або розбивати процес заповнення форми на кілька етапів. У програмі AJAX додаткова сторінка не потрібна.

Вибір запису буде реалізовано за допомогою двох елементів веб-форми. Перший елемент - це текстове поле, в якому користувач вводить ключове слово. Воно відсилається на сервер, а той повертає ті рядки з таблиці, які задовольняють умові пошуку. Відповідь сервера (у вигляді списку) розміщується в полі SELECT, в якому користувач і зробить остаточний вибір. Таким чином, при надсиланні всієї форми на сервер потрапить обране в поле SELECT значення як ID запису з великої таблиці.

У HTML виглядати це може так:

<input type="text"

onkeyup="lookup(this.value, 'id\_select',

'http://localhost/cgi-bin/xmlhttp.cgi')" />

<select id="id\_select" name="id\_select">

<option selected="selected" value=""></option>

</select>

На будь-яку подію KeyUp (віджимання кнопки) у текстовому полі викликається функція lookup ('текст', 'id-selecta', 'url')

function lookup(text, select\_id, url) {

// Отримуємо об'єкт XMLHTTPRequest

if(!this.http){

this.http = get\_http();

this.working = false;

}

// Запит

if (!this.working && this.http) {

var http = this.http;

// Якщо текстовому полі менше трьох

// Символів - не робимо нічого

if (text.length <3) return;

//додаємо закодований текст

//в URL запиту

url = url + "?text="+encodeURIComponent(text);

//створюємо запит

this.http.open("GET", url, true);

//Прикріплюємо до запиту функцію-обробник

//подій

this.http.onreadystatechange = function() {

// 4 - дані готові до обробки

if (http.readyState == 4) {

fill(select\_id, http.responseText);

this.working = false;

}else{

// дані у процесі отримання,

// можна порадувати користувача

//повідомленнями

// ЧЕКАЙТЕ ВІДПОВІДІ

}

}

this.working = true;

this.http.send(null);

}

if(!this.http){

alert('Помилка створення XMLHTTP об'єкта!')

}

}

Як видно, спочатку ми отримуємо XMLHTTP-об'єкт за допомогою функції get\_http(). Потім пошуковий текст кодується в стилі URL-адреси і формується GET-запит до сервера. URL запиту в даному випадку виглядатиме приблизно так: http://localhost/cgi-bin/xmlhttp.cgi?text=...

Скрипт на сервері, отримавши значення text, робить пошук у таблиці та надсилає результат клієнту. В обробнику подій об'єкта XMLHTTP, коли дані від сервера отримані та готові до використання, викликається функція fill('select\_id', 'data'), яка заповнить список SELECT отриманими даними.

Функція get\_http() - це крос-браузерна реалізація отримання об'єкта XMLHTTP (у кожному браузері він виходить по-своєму). Її реалізацію з коментарями ви можете легко знайти в інтернеті, це, так би мовити, приклад підручника.

function get\_http(){

var xmlhttp;

/\*@cc\_on

@if (@\_jscript\_version >= 5)

try {

xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");

} catch (e) {

try {

xmlhttp = new

ActiveXObject("Microsoft.XMLHTTP");

} catch (E) {

xmlhttp = false;

}

}

@else

xmlhttp = false;

@end @\*/

if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {

try {

xmlhttp = new XMLHttpRequest();

} catch (e) {

xmlhttp = false;

}

}

return xmlhttp;

}

Функція fill() отримує на вхід значення параметра ID списку SELECT, який необхідно заповнити, і дані, отримані з сервера.

Для простоти припустимо, що дані з сервера ми отримуємо у вигляді таблиці, поля якої розділені символом табуляції 't', а рядки - символом перенесення рядка 'n' (зручніше використовувати формат JSON – JavaScript Object Notation):

id1 t name1 n

id2 t name2 n

...

На основі вмісту цієї таблиці ми заповнюватимемо поле SELECT елементами OPTION.

function fill (select\_id, data) {

// поле SELECT змінну як об'єкта

var select = document.getElementById(select\_id);

// Очищаємо SELECT

select.options.length = 0;

// якщо даних немає - не робимо більше нічого

if(data.length == 0) return;

// у масиві arr - рядки отриманої таблиці

var arr = data.split('n');

// для кожного рядка

for(var i in arr) {

// у масиві val - поля отриманої таблиці

val = arr[i].split('t');

// додаємо новий об'єкт OPTION до нашого SELECT

select.options[select.options.length]=

new Option(val[1], val[0], false, false);

}

}

Готово. Тепер для будь-якої веб-форми програми ми можемо реалізувати подібний вибір значення з багатомільйонного списку, який буде виглядати як лічені натискання клавіш. У локальній мережі вибір відбувається майже миттєво. У разі нестабільного або низькошвидкісного з'єднання з сервером, необхідно також сповіщати користувача про те, що завантаження даних із сервера ще не завершено. Корисно передбачити засоби для реакції на обрив з'єднання.

**5. Контрольні питання**

1. Яка основна сутність технології AJAX?
2. Які переваги технології в порівнянні з класичним перезавантаженням сторінок?
3. Що таке асинхронний обмін даними?
4. Які технології використовують у концепції AJAX?
5. Що таке DOM?
6. Як використовується JavaScript для динамічних змін на сторінці?
7. Який об'єкт використовується для асинхронних запитів?
8. Як отримати доступ до елемента сторінки у браузері?
9. **Варіанти завдань**

У кожному варіанті вихідна форма документа повинна містити заголовок з назвою роботи, внизу документа – ПІБ виконавця (можна брати з попередньої роботи).

Сервер запускається/зупиняється файлами run.exe/stop.exe з папки /webservers/etc/ , яка знаходиться у робочій директорії.

Файли ЛР повинні бути розміщені (при запущеному сервері) у папці

*z:/home/localhost/www/прізвище\_виконавця/*

та будуть відповідно доступні за адресою

*http://localhost/прізвище\_виконавця/ім'я\_скрипту.php*

**Усі варіанти завдань ґрунтуються на завданнях попередньої лабораторної роботи.**

**Всі оновлення на сторінці проводити ДИНАМІЧНО за допомогою AJAX (без перезавантаження сторінки).**

Варіанти:

1. "Онлайн газета".

Редагувати режим перегляду. Вивести лише заголовки статей у вигляді посилань і порожню область для перегляду. При натисканні на завантажити в область перегляду текст статті.

1. "Міні фотогалерея".

Вивести список завантажених файлів в ідеї посилань. При натисканні на посилання завантажувати картинку та відображати її в області перегляду.

1. "Гостьова книга"

Здійснити перевірку адреси електронної пошти на коректність динамічної за допомогою AJAX (додати поруч із полем для введення посилання або кнопку, яка запускає перевірку). У разі неправильної адреси виводитиме попередження.

1. "Прайс лист"

Вивести список товарів як посилань. При натисканні на посилання вивести в області перегляду опис обраного товару.

1. «Список студентів»

Вивести список студентів у вигляді посилань. При натисканні на посилання виводити середню групу та бал студента в області перегляду.

1. «Посторінковий висновок записів»

Виводити кожну наступну порцію даних без перезавантаження сторінки (генеровану частину сторінки призначати як  
innerHtml для блоку DIV, який буде областю перегляду).

**6. Список літератури, що рекомендується**

1. Розробка Web-додатків на РНР та MySQL: Пер. з англ. / Лаура Томсон, Люк Веллінг. - 2-ге вид., Випр. - СПб: ТОВ «ДіаСофт», 2003. - 672 с.
2. Аргеріх Л. та ін. Професійне РНР програмування, 2-ге видання. - Пров. з англ. – СПб: Символ-Плюс, 2003. – 1048 с., іл.
3. Кузнєцов І.В., Симдянов І.В., Голишев С.В. PHP 5. Практика розробки веб-сайтів. - СПб.: БХВ-Петербург, 2005. - 960 с.
4. Кузнєцов І.В., Симдянов І.В., Голишев С.В. PHP 5 на прикладах. - СПб.: БХВ-Петербург, 2005. - 576 с.
5. Коггзолл, Джон. PHP5. Повне керівництво: Пер. з англ. - М.: Видавничий дім "Вільямс", 2006. - 752 с.
6. Колісніченко Д.М. Самовчитель PHP 5. СПб.: Наука та техніка, 2004. - 576 с.
7. http://www.php.net
8. http://www.mysql.org
9. www.ajaxian.com
10. http://ua.wikipedia.org/wiki/AJAX