МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

В. В. Огурцов
Д. В. Гриньов

ВЕБ-ПРОГРАМУВАННЯ НА БОЦІ СЕРВЕРА ЗА ДОПОМОГОЮ МОВИ PHP

Лабораторний практикум
з навчальної дисципліни
"Веб-технології та веб-дизайн"
для студентів напряму підготовки
6.050101 "Комп'ютерні науки"

Харків
ХНЕУ ім. С. Кузнеця
2016

УДК 004.7(07)
О-39

Авторський колектив: канд. екон. наук В. В. Огурцов – лабораторні роботи 1 – 4; канд. техн. наук Д. В. Гриньов – лабораторні роботи 5, 6.

Рецензенти: професор кафедри електронних обчислювальних машин Харківського національного університету радіоелектроніки, д-р техн. наук *С. Г. Удовенко*; професор кафедри обчислювальної техніки та програмування НТУ "ХПІ", д-р техн. наук *Г. А. Кучук.*

Рекомендовано до видання рішенням ученої ради Харківського національного економічного університету імені Семена Кузнеця.
Протокол № 3 від 31.10.2016 р.

Самостійне електронне текстове мережеве видання

Огурцов В. В.

О-39 Веб-програмування на боці сервера за допомогою мови PHP : лабораторний практикум з навчальної дисципліни "Веб-технології та веб-дизайн" для студентів напряму підготовки 6.050101 "Комп'ютерні науки" [Електронний ресурс] / В. В. Огурцов, Д. В. Гриньов. – Харків : ХНЕУ ім. С. Кузнеця, 2016. – 132 с.
ISBN 978-966-676-653-6

Призначено для практичного вивчення основ веб-технологій, веб-програмування на боці сервера за допомогою мови PHP. Розглянуто основи побудови веб-сайта на боці сервера з використанням PHP і SQL. Досліджено принципи клієнт-серверних технологій. Запропоновано практичні основи створення якісних веб-сайтів.

Рекомендовано для студентів та аспірантів, які спеціалізуються в галузі інформаційних технологій у різних сферах діяльності.

УДК 004.7(07)

ISBN 978-966-676-653-6

Вступ

Нова інформаційна технологія досягла такого розвитку, що, напевне, не залишилося сфер людського життя, які не задіяні в глобальній мережі Інтернет. Наразі інтерес до мережі Інтернет продовжує зростати. Розроблений у роки інформаційного вибуху, Інтернет стає невід'ємною частиною життя більшості людей усього світу.

Якісний сайт стає важливим, а у деяких галузях – єдиним засобом досягнення економічних, політичних, соціальних, рекламних та інших цілей. Якісний сайт відрізняється від інших сайтів глобальної мережі такими рисами: висока якість інформаційного наповнення та грамотність його подання; оригінальність і естетична привабливість зовнішнього вигляду сторінок; доступність змісту сайта для максимально широкого кола користувачів поза залежністю від застосовуваних ними типів пристроїв і версій браузерів, а також від обмежень за станом здоров'я; ергономічність елементів користувальницького інтерфейсу сайта, що забезпечує високу ефективність, але водночас легкість і невимушеність взаємодії відвідувача з веб-ресурсом; надійність і безпека використовуваних технологічних рішень, чітка погодженість роботи всіх компонентів; бездоганне пророблення всіх деталей і нюансів.

Звісно, для створення якісного веб-сайта, тобто відповідності до цих рис, потрібна плідна робота висококваліфікованих спеціалістів із різних веб-технологій, які повинні розуміти не лише вузьку галузь знань, але і добре уявляти весь спектр веб-технологій.

У практикумі лабораторні завдання підібрані таким чином, який дозволяє поступово вивчати важливі питання дисципліни з веб-дизайну та програмування на боці сервера. Поступове навантаження понятійного апарату дасть змогу глибше зрозуміти сутність питань, пов'язаних з механізмами побудови веб-сайтів. Лабораторний практикум містить опис шести лабораторних робіт. Кожна з них розкриває відповідну тему із навчальної дисципліни, освітлюючи практичні питання з веб-дизайну та програмування на боці сервера. Склад кожної лабораторної роботи містить такі пункти:

мета лабораторної роботи;
рекомендації щодо підготовки до виконання роботи;
основні тези;
завдання до лабораторної роботи за варіантами;
хід виконання лабораторної роботи (у разі потреби);
контрольні запитання.

У ході проведення лабораторної роботи студент повинен продемонструвати:

а) знання понятійного апарату й основних принципів побудови веб-сайта;

б) розуміння принципів функціонування та використання процесів клієнт-серверних технологій;

в) грамотне використання програмного забезпечення для розроблення веб-сайтів.

Вивчення тем і виконання завдань містить такі етапи.

1. Підготовчий етап (до проведення практичного або лабораторного заняття):

а) отримання відповідного матеріалу до даних методичних рекомендацій;

б) завдання, номера варіанта та вимог від викладача;

в) вивчення теоретичного матеріалу за відповідною темою;

г) розроблення алгоритму виконання завдання.

2. Безпосереднє виконання завдання в аудиторії, в комп'ютерному класі обчислювального центру або самостійно (наприклад, вдома):

а) проходження допуску до лабораторної роботи;

б) установлення (за необхідності) конфігурування програмного забезпечення;

в) відпрацьовування завдання за варіантом;

г) аналіз отриманих результатів.

3. Складання звіту та захист роботи. Звіт про лабораторну роботу повинен містити:

а) титульний аркуш із найменуванням лабораторної роботи (ЛР) і даними виконавця;

б) дату виконання;

в) мету роботи;

г) опис завдання;

д) тексти розроблених HTML-документів і програм;

е) результати роботи й їх аналіз;

ж) висновки про роботу.

Усі матеріали звіту необхідно зброшурувати.

У процесі викладання навчальної дисципліни "Веб-технології та веб-дизайн" основна увага приділяється оволодінню студентами професійними компетентностями згідно з Національною рамкою кваліфікацій України.

Лабораторна робота 1
Моніторинг та аналіз http-трафіка

1.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення специфікації протоколу HTTP і моделювання та моніторинг http-трафіка.

1.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити основи роботи між клієнтом і сервером. Ознайомитись зі специфікацією протоколу передачі даних між клієнтом і сервером HTTP.

1.3. Загальні тези лабораторної роботи

Загальні відомості щодо протоколу HTTP

Протокол передачі Гіпертексту (HTTP) є протоколом прикладного рівня сімейства протоколів TCP/IP [1; 3; 6; 14]. Опис протоколу HTTP можна знайти в RFC-1945, RFC-2068, RFC-2069, RFC-2616, RFC-2617, RFC-7540. Протокол HTTP – найпоширеніший прикладний протокол стека TCP/IP. Коли ви відвідуєте різні WWW-сайти за допомогою браузера, він (браузер) взаємодіє з веб-серверами, використовуючи саме протокол HTTP. Щоразу під час переходу за посиланням від одного ресурсу до іншого браузер звертається до HTTP для доступу до сервера, що зберігає необхідну інформацію. HTTP забезпечує високопродуктивний механізм тиражування інформації мультимедійних систем незалежно від типу подання даних. Протокол побудований за об'єктно-орієнтованої технології та може використовуватися для вирішення різних завдань, наприклад, роботи з серверами імен або управління розподіленими інформаційними системами.

На даний момент існує кілька версій протоколу HTTP. Першою версією протоколу HTTP була версія 0.9, в якій використовувався простий запит і підтримувався єдиний метод GET. Таким чином, за допомогою цього протоколу можна було тільки отримати документ або файл з сервера. Найбільш поширеною є версія 1.1, яка була стандартизована в 1997 р.

(RFC-2068) і оновлена в 1999 р. (RFC-2616) [15]. У цій версії використовується повний запит, і набір методів значно розширений. Зокрема, можна не тільки отримувати файли з сервера, але і передавати файли на сервер, видаляти їх з сервера (природно, за наявності певних прав), передавати різними способами інформацію спеціальними програмами, які працюють на сервері, а також керувати параметрами з'єднання, кешування, виду, типу та кодування ресурсу тощо. У 2015 році була стандартизована версія 2 (RFC-7540) [16], яка сьогодні (на травень 2016 р.) підтримується більшістю популярних веб-браузерів і веб-серверів. Але частка сайтів, що працюють на HTTP/2 поки (на 01.05.2016 р.) становить 7.5 % – за даними w3techs.com [30]. Відмінності HTTP/2 від HTTP/1.1 [16; 17] не стосуються HTTP-методів, кодів стану та семантики, а це є основними частинами протоколу для розробників веб-додатків. Основні зміни спрямовані на оптимізацію продуктивності, використання всіх можливостей протоколу TCP, поверх якого працює HTTP.

Сервіси WWW

Протокол HTTP дозволяє отримувати доступ до ресурсів і сервісів WWW-серверів. Для уніфікації доступу до багатофункціональних ресурсів мережі WWW-сервера підтримують комплекс інтерфейсів, що дозволяють структурувати рівні та методи роботи з різними ресурсами мережі.

URI (Uniform Resource Identifier – Ідентифікатор ресурсу), URL (Uniform Resource Locator – Місцезнаходження ресурсу), URN (Uniform Resource Name – Ім'я ресурсу) – різні імена того самого сервісу, який призначений для ідентифікації типів, методів роботи та комп'ютерів, на яких знаходяться певні ресурси, доступні через Internet. Цей сервіс складається з трьох частин:

схема – ідентифікує тип сервісу, через який можна отримати доступ до сервісу (наприклад, FTP або WWW-сервер);

адреса – ідентифікує адресу (хост) ресурсу, (наприклад, www.hneu.edu.ua);

ім'я або шлях доступу – ідентифікує повний шлях до ресурсу на обраному хості, який ми хочемо використовувати для доступу до ресурсу (наприклад, /home/images/imagel.gif).

Існує два типи адрес: абсолютний URI і відносний URI. *Абсолютний URI* містить повну адресу об'єкта та протокол, тобто всі три наведені частини. Наприклад, файл hneu.jpg, розташований в каталозі images на сервері www.ukr.net, матиме такий абсолютний URI: http://www.ukr.net/images/hneu.jpg.

Це означає, що буде використовуватися тип доступу через HTTP, схема доступу відокремлена двокрапкою ":" і вказує на використання протоколу HTTP. Наступні два слеша відокремлюють наступну адресу сервера www.hneu.edu.ua, далі йде шлях до файла та сам файл. Абсолютний URI завжди повинен починатися зі схеми доступу. Якщо шлях до файла й ім'я файла відсутні, то передбачається, що звернення відбувається до кореневого каталогу сервера "/". Слід зазначити, що схема доступу та DNS адреси сервера є регістрозалежному незалежними, однак шлях та ім'я файла в загальному випадку до регістру залежні.

Відносний URI використовує URI поточного документа, запитаного HTTP-клієнтом. Застосовуючи ту ж схему, HTTP клієнт реконструює URI, змінюючи тільки деякі імена та розширення файлів. Указівка перед відносним URI ".." означає перехід на рівень рейтингів вгору, а вказівка слеша – ігнорує всі каталоги і додає вказаний далі шлях безпосередньо за адресою сервера.

Крім того, URI ресурсу може містити не тільки ім'я ресурсу, а й параметри, необхідні для його роботи. Ім'я ресурсу відокремлене від рядка параметрів символом "?". Рядок параметрів складається з лексем, що розділяються символом "&". Кожна така лексема складається з імені параметра і його значення, розділених символом "=". Символи, що не входять у набір символів ASCII, замінюються знаком "%" і шістнадцятковим значенням цього символу. Наприклад, символ "?" замінюється на "% 3F". Оскільки символ пробілу зустрічається в рядку параметрів досить часто, то він замінюється не на "% 20", а на символ "+" (якщо ж в рядку параметрів зустрічається символ "+", то він замінюється на "% 2B"). Для зазначеного ресурсу весь рядок параметрів є одним строковим параметром, тому тип, черговість або унікальність імен окремих параметрів рядка не істотні. Наприклад: ***http://www.is.hneu.edu.ua/?q=node/51***.

Принципи побудови HTTP-з'єднання

Протокол HTTP побудований за моделлю "запит/відповідь". Іншими словами, клієнт встановлює з'єднання з сервером і відправляє запит. У ньому зазначені тип запиту, URL, версія протоколу HTTP і зміст запиту: інформація клієнта (параметри) і, можливо, супроводжувальна інформація або тіло повідомлення. Сервер HTTP після обробки запиту повертає відповідь, що містить: версію підтримуваного протоколу, код обробки запиту або код помилки й інформацію, що повертається за запитом. Інформація тіла

повідомлень як клієнта, так і сервера повинна бути представлена в MIME-форматі.

HTTP-з'єднання ініціюється клієнтом і складається з запиту до ресурсу певного сервера. У найпростішому випадку, з'єднання є потоком даних між клієнтом-ініціатором з'єднання та сервером. Однак досить часто в поєднанні може брати участь проміжний агент або proxy-сервер.

Proxy-сервер або просто proxy – це проміжний агент, який приймає запит клієнта і залежно від своїх налаштувань змінює частину або всі повідомлення запиту та передає переформатований запит запитуваній сервера. У момент прийняття запитів proxy може працювати як сервер, а під час передачі запитів – як клієнт. Крім того, proxy може підтримувати внутрішній кеш запитів і відповідей. Кеш зберігає відповіді серверів і повертає їх на запит клієнта, не передаючи запит безпосередньо запрошенням сервера. Тим самим зменшується час з'єднання і збільшується продуктивність роботи з віддаленими та повільними серверами. Однак далеко не всі відповіді можуть кешуватися. Деякі запити можуть містити параметри, які накладають обмеження на роботу кешу.

HTTP – це протокол прикладного рівня, який працює поверх транспортного протоколу TCP [1; 3; 6; 14 – 16; 30]. Однак, як будь-який протокол прикладного рівня, може працювати на будь-якому іншому транспортному протоколі, орієнтованому на установку з'єднання, та забезпечує гарантовану доставку. За замовчуванням HTTP-протокол використовує TCP порт 80. На відміну від інших прикладних протоколів, що забезпечують нерозривне з'єднання до тих пір, поки не відбудеться помилка чи не буде поданий сигнал до завершення з'єднання, HTTP працює по-іншому. HTTP-з'єднання повинне відкриватися клієнтом перед кожним запитом і закриватися сервером після відправки відповіді.

Ні браузер (клієнт), ні сервер не зберігають інформацію навіть про останнє з'єднання. Такий стиль роботи дозволяє серверу швидше переходити до обслуговування інших клієнтів, що збільшує ефективність його роботи. Однак з отриманням гіпертекстових документів, які містять вбудовану графіку або інші асоційовані об'єкти, за короткий проміжок часу браузер відправляє кілька запитів до одного й того ж сервера. У цьому випадку знижується ефективність роботи та збільшується завантаження мережі за рахунок великої кількості службових TCP-пакетів, призначених для відкриття та закриття з'єднання. Тому вже в специфікації HTTP/1.0 був передбачений режим нерозривного або постійного з'єднання. У цьому випадку сервер не розриває з'єднання після відправки відповіді, і клієнт може

відправити наступний запит. У специфікації HTTP/1.1 постійні з'єднання використовуються за замовчуванням.

За вибір режиму й управління параметрами з'єднання відповідає поле заголовка Connection. Крім цього в HTTP/1.1 для постійних з'єднань може бути використана конвеєрна обробка запитів. Клієнт може відправити кілька запитів, не чекаючи відповіді на кожен, а потім отримати кілька відповідей від сервера. Слід також зазначити, що навіть з використанням постійних з'єднань, сервер все ж розриває з'єднання з клієнтом, якщо після закінчення деякого часу очікування не отримує від нього ніяких даних. Тайм-аут зазвичай становить близько кількох десятків секунд. Тому, незважаючи на можливість використання постійних з'єднань, протокол HTTP відрізняється від інших прикладних протоколів стека TCP/IP, в яких з'єднання зазвичай закривається з боку клієнта.

Формат HTTP-транзакції

У HTTP-транзакції мають один загальний формат. Кожен запит клієнта та відповідь сервера складається з трьох частин: першого рядка запиту (відповіді), заголовка та тіла.

Клієнт ініціює транзакцію таким чином:

- установлює TCP-з'єднання з сервером на номер телефону порту (за замовчуванням - 80);
- посилає перший рядок запиту, вказавши HTTP-команду, яка називається методом, адресу документа і номер версії HTTP;
- посилає заголовок запиту, щоб повідомити серверу інформацію про свою конфігурації та дані про формати документів, які він може приймати. Уся інформація заголовка вказується за рядком, обов'язково в кожному рядку наводиться ім'я та значення. Завершується заголовок символом нового рядка;
- посилає тіло запиту, яке для деяких запитів може бути відсутнім. Дані, що знаходяться в тілі, використовуються головним чином тими веб-додатками, які застосовують за методом POST.

Сервер відповідає на запит клієнта таким чином:

- посилає перший рядок відповіді, що містить три поля: версію HTTP, код стану й опис;
- посилає клієнту заголовок відповіді, що містить дані про самий сервер і викликані документи. Як і в запиті, заголовок завершується символом нового рядка;

- посилає клієнту тіло відповіді. У разі успішного запиту тіло містить викликані дані. Це може бути копія файла або результат виконання веб-додатка. Якщо запит клієнта задовольнити не можна, тіло може містити додаткові дані у вигляді зрозумілого для користувача роз'яснення причин, за яких сервер не зміг виконати даний запит;
- розриває з'єднання з клієнтом.

Структура та параметри запиту клієнта

Перший рядок запиту подають у вигляді:

METHOD <SP> Request-URI <SP>HTTP-Version.

Тут і далі <SP> позначає символ пропуску, /* line */ – позначення порожнього рядка. Кожен рядок закінчується парою <CRLF> – повернення каретки, переведення рядка.

Поле "METHOD" містить ідентифікатор методу звернення до HTTP-сервера. Метод визначає склад і структуру заголовків запиту, вид передачі та структуру параметрів запиту. Найбільш поширеними методами є "GET" і "POST". Іноді також використовують методи "HEAD", "DELETE", "PUT". У специфікації HTTP/1.1 додані деякі нові методи, наприклад, "OPTIONS" і "TRACE". Усі ідентифікатори методів є регістрозалежними.

"Request-URI" містить абсолютний або відносний URI-ресурс. У разі відносного URI схема доступу вважається http, а в якості адреси сервера береться адреса сервера, з якою встановлене з'єднання.

"HTTP-Version" указує на версію HTTP, підтримувану клієнтом. За замовчуванням використовується версія 0.9. Для ідентифікації версії протоколу HTTP використовує схему "major.minor". Тобто версія HTTP/1.0 старша за версію HTTP/0.9. Наприклад, перший рядок запиту клієнта може виглядати так:

GET /index.html HTTP/1.1.

Далі йде заголовок, який складається з рядків виду FieldName: FieldValue, де FieldName – назва поля, FieldValue – значення поля. Кожен рядок завершується символами <CRLF>; порядок проходження полів не важливий. Імена полів є регістрозалежному незалежними, значення полів можуть бути регістрозалежному залежними. Специфікація HTTP/1.0 вимагає, щоб символ ":" слідував відразу за ім'ям поля, а між символом ":" і значенням поля був тільки один символ <SP>. Специфікація HTTP/1.1

висуває менш жорсткі вимоги. Умовно можна розділити всі поля запиту на наступні розділи: General-Header – основні поля, Request-Header – поля запиту, Response-Header – поля відповіді, Entity-Header – поля об'єкта, Extension-Header – додаткові поля. Найбільш поширені поля для кожного розділу треба подати таким чином:

[General-Header]
Date
Pragma
Cache-Control
Connection
Trailer
Transfer-Encoding
Upgrade
Via
[Request-Header] Authorization
From
If-Modified-Since
Referer
User-Agent Accept
Accept-Charset
Accept-Encoding
Accept-Language
Expect
Host
If-Match
If-None-Match
If-Range
If-Unmodified-Since Max-Forwards
Proxy-Authorization
Range
TE
[Response-Header] Location
Server
WWW-Authenticate
Age
Proxy-Authenticate Retry-After
Vary

ETag
Warning
[Entity-Header] Allow
Content-Encoding
Content-Length
Content-Type
Expires
Last-Modified
Content-Language
Content-Location
Content-MD5
Content-Range
[Extension-Header]

Поля розділів "General-Header" і "Entity-Header" можуть використовуватися як в запитах клієнта, так і у відповідях сервера. Поля розділу "RequestHeader" використовуються тільки в запитах клієнта. Поля розділу "ResponseHeader" використовуються тільки у відповідях сервера. У розділі "ExtensionHeader" містяться деякі додаткові поля.

Заголовок запиту може виглядати, наприклад, таким чином:

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94
Safari/537.36
*Accept: image/gif, text/html, */**
From: webmaster@hneu.edu.ua
If-Modified-Since: Tue, 29 Mar 2016 11:20:46 GMT
Connection: Keep-Alive

Заголовок відділяється від тіла повідомлення порожнім рядком. Тіло повідомлення може міститися в запиті клієнта за використання методів "PUT" і "POST". Указувати поле заголовка "ContentLength" – обов'язково.

Структура та параметри відповіді сервера

Перший рядок відповіді подають у вигляді:

HTTP-Version <SP> Ret-Code <SP>Comment-String

Поле версії містить номер версії HTTP, яку використовує даний сервер для передачі відповіді. "Ret-Code" – це трирозрядне число, що позначає результат обробки сервером запиту клієнта. Формат коду повернення

аналогічний кодам повернення інших прикладних протоколів стека TCP/IP – таких, як FTP і SMTP. Перша цифра коду статусу HTTP визначає клас коду відповіді, а кожен з п'яти класів містить групу значень статусу коду.

Існує п'ять можливих значень для першої цифри (від 1 до 5):

1xx: інформаційні коди – запит отриманий, триває обробка;

2xx: успішні коди – дані були успішно прийняті, оброблені та використані;

3xx: коди перенаправлення – для виконання запиту потрібні додаткові дії

4xx: коди помилок клієнта – запит містить синтаксичні помилки або запитані ресурси недоступні;

5xx: коди помилок сервера – сервер не зміг обробити або виконати запит.

"Comment-String" – опис, який є простим, зрозумілим для користувача текстом, що пояснює код відповіді.

Найбільш поширені коди відповіді:

100 – продовжувати, Continue;

200 – OK;

201 – створений, Created;

202 – прийнято, Accepted;

204 – нема вмісту No Content;

206 – частковий вміст, Partial Content;

300 – множинний вибір, Multiple Choices;

301 – постійно перенесений, Moved Permanently;

302 – тимчасово переміщений, Moved Temporarily;

304 – немодифікована, Not Modified;

305 – необхідно використовувати проксі-сервер, Use Proxy;

400 – некоректний запит, Bad Request;

401 – несанкціоновано, Unauthorized;

403 – заборонено, Forbidden;

404 – не знайдено, Not Found;

405 – метод не дозволений, Method Not Allowed;

407 – потрібна аутентифікація через proxy-сервер, Proxy Authentication Required;

408 – минув час очікування запиту, Request Timeout;

413 – об'єкт запиту занадто великий, Request Entity Too Large;

414 – URI запиту занадто довгий, Request-URI Too Long;

500 – внутрішня помилка сервера, Internal Server Error;

501 – не реалізоване, Not Implemented;
505 – не підтримує версію HTTP, HTTP Version Not Supported;
Наприклад, перший рядок відповіді сервера може виглядати так:
HTTP / 1.1 200 OK
або
HTTP / 1.1 404 Not Found

Далі заголовок, аналогічний за форматом заголовку запиту. Тема відділяється від тексту повідомлення порожнім рядком. Розмір тіла повідомлення вказується в полі заголовка "Content-Length". Допускається також у відповіді не вказувати поле Content-Length. У цьому випадку клієнт може визначити розмір тіла щодо закриття з'єднання сервером. Крім того, в специфікації HTTP/1.1 можна за допомогою поля Transfer-Encoding вказати тип передачі "chunked". У цьому випадку розмір тіла повідомлення вказується в самому тілі, і поле Content-Length також може бути відсутнім. Тема відповіді може виглядати, наприклад, таким чином:

Date: Mon, 09 May 2016 14:04:40 GMT
Server: Apache/2.2.22 (FreeBSD) mod_ssl/2.2.22 OpenSSL/0.9.8q
DAV/2 PHP/5.3.10 with Suhosin-Patch mod_jk/1.2.32
Last-modified: Mon, 09 May 2016 14:04:38 GMT
Content-type: text/html
Content-length: 2482

Методи запиту

Метод "GET" запитує інформацію про ресурс, розташований за заданим URI. Як правило, запитувана інформація є HTML, TXT або інший текстовий або графічний файл. Тіло запиту з використанням методу GET завжди порожнє. Тому, якщо необхідно передати додаткові дані, вони приєднуються до URI в першому рядку запиту. Якщо в тезі <form> документа задане значення атрибута method = "GET", то пари "ключ = значення", тобто введені дані з форми, приєднуються до URI після знаку питання (?). Пари відокремлюються одна від одної амперсантом (&). Символи, що не входять у набір символів ASCII, замінюються знаком "%" і шістнадцятковим значенням цього символу. Крім того, в методі GET може передаватися інформація про додаткові шляхи. Додатковий шлях вказується після URL (після імені файла та до знака запитання), тобто /path/filename.ext?param=value.

Запит за допомогою методу GET може виглядати таким чином:

GET /index.html HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/50.0.2661.94 Safari/537.36
Host: www.hneu.edu.ua
Accept: text/html, application/xhtml+xml, application/xml;q=0.9,
*image/webp, */*;q=0.8*

Метод "POST", як правило, використовується для передачі клієнтом на сервер даних, які обробляє ресурс, зазначений у URI. Даний метод найчастіше використовується не зі статичними ресурсами сайта, а для роботи з веб-додатком (PHP, ASP.NET, JSP, Ruby, Python, Node.js, CGI- або Fast CGI-сервісом і т. п.). Метод "POST" передає параметри ресурсу URI в тексті листа, тому в його використанні не потрібно дотримуватися обмежень на довжину переданого рядка параметрів. Якщо в тезі <form> документа задане значення атрибута method = "POST", то пари "ключ = значення", тобто введені дані з форми, поміщаються в тіло запиту. Розмір тіла запиту визначається полем заголовка "Content-Length", це поле є обов'язковим у використанні методу POST. Відповіді на запити методом POST, як правило, не кешуються. Приклад запиту за допомогою методу POST:

POST http://www.ikt.hneu.edu.ua/course/modedit.php HTTP/1.1
Host: www.ikt.hneu.edu.ua
Connection: Keep-Alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
image/webp,/*;q=0.8*
Accept-Encoding: gzip, deflate
Accept-Language: uk,en;q=0.8,en-US;q=0.6,ru;q=0.4
Cache-Control :max-age=0
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94
Safari/537.36
Content-type: application/x-www-form-urlencoded
Content-Length: 835
/* line */*Course=2103*

&coursemodule=95384
&section=7
&module=13
&modulename=resource

Метод "HEAD" аналогічний методу "GET", за винятком того, що клієнту повертається тільки заголовок повідомлення відповіді. Цей метод використовується, коли клієнт хоче мати інформацію про документ, не отримуючи сам документ. Для методу HEAD існує безліч додатків. Наприклад, клієнт може зажадати таку інформацію:

- час зміни документа (ці дані корисні для запитів, пов'язаних з кеш-пам'яттю);
- розмір документа (необхідний для компонування сторінки, оцінки часу передачі, визначення необхідності запиту більш компактною версією документа);
- тип документа (дозволяє клієнту отримувати документи тільки певного типу);
- тип сервера (дозволяє створювати спеціалізовані запити).

Метод "DELETE" використовується для видалення ресурсу певного URI. Для виконання цього методу необхідно, щоб користувач мав відповідні права на видалення URI. Зазвичай ця можливість для користувачів, які не пройшли авторизацію на сервері, заборонена. Запит за допомогою методу DELETE може виглядати таким чином:

DELETE /pub/user1/temp.txt HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94
Safari/537.36

Метод "PUT" використовується, коли клієнт бажає зберегти переданий на сервер ресурс з ідентифікатором URI. Сам збережений ресурс знаходиться в тілі запиту. Як і для методу DELETE, для виконання цього методу користувач повинен мати необхідні права на каталог, куди поміщається ресурс. Розмір тіла повідомлення визначається полем заголовка "Content-Length", це поле є обов'язковим для використання методу PUT. Приклад запиту за допомогою методу PUT:

PUT /pub/user1/upload/newpage.htm HTTP/1.1
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94
Safari/537.36
Content-type: text/html
Content-Length: 60
/ line */*
<HTML><BODY>My new Page is UNDER CONSTRUCTION
</BODY></HTML>

До специфікації HTTP/1.1 були додані кілька нових методів [15].

Метод "OPTIONS" запитує інформацію про комунікаційні параметри сервера. Зазначений ресурс не запитується і, якщо це CGI-модуль, не запускається. Щоб зробити запит про всі сервери в цілому, замість URI-запиту слід використовувати символ '*'. Запит за допомогою методу OPTIONS може виглядати таким чином:

*OPTIONS * HTTP/1.1*
Host: www.hneu.edu.ua
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94
Safari/537.36

Метод "TRACE" застосовується для організації локальної петлі (loopback) на прикладному рівні. HTTP-запит у використанні методу TRACE не повинен містити тіла. З отриманням такого запиту сервер повинен негайно відправити інформацію, що надійшла назад в тілі HTTP-відповіді. Цей метод дозволяє трасирувати можливі з'єднання й отримувати інформацію про час проходження й обробки запитів. Крім того, проаналізувавши поле заголовка Via, клієнт може дізнатися, через які proxy-сервери пройшов запит. Поле заголовка Max-Forwards дозволяє обмежувати довжину ланцюжка запитів, що може бути корисним для тестування нескінченних циклів в ланцюжку proxy-серверів.

HTTP-методи характеризуються двома властивостями: ідемпотентністю та безпечністю.

Ідемпотентність – це можливість виконати той самий запит до сервісу кілька разів, проте відповідь кожен раз буде однаковою (крім помилок і застарівання).

Безпечність означає, що звернення до сервера не змінює вміст. Так, GET може бути викликаний багато разів, але він не змінить вмісту.

У табл. 1.1 наведені основні HTTP-методи із відповідністю до цих властивостей.

Таблиця 1.1

Властивості HTTP-методів

HTTP-метод	Ідемпотентність	Безпечність
DELETE	Так	Ні
GET	Так	Так
HEAD	Так	Так
OPTIONS	Так	Так
PATCH	Так	Ні
POST	Ні	Ні
PUT	Так	Ні

Поля заголовків запиту та відповіді

Загальний заголовок "General Header". Цей заголовок може використовуватися як в запиті, так і відповіді сервера:

- *поле "Date:"* містить дату та час побудови повідомлення (запиту або відповіді). Формат дати описаний в RFC-822;
- *поле "Pragma:"* використовується для установки спеціальних директив учасникам з'єднання. Наприклад, опція "no-cache" не дозволяє проміжним об'єктам з'єднання використовувати кеш у відповіді на запит.

У специфікації HTTP/1.1 в основний заголовок "General-Header" повідомлення були додані такі поля.

- *Поле "Cache-Control:"* дозволяє більш гнучко управляти механізмами кешування інформації, встановлювати обмеження на час і об'єкти кешування, управляти обмеженнями на запити з кешу певних клієнтів та ін.
- *Поле "Connection:"* управляє параметрами з'єднання. У специфікації HTTP/1.0 для використання постійного з'єднання клієнт повинен вказати в цьому полі значення "Keep-Alive". Як зазначалось в специфікації HTTP/1.1 постійні з'єднання використовуються за замовчуванням. Тому клієнтові HTTP/1.1 не обов'язково вказувати значення "KeepAlive" в цьому полі для використання постійного з'єднання, і в загальному випадку це поле може бути відсутнім. Якщо ж клієнт, який використовує HTTP/1.1, не бажає встановлювати постійне з'єднання, то він повинен вказати значення "Close"

в поле Connection. У цьому випадку сервер розірве з'єднання відразу після відправки відповіді клієнту. У специфікації HTTP/1.1 поле Connection зі значенням "Close" може також вказуватися й у відповіді. Таким чином, сервер повідомляє клієнтові, що з'єднання буде розірване відразу після закінчення передачі HTTP-відповіді, і клієнт не повинен більше посилати запити за цим з'єднанням. Сервер, отримавши в заголовку запиту поля Connection зі значенням "Keep-Alive", може включити в заголовок відповіді поле "Keep-Alive". У цьому полі він може вказати параметри постійного з'єднання. Наприклад, час тайм-ауту, після якого дзвінки будуть розірвані, якщо від клієнта не отримано жодних даних.

- *Поле "Transfer-Encoding:"* використовується для вказівки типу кодування тіла запиту або відповіді під час передачі його мережею. У специфікації HTTP/1.1 в основному використовується тип кодування "chunked". У цьому випадку тіло передається частинами (chunk). Для кожної частини в першому рядку вказується розмір частини, потім – вміст, після якого йдуть символи <CRLF>. Остання частина має нульовий розмір і може містити додаткові параметри, що завершуються також символами <CRLF>. Таким чином, можна передавати тіло будь-якого розміру, не вказуючи в заголовку поля ContentLength. Це може бути корисним, наприклад, у динамічній генерації HTTP-відповідей.
- *Поле "Upgrade:"* містить специфікації інших протоколів, за якими можна встановити з'єднання з даним сервером (або клієнтом), крім HTTP.
- *Поле "Via:"* використовується для журналізації проміжних Proxy-агентів або шлюзів, через які пройшов запит (або відповідь). Це поле подібне полю "Received:" в форматі поштового повідомлення.

Наприклад, заголовок "General Header" може виглядати так:

Date: Mon, 09 May 2016 14:16:34 GMT
Pragma: no-cache
Connection: Close

Заголовок запиту "Request-Header" з'являється тільки в запитах клієнтів. Він дозволяє клієнтам відправляти на сервер додаткову інформацію про себе.

- *Поле "Authorization:"* містить інформацію для аутентифікації користувача. Відповідь на запит, що містить поле "Authorization:", не кешується проміжними об'єктами з'єднання.
- *Поле "From:"* може містити email-адресу користувача. Поле може використовуватися для збирання статистики відвідування Web-сервера.

- *Поле "If-Modified-Since:"* використовується у роботі методом "GET". Якщо запитуваний ресурс не змінювався з моменту, зазначеного в цьому параметрі, даний ресурс не повертається; за запитом повертається тільки заголовок повідомлення відповіді та відповідний код повернення. Це поле може використовуватися для контролю над вмістом кешу.
- *Поле "Referer:"* містить URI попереднього ресурсу. Цей параметр може використовуватися сервером, наприклад, для оптимізації кешування, побудови шляхів пошуку тощо.
- *Поле "User-Agent:"* містить інформацію про програмне забезпечення клієнта.

Заголовок запиту "Request-Header" був розширений новими полями в специфікації HTTP/1.1:

- *Поле "Accept:"* наказує відповідному серверу використовувати у відповіді тільки зазначені формати даних. Наприклад, якщо клієнт розуміє тільки текстові дані та картинки в форматі "gif", запит повинен містити:

Accept: text/, image/gif*

- *Поля "Accept-Charset:", "Accept-Encoding:", "Accept-Language:"* визначають, відповідно, допустимі у відповіді значення символьного набору, типів кодувань і використовуваної національної мови. Крім того, запит може містити поле "Content-Language:", що описує основні (але не всі) національні мови, які використовуються у формуванні тексту повідомлення.
- *Поле "Host:"* повинно містити IP або DNS-адресу хоста, у якого запитується URI ресурс. У загальному випадку через двокрапку може бути вказаний HTTP-порт, який використовується цим хостом (якщо порт не вказаний, то за замовчуванням використовується 80 порт). Це може знадобитися, наприклад, для організації прозорих (transparent) proxy-серверів. Зазвичай за використанням поля Host у запиті вказується відносний URI, тобто без адреси хоста, а адреса хоста вказується в полі Host (наприклад, Internet Explorer завжди вказує в запиті відносний URI, за винятком випадку, коли він налаштований на роботу безпосередньо через proxy-сервер). Таким чином, запит може пройти через ланцюжок proxy-серверів без зазначення будь-яких додаткових полів і опцій. Це поле є обов'язковим у використанні HTTP версії 1.1.

GET /examples/mainpage.htm HTTP/1.1

…

Host: www.hneu.edu.ua:8000

- *Поля "If-Match:", "If-None-Match:", "If-Range:", "If-Unmodified-Since:"* встановлюють різні прапори на попередні запити зазначеного ресурсу для більш ефективного управління кешем сервера і/або проміжних агентів.
- *Поле "Max-Forwards:"* містить максимальну кількість проміжних агентів, через які може пройти запит. Це поле може використовуватися тільки разом з методами TRACE й OPTIONS. Кожен проміжний агент зменшує це поле на одиницю перед передачею запиту наступному агенту або кінцевому серверу. Якщо у прийомі запиту це поле містить нуль, то проміжний агент не повинен передавати запит далі, а має сформувати HTTP-відповідь для клієнта.
- *Поле "Proxy-Authorization:"* дозволяє клієнту реєструватися на хостах проміжних агентів з'єднання. Це поле може слугувати для аутентифікації клієнта на всьому шляху до запитуваного ресурсу.
- *Поле "Range:"* дозволяє контролювати розмір переданого повідомлення. За допомогою цього поля можна отримати не весь документ цілком, а тільки деяку його частину. Це дозволяє отримати запитаний ресурс частинами або отримати тільки частину ресурсу розривом з'єднання. Як параметр для цього поля вказується одиниця вимірювання розміру частин, а далі після символу "=" задається один або кілька діапазонів, розділених символом ",". Наприклад, bytes = 0-19,100-149, -10 дозволяє отримати перші 20 байт, 50 байт, починаючи з позиції 100, і останні 10 байт.

Наприклад, заголовок "Request-Header" може виглядати так:

Host: www.hneu.edu.ua
Authorization: Basic U2Fua292OlNlcmdleQ==
From: is@hneu.edu.ua
If-Modified-Since: Mon, 09 May 2016 13:41:57 GMT
Referer: http://www.hneu.edu.ua/home.html
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94
Safari/537.36
*Accept: image/gif, image/x-xbitmap, image/jpeg, image/png, */**
Accept-Language: uk
Accept-Encoding: gzip, deflate
Range: bytes=0-99

Заголовок "Response-Header" дозволяє передавати додаткову інформацію обробки запиту, яку не можна помістити в рядок статусу.

- *Поле "Location:"* містить повний URI створеного або переміщеного ресурсу. Наприклад, якщо клієнт відправив запит на WWW-сервер з метою отримання будь-якого документа, який був переміщений в інший каталог або на інший сервер, і запитуваний сервер відповідає "Location: ...", то запит клієнта автоматично перенаправляється за URI, вказаною сервером.
- *Поле "Server:"* містить специфікацію програмного забезпечення WWW-сервера, що відповідає на запит.
- *Поле "WWW-Authenticate:"* містить параметри схеми та простору аутентифікації.

У специфікації HTTP/1.1 в основний заголовок "Response-Header" повідомлення були додані такі поля.

- Поле "Age:" містить кількість секунд, витрачений сервером на відправку відповіді на цей запит. Поле дозволяє контролювати продуктивність різних серверів і проміжних об'єктів мережі.
- *Поле "ETag:"* містить ключові слова даного ресурсу, за якими ресурс ідентифікується як на сервері, так і в кеші проміжних агентів.
- *Поле "Proxy-Authenticate:"* містить параметри схеми та простору аутентифікації для proxy-сервера.
- *Поле "Retry-After:"* наказує клієнту повторити запит через указану кількість секунд або після вказаної дати.
- *Поле "Warning:"* містить додаткову інформацію про статус обробки запиту – своєрідна система підлеглих кодів коду статусу відповіді.

Наприклад, заголовок "Response-Header" може виглядати так:

Location: http://www.hneu.edu.ua/newsite/NewHome.html
WWW-Authenticate: basic realm="HNEU Security"
Server: Apache/2.2.21 (FreeBSD) mod_ssl/2.2.21 OpenSSL/0.9.8q DAV/2 PHP/5.3.8 with Suhosin-Patch

Заголовок переданого повідомлення "Entity-Header". Містить інформацію про структуру та формат тіла запиту або, якщо тіло повідомлення відсутнє, інформацію про запитуваний ресурс. Може з'являтися як у запиті клієнта, так і відповіді сервера.

- *Поле "Allow:"* містить список методів, підтримуваних ресурсом. Це поле має виключно інформаційний характер.
- *Поле "Content-Encoding:"* містить ідентифікатор типу додаткового кодування ресурсу (для використання цього ресурсу він повинен стверджуватися зазначеним алгоритмом).
- *Поле "Content-Length:"* визначає довжину тіла повідомлення.
- *Поле "Content-Type:"* містить тип ресурсу, відповідно до специфікації MIME, і таблицю кодування поданих даних (UTF-8, windows-1251 та ін.).
- *Поле "Expires:"* містить дату закінчення терміну дії ресурсу. Це поле, наприклад, може використовуватися в механізмах кешування як параметр заборони кешування застарілих ресурсів. Це поле може також інформувати користувача про те, що інформація, надана цим ресурсом, застаріла.
- *Поле "Last-Modified:"* містить дату та час останньої зміни ресурсу.

У специфікації HTTP/1.1 розділ "Entity-Header" розширений полями сімейства "Content *:", а саме: "Content-Base:", "ContentLanguage:", "Content-Location:", "Content-MD5:", "Content-Range:". Вони описують, відповідно, формат, національний алфавіт, розташування частин ресурсу (якщо такі є), алгоритм шифрування, розмір частин і всього повідомлення.

Наприклад, заголовок "Entity-Header" може виглядати так:

Allow: GET, HEAD
Content-Encoding: x-gzip
Content-Length: 3495
Content-Type: text/html; charset= utf-8
Expires: Mon, 2 May 2016 14:04:40 GMT
Last-Modified: Sun, 01 May 2016 14:04:38 GMT

У розділі "Extension-Header" містяться додаткові поля, які можуть бути включені в специфікацію пізніше. Усі нерозпізнані поля повинні ігноруватися HTTP-клієнтами, а proxy-сервери – передавати їх без змін. Таким чином, функціональність HTTP-протоколу може бути розширена без зміни загальної структури запитів і відповідей. Слід зазначити, що деякі з полів, додані до специфікації HTTP/1.1, для забезпечення сумісності включені як додаткові поля в специфікацію HTTP/1.0. Одне з таких полів – це поле "Connection:", яке спочатку було відсутнє в специфікації HTTP/1.0. До специфікації HTTP/1.1 додані декілька полів, які використовуються для

роботи з proxy-серверами. Наприклад, поле X-Forwarder-For використовується proxy-серверами для вказівки IP-адреси клієнта, для якого виконується запит. Поле Proxy-Connection управляє параметрами з'єднання з proxy-сервером.

Механізми аутентифікації

Протокол HTTP надає простий механізм аутентифікації користувача ресурсів WWW-сервера. Механізм побудований на обміні інформацією аутентифікації між клієнтом і сервером, що містить ресурс. Механізм дозволяє вибирати схему аутентифікації та рівень секретності переданих даних.

Доцільно розглянути найбільш поширену схему, яка застосовується в аутентифікації, – "Basic".

Наприклад, клієнт зробив запит ресурсу, для доступу до якого потрібна аутентифікація, наприклад:

GET /secret/security.html HTTP/1.1

У цьому випадку відповідний сервер відправляє код відповіді 401, який означає, що потрібна аутентифікація, і передає в заголовку відповіді поле "WWW-Authenticate:", що містить метод аутентифікації – схему та принаймні одну назву області аутентифікації. Наприклад, так:

HTTP / 1.1 401 Unauthorized
Date: Mon, 09 May 2016 14:04:38 GMT
Server: SAMBAR 4.3
MIME-version: 1.0
WWW-Authenticate: basic realm = "Sambar Security"

Простір ресурсів WWW-сервера може складатися з декількох областей аутентифікації, і кожна з них може використовувати свою схему аутентифікації. Параметр "realm" містить ім'я даної області аутентифікації – символьний рядок. У комбінації з URI-ресурсу це ім'я однозначно визначає ресурс WWW в мережі. У даному прикладі шукана область – "Sambar Security" зі схемою аутентифікації – "Basic".

За цією схемою клієнт повинен передати свій ідентифікатор і пароль для області аутентифікації, зазначеної в параметрі "realm". Якщо на сервері кілька областей і кожна вимагає своїх параметрів аутентифікації – ідентифікаторів і паролів користувача, тоді, щоб скористатися ресурсами будь-якої з них, клієнт повинен зареєструватися в кожній окремо.

Після того як клієнт отримав відповідь сервера, що пропонує йому зареєструватися, клієнт повинен відправити на сервер повторний запит цього ж ресурсу з полем "Authorization:" у заголовку запиту, що містить рядок з ідентифікатором і паролем користувача в кодуванні base64, – це єдиний вид кодування, який використовує схема "Basic". Тоді браузер клієнта зазвичай видає вікно для запиту у користувача імені користувача та пароля. Потім він формує рядок виду ім'я_користувача: пароль, кодує її за base64 і записує в поле "Authorization:". Наприклад, для імені користувача gst і пароля gst цей рядок буде виглядати так:

Authorization: Basic Z3N0OmdzdA ==

Після обробки даного запиту сервер повинен дозволити клієнту користуватися ресурсом. Якщо аутентифікація пройшла невдало, сервер повертає код 403 (forbidden).

Схема "basic" не передбачає будь-якого шифрування для користувача ідентифікатора та пароля для передачі їх мережею, оскільки алгоритм base64 призначений для кодування, а не для шифрування. Тому цю схему можна назвати дуже сумнівною схемою захисту, застосовуваною переважно для тестів, ідентифікації або журналізації входів користувачів, ніж для серйозного захисту ресурсів WWW-сервера.

Механізм Cookies

Як вже було зазначено, в загальному випадку для кожного запитуваного ресурсу відкривається нове з'єднання. Проте ні HTTP-клієнт, ні HTTP-сервер не зберігають інформацію навіть про останнє з'єднання. Тому, якщо HTTP-клієнт після запиту будь-якого ресурсу повторно запитує з того ж сервера інший ресурс, сервер не може визначити, той же це клієнт чи інший. Водночас для вирішення багатьох завдань необхідно ідентифікувати клієнта на час сеансу роботи користувача, або навіть між сеансами користувача [1 – 4]. Наприклад, це може знадобитися для організації різних систем голосування й опитування користувачів, віртуальних магазинів, для аутентифікації користувачів і збереження різних налаштувань для кожного користувача. Використання постійного HTTP-з'єднання в цьому випадку не вирішує проблему, оскільки таке з'єднання призначене для послідовного запиту декількох документів. Постійне з'єднання не може залишатися активним на час усього сеансу роботи користувача, оскільки воно буде розірване сервером після закінчення порівняно невеликого тайм-аута, якщо від клієнта не надходить жодних даних.

Для вирішення цієї проблеми протоколу HTTP розроблений спеціальний механізм cookies. У цьому випадку інформацію про з'єднання повинен зберігати HTTP-клієнт, наприклад, браузер. Для реалізації цього механізму використовуються два додаткових поля HTTP-заголовка Set-Cookie та Cookie. Схема взаємодії клієнта та сервера з використанням механізму cookies виглядає таким чином:

- клієнт запитує який-небудь документ з сервера;
- сервер включає в заголовок відповіді поле Set-Cookie, в якому повідомляє клієнту інформацію, призначену для збереження, і параметри, що задають область дії цієї інформації;
- з кожним наступним запитом клієнт повертає серверу збережену інформацію за допомогою поля Cookie в заголовку запиту, якщо запитуваний ресурс перебуває в радіусі дії, заданої сервером;
- веб-додаток може проаналізувати значення поля Cookie й ідентифікувати клієнта.

Спочатку механізм cookies був описаний в специфікації "Netscape Communications Persistent Client State HTTP Cookies". У подальшому був прийнятий RFC-2109 "HTTP State Management Mechanism", що описує цей механізм. Загальний принцип роботи та формат полів заголовка у цих специфікацій збігаються, однак вони дещо розрізняються за параметрами заголовків.

У специфікації RFC-2109 поле Set-Cookie виглядає так:

Set-Cookie: NAME = VALUE; Comment = comment; Domain = domain; Max-Age = delta-seconds; Path = path; Secure; Version = 1.

Обов'язковими параметрами є NAME і Version. RFC-2109 описує версію 1. Якщо параметр Version опущений, то використовується специфікація netscape. У цій специфікації замість параметра Max-Age використовується параметр Expires і відсутній параметр Comment. Інші параметри ідентичні в обох специфікаціях. Параметр NAME задає дані, які повинні бути збережені клієнтом. Параметр Domain задає ім'я домена, для якого дійсне значення переданого cookie. За замовчуванням використовується доменне ім'я сервера, який сформував відповідь. Параметр Path задає каталог на сервері, для документів якого дійсне значення переданого cookie. Якщо вказане "Path = /", то cookie дійсні для всього сервера. Якщо параметр опущений, то використовується каталог, з якого був запитаний документ. Якщо присутнє поле Secure, то cookie передаються тільки на захищеному

HTTPS-з'єднанні. Параметри Max-Age й Expires задають термін дії cookie. Водночас Max-Age містить кількість секунд, після закінчення яких переданий cookie вважається недійсним і повинен бути вилучений браузером. Поле Expires містить дату, після настання якої переданий cookie вважається недійсним. Якщо поля опущені, то cookie дійсні тільки на час одного сеансу користувача (тобто поки вікно браузера не буде закрите), і браузер не повинен зберігати значення cookie на диску. Поле Set-Cookie не повинно кешуватися proxy-серверами.

У специфікації RFC-2109 поле Cookie виглядає таким чином:

Cookie: $ Version = 1; NAME1 = VALUE1; $ Path = path1; $ Domain = domain1;
NAME2 = VALUE2; $ Path = path2; $ Domain = domain2

Для кожного значення cookie, параметри якого задовольняють запитуваному ресурсу, браузер включає параметр NAME = VALUE. Параметри $ Path і $ Domain є необов'язковими в RFC-2109 і відсутні в специфікації netscape.

Cookie є лише блоком даних, що зберігається на боці клієнта. Формування та аналіз цих даних повинні виконуватися на стороні сервера, наприклад, за допомогою веб-додатку. Зі збереженням cookie браузер повинен дотримуватися таких обмежень:

- загалом може зберігатися до трьохсот значень cookies;
- кожен cookie не може перевищувати 4Кбайт;
- з одного сервера або домена може зберігатися до двадцяти значень cookie.

Слід зазначити, що механізм cookies не є надійним для ідентифікації користувачів, оскільки браузер може видалити збережені cookies (наприклад, через нестачу вільного місця) або взагалі не підтримувати механізм cookies. Крім того, користувач може заборонити в браузері підтримку cookies або видалити збережені cookies вручну.

Засоби моніторингу HTTP-трафіка

Для моніторингу обміну даними між http-клієнтом і http-сервером існують засоби моніторингу як вбудовані в браузер, так і зовнішні (працюють як локальні проксі). Найбільш популярними зі вбудованих в браузер є Developer Tools (вкладка "мережа" для різних браузерів – рис. 1.1 – 1.4) та firebug (рис. 1.5), а зі зовнішніх засобів моніторингу – це fiddler (http://www.fiddler2.com/fiddler2/).

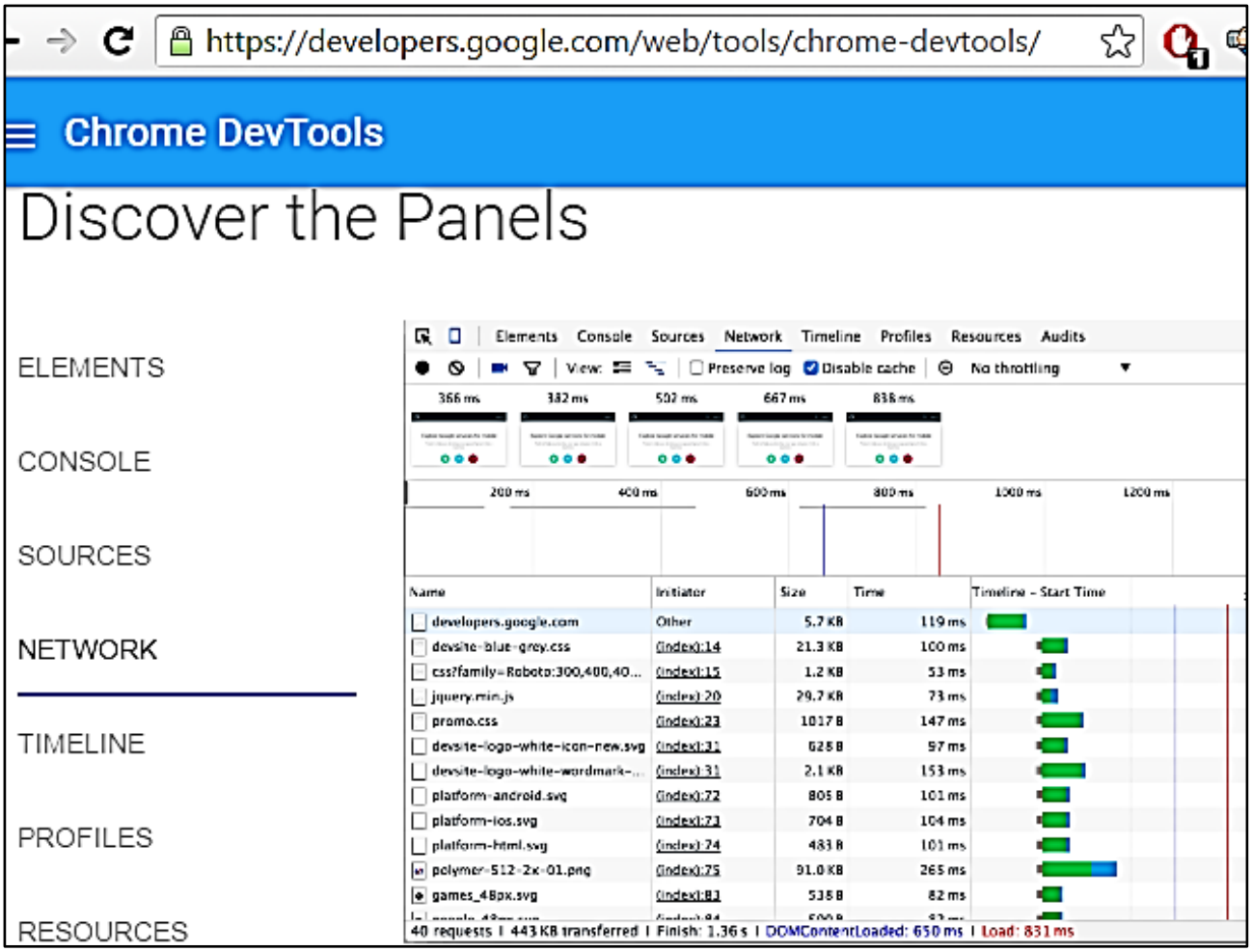


Рис. 1.1. **Сторінка керівництва користувача (розробника) з Chrome Developer Tools з наочним прикладом засобу моніторингу**

На рис. 1.2 зображена панель веб-розробника Google Chrome та вибір стовпця "Protocol" з можливого набору стовпців, що характеризують параметри HTTP-запиту/відповіді. Серед цих параметрів є найбільш важливі поля заголовків і деякі додаткові характеристики запиту/відповіді.

На рис. 1.2 перший запит відправлений за протоколом HTTP/1.1, а наступні – за протоколом HTTP/2 (у панелі розробника позначений як "h2"). Крім того, на рис. 1.2 обрана опція "Disable cache", що вимикає кешування браузера, тому серед кодів стану відсутній код 304.

Якщо клацнути по будь-якому рядку з запитів, то відкриється додаткове вікно з детальною інформацією про відповідний http-запит, яка містить: Request URL, Request Method, Status Code, Remote Address, заголовки відповіді, заголовки запиту та параметри URL та/або дані форми, якщо такі були передані (метод POST).

Рис. 1.2. **Моніторинг HTTP в Google Chrome Developer Tools**

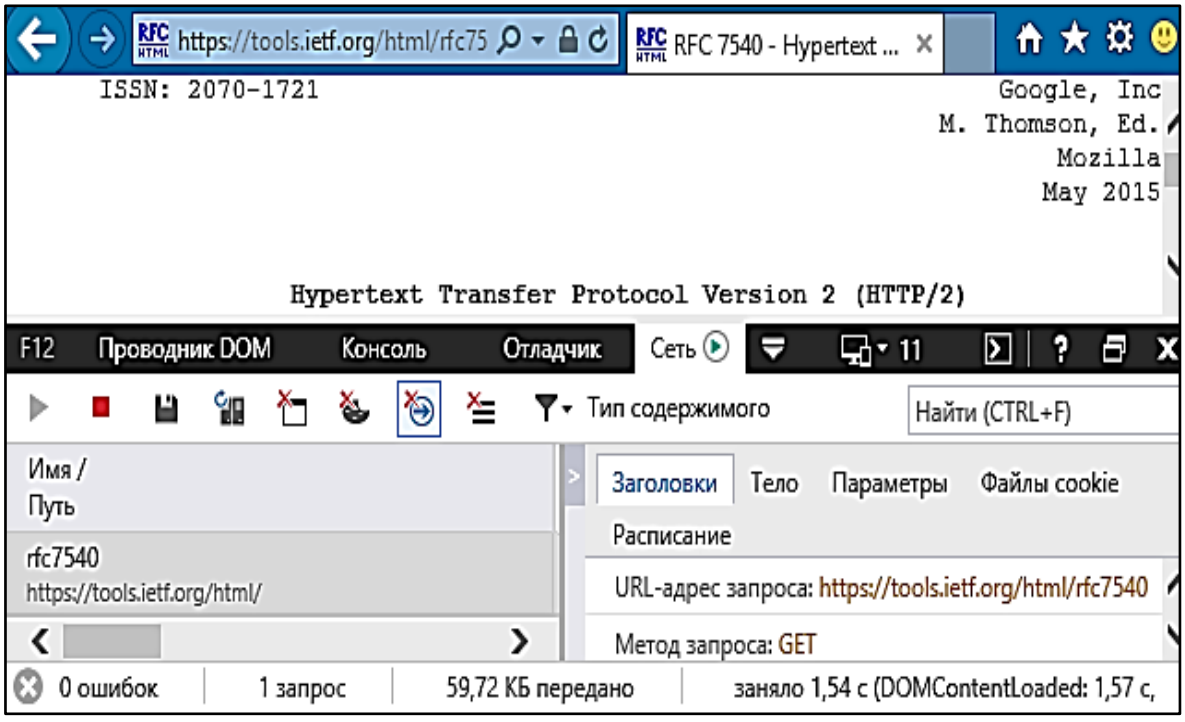


Рис. 1.3. **Моніторинг HTTP в Internet Explorer**

13 лютого 2013 року розробники *Opera* заявили про перехід усієї лінії їх продукції на движки WebKit і V8 в оболонці Chromium, що мають відкритий вихідний код. 4 квітня в світлі подій щодо движка Blink, створеного

корпорацією Google і є Форком движка WebKit, *Opera Software* підтвердила свою позицію щодо прив'язки браузера до проекту Chromium – у нових версіях буде використаний Блинк [26]. Тому і панель веб-розробника має вигляд, аналогічний панелі у Google Chrome (див. рис. 1.4).

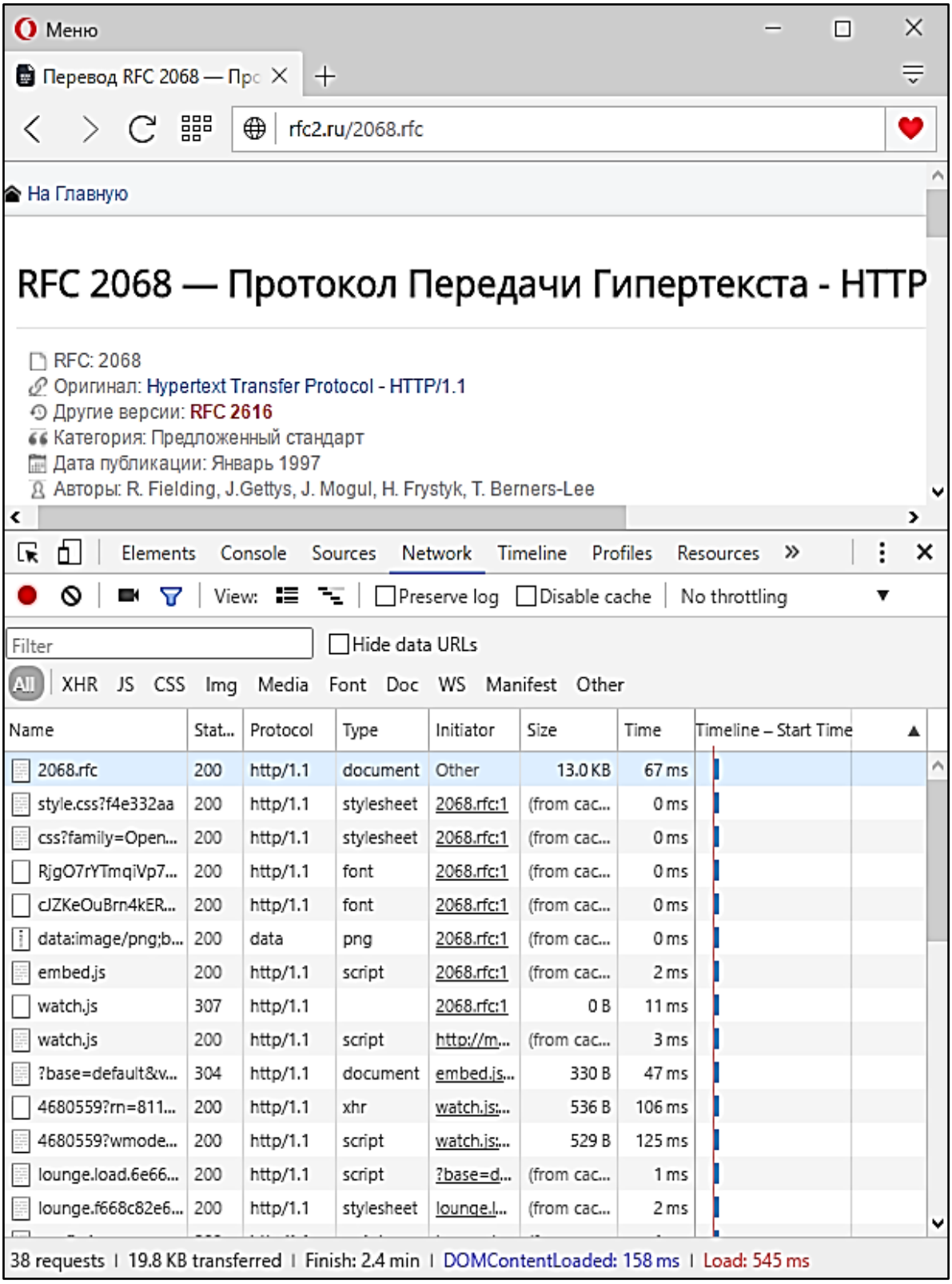


Рис. 1.4. **Моніторинг HTTP в Opera**

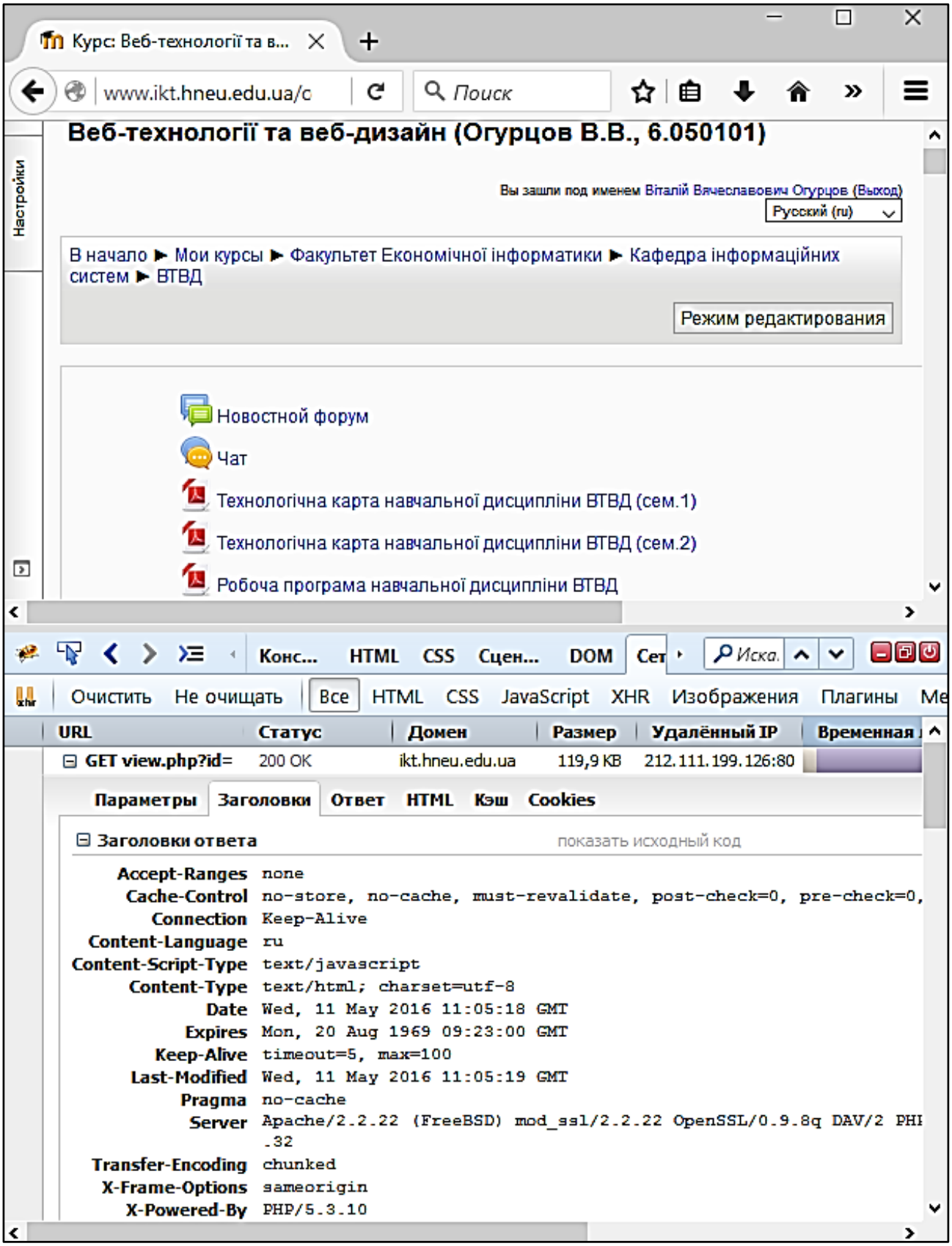


Рис. 1.5. **Моніторинг HTTP у Firebug (Mozilla Firefox)**

Але існують й інші приклади засобів моніторингу http-трафіка, які не поступаються попереднім. Це, наприклад, httpwatch, який встановлюється як плагін для IE (рис. 1.6).

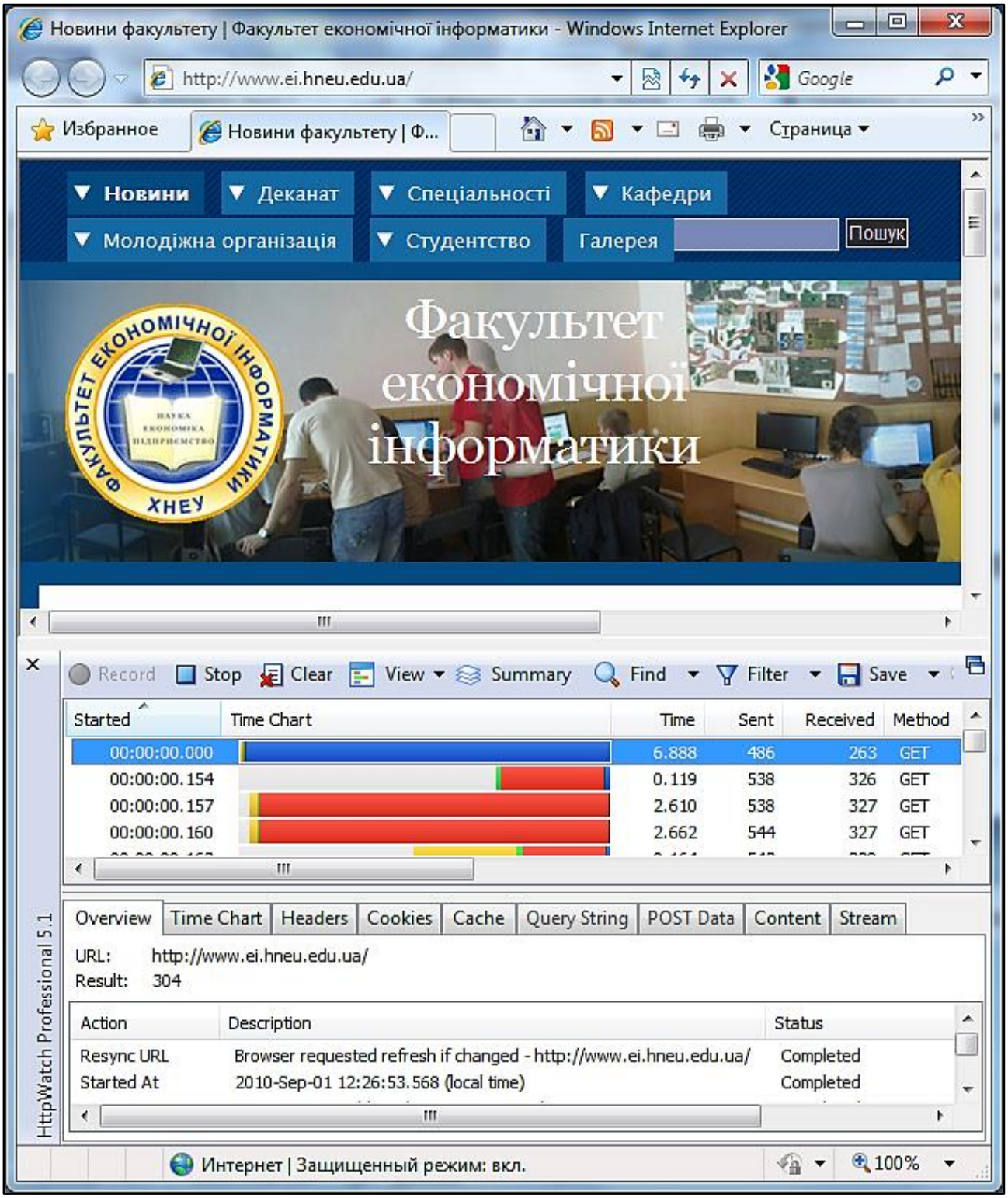


Рис. 1.6. **HTTP-моніторинг за допомогою HttpWatch Professional**

Не завжди в процесі розроблення веб-додатків вистачає вбудованих у браузер засобів моніторингу http-трафіка. Наприклад, у розробці веб-додатка, який звертається до стороннього веб-сервісу, потрібно, щоб http-запит відправлявся з самого додатка, а не з браузера, тобто серверний код додатка, що в цю мить виступає в ролі http-клієнта, і для моніторингу цього трафіка без окремих утиліт не обійтись. Такими утилітами є проксірувальні аналізатори http-трафіка, наприклад Fiddler (рис. 1.7) або Charles (див. рис. 1.8).

Перевагою Fiddler перед Charles є можливість автоматичного перехоплення http-трафіка. А для моніторингу (аналізу) http-трафіка за допомогою Charles необхідно явно в веб-додатку конфігурувати його як proxy-сервер.

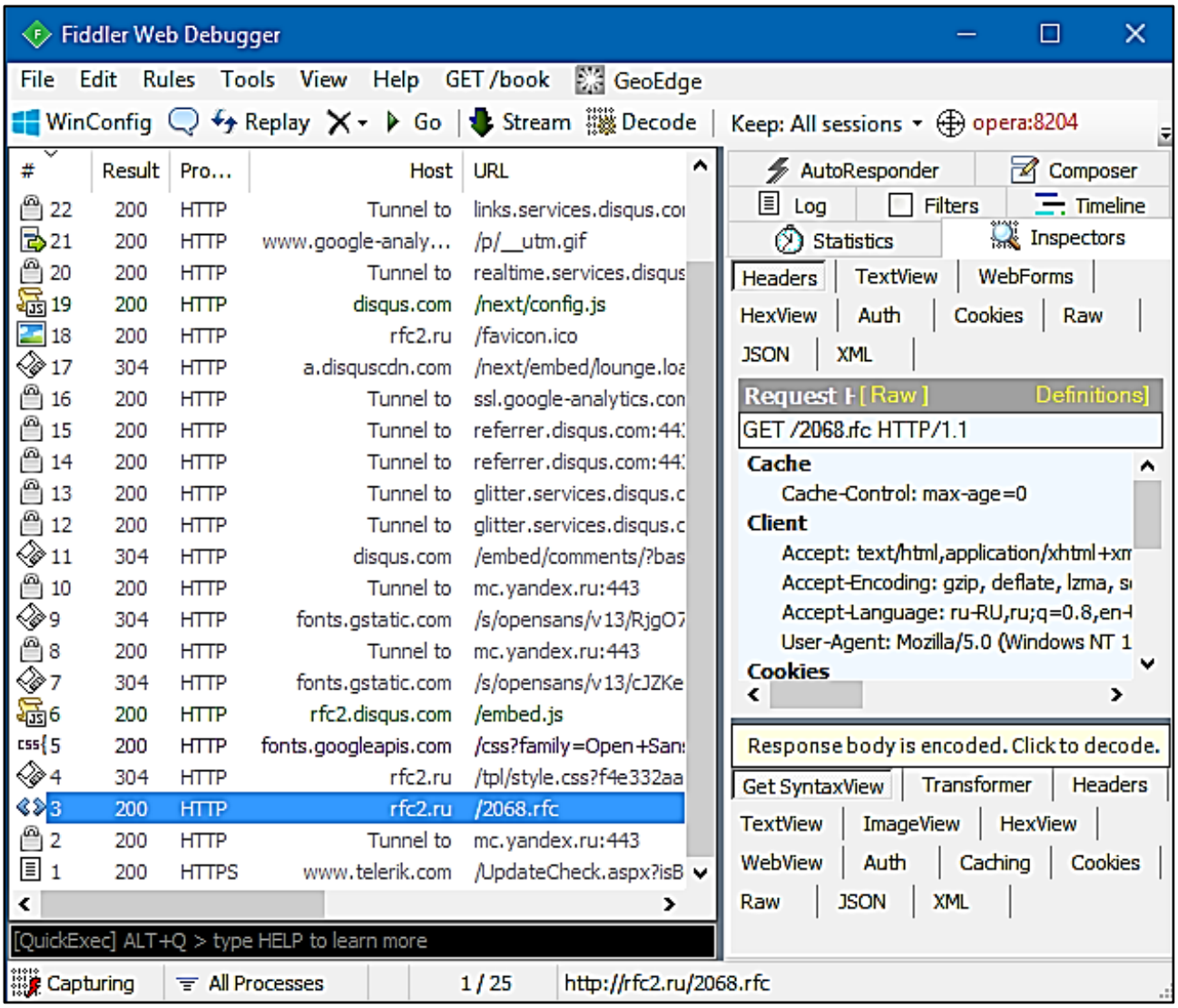


Рис. 1.7. **Веб-відладник Fiddler**

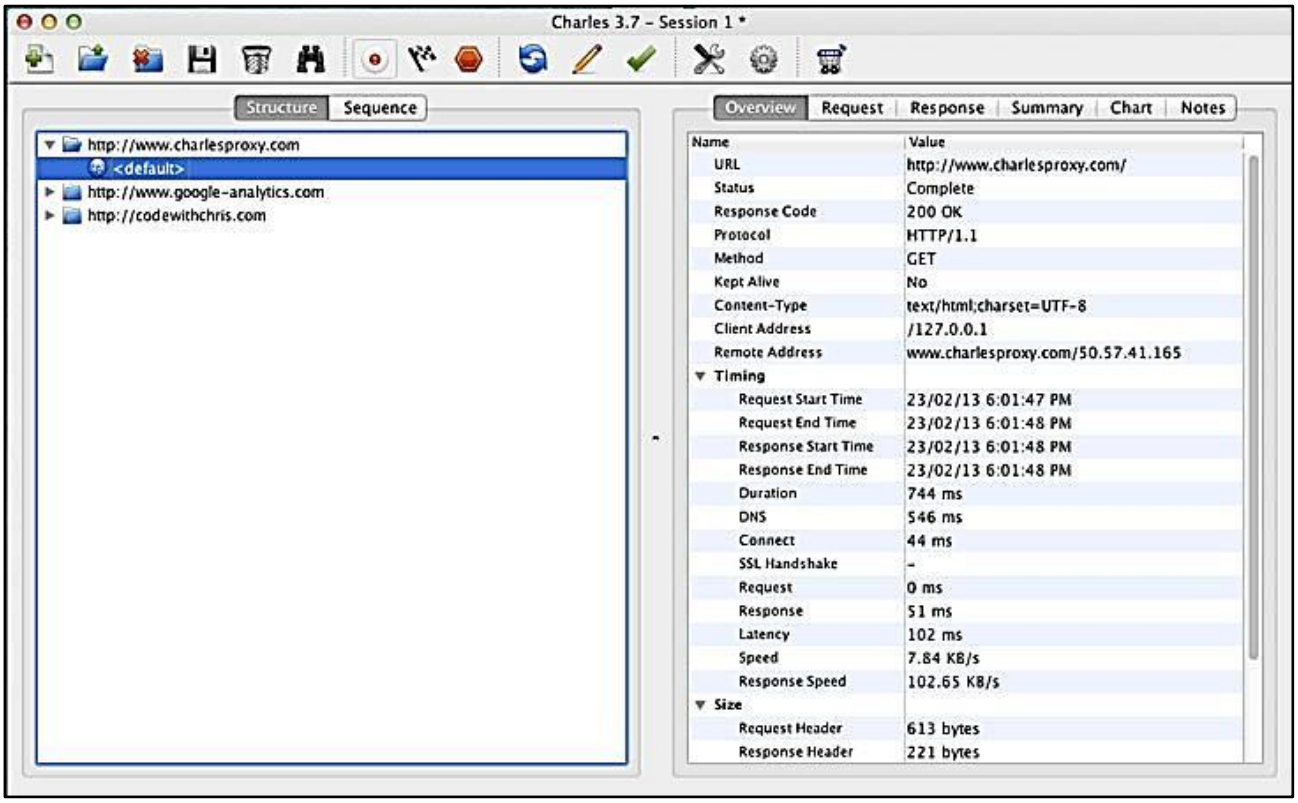


Рис. 1.8. **HTTP-моніторинг Charles**

Моделювання http-запитів і відповідей

У B2B-секторі одні організації надають свої товари (послуги), інші їх купують (використовують). У кожної компанії є своя ІС, і в результаті взаємодії компаній дані у вигляді документів (у загальному випадку – інформаційних масивів) передаються з однієї ІС в іншу. Для автоматизації цих інформаційних процесів усе більше компаній розробляють програмні інтерфейси, які засновані на протоколі HTTP (S). Тому використання засобів моделювання обміну http-повідомленнями ефективне не тільки для вивчення цього протоколу, але й у розробці та тестуванні веб-API.

Доцільно розглянути якісний засіб моделювання обміну http-повідомленнями "Postman" (рис. 1.9). "Postman" існує у вигляді додатка до браузера Google Chrome та додатка до Mac OS.

Рис. 1.9. **Веб-сторінка "Postman"**

Після встановлення "Postman" для його ефективного використання потрібно зареєструватися (рис. 1.10) (для можливості зберігати свої дані та колекції запитів) та увійти під своїм обліковім записом (див. рис. 1.11).

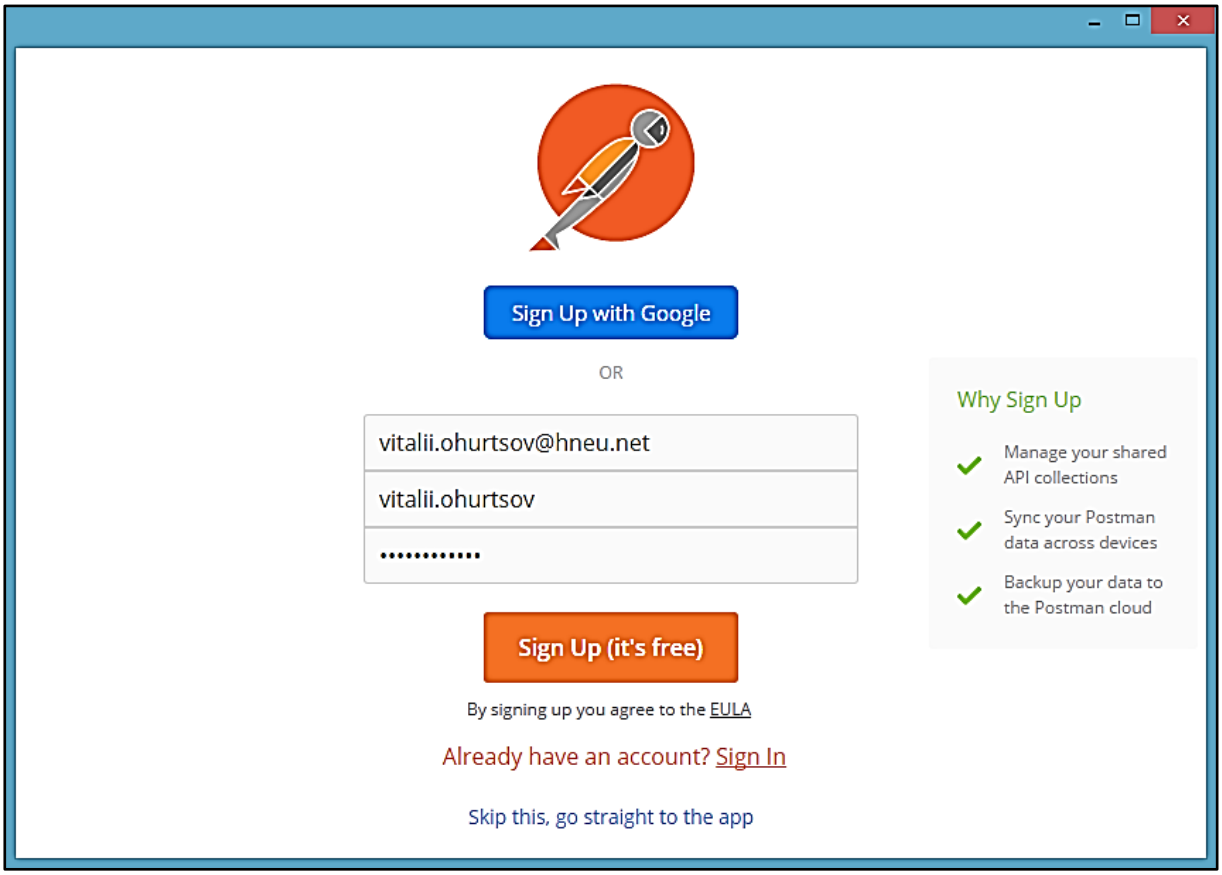


Рис. 1.10. **Реєстрація в "Postman"**

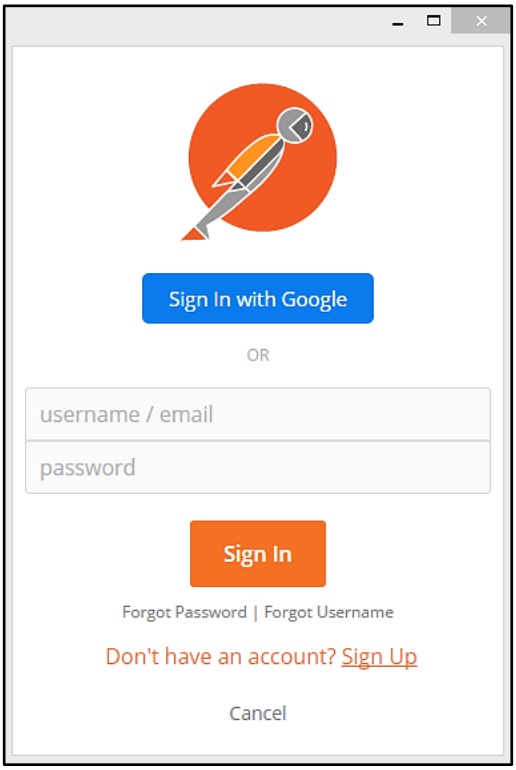


Рис. 1.11. **Вхід в "Postman"**

Для групування запитів можна створити колекції (рис. 1.12), наприклад, для кожного проекту – окрему колекцію запитів або декілька залежно від масштабів проекту.

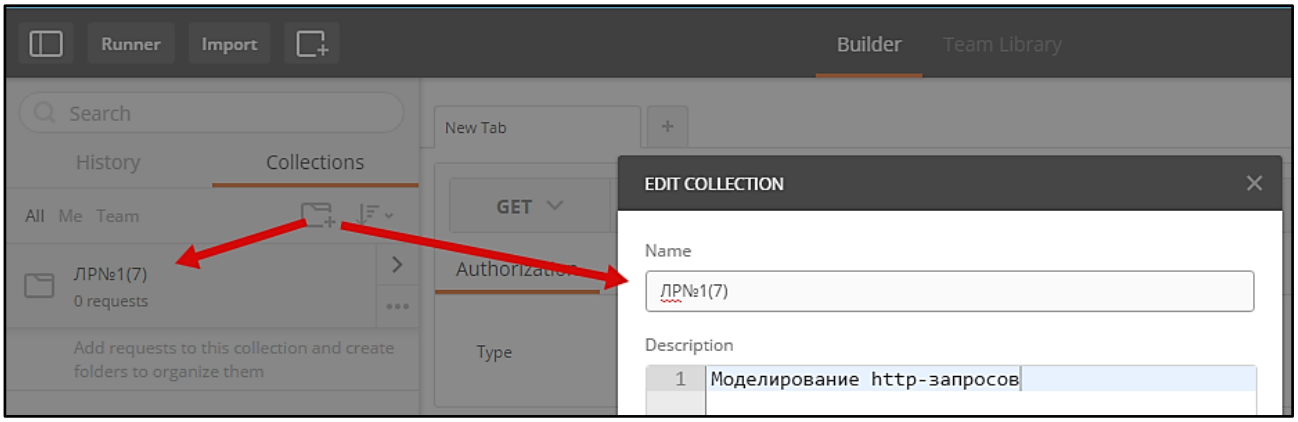


Рис. 1.12. **Створення й оновлення колекції запитів**

Далі можна приступати до створення самих запитів (рис. 1.13). Для подальшого використання запиту його потрібно зберегти у попередньо створену колекцію (рис. 1.14, 1.15). Далі можна відправити запит на вказаний URL, для чого потрібно натиснути на кнопку "Send".

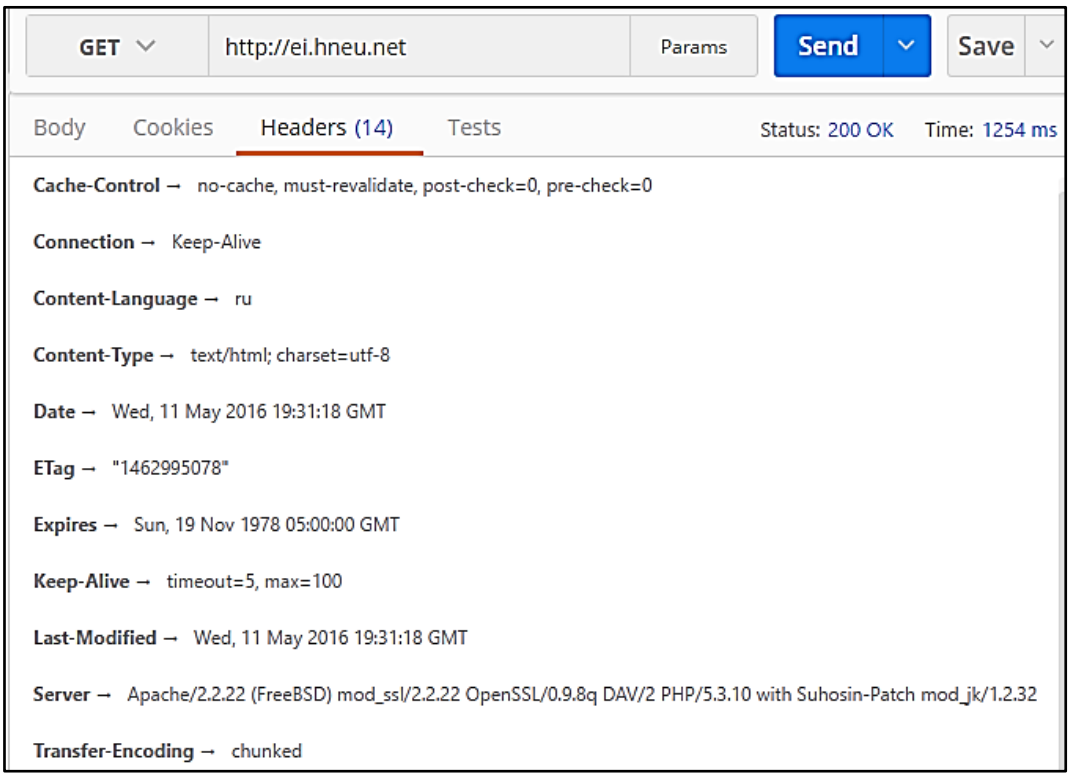


Рис. 1.13. **Створення та відправлення http-запиту**

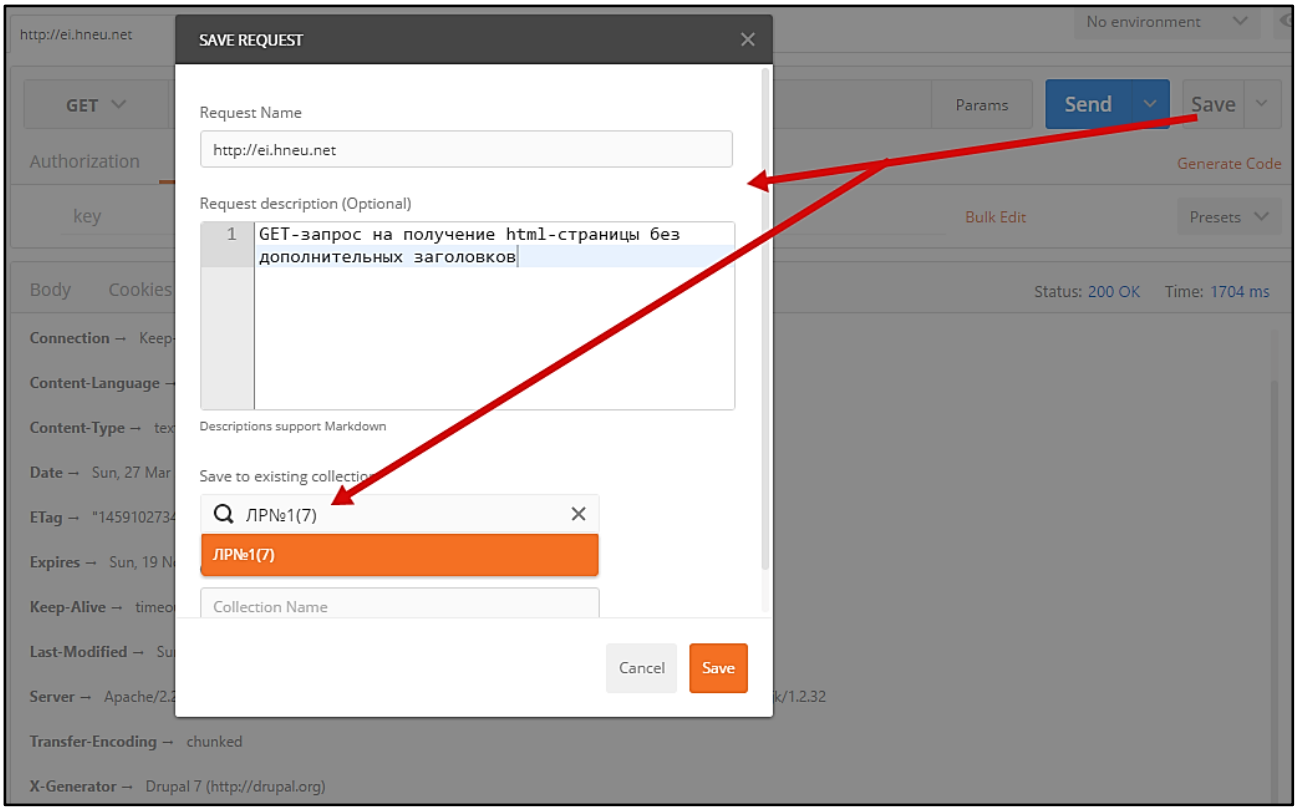


Рис. 1.14. **Збереження запиту до колекції**

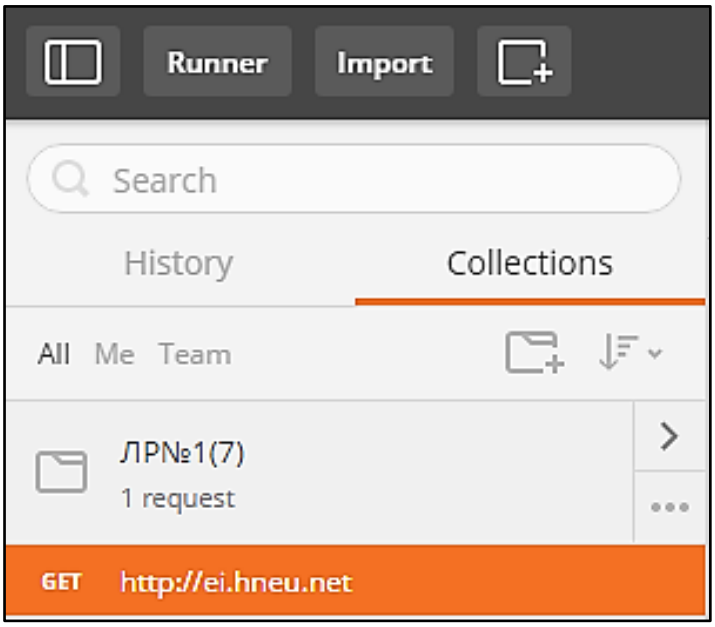


Рис. 1.15. **Приклад колекції із запитами**

Як що ви налаштували запит, відправили його, але не зберегли, то відновити його можна за допомогою історії запитів, доступ до якої розташований у лівій боковій панелі (рис. 1.16).

Рис. 1.16. **Історія відправлених запитів**

У "Postman" доступні для налаштування різні методи авторизації (рис. 1.17).

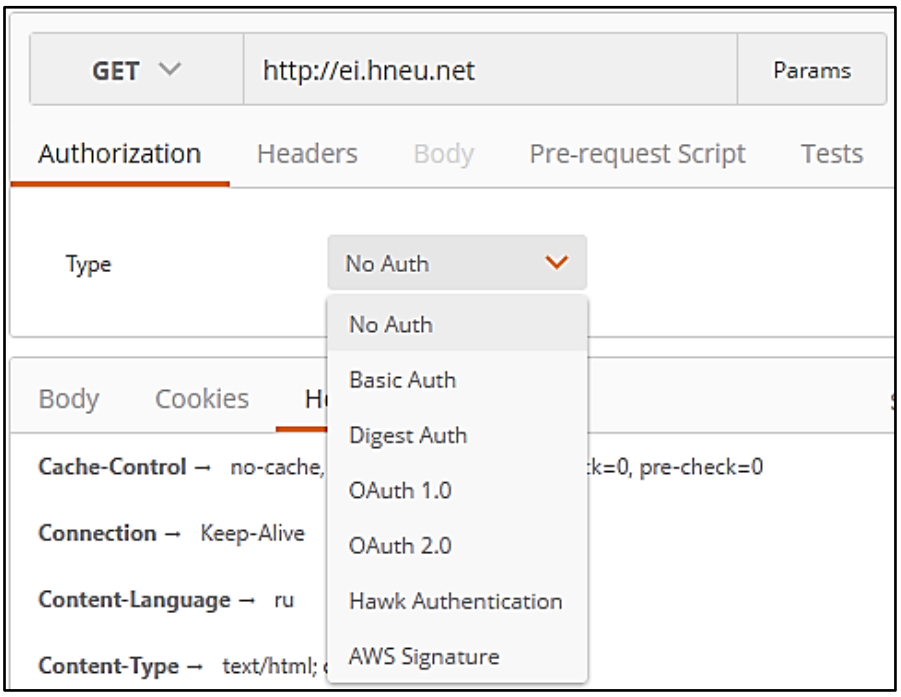
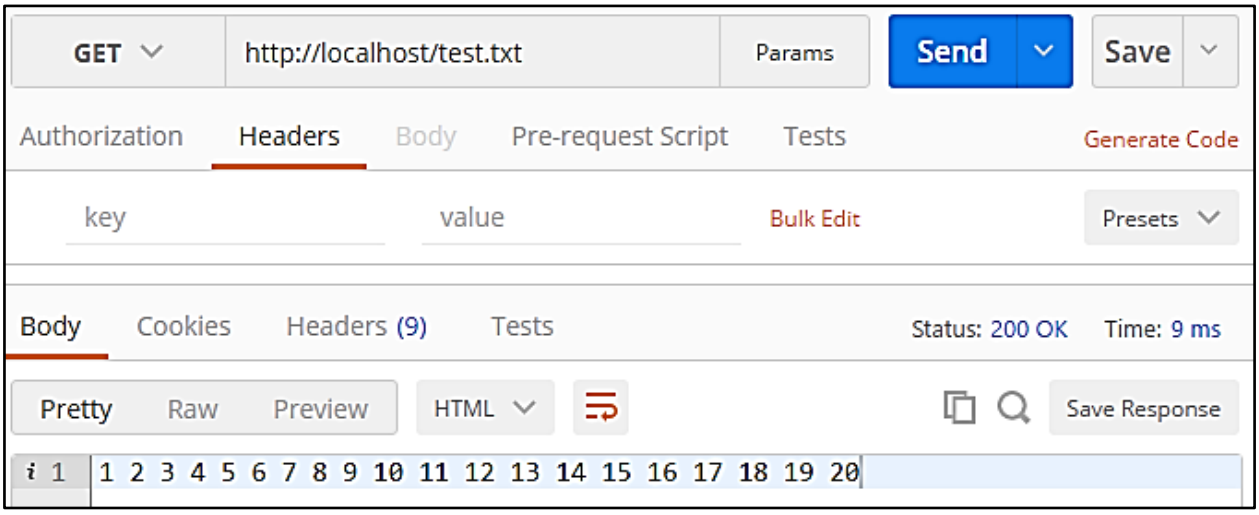
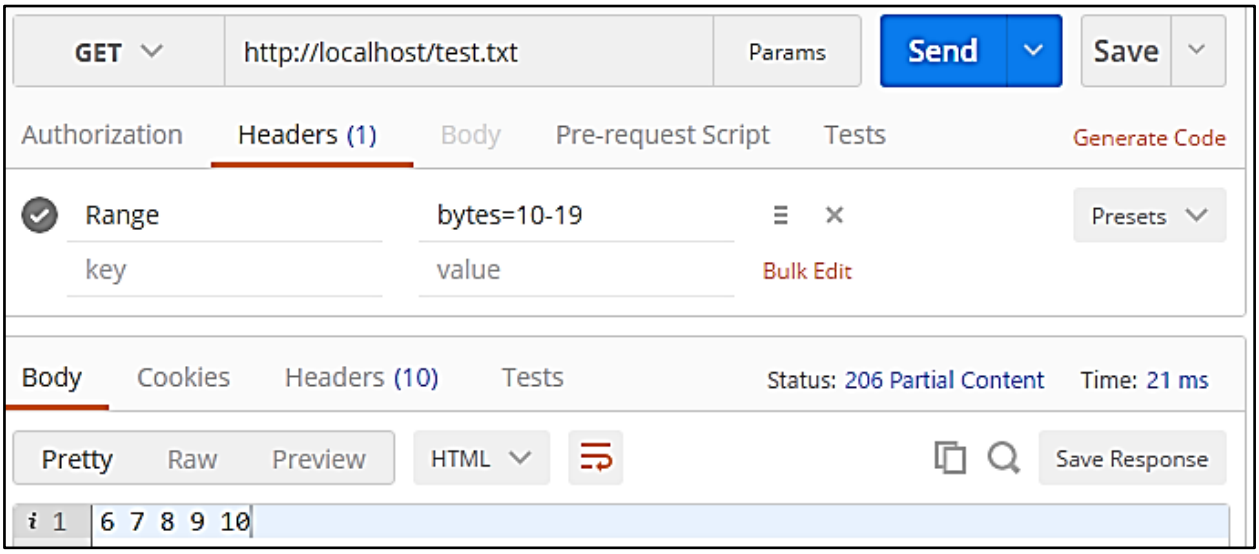


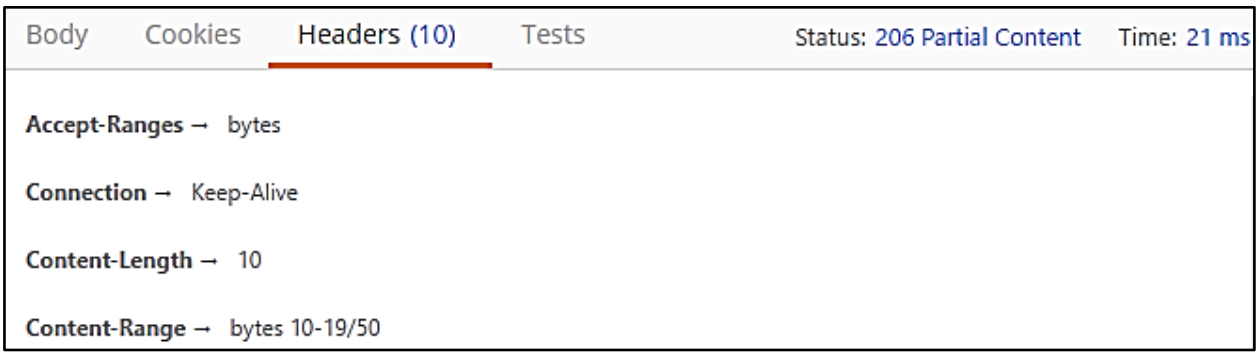
Рис. 1.17. **Методи авторизації**

Для налаштування http-запитів можна встановлювати різні заголовки. Наприклад для отримання частини файла можна додати заголовок діапазону "Range" (див. рис. 1.18). Якщо веб-сервер підтримує цей заголовок і дозволяє для цього ресурсу отримання частини даних, то в тіло відповіді прийде потрібна частина даних. У випадку успішного отримання часткового вмісту файла замість коду стану "200 OK" буде отриманий код стану "206 Partial Content" і додаткові заголовки (Accept-Ranges, Content-Range).



а) запит на отримання повного текстового файла із цифрами від 1 до 20, які розділені між собою пробілом





б) запит на отримання частини текстового файла з 10-го до 19-го байта

Рис. 1.18. **Приклад налаштування запита із додатковими заголовками**

Крім того, "Postman" надає можливість зручно налаштовувати параметри URL (рис. 1.19).

Рис. 1.19. **Приклад використання URL-параметрів запита**

А у випадку зміни методу на POST з'являється можливість додати параметри в тіло запита (рис. 1.20) та обрати тип кодування для них.

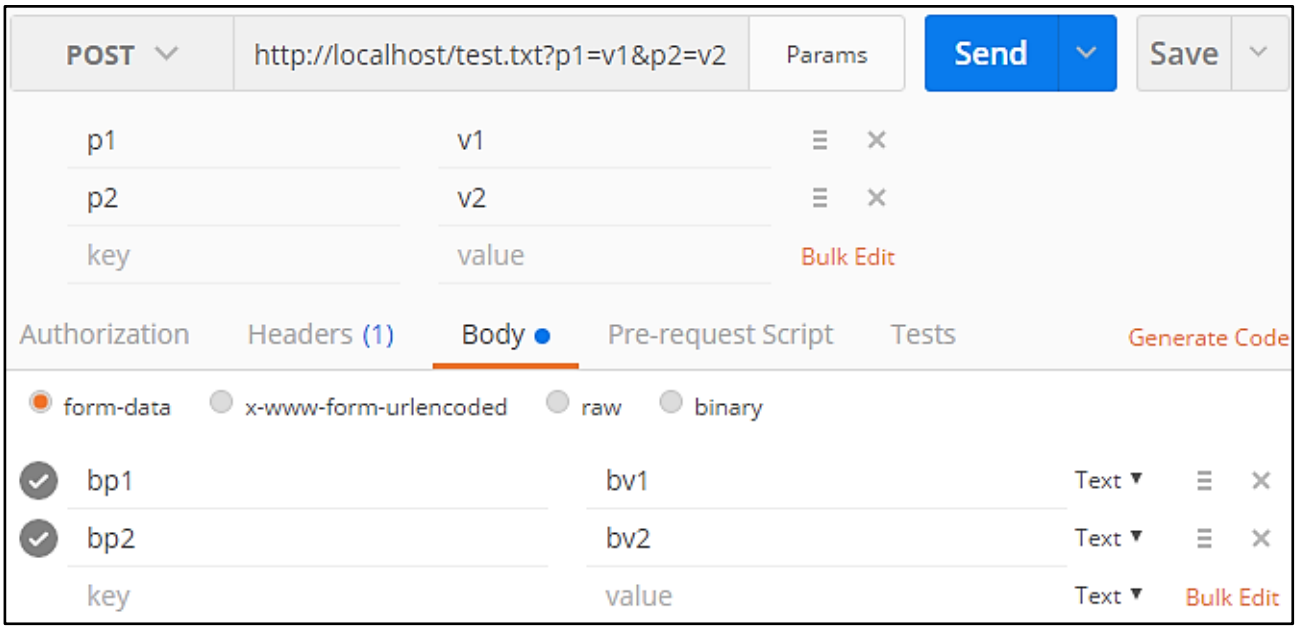


Рис. 1.20. **Приклад використання параметрів у тілі запита методом POST**

Якщо створений запит влаштовує, то за допомогою "Postman" можна згенерувати код різними мовами програмування (рис. 1.21) або в первинному вигляді http-запита. Для цього потрібно обрати "HTTP" (див. рис. 1.22).

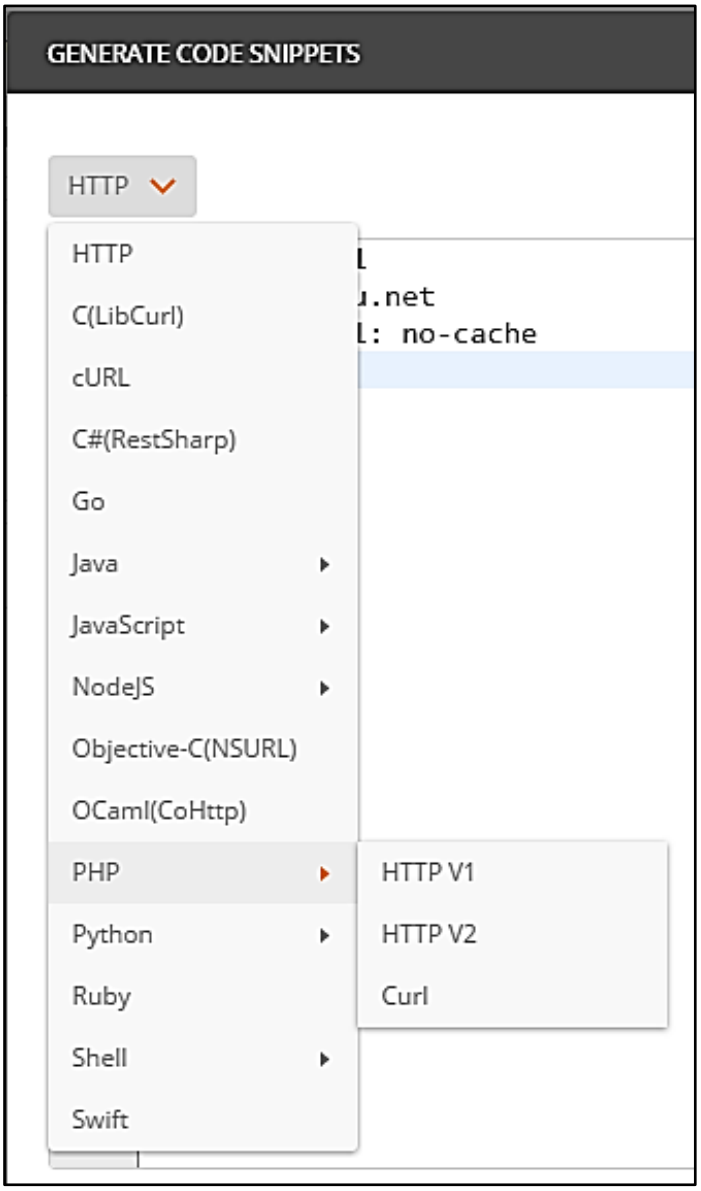


Рис. 1.21. **Генератор програмного коду відправлення http-запита та отримання відповіді різними мовами**

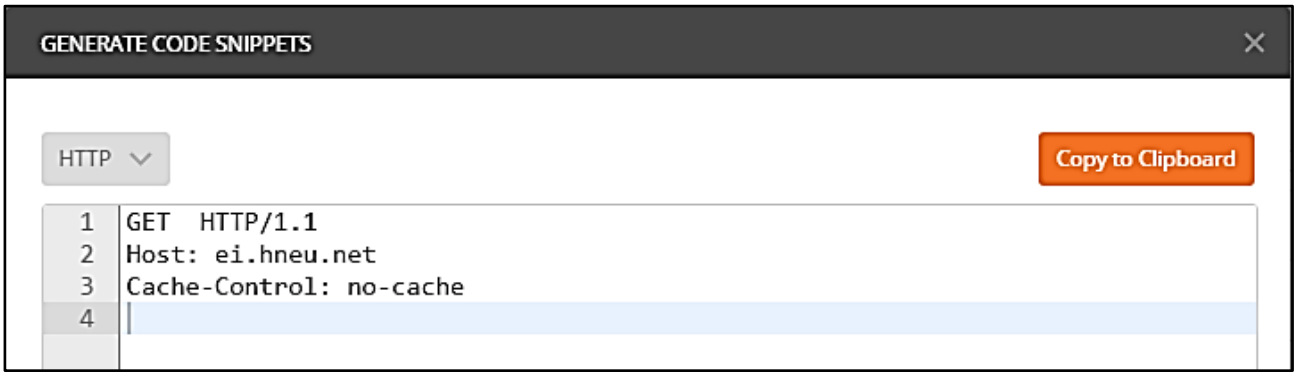


Рис. 1.22. **Генератор http-запита в первинному вигляді**

1.4. Завдання до лабораторної роботи

1. Вивчити специфікацію протоколу HTTP 1.1 [14].
2. Змоделювати обмін http-повідомленнями з максимальною кількістю різних кодів стану (мінімум 7) і різних методів доступу (мінімум 2) або знайти відповідні цьому сайти.
3. За допомогою засобу моніторингу обміну даними за протоколом HTTP (наприклад, firebug: вкладка "мережа") відстежити та зафіксувати в звітах (в електронному вигляді) у вигляді скриншотів кожну частину виконання завдання з пункту 2. Розібратися з усіма атрибутами http-заголовків, зафіксованих у звіті.

1.5. Контрольні запитання

1. Що таке HTTP?
2. Назвіть методи протоколу HTTP.
3. Чим відрізняються методи GET і POST?
4. Назвіть безпечні методи протоколу HTTP.
5. Опишіть (назвіть) загальну структуру HTTP-повідомлення.
6. Які атрибути тега <FORM> і які значення потрібно встановити, щоб можна було передавати файли на сервер за протоколом HTTP?
7. Що таке MIME й яке відношення він має до HTTP?

Лабораторна робота 2
Розроблення веб-додатків за допомогою мови PHP

2.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення основ мови PHP.

2.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити основи протоколу передачі даних прикладного рівня HTTP.

2.3. Загальні тези лабораторної роботи

Історія створення PHP

На відміну від багатьох інших мов програмування, PHP був створений не корпорацією або генієм-програмістом, а звичайним користувачем, Расмусом Лердорфом, у далекому 1994 році. Мета розроблення мови була проста – зробити домашню сторінку Лердорфа більш інтерактивною, а отже, і більш привабливою для відвідувачів. Лердорф розробив базовий синтаксис і написав перший інтерпретатор своєї мови, який отримав назву Personal Home Page Tools – PHP. Цей інтерпретатор міг обробляти лише кілька основних команд, проте початок був покладений [4; 7; 13].

У 1995 році Лердорф допрацював інтерпретатор PHP, з'єднавши його з іншою своєю програмою, яка вміла обробляти HTML-форми (вона іменувалася FI – від "Form Interpretator"). Розроблювач зробив так, щоб інтерпретатор, який отримав назву PHP/FI Version 2, міг ставати частиною веб-сервера. Це нововведення дозволило програмам на PHP виконуватися дуже швидко. Крім того, в тому ж 1995 році інтерпретатор PHP був доповнений можливостями обробки нових команд, зокрема, команд для роботи з серверами баз даних і автоматичного створення gif-файлів (останнє, наприклад, можна використати для генерації кнопок-лічильників відвідувань). PHP/FI був розміщений в Мережі для загального використання, і почалося його розповсюдження.

До кінця 1997 року PHP використовувався понад п'ятдесятьма тисячами сайтів. Веб-майстри швидко оцінили переваги нової мови веб-програмування – такі, як легкість освоєння та багатство можливостей, і незабаром традиційні Perl і C стали здавати свої позиції.

Оскільки вихідний код інтерпретатора був відкритий (а сам інтерпретатор, зрозуміло, безкоштовний), то ентузіасти займалися його доопрацюванням, і влітку 1998 року з'явився на світ PHP3 – дітище Зива Сураські й Енді Гутманса (Zeev Suraski and Andi Gutmans). *PHP3 був створений практично "з нуля"*, позаяк його автори вважали код попередніх версій недостатньо ефективним. Крім того, PHP3 став легко розширюваним продуктом. Будь-хто, створивши на основі певних стандартів модуль розширення PHP (що дозволяє, наприклад, працювати з архівами будь-якого типу), міг цей модуль інтегрувати з програмними файлами PHP без серйозних витрат часу та зусиль.

Уже до кінця 1999 року кількість сайтів, побудованих на основі PHP, переважила за мільйон. Дуже важливою перевагою PHP також було те, що програми, які дозволяли обробляти команди PHP, були створені практично для всіх операційних систем, від Windows до Unix і Linux.

У 2000 році вийшла розроблена компанією Zend Technologies четверта версія інтерпретатора PHP, доповнена безліччю нових функцій.

Сьогодні PHP використовується більш ніж на 20 % сайтів.

Синтаксис PHP

Синтаксис PHP багато запозичив з таких мов, як C, Java і Perl. Файл, оброблюваний сервером, як правило, має розширення php [4; 7; 13].

PHP-код включається в html-код, як зображено на рис. 2.1.

```
<?php  текст_коду  ?>

        або

<? текст_коду; ?>
```

Рис. 2.1. **Включення PHP у HTML**

Коментарі

PHP підтримує коментарі 'C', 'C ++' і оболонки Unix (рис. 2.2).

```
<?php echo "Якийсь текст"; /* Це однорядковий
коментар у стилі c++ */

/* Це багаторядковий коментар у стилі C,
це ще один його рядок */
echo "Ця інформація буде виведена в HTML";

# Це коментар у shell-стилі
echo "Ця інформація буде виведена в HTML";?>
```

Рис. 2.2. **Коментарі в PHP**

Присвоєння значень змінним

Змінні в програмах на PHP відокремлюються символами $ (рис. 2.3).

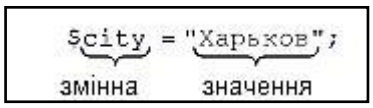


Рис. 2.3. **Змінні в PHP**

Деякі операції

Інкремент/декремент:

++$a Pre-increment (Збільшує $a на 1, потім повертає $a).

$a++ Post-increment (Повертає $a, потім збільшує $a на 1).

--$a Pre-decrement (Зменшує $a на 1, потім повертає $a).

$a-- Post-decrement (Повертає $a, потім зменшує $a на 1).

Арифметичні:

$a + $b Додавання (Сума $a і $b).

$a - $b Віднімання (Різниця $a і $b).

$a * $b Множення (Добуток $a і $b).

$a / $b Ділення (Частка від ділення $a на $b).

$a % $b Modulus (Цілочисельний залишок від ділення $a на $b).

рядкові:

Є дві рядкові операції. Перша – операція ('.'), яка повертає об'єднання з правого та лівого аргументів. Друга – операція присвоєння ('. ='), яка приєднує правий аргумент до лівого аргументу (рис. 2.4).

```php
<?php

    $a = "Hello "; $b = $a . "World!"; // тепер $b
містить "Hello World!"

    $a = "Hello "; $a .= "World!"; // тепер $a
містить "Hello World!"

?>
```

Рис. 2.4. **Строкові операції в PHP**

Вирази порівняння

Вирази порівняння обчислюються в 0 або 1, означаючи FALSE або TRUE (відповідно).

PHP підтримує

> (більше),

>= (більше або дорівнює),

== (дорівнює),

!= (не дорівнює),

< (менше) и <= (менше або дорівнює).

Ці вирази найчастіше використовуються всередині умовних операторів – таких, як **if**.

порівняння:

$a == $b дорівнює TRUE, якщо $a дорівнює $b.

$a != $b не дорівнює TRUE, якщо $a не дорівнює $b.

$a <> $b не дорівнює TRUE, якщо $a не дорівнює $b.

$a < $b менше TRUE, якщо $a строго менше $b.

$a > $b більше TRUE, якщо $a строго більше $b.

$a <= $b менше або дорівнює TRUE, якщо $a менше або дорівнює $b.

$a >= $b більше або дорівнює TRUE, якщо $a більше або дорівнює $b.

Табл. 2.1 – 2.3 демонструють роботу PHP з типами змінних і операторами порівняння, як у випадку вільного, так і в разі суворого порівняння.

Таблиця 2.1

Порівняння типів *$x* і результатів функцій PHP, пов'язаних з типами

Вираз	gettype()	empty()	is_null()	isset()	логічне: *if($x)*
$x = "";	рядок	TRUE	FALSE	TRUE	FALSE
$x = NULL	NULL	TRUE	TRUE	FALSE	FALSE
var $x;	NULL	TRUE	TRUE	FALSE	FALSE
$x не визначена	NULL	TRUE	TRUE	FALSE	FALSE
$x = array();	масив	TRUE	FALSE	TRUE	FALSE
$x = FALSE;	логічне	TRUE	FALSE	TRUE	FALSE
$x = true;	логічне	FALSE	FALSE	TRUE	TRUE
$x = 1;	ціле	FALSE	FALSE	TRUE	TRUE
$x = 42;	ціле	FALSE	FALSE	TRUE	TRUE
$x = 0;	ціле	TRUE	FALSE	TRUE	FALSE
$x = -1;	ціле	FALSE	FALSE	TRUE	TRUE
$x = "1";	рядок	FALSE	FALSE	TRUE	TRUE
$x = "0";	рядок	TRUE	FALSE	TRUE	FALSE
$x = "-1";	рядок	FALSE	FALSE	TRUE	TRUE
$x = "php";	рядок	FALSE	FALSE	TRUE	TRUE
$x = "true";	рядок	FALSE	FALSE	TRUE	TRUE
$x = "FALSE";	рядок	FALSE	FALSE	TRUE	TRUE

Таблиця 2.2

Гнучке порівняння за допомогою ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Таблиця 2.3

Жорстке порівняння за допомогою ===

	TRUE	FALSE	*1*	*0*	*-1*	*"1"*	*"0"*	*"-1"*	NULL	*array()*	*"php"*
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-1	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"1"	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"-1"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
array()	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
"php"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Деякі оператори

include "ім'я файла" – команда для включення вмісту одного файла в інший. Вміст файла, ім'я якого вказується в команді, цілком і повністю вставляється на те місце, де розташовується ця команда. Усі коди PHP, що містяться у встановленому файлі, виконуються так, ніби вони були на місці цієї команди (рис. 2.5) [21].

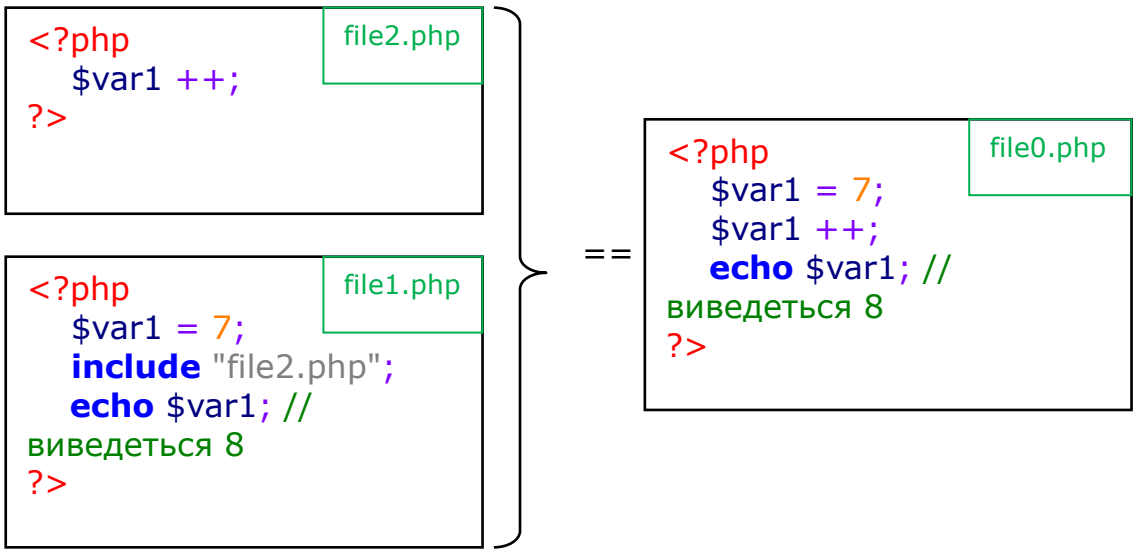


Рис. 2.5. **Оператор include**

Пам'ятайте, що *файл саме вставляється* (наприклад, шляхи до картинок, які повинні бути присутніми у файлі, що вставляється, слід вказувати від місцезнаходження того файла, в якому перебувала команда include). Якщо файл, що включається в сторінку за допомогою команди include, відсутній, то замість нього розміщується повідомлення про це, а програма на PHP виконується далі. За необхідності завершення обробки та видачі веб-сторінки в разі відсутності файла, що включається, замість команди include слід використовувати команду require.

Якщо ж необхідно, щоб потрібний файл був включений тільки один раз в рамках однієї сторінки (з урахуванням інших включень на ній), то замість include та require потрібно використовувати include_once та require_once, відповідно. Як приклад можна привести підключення до бази даних, яке обов'язково потрібне для роботи з даними, але воно повинне бути одне і те ж в рамках однієї сторінки (разом зі включеними файлами, якщо такі є на сторінці).

mail ("Кому", "Тема", " Текст повідомлення", "Додаткові заголовки") – відправка поштового повідомлення. Виконанням даної команди на сервері відповідно до вказаних параметрів електронний лист формується і вирушає за допомогою встановленої на сервері поштової програми. В якості параметра "Кому" може виступати набір адрес, розділених комами. "Додаткові заголовки" можуть бути довільні (звичайно, допустимі поштовими протоколами!), розділятися вони повинні комбінацією символів /n, яка в PHP означає переведення рядка. Якщо серед "Додаткових заголовків" не вказане поле From, то воно заповнюється за замовчуванням поштовою програмою веб-сервера, наприклад, іменем "Unprivileged User".

echo ("текст") – виведення на веб-сторінку будь-якого тексту. Щоб вивести на веб-сторінку значення будь-якої змінної, досить просто написати її ім'я всередині рядка, що виводиться: команда echo "це цифра $a" виведе в веб-сторінку текст "це цифра 1", якщо раніше змінній $a було присвоєне значення, дорівнене одиниці. У разі необхідності використовувати в виведеному рядку лапки або інші спеціальні символи перед цими символами слід ставити знак ".

if (умова) {... команди, які повинні виконуватися, якщо умова правильна...;} else {... команди, які повинні виконуватися, якщо умова неправильна...} – команда, що дозволяє виконати ту чи іншу дію

залежно від істинності чи хибності тієї чи іншої умови. У фігурних дужках може розташовуватися кілька команд, між якими ставиться крапка з комою.

for (початкове значення лічильника, умова продовження циклу, зміна лічильника на кожному циклі) {... команди... ;} – цикл, тобто повторення вказаних в ньому команд стільки разів, скільки дозволить умова зміни лічильника циклу (змінної, спеціально виділеної для підрахунку кількості виконань команд циклу).

while (умова) { ...команди... *}* – цикл з умовою. Команди в фігурних дужках виконуються до тих пір, поки виконується умова в заголовку циклу. Для того щоб цикл перервався, потрібно, щоб припинилось виконання умови. Тому всередині циклу необхідно передбачити можливість впливати на цю умову.

Цикл do {. . .команди. . . } while (умова) – працює так само, проте команди, зазначені в фігурних дужках, будуть виконані щонайменше один раз – навіть якщо умова виконуватися не буде. Перервати виконання будь-якого циклу можна оператором break – подальше виконання програми піде з команди, наступної після закриття фігурної дужки. Оператор же continue перериває поточну стадію виконання циклу, тобто після цього оператора подальше виконання програми розпочнеться з чергової перевірки умови заголовка циклу.

switch (вираз) {case значення: ... команди...; break; case інше значення: ... команди...; break;} – оператор вибору. Під час його роботи вміст, укладений в фігурні дужки, проглядається зверху вниз. Як тільки буде знайдений оператор case зі значенням, що збігається зі значенням виразу, PHP почне виконувати весь код, наступний за цим оператором case до останньої фігурної дужки оператора switch або до першого оператора break залежно від того, що з'явиться раніше. (Зверніть увагу, що, якщо команду break не вказати в кінці коду, що належить до одного варіанту значення виразу в заголовку оператора switch, PHP буде виконувати код далі – той, який належить вже наступному оператору case! Це одна з відмінностей даного оператора від аналогічних в інших мовах програмування). У кінці оператора switch можна вказати оператор default. Код, що стоїть після нього, виконається в тому випадку, якщо значення виразу в заголовку оператора не співпаде з жодним зі значень після операторів case.

foreach (змінна as масив) {... команди...;} – почергове зчитування всіх елементів масиву. Foreach зчитує у вказану в його параметрах змінну почергово всі елементи зазначеного в них же масиву, виконуючи кожного разу вказаний в фігурних дужках код, в якому може використовуватися зазначена змінна. (Значення елементів масиву цим оператором тільки зчитуються, їх модифікація за допомогою команди foreach неможлива).

Програма на PHP може перериватися кодом веб-сторінки. Для цього досить вставити закривальний тег до цього коду та відкривальний – після. Все, що знаходиться між ними, буде видаватися в браузер без будь-якої обробки, розглядаючись як виведене за допомогою команди echo (рис. 2.6).

```
<?php if ($a==1) { ?> <p>Змінна а дорівнює 1</p>
<?php> }?>
//еквівалентний коду
<?php if ($a==1) { echo "<p> Змінна а дорівнює
1</p>";}?>
```

Рис. 2.6. **Розрив/вставка PHP-коду**

Однак перший варіант менше навантажує процесор комп'ютера, на якому розташований інтерпретатор PHP. Зі сказаного також випливає, що всі програми на PHP, розташовані на одній веб-сторінці, є однією великою програмою, незважаючи на те, що вони поділяються блоками звичайного тексту сторінки. Саме тому змінна, оголошена в розташованому на початку сторінки коді, зберігає своє значення не тільки до її кінця, але й у всіх приєднуваних за допомогою команди include файлах.

Область видимості змінної

Область видимості змінної – це середовище, в якому вона визначена. У більшості випадків усі змінні PHP мають єдину зону видимості. Ця єдина область видимості охоплює також файли, що включаються (include), та необхідні (require) файли. У прикладі на рис. 2.7 змінна $a буде доступна всередині включеного скрипта b.inc.

```php
<?php
    $a = 1;
    include "b.inc";
?>
```

Рис. 2.7. **Область видимості змінних (включення файла)**

Однак усередині визначених користувачем функцій вводиться локальна область видимості функції. Будь-яка, використовувана всередині функції, змінна за замовчуванням обмежена локальною областю видимості функції. У прикладі на рис. 2.8 скрипт не згенерує ніякого виводу, оскільки вираз echo вказує на локальну версію змінної $a, а в межах цієї зони видимості їй не було присвоєно значення.

```php
<?php

    $a = 1; /* глобальна область видимості */

    function Test()
    {
        echo $a; /* посилання на змінну локальної
області видимості */
    }
    Test();

?>
```

Рис. 2.8. **Область видимості змінних (функції)**

Це дещо відрізняється від мови C тим, що глобальні змінні в C автоматично доступні функціям, якщо тільки вони не були перезаписані локальним визначенням. Така ситуація може викликати деякі проблеми, оскільки користувачі можуть ненавмисно змінити глобальну змінну. У PHP, якщо глобальна змінна буде використовуватися всередині функції, вона повинна бути оголошена глобальною всередині неї.

Ключове слово global

На рис. 2.9 зображений приклад використання global. Наведений скрипт виведе "3". Після визначення $a та $b усередині функції як global

усі посилання на будь-яку з цих змінних будуть вказувати на їх глобальну версію. Не існує ніяких обмежень на кількість глобальних змінних, які можуть оброблятися функцією.

```
<?php
     $a = 1;
     $b = 2;

     function Sum()
     {
          global $a, $b;
          $b = $a + $b;
     }
     Sum();
     echo $b;
?>
```

Рис. 2.9. **Використання ключового слова global**

Інший спосіб доступу до змінних глобальної області видимості – використання спеціального, визначеного PHP масиву $GLOBALS. Попередній приклад може бути переписаний так, як показано на рис. 2.10.

```
<?php
     $a = 1;
     $b = 2;

     function Sum()
     {
   $GLOBALS["b"] = $GLOBALS["a"] +
   $GLOBALS["b"];
     }
     Sum();
     echo $b;
?>
```

Рис. 2.10. **Використання $GLOBALS замість global**

$GLOBALS – це асоціативний масив, ключем якого є ім'я, а значення – вміст глобальної змінної. Зверніть увагу, що $GLOBALS існує в будь-якій області видимості. Це пояснюється тим, що цей масив є суперглобальним.

Масиви (Array)

Масиви – це впорядковані набори даних, які є списком однотипних елементів [4; 7; 13].

Існує два типи масивів, що розрізняються за способом ідентифікації елементів. У масивах першого типу елемент визначається індексом в послідовності. Такі масиви називаються *простими масивами*. Масиви другого типу мають асоціативну природу, і для звернення до елементів використовуються ключі, логічно пов'язані зі значеннями. Такі масиви називають *асоціативними масивами*.

Важливою особливістю PHP є те, що PHP, на відміну від інших мов, дозволяє створювати масиви будь-якої складності безпосередньо в тілі програми (скрипта).

Масиви можуть бути як одновимірними, так і багатовимірними.

Прості масиви та списки в PHP

Для звернення до елементів простих індексованих масивів використовується цілочисельний індекс, який визначає позицію заданого елемента.

Прості одномірні масиви

Узагальнений синтаксис елементів простого одновимірного масиву наведений на рис. 2.11.

```
$имя[індекс];
```

Рис. 2.11. **Узагальнений синтаксис елементів простого одновимірного масиву**

Масиви, індексами яких є числа, що починаються з нуля – це **списки** (рис. 2.12).

```php
<?php
// Простий спосіб ініціалізації масиву

$names[0]="Апельсин";
$names[1]="Банан";
$names[2]="Груша";
$names[3]="Помідор";

// Тут: names - ім'я масиву, а 0, 1, 2, 3
- індекси масиву
?>
```

Рис. 2.12. **Списки**

Доступ до елементів простих масивів (списків) здійснюється таким чином (рис. 2.13).

```php
<?php
...
// Виводимо елементи масивів в браузер:

echo $names[0]; // Вивід елемента масиву
names з індексом 0
echo "<br>";
echo $names[3]; // Вивід елемента масиву
names з індексом 3

// Виводить:
// Апельсин
// Помідор
?>
```

Рис. 2.13. **Вивід елементів простих масивів**

З технічної точки зору різниці між простими масивами та списками немає.

Прості масиви можна створювати, не вказуючи індекс нового елемента масиву (рис. 2.14), це зробить PHP.

```php
<?php
// Простий спосіб ініціалізації масиву, без
вказівки індексів

$names[]="Апельсин";
$names[]="Банан";
$names[]="Груша";
$names[]="Помідор";

// PHP автоматично присвоїть індекси елементів
масиву, починаючи з 0

// Виводимо елементи масивів в браузер:
echo $names[0]; // Вивід елемента масиву names
з індексом 0
echo "<br>";
echo $names[3]; // Вивід елемента масиву names
з індексом 3

// Виводить:
// Апельсин
// Помідор
?>
```

Рис. 2.14. **Створення простих масивів**

У розглянутому прикладі можна додавати елементи масиву names простим способом, тобто не вказуючи індекс елемента масиву (рис. 2.15).

```php
$names[]="Яблуко";
```

Рис. 2.15. **Додавання елементів в простий масив**

Новий елемент простого масиву (списку) буде доданий в кінець масиву. Надалі, з кожним новим елементом масиву, індекс буде збільшуватися на одиницю.

Прості багатовимірні масиви

Узагальнений синтаксис елементів багатовимірного простого масиву наведений на рис. 2.16.

```
$ім'я[індекс1][індекс2]..[індексN];
```

Рис. 2.16. **Узагальнений синтаксис багатовимірного простого масиву**

Приклад створення та виведення вмісту простого багатовимірного масиву поданий на рис. 2.17.

```
<?php
// Багатовимірний простий масив:
$arr[0][0]="Овочі";
$arr[0][1]="Фрукти";
$arr[1][0]="Абрикос";
$arr[1][1]="Апельсин";
$arr[1][2]="Банан";
$arr[2][0]="Огірок";
$arr[2][1]="Помідор";
$arr[2][2]="Гарбуз";
// Виводимо елементи масиву:
echo "<h3>".$arr[0][0].":</h3>";
for ($q=0; $q<=2; $q++) {
echo $arr[2][$q]."<br>";
}
echo "<h3>".$arr[0][1].":</h3>";
for ($w=0; $w<=2; $w++) {
echo $arr[1][$w]."<br>";
}
?>
```

Рис. 2.17. **Приклад роботи з простим багатовимірним масивом**

Асоціативні масиви в PHP

У PHP індексом масиву може бути не тільки число, а й рядок. Причому на такий рядок не накладається ніяких обмежень: він може містити пробіли, довжина такого рядка може бути довільною.

Асоціативні масиви особливо зручні в ситуаціях, коли елементи масиву зручніше пов'язувати зі словами, а не з числами.

Отже, масиви, індексами яких є рядки, називаються асоціативними масивами.

Одновимірні асоціативні масиви

Одновимірні асоціативні масиви містять тільки один ключ (елемент), відповідний конкретному індексу асоціативного масиву (рис. 2.18).

```
<?php
// Асоціативний масив

$names["Іванов"]="Іван";
$names["Сидоров"]=" Микола";
$names["Петров"]="Петро";

/* У даному прикладі: прізвища - ключі асоціативного масиву,
а імена – елементи масиву names */
?>
```

Рис. 2.18. **Приклад створення одновимірного асоціативного масиву**

Доступ до елементів одновимірних асоціативних масивів здійснюється так само, як і до елементів звичайних масивів, і називається доступом за ключем (рис. 2.19).

```
echo $names["Іванов"];
```

Рис. 2.19. **Доступ за ключем**

Багатовимірні асоціативні масиви

Багатовимірні асоціативні масиви можуть містити декілька ключів, які відповідають конкретному індексу асоціативного масиву. Приклад багатовимірного асоціативного масиву поданий на рис. 2.20.

Багатовимірні масиви схожі на записи мовою Pascal або структури мовою C.

```php
<?php
// Багатовимірний масив

$A["Ivanov"] = array("name"=>"Іванов І.І.",
"age"=>"25", "email"=>"ivanov@mail.ru");
$A["Petrov"] = array("name"=>"Петров П.П.",
"age"=>"34", "email"=>"petrov@mail.ru");
$A["Sidorov"] = array("name"=>"Сидоров С.С.",
"age"=>"47", "email"=>"sidorov@mail.ru");
?>
```

Рис. 2.20. **Приклад створення багатовимірного асоціативного масиву**

Доступ до елементів багатовимірного асоціативного масиву здійснюється таким чином (рис. 2.21).

```php
echo $A["Sidorov"]["name"]; // Виводить
Сидоров С.С.
echo $A["Petrov"]["email"]; // Виводить
petrov@ukr.net
```

Рис. 2.21. **Доступ до елементів багатовимірного асоціативного масиву**

2.4. Завдання до лабораторної роботи

1. Шаблон сторінок сайта, створений в попередніх лабораторних роботах, розділити на логічні блоки (модулі).
2. Виділені блоки перенести в окремі файли (рекомендовано в окремий підкаталог), які, в свою чергу, підключити до основного шаблону.
3. Для облікових елементів (меню, новини, статті тощо) створити (в окремих файлах, які потрібно підключити до відповідних модулів) масиви (різних типів) і використовувати їх в якості джерела цих облікових даних (елементів).
4. PHP-код прокоментувати.

2.5. Контрольні запитання

1. Що таке PHP?
2. Назвіть основних авторів PHP.
3. Яким чином можна звернутися до змінної в PHP?
4. Які оператори PHP ви знаєте?
5. Який оператор почергово зчитує елементи масиву?
6. Чим відрізняються оператори include та require від операторів include_once та require_once?
7. У чому особливість області видимості змінних у PHP порівняно з такими мовами, як C++ і C#?
8. Чим асоціативний масив відрізняється від звичайного масиву?

Лабораторна робота 3
Розроблення веб-додатків мовою PHP з використанням механізмів управління станом

3.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення способів обробки даних http-запитів засобами PHP і механізмів управління станом веб-додатків.

3.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити основи мови PHP для подальшого застосування під час розроблення веб-сайтів.

3.3. Загальні тези лабораторної роботи

Обробка параметрів http-запитів

Веб-програмування насамперед є обробкою різних даних, введених користувачем, тобто обробкою HTML-форм [9].

На рис. 3.1 наведений сценарій, який приймає і виводить у параметрах ім'я користувача та його вік.

```
" Привіт, <ім'я>! Вам <вік> років!"
```

Рис. 3.1. **Сценарій виводу з параметрами**

Тобто в скрипт потрібно передати два параметра: name й age.

Тепер потрібно написати скрипт script.php, що приймає два параметри: name й age, а також HTML-документ з формою, котра ці два параметри буде передавати в новий скрипт (рис. 3.2).

```php
<?php
    echo "Привіт, {$_GET['name']} ! Вам {$_GET['age']} років!";
?>
```

Рис. 3.2. **Приклад виводу рядка з http-параметрами**

Параметри URL можна використовувати, наприклад, для реалізації навігації сайта, тобто посилання на сторінки сайта будуть не на різні файли сторінок сайта, а на один PHP-файл, в якому буде аналізуватися URL-параметр(и), який ідентифікує відповідну сторінку. Такий підхід (шаблон проектування) називається Front Controller, а блок коду, який ідентифікує сторінку сайта за URL-параметрами (і можливо за HTTP-методом), називають роутингом (маршрутизацією) [9].

Тобто у блоці меню сайта в пункті "Сторінка 1" замість гіперпосилання Сторінка 1 або <a href="**/page1.php**">… буде гіперпосилання Сторінка 1 або без імені файлу, тому що це ім'я файла за замовчуванням (у веб-сервері apache): Сторінка 1.

Подібний підхід дозволяє розташувати перелік даних сторінок в окремому сховищі (зазвичай у БД) та у циклі виводити їх згідно з потрібним шаблоном. Тобто навігація на сторінках сайта буде формуватися динамічно, та у випадку зміни даних у сховищі сторінок навігація автоматично зміниться відповідно до зміни даних.

Для прикладу у якості сховища даних сторінок сайта обраний асоціативний PHP-масив (у випадку використання БД у якості подібного сховища дані на початку будуть попадати у масив, а потім з масиву використовуватися), наприклад, як наведено на рис. 3.3.

Для реалізації мінімальної маршрутизації у фронт-контролері (див. рис. 3.4) потрібно включити масив даних про сторінки сайта, та провести аналіз URL-параметра, що ідентифікує сторінку (у наведеному прикладі – "page"). У разі відсутності цього параметру потрібно відобразити користувачу (перенаправити клієнта), наприклад, головну сторінку сайту. А у разі невідповідності ідентифікатора сторінки жодній у масиві сторінок, потрібно відобразити користувачу відповідне повідомлення та повернути код стану (404 Not Found) відповідно до протоколу HTTP.

```php
<?php
$pages=array(
     'index' => array(
          'name'    => 'Головна',
          'title'   => 'Головна сторінка демонстраційного сайту
за курсом ВТВД',
          'content' => 'Це основна інформаційна частина голов-
ної сторінки сайту. <p>Вона може містить не тільки текст але й
html-теги</p>'),
     '404' => array(
          'name'    => 'Не знайдено',
          'title'   => 'Сторінку не знайдено',
          'content' => '<strong>Вибачте, але такої сторінки на
сайті не знайдено</strong>'),
     'examples' => array(
          'name'    => 'Приклади',
          'title'   => 'Приклади робот',
          'content' => '<ul><li>Проект 1</li><li>Проект
2</li><li>Проект 3</li></ul>'),
     'contacts' => array(
          'name'    => 'Контакти',
          'title'   => 'Сторінка контактів',
          'content' => '<div class="vcard"><div class="fn
org">Харківський національний економічний університет
ім. С. Кузнеця</div><div class="adr"><div class="street-
address">пр.Науки, 9а</div><div><span
class="locality">м.Харків</span>, <abbr class="region"
title="Харківська">ХА</abbr><span class="postal-
code">61001</span></div><div class="country-
name">Україна</div></div><div>Телефон: <span
class="tel">+38(057)702-03-04</span></div><div>E-mail: <span
class="email">deppost@hneu.edu.ua</span></div><div></div></div
>'),
     'sitemap' => array(
          'name'    => 'Мапа сайту',
          'title'    => 'Мапа сайту',
          'content' => '<div class="sitemap">Мапа сайту</div>',
     ),
);
```

Рис. 3.3. **Приклад масиву даних сторінок сайта**

```php
<?php
// включення масиву даних про сторінки сайта
require_once ('pages.php');
// маршрутизація (ролутинг)
if (!isset($_GET['page'])) { // якщо URL-параметр  page
відсутній
// то відкриваємо головну сторінку
    $_GET['page'] = 'index';
} // якщо значення параметра не відповідає жодній
сторінці
elseif(!isset($pages[$_GET['page']])){
    // то повернення відповідного кода стану
    header("HTTP/1.1 404 Not Found");
    // та відкриваємо сторіну з повідомленням,
    // що таку сторінку не знайдено
    $_GET['page']='404';
}
// включення HTML-шаблону, в якому виводяться дані
поточної сторінки
include 'layout.php';
?>
```

Рис. 3.4. **Приклад Front Controller (index.php) з маршрутизацією**

У прикладі всі файли розташовані у тому ж каталозі, що і фронт-контролер. Але на практиці зазвичай у проектах існує певна структура каталогів, у кожному з котрих розташовують файли за однаковим призначенням (наприклад, файли, у яких реалізують бізнес-логіку). Роботу із даними називають моделями та розташовують у каталозі з назвою (наприклад, "models"), а шаблони виводу – у каталозі "views".

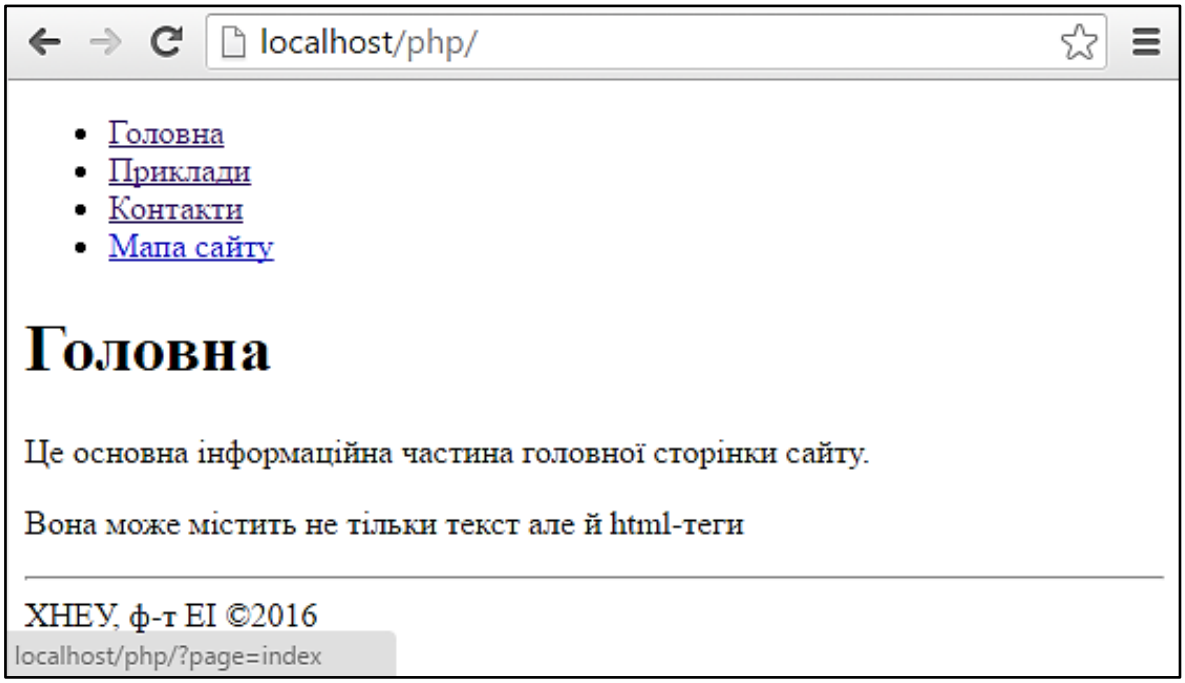
Найпростіший шаблон сторінок сайта наведений на рис. 3.5.

У даному прикладі для всіх сторінок сайта використовується один шаблон, а за необхідності використання іншого шаблону для певної сторінки або групи сторінок цей код потрібно модифікувати. Але це не потребує не багато змін: по-перше потрібно додати у сховище даних про сторінки (у прикладі – масив) ще один елемент (наприклад, "layout"), у якому буде знаходитися ідентифікатор шаблону; по-друге, у місці включення шаблону (include 'layout.php';) написати перевірку наявності у поточної сторінки параметра "layout" й у разі його присутності включити відповідний шаблон замість того, що використовується за замовчуванням.

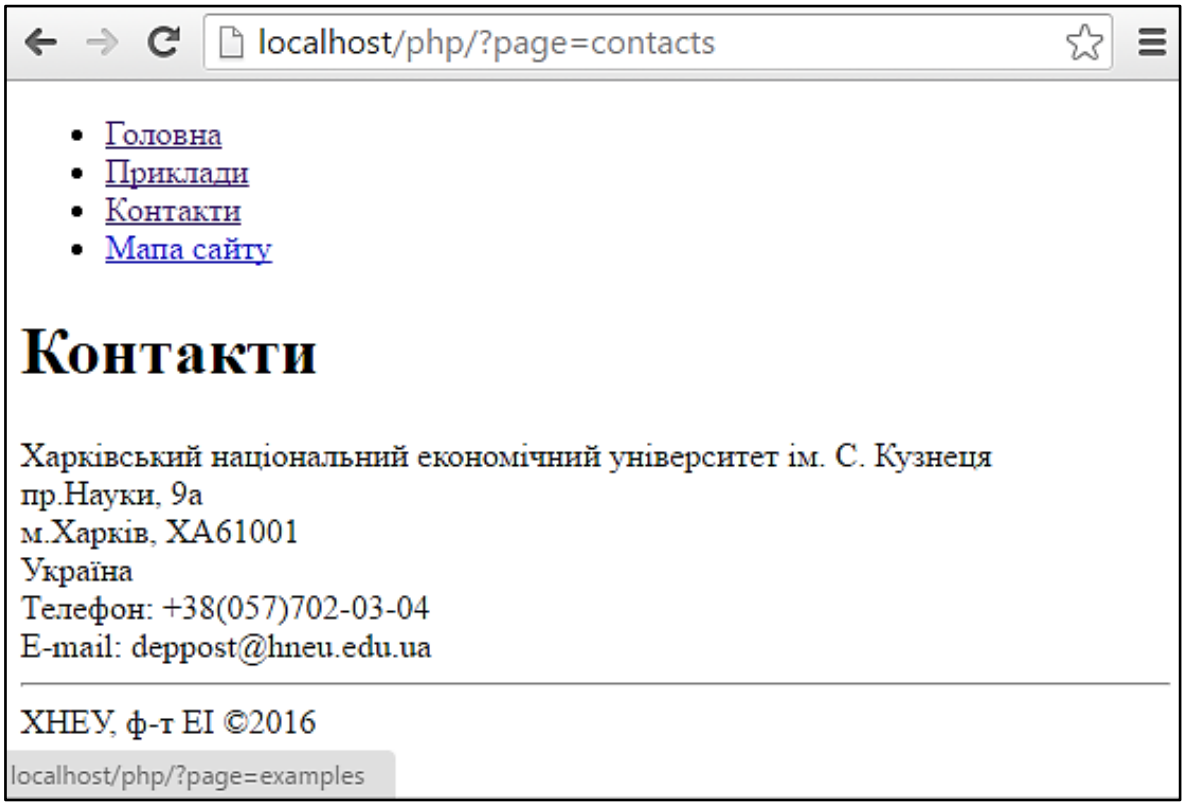
```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
        <title><?php echo
$pages[$_GET['page']]['title']; ?> | Демонстраційний
сайт з ВТВД</title>
    </head>
    <body>
        <div id="wrapper">
            <div id="header">
                <ul class="nav">
                    <?php foreach ($pages as $href =>
$page) { ?>
                        <?php if ($href != '404') { ?>
                            <li><a href="?page=<?php
echo $href; ?>">
                                    <?php echo
$page['name']; ?>
                                </a></li>
                        <?php } ?>
                    <?php } ?>
                </ul>
            </div>
            <div id="middle">
                <h1><?php echo
$pages[$_GET['page']]['name'];?></h1>
                <div class="content"><?php echo
$pages[$_GET['page']]['content']; ?></div>
            </div>
            <div id="footer">
                 <hr>ХНЕУ, ф-т ЕІ &copy;2016
             </div>
        </div>
    </body>
</html>
```

Рис. 3.5. **Приклад шаблону сторінок сайта (layout.php)**

Результат роботи демонстраційного сайта наведений на рис. 3.6.



а) приклад головної сторінки (без URL-параметра "page")



б) приклад сторінки "Контакти" (з URL-параметром "page")

Рис. 3.6. **Приклади роботи демонстраційного сайта**

На рис. 3.7 наведений код HTML-документа send.html, за допомогою якого параметри name й age передаються скрипту script.php.

```
<html>
    <body>
        <form action="script.php">
            Введіть ім'я: <input type=text
name="name"><br>
            Введіть вік: <input type=text
name="age"><br>
            <input type=submit value="GO!">
        </form>
    </body>
</html>
```

Рис. 3.7. **Приклад html-форми**

Зверніть увагу на адресний рядок браузера після передачі параметрів сценарію, вона буде виглядати приблизно так (без URL-кодування кирилиці) (рис. 3.8).

```
http://localhost/script.php?name=Саша&age=23
```

Рис. 3.8. **Приклад URL http-запиту методом GET**

Причиною додавання браузером параметрів з форми в URL є метод передачі даних за http, а саме – методом GET, використовуваний за замовчуванням. А для використання методу POST його потрібно явно вказати в атрибуті форми method (рис. 3.9).

```
<form action="script.php" method="POST">
```

Рис. 3.9. **Установка атрибута html-форми method для відправки даних методом POST**

Cookie

Cookie – це механізм зберігання даних браузером віддаленого комп'ютера для ідентифікації відвідувачів, що повертаються, та зберігання параметрів веб-сторінок (наприклад, змінних) [10; 19; 20].

Використання Cookie слід розглянути на конкретному прикладі.

Припустимо, що потрібно написати лічильник відвідування сайта. Для цього треба знати, яка кількість відвідувань сайта здійснювалась кожним конкретним відвідувачем.

Дану задачу можна вирішити двома способами. Перший з них полягає у веденні обліку IP-адрес користувачів. Для цього потрібна база даних всього з однієї таблиці, приблизна структура якої приведена в табл. 3.1.

Таблиця 3.1

Таблиця даних обліку відвідувань сайта користувачами

IP-адреса	Кількість відвідувань
210.124.134.203	7
212.201.78.207	14
83.103.203.73	3

Коли користувач заходить на сайт, потрібно визначити його IP-адресу, знайти в базі даних інформацію про його відвідування, збільшити лічильник і вивести його в браузер відвідувача. Написати обробник (скрипт) подібної процедури нескладно. Однак з використанням такого методу з'являються проблеми такого характеру:

- для кожної IP-адреси потрібно вести облік в одній таблиці, котра може бути дуже великою. А з цього випливає, що нераціонально використовуваний процесорний час і дисковий простір;
- у більшості домашніх користувачів IP-адреси є динамічними. Тобто сьогодні у нього адреса 212.218.78.124, а завтра – 212.218.78.137. Таким чином, велика ймовірність повторного ідентифікування одного користувача кілька разів.

Можна використовувати інший спосіб, набагато легший в реалізації та більш ефективний. Можна встановити в Cookie змінну, яка буде зберігатися на диску віддаленого користувача. Ця змінна і зберігатиме інформацію про відвідування. Вона буде зчитуватися скриптом у разі звернення відвідувача до сервера. Вигода такого методу ідентифікації очевидна: не треба зберігати безліч "непотрібної" інформації про IP-адреси; нас

не цікавлять динамічні IP-адреси, оскільки дані про свої відвідування зберігаються конкретно у кожного відвідувача сайта.

Тепер зрозуміло, для чого можна використовувати Cookie – для зберігання невеликої за обсягом інформації у клієнта (браузера) користувача сайта, наприклад: налаштування сайта (колір фону сторінок, мова, оформлення таблиць і т.д.), а також іншої інформації.

Файли Cookie – це звичайні текстові файли, які зберігаються на диску у відвідувачів сайтів. Файли Cookie і містять ту інформацію, котра була в них записана сервером (або клієнтом).

Програмування Cookie

Для установки Cookie використовують функцію SetCookie(). Для цієї функції можна вказати шість параметрів, один з яких є обов'язковим [10]:

name – задає ім'я (рядків), закріплене за Cookie;

value – визначає значення змінної (рядок);

expire – час "життя" змінної (ціле число). Якщо даний параметр не вказати, то Cookie будуть "жити" до кінця сесії, тобто до закриття браузера. Якщо час вказаний, то, коли він настане, Cookie само знищиться;

path – шлях до Cookie (рядок);

domain – домен (рядок). Як значення встановлюється ім'я хоста, з якого Cookie було встановлене;

secure – передача Cookie через захищене HTTPS-з'єднання.

Зазвичай використовуються тільки три перші параметра.

Приклад установки Cookie наведений на рис. 3.10.

```
<?php
// Встановлюємо Cookie до кінця сесії:
SetCookie("Test","Value");
// Встановлюємо Cookie на одну годину:
SetCookie("My_Cookie","Value",time()+3600);
?>
```

Рис. 3.10. **Установлення Cookie**

Під час використання Cookie необхідно мати на увазі, що **Cookie повинні встановлюватися до першого виводу інформації** в браузер (наприклад, оператором echo або виводом якої-небудь функції). Тому бажано встановлювати Cookie на самому початку скрипта. *Cookie встановлюються*

за допомогою певного http-заголовка сервера, а якщо скрипт виводить що-небудь, то це означає, що починається тіло документа (http-тіло). У результаті Cookie не будуть встановлені, і може бути виведене попередження. Для перевірки успішності установки Cookie можна використовувати метод, який наведений на рис. 3.11.

```
<?php

// Встановлюємо Cookie до кінця сесії:

// У разі успішного встановлення Cookie, функція
SetCookie повертає TRUE:

if (SetCookie("Test","Value"))
     echo "<h3>Cookie успішно встановлені!</h3>";

?>
```

Рис. 3.11. **Установка Cookie з перевіркою на успішність**

Функція SetCookie() повертає TRUE в разі успішного встановлення Cookie. У разі якщо Cookie встановити не вдається, SetCookie() поверне FALSE і, можливо, попередження (залежить від налаштувань PHP).

Читання значень Cookies

Отримати доступ до Cookies та їх значень досить просто. Вони зберігаються в суперглобальному масиві $_COOKIE, тобто з передачею в http-запиті будь-яких Cookie PHP-інтерпретатор поміщає їх в суперглобальний масив $_COOKIE.

Доступ до значень здійснюється за іменем встановлених Cookies, (рис. 3.12).

```
<?php
echo $_COOKIE['my_cookie'];
// Виводить значення встановленої Cookie 'My_Cookie'
?>
```

Рис. 3.12. **Доступ до значень Cookie в PHP**

Під час читання значень Cookies звертайте увагу на перевірку існування Cookies, наприклад, використовуючи оператор **isset**(), або шляхом зменшення виводу помилок оператором @.

Приклад реалізації лічильника кількості завантажень сторінки за допомогою Cookies наведений на рис. 3.13.

```
<?php

// Перевіряємо, чи було вже встановлено Cookie 'Counter',
// Якщо так, то читаємо його значення,
// І збільшуємо значення лічильника звернень на сторінку:
if (isset($_COOKIE['Counter']))
$cnt=$_COOKIE['Counter']+1;
else $cnt=0;
// Установлюємо Cookie 'Counter' зі значенням лічильника,
// З часом "життя" до 18/07/29,
// Тобто на дуже довгий час:
setcookie("Counter",$cnt,0x6FFFFFFF);
// Виводить кількість відвідувань (завантажень) цієї
сторінки:
echo "<p>Ви відвідували цю сторінку
<b>".@$_COOKIE['Counter']."</b> раз</p>";

?>
```

Рис. 3.13. **Реалізація лічильника кількості завантажень сторінки за допомогою Cookies**

Видалення Cookies

Іноді виникає необхідність видалення Cookies. Зробити це нескладно, необхідно лише знову встановити Cookie з ідентичним ім'ям та/або з терміном життя, який вже пройшов, або з порожнім значенням (рис. 3.14) [10].

```
<?php
// Видаляємо Cookie 'Test':
SetCookie("Test","");
?>
```

Рис. 3.14. **Видалення Cookies**

Переваги використання Cookies незаперечні. Однак існують і деякі проблеми їх використання. Перша з них полягає в тому, що відвідувач може блокувати прийом Cookies браузером або просто видалити всі Cookies або їх частину.

Сесії в PHP

Веб-сервер не підтримує постійного з'єднання з клієнтом, і кожен запит обробляється як новий, без зв'язку з попередніми. Тобто неможливо ні відстежити запити від одного і того ж самого відвідувача, ні зберегти для нього змінні між переглядами окремих сторінок. Тому для вирішення цих двох завдань були винайдені сесії [19; 20].

Сесії – це механізм, що дозволяє однозначно ідентифікувати браузер і створює для цього браузера файл на сервері, в якому зберігаються змінні сеансу.

Як влаштовані й як працюють сесії?

Передусім необхідно ідентифікувати браузер. Сесії використовують стандартні способи передачі даних. Ідентифікатор – це звичайна змінна. За замовчуванням її ім'я – PHPSESSID.

Завдання PHP – відправити її браузеру, щоб той повернув її з наступним запитом. Змінну можна передати тільки двома способами: в Cookies або POST/GET запитом. PHP використовує обидва варіанти. За це відповідають два налаштування в php.ini (див. рис. 3.15).

```
session.use_cookies - якщо дорівнює 1, то PHP
передає ідентифікатор в cookies, якщо 0 - то ні.

session.use_trans_sid якщо дорівнює 1, то PHP
передає його, додаючи до URL і формам, якщо 0 -
то ні.
```

Рис. 3.15. **Налаштування способів ідентифікації сесій в PHP**

Міняти ці й інші параметри сесій можна так само, як і інші налаштування PHP – у файлі php.ini, а також за допомогою команди ini_set() або в файлах налаштування веб-сервера.

Якщо включена тільки перша, то зі стартом сесії – за кожним викликом session_start() – клієнту встановлюється cookie. Браузер справно, з кожним наступним запитом цей cookie повертає, і PHP має ідентифікатор сесії. Проблеми починаються, якщо браузер cookies не повертає. У цьому випадку, не отримуючи cookies з ідентифікатором, PHP буде весь час стартувати нову сесію, і механізм працювати не буде.

Якщо включена тільки друга, то cookies не виставляються. А відбувається те, заради чого, власне, і варто використовувати вбудований механізм сесій. Після того, як скрипт виконує свою роботу і сторінка повністю сформована, PHP переглядає її всю і дописує до кожного посилання та до кожної форми – передачу ідентифікатора сесії.

Старт (ініціалізація) сесії PHP

Старт (ініціалізація) сесії в PHP здійснюється функцією **session_start()**. Для того щоб мати доступ до змінних сесії на будь-яких сторінках сайта, потрібно викликати цю функцію на самому початку к**ожного** файла [19; 20].

PHP визначає, яку саме сесію потрібно запустити, на ім'я cookie, переданому браузером в заголовку запиту. Браузер ж, в свою чергу, отримує цей cookie від сервера, куди поміщає його функція session_start(). Якщо час життя cookie в браузері минув, він не буде переданий в запиті. Тому PHP не зможе визначити, яку сесію потрібно запустити, і розцінить це як створення нової сесії. Параметр налаштувань PHP session.gc_maxlifetime, який встановлюється дорівненим нашому тайм-ауту відсутності активності користувача, задає час життя PHP-сесії і контролюється сервером. Працює контроль часу життя сесії таким чином (тут розглядається приклад сховища сесій у тимчасових файлах як найпоширеніший і повернеться до типової в PHP варіант).

У момент створення нової сесії в каталозі, встановленому як каталог для зберігання сесій в параметрі налаштувань PHP session.save_path, створюється файл з ім'ям sess_ <sessionid>, де <sessionid> – ідентифікатор сесії. Далі в кожному запиті, в момент запуску вже існуючої сесії, PHP оновлює час модифікації цього файла. Таким чином, в кожному наступному запиті PHP (шляхом різниці між поточним часом і часом останньої модифікації файла сесії) може визначити, чи є сесія активною, або її час життя вже минув.

Слід зазначити, що параметр session.gc_maxlifetime діє на всі сесії в межах одного сервера (точніше – в межах одного головного процесу

PHP). На практиці це означає, що якщо на сервері працює кілька сайтів і кожен з них має власний тайм-аут відсутності активності користувачів, то установка цього параметра на одному з сайтів призведе до його установки і для інших сайтів. Те ж стосується і shared-хостингу. Для уникнення подібної ситуації використовуються окремі каталоги сесій для кожного сайту в межах одного сервера. Установка шляху до каталогу сесій проводиться за допомогою параметра session.save_path у файлі налаштувань php.ini або шляхом виклику функції ini_set (). Після цього сесії кожного сайта будуть зберігатися в окремих каталогах, і параметр session.gc_maxlifetime, встановлений на одному з сайтів, буде діяти тільки на сесії цього сайта.

Ініціалізація та доступ до змінних сесії

Змінні сесії в PHP зберігаються (до них здійснюється доступ) у суперглобальному масиві $_SESSION. Для запису змінної в сесію досить ініціалізувати елемент масиву $_SESSION з відповідним назві змінної ключем (див. рис. 3.16).

```php
<?php

// Ініціалізації сесії в PHP
session_start();

// Установка змінної 'test' в сесії
$_SESSION['test']='Hello world!';

// Вивід значення змінної сесії 'test'
echo $_SESSION['test'];
?>
```

Рис. 3.16. **Установка та читання змінних в сесії PHP**

Прибиранням сміття – видаленням застарілих файлів PHP теж займається сам, як і кодуванням даних і безліччю інших завдань. У результаті цього робота з сесіями виявляється дуже простою. Наприклад, перевірка авторизації може виглядати так (рис. 3.17).

```php
<?php

session_start();

if ($_SESSION['authorized']<>1) {
header("Location: /auth.php");
exit;
}
?>
```

Рис. 3.17. **Перевірка авторизації в PHP**

Видалення змінних з сесії

Видалення змінних з сесії здійснюється як видалення звичайних змінних за допомогою функції **unset**() (див. рис. 3.18).

```php
<?php
session_start();
unset($_SESSION['var']);
?>
```

Рис. 3.18. **Видалення змінних з сесії**

А що станеться, якщо браузер був закритий, і cookies з ім'ям сесії був автоматично знищений? Запит до сервера під час наступного відкривання браузера не буде містити cookies сесії, і сервер не зможе відкрити сесію, щоб переглянути тайм-аут відсутності активності користувача. Для нас це рівносильно створенню нової сесії і ніяк не впливає на функціонал і безпеку. Але виникає справедливе запитання: а хто ж тоді знищить стару сесію, якщо до сих пір її знищували ми після закінчення тайм-ауту? Або вона тепер буде висіти в каталозі сесій вічно? Для очищення старих сесій в PHP існує механізм під назвою garbage collection. Він запускається в момент чергового запиту до сервера й очищує всі старі сесії на підставі дати останньої зміни файлів сесій. Але запуск механізму garbage collection відбувається не за кожним запитом до сервера. Частота (а точніше – ймовірність) запуску визначається двома параметрами налаштувань: session.gc_probability і session.gc_divisor. Результат від ділення першого параметра на другий

і є ймовірністю запуску механізму garbage collection. Таким чином, для того щоб механізм очищення сесій запускався за кожним запитом до сервера, ці параметри потрібно встановити в дорівнені значення, наприклад "1". Такий підхід гарантує чистоту каталогу сесій, але є занадто складним для сервера. Тому в production-системах за замовчуванням встановлюється значення session.gc_divisor, яке дорівнює 1 000. Це означає, що механізм garbage collection буде запускатися з ймовірністю 1/1000. Якщо ви проекспериментуєте з цим в своєму файлі php.ini, то зможете помітити, що в описаному випадку, коли браузер закривається і очищає всі свої cookies, в каталозі сесій якийсь час все ще залишаються старі сесії [12].

Запобігання "зависання" скриптів через блокування файлу сесії

Ця проблема залежить від завантаженості сервера або кількості користувачів. Звичайно, чим більше запитів, тим повільніше виконуються скрипти. Але це непряма залежність. Проблема з'являється тільки в межах однієї сесії, коли до сервера приходить кілька запитів від імені одного користувача (наприклад, один з них long poll, а решта – звичайні запити). Кожен запит намагається отримати доступ до одного і того ж файла сесії, і якщо попередньо не розблокувати файл, то наступний буде висіти в очікуванні.

Для зведення блокування файлів сесій до мінімуму настійно рекомендується закривати сесію шляхом виклику функції session_write_close відразу після того, як виконані всі дії з сесійними змінними. На практиці це означає, що не слід зберігати в сесійних змінних все взагалі та звертатися до них протягом часу виконання скрипта. А якщо треба зберігати в сесійних змінних якісь робочі дані, то зчитувати їх відразу після старту сесії, зберігати в локальні змінні для подальшого використання та закривати сесію (мається на увазі закриття сесії за допомогою функції session_write_close, а не знищення за допомогою session_destroy) (рис. 3.19).

У даному прикладі це означає, що відразу після відкриття сесії, перевірки часу її життя й існування авторизованого користувача необхідно вважати та зберегти всі додаткові необхідні сесійні змінні (якщо такі існують), після чого закрити сесію за допомогою виклику session_write_close () і продовжити виконання скрипта, будь то long poll або звичайний запит [12; 25].

Захист від атаки "Impersonation" (Видача Себе за Іншого)

Поширене помилкове уявлення, що вбудований у PHP механізм управління сесією сам вживає заходів безпеки проти атак, заснованих на сесії.

Навпаки, PHP тільки надає відповідний для цього механізм. Відповідальність же за забезпечення заходів безпеки лягає на розробника. Не існує ідеального способу вирішення, як і найкращого рішення, правильного для кожного [25].

```php
<?php
session_start();
//Робота з сесією
$user=$_SESSION['username']; //приклад
if ($user=="") { //приклад
   session_write_close();
  /*
   "важкий" код без потреби роботи із сесією
     ...
   $user=....;
  */
  //повторне відкриття сесії
   ini_set('session.use_only_cookies', false);
   ini_set('session.use_cookies', false);
   ini_set('session.use_trans_sid', false);
   ini_set('session.cache_limiter', null);
   session_start();
}
//відновлення роботи із сесією
$_SESSION['username']=$user; //приклад
```

Рис. 3.19. **Приклад запобігання "зависанню" скриптів через блокування файла сесії**

Для пояснення небезпеки BCI слід розглянути таку послідовність подій (рис. 3.20):

1) Користувач заходить на http://www.example.org/ і авторизується;
2) Веб-сайт встановлює cookies PHPSESSID = 12345;
3) Зловмисник заходить на http://www.example.org/ і надає cookies PHPSESSID = 12345;
4) веб-сайт помилково думає, що Зловмисник насправді є Користувач.

Звичайно, цей сценарій вимагає, щоб Зловмисник якимось чином дізнався або вгадав правильний PHPSESSID, що належить Користувачу. Хоч це і здається малоефективним, проте прикладом забезпечення безпеки за допомогою НДС (obscurity) – Невідомості Для Стороннього: він не є чимось, на що варто покладатися. Така невідомість – річ, звичайно, непогана, і вона може допомагати, але необхідно щось більш суттєве, що запропонує надійний захист проти подібної атаки.

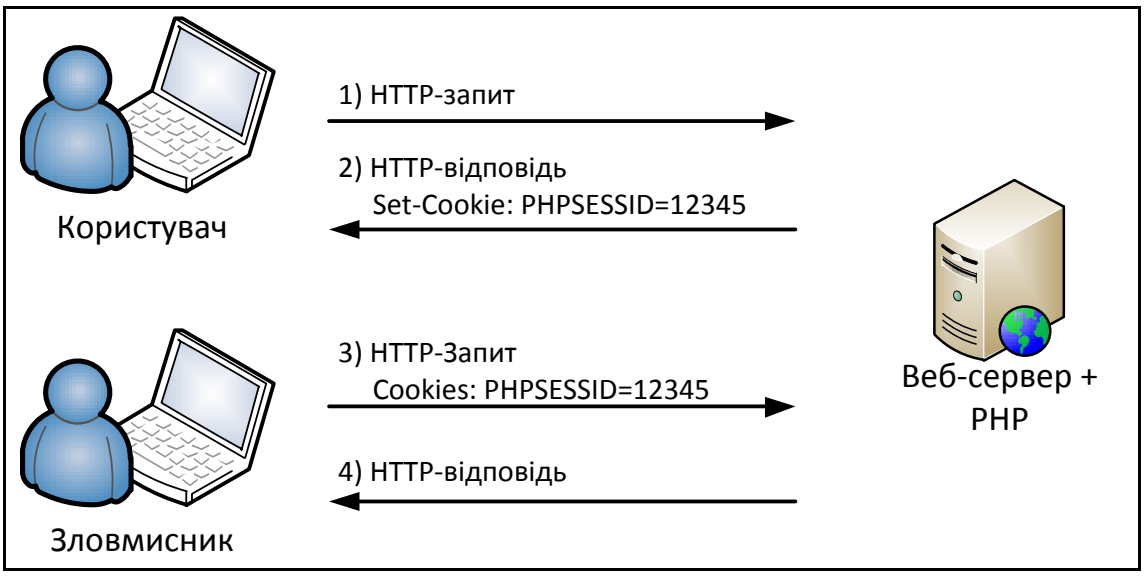


Рис. 3.20. **Сценарій "Видача Себе за Іншого"**

Існує багато способів, які можуть бути використані для утруднення BCI або інших заснованих на сесії атак. Загальним підходом тут є створення систем настільки, наскільки це можливо, зручними для ваших законних користувачів і, настільки, наскільки можливо, – складними та заплутаними для атакуючих. Це може виявитися важкодосяжним балансом, і в значній мірі ідеальний баланс залежить від дизайну програми. Тому, в кінцевому рахунку, кращий суддя – це ви самі.

Найпростіший коректний з точки зору HTTP/1.1 запит, як відомо, складається з рядка-запиту і заголовка Host:

```
GET / HTTP/1.1
Host: www.example.org
Cookie: PHPSESSID=12345
User-Agent: Mozilla/5.0 Galeon/1.2.6 (X11; Linux i686; U;)
Gecko/20020916
Accept: text/html;q=0.9, */*;q=0.1
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Accept-Language: en
```

Цей запит містить чотири необов'язкових заголовка: User-Agent, Accept, Accept-Charset і Accept-Language. Через те, що ці заголовки необов'язкові, буде не дуже розумно сподіватися на їх наявність. Однак якщо браузер користувача все ж послав ці заголовки, чи безпечно вважати, що вони

будуть присутні і в повторних запитах від того ж браузера? Відповіддю є "так" з невеликим винятком. Припускаючи, що попередній приклад – це запит, надісланий користувачем з активною сесією, слід розглянути наступний запит, надісланий незадовго після цього:

```
GET / HTTP/1.1
Host: www.example.org
Cookie: PHPSESSID=12345
User-Agent: Mozilla/5.0 (compatible; IE 6.0 Microsoft Windows XP)
```

Оскільки наданий той же унікальний ідентифікатор, дістане доступ до тієї ж PHP-сесії. Якщо браузер ідентифікує себе інакше, ніж в попередній взаємодії, чи треба думати, що це той самий користувач?

Звісно це небажано. Проте це саме те, що станеться, якщо ви пишете код, який не перевіряє спеціально такі ситуації. Навіть у випадках, коли ви не можете бути впевнені, що запит є BCI-атакою, проста вимога введення користувачем пароля може допомогти запобігти BCI без заподіяння незручності вашим користувачам.

Ви можете додати перевірку браузера користувача в вашу модель безпеки за допомогою коду, подібного до поданого на рис. 3.21.

```
<?php
    session_start();
    if ( sha1($_SERVER['HTTP_USER_AGENT']) !=
$_SESSION['HTTP_USER_AGENT'] ) {
        exit;
    }
    /* Код далі */
?>
```

Рис. 3.21. **Приклад перевірки клієнта**

Звичайно, перед цим необхідно зберігати SHA1-хеш від повної назви браузера користувача щоразу, коли вперше починаєте сесію, як показано на рис. 3.22.

```php
<?php
    session_start();
    if(!isset($_SESSION['HTTP_USER_AGENT'])){
        $_SESSION['HTTP_USER_AGENT'] =
sha1($_SERVER['HTTP_USER_AGENT']);
    } elseif ( sha1($_SERVER['HTTP_USER_AGENT']) !=
$_SESSION['HTTP_USER_AGENT'] ) {
        exit;
    }
?>
```

Рис. 3.22. **Приклад ініціалізації ідентифікатора клієнта**

Хоча і не обов'язково використовувати SHA1-хеш замість повної назви браузера користувача, це допоможе забезпечити певну сталість і виключити необхідність перевірки правильності значення $_SERVER['HTTP_USER_AGENT'] перед збереженням його в сесії. Оскільки ці дані беруться від клієнта, довіряти їм наосліп не можна, але формат SHA1-хеша незалежний від вхідних даних. Тепер, коли додана перевірка браузера користувача, атакуючий повинен здійснити два кроки для того, щоб підробити сесію:

- отримати правильний ідентифікатор сесії;
- надати для BCI-атаки заголовок User-Agent.

Це, безсумнівно, можливо, проте складніше, ніж, якби другий крок був пропущений. Таким чином, безпека сесійного механізму дещо посилена.

Інші заголовки можуть бути додані таким же шляхом, і ви можете навіть використовувати в якості "відбитків пальців" комбінацію заголовків. Якщо ви допишете якийсь додатковий секретний префікс, то такі "відбитки" практично неможливо вгадати (рис. 3.23).

```php
<?php
    session_start();
    $fingerprint = 'SECRETSTUFF' .
$_SERVER['HTTP_USER_AGENT'] .
$_SERVER['HTTP_ACCEPT_CHARSET'];
    $_SESSION['fingerprint'] = sha1($fingerprint .
session_id());
?>
```

Рис. 3.23. **Приклад захисту від BCI**

Заголовок Accept не повинен використовуватися в "відбитки пальців", оскільки Microsoft'овський Internet Explorer відомий розбіжністю в значеннях цього заголовка у випадках, коли користувач оновлює сторінку та клацає по посиланню.

З важковгадуваними "відбитками пальців" певний виграш досягається і без ускладнення цієї інформації іншим, ніж демонструвалося досі, способом. З існуючим механізмом для ВСД потрібні ті ж два кроки, хоча другий крок тепер набагато складніший, тому що атакуючий повинен відтворити множинні заголовки.

Для збільшення безпеки необхідно почати включати дані на додаток до унікальних ідентифікаторів. Розглянемо механізм управління сесією, яким унікальний ідентифікатор передається в GET-даних. Якщо "відбитки пальців", згенеровані в попередньому прикладі, також передаються як GET-дані, атакуючий повинен здійснити наступні три кроки для успішної підробки сесії:

- отримати правильний ідентифікатор сесії;
- надати ті ж HTTP-заголовки;
- надати правильні "відбитки пальців".

Якщо і унікальний ідентифікатор, і "відбитки пальців" передаються як GET-дані, то можливо, що атакуючий, який зможе отримати щось одне з них, також отримає доступ і до іншого. Більш безпечним підходом є використання двох різних методів передачі: GET-дані та cookies. Звичайно, це залежить від налаштувань, але для тих, хто дозволив cookies, може бути забезпечений додатковий рівень захисту. Таким чином, якщо атакуючий отримає унікальний ідентифікатор через уразливість браузера, "відбитки пальців", ймовірно, все ще будуть йому невідомі.

Є багато інших способів, які можуть бути використані, щоб збільшити безпеку вашого механізму управління сесією. Будемо сподіватися, ви досягнете успіху на шляху створення деяких власних технік. Адже саме ви є експертом ваших власних додатків, тому, озброєні хорошим розумінням сесій, ви – найкраща особа для реалізації додаткової безпеки.

Невідомість для стороннього (НДП)

Існує думка, що невідомість для стороннього не пропонує адекватний захист, і на не варто покладатися. Навпаки, вже маючи в якості основи безпечний механізм управління сесією, НДП може запропонувати певну ступінь додаткової надійності.

Допомогти може просте використання введення в оману імен змінних для унікального ідентифікатора і "відбитків пальців". Ви можете також передавати неправдиві дані, для того щоб збити з шляху потенційного атакуючого і ввести його в оману. На ці техніки для захисту, безумовно, покладатися ніколи не варто, але ви не дарма витратите час, якщо організуєте НДП у вашому власному механізмі.

3.4. Завдання до лабораторної роботи

1. У шаблон сторінок сайта, створений в попередній лабораторній роботі, додати модуль авторизації користувачів на сайті (дані користувачів зберігати в масиві в окремому файлі, підключеному до даного модулю).
2. Реалізувати модуль меню через параметри http-запитів (ідентифікатори пунктів меню зберігати в багатовимірному масиві, а контент сторінок у відповідних цим ідентифікаторам файлах).
3. Реалізувати модуль (розташувати або в "футері", або в нижній частині бічної панелі сайта) обліку відвідуваності користувачів сайта й окремих сторінок сайта за допомогою Cookie.

3.5. Контрольні запитання

1. Назвіть джерела http-запитів з параметрами.
2. Від чого залежить спосіб передачі параметрів у http-запиті?
3. Яким чином можна отримати доступ до параметрів http-запиту в PHP-обробнику?
4. Яке призначення тега FORM?
5. Назвіть елементи управління й їх html-реалізацію.
6. Що таке Cookie?
7. Назвіть параметри Cookie.
8. Назвіть способи установки Cookie.
9. Які ви знаєте суперглобальні масиви в PHP?
10. Для чого застосовують масив $_SESSION?

Лабораторна робота 4
Розроблення веб-додатків з використанням ООП у PHP

4.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення об'єктно-орієнтованого механізму взаємодії PHP-скриптів.

4.2. Рекомендації щодо підготовки до виконання лабораторної роботи

У процесі підготовки до лабораторної роботи необхідно вивчити об'єктно-орієнтований механізм взаємодії PHP-скриптів.

4.3. Загальні тези лабораторної роботи

Об'єктно-орієнтований підхід є загальною тенденцією в сучасному програмуванні. Мова PHP не є винятком. Крім того, робота з БД є невід'ємною та дуже важливою частиною сучасних веб-додатків і веб-проектів. Тому доцільно розглянути механізм роботи з БД в PHP, який є повністю об'єктно-орієнтованим. Цей механізм називається PDO [5; 8; 28].

PHP Data Objects (PDO) – розширення для PHP, що надає розробнику простий і універсальний інтерфейс для доступу до різних баз даних.

PDO пропонує єдині методи для роботи з різними базами даних, хоча текст запитів може дещо відрізнятися. Оскільки багато СУБД реалізують свій діалект SQL, який в тій чи іншій мірі підтримує стандарти ANSI й ISO, то використанням простих запитів можна досягти сумісності між різними мовами. На практиці це означає, що можна досить легко перейти на іншу СУБД, не змінюючи або частково змінюючи код програми.

PDO не використовує абстрактних шарів для підключення до БД, на зразок ODBC, а використовує для різних БД їх "рідні" драйвери, що дозволяє добитися високої продуктивності. Сьогодні для PDO існують драйвери практично до всіх загальновідомих СУБД та інтерфейсів (табл. 4.1). Утім, є і драйвер для підключення до ODBC.

Таблиця 4.1

Список драйверів СУБД в складі PDO

Ім'я драйвера	Підтримувані СУБД
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird/Interbase 6
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC и win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 3 и SQLite 2
PDO_4D	4D

PDO входить до складу PHP, починаючи з версії 4.1.

Конструювання об'єкта PDO

Під час створення об'єкта PDO, що є базою даних або з'єднанням, конструктору як параметр передається ім'я джерела даних (DSN – Data Source Name). До складу DSN входить ім'я драйвера, потім – двокрапка, а потім – рядок, синтаксис якого залежить від конкретного драйвера. Приклад створення з'єднання з базою даних MySQL наведений на рис. 4.1.

Об'єкт PDO конструюється майже так само, як встановлюється з'єднання з сервером БД MySQL. Основною відмінністю є те, що спочатку ще потрібно вказати ім'я драйвера БД і двокрапку [8; 22].

```php
<?php

//--- конфігураційні параметри ---
$db_drivername = "mysql";
$hostname      = "localhost";
$dbname        = "weblabdb";
$username      = "root";
$password      = "";

//--- Створення PDO для з'єднання з сервером БД
$pdo = new PDO("{$db_drivername}:host={$hostname};
dbname={$dbname}", $username, $password);

//--- 1 параметр PDO: "mysql:host=localhost;dbname=weblabdb"
//--- 2 параметр PDO: "root"
//--- 3 параметр PDO: ""
```

Рис. 4.1. **Створення об'єкта PDO**

Виконання запитів до БД

Для виконання SQL-запитів до БД в PDO існує два способи [5; 8; 28]:

1. Виклик методу об'єкта PDO-> query (), якому в якості параметра передається SQL-запит у вигляді рядка (див. рис. 4.2), а повертає об'єкт класу PDOStatement. PDOStatement – це один з класів PDO, який схожий з результівним набором даних, але є об'єктом і надає більший функціонал у вигляді своїх методів;

2. Виклик методу *prepare()* об'єкта класу PDO. Метод *prepare()* готує SQL-запит, який отримують в якості строкового параметра, і повертає об'єкт класу PDOSatement. Для виконання підготовленого запиту потрібно викликати метод *execute()* повернутого об'єкта класу PDOSatement (див. рис. 4.3).

Якщо в підготовленому запиті присутні параметри, то в метод execute() в якості параметра необхідно передати масив елементів, які будуть передані в запит.

```
<?php $pdo->query("SET CHARACTER SET utf8"); ?>
...
<div id="menu">
    <?php
    $query = 'SELECT * FROM page';
    $q_res = $pdo->query($query);
    if ($q_res)
      foreach ($q_res as $row_menu) { ?>
       <a href="index.php?page=<?php echo
$row_menu["id"]; ?>">
        <?php echo $row_menu["menu_item_name"]; ?> </a>
    <?php } ?>
</div>
```

Рис. 4.2. **Приклад виконання запитів до БД за допомогою методу PDO->query()**

У прикладі на рис. 4.3 поданий параметризований запит зі зазначенням параметра за допомогою знака "**?**". Значення підставляються з простого масиву, який передається в якості параметра в метод *execute()*. Необхідність цього стає очевидною в разі, коли в запиті більше одного параметра, а значення підставляються в нього з масиву в порядку проходження елементів цього масиву.

```php
<?php
   $query_getPage = "SELECT * FROM page WHERE id =?";
   $stmt = $pdo->prepare($query_getPage);
   if (isset($_GET["page"])) { $page = $_GET["page"];}
   else { $page = "1"; }
   if ($stmt->execute(array($page))) {
       $row_page = $stmt->fetch(PDO::FETCH_ASSOC);
   } else {
       die("Неможливо отримати інформацію про поточну
сторінку з БД!");
   }
?>
```

Рис. 4.3. **Приклад виконання підготовлених запитів до БД за допомогою методів PDO->prepare() і PDO->execute()**

У PDO існує спосіб зазначення іменованих параметрів запиту. Для цього в запиті потрібно вказувати через двокрапку ім'я параметра (рис. 4.4), і за допомогою методу *bind()* зв'язати потрібний параметр з відповідною змінною. Після зв'язування всіх параметрів можна викликати метод *execute()* для виконання підготовленого запиту.

```php
<?php
  $query_ins =    "INSERT   INTO   page   (menu_item_name,
content)"."VALUES (:menu_name, :content)";
  $stmt_ins = $pdo->prepare($query_ins);

  // зв'язування параметрів зі змінними
  $stmt_ins->bindParam(':menu_name', $menu_item_name);
  $stmt_ins->bindParam(':content', $content);

  // вставка сторінки в таблицю 'page'
  $menu_item_name = 'Хобі';
  $content = "Основний вміст сторінки Хобі";
  $stmt_ins->execute();

  // Вставка чергової сторінки в таблицю 'page'
  $menu_item_name = 'Галерея';
  $content = "Основний вміст сторінки Галерея";
  $stmt_ins->execute();
?>
```

Рис. 4.4. **Приклад виконання підготовлених параметризованих запитів до БД з іменованими параметрами**

Пов'язувати зі змінними можна і неіменовані параметри. Для цього в якості першого параметра методу *bind()* замість імені з двокрапкою потрібно вказувати номер в порядку слідування в запиті параметра, який вказується знаком "?" (див. рис. 4.5).

Отримання кількості записів

Кількість записів, отриману в результаті виконання запиту, можна дізнатися за допомогою методу *rowCount()*.

Вибір рядка з результату

Вибрати рядок з результату запиту можна за допомогою методу ***fetch()*** (див. рис. 4.3).

```
<?php
  $query_upd =
           "UPDATE page SET content = ? WHERE id = ? ";
  $stmt_upd = $pdo->prepare($query_upd);

  // зв'язування параметрів зі змінними
  $stmt_upd->bindParam(1, $content);
  $stmt_upd->bindParam(2, $id);

  // Оновлення контенту сторінки в таблиці page з id=1
  $id = 1;
  $content = " Оновлений зміст основної частини Головної
сторінки ";
  $stmt_upd->execute();
?>
```

Рис. 4.5. **Приклад зв'язування підготовлених параметризованих запитів до БД з неіменованими параметрами**

Результат виконання методу ***fetch()*** аналогічний результату функції ***mysql_fetch_array()***. А для того, щоб даний метод повернув тільки асоціативний масив, потрібно як параметр передати константу об'єкта PDO::FETCH_ASSOC.

Завершення з'єднання з БД

Для завершення з'єднання з БД потрібно знищити об'єкт PDO. Для цього досить привласнити об'єкт NULL.

Обробка винятків у PDO

Для коректної обробки помилок (виняткових ситуацій) блоки коду, які повинні виконуватися разом, рекомендується поміщати ("обертати") в конструкцію "*try{ }catch{ }*". А щоб в разі помилки виконання запиту порушувався виняток, потрібно встановити відповідні атрибути об'єкта з'єднання з БД. Приклад обробки винятків для з'єднання з БД за допомогою PDO наведений на рис. 4.6.

```
<?php
try {
   $pdo=new
PDO("mysql:host=localhost;dbname=weblabdb",$username,$
password);

   $pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);

   $pdo->query("SET CHARACTER SET utf8");

}
catch (PDOException $e) {
   die($e->getMessage());
}
```

Рис. 4.6. **Обробка винятків у процесі створення об'єкта PDO**

Для оптимізації обробки виключень у ході з'єднання з БД можна визначити клас, який буде наслідувати клас PDO, і в конструкторі визначеного класу на час створення об'єкта PDO перевизначити стандартний обробник виключень, як показано в прикладі (див. рис. 4.7).

Такий підхід також корисний з точки зору безпеки: якщо веб-додаток не оброблює виняток з конструктора PDO, дією, прийнятою за замовчуванням в Zend Engine, є завершення сценарію та відображення налагоджувальних даних, в яких міститься повна інформація про з'єднання з БД, включаючи ім'я користувача та пароль.

Створення адміністраторської частини сайта ("адмінки")

Частину сайта, призначену тільки для адміністратора(ів) та інших привілейованих користувачів, рекомендується створювати в окремому

каталозі з метою не тільки логічного розмежування користувачів (залежно від ролі користувача), але і фізичного розмежування [27]. Даний підхід вважається більш безпечним.

```
<?php
Class SafePDO extends PDO {
    public static function
exception_handler($exception) {
        // Завершення скрипта і вивід даних винятку
        die('Виняток: ' . $exception->getMessage());
    }
    public function __construct($dsn, $user='',
$pass='', $drv_opts=array()) {
        // Тимчасова зміна обробника винятків PHP
        set_exception_handler(array(__CLASS__,
'exception_handler'));
        // створення об'єкта PDO
        parent::__construct($dsn, $user, $pass,
$drv_opts);
        // Відновлення обробника винятків PHP за
замовчуванням
        restore_exception_handler();
    }
}
// Підключення (безпечне) до БД
$pdo = new
SafePDO("mysql:host=localhost;dbname=weblabdb",
"root", "");
$pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION) ;
?>
```

Рис. 4.7. **Безпечне створення об'єкта PDO**

У разі використання ООП у роботі з БД рекомендується для кожної сутності БД (таблиця, збережена процедура, подання тощо) створювати окремий клас. А кожній дії над даними сутності (перегляд даних у табличній формі, додавання, редагування, видалення) повинен відповідати метод класу.

Крім того, гарним тоном в об'єктно-орієнтованому програмуванні вважається розміщення кожного класу в окремому файлі.

Коментарі в форматі PHPDoc

Система PHPDoc заснована на тому, що перед оголошенням кожної функції, класу, методу або властивості вставляється блок коментарів. Його наявність не обов'язкова, але дуже допомагає в прочитанні коду та дає можливість редактору (середовищу розробки), що володіє "IntelliSense", виводити підказки та докладний опис за відповідними елементами програмного коду.

Кожен блок коментарів починається з опису, а за ним можуть слідувати ті чи інші параметри. Наприклад, якщо PHPDoc-коментар належить до функції, слід описати її вхідні параметри та значення, яке повертається. Очевидно, в разі оголошення властивості класу його опис буде містити іншу інформацію.

У фрагменті коду (див. рис. 4.8) показаний приклад коментаря в стилі PHPDoc для простої функції.

Перше, що слід відзначити, – це початок блоку коментарів. Група символів ***/***** вказує синтаксичному аналізатору PHPDoc на початок блоку коментарів у даному форматі.

```
<?php
/**

* mySimpleFunction
* Проста функція, яка повертає вітання
* користувачеві за його іменем та віком
* @param   string  Sname   Ім'я користувача
* @param   int     Sage    Вік користувача
* @return  string          Готове повідомлення

*/
function mySimpleFunctiont($name, $age)
    $str = sprintf('Привіт %s, тобі %d років', $name, $age);
    return $str;
}
?>
```

Рис. 4.8. **Приклад коментування PHP-коду за допомогою PHPDoc**

У першому рядку блоку дається короткий опис. Зазвичай тут найкраще вказувати просто ім'я функції, класу, властивості або методу.

Наступний розділ коментаря – це більш розгорнутий опис. Тут треба спробувати описати, чим є функція, клас, властивість або метод з точки зору їх зовнішнього використання (тобто не як вони влаштовані, а що

конкретно роблять). Докладні описи операцій та тонкощів роботи розташовані не тут, а в стандартних коментарях всередині коду.

В останньому розділі блоку коментарів містяться різні параметри формату PHPDoc, які використовуються синтаксичним аналізатором для генерації якісної та зручної документації до програмного інтерфейса. Кожен параметр починається з символу @, за яким без пробілу слідує ім'я параметра. Після нього розташовується інформація, яку вимагає цей параметр згідно з синтаксисом.

У наведеному на рис. 4.8 прикладі є параметри @param і @return. Параметр @param описує аргументи функції:

- тип аргументу (в даному випадку перший аргумент – рядок символів);
- його ім'я (тут це $name);
- короткий опис даних, що передаються через цей аргумент.

Параметр @return описує, якого роду результат повертає функція; тут вказується його тип даних і дається короткий опис сутності переданої інформації.

4.4. Завдання до лабораторної роботи

1. Удосконалити сайт, розроблений в попередній лабораторній роботі, змінивши спосіб взаємодії з сервером БД на PDO.
2. Реалізувати адміністраторську частину сайта.
3. Реалізувати обробку винятків.
4. Прокоментувати кожен клас і кожен метод, використовуючи PHPDoc

4.5. Контрольні запитання

1. Що таке PDO?
2. У чому перевага PDO над іншими способами взаємодії PHP-скриптів з БД?
3. Що таке DSN?
4. Які ви знаєте способи виконання запитів в PDO?
5. Які переваги мають підготовлені запити?
6. Як можна передати параметр в запит в PDO?
7. Назвіть рекомендації для ООП з використанням БД.
8. Навіщо потрібні PHPDoc?

Лабораторна робота 5
Розроблення веб-додатків з використанням серверу БД MySQL з використанням процедурного й об'єктно-орієнтованого підходу (модуль PDO)

5.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення способів взаємодії PHP-скриптів із сервером БД MySQL для створення веб-додатків.

5.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Необхідно вивчити основи мови PHP для подальшого застосування під час розроблення веб-сайтів.

5.3. Загальні тези лабораторної роботи

Однією зі значних переваг PHP є підтримка широкого кола баз даних. Створення скрипта, що використовує бази даних, неймовірно просте. Сучасні PHP підтримують такі бази даних [2; 18; 22]:

- Adabas D
- PostgreSQL
- FrontBase
- Empress
- Solid
- Direct MS-SQL
- Hyperwave
- Velocis
- DB2 O
- Unix dbm
- Oracle (OCI7 і OCI8)
- Ovrimos
- InterBase
- dBase
- SQLite
- mSQL
- FilePro (тільки читання)
- Sybase
- **MySQL**
- IBM
- DBC
- Informix
- Ingres

Також у PHP включена підтримка DBX для роботи на абстрактному рівні, тому можна працювати з будь-якою базою даних, що використовує DBX. Крім того, PHP підтримує ODBC (Open Database Connection standard). Таким чином, можна працювати з будь-якою базою даних, яка підтримує цей всесвітньо визнаний стандарт.

Найбільш популярною БД у веб-додатках на PHP є MySQL. Даний сервер БД є дуже якісним і активно розвивається компанією Oracle. Ці причини зумовили вибір MySQL в курсі "Веб-програмування".

Як працює архітектура веб-баз даних

Необхідно розглянути послідовність дій, наведену на рис. 5.1.

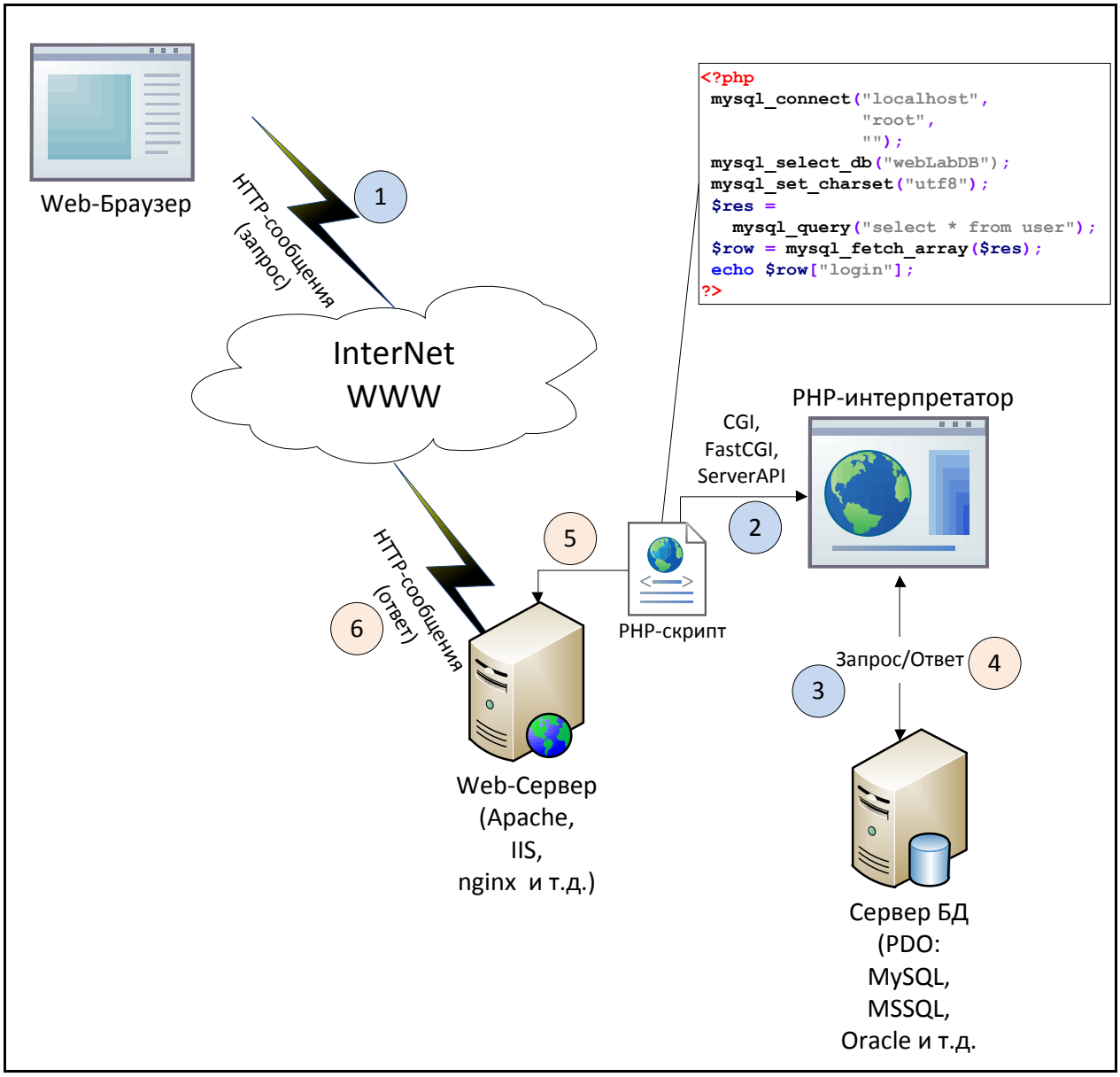


Рис. 5.1. **Архітектура веб-баз даних**

Виходячи з рис. 5.1 архітектура Веб-баз даних має таку послідовність.

1. Веб-браузер користувача видає HTTP-запит деякої Веб-сторінки. Наприклад, користувач шукає певну інформацію, використовуючи HTML-форму. Сторінка з результатами пошуку буде називатися results.php.

2. Веб-сервер приймає запит на results.php, витягує цей файл і передає на обробку інтерпретатору PHP.

3. Інтерпретатор PHP приступає до розбору сценарію. Сценарій містить команду **з'єднання з базою даних** і **виконання запиту** (пошуку інформації). PHP відкриває з'єднання з MySQL-сервером і відправляє відповідний запит.

4. Сервер MySQL приймає запит до бази даних, обробляє його та відправляє результат (набір шуканих даних) назад до інтерпретатора PHP.

5. Інтерпретатор PHP завершує виконання сценарію, що зазвичай включає в себе форматування результатів запиту в HTML. Після цього результат у вигляді HTML повертається веб-серверу.

6. Веб-сервер пересилає HTML в браузер, і користувач має можливість переглянути запитану інформацію [2; 18; 22].

Управління сервером БД MySQL

Для роботи (управління) з сервером MySQL існує кілька способів:

- командний рядок;
- клієнтський додаток (наприклад, dbForge Studio for MySQL);
- стандартна php-утиліта phpMyAdmin (див. рис. 5.2).

Порівняння (collation)

Під час створення БД необхідно звернути увагу на коректну установку такого параметра, як **порівняння (collation)**. Цей параметр є комплексним і включає **кодування даних** БД (це utf8 на рис. 5.2), **порядок символів** (general на рис. 5.2) у відповідному кодуванні (він впливає на сортування даних) і **чутливість до регістру символів** ("ci" на рис. 5.2 – **c**ase **i**nsensitive, тобто нечутливий до регістру символів), який впливає і на сортування, і на порівняння даних. Даний параметр обраної (або щойно створеної) БД можна змінити у вкладці "Дії".

Вибір потрібної БД або таблиці проводиться в лівій панелі (рис. 5.2).

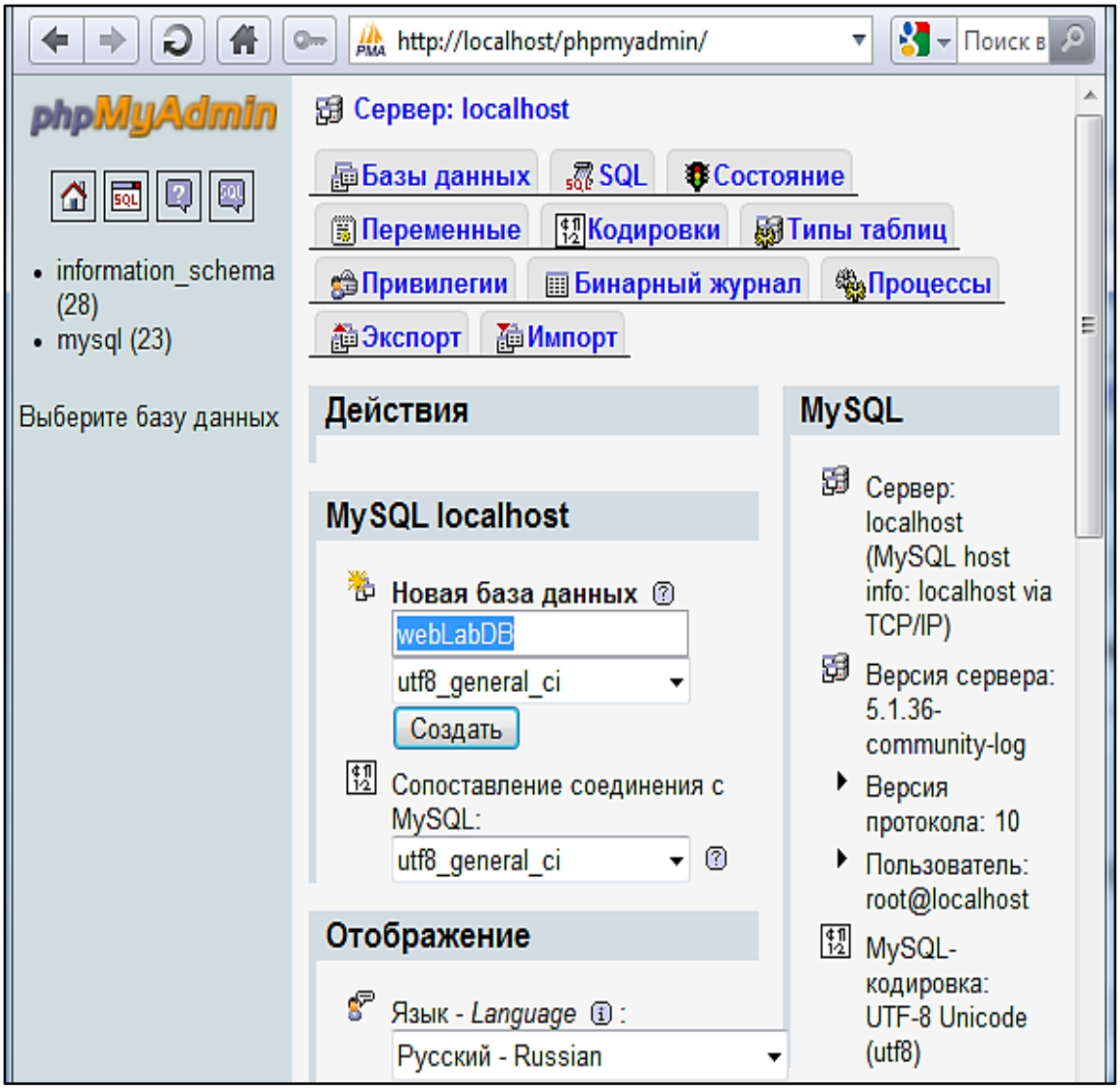


Рис. 5.2. **Вікно утиліти phpMyAdmin. Створення БД**

Створення таблиць БД

Для створення таблиць потрібно вибрати БД. У вікні, що з'явилося, (рис. 5.3) потрібно вказати ім'я створюваної таблиці та кількість полів, які можна буде змінити згодом.

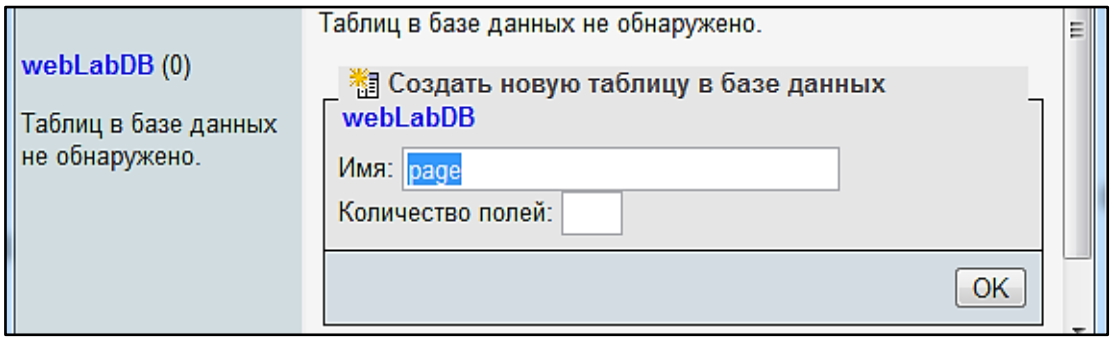


Рис. 5.3. **Вікно утиліти phpMyAdmin. Створення таблиці**

Типи стовпців в MySQL

У MySQL визначені три базових типи стовпців: числовий, дата та час, а також строковий (табл. 5.1). Кожна з цих категорій поділяється на безліч типів.

Таблиця 5.1

Основні типи стовпців в MySQL

Тип	Діапазон	Опис
INT	-2^{31} .. $2^{31}-1$ або 0 .. $2^{32}-1$	Звичайні цілі числа
FLOAT(точність)	залежить від точності	Може використовуватися для визначення чисел зі змінною точкою одинарної або подвійної точності
DATE	1000-01-01 9999-12-31	Дата. Відображується у вигляді РРРР-ММ-ДД
DATETIME	1000-01-01 00:00:00 9999-12-31 23:59:59	Дата та час. Відображуються у вигляді РРРР-ММ-ДД ГГ:ХХ:СС
VARCHAR	1 .. 255 символу	Рядок довільної довжини
TEXT	Максимальна довжина в символах: $2^{16}-1$	Нормальне поле TEXT

Кожен із трьох типів вимагає використання різного обсягу пам'яті. У процесі вибору типу стовпця головне – вибрати тип, що вимагає найменшого обсягу пам'яті, в якому поміщаються дані.

Для завершення процесу створення таблиці необхідно задати її параметри та параметри її полів (стовпців): типи полів, розмірність полів, значення за замовчуванням, чи є поле обов'язковим для заповнення, автоматичний інкремент (використовується для забезпечення унікальності полів) тощо (рис. 5.4) [2; 18; 22].

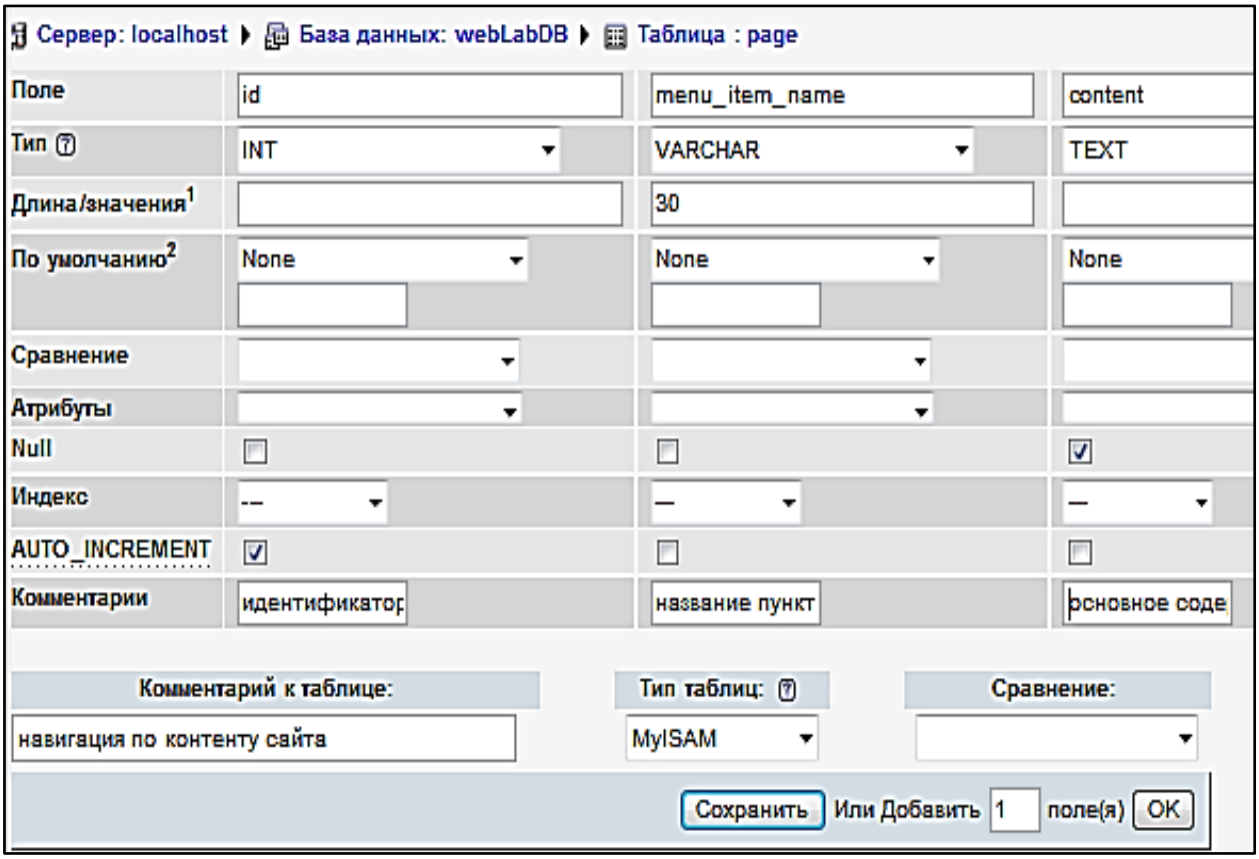


Рис. 5.4. **Створення таблиці. Налаштування параметрів**

На будь-яку активну дію з БД і її елементами утиліта phpMyAdmin генерує SQL-запит до сервера MySQL і відображає його (див. рис. 5.5) у результаті виконання.

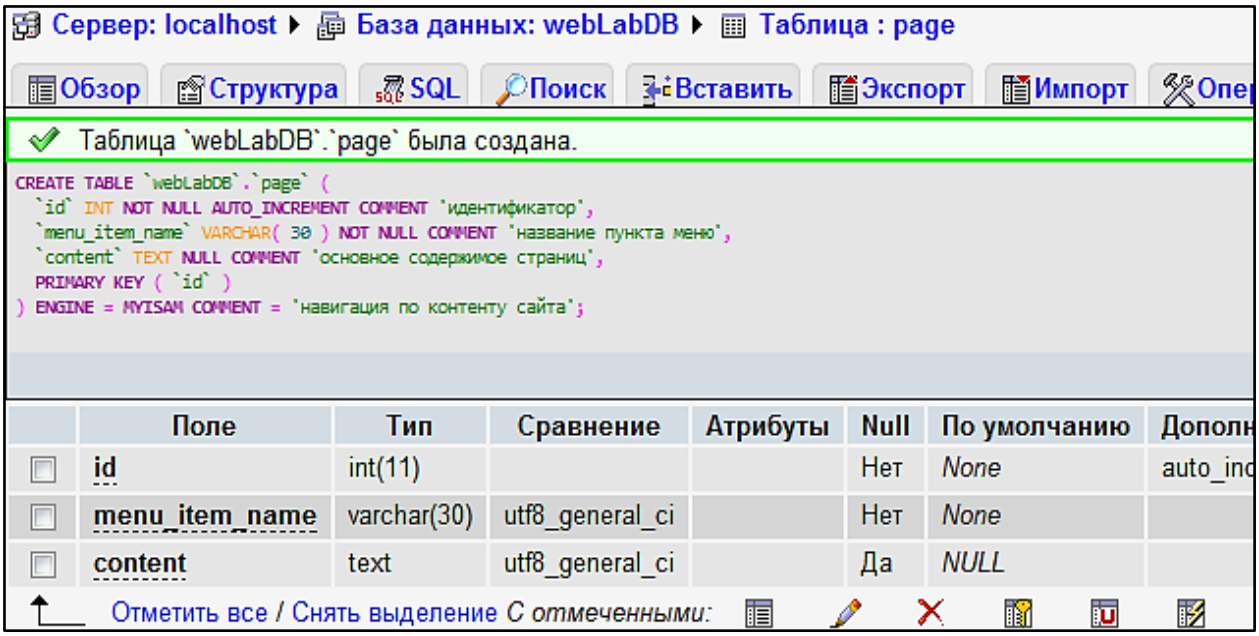


Рис. 5.5. **Результат виконання запиту на створення таблиці**

Наповнення даними

Щоб заповнити таблицю даними, потрібно скористатися вкладкою "Вставить", в якій можна додати один або кілька записів (рис. 5.6), наповнивши їх вручну. Поле "id" не потрібно заповнювати, оскільки воно заповнюється автоматично лічильником (autoincrement).

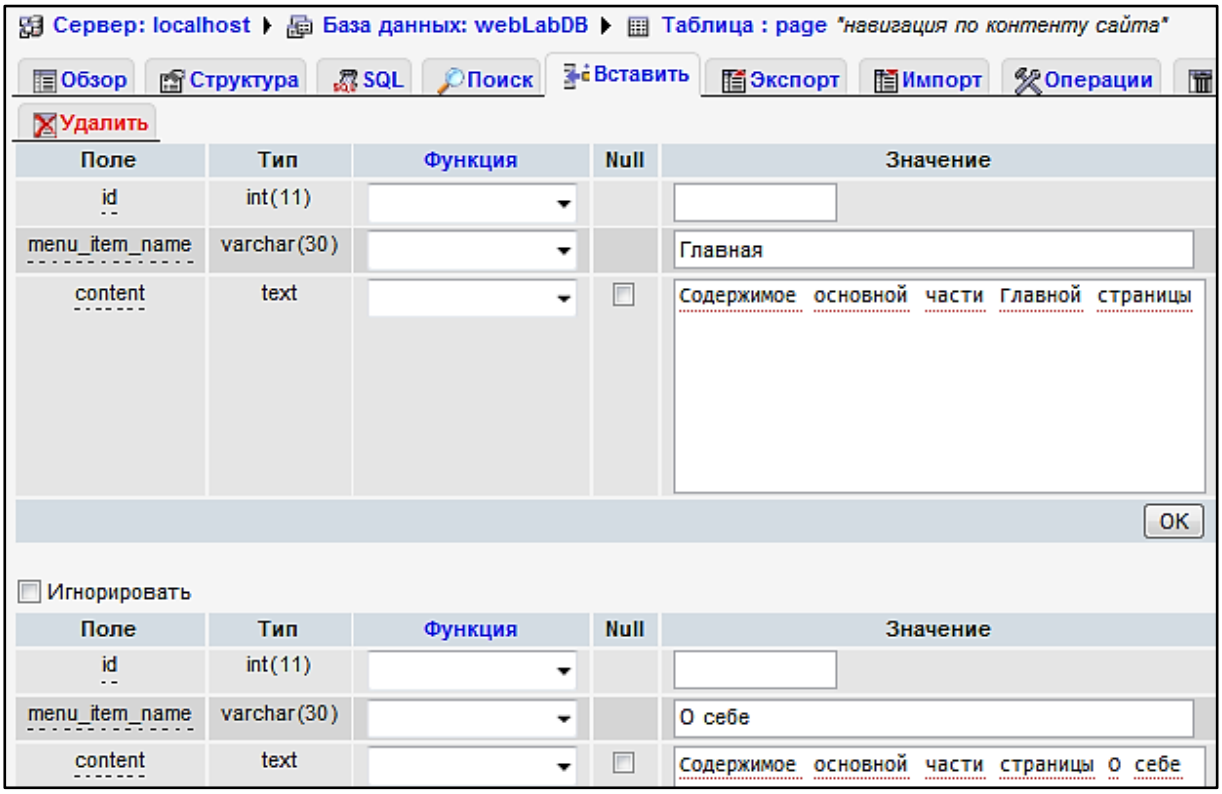


Рис. 5.6. **Вставка даних у таблицю**

Результат виконання вставки у вигляді SQL-запиту наведений на рис. 5.7.

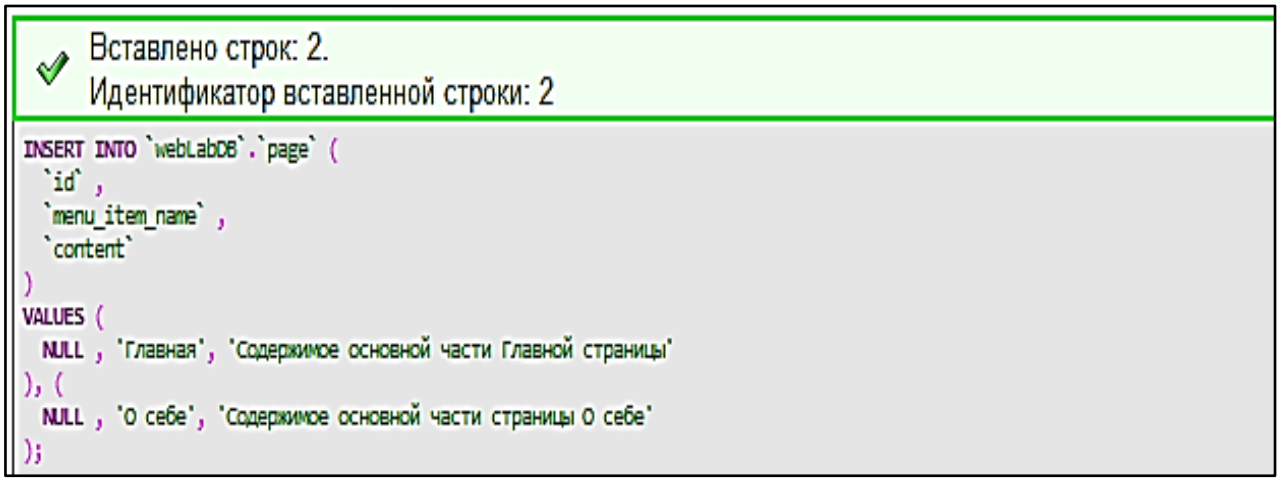


Рис. 5.7. **SQL-запит вставки даних у таблицю**

Для автоматичного заповнення таблиці даними можна скористатися імпортом (вкладка "Импорт") попередньо експортованих (вкладка "Экспорт") даних.

Створення PHP-скрипту навігації

Для організації навігації за контентом сайта з використанням БД даних з БД "webLabDB" і таблиці "page" необхідно реалізувати декілька стандартних кроків під час роботи з БД з PHP-скрипту (див. рис. 5.8).

1. Потрібно встановити з'єднання з сервером MySQL, яке буде використовуватися в усіх наступних запитах до сервера MySQL. Для цього служить функція mysql_connect(), в яку необхідно передати **ім'я** (або IP-адресу) **хоста**, на якому встановлений сервер MySQL; **ім'я облікового запису** на даному сервері, який має необхідні права на потрібну БД (у даному випадку – на webLabDB) і, звичайно, **пароль** цього облікового запису.
2. Якщо планується робота з однією БД, то зручніше вибрати поточну БД і в наступних зверненнях до об'єктів БД не вказувати її ім'я. Для цього застосовується функція mysql_select_db(), до якої як параметр передається ім'я БД.
3. Для того щоб напевно забезпечити ідентичність кодування з'єднання скрипта з сервером MySQL і кодування БД, застосовується функція mysql_set_charset(), котра в якості параметра приймає назву кодування (в даному випадку – UTF8).
4. Для виконання будь-якого SQL-запиту до MySQL з PHP застосовується функція mysql_query(), котра виконує SQL-запит, переданий їй в строковому вигляді в якості параметра.
5. У разі виконання SQL-запиту на вибірку даних функція mysql_query() повертає дескриптор отриманого набору даних. Щоб скористатися цими даними, їх потрібно порядково в циклі вибрати в масив. Функція mysql_fetch_assoc() вибирає черговий рядок з набору даних і повертає у вигляді асоціативного масиву. Якщо даних більше немає, то ця функція повертає FALSE.
6. Далі здійснюються стандартні робочі дії з асоціативними масивами.

Приклад, наведений на рис. 5.8, не розбитий на окремі функції та модулі для наочності, але ці дії рекомендовано провести. Крім того бажано перевіряти роботу з MySQL на помилки, для чого можуть стати в нагоді функції mysql_error() і mysql_errno(), які повертають повідомлення про помилку для останньої запущеної функції MySQL і номер помилки, відповідно.

```
<body>
  <?php
    mysql_connect("localhost", "root", "");
    mysql_select_db("webLabDB");
    mysql_set_charset("utf8");
    $res_query = mysql_query("SELECT * FROM pages");
  ?>
  <div id="menu">
  <?php
    while ($row = mysql_fetch_assoc($res_query)) {
  ?>
    <a href="index.php?page=<?php echo $row["id"]; ?>">
      <?php echo $row["menu_item_name"]; ?>
    </a>

  <?php
    }
  ?>
  </div>
  <div id="contenet">
<?php

    if (isset($_GET["page"])) {
        $page = $_GET["page"];
    }
    else {
        $page = "1";
    }

    $res_query = mysql_query("SELECT * FROM `pages`
WHERE `id` =" . $page);

    $row = mysql_fetch_assoc($res_query);

    echo $row["content"];
  ?>
  </div>
</body>
```

Рис. 5.8. **Реалізація навігації на контенті сайта в PHP з використанням БД MySQL**

5.4. Завдання до лабораторної роботи

1. Удосконалити сайт, розроблений в лабораторній роботі 5, використовуючи в якості джерел даних не масиви, а БД MySQL.
2. Наповнення БД можна здійснювати сторонніми засобами.
3. Реалізувати обробку помилок БД.

5.5. Контрольні запитання

1. За допомогою якої PHP-функції можна підключитися до сервера MySQL?
2. За допомогою якої PHP-функції можна вибрати необхідну БД MySQL?
3. За допомогою якої PHP-функції можна виконати запит до БД MySQL?
4. За допомогою якої PHP-функції можна зберегти рядок результату виконання запиту в асоціативний масив?
5. Для чого в налаштуваннях БД використовується такий параметр, як ″порівняння″ (collation)?
6. За допомогою якої PHP-функції можна отримати повідомлення про останню помилку MySQL?
7. Для чого застосовується PHP-функція mysql_set_charset() ?

Лабораторна робота 6
Розроблення веб-додатків мовою PHP з використанням API (XML DOM і Simple XML) для обробки XML-документів PHP і XML

6.1. Мета лабораторної роботи

Закріплення теоретичного матеріалу з попереднього лекційного матеріалу, вивчення основних механізмів роботи з XML у PHP.

6.2. Рекомендації щодо підготовки до виконання лабораторної роботи

Під час підготовки до лабораторної роботи необхідно вивчити основні можливості бібліотек JavaScript, зокрема jQuery.

6.3. Загальні тези лабораторної роботи

Основи XML

Стислий вступ в основи XML дозволить зрозуміти важливість цієї мови для PHP-розробника та навчитися створювати прості XML-документи.

Відомості про XML

Мову Extensible Markup Language (XML) можна назвати і мовою розмітки, і форматом зберігання текстових даних. Це підмножина мови Standard Generalized Markup Language (SGML); вона надає текстові засоби для опису деревовидних структур і їх застосування до інформації. XML слугує основою для цілого ряду мов і форматів – таких, як Really Simple Syndication (RSS), Mozilla XML User Interface Language (XUL), Macromedia Maximum eXperience Markup Language (MXML), Microsoft eXtensible Application Markup Language (XAML) і open source-мови Java XML UI Markup Language (XAMJ) [11; 23; 24; 29].

Структура XML

Базовим блоком даних в XML є елемент. Елементи виділяються початковим тегом – таким, як <book>, і кінцевим тегом – таким, як </book>. Кожному початковому тегу повинен відповідати кінцевий тег. Якщо для якогось початкового тега відсутній кінцевий тег, XML-документ оформлений неправильно, і синтаксичний аналізатор (парсер) не зможе проаналізувати його належним чином. Назви тегів зазвичай відображують тип елемента. Можна очікувати, що елемент book містить назву книги, наприклад, "Великий американський роман" (рис. 6.1). Текст, що міститься між тегами, включаючи пробіли, називається символьними даними.

```
<books>
 <book>
  <title>Великий американський роман</title>
  <characters>
   <character>
    <name>Кліфф</name>
    <desc>чудовий хлопець</desc>
   </character>
   <character>
    <name>Миловидна жінка</name>
    <desc>рідкісна красуня</desc>
   </character>
   <character>
    <name>Вірний Пес</name>
    <desc>любить поспати</desc>
   </character>
  </characters>
  <plot>
   Кліфф зустрічає Миловиду Жінку. Вірний Пес
спить, але прокидається, щоб
   обгавкати листоношу.
  </plot>
  <success type="bestseller">4</success>
  <success type="bookclubs">9</success>
  </book>
 </books>
```

Рис. 6.1. **Приклад XML-документа**

Імена XML-елементів та атрибутів можуть складатися лише з латинських літер верхнього (A-Z) і нижнього (a-z) регістрів, цифр (0 – 9), деяких спеціальних і неанглійських символів, а також трьох знаків пунктуації: дефіса, знака підкреслення та точки. Інші символи в іменах не допускаються.

XML чутливий до регістру. У наведеному прикладі <Book> і <book> описують два різних елемента. Обидва імені є прийнятними. Однак опис двох різних елементів іменами <Book> і <book> не можна вважати розумним рішенням з огляду на високу ймовірність помилок.

Кожен документ XML містить один і тільки один кореневий елемент. **Кореневий елемент** – це єдиний елемент XML-документа, для якого немає батьківського елемента. У наведеному прикладі кореневим є елемент <books>. Більшість XML-документів містить батьківські та дочірні елементи. Елемент <books> має один дочірній елемент <book>. У елемента <book> чотири дочірніх елемента: <title>, <characters>, <plot> і <success>. У елемента <characters> три дочірніх елемента, кожен з яких є елементом <character>. У кожного елемента <character> по два дочірніх елемента, <name> і <desc>.

Крім вкладених елементів, що створюють відносини батьківський – дочірній, XML-елементи можуть мати атрибути. Це пари ім'я – значення, приєднані до початкового тегу елемента. Імена відокремлюються від значень знаком рівності (=). Значення вкладенні в одинарні або подвійні лапки. На рис. 6.1 елемент <success> має два атрибути: "bestseller" і "bookclubs". XML-розробники практикують різні підходи до використання атрибутів. Більшу частину інформації, що міститься в атрибуті, можна помістити в дочірній елемент. Деякі розробники наполягають на тому, щоб інформація атрибутів складалася не з даних, а з метаданих, тобто відомостей про дані. Самі дані повинні міститися в елементах. Насправді рішення про те, чи використовувати атрибути, залежить від природи даних і від того, як вони витягуються з XML [11; 23; 24; 29].

Переваги XML

Одна з переваг XML полягає в його відносній простоті. XML-документ можна скласти в простому текстовому редакторі або текстовому процесорі, не вдаючись до спеціальних інструментів або ПЗ. Базовий синтаксис XML складається зі вкладених елементів, деякі з яких мають атрибути та вміст. Зазвичай елемент починається відкривальним тегом <тег> і закінчується відповідним закривальним тегом </тег>. XML чутливий до регістру та не

ігнорує пробіли і табуляції. Він дуже схожий на HTML, але, на відміну від HTML, дозволяє привласнювати тегам імена для кращого опису своїх даних. До переваг XML належить самодокументування, читабельний для користувачів і комп'ютерів формат, підтримка Unicode (що дозволяє створювати документи на різних мовах) і прості вимоги до синтаксису та синтаксичного аналізу.

Недоліки XML

XML багатослівний і надлишковий. Це породжує документи великого обсягу, що займають багато дискового простору і мережевих ресурсів. Передбачається, що він повинен бути читабельним для користувачів, але важко уявити собі людину, яка намагається прочитати файл XML з 7 млн вузлів. Найпростіші синтаксичні аналізатори функціонально не здатні підтримувати широкий набір типів даних [11; 23; 24; 29].

Правильно побудований XML

XML-документ вважається побудованим правильно, якщо в ньому дотримані правила XML-синтаксису. Неправильно побудований формат у технічному сенсі не є XML-документом. Наприклад, такий HTML-тег, як
, в XML неприйнятний; відповідний правильний тег виглядає як
. Кореневий елемент можна уявити собі як нескінченну шафу з документами. У вас всього одна шафа, але майже немає обмежень на тип і кількість її вмісту. У вашій шафі вміщується нескінченна кількість ящиків і папок для документів.

XML-документи мають як фізичну, так і логічну структуру.

Фізична структура

Сутності (англ. – Entity). Головною сутністю є зміст документа. Інші можливі сутності вказуються за допомогою посилання на сутності та можуть слугувати в ролі позначень спеціальних символів, посилань на спеціальні символи або окремі документи чи фрагменти тексту.

XML-декларація, в ній вказується версія XML, кодування й інша допоміжна інформація.

Декларація типу документа може застосовуватись для того, щоб додавати нові типи сутностей та визначати логічну структуру документа.

Логічна структура

XML-документ має ієрархічну логічну структуру та може подаватись у вигляді дерева. Вузлами цього дерева можуть бути:

- *елементи*, фізична структура яких складається із коректної пари відкривального та закривального тегів (<Назва-тега>) і (</Назва-тега>), або тега порожнього елемента (<Назва-тега />);
- *атрибути*, що мають вигляд пар ключ – значення (назва атрибута="значення атрибута") і знаходяться або у відкривальному, або у порожньому тегу (подібно до метаданих);
- *указівки щодо обробки документа* (англ. – Processing Instruction) (<?Обробник параметр ?>);
- *коментарі* (<!-- Текст коментаря -->);
- *текст* або у вигляді простого тексту, або фрагментів CDATA (<![CDATA[довільний текст]]>).

XML-документ повинен мати лише один кореневий елемент. Решта елементів є піделементами цього кореневого елемента.

Деякі веб-браузери здатні безпосередньо відображувати XML-документи. Це може досягатись шляхом застосування таблиці стилів (англ. – Stylesheet). Указані у таблиці стилів операції можуть призводити до перетворення XML-документа на інший, відмінний від XML формат.

Коректність XML-документів

Залишивши назви, дозволену ієрархію та значення елементів і атрибутів відкритими та з можливістю бути визначеними в спеціалізованих схемах або визначеннях типу документа (DTD), XML утворює синтаксичну основу для створення спеціалізованих, заснованих на XML мовах розмітки даних. Загальний синтаксис таких документів стабільний і наперед визначений – документи мають відповідати базовим вимогам XML, гарантуючи те, що довільне програмне забезпечення з підтримкою XML буде здатне щонайменше зчитати та відтворити відносну структуру інформації, яка міститься в них. Схема лише доповнює синтаксичні правила множиною обмежень. Зазвичай схеми обмежують назви елементів та атрибутів, дозволені типи значень і допустиму ієрархію елементів (наприклад, дозволяючи лише елементу з назвою ″народження″ містити піделемент з назвою ″місяць″ і з назвою ″день″) і кожен із них мусить містити лише літери. Обмеження, вказані в схемі, можуть також включати присвоєння певних типів

даних для впливу на те, як обробляється інформація. Наприклад, дані елемента ″місяць″ можна визначити як такі, що містять лише місяць, як це визначено відповідно до використаної мови схем.

Коректний XML-документ, що відповідає обмеженням схеми або DTD, називається *валідним*.

DTD (Document Type Definition)

Найдавнішим форматом схем для XML є успадкований від SGML формат визначення типу документа (Document Type Definition, DTD). Ставши через включення до стандарту XML 1.0 DTD поширеним форматом схем, він має такі обмеження:

- відсутність нових можливостей XML, із найважливішою з посеред них простори назв;
- брак виразності. Деякі формальні аспекти XML-документів неможливо відобразити в DTD;
- використовується спеціалізований, заснований не на XML, синтаксис для опису схем.

DTD все ще використовується в багатьох програмах, оскільки він вважається найпростішим форматом для аналізу та збереження, але він застарілий.

XML Schema

XML Schema (W3C) – мова схем XML-документів, опублікована в травні 2001 р. консорціумом W3C як ″рекомендація″ (англ. – Recommendation). Це була перша спеціалізована мова схем, що отримала статус ″рекомендації″ від консорціуму W3C.

XML Schema (буквально — XML-схема) прийшла на заміну DTD; скорочено позначається як XSD (від англ. XML Schema Definition). XSD набагато потужніші за DTD в описанні заснованих на XML мов. Вони використовують багатий набір типів даних, підтримують детальніші обмеження на структуру документів, і повинні оброблятись складнішими системами. XSD побудований на основі XML, що робить можливим використання звичайних інструментів XML для їх обробки, хоча реалізації XSD вимагають набагато більшого ніж просто можливість читати XML.

Як і решта мов схем, XML Schema використовується для визначення певних правил ″валідності″ XML-документів. Разом із правилами

валідності структури документа стандарт XML Schema дозволяє визначати типи даних значень елементів та атрибутів у XML-документах.

Екземпляр XML Schema називається визначенням XML-схеми (англ. – XML Schema Definition, скорочено – XSD) і зазвичай має розширення в імені файлу ".xsd". Саму мову інколи неформально називають XSD (ікс-ес-ді). Було зроблене припущення, що WXS (скорочено від англ. W3C XML Schema) є кращим акронімом назви мови. Однак цей акронім не мав широкого застосування і був відкинутий робочою групою консорціума. XSD є скороченням від XML Schema Datatypes, частини специфікації XML Schema, що визначає типи даних. Для роботи з великими специфікаціями XML Schema розроблений пакет JAXB, який автоматично створює класи Java на основі XML Schema та в зворотному напрямі – на основі класів Java генерує файл XML Schema.

Серед недоліків XSD називають такі:

- специфікація дуже велика, що робить її складною для розуміння та реалізації;
- оснований на XML синтаксис додає надмірності мові, що ускладнює читання та запис XSD;
- валідація відносно схеми може бути дорогим додатком до синтаксичного аналізу XML-документів;
- можливості моделювання дуже обмежені, без надання впливу значень атрибутів на вміст елементів;
- модель отримання типів даних є дуже обмеженою, зокрема в тому, що набуття шляхом розширення не є корисним;
- механізми ключа/посилання на ключ/унікальність не враховують тип даних;
- концепція PSVI (англ. – Post Schema Validation Infoset) не має стандартного подання або прикладного програмного інтерфейса, що працює проти незалежності від реалізації, якщо не виконується повторна валідація.

RELAX NG

RELAX NG є іншою поширеною мовою схем для XML. Уперше RELAX NG був визначений стандартом OASIS, а нині — Міжнародним стандартом ISO (як частина DSDL). Ця мова схем має два формати:

оснований на XML і компактний – не-XML. Компактний синтаксис призначений для покращення можливості читання та написання схем. Існує точно визначений спосіб перетворення компактного формату в оснований на XML і навпаки – не втрачаються переваги від використання стандартних XML-інструментів. Тому RELAX NG має простіші системи для визначення та валідації у порівнянні з XML Schema. Це робить її привабливішою для використання та реалізації. Також існує можливість використання модулів роботи з типом даних; наприклад, автор схеми RELAX NG може вказати, що значення XML-документа мають відповідати визначенням типам даних у форматі XML Schema Datatypes.

ISO DSDL та інші мови схем

Стандарт ISO DSDL (англ. – Document Schema Description Languages, мови описання схем документів) об'єднує широке коло малих мов схем, кожна із яких призначена для розв'язання окремих проблем. До DSDL належить: RELAX NG із повним і компактним синтаксисом; мова припущень Schematron; мови для визначення типів даних, обмежень на літери, перейменування та розкривання мнемонік, та основане на просторах назв перенаправлення фрагментів документів у різні валідатори. Мови DSDL все ще не мають підтримки, як у XML Schema. Певною мірою вони є реакцією видавців на брак можливостей XML Schema для видавничої справи.

Деякі мови схем не тільки описують структуру певного формату XML-документів, а мають обмежені можливості впливу на обробку документів цього формату. У DTD і XSD є така можливість; наприклад, вони можуть визначати значення для атрибутів "за замовчуванням". Натомість, як RELAX NG, так і Schematron таких можливостей не мають.

Читання, обробка та написання XML у PHP5

SimpleXML, за необхідності в поєднанні з DOM, – ідеальний вибір для читання, обробки та складання в PHP5 простих, передбачуваних і відносно компактних документів.

З множини API, присутніх в PHP 5, DOM – найвідоміший, а на SimpleXML найпростіше програмувати. У типових ситуаціях – таких, як розглянуті, вони найбільш ефективні.

Розширення DOM

Document Object Model (DOM) – це прийнятий W3C стандартний набір об'єктів для документів HTML і XML, стандартна модель поєднання цих об'єктів і стандартний інтерфейс для доступу до них і маніпуляцій з ними. Багато постачальників підтримують DOM в якості інтерфейса до своїх спеціальних структур даних і API, завдяки чому модель DOM відома масі розробників. DOM легко освоїти та застосовувати, оскільки його структура в пам'яті нагадує початковий документ XML. Щоб передати інформацію з додатком, DOM створює дерево об'єктів, яке точно повторює дерево елементів файла XML, тому кожен елемент XML слугує вузлом цього дерева. DOM – це парсер, заснований на деревоподібній структурі. Оскільки DOM будує дерево всього документа, він споживає багато ресурсів пам'яті та часу процесора. Тому аналіз дуже великих документів за допомогою DOM непрактичний через проблеми продуктивності. Отже розширення DOM використовується головним чином через його здатність імпортувати формат SimpleXML і виводити XML в форматі DOM, або, навпаки, для використання в якості строкових даних або XML-файла.

SimpleXML

Розширення SimpleXML – це кращий інструмент для синтаксичного аналізу XML. Для його роботи потрібне PHP5. Воно взаємодіє з DOM для складання XML-файлів і має вбудовану підтримку XPath. SimpleXML найкраще працює з нескладними даними типу записів – такими, як XML, який передається у вигляді документа або рядка з іншої частини того ж додатка. Якщо XML-документ не надто складний, не дуже глибокий і не містить змішаного контенту, для SimpleXML кодувати простіше, ніж для DOM, як і випливає з назви. До того ж він надійніший у роботі з відомою структурою документа.

DOM в дії

Модель DOM, реалізована в PHP5, – це та ж специфікація W3C DOM, яка використовується в браузерах і з якою працюють за допомогою JavaScript. Застосовують ті ж методи. Отже, способи кодування відомі. На рис. 6.2 проілюстроване використання DOM для створення XML-рядка та XML-документа, відформатованих в цілях читабельності.

```php
<?php
// Створює XML-рядок і XML-документ за допомогою DOM
$dom = new DomDocument('1.0');
// додавання кореня - <books>
$books =
 $dom->appendChild($dom->createElement('books'));
// додавання елемента <book> в <books>
$book =
 $books->appendChild($dom->createElement('book'));
// додавання елемента <title> в <book>
$title =
 $book->appendChild($dom->createElement('title'));
//додавання елемента текстового вузла <title> в <title>
$title->appendChild(
    $dom->createTextNode('Great American Novel'));
//генерація xml
$dom->formatOutput = true; // установка атрибуту
formatOutput domDocument в значення true
// save XML as string or file
$test1 = $dom->saveXML(); // передача рядка в test1
$dom->save('test1.xml'); // збереження файла
?>
```

Рис. 6.2. **Приклад XML-документа**

Це призводить до створення вихідного файла (рис. 6.3).

```xml
<?xml version="1.0"?>
<books>
  <book>
    <title>Great American Novel</title>
  </book>
</books>
```

Рис. 6.3. **Вихідний файл XML**

Цей код імпортує об'єкт SimpleXMLElement в об'єкт DOM Element, ілюструючи взаємодію DOM і SimpleXML

SimpleXML в дії

На рис. 6.4 розглянутий тестовий XML-файл як include-файл з метою читабельності.

```
<?php
 $xmlstr = <<<XML
 <books>
 <book>
  <title>Great American Novel</title>
  <characters>
   <character>
    <name>Cliff</name>
    <desc>really great guy</desc>
   </character>
   <character>
    <name>Lovely Woman</name>
    <desc>matchless beauty</desc>
   </character>
   <character>
    <name>Loyal Dog</name>
    <desc>sleepy</desc>
   </character>
  </characters>
  <plot>
   Cliff meets Lovely Woman.  Loyal Dog sleeps, but
wakes up to bark at mailman.
  </plot>
  <success type="bestseller">4</success>
  <success type="bookclubs">9</success>
 </book>
 </books>
XML;
?>
```

Рис. 6.4. **Тестовий XML-файл**

У гіпотетичному Ajax-додатку може, наприклад, знадобитися витягти з XML-документа поштовий індекс і звернутися до бази даних.

На рис. 6.5 з XML-файла попереднього прикладу витягується елемент **<plot>.**

```
<?php

  include 'example.php';
  $xml = new SimpleXMLElement($xmlstr);

  echo $xml->book[0]->plot; /* "Cliff meets Lovely
Woman. ..." */

  ?>
```

Рис. 6.5. **Витягування вузла XML**

З іншого боку, може знадобитися отримати багаторядкову адресу. Коли у одного батьківського елемента кілька примірників дочірнього елемента, застосовується звичайна методика ітерування. Ця функціональність показана на рис 6.6.

```
<?php
 include 'example.php';

 $xml = new SimpleXMLElement($xmlstr);
 /* For each <book> node, echo a separate <plot>. */
 foreach ($xml->book as $book) {
   echo $book->plot, '<br />';
 }
 ?>
```

Рис. 6.6. **Витягування декількох екземплярів елемента XML**

Крім читання імен елементів і їх значень, SimpleXML може також звертатися до атрибутів елемента. На рис. 6.7 показане звернення до атрибутів елемента з попереднього прикладу (див. рис. 6.7); це робиться так само, як звернення до елементів масиву.

```php
<?php
 include 'example.php';

 $xml = new SimpleXMLElement($xmlstr);

 /* Access the <success> nodes of the first book.
 * Output the success indications, too. */
 foreach ($xml->book[0]->success as $success) {
    switch((string) $success['type']) {
        // Get attributes as element indices
    case 'bestseller':
        echo $success, ' months on bestseller list';
        break;
    case 'bookclubs':
        echo $success, ' bookclub listings';
        break;
    }
 }
 ?>
```

Рис. 6.7. **Демонстрація звернення SimpleXML до атрибутів елемента**

Щоб порівняти елемент або атрибут з рядком або передати його функції, якій потрібний рядок, треба перетворити його в рядок за допомогою оператора (string). Інакше, за замовчуванням, PHP розглядає елемент як об'єкт (рис. 6.8).

```php
<?php
 include 'example.php';
 $xml = new SimpleXMLElement($xmlstr);

 if ((string)$xml->book->title ==
        'Great American Novel') {
    print 'My favorite book.';
 }
 htmlentities((string) $xml->book->title);
 ?>
```

Рис. 6.8. **Перетворення на рядок**

Дані в SimpleXML не обов'язково мають бути незмінними. На рис. 6.9 виводиться новий XML-документ – точна копія вихідного за винятком того, що в новому ім'я Cliff змінено на Big Cliff.

```
<?php

 include 'example.php';
 $xml = new SimpleXMLElement($xmlstr);

 $xml->book[0]->characters->character[0]->name =
     'Big Cliff';
 echo $xml->asXML();
 ?>
```

Рис. 6.9. **Зміна текстового вузла за допомогою SimpleXML**

Починаючи з версії PHP 5.1.3 SimpleXML доповнений можливістю легко додавати дочірні елементи й атрибути. На рис. 6.10 виводиться XML-документ, заснований на вихідному, але з доданим новим персонажем і описом.

```
<?php

include 'example.php';
 $xml = new SimpleXMLElement($xmlstr);

 $character =
    $xml->book[0]->characters->addChild('character');
 $character->addChild('name', 'Yellow Cat');
 $character->addChild('desc', 'aloof');

 $success = $xml->book[0]->addChild('success', '2');
 $success->addAttribute('type', 'reprints');

 echo $xml->asXML();
 ?>
```

Рис. 6.10. **Додавання дочірніх і текстових вузлів за допомогою SimpleXML**

SimpleXML, за необхідності в поєднанні з DOM, слугує ідеальним інструментом для розробників, які мають справу з простими, передбачуваними і відносно компактними XML-документами. Версія PHP5 значно розширила можливості програмістів в частині роботи з XML у PHP.

Розширені методи парсинга XML

У PHP5 збільшена різноманітність методів синтаксичного аналізу (парсинга) XML. Парсер Expat SAX Джеймса Кларка, який тепер заснований на бібліотеці libxml2, більше не є єдиним повнофункціональним парсером. Нам вже знайома можливість парсинга за допомогою DOM, узгоджена зі стандартом W3C. Додаткові можливості пропонує SimpleXML і XMLReader, який простіший в розумінні та швидший в роботі, ніж SAX. Усі розширення XML тепер засновані на бібліотеці libxml2 проекту GNOME. Ця уніфікована бібліотека дозволяє різним розширенням взаємодіяти між собою. Слід розглянути методи парсинга XML в PHP5 з акцентом на обсяжні або складні XML-документи [11; 23; 24; 29].

Основи XML-парсинга

Існує два основних способи XML-парсинга: на базі дерев і на базі потоків.

Метод дерева передбачає завантаження в пам'ять XML-документа цілком. Деревоподібна структура файла дозволяє довільно звертатися до елементів документа та редагувати XML. Прикладами парсерів за методом дерева слугують DOM і SimpleXML. Вони зберігають деревоподібну структуру в пам'яті в різних, але взаємодіючих форматах.

У *потоковому парсінгу* весь документ в пам'ять не завантажується. У даному випадку термін "потік" уживається в тому ж сенсі, що і в описі поточного аудіо. Відбувається те ж саме і з тих же причин: дані надходять дрібними порціями з метою економії смуги пропускання та ресурсів пам'яті. Потоковому парсінгу доступний тільки той вузол, який аналізується в даний момент, а редагування повного XML-документа неможливе. Прикладами поточних парсерів є XMLReader і SAX.

Парсери, що працюють за методом дерева

Парсери, що працюють за методом дерева, завантажують у пам'ять весь документ, тому корінь нагадує стовбур дерева, а всі дочірні, внучаті

та більш віддалені нащадки й атрибути слугують гілками. Найвідоміший парсер, який працює за методом дерева, це DOM. Найпростіший – SimpleXML.

Парсинг за допомогою DOM

Стандарт DOM, згідно з W3C, – це ″... незалежний від платформи та мови програмування інтерфейс, який дозволяє програмам і сценаріям динамічно звертатися до документів і редагувати їх зміст, структуру та стиль″. Бібліотека libxml2 проекту GNOME реалізує DOM разом з усіма його методами на мові C. Оскільки всі XML-розширення PHP5 засновані на libxml2, вони підтримують повну взаємодію між собою. Ця взаємодія значно покращує їх функціональність. Наприклад, можна витягти елемент за допомогою поточного парсера XMLReader, імпортувати його в DOM і витягти дані з використанням XPath.

Парсер DOM працює за методом дерева. Він простий для розуміння та застосування, тому що його структура в пам'яті нагадує оригінальний XML-документ. DOM передає інформацію з додатком, створюючи дерево об'єктів, в точності повторює дерево елементів з XML-файла, а кожен елемент XML слугує вузлом цього дерева. DOM – це стандарт W3C, що принесло йому визнання розробників зважаючи на його узгодженість з іншими мовами програмування. Оскільки DOM будує дерево всього документа, він споживає великий обсяг пам'яті та багато ресурсів процесора.

DOM у дії

Якщо через якесь обмеження ви змушені вибрати єдиний парсер, має сенс вибрати DOM хоча б в силу його гнучкості. DOM дозволяє складати XML-документи, модифікувати їх, звертатися до них, перевіряти та перетворювати їх. До того ж можна використовувати всі методи та властивості DOM. Більшість методів DOM другого рівня реалізовані з належною підтримкою властивостей. Завдяки надзвичайній гнучкості DOM аналізовані документи можуть бути надто складними. Однак пам'ятайте, що за гнучкість доводиться платити тим, що весь документ завантажується в пам'ять цілком.

У прикладі, наведеному на рис. 6.11, DOM застосовується для парсинга документа та вилучення елемента за допомогою функції getElementById. Перед посиланням на ідентифікатор документ необхідно перевірити,

встановивши validateOnParse = true. Відповідно до стандарту DOM, для цього потрібний DTD, який визначає атрибут ID як тип ID.

```
<?php

$doc = new DomDocument;

// Перш ніж посилатися на id, документ потрібно
перевірити
$doc->validateOnParse = true;
$doc->Load('basic.xml');

echo "The element whose id is myelement is: " .
$doc->getElementById('myelement')->tagName . "\n"; ?>
```

Рис. 6.11. **Використання DOM з простим документом**

Функція getElementsByTagName () повертає новий екземпляр класу DOMNodeList, що містить елементи з заданим ім'ям тега. Звичайно, потрібно перегорнути список. Зміна структури документа під час роботи зі списком NodeList, виданими функцією getElementsByTagName (), впливає на список NodeList, з яким ви працюєте (рис. 6.12). Перевірка в даному випадку не потрібна.

```
<?php

DOMDocument {
 DOMNodeList getElementsByTagName(string name);
}
?>
```

Рис. 6.12. **Використання DOM з простим документом**

У прикладі на рис. 6.13 DOM використовується з XPath.

```php
<?php

$doc = new DOMDocument;

// Ми не хочемо мати справи з пробілами
$doc->preserveWhiteSpace = false;

$doc->Load('book.xml');

$xpath = new DOMXPath($doc);

// Ми почали з кореневого елемента
$query =
'//book/chapter/para/informaltable/tgroup/tbody/row/e
ntry[. = "en"]';

$entries = $xpath->query($query);

foreach ($entries as $entry) {

  echo "Found {
    $entry->previousSibling->previousSibling-
>nodeValue
    }," .
       " by {$entry->previousSibling->nodeValue}\n";
}

?>
```

Рис. 6.13. **Використання DOM і парсинг із застосуванням XPath**

Парсинг за допомогою SimpleXML

Розширення SimpleXML – ще один спосіб парсинга XML-документа. Для розширення SimpleXML потрібний PHP5 і використовується вбудована підтримка XPath. SimpleXML найкраще працює з нескладними даними XML. У тому випадку, якщо XML-документ не надто складний, глибокий і не має змішаного контенту, SimpleXML простіший в застосуванні, ніж DOM, як і передбачає його назва. Він інтуїтивно зрозумілий, якщо ви працюєте з відомою структурою документа.

SimpleXML в дії

SimpleXML володіє багатьма перевагами DOM і простіший в програмуванні. Він дозволяє легко звертатися до дерева XML, має вбудовану підтримку перевірки та XPath, а також взаємодіє з DOM, забезпечуючи йому підтримку в читанні та записі XML-документів. Документи, аналізовані SimpleXML, пишуться легко та швидко. Однак пам'ятайте, що, як і в DOM, простота і гнучкість SimpleXML досягається ціною завантаження в пам'ять XML-документа цілком.

Код, наведений на рис. 6.14, витягує з прикладу XML-документа сюжет твору, що міститься в елементі <plot>.

```
<?php

$xmlstr = <<<XML
<?xml version='1.0' standalone='yes'?>
<books>
  <book>
     <title>Great American Novel</title>
     <plot>
        Cliff meets Lovely Woman. Loyal Dog sleeps,
but wakes up to bark at mailman.
     </plot>
     <success type="bestseller">4</success>
     <success type="bookclubs">9</success>
  </book>
</books>
XML;
?>

<?php

$xml = new SimpleXMLElement($xmlstr);

echo $xml->book[0]->plot; /* "Cliff meets Lovely
Woman. ..." */

?>
```

Рис. 6.14. **Витягування фрагмента тексту**

З іншого боку, може знадобитися отримати багаторядкову адресу. Коли у одного батьківського елемента є кілька примірників дочірнього

елемента, застосовується звичайна методика ітерування. Ця функціональність демонструється на рис. 6.15.

```
<?php

$xmlstr = <<<XML
<xml version='1.0' standalone='yes'?>
<books>
  <book>
     <title>Great American Novel</title>
     <plot>
        Cliff meets Lovely Woman.
     </plot>
     <success type="bestseller">4</success>
     <success type="bookclubs">9</success>
  </book>
  <book>
     <title>Man Bites Dog</title>
     <plot>
        Reporter invents a prize-winning story.
     </plot>
     <success type="bestseller">22</success>
     <success type="bookclubs">3</success>
  </book>
</books>
XML;
?>

<php

$xml = new SimpleXMLElement($xmlstr);

foreach ($xml->book as $book) {
  echo $book->plot, '<br />';
}
?>
```

Рис. 6.15. **Витягування декількох екземплярів елемента**

Крім читання імен елементів і їх значень, SimpleXML може звертатися до атрибутів елемента. На рис. 6.16 здійснюється звернення до атрибутів елемента; це робиться точно так же, як звернення до елементів масиву.

```
<?php

$xml = new SimpleXMLElement($xmlstr);

foreach ($xml->book[0]->success as $success) {

  switch((string) $success['type']) {

  case 'bestseller':
     echo $success, ' months on bestseller list <br
/>';
     break;

  case 'bookclubs':
     echo $success, ' bookclub listings<br />';
     break;
  }
}
?>
```

Рис. 6.16. **Демонстрація звернення SimpleXML до атрибутів елемента**

В останньому прикладі (див. рис. 6.17) SimpleXML і DOM використовуються з розширенням XMLReader. За допомогою XMLReader дані передаються послідовно, між окремими елементами з використанням методу expand (). Цим методом вузол, переданий XMLReader, можна перетворити на DOMElement, а потім передати SimpleXML.

Потокові парсери

Потокові парсери називаються так тому, що вони аналізують XML в потоці, багато в чому нагадуючи роботу потокового аудіо. У кожен момент часу вони працюють з одним окремим вузлом, а закінчивши, зовсім забувають про його існування. XMLReader – це pull-парсер, і програмування для нього багато в чому нагадує витягування результату запиту до таблиці бази даних за допомогою курсора. Це полегшує роботу з незнайомими або непередбачуваними XML-файлами.

```php
<?php

// Парсинг великого документа за допомогою Expand і
SimpleXML
$reader = new XMLReader();

$reader->open("tooBig.xml");

while ($reader->read()) {

    switch ($reader->nodeType) {

        case (XMLREADER::ELEMENT):

        if ($reader->localName == "entry") {

            if ($reader->getAttribute("ID") == 5225) {
                $node = $reader->expand();
                $dom = new DomDocument();
                $n = $dom->importNode($node,true);
                $dom->appendChild($n);
                $sxe = simplexml_import_dom($n);
                echo $sxe->title;
            }
        }
    }
}
?>
```

Рис. 6.17. **Використання SimpleXML і DOM з розширенням XMLReader для аналізу об'ємного XML-документа**

Парсинг за допомогою XMLReader

XMLReader – це потоковий парсер того типу, який часто називають курсорними або pull-парсером. XMLReader витягує інформацію з XML-документа на вимогу. Він заснований на API, отриманому з C # XmlTextReader. У PHP 5.1 він включений задіяний за замовчуванням. І заснований на бібліотеці libxml2. До виходу PHP 5.1 розширення XMLReader не включене за замовчуванням, але доступне в PECL. XMLReader підтримує простору імен і перевірку, включаючи DTD і Relaxed NG.

XMLReader у дії

Як потоковий парсер XMLReader зручний для роботи з об'ємними XML-документами; програмувати в ньому набагато легше та зазвичай швидше, ніж в SAX. Це кращий потоковий парсер.

У наступному прикладі (рис. 6.18) об'ємний XML-документ аналізується за допомогою XMLReader.

```
<?php
$reader = new XMLReader();
$reader->open("tooBig.xml");
while ($reader->read()) {
  switch ($reader->nodeType) {
  case (XMLREADER::ELEMENT):
   if ($reader->localName == "entry") {
    if ($reader->getAttribute("ID") == 5225){
     while ($reader->read()) {
      if ($reader->nodeType == XMLREADER::ELEMENT){
        if ($reader->localName == "title"){
                $reader->read();
                echo $reader->value;
                break;
        }
        if ($reader->localName == "entry"){break;} }
     }}
   }}
}?>
```

Рис. 6.18. **XMLReader з об'ємним XML-файлом**

Парсинг за допомогою SAX

Simple API for XML (SAX) – це потоковий парсер. Події пов'язані з читаним XML-документом, тому SAX програмується в стилі зворотних викликів. Існують події для відкривальних і закривальних тегів елемента, сутностей і помилок парсинга. Головна причина використання парсера SAX замість XMLReader полягає в тому, що парсер SAX іноді ефективніший і зазвичай краще відомий. Важливий недолік – код для парсера SAX складніший і його важче писати, ніж для XMLReader.

SAX у дії

SAX відомий тим, хто працював з XML у PHP4, а розширення SAX у PHP5 сумісне з версією, до якої користувачі звикли. Оскільки це потоковий парсер, він добре справляється з об'ємними файлами, але XMLReader кращий за нього.

На рис. 6.19 наведений приклад обробки об'ємного XML-документа парсером SAX.

```
<?php
class SaxClass {
  private $hit = false;
  private $titleHit = false;
  // зворотний виклик для початку кожного елементу
  function startElement($parser_object,
                        $elementname, $attribute) {
     if ($elementname == "entry") {
        if ( $attribute['ID'] == 5225) {
           $this->hit = true;
        } else {
           $this->hit = false;
        }
     }
     if ($this->hit && $elementname == "title") {
        $this->titleHit = true;
     } else {
        $this->titleHit =false;
     }
// зворотний виклик для кінця кожного елемента
function endElement($parser_object, $elementname){ }

// зворотний виклик для вмісту кожного елемента
function contentHandler($parser_object,$data)
  {
     if ($this->titleHit) {
        echo trim($data)."<br />";
     }
  }
}
```

Рис. 6.19. **Використання SAX для аналізу об'ємного XML-файла** (початок)

```
 /* Функція запуску парсинга, коли всі значення
встановлені і файл відкритий */
 function doParse($parser_object) {
   if (!($fp = fopen("tooBig.xml", "r")));

   //прокрутка данных
   while ($data = fread($fp, 4096)) {
      //анализ фрагмента
      xml_parse($parser_object, $data, feof($fp));
   }
 }

 $SaxObject = new SaxClass();
 $parser_object = xml_parser_create();
 xml_set_object ($parser_object, $SaxObject);

 xml_parser_set_option($parser_object,
            XML_OPTION_CASE_FOLDING, false);

 xml_set_element_handler($parser_object,
            "startElement","endElement");
 xml_set_character_data_handler($parser_object,
            "contentHandler");

 doParse($parser_object);

 ?>
```

Рис. 6.19. **Використання SAX для аналізу об'ємного XML-файла** (закінчення)

PHP5 пропонує різноманітні методи парсинга. Парсинг за допомогою DOM, який тепер повністю сумісний зі стандартом W3C, – відомий варіант, який підходить для складних, але компактних документів. SimpleXML – це спосіб роботи з простими та не дуже об'ємними документами, а потоковий парсер XMLReader, легший та швидший за SAX, зручніший для дуже великих документів.

6.4. Завдання до лабораторної роботи

1. Удосконалити сайт, розроблений у попередній роботі, додати на сторінку flashxml- компонент відповідно до варіанта.
2. В адміністративній частині сайта додати можливість налаштування конфігураційного XML-файла або можливість формування елементів flashxml-компонента (файл images.xml або menu.xml або tagcloud.xml).
3. Для роботи з конфігураційним файлом (settings.xml) достатньо забезпечити можливість управління трьома параметрами, а решту можна підставляти статично (наприклад, зі заздалегідь підготовленого файла).
4. Для роботи з файлом елементів (images.xml, або menu.xml, або tagcloud.xml) flashxml-компонента дані бажано брати з БД (але необов'язково).
5. За необхідності – модифікувати структуру БД (наприклад, додати поле в таблицю).

Варіанти

1)	image sub-menu	2)	dock menu
3)	multilevel menu	4)	tag cloud
5)	accordion	6)	image-slider
7)	dropdown menu	8)	photo dragger
9)	3d carousel	10)	banner rotator

6.5. Контрольні запитання

1. Що таке XML?
2. Які переваги та недоліки XML?
3. Опишіть структуру XML.
4. Що таке XML схема?

Рекомендована література

1. Введення в CGI [Електронний ресурс]. – Режим доступу : http://www.intuit.ru/department/internet/cgi/1/1.html.

2. Веллинг Л. Розробка Web-приложений с помощью PHP і MySQL / Л. Веллинг. – 3-е изд. – Москва : ИД "Вильямс", 2008. – 880 с. [Электронный ресурс]. – Режим доступа : http://www.velling.ru.

3. Генерація HTTP запитів [Електронний ресурс]. – Режим доступу : http://www.codenet.ru/webmast/php/HTTP-POST.php.

4. Гутманс Е. PHP 5. Професійне програмування / Е. Гутманс [Електронний ресурс]– Режим доступу : http://docs.php.net.

5. Зервас К. Web 2.0: создание приложений на PHP / К. Зервас; пер. с англ. – Москва : ИД "Вильямс", 2010. – 544 с. [Электронный ресурс]. – Режим доступа : http://www.php.web2.0.pdo.php.

6. CGI крок за кроком [Електронний ресурс]. – Режим доступу : http://www.firststeps.ru/cgi/cgi1.html.

7. Керівництво по PHP [Електронний ресурс]. – Режим доступу : http://docs.php.net/manual/ru/index.php, http://www.php.ru/manual/.

8. Ловейн П. Объектно-ориентированное программирование на PHP5 / П. Ловейн. – Москва : НТ Пресс, 2007. – 224 с. [Электронный ресурс]. – Режим доступа : http://www.Loveine.ru.

9. PHP і HTTP – Робота з HTML-формами [Електронний ресурс]. – Режим доступу : http://php.su/phphttp/forms/.

10. PHP і Cookies [Електронний ресурс]. – Режим доступу : http://php.su/phphttp/?cookie.

11. PHP5/XML [Электронный ресурс]. – Режим доступа : http://phpclub.ru/faq/PHP5/XML.

12. Подводные камни использования сессий в PHP (2013) [Электронный ресурс]. – Режим доступа : https://habrahabr.ru/post/182352/.

13. Портал по PHP [Електронний ресурс]. – Режим доступу : http://php.su/. (Перелік функцій PHP за категоріями – http://php.su/functions/?page=cat).

14. Протокол передачі гіпертексту – HTTP/1.1 (оригінал – Hypertext Transfer Protocol -- HTTP/1.1) [Електронний ресурс]. – Режим доступу : https://tools.ietf.org/html/rfc2616.

15. Протокол передачи гипертекста – HTTP/1.1 [Электронный ресурс]. – Режим доступа : http://rfc2.ru/2068.rfc.

16. Протокол передачі гіпертексту версії 2 (HTTP/2) [Електронний ресурс]. – Режим доступу : https://tools.ietf.org/html/rfc7540.

17. Разъяснение http2 [Электронный ресурс]. – Режим доступа : https://bagder.gitbooks.io/http2-explained/content/ru/.

18. Робота з базами даних. Початок [Електронний ресурс]. – Режим доступу : http://www.php.su/articles/?cat=phpdb&page=002.

19. Сесії в PHP [Електронний ресурс]. – Режим доступу : http://www.php.su/articles/?cat=examples&page=070.

20. Сесії. Навчання і правильне використання [Електронний ресурс]. – Режим доступу : http://www.php.su/articles/?cat=protocols&page=009.

21. Стандарти оформлення коду PHP [Електронний ресурс]. – Режим доступу : http://www.opennet.ru/docs/RUS/php_code_standart/.

22. Функції СУБД MySQL [Електронний ресурс]. – Режим доступу : http://www.php.su/functions/?cat=mysql.

23. XML для PHP-розробників: Частина 1. Робота з XML в PHP за 15 хвилин [Електронний ресурс]. – Режим доступу : http://www.ibm.com/developerworks/ru/library/x-xmlphp1/.

24. XML: специфікація і функції DOM в PHP [Електронний ресурс]. – Режим доступу : http://php.su/articles/?cat=xml&page=002.

25. Шифлетт К. Правда о Сессиях / К. Шифлетт [Электронный ресурс]. – Режим доступа : http://phpclub.ru/detail/article/sessions.

26. Opera [Electronic resource]. – Access mode : https://ru.wikipedia.org/wiki/Opera.

27. PHPDoc and PHP IDE [Electronic resource]. – Access mode : https://docs.google.com/View?id=dchkhbfc_34zpkg3gq.

28. PHP: PDO – Manual [Electronic resource]. – Access mode : http://www.php.net/manual/en/book.pdo.php.

29. PHP Manual (en). XML Manipulation [Electronic resource]. – Access mode : http://www.php.net/manual/en/refs.xml.php.

30. Usage of HTTP/2 for websites [Electronic resource]. – Access mode : http://w3techs.com/technologies/details/ce-http2/all/all.

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Огурцов Віталій В'ячеславович
Гриньов Денис Валерійович

ВЕБ-ПРОГРАМУВАННЯ НА БОЦІ СЕРВЕРА ЗА ДОПОМОГОЮ МОВИ PHP

Лабораторний практикум
з навчальної дисципліни
"Веб-технології та веб-дизайн"
для студентів напряму підготовки
6.050101 "Комп'ютерні науки"

Самостійне електронне текстове мережеве видання

Відповідальний за видання *О. Г. Руденко*

Відповідальний редактор *М. М. Оленич*

Редактор *Н. І. Ганцевич*

Коректор *Т. А. Маркова*

План 2016 р. Поз. № 11-ЕНП. Обсяг 132 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***