МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

С. Б. Приходько,
А. В. Пухалевич, Л. М. Макарова

МЕТОДИЧНІ ВКАЗІВКИ ТА ЗАВДАННЯ

до виконання лабораторних робіт з дисципліни

«WEB-програмування»

Рекомендовано Методичною радою НУК

ВИДАВНИЦТВО
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КОРАБЛЕБУДУВАННЯ
ІМ. АДМІРАЛА МАКАРОВА

2025

УДК 004.77(076)
П75

Автори:
С. Б. Приходько, д-р техн. наук, професор;
А. В. Пухалевич, канд. техн. наук;
Л. М. Макарова, канд. техн. наук, доцент

Рецензент
М. В. Ушкац, д-р фіз.-мат. наук, професор

Рекомендовано Методичною радою НУК

П75 **Методичні** вказівки та завдання до виконання лабораторних робіт з дисципліни «WEB-програмування» / С. Б. Приходько, А. В. Пухалевич, Л. М. Макарова. – Миколаїв : НУК, 2025. – 52 с.

Методичні вказівки призначені для студентів третього курсу спеціальності 121 «Інженерія програмного забезпечення», які вивчають дисципліну «WEB-програмування». Також можуть бути корисними всім, хто вивчає HTML, CSS та JavaScript.

УДК 004.77(076)

ВСТУП

У Національному університеті кораблебудування імені адмірала Макарова (НУК) дисципліна «WEB-програмування» вивчається студентами спеціальності 121 «Інженерія програмного забезпечення» у п'ятому семестрі. Вона відноситься до циклу професійно-орієнтованих дисциплін навчального плану підготовки бакалаврів. Дані методичні вказівки мають допомогти студентам третього курсу зазначеної спеціальності у підготовці та виконанні лабораторних робіт з дисципліни «WEB-програмування». Вони містять завдання для лабораторних робіт, стислий виклад теоретичних відомостей, етапи виконання робіт, завдання для самостійної роботи, допоміжну літературу та питання для самоконтролю. При виконанні кожної лабораторної роботи рекомендується:

- засвоїти теоретичні відомості;
- розібратися з завданням та етапами виконання роботи;
- виконати запропоновані завдання й оформити звіт з роботи.

Лабораторну роботу необхідно виконувати відповідно за наведеними етапами. Звіт про лабораторну роботу оформлюється кожним студентом індивідуально. Звіт повинен містити: назву і мету роботи; завдання (постановку задачі); методику й алгоритм розв'язання задачі; текст програми; результати роботи; аналіз результатів і висновки. Текст програми бажано розміщувати у додатку.

ЛАБОРАТОРНА РОБОТА № 1
Створення HTML-шаблона вебсайту

Мета: вивчити основні HTML-теги, їх атрибути та навчитись створювати шаблони вебсторінок.

Завдання: використовуючи мову розмітки гіпертексту HTML5, створити шаблон сайту-блогу, який має семантично вірну розмітку і буде задовольняти наступним умовам:

1) повинен складатися з таких сторінок: головна сторінка зі списком публікацій блогу, сторінка перегляду публікації, сторінка з інформацією про сайт;

2) кожна сторінка має містити верхню частину (включає зображення-логотип, який є посиланням на головну сторінку, навігацію та назву сайту), нижню частину (включає поточний рік та інформацію про авторські права – прізвище та ім'я студента);

3) кожна публікація повинна мати назву, яка є посиланням на сторінку перегляду публікації, зображення, текст з декількох параграфів, дату публікації та ім'я автора;

4) шаблон не має містити будь які класи, стилі та сценарії;

5) файли шаблона повинні проходити валідацію на ресурсі https://validator.w3.org.

Зробити висновки щодо отриманих результатів.

Загальні теоретичні відомості

HTML (HyperText Markup Language) – мова розмітки гіпертексту, призначена для створення вебсторінок.

HTML-документ складається з тексту, який являє собою інформаційний вміст і спеціальних засобів мови HTML – тегів розмітки, які визначають структуру і зовнішній вигляд документа при його відображенні браузером.

Тег (html-тег, тег розмітки) – керуюча символьна послідовність, яка задає спосіб відображення гіпертекстової інформації.

HTML-тег складається з імені, за яким може слідувати необов'язковий список атрибутів. Весь тег (разом з атрибутами) береться в кутові дужки: `<імя_тега [атрибути]>`

Як правило, теги є парними і складаються з початкового та кінцевого тегів. Ім'я кінцевого тега збігається з ім'ям початкового, але перед ім'ям кінцевого тега ставиться коса риска (наприклад, <*html*> ... </*html*>). Кінцеві теги ніколи не містять атрибутів. Деякі теги не мають кінцевого елемента, наприклад тег <*img*>. Регістр символів для тегів не має значення, але прийнято використовувати нижній регістр.

Структура будь якого HTML-документа є такою:

1. Опис документа, який починається з вказівки його типу (секція DOCTYPE).
2. Текст документа, що поміщується в тег <html>.

Наявність секції DOCTYPE дозволяє вказати браузеру, яка версія HTML використовується в документі та які вимоги потрібно виконувати при обробці гіпертексту.

Текст документа (<*html*> ... </*html*>) є кореневим елементом документа. Всі інші елементи містяться всередині тегів <*html*> ... </*html*>. Все, що знаходиться за межами цих тегів, не сприймається браузером як код HTML і ніяк не обробляється. Текст документа складається з обов'язкових елементів – заголовка і тіла, які виділяються відповідно тегами <*head*> і <*body*>.

Заголовок документа (<*head*> ... </*head*>) містить технічну інформацію про сторінку: назва, опис, ключові слова для пошукових машин, кодування тощо. Ця інформація не відображається у вікні браузера, однак вказує браузеру, як слід обробляти сторінку. В розділі head обов'язково має бути присутнім тег *title*.

Обов'язковий елемент <*title*> ... </*title*> задає назву документа (його значення відображається в заголовку вікна браузера).

Елемент <*meta*> описує властивості документа (метадані). Призначення мета-тега визначається набором його атрибутів.

Приклад опису метаданих:

```
<meta charset="utf-8"> кодування сторінки
<meta name="description" content="рядок"> -
короткий опис сторінки
<meta name="keywords" content="рядок"> - список
ключових слів
```

```
<meta name="robots" content="noindex,nofollow">
- заборона індексування
<meta name="viewport" content="width=device-
width, initial-scale=1" /> - управління режимом
відображення на мобільних пристроях
```

В тілі документа (<*body*> ... </*body*>) знаходиться весь вміст документа, який буде відображено користувачам.

Теги в тілі документа поділяються на дві групи: блочні та рядкові.

Блочні теги займають всю доступну ширину, висота елемента визначається його вмістом, і він завжди починається з нового рядка. Всередині блочних тегів можна розміщувати інші блочні теги, а також рядкові теги.

Рядкові теги використовуються для зміни вигляду тексту або його логічного виділення. Вони розміщуються всередині інших елементів, наприклад, текстового абзацу. Рядкові теги не займають всю ширину батьківського елементу. Всередині рядкових тегів можна розміщувати інші рядкові теги, але блочні теги розміщувати не можна.

Перелік основних HTML тегів, що використовуються в тілі документа:

<*div*> ... </*div*> – контейнер загального призначення (структурний блок);

<*hN*> ... </*hN*> – заголовок *N*-ого рівня (*N* = 1 ... 6), наприклад <*h1*>;

<*p*> ... </*p*> – параграф;

... – гіперпосилання;

<*ol*> ... </*ol*> – нумерований список;

<*ul*> ... </*ul*> – маркований список;

<*li*> ... </*li*> – елемент нумерованого чи маркованого списку;

<*table*> ... </*table*> – контейнер таблиці;

<*tr*> ... </*tr*> – рядок таблиці;

<*td*> ... </*td*> – елемент таблиці;

<img src="..." alt="..."> – зображення;

<*strong*> ... </*strong*> – посилення;
<*span*> ... </*span*> – виділення;
<*em*> ... </*em*> – виділення;
<*br*> – примусовий розрив рядка.

У стандарті HTML5 додано нові семантичні елементи для структурування, групування і розмітки текстового вмісту. Додані елементи дозволили поліпшити структуру вебсторінки, додавши смислове значення вмісту таких елементів.

Елемент <*main*> – групує основний контент вебсторінки. Вміст елемента має бути унікальним на сторінці і не повинен відображатися де-небудь ще на сайті.

Елемент <*header*> – визначає вміст верхньої частини сторінки або її секції. Може містити основний заголовок, або групу заголовків (<h1> - <h6>). Тег <header> не можна поміщати всередину елементів <footer> або іншого елемента <header>.

Елемент <*footer*> – визначає вміст нижньої частини сторінки або її секції.

Елемент <*section*> – визначає розділ документа, як правило, з заголовком. Наприклад, розділами можуть бути глави, вкладки в діалоговому вікні тощо.

Елемент <*nav*> – являє собою розділ навігаційних посилань, що містить посилання на інші сторінки.

Елемент <*article*> – задає компонент сторінки, призначений для самостійного розповсюдження. Це може бути запис в блозі, повідомлення форуму, стаття тощо.

Елемент <*time datetime*="…"> – являє собою або час в 24-годинному форматі, або точну дату, яку при необхідності можна поєднувати з часом і зазначенням часового поясу. У елемента <time> дві частини: машинний шаблон часу (атрибут *datetime*); зрозумілий для людини текст.

Атрибути – це пари виду «властивість=значення», уточнюючі представлення відповідного тега:

```
<тег атрибут="значення"> ... </тег>
```

Атрибути можуть бути обов'язковими і необов'язковими. Необов'язкові атрибути можуть бути опущені, тоді для тега

застосовується значення цього атрибута за замовчуванням. Якщо не вказано обов'язковий атрибут, то вміст тега швидше за все буде відображено неправильно.

Крім тегів, у HTML-документах можуть бути присутні спеціальні символи. Для відображення спеціальних символів використовується мнемонічний або числовий код виду *&ім'я*; або *&#NNNN;*, де *NNNN* – код символу в Юнікоді в десятковій системі числення. Наприклад:

< – символ «менше» (<);

> – символ «більше» (>);

" – подвійні лапки (");

& – амперсанд (&);

* * – нерозривний пробіл.

Приклад семантичної розмітки HTML5:

```
<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="UTF-8">
   <title>Назва сторінки - Назва вебсайту</title>
</head>
<body>
   <header><!—Верхня частина сторінки-->
    <a   href="/"><img   src="logo.png"   alt=
"Логотип"></a>
    <nav>
        <ul>
          <li><a href="/">Головна</a></li>
          <li><a  href="/contact">Контакти</a>
</li>
        </ul>
    </nav>
    <h1>Назва сайту</h1>
   </header>
   <main><!--Основний вміст сторінки-->
    <article>
```

```
        <header><h2>Назва статі</h2></header>
        <div>
          <p>Текст статі</p>
          <p>Продовження тексту</p>
        </div>
        <footer>Додаткова інформація про статтю
</footer>
    </article>
   </main>
   <footer>© Інформація про авторські права вебсайту
</footer>
</body>
</html>
```

Гіперпосилання – це особливим чином позначений фрагмент вебсторінки (текст, зображення тощо), який пов'язаний з іншим документом. Для вказівки гіперпосилань використовується тег <a>. Гіперпосилання дозволяють переміщатися між пов'язаними вебсторінками.

```
<a  href="https://example.com/">Приклад</a>
<a     href="ftp://example.com/archive.tar.gz">
Завантажити файл</a>
<a href="mailto://user@example.com">Написати нам
</a>
<a href="tel://+380123456789">Зателефонувати нам
</a>
```

Посилання можуть бути абсолютними і відносними.

Абсолютні посилання вказують, як правило, на зовнішній ресурс. Для них потрібно вказувати повний шлях, включаючи назву домену:

Абсолютне посилання

Посилання на рисунок

Відносні посилання використовуються для переходу на внутрішні сторінки сайту. Для них потрібно вказувати тільки шлях, без назви домену:

```
<a href="/index.html">Посилання на сторінку в кореневому каталозі</a>
<a href="/help/index.html">Посилання на сторінку в підкаталозі кореневого каталогу</a>
<a href="../uploads/index.html">Посилання на сторінку в вищому каталозі</a>
```

Етапи виконання роботи

Виконання лабораторної роботи включає в себе наступні етапи.

1. Створити файли з розширенням *html* для головної сторінки, сторінки перегляду публікації, сторінки з інформацією про сайт (файл головної сторінки потрібно назвати *index.html*).
2. У створених файлах додати секцію DOCTYPE, теги html, head, title, body, header, main та footer.
3. У файлі головної сторінки, в елементі main, додати список публікацій: 2-3 елементи article, які містять теги header, h2, img, div, p, footer та time.
4. У файлі сторінки перегляду публікації, в елементі main, додати одну з публікацій з головної сторінки, використовуючи елемент article, який містить теги header, h1, img, div, p, footer та time.
5. У файлі сторінки з інформацією про сайт, в елементі main, додати відомості про призначення вебсайту, використовуючи елемент article, який містить теги header, h1, div, p та footer.
6. Провести валідацію html-файлів на ресурсі https://validator.w3.org. Виправити помилки, вказані валідатором при їх наявності.

Зробити висновки щодо отриманих результатів.

Список джерел

1. Структура HTML-сторінки: https://w3schoolsua.github.io/hyperskillua/34/html-page-structure.html.
2. HTML Елемент Head: https://w3schoolsua.github.io/html/html_head.html.
3. HTML5 Семантичні елементи: https://w3schoolsua.github.io/html/html5_semantic_elements.html.
4. HTML Зображення: https://w3schoolsua.github.io/html/html_images.html.

ЛАБОРАТОРНА РОБОТА № 2
Візуальне оформлення вебсайту на CSS

Мета: навчитись додавати візуальне оформлення до гіпертекстових документів з використанням каскадних таблиць стилів CSS.

Завдання: використовуючи каскадні таблиці стилів CSS, додати візуальне оформлення до шаблона сайту-блогу з лабораторної роботи № 1. Таблиця стилів має задовольняти наступним умовам:

1) не містити стилі, які не застосовуються в шаблоні;
2) повинна проходити валідацію на ресурсі https://jigsaw.w3.org/css-validator/;
3) повинна містити стилі для ширини вікна 768 і 320 точок (зменшені відступи між елементами);
4) назви класів мають відповідати логічному призначенню.

Зробити висновки щодо отриманих результатів.

Загальні теоретичні відомості

CSS дозволяє представляти один і той же документ у різних стилях або способах виведення, таких як екранне представлення, друк, читання голосом (спеціальним голосовим браузером або програмою читання з екрана), або при виведенні пристроями, що використовують шрифт Брайля.

Стиль – це сукупність правил, що застосовуються до елементу гіпертексту і визначають спосіб його відображення. Стиль включає всі типи елементів дизайну: шрифт, фон, текст, кольори посилань, поля і розташування об'єктів на сторінці.

Таблиця стилів – це сукупність стилів, які можна застосувати до гіпертекстових документів.

Таблиці стилів будуються відповідно до визначеного в стандарті синтаксису. Вони складаються з наступних частин:

```
селектор [, селектор [...]] {
   властивість1: значення1;
   [властивість2: значення2;]
```

```
    [властивість3: значення3;]
}
```

Селектор (Selector) – це елемент, до якого будуть застосовуватися стилі. Селектором може бути тег, клас або ідентифікатор об'єкта гіпертекстового документа.

Властивість (Property) – визначає одну або декілька характеристик селектора: відступи, шрифти, вирівнювання, розміри тощо.

Значення (Value) – це фактичні числові або строкові константи, що визначають властивість селектора.

Опис (Declaration) – сукупність властивостей та їх значень.

Правило (Rule) – повний опис стилю (селектор + опис).

Таким чином, таблиця стилів – це набір правил, які задають значення властивостей селекторів, перерахованих далі.

Деякі властивості CSS та їх можливі значення:

background	[background-color || background-image || background-repeat || background-attachment || background-position] | inherit	Елемент управління фоном
background-color	<color> | transparent | inherit	Колір фону
background-image	<uri> | none | inherit	Фонове зображення
background-position	[[<percentage> | <length>] {1,2} | [[top | center | bottom] || [left | center | right]]] | inherit	Положення фонового зображення
background-repeat	repeat | repeat-x | repeat-y | no-repeat | inherit	Повторення фонової зображення
border	[border-width || border-style || <color>] | inherit	Границі елемента
border-collapse	collapse | separate | inherit	Об'єднання/поділ суміжних границь
border-color	<color>{1,4} | transparent | inherit	Колір границі
border-style	<border-style>{1,4} | inherit	Стиль лінії границі

border-top border-right border-bottom border-left	[border-top-width || border-style || <color>] | inherit	Управління стилем заданої границі
border-width	<border-width>{1,4} | inherit	Товщина лінії границі
clear	none | left | right | both | inherit	Заборона заповнення вільного простору поруч із елементом
color	<color> | inherit	Колір тексту
display	inline | block | inline- block | table | table-row | table-cell | none | inherit | flex та інші	Спосіб відображення елемента
float	left | right | none | inherit	Плаваюче розміщення елемента
font	[[font-style || font-variant || font-weight]? font-size [/ line-height]? font-family] | inherit	Керування шрифтом
font-family	[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>] | inherit	Гарнітура шрифта
font-size	<absolute-size> | <relative-size> | <length> | <percentage> | inherit	Кегль шрифта
font-style	normal | italic | oblique | inherit	Стиль шрифта
font-weight	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit	Товщина шрифта
height	<length> | <percentage> | auto | inherit	Ширина елемента
line-height	normal | <number> | <length> | <percentage> | inherit	Висота рядка

list-style	[list-style-type || list-style-position || list-style-image] | inherit	Стиль списку
margin	<margin-width>{1,4} | inherit	Зовнішній відступ
margin-top margin-right margin-bottom margin-left	<margin-width> | inherit	Зовнішній відступ по заданій стороні
padding	<padding-width>{1,4} | inherit	Внутрішній відступ
padding-top padding-right padding-bottom padding-left	<padding-width> | auto | inherit	Внутрішній відступ по заданій стороні
position	static | relative | absolute | fixed | inherit	Позиціонування елемента
text-align	left | right | center | justify | <string> | inherit	Вирівнювання текстового блоку
text-decoration	none | [underline || overline || line-through || blink] | inherit	Текстові ефекти
text-indent	<length> | <percentage> | inherit	Абзацний відступ
text-transform	capitalize | uppercase | lowercase | none | inherit	Накреслення тексту
vertical-align	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit	Вертикальне вирівнювання в межах блоку
width	<length> | <percentage> | auto | inherit	Ширина елемента

Існує три способи додавання таблиці стилів до документа HTML:

Вбудовування (Inline) дозволяє застосувати стиль до заданого тегу HTML за допомогою атрибута style.

Впровадження (Embedded) дозволяє управляти стилями сторінки цілком. Для впроваджених стилів використовується тег <style> з таблицею стилів, який зазвичай розміщують у заголовку HTML-документа.

Зв'язування (Linked або External) дозволяє винести таблицю стилів у зовнішній файл (зазвичай, з розширенням .css), посилаючись на який за допомогою тега <link> можна змінювати відображення всіх сторінок сайту. Наприклад:

```
<link rel="stylesheet" href="sample.css">
```

Будь-яка вебсторінка, що містить такий зв'язок, буде оформлена відповідно до стилів, вказаних у файлі sample.css.

Для того, щоб шаблон зі стилями виглядав однаково у всіх браузерах, на початку таблиці стилів потрібно додавати «скидання» стилів за замовчуванням. Наприклад:

```
* {
   margin: 0;
   padding: 0;
   box-sizing: border-box;
}
body {
   font-family: Tahoma, Verdana, sans-serif;
   font-size: 16px;
   line-height: 1.4;
}
img {
   border: none;
   max-width: 100%;
}
```

Таблиці стилів можуть застосовуватися для управління відображенням вмісту в залежності від використовуваного пристрою виведення (монітор, проєктор, пристрій друку тощо). Для цього в опис стилів включити тип пристрою, наприклад так:

```
@media print {/* принтер * /
     body {font-size: 10pt; }
}
```

```
@media screen {/* монітор * /
     body {font-size: 12pt; }
}
```

За допомогою CSS медіа-запитів можливо також додати стилі, які будуть застосовуватися тільки при виконанні певних умов (наприклад, при певній ширині, висоті вікна, екрана або орієнтації пристрою). Для цього в тег *head* вебсторінки потрібно додати тег *<meta name="viewport" content="width=device-width">*, а в стилі включити умови їх застосування. Наприклад:

```
/*  -----  Вікно шириною менше ніж 1023
---------- */
@media only screen and (max-width: 1023px) {
   /* CSS-правила */
}
/*  -----  Вікно шириною менше ніж 767
---------- */
@media only screen and (max-width: 767px) {
   /* CSS-правила */
}
```

Етапи виконання роботи

Виконання лабораторної роботи включає в себе наступні етапи.

1. До всіх сторінок лабораторної роботи № 1 підключити зовнішній файл зі стилями (файл можна назвати *styles.css*).
2. До html-тегів додати класи з назвами, які будуть відповідати логічному призначенню тегів (наприклад, *site-header*, *site-footer*, *site-navigation*, *blog-article*).
3. У файл зі стилями додати правила, що дозволять сторінкам виглядати згідно з дизайном, що відповідає номеру варіанта.
4. Перевірити вигляд сторінок в емуляторі для ширини вікна 768 і 320 точок. Зменшити відступи там, де це необхідно при ширині вікна 768 і 320 точок.

5. Провести валідацію файлу зі стилями на ресурсі https://jigsaw.w3.org/css-validator/. Виправити помилки, вказані валідатором при їх наявності.

Зробити висновки щодо отриманих результатів.

Список джерел

1. Метатег Viewport: https://www.w3schools.com/css/css_rwd_viewport.asp.
2. Медіазапити: https://www.w3schools.com/css/css_rwd_mediaqueries.asp.

ЛАБОРАТОРНА РОБОТА № 3
Створення динамічних вебсторінок на PHP

Мета: вивчити синтаксис, типи даних, оператори та управляючі конструкції мови PHP. Навчитися створювати користувацькі функції.

Завдання: перевести з HTML на PHP шаблон сайту-блогу з лабораторної роботи № 2:

1) змінити розширення файлів з html на php і змінити посилання у файлах;

2) використовуючи оператори включення, перемістити в окремі файли:

– вміст тегу head;
– заголовок сайту;
– підвал сайту;
– публікації (вміст тегів article);

3) в окремому файлі створити функцію, яка буде повертати список публікацій – масив, кожен елемент якого є окремою публікацією (вкладеним масивом з ключами id, title, content, author, image, created);

4) дані, що повертає створена функція, використати на сторінці зі списком публікацій і сторінці перегляду публікації;

5) рік у підвалі сайту має генеруватися на php.

Зробити висновки щодо отриманих результатів.

Загальні теоретичні відомості

PHP вбудовується в HTML-код за допомогою тегів <?php ?>, скорочений варіант для виведення тексту <?= ?>:

```
<?= 'Hello, World!'; ?>
```

Вирази *require*, *include*, *require_once* та *include_once* дозволяють включати код, що міститься у вказаному файлі, і виконувати його. Включений файл успадковує область видимості змінних рядка, де він з'явився. У випадку помилок *include* видає попередження, і робота скрипта триває, а помилка в *require*

викликає фатальну помилку роботи скрипта і припиняє його виконання. Аналогічно працюють вирази *include_once* та *require_once* з тією лише різницею, що якщо файл уже один раз був включений, він не буде включений і виконаний повторно, і поверне вираз *true*.

Змінна у PHP позначається знаком долара, за яким слідує її ім'я. Наприклад: $myVariable. Імена змінних чутливі до регістру. Імена змінних, як і інші найменування в PHP, відповідають наступним правилам: починаються з букви або символу підкреслення, з подальшими, у будь-якій кількості, літерами, цифрами або символами підкреслення. Приклад присвоєння та виведення значення змінної:

```
<?php
   $first = 'Text'; //Надаємо $first значення
'Text'
   echo "Змінна з ім'ям first дорівнює " .
$first;
?>
```

PHP підтримує 10 простих типів даних. Скалярні типи: *boolean* (логічний тип – *true* або *false*); *integer* (цілі числа, розмір яких залежить від платформи); *float* (число з плаваючою крапкою); *string* (рядки). Змішані типи: *array* (масиви); *object* (об'єкти); *callable* (функції зворотного виклику); *iterable*. Спеціальні типи: *resource* (ресурси); *null* (порожній тип).

Рядок – це набір символів. На довжину рядків не існує обмежень. Рядок може бути визначений різними способами: за допомогою одинарних чи подвійних лапок, heredoc-синтаксисом, nowdoc-синтаксисом.

Якщо рядок помістити у подвійні лапки, то PHP розпізнає більшу кількість управляючих послідовностей: \" – подвійні лапки; \\ – зворотний слеш; *n* – новий рядок (LF або 0x0A (10) в ASCII); *r* – повернення каретки (CR або 0x0D (13) в ASCII); *t* – горизонтальна табуляція (HT або 0x09 (9) в ASCII); *$* – знак долара.

В рядках у подвійних лапках виконується обробка змінних. Якщо інтерпретатор зустрічає знак долара ($), він захоплює стільки символів, скільки можливо, щоб сформувати правильне ім'я змінної. Щоб точно визначити кінець імені, ім'я змінної потрібно вкласти у фігурні дужки. Приклад:

```
<?php
   $name = 'Іван';
   echo "Мене звати {$name}"; //Це виведе "Мене звати Іван"
?>
```

Масиви (array) – це впорядковане відображення, яке встановлює відповідність між значенням і ключем. Ключ може бути або типу *integer*, або типу *string*. Значення може бути будь-якого типу. Цей тип може використовуватися як власне масив, список, хеш-таблиця, словник, колекція, стек, черга. Так як значенням масиву може бути інший масив, можна також створювати дерева і багатовимірні масиви. Масив може бути створений мовною конструкцією *array*() або []. Як параметри вона приймає будь-яку кількість розділених комами значень чи пар ключ => значення:

```
$a = ['value', 'key2' => 'value2', 'key3' => $name, 'value4'];
```

Доступ до елементів масиву може бути здійснений за допомогою ключа, вказаного в квадратних дужках. При додаванні нового елемента індекс можна не вказувати, тоді буде автоматично присвоєно індекси, починаючи з 0.

```
<?php
   $names = [];
   $names[] = 'Апельсин';
   $names[2] = 'Груша'; //Індекс 1 пропущено
   $names['Іванов'] = 'Іван'; //Індексом є рядок
```

```
   echo $names[0]; //Друк елемента масиву names
з індексом 0
   echo $names[2]; //Друк елемента масиву names
з індексом 2
   echo   $names['Іванов'];   //Друк   елемента
з індексом 'Іванов'
?>
```

Приклад створення масиву із даними публікації та виведення цих даних разом з додаванням html-тегів:

```
<?php
   $article   =   ['id'=>1,   'title'=>'Article
1', 'content'=>'<p>Line 1</p><p>Line 2</p>',
'author'=>'Sergiy',     'created'=>'2021-01-01
01:02:03'];
   $url  =  "article.php?id={$article['id']}";
// id публікації використовується як параметр
адреси
?>
<article>
   <header>
    <h1><a    href="    <?=    $url;    ?>"><?=
$article['title']; ?></a></h1>
   </header>
   <div><?= $article['content']; ?></div>
   <footer>
    <div>Published:     <time     datetime=<?=
$article['created']; ?>><?= $article['created'];
?></time></div>
    <div>Author: <?= $article['author']; ?></
div>
   </footer>
</article>
```

У PHP існує кілька конструкцій, що дозволяють виконувати повторювані дії в залежності від умови. Це цикли *while*, *do .. while*, *foreach*. Конструкція *foreach* призначена виключно для роботи з масивами, її синтаксис:

```
foreach ($array as $value) {
   блок_виконання
}
або
foreach ($array as $key => $value) {
   блок_виконання
}
```

Виконання *блоку_виконання* відбувається стільки разів, скільки елементів у масиві *$array*. Усередині *блоку_виконання* значення поточного елемента масиву може бути отримано за допомогою змінної *$value*, а ключ поточного елемента масиву – за допомогою змінної *$key*

Приклад створення багатовимірного масиву зі списком публікацій і виведення даних з цього використанням циклу *foreach*:

```
<?php
$articles = [
   ['id'=>1,        'title'=>'Article        1',
'content'=>'<p>Line    1</p><p>Line    2</p>',
'author'=>'Sergiy',      'created'=>'2021-01-01
01:02:03'],
   ['id'=>2,        'title'=>'Article        2',
'content'=>'<p>Line    1</p><p>Line    2</p>',
'author'=>'Andrii',      'created'=>'2022-02-02
15:20:30'],
];
?>
<?php foreach ($articles as $article) { ?>
   <?php    require    'inc/article.php';    //
Виводиться елемент article ?>
<?php } ?>
```

У наведеному прикладі html публікації винесено окремий файл *inc/article.php*, а в циклі відбувається включення цього файлу.

Для доступу до даних запитів HTTP GET використовується суперглобальна змінна *$_GET*.

Приклад доступу та обробки ідентифікатора публікації, переданого в адресі як article.php?*id=1*:

```
<?php
   $articleId = $_GET['id'] ?? null;
   if (null === $articleId) { //Параметр
в адресі відсутній
    http_response_code(400);
    exit();
   }

   $articleId = (int)$articleId;
   $article = getArticle($articleId); //Пошук
публікації
   if (null === $article) { //Публікацію не
знайдено
    http_response_code(404);
    exit();
   }

   //Змінна $article містить публікацію
з заданим ідентифікатором
?>
```

Встановлення інтерпретатора PHP в операційних системах сімейства unix можна зробити з використанням менеджерів пакетів, а PHP для ОС Windows можна скачати за адресою https://windows.php.net/download/.

Для того, щоб обробляти HTTP запити, в операційній системі повинен бути встановлений та запущений вебсервер. В операційних системах сімейства unix можна встановити вебсервер

Apache, а на ОС Windows можна використовувати вебсервер, вбудований у PHP. Вбудований вебсервер запускається командою, якій необхідно вказати інтерфейс, порт прослуховування та шлях до каталогу з php-файлами (webroot). Наприклад: *php.exe -S localhost:8080 –t D:\lab-web\webroot*. Також може використовуватись вебсервер, вбудований в IDE. Перевірити, що вебсервер коректно налаштований і запущений, можна відкривши відповідну адресу у браузері, наприклад http://localhost:8080/index.php.

Етапи виконання роботи

Виконання лабораторної роботи включає в себе наступні етапи.

1. Змінити розширення фалів з лабораторної роботи № 2 з *html* на *php*, та відповідно змінити адреси в атрибутах *href* посилань у цих файлах.

2. Використовуючи оператор require, перемістити в окремий файл вміст тегу *head* (файл можна назвати *inc/head.php*). Вміст тегу *title* виводити зі змінної, об'явленої в основному файлі (змінну можна назвати *$pageTitle*).

3. Використовуючи оператор require, перемістити в окремий файл заголовок сайту (файл можна назвати *inc/header.php*). За допомогою умовного оператора та змінної, об'явленої в основному файлі, забезпечити використання для назви сайту тегу *h1* на головній сторінці та тегу *div* на інших сторінках (змінну можна назвати *$currentPage*). Аналогічно забезпечити виділення поточної сторінки в навігації.

4. Використовуючи оператор require, перемістити в окремий файл підвал сайту (файл можна назвати *inc/footer.php*). Замінити статичне значення року в підвалі сайту на значення, що генерується за допомогою вбудованої функції *date*().

5. Створити файл для розміщення функцій і налаштувань вебсайту (файл можна назвати *inc/functions.php*). Використовуючи оператор *require_once*, підключити створений файл на початку всіх сторінок вебсайту.

6. У файлі для функцій створити функцію, яка буде повертати список публікацій (функцію можна назвати *getArticles(): array*). Дані, що повертає функція, використати на сторінці зі списком публікацій.

7. У файлі для функцій створити функцію, яка буде повертати публікацію по її ідентифікатору (функцію можна назвати *getArticle(int $articleId): ?array*). Дані, що повертає функція, використати на сторінці перегляду публікації.

8. Використовуючи оператор require, перемістити в окремий файл код публікації (файл можна назвати *inc/article.php*). Параметр *id* публікацій використовувати як частину посилання на сторінку перегляду цієї публікації. Забезпечити використання для назви основної публікації тегу *h2* на головній сторінці та тегу *h1* на інших сторінках.

9. На сторінці перегляду публікації використовувати переданий в адресі параметр *id* як ідентифікатор публікації, яку необхідно відобразити. Якщо публікація не знайдена, то потрібно повернути HTTP статус *404* і зупинити виконання скрипта (потрібно використовувати вбудовані функції *http_response_code()* та *exit()*).

Зробити висновки щодо отриманих результатів.

Список джерел

1. Оператор require: https://www.php.net/manual/uk/function.require.php.
2. Цикли foreach: https://www.php.net/manual/uk/control-structures.foreach.php.
3. Функція date: https://www.php.net/manual/uk/function.date.php.
4. Опис параметру format для date: https://www.php.net/manual/uk/datetime.format.php.
5. Функція http_response_code: https://www.php.net/manual/uk/function.http-response-code.php.

ЛАБОРАТОРНА РОБОТА № 4
Робота з базами даних на PHP

Мета: навчитись працювати з базами даних на PHP.

Завдання: підключити шаблон сайту-блогу з лабораторної роботи № 3 до бази даних:

1) додати функцію, яка повертатиме підключення до бази даних;

2) додати php-файл, в якому буде відбуватися ініціалізація бази даних, виконати ініціалізацію;

3) публікації отримувати через запит до бази даних.

Зробити висновки щодо отриманих результатів.

Загальні теоретичні відомості

Модуль *Об'єкти даних PHP* (PHP Data Objects – PDO) визначає простий та узгоджений інтерфейс для доступу до баз даних у PHP. Кожен драйвер бази даних, у якому реалізований цей інтерфейс, може надати специфічну для бази даних функціональність у вигляді стандартних функцій модуля.

PDO забезпечує абстракцію доступу до даних, це означає, що незалежно від того, яка конкретна база даних використовується, можна користуватися одними і тими ж функціями для виконання запитів та вибірки даних. PDO не абстрагує саму базу даних, цей модуль не переписує SQL-запити і не емулює відсутній СУБД функціонал. Якщо саме це потрібно, необхідно скористатися повноцінною абстракцією бази даних.

Для доступу до всіх СУБД, у драйверах яких реалізований інтерфейс PDO (в тому числі MySQL, SQLite, PostgreSQL, Microsoft SQL Server), використовується клас *PDO*. Принцип роботи класу *PDO* полягає у створенні екземпляру класу та виклику його методів для виконання запитів.

Метод ***PDO::__construct()*** створює екземпляр об'єкта PDO

```
public PDO::__construct(string $dsn, ?string
$username = null, ?string $password = null,
?array $options = null): PDO
```

dsn – ім'я джерела даних або DSN, що містить інформацію, необхідну для підключення до бази даних. Загалом, DSN складається з імені драйвера PDO, за яким слідує двокрапка та специфічний синтаксис підключення драйвера PDO. Додаткову інформацію можна отримати з розділу Документація щодо специфічних драйверів PDO документації PHP.

Параметр *dsn* підтримує три різні методи вказівки аргументів, необхідних для створення з'єднання з базою даних:

1. Виклик драйвера – *dsn* містить повний DSN.
2. Виклик URI – параметр *dsn* складається з рядка uri: з наступним URI, який визначає розташування файлу, що містить рядок DSN. URI може вказувати на локальний файл або віддалений URL-адресу. Наприклад: uri:file:///path/to/dsnfile.
3. Поєднання імен – *dsn* складається з імені name, яке відповідає параметру *pdo.dsn.name* у php.ini, що визначає рядок DSN.

username – ім'я користувача для рядка DSN. Цей параметр опціональний для деяких драйверів PDO.

passwd – пароль для рядка DSN. Цей параметр опціональний для деяких драйверів PDO.

options – масив ключ=>значення специфічних для драйвера налаштувань підключення.

PDO::__construct() викидає *PDOException*, якщо спроба підключення до бази даних завершується з помилкою.

Приклад підключення до бази даних MySQL:

```
try {
     $connection = new PDO('mysql:host=localho
st;dbname=blog', 'dbuser', 'dbpass');
} catch (PDOException $e) {
     echo "Error!: {$e->getMessage()}";
     die;
}
```

Приклад підключення до бази даних SQLite:

```
$connection = new PDO('sqlite:blog.sqlite');
```

Метод ***PDO::exec()*** виконує SQL-запит, який не потребує повернення даних, і повертає кількість задіяних рядків:

```
public PDO::exec(string $statement): int|false
```

statement – SQL-вираз, який необхідно підготувати та запустити (зазвичай запит INSERT, UPDATE або DELETE). Дані всередині запиту мають бути правильно екрановані.

PDO::exec() повертає кількість рядків, які були модифіковані або видалені під час виконання. Якщо таких рядків немає, PDO::exec() поверне *0*.

Метод ***PDO::query()*** готує та виконує вираз SQL за один виклик функції.

```
public PDO::query(string $query, ?int $fetchMode
= null): PDOStatement|false
```

query – SQL-запит для підготовки та виконання.

fetchMode – режим вибірки за замовчуванням для повернутого *PDOStatement*. Має бути однією з констант *PDO::FETCH_**, наприклад *PDO::FETCH_ASSOC*.

PDO::query() повертає *false* при виникненні помилок. При успішному виконанні повертається об'єкт *PDOStatement*, який можна ітерувати за допомогою циклу *foreach*, чи викликати методи *PDOStatement::fetch()* або *PDOStatement::fetchAll()*.

Якщо SQL містить заповнювачі (певні значення, отримані від користувача), слід використовувати *PDO::prepare()* і *PDOStatement::execute()*. Як альтернативу SQL можна підготувати вручну перед викликом *PDO::query()*, при цьому дані повинні бути правильно відформатовані з використанням *PDO::quote()*, якщо це підтримується драйвером.

Якщо запит запускатиметься багаторазово, для покращення продуктивності запит краще один раз підготувати *PDOStatement* методом *PDO::prepare()*, а потім виконувати його методом *PDOStatement::execute()* стільки разів, скільки потрібно.

Метод ***PDO::quote()*** вміщує рядок у лапки (якщо потрібно) і екранує спеціальні символи всередині рядка відповідним для

драйвера способом. Повертає екранований рядок, який теоретично безпечно використовувати в тілі запиту SQL. Повертає *false*, якщо драйвер не підтримує екранування.

```
public PDO::quote(string $string, int $type =
PDO::PARAM_STR): string|false
```

string – рядок, що екранується.

type – надає підказку про тип даних для драйверів, які мають альтернативні способи екранування. Наприклад, *PDO_PARAM_LOB* вкаже драйверу екранувати двійкові дані.

Метод ***PDO::lastInsertId()*** повертає ID останнього вставленого рядка.

```
public PDO::lastInsertId(?string $name = null):
string|false
```

Метод ***PDO::prepare()*** готує запит до виконання та повертає пов'язаний з цим запитом об'єкт.

```
public PDO::prepare(string $query, array
$options = []): PDOStatement|false
```

query – SQL-запит для підготовки. Запит може містити іменовані (:*name*) або неіменовані (*?*) псевдозмінні, які будуть замінені на реальні значення під час запуску запиту на виконання. Використовувати одночасно іменовані і неіменовані псевдозмінні в одному запиті не можна. Використовуйте псевдозмінні, щоб прив'язати до запиту дані, отримані від користувача. Не включайте дані, введені користувачем, безпосередньо в запит.

options – масив, що містить одну або більше пар ключ => значення для встановлення значень атрибутів об'єкта *PDOStatement*, який буде повернутий з цього методу.

Якщо СУБД успішно підготувала запит, *PDO::prepare()* повертає об'єкт *PDOStatement*. Якщо підготувати запит не вдалося, *PDO::prepare()* повертає false або викидає виключення *PDOException* (залежить від режиму обробки помилок).

Метод ***PDOStatement::execute()*** запускає підготовлений запит *PDOStatement* на виконання.

```
public PDOStatement::execute(?array $params =
null): bool
```

params – масив, що має містити стільки елементів, скільки псевдозмінних заявлено в SQL-запиті. Усі значення будуть прийняті як такі, що мають тип *PDO::PARAM_STR*.

Після успішного запуску об'єкт *PDOStatement* можна ітерувати за допомогою циклу *foreach*, чи викликати на ньому методи *PDOStatement::fetch()* або *PDOStatement::fetchAll()*.

Метод ***PDOStatement::fetch()*** вибирає наступний рядок з набору результатів.

```
public PDOStatement::fetch(int $mode =
PDO::FETCH_DEFAULT): mixed
```

Метод ***PDOStatement::fetchAll()*** вибирає всі рядки, що залишилися, з набору результатів.

```
public PDOStatement::fetchAll(int $mode =
PDO::FETCH_DEFAULT): array
```

Важливо, щоб при виникненні помилок, сайт повертав відповідний HTTP код у заголовках (наприклад *404 Not Found*, *500 Internal Server Error*). **HTTP заголовки можливо відправити тільки до того, як були виведені будь-які дані або теги**, тому підключення та запити до БД необхідно виконати до виведення <!DOCTYPE html>.

Етапи виконання роботи

1. Додати функцію, яка повертатиме підключення до бази даних (функцію можна назвати *getDbConnection(): PDO*).
2. Додати php-файл, в якому буде відбуватися ініціалізація бази даних (файл можна назвати *init.php*). У цьому файлі потрібно виконати SQL-запит на створення таблиці articles з полями *id*, *title*, *content*, *author*, *image*, *created* (можна використовувати

метод *PDO::exec()*). Приклад SQL-запиту для створення таблиці з публікаціями (СУБД SQLite3):

```
$query = 'CREATE TABLE `articles` (
   `id`   INTEGER   NOT   NULL   PRIMARY   KEY
AUTOINCREMENT,
   `title` TEXT NOT NULL,
   `content` TEXT NOT NULL,
   `author` TEXT NOT NULL,
   `image` TEXT NOT NULL,
   `created` TEXT NOT NULL
)';
```

3. Модифікувати функцію *getArticles()* так, щоб список публікацій отримувався шляхом виконання запиту до бази даних (можна використовувати метод *PDO::query()* та цикл *while* чи функцію *iterator_to_array()*).

4. Модифікувати функцію *getArticles()* так, щоб публікацію з заданим ідентифікатором отримувати шляхом виконання запиту до бази даних (можна використовувати метод *PDO::query()* або підготовлений вираз – prepared statement, використовуючи методи *PDO::prepare()* та *PDOStatement::execute()*). Запит має містити частину *WHERE*.

Зробити висновки щодо отриманих результатів.

Список джерел

1. Клас PDO: https://www.php.net/manual/uk/class.pdo.php.
2. Метод PDO::prepare: https://www.php.net/manual/uk/pdo.prepare.php.
3. Цикли while: https://www.php.net/manual/uk/control-structures.while.php.

ЛАБОРАТОРНА РОБОТА № 5
Створення HTML-форм та їх обробка на PHP

Мета: навчитись створювати HTML-форми та обробляти отримані з них дані.

Завдання: реалізувати додавання коментарів до сайту-блогу з лабораторної роботи № 4:

1) на сторінці перегляду публікації зробити можливість додавання коментаря через форму з трьох полів: ім'я автора коментаря (однорядкове текстове поле), оцінка публікації (випадаючий список з числами від 1 до 5), текст коментаря (багаторядкове текстове поле);

2) на сервері виконати валідацію отриманих даних (довжина імені автора коментаря має бути довжиною від 1 до 50 символів, довжина тексту коментаря має бути довжиною від 1 до 200 символів, оцінка – ціле число від 1 до 5);

3) коментарі зберігати в таблицю comments (id, article_id, author, rate, content, created);

4) на сторінці перегляду публікації, під формою, відобразити раніше додані коментарі;

5) на сторінці зі списком публікацій біля кожної публікації вивести кількість коментарів та середню оцінку публікації.

Зробити висновки щодо отриманих результатів.

Загальні теоретичні відомості

Елемент HTML <form> задає розділ документа, що містить інтерактивні елементи управління, які дозволяють користувачеві відправляти інформацію на вебсервер. Обов'язковий атрибут *action* задає URI-адресу сервера, який обробляє інформацію, передану через форму. Атрибут *method* задає HTTP метод, який браузер використовує для відправки форми. Можливі значення: *get* (при відправці форми дані додаються до URI атрибута action), *post* (дані з форми включаються в тіло форми і надсилаються на сервер методом HTTP POST).

Елемент HTML <input> використовується для створення інтерактивних елементів управління. Атрибут *name* задає ім'я поля, щоб на сервері було можливо визначити, якому полю належать відправлені дані. Атрибут *value* задає значення поля за замовчуванням. Тип елемента <input> задається атрибутом *type*. Основними значеннями є: *text* (однорядкове текстове поле); *hidden* (елемент, який не відображається, але значення якого відправляється серверу); *number* (поле для введення числа з плаваючою крапкою); *password* (однорядкове текстове поле, значення якого приховано); *checkbox* (прапорець, атрибут *checked* вказує, чи повинен прапорець бути виставлений); *radio* (радіокнопка), *submit* (кнопка, яка відправляє форму, текст на кнопці задається атрибутом value).

HTML-елемент <select> задає випадаючий список. Для визначення пункту списку використовується HTML-елемент <option>. У пункті відображається текстовий вміст елемента <option>, а атрибут *value* задає значення, що відправляється формою, якщо вибраний даний пункт. Атрибут *selected* задає пункт списку, що є вибраним за замовчуванням.

Елемент HTML <textarea> являє собою багаторядкове текстове поле, яке дозволяє користувачам вводити значну кількість тексту, наприклад коментар до форми огляду або зворотного зв'язку. Значення за замовчуванням вказується між відкриваючим і закриваючим тегами.

HTML-елемент <label> являє собою підпис до елемента управління. Підпис можна пов'язати з елементом управління, розмістивши елемент управління всередині <label>, або додаючи атрибут *for*, що дорівнює атрибуту *id* елемента управління.

Приклад форми з інтерактивними елементами управління та підписами до них:

```
<form action="" method="post">
   <input      type="hidden"      name="action"
value="new-comment">
```

```
    <div class="input">
     <label>
         Name:<br>
         <input type="text" name="author">
     </label>
    </div>

    <div class="input">
     <label>
         Article rate:<br>
         <select name="rate">
           <option>Select rate</option>
           <option value="5">Rate 5</option>
           <option value="4">Rate 4</option>
           <option value="3">Rate 3</option>
           <option value="2">Rate 2</option>
           <option value="1">Rate 1</option>
         </select>
     </label>
    </div>

    <div class="input">
     <label>
         Comment:<br>
         <textarea   name="content"   cols="60"
rows="5"></textarea>
     </label>
    </div>

    <div class="submit">
     <input type="submit" value="Send comment">
    </div>
</form>
```

На PHP отримати доступ до даних, відправлених з вебформ, можливо, використовуючи відповідні суперглобальні змінні *$_GET* та *$_POST*. Інформація з адресної строки веббраузера, в тому числі дані форм, відправлених методом HTTP GET, доступна на PHP через суперглобальну змінну *$_GET*. Дані форм, відправлених методом HTTP POST, доступні на PHP через суперглобальну змінну *$_POST*.

Приклад обробки даних вебформи, надісланої методом HTTP POST:

```
<?php
$articleId = $_GET['article'] ?? null;
...
$action = $_POST['action'] ?? null;
$errors = [];
if ('new-comment' === $action) {
   $author = trim((string)($_POST['author'] ??
null));
   if ('' === $author) {
    $errors['author'] = 'This field is can not be
empty';
   } elseif (mb_strlen($author) > 50) {
    $errors['author'] = 'Length can not be more
than 50 characters';
   }

   $rate = (int)($_POST['rate'] ?? null);
   if ($rate < 1 || $rate > 5) {
    $errors['rate'] = 'Invalid rate';
   }

   $content = trim((string)($_POST['content']
?? null));
   if ('' === $content) {
    $errors['content'] = 'This field is can not be
empty';
```

```
    } elseif (mb_strlen($content) > 200) {
      $errors['content'] = 'Length can not be
more than 200 characters';
    }

    if (0 === count($errors)) {
      //No errors
      //TODO Insert comment into database
      //Redirect user to current page using HTTP GET
      header("Location:      article.php?article=
{$articleId}");
      exit;
    }
}
?>
```

У прикладі вище, при відсутності помилок у введених даних, користувач перенаправляється на поточну сторінку, щоб перезавантажити її методом HTTP GET.

Якщо ж введені дані містять помилки, замість перенаправлення потрібно вивести інформацію про помилки біля відповідних інтерактивних елементів форми. При цьому інтерактивним елементам потрібно встановити значення за замовчуванням, щоб користувачеві не довелось заново заповнювати форму. **Валідація на вебсервері даних, отриманих від користувача, є обов'язковою**, так як користувач, використовуючи інструменти розробника в браузері, може змінити атрибути елементів управління (в тому числі імена, значення тощо).

Приклад виведення інформації про помилки та встановлення значень:

```
<div class="input">
    <label>
      Name:<br>
      <input type="text" name="author"
      value="<?=          htmlspecialchars($_POST
['author']??''); ?>">
```

```
    </label>
    <?php     if     (array_key_exists('author',
$errors)) { ?>
      <div class="error"><?= $errors['author'];
?></div>
    <?php } ?>
</div>
```

При відображенні будь-яких даних, прийнятих з форми чи адресного рядка, обов'язково потрібно використовувати функцію *htmlspecialchars()*, яка забезпечує правильне кодування «особливих» HTML-символів, щоб запобігти впровадженню зловмисного HTML або Javascript на вебсторінку (англ. Cross Site Scripting – XSS).

При використанні даних, отриманих від користувача, для запитів у базу даних, обов'язково потрібно виконувати їх екранування, використовуючи підготовлені вирази (prepared statements), або за допомогою відповідних функцій, наприклад *PDO::quote()*, *SQLite3::escapeString()* або *mysqli_real_escape_string()*, щоб запобігти впровадженню і виконанню довільного sql-запиту (англ. SQL injection).

Приклад збереження даних, отриманих від користувача, в базу даних:

```
$connection = getDbConnection();
$query = '
    INSERT INTO comments (article_id, rate,
content, author, created)
    VALUES (:articleId, :rate, :content, :author,
:created)
    ';
$statement = $connection->prepare($query);
$data = ['articleId'=>$articleId, 'rate'=>$rate,
'content'=>$content, 'author'=>$author];
$data['created'] = date('Y-m-d H:i:s');
$result = $statement->execute($data);
```

```
if (false === $result) {
   http_response_code(500);
   exit();
}
```

Етапи виконання роботи

1. На сторінці перегляду публікації додати HTML-форму (елемент *form[method=post]*) з трьома елементами управління: ім'я автора коментаря (елемент *input[type=text]*), оцінка публікації (число від 1 до 5, елемент *select*), текст коментаря (елемент *textarea*). Також додати на форму прихований елемент (*input[type=hidden]*) для подальшої ідентифікації надсилання форми на PHP.

2. На сторінці перегляду публікації (перед виведенням HTML-тегів) виконати ідентифікацію надсилання форми і валідацію отриманих даних. Виявлені помилки зберегти в масив *$errors*, ключі якого відповідатимуть назвам елементів управління.

3. Якщо введені дані містять помилки, вивести інформацію про помилки, збережену у масиві *$errors*, біля відповідних інтерактивних елементів форми. Інтерактивним елементам потрібно встановити значення за замовчуванням, що відповідають попереднім введеним даним.

4. У php-файлі з кодом ініціалізації бази даних (*init.php*) додати виконання SQL-запиту на створення таблиці *comments* з полями *id*, *article_id*, *author*, *rate*, *content*, *created* (можна використовувати метод *PDO::exec()*). Приклад SQL-запиту для створення таблиці з коментарями (СУБД SQLite3):

```
$query = 'CREATE TABLE `comments` (
   `id`   INTEGER   NOT   NULL   PRIMARY   KEY
AUTOINCREMENT,
   `article_id` INTEGER NOT NULL,
   `author` TEXT NOT NULL,
   `rate` INTEGER NOT NULL,
   `content` TEXT NOT NULL,
   `created` TEXT NOT NULL
)';
```

5. При відсутності помилок у введених даних (масив $errors порожній), зберегти надісланий коментар у базу даних, використовуючи методи *PDO::prepare()* та *PDOStatement::execute()*. Перенаправити користувача на поточну сторінку методом HTTP GET, використовуючи функцію *header*() з заголовком *Location*.

6. На сторінці перегляду публікації (перед виведенням HTML-тегів) завантажити з бази даних список раніше доданих коментарів до публікації. Під формою відобразити раніше додані коментарі, екрануючи текст коментаря функцією *htmlspecialchars()* та відобразивши розриви рядків функцією *nl2br()*.

7. Модифікувати функцію *getArticles()*, щоб вона повертала кількість коментарів і середню оцінку публікації (в SQL запит потрібно додати *JOIN* та *GROUP BY articles.id*, а також додати функціональні вирази *COUNT(comments.id)* comments_count, *AVG(comments.rate) avg_rate* в частині *SELECT*).

8. На сторінці зі списком публікацій біля кожної публікації вивести кількість коментарів та середню оцінку публікації, які повертатимуться функцією *getArticles()*.

Список джерел

1. HTML форми: https://www.w3schools.com/html/html_forms.asp.
2. Елементи HTML-форм: https://w3schools.com/html/html_form_elements.asp.
3. Атрибути елементу input: https://w3schools.com/html/html_form_attributes.asp.
4. Функція mb_strlen: https://www.php.net/manual/uk/function.mb-strlen.php.
5. Фунція htmlspecialchars: https://www.php.net/manual/uk/function.htmlspecialchars.
6. Функція nl2br: https://www.php.net/manual/uk/function.nl2br.php.
7. Функція header: https://www.php.net/manual/uk/function.header.php.

ЛАБОРАТОРНА РОБОТА № 6
Реалізація асинхронної взаємодії браузера з вебсервером

Мета: навчитись відправляти та обробляти ajax-запити.

Завдання: реалізувати додавання коментарів до сайту-блогу без перезавантаження сторінки:

1) на сторінці перегляду публікації, після відправки форми з коментарем, надсилати дані на сервер з допомогою технології ajax, замість повного перезавантаження сторінки;

2) після додавання нового коментаря список коментарів повинен оновитися;

3) коментування повинне працювати при відключенні Javascript у браузері.

Зробити висновки щодо отриманих результатів.

Загальні теоретичні відомості

JavaScript – це мова програмування, що використовується в складі HTML-сторінок для збільшення їх функціональності та можливостей взаємодії з користувачем. JavaScript є однією із складових динамічного HTML.

Виконання програми JavaScript відбувається при перегляді HTML або ініціюється діями користувача.

Скрипт може бути пов'язаний з HTML-сторінкою двома способами:

1) за допомогою парного тегу *script*;

2) як оброблювач події, що стосується конкретного тегу HTML.

Сценарій, вбудований в HTML-сторінку з використанням тегу *script*, має наступний формат:

```
<script>
     alert('Hello world!'); //Код програми
</script>
```

Для додавання оброблювачів подій у теги деяких елементів введені атрибути обробки подій. Ім'я атрибуту обробки події

починається з префікса on, за яким йде назва події. Наприклад, події натискання кнопки миші click відповідає параметр обробки події з назвою onclick.

Все, що розміщується між тегами *script*, інтерпретується як код програми на мові JavaScript. Атрибут src дозволяє задати зовнішній файл з кодом сценарію:

```
<script src="functions.js"></script>
```

JavaScript підтримує чотири простих типи даних: ціле число, число з плаваючою крапкою, рядок ("рядок" або 'рядок'), логічний (*true*/*false*).

Також існують спеціальні типи даних: null, об'єкт (object), функція (function).

Імена змінних повинні містити тільки букви латинського алфавіту, цифри, символ підкреслення _, знак долара $, та починатись з букви або символу підкреслення _ або знаку долара $. Імена змінних чутливі до регістру. Заборонено використовувати імена, що збігаються з ключовими словами JavaScript.

Визначити змінну можливо:

1) операторами *var* та *let*;

2) при ініціалізації, за допомогою оператору присвоєння (=), наприклад, b=126.

JavaScript – слабо типізована мова. Тип змінної залежить від того, який тип інформації в ній зберігається. В декларації змінної тип не вказується, і надалі змінній можна надавати значення будь-яких типів. Інтерпретатор JavaScript сам виконує автоматичне перетворення типів даних у міру необхідності.

Функція JavaScript – це іменована група команд, які вирішують певну задачу та можуть повернути деяке значення. Функція визначається за допомогою оператору *function*, що має такий синтаксис:

```
function Ім'я_функції([параметри]) {
   [оператори]
   return [значення_що повертається];
}
```

У JavaScript існує два способи створення об'єктів – за допомогою оператора *new*, за яким іде функція конструктор, або за допомогою літерала об'єкта:

```
var empty = new Object(); //Об'єкт створюється оператором new
var empty = {}; //Об'єкт створюється за допомогою літерала об'єкта
```

Літерал об'єкта – це поміщений у фігурні дужки список з нуля або більше властивостей (пари ім'я: значення), що розділені комами. Ім'ям властивості може бути ідентифікатор або рядковий літерал. Значенням властивості може будь-яке значення примітивного типу або типу посилального, а також будь-який JavaScript-вираз:

```
var empty ={}; //Порожній об'єкт (без властивостей)
var point ={x: 0,y: 0}; //Об'єкт з двома властивостями x і y
var homer ={ //Об'єкт з декількома властивостями
   name: "Гомер Сімпсон",
   age: 34,
   married: true
};
```

Зазвичай для доступу до значень властивостей об'єкта використовується оператор крапка. Властивості об'єкта працюють як змінні: в них можна зберігати значення і зчитувати їх:

```
homer.age = 35; //надаємо властивості нове значення
document.write(homer.age+'<br>'); //виведення значення властивості
homer.male = 'чоловік'; //додаємо в об'єкт нову властивість
document.write(homer.male) //виведення значення нової властивості
```

В ядрі JavaScript визначені об'єкти, які можливо використовувати не використовуючи контекст завантаженої сторінки. До основних об'єктів відносяться: Array, Date, Math, String.

Array – масив. Масив – це упорядкований набір однотипних даних, до елементів якого можна звернутись по імені або по індексу. Для створення масиву необхідно використати одну із наступних конструкцій:

```
ім'я_масиву = [[елемент1],[елемент2],[елемент3],
...];
ім'я_масиву = new Array([елемент1],[елемент2],
[елемент3],...);
ім'я_масиву = new Array([довжина масиву]);
```

Наприклад:

```
ar0 = [1, 2, 3];
ar1 = new Array(1, 2, 3);
ar2 = new Array(3);
```

Перший елемент масиву має номер 0.

Для доступу до значень елементів масиву в квадратних дужках біля імені масиву необхідно вказати порядковий номер елементу. Наприклад:

```
a = ar1[2];
ar1[0] = 7;
```

Об'єкт ***Date*** використовується для роботи з датами.

Об'єкт ***Math*** дозволяє використовувати вбудовані в JavaScript математичні функції та константи. При зверненні до методів та властивостей цього об'єкта створювати його не потрібно, але необхідно явно вказувати його ім'я.

Об'єкт ***String*** використовується для роботи з рядковими типами даних.

На додаток до стандартних об'єктів JavaScript існує декілька функцій, для виклику яких не потрібно створювати об'єкти.

До цих функцій належать: parseFloat(параметр) та parseInt(параметр). Також досить часто використовуються функції Number(об'єкт) та String(об'єкт), які перетворюють об'єкт, що використовується як параметр, у число або рядок.

Об'єкт ***window*** створюється автоматично при запуску браузера. Крім того, нове вікно можливо створити і засобами JavaScript. Для цього необхідно використати метод open. Для закриття вікна браузера використовується метод close. Наприклад, для закриття вікна попереднього прикладу необхідно:

При зверненні до методів та властивостей вікна браузера, в якому знаходиться програма JavaScript, ключове слово window можна не вказувати. Наприклад, закрити поточне вікно браузера можна так:

```
close()
```

Розглянемо деякі методи об'єкта window.

Метод **alert()** викликає вікно сповіщення, яке містить текст повідомлення і клавішу ОК. Приклад:

```
alert('Вітаю, це сповіщення');
```

Метод **setInterval()** активує періодичне виконання деяких команд з заданим інтервалом. Приклад:

```
var id = setInterval('Код_JavaScript',
інтервал_часу);
```

Метод **clearInterval()** припиняє повторне виконання коду заданого setInterval(). Приклад:

```
clearInterval(id);
```

Метод **setTimeout()** активує виконання деяких команд після закінчення встановленого терміну часу. Приклад:

```
var id = setTimeout('Код_JavaScript',
інтервал_часу);
```

Метод **clearTimeout()** відміняє заплановане методом setTimeout() виконання коду. Приклад:

```
clearTimeout(id);
```

Метод **confirm()** Викликає вікно підтвердження, що містить текст повідомлення і клавіші ОК і Відміна. Повертає true або false. Приклад:

```
var x = confirm('Натисніть ОК або Отмена.');
```

Метод **prompt()** викликає вікно запиту, спонукаюче відвідувача ввести в нього певні дані. Приклад:

```
var x = prompt('Введіть Ваше ім'я:','Ім'я');
```

Метод **scrollBy()** прокручує вміст вікна на вказану кількість пікселів. Приклад:

```
scrollBy(0,100);
```

Метод **scrollTo()** прокручує вміст вікна до вказаних координат. Приклад:

```
scrollTo(0,960);
```

Об'єкт ***document*** містить інформацію про завантажену сторінку. Всі елементи HTML-сторінки є властивостями цього об'єкта.

Стандарт DOM передбачає декілька засобів пошуку елементу. Це методи getElementById, getElementsByTagName, getElementsByName, а також querySelector і querySelectorAll.

Важливою ознакою інтерактивних HTML-сторінок є можливість реакції на дії користувача. В JavaScript інтерактивність реалізована за допомогою перехоплення та обробки подій, викликаних у результаті дій користувача. Розширені способи пошуку, фільтрації елементів та маніпуляцій над ними, а також додавання обробки подій пропонують JavaScript-бібліотеки.

jQuery – бібліотека JavaScript, яка дозволяє отримувати доступ до будь-якого елементу моделі DOM, звертатися до атрибутів і вмісту, а також змінювати їх. Крім цього, бібліотека надає зручний API для роботи з технологією AJAX.

Бібліотеку jQuery можна скачати з сайту http://jquery.com/.

Перед початком роботи бібліотеку треба підключити на вебсторінці:

```
<script src="jquery.js"> </script>
```

Вся робота з бібліотекою ведеться з використанням функції $. Загальна ідея використання jQuery:

1) вибирається елемент або група елементів;
2) виконуються дії над вибраними елементами.

Вибір елементів заснований на синтаксисі CSS. Наприклад:

1. $('p a') – всі посилання, розташовані в абзацах.
2. $('#myId') – вибір елемента з указаним ідентифікатором.
3. $('myClass') – використання класів.
4. $('body> div') – вибір елементів div, є прямими нащадками елемента body.
5. $('p:even') – парні абзаци.
6. $('#site-nav li:first-child') – перший пункт головного меню сайту.
7. Використання значень атрибутів.

$('a[href^=http://]') – вибір посилань, адреси яких починаються з символів http://, на це вказує символ ^.

$('a[href$=.pdf]') – посилання, адреси яких закінчуються на «pdf».

$('a[href*=youtube]') – посилання, адреси яких містять youtube в довільному місці.

Методи jQuery дозволяють маніпулювати властивостями, атрибутами, стилями і вмістом елементів. Приклад:

```
$('.page-header').text('New     title');     //
Встановлюємо текстовий вміст
$('.result').html('<strong>0<strong>     items
found'); //Встановлюємо html вміст
$('#main-menu  li').removeClass('hidden');  //
Видаляємо клас
$('#main-menu          li:last-child').addClass
('hidden'); //Додаємо клас
$('#b3').toggleClass('hidden',  a  >  0);  //
Перемикаємо клас по умові
$('#b3').css('display',  'none');  //Змінюємо
стилі
```

```
$('#b3').css({'background-color':      'black',
'color': white'});
$('#b4').attr('title',    'New    item');    //
Встановлюємо атрибут
$('#b4').removeAttr('title'); //Видаляємо атрибут
```

Приклад встановлення властивості *title* з номером у всіх елементів з класом b2:

```
$('.b2').each(function (n) { $(this).attr('title',
'New ' + n); })
```

Тут використаний метод *each*, який виконує обхід усіх елементів у наборі і викликає для них функцію. Як параметр функції передається індекс елемента в наборі, а сам елемент є доступним через *this*.

Для встановлення обробника подій використовується метод *on*.

Приклад функції, яка буде спрацьовувати при кліку на будь-якому рисунку:

```
$('img').on('click', function() {
   alert('На рисунку зображено ' + $(this).
attr('alt'));
});
```

Для того, щоб виконувати HTTP запити та завантажувати потрібні дані без перезавантаження вебсторінки, використовується технологія ***AJAX*** (Asynchronous JavaScript and XML).

Запити AJAX можна робити за допомогою об'єкта XMLHttpRequest, або за допомогою методів бібліотеки jQuery. При цьому у запитах буде присутній HTTP заголовок *X-Requested-With* зі значенням: *XMLHttpRequest*. Наявність вказаного заголовку дозволяє ідентифікувати запити AJAX на сервері й обробляти їх особливим чином.

Наприклад, на PHP такі запити можна відслідковувати, перевіривши наявність і значення вказаного HTTP заголовку в суперглобальній змінній $_SERVER.

```
$isAjax = 'xmlhttprequest' === strtolower
($_SERVER['HTTP_X_REQUESTED_WITH'] ?? '');
```

Бібліотека jQuery дозволяє виконувати запити AJAX за допомогою методів $.get(), $.post(), $.ajax(), $.getJSON(), $.getScript().

Наприклад, щоб не було перезавантаження сторінки при відправці форми, треба додати обробник події відправки форми та надіслати дані форми запитом AJAX:

```
$('#comments-form').on('submit', function() {
   var $form = $(this);
   jQuery.post($form.prop('action'), $form.
serialize(), function(response) {
    //Обробка асинхронно отриманої відповіді
від сервера
   });

   return false; //Не відправляти форму
звичайним способом
});
```

Етапи виконання роботи

1. В кінці сторінки перегляду публікації додати код на мові JavaScript з використанням бібліотеки jQuery, який буде обробляти подію *submit* у форми для додавання коментаря, і надіслати дані форми на сервер з допомогою технології ajax, замість повного перезавантаження сторінки.

2. В коді на PHP, який обробляє відправлені форми коментарів, додати перевірку, що запит було надіслано з допомогою AJAX. В таких випадках у відповіді сервера повертати HTML лише для оновленого списку коментарів (якщо всі дані форми

пройшли валідацію). Або ж повертати HTML лише для форми із відображеними повідомленнями про помилки біля полів, які не пройшли валідацію.

3. Додати обробку асинхронно отриманої відповіді від сервера, щоб повернений HTML встановлювався на списку коментарів, або ж на власне формі.

4. Вимкнути JavaScript у браузері і перевірити, чи працює після цього форма.

Список джерел

1. Лазарович І.М. Конспект лекцій з дисципліни «Програмування та підтримка веб-застосувань» для студентів напряму підготовки «Інформатика». Івано-Франківськ : Вид-во Прикарпатського національного університету ім. Василя Стефаника, 2015. 153 с.
2. Метод on() бібліотеки jQuery: https://api.jquery.com/on/.
3. Метод serialize() бібліотеки jQuery:https://api.jquery.com/serialize/.
4. Метод jQuery.get ()бібліотеки jQuery:https://api.jquery.com/jQuery.get/.
5. Метод jQuery.post() бібліотеки jQuery:https://api.jquery.com/jQuery.post/.

ЗМІСТ

Навчальне видання

ПРИХОДЬКО Сергій Борисович
ПУХАЛЕВИЧ Андрій Володимирович
МАКАРОВА Лідія Миколаївна

МЕТОДИЧНІ ВКАЗІВКИ ТА ЗАВДАННЯ
до виконання лабораторних робіт з дисципліни
«WEB-програмування»

Комп'ютерне верстання *Н. М. Ковальчук*
Коректор *О. Є. Вакула*

Формат 60×84/16. Ум. друк. арк. 3,02. Вид. № 14. Зам. № 0604-16.
Видавець і виготівник Національний університет кораблебудування
імені адмірала Макарова
просп. Героїв України, 9, м. Миколаїв, 54007
E-mail : publishing@nuos.edu.ua
Свідоцтво суб'єкта видавничої справи ДК № 6402 від 19.09.2018 р.