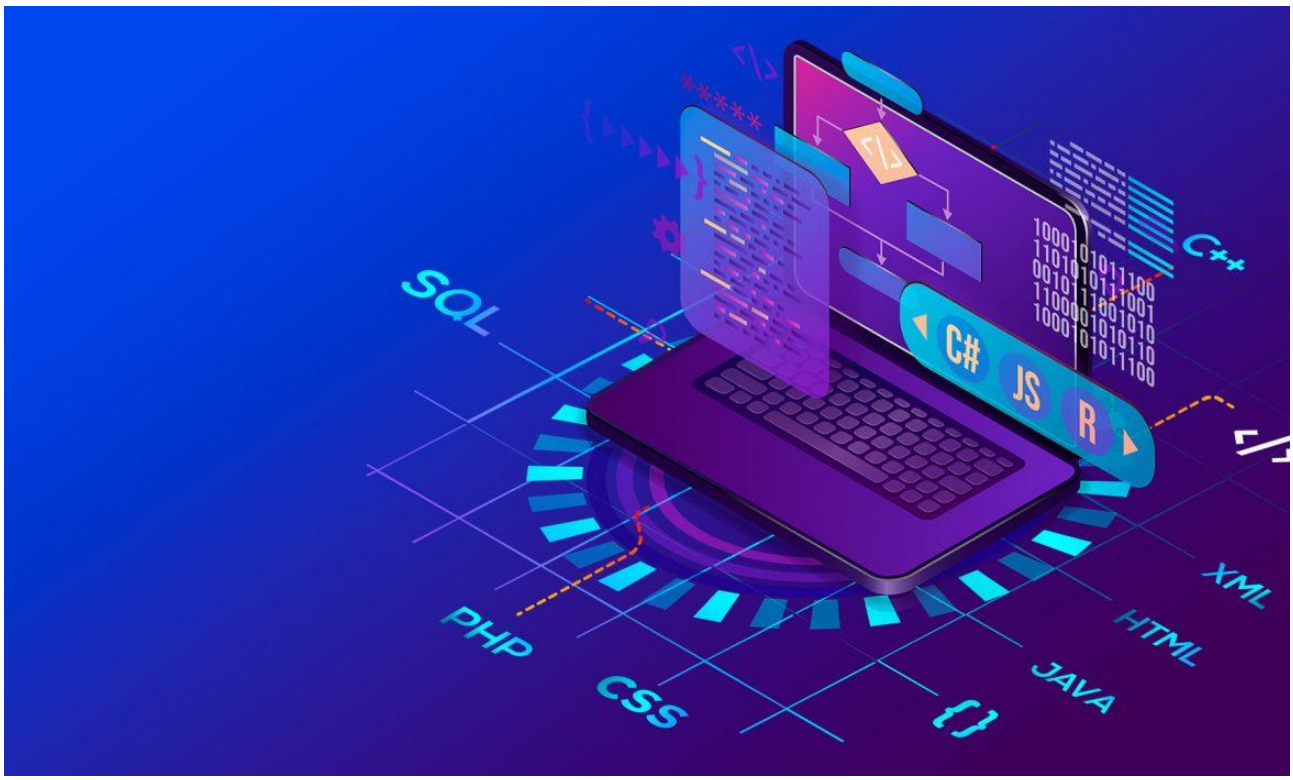
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

ФАКУЛЬТЕТ МЕНЕДЖМЕНТУ
КАФЕДРА ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ, КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ВЕБТЕХНОЛОГІЇ ТА ВЕБДИЗАЙН

Методичні рекомендації до виконання лабораторних робіт
для здобувачів першого (бакалаврського) рівня вищої освіти ОПП
«Комп'ютерні науки» спеціальності F3 (122) Комп'ютерні науки
денної форми здобуття вищої освіти



Миколаїв
2025

УДК 004.777
В26

Друкується за рішенням науково-методичної комісії факультету менеджменту Миколаївського національного аграрного університету (протокол № 1 від 28 серпня 2025 року)

Укладачі:

С. І. Тищенко – в.о. завідувача кафедри, к.п.н., доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. Ю. Пархоменко – к.ф.-м.н., доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

С. І. Ємельянов – PhD, старший викладач кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

Т.С.Кучмійова – к.ф.-м.н., доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. О. Жебко – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

О. Є. Богатєнкова – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету;

А.М. Коломієць – асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету

Рецензенти:

Р.В. Кураченко – Head o Production / Learning Experience at Interaction Design Foundation (IxDF)

О. С. Садовий - канд. техн. наук, доцент, завідувач кафедри агроінженерії Миколаївського національного аграрного університету

ЗМІСТ

ПЕРЕДМОВА

Навчальна дисципліна «Вебтехнології та вебдизайн» призначена для забезпечення здобувачів вищої освіти за спеціальністю F3 (122) «Комп'ютерні науки» фундаментальними теоретичними знаннями та практичними навичками у сфері проєктування, розробки, розгортання та супроводу сучасних вебзастосунків. Курс покликаний ознайомити майбутніх фахівців з архітектурою клієнт-серверної взаємодії, методологіями створення інтерактивних інтерфейсів, базовими технологіями фронтенд- та бекенд-розробки, а також сформувати глибоке розуміння принципів UI/UX дизайну в цифровій індустрії.

Предметом вивчення дисципліни є принципи побудови та функціонування вебзастосунків, методи семантичної розмітки й стилізації вебдокументів, технології алгоритмічного програмування на стороні клієнта (JavaScript) та сервера (PHP), механізми взаємодії з реляційними базами даних (MySQL), а також сучасні підходи до вебдизайну й забезпечення адаптивності.

Об'єктом вивчення дисципліни є клієнт-серверна архітектура, життєвий цикл розробки вебпроєктів, об'єктна модель документа (DOM), серверні технології обробки запитів, бази даних для вебу, принципи чуйного вебдизайну (Responsive Web Design) та інструментальні засоби проєктування інтерфейсів.

Метою є формування у здобувачів вищої освіти цілісного інженерно-системного світогляду в галузі веброзробки. Освітня компонента спрямована на розвиток здатності до проєктування архітектури вебзастосунків та обґрунтованого вибору оптимальних технологічних стеків (Frontend та Backend) для вирішення комплексних фахових задач. Окрема увага приділяється формуванню глибокого розуміння принципів юзабіліті, стандартів вебдоступності (a11y) та фундаментальних аспектів кібербезпеки у сучасній вебінфраструктурі.

Основними завданнями є:

- засвоєння базових концепцій, класифікації та багаторівневої архітектури вебзастосунків, а також принципів функціонування мережевих протоколів HTTP/HTTPS;
- набуття практичних навичок розробки семантичної, адаптивної та кросбраузерної верстки із застосуванням стандартів HTML5 та CSS3;
- ознайомлення з парадигмами клієнтського програмування мовою JavaScript для реалізації динамічних інтерфейсів, маніпуляцій об'єктною моделлю документа (DOM) та організації асинхронної взаємодії з сервером (AJAX, Fetch API);
- формування вмінь розробки серверної логіки з використанням мови PHP, налаштування маршрутизації та проєктування безпечної взаємодії з

реляційними базами даних (MySQL);

- вивчення сучасних підходів до проєктування користувацьких інтерфейсів (UX/UI) та набуття навичок роботи з відповідними інструментами (зокрема, Figma) для створення ергономічних та користувачоорієнтованих рішень;
- опанування методів забезпечення якості програмного продукту, що включає тестування, оптимізацію продуктивності, реалізацію базових механізмів захисту від типових вразливостей, а також процедур розгортання (деплою) готових проєктів на віддаленому сервері (хостингу).

ЛАБОРАТОРНА РОБОТА № 1

Тема: Основи HTML5. Структурування контенту, робота з текстом, списками, посиланнями та мультимедіа.

Мета: Ознайомитися з базовою структурою HTML-документа. Набути практичних навичок використання основних тегів для розмітки текстової інформації, створення списків, гіперпосилань та вбудовування зображень. Навчитися працювати з редактором коду (VS Code), структурувати файли проєкту та публікувати код на платформі GitHub.

Теоретичні відомості:

HTML (HyperText Markup Language) – це стандартизована мова розмітки, яка використовується для створення структури вебсторінок. HTML-документ складається з елементів (тегів), які вказують браузеру, як саме відображати контент: як заголовок, абзац, посилання чи зображення.

Сучасний стандарт HTML5 базується на принципах семантичної розмітки. Це означає, що теги повинні описувати сенс контенту, а не його зовнішній вигляд.

Обов'язкова структура сучасного HTML-документа визначається кореневим елементом <html>, який об'єднує дві основні частини: контейнер <head> та контейнер <body>. Декларація <!DOCTYPE html> передує кореневому елементу та ідентифікує версію HTML5, забезпечуючи коректне відображення сторінки браузером. У розділі <head> розміщуються метадані, зокрема інформація про кодування символів, заголовок документа, а також посилання на зовнішні ресурси (стилі, скрипти); цей вміст не візуалізується безпосередньо у вікні браузера.

Розділ <body> охоплює весь видимий контент і структурується за допомогою семантичних елементів, таких як <header> (верхній колонтитул), <main> (основна зона вмісту), <section> (логічно завершені розділи) та <footer> (нижній колонтитул), що сприяє формуванню чіткої інформаційної архітектури вебсторінки.

Завдання до виконання:

Крок 1. Підготовка робочого середовища

Першочергово необхідно завантажити та встановити редактор коду Visual Studio Code (VS Code), якщо він ще не присутній у системі. Після встановлення слід додати корисні розширення, зокрема Live Server, яке дозволяє автоматично оновлювати сторінку в браузері при збереженні коду, та Prettier – Code formatter для автоматичного вирівнювання і форматування програмного коду.

Крок 2. Ініціалізація проєкту

На комп'ютері створюється нова папка з назвою web-lab-1-Прізвище (використовується латиниця). Цю папку слід відкрити у VS Code через меню File -> Open Folder. Усередині папки створюється файл index.html, який є головним файлом будь-якого вебсайту. Додатково створюється підпапка images, до якої поміщається власне фото або будь-яке інше зображення, що виконуватиме роль аватара.

Крок 3. Створення базового каркаса (Boilerplate)

Відкривши файл index.html, за допомогою вбудованого інструменту Emmet (введення символу ! і натискання Tab або Enter) автоматично генерується базова структура HTML-документа. Далі необхідно змінити атрибут мови в тегу <html> на українську: <html lang=«uk»>. У тегу <title>, розташованому всередині <head>, вказується заголовок «Резюме - Ваше Прізвище».

Крок 4. Семантична розмітка сторінки

Перейшовши всередину тегу <body>, створюються три головні семантичні блоки: <header>, <main> та <footer>. Усі подальші дії з наповнення контентом виконуються всередині цих блоків. Для зручності розробки запускається Live Server (кнопка «Go Live» у нижньому правому куті VS Code), що дає змогу бачити зміни в реальному часі.

Крок 5. Наповнення шапки сайту (<header>)

У блок <header> додається головний заголовок сторінки <h1>, у якому зазначається ім'я та прізвище студента. Під заголовком розміщується абзац <p> із короткою інформацією (наприклад, спеціальність і курс). Також додається фотографія за допомогою тегу <img>: в атрибуті src прописується відносний шлях до файлу (наприклад, images/photo.jpg), а в атрибуті alt обов'язково надається опис, наприклад «Фото студента Ваше Прізвище».

Крок 6. Наповнення основного контенту (<main>)

У межах блоку <main> контент розбивається на логічні секції за допомогою тегу <section>. Для кожної секції додається відповідний заголовок <h2>.

Перша секція, «Про мене», містить кілька абзаців тексту <p>, у яких використовуються теги форматування: для виділення важливого тексту жирним шрифтом, для курсиву та для примусового переносу рядка.

Друга секція, «Мої навички» або «Хобі», включає маркований список <ul> із трьома-п'ятьма пунктами <li> (наприклад, HTML, базові навички роботи з ПК, англійська мова).

Третя секція, «Освіта» або «Етапи проєкту», оформлюється як нумерований список <ol> із кількома елементами <li>, щоб продемонструвати

автоматичну нумерацію.

Четверта секція, «Контакти та соціальні мережі», містить гіперпосилання <a> на профілі у соцмережах, GitHub або Telegram; для кожного посилання в атрибуті href зазначається повна URL-адреса, а за допомогою атрибута target=«_blank» забезпечується відкриття посилання у новій вкладці.

Крок 7. Оформлення підвалу (<footer>)

У блок <footer> додається текстовий абзац <p>, що містить знак копірайту (спецсимвол ©), поточний рік та прізвище студента, наприклад: © 2025 Іван Іваненко. Усі права захищені.

Крок 8. Збереження та відправка на GitHub

Після завершення верстки слід переконатися, що код відформатовано рівномірно та відсутні незакриті теги. На обліковому записі GitHub створюється новий публічний репозиторій, наприклад, з назвою web-technologies-labs. За допомогою терміналу або програми GitHub Desktop виконується комміт файлів та їх відправлення (push) до віддаленого репозиторію.

Крок 9. Оформлення звіту

Створюється файл звіту у форматі Word або Google Docs, заповнюється титульний аркуш, зазначаються тема та мета роботи. До звіту додаються скриншоти написаного коду та результату його виконання у браузері, а також посилання на створений репозиторій у GitHub. Готовий звіт зберігається у форматі .pdf для подальшого подання викладачеві.

Контрольні питання для захисту:

1. Поняття семантичної розмітки та приклади семантичних тегів, використаних у роботі, з поясненням їхнього призначення.
2. Призначення тегів форматування тексту , , та спецсимволів (наприклад, ©).
3. Структура та відмінності між нумерованими (<ol>) і маркованими (<ul>) списками.
4. Різниця між абсолютними та відносними посиланнями (шляхами до файлів) на прикладі атрибута src тегу <img>.
5. Призначення декларації <!DOCTYPE html> та структура HTML-документа (елементи <html>, <head>, <body>).

ЛАБОРАТОРНА РОБОТА № 2

Тема: Форми та таблиці в HTML. Створення інтерфейсів введення даних.

Мета: Навчитися створювати таблиці для структурування інформації та

розробляти інтерактивні вебформи для збору даних від користувачів. Опанувати використання сучасних типів полів введення HTML5, семантичних зв'язків між елементами форми та базових атрибутів валідації.

Теоретичні відомості:

Таблиці в HTML використовуються виключно для представлення табличних даних (розклади, звіти, прайс-листи), а не для побудови макета сторінки. Створення таблиці починається з парного тегу <table>. Рядки визначаються тегом <tr>, заголовки стовпців – <th>, а звичайні комірки з даними – <td>. Для об'єднання комірок по горизонталі або вертикалі використовуються атрибути colspan та rowspan.

Форми є головним інструментом взаємодії користувача з сервером. Уся форма обгортається в тег <form>, який має два ключових атрибути: action (вказує адресу скрипта на сервері, який буде обробляти дані) та method (визначає HTTP-метод передачі, зазвичай GET або POST). Основним елементом форми є тег <input>, зовнішній вигляд та поведінка якого кардинально змінюється залежно від атрибута type (text, password, email, radio, checkbox тощо). Для пояснення призначення поля використовується тег <label>, який зв'язується з <input> за допомогою атрибутів for та id. Також форми містять випадаючі списки (<select>), багаторядкові текстові поля (<textarea>) та кнопки відправки (<button type=«submit»>).

Завдання до виконання:

Крок 1. Підготовка робочого середовища

Необхідно створити нову папку з назвою web-lab-2-Прізвище та відкрити її у VS Code. У цій папці слід створити файл index.html та згенерувати базову HTML5-структуру за допомогою Emmet, ввівши знак оклику та натиснувши Tab. Далі потрібно встановити мову документа lang="uk" та змінити заголовок <title> на «Лабораторна робота 2». У тілі документа (<body>) необхідно створити два семантичні розділи <section>: один для таблиці, інший для форми, після чого запустити Live Server.

Крок 2. Створення таблиці (Розклад занять)

У першій секції слід створити заголовок другого рівня з текстом «Розклад занять» та додати тег <table>. Для візуального виділення меж таблиці тимчасово додається атрибут border="1". Далі створюється рядок заголовків <tr>, у якому розміщуються комірки <th> з назвами днів тижня. Після цього додаються кілька рядків <tr> для навчальних пар, у яких розташовуються комірки <td> з назвами дисциплін. Нарешті, застосовується об'єднання комірок: знаходяться дві однакові пари, що йдуть поспіль, і об'єднуються по вертикалі

(rowspan="2") або горизонталі (colspan="2"), після чого видаляється зайва комірка в наступному рядку/стовпці, щоб структура таблиці залишалася коректною.

Крок 3. Створення основи форми реєстрації

У другій секції необхідно створити заголовок другого рівня з текстом «Форма реєстрації студента» та додати тег <form>. Для цієї форми вказуються атрибути action="#" (дані надсилатимуться на поточну сторінку) та method="POST".

Крок 4. Додавання текстових полів та паролів

Для введення імені додається тег <label> з текстом «Ім'я:» та атрибутом for="firstName", а також тег <input> з type="text", id="firstName", name="first_name" та атрибутом required, що робить поле обов'язковим для заповнення. Аналогічно створюється поле для електронної пошти з використанням типу email, а також поле для пароля з типом password, яке приховує введені символи.

Крок 5. Додавання елементів вибору (Radio та Checkbox)

Для вибору статі додаються дві радіокнопки (<input type="radio">) з однаковим атрибутом name="gender", що забезпечує їхню роботу як однієї групи. Кожне поле зв'язується з відповідним <label>. Для вибору курсів створюються три чекбокси (<input type="checkbox">) з назвами дисциплін, наприклад HTML, CSS, JS, що дозволяють обирати кілька варіантів.

Крок 6. Додавання випадаючого списку та текстової області

Для вибору міста проживання додається випадаючий список <select> з атрибутом name="city", усередині якого розміщуються теги <option> з відповідними значеннями. Для коментаря передбачається текстова область <textarea> з атрибутами name="comment", rows="4" та cols="50", при цьому текст-підказка задається через атрибут placeholder. У кінці форми додається кнопка відправки <button type="submit"> з текстом «Зареєструватися». Після цього слід перевірити роботу форми, натиснувши кнопку відправки з порожнім полем імені; браузер має видати попередження завдяки атрибуту required.

Крок 7. Збереження та відправка на GitHub

Необхідно створити новий репозиторій для другої лабораторної роботи (або додати папку до спільного репозиторію курсу). Після цього виконується комміт змін з відповідним повідомленням, наприклад «Додано таблицю та форму реєстрації», і код відправляється на віддалений сервер.

Крок 8. Оформлення звіту

Заповнюється титульний аркуш стандартного звіту. Додаються скриншоти коду таблиці та форми, а також скриншот зовнішнього вигляду сторінки в браузері. До звіту включається посилання на репозиторій, після чого

звіт зберігається у форматі PDF.

Контрольні питання для захисту:

1. Поясніть різницю між використанням тегів <th> та <td> у процесі створення таблиць.
2. Охарактеризуйте атрибути, що застосовуються для об'єднання кількох комірок таблиці по горизонталі та по вертикалі, навівши відповідні приклади.
3. Розкрийте призначення атрибутів action та method тегу <form> та вкажіть два основні методи передачі даних.
4. Обґрунтуйте доцільність використання тегу <label> для кожного поля <input> та опишіть спосіб їх зв'язування за допомогою відповідних атрибутів.
5. Визначте принципову відмінність у функціонуванні полів вводу з типами radio та checkbox та зазначте обов'язкову умову для коректної роботи групи радіокнопок.

ЛАБОРАТОРНА РОБОТА № 3

Тема: Каскадні таблиці стилів (CSS3). Селектори, типографіка та блокова модель.

Мета: Навчитися підключати зовнішні файли стилів до HTML-документа. Опанувати синтаксис CSS, використання базових та комбінованих селекторів. Набути практичних навичок стилізації тексту (типографіки), управління кольорами фону та застосування CSS блокової моделі (Box Model) для створення відступів та рамок навколо елементів.

Теоретичні відомості:

CSS (Cascading Style Sheets) є мовою таблиць стилів, призначеною для опису зовнішнього вигляду HTML-документа. Якщо HTML забезпечує формування структури, то CSS відповідає за візуальне оформлення, зокрема кольори, шрифти, відступи та позиціювання елементів.

Існує три способи додавання CSS:

- вбудований (Inline) через атрибут style безпосередньо в тегу, що не рекомендується для масштабованих проєктів;
- внутрішній (Internal) за допомогою тегу <style> у секції <head>;
- зовнішній (External) шляхом підключення окремого CSS-файлу через тег <link>, що вважається найкращою практикою.

Основу CSS складає правило, яке містить селектор, що визначає елементи для стилізації, та блок оголошень із властивостями та їх значеннями. До основних типів селекторів належать селектори за тегом, за класом та за

ідентифікатором.

Ключовою концепцією для розуміння розмірів і відступів є блокова модель (CSS Box Model), згідно з якою кожен HTML-елемент розглядається браузером як прямокутний блок, що складається з чотирьох шарів: вмісту (content), внутрішнього відступу (padding), рамки (border) та зовнішнього відступу (margin).

Завдання до виконання:

Крок 1. Підготовка робочого середовища та файлів

Підготувати робоче середовище: створити папку з назвою web-lab-3-Прізвище, відкрити її в редакторі VS Code, створити файл index.html з базовою структурою HTML5 (за допомогою Emmet) та організувати підпапку css, у якій розмістити файл style.css.

Крок 2. Підключення зовнішнього файлу стилів

У розділі <head> документа index.html додається елемент <link> із атрибутами rel="stylesheet" та href="css/style.css". Для перевірки коректності підключення у файлі style.css записується тестове правило, наприклад, зміна фону елемента body, після чого запускається Live Server для візуального контролю.

Крок 3. Підготовка HTML-розмітки

У тілі документа створюється елемент <header> із заголовком <h1>«Блог про вебтехнології»</h1>. Після цього додається елемент <main>, у якому розміщуються три статті <article>. Кожна стаття містить заголовок другого рівня <h2> (наприклад, «Що таке CSS», «Селектори», «Блокова модель») та два абзаци тексту <p>. Для подальшої стилізації кожному <article> присвоюється клас post, останньому абзацу – клас highlight, а одному із заголовків <h2> – унікальний ідентифікатор special-title.

Крок 4. Робота з типографікою та кольорами

Після формування структури документа здійснюється перехід до стилізації у файлі style.css. Спочатку задаються загальні типографічні властивості: для селектора body встановлюється гарнітура шрифту Arial, sans-serif; колір тексту #333333; міжрядковий інтервал 1.6. Для заголовка <h1> визначається вирівнювання по центру (text-align: center) та колір темно-синього відтінку, наприклад #1a5276.

Крок 5. Використання класових та ID-селекторів

Відпрацюйте застосування селекторів за класом та ідентифікатором. Для елемента з ідентифікатором #special-title задається колір тексту #e74c3c та підкреслення (text-decoration: underline). Для елементів із класом .highlight встановлюється напівжирне накреслення, курсив та фоновий колір #f9e79f.

Крок 6. Застосування блокової моделі (Box Model)

Ключовим етапом роботи є застосування блокової моделі (Box Model) до елементів із класом .post. Для них визначаються:

- фоновий колір #ffffff; ширина 80% (або max-width: 600px);
- зовнішні відступи margin: 20px auto (що забезпечує вертикальні відступи та центрування по горизонталі);
- внутрішні відступи padding: 20px;
- рамка border: 2px solid #bdc3c7;
- заокруглення кутів border-radius: 8px.

Після збереження стилів результат перевіряється у браузері, а за допомогою інструментів розробника (F12 → вкладка Elements → блок Computed) аналізуються складові блокової моделі – вміст (content), внутрішні відступи (padding), рамка (border) та зовнішні відступи (margin).

Крок 7. Збереження та відправка на GitHub

Збережіть виконану роботу у системі контролю версій: після форматування коду та виконайте комміт із повідомленням, наприклад «Додано базові стилі та застосовано Box Model», здійсніть відправку змін до публічного репозиторію на GitHub.

Крок 8. Оформлення звіту

Оформіть звіт за встановленим зразком. До звіту додаються скриншоти вихідного коду файлів index.html та style.css, зовнішнього вигляду сторінки у браузері, а також обов'язково скриншот панелі розробника із відображенням блокової моделі для елемента .post. Після цього до звіту вноситься посилання на відповідний репозиторій GitHub, і документ зберігається у форматі PDF.

Контрольні питання для захисту:

1. Охарактеризуйте три способи підключення CSS до HTML-документа та визначте, який із них вважається найкращою практикою, навівши відповідне обґрунтування.
2. Поясніть відмінності між селектором класу (.name) та селектором ідентифікатора (#name), а також укажіть допустиму кількість використань однакового ідентифікатора в межах однієї вебсторінки.
3. Розкрийте поняття CSS Box Model (Блокова модель) та перелічіть її чотири основні складові у порядку від центру до зовнішнього краю.
4. Визначте принципову різницю між властивостями padding та margin.
5. Опишіть, як впливає на HTML-елемент одночасне застосування властивостей margin: 0 auto; та фіксованої ширини (наприклад, width: 50%).

ЛАБОРАТОРНА РОБОТА № 4

Тема: Позиціювання елементів. Створення базового макета сторінки.

Мета: Зрозуміти принципи нормального потоку документа (Normal Flow) та властивості display. Опанувати різні методи позиціювання елементів за допомогою властивості position (relative, absolute, fixed, sticky). Навчитися керувати накладанням елементів один на одного за допомогою z-index та створити класичний двоколонковий макет сторінки.

Теоретичні відомості:

За замовчуванням браузер розміщує елементи на сторінці відповідно до нормального потоку документа (Normal Flow). Блокові елементи (display: block – наприклад, , <p>, <h1>) займають усю доступну ширину і розташовуються один під одним. Рядкові елементи (display: inline – наприклад, , <a>) розташовуються в один рядок і не приймають властивостей ширини (width) та висоти (height). Гібридний тип inline-block дозволяє елементам стояти в ряд, але при цьому мати задані розміри.

Для складнішого керування розташуванням використовується властивість position. Вона вириває елемент з нормального потоку або змінює його поведінку:

- static: Поведінка за замовчуванням. Елемент залишається в нормальному потоці.
- relative (Відносне): Елемент залишається у потоці, але його можна зсунути (властивостями top, bottom, left, right) відносно його *початкового* місця.
- absolute (Абсолютне): Елемент повністю вирівається з потоку (інші елементи його ігнорують). Він позиціюється відносно *найближчого батьківського елемента*, який має будь-яке позиціювання, крім static.
- fixed (Фіксоване): Елемент позиціюється відносно *вікна браузера* (viewport) і залишається на місці під час прокручування сторінки.
- sticky (Липке): Гібрид відносного та фіксованого. Елемент прокручується разом зі сторінкою до певної точки, а потім «прилипає» до краю екрана.

Властивість z-index визначає порядок накладання елементів по осі Z (глибина). Елемент із більшим значенням перекриває елемент із меншим. z-index працює тільки для позиційованих елементів (relative, absolute, fixed, sticky).

Завдання до виконання:

Крок 1. Підготовка робочого середовища

На початку роботи необхідно створити окрему папку з назвою web-lab-4-Прізвище та відкрити її в редакторі VS Code. У цій папці створюється файл index.html із базовою структурою HTML5, а також папка css, у якій розміщується файл style.css. Підключення зовнішньої таблиці стилів до HTML-документа здійснюється у розділі <head> за допомогою тега <link>.

Крок 2. Створення HTML-структури макета

У тілі документа (<body>) формується каркас вебсторінки.

Спочатку додається шапка сайту з використанням елемента <header class="site-header">, яка містить заголовок першого рівня. Далі створюється блок навігації <nav class="main-nav"> з трьома гіперпосиланнями.

Основний вміст сторінки групується в контейнері , усередині якого розташовуються дві колонки: основна частина <main class="content"> із заголовком та кількома абзацами тексту, а також бічна панель <aside class="sidebar"> із заголовком та абзацом.

Після контейнера розміщується підвал <footer class="site-footer"> з інформацією про авторські права. У нижній частині тіла документа додається елемент для повернення на початок сторінки – посилання з класом btn-up.

Крок 3. Базова стилізація та створення двоколонкового макета

У файлі style.css насамперед виконується скидання стандартних відступів для всіх елементів за допомогою універсального селектора, що встановлює margin: 0; padding: 0; та box-sizing: border-box;.

Для контейнера .container задаються обмеження максимальної ширини (1000px) та автоматичні поля для горизонтального центрування.

Двоколонкова структура реалізується через властивість display: inline-block: для елемента .content встановлюється ширина 70%, а для .sidebar – 29%, з увімкненням вертикального вирівнювання vertical-align: top та внутрішніх відступів padding. Невелика різниця у відсотках (70%+29%) компенсує пробільні символи між inline-блоками, що виникають через особливості форматування HTML-коду.

Крок 4. Робота з фіксованим та «липким» позиціюванням (fixed, sticky)

Для забезпечення постійної видимості навігаційного меню при прокручуванні сторінки до селектора .main-nav застосовується властивість position: sticky зі значенням top: 0. Це змушує елемент «прилипати» до верхньої межі вікна перегляду після досягнення відповідної позиції. Додатково меню отримує темний фон, білий текст та внутрішні відступи. Кнопка «Вгору» позиціонується абсолютно відносно вікна браузера за допомогою position: fixed із координатами bottom: 30px та right: 30px, що гарантує її нерухомість у правому нижньому куті незалежно від прокрутки. Для кнопки також задаються кольори, заокруглення та скасування підкреслення тексту.

Крок 5. Абсолютне та відносне позиціювання (absolute + relative)

Для демонстрації взаємодії absolute та relative у бічну панель (<aside class="sidebar">) додається елемент-маркер Нове!. У CSS для батьківського контейнера .sidebar встановлюється position: relative, що робить його точкою відліку для дочірніх елементів із абсолютним позиціюванням. Сам бейдж отримує position: absolute зі зміщенням top: -10px та right: -10px, завдяки чому він розташовується у правому верхньому куті панелі, частково виходячи за її межі. Для коректного відображення поверх іншого вмісту за потреби додається z-index. Бейдж стилізується яскравим фоном, округлими краями та контрастним текстом.

Крок 6. Збереження, тестування та відправка на GitHub

Після завершення стилізації необхідно перевірити функціональність усіх ефектів у браузері: липке меню має залишатися зверху під час скролу, кнопка «Вгору» фіксуватися в кутку, а бейдж – точно прилягати до кута сайдбару. Успішно протестований код додається до локального Git-репозиторію з коментарем «Додано макет та різні типи позиціювання» та відправляється на віддалений сервер GitHub.

Крок 7. Оформлення звіту

Звіт виконується у форматі PDF і має містити титульний аркуш із темою та метою роботи, теоретичні відомості, скриншоти HTM- та CSS-коду, а також скриншот браузера, на якому чітко видно поведінку елементів при прокручуванні. Обов'язково додається посилання на репозиторій із виконаним завданням.

Контрольні питання для захисту:

1. Охарактеризуйте поняття нормального потоку документа (Normal Flow) та визначте відмінності в поведінці елементів із display: block та display: inline.

2. Назвіть CSS-властивості, що дозволяють змістити елемент відносно його початкового положення без виривання з потоку, залишаючи на старому місці порожній простір.

3. Поясніть принцип («золоте правило») абсолютного позиціювання: відносно якого контейнера визначаються координати елемента з position: absolute, якщо жоден із його предків не має заданої властивості position?

4. Проведіть порівняльний аналіз значень fixed та sticky властивості position, вказавши ключові відмінності в їхній поведінці при прокручуванні сторінки.

5. Розкрийте призначення властивості z-index та умови, за яких вона починає діяти. Чому z-index не працює для елементів, що не мають встановленої властивості position?

ЛАБОРАТОРНА РОБОТА № 5

Тема: Основи JavaScript. Змінні, типи даних, цикли та функції.

Мета: Ознайомитися з основами клієнтського програмування. Навчитися підключати файли JavaScript до HTML-документа. Опанувати синтаксис мови: оголошення змінних, використання примітивних типів даних та структур даних (масиви, об'єкти), написання умовних конструкцій, циклів та створення функцій. Набути навичок роботи з консоллю розробника у браузері.

Теоретичні відомості:

JavaScript (JS) – це високорівнева мова програмування, яка відповідає за інтерактивність та динамічну поведінку вебсторінок. Якщо HTML створює структуру, а CSS – зовнішній вигляд, то JavaScript «оживляє» елементи (обробляє кліки, відправляє дані на сервер, створює анімації).

JavaScript код можна писати безпосередньо в HTML-файлі всередині парного тегу <script>, але найкращою практикою є винесення коду в окремий файл із розширенням .js та його підключення: <script src=«script.js» defer></script>. Атрибут defer гарантує, що скрипт завантажиться асинхронно і виконається лише після повного завантаження HTML-документа.

Основи синтаксису:

-Змінні: для збереження даних використовуються ключові слова let (змінна, значення якої можна змінити) та const (константа, значення якої не змінюється). Застаріле ключове слово var у сучасному коді не використовується.

-Типи даних: Примітивні (Number, String, Boolean, Null, Undefined) та складні (Object, Array).

-Умовні конструкції: if, else if, else дозволяють виконувати різний код залежно від умов.

-Цикли: for та while використовуються для багаторазового виконання блоку коду (наприклад, перебору елементів масиву).

-Функції: іменовані блоки коду, які виконують конкретне завдання і можуть бути викликані багаторазово. Сучасний стандарт ES6 ввів поняття «стрілкових функцій» (Arrow functions), які мають більш компактний синтаксис.

Головним інструментом для тестування JS-коду на етапі розробки є консоль браузера (відкривається клавішею F12). Виведення даних у консоль здійснюється командою console.log().

Завдання до виконання:

Крок 1. Підготовка робочого середовища

На початковому етапі необхідно створити нову папку з назвою web-lab-5-Прізвище та відкрити її у VS Code. У корені цієї папки створюється файл index.html з базовою структурою HTML-документа. У тілі документа (<body>) розміщується заголовок першого рівня <h1> з текстом «Основи JavaScript» та абзац <p>, який інформує користувача про необхідність відкриття консолі розробника (F12) для перегляду результатів виконання скрипту. Далі створюється папка js, у якій розміщується файл script.js. Підключення зовнішнього скрипта до HTML-документа здійснюється за допомогою тегу <script src="js/script.js"></script>, який рекомендується розміщувати перед закриваючим тегом </body> або в розділі <head> з атрибутом defer.

Крок 2. Робота зі змінними та типами даних

У файлі script.js необхідно реалізувати наступний код. Спочатку оголошується константа (const) з іменем studentName, якій присвоюється рядкове значення (String) – ім'я студента. Потім оголошується змінна (let) з іменем studentAge для зберігання числового значення (Number) віку. Також оголошується змінна isEnrolled, якій присвоюється логічне значення true (Boolean). Для виведення значень цих змінних у консоль використовується метод console.log() із застосуванням шаблонних рядків (Template literals), які дозволяють вбудовувати вирази у рядок за допомогою зворотних лапок та синтаксису ${}. Після збереження коду слід відкрити файл index.html у браузері, активувати консоль розробника клавішею F12 та переконатися у коректному виведенні повідомлення на вкладці «Console».

Крок 3. Робота зі структурами даних (Масиви та Об'єкти)

Наступним кроком передбачається створення структурованих даних. Оголошується константа subjects, яка є масивом (Array) і містить перелік улюблених дисциплін, наприклад: «Вища математика», «Вебдизайн», «Бази даних». Також створюється об'єкт (Object) з іменем user, який описує користувача за допомогою властивостей firstName, lastName, role та loginCount. Після створення цих структур необхідно вивести у консоль другий елемент масиву subjects (з урахуванням нульової індексації) та значення властивості role з об'єкта user.

Крок 4. Умовні конструкції (if/else)

Для закріплення навичок роботи з логічними конструкціями необхідно написати умовне розгалуження, яке аналізує значення властивості loginCount об'єкта user. Якщо значення цієї властивості перевищує 10, у консоль виводиться повідомлення «Це активний користувач.». У випадку, коли

loginCount дорівнює нулю, виводиться повідомлення «Це новий користувач.». Для всіх інших значень (блок else) передбачено виведення повідомлення «Звичайний користувач.».

Крок 5. Використання циклів (for)

Для демонстрації роботи ітераційних процесів слід використати цикл for, який перебирає всі елементи масиву subjects. У тілі циклу необхідно організувати виведення в консоль кожного елемента масиву у форматі «Дисципліна №X: Назва», де номер дисципліни визначається лічильником циклу.

Крок 6. Створення функцій

На цьому етапі потрібно створити дві функції з різними синтаксичними підходами. Перша функція (Function Declaration) має назву calculateArea, приймає два параметри (width та height) і повертає значення площі прямокутника як результат їх добутку. Після оголошення функція викликається з аргументами, наприклад 5 та 10, а результат виводиться у консоль. Друга функція реалізується як стрілкова функція (Arrow Function) з іменем isAdult, яка приймає один параметр (age) та повертає true, якщо вік більше або дорівнює 18, і false в іншому випадку. Для перевірки роботи функції їй передається раніше створена змінна studentAge.

Крок 7. Збереження, тестування та відправка на GitHub

Після написання всього коду необхідно переконатися у відсутності помилок у консолі браузера (червоних повідомлень Errors) та коректному відпрацюванні всіх команд console.log(). Потім виконується комміт змін з відповідним повідомленням, наприклад «Додано основи JS: змінні, цикли, функції», і код відправляється на віддалений репозиторій GitHub.

Крок 8. Оформлення звіту

На завершальному етапі оформлюється звіт у форматі PDF. До звіту додаються скриншот коду файлу script.js та скриншот вікна браузера з відкритою консоллю розробника, де чітко видно всі результати виконання скрипта (результати кроків 2-6). Також до звіту обов'язково вноситься посилання на створений репозиторій.

Контрольні питання для захисту:

1. Поясніть принципову різницю між використанням ключових слів let, const та var при оголошенні змінних у JavaScript.
2. Перелічіть примітивні типи даних, які існують у мові JavaScript.
3. Визначте відмінності між масивом (Array) та об'єктом (Object) у JavaScript та опишіть доцільність застосування кожної з цих структур даних.
4. Поясніть призначення ключового слова return у функціях та опишіть,

що поверне функція, якщо оператор return не використовується.

5. Наведіть приклад базової стрілкової функції (Arrow function) та охарактеризуйте відмінності її синтаксису від синтаксису звичайної функції (Function Declaration).

ЛАБОРАТОРНА РОБОТА № 6

Тема: Робота з DOM-деревом. Динамічна зміна вмісту сторінки.

Мета: Зрозуміти концепцію об'єктної моделі документа (DOM). Опанувати методи пошуку HTML-елементів за допомогою JavaScript. Навчитися динамічно зчитувати та змінювати текстовий вміст, HTML-код, CSS-класи та стилі елементів сторінки. Набути навичок програмного створення нових вузлів та їх додавання до DOM-дерева.

Теоретичні відомості:

DOM (Document Object Model) Об'єктна модель документа – це програмний інтерфейс (API) для HTML-документів. Коли браузер завантажує сторінку, він створює її внутрішню репрезентацію у вигляді ієрархічного дерева вузлів (nodes). Завдяки DOM мова JavaScript може отримувати доступ до будь-якого елемента сторінки, змінювати його, видаляти або додавати нові.

Точкою входу до DOM є глобальний об'єкт document. Основні методи для пошуку елементів:

- document.getElementById('id') – знаходить один елемент за його унікальним ідентифікатором.
- document.querySelector('.class') – знаходить *перший* елемент, що відповідає вказаному CSS-селектору.
- document.querySelectorAll('p') – повертає колекцію (NodeList) *усіх* елементів, що відповідають селектору.

Зміна вмісту:

- element.textContent – зчитує або задає лише текст всередині елемента (безпечно).
- element.innerHTML – зчитує або задає HTML-вміст елемента (можна вставляти теги, але вимагає обережності через ризик XSS-атак).

Управління стилями та класами: Хоча можна змінювати стилі напряму через об'єкт style (наприклад, element.style.color = «red»;), найкращою практикою є додавання або видалення заздалегідь підготовлених CSS-класів за допомогою властивості classList (методи .add(), .remove(), .toggle()).

Завдання до виконання:

Крок 1. Підготовка робочого середовища

Створіть папку з назвою web-lab-6-Прізвище та відкрийте її в редакторі VS Code. Організуйте базову структуру проєкту: у кореневій папці створіть файл index.html, а також дві підпапки – css для файлу style.css та js для файлу script.js. Після цього у файлі index.html підключіть створені файли стилів (у розділі <head>) та скриптів (перед закриваючим тегом </body> або з атрибутом defer).

Крок 2. Підготовка HTML-розмітки

У тілі HTML-документа (<body>) розмістіть такі елементи: заголовок першого рівня з ідентифікатором «main-title» та текстом «Старий заголовок сторінки»; блок із класом «content-box», який містить абзац <p> із класом «text» та початковим текстом; а також заголовок другого рівня «Список покупок:» і порожній невпорядкований список із ідентифікатором «shopping-list».

Крок 3. Підготовка CSS-класів

У файлі style.css визначте кілька класів, які згодом будуть динамічно застосовуватися за допомогою JavaScript. Клас .highlight має встановлювати, наприклад, жовтий фон, жирне накреслення та внутрішні відступи. Клас .hidden повинен приховувати елемент за допомогою display: none;.

Крок 4. Пошук елементів та зміна тексту (JavaScript)

У файлі script.js реалізуйте пошук елементів і зміну їхнього вмісту. Спочатку знайдіть заголовок за його ідентифікатором за допомогою методу getElementById та збережіть його в константу. Замініть текстовий вміст цього елемента через властивість textContent. Далі за допомогою методу querySelector знайдіть абзац із класом «text» та змініть його внутрішній HTML, додавши виділення жирним шрифтом, через властивість innerHTML.

Крок 5. Робота з CSS-класами та стилями через JS

Знайдіть елемент div із класом «content-box» та додайте йому раніше створений клас «highlight», використовуючи метод classList.add(). Потім безпосередньо змініть стилі абзацу, звернувшись до його властивості style: встановіть синій колір тексту та розмір шрифту 20 пікселів.

Крок 6. Динамічне створення елементів DOM

Для наповнення порожнього списку створіть масив рядків із назвами продуктів. Знайдіть елемент ul за його ідентифікатором. За допомогою циклу переберіть елементи масиву, для кожного з них створіть новий тег <li> методом createElement, встановіть його текстовий вміст рівним поточному значенню, а потім додайте створений вузол до списку через appendChild.

Крок 7. Перевірка результату та відправка на GitHub

Запустіть index.html за допомогою Live Server та переконайтеся, що всі

зміни відобразилися коректно: заголовок змінився, абзац містить жирне виділення, блок має жовтий фон, а список відображає три пункти. Після перевірки виконайте комміт змін та відправте репозиторій на GitHub.

Крок 8. Оформлення звіту

Підготуйте звіт у форматі PDF, який має містити титульний аркуш, скриншоти коду файлів index.html, style.css, script.js, а також скриншот візуального результату у браузері. Додайте посилання на створений репозиторій.

Контрольні питання для захисту:

1. Поясніть сутність поняття DOM та його зв'язок із HTML-документом.
2. Проведіть порівняльний аналіз методів document.getElementById та document.querySelector, зазначивши їхні особливості.
3. Розкрийте відмінності між властивостями textContent та innerHTML, а також обґрунтуйте, яка з них є безпечнішою з точки зору запобігання XSS-атакам.
4. Охарактеризуйте призначення властивості classList, наведіть щонайменше два її методи та опишіть їхню дію.
5. Опишіть послідовність дій у JavaScript, необхідних для створення нового HTML-елемента (наприклад,) та його відображення на сторінці.

ЛАБОРАТОРНА РОБОТА № 7

Тема: Обробка подій у JavaScript. Валідація форм на стороні клієнта.

Мета: Опанувати механізми обробки подій (Events) у JavaScript. Навчитися призначати обробники подій на HTML-елементи за допомогою методу addEventListener. Реалізувати клієнтську валідацію вебформи: перевірку правильності заповнення полів введення до моменту відправки даних на сервер, виведення користувацьких повідомлень про помилки та блокування стандартної поведінки браузера.

Теоретичні відомості:

Подія (Event) – це сигнал від браузера про те, що щось відбулося. Наприклад: користувач клікнув мишкою (click), натиснув клавішу на клавіатурі (keydown), відправив форму (submit), ввів текст у поле (input) або сторінка повністю завантажилася (DOMContentLoaded).

Щоб JavaScript міг реагувати на ці події, використовується метод addEventListener. Він «прикріплює» функцію (обробник події) до певного HTML-елемента і чекає, поки вказана подія не відбудеться.

Синтаксис: element.addEventListener('event_name', function(event) { ... });

Обробник події автоматично отримує об'єкт події (event або скорочено e), який містить усю інформацію про те, що сталося (наприклад, координати кліку або те, яка саме клавіша була натиснута).

Валідація форм – це процес перевірки введених користувачем даних на відповідність певним правилам (наприклад, пароль має бути не коротшим за 8 символів). Коли користувач натискає кнопку «Відправити» (Submit), браузер за замовчуванням намагається миттєво перезавантажити сторінку і відправити дані. Щоб цього уникнути і спочатку перевірити дані за допомогою JS, потрібно зупинити цю стандартну поведінку за допомогою методу e.preventDefault().

Завдання до виконання:

Крок 1. Підготовка робочого середовища

На початку роботи необхідно створити папку з назвою web-lab-7-Прізвище та відкрити її в інтегрованому середовищі розробки VS Code. У цій папці створюються три файли: index.html, css/style.css та js/script.js. Після створення файлів слід підключити таблицю стилів та скрипт до HTML-документа.

Крок 2. Створення HTML-розмітки форми

У файлі index.html розробляється форма реєстрації. Важливою особливістю є відсутність атрибута required, оскільки валідація буде реалізована засобами JavaScript. Спочатку створюється елемент <form> з ідентифікатором registration-form. Для поля введення імені користувача додається елемент <label> з відповідним текстом та поле <input type="text">, а також порожній елемент із класом error-text для виведення повідомлень про помилки. Аналогічним чином створюються поля для електронної пошти (з типом email) та пароля (з типом password), кожне з яких супроводжується власним елементом для виведення помилок. Завершується форма кнопкою відправки з типом submit.

Крок 3. Додавання стилів для стану помилки

У файлі style.css визначаються базові стилі для форми та спеціальні стилі, що візуалізують стан помилки. Для елементів із класом error-text встановлюються червоний колір тексту, малий розмір шрифту, блочний тип відображення, відступи та мінімальна висота. Для підсвічування полів, що містять помилку, створюється клас .input-error, який задає червону рамку та світло-червоний фон. Крім того, форма стилізується для досягнення візуальної привабливості: визначаються ширина, внутрішні відступи, та тип відображення flex із вертикальним напрямком.

Крок 4. Налаштування обробника події submit у JavaScript

У файлі script.js спочатку здійснюється пошук елемента форми за її ідентифікатором з подальшим збереженням у константу. До форми додається обробник події submit, у якому першочергово викликається метод preventDefault() для блокування стандартної відправки форми. Після цього ініціюється виклик функції валідації.

Крок 5. Написання логіки валідації

Створюється функція validateForm(), яка реалізує перевірку кожного поля форми. У тілі функції отримується доступ до значень полів за допомогою властивості value та методу trim(), що видаляє зайві пробіли. Вводиться змінна-флаг isValid, початкове значення якої встановлюється як true; вона набуває значення false у разі виявлення будь-якої помилки валідації. Для поля імені перевіряється, чи не є воно порожнім: якщо умова виконується, у відповідний елемент span виводиться повідомлення про помилку, полю введення присвоюється клас .input-error, а змінна isValid встановлюється як false. За відсутності помилки повідомлення очищується та видаляється клас підсвічування. Для поля пароля перевіряється, чи довжина введеного значення становить не менше восьми символів; у разі порушення цієї умови виводиться відповідне повідомлення про помилку та змінна isValid встановлюється як false. Після завершення перевірок аналізується значення змінної isValid: якщо воно дорівнює true, у консоль (або за допомогою alert) виводиться повідомлення про успішну валідацію та готовність даних до відправки, після чого форма очищується методом reset().

Крок 6. Тестування та відправка на GitHub

Сторінка відкривається у браузері для тестування функціональності. При спробі надсилання форми з порожніми полями мають відображатися червоні повідомлення про помилки та червоні рамки навколо відповідних полів, при цьому сторінка не повинна перезавантажуватися. Після введення коректних даних попередження мають зникати, а форма – очищатися. Після успішного тестування виконується комміт змін з відповідним повідомленням та здійснюється відправка коду на віддалений репозиторій GitHub.

Крок 7. Оформлення звіту

Звіт оформлюється у форматі PDF. До нього додаються скриншоти вмісту файлів HTML, CSS та JavaScript. Також додаються скриншоти браузера, що демонструють вигляд форми до валідації, вигляд форми з відображеними помилками при введенні некоректних даних, а також спливаюче вікно або повідомлення в консолі про успішну реєстрацію. Наприкінці звіту наводиться посилання на репозиторій.

Контрольні питання для захисту:

1. Поняття події (Event) у контексті браузерного середовища.
2. Функціональне призначення методу addEventListener.
3. Обґрунтуйте недоцільність використання атрибутів типу onclick безпосередньо в HTML-розмітці порівняно із застосуванням методу addEventListener.
4. Розкрийте роль методу preventDefault() у функції обробки події відправки форми та описати наслідки його відсутності.
5. Поясніть спосіб отримання в JavaScript текстового вмісту, введеного користувачем у поле <input>.

ЛАБОРАТОРНА РОБОТА № 8

Тема: Асинхронний JavaScript. Робота з Fetch API та форматом JSON.

Мета: Опанувати концепцію асинхронного програмування у JavaScript. Навчитися формувати та надсилати HTTP-запити до зовнішніх серверів (REST API) за допомогою сучасної функції fetch(). Зрозуміти структуру формату обміну даними JSON та набути навичок динамічного рендерингу отриманої інформації на вебсторінці.

Теоретичні відомості:

Зазвичай код у JavaScript виконується синхронно (рядок за рядком). Однак мережеві запити (наприклад, завантаження списку товарів із бази даних) потребують часу. Якби JS чекав на відповідь сервера синхронно, уся сторінка б «зависала» і не реагувала на кліки користувача. Для вирішення цього використовується асинхронне програмування.

Головним сучасним інструментом для мережевих запитів є Fetch API. Метод fetch() ініціює запит до вказаної URL-адреси і повертає спеціальний об'єкт – Promise (Обіцянку), який представляє результат асинхронної операції, що завершиться в майбутньому. Для зручної роботи з промісами сучасний стандарт мови використовує ключові слова async (оголошує асинхронну функцію) та await (змушує код дочекатися результату конкретного запиту, не блокуючи при цьому решту сторінки).

Більшість сучасних серверів повертають дані у форматі JSON (JavaScript Object Notation). Це легкий текстовий формат, який дуже нагадує звичайні об'єкти та масиви в JS, але всі ключі в ньому обов'язково беруться у подвійні лапки (наприклад, {«name»: «Іван», «age»: 20}). Для перетворення цієї текстової відповіді у повноцінний JS-об'єкт використовується метод .json().

Завдання до виконання:

Крок 1. Підготовка робочого середовища

Необхідно створити папку з назвою web-lab-8-Прізвище та відкрити її у VS Code. У цій папці створюється структура проєкту, що складається з файлів index.html, css/style.css та js/script.js. Виконується підключення створених файлів стилів та скриптів до HTML-документа.

Крок 2. Підготовка HTML-розмітки

У файлі index.html розробляється інтерфейс для завантаження списку користувачів із безкоштовного тестового API JSONPlaceholder. До тіла документа додається головний заголовок першого рівня з текстом «Каталог користувачів». Створюється кнопка з ідентифікатором load-users-btn та текстом «Завантажити дані», яка ініціюватиме виконання запиту. Також додається порожній контейнер-блок з ідентифікатором users-container та класом grid-container, призначений для динамічного додавання карток користувачів засобами JavaScript.

Крок 3. Базова стилізація карток (CSS)

У файлі style.css виконується налаштування стилів для сітки та майбутніх карток. Для елемента з класом grid-container задається відображення у вигляді сітки за допомогою CSS Grid або Flexbox, що забезпечує розташування карток у кілька колонок. Створюється клас user-card, який згодом динамічно додаватиметься до елементів кожного користувача; для цього класу визначаються такі властивості, як рамка, внутрішні відступи, заокруглені кути та колір фону.

Крок 4. Створення асинхронної функції (JavaScript)

У файлі script.js оголошується асинхронна функція з назвою fetchUsers. Для запобігання некоректному завершенню скрипту у разі виникнення проблем із мережею, тіло функції обгортається конструкцією try...catch, де у блоці catch передбачено виведення повідомлення про помилку до консолі.

Крок 5. Виконання запиту та парсинг JSON

У блоці try за допомогою функції fetch виконується GET-запит до тестового API за адресою https://jsonplaceholder.typicode.com/users, при цьому очікується отримання відповіді з використанням оператора await. Здійснюється перевірка статусу відповіді: якщо response.ok має значення false, генерується виняток із повідомленням про помилку мережі. Отримана відповідь перетворюється з формату JSON у масив об'єктів JavaScript за допомогою методу response.json() з очікуванням результату. Отриманий масив виводиться у консоль для візуального аналізу структури даних.

Крок 6. Динамічний рендеринг елементів у DOM

Після отримання масиву користувачів у межах блоку try виконуються

наступні дії. Контейнер для карток знаходиться за допомогою методу getElementById. Попередній вміст контейнера очищається шляхом присвоєння порожнього рядка властивості innerHTML, що запобігає накопиченню карток при повторних натисканнях кнопки. За допомогою циклу forEach здійснюється перебір елементів масиву. Для кожного об'єкта користувача створюється новий елемент div, якому додається клас user-card. Наповнення картки даними виконується через властивість innerHTML із використанням шаблонних рядків, куди підставляються значення властивостей імені, електронної пошти та міста з вкладеного об'єкта адреси. Створена картка додається до контейнера за допомогою методу appendChild.

Крок 7. Прив'язка функції до події кліку

Поза межами функції fetchUsers виконується пошук кнопки завантаження за її ідентифікатором. До знайденої кнопки додається обробник події click, який запускає функцію fetchUsers.

Крок 8. Тестування, збереження та відправка на GitHub

Виконується відкриття сторінки у браузері, де контейнер спочатку залишається порожнім. Після відкриття консолі розробника та натискання кнопки «Завантажити дані» на сторінці мають відобразитися десять карток користувачів із відповідним стильовим оформленням, а в консолі – виведений масив об'єктів. Після успішного тестування виконується комміт змін із повідомленням «Додано роботу з Fetch API та JSON» та здійснюється відправка коду до віддаленого репозиторію на GitHub.

Крок 9. Оформлення звіту

Звіт оформлюється у форматі PDF із зазначенням теми, мети роботи та теоретичних відомостей. До звіту додаються скриншоти вмісту файлів index.html, style.css та script.js, а також скриншот браузера після натискання кнопки із відображеними картками та відкритою консоллю, де видно отриманий масив users. Обов'язковим елементом є посилання на репозиторій із виконаною роботою.

Контрольні питання для захисту:

1. Поясніть принципову різницю між синхронним та асинхронним виконанням коду та обґрунтуйте доцільність застосування асинхронного підходу для виконання мережевих запитів.

2. Дайте визначення формату JSON та вкажіть ключові відмінності його синтаксису від синтаксису звичайного об'єкта JavaScript (зокрема, щодо оформлення ключів лапками).

3. Розкрийте функціональне призначення ключових слів async та await та опишіть їхній вплив на виконання функції.

4. Обґрунтуйте необхідність використання конструкції try...catch при

роботі з мережевими запитами та опишіть наслідки її відсутності у разі недоступності сервера.

5. Визначте, який метод об'єкта response застосовується у Fetch API для перетворення тіла відповіді з текстового формату JSON у структури даних JavaScript.

ЛАБОРАТОРНА РОБОТА № 9

Тема: Основи PHP. Синтаксис, масиви, керуючі конструкції.

Мета: Ознайомитися з принципами роботи серверних мов програмування. Налаштувати локальне серверне середовище (наприклад, XAMPP або вбудований сервер PHP). Опанувати базовий синтаксис PHP: змінні, типи даних, створення індексованих та асоціативних масивів, використання умовних конструкцій та циклів. Навчитися генерувати HTML-код динамічно за допомогою PHP.

Теоретичні відомості:

PHP (PHP: Hypertext Preprocessor) – це популярна мова програмування загального призначення, яка використовується переважно для серверної (backend) веброзробки.

Фундаментальна різниця між JavaScript та PHP полягає у місці їх виконання. JS виконується безпосередньо у браузері користувача (на клієнті). PHP виконується на сервері. Коли користувач запитує сторінку .php, сервер запускає PHP-код, виконує обчислення (наприклад, дістає дані з бази), формує готову HTML-сторінку і відправляє її браузеру. Користувач ніколи не бачить вихідного PHP-коду, лише результат його роботи (HTML/CSS/JS).

Щоб комп'ютер міг обробляти файли .php, на ньому має бути встановлений вебсервер (наприклад, Apache або Nginx) та інтерпретатор PHP. Найчастіше для навчання використовують готові збірки типу XAMPP, MAMP або локальний сервер, вбудований у сам PHP.

Основи синтаксису:

- Код PHP завжди пишеться всередині спеціальних тегів: <?php ... ?>.
- Кожна інструкція повинна закінчуватися крапкою з комою (;).
- Усі змінні в PHP починаються зі знака долара $. Мова має динамічну типізацію.
- Для виведення даних на екран (генерування тексту або HTML) найчастіше використовується конструкція echo.
- Масиви в PHP бувають індексованими (де ключами є числа від 0) та асоціативними (де ключами є рядки-імена, аналог об'єктів у JS). Для зручного

перебору масивів використовується цикл foreach.

Завдання до виконання:

Крок 1. Підготовка робочого середовища (Локальний сервер)

Оскільки PHP не виконується при безпосередньому відкритті файлу в браузері, як це відбувається з HTML, необхідно забезпечити функціонування серверного середовища. Для цього слід встановити та запустити локальну серверну збірку (наприклад, XAMPP) і активувати модуль Apache. Після запуску сервера потрібно перейти до його кореневої директорії (у XAMPP це папка htdocs на диску C: або в каталозі програм). Усередині цієї папки створюється нова директорія з назвою web-lab-9-Прізвище, яка потім відкривається у VS Code. Альтернативним способом є використання вбудованого сервера PHP: за наявності глобально встановленого PHP достатньо створити папку проєкту в будь-якому місці, відкрити термінал у цій папці та виконати команду php -S localhost:8000.

Крок 2. Створення першого PHP-скрипту

У створеній папці проєкту необхідно створити файл index.php і сформувати в ньому базову HTML-структуру за допомогою Emmet. Усередині тіла документа (тег <body>) відкриваються PHP-теги, у яких за допомогою конструкції echo виводиться заголовок, наприклад: <?php echo "<h1>Привіт, світе! Це мій перший PHP-скрипт.</h1>"; ?>. Для перевірки роботи скрипту в браузері вводиться адреса http://localhost/web-lab-9-Прізвище/ (або http://localhost:8000 при використанні вбудованого сервера), де має відобразитися відповідний заголовок.

Крок 3. Робота зі змінними та типами даних

У межах блоку PHP необхідно оголосити рядкову змінну $name із власним іменем та цілочисельну змінну $age із зазначенням віку. Для виведення даних використовується echo, причому в PHP існує два способи об'єднання рядків: конкатенація за допомогою крапки (.) або безпосереднє розміщення змінних у подвійних лапках, де вони автоматично інтерпретуються. Наприклад: echo " Мене звати $name, мені $age років. ";.

Крок 4. Створення та використання масивів

Спочатку створюється індексований масив улюблених мов програмування, наприклад: $languages = ["HTML", "CSS", "JavaScript", "PHP"];. Далі формується асоціативний масив, що описує студента, де кожному ключу відповідає певне значення: $student = ["first_name" => "Іван", "last_name" => "Іваненко", "specialty" => "Комп'ютерні науки", "gpa" => 4.8];. Для демонстрації роботи з асоціативним масивом на сторінку виводиться значення за ключем specialty за допомогою echo.

Крок 5. Умовні конструкції (if/else)

На основі значення середнього балу з масиву $student реалізується умовна конструкція. Якщо $student["gpa"] більше або дорівнює 4.0, виводиться повідомлення із зеленим кольором тексту про право на стипендію. У протилежному випадку виводиться повідомлення червоним кольором про недостатній бал.

Крок 6. Використання циклу foreach для генерації HTML

PHP надає зручну можливість динамічної генерації HTML-розмітки на основі даних із масивів. Спочатку виводиться заголовок другого рівня «Мої навички:». Потім за допомогою echo відкривається тег невпорядкованого списку <ul>. Далі запускається цикл foreach, який перебирає елементи масиву $languages; для кожного елемента генерується тег <li> із відповідним значенням. Після завершення циклу список закривається. При перегляді сторінки в браузері та аналізі її вихідного коду (Ctrl+U) видно лише чистий HTML-список без жодних слідів PHP-коду чи циклів, що підтверджує серверний характер обробки.

Крок 7. Збереження, відправка на GitHub та звіт

Після перевірки працездатності коду на локальному сервері без помилок виконується комміт змін із відповідним повідомленням (наприклад, «Додано основи PHP: змінні, масиви, foreach») та відправка коду до віддаленого репозиторію. Далі оформлюється звіт у форматі PDF, який має містити тему, мету роботи та теоретичні відомості. До звіту додаються скриншот коду файлу index.php та скриншот результату виконання в браузері, де в адресному рядку чітко видно localhost. Обов'язково вказується посилання на репозиторій.

Контрольні питання для захисту:

1. Поясніть принципову різницю між виконанням коду на стороні клієнта (JavaScript) та на стороні сервера (PHP).
2. Вкажіть символ, яким обов'язково має починатися ім'я будь-якої змінної в мові PHP.
3. Охарактеризуйте відмінності між індексованим та асоціативним масивами в PHP, зазначивши, що використовується як ключі в асоціативному масиві.
4. Розкрийте призначення циклу foreach та обґрунтуйте, чому він є найзручнішим інструментом для перебору масивів.
5. Опишіть, що побачить користувач у вихідному коді сторінки (відкритому через інструменти розробника браузера), яка була згенерована за допомогою PHP, та чи будуть там відображатися змінні й цикли.

ЛАБОРАТОРНА РОБОТА № 10

Тема: Робота з формами в PHP. Обробка GET та POST запитів.

Мета: Зрозуміти механізми передачі даних між клієнтом (браузером) та сервером. Опанувати роботу з суперглобальними масивами $_GET та $_POST у PHP. Навчитися приймати дані з HTML-форм, перевіряти їх на існування та проводити базову серверну санітизацію для захисту від XSS-атак.

Теоретичні відомості:

Коли користувач заповнює HTML-форму і натискає кнопку відправки, браузер формує HTTP-запит до сервера. Спосіб передачі цих даних визначається атрибутом method у тегу <form>. Існує два основні методи:

1. GET-запит: Дані передаються безпосередньо у вигляді параметрів в URL-адресі (наприклад, site.com/search.php?query=телефон). Цей метод підходить для пошукових запитів або фільтрації, оскільки таку сторінку можна додати в закладки. Однак він небезпечний для передачі паролів і має обмеження по довжині.

2. POST-запит: Дані передаються приховано у тілі самого HTTP-запиту. Вони не відображаються в адресному рядку браузера. Цей метод використовується для реєстрації, авторизації та передачі великих обсягів інформації.

У PHP для отримання цих даних існують спеціальні вбудовані змінні – суперглобальні масиви $_GET та $_POST. Вони є асоціативними масивами, де ключами виступають значення атрибутів name з HTML-полів форми (наприклад, <input name=«username»> буде доступно як $_POST['username']).

Оскільки даним від користувача ніколи не можна довіряти (користувач може ввести шкідливий JS-код у поле імені), перед виведенням цих даних на екран їх обов'язково потрібно очищати (санітизувати). Основна функція для цього в PHP – htmlspecialchars(), яка перетворює спеціальні HTML-символи (як-от < та >) на безпечні сутності.

Завдання до виконання:

Крок 1. Підготовка робочого середовища

Спочатку необхідно запустити локальний сервер, зокрема Apache у складі XAMPP. У папці htdocs слід створити теку з назвою web-lab-10-Прізвище та відкрити її у VS Code. Далі потрібно створити два файли: index.php, де буде розміщено HTML-форму, та process.php, який міститиме логіку обробки даних.

Крок 2. Створення HTML-форми у файлі index.php

У файлі index.php необхідно створити стандартну структуру HTML5-

документа. Слід додати заголовок другого рівня з текстом «Форма реєстрації (POST)». Після цього створюється форма з атрибутами action="process.php" та method="POST". У межах форми додаються такі поля введення: текстове поле для імені з атрибутом name="user_name" та обов'язковим заповненням (required), поле для електронної пошти з типом email, name="user_email" та required, поле для пароля з типом password, name="user_password" та required. У кінці форми розміщується кнопка відправлення з типом submit та текстом «Відправити».

Крок 3. Прийом та перевірка даних на сервері у файлі process.php

Файл process.php починається з відкриваючого тегу <?php. Перед обробкою даних слід переконатися, що запит надійшов саме методом POST. Для цього додається перевірка суперглобального масиву $_SERVER: якщо значення REQUEST_METHOD дорівнює POST, виконується обробка даних; в іншому випадку виводиться повідомлення про помилку.

Крок 4. Отримання та санітизація даних

Усередині блоку умови для POST отримуються дані з масиву $_POST. Для захисту від XSS-атак та очищення від зайвих пробілів застосовуються функції htmlspecialchars() та trim(). Ім'я, email та пароль обробляються відповідним чином; пароль з міркувань безпеки не виводиться на екран. Після цього виконується базова валідація: якщо будь-яке з полів порожнє, виводиться повідомлення про необхідність заповнення всіх полів. У разі успішної валідації виводиться повідомлення про реєстрацію з іменем та email користувача, а пароль позначається як прихований.

Крок 5. Тестування методу POST

Для тестування слід відкрити в браузері адресу http://localhost/web-lab-10-Прізвище/index.php, заповнити форму та натиснути кнопку відправлення. Після цього відбувається перехід на сторінку process.php, де відображається привітання з іменем користувача. Важливо звернути увагу, що в адресному рядку параметри форми не відображаються.

Крок 6. Експеримент з методом GET

Повернувшись до файлу index.php, слід створити другу форму для пошуку з методом GET. Її action також вказує на process.php. Форма містить одне текстове поле з name="search_query" та кнопку «Знайти». У файлі process.php під попереднім кодом додається логіка обробки GET-запиту: перевіряється наявність параметра search_query у масиві $_GET, після чого виконується санітизація та виведення пошукового запиту. При використанні цієї форми в адресному рядку з'являються параметри у форматі ?search_query=...

Крок 7. Збереження та відправка на GitHub

Після перевірки коректної роботи обох форм необхідно виконати комміт змін з повідомленням «Додано обробку форм POST та GET у PHP» та відправити код на віддалений репозиторій GitHub.

Крок 8. Оформлення звіту

Звіт оформлюється у форматі PDF. До нього додаються скриншоти коду файлів index.php та process.php, а також скриншоти браузера: заповнена форма до відправлення, результат обробки форми POST, результат обробки форми GET із чітко видимим адресним рядком, що містить параметри ?search_query=... . У звіті обов'язково вказується посилання на репозиторій.

Контрольні питання для захисту:

1. Визначте основні відмінності між HTTP-методами GET та POST.
2. Поясніть, для яких завдань категорично не рекомендується використовувати метод GET, та обґрунтуйте причини.
3. Охарактеризуйте суперглобальні масиви $_GET та $_POST у PHP, вказавши, до якого типу даних вони належать.
4. Розкрийте призначення функції htmlspecialchars() при обробці даних, отриманих від користувача, та наведіть приклад загрози, якій вона запобігає.
5. Опишіть спосіб перевірки у PHP, чи була відправлена форма (тобто чи існують дані у відповідному масиві) перед зверненням до $_POST.

ЛАБОРАТОРНА РОБОТА № 11

Тема: Керування станом на сервері: Сесії та Куки (Sessions & Cookies).

Мета: Ознайомитися з проблемою відсутності стану (stateless) у протоколі HTTP. Навчитися використовувати механізми сесій ($_SESSION) та куків ($_COOKIE) у PHP для збереження інформації про користувача. Реалізувати базовий функціонал авторизації: створення сесії, перевірка прав доступу до сторінки та коректне завершення сеансу (Logout).

Теоретичні відомості:

Протокол HTTP є stateless (без збереження стану) – це означає, що кожен новий запит від браузера до сервера є незалежним. Сервер «не пам'ятає», що ви вже заходили на сторінку секунду тому. Для вирішення цієї проблеми існують два основні інструменти:

1. Cookies (Куки): Це невеликі текстові файли, які сервер просить браузер зберегти на комп'ютері користувача. При кожному наступному запиті браузер автоматично відправляє ці куки назад на сервер. Вони підходять для довгострокових налаштувань (наприклад, вибір мови або темна тема).

2. Sessions (Сесії): Це механізм, де дані зберігаються на сервері, а браузеру видається лише унікальний ідентифікатор сесії (Session ID) через куки. Це набагато безпечніше для зберігання конфіденційних даних (логін користувача, статус адміністратора). Сесія зазвичай триває до закриття браузера.

У PHP для роботи з сесіями обов'язково потрібно викликати функцію session_start() на початку кожного файлу. Дані зберігаються у суперглобальному масиві $_SESSION.

Завдання до виконання:

Крок 1. Підготовка проєкту

У каталозі htdocs сервера створюється папка web-lab-11-Прізвище, в якій розміщуються три файли: login.php (форма входу), admin.php (захищена сторінка) та logout.php (скрипт виходу).

Крок 2. Реалізація форми входу (login.php)

Створюється HTML-форма з полями «Логін» та «Пароль». На початку файлу додається PHP-код, який ініціалізує сесію (session_start()) та перевіряє, чи не виконано вхід раніше: якщо в сесії існує ключ 'user', користувач перенаправляється на admin.php за допомогою header("Location: admin.php"). Далі перевіряється метод запиту: якщо це POST, отримані логін та пароль порівнюються з жорстко заданими значеннями (admin та 1234). У разі збігу в сесію записується змінна $_SESSION['user'] і знову виконується перенаправлення на admin.php. При невдалій спробі автентифікації формується змінна $error з повідомленням про помилку, яке виводиться під заголовком форми.

Крок 3. Створення захищеної сторінки (admin.php)

Сторінка призначена лише для автентифікованих користувачів. На початку файлу викликається session_start(), після чого перевіряється наявність ключа 'user' у масиві $_SESSION. Якщо його немає, виконується перенаправлення на login.php. У HTML-частині виводиться привітання з ім'ям користувача (наприклад, <h1>Вітаємо в панелі керування, <?php echo $_SESSION['user']; ?>!</h1>) та посилання на logout.php.

Крок 4. Робота з Cookies

У файлі admin.php додається код, який зчитує значення куки last_visit (якщо вона існує) та виводить повідомлення про час останнього візиту. Після цього встановлюється нова кука last_visit із поточним часом та терміном дії 30 днів за допомогою функції setcookie("last_visit", date("d-m-Y H:i:s"), time() + (86400 * 30), "/").

Крок 5. Скрипт виходу (logout.php)

У цьому файлі викликається session_start(), після чого сесія знищується функцією session_destroy(), і користувач перенаправляється на login.php за допомогою header("Location: login.php").

Крок 6. Тестування

Перевіряється робота механізму автентифікації: при спробі прямого доступу до admin.php через адресний рядок відбувається редірект на login.php. Введення правильних даних (admin / 1234) надає доступ до адмін-панелі. Оновлення сторінки admin.php демонструє зміну дати останнього візиту завдяки куці. Після натискання посилання «Вийти» сесія завершується, і доступ до захищеної сторінки блокується.

Крок 7. Збереження та відправка на GitHub

Файли проєкту додаються до системи контролю версій (git commit) із відповідним повідомленням, після чого зміни відправляються до віддаленого репозиторію.

Крок 8. Оформлення звіту

До звіту додаються скриншоти кодів трьох файлів, скриншот сторінки admin.php із відображенням імені користувача (з сесії) та дати останнього візиту (з куків), а також скриншот вкладки Application -> Cookies інструментів розробника браузера, де видно створені куки PHPSESSID та last_visit.

Контрольні питання для захисту:

1. Поясніть поняття сесії в PHP та вкажіть місце фізичного зберігання даних сесії.
2. Визначте відмінності між Cookie та Session, а також переваги використання кожного методу.
3. Обґрунтуйте призначення функції session_start() та вкажіть місце її виклику у файлі.
4. Охарактеризуйте функцію PHP для встановлення cookie та перелічіть її основні параметри.
5. Поясніть призначення функції header("Location: ...") та умови її коректного виконання.

ЛАБОРАТОРНА РОБОТА № 12

Тема: Взаємодія PHP з базами даних. Робота з MySQL та SQL-запитами.

Мета: Навчитися створювати бази даних та таблиці за допомогою інтерфейсу phpMyAdmin. Опанувати розширення mysqli для підключення до БД із PHP-скриптів. Навчитися виконувати основні операції з даними (CRUD): додавання записів (INSERT) та їх читання (SELECT).

Теоретичні відомості:

Для зберігання великих обсягів структурованої інформації (користувачі, товари, повідомлення) використовуються системи керування базами даних (СКБД). Найпопулярнішою для PHP є MySQL.

Взаємодія відбувається за допомогою мови запитів SQL (Structured Query Language). Основні команди:

- CREATE DATABASE / CREATE TABLE – створення структури.
- INSERT INTO – додавання даних.
- SELECT – отримання даних.

Для роботи з MySQL з PHP використовується об'єктно-орієнтований або процедурний інтерфейс mysqli. Процес роботи завжди складається з чотирьох етапів:

1. Встановлення з'єднання з сервером БД.
2. Формування та відправка SQL-запиту.
3. Отримання та обробка результатів.
4. Закриття з'єднання.

Завдання до виконання:

Крок 1. Створення бази даних у phpMyAdmin

Перед початком роботи необхідно запустити модулі Apache та MySQL у середовищі XAMPP. Після запуску слід відкрити у браузері інтерфейс phpMyAdmin за адресою http://localhost/phpmyadmin/. У цьому інструменті створюється нова база даних з іменем web_lab_db та кодуванням utf8mb4_general_ci. У межах створеної бази даних проєктується таблиця users, що містить три стовпці: id (тип INT, з позначками Auto Increment та Primary Key), name (тип VARCHAR довжиною 100) та email (тип VARCHAR довжиною 100). Після налаштування структури таблиця зберігається.

Крок 2. Підключення до бази даних (db.php)

У папці лабораторної роботи створюється файл db.php, який містить конфігурацію з'єднання з базою даних. У цьому файлі визначаються параметри сервера (localhost), ім'я користувача (root), пароль (порожній) та назва бази даних (web_lab_db). За допомогою функції mysqli_connect() встановлюється з'єднання, після чого виконується його перевірка: у разі помилки скрипт завершується з відповідним повідомленням.

Крок 3. Форма додавання користувача (index.php)

Створюється файл index.php, який реалізує інтерфейс для додавання користувачів та відображення списку наявних записів. На початку файлу за допомогою директиви include підключається створений раніше файл db.php.

Далі розміщується HTML-форма з полями введення імені (name) та електронної пошти (email). Обробник форми активується за наявності POST-запиту з ключем add_user. У ньому отримані дані проходять через функцію mysqli_real_escape_string() для запобігання SQL-ін'єкціям, після чого формується запит INSERT на додавання нового запису до таблиці users. Результат виконання запиту виводиться користувачеві у вигляді повідомлення про успішне додавання або опис помилки.

Крок 4. Виведення даних із бази на сторінку

Під формою у тому ж файлі index.php реалізується вибірка даних із таблиці users за допомогою SQL-запиту SELECT * FROM users. Отриманий результат обробляється: якщо кількість рядків більша за нуль, вони виводяться у вигляді невпорядкованого списку, де для кожного користувача зазначаються його ідентифікатор, ім'я та електронна пошта. Якщо записи відсутні, виводиться відповідне повідомлення. Наприкінці з'єднання з базою даних закривається функцією mysqli_close().

Крок 5. Тестування та робота з GitHub

Для перевірки коректності роботи створеного вебзастосунку необхідно відкрити файл index.php у браузері через локальний сервер. Через форму додаються кілька записів (наприклад, ім'я виконавця та його колег). Після надсилання форми слід переконатися, що нові дані одразу відображаються у списку нижче. Для підтвердження збереження даних на рівні бази даних можна переглянути вміст таблиці users у phpMyAdmin на вкладці «Огляд». Усі зміни фіксуються у системі контролю версій Git за допомогою комміту з відповідним повідомленням та відправляються до віддаленого репозиторію на GitHub.

Крок 6. Оформлення звіту

Підсумковий звіт має містити титульний аркуш, скриншоти коду файлів db.php та index.php, зображення структури таблиці users з інтерфейсу phpMyAdmin, а також скриншот вебсторінки, на якій відображено форму додавання та список користувачів, виведений із бази даних. До звіту додається посилання на репозиторій. Готовий звіт зберігається у форматі PDF.

Контрольні питання для захисту:

1. Поясніть призначення файлу db.php та опишіть процес встановлення з'єднання з базою даних MySQL за допомогою функції mysqli_connect().
2. Охарактеризуйте роль атрибутів Auto Increment та Primary Key для поля id у таблиці users та обґрунтуйте необхідність їх використання.
3. Розкрийте механізм захисту від SQL-ін'єкцій, застосований у коді обробника форми, зокрема функцію mysqli_real_escape_string().
4. Визначте відмінності між методами передачі даних GET та POST у

контексті обробки HTML-форм, наведіть приклади доцільності використання кожного з них.

5. Опишіть алгоритм виведення даних із таблиці users на вебсторінку: від формування SQL-запиту до ітерації по результаті вибірки за допомогою mysqli_fetch_assoc().

ЛАБОРАТОРНА РОБОТА № 13

Тема: Основи CSS-препроцесорів (SASS/SCSS). Змінні, вкладеність та міксини.

Мета: Ознайомитися з концепцією препроцесорів для оптимізації та автоматизації написання каскадних таблиць стилів. Навчитися налаштовувати середовище для компіляції SCSS у CSS. Опанувати використання змінних для створення єдиної дизайн-системи, застосовувати вкладені селектори (nesting) для покращення читабельності коду та створювати міксини (mixins) для повторного використання блоків стилів.

Теоретичні відомості:

CSS-препроцесор – це програма, яка має власний синтаксис (схожий на CSS, але з додатковими можливостями програмування) і генерує з нього звичайний CSS-код, який вже може прочитати браузер. Найпопулярнішим препроцесором є Sass (Syntactically Awesome Style Sheets).

Сьогодні найчастіше використовується синтаксис SCSS (Sassy CSS). Він повністю сумісний зі звичайним CSS (будь-який валідний CSS-код є валідним SCSS-кодом), але додає потужні інструменти:

-Змінні (Variables): Дозволяють зберігати кольори, шрифти або розміри в одному місці. Якщо колір бренду зміниться, достатньо змінити одну змінну, а не шукати її по всьому файлу.

-Вкладеність (Nesting): Дозволяє вкладати селектори один в один, візуально повторюючи ієрархію HTML-документа.

-Міксини (Mixins): Це своєрідні функції або шаблони. Вони дозволяють згрупувати кілька рядків CSS-коду і використовувати їх у різних місцях, передаючи власні параметри.

Оскільки браузери не розуміють файли .scss, їх обов'язково потрібно компілювати (перетворювати) у .css за допомогою спеціальних програм або розширень для редактора коду.

Завдання до виконання:

Крок 1. Підготовка середовища та компілятора

Спочатку необхідно створити папку з назвою web-lab-13-Прізвище та відкрити її у VS Code. Далі слід перейти до вкладки розширень, знайти та встановити розширення Live Sass Compiler. Після цього створюється базовий файл index.html, а також папка scss, усередині якої створюється файл style.scss. Відкривши файл style.scss, потрібно натиснути кнопку «Watch Sass» на нижній панелі VS Code, внаслідок чого розширення автоматично створить папку css із згенерованим файлом style.css. Нарешті, у файлі index.html у розділі <head> необхідно підключити згенерований CSS-файл за допомогою тегу <link>.

Крок 2. Робота зі змінними

У файлі style.scss на самому початку оголошуються змінні для кольорів та відступів, які позначаються символом долара. Прикладами таких змінних можуть бути $primary-color, $secondary-color, $text-color, $bg-light та $base-padding. Після оголошення ці змінні використовуються у базових стилях, зокрема для тіла документа задаються шрифт, колір фону та колір тексту із застосуванням відповідних змінних.

Крок 3. Використання вкладеності (Nesting)

У файлі index.html створюється навігаційне меню з використанням елементів nav, ul, li та a. У style.scss для стилізації цього меню застосовується принцип вкладеності: всередині селектора .main-nav розміщуються правила для ul, всередині ul – для li, а всередині li – для a. Для оформлення стану наведення на посилання використовується символ амперсанда (&), який дозволяє створити псевдоклас :hover для поточного елемента, забезпечуючи зміну кольору тексту та появу підкреслення.

Крок 4. Створення та використання міксинів

До файлу index.html додаються дві кнопки з класами btn-success та btn-danger. У style.scss створюється міксин з іменем button-style, який приймає параметр кольору фону та містить спільні стилі для кнопок. Цей міксин застосовується до класів btn-success та btn-danger за допомогою директиви @include, причому для кнопки Видалити передається інший колір фону безпосередньо під час виклику міксину.

Крок 5. Перевірка результату та збереження

Після написання коду слід відкрити згенерований файл style.css та переконатися, що компілятор правильно розгорнув усі вкладення у звичайні CSS-селектори та підставив значення замість змінних і міксинів. Важливо пам'ятати, що редагувати згенерований CSS-файл безпосередньо не слід. Далі запускається Live Server для перевірки застосування стилів у браузері. Після успішної перевірки виконується комміт змін та відправка файлів на GitHub, включаючи як вихідні SCSS-файли, так і згенеровані CSS-файли.

Крок 6. Оформлення звіту

Звіт оформлюється у форматі PDF. До нього додаються скриншот коду файлу style.scss із відображенням змінних, міксинів та вкладеності, скриншот згенерованого файлу style.css для демонстрації розуміння процесу компіляції, а також скриншот браузера з відкритою сторінкою, на якій видно меню та кнопки. Наприкінці вказується посилання на репозиторій.

Контрольні питання для захисту:

1. Дайте визначення CSS-препроцесора та обґрунтуйте доцільність його використання у сучасній веброзробці.
2. Опишіть механізм обробки браузером файлів формату .scss та поясніть процес компіляції.
3. Розкрийте переваги застосування змінних у SCSS порівняно з традиційним написанням фіксованих значень кольорів та розмірів у CSS.
4. Поясніть функціональне призначення символу амперсанда (&) у вкладених правилах SCSS та проаналізуйте наслідки його відсутності перед псевдокласом :hover.
5. Визначте поняття міксину (mixin) та вкажіть принципову відмінність між міксином і звичайним класом у CSS.

ЛАБОРАТОРНА РОБОТА № 14

Тема: Методологія BEM (Block, Element, Modifier). Компонентний підхід у верстці.

Мета: Ознайомитися з концепцією компонентного підходу до побудови веб-інтерфейсів. Опанувати правила методології BEM для іменування CSS-класів. Навчитися виділяти незалежні блоки на сторінці та стилізувати їх, поєднуючи синтаксис SCSS (зокрема батьківський селектор &) з номенклатурою BEM для уникнення дублювання коду та конфліктів стилів.

Теоретичні відомості:

У великих проєктах звичайне іменування класів (наприклад, .title або .btn) швидко призводить до хаосу та конфліктів стилів, коли один і той самий клас випадково використовується для різних елементів. Для вирішення цієї проблеми компанія Яндекс розробила методологію BEM (Блок, Елемент, Модифікатор).

BEM пропонує розділяти інтерфейс на незалежні компоненти і давати їм імена за чіткими правилами:

1. Блок (Block): Логічно незалежний компонент сторінки, який можна перенести в інше місце, і він не зламається (наприклад, картка товару, меню,

шапка). Іменується звичайним словом: .card, .nav.

2. Елемент (Element): Частина блоку, яка не має самостійного змісту і не може використовуватися поза своїм блоком (наприклад, заголовок картки, кнопка всередині картки). Відділяється від імені блоку двома підкресленнями __. Наприклад: .card__title, .card__button.

3. Модифікатор (Modifier): Прапорець, який змінює зовнішній вигляд, стан або поведінку блоку чи елемента (наприклад, «активна» кнопка, «велика» картка). Відділяється двома дефісами --. Наприклад: .card--featured, .card__button--active.

Завдяки SCSS писати класи за BEM дуже зручно. Символ амперсанда (&) компілятор замінює на ім'я батьківського селектора, що дозволяє не писати назву блоку щоразу:

SCSS

```
.card {
   // стилі блоку
   &__title { ... } // скомпілюється у .card__title
   &--dark { ... }  // скомпілюється у .card--dark
}
```

Завдання до виконання:

Крок 1. Підготовка робочого середовища

Для виконання лабораторної роботи необхідно створити окрему директорію з назвою web-lab-14-Прізвище та відкрити її в редакторі VS Code. У цій директорії створюється файл index.html, а також папка scss із файлом style.scss. За допомогою розширення Live Sass Compiler активується режим відстеження (кнопка «Watch Sass»), що забезпечує автоматичну компіляцію SCSS-файлів у CSS та генерує теку css із файлом style.css. Згенерований файл стилів підключається до HTML-документа стандартним способом.

Крок 2. Розмітка компонента (Блок та Елементи)

На даному етапі створюється картка профілю користувача з дотриманням методології BEM. У файлі index.html розміщується контейнер блоку – елемент <article> із класом profile-card. Усередині нього формуються елементи цього блоку, імена яких утворюються за схемою блок__елемент:

- зображення аватара (<img class="profile-card__avatar" src="..." alt="Аватар">);
- заголовок з іменем (<h2 class="profile-card__name">Іван Іваненко</h2>);
- текст опису (Фронтенд розробник);
- кнопка (<button class="profile-card__btn">Підписатися</button>).

Крок 3. Додавання модифікаторів

Для демонстрації можливостей модифікаторів створюється друга картка, яка відрізняється за дизайном (наприклад, для преміум-користувача). Код першої картки копіюється та розміщується нижче. До головного контейнера другої картки додається модифікатор блоку: клас profile-card--premium (підсумковий атрибут class має вигляд class="profile-card profile-card--premium"). Для кнопки другої картки застосовується модифікатор елемента profile-card__btn--active, а текст кнопки змінюється на «Відписатися».

Крок 4. Стилізація за допомогою SCSS та амперсанда (&)

У файлі style.scss описуються стилі для блоку profile-card та його елементів. Спочатку задаються базові властивості самої картки: рамка, внутрішні відступи, ширина, вирівнювання тексту, тінь. Для стилізації елементів використовується конструкція &__, яка розгортається компілятором у повний клас BEM. Приклад реалізації:

```
.profile-card {
  border: 1px solid #ccc;
  border-radius: 8px;
  padding: 20px;
  width: 250px;
  text-align: center;

  &__avatar {
    width: 100px;
    border-radius: 50%;
  }

  &__name {
    font-size: 1.2rem;
    color: #333;
  }

  &__btn {
    padding: 8px 16px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;

    &--active {
```

```
            background-color: #dc3545;
        }
    }

    &--premium {
        background-color: #fff3cd;
        border-color: #ffecb5;
    }
}
```

Крок 5. Перевірка та відправка на GitHub

Після завершення стилізації сторінка відкривається у браузері. Очікується, що дві картки матимуть різне оформлення завдяки застосованим модифікаторам: друга картка відрізнятиметься фоном і кольором кнопки. Для контролю рекомендується переглянути згенерований файл style.css і переконатися, що всі конструкції з амперсандом перетворилися на повноцінні селектори BEM (наприклад, .profile-card__btn--active). Після успішної перевірки зміни фіксуються коммітом із повідомленням «Додано компонент картки за методологією BEM» та відправляються на віддалений репозиторій GitHub.

Крок 6. Оформлення звіту

Звіт оформлюється у форматі PDF і містить такі обов'язкові елементи: тему роботи, мету, теоретичні відомості. До звіту додаються скриншоти: коду файла index.html із чітким відображенням класів за правилами BEM, вмісту файла style.scss (для демонстрації використання амперсанда), зовнішнього вигляду сторінки з двома картками профілю у браузері. Також обов'язково вказується посилання на репозиторій.

Контрольні питання для захисту:

1. Охарактеризуйте основну проблему CSS, яку вирішує методологія BEM, та обґрунтуйте, чому використання селекторів тегів або коротких класів (наприклад, .title) не є достатньо ефективним підходом.

2. Розкрийте зміст термінів «Блок», «Елемент» та «Модифікатор» відповідно до методології BEM, навівши власний приклад реалізації кожного з них.

3. Поясніть, чому згідно з правилами BEM не допускається створення класів виду .block__element1__element2 (вкладення елементів у назві), і які наслідки може мати порушення цього правила.

4. Опишіть роль символу амперсанда (&) у препроцесорі SCSS при написанні стилів за методологією BEM та вкажіть, у який результуючий селектор він розгортається компілятором.

5. Визначте, чи може елемент BEM існувати поза межами свого батьківського блоку, та аргументуйте свою відповідь з точки зору методології.

ЛАБОРАТОРНА РОБОТА № 15

Тема: Основи Git: Робота з гілками (Branching) та злиття (Merge).

Мета: Навчитися використовувати систему контролю версій Git як інструмент для паралельної розробки. Опанувати концепцію розгалуження (Branching), створення гілок для ізольованої розробки нових функцій (Feature Branch Workflow), перемикання між гілками, а також злиття (Merge) гілок в основну лінію розробки та вирішення конфліктів.

Теоретичні відомості:

У професійній розробці ніхто не пише код одразу в головну гілку проєкту (яка зазвичай називається main або master). Якщо кілька розробників одночасно будуть зберігати зміни в один файл, це призведе до хаосу.

Для вирішення цієї проблеми Git пропонує механізм гілок (branches). Гілка – це незалежна лінія розробки. Ви створюєте відгалуження від головної гілки, спокійно пишете свій код, тестуєте його, і лише коли функція повністю готова, «зливаєте» її назад у головну гілку. Цей підхід називається Feature Branch Workflow.

Основні команди для роботи з гілками:

- git branch – переглянути список існуючих гілок.
- git branch <назва_гілки> – створити нову гілку.
- git checkout <назва_гілки> (або сучасніший варіант git switch <назва_гілки>) – перемкнутися на вказану гілку.
- git checkout -b <назва_гілки> – створити нову гілку та одразу перемкнутися на неї.
- git merge <назва_гілки> – злити вказану гілку в ту гілку, на якій ви зараз знаходитеся.

Коли дві гілки змінюють один і той самий рядок коду по-різному, під час злиття виникає конфлікт (Merge Conflict). Git зупиняє злиття і просить розробника вручну обрати, який саме варіант коду потрібно залишити.

Завдання до виконання:

Крок 1. Підготовка базового репозиторію

Необхідно створити теку з назвою web-lab-15-Прізвище та відкрити її у VS Code. Після цього слід відкрити вбудований термінал (комбінація Ctrl + ~) та ініціалізувати репозиторій за допомогою команди git init. Далі створюється

файл index.html, до якого додається базова HTML-структура (за допомогою знаку оклику). Завершальним кроком цього етапу є виконання першого коміту в основну гілку main з повідомленням «Ініціалізація проєкту, базовий HTML».

Крок 2. Робота в першій гілці (Додавання шапки)

Для розробки шапки сайту створюється нова гілка feature-header, на яку негайно виконується перехід за допомогою команди git checkout -b feature-header. У файлі index.html всередині елемента <body> додається розмітка: <header><h1>Мій супер сайт</h1></header>. Після збереження файлу зміни індексуються та фіксуються комітом з повідомленням «Додано шапку сайту».

Крок 3. Робота в другій гілці (Додавання підвалу)

Спочатку виконується перемикання на головну гілку (git checkout main). На цьому етапі шапка сайту у файлі index.html тимчасово зникає, що є очікуваною поведінкою, оскільки гілка main ще не містить відповідних змін. Далі створюється нова гілка feature-footer, яка базується на чистому main (git checkout -b feature-footer). У тіло документа додається елемент <footer> Копірайт 2026 </footer>. Зміни фіксуються комітом з повідомленням «Додано підвал сайту».

Крок 4. Злиття гілок (Merge) без конфліктів

Для збирання завершеного сайту виконується перехід на головну гілку (git checkout main). Першим зливається гілка шапки за допомогою команди git merge feature-header. У терміналі з'являється повідомлення про успішне злиття, а шапка з'являється у файлі. Після цього аналогічним чином зливається гілка підвалу (git merge feature-footer). Оскільки шапка і підвал додавалися в різні ділянки файлу, Git автоматично виконує об'єднання без конфліктів, і в результаті документ містить обидва елементи.

Крок 5. Створення та вирішення конфлікту

Перебуваючи в гілці main, необхідно змінити заголовок сайту на <title>Головна сторінка</title> та закомітити цю зміну (git commit -am «Змінено title в main»). Далі створюється нова гілка feature-seo, у якій той самий тег змінюється на <title>SEO Заголовок - Найкращий сайт</title> з наступним комітом. Після повернення в main заголовок знову змінюється на <title>Офіційний сайт</title> та фіксується комітом. Спроба злиття гілки SEO (git merge feature-seo) призводить до виникнення конфлікту, про що сигналізує підсвічування файлу index.html червоним кольором. У VS Code відкривається файл з конфліктними рядками, де пропонуються варіанти вирішення: Accept Current Change (залишити поточні зміни з main), Accept Incoming Change (прийняти зміни з гілки SEO) або Accept Both Changes (залишити обидва варіанти). Необхідно обрати варіант Accept Incoming Change, зберегти файл, проіндексувати його та завершити злиття комітом з повідомленням «Вирішено

конфлікт з тегом title».

Крок 6. Відправка на GitHub

Створюється відповідний репозиторій на GitHub. Локальний репозиторій прив'язується до віддаленого за допомогою команди git remote add origin. Головна гілка відправляється командою git push -u origin main. Для демонстрації навичок роботи з гілками також відправляється будь-яка інша гілка, наприклад feature-header, командою git push origin feature-header.

Крок 7. Оформлення звіту

Звіт оформлюється у форматі PDF і має містити тему, мету роботи та теоретичні відомості. До звіту додаються скриншоти терміналу з командами створення гілок та успішного злиття, а також скриншот коду в момент конфлікту до його вирішення, де видно маркери <<<<<<< HEAD та =======. Обов'язково вказується посилання на створений репозиторій.

Контрольні питання для захисту:

1. Поясніть сутність поняття «гілка» (branch) у системі Git та її призначення в контексті командної роботи.
2. Охарактеризуйте команди, що використовуються для створення нової гілки та перемикання на неї.
3. Розкрийте суть процесу, що відбувається під час виконання команди git merge, та вкажіть, на якій гілці необхідно перебувати перед її запуском.
4. Визначте поняття конфлікту злиття (merge conflict) та причини його виникнення.
5. Поясніть значення маркерів <<<<<<< HEAD та >>>>>>> <назва_гілки>, які з'являються всередині файлу при виникненні конфлікту.

ЛАБОРАТОРНА РОБОТА № 16

Тема: Основи адаптивної графіки та SVG. Анімації CSS.

Мета: Ознайомитися з форматом векторної графіки SVG та його перевагами над растровими зображеннями. Навчитися вбудовувати SVG-код безпосередньо в HTML-документ та керувати його властивостями (кольором, контуром) за допомогою CSS. Опанувати створення плавних переходів (transition) та складних багатоетапних анімацій за допомогою правил @keyframes.

Теоретичні відомості:

У веб-розробці використовують два основні типи графіки: растрів (JPEG, PNG, GIF) та векторну (SVG). Растрові зображення складаються з пікселів і при

збільшенні втрачають якість (стають «квадратними»). Векторна графіка базується на математичних формулах (лініях, кривих, багатокутниках), тому вона ідеально масштабується до будь-яких розмірів без втрати якості і зазвичай важить набагато менше.

SVG (Scalable Vector Graphics) – це формат векторної графіки, заснований на мові розмітки XML. Це означає, що SVG-зображення – це просто текстовий код (теги <svg>, <circle>, <path> тощо). Якщо вставити цей код безпосередньо в HTML (Inline SVG), ви зможете змінювати його колір та розмір за допомогою звичайного CSS (властивості fill для заливки та stroke для контуру).

CSS Анімації бувають двох видів:

1. Транзиції (Transitions): Дозволяють плавно змінювати значення CSS-властивостей від одного стану до іншого (найчастіше при наведенні миші – :hover).

2. Ключові кадри (@keyframes): Дозволяють створювати складні анімації, що складаються з багатьох кроків (від 0% до 100% часу виконання), які можуть працювати безперервно, без втручання користувача.

Завдання до виконання:

Крок 1. Підготовка робочого середовища

На початку роботи необхідно створити папку з назвою web-lab-16-Прізвище та відкрити її в редакторі VS Code. У цій папці слід створити базові файли: index.html та папку css, у якій розмістити файл style.css. Після цього потрібно підключити створений файл стилів до HTML-документа та запустити Live Server для забезпечення автоматичного оновлення сторінки в процесі розробки.

Крок 2. Робота з Inline SVG

Для виконання цього етапу необхідно знайти в мережі безкоштовну векторну іконку, наприклад на сайті Heroicons або FontAwesome, та скопіювати її SVG-код. У файлі index.html всередині елемента <body> вставляється скопійований код, який починається з тегу <svg>. Цей тег слід обгорнути в посилання за допомогою елемента <a> з класом icon-link. Далі потрібно видалити атрибути fill та stroke всередині тегу <svg>, щоб уникнути конфлікту з CSS-стилями, та додати до тегу <svg> клас my-icon.

Крок 3. Стилізація SVG та плавні переходи (Transition)

У файлі style.css за допомогою селектора .my-icon задаються розміри іконки (ширина та висота 100 пікселів), колір заливки fill: #3498db та властивість transition, яка забезпечує плавну зміну кольору та трансформації протягом 0,3 секунди з функцією ease. Далі для селектора .icon-link:hover .my-icon визначаються зміни при наведенні: колір fill: #e74c3c та трансформація

scale(1.2), що збільшує іконку на 20%. Після написання стилів слід перевірити в браузері, чи при наведенні іконка плавно збільшується та змінює колір.

Крок 4. Створення складної анімації (Keyframes)

Для створення індикатора завантаження (спіннера), який постійно обертається, у HTML-документ додається елемент . У CSS-файлі для цього класу задаються розміри 50×50 пікселів, відступи, рамка завтовшки 5 пікселів світло-сірого кольору з верхньою частиною синього кольору, а також border-radius: 50% для утворення кола. За допомогою властивості animation запускається анімація з назвою spin тривалістю 1 секунда, лінійною функцією часу та нескінченним повторенням. У правилі @keyframes spin визначаються ключові кадри: на початку (0%) трансформація rotate(0deg), у кінці (100%) – rotate(360deg).

Крок 5. Створення анімації «м'ячика, що стрибає»

До HTML-документа додається елемент . У CSS для цього класу задаються стилі, що формують круглу форму (border-radius: 50%), колір фону, а також анімація bounce тривалістю 0,5 секунди з чергуванням напрямку (alternate), нескінченним повторенням та функцією часу ease-in. У @keyframes bounce визначаються кадри: на 0% трансформація translateY(0), на 100% – translateY(100px), що імітує падіння м'яча на 100 пікселів униз.

Крок 6. Збереження та відправка на GitHub

Після завершення розробки необхідно переконатися в коректній роботі всіх анімацій: спіннер має безперервно обертатися, м'яч – стрибати, а іконка SVG – реагувати на наведення. Далі виконується комміт змін із повідомленням «Додано SVG та CSS анімації» та здійснюється відправка репозиторію на GitHub.

Крок 7. Оформлення звіту

Звіт оформлюється у форматі PDF. До нього додаються скриншоти файлів index.html (з видимим кодом SVG) та style.css (з правилами @keyframes). Також слід зробити скриншоти браузера у двох станах: у спокійному стані та в момент наведення на SVG-іконку. Наприкінці звіту вказується посилання на репозиторій.

Контрольні питання для захисту:

1. Поясніть фундаментальну різницю між растровою та векторною графікою, а також визначте, для яких елементів вебсайту доцільніше використовувати формат SVG.

2. Назвіть дві основні CSS-властивості, що застосовуються для зміни кольору SVG-зображень замість стандартних властивостей color та background-

color.

3. Обґрунтуйте необхідність використання inline SVG (вставка коду безпосередньо в HTML) для стилізації через псевдоклас :hover, пояснивши, чому підключення через тег <img> не дає такого ефекту.

4. Розкрийте призначення CSS-властивості transition та опишіть параметри, які вона приймає для керування плавними змінами.

5. Поясніть роль директиви @keyframes у створенні анімацій та інтерпретуйте значення відсоткових точок (0%, 50%, 100%) у межах цього правила.

ЛАБОРАТОРНА РОБОТА № 17

Тема: Вступ до JavaScript ES6+: Деструктуризація, Spread/Rest, Модулі.

Мета: Ознайомитися з ключовими нововведеннями стандарту ECMAScript 2015 (ES6) та новіших версій. Навчитися розпаковувати дані з масивів та об'єктів за допомогою деструктуризації. Опанувати роботу з операторами Spread та Rest (...) для копіювання і об'єднання даних. Навчитися структурувати код, розбиваючи його на незалежні модулі за допомогою import та export.

Теоретичні відомості:

До 2015 року JavaScript мав певні обмеження у синтаксисі. Стандарт ES6 докорінно змінив мову, додавши такі інструменти:

1. Деструктуризація (Destructuring): Це спеціальний синтаксис, який дозволяє «розпакувати» властивості об'єкта або елементи масиву в окремі змінні одним рядком коду, замість того, щоб звертатися до них через крапку (user.name, user.age).

2. Spread (Оператор розширення ...): Дозволяє «розкласти» масив або об'єкт на окремі елементи. Це найзручніший спосіб скопіювати масив або об'єднати два об'єкти без мутації оригінальних даних.

3. Rest (Залишковий параметр ...): Виглядає так само як Spread, але робить протилежне – збирає безліч окремих елементів (наприклад, аргументів функції) у єдиний масив.

4. ES6 Модулі (Modules): Раніше весь JS-код доводилося писати в одному гігантському файлі або підключати купу <script> у правильному порядку. Модулі дозволяють експортувати (export) функцію з одного файлу і імпортувати (import) її в інший. Важливо: браузерні модулі працюють лише через HTTP-сервер (Live Server), а не при простому відкритті файлу.

Завдання до виконання:

Крок 1. Підготовка робочого середовища

На початку роботи необхідно створити папку з назвою web-lab-17-Прізвище та відкрити її у редакторі VS Code. У кореневій папці створюється файл index.html. Далі створюється папка js, всередині якої розміщуються два файли: main.js (призначений для головного скрипта) та mathUtils.js (модуль, що міститиме допоміжні функції). У файлі index.html здійснюється підключення головного скрипта з обов'язковим додаванням атрибута type="module", оскільки без нього браузер не зможе обробляти команди імпорту. Рядок підключення має вигляд: <script type="module" src="js/main.js"></script>.

Крок 2. Робота з модулями (export / import)

У файлі mathUtils.js створюється стрілкова функція для розрахунку знижки, яка одразу експортується. Функція приймає два параметри: ціну товару та відсоток знижки, після чого повертає кінцеву вартість з урахуванням знижки. У файлі main.js на самому початку виконується імпорт цієї функції з відносним шляхом './mathUtils.js'. Для перевірки працездатності модулів у консоль виводиться результат виконання функції з тестовими значеннями. Важливо, що сторінка має бути запущена через Live Server, інакше виникне помилка CORS, пов'язана з політикою безпеки браузера.

Крок 3. Деструктуризація об'єктів та масивів

Продовжуючи роботу у файлі main.js, створюється об'єкт користувача, який містить властивості імені, прізвища, віку та ролі. Замість традиційного звернення до властивостей через крапку, застосовується деструктуризація об'єкта: створюються окремі змінні firstName та role, яким присвоюються відповідні значення з об'єкта user. Далі створюється масив кольорів, після чого виконується деструктуризація масиву для отримання перших двох елементів у окремі змінні. Результати деструктуризації виводяться у консоль для перевірки.

Крок 4. Spread оператор (копіювання та об'єднання)

На цьому етапі створюються два масиви: fruits та berries. За допомогою spread-оператора (три крапки) вони об'єднуються у новий масив allFruits, до якого також додається додатковий елемент. Отриманий масив виводиться у консоль. Аналогічна операція виконується з об'єктами: створюється копія раніше створеного об'єкта user з одночасною зміною значення віку та додаванням нової властивості (міста). Це демонструє можливість створення нових об'єктів на основі існуючих із розширенням їхніх властивостей.

Крок 5. Rest параметр (збір аргументів)

Для демонстрації роботи rest-параметра створюється функція sumAll, яка здатна приймати будь-яку кількість аргументів. За допомогою rest-параметра (...numbers) усі передані аргументи збираються у масив, після чого метод reduce

обчислює їхню суму. Функція викликається з чотирма числовими аргументами, а результат її роботи виводиться у консоль.

Крок 6. Збереження та звіт

Після виконання всіх попередніх кроків необхідно переконатися, що у консолі браузера відображаються результати роботи всіх скриптів без помилок. Виконані зміни зберігаються у репозиторії GitHub з відповідним комітом. Для оформлення звіту створюється документ у форматі PDF, який містить тему роботи, мету, теоретичні відомості, скриншоти вмісту файлів main.js та mathUtils.js, скриншот консолі браузера з результатами виконання коду, а також посилання на створений репозиторій.

Контрольні питання для захисту:

1. Визначте поняття деструктуризації та поясніть, яку проблему вона вирішує при роботі з об'єктами, навівши відповідний приклад.
2. Поясніть відмінність у поведінці spread-оператора та rest-параметра, враховуючи, що обидва позначаються однаково (три крапки), але використовуються в різних контекстах.
3. Охарактеризуйте причину виникнення помилки CORS при спробі використання ES6-модулів (import/export) під час відкриття HTML-файлу безпосередньо з файлової системи (за протоколом file://) та вкажіть спосіб її усунення.
4. Назвіть обов'язковий атрибут, який необхідно додати до тегу <script> у HTML-документі для забезпечення коректної роботи модульної системи JavaScript.
5. Розкрийте щонайменше три переваги використання модульного підходу до організації коду порівняно зі зберіганням усієї логіки в одному великому файлі.

ЛАБОРАТОРНА РОБОТА № 18

Тема: Тестування JavaScript коду (Unit Testing). Основи фреймворку Jest.

Мета: Ознайомитися з концепцією модульного тестування (Unit Testing). Зрозуміти важливість автоматизованої перевірки коду перед розгортанням проєкту. Навчитися ініціалізувати проєкт за допомогою NPM (Node Package Manager), встановлювати тестовий фреймворк Jest та писати базові тести для перевірки правильності роботи JavaScript-функцій.

Теоретичні відомості:

Коли проєкт стає великим, ручна перевірка кожної функції в браузері

(через console.log або кліки по кнопках) стає неможливою. Зміна одного рядка коду може непомітно зламати іншу частину сайту. Для вирішення цієї проблеми пишуть автоматичні тести – окремі скрипти, які перевіряють, чи правильно працює ваш основний код.

Найбазовіший рівень тестування – це Unit Testing (модульне тестування). Його суть полягає в тому, щоб взяти найменшу неподільну частину програми (зазвичай це одна функція), передати їй певні вхідні дані і перевірити, чи збігається результат з очікуваним.

Для JavaScript стандартом де-факто є фреймворк Jest (створений Facebook). Він не працює безпосередньо в браузері – для його запуску потрібне середовище Node.js та пакетний менеджер NPM, який дозволяє завантажувати сторонні бібліотеки в проєкт.

Анатомія тесту в Jest складається з трьох основних блоків:

1. describe('назва', () => {...}) – групує кілька пов'язаних тестів.

2. test('опис', () => {...}) (або it) – безпосередньо сам тест.

3. expect(результат).toBe(очікування) – твердження (assertion), яке перевіряє значення.

Завдання до виконання:

Крок 1. Встановлення Node.js та ініціалізація проєкту

На початку роботи необхідно переконатися, що на комп'ютері встановлено Node.js, виконавши в терміналі команду node -v для перевірки версії. Далі створюється окрема папка з назвою web-lab-18-Прізвище, яка відкривається у редакторі VS Code. У відкритому терміналі VS Code виконується команда npm init -y, яка генерує файл package.json – основний конфігураційний файл, що містить метадані проєкту. Після цього встановлюється фреймворк Jest як залежність для розробки за допомогою команди npm install --save-dev jest. Завершальним кроком цього етапу є редагування файлу package.json: у секції "scripts" потрібно змінити значення команди "test" на "jest", що дозволить запускати тести командою npm test.

Крок 2. Створення модуля для тестування

Для подальшого тестування створюється файл mathUtils.js. У цьому файлі реалізуються дві прості функції:

- функція add(a, b), яка повертає суму двох чисел;
- функція isEven(number), яка перевіряє, чи є число парним (повертає true або false).

Обидві функції експортуються за допомогою синтаксису CommonJS (module.exports = { add, isEven }), який за замовчуванням підтримується Node.js та Jest.

Крок 3. Написання тестів

Jest автоматично виявляє файли з розширенням .test.js або файли, розташовані в теці __tests__. Тому створюється файл mathUtils.test.js. У ньому за допомогою require імпортуються функції з модуля mathUtils.js. Далі створюється тестовий набір (test suite) з використанням функції describe, який об'єднує групу тестів для математичних функцій. У межах цього набору за допомогою функції test формується два тестових випадки:

- перший перевіряє коректність додавання для різних вхідних значень, використовуючи метод expect().toBe();
- другий перевіряє роботу функції визначення парності за допомогою методів toBeTruthy() та toBeFalsy().

Таким чином, структура тестового файлу має вигляд:

```
const { add, isEven } = require('./mathUtils');

describe('Тестування математичних функцій', () => {
  test('Функція add повинна правильно додавати два числа', () => {
    expect(add(2, 3)).toBe(5);
    expect(add(-1, 1)).toBe(0);
  });

  test('Функція isEven повинна повертати true для парних чисел', () => {
    expect(isEven(4)).toBeTruthy();
    expect(isEven(7)).toBeFalsy();
  });
});
```

Крок 4. Запуск тестів та аналіз результатів

Для виконання тестів у терміналі вводиться команда npm test. У разі успішного проходження всіх перевірок у консолі з'являється зелений звіт PASS. Для демонстрації діагностичних можливостей Jest навмисно вноситься помилка у функцію add у файлі mathUtils.js (наприклад, заміна оператора + на -). Повторний запуск npm test призводить до появи червоного повідомлення FAIL, яке містить детальну інформацію про розбіжність між очікуваним (Expected) та отриманим (Received) результатами. Після аналізу помилка виправляється, і тести знову виконуються успішно.

Крок 5. Збереження та підготовка звіту

Перед відправкою коду на GitHub створюється файл .gitignore, до якого додається рядок node_modules/. Це необхідно, оскільки директорія node_modules не повинна зберігатися у репозиторії (вона може бути відтворена командою npm install). Після цього виконується комміт змін та завантаження

проєкту на віддалений репозиторій. Для оформлення звіту у форматі PDF необхідно додати: титульний аркуш, скриншоти вмісту файлів mathUtils.js та mathUtils.test.js, скриншот терміналу з результатами успішного виконання тестів (зелений лог Jest) та пряме посилання на створений репозиторій.

Контрольні питання для захисту:

1. Визначте поняття модульного тестування (Unit Testing) та поясніть, яку ключову проблему процесу розробки програмного забезпечення воно вирішує.
2. Поясніть призначення файлу package.json у проєктах на базі Node.js та опишіть його основну роль.
3. Розкрийте функціональне призначення ключових слів (функцій) describe, test та expect у контексті роботи фреймворку Jest.
4. Охарактеризуйте значення прапорця --save-dev при встановленні пакетів через менеджер залежностей NPM.
5. Обґрунтуйте необхідність додавання папки node_modules до файлу .gitignore та опишіть механізм, за допомогою якого інший розробник зможе розгорнути проєкт і запустити його без цієї папки.

ЛАБОРАТОРНА РОБОТА № 19

Тема: Безпека вебзастосунків. Захист від SQL Injection та XSS-атак.

Мета: Зрозуміти природу найпоширеніших вразливостей вебзастосунків за класифікацією OWASP. Навчитися реалізовувати захист від міжсайтового скриптингу (XSS) за допомогою санітизації даних. Опанувати механізм підготовлених запитів (Prepared Statements) у PHP для повного захисту бази даних від SQL-ін'єкцій.

Теоретичні відомості:

У світі веброзробки існує золоте правило: «Ніколи не довіряйте даним від користувача». Все, що приходить із форм (POST/GET запити), може містити шкідливий код.

1. XSS (Cross-Site Scripting): Це вразливість, при якій зловмисник вводить у форму JavaScript-код (наприклад, <script>alert('Зламано!');</script>). Якщо сервер просто збереже це і виведе іншим користувачам (наприклад, у коментарях), їхні браузери виконають цей код, що може призвести до крадіжки сесій (куків) або паролів.

2. SQL Injection (SQL-ін'єкція): Це атака на базу даних. Якщо сервер підставляє введені користувачем дані напряму в рядок SQL-запиту, хакер може ввести спеціальні символи (наприклад, 1' OR '1'='1), що докорінно змінить

логіку запиту. Це дозволяє обійти авторизацію, видалити таблиці або вкрасти всі дані.

Рішення:

-Від XSS рятує функція PHP htmlspecialchars(), яка перетворює небезпечні символи (як < і >) на безпечні HTML-сутності (< і >).

-Від SQL-ін'єкцій на 100% рятують Підготовлені запити (Prepared Statements). У них структура запиту (шаблон) відправляється на сервер бази даних *окремо* від самих даних.

Завдання до виконання:

Крок 1. Підготовка робочого середовища та бази даних

Спочатку необхідно запустити модулі Apache та MySQL у середовищі XAMPP. Після цього створюється робоча папка з назвою web-lab-19-Прізвище. За допомогою інструменту phpMyAdmin слід створити таблицю secret_users, яка міститиме поля id, login та password, а також додати до неї одного тестового користувача (наприклад, з логіном admin та паролем qwerty).

Крок 2. Демонстрація вразливості до XSS

Наступним кроком створюється файл xss.php, який містить форму для додавання коментарів з методом відправки POST. У ньому реалізується вразливий PHP-обробник, що безпосередньо виводить введені користувачем дані без будь-якої обробки. При введенні у форму рядка <script>alert('Вас зламано!');</script> відбувається виконання JavaScript-коду, що проявляється у вигляді спливаючого вікна, чим демонструється наявність вразливості.

Крок 3. Захист від XSS

Для усунення виявленої вразливості обробник модифікується шляхом додавання функції htmlspecialchars() з параметрами ENT_QUOTES та UTF-8. Ця функція перетворює спеціальні символи HTML на відповідні сутності, що унеможливлює виконання введеного скрипта. При повторній спробі введення шкідливого коду останній відображається як звичайний текст, що підтверджує ефективність захисту.

Крок 4. Захист від SQL Injection за допомогою підготовлених запитів

Створюється файл login.php з формою авторизації для введення логіна та пароля. Після встановлення з'єднання з базою даних реалізується безпечний обробник авторизації з використанням підготовлених запитів. Спочатку створюється шаблон SQL-запиту із знаками питання замість безпосередніх значень, після чого змінні прив'язуються до цього шаблону за допомогою методу bind_param() із зазначенням типів даних. Після виконання запиту отримується результат, на основі якого формується відповідне повідомлення. Навіть при спробі введення шкідливого рядка на кшталт admin' OR '1'='1 він

інтерпретується буквально як значення, а не як частина SQL-команди, що робить атаку неефективною.

Крок 5. Оформлення звіту

Виконані розробки зберігаються у системі контролю версій Git шляхом створення комміту з відповідним повідомленням та відправляються на віддалений репозиторій GitHub. Для звіту готується документ у форматі PDF, до якого додаються скриншоти файлів xss.php та login.php з акцентом на використанні функції htmlspecialchars() та методу prepare(). Також обов'язково вказується посилання на створений репозиторій.

Контрольні питання для захисту:

1. Поясніть принцип дії атаки Cross-Site Scripting (XSS) та визначте основну мету зловмисника при її реалізації.
2. Опишіть механізм нейтралізації небезпечного JavaScript-коду за допомогою функції htmlspecialchars().
3. Розкрийте сутність SQL-ін'єкції та наведіть приклад небезпечного рядка, який може бути використаний хакером у формі авторизації.
4. Поясніть концепцію підготовлених запитів (Prepared Statements) та обґрунтуйте, чому вони вважаються найнадійнішим способом захисту від SQL-ін'єкцій.
5. Визначте призначення символів "?" (знаків питання) у SQL-шаблоні при використанні функції prepare().

СПИСОК РЕКОМЕНДОВАНОЇ ТА ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Архітектура сучасних вебзастосунків: REST API, GraphQL та мікросервіси: конспект лекцій. Львів: НУ «Львівська політехніка», 2024. 190 с.
2. Бородкіна І. Л., Бородкін Г. О. WEB-технології та WEB-дизайн: застосування мови HTML для створення електронних ресурсів. Київ : Ліра-К, 2022. 212 с.
3. Босько В. В., Константинова Л. В., Марченко К. М., Улічев О. С. Web-програмування. Частина 1 (frontend) : навч. посіб. Кропивницький: ЦНТУ, 2022. 208 с. URL:https://dspace.kntu.kr.ua/server/api/core/bitstreams/2bfb5a3c-54d9-4283-89c1-5e190e8aa151/content
4. Брюханова Г. Комп'ютерні дизайн-технології : навчальний посібник. Київ : Центр учбової літератури, 2021. 180 с.
5. Двірничук К. В., Вацек Д. О. Веб-програмування та веб-дизайн : навч. посіб. Чернівці : Чернівец. нац. ун-т ім. Ю. Федьковича, 2022. 472 с. URL : https://files.znu.edu.ua/files/Bibliobooks/Inshi74/0054410.pdf
6. Документація Docker: Контейнеризація інфраструктури для розробників. *Docker Inc.*, 2025. URL: https://docs.docker.com/.
7. Евристики юзабіліті та проєктування користувацького досвіду (UX). *Nielsen Norman Group*, 2025. URL: https://www.nngroup.com/articles/ten-usability-heuristics/.
8. Лобода Ю. Г. Веб-технології та веб-дизайн: навч.-метод. посіб. для здобувачів вищої освіти галузі знань 12 «Інформаційні технології» / уклад.: Ю. Г. Лобода, А. А. Толокнов ; Нац. ун-т «Одеська юрид. академія». Одеса : Фенікс, 2023. 260 с. URL: https://dspace.onua.edu.ua/items/dd3015d5-b60a-4152-b6d6-00c8250f67b7
9. Налаштування подій та веб-аналітика: Документація Google Analytics 4 (GA4). *Google Довідка*, 2025. URL: https://support.google.com/analytics/.
10. Оптимізація вебпродуктивності та показники Core Web Vitals. *Google for Developers (web.dev)*, 2025. URL: https://web.dev/explore/fast.
11. Основи веброзробки : навчально-методичний посібник / уклад. Г. Є. Заволодько, О. В. Касілов, С. В. Бронін. Харків : НТУ «ХПІ», 2025. 322 с. https://repository.kpi.kharkov.ua/server/api/core/bitstreams/735f2aa3-ed27-419f-90a4-4a5658379f41/content
12. Офіційна документація React: Створення користувацьких інтерфейсів та SPA. *Meta Platforms*, 2025. URL: https://react.dev/.
13. Офіційне керівництво з PHP: документація (розділи бази даних PDO та керування сесіями). *The PHP Group*, 2025. URL: https://www.php.net/manual/uk/.
14. Пасічник В. В., Пасічник О. В. Веб-дизайн : підручник. Львів : Магнолія-2006, 2024. 520 с.

15. Пасічник В. В., Пасічник О. В., Угрин Д. І. Веб-технології та веб-дизайн. Книга 1. Веб-технології : підручник. Львів : Магнолія, 2021. 336 с.
16. Пастернак І. І., Костик А. Т. Інструментальні засоби веб-технологій : навч. посібник. Львів : Магнолія, 2024. 197 с. Баран С. В. Основи web-програмування : навчальний посібник. Кривий Ріг : Державний університет економіки і технологій, 2023. 316 с. URL: https://dspace.duet.edu.ua/jspui/handle/123456789/832
17. Пустюльга С. І., Самчук В. П. Технології вебдизайну : навчальний посібник. Луцьк : Вежа, 2023. 604 с. https://surl.li/pxaqwp
18. Сучасні тенденції веброзробки та інтеграція ШІ-інструментів (Copilot, Cursor): онлайн-курс. *Prometheus*, 2025. URL: https://prometheus.org.ua/courses/modern-web-dev.
19. Ушенко Ю.О ., Олар О. В., Галочкін О. В., Д'яченко Л. І. Сучасні технології розробки web-додатків: Фронтенд розробка : навч. посібник. Чернівці : Чернівецький нац. ун-т, 2022. 222 с.
20. Юрчак І. Ю., Гузинець Н. В. Базові засади веб-розробки : навчальний посібник. Львів : Магнолія, 2023. 180 с.
21. MDN Web Docs: Адаптивний вебдизайн, CSS Grid та Flexbox. *Mozilla Foundation*, 2025. URL: https://developer.mozilla.org/uk/docs/Learn/CSS/CSS_layout/Responsive_Design.

Навчальне видання

ВЕБТЕХНОЛОГІЇ ТА ВЕБДИЗАЙН

Методичні рекомендації

Укладачі: **Тищенко** Світлана Іванівна
Пархоменко Олександр Юрійович
Ємельянов Святослав Ігорович
Кучмійова Тетяна Сергіївна
Жебко Олександр Олегович
Богатєнкова Олександра Євгенівна
Коломієць Андрій Миколайович

Формат 60х84 1/16. Ум. друк. арк. 5,6.
Наклад 50 прим. Зам. № _____

Надруковано у видавничому відділі
Миколаївського національного аграрного університету
54020, м. Миколаїв, вул. Георгія Гонгадзе, 9

Свідоцтво суб'єкта видавничої справи ДК № 4490 від 20.02.2013 р.