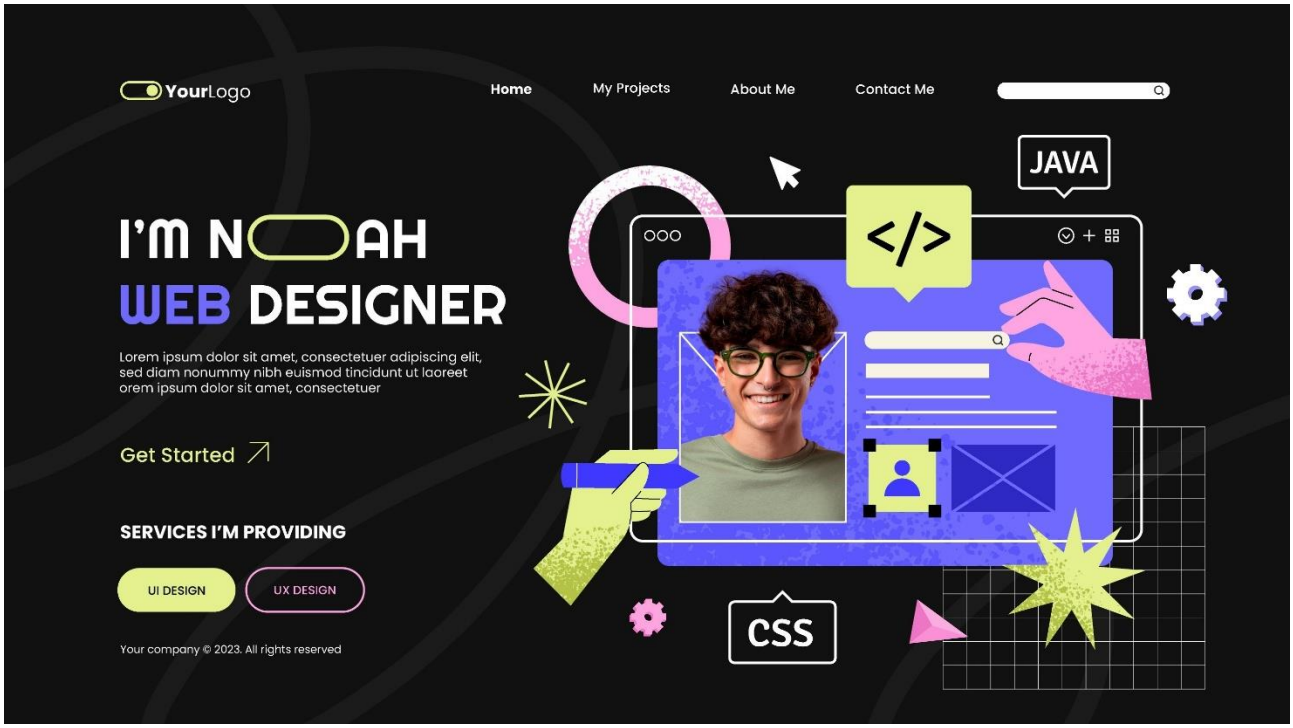
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

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

ВЕБТЕХНОЛОГІЇ ТА ВЕБДИЗАЙН

Конспект лекцій
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 5
ЗМІСТОВИЙ МОДУЛЬ 1. ОСНОВИ ВЕБ-РОЗРОБКИ ТА ВЕБ-ДИЗАЙНУ 7
ТЕМА 1.1. ПРИНЦИПИ ФУНКЦІОНУВАННЯ ІНТЕРНЕТУ ТА БАЗОВА АРХІТЕКТУРА ВЕБЗАСТОСУНКІВ 7
ТЕМА 1.2. HTML5: ОСНОВА СТРУКТУРИ ВЕБ-СТОРІНКИ..... 11
ТЕМА 1.3. CSS3: ПРИНЦИПИ ОФОРМЛЕННЯ ТА ВЕРСТКИ..... 15
ТЕМА 1.4. АДАПТИВНИЙ ТА МОБІЛЬНИЙ ДИЗАЙН, РОБОТА З МЕДІА-КОНТЕНТОМ 19
ЗМІСТОВИЙ МОДУЛЬ 2. JAVASCRIPT 24
ТЕМА 2.1. JAVASCRIPT: СИНТАКСИС, ТИПИ ДАНИХ ТА ОСНОВИ РОБОТИ 24
ТЕМА 2.2. ФУНКЦІЇ, ЗАМИКАННЯ ТА ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ 27
ТЕМА 2.3. РОБОТА З DOCUMENT OBJECT MODEL (DOM)..... 31
ТЕМА 2.4. ПОДІЇ ТА ІНТЕРАКТИВНІСТЬ НА КЛІЄНТСЬКІЙ СТОРОНІ..... 35
ЗМІСТОВИЙ МОДУЛЬ 3. ОСНОВИ PHP 39
ТЕМА 3.1. ВСТУП ДО PHP ТА СЕРВЕРНОГО ПРОГРАМУВАННЯ 39
ТЕМА 3.2. УПРАВЛЯЮЧІ КОНСТРУКЦІЇ ТА ЦИКЛИ В PHP..... 43
ТЕМА 3.3. МАСИВИ, РЯДКИ ТА ЇХ ОБРОБКА 47
ТЕМА 3.4. ФУНКЦІЇ ТА ОРГАНІЗАЦІЯ КОДУ В PHP 51
ЗМІСТОВИЙ МОДУЛЬ 4. ВЗАЄМОДІЯ КЛІЄНТА ТА СЕРВЕРА..... 56
ТЕМА 4.1. ОБРОБКА ФОРМ ТА ДАНИХ КОРИСТУВАЧА..... 56

ТЕМА 4.2. СЕСІЇ ТА COOKIES: УПРАВЛІННЯ СТАНОМ КОРИСТУВАЧА 60
ТЕМА 4.3. ОСНОВИ РЕЛЯЦІЙНИХ БАЗ ДАНИХ ТА МОВА SQL .. 63
ТЕМА 4.4. ПІДКЛЮЧЕННЯ PHP ДО БАЗИ ДАНИХ (PDO / MYSQLI) 67
ЗМІСТОВИЙ МОДУЛЬ 5. СУЧАСНІ ПІДХОДИ ДО ВЕБДИЗАЙНУ, ОПТИМІЗАЦІЇ ТА ВЕБРОЗРОБКИ 72
ТЕМА 5.1. ОСНОВИ UX/UI ДИЗАЙНУ ТА ПРИНЦИПИ СТВОРЕННЯ ЗРУЧНИХ ІНТЕРФЕЙСІВ 72
ТЕМА 5.2. АДАПТИВНИЙ ТА МОБІЛЬНИЙ ДИЗАЙН: СУЧАСНІ ТЕХНІКИ ТА ТРЕНДИ 75
ТЕМА 5.3. ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ВЕБ-САЙТІВ ТА ВЕБ-АНАЛІТИКА 79
ТЕМА 5.4. СУЧАСНІ ІНСТРУМЕНТИ ТА ТЕНДЕНЦІЇ ФРОНТЕНД- ТА БЕКЕНД-РОЗРОБКИ 82
СПИСОК РЕКОМЕНДОВАНОЇ ТА ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 87

ПЕРЕДМОВА

Курс дисципліни «Вебтехнології та вебдизайн» призначений для забезпечення здобувачів вищої освіти спеціальності F3 (122) «Комп'ютерні науки» фундаментальними теоретичними знаннями та практичними навичками у сфері проєктування, розробки, розгортання та супроводу сучасних вебзастосунків. Курс має на меті 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 та 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 та 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 адаптивного та чуйного вебдизайну (Responsive Web Design), а також інструментальні засоби проєктування інтерфейсів.

Викладання дисципліни ставить за мету сформувати у здобувачів вищої освіти інженерне та системне мислення у галузі веброзробки, розвинути навички проєктування архітектури вебзастосунків, вибору оптимальних технологічних стеків (Frontend + Backend) для вирішення комплексних завдань, а також виробити розуміння стандартів юзабіліті, вебдоступності (a11y) та кібербезпеки у вебінфраструктурі.

Основними завданнями, що мають бути вирішені у процесі викладання дисципліни, є:

– навчити розуміти базові концепції, класифікацію та багаторівневу архітектуру сучасних вебзастосунків і мережевих протоколів (HTTP/HTTPS);

– надати практичні навички розробки семантичних, адаптивних та кросбраузерних інтерфейсів з використанням сучасних стандартів HTML5 та CSS3;

– ознайомити з принципами клієнтського програмування (JavaScript) для створення динамічних елементів, маніпуляції DOM-деревом та асинхронного обміну даними (AJAX/Fetch);

– сформувати вміння розробляти серверну логіку (PHP), налаштовувати маршрутизацію та організовувати ефективну і безпечну взаємодію з базами

даних (MySQL);

– навчити застосовувати сучасні принципи UI/UX дизайну для проєктування користувачоцентричних та ергономічних інтерфейсів (за допомогою інструментів типу Figma);

– сформувати практичні навички тестування, оптимізації продуктивності вебсторінок, забезпечення базового захисту від вебвразливостей та розгортання готових проєктів на сервері (хостингу).

ЗМІСТОВИЙ МОДУЛЬ 1. ОСНОВИ ВЕБ-РОЗРОБКИ ТА ВЕБ-ДИЗАЙНУ

ТЕМА 1.1. ПРИНЦИПИ ФУНКЦІОНУВАННЯ ІНТЕРНЕТУ ТА БАЗОВА АРХІТЕКТУРА ВЕБЗАСТОСУНКІВ

Мета: сформувати у здобувачів вищої освіти системне розуміння принципів роботи глобальної мережі Інтернет, архітектури «клієнт-сервер», стеку мережевих протоколів, системи адресації та детального життєвого циклу обробки вебзапиту.

План лекції:

1. Еволюція та базова термінологія: Інтернет проти World Wide Web (WWW).
2. Архітектура «Клієнт-Сервер» та багаторівневі системи (N-Tier).
3. Протоколи передачі даних: HTTP та HTTPS (Анатомія запитів та відповідей).
4. IP-адресація та ієрархія системи доменних імен (DNS).
5. Деталізований життєвий цикл відображення вебсторінки (Critical Rendering Path).

1. Еволюція та базова термінологія: Інтернет проти World Wide Web (WWW)

Для фахівця з комп'ютерних наук критично важливо розрізняти базові поняття, які пересічні користувачі часто вважають синонімами.

- Інтернет (Internet – Interconnected Networks): Це глобальна апаратна та логічна інфраструктура, величезна мережа комп’ютерних мереж. Вона базується на принципі пакетної комутації даних і використовує стандартизований стек протоколів TCP/IP. Інтернет з'явився наприкінці 1960-х років (проєкт ARPANET) і забезпечує роботу багатьох сервісів: електронної пошти (SMTP/POP3), передачі файлів (FTP), IP-телефонії та, власне, Вебу.
- World Wide Web (WWW, Веб, Всесвітня павутина): Це інформаційний простір (служба), побудований поверх Інтернету. Його винайшов Тім Бернерс-Лі у 1989 році в CERN. Веб базується на трьох основних технологіях:

1. URL (Uniform Resource Locator) – система унікальних адрес для ідентифікації ресурсів.
2. HTTP (HyperText Transfer Protocol) – протокол передачі даних.
3. HTML (HyperText Markup Language) – мова гіпертекстової розмітки документів.

2. Архітектура «Клієнт-Сервер» та багаторівневі системи

Більшість сучасних вебзастосунків функціонують на основі архітектури «клієнт-сервер», яка розподіляє обчислювальне навантаження.

- Клієнт (Frontend): Програма (найчастіше веббраузер: Chrome, Firefox, Safari), яка працює на пристрої користувача. Її головні завдання: формування правильного запиту до сервера, отримання відповіді, парсинг коду (HTML/CSS/JS) та рендеринг (відмальовування) графічного інтерфейсу (UI).

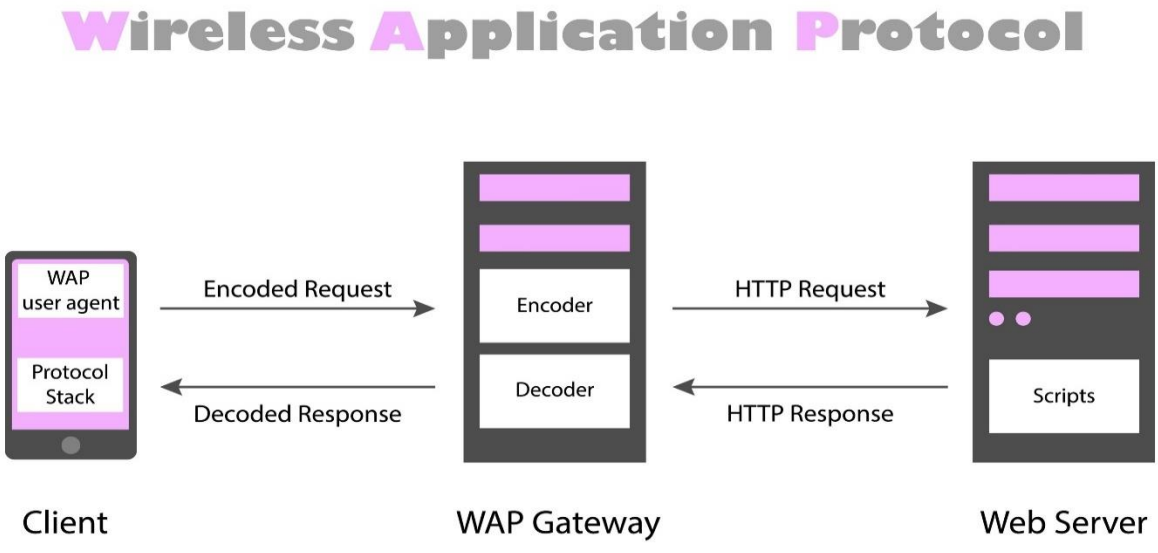
- Сервер (Backend): Потужний апаратно-програмний комплекс, який безперервно (24/7) прослуховує визначені порти (наприклад, порт 80 для HTTP або 443 для HTTPS). Отримавши запит, сервер маршрутизує його, виконує бізнес-логіку (за допомогою мов PHP, Python, Node.js), звертається до бази даних (MySQL, PostgreSQL) і формує відповідь.

Еволюція до N-рівневої (N-Tier) архітектури: Сучасні складні системи (наприклад, електронна комерція або ERP-системи) рідко використовують просту двоярусну модель. Вони застосовують трирівневу (3-Tier) архітектуру:

1. Рівень представлення (Presentation Layer): Браузер клієнта (UI/UX).
2. Рівень застосунку (Application Layer): Вебсервер, де виконується бізнес-логіка програми (API).
3. Рівень даних (Data Layer): Окремий сервер бази даних (СУБД), який відповідає виключно за збереження, цілісність та швидкий пошук інформації.

3. Протоколи передачі даних: HTTP та HTTPS

Взаємодія між клієнтом і сервером відбувається у форматі транзакцій «Запит – Відповідь» за протоколом HTTP. Це протокол *без збереження стану (stateless)*: кожен новий запит розглядається сервером як незалежний (саме тому для ідентифікації користувачів були придумані Cookies та сесії).



Анатомія HTTP-запиту

HTTP-запит, що ініціюється клієнтом (як правило, веб-браузером або іншим мережевим додатком), структурно складається з чотирьох основних компонентів.

1. Метод, який визначає операцію, котру клієнт має намір виконати над ресурсом. У протоколі HTTP реалізовано основні методи, що відповідають CRUD-операціям:

- метод GET застосовується для отримання даних без внесення змін на сервері;
- POST використовується для надсилання нових даних, наприклад, під час реєстрації користувача;
- PUT та PATCH призначені для оновлення вже наявних даних; нарешті, DELETE забезпечує видалення ресурсу.

2. URL – уніфікований локатор ресурсу, що вказує шлях до конкретного об'єкта на сервері.

3. Заголовки (headers), які містять службову інформацію, зокрема дані про користувацький агент, прийняті типи контенту, мову, а також токени авторизації.

4. Тіло запиту (body), що являє собою корисне навантаження і передається переважно в методах POST та PUT у форматах JSON або Form-Data.

Анатомія HTTP-відповіді

Ключовим елементом HTTP-відповіді, яку повертає сервер, є статусний код (status code), що інформує клієнта про результат виконання запиту.

Коди стану класифікуються за п'яти категоріями.

1. Коди 2xx сигналізують про успішне виконання операції: зокрема, код 200 OK підтверджує коректне виконання запиту, тоді як 201 Created вказує на успішне створення нового ресурсу.

2. Коди 3xx позначають перенаправлення: наприклад, код 301 Moved Permanently свідчить про остаточне переміщення ресурсу на іншу адресу.

3. Коди 4xx інформують про помилки на стороні клієнта: код 400 Bad Request вказує на некоректний синтаксис запиту, 401 Unauthorized сигналізує про необхідність автентифікації, а 404 Not Found повідомляє, що запитуваний ресурс не знайдено.

4. Коди 5xx констатують помилки на стороні сервера, зокрема код 500 Internal Server Error вказує на внутрішню помилку в роботі серверного програмного забезпечення.

HTTPS та криптографічний захист

HTTPS є розширенням протоколу HTTP, що функціонує поверх

криптографічного протоколу TLS (раніше SSL). Дана архітектура забезпечує три ключові аспекти безпеки мережевої взаємодії:

- шифрування даних, яке унеможливлює їх перехоплення зловмисниками (атаки типу «людина посередині»);
- автентифікацію сервера за допомогою цифрових сертифікатів, що гарантує справжність веб-ресурсу;
- цілісність даних, яка запобігає їх модифікації під час передачі.

4. IP-адресація та ієрархія системи доменних імен (DNS)

Для того щоб пакети даних знаходили правильного адресата в глобальній мережі, кожен вузол повинен мати унікальну адресу.

- IPv4: Класичний 32-бітний формат (наприклад, 192.168.0.1), що дозволяє створити близько 4.3 млрд адрес.
- IPv6: Сучасний 128-бітний формат (наприклад, 2001:0db8:85a3:0000:0000:8a2e:0370:7334), який забезпечує практично невичерпну кількість адрес.

DNS (Domain Name System): Оскільки запам'ятовувати IP-адреси незручно, створено ієрархічну розподілену базу даних доменних імен. Процес резолвінгу (перетворення імені на IP) проходить кілька етапів:

1. Перевірка локального кешу браузера та ОС.
2. Запит до DNS-сервера провайдера (Recursive Resolver).
3. Якщо адреса невідома, запит йде до Кореневих серверів (Root Servers).
4. Запит до серверів доменних зон верхнього рівня (TLD), наприклад .com або .ua.
5. Запит до авторитетного (Authoritative) сервера конкретного домену, який і віддає кінцеву IP-адресу.

5. Деталізований життєвий цикл відображення вебсторінки (Critical Rendering Path)

Що технічно відбувається між натисканням клавіші Enter в адресному рядку та появою сторінки:

1. DNS Lookup: Браузер з'ясовує IP-адресу сервера.
2. TCP Handshake (і TLS Handshake для HTTPS): Встановлення надійного з'єднання з сервером за протоколом TCP.
3. Відправка HTTP-запиту та отримання відповіді: Браузер отримує початковий HTML-документ.
4. Побудова DOM (Document Object Model): Движок браузера читає HTML-код зверху вниз і створює деревоподібну структуру об'єктів.
5. Побудова CSSOM (CSS Object Model): Браузер завантажує CSS-файли і

створює дерево стилів.

6. Виконання JavaScript: Якщо браузер зустрічає тег <script>, він призупиняє рендеринг, завантажує і виконує код (який може змінювати DOM або CSSOM).

7. Побудова Render Tree: Об'єднання DOM та CSSOM (приховуються елементи display: none).

8. Layout / Reflow (Компонування): Розрахунок геометричних розмірів та точних координат кожного елемента на екрані.

9. Paint (Відмальовування): Перетворення обчислених блоків у реальні пікселі на моніторі.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Для глибшого засвоєння матеріалу здобувачам рекомендується використати генеративні ШІ (ChatGPT, Claude, Gemini).

- Приклад промпту №1: *«Дій як Senior Web Developer. Поясни мені різницю між HTTP методами PUT та PATCH простими словами з прикладами з реального життя.»*

- Приклад промпту №2: *«Проаналізуй процес 'TCP Handshake'. Які проблеми можуть виникнути на цьому етапі і як вони впливають на швидкість завантаження сайту (latency)?»*

Питання для обговорення та контролю знань:

1. Опишіть роль протоколу TCP у стеку TCP/IP та механізми, за допомогою яких він забезпечує надійність доставки даних.

2. Поясніть, чому архітектуру сучасних корпоративних вебзастосунків часто розбивають на три рівні (3-Tier).

3. Назвіть основні категорії HTTP Status Codes та розкрийте значення кодів 404 і 500 для розробника.

4. Поясніть, чому розміщення тегу <script> у розділі <head> (без атрибутів defer/async) може уповільнити відображення сторінки.

ТЕМА 1.2. HTML5: ОСНОВА СТРУКТУРИ ВЕБ-СТОРІНКИ

Мета: засвоїти принципи сучасного структурування вебдокументів; вивчити концепцію семантичної розмітки та її вплив на пошукову оптимізацію (SEO) і доступність (Accessibility); опанувати інструменти створення інтерактивних вебформ із вбудованою клієнтською валідацією та роботу з мультимедійними елементами.

План лекції:

1. Основи мови розмітки HTML5: парадигма семантичної розмітки.
2. Управління метаданими: кодування, viewport та Open Graph.
3. Робота з користувацькими даними: вебформи та вбудована валідація.
4. Мультимедійні елементи та вбудовування зовнішнього контенту.
5. Вебдоступність (Accessibility) та використання ARIA-атрибутів.

1. Основи мови розмітки HTML5: парадигма семантичної розмітки

HTML (HyperText Markup Language) – це стандартизована мова розмітки, яка будує DOM-дерево (Document Object Model) вебсторінки. Головним нововведенням стандарту HTML5 стала відмова від надмірного використання несемантичних тегів на користь семантичної розмітки.

Семантичний тег чітко пояснює свій зміст як розробнику, так і пошуковим роботам (Googlebot) та допоміжним технологіям (Screen Readers), що критично покращує SEO та доступність.

Основні семантичні теги HTML5:

- <header> – вступна частина сторінки або окремої секції (зазвичай містить логотип та головний заголовок).
- <nav> – блок основної навігації сайту (меню).
- <main> – унікальний основний контент сторінки. На одній сторінці може бути лише один видимий тег <main>.
- <article> – самостійний, незалежний блок контенту, який має сенс у відриві від решти сторінки (наприклад, стаття в блозі, картка товару).
- <section> – логічне або тематичне групування контенту, зазвичай із власним заголовком.
- <aside> – контент, який побічно пов'язаний з основним (бокові панелі, рекламні блоки, хмари тегів).
- <footer> – підвал сторінки або секції (копірайт, юридична інформація).

2. Управління метаданими: кодування, viewport та Open Graph

Тег <head> містить метадані, які не відображаються у вікні браузера, але є необхідними для коректного рендерингу сторінки та взаємодії із соціальними мережами.

- Кодування: <meta charset=«UTF-8»> – встановлює універсальне кодування символів, що підтримує всі мови світу.
- Адаптивність (Viewport): <meta name=«viewport» content=«width=device-width, initial-scale=1.0»> – ключовий тег для мобільних пристроїв. Він вказує браузеру, що ширина області перегляду має дорівнювати фізичній ширині екрана, відключаючи примусове зменшення десктопної версії

сайту.

- SEO-теги: <meta name=«description» content=«...»> – короткий опис сторінки, який пошукові системи часто виводять у сніпеті під посиланням.

- Open Graph (OG): Спеціальні метатеги (наприклад, og:title, og:image, og:description), що визначають, як виглядатиме посилання на ваш сайт при поширенні у Facebook, Telegram, Viber чи інших соціальних мережах.

3. Робота з користувацькими даними: вебформи та вбудована валідація

Тег <form> забезпечує збір та передачу даних від клієнта до сервера. Стандарт HTML5 значно розширив типи полів вводу <input>, переклавши частину відповідальності за перевірку даних (валідацію) з JavaScript на рушій браузера.

Сучасні типи полів (type):

- email, url, tel – не лише перевіряють базовий синтаксис (наявність @ або http://), але й викликають оптимізовану віртуальну клавіатуру на мобільних пристроях.

- date, time, color, range – нативно викликають відповідні віджети (календар, палітру) без необхідності підключення сторонніх JS-бібліотек.

Атрибути клієнтської валідації та UX:

- required – робить поле обов'язковим для заповнення. Форма не відправиться, якщо поле порожнє.

- pattern – дозволяє задати регулярний вираз (Regex) для складної перевірки (наприклад, pattern=«[0-9]{3}-[0-9]{3}-[0-9]{4}» для формату телефону).

- autocomplete – підказує браузеру, які дані з автозаповнення сюди підставити (наприклад, autocomplete=«off» для вимкнення підказок).

4. Мультимедійні елементи та вбудовування зовнішнього контенту

Сучасний HTML дозволяє працювати з медіафайлами нативно, без використання сторонніх плагінів (як це було раніше з Flash).

- Аудіо та Відео: Теги <audio> та <video>. Ключові атрибути: controls (стандартна панель керування), autoplay (автоматичний запуск – у сучасних браузерах працює лише разом із атрибутом muted), loop (зациклення). Для кросбраузерності всередині використовується тег <source> із різними форматами кодеків (наприклад, mp4, webm).

- Адаптивні зображення (<picture>): Елемент дозволяє браузеру обирати найоптимальніший графічний файл. Всередині <picture> використовуються теги <source>, які вказують браузеру завантажувати, наприклад, легкий формат

WebP, а якщо браузер його не підтримує – резервний JPEG через тег <img>.

- Зовнішній контент (<iframe>): Тег для вбудовування іншої вебсторінки всередину поточної (найчастіше використовується для інтеграції Google Maps або YouTube-плеєрів).

5. Вебдоступність (Accessibility) та використання ARIA-атрибутів

Доступність (Accessibility, скорочено a11y) – це практика створення вебінтерфейсів, якими можуть користуватися абсолютно всі люди, включно з користувачами з порушеннями зору, моторики або когнітивними розладами.

Коли можливостей стандартних семантичних тегів недостатньо для опису складної взаємодії (наприклад, кастомні модальні вікна чи випадаючі списки), використовуються атрибути WAI-ARIA (Accessible Rich Internet Applications).

- role=«...» – чітко визначає семантичну роль елемента (наприклад, role=«button», role=«dialog», role=«navigation»).
- aria-label=«...» – прихований текстовий опис елемента для скрінрідерів (наприклад, для кнопок-іконок закриття «X», які не містять видимого тексту).
- aria-hidden=«true» – ховає декоративний елемент (наприклад, іконку FontAwesome) від скрінрідера, щоб не створювати звуковий «шум».

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Для ефективного застосування сучасних інструментів розробки здобувачам рекомендується використовувати ШІ-асистенти.

- Приклад промпту №1 (Валідація): *«Згенеруй регулярний вираз (Regex) для атрибута pattern в HTML5, який дозволятиме введення лише українських номерів телефонів. Детально поясни синтаксис цього регулярного виразу.»*
- Приклад промпту №2 (Доступність): *«Проаналізуй цей HTML-код модального вікна (надай ШІ свій код). Яких ARIA-атрибутів йому не вистачає для повної відповідності стандартам доступності? Запропонуй виправлений варіант.»*

Питання для обговорення та контролю знань:

1. Поясніть принципову різницю між тегами <section>, <article> та і визначте випадки, коли використання залишається виправданим
2. Визначте метатеги, необхідні для коректного відображення посилання на вебсторінку в месенджерах (Telegram/Viber)
3. Обґрунтуйте, чому наявність атрибута required у HTML-формі на стороні клієнта не скасовує необхідності валідації даних на стороні сервера (наприклад, за допомогою PHP)
4. Укажіть атрибути, які необхідно застосувати до тегу <video>, щоб відео

автоматично відтворювалося як фонове зображення (без звуку та панелі керування)

5. Поясніть, що таке ARIA-ролі, та опишіть випадки, коли їх використання є критично необхідним для забезпечення доступності (a11y) вебсайту.

ТЕМА 1.3. CSS3: ПРИНЦИПИ ОФОРМЛЕННЯ ТА ВЕРСТКИ

Мета: засвоїти фундаментальні механізми роботи каскадних таблиць стилів (CSS); зрозуміти принципи каскадності, специфічності та успадкування; опанувати роботу з базовими та комбінованими селекторами, блоковою моделлю (Box Model) документа, а також класичними методами позиціонування елементів на вебсторінці.

План лекції:

1. Роль CSS у веброзробці: синтаксис та способи підключення стилів.
2. Механізми CSS: каскадність, специфічність та успадкування.
3. Селектори в CSS3: від базових до псевдокласів та псевдоелементів.
4. Блокова модель документа (CSS Box Model) та розрахунок розмірів.
5. Управління потоком документа: властивість display та позиціонування.

1. Роль CSS у веброзробці: синтаксис та способи підключення стилів

CSS (Cascading Style Sheets, Каскадні таблиці стилів) – це спеціальна мова, яка використовується для опису зовнішнього вигляду (представлення) документа, написаного мовою розмітки (HTML). CSS дозволяє відокремити зміст сторінки (HTML) від її візуального оформлення (кольорів, шрифтів, відступів, анімацій), що є фундаментальним правилом сучасної веброзробки.

Базовий синтаксис CSS-правила: Кожне CSS-правило складається з селектора (вказує, до якого елемента застосувати стиль) та блоку оголошень у фігурних дужках. Блок оголошень містить пари *властивість: значення*;.

CSS

```
h1 {
   color: blue;
   font-size: 24px;
}
```

Способи підключення CSS до HTML:

1. Вбудований (Inline): Стилі пишуться безпосередньо в тезі через атрибут style (напр., <p style=«color: red;»>). *Вважається антипатерном*, оскільки порушує принцип розділення коду та ускладнює підтримку проєкту.

2. Внутрішній (Internal): Стилі розміщуються у тезі <style> всередині блоку <head>. Доцільно використовувати лише для дуже малих сторінок або для критичного CSS (Critical Rendering Path).

3. Зовнішній (External): Стилі зберігаються в окремому файлі з розширенням .css та підключаються в HTML за допомогою тегу <link rel=«stylesheet» href=«styles.css»>. *Це головний та правильний стандарт.*

2. Механізми CSS: каскадність, специфічність та успадкування

Розуміння того, як браузер вирішує, який саме стиль застосувати до елемента при виникненні конфліктів, є критичним для фронтенд-розробника.

1) Каскадність (Cascade): Це алгоритм, за яким браузер визначає пріоритет стилів. За загальним правилом, якщо правила мають однакову вагу, перемагає те, яке оголошено *нижче* у коді (останнім).

2) Специфічність (Вага селектора): Це математична система балів, що визначає пріоритет.

a) Теги (напр., div, p) та псевдоелементи мають найменшу вагу.

b) Класи (напр., .button), атрибути та псевдокласи мають середню вагу.

c) Ідентифікатори (ID, напр., #header) мають високу вагу.

d) Inline-стилі (атрибут style) перебивають усе вищезазначене.

e) Ключове слово !important має абсолютний пріоритет, але його використання є поганою практикою і допускається лише для тимчасового «латання» багів.

3) Успадкування (Inheritance): Деякі властивості (переважно ті, що стосуються типографіки – color, font-family, font-size) автоматично передаються від батьківського контейнера до всіх його дочірніх елементів. Властивості, пов'язані з розмірами та відступами (margin, padding, border), *не успадковуються*.

3. Селектори в CSS3: від базових до псевдокласів та псевдоелементів

Селектори дозволяють точно «націлюватися» на необхідні вузли DOM-дерева.

Базові та комбіновані селектори:

- Універсальний: * (обирає абсолютно всі елементи сторінки).
- За тегом: p (всі абзаци).
- За класом: .container (всі елементи з класом «container»). Класи можна використовувати багаторазово.
- За ID: #main-menu (ідентифікатор має бути унікальним на сторінці).
- Вкладені (Нащадки): .card img (всі зображення всередині блоку .card).

Псевдокласи (визначають стан елемента):

- :hover – стан при наведенні курсора миші.
- :focus – коли елемент (наприклад, текстове поле форми) у фокусі (активний).
- :nth-child(n) – обирає елемент за його порядковим номером у батьківському контейнері (наприклад, для створення «зебри» у таблицях).

Псевдоелементи (дозволяють стилізувати частину елемента або додати віртуальний контент):

- ::before та ::after – створюють віртуальні вузли перед або після вмісту основного елемента. Часто використовуються для декоративних цілей (іконки, графічні елементи) без засмічення HTML-коду зайвими тегами або . Вимагають обов'язкової наявності властивості content.

4. Блокова модель документа (CSS Box Model) та розрахунок розмірів

Блокова модель (Box Model) – це фундаментальна концепція рендерингу. Браузер розглядає кожен HTML-елемент як прямокутний контейнер, що складається з чотирьох шарів (зсередини назовні):

1. Content (Контент): Сама область із текстом або зображенням. Визначається властивостями width та height.
2. Padding (Внутрішній відступ): Прозорий простір між контентом і межею (border). Колір фону елемента поширюється і на padding.
3. Border (Межа): Лінія, що оточує padding та контент (border: 1px solid black;).
4. Margin (Зовнішній відступ): Прозорий простір *за межами* елемента, що відокремлює його від сусідніх блоків.

Критично важливе налаштування box-sizing: За замовчуванням у стандартному CSS ширина елемента розраховується як width + padding + border. Якщо задати блоку width: 100% і додати padding: 20px, він стане ширшим за екран і з'явиться горизонтальна прокрутка. Щоб уникнути цього, у сучасній верстці завжди використовується універсальне скидання:

CSS

```
* {
   box-sizing: border-box;
}
```

Це правило змушує браузер включати padding і border у задану ширину (width), залишаючи реальні зовнішні габарити блоку стабільними.

5. Управління потоком документа: властивість display та позиціонування

За замовчуванням елементи розміщуються на сторінці згідно зі

стандартним потоком (Normal Flow), зверху вниз.

Властивість display (Визначає тип поведінки елемента):

- block: Елемент (напр., , <h1>) займає всю доступну ширину (100%), починається з нового рядка. Йому можна задати width і height.
- inline: Елемент (напр., , <a>) займає рівно стільки місця, скільки вимагає контент. Йому *неможливо* задати width, height або вертикальні відступи (margin-top).
- inline-block: Гібрид. Не переносить на новий рядок, але дозволяє задавати ширину, висоту та відступи.
- none: Повністю вилучає елемент із потоку та ховає його (на відміну від visibility: hidden, де елемент ховається, але залишає після себе порожній простір).

Позиціонування (Властивість position): Дозволяє вирвати елемент із нормального потоку та розмістити його за точними координатами (top, bottom, left, right):

- static: За замовчуванням (координати ігноруються).
- relative: Елемент зсувається відносно свого *початкового положення*. Його оригінальне місце у потоці зберігається порожнім.
- absolute: Елемент повністю вивертається з потоку і позиціонується відносно *найближчого предка з позиціонуванням, відмінним від static*.
- fixed: Елемент «приклеюється» до вікна браузера і залишається на місці під час прокручування сторінки (наприклад, липка шапка меню).

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Згідно з вимогами дисципліни, здобувачі можуть використовувати інструменти ШІ (ChatGPT, Claude) для глибшого розуміння специфіки CSS:

- Приклад промпту №1 (Розрахунок ваги): *«Дій як Senior Frontend Developer. У мене є HTML-елемент <a class=«link» id=«nav-item» style=«color: black;»>. І є такий CSS: a.link { color: red; }, #nav-item { color: blue; }. Якого кольору буде посилання і чому? Поясни, як працює математика специфічності в CSS.»*
- Приклад промпту №2 (Налагодження/Debugging): *«Я застосував position: absolute до модального вікна, щоб вирівняти його по центру батьківського блоку .wrapper, але воно зсунулося відносно всього екрана. У чому моя логічна помилка і як її виправити?»*

Питання для обговорення та контролю знань:

1. Поясніть концепцію каскадності в css. Опишіть, який колір буде застосовано до елемента h1, якщо у зовнішньому файлі стилів спочатку задано

правило h1 { color: red; }, а нижче – h1 { color: green; }.

2. Визначте принципову різницю між псевдокласами (наприклад, :hover) та псевдоелементами (наприклад, ::before). Зазначте обов'язковий атрибут, якого вимагають псевдоелементи для коректного відображення.

3. Обґрунтуйте, чому правило * { box-sizing: border-box; } стало індустріальним стандартом у сучасній верстці. Поясніть, як воно вирішує проблему класичної моделі box model.

4. Порівняйте вплив властивостей display: none та visibility: hidden на dom та сусідні елементи. Вкажіть, як кожна з них змінює відображення елемента та його місце в потоці документа.

5. Опишіть сценарії використання position: relative та position: absolute. Поясніть, як ці значення властивості position взаємодіють між собою при побудові макету.

ТЕМА 1.4. АДАПТИВНИЙ ТА МОБІЛЬНИЙ ДИЗАЙН, РОБОТА З МЕДІА-КОНТЕНТОМ

Мета: засвоїти сучасні парадигми створення кросплатформних інтерфейсів; зрозуміти концептуальну різницю між чуйним (RWD) та адаптивним (AWD) дизайном; опанувати методологію Mobile-First; навчитися використовувати CSS-медіазапити, відносні одиниці виміру та гнучкий медіа-контент для оптимізації відображення вебсторінок на пристроях із різною роздільною здатністю екрана.

План лекції:

1. Еволюція підходів: Фіксований, Адаптивний (AWD) та Чуйний (RWD) вебдизайн.
2. Парадигма Mobile-First та поступове покращення (Progressive Enhancement).
3. Медіазапити (Media Queries) в CSS3: синтаксис та точки зламу (Breakpoints).
4. Гнучка сітка (Fluid Grid) та сучасні відносні одиниці виміру.
5. Робота з адаптивним медіа-контентом (зображення, відео) та атрибут srcset.

1. Еволюція підходів: Фіксований, Адаптивний (AWD) та Чуйний (RWD) вебдизайн

З появою смартфонів виникла критична необхідність оптимізувати вебсторінки під екрани різних розмірів. Історично сформувалося кілька

підходів:

-Фіксований дизайн (Fixed Layout): Сторінка має жорстко задану ширину в пікселях (наприклад, 960px). На мобільних пристроях такий сайт доводиться масштабувати пальцями (pinch-to-zoom), що створює жахливий користувацький досвід (UX).

-Адаптивний дизайн (AWD - Adaptive Web Design): Створюється кілька окремих, жорстко фіксованих макетів для конкретних пристроїв (наприклад, 320px для смартфонів, 768px для планшетів, 1024px для ПК). Сервер або клієнтський скрипт визначає тип пристрою і «підкидає» відповідний макет. Між цими контрольними точками дизайн не змінюється.

-Чуйний дизайн (RWD - Responsive Web Design): Сучасний індустріальний стандарт, запропонований Ітаном Маркоттом у 2010 році. Макет плавно і динамічно трансформується під *будь-яку* ширину екрана за рахунок використання відносних одиниць (відсотків), гнучких зображень та медіазапитів. Сторінка виглядає коректно навіть на проміжних або нестандартних роздільних здатностях.

2. Парадигма Mobile-First та поступове покращення

Підхід Mobile-First (Мобільні пристрої насамперед) є фундаментальною зміною у мисленні веброзробника. Замість того, щоб проєктувати складний десктопний сайт, а потім намагатися «сховати» або «зжати» його елементи для смартфонів (Graceful Degradation), розробка починається з мобільної версії.

Принципи Mobile-First:

1. Базовий CSS: Усі стилі, що пишуться за замовчуванням (без медіазапитів), призначені для найменших екранів. Вони формують лінійний, спрощений макет (в один рядок).

2. Progressive Enhancement (Поступове покращення): У міру того, як ширина екрана збільшується (планшет, ноутбук), розробник за допомогою медіазапитів *додає* нові стилі, ускладнюючи сітку (перехід від одного стовпця до трьох, додавання складних ховер-ефектів).

3. Продуктивність: Мобільні пристрої часто мають слабший процесор і повільніший інтернет. Завантажуючи спочатку лише мобільний CSS, ми зменшуємо час початкового рендерингу.

3. Медіазапити (Media Queries) в CSS3: синтаксис та точки зламу

Media Queries – це модуль CSS3, який дозволяє застосовувати стилі лише тоді, коли браузер або пристрій відповідає певним умовам (наприклад, певній ширині вікна, орієнтації екрана тощо).

Базовий синтаксис для Mobile-First (використання min-width):

```
CSS
/* Базові стилі для смартфонів (і всіх пристроїв за замовчуванням) */
.container {
   width: 100%;
}

/* Точка зламу для планшетів (від 768px і ширше) */
@media screen and (min-width: 768px) {
   .container {
      width: 750px;
   }
}

/* Точка зламу для десктопів (від 1024px і ширше) */
@media screen and (min-width: 1024px) {
   .container {
      width: 960px;
   }
}
```

Точки зламу (Breakpoints): Це конкретні значення ширини екрана, на яких макет зазнає значних структурних змін. Не варто прив'язувати брейкпоінти до конкретних моделей телефонів (наприклад, iPhone 14), оскільки пристроїв тисячі. Точки зламу мають залежати виключно від того, коли *сам контент* починає виглядати погано і потребує перелаштування.

4. Гнучка сітка (Fluid Grid) та сучасні відносні одиниці виміру

Для забезпечення плавності RWD необхідно відмовитися від жорстких одиниць виміру (пікселів – px) на користь відносних під час задання ширини блоків.

Основні відносні одиниці:

- % (Відсотки): Розраховуються відносно розміру *батьківського* елемента. Наприклад, якщо батьківський div має ширину 1000px, то дочірній елемент із width: 50% займе 500px.
- vw (Viewport Width) та vh (Viewport Height): Відносні до розміру самого вікна браузера. 1vw дорівнює 1% від ширини вікна. Ідеально підходять для створення секцій, що займають весь екран (height: 100vh;).
- em: Розраховується відносно розміру шрифту (font-size) *поточного* елемента або його безпосереднього предка. Часто призводить до каскадних проблем масштабування.

-rem (Root em): Розраховується відносно розміру шрифту *кореневого* елемента (тегу <html>, зазвичай це 16px). Це сучасний стандарт для задання розмірів шрифтів та відступів в адаптивному дизайні, оскільки він дозволяє користувачам з вадами зору легко масштабувати весь сайт через налаштування браузера.

5. Робота з адаптивним медіа-контентом (зображення, відео) та атрибут srcset

Велике зображення фіксованого розміру (наприклад, 800px за шириною) «розірве» макет на екрані смартфона (шириною 360px), змусивши з'явитися горизонтальну прокрутку.

1. CSS-правило гнучких зображень (Fluid Media): Фундаментальне правило RWD, яке має застосовуватися до всіх медіафайлів:

CSS

```
img, video, iframe {
   max-width: 100%;
   height: auto;
}
```

max-width: 100% гарантує, що зображення ніколи не стане ширшим за свій батьківський контейнер, але при цьому воно не буде розтягуватися більше за свій оригінальний розмір (на відміну від width: 100%).

2. Оптимізація продуктивності: <picture> та атрибут srcset: Хоча CSS-правило робить зображення візуально гнучким, воно не вирішує проблему ваги файлу: мобільний телефон все одно завантажуватиме гігантську картинку, призначену для 4K-монітора. Для цього HTML5 пропонує атрибут srcset:

HTML

```
<img src=«small.jpg»
    srcset=«small.jpg 480w, medium.jpg 800w, large.jpg 1200w»
    sizes=«(max-width: 600px) 480px, 800px»
    alt=«Опис»>
```

Браузер самостійно аналізує ширину свого вікна та щільність пікселів екрана (наприклад, екрани Retina), і завантажує лише той файл, який є найбільш оптимальним у даний момент, суттєво заощаджуючи мобільний трафік.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

-Приклад промпту №1 (Архітектура Mobile-First): *«Дій як Senior Frontend Розробник. Напиши базовий CSS-каркас (boilerplate) з медіазапитами за методологією Mobile-First для типового інтернет-магазину. Визнач 3 основні*

точки зламу (breakpoints) та поясни, чому ти обрав саме ці значення в пікселях.»

-Приклад промпту №2 (Типографіка): *«Поясни мені концепцію 'Fluid Typography' (Адаптивна типографіка) в CSS. Напиши приклад коду, використовуючи функцію clamp() для розміру заголовка H1, щоб він плавно масштабувався від 24px на мобільному до 48px на десктопі.»*

Питання для обговорення та контролю знань:

1. Розкрийте фундаментальну відмінність між чуйним (Responsive) та адаптивним (Adaptive) вебдизайном з точки зору технічної реалізації.
2. Поясніть логіку застосування методології Mobile-First. Обґрунтуйте, чому розробку CSS-коду починають з найменших екранів, використовуючи медіазапити min-width.
3. Обґрунтуйте, чому одиниця виміру rem є кращою за px при заданні розмірів шрифтів (font-size) та відступів (margin/padding) у сучасному вебі.
4. Опишіть, що станеться, якщо зображенню застосувати width: 100% замість max-width: 100% в адаптивному контейнері.
5. Визначте, яку проблему продуктивності вирішує використання атрибута srcset у тезі <img> для мобільних користувачів.

ЗМІСТОВИЙ МОДУЛЬ 2. JAVASCRIPT
ТЕМА 2.1. JAVASCRIPT: СИНТАКСИС, ТИПИ ДАНИХ ТА ОСНОВИ РОБОТИ

Мета: засвоїти фундаментальні основи клієнтського програмування мовою JavaScript; вивчити правила оголошення змінних, специфіку динамічної типізації, базові типи даних, механізми перетворення типів, а також алгоритмічні конструкції керування потоком виконання (умови та цикли).

План лекції:

1. Роль JavaScript у веброзробці та способи інтеграції скриптів.
2. Змінні та константи: еволюція від var до let і const.
3. Система типів даних: примітиви та об'єктні типи (Reference types).
4. Базові оператори, приведення типів та строга рівність (===).
5. Основи алгоритмізації: умовні конструкції та цикли.

1. Роль JavaScript у веброзробці та способи інтеграції скриптів

JavaScript (JS) – це високорівнева, інтерпретована мова програмування, яка відповідає за інтерактивність та поведінку вебсторінок на стороні клієнта (у браузері). Стандартизацією мови займається специфікація ECMAScript (ES). Код JS виконується спеціальними рушіями браузерів (наприклад, V8 у Google Chrome або SpiderMonkey у Firefox).

Способи підключення JS до HTML-документа: Як і у випадку з CSS, найкращою практикою є винесення коду в зовнішній файл .js:

HTML

<script src=«main.js»></script>

<script src=«main.js» defer></script>

Атрибут defer вказує браузеру завантажувати скрипт у фоновому режимі (асинхронно), не блокуючи побудову DOM-дерева, але виконати його лише тоді, коли HTML буде повністю розпарсено.

2. Змінні та константи: еволюція від var до let і const

До виходу стандарту ES6 (2015 рік) єдиним способом оголосити змінну було ключове слово var. Сьогодні це вважається застарілим підходом через проблеми з «підняттям» (hoisting) та областю видимості.

Сучасні стандарти оголошення:

- let: Використовується для змінних, значення яких може змінюватися в процесі виконання програми. Має *блокову область видимості* (змінна існує лише всередині фігурних дужок {}, де її оголошено).
- const: Використовується для констант – значень, які не повинні

переприсвоюватися. Також має блокову область видимості. *Важливо:* const захищає від переприсвоєння самого ідентифікатора, але якщо константа зберігає масив або об'єкт, їхній внутрішній вміст можна змінювати (мутувати).

JavaScript

let userAge = 25;

userAge = 26; // Дозволено

const birthYear = 1998;

birthYear = 1999; // Помилка: TypeError

3. Система типів даних: примітиви та об'єктні типи

JavaScript є мовою з динамічною типізацією (Dynamic Typing). Це означає, що тип даних прив'язаний до самого значення, а не до змінної. Одна змінна може спочатку зберігати число, а потім – рядок тексту.

За стандартом ECMAScript існує 8 базових типів даних, які поділяються на дві категорії:

Примітивні типи (передаються за значенням):

1. Number: Цілі числа та числа з рухомою комою (наприклад, 42, 3.14).

2. String: Текстові рядки, взяті в одинарні, подвійні або зворотні (шаблонні) лапки.

3. Boolean: Логічний тип (true або false).

4. Undefined: Значення за замовчуванням для змінних, які було оголошено, але яким не було присвоєно жодного значення.

5. Null: Спеціальне значення, яке явно вказує на «ніщо» або порожнечу.

6. Symbol: Унікальні та незмінні ідентифікатори об'єктів.

7. BigInt: Для роботи з дуже великими числами.

Об'єктні типи (передаються за посиланням): 8. Object: Складні структури даних (масиви, функції, дати тощо). На відміну від примітивів, об'єкти зберігаються в купі (Heap), а змінна містить лише посилання на цю ділянку пам'яті.

4. Базові оператори, приведення типів та строга рівність

Слабкою (але потужною) стороною JS є неявне приведення типів (Type Coercion). Браузер часто намагається самостійно вгадати, що мав на увазі розробник, якщо той змішує різні типи даних. Наприклад, вираз '5' + 1 поверне рядок '51', оскільки оператор + ініціює конкатенацію (склеювання). Але вираз '5' - 1 поверне число 4, бо мінус працює лише з числами.

Оператори порівняння (Критично важливо):

- Нестрога рівність (==): Порівнює значення *після* приведення їх до одного типу. Вираз 5 == '5' поверне true. Використання цього оператора часто

призводить до логічних помилок.

- Строга рівність (===): Порівнює і значення, і тип даних. Вираз 5 === '5' поверне false. У сучасній розробці слід завжди використовувати лише строгу рівність.

5. Основи алгоритмізації: умовні конструкції та цикли

Для керування логікою (Control Flow) JS використовує класичні C-подібні конструкції.

Умовні конструкції (if/else та тернарний оператор):

JavaScript

```
if (userAge >= 18) {
   console.log(«Доступ дозволено»);
} else if (userAge === 17) {
   console.log(«Зачекайте ще рік»);
} else {
   console.log(«Доступ заборонено»);
}
// Тернарний оператор (спрощений if/else для простих умов)
let status = (userAge >= 18) ? «Дорослий» : «Неповнолітній»;
```

У контексті логічних перевірок деякі значення автоматично приводяться до false (так звані *falsy*-значення): 0, «» (порожній рядок), null, undefined, NaN. Усі інші значення є *truthy* (навіть порожні масиви чи об'єкти).

Цикли (for, while):

- Цикл while: Виконується, поки умова є істинною. Ідеальний для ситуацій, коли точна кількість ітерацій заздалегідь невідома. Головний ризик – забути оновити умову та створити нескінченний цикл, який «підвісить» вкладку браузера.

- Цикл for: Використовується, коли кількість ітерацій відома. Має вбудований лічильник.

JavaScript

```
for (let i = 0; i < 5; i++) {
   if (i === 3) continue; // Пропускає 3-тю ітерацію
   if (i === 4) break;    // Повністю зупиняє цикл
   console.log(i);
}
```

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Згідно з вимогами дисципліни, ШІ-асистенти можуть значно прискорити розуміння базового синтаксису:

- Приклад промпту №1 (Налагодження коду): *«Дій як ментор з JavaScript. Чому виклик console.log(0 == false) повертає true, а console.log(0 === false) повертає false? Поясни концепцію Type Coercion на цьому прикладі простими словами.»*

- Приклад промпту №2 (Алгоритми): *«Напиши класичний алгоритм FizzBuzz мовою JavaScript, використовуючи цикл for та строгі перевірки кратності (оператор %). Поясни кожен крок логіки.»*

Питання для обговорення та контролю знань:

1. Опишіть роботу умовної конструкції if-else у JavaScript та поясніть різницю між конструкціями else if та else (згідно з питаннями до екзамену).
2. Визначте, які типи даних та значення вважаються істинними (truthy) та хибними (falsy) в умовних виразах JavaScript.
3. Поясніть, чому використання ключового слова var сьогодні є небажаним, і чим воно відрізняється від let за своєю областю видимості.
4. Охарактеризуйте відмінності між циклами for і while у клієнтських скриптах.
5. Розкрийте значення використання ключових слів break та continue під час ітерації циклу та опишіть, як уникнути нескінченного циклу while.

ТЕМА 2.2. ФУНКЦІЇ, ЗАМИКАННЯ ТА ФУНКЦІОНАЛЬНЕ ПРОГРАМУВАННЯ

Мета: засвоїти різні способи оголошення та виклику функцій у JavaScript; зрозуміти концепцію лексичного оточення та замикань (closures); опанувати роботу з функціями вищого порядку для обробки масивів; ознайомитися з базовими концепціями функціонального програмування (зокрема, чистими функціями).

План лекції:

1. Способи створення функцій: Declaration, Expression та Arrow Functions.
2. Робота з аргументами: параметри за замовчуванням та rest-параметри.
3. Область видимості, лексичне оточення та Замикання (Closures).
4. Функціональне програмування: Чисті функції (Pure Functions).
5. Функції вищого порядку та методи обробки масивів (map, filter, reduce).

1. Способи створення функцій: Declaration, Expression та Arrow Functions

Функції в JavaScript є об'єктами першого класу (First-class citizens). Це

означає, що їх можна передавати як аргументи, повертати з інших функцій та присвоювати змінним.

Існує три основні способи створення функцій:

- Function Declaration (Оголошення функції): Класичний спосіб. Головна особливість – такі функції піднімаються (hoisted) на самий початок області видимості під час виконання. Це дозволяє викликати функцію в коді ще до того, як її було оголошено.

JavaScript

```
function calculateTotal(price, quantity) {
   return price * quantity;
}
```

- Function Expression (Функціональний вираз): Функція створюється в контексті якогось виразу (наприклад, присвоєння змінній). Вона не піднімається (no hoisting), тому її можна викликати лише після рядка з оголошенням.

JavaScript

```
const calculateTotal = function(price, quantity) {
   return price * quantity;
};
```

- Arrow Functions (Стрілкові функції): Сучасний синтаксис, доданий у стандарті ES6 (2015). Він більш лаконічний. Стрілкові функції не мають власного контексту this (що робить їх ідеальними для колбеків) і не можуть бути використані як конструктори.

JavaScript

```
const calculateTotal = (price, quantity) => price * quantity;
```

2. Робота з аргументами: параметри за замовчуванням та rest-параметри

Сучасний JavaScript пропонує зручні інструменти для роботи з аргументами, які роблять код стійкішим до помилок.

- Параметри за замовчуванням (Default Parameters): Дозволяють задати значення, яке буде використано, якщо під час виклику функції аргумент не передали (або передали undefined).

JavaScript

```
function greet(name = «Гість») {
   console.log(`Привіт, ${name}!`);
}
greet(); // Виведе: Привіт, Гість!
```

- Rest-параметри (...): Дозволяють зібрати будь-яку кількість «зайвих»

аргументів, переданих у функцію, в єдиний масив. Це сучасна альтернатива застарілому об'єкту arguments.

JavaScript

```
function sumAll(...numbers) {
   // numbers - це справжній масив
   return numbers.reduce((acc, curr) => acc + curr, 0);
}
console.log(sumAll(1, 2, 3, 4)); // Виведе: 10
```

3. Область видимості, лексичне оточення та Замикання (Closures)

Замикання (Closure) – це одна з найважливіших і найскладніших концепцій у JavaScript.

Кожна функція під час свого створення отримує приховане посилання на своє лексичне оточення (Lexical Environment) – місце в коді, де вона була фізично написана. *Замикання* – це здатність функції «запам'ятовувати» змінні зі свого зовнішнього лексичного оточення і мати до них доступ навіть тоді, коли зовнішня функція вже завершила своє виконання.

Практичне застосування (Фабрика функцій та інкапсуляція):

JavaScript

```
function createCounter() {
   let count = 0; // «Прихована» змінна
   return function() {
      count++;
      return count;
   };
}

const counter1 = createCounter();
console.log(counter1()); // 1
console.log(counter1()); // 2
// Достукатися до змінної count напряму ззовні - неможливо! Це і є інкапсуляція.
```

4. Функціональне програмування: Чисті функції (Pure Functions)

JavaScript є мультипарадигмальною мовою. Останніми роками великої популярності набуло функціональне програмування. Його базовий будівельний блок – чиста функція.

Функція вважається чистою, якщо вона відповідає двом суворим правилам:

1. Детермінованість: За одних і тих самих вхідних аргументів вона завжди повертає абсолютно однаковий результат.

2. Відсутність побічних ефектів (Side Effects): Функція не змінює жодних зовнішніх змінних, не модифікує DOM-дерево, не робить запитів до бази даних і не мутує вхідні параметри. Вона лише створює і повертає нові дані.

Приклад брудної функції (мутує зовнішній масив):

JavaScript

```
let cart = [];
function addToCart(item) {
   cart.push(item); // Побічний ефект!
}
```

Приклад чистої функції:

JavaScript

```
function addToCart(cart, item) {
   return [...cart, item]; // Повертає новий масив, не чіпаючи старий
}
```

5. Функції вищого порядку та методи обробки масивів

Функція вищого порядку (Higher-Order Function) – це функція, яка приймає іншу функцію як аргумент (колбек) АБО повертає функцію як результат (як у прикладі із замиканням).

Найбільш практичне їх застосування – це вбудовані методи масивів:

-forEach(callback): Виконує передану функцію для кожного елемента масиву. Нічого не повертає (undefined). Використовується просто для ітерації.

-map(callback): Трансформує масив. Створює і повертає *новий* масив такої ж довжини, де кожен елемент є результатом виконання колбеку.

-filter(callback): Фільтрує масив. Повертає *новий* масив, до якого потраплять лише ті елементи, для яких колбек повернув true.

-reduce(callback, initialValue): Згортає (редукує) масив до одного єдиного значення (наприклад, сума всіх чисел, об'єднання в об'єкт тощо).

-find(callback): Повертає перший елемент масиву, який задовольняє умову, і зупиняє пошук.

-some(callback) / every(callback): Повертають true або false. some перевіряє, чи хоча б один елемент відповідає умові. every перевіряє, чи абсолютно всі елементи відповідають умові.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Згідно з вимогами курсу, використання ШІ допомагає краще зрозуміти складні алгоритмічні концепції:

-Приклад промпту №1 (Замикання): *«Поясни концепцію замикань (closures) у JavaScript на прикладі з реального життя (наприклад, рюкзак з речами). Наведи приклад коду, де замикання використовується для створення приватних змінних.»*

-Приклад промпту №2 (Методи масивів): *«У мене є масив об'єктів users (id, ім'я, вік). Напиши чисту функцію з використанням комбінації методів filter та map, яка поверне масив імен користувачів, старших за 18 років. Поясни, чому такий ланцюжок методів є кращим за використання циклу for.»*

Питання для обговорення та контролю знань:

1. Визначте принципову відмінність між Function Declaration та Function Expression щодо етапу їх ініціалізації в коді (hoisting).
2. Поясніть, чому стрілкові функції (Arrow Functions) не можна використовувати як методи об'єктів, якщо всередині планується звернення до контексту через this.
3. Охарактеризуйте лексичне оточення функції та його вплив на механізм роботи замикань (closures). Наведіть практичний приклад.
4. Визначте дві обов'язкові умови, які повинні виконуватися, щоб функцію можна було вважати чистою (Pure Function).
5. Поясніть різницю між методами масиву map() та forEach(). Який з них слід використовувати, якщо потрібно отримати новий трансформований масив.

ТЕМА 2.3. РОБОТА З DOCUMENT OBJECT MODEL (DOM)

Мета: засвоїти концепцію об'єктної моделі документа (DOM) як програмного інтерфейсу для HTML; опанувати методи пошуку, вибірки та навігації по вузлах DOM-дерева; навчитися динамічно змінювати вміст, атрибути та стилі елементів за допомогою JavaScript; вивчити процеси створення, вставлення та видалення нових HTML-вузлів.

План лекції:

1. Концепція DOM-дерева та глобальний об'єкт document.
2. Пошук та вибірка елементів: сучасні методи та типи колекцій.
3. Навігація по DOM-дереву: переміщення між родичами та нащадками.
4. Маніпуляція контентом: зміна тексту, HTML-розмітки та атрибутів.
5. Динамічна зміна структури: створення, додавання та видалення вузлів.

1. Концепція DOM-дерева та глобальний об'єкт document

DOM (Document Object Model – Об'єктна модель документа) – це

стандартизований W3C програмний інтерфейс (API) для HTML-документів. Коли браузер завантажує вебсторінку, він перетворює HTML-код на ієрархічну деревоподібну структуру об'єктів (вузлів), якою може керувати JavaScript.

Кожен тег (наприклад, <body>, <h1>, <a>) стає об'єктом-вузлом. Текст всередині тегів стає текстовим вузлом, а коментарі – вузлами-коментарями. Головною «точкою входу» для будь-яких маніпуляцій зі сторінкою є глобальний об'єкт document.

2. Пошук та вибірка елементів: сучасні методи та типи колекцій

Щоб змінити будь-який елемент на сторінці, скрипт спочатку повинен знайти його в DOM-дереві.

Класичні (застарілі) методи:

- document.getElementById('header') – швидкий пошук одного унікального елемента за його id.
- document.getElementsByClassName('item') та document.getElementsByTagName('div'). Ці методи повертають «живу» колекцію (HTMLCollection). Якщо після вибірки додати новий елемент з таким класом у DOM, колекція автоматично оновиться, що може призвести до неочікуваних результатів.

Сучасні універсальні методи (через CSS-селектори):

- document.querySelector('.card > p') – повертає перший знайдений елемент, що відповідає вказаному CSS-селектору. Якщо нічого не знайдено, повертає null.
- document.querySelectorAll('.list-item') – повертає статичну колекцію (NodeList) усіх відповідних елементів. «Статична» означає, що колекція фіксує стан DOM на момент виклику і не змінюється автоматично. З цією колекцією дуже зручно працювати через вбудований метод forEach.

3. Навігація по DOM-дереву: переміщення між родичами та нащадками

Часто елемент простіше знайти відносно іншого (вже знайденого) елемента. Цей процес називається DOM-навігацією (DOM Traversal).

Існує два набори властивостей для навігації: для всіх типів вузлів (включаючи текстові пробіли) і виключно для вузлів-елементів (тегів). На практиці найчастіше використовують другі:

- Батьківський елемент: element.parentElement (повертає тег, у який вкладено поточний елемент).
- Дочірні елементи: element.children (повертає колекцію всіх вкладених тегів). element.firstElementChild та element.lastElementChild дозволяють швидко

отримати першого чи останнього нащадка.

- Сусідні елементи (брати/сестри): element.previousElementSibling (попередній сусід) та element.nextElementSibling (наступний сусід на тому ж рівні вкладеності).

Важливий метод: element.closest('.container') – шукає найближчого предка, який відповідає CSS-селектору (рухаючись вгору по дереву від самого елемента). Дуже корисний при делегуванні подій.

4. Маніпуляція контентом: зміна тексту, HTML-розмітки та атрибутів

Отримавши посилання на елемент, ми можемо динамічно змінювати його властивості.

Текст та HTML:

- element.textContent – дозволяє безпечно читати або змінювати лише текстовий вміст. Будь-які HTML-теги, передані сюди, будуть екрановані та відображені як звичайний текст (захист від XSS-атак).
- element.innerHTML – дозволяє читати або вставляти HTML-розмітку. Використовувати слід вкрай обережно, лише якщо ви на 100% довіряєте джерелу даних.

Робота з атрибутами: Стандартні атрибути (як id, src, href) можна змінювати напряму (наприклад, img.src = 'new.jpg'). Для нестандартних (кастомних) атрибутів використовуються спеціальні методи:

- element.setAttribute('data-id', '123') – встановити атрибут.
- element.getAttribute('data-id') – отримати значення.
- element.removeAttribute('disabled') – видалити атрибут.

Управління стилями через CSS-класи: Найкраща практика – не змінювати стилі через інлайн-властивість style, а маніпулювати класами за допомогою об'єкта classList:

- element.classList.add('active') / element.classList.remove('hidden')
- element.classList.toggle('dark-theme') – ідеально для перемикачів (додає клас, якщо його немає, і видаляє, якщо є).

5. Динамічна зміна структури: створення, додавання та видалення вузлів

JavaScript дозволяє «будувати» та руйнувати інтерфейс на льоту, не перезавантажуючи сторінку клієнта.

Створення нового елемента:

- const newDiv = document.createElement('div'); (Елемент створюється лише в оперативній пам'яті, його ще не існує на сторінці).

- Після створення ми його наповнюємо: newDiv.textContent = 'Привіт!'; newDiv.classList.add('box');.

Вставлення у DOM-дерево (Рендеринг):

- parentElement.append(newDiv) – вставляє вузол всередину parentElement, у самий кінець (після всіх інших нащадків).
- parentElement.prepend(newDiv) – вставляє на початок (перед першим нащадком).
- element.before(newDiv) та element.after(newDiv) – вставляють вузол на той самий рівень ієрархії, безпосередньо перед або після вказаного елемента.

Видалення:

- element.remove() – повністю видаляє вибраний вузол (та всіх його нащадків) із DOM-дерева.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Здобувачі можуть використовувати інструменти штучного інтелекту для генерації шаблонів складних маніпуляцій з DOM.

- Приклад промпту №1 (Рефакторинг): *«Дій як Senior Frontend Developer. У мене є код, який використовує innerHTML для додавання 100 елементів списку <li> у циклі for. Поясни, чому це повільно та небезпечно, і перепиши код з використанням document.createElement та DocumentFragment.»*
- Приклад промпту №2 (Навігація по DOM): *«Поясни різницю між властивостями childNodes та children у JavaScript. Напиши приклад, де використання childNodes може призвести до помилки через текстові вузли (пробіли в HTML).»*

Питання для обговорення та контролю знань:

1. Поясніть концепцію DOM та обґрунтуйте, чому DOM не є частиною мови JavaScript, а лише API, яке надається браузером
2. Визначте критичну різницю між «живими» колекціями (HTMLCollection) та «статичними» (NodeList), які повертає метод querySelectorAll
3. Поясніть, чому для додавання користувацьких даних на вебсторінку безпечніше використовувати властивість textContent, ніж innerHTML, та дайте визначення XSS-вразливості
4. Назвіть методи DOM API, які дозволяють вставити новостворений елемент div одразу після існуючого елемента h1 на одному рівні вкладеності
5. Опишіть призначення методу closest() та його відмінність від методу querySelector

ТЕМА 2.4. ПОДІЇ ТА ІНТЕРАКТИВНІСТЬ НА КЛІЄНТСЬКІЙ СТОРОНІ

Мета: засвоїти архітектуру подієвої моделі веббраузера; глибоко зрозуміти фази розповсюдження подій (спливання та занурення); опанувати архітектурний патерн делегування подій для оптимізації використання оперативної пам'яті; вивчити специфіку обробки різних категорій подій (миші, клавіатури, форм) та методи керування їхньою стандартною поведінкою.

План лекції:

1. Архітектура подієвої моделі браузера та об'єкт Event.
2. Фази розповсюдження подій: занурення (capturing) та спливання (bubbling).
3. Оптимізація продуктивності: патерн делегування подій (Event Delegation).
4. Класифікація подій: взаємодія з мишею, клавіатурою та формами.
5. Керування потоком: preventDefault, stopPropagation та пасивні слухачі.

1. Архітектура подієвої моделі браузера та об'єкт Event

Браузерне середовище є подієво-орієнтованим (Event-Driven). Це означає, що виконання JavaScript-коду найчастіше ініціюється не послідовно зверху вниз, а у відповідь на певні тригери (події), що генеруються користувачем (клік, скрол) або самою системою (завершення завантаження файлу).

Реєстрація обробників подій: Сучасним стандартом призначення обробників є метод addEventListener(). Його головна перевага над застарілими HTML-атрибутами (на кшталт onclick) полягає в можливості «повісити» на одну подію одразу кілька незалежних функцій-слухачів.

JavaScript

```
const btn = document.querySelector('.action-btn');
const handleClick = (event) => {
   console.log('Подія відбулася!');
};
btn.addEventListener('click', handleClick);
// За необхідності слухач можна видалити (критично для очищення пам'яті у SPA):
// btn.removeEventListener('click', handleClick);
```

Об'єкт події (Event): Щоразу, коли спрацьовує подія, браузер автоматично генерує об'єкт Event і передає його першим аргументом у функцію-обробник. Він містить вичерпну інформацію про подію:

- event.type – тип події (наприклад, «click», «keydown»).
- event.target – найглибший (вихідний) елемент у DOM-дереві, на якому фізично виникла подія.
- event.currentTarget – елемент, до якого безпосередньо прив'язаний слухач addEventListener (він же this всередині класичної функції-обробника).

2. Фази розповсюдження подій: занурення та спливання

Коли на сторінці відбувається подія (наприклад, клік по кнопці всередині кількох вкладених), вона не просто спрацьовує на цій кнопці. Подія проходить складний шлях крізь DOM-дерево.

Стандарт W3C визначає 3 фази розповсюдження події:

1. Фаза занурення (Capturing Phase): Подія починає свій шлях від кореня документа (Window -> Document -> <html> -> <body>) і спускається вниз по ієрархії предків до цільового елемента. За замовчуванням обробники на цій фазі *не спрацьовують*.

2. Фаза цілі (Target Phase): Подія досягає самого елемента (event.target), на якому відбулася дія.

3. Фаза спливання (Bubbling Phase): Подія починає «спливати» у зворотному напрямку – від цільового елемента вгору по ланцюжку його батьків аж до об'єкта Window. Саме на цій фазі за замовчуванням спрацьовує більшість обробників (addEventListener).

Примітка: Деякі події, такі як focus, blur або scroll, не спливають.

3. Оптимізація продуктивності: патерн Делегування подій

Механізм спливання подій лежить в основі одного з найважливіших архітектурних патернів у JavaScript – Делегування подій (Event Delegation).

Проблема: Якщо у вас є динамічний список товарів (<ul>) із 1000 елементів (<li>), і кожному потрібна кнопка «Видалити», додавання 1000 окремих обробників addEventListener призведе до катастрофічного витоку оперативної пам'яті браузера. Крім того, при додаванні 1001-го елемента через JS, йому доведеться вручну призначати новий обробник.

Рішення (Делегування): Замість того, щоб вішати слухачі на кожного нащадка, ми призначаємо один єдиний обробник на їхнього спільного батька (наприклад, на <ul>). Коли користувач клікає на кнопку в <li>, подія спливає до <ul>, де її перехоплює наш загальний обробник. Далі за допомогою event.target ми визначаємо, чи клік був саме по кнопці.

JavaScript

```
const list = document.querySelector('.product-list');
list.addEventListener('click', (event) => {
```

```
    // Перевіряємо, чи клік був саме по кнопці з класом .delete-btn
    if (event.target.closest('.delete-btn')) {
        const itemToRemove = event.target.closest('li');
        itemToRemove.remove();
    }
});
```

4. Класифікація подій: взаємодія з мишею, клавіатурою та формами

Розробник повинен вміти обирати правильний тип події для конкретної задачі інтерактивності.

1) Події миші (Mouse Events):

a) click, dblclick (подвійний клік).

b) mousedown / mouseup (кнопку натиснуто / відпущено).

c) mousemove (рух курсора – генерує події безперервно, вимагає обережності для збереження продуктивності).

d) mouseenter / mouseleave (наведення/зняття курсора; на відміну від mouseover/mouseout, ці події *не спливають*).

2) Події клавіатури (Keyboard Events):

a) keydown (клавішу натиснуто) та keyup (відпущено). Об'єкт події містить властивості event.key (наприклад, «Enter», «a») та event.code (фізичне розташування клавіші).

3) Події форм (Form Events):

a) submit (відправка форми). Завжди вішається на сам тег <form>, а не на кнопку відправки.

b) input (спрацьовує синхронно при кожній зміні тексту в <input> або <textarea>).

c) change (спрацьовує при зміні значення, але після втрати фокусу елементом).

5. Керування потоком: preventDefault, stopPropagation та пасивні слухачі

Часто стандартна логіка браузера заважає реалізації кастомного інтерфейсу. JavaScript надає методи для контролю цього процесу безпосередньо через об'єкт Event:

- event.preventDefault() (Скасування стандартної дії): Зупиняє дефолтну поведінку браузера. Наприклад, запобігає перезавантаженню сторінки під час події submit у формах, або забороняє перехід за посиланням <a>, якщо воно використовується як UI-кнопка.

- event.stopPropagation() (Зупинка спливання): Припиняє подальше

просування події вгору по DOM-дереву. Якщо клікнути на елемент із цим методом, жоден батьківський обробник цієї ж події не спрацює. *Увага: цей метод слід використовувати дуже обережно, оскільки він ламає архітектуру для скриптів аналітики (наприклад, Google Analytics), які слухають події на рівні document.*

- Пасивні слухачі ({ passive: true }): Спеціальний параметр для addEventListener, який повідомляє браузеру, що всередині обробника *не буде* викликано preventDefault(). Це критично важливо для подій scroll та touchmove на мобільних пристроях – браузер може одразу виконувати плавне прокручування сторінки, не чекаючи завершення виконання вашого JS-коду.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Дебагінг спливання): *«Дій як Senior Frontend Developer. У мене є вкладена структура HTML: div.card > button.like. На обох елементах висить обробник події click. Коли я клікаю на кнопку, спрацьовує клік і на картці. Як мені це ізолювати, і чи є stopPropagation найкращим рішенням у цій ситуації? Запропонуй альтернативний підхід з використанням event.target.»*

Приклад промпту №2 (Клавіатурні події): *«Напиши JS-код для модального вікна. Реалізуй логіку, за якою вікно закривається при натисканні клавіші Escape. Додай перевірку через event.key та поясни, на який елемент (window, document чи саме модальне вікно) краще вішати слухач keydown.»*

Питання для обговорення та контролю знань:

1. Опишіть життєвий цикл події в браузері. Вкажіть три фази розповсюдження події та визначте, на якій з них зазвичай спрацьовують обробники, додані через addEventListener.
2. Поясніть різницю між властивостями event.target та event.currentTarget в об'єкті події.
3. Дайте визначення делегування подій (Event Delegation) та опишіть, які проблеми з продуктивністю та підтримкою коду воно вирішує.
4. Назвіть типові сценарії веброзробки, в яких використання методу event.preventDefault() є критично необхідним.
5. Поясніть, чому зловживання методом event.stopPropagation() вважається антипатерном і як це може зашкодити інтеграції сторонніх сервісів, наприклад, систем аналітики.

ЗМІСТОВИЙ МОДУЛЬ 3. ОСНОВИ PHP

ТЕМА 3.1. ВСТУП ДО PHP ТА СЕРВЕРНОГО ПРОГРАМУВАННЯ

Мета: засвоїти фундаментальні принципи серверного програмування (Backend); зрозуміти роль мови PHP в архітектурі сучасних вебзастосунків та її взаємодію з вебсервером; вивчити базовий синтаксис, систему типів даних, керуючі конструкції мови; опанувати основи отримання та обробки даних від клієнта через суперглобальні масиви.

План лекції:

1. Роль PHP у веброзробці та життєвий цикл серверного запиту.
2. Локальне середовище розробки та базова структура PHP-скрипта.
3. Синтаксис мови: змінні, константи та динамічна типізація.
4. Керуючі алгоритмічні конструкції: умови та цикли (foreach).
5. Взаємодія з клієнтом: обробка форм (методи GET та POST).

1. Роль PHP у веброзробці та життєвий цикл серверного запиту

PHP (PHP: Hypertext Preprocessor) – це широко розповсюджена мова програмування загального призначення з відкритим вихідним кодом (Open Source), яка спеціально сконструйована для веброзробки і виконується на стороні сервера (Backend).

На відміну від JavaScript (який виконується у браузері клієнта і маніпулює DOM), PHP працює «під капотом» системи. Користувач ніколи не бачить вихідного PHP-коду, він отримує лише результат його виконання (зазвичай чистий HTML-код).

Життєвий цикл обробки запиту:

1. Запит (Request): Браузер користувача надсилає HTTP-запит до сервера (наприклад, перехід за адресою site.com/index.php).
2. Вебсервер (Apache або Nginx): Приймає запит. Бачачи розширення .php, вебсервер розуміє, що це не статичний файл, і передає його на обробку інтерпретатору PHP.
3. Виконання: Інтерпретатор читає код зверху вниз, виконує логіку (звертається до бази даних, виконує обчислення, перевіряє сесії).
4. Генерація відповіді (Response): Інтерпретатор формує готовий HTML-документ (або JSON-рядок для API) і повертає його вебсерверу.
5. Доставка: Вебсервер відправляє згенерований результат назад у браузер клієнта для рендерингу.

2. Локальне середовище розробки та базова структура PHP-скрипта

Оскільки PHP виконується на сервері, ви не можете просто відкрити файл

.php у браузері, як HTML-сторінку. Для розробки необхідно розгорнути локальне серверне середовище (Localhost). У навчальних цілях найчастіше використовують готові збірки: XAMPP, MAMP, OpenServer або сучасні рішення на базі Docker. До складу такої збірки зазвичай входить вебсервер (Apache), інтерпретатор PHP та СУБД (MySQL).

Базова структура скрипта: PHP-код інтегрується безпосередньо в HTML-документ за допомогою спеціальних тегів-обмежувачів <?php ... ?>.

```
PHP
<!DOCTYPE html>
<html lang=«uk»>
<head><title>Перший скрипт</title></head>
<body>
  <h1>Тестова сторінка</h1>
  <?php
    // Все, що знаходиться між цими тегами, обробляється як PHP
    echo «<p>Цей абзац згенеровано динамічно на сервері!</p>«;
  ?>
</body>
</html>
```

Конструкція echo (або функція print()) використовується для виводу даних у вихідний HTML-потік. Кожна інструкція в PHP обов'язково має закінчуватися крапкою з комою (;).

3. Синтаксис мови: змінні, константи та динамічна типізація

Як і JavaScript, PHP є мовою з динамічною слабкою типізацією (хоча сучасні версії PHP 7 і 8 дозволяють використовувати строгу типізацію).

Змінні: Усі змінні в PHP обов'язково починаються зі знака долара ($). Регістр літер має значення ($name і $Name – це різні змінні).

```
PHP
$userName = «Олександр»; // Тип String
$age = 20;               // Тип Integer
$price = 199.99;         // Тип Float
$isStudent = true;       // Тип Boolean
```

Константи: Константи використовуються для збереження значень, які ніколи не змінюються (наприклад, налаштування підключення до БД). Їх можна визначити двома способами:

```
PHP
define(«DB_HOST», «localhost»); // Традиційний спосіб
const SITE_NAME = «My Portal»;  // Сучасний спосіб
```

Звернення до константи відбувається без знака $.

Рядки та інтерполяція: Рядки в подвійних лапках (« «) підтримують інтерполяцію – можливість вбудовувати змінні безпосередньо в текст. Одинарні лапки (' ') виводять текст рівно так, як він написаний, без обробки змінних.

PHP

```
echo «Привіт, $userName!»; // Виведе: Привіт, Олександр!
echo 'Привіт, $userName!'; // Виведе: Привіт, $userName!
```

4. Керуючі алгоритмічні конструкції: умови та цикли

Синтаксис умов та циклів у PHP був успадкований від мови C і є практично ідентичним до JavaScript.

- Умови: if, else, elseif. Існує також зручний синтаксис match (з'явився у PHP 8), який замінює громіздкий switch.
- Цикли: while, do-while, for.

Особливість PHP – цикл foreach: Оскільки в PHP найпопулярнішою структурою даних є асоціативні масиви (де замість числових індексів використовуються ключі-рядки), цикл foreach є незамінним інструментом для ітерації.

PHP

```
$user = [
   «name» => «Олександр»,
   «role» => «Адміністратор»,
   «age» => 25
];
// Ітерація з отриманням ключа та значення
foreach ($user as $key => $value) {
   echo «<p>Властивість <b>$key</b> має значення: $value</p>«;
}
```

5. Взаємодія з клієнтом: обробка форм (методи GET та POST)

Головна задача PHP – приймати дані від користувача (наприклад, з HTML-форми), обробляти їх і видавати результат. Дані передаються за допомогою методів HTTP-запиту, а інтерпретатор PHP автоматично поміщає їх у суперглобальні асоціативні масиви ($_GET та $_POST).

- Метод GET ($_GET): Дані передаються відкрито, безпосередньо в URL-адресі сторінки (наприклад, search.php?query=телефон&sort=price). Використовується виключно для отримання даних (пошук, фільтрація, пагінація). Не підходить для передачі паролів або великих текстів, оскільки

URL має обмеження по довжині і зберігається в історії браузера.

- Метод POST ($_POST): Дані передаються приховано у тілі (body) HTTP-запиту. Використовується для відправки чутливих даних або зміни стану сервера (реєстрація, авторизація, завантаження файлу, оплата).

Приклад обробки простої форми (POST):

HTML

```
<form action=«process.php» method=«POST»>
   <input type=«text» name=«login» required>
   <button type=«submit»>Увійти</button>
</form>
```

PHP

```
// process.php (Серверна частина)
if ($_SERVER[«REQUEST_METHOD»] === «POST») {
   // Отримуємо дані за атрибутом name=«login»
   // Функція htmlspecialchars захищає від XSS-атак (впровадження JS-коду)
   $login = htmlspecialchars($_POST['login']);

   echo «Вітаємо, $login! Ви успішно авторизувались.»;
}
```

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Штучний інтелект допоможе швидко налаштувати робоче середовище та зрозуміти нюанси серверної безпеки.

- Приклад промпту №1 (Налаштування сервера): *«Дій як DevOps інженер. Я встановив пакет XAMPP на Windows, але при запуску Apache виникає помилка конфлікту порту 80. Поясни покроково, як мені змінити конфігураційні файли (httpd.conf), щоб перенести Apache на порт 8080.»*

- Приклад промпту №2 (Безпека даних): *«Що таке XSS (Cross-Site Scripting) у контексті PHP? Напиши вразливий код обробки форми, а потім продемонструй, як виправити його за допомогою функції htmlspecialchars(). Поясни, що саме робить ця функція.»*

Питання для обговорення та контролю знань:

1. Поясніть принципову різницю між виконанням коду мовою JavaScript та мовою PHP. Визначте, де саме виконується кожен з них.

2. Поясніть призначення локальних серверів типу XAMPP або Docker для розробника. Обґрунтуйте, чому не можна просто відкрити PHP-файл у браузері подвійним кліком.

3. Опишіть різницю між одинарними (' ') та подвійними (« «) лапками при створенні рядкових змінних у PHP.

4. Поясніть, чому цикл foreach є більш зручним для роботи з масивами у PHP порівняно з класичним циклом for.

5. Порівняйте HTTP-методи GET та POST. Визначте випадки, в яких використання методу GET є категорично забороненим з міркувань кібербезпеки.

ТЕМА 3.2. УПРАВЛЯЮЧІ КОНСТРУКЦІЇ ТА ЦИКЛИ В PHP

Мета: засвоїти механізми керування потоком виконання PHP-скриптів; глибоко зрозуміти логіку роботи умовних операторів та конструкцій множинного вибору (зокрема сучасного виразу match); опанувати всі види циклів для обробки наборів даних; вивчити специфіку ітерації асоціативних масивів за допомогою конструкції foreach.

План лекції:

1. Логічні оператори, вирази та перевірка на істинність.
2. Базові умовні конструкції (if, elseif, else) та тернарний оператор.
3. Еволюція множинного вибору: від switch до виразу match (PHP 8).
4. Класичні циклічні конструкції: while, do-while та for.
5. Специфіка PHP: ітерація масивів через foreach та керування ітераціями.

1. Логічні оператори, вирази та перевірка на істинність

Керуючі конструкції базуються на оцінці виразів як істинних (true) або хибних (false). У PHP, як і в багатьох інших мовах, діє правило неявного приведення типів (Type Juggling) при логічних перевірках.

- Хибні (Falsy) значення в PHP: false, 0 (ціле), 0.0 (дробове), «» (порожній рядок), рядок «0», масив без елементів [], та спеціальне значення null.
- Істинні (Truthy): Усе інше (включно з від'ємними числами та рядками, що містять лише пробіли).

Логічні оператори:

- && (АБО and) – Логічне І. Повертає true, якщо обидва операнди істинні.
- || (АБО or) – Логічне АБО. Повертає true, якщо хоча б один операнд істинний.
- ! – Логічне НІ (заперечення).
- *Важливий нюанс PHP:* Оператори && / || мають вищий пріоритет

виконання, ніж and / or. У повсякденній практиці рекомендується використовувати саме символьні варіанти (&&, ||), щоб уникнути несподіваних помилок при присвоєнні змінних.

2. Базові умовні конструкції та тернарний оператор

Конструкція if-elseif-else є фундаментом алгоритмізації. Синтаксис повністю ідентичний до мов C, Java чи JavaScript.

PHP

```
$userAge = 20;

if ($userAge >= 18) {
   echo «Доступ дозволено.»;
} elseif ($userAge === 17) {
   echo «Зачекайте ще один рік.»;
} else {
   echo «Доступ заборонено.»;
}
```

Альтернативний синтаксис (для шаблонів): Коли PHP-код змішується з великими блоками HTML, використання фігурних дужок стає незручним і нечитабельним. Для цього PHP пропонує синтаксис на основі двокрапки:

PHP

```
<?php if ($userAge >= 18): ?>
   <h1>Вітаємо на сайті!</h1>
<?php else: ?>
   <h1>Вам сюди не можна.</h1>
<?php endif; ?>
```

Короткі записи (Тернарний оператор та Null Coalescing):

- Тернарний оператор (?:): Дозволяє записати простий if-else в один рядок: $status = ($userAge >= 18) ? «Дорослий» : «Дитина»;

- Оператор об'єднання з null (??): З'явився в PHP 7. Ідеальний для безпечного отримання даних з форм або масивів. Він повертає лівий операнд, якщо той існує і не дорівнює null, інакше – правий. $username = $_POST['username'] ?? «Гість»;

3. Еволюція множинного вибору: від switch до виразу match

Коли необхідно порівняти одну змінну з великою кількістю різних значень, ланцюжки elseif стають громіздкими.

Класичний switch: Працює на основі *нестрогого порівняння* (==). Вимагає обов'язкового використання break після кожного блоку case, інакше відбудеться

«провалювання» (fall-through) і виконається код наступних блоків.

```
PHP
switch ($role) {
   case 'admin':
      echo «Повний доступ»;
      break; // Критично важливо!
   case 'editor':
      echo «Доступ до статей»;
      break;
   default:
      echo «Доступ лише для читання»;
}
```

Сучасний вираз match (починаючи з PHP 8.0): Це значно безпечніша та компактніша альтернатива switch. Головні відмінності:

1. Використовує строге порівняння (===). Значення «1» і число 1 не співпадуть.

2. Не потребує break (провалювання неможливе за задумом).

3. Є *виразом* (expression), тобто повертає значення, яке можна одразу присвоїти змінній.

```
PHP
$message = match ($role) {
   'admin' => «Повний доступ»,
   'editor' => «Доступ до статей»,
   default => «Доступ лише для читання»,
};
echo $message;
```

4. Класичні циклічні конструкції: while, do-while та for

Цикли дозволяють багаторазово виконувати один і той самий блок коду.

- while (Цикл з передумовою): Виконується, поки умова істинна. Якщо умова хибна від самого початку, тіло циклу не виконається жодного разу. Часто використовується для зчитування рядків із файлу або бази даних.

- do-while (Цикл з постумовою): Гарантує виконання тіла циклу щонайменше один раз, навіть якщо умова спочатку хибна, оскільки перевірка відбувається в кінці.

- for (Цикл із лічильником): Найкраще підходить для ситуацій, коли кількість ітерацій відома заздалегідь.

```
PHP
for ($i = 1; $i <= 5; $i++) {
```

```
    echo «Ітерація номер $i <br>«;
}
```

5. Специфіка PHP: ітерація масивів через foreach та керування ітераціями

Оскільки в PHP структури даних переважно реалізуються через асоціативні масиви, класичний цикл for (який покладається на числові індекси) є незручним. Для цієї мети існує спеціальний цикл foreach.

Синтаксис отримання лише значень:

```
PHP
$fruits = [«Яблуко», «Банан», «Апельсин»];
foreach ($fruits as $fruit) {
    echo «Фрукт: $fruit <br>«;
}
```

Синтаксис отримання ключів та значень (Критично для асоціативних масивів):

```
PHP
$user = [
    «name» => «Олена»,
    «email» => «olena@test.com»,
    «role» => «admin»
];
foreach ($user as $key => $value) {
    echo «Поле <b>$key</b> містить значення: $value <br>«;
}
```

Керування ітераціями (break та continue): Ці ключові слова працюють у всіх видах циклів.

- break; – негайно та повністю зупиняє виконання циклу.
- continue; – перериває лише поточну ітерацію і одразу переходить до наступної (або до перевірки умови).

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Інструменти генеративного ШІ допомагають швидко засвоювати синтаксичні особливості нових версій PHP.

- Приклад промпту №1 (Рефакторинг на сучасний PHP): *«Дій як Senior Backend Developer. У мене є старий PHP-код, який використовує довгий ланцюжок switch-case для визначення знижки за категорією клієнта. Перепиши цей код, використовуючи сучасний вираз match (PHP 8). Поясни, чому новий підхід є безпечнішим щодо типізації.»*

- Приклад промпту №2 (Оператори порівняння): *«Поясни мені різницю між тернарним оператором ?: та оператором об'єднання з null ?? у PHP. Наведи два практичні приклади: один для перевірки даних з $_GET, а інший – для присвоєння дефолтного значення змінній.»*

Питання для обговорення та контролю знань:

1. Поясніть, що таке «хибні» (falsy) значення в PHP. Визначте, чи буде рядок «0» сприйнятий конструкцією if як true чи як false.
2. Опишіть, у чому полягає критична небезпека відсутності ключового слова break у структурі switch. Поясніть поняття «fall-through».
3. Назвіть три головні відмінності сучасного виразу match (PHP 8) від класичного оператора switch.
4. Визначте, який цикл гарантує, що блок коду буде виконано щонайменше один раз, незалежно від початкової істинності умови.
5. Поясніть, чому цикл foreach є кращим вибором для обробки даних, отриманих з бази даних (які зазвичай повертаються у вигляді асоціативних масивів), порівняно з циклом for.

ТЕМА 3.3. МАСИВИ, РЯДКИ ТА ЇХ ОБРОБКА

Мета: засвоїти внутрішню архітектуру масивів у PHP (як упорядкованих хеш-таблиць); вивчити синтаксис створення та маніпуляції індексованими, асоціативними та багатовимірними масивами; опанувати багату бібліотеку вбудованих функцій PHP для сортування, фільтрації та пошуку даних; зрозуміти принципи роботи з текстовими рядками, зокрема безпечну обробку багатобайтових кодувань (UTF-8) та базове використання регулярних виразів.

План лекції:

1. Архітектура масивів у PHP: індексовані, асоціативні та багатовимірні структури.
2. Маніпуляції з елементами: додавання, видалення та об'єднання масивів.
3. Вбудовані функції для роботи з масивами: сортування, пошук та трансформація.
4. Основи обробки рядків: інтерполяція, форматування та багатобайтові функції (mbstring).
5. Пошук і заміна в тексті: строкові функції та основи регулярних виразів (PCRE).

1. Архітектура масивів у PHP: індексовані, асоціативні та

багатовимірні

У PHP масив (Array) насправді є складною структурою даних – упорядкованою хеш-таблицею (карткою), яка зіставляє значення з ключами. Це робить масиви у PHP надзвичайно гнучкими: вони можуть працювати і як класичні списки, і як словники (хеші), і як стеки або черги.

Індексовані масиви (з числовими ключами): Ключі призначаються автоматично, починаючи з нуля, якщо їх не вказати явно.

PHP

$colors = [«Червоний», «Зелений», «Синій»]; // Сучасний короткий синтаксис (PHP 5.4+)

$legacyColors = array(«Червоний», «Зелений»); // Класичний синтаксис

echo $colors[0]; // Виведе: Червоний

Асоціативні масиви (зі строковими ключами): Використовуються, коли елементам потрібно дати логічні імена (аналог об'єктів у JavaScript). Для присвоєння значення ключу використовується оператор =>.

PHP

```
$user = [
   «id» => 101,
   «name» => «Олексій»,
   «role» => «Менеджер»
];
echo $user[«name»]; // Виведе: Олексій
```

Багатовимірні масиви: Це масиви, елементами яких є інші масиви. Найчастіше використовуються для представлення табличних даних з бази даних.

PHP

```
$catalog = [
   [«id» => 1, «title» => «Ноутбук», «price» => 15000],
   [«id» => 2, «title» => «Миша», «price» => 600]
];
echo $catalog[1][«title»]; // Виведе: Миша
```

2. Маніпуляції з елементами: додавання, видалення та об'єднання

У PHP не потрібно заздалегідь оголошувати розмір масиву – він розширюється динамічно.

- Додавання елементів: Найшвидший спосіб додати елемент у кінець індексованого масиву – використати порожні квадратні дужки []:

PHP

```
$fruits = [«Яблуко»];
```

$fruits[] = «Груша»; // Масив тепер: [«Яблуко», «Груша»]

// Для додавання кількох елементів одразу:

array_push($fruits, «Банан», «Апельсин»);

- Додавання в асоціативний масив: Вказується конкретний ключ. $user[«email»] = «test@mail.com»;

- Видалення елементів: Конструкція unset() видаляє елемент за ключем або індексом. *Важливо:* при видаленні з індексованого масиву за допомогою unset, ключі не переіндексовуються автоматично (виникає «дірка» в нумерації). unset($fruits[1]);

- Об'єднання масивів: Використовується функція array_merge($arr1, $arr2) або сучасний оператор розпакування (Spread operator) [...$arr1, ...$arr2] (починаючи з PHP 7.4).

3. Вбудовані функції для роботи з масивами

PHP славиться величезною стандартною бібліотекою функцій (понад 80 функцій лише для масивів).

Сортування:

- sort(&$arr) – сортує індексований масив за зростанням значень (ключі скидаються).

- rsort(&$arr) – сортує за спаданням.

- asort(&$arr) – сортує асоціативний масив за значеннями, зберігаючи прив'язку ключів!

- ksort(&$arr) – сортує асоціативний масив за ключами.

Пошук та перевірка:

- count($arr) – повертає кількість елементів (довжину масиву).

- in_array($needle, $haystack) – перевіряє, чи існує певне значення в масиві (повертає boolean).

- array_key_exists($key, $arr) – перевіряє наявність ключа.

Трансформація (Функціональний підхід):

- array_map($callback, $arr) – застосовує функцію до кожного елемента і повертає новий масив (аналог map у JS).

- array_filter($arr, $callback) – фільтрує масив, залишаючи лише ті елементи, для яких callback повертає true.

4. Основи обробки рядків: інтерполяція та багатобайтові кодування

Робота з текстом є однією з найчастіших задач на бекенді.

Безпека та кодування (mbstring): Стандартні строкові функції PHP (наприклад, strlen або strtolower) розроблялися для однобайтових кодувань (ASCII). Якщо передати їм український текст (UTF-8, де одна літера може

займати 2 байти), вони повернуть некоректний результат. Тому для роботи з кирилицею завжди слід використовувати функції з префіксом mb_ (Multi-Byte).

- strlen(«Київ») поверне 8 (неправильно, рахує байти).
- mb_strlen(«Київ», «UTF-8») поверне 4 (правильно, рахує символи).
- mb_strtoupper(«привіт») – переведе у верхній регістр («ПРИВІТ»).

Очищення та форматування:

- trim($str) – видаляє пробіли та невидимі символи (переноси рядків) з початку та кінця рядка. Це обов'язковий крок при отриманні даних з форм ($_POST).
- htmlspecialchars($str) – перетворює спеціальні HTML-символи (на кшталт < та >) на безпечні сутності (<). Критично важливо для захисту від XSS-атак при виведенні даних користувача на сторінку.

5. Пошук і заміна в тексті: строкові функції та регулярні вирази

Класичні строкові функції (швидкі, але прості):

1) strpos($haystack, $needle) – шукає позицію першого входження підрядка. Якщо не знайдено, повертає false.
2) str_replace($search, $replace, $subject) – замінює всі входження рядка пошуку на рядок заміни.
3) Конвертація «Масив <-> Рядок»:

a) explode($separator, $string) – розбиває рядок на масив (наприклад, CSV-рядок «яблуко,банан,груша» за розділювачем «,»).

b) implode($glue, $array) – склеює масив у єдиний текстовий рядок.

Регулярні вирази (PCRE – Perl Compatible Regular Expressions): Коли базових функцій недостатньо (наприклад, потрібно перевірити валідність email-адреси або знайти всі номери телефонів у тексті), використовується сімейство функцій preg_*.

- preg_match($pattern, $subject) – перевіряє, чи відповідає рядок регулярному виразу.
- preg_replace($pattern, $replacement, $subject) – виконує складну заміну за патерном.
- *Приклад патерну для перевірки лише літер та цифр:* /^[a-zA-Z0-9]+$/

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Функціональний PHP): *«Дій як PHP Senior Developer. У мене є асоціативний масив товарів (id, назва, ціна, категорія). Напиши код, який використовує функції array_filter та array_map, щоб спочатку відфільтрувати товари з категорії 'Електроніка', а потім повернути новий масив, що містить лише їхні назви. Поясни використання стрілкових*

функцій (arrow functions fn()) у PHP 7.4+.»

Приклад промпту №2 (Регулярні вирази): *«Згенеруй регулярний вираз для функції preg_match у PHP, який ідеально підходить для валідації складності пароля (мінімум 8 символів, хоча б одна велика літера, одна мала та одна цифра). Детально поясни синтаксис (lookahead assertions), який ти використав у патерні.»*

Питання для обговорення та контролю знань:

1. Поясніть, чим асоціативні масиви в PHP відрізняються від індексованих. Опишіть синтаксис, який використовується для створення асоціативних ключів.
2. Поясніть, чому при роботі з українським текстом (кодування UTF-8) використання стандартної функції strlen призводить до логічних помилок. Визначте, яку функцію слід використовувати замість неї.
3. Опишіть різницю між функціями sort() та asort(). Визначте, яку з них необхідно використовувати для асоціативних масивів, щоб не втратити прив'язку ключ-значення.
4. Поясніть, для чого використовується функція explode(). Наведіть практичний приклад, де конвертація рядка в масив є необхідною.
5. Поясніть, що відбувається з ключами індексованого масиву, якщо видалити елемент із середини за допомогою конструкції unset(). Опишіть, як можна відновити правильну нумерацію.

ТЕМА 3.4. ФУНКЦІЇ ТА ОРГАНІЗАЦІЯ КОДУ В PHP

Мета: засвоїти синтаксис створення функцій користувача в PHP; глибоко зрозуміти специфіку локальної та глобальної області видимості змінних (яка кардинально відрізняється від JavaScript); опанувати роботу з анонімними та стрілковими функціями; вивчити механізми строгої типізації аргументів і значень, що повертаються; навчитися модульно організовувати проєкт за допомогою конструкцій підключення файлів.

План лекції:

1. Створення функцій користувача: аргументи, параметри за замовчуванням та іменовані аргументи.
2. Область видимості змінних: локальна, глобальна та статична (static).
3. Анонімні функції (Closures) та стрілкові функції (fn()).
4. Сучасний PHP: строга типізація (declare) та типізація повернення.
5. Організація коду: модульність через include та require.

1. Створення функцій користувача: аргументи та іменовані параметри

Функції в PHP оголошуються за допомогою ключового слова function. Вони можуть приймати аргументи та повертати результат за допомогою return. Якщо функція нічого не повертає явно, вона автоматично повертає null.

Базовий синтаксис та параметри за замовчуванням:

PHP

```
function calculateDiscount($price, $discount = 10) {
   return $price - ($price * ($discount / 100));
}
echo calculateDiscount(1000); // Виведе 900 (використано 10%)
```

Іменовані аргументи (Нововведення PHP 8.0): Дозволяють передавати значення у функцію, вказуючи ім'я параметра, а не його позицію. Це робить код значно читабельнішим, особливо якщо функція має багато параметрів за замовчуванням.

PHP

```
// Передаємо лише ціну та тип клієнта, пропускаючи знижку
$finalPrice = calculateDiscount(price: 1500, customerType: 'VIP');
```

2. Область видимості змінних: локальна, глобальна та статична

Це найважливіша відмінність PHP від JavaScript. У PHP функції повністю ізольовані від зовнішнього середовища. Функція не бачить змінних, створених поза нею, а зовнішній код не бачить змінних, створених всередині функції.

- Локальна область: Змінна існує лише під час виконання функції і знищується після її завершення.
- Глобальна область (global): Щоб достукатися до зовнішньої змінної зсередини функції, її треба явно імпортувати за допомогою ключового слова global. *Увага: у сучасній розробці використання global вважається поганою практикою (антипатерном), оскільки порушує інкапсуляцію. Дані слід передавати через аргументи!*

PHP

```
$tax = 20; // Глобальна змінна
function calculateTotal($price) {
   global $tax; // Без цього рядка PHP видасть помилку «Undefined variable»
   return $price + $tax;
}
```

- Статична область (static): Локальна змінна, яка не знищується після

завершення функції, а зберігає своє значення між викликами. Ідеально підходить для створення лічильників.

3. Анонімні функції (Closures) та стрілкові функції

Функції, які не мають імені, часто використовуються як колбеки (наприклад, у функціях array_map чи array_filter).

Анонімні функції та ключове слово use: Оскільки PHP ізолює область видимості, анонімна функція також не бачить зовнішніх змінних. Щоб передати їй зовнішню змінну (створити замикання), використовується конструкція use.

PHP

```
$multiplier = 3;
$numbers = [1, 2, 3];

// Без 'use ($multiplier)' функція не знала б, що таке $multiplier
$result = array_map(function($n) use ($multiplier) {
   return $n * $multiplier;
}, $numbers);
```

Стрілкові функції (Arrow Functions, PHP 7.4+): Більш лаконічний синтаксис fn() => вираз. Головна їхня перевага – вони автоматично захоплюють змінні із зовнішньої області видимості за значенням (не потрібно писати use).

PHP

```
$multiplier = 3;
$result = array_map(fn($n) => $n * $multiplier, $numbers);
```

4. Сучасний PHP: строга типізація та типізація повернення

Історично PHP був мовою зі слабкою типізацією. Але для створення надійних корпоративних застосунків цього недостатньо. Сучасний PHP дозволяє (і заохочує) вказувати типи даних для аргументів та значень, що повертаються.

Оголошення типів (Type Hinting):

PHP

```
// Приймає два цілих числа, повертає число з рухомою комою (float)
function divide(int $a, int $b): float {
   return $a / $b;
}
```

Строга типізація (strict_types): За замовчуванням PHP спробує перетворити типи (наприклад, рядок «5» на число 5). Щоб заборонити це і змусити PHP видавати фатальну помилку (TypeError) при невідповідності типів, на самому початку файлу (першим рядком) пишуть:

PHP

declare(strict_types=1);

-*Об'єднані типи (Union Types, PHP 8.0):* Дозволяють вказати кілька можливих типів: function process(int|float $value).

5. Організація коду: модульність через include та require

Щоб не писати тисячі рядків коду в одному файлі index.php, проєкт розбивають на логічні модулі (наприклад, функції для роботи з БД виносять в окремий файл).

Для підключення одного PHP-файлу до іншого існують 4 мовні конструкції:

1. include 'filename.php'; – намагається підключити файл. Якщо файл не знайдено, видає попередження (Warning), але виконання скрипта продовжується. Використовується для підключення некритичних елементів (наприклад, HTML-футера сторінки).

2. require 'filename.php'; – намагається підключити файл. Якщо файл не знайдено, видає фатальну помилку (Fatal Error) і негайно зупиняє виконання всього скрипта. Використовується для критично важливих файлів (конфігурації, підключення до БД, класи).

3. include_once / require_once: – працюють так само, але PHP запам'ятовує, чи був уже підключений цей файл раніше. Якщо так, він не буде підключати його вдруге (запобігає помилкам переозначення функцій).

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Область видимості): *«Дій як досвідчений PHP-ментор. Я вивчав JavaScript, де функції мають доступ до зовнішніх змінних за замовчуванням. Чому розробники PHP вирішили зробити функції повністю ізольованими? Поясни концепцію ключового слова use в анонімних функціях.»*

Приклад промпту №2 (Типізація та архітектура): *«Напиши приклад PHP-файлу, який включає declare(strict_types=1);. Створи функцію, яка приймає масив цін (тільки масив чисел) та повертає їх суму (float). Продемонструй, як PHP відреагує, якщо я спробую передати масив з одним рядковим значенням.»*

Питання для обговорення та контролю знань:

1. Поясніть, чим відрізняється поведінка локальних змінних у PHP від JavaScript щодо доступу до зовнішнього (глобального) середовища.

2. Поясніть, для чого використовується ключове слово static під час оголошення змінної всередині функції. Наведіть приклад використання такої змінної.

3. Опишіть принципову різницю в поведінці конструкцій include та require, якщо файл, який намагаються підключити, не існує.

4. Поясніть, чому конструкціям require_once віддається перевага перед звичайним require при підключенні файлів із бібліотеками функцій або конфігураціями бази даних.

5. Поясніть, що робить директива declare(strict_types=1) і чому її використання вважається стандартом якості (Best Practice) у сучасній розробці на PHP.

ЗМІСТОВИЙ МОДУЛЬ 4. ВЗАЄМОДІЯ КЛІЄНТА ТА СЕРВЕРА
ТЕМА 4.1. ОБРОБКА ФОРМ ТА ДАНИХ КОРИСТУВАЧА

Мета: зрозуміти механізми передачі даних від клієнтського браузера до сервера через протокол HTTP; навчитися працювати із суперглобальними масивами $_GET, $_POST та $_FILES; опанувати критично важливі принципи безпеки: валідацію (перевірку) та санітизацію (очищення) користувацьких даних; засвоїти алгоритм безпечного завантаження файлів на сервер; вивчити патерн PRG (Post/Redirect/Get).

План лекції:

1. Механізми передачі даних: HTTP-методи та суперглобальні масиви.
2. Валідація даних: перевірка на наявність, типи та використання filter_var().
3. Санітизація та кібербезпека: правило «Ніколи не довіряй користувачу» та захист від XSS.
4. Обробка завантаження файлів: форма multipart/form-data та масив $_FILES.
5. Архітектурний патерн PRG (Post/Redirect/Get) та «липкі» форми (Sticky Forms).

1. Механізми передачі даних: HTTP-методи та суперглобальні масиви

Взаємодія між клієнтом (браузером) і сервером (PHP) відбувається переважно через HTML-форми. Атрибут method тегу <form> визначає, як саме дані будуть запаковані в HTTP-запит.

Метод GET ($_GET): Дані передаються відкрито як частина URL-адреси (Query String). Використовується для ідемпотентних запитів – таких, що не змінюють стан сервера (пошук, фільтрація, перехід по сторінках).

Метод POST ($_POST): Дані передаються приховано в тілі HTTP-запиту. Використовується для чутливої інформації (паролі) та для дій, що змінюють дані на сервері (створення акаунта, оплата, публікація статті).

Суперглобальний масив $_REQUEST: Містить дані з $_GET, $_POST та $_COOKIE одночасно. Його використання не рекомендується з міркувань безпеки, оскільки він розмиває розуміння того, звідки саме прийшли дані.

2. Валідація даних: перевірка на наявність та типи

Перш ніж працювати з даними, сервер повинен переконатися, що вони взагалі існують і відповідають очікуваному формату. Це називається валідацією.

Базові перевірки:

-isset($_POST['username']) – перевіряє, чи була змінна передана у запиті (навіть якщо вона порожня).

-empty($_POST['username']) – перевіряє, чи змінна порожня (порожній рядок, 0, null, false).

Спеціалізована валідація (функція filter_var): PHP має вбудований потужний інструмент для перевірки форматів даних:

PHP

$email = $_POST['email'] ?? '';

// Перевірка, чи є рядок валідною email-адресою

if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {

echo «Помилка: Некоректний формат електронної пошти.»;

}

Існують також фільтри FILTER_VALIDATE_URL, FILTER_VALIDATE_INT, FILTER_VALIDATE_IP тощо.

3. Санітизація та кібербезпека: захист від XSS

Головне правило бекенд-розробника: «All input is evil» (Ніколи не довіряй даним від користувача). Навіть якщо на фронтенді (в HTML/JS) є перевірки, зловмисник може легко їх обійти, надіславши POST-запит напряму через інструменти на кшталт Postman.

Санітизація – це процес очищення даних від потенційно небезпечного коду.

1. Очищення від зайвих пробілів: Завжди використовуйте trim(), щоб користувач випадково (або навмисно) не зареєстрував логін, який складається лише з пробілів.

2. Захист від XSS (Cross-Site Scripting): Якщо користувач введе в поле коментаря <script>alert('Зламано!');</script>, і ви виведете це на сторінку «як є», браузер виконає цей JavaScript-код. Щоб цього уникнути, усі дані перед виведенням у HTML потрібно пропускати через функцію htmlspecialchars().

PHP

$safeComment = htmlspecialchars($_POST['comment'], ENT_QUOTES, 'UTF-8');

echo «Ваш коментар: « . $safeComment; // Виведе безпечний текст: <script>...

4. Обробка завантаження файлів: масив $_FILES

Передача файлів (зображень, документів) докорінно відрізняється від передачі звичайного тексту.

Вимоги до HTML-форми: Форма обов'язково повинна мати метод POST

та спеціальний атрибут кодування enctype=«multipart/form-data». Без нього сервер отримає лише ім'я файлу, а не сам файл.

Масив $_FILES: PHP приймає файл, тимчасово зберігає його в системній папці сервера і заповнює суперглобальний масив $_FILES['input_name'], який містить:

- name – оригінальне ім'я файлу на комп'ютері клієнта.
- type – MIME-тип файлу (наприклад, image/jpeg).
- tmp_name – шлях до тимчасового файлу на сервері.
- error – код помилки (якщо 0 або UPLOAD_ERR_OK, все добре).
- size – розмір файлу в байтах.

Алгоритм збереження: Щоб файл не видалився після завершення скрипта, його треба перемістити з тимчасової папки у постійну за допомогою move_uploaded_file():

PHP

```
if ($_FILES['avatar']['error'] === UPLOAD_ERR_OK) {
   $tmpPath = $_FILES['avatar']['tmp_name'];
   $destination = «uploads/» . basename($_FILES['avatar']['name']);

   // Переміщуємо файл
   if (move_uploaded_file($tmpPath, $destination)) {
      echo «Файл успішно завантажено!»;
   }
}
```

Важливо: на практиці завжди треба перевіряти розширення файлу та його розмір, щоб зловмисники не завантажили виконуваний .php скрипт замість картинки.

5. Архітектурний патерн PRG та «липкі» форми (Sticky Forms)

Проблема подвійного надсилання (Double Submit): Якщо після успішної обробки POST-запиту (наприклад, після оплати товару) користувач натисне в браузері кнопку «Оновити сторінку» (F5), браузер запитає: *«Підтвердити повторне надсилання форми?»*. Якщо користувач погодиться, гроші знімуться вдруге.

Рішення – патерн PRG (Post / Redirect / Get): Після успішної обробки POST-даних сервер ніколи не повинен віддавати HTML-сторінку одразу. Він повинен відправити HTTP-заголовок перенаправлення (Redirect) на іншу сторінку (або на ту саму), яка завантажиться методом GET.

PHP

```
// Обробка даних...
```

```
if ($paymentSuccess) {
   header(«Location: success.php»); // Перенаправлення (Redirect)
   exit; // Обов'язкова зупинка подальшого виконання скрипта!
}
```

«Липкі» форми (Sticky Forms): Якщо під час валідації виникла помилка (наприклад, пароль закороткий), користувачеві слід повернути форму з уже заповненими даними (крім паролів), щоб йому не довелося вводити все з нуля. Для цього в атрибут value інпутів підставляють безпечно очищені дані з $_POST.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Безпечне завантаження файлів): *«Дій як спеціаліст з кібербезпеки. Я пишу скрипт на PHP для завантаження аватарів користувачів. Напиши надійну функцію для перевірки того, що завантажений файл дійсно є зображенням (не покладаючись лише на розширення або $_FILES['type'], оскільки їх можна підробити). Використай функцію finfo_file або mime_content_type та поясни їхню роботу.»*

Приклад промпту №2 (PRG патерн): *«Поясни мені архітектурний патерн Post/Redirect/Get (PRG) на прикладі форми реєстрації користувача. Чому використання функції header('Location: ...') потребує наявності exit; одразу після неї, і чому перед header() не повинно бути жодного HTML-виводу?»*

Питання для обговорення та контролю знань:

1. Поясніть, чому передача логіна та пароля через метод GET є критичною помилкою проєктування вебзастосунку. Визначте, де саме ці дані зможе побачити сторонній користувач.
2. Вкажіть, яка функція в PHP використовується для перетворення спеціальних символів HTML на безпечні сутності для захисту від XSS-атак.
3. Поясніть, чому форма для завантаження файлів обов’язково повинна містити атрибут enctype=«multipart/form-data» і метод POST. Опишіть, що відбудеться, якщо забути цей атрибут.
4. Поясніть, з якою метою використовується функція move_uploaded_file(). Опишіть, де фізично знаходиться файл одразу після того, як його прийняв сервер, але до виклику цієї функції.
5. Опишіть, яку проблему (взаємодію з користувачем) вирішує використання архітектурного патерну PRG (Post/Redirect/Get) після успішної обробки даних форми.

ТЕМА 4.2. СЕСІЇ ТА COOKIES: УПРАВЛІННЯ СТАНОМ КОРИСТУВАЧА

Мета: зрозуміти фундаментальну проблему відсутності стану (stateless) у протоколі HTTP; опанувати механізми збереження даних на стороні клієнта (Cookies) та на стороні сервера (Sessions); навчитися реалізовувати процеси авторизації користувачів, створення кошика покупок та персоналізації налаштувань; засвоїти ключові правила безпеки при роботі із сесіями (запобігання викраденню та фіксації сесії).

План лекції:

1. Проблема HTTP Stateless та необхідність управління станом.
2. Робота з Cookies у PHP: створення, читання, видалення та обмеження.
3. Механізм сесій (Sessions): архітектура, суперглобальний масив $_SESSION.
4. Життєвий цикл сесії: старт, збереження даних та повне знищення.
5. Порівняння підходів та кібербезпека: прапорці HttpOnly, Secure та регенерація ID.

1. Проблема HTTP Stateless та необхідність управління станом

Протокол HTTP (HyperText Transfer Protocol), на якому базується весь веб, за своєю природою є stateless (таким, що не зберігає стан). Це означає, що сервер розглядає кожен новий запит від браузера як абсолютно незалежну подію. Коли ви переходите зі сторінки login.php на profile.php, сервер «забуває», хто ви такі. Без додаткових механізмів було б неможливо створити інтернет-магазин (сервер би забував товари в кошику при переході на сторінку оплати) або закриті адмін-панелі.

Для вирішення цієї проблеми були розроблені два взаємопов'язані механізми управління станом (State Management): Cookies (печиво) та Sessions (сесії).

2. Робота з Cookies у PHP: створення, читання та видалення

Cookie – це невеликий фрагмент даних (до 4 КБ), який сервер відправляє браузеру, а браузер зберігає у себе на жорсткому диску і автоматично додає до *кожного* наступного запиту до цього ж сервера.

Створення Cookie: Оскільки Cookies передаються через HTTP-заголовки (Headers), функція setcookie() обов'язково має викликатися до будь-якого виводу HTML-коду або навіть пробілу у PHP-скрипті.

PHP

```
// Синтаксис: setcookie(назва, значення, час_життя, шлях);
```

$expireTime = time() + (86400 * 30); // Поточний час + 30 днів (у секундах)

setcookie(«user_lang», «uk», $expireTime, «/»);

Читання Cookie: Після того як браузер зберіг Cookie, при наступному оновленні сторінки дані стануть доступними у суперглобальному масиві $_COOKIE.

PHP

if (isset($_COOKIE['user_lang'])) {

  echo «Ваша обрана мова: « . $_COOKIE['user_lang'];

}

Видалення Cookie: У PHP немає спеціальної функції для видалення. Щоб видалити Cookie, потрібно перезаписати його з тією ж назвою, але встановити час життя у минуле.

PHP

setcookie(«user_lang», «», time() - 3600, «/»); // Час життя вийшов годину тому

3. Механізм сесій (Sessions): архітектура та масив $_SESSION

Cookies зберігаються у клієнта, тому їх можна легко підробити, вкрасти або видалити. Для збереження критичних даних (наприклад, ID авторизованого користувача, його роль чи баланс) використовують Сесії.

Як це працює «під капотом»:

1. Коли сесія стартує, сервер створює у себе на жорсткому диску (або в базі даних/пам'яті) унікальний файл.

2. Сервер генерує випадковий, довгий ідентифікатор сесії (Session ID).

3. Сервер відправляє цей Session ID браузеру у вигляді тимчасової Cookie (зазвичай під назвою PHPSESSID).

4. При наступних запитах браузер надсилає PHPSESSID, сервер знаходить відповідний файл у себе і відновлює масив $_SESSION.

Тобто: сесія зберігає дані на сервері, але використовує клієнтську Cookie лише як «ключик» від комірки з цими даними.

4. Життєвий цикл сесії: старт, збереження даних та знищення

Старт сесії: Як і setcookie, функція session_start() відправляє HTTP-заголовки, тому має бути першим рядком вашого скрипта (або принаймні викликатися до будь-якого виводу HTML). Вона або створює нову сесію, або відновлює існуючу.

PHP

<?php

session_start(); // Обов'язковий старт!

```
// Запис даних у сесію
$_SESSION['user_id'] = 42;
$_SESSION['role'] = 'admin';
?>
```

Повне знищення сесії (наприклад, при натисканні кнопки «Вийти»): Просто очистити масив недостатньо для повної безпеки. Потрібно знищити саму сесію та бажано видалити Cookie з ідентифікатором.

```
PHP
session_start();
// 1. Очищуємо всі змінні сесії
$_SESSION = array();
// 2. Знищуємо сесійну Cookie на стороні клієнта
if (ini_get(«session.use_cookies»)) {
   $params = session_get_cookie_params();
   setcookie(session_name(), '', time() - 42000,
      $params[«path»], $params[«domain»],
      $params[«secure»], $params[«httponly»]
   );
}
// 3. Знищуємо файл сесії на сервері
session_destroy();
```

5. Порівняння підходів та кібербезпека

Cookies: Налаштування інтерфейсу (темна/світла тема), мова сайту, аналітика, некритичні дані кошика (для неавторизованих користувачів), токени «Запам'ятати мене».

Sessions: Статус авторизації (is_logged_in), права доступу, конфіденційні дані, поточний процес оформлення замовлення.

Кібербезпека (Захист сесій):

1. Прапорець HttpOnly: Якщо при створенні Cookie (або конфігурації сесії в php.ini) вказати цей прапорець, доступ до Cookie через JavaScript (document.cookie) буде заблоковано. Це головний захист від крадіжки сесії через XSS-атаки.

2. Прапорець Secure: Дозволяє передавати Cookie виключно через зашифроване з'єднання HTTPS. Захищає від перехоплення трафіку (Man-in-the-Middle).

3. Запобігання фіксації сесії (Session Fixation): Коли користувач успішно вводить логін та пароль (переходить зі статусу «Гість» у статус «Авторизований»), критично важливо змінити його Session ID, зберігши при

цьому самі дані.

```
PHP
if ($loginSuccessful) {
   session_regenerate_id(true); // Генерує новий ID та видаляє старий файл
   $_SESSION['logged_in'] = true;
}
```

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Функціонал «Запам'ятати мене»): *«Дій як Senior Backend Developer. Поясни, чому не можна зберігати логін і пароль користувача у звичайних Cookies для реалізації функції 'Remember Me'. Напиши безпечний приклад на PHP, який використовує довготривалі Cookies з криптографічними токенами для автоматичного відновлення сесії.»*

Приклад промпту №2 (Дебагінг помилки Headers Already Sent): *«Я постійно отримую помилку 'Warning: Cannot modify header information - headers already sent' при спробі використати session_start() або setcookie(). Поясни технічну причину цієї помилки. Що означає 'буферизація виводу' (output buffering) і як функція ob_start() може тимчасово вирішити цю проблему?»*

Питання для обговорення та контролю знань:

1. Поясніть концепцію «Stateless» у контексті HTTP. Обґрунтуйте, чому без спеціальних інструментів неможливо створити сторінку профілю користувача.
2. Опишіть принципову різницю між Cookies та Sessions з точки зору місця фізичного зберігання даних та рівня безпеки.
3. Поясніть, чому виклик функцій setcookie() або session_start() повинен відбуватися до виводу будь-яких HTML-тегів або тексту на сторінку.
4. Визначте, яку роль виконує Cookie з назвою PHPSESSID. Опишіть, що станеться з сесією на сервері, якщо користувач вручну видалить цю Cookie у своєму браузері.
5. Поясніть, як використання функції session_regenerate_id(true) під час авторизації допомагає захистити користувача від атак типу Session Fixation (фіксація сесії).

ТЕМА 4.3. ОСНОВИ РЕЛЯЦІЙНИХ БАЗ ДАНИХ ТА МОВА SQL

Мета: засвоїти концепцію реляційних баз даних (РБД) та принципи зберігання інформації; зрозуміти архітектуру таблиць, записів та полів; опанувати механізми зв'язків між даними за допомогою первинних і зовнішніх

ключів; вивчити основи мови структурованих запитів (SQL) для виконання базових CRUD-операцій (створення, читання, оновлення, видалення даних).

План лекції:

1. Концепція СУБД та архітектура реляційних баз даних.
2. Проєктування зв'язків: первинні (Primary) та зовнішні (Foreign) ключі.
3. Основи мови SQL: створення структури (DDL – Data Definition Language).
4. Маніпуляція даними (DML – Data Manipulation Language): INSERT, UPDATE, DELETE.
5. Вибірка даних (DQL – Data Query Language): фільтрація SELECT та основи JOIN.

1. Концепція СУБД та архітектура реляційних баз даних

Для збереження даних вебзастосунків (користувачі, товари, замовлення) звичайні текстові файли не підходять через низьку швидкість пошуку, відсутність конкурентного доступу та безпеки. Для цього використовують Системи Управління Базами Даних (СУБД). Найпопулярнішими у зв'язці з PHP є MySQL та PostgreSQL.

Реляційна база даних (Relational Database) базується на реляційній моделі, де вся інформація зберігається у вигляді двовимірних масивів – таблиць.

Таблиця (Table / Сутність): Логічне угруповання даних однієї тематики (наприклад, таблиця users або products).

Рядок (Row / Запис / Кортеж): Один конкретний об'єкт у таблиці (наприклад, конкретний користувач «Іван»).

Стовпець (Column / Поле / Атрибут): Властивість об'єкта (наприклад, email, password, created_at). Кожен стовпець має строго визначений тип даних (ціле число, текст, дата).

2. Проєктування зв'язків: первинні та зовнішні ключі

Головна сила реляційних баз даних полягає в тому, що дані в різних таблицях можуть бути пов'язані між собою. Для забезпечення цілісності цих зв'язків існують ключі.

Первинний ключ (Primary Key - PK): Унікальний ідентифікатор кожного запису в таблиці. Жодні два рядки не можуть мати однаковий PK. Він не може бути порожнім (NULL). Зазвичай це поле id з автоматичним збільшенням (AUTO_INCREMENT).

Зовнішній ключ (Foreign Key - FK): Стовпець в одній таблиці, який посилається на первинний ключ іншої таблиці. Він створює фізичний зв'язок.

Типи зв'язків:

1. Один-до-багатьох (1:N): Найпоширеніший тип. Один користувач може мати багато замовлень, але одне замовлення належить лише одному користувачу. Таблиця orders матиме зовнішній ключ user_id.

2. Багато-до-багатьох (M:N): Один студент слухає багато курсів, а один курс слухає багато студентів. Для реалізації такого зв'язку завжди створюється третя, проміжна (звідна) таблиця (наприклад, student_courses).

3. Один-до-одного (1:1): Використовується рідше, зазвичай для розділення великих таблиць (наприклад, users та user_profiles).

3. Основи мови SQL: створення структури (DDL)

SQL (Structured Query Language) – це стандартизована мова для взаємодії з реляційними базами даних. Вона не є мовою програмування у класичному розумінні (не має циклів чи масивів), це мова декларативних запитів.

Підмножина DDL відповідає за структуру таблиць.

SQL

```
-- Створення нової таблиці користувачів
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- INT, VARCHAR – типи даних (число, рядок змінної довжини).
- NOT NULL – поле обов'язкове для заповнення.
- UNIQUE – значення в цьому стовпці не можуть повторюватися у різних записах.

4. Маніпуляція даними (DML): INSERT, UPDATE, DELETE

Ця підмножина SQL реалізує три з чотирьох операцій CRUD (Create, Read, Update, Delete), що модифікують самі записи.

Створення запису (Create):

SQL

```
INSERT INTO users (username, email)
VALUES ('Олександр', 'alex@example.com');
```

Оновлення запису (Update): *Критично важливо:* Завжди використовуйте умову WHERE. Якщо її забути, оновляться всі записи в таблиці!

SQL

```
UPDATE users
```

SET username = 'Олександр Новий'

WHERE id = 1;

Видалення запису (Delete):

SQL

DELETE FROM users

WHERE id = 1;

(Примітка: У сучасних системах часто використовують «м'яке видалення» (Soft Delete) – замість DELETE запису просто оновлюють статус is_deleted = 1 за допомогою UPDATE, щоб зберегти історію).

5. Вибірка даних (DQL): фільтрація SELECT та основи JOIN

Підмножина DQL відповідає за отримання даних (операція Read у CRUD).

Базовий запит:

SQL

-- Вибрати всі поля (*) з таблиці users

SELECT * FROM users;

-- Вибрати конкретні поля з фільтрацією та сортуванням

SELECT username, email

FROM users

WHERE created_at >= '2023-01-01'

ORDER BY username ASC

LIMIT 10;

-WHERE – фільтрує рядки за умовою.

-ORDER BY ... ASC/DESC – сортує результат за зростанням або спаданням.

-LIMIT – обмежує кількість повернутих записів (необхідно для пагінації).

Об'єднання таблиць (JOIN): Оскільки бази даних нормалізовані (дані розділені по різних таблицях), нам часто потрібно збирати їх разом в одному запиті.

-INNER JOIN: Повертає лише ті записи, які мають збіги в обох таблицях.

SQL

-- Отримати ім'я користувача та суму його замовлення

SELECT users.username, orders.total_amount

FROM users

INNER JOIN orders ON users.id = orders.user_id;

-LEFT JOIN: Повертає всі записи з лівої (першої) таблиці, навіть якщо для них немає відповідників у правій таблиці (замість значень правої таблиці буде підставлено NULL).

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Проєктування БД): *«Дій як Database Architect. Я хочу створити базу даних для інтернет-магазину. Мені потрібні таблиці для Користувачів, Товарів, Категорій (один товар – одна категорія) та Замовлень. Напиши SQL-код (DDL) для створення цих таблиць у MySQL, включаючи правильне налаштування Primary та Foreign ключів.»*

Приклад промпту №2 (Оптимізація запитів): *«Поясни мені різницю між INNER JOIN та LEFT JOIN у SQL простими словами. Наведи приклад з таблицями 'Студенти' та 'Іспити', де використання LEFT JOIN покаже студентів, які ще не склали жодного іспиту. Напиши відповідний SQL-запит.»*

Питання для обговорення та контролю знань:

1. Поясніть, що таке «реляційна» база даних і чим вона відрізняється від збереження даних у звичайному текстовому файлі або таблиці Excel.
2. Опишіть, у чому полягає критична важливість Первинного ключа (Primary Key) для кожної таблиці бази даних.
3. Опишіть концепцію зв'язку «багато-до-багатьох» (M:N). Поясніть, чому такий зв'язок неможливо реалізувати напряму за допомогою лише двох таблиць і зовнішнього ключа.
4. Поясніть, що станеться, якщо виконати SQL-запит UPDATE products SET price = 0; без використання оператора WHERE.
5. Поясніть, для чого в SQL-запитах використовується конструкція LIMIT і як вона допомагає при розробці вебзастосунків із великою кількістю даних.

ТЕМА 4.4. ПІДКЛЮЧЕННЯ PHP ДО БАЗИ ДАНИХ (PDO / MYSQLI)

Мета: навчитися встановлювати безпечне з'єднання між PHP-застосунком та сервером баз даних; зрозуміти принципову різницю між розширеннями MySQLi та PDO; опанувати механізм підготовлених запитів (Prepared Statements) для абсолютного захисту від SQL-ін'єкцій; засвоїти патерни виконання CRUD-операцій та обробки помилок бази даних через винятки (Exceptions).

План лекції:

1. Еволюція підключень у PHP: від mysql_* до MySQLi та PDO.
2. Встановлення з'єднання через PDO та обробка винятків (try-catch).
3. Загроза SQL-ін'єкцій та захист: Підготовлені запити (Prepared Statements).

4. Виконання CRUD-операцій: вибірка, додавання, оновлення та видалення.

5. Архітектурний підхід: винесення конфігурації та безпека облікових даних.

1. Еволюція підключень у PHP: від mysql_* до MySQLi та PDO

Історично в PHP використовувалося розширення mysql_ (наприклад, функція mysql_connect). Сьогодні воно повністю видалене з мови через критичні вразливості безпеки. Сучасний PHP пропонує два актуальні підходи для роботи з базами даних:

1) MySQLi (MySQL Improved):

a) Працює виключно з базами даних MySQL та MariaDB.

b) Підтримує як процедурний, так і об'єктно-орієнтований стиль написання коду.

c) Трохи швидший за PDO, але менш гнучкий.

2) PDO (PHP Data Objects):

a) Є універсальним інтерфейсом. Підтримує понад 12 різних систем керування базами даних (MySQL, PostgreSQL, SQLite, Oracle тощо). Якщо в майбутньому проєкт переїде з MySQL на PostgreSQL, код підключення доведеться змінити мінімально.

b) Працює тільки в об'єктно-орієнтованому стилі.

c) Є беззаперечним індустріальним стандартом сучасної веброзробки. Саме на ньому ми зосередимо увагу.

2. Встановлення з'єднання через PDO та обробка винятків

Для підключення через PDO необхідно створити новий об'єкт класу PDO. Конструктор приймає три основні параметри: DSN, логін та пароль.

DSN (Data Source Name) – це рядок, що містить інформацію про тип бази даних, хост, порт і назву самої БД.

Оскільки підключення до стороннього сервера (яким є БД) – це операція, що може завершитися невдачею (сервер впав, неправильний пароль), цей процес обов'язково загортають у блок відловлювання винятків try...catch.

PHP

```
$host = '127.0.0.1';
$db   = 'my_shop';
$user = 'root';
$pass = '';
$charset = 'utf8mb4';
// Формуємо рядок DSN
```

```
$dsn = «mysql:host=$host;dbname=$db;charset=$charset»;
// Опції для налаштування поведінки PDO
$options = [
  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // Викидати винятки при помилках SQL
  PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC, // Повертати дані як асоціативний масив
  PDO::ATTR_EMULATE_PREPARES => false, // Використовувати справжні підготовлені запити
];
try {
  // Створення екземпляра підключення
  $pdo = new PDO($dsn, $user, $pass, $options);
  // echo «Підключення успішне!»;
} catch (\PDOException $e) {
  // Якщо сталася помилка, скрипт зупиняється і виводить повідомлення (у реальному проєкті логується)
  throw new \PDOException($e->getMessage(), (int)$e->getCode());
}
```

3. Загроза SQL-ін'єкцій та захист: Підготовлені запити

SQL-ін'єкція – це найпопулярніший і найнебезпечніший метод злому баз даних. Він виникає, коли дані від користувача (наприклад, з $_POST) напряму вставляються (конкатенуються) в рядок SQL-запиту.

Вразливий код (НІКОЛИ ТАК НЕ РОБІТЬ!):

PHP

```
$email = $_POST['email']; // Зловмисник ввів: ' OR '1'='1
$sql = «SELECT * FROM users WHERE email = '$email'«;
// Сервер виконає: SELECT * FROM users WHERE email = '' OR '1'='1' (і поверне всіх користувачів!)
$pdo->query($sql);
```

Рішення: Підготовлені запити (Prepared Statements): Процес розбивається на два етапи:

1. Підготовка (prepare): Ми відправляємо в БД шаблон запиту з «маркерами» (плейсхолдерами ? або :name) замість реальних даних. БД компілює цей шаблон.

2. Виконання (execute): Ми передаємо масив із даними. БД вставляє ці дані в уже скомпільований шаблон виключно як текстові значення, а не як частину виконуваного коду.

PHP

$email = $_POST['email'];

// 1. Підготовка (використовуємо іменований маркер :email)

$stmt = $pdo->prepare(«SELECT * FROM users WHERE email = :email»);

// 2. Виконання (передаємо дані)

$stmt->execute(['email' => $email]);

4. Виконання CRUD-операцій: вибірка, додавання, оновлення та видалення

Маючи об'єкт $stmt (Statement), ми можемо отримувати результати або перевіряти статус операції.

Вибірка даних (Read):

-$stmt->fetch() – повертає лише один (наступний) рядок у вигляді масиву. Ідеально для пошуку за ID.

-$stmt->fetchAll() – повертає всі знайдені рядки у вигляді двовимірного масиву. Використовується для списків товарів/статей.

PHP

// Отримання одного користувача

$stmt = $pdo->prepare(«SELECT id, username FROM users WHERE id = ?»);

$stmt->execute([1]);

$user = $stmt->fetch();

echo $user['username'];

Додавання даних (Create): Після виконання INSERT, часто потрібно дізнатися id щойно створеного запису (наприклад, щоб прив'язати до нього фотографію). Для цього використовується $pdo->lastInsertId().

PHP

$stmt = $pdo->prepare(«INSERT INTO users (username, email) VALUES (:name, :email)»);

$stmt->execute(['name' => 'Олег', 'email' => 'oleg@test.com']);

echo «Створено користувача з ID: « . $pdo->lastInsertId();

Оновлення та Видалення (Update & Delete): Для цих операцій важливо знати, скільки рядків було фактично змінено/видалено. Для цього використовується метод $stmt->rowCount().

PHP

$stmt = $pdo->prepare(«DELETE FROM users WHERE id = ?»);

$stmt->execute([5]);

echo «Видалено рядків: « . $stmt->rowCount();

5. Архітектурний підхід: винесення конфігурації

Якщо у вашому проєкті 20 сторінок, кожна з яких звертається до бази даних, писати код підключення (new PDO...) у кожному файлі – жахлива практика.

Правильний підхід:

1. Створюється окремий файл (наприклад, db.php або config.php), у якому відбувається підключення і створюється об'єкт $pdo.

2. Цей файл підключається на потрібних сторінках за допомогою require_once 'db.php';.

3. *Кібербезпека:* У реальних проєктах логіни та паролі від бази даних ніколи не зберігають у самому PHP-коді. Їх виносять у спеціальні файли змінних середовища .env, які додаються у .gitignore, щоб пароль не потрапив у публічний репозиторій на GitHub.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (SQL Injections): *«Дій як Senior Security Engineer. Поясни мені концепцію 'Сліпої SQL-ін'єкції' (Blind SQL Injection). Наведи приклад на PHP, де використання конкатенації рядків у запиті створює вразливість, і покажи, як метод $pdo->prepare() повністю усуває цю загрозу.»*

Приклад промпту №2 (ООП Обгортка для БД): *«Напиши для мене простий PHP-клас Database, який використовує патерн Singleton для підключення до БД через PDO. Клас повинен мати один публічний метод query($sql, $params), який автоматично робить prepare та execute. Поясни, чому патерн Singleton корисний для з'єднань з БД.»*

Питання для обговорення та контролю знань:

1. Поясніть, які дві головні переваги використання розширення PDO порівняно з MySQLi у сучасній PHP-розробці.

2. Визначте, що таке DSN (Data Source Name) і яку інформацію він у собі містить.

3. Опишіть механізм виникнення вразливості SQL-ін'єкції. Поясніть, чому підготовлені запити (Prepared Statements) роблять цю атаку технічно неможливою.

4. Поясніть різницю між методами fetch() та fetchAll() об'єкта PDOStatement. Визначте, коли який з них доречно використовувати.

5. Опишіть, як дізнатися id запису (при налаштуванні AUTO_INCREMENT), який був щойно доданий у таблицю за допомогою запиту INSERT через PDO.

ЗМІСТОВИЙ МОДУЛЬ 5. СУЧАСНІ ПІДХОДИ ДО ВЕБДИЗАЙНУ, ОПТИМІЗАЦІЇ ТА ВЕБРОЗРОБКИ

ТЕМА 5.1. ОСНОВИ UX/UI ДИЗАЙНУ ТА ПРИНЦИПИ СТВОРЕННЯ ЗРУЧНИХ ІНТЕРФЕЙСІВ

Мета: зрозуміти принципову різницю між UX (користувацьким досвідом) та UI (користувацьким інтерфейсом); засвоїти базові закони візуальної ієрархії та композиції; вивчити ключові евристики юзабіліті Якоба Нільсена; опанувати процес проєктування інтерфейсів від низькодеталізованих вайрфреймів до інтерактивних прототипів (на базі Figma).

План лекції:

1. Розмежування понять: UX (User Experience) проти UI (User Interface).
2. Дизайн, орієнтований на користувача (UCD), та дослідження аудиторії.
3. Фундаментальні принципи візуального дизайну: ієрархія, контраст та «повітря» (Whitespace).
4. Закони юзабіліті: ключові евристики Якоба Нільсена.
5. Етапи проєктування: Wireframe, Mockup, Prototype та інструментарій (Figma).

1. Розмежування понять: UX (User Experience) проти UI (User Interface)

Хоча ці терміни часто пишуть через скісну риску, вони позначають абсолютно різні, хоч і взаємопов'язані, дисципліни.

-UX (User Experience – Користувацький досвід): Це «невидима» частина роботи, аналітика та логіка. UX визначає, *як* продукт працює. Він фокусується на вирішенні проблем користувача: наскільки легко знайти потрібний товар, скільки кроків займає реєстрація, чи логічна структура меню. Головна мета UX – зробити шлях користувача максимально безшовним та ефективним.

-UI (User Interface – Користувацький інтерфейс): Це візуальна частина, естетика. UI визначає, *як* продукт виглядає. Сюди входять кольорові палітри, типографіка (шрифти), форми кнопок, анімації, іконки та відступи. Головна мета UI – зробити продукт візуально привабливим, викликати довіру та емоційний відгук, а також допомагати користувачеві орієнтуватися (наприклад, червона кнопка попереджає про небезпеку).

Аналогія: Якщо уявити сайт як будинок, то UX – це фундамент, несучі стіни та планування кімнат (щоб з кухні не треба було йти через спальню у ванну). UI – це шпалери, меблі, освітлення та декор.

2. Дизайн, орієнтований на користувача (UCD), та дослідження

аудиторії

User-Centered Design (UCD) – це філософія розробки, де в центрі кожного рішення стоять потреби, обмеження та поведінка кінцевого користувача, а не вподобання розробника чи замовника.

Ключові інструменти дослідження:

- Персони (User Personas): Узагальнені портрети типових представників цільової аудиторії (ім'я, вік, професія, цілі, страхи, технічна грамотність). Допомагають команді розробників фокусуватися на реальних людях.
- Шлях користувача (User Journey Map): Візуалізація всіх кроків, які проходить клієнт для досягнення своєї мети (від першого заходу на сайт до успішної покупки та отримання листа на пошту). Допомагає знайти «вузькі місця» (pain points), де користувачі найчастіше покидають сайт.

3. Фундаментальні принципи візуального дизайну: ієрархія та контраст

Щоб користувач міг швидко сканувати сторінку, дизайнер використовує принципи композиції, засновані на психології сприйняття (Гештальт-принципи).

- Візуальна ієрархія: Око людини читає екран не так, як книгу. Зазвичай використовується F-патерн (для сторінок з великою кількістю тексту, як блоги) або Z-патерн (для лендингів, де погляд ковзає від логотипа до меню, а потім по діагоналі до головної кнопки). Найважливіші елементи (заголовки, кнопки заклику до дії – CTA) роблять найбільшими та найяскравішими.
- Прогалини / «Повітря» (Whitespace / Negative Space): Порожній простір між елементами. Це не «втрачене місце», а найпотужніший інструмент дизайну. Відступи допомагають групувати пов'язані елементи (Закон близькості) та дають очам «відпочити», роблячи інтерфейс легким і зрозумілим.
- Контраст та Типографіка: Текст має бути читабельним (високий контраст між фоном і текстом). Рекомендується використовувати не більше двох-трьох шрифтів на весь проєкт (один для заголовків, інший для основного тексту).

4. Закони юзабіліті: ключові евристики Якоба Нільсена

Якоб Нільсен ще у 1994 році сформулював 10 загальних принципів (евристик) оцінки інтерфейсів, які актуальні й досі. Ось найважливіші з них:

1. Видимість стану системи (Visibility of system status): Користувач завжди повинен знати, що відбувається. (Наприклад, індикатор завантаження під час оплати, підсвічування активного пункту меню).
2. Зв'язок між системою та реальним світом: Розмовляйте мовою

користувача, уникайте специфічних технічних термінів. Іконки мають бути інтуїтивними (кошик для покупок, лупа для пошуку).

3. Свобода та контроль (User control and freedom): Люди часто роблять помилки. Обов'язково має бути функція скасування дії («Undo», кнопка «Назад», легке видалення товару з кошика).

4. Одноманітність та стандарти (Consistency and standards): Кнопки одного типу мають виглядати однаково на всіх сторінках. Не змушуйте користувача вгадувати, чи означають різні слова або дії одне й те саме.

5. Запобігання помилкам (Error prevention): Краще попередити помилку, ніж показувати повідомлення про неї. (Наприклад, блокування кнопки «Відправити», поки не заповнені всі обов'язкові поля форми).

5. Етапи проєктування: Wireframe, Mockup, Prototype

Створення дизайну – це ітеративний процес. Ніхто не починає малювати готові кнопки з тінями в перший же день.

1. Wireframe (Вайрфрейм / Каркас): Це «скелет» сторінки. Низькодеталізована чорно-біла схема, яка показує лише розташування блоків, тексту та зображень. Створюється для швидкого узгодження логіки з замовником. (Часто малюється просто на папері).

2. Mockup (Мокап / Візуальний макет): Це статичне зображення майбутнього сайту у високій деталізації. Тут вже є реальні кольори, шрифти, фотографії та іконки. Це те, як сайт виглядатиме у фіналі.

3. Prototype (Прототип): Це інтерактивна модель. У прототипі можна клікати на кнопки, відкривати меню, переходити між сторінками. Він імітує роботу справжнього сайту, хоча під ним ще немає жодного рядка коду.

Figma – сучасний індустріальний стандарт для UI/UX дизайну. Це хмарний інструмент, який дозволяє дизайнерам працювати в реальному часі разом, створювати дизайн-системи та передавати готові макети розробникам (з автоматично згенерованим CSS-кодом для елементів).

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (UX Дослідження): *«Дій як досвідчений UX-дослідник. Згенеруй детальну User Persona (Персону користувача) для нового мобільного застосунку з доставки здорового харчування у великому місті. Опиши її демографію, ключові цілі, фрустрації (страхи) та технічний рівень.»*

Приклад промпту №2 (UI Аналіз та Кольори): *«Дій як UI-дизайнер. Порадь кольорову палітру (у форматі HEX-кодів) для сайту ветеринарної клініки. Поясни свій вибір з точки зору психології кольору. Також запропонуй пару Google-шрифтів, які добре поєднуються для заголовків та основного*

тексту в цьому контексті.»

Питання для обговорення та контролю знань:

1. Поясніть різницю між UX-дизайном та UI-дизайном. Наведіть приклад завдання для кожного з цих напрямків.
2. Визначте, що таке «Negative space» (або «повітря») у вебдизайні, та опишіть його функцію.
3. Опишіть суть евристики Нільсена «Видимість стану системи». Поясніть, чому ігнорування цього правила найбільше дратує користувачів.
4. Поясніть, у яких випадках при проєктуванні інтерфейсу використовується F-патерн, а в яких – Z-патерн читання.
5. Опишіть, чим Wireframe (каркас) відрізняється від Prototype (прототипу). Поясніть, чому етап створення чорно-білих вайрфреймів не варто пропускати.

ТЕМА 5.2. АДАПТИВНИЙ ТА МОБІЛЬНИЙ ДИЗАЙН: СУЧАСНІ ТЕХНІКИ ТА ТРЕНДИ

Мета: зрозуміти фундаментальну різницю між адаптивним та чуйним (responsive) вебдизайном; засвоїти філософію «Mobile First»; опанувати технічний інструментарій CSS (Media Queries, Flexbox, Grid) для побудови гнучких макетів; вивчити методи оптимізації медіаконтенту та типографіки; ознайомитися з трендами мобільної ергономіки та темними темами (Dark Mode).

План лекції:

1. Еволюція підходів: Responsive vs. Adaptive та філософія Mobile First.
2. Технічний фундамент: метатег Viewport та CSS Media Queries.
3. Сучасні системи верстки: макро- та мікро-лейаути (CSS Grid та Flexbox).
4. «Гумова» типографіка та оптимізація медіа (srcset, picture, clamp).
5. Сучасні тренди мобільних інтерфейсів: Thumb Zone та Dark Mode.

1. Еволюція підходів: Responsive vs. Adaptive та філософія Mobile First

Історично сайти створювалися лише для десктопних моніторів. З появою смартфонів виникла потреба оптимізувати контент під маленькі екрани. З'явилися два основні підходи.

-Адаптивний дизайн (Adaptive Web Design - AWD): Створення кількох *фіксованих* макетів під конкретні розміри екранів (наприклад, 320px, 768px,

1024px). Сервер або браузер визначає пристрій і завантажує відповідний макет. Сьогодні використовується рідко через складність підтримки.

-Чуйний дизайн (Responsive Web Design - RWD): Створення *одного* гнучкого макету, який плавно змінює свої пропорції та структуру залежно від ширини екрана. Це сучасний індустріальний стандарт.

Філософія Mobile First (Мобільні пристрої насамперед): Оскільки понад 50% світового інтернет-трафіку генерують смартфони, проєктування та написання коду починають саме з мобільної версії.

Чому це важливо? Набагато легше спочатку розмістити найважливіший контент на вузькому екрані, а потім додавати декоративні елементи та додаткові колонки для десктопу (Progressive Enhancement), ніж намагатися «втиснути» складний десктопний сайт у мобільний формат (Graceful Degradation).

2. Технічний фундамент: метатег Viewport та CSS Media Queries

Щоб чуйний дизайн запрацював, мобільному браузеру потрібно «пояснити», що сайт вміє адаптуватися. Без цього браузер спробує відобразити десктопну версію у зменшеному масштабі (дрібний нечитабельний текст).

Метатег Viewport (додається в <head>):

HTML

<meta name=«viewport» content=«width=device-width, initial-scale=1.0»>

-width=device-width – наказує браузеру встановити ширину сторінки рівною фізичній ширині екрана пристрою.

-initial-scale=1.0 – встановлює початковий масштаб 100%.

Медіа-запити (CSS Media Queries): Це інструмент CSS, що дозволяє застосовувати стилі лише за виконання певних умов (контрольні точки або «breakpoints»). Згідно з підходом Mobile First, базові стилі пишуться для мобільних, а медіа-запити використовують min-width для більших екранів.

CSS

```
/* Базові стилі для смартфонів (Mobile First) */
.container { width: 100%; padding: 10px; }
/* Стилі для планшетів (і ширше) */
@media (min-width: 768px) {
   .container { width: 750px; padding: 20px; }
}
/* Стилі для десктопів (і ширше) */
@media (min-width: 1024px) {
   .container { width: 960px; }
}
```

3. Сучасні системи верстки: макро- та мікро-лейаути

У минулому для верстки використовували таблиці або «плаваючі» блоки (float), що було незручно. Сьогодні стандарт – це Flexbox та Grid.

CSS Flexbox (Гнучкі блоки): Одновимірна система. Використовується для вирівнювання елементів в один ряд (або стовпець) та розподілу вільного простору між ними. Ідеально підходить для мікро-компонентів: навігаційного меню, карток товарів, кнопок.

CSS Grid (Сітки): Двовимірна система. Дозволяє створювати складні макети, керуючи одночасно і рядками, і стовпцями. Ідеально підходить для макро-лейауту (загальної структури сторінки: шапка, бічна панель, основний контент, підвал). За допомогою функції minmax() та одиниці виміру fr (фракція) можна створювати сітки, які адаптуються взагалі без Media Queries.

4. «Гумова» типографіка та оптимізація медіа

Адаптивні зображення: Зображення не повинні виходити за межі свого контейнера на вузьких екранах.

CSS

```
img {
   max-width: 100%;
   height: auto;
}
```

Тег <picture> та атрибут srcset: Щоб не завантажувати величезне десктопне зображення на смартфон (економія мобільного трафіку), використовують srcset. Він дозволяє браузеру обрати найменший підходящий файл.

HTML

```
<picture>
  <source media=«(min-width: 800px)» srcset=«large-banner.jpg»>
  <img src=«small-banner.jpg» alt=«Банер»>
</picture>
```

«Гумова» (Fluid) типографіка: Замість стрибкоподібної зміни розміру шрифту через медіа-запити, сучасний CSS використовує функцію clamp(). Вона плавно масштабує шрифт між мінімальним та максимальним значенням залежно від ширини екрана (vw).

CSS

```
h1 {
   /* Мінімум 1.5rem, ідеальний розмір залежить від ширини вікна, максимум 3rem */
```

```
  font-size: clamp(1.5rem, 5vw, 3rem);
}
```

5. Сучасні тренди мобільних інтерфейсів: Thumb Zone та Dark Mode

Адаптивний дизайн – це не лише про те, щоб елементи «влізли» в екран, а й про те, щоб ними було зручно користуватися.

Ергономіка та Зона великого пальця (Thumb Zone): Оскільки екрани смартфонів стають дедалі більшими, тягнутися до «гамбургер-меню» у лівому верхньому куті стає фізично незручно. Сучасний тренд – перенесення головної навігації (Bottom Navigation Bar) та ключових кнопок дій (CTA) у нижню частину екрана, куди легко дістає великий палець.

Розмір тач-таргетів (Touch Targets): Кнопки та посилання для пальців мають бути значно більшими, ніж для курсора миші. Мінімальний рекомендований розмір інтерактивної зони – 44x44 пікселі (Apple) або 48x48 пікселів (Google Material Design).

Темна тема (Dark Mode): Користувачі масово переходять на темні теми для економії заряду батареї (на OLED-екранах) та зменшення навантаження на очі. CSS дозволяє автоматично адаптувати сайт під системні налаштування пристрою:

```
CSS
@media (prefers-color-scheme: dark) {
  body { background-color: #121212; color: #ffffff; }
}
```

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Генерація CSS Grid): *«Дій як Senior Frontend Developer. Напиши HTML та CSS код для створення адаптивної галереї зображень за допомогою CSS Grid. Галерея повинна мати 1 колонку на мобільних, 2 на планшетах і 4 на десктопах. Використай функцію repeat(auto-fit, minmax(...)) щоб зробити це без використання @media запитів.»*

Приклад промпту №2 (Оптимізація зображень): *«Поясни мені концепцію атрибута srcset та sizes у тегу <img> простими словами. Як браузер вирішує, яке зображення завантажити (наприклад, image-320w.jpg чи image-800w.jpg)? Наведи практичний приклад коду.»*

Питання для обговорення та контролю знань:

1. Поясніть фундаментальну різницю між Adaptive Web Design (фіксовані макети) та Responsive Web Design (чуйні макети).

2. Поясніть призначення метатегу <meta name=«viewport» ...>. Опишіть,

що побачить користувач на смартфоні, якщо цей тег видалити з коду.

3. Поясніть логіку підходу «Mobile First». Визначте, чому медіа-запити в цьому підході зазвичай використовують умову min-width, а не max-width.

4. Визначте, для яких задач краще підходить CSS Flexbox, а для яких – CSS Grid. Опишіть їхню головну відмінність (вимірність).

5. Поясніть, що таке «Thumb Zone» у мобільному дизайні і як цей концепт впливає на розташування елементів навігації у сучасних вебзастосунках.

ТЕМА 5.3. ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ВЕБ-САЙТІВ ТА ВЕБ-АНАЛІТИКА

Мета: зрозуміти вплив швидкості завантаження сайту на користувацький досвід (UX) та ранжування в пошукових системах; вивчити ключові метрики продуктивності від Google (Core Web Vitals); опанувати техніки оптимізації критичного шляху рендерингу (CRP), кешування та стиснення ресурсів; познайомитися з основами налаштування подій у Google Analytics 4 (GA4) та методологією A/B тестування.

План лекції:

1. Метрики продуктивності: Core Web Vitals (LCP, INP, CLS).
2. Критичний шлях рендерингу (CRP) та оптимізація коду.
3. Оптимізація медіаконтенту: Lazy Loading, WebP/AVIF та CDN.
4. Введення у веб-аналітику: Google Analytics 4 та Google Tag Manager.
5. Оптимізація конверсії (CRO) та A/B тестування.

1. Метрики продуктивності: Core Web Vitals

Швидкість сайту – це не лише про те, коли з'явиться перший піксель, а й про те, наскільки плавно працює інтерфейс. Google використовує набір стандартизованих метрик Core Web Vitals як один із факторів ранжування сайтів у пошуку.

1) LCP (Largest Contentful Paint): Вимірює швидкість завантаження *основного* контенту (найбільшої картинки або текстового блоку у видимості екрана).

а) *Хороший показник:* до 2.5 секунд.

2) INP (Interaction to Next Paint): Замінив застарілу метрику FID. Вимірює затримку реакції інтерфейсу на дії користувача (клік по кнопці, відкриття меню). Показує, наскільки швидко сайт реагує після взаємодії.

а) *Хороший показник:* до 200 мілісекунд.

3) CLS (Cumulative Layout Shift): Вимірює візуальну стабільність. Фіксує

несподівані зсуви контенту (наприклад, коли ви хочете натиснути на посилання, але зверху раптово завантажується рекламний банер, і ви клікаєте не туди).

а) *Хороший показник:* менше ніж 0.1.

Інструменти для перевірки: Google PageSpeed Insights, Lighthouse у Chrome DevTools.

2. Критичний шлях рендерингу (CRP) та оптимізація коду

Critical Rendering Path (CRP) – це послідовність кроків, які робить браузер, щоб перетворити HTML, CSS та JavaScript на видиме зображення на екрані.

Щоб сайт завантажувався миттєво, цей шлях треба оптимізувати:

- Мініфікація (Minification): Видалення всіх зайвих пробілів, коментарів та переносів рядків із файлів .css, .js та .html. Це зменшує розмір файлів на 20-30%. (Файли часто мають суфікс .min.js).

- Стиснення на сервері (Gzip / Brotli): Сервер стискає текстові файли перед відправкою, а браузер розпаковує їх на льоту. Це зменшує обсяг переданих даних до 70%.

- Асинхронне завантаження JavaScript: За замовчуванням, коли браузер зустрічає тег <script>, він зупиняє побудову сторінки (рендеринг), завантажує і виконує скрипт (це називається *render-blocking resource*). Використання атрибутів defer або async вирішує цю проблему:

HTML

<script src=«app.js» defer></script>

- Кешування браузера (Browser Caching): Сервер наказує браузеру зберегти логотипи, стилі та скрипти на жорсткому диску користувача. При повторному візиті ці файли беруться з пам'яті комп'ютера миттєво, а не завантажуються з інтернету.

3. Оптимізація медіаконтенту: Lazy Loading, WebP та CDN

Зображення та відео зазвичай становлять понад 60% ваги вебсторінки.

- Сучасні формати: Використання форматів WebP або AVIF замість застарілих JPEG чи PNG. Вони забезпечують таку ж якість зображення при меншій вазі (на 30-50%).

- Ліниве завантаження (Lazy Loading): Техніка, при якій зображення завантажуються не одразу при відкритті сторінки, а лише тоді, коли користувач доскролює до них. У сучасних браузерах це робиться одним атрибутом:

HTML

<img src=«photo.webp» loading=«lazy» alt=«Опис»>

- CDN (Content Delivery Network): Мережа розподілених серверів по всьому світу. Якщо ваш основний сервер знаходиться в Києві, а користувач заходить із Нью-Йорка, картинки будуть завантажуватися з найближчого до нього американського вузла CDN, що кардинально зменшує затримку (Ping).

4. Введення у веб-аналітику: Google Analytics 4 (GA4)

Зробити швидкий сайт – половина справи. Треба розуміти, що на ньому роблять люди.

Google Analytics 4 базується на подієвій моделі (Event-based data model). Будь-яка взаємодія (перегляд сторінки, клік, прокрутка, покупка) – це Подія (Event), яка може мати додаткові параметри (наприклад, подія purchase матиме параметр value – суму покупки).

Google Tag Manager (GTM): Це інструмент-посередник (контейнер). Замість того, щоб програміст постійно додавав нові скрипти (Facebook Pixel, TikTok Tag, Hotjar) напряму в код сайту, програміст один раз встановлює GTM. Далі маркетолог може самостійно додавати будь-які аналітичні скрипти через зручний веб-інтерфейс GTM, не чіпаючи вихідний код проєкту.

5. Оптимізація конверсії (CRO) та A/B тестування

Конверсія (Conversion Rate) – це відсоток відвідувачів, які виконали бажану цільову дію (купили товар, залишили email, зареєструвалися) відносно загальної кількості відвідувачів.

Теплові карти (Heatmaps): Інструменти на кшталт Hotjar або Microsoft Clarity, які показують, куди користувачі клікають найчастіше, як далеко вони скролять сторінку вниз і де водять мишкою. Це допомагає знайти «мертві зони» на сайті.

A/B Тестування (Split Testing): Метод дослідження, при якому трафік сайту ділиться навпіл. Варіант «А» (оригінал) бачить 50% користувачів, а варіант «Б» (наприклад, змінений колір або текст кнопки) – інші 50%. Через тиждень аналітика показує, який варіант приніс більше покупок. Рішення приймаються на основі точних даних, а не інтуїції дизайнера.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (PageSpeed Insights): *«Дій як Tech Lead. Я прогнав свій сайт через Google PageSpeed Insights і отримав помилку: 'Eliminate render-blocking resources'. Поясни мені простими словами, що це означає, які саме файли (CSS/JS) зазвичай викликають цю проблему, і як атрибути async та defer допомагають її вирішити.»*

Приклад промпту №2 (Аналітика подій): *«Як налаштувати відстеження*

події 'Додавання в кошик' (add_to_cart) для електронної комерції в Google Analytics 4? Які параметри події (наприклад, currency, value, items) є обов'язковими для передачі згідно з офіційною документацією GA4?»

Питання для обговорення та контролю знань:

1. Поясніть метрику LCP (Largest Contentful Paint) та обґрунтуйте, чому великий банер у верхній частині сторінки безпосередньо впливає на цей показник.

2. Опишіть принцип лінивого завантаження (Lazy Loading) зображень та поясніть, як він економить трафік на мобільних пристроях.

3. Поясніть, чому мініфікація коду та стиснення на сервері (Gzip/Brotli) є критичними кроками перед викачуванням сайту в продакшн.

4. Визначте різницю між Google Analytics та Google Tag Manager. Поясніть, чому їх часто використовують разом.

5. Поясніть принцип A/B тестування. Наведіть гіпотетичний приклад, як за його допомогою можна збільшити кількість реєстрацій на сайті.

ТЕМА 5.4. СУЧАСНІ ІНСТРУМЕНТИ ТА ТЕНДЕНЦІЇ ФРОНТЕНД- ТА БЕКЕНД-РОЗРОБКИ

Мета: осягнути еволюцію веброзробки від простих скриптів до складних інженерних систем; зрозуміти призначення сучасних JavaScript-фреймворків (React, Vue) та концепцію Virtual DOM; засвоїти різницю між монолітною та мікросервісною архітектурою; порівняти підходи REST API та GraphQL; ознайомитися з технологіями контейнеризації (Docker) та впливом штучного інтелекту на процес написання коду.

План лекції:

1. Еволюція фронтенду: від Vanilla JS до SPA-фреймворків (React, Vue, Angular).
2. Інструменти збірки та екосистема Node.js (NPM, Vite).
3. Сучасна архітектура бекенду: Моноліт проти Мікросервісів.
4. Взаємодія систем: порівняння REST API та GraphQL.
5. DevOps, контейнеризація (Docker) та тренди майбутнього (AI-асистенти).

1. Еволюція фронтенду: від Vanilla JS до SPA-фреймворків

Раніше кожна взаємодія на сайті (наприклад, перехід на іншу сторінку) вимагала повного перезавантаження сторінки з сервера. Сьогодні стандартом є

SPA (Single Page Application – Односторінковий застосунок). Браузер завантажує сторінку лише один раз, а далі JavaScript динамічно підміняє контент, створюючи ілюзію миттєвої роботи, як у десктопних програмах.

Для створення складних SPA «чистого» JavaScript (Vanilla JS) недостатньо – код стає заплутаним і важким для підтримки. Тому індустрія перейшла на компонентний підхід та фреймворки/бібліотеки:

- React (від Meta): Найпопулярніша бібліотека. Використовує синтаксис JSX (HTML всередині JS).
- Vue.js: Дуже гнучкий і дружній до новачків фреймворк, який часто використовується в екосистемі PHP (зокрема з Laravel).
- Angular (від Google): Потужний, але складний фреймворк, який переважно використовується у великих корпоративних (Enterprise) проєктах.

Головна інновація – Virtual DOM: Замість того, щоб напряму і повільно змінювати реальний HTML-документ (DOM) при кожному кліку, фреймворки (як-от React) створюють його легку віртуальну копію в пам'яті. Вони обчислюють різницю між старим і новим станом, і вносять у реальний DOM лише мінімально необхідні точкові зміни. Це кардинально підвищує продуктивність.

2. Інструменти збірки та екосистема Node.js

Сучасний фронтенд-код перед відправкою в браузер потрібно «зібрати» (перекласти з сучасного синтаксису на старий для сумісності, мініфікувати, об'єднати файли).

Node.js та NPM: Хоча Node.js – це серверне середовище для виконання JavaScript, воно подарувало світові NPM (Node Package Manager). Це гігантський репозиторій готових бібліотек (слайдери, календарі, інструменти для роботи з часом), які розробник може встановити однією командою (npm install name).

Vite: Сучасний, надшвидкий інструмент збірки (bundler), який прийшов на зміну важкому Webpack. Він миттєво запускає локальний сервер для розробки та оновлює сторінку за мілісекунди після збереження коду (Hot Module Replacement).

3. Сучасна архітектура бекенду: Моноліт проти Мікросервісів

Зі зростанням складності вебзастосунків змінився і підхід до проєктування бекенду.

Монолітна архітектура: Традиційний підхід. Весь код (авторизація, каталог товарів, кошик, платіжна система, підключення до БД) знаходиться в одному великому проєкті.

Плюси: Легко почати розробку, просто тестувати та розгортати.

Мінуси: При падінні одного модуля (наприклад, помилка в модулі розсилки email) може впасти весь сайт. Масштабувати доводиться весь застосунок цілком.

Мікросервісна архітектура: Застосунок розбивається на десятки маленьких, незалежних програм (сервісів), кожна з яких виконує лише одну функцію (Сервіс Користувачів, Сервіс Оплат, Сервіс Сповіщень). Вони спілкуються між собою через мережу.

Плюси: Якщо падає сервіс відгуків, користувачі все одно можуть купувати товари. Сервіси можна писати на різних мовах (один на PHP, інший на Python).

Мінуси: Дуже складна інфраструктура та налаштування мережевої взаємодії.

4. Взаємодія систем: порівняння REST API та GraphQL

Оскільки фронтенд (React) і бекенд (PHP/Node.js) тепер існують окремо, їм потрібен міст для спілкування. Цим мостом є API (Application Programming Interface). Замість HTML-сторінок сучасний бекенд повертає сирі дані у форматі JSON.

Таблиця 5.1. Існують два головні стандарти побудови API:

Характеристика	REST API	GraphQL
Точки доступу (Endpoints)	Багато різних URL (напр., /users, /posts, /comments).	Лише один URL (напр., /graphql).
Отримання даних	Сервер сам вирішує, який набір даних повернути. Часто виникає проблема Overfetching (отримання зайвих даних).	Клієнт (фронтенд) формує запит і точно вказує, які саме поля йому потрібні, і отримує лише їх.
Гнучкість	Для нового типу відображення на фронтенді часто треба просити бекенд-розробника створити новий Endpoint.	Фронтенд-розробник може самостійно комбінувати дані з різних сутностей в одному запиті.
Крива навчання	Простий, базується на стандартних HTTP-методах (GET, POST).	Складніший, вимагає вивчення спеціальної мови запитів.

5. DevOps, контейнеризація (Docker) та тренди майбутнього

Docker (Контейнеризація):

Раніше розробники часто стикалися з проблемою: *«На моєму комп'ютері*

цей код працює, а на сервері – ні». Це відбувалося через різні версії операційних систем, PHP або баз даних.

Docker вирішує цю проблему. Він запаковує ваш код разом з усім його оточенням (потрібною версією PHP, налаштуваннями сервера) у стандартизований «контейнер». Цей контейнер гарантовано працюватиме абсолютно однаково на будь-якому комп'ютері чи сервері.

Тренди сьогодення та майбутнього:

Штучний інтелект у розробці (AI Copilots): Інструменти на кшталт GitHub Copilot або Cursor кардинально змінюють роботу програміста. Вони не замінюють розробника, але беруть на себе рутину: автодоповнення цілих блоків коду, написання тестів, пошук багів та пояснення складних ділянок коду. Навичка правильного написання промптів стає ключовою для розробника.

Serverless (Безсерверні обчислення): Вам більше не потрібно орендувати цілий сервер і платити за нього щомісяця. Ви просто завантажуєте свій код у хмару (наприклад, AWS Lambda або Vercel), і платформа сама запускає його лише в момент запиту від користувача. Ви платите виключно за мілісекунди фактичного виконання коду.

Інтеграція ШІ в освітній процес (Самостійне опрацювання)

Приклад промпту №1 (Docker): *«Дій як DevOps інженер. Поясни мені концепцію Docker на зрозумілому прикладі з реального життя (наприклад, перевезення вантажів на кораблях). Чим контейнер відрізняється від традиційної віртуальної машини (Virtual Machine) з точки зору споживання ресурсів комп'ютера?»*

Приклад промпту №2 (REST vs GraphQL): *«Я планую створити мобільний застосунок для блогу. У мене є вибір між створенням REST API та GraphQL. Поясни мені концепцію 'Overfetching' (надмірне отримання даних) у REST API на прикладі екрану списку статей, де мені потрібен лише заголовок і дата, і покажи, як GraphQL вирішує цю проблему.»*

Питання для обговорення та контролю знань:

1. Поясніть концепцію Single Page Application (SPA) та опишіть, як вона покращує користувацький досвід порівняно з традиційними багатосторінковими сайтами.
2. Поясніть принцип роботи Virtual DOM. Обґрунтуйте, чому пряме маніпулювання реальним DOM через document.getElementById у великих застосунках вважається неефективним.
3. Назвіть одну головну перевагу та один критичний недолік мікросервісної архітектури порівняно з монолітною.

4. Опишіть фундаментальну різницю у способі отримання даних між REST API (з багатьма ендпоінтами) та GraphQL.

5. Поясніть, яку фундаментальну проблему розробки та розгортання застосунків вирішує використання технології Docker.

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

Конспект лекцій

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.