

Дисципліна

**«Бекенд-розробка, сервіси та мікросервіси»**

## **КОНСПЕКТ ЛЕКЦІЇ. ТЕМА 1**

*Клієнт-серверна архітектура вебу, HTTP протокол, огляд серверних мов програмування, еволюція вебу та місце PHP серед бекенд-технологій*

Спеціальність: Комп'ютерна інженерія, 3 курс  
Обсяг: 1 год 20 хв (лекція 1 з 18)

## План лекції

1. Клієнт-серверна архітектура вебу
2. HTTP протокол: запити, відповіді, заголовки, статус-коди
3. Серверні мови програмування та їх призначення
4. Еволюція вебу: Web 1.0 → Web 2.0 → Mobile Web
5. Server-Side Rendering (SSR) та Client-Side Rendering (CSR)
6. Місце PHP серед бекенд-технологій
7. Коротка історія PHP
8. Базовий синтаксис PHP. Перша програма Hello World
9. Розгортання середовища розробки: XAMPP

*Мета лекції: сформувати цілісне розуміння того, як влаштована взаємодія клієнта й сервера в вебi, яке місце в цій взаємодії посідає PHP, і підготувати робоче середовище для написання перших серверних скриптів.*

# 1. Клієнт-серверна архітектура вебу

## 1.1. Що таке клієнт-серверна модель

Клієнт-серверна архітектура — це спосіб організації розподілених обчислень, за якого функціональність програмної системи розділена між двома типами учасників: клієнтом, який ініціює запити, та сервером, який ці запити обробляє й повертає результат. У випадку Вебу клієнтом найчастіше виступає браузер (Chrome, Firefox, Safari) або будь-яка інша програма, здатна виконувати HTTP-запити (мобільний застосунок, скрипт, IoT-пристрій), а сервером — програма (веб-сервер), яка приймає ці запити, виконує потрібну логіку та повертає відповідь.

Головна ідея розподілу: клієнт відповідає за представлення (як виглядає інтерфейс, взаємодія з користувачем), а сервер — за дані, бізнес-логіку та їх безпеку. Саме тому серверну частину прийнято називати «бекендом» (від англ. back end — те, що «позаду», приховане від користувача), а клієнтську — «фронтендом».

## 1.2. Життєвий цикл запиту

1. Користувач виконує дію в браузері (вводить URL, натискає кнопку, надсилає форму).
2. Браузер формує HTTP-запит і надсилає його на сервер за вказаною адресою (доменне ім'я перетворюється на IP-адресу через DNS).
3. Запит проходить мережею (можливо, через проміжні вузли — проху, CDN, балансувальники навантаження) і потрапляє на веб-сервер (Apache, Nginx).
4. Веб-сервер визначає, чи потрібно віддати статичний файл, чи передати запит на виконання серверному скрипту (наприклад, PHP-інтерпретатору через модуль mod\_php або PHP-FPM).
5. Серверний скрипт виконує логіку: може звертатися до бази даних, файлової системи, зовнішніх сервісів.
6. Сервер формує HTTP-відповідь (як правило, HTML, JSON або файл) і надсилає її назад клієнту.
7. Браузер отримує відповідь і відображає її користувачу (рендерить HTML, виконує JS, оновлює DOM).

## 1.3. Багаторівнева (N-tier) архітектура

У найпростішому вигляді архітектура складається з двох рівнів (2-tier): клієнт і сервер, який одночасно виконує логіку та зберігає дані. У промислових застосунках частіше застосовують трирівневу (3-tier) архітектуру, яка розділяє відповідальність ще детальніше:

| Рівень                              | Призначення                                                 | Приклади технологій          |
|-------------------------------------|-------------------------------------------------------------|------------------------------|
| Рівень представлення (Presentation) | Інтерфейс користувача, взаємодія з браузером                | HTML, CSS, Vanilla JS, React |
| Рівень бізнес-логіки (Application)  | Обробка запитів, правила предметної області, автентифікація | PHP, Laravel, Node.js, Java  |
| Рівень даних (Data)                 | Зберігання та вибірка даних                                 | MySQL, PostgreSQL,           |

| Рівень | Призначення | Приклади технологій |
|--------|-------------|---------------------|
|        |             | MongoDB, Redis      |

У курсі, який ми вивчаємо, рівень представлення студенти вже опанували (HTML, Vanilla JS). Наше завдання — навчитися будувати рівень бізнес-логіки на PHP і грамотно поєднувати його з рівнем даних, а надалі — розкласти цю логіку на окремі сервіси (мікросервісну архітектуру).

## 2. HTTP протокол

### 2.1. Місце HTTP у мережевому стеку

HTTP (HyperText Transfer Protocol) — прикладний протокол передачі даних, який визначає формат повідомлень між клієнтом і сервером у Вебі. HTTP працює поверх транспортного протоколу TCP (а в HTTP/3 — поверх QUIC, що базується на UDP), який гарантує надійну доставку пакетів. Захищена версія — HTTPS — додає шар шифрування TLS/SSL між HTTP та TCP, забезпечуючи конфіденційність і цілісність переданих даних.

Ключова властивість HTTP — його stateless-природа (відсутність стану): кожен запит обробляється сервером незалежно від попередніх, сервер «не пам'ятає» користувача між запитами. Саме тому для збереження стану (наприклад, факту авторизації) використовують додаткові механізми — cookies та сесії, які ми детально розглянемо в наступних темах.

### 2.2. Структура HTTP-запиту

HTTP-запит складається з трьох частин:

```
GET /catalog/products?category=laptops HTTP/1.1
Host: shop.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xhtml+xml
Accept-Language: uk-UA,uk;q=0.9
Cookie: session_id=a1b2c3d4
Connection: keep-alive
```

(тіло запиту — відсутнє для GET)

- Рядок запиту (request line) — метод, шлях ресурсу (URI) та версія протоколу.
- Заголовки (headers) — метадані запиту: тип клієнта, мова, куки, тип очікуваного контенту тощо.
- Тіло запиту (body) — необов'язкова частина; містить дані, які клієнт надсилає серверу (наприклад, вміст форми при POST-запиті).

### 2.3. HTTP-методи

| Метод  | Призначення                                  | Наявність тіла |
|--------|----------------------------------------------|----------------|
| GET    | Отримати ресурс (не змінює стан на сервері)  | Немає          |
| POST   | Створити новий ресурс / надіслати дані форми | Є              |
| PUT    | Повністю оновити (замінити) наявний ресурс   | Є              |
| PATCH  | Частково оновити наявний ресурс              | Є              |
| DELETE | Видалити ресурс                              | Зазвичай немає |

| Метод   | Призначення                                               | Наявність тіла |
|---------|-----------------------------------------------------------|----------------|
| HEAD    | Те саме, що GET, але без тіла відповіді (лише заголовки)  | Немає          |
| OPTIONS | Дізнатися, які методи підтримує ресурс (важливо для CORS) | Немає          |

*Методи GET, HEAD, OPTIONS, PUT, DELETE вважаються ідемпотентними — повторне виконання того самого запиту не змінює результат порівняно з одноразовим виконанням. POST і PATCH — ні. Це важливо враховувати під час проектування REST API (тема 6).*

## 2.4. Структура HTTP-відповіді

```
HTTP/1.1 200 OK
Date: Tue, 01 Sep 2026 10:15:32 GMT
Server: Apache/2.4.58 (Win64) PHP/8.3.9
Content-Type: text/html; charset=UTF-8
Content-Length: 1284
Set-Cookie: session_id=a1b2c3d4; HttpOnly; Path=/

<!DOCTYPE html><html>...</html>
```

Статус-рядок відповіді містить версію протоколу, тризначний код статусу та текстове пояснення (reason phrase). Заголовки описують метадані відповіді (тип контенту, розмір, сервер), а тіло — власне вміст (HTML-сторінка, JSON-об'єкт, файл).

## 2.5. Коди статусів HTTP

| Клас | Значення                                                  | Приклади кодів                                                      |
|------|-----------------------------------------------------------|---------------------------------------------------------------------|
| 1xx  | Інформаційні — запит прийнято, обробка триває             | 100 Continue                                                        |
| 2xx  | Успіх — запит успішно оброблено                           | 200 OK, 201 Created, 204 No Content                                 |
| 3xx  | Перенаправлення — потрібні додаткові дії клієнта          | 301 Moved Permanently, 302 Found, 304 Not Modified                  |
| 4xx  | Помилка клієнта — некоректний запит                       | 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found     |
| 5xx  | Помилка сервера — сервер не зміг обробити коректний запит | 500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable |

## 2.6. Ключові заголовки

- Content-Type — MIME-тип тіла запиту/відповіді (text/html, application/json, multipart/form-data тощо).
- Content-Length — розмір тіла в байтах.
- Authorization — облікові дані для автентифікації (Basic, Bearer-токен).
- Cookie / Set-Cookie — передача та встановлення кук для збереження стану сесії.
- Cache-Control — правила кешування ресурсу браузером і проміжними вузлами.
- Location — адреса для перенаправлення (використовується разом із кодами 3xx).

## 2.7. Еволюція версій протоколу

HTTP/1.1 (1997) — базова версія з постійними з'єднаннями (keep-alive), досі найпоширеніша. HTTP/2 (2015) додав мультиплексування кількох запитів в одному TCP-з'єднанні, стиснення заголовків і пріоритезацію, що суттєво пришвидшило завантаження сторінок. HTTP/3 (2022) відмовився від TCP на користь протоколу QUIC (поверх UDP), що усуває проблему «head-of-line blocking» і ще більше скорочує затримки, особливо в мобільних мережах.

### 3. Серверні мови програмування та їх призначення

Серверна (бекенд) мова програмування — це мова, скрипти якої виконуються не в браузері користувача, а на сервері, і результат їх роботи (як правило, готовий HTML, JSON або інші дані) передається клієнту. На відміну від клієнтських мов (JavaScript у браузері), серверний код має доступ до бази даних, файлової системи сервера, зовнішніх API з приватними ключами, і його вихідний код ніколи не потрапляє до користувача — користувач бачить лише результат виконання.

#### 3.1. Огляд поширених серверних мов і платформ

| Мова / платформа     | Типове застосування                                          | Особливість                                        |
|----------------------|--------------------------------------------------------------|----------------------------------------------------|
| PHP                  | Веб-сайти, CMS (WordPress), API, e-commerce                  | Найпоширеніша мова для вебу; низький поріг входу   |
| JavaScript (Node.js) | API реального часу, чати, стріми, SPA-бекенди                | Той самий синтаксис, що й на фронтенді             |
| Python               | Дата-сервіси, ML-інтеграції, Django/FastAPI застосунки       | Лаконічний синтаксис, потужна екосистема аналітики |
| Java                 | Корпоративні системи, банкінг, високонавантажені сервіси     | Строга типізація, висока продуктивність JVM        |
| C# (.NET)            | Корпоративні застосунки, застосунки на Windows-стеку         | Глибока інтеграція з екосистемою Microsoft         |
| Ruby                 | Швидкий прототип, стартапи (Ruby on Rails)                   | Орієнтація на «щастя розробника» й конвенції       |
| Go                   | Мікросервіси, високонавантажені API, інфраструктурні утиліти | Компільована мова, вбудована паралельність         |

#### 3.2. Чому саме PHP у цьому курсі

- Мова створювалася спеціально для Вебу — від синтаксису до вбудованих функцій (робота з формами, куками, сесіями «з коробки»).
- Мінімальний поріг входу: перший динамічний скрипт можна написати за кілька хвилин після встановлення XAMPP.
- Величезна поширеність — сотні мільйонів сайтів у продакшн (детальніше в розділі 6), а отже — величезний ринок праці й підтримки.
- Сучасний PHP (8.x) — це вже не «скриптова мова для вставок у HTML», а повноцінна об'єктно-орієнтована мова з типізацією, атрибутами, JIT-компіляцією, на якій побудовані індустріальні фреймворки (Laravel, Symfony).
- Логічний місток від фронтенду (HTML + Vanilla JS), який студенти вже знають, до повноцінної розробки динамічних застосунків.

## 4. Еволюція вебу: Web 1.0 → Web 2.0 → Mobile Web

### 4.1. Web 1.0 (приблизно 1991–2004)

Перше покоління Вебу — це переважно статичні HTML-сторінки в режимі «тільки читання» (read-only). Сайти створювалися вручну або невеликою кількістю авторів, взаємодія користувача обмежувалася переходами за гіперпосиланнями та переглядом контенту. Динамічний контент, якщо й був, генерувався на сервері (ранні CGI-скрипти, Perl, перші версії PHP) і повністю перезавантажував сторінку при кожній дії.

### 4.2. Web 2.0 (приблизно 2004 — і дотепер)

Друге покоління принесло інтерактивність і контент, який створюють самі користувачі (User-Generated Content): блоги, соціальні мережі, вікі, форуми. Ключову роль відіграла технологія AJAX (Asynchronous JavaScript and XML), яка дозволила оновлювати частину сторінки без повного перезавантаження — саме на цьому базуються сучасні динамічні інтерфейси. Бекенд у цю епоху ускладнюється: з'являються повноцінні фреймворки, ORM, API для обміну даними між сервісами.

### 4.3. Mobile Web та сучасний етап

З масовим поширенням смартфонів (з 2007–2010 рр.) веб змушений був адаптуватися до малих екранів і нестабільного з'єднання: з'явився responsive-дизайн (адаптивна верстка), а пізніше — Progressive Web Apps (PWA), які поєднують переваги сайту (доступність за посиланням) і нативного застосунку (робота офлайн, push-сповіщення, іконка на робочому столі). Бекенду тепер часто доводиться обслуговувати одночасно і веб-клієнт, і мобільні застосунки через єдине API — це один з ключових аргументів на користь REST/JSON API, які ми розглянемо в темі 6.

*Іноді згадують і термін Web 3.0 — децентралізований веб на основі блокчейну та семантичних технологій. Це радше перспективний напрям, ніж усталений стандарт, тому в курсі ми на ньому не фокусуємось.*

## 5. Server-Side Rendering (SSR) та Client-Side Rendering (CSR)

### 5.1. Server-Side Rendering

При SSR сервер сам формує готову HTML-розмітку у відповідь на запит: PHP-скрипт виконується, звертається до бази даних, «вставляє» дані у шаблон і повертає браузеру вже готову сторінку. Браузеру залишається лише відобразити отриманий HTML. Це класичний підхід, на якому побудовані WordPress, більшість корпоративних сайтів і те, що ми будемо писати на PHP у першій половині курсу.

### 5.2. Client-Side Rendering

При CSR сервер віддає майже порожню HTML-сторінку та файл JavaScript. Браузер завантажує цей JS (наприклад, застосунок на React, Vue або навіть Vanilla JS), і вже сам, на стороні клієнта, звертається до сервера за даними (як правило, через JSON API) та будує (рендерить) інтерфейс. Сервер у цьому підході здебільшого перетворюється на постачальника даних — саме такий сервер ми навчимося будувати в темах 6–9 (REST API, JSON, фреймворки).

### 5.3. Порівняння підходів

| Критерій                              | SSR                                  | CSR                                             |
|---------------------------------------|--------------------------------------|-------------------------------------------------|
| Перше відображення сторінки           | Швидше (готовий HTML одразу)         | Повільніше (потрібно завантажити й виконати JS) |
| Індексація пошуковими системами (SEO) | Простіша, контент одразу в HTML      | Складніша, потребує додаткових технік           |
| Навантаження на сервер                | Вище (сервер формує розмітку щоразу) | Нижче (сервер лише віддає дані)                 |
| Взаємодія без перезавантаження        | Обмежена без додаткового JS          | Природна (SPA)                                  |
| Приклад                               | Класичний сайт на PHP                | SPA на React/Vue + PHP JSON API                 |

У продакшні межа між підходами часто розмита: сучасні застосунки поєднують SSR (для швидкого першого завантаження та SEO) з подальшою «гідратацією» на клієнті (hydration), коли JS «оживляє» вже відрендерений HTML. У нашому курсі ми пройдемо шлях від чистого SSR на PHP (теми 1–5) до побудови JSON API, яке живить окремий клієнтський застосунок (теми 6 і далі).

## 6. Місце PHP серед бекенд-технологій

За даними аналітичного сервісу W3Techs (станом на вересень 2026 року), PHP використовується на 70,2% усіх сайтів, для яких відома серверна мова програмування, — це набагато більше, ніж у будь-якого окремого конкурента:

| Мова                 | Частка сайтів |
|----------------------|---------------|
| PHP                  | 70,2%         |
| JavaScript (Node.js) | 7,2%          |
| Ruby                 | 7,0%          |
| Java                 | 5,5%          |
| ASP.NET              | 4,2%          |
| Python               | 1,2%          |

- Ядром популярності PHP залишається CMS WordPress, яка керує величезною часткою всього Вебу — і сама написана на PHP.
- Серед сучасних PHP-проектів домінує фреймворк Laravel — за даними опитування JetBrains «The State of PHP», його регулярно використовує більшість PHP-розробників.
- У межах самої мови відбувся перехід на 8-му гілку: PHP 8 нині встановлено на більшості активних PHP-сайтів, що свідчить про активний розвиток екосистеми, а не «застигання» на старих версіях.
- PHP лежить в основі таких масштабних продуктів, як Wikipedia, Facebook (історично) і безліч e-commerce платформ.

*Висновок для курсу: попри те, що навколо нових технологій (Node.js, Go) більше «хайпу» в професійних спільнотах, саме глибоке знання PHP відкриває найбільший ринок вакансій — від підтримки й розвитку CMS-сайтів до побудови власних API на Laravel.*

## 7. Коротка історія PHP

| Рік       | Подія                                                                                                                                                                   |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1994–1995 | Расмус Лердорф (Rasmus Lerdorf) створює набір CGI-скриптів «Personal Home Page Tools» (PHP Tools) для відстеження відвідувань власної сторінки.                         |
| 1997–1998 | Вихід PHP/FI 2.0, а згодом PHP 3 — мова переписана «з нуля» командою Енді Гутманса та Зеєва Сураскі, отримує назву «PHP: Hypertext Preprocessor» (рекурсивний акронім). |
| 2000      | PHP 4 на новому русії Zend Engine — суттєве підвищення продуктивності та стабільності.                                                                                  |
| 2004      | PHP 5 — повноцінна підтримка ООП (класи, інтерфейси, винятки), що наближає мову до «дорослих» об'єктно-орієнтованих мов.                                                |
| 2015      | PHP 7 (після пропущеної версії 6) — русій Zend Engine 3 підвищує продуктивність удвічі-втричі порівняно з PHP 5.                                                        |
| 2020      | PHP 8.0 — додано JIT-компіляцію, атрибути (Attributes), union-типи, значно суворіша типізація.                                                                          |
| 2023–2026 | PHP 8.2–8.4 — подальше вдосконалення типізації (readonly-класи, enum-покращення), продуктивності та інструментів розробника.                                            |

Важливо розуміти: сучасний PHP 8.x кардинально відрізняється від того, яким мова була у 2000-х роках. Стереотип «PHP — це мова для аматорського програмування» сформувався на прикладі коду, написаного 15–20 років тому; сучасна практика розробки на PHP (з типізацією, ООП, автозавантаженням через Composer, тестуванням) не поступається практикам, прийнятим в інших мовах.

## 8. Базовий синтаксис PHP. Перша програма Hello World

### 8.1. Як PHP-код потрапляє в HTML

PHP — мова, вбудована в HTML: інтерпретатор виконує лише той код, що обрамлений спеціальними тегамі `<?php ... ?>`, а весь інший текст файлу передає у відповідь «як є». Файл із PHP-кодом зазвичай має розширення `.php` і за замовчуванням обробляється веб-сервером, а не браузером напямую.

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Перша PHP-сторінка</title>
</head>
<body>

  <?php
    // Це коментар. echo — виводить рядок у HTML-відповідь
    echo "Hello, World!";
  ?>

</body>
</html>
```

Коли браузер запитує цей файл, веб-сервер (Apache) передає його на обробку PHP-інтерпретатору. Інтерпретатор виконує лише вміст між `<?php` і `?>`, а результат (рядок «Hello, World!») підставляє в те саме місце документа. Браузер отримує вже чистий HTML — «Hello, World!» всередині тегу `<body>` — і навіть не підозрює, що там була якась логіка на PHP: перегляд вихідного коду сторінки (View Source) ніколи не покаже сам PHP-код, лише результат його роботи.

### 8.2. Мінімальний синтаксис: змінні, коментарі, echo

```
<?php
// Одноряковий коментар
/* Багаторяковий
   коментар */

// Змінна в PHP завжди починається з символу $
$name = "Мар'ян";
$year = 2026;

// echo та print виводять дані в потік відповіді
echo "Привіт, " . $name . "! Зараз " . $year . " рік.";

// Підстановка змінних прямо в рядок у подвійних лапках
echo "Привіт, $name! Зараз $year рік.";
```

- Кожна інструкція завершується крапкою з комою ;
- Змінні не оголошуються заздалегідь і не мають фіксованого типу (динамічна типізація) — тип визначається за значенням.
- Крапка . — оператор конкатенації (з'єднання) рядків.
- У рядках у подвійних лапках "\"" змінні автоматично підставляються; в одинарних "'" — ні.

*Детальний розгляд типів даних, операторів, умовних конструкцій та циклів PHP — тема наступної лекції. Зараз наша мета — переконатися, що середовище працює й перший скрипт виконується.*

## 9. Розгортання середовища розробки: ХАМРР

### 9.1. Що таке ХАМРР

ХАМРР — це безкоштовний крос-платформний дистрибутив, який в одному інсталяторі об'єднує всі компоненти, потрібні для локальної розробки PHP-застосунків: (X) крос-платформність (Windows/Linux/macOS), (A) Apache — веб-сервер, (M) MariaDB/MySQL — сервер бази даних, (P) PHP — інтерпретатор, (P) Perl — рідше використовується у навчальних курсах. Замість того, щоб окремо встановлювати й налаштовувати кожен компонент, ХАМРР дозволяє розгорнути повноцінне середовище розробки за кілька хвилин.

### 9.2. Мінімальні вимоги

| Параметр              | Мінімальні вимоги                                                  |
|-----------------------|--------------------------------------------------------------------|
| Операційна система    | Windows 10/11 (64-біт), macOS 12+, або сучасний дистрибутив Linux  |
| Оперативна пам'ять    | від 2 ГБ (рекомендовано 4+ ГБ для комфортної роботи з IDE)         |
| Вільне місце на диску | від 1 ГБ (сам дистрибутив) + місце під проекти й бази даних        |
| Права доступу         | Права адміністратора для встановлення та роботи портів 80/443/3306 |
| Мережа                | Не обов'язкова — ХАМРР працює повністю локально (localhost)        |

### 9.3. Встановлення та перший запуск

1. Завантажити інсталятор з офіційного сайту [apachefriends.org](http://apachefriends.org), обравши версію під свою ОС.
2. Запустити інсталятор, залишивши стандартний набір компонентів (Apache, MySQL/MariaDB, PHP, phpMyAdmin).
3. Обрати теку встановлення (типово C:\xampp у Windows).
4. Відкрити ХАМРР Control Panel і натиснути Start навпроти модулів Apache та MySQL — індикатор стане зеленим, коли служба запущена.
5. Перевірити роботу, відкривши в браузері <http://localhost> — має з'явитися стартова сторінка ХАМРР.

### 9.4. Структура проєкту та запуск PHP-скриптів

Усі файли, які мають бути доступні через веб-сервер, розміщуються в кореневій теці документів — за замовчуванням це C:\xampp\htdocs (Windows) або /opt/lampp/htdocs (Linux). Кожна підтека всередині htdocs стає доступною за відповідним шляхом у браузері:

```
htdocs/
├─ index.php           → http://localhost/
├─ hello-world/
│   └─ index.php       → http://localhost/hello-world/
```

```
└─ lab1/  
  └─ task1.php      → http://localhost/lab1/task1.php
```

Важливо: PHP-файл потрібно відкривати саме через адресу `http://localhost/...`, а не подвійним клацанням у Провіднику (це відкриє його як текст або через `file://`, і PHP-код виконано не буде) — файл мусить пройти через Apache, який передасть його PHP-інтерпретатору.

### 9.5. Перевірка встановлення: `phpinfo()`

Стандартний спосіб перевірити коректність налаштування PHP — створити файл `info.php` з єдиним рядком коду й відкрити його в браузері:

```
<?php  
phpinfo();  
?>
```

Функція `phpinfo()` виводить детальну довідкову сторінку: встановлену версію PHP, підключені розширення (extensions), шлях до конфігураційного файлу `php.ini`, ліміти пам'яті й розміру завантажуваних файлів тощо. Цю сторінку зручно використовувати для діагностики середовища, але в реальному (продакшн) проєкті такий файл ніколи не залишають доступним — він розкриває забагато технічної інформації.

*На лабораторній роботі 1 кожен студент встановить XAMPP, запустить Apache й MySQL, створить файл `hello.php` з виводом власного імені та акаdemічної групи, а також перевірить середовище через `phpinfo()`.*

## Питання для самоперевірки

- Поясніть, чим клієнт відрізняється від сервера у клієнт-серверній архітектурі. Наведіть приклад із повсякденного використання Вебу.
- Опишіть повний шлях HTTP-запиту від натискання кнопки в браузері до відображення результату користувачу.
- Чим відрізняються методи GET і POST? Коли варто використовувати кожен з них?
- Що означає stateless-природа HTTP і як на практиці вирішується проблема збереження стану користувача?
- Назвіть по три коди статусів кожного класу (2xx, 3xx, 4xx, 5xx) і поясніть їх значення.
- Порівняйте Server-Side Rendering та Client-Side Rendering: назвіть по два переваги й недоліки кожного підходу.
- Чому, попри наявність десятків серверних мов, PHP досі займає понад 70% ринку серверних технологій?
- Які компоненти входять до складу XAMPP і за що відповідає кожен з них?
- Поясніть, чому подвійне клацання на .php-файлі в Провіднику не є коректним способом його запуску.

## Короткий глосарій

| Термін              | Означення                                                                          |
|---------------------|------------------------------------------------------------------------------------|
| Бекенд (backend)    | Серверна частина застосунку: логіка, дані, безпека, недоступна користувачу напряду |
| Фронтенд (frontend) | Клієнтська частина застосунку: інтерфейс, з яким взаємодіє користувач              |
| HTTP                | Прикладний протокол передачі гіпертекстових даних між клієнтом і сервером          |
| Stateless           | Властивість протоколу не зберігати стан між окремими запитами                      |
| SSR                 | Server-Side Rendering — формування готової HTML-розмітки на сервері                |
| CSR                 | Client-Side Rendering — побудова інтерфейсу засобами JavaScript у браузері         |
| XAMPP               | Дистрибутив для локального розгортання Apache, MySQL/MariaDB, PHP і Perl           |
| htdocs              | Коренева тека веб-сервера Apache, з якої обслуговуються файли сайту                |