

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО»

# СЕРВЕРНІ WEB-ТЕХНОЛОГІЇ

**Навчальний посібник**

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського  
як навчальний посібник для здобувачів ступеня бакалавра  
за освітньою програмою «Автоматизація та комп'ютерно-інтегровані технології кібер-енер-  
гетичних систем»  
спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології

Укладач: О. С. Бунке

Електронне мережне навчальне видання

Київ  
КПІ ім. Ігоря Сікорського  
2023

Рецензент

*Сірий О.А.*, к.т.н., доцент,  
кафедра теплової та альтернативної енергетики, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

Відповідальний редактор

*Новіков, П.В.*, к.т.н., доцент

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського  
(протокол № 8 від 02.06.2023 р.)  
за поданням Вченої ради ННІАТЕ  
(протокол № 13 від 29.05.2023 р.)*

Посібник розроблений на підставі силабусу навчальної дисципліни «Серверні веб-технології» та призначений для проведення лабораторних занять, підвищення розуміння основ WEB-технологій.

Призначений для студентів, які навчаються за освітньою програмою підготовки бакалаврів за спеціальністю 151 "Автоматизація та комп'ютерно-інтегровані технології".

Спрямований на формування у студентів умінь та навичок з проектування та розробки серверної частини WEB-застосунків. Забезпечує студентів необхідними теоретичними знаннями для виконання практичних завдань, запланованих впродовж семестру.

Реєстр. № 22/23-754. Обсяг 3,65 авт. арк.

Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
проспект Перемоги, 37, м. Київ, 03056  
<https://kpi.ua>

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© КПІ ім. Ігоря Сікорського, 2023

## Зміст

<b>ВСТУП</b> .....	6
<b>РОЗДІЛ 1 КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА</b> .....	7
1.1 Протокол HTTP .....	9
1.2 Опис роботи HTTP .....	9
1.3 Протокол HTTPs .....	10
1.4 Детальніше про бекенд .....	11
1.5 Інструменти розробки серверної частини WEB-застосунку.....	11
1.6 Відмінність Frontend і Backend розробки.....	12
1.7 Application Programming Interface (API).....	12
1.8 Чому API називають інтерфейсом .....	13
1.9 Призначення API .....	14
1.10 API на прикладі.....	14
1.11 REST API .....	15
1.12 Опис роботи REST API .....	15
1.13 Файли-куки.....	16
1.14 Сесії.....	17
1.15 Питання для самоконтролю.....	17
<b>РОЗДІЛ 2 ТИПИ ВЕБ-САЙТІВ</b> .....	18
2.1 Статичні сайти на зорі Інтернету.....	18
2.2 Технологія AJAX .....	19
2.3 SPA .....	19
2.4 Повернення до Server Side Rendering .....	20
2.5 Static Site Generator.....	21
2.6 Питання для самоконтролю.....	23
<b>РОЗДІЛ 3 ОГЛЯД МОВ ДЛЯ СЕРВЕРНИХ ЗАСТОСУНКІВ</b> .....	24
3.1 Типи мов програмування.....	24
3.2 HTML та CSS .....	25
3.3 Python .....	26
3.4 JavaScript.....	26
3.5 PHP .....	27
3.6 Java .....	28
3.7 C#.....	28
3.8 Питання для самоконтролю.....	29
<b>РОЗДІЛ 4 СЕРВІСНА АРХІТЕКТУРА</b> .....	30
4.1 Основні принципи мікросервісної архітектури.....	30
4.2 Переваги мікросервісної архітектури.....	31
4.3 Монолітна архітектура .....	31
4.4 Переваги монолітної архітектури .....	32

4.5	Порівняння сервісного та монолітного підходів .....	32
4.6	Перехід від моноліту до мікросервісів .....	33
4.7	Підсумки до розділу .....	33
4.8	Питання для самоконтролю .....	33
<b>РОЗДІЛ 5 ОСНОВИ МОВИ PHP .....</b>		<b>35</b>
5.1	Створення змінної.....	35
5.2	Константи в PHP .....	36
5.3	Функції в PHP.....	37
5.4	Посилання.....	38
5.5	Масив .....	39
5.6	Генератори.....	39
5.7	Підхід до написання чистого коду .....	40
5.8	Найменування та розділення .....	41
5.9	Функції.....	42
5.10	Коментарі.....	43
5.11	Форматування та правила .....	43
5.12	Класи .....	44
5.13	Обробка помилок.....	45
5.14	Межі .....	45
5.15	Приклади коду PHP .....	45
5.16	Питання для самоконтролю .....	74
<b>РОЗДІЛ 6 ПАКЕТНИЙ МЕНЕДЖЕР COMPOSER.....</b>		<b>76</b>
6.1	Інсталяція Composer .....	76
6.2	Встановлення на Linux/Unix/macOS .....	76
6.3	Встановлення на Windows .....	77
6.4	Синтаксис та опції Composer.....	77
6.5	Оновлення пакетів .....	79
6.6	Видалення пакетів .....	79
6.7	Скидання автозавантаження.....	79
6.8	Питання для самоконтролю .....	80
<b>РОЗДІЛ 7 ФРЕЙМВОРК SYMFONY .....</b>		<b>81</b>
7.1	Основні компоненти фреймворка Symfony .....	81
7.2	Структура веб-додатку на Symfony .....	82
7.3	Процес запиту .....	82
7.4	Розгортання проєкту на Symfony .....	83
7.5	Підсумки до розділу .....	84
7.6	Питання для самоконтролю .....	84
<b>РОЗДІЛ 8 WEB-СЕРВЕР НА NODE.JS.....</b>		<b>85</b>
8.1	Блокуючі введення/виведення.....	86
8.2	Проблема C10K.....	87

8.3	Програмна платформа Node.js.....	88
8.4	Node.js та цикл подій.....	88
8.5	Цикл подій.....	90
8.6	Проблема CPU-ємних завдань.....	92
8.7	Воркери та їх потоки.....	95
8.8	Директиви Import та Export.....	97
8.9	Cookie в node.js.....	98
8.10	Сесії в node.js.....	99
8.11	Питання для самоконтролю.....	100
<b>РОЗДІЛ 9 РОЗГОРТАННЯ ВЕБ-ДОДАТКІВ В МЕРЕЖІ.....</b>		<b>101</b>
9.1	Завантаження файлів на хостинг за допомогою FTP.....	101
9.2	Docker.....	102
9.3	GitLab CI.....	103
9.4	Практичні поради та кращі практики.....	105
9.5	Поради при роботі з GitLab CI.....	106
9.6	Підсумки до розділу.....	106
9.7	Питання для самоконтролю.....	106
<b>ПЕРЕЛІК ПОСИЛАНЬ.....</b>		<b>108</b>

## ВСТУП

Практично всі WEB-сайти та Мобільні застосунки звідкись отримують і кудись зберігають данні – ця непомітна для користувача компонента називається серверною, або «бек-ендом». Розробка серверних продуктів для WEB базується на знаннях архітектури, баз даних та деяких популярних мов програмування.

У посібнику наводиться опис принципів роботи серверних веб-додатків, порядок взаємодії користувача із веб-сервером, принципи роботи програмних інтерфейсів (API), мікросервісна архітектура, огляд популярної мови програмування PHP та програмної платформи NodeJs.

Метою навчальної дисципліни «Серверні веб-технології» є формування та розвиток у студентів таких компетентностей:

- Здатність застосовувати знання у практичних ситуаціях;
- Здатність до пошуку, опрацювання та аналізу інформації з різних джерел;
- Здатність використовувати для вирішення професійних завдань новітні технології у галузі автоматизації та комп'ютерно-інтегрованих технологій, зокрема, проектування багаторівневих систем керування, збору даних та їх архівування для формування бази даних параметрів процесу та їх візуалізації за допомогою засобів людино-машинного інтерфейсу.
- Здатність вільно користуватись сучасними комп'ютерними та інформаційними технологіями для вирішення професійних завдань, програмувати та використовувати прикладні та спеціалізовані комп'ютерно-інтегровані середовища для вирішення задач автоматизації.

У результаті вивчення навчальної дисципліни студенти набудуть таких загальних програмних результатів навчання:

- Вміти програмувати інформаційні сервіси з використанням клієнт-серверних технологій.

Для успішного опанування дисципліни студент має володіти базовими знаннями з алгоритмів, прослухати дисципліни «сучасні технології програмування» та «проектування та розробка баз даних».

Результати навчання використовуються при дипломному проектуванні, у вибіркових дисциплінах подібного спрямування та у професійній діяльності.

## РОЗДІЛ 1 КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА

Функціонування мережі Інтернет засноване на архітектурі клієнт-сервер. Водночас програма-клієнт (зазвичай браузер) на комп'ютері користувача відправляє запити для отримання інформації або ресурсів (зазвичай це запити за URL-адресою) програмі-серверу, що встановлена на одному із серверних комп'ютерів, підключених до мережі та ідентифікованому IP-адресою. Логіка роботи сервера забезпечує видачу файлів, контенту, ресурсів, які зберігаються на сервері, браузеру і кілька допоміжних функцій (рис. 1) [1].

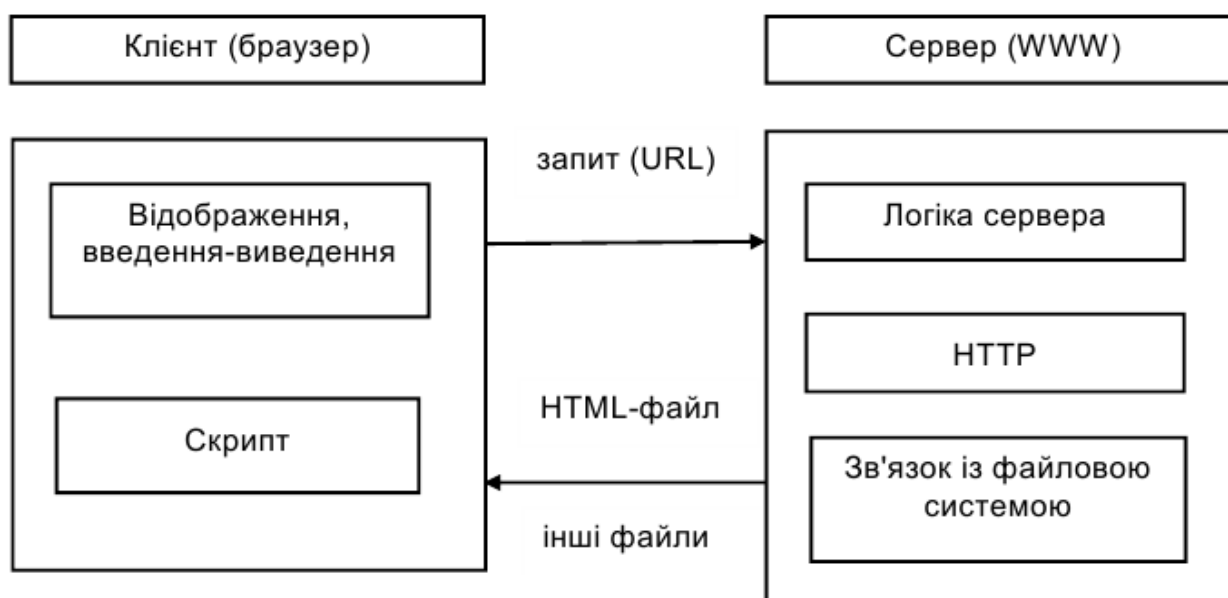


Рисунок 1 – Клієнт-серверна архітектура у WEB

За такої взаємодії сервер грає роль сховища. Для розширення можливостей WEB-сторінок із реалізації більш складних функцій необхідно розширювати логіку сервера за рахунок виконання спеціально розроблених програм. Ці програми можуть виконувати найрізноманітніші функції: оброблення сторінок перед відправкою, зв'язок із базами даних, проведення обчислень тощо. Таким чином під серверними технологіями будемо розуміти сукупність засобів і методів запуску виконуваних програм на сервері.

Для забезпечення функціонування сервісу WWW було розроблено досить багато серверів. Вони володіють різними характеристиками, але найбільш масовими є два: IIS (Internet Information Services), розроблений компанією "Майкрософт", і вільно поширювані Apache та Nginx.

Програми, які розробляються для виконання на сервері крім реалізації основних функцій, повинні мати кілька особливостей (порівняно з тими, що виконуються в середовищі операційної системи):

- запуск здійснюється сервером;
- має бути забезпечена можливість взаємодії із сервером;
- має бути забезпечена можливість відправляти результати (дані) клієнту, який видав запит (браузеру).

Для створення серверних розширень було розроблено кілька інтерфейсів (специфікацій на взаємодію між сервером та програмою), найбільшого поширення набули: CGI, FCGI і API-інтерфейси. Кожен із них має свої переваги, так само, як і певні недоліки [1].

Технологія активних серверних сторінок (включення коду в саму HTML-сторінку) полягає в тому, що запитувана сторінка з розміченим текстом містить програмний код, який виконується на сервері перед відправкою клієнту, змінюючи запитовану сторінку. Цей код може витягувати дані з бази даних, будувати призначений для користувача інтерфейс залежно від отриманих даних, а потім відправляти HTML-сторінку назад клієнту. Кінцевим результатом такого процесу буде відправка браузеру стандартної WEB-сторінки на HTML. Цей процес являє собою динамічне створення WEB-сторінок. Сторінка, яка показується в браузері, насправді є продуктом виконання коду, який був запущений після клієнтського запиту, на сервері такої сторінки фізично не існує.

Гнучкість і зручність WEB-додатків ґрунтується на відокремленні візуальної частини застосунку від логіки його роботи. WEB-додаток по суті є розподіленою програмою, яка використовує для обміну даними протокол HTTP. Перевагою такого додатка є стандартний, всім відомий клієнт-браузер, і зниження витрат на супровід додатків розробниками. Замість того, щоб встановлювати додаток Windows на кожне робоче місце в компанії, можна створити один WEB-додаток, до якого є загальний доступ. Якщо виникає необхідність внесення в цей додаток змін або виправлень, то досить оновити додаток тільки на сервері, а не на всіх клієнтських комп'ютерах. Це також



дозволяє кожному клієнту постійно користуватися самою останньою версією програми, оскільки саме вона завжди буде знаходитися на сервері [1].

Незважаючи на те, що мова йде, по суті, про розподілений додаток, користувач, який працює в браузері (набираючи URL в рядку адреси або клацаючи по посиланнях), може цього не помічати. У зв'язку із цим додатки, що складаються з активних сторінок, продовжують називати сайтами.

## **1.1 Протокол HTTP**

HTTP — це протокол передачі даних, на основі якого працює всесвітня павутина. Завдяки цьому протоколу ми можемо заходити на сайти в браузері та взаємодіяти з ними: переходити з однієї сторінки на іншу, завантажувати файли та переглядати зображення, обмінюватися повідомленнями та оплачувати покупки.

Абревіатура HTTP розшифровується як HyperText Transfer Protocol — протокол передачі гіпертексту (тобто текстових документів, які містять посилання інші документи). Нині за його допомогою передають будь-які формати даних.

## **1.2 Опис роботи HTTP**

Протокол HTTP використовує в роботі технологію клієнт-сервер: клієнт відправляє на сервер запит, де спеціальна програма його обробляє, формує відповідь і повертає клієнту.

У ролі клієнта зазвичай виступає браузер, але його функції може виконувати й інша програма. Наприклад, пошуковий робот, який подорожує сайтами і сканує їх для індексації в пошукових системах. У ролі сервера виступає веб-сервер — спеціальна програма на фізичному сервері, де зберігається сайт.

Наприклад, щоб відобразити будь-яку сторінку, ваш браузер надсилає серверу запит на отримання HTML-документа з її вмістом. Далі браузер вивчає документ і запитує додаткові файли, необхідні відображення сторінки: стилі елементів, зображення, скрипти. Потім браузер збирає все це разом, як зазначено в HTML-документі, та виводить на екран комп'ютера результат [2].

Інший приклад — коли браузер надсилає якісь дані серверу. Скажімо, логін та пароль для входу до облікового запису. Щоб сформувати HTML-документ з головною сторінкою облікового запису, сервер перевіряє деталі входу в спеціальній таблиці всередині бази даних, де зберігаються дані всіх користувачів. Якщо у таблиці є користувач із зазначеними даними, сервер формує HTML-документ і відправляє браузеру, як у прикладі вище. Якщо такого користувача немає або деталі входу неправильні, в движку сайту прописаний спеціальний сценарій на цей випадок — надіслати повідомлення з помилкою, яку браузер відобразить на екрані.

### 1.3 Протокол HTTPS

HTTPS — це розширення для протоколу HTTP, що робить його безпечним. Справа в тому, що дані передаються за HTTP у відкритому вигляді. Це створює ризик розкрити конфіденційну інформацію, якщо хтось перехопить трафік. HTTPS вирішує цю проблему, додаючи в початковий протокол можливість шифрування даних [3].

Абревіатура HTTPS розшифровується як HyperText Transfer Protocol Secure — безпечний протокол передачі гіпертексту. Безпека досягається за рахунок об'єднання протоколу HTTP з криптографічним протоколом TLS (рис. 2).

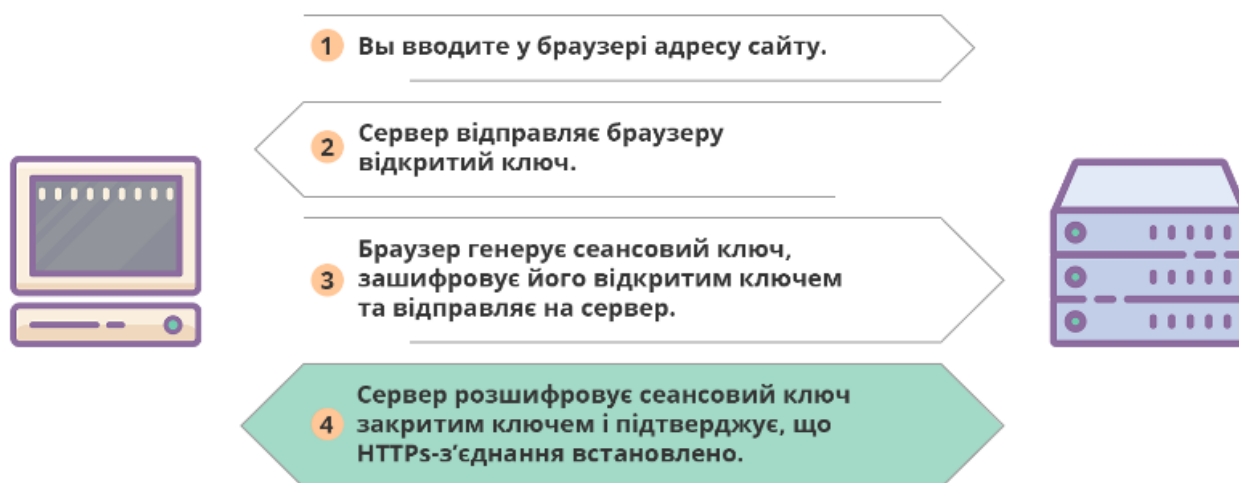


Рисунок 2 – Обмін запитамми по HTTPS

## 1.4 Детальніше про бекенд

Бекенд – це узагальнене поняття програмно-апаратного комплексу серверної частини WEB-застосунку. Простіше кажучи, це все, що працює на сервері. Виходячи з цього, бекенд розробка – це робота над програмними засобами, спрямованими на реалізацію логіки ресурсу [1]. Ця частина прихована від очей користувача, оскільки відбувається за межами його браузера або конкретно взятого комп'ютера. Розробник сайтів в даному випадку використовує ті ресурси, які є на сервері. При цьому його обов'язки можуть значно варіюватися, залежно від того про який продукт йде мова. Так, фахівець може займатися створенням, інтеграцією баз даних, забезпечувати безпеку ресурсу, налаштовувати технології резервного копіювання або ж відновлення інформації. Бекенд являє собою процес об'єднання користувача з сервером, який неможливо відстежити неозброєним поглядом [4].

## 1.5 Інструменти розробки серверної частини WEB-застосунку

Для створення серверної частини сайту необхідно освоїти повноцінну мову програмування. Вона може бути практично будь-якою, але зараз у WEB-розробці найчастіше використовують PHP, Javascript та Python [4]. Це мови загального призначення, але для створення сайтів вони підходять у більшості випадків. Для розробки серверної частини необхідно розібратися з базами даних. Найпопулярніші СУБД для розробки веб-додатків: MySQL, PostgreSQL, MongoDB, також для хмарних швидких рішень використовується платформа Google Firebase [5].

Сучасна розробка бекенду - все частіше це розробка програмного інтерфейсу (API), яким користуються як браузери так і мобільні додатки [4].

Для розробки повноцінного серверного додатку зазвичай використовуються підготовлені інструментарії з набором вже стабільних протестованих компонентів та структурою проекту - фреймворки.

Для бекенду рекомендую:

PHP: Symfony 5+, Laravel 9

NodeJS: NestJS, Express

Python: Flask, Django

## 1.6 Відмінність Frontend і Backend розробки

Основні відмінності даних понять полягають в тому, що одне з них (фронтенд) являє собою все, що бачить користувач при роботі з сайтом, а інше (бекенд), навпаки, перебуває поза полем зору людини. Це дві частини одного і того ж проекту, одного цілого, і є кілька варіантів, як вони будуть взаємодіяти один з одним [5].

Зазвичай весь процес проходить циклічно:

- Frontend збирає призначені для користувача дані і перенаправляє їх в Backend;
- відбувається обробка даних;
- інформація повертається, прийнявши зрозумілу форму і виконавши запит.

Відмінності Frontend від Backend сайту істотні, так як за кожну з названих вище задачу відповідає окремий фахівець, а успішний результат можливий тільки при взаємодоповнюючій командній роботі.

Зрозуміти особливості їх взаємодії найпростіше на прикладі. Так, оплачуючи покупку в інтернеті, ви заповнюєте дані своєї карти, натискаєте кнопку «оплатити» і отримуєте сповіщення про те, що оплата пройшла. Це чистої води фронтенд. А ось як далі гроші рухаються по мережі, як продавець отримує ваше замовлення – ви не бачите, це бекенд.

## 1.7 Application Programming Interface (API)

Абревіатура API розшифровується як Application Programming Interface або програмний інтерфейс застосунку. API – це набір способів і правил взаємодії та обміну даними між різними програмами [4]. Їх спілкування, якщо можна так висловитися, відбувається за допомогою класів, методів, функцій і структур. Іноді комунікації можливі за допомогою констант однієї програми, до яких звертаються інші. Говорячи простими словами, API виступає в ролі посередника, що дозволяє програмі 1 отримати доступ до даних і навіть деяким можливостям програми 2. Все це дозволяє розробникам поліпшити функціональність продуктів, що випускаються і пов'язати їх з іншими додатками.

Кожен раз, коли користувач відвідує будь-яку сторінку в мережі, він взаємодіє з API віддаленого сервера. API-це складова частина сервера, яка отримує запити і відправляє відповіді.

## 1.8 Чому API називають інтерфейсом

Інтерфейс – це межа між двома програмами, за допомогою якої вони можуть обмінюватися інформацією і виконувати пов’язані один з одним функції. При цьому процеси, що відбуваються всередині кожної з них, недоступні для іншої програми. Схоже визначення раніше ми дали для API. Завдяки такому підходу вдається налагодити взаємодію між декількома програмами, не переймаючись питанням про їх устрій, програмну логіку або способи обробки даних (див. ілюстрацію на рис. 3). За допомогою інтерфейсів програмістам не потрібно розбиратися в програмному коді інших фахівців, щоб підключити свій продукт до іншого. Користувачам же при цьому не потрібно замислюватися про те, що стоїть за звичними функціями в девайсах. Наприклад, для відправки повідомлення зі смартфона їм не потрібно знати принцип роботи тачскріна. Замість цього достатньо набрати потрібний текст, натиснувши певне поєднання віртуальних клавіш. API дозволяє користуватися функціями програми, не маючи уявлення про те, як вона працює. З цієї причини даний термін прирівнюють до інтерфейсу.

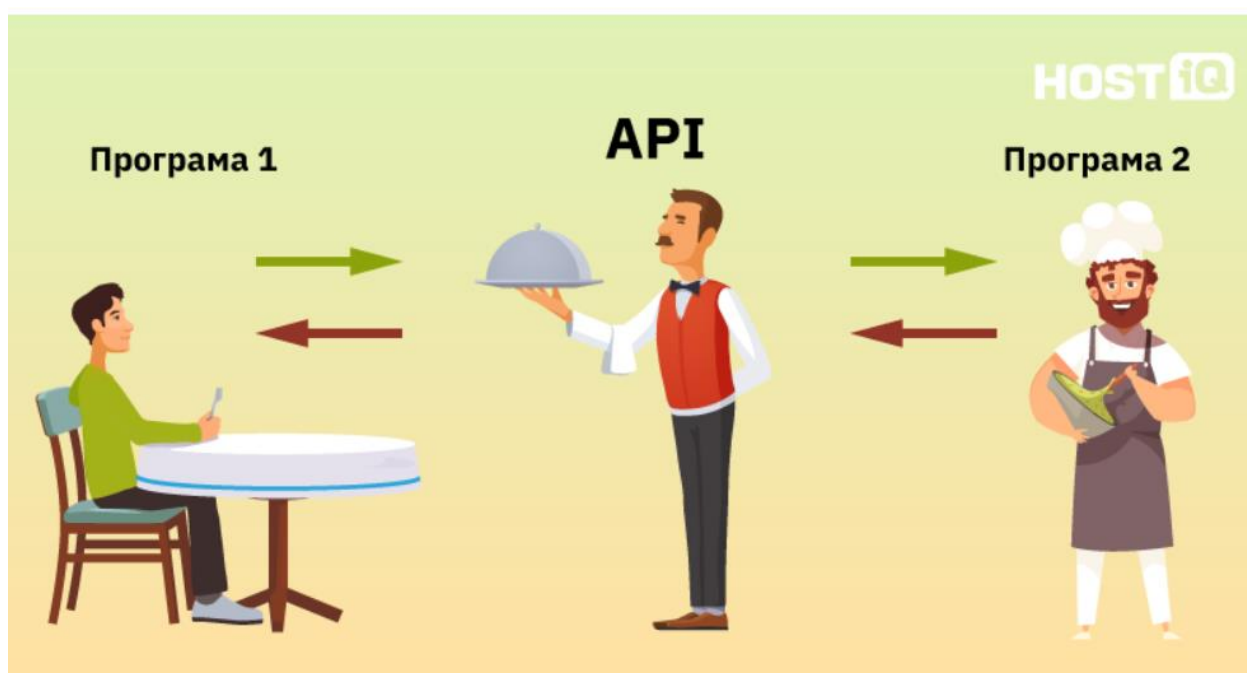


Рисунок 3 – Суть роботи API на прикладі обслуговування в ресторані

## 1.9 Призначення API

Головне завдання API полягає в тому, щоб налагодити взаємодію між певними утилітами і не замислюватися при цьому, як вони влаштовані або як вони обробляють дані.

Але у подібного інструменту є і інші цілі:

1. зробити програмне забезпечення більш безпечним;
2. заробити на тому, що інші розробники використовують функціонал готового продукту для удосконалення власних напрацювань.

Інтерес програмістів до API обумовлений наступними причинами:

1. Збільшення безпеки розробки. За допомогою API можна винести в окремий додаток той функціонал, який повинен бути захищений від стороннього втручання. Це знижує ймовірність некоректного його використання сторонніми програмами.

2. Зниження вартості розробки і економія часу. Купити готовий API іноді вигідніше, ніж витратити час на створення функціоналу з нуля з усіма наслідками, що випливають звідси наслідками. Можна користуватися вже готовими інструментами, а не винаходити велосипед з нуля.

3. Спрощення зв'язків між різними системами і сервісами. Розробники можуть впроваджувати у свої продукти підтримку сторонніх сервісів, навіть не замислюючись про те, хто їх створював.

## 1.10 API на прикладі

Розробник зробив сайт із онлайн-конвертером валют. Він хоче, щоб конвертером користувалися не лише на вихідному сайті, а й у банківських програмах.

Як це зробити:

1. Розробник конвертера створює API та викладає його на своєму сайті.

Описує в документації, як працює API та які в нього є функції.

2. Розробники банківських програм беруть API конвертера та додають його до свого коду, якщо хочуть дати своїм користувачам функції цього конвертера.

3. Користувач програми вводить суму, яку хоче «обміняти», та вибирає валюту.
4. Програма надсилає на сервер конвертера спеціальний запит із сумою, яку потрібно перевести у вибрану валюту.
5. Конвертер переводить суму та відправляє відповідь з результатом у потрібній валюті.
6. Програма отримує відповідь та показує її користувачу.

У результаті всі задоволені: користувач переклав цифри в іншу валюту в додатку, конвертером користуються не лише на сайті, а банківські програми не писали конвертер з нуля, але покращили свій продукт.

### **1.11 REST API**

REST або Representational state transfer, що перекладається як передача стану подання – це архітектурний стиль проектування API з використанням протоколу HTTP [5]. Головна перевага REST – велика гнучкість.

REST API застосовується скрізь, де є необхідність надання даних із сервера користувачеві веб-програми або сайту.

Головними компонентами REST API є:

Client – клієнт або програма, яка запущена на стороні користувача (на його девайсі) та ініціює комунікацію.

Server — сервер, який надає API як доступ до своїх даних та функцій.

Resource - ресурс є будь-яким видом контенту (відео, текст, картинка), який сервер може передати клієнту.

### **1.12 Опис роботи REST API**

REST API взаємодіє з допомогою HTTP запитів, виконуючи стандартні функції: створення, оновлення, читання, видалення записів у ресурсі. Існує чотири методи, що описують, що потрібно робити з ресурсом [1]:

POST - створення ресурсу;

GET – отримання ресурсу;

PUT – оновлення ресурсу;

DELETE – видалення ресурсу.

### 1.13 Файли-куки

Файли-куки (з англійської cookie перекладається як печиво) – це невеликі текстові файли, в яких зберігається інформація про наші дії на певних ресурсах [6].

Файли cookie корисні і для самого веб-ресурсу, і для користувача, і можуть виконувати такі функції:

- Збір даних про переваги власника пристрою. Cookie дозволяють запропонувати йому тільки рекламу або сторінки ресурсу, які будуть затребувані.
- Збереження налаштувань користувача та даних для входу на сайт. Це позбавляє нас необхідності щоразу заново вводити цю інформацію.
- Прискорення серфінгу на сторінках веб-ресурсу.
- Збір даних про затребуваність запропонованого продукту – рекламного чи інформаційного.

Механізм дії cookie дуже простий. Будь-яка наша дія на сайті фіксується та перенаправляється браузеру у вигляді куки, завдяки чому ресурс нас ідентифікує. Після того, як ми залишаємо сторінку, файли відправляються назад.

*Яку інформацію передають куки-файли.*

Веб-ресурси за допомогою cookies збирають дані, які допоможуть їм краще просувати свої продукти. Це можуть бути [6]:

- Технічні параметри, які вибирає користувач, – масштаб, шрифт, валюта, мова.
- Товарні переваги – дані про переглянуту рекламу чи куплену продукцію.
- Особисті дані, розташування та IP-пристрої для ідентифікації користувача.
- Лайки та коментарі.
- Тип ОС.

Всі ці файли аналізуються та роблять пошук інформації більш комфортним. Тому, якщо цей сайт використовує cookies, це не означає, що він намагається вивідати особисті дані та нашкодити вам.



## 1.14 Сесії

Особливість веб-серверу полягає в тому, що він не здатний розпізнати, чи надходять запити від одного і того ж браузера, чи від різних, оскільки HTTP протокол не дозволяє відстежувати ці стани та підтримувати постійний зв'язок з клієнтом. Кожен новий запит обробляється окремо, без прив'язки до попередніх. Справитися з цією проблемою допомагає сесія браузера (сеанс) – механізм, який дозволяє відстежити запити від одного браузера та зберегти деякі зміни під час переходів по сторінках сайту.

З початком сесії на стороні сервера створюється файл, який містить інформацію про користувача, про його дії та події, що відбулися в рамках одного сеансу [7]. Такими подіями можуть бути перегляд сторінок сайту, різні взаємодії користувача з елементами сторінки, вчинення транзакцій і т. ін.

До тих пір, поки попередня сесія активна, нова не може бути розпочата. Старий сеанс може завершитися при дотриманні однієї з умов (в залежності від налаштувань):

- після закінчення певної години бездіяльності (таймаут);
- у певний час доби (опівночі, наприклад);
- при закритті браузера

## 1.15 Питання для самоконтролю

1. Які переваги архітектури клієнт-сервер перед програмами, що безпосередньо встановлюються на комп'ютері користувача?
2. Яка програма на комп'ютері користувача зазвичай виступає в ролі клієнта?
3. Що таке API?
4. Назвіть найбільш розповсюджені WEB-сервери.
5. В чому полягає суть технологія активних серверних сторінок?
6. Опишіть роботу протоколу HTTPs.
7. Які програми відносяться до серверної частини WEB-застосунку?
8. Що таке Frontend і Backend?
9. Для чого слугують Файли-куки?

## РОЗДІЛ 2 ТИПИ ВЕБ-САЙТІВ

### 2.1 Статичні сайти на зорі Інтернету

На зорі Інтернету були лише статичні сторінки, нічого динамічно не генерувалося. Звичайні, заздалегідь створені, статичні HTML документи надсилалися клієнту.

Коли користувач заходив на веб-сайт, простий HTTP запит йшов на сервер і далі він відповідав розміткою, яка відображалася в браузері.

Потім з'явилася можливість використовувати динамічний рендеринг і будувати шаблони для розмітки, наприклад PHP.

Ці шаблони дозволяли заповнювати інформацією, що надсилається клієнту, HTML. Кожен запит HTTP проходив через серверну частину веб-сайту і збирав необхідні дані. Наприклад, завдяки цьому з'явилася можливість додавати такі елементи як імена користувачів, поточні дати, дані з бази даних та інші.

Це і був початковий сервер Side Rendering. Клієнт (браузер) запитував у сервера, необхідний для відображення, HTML, який був згенерований саме на сервері і далі відображався у браузері.

Дамо визначення основним термінам:

- **SPA** – Single Page Application . Односторінкова програма – це веб-додаток або веб-сайт, що використовує єдиний HTML-документ як оболонку для всіх веб-сторінок і організує взаємодію з користувачем через HTML, CSS, JavaScript, що динамічно підвантажуються, зазвичай за допомогою AJAX .
- **CSR** – Client Side Rendering. Рендеринг на клієнті – це рендеринг програми у браузері за допомогою DOM (Document Object Model).
- **SSR** – Server Side Rendering . Рендеринг на сервері – це рендеринг клієнтської частини програми на сервері.
- **SSG** – Static Site Generator . Статична генерація сайтів – це генерація всіх HTML сторінок програми в момент збирання.

## 2.2 Технологія AJAX

З появою такої технології, як AJAX (Asynchronous JavaScript and XML), з'явилася можливість отримувати дані асинхронно, без перезавантаження цілої сторінки.

З точки зору UX , це було величезним поліпшенням. Зменшилася кількість перезавантажень сторінок, тим самим почався рух в бік Client Side Rendering .

Для спрощення роботи з фронтендом стали з'являтися різні бібліотеки і фреймворки. Однією з найпопулярніших бібліотек стала JQuery , але вона ще не була повноцінним інструментом для CSR.

## 2.3 SPA

Потім на зміну прийшли React, Angular і Vue , завдяки яким переважна більшість розробників познайомилося з повноцінним рендерингом на клієнті. Ці три фреймворки використовують компонентний підхід і дозволяють розбивати розмітку на дрібні частини, що перевикористовуються, тепер вся генерація HTML відбувається саме на фронтенді. Бекенд використовується тільки для складних обчислень, роботи з базами даних, насичення фронтенду даними та багато іншого.

Завдяки можливості легко генерувати розмітку на стороні клієнта, SPA завоювали величезну популярність.

Якщо трохи заглибитися в деталі, то тепер сервер став відправляти клієнту практично порожній HTML , який містить лише базову розмітку, з яким-небудь порожнім тегом, наприклад:

```
<div> class = ' root '> </div>
```

і єдиний файл зі скриптом, наприклад:

```
< script src = ' bundle . js '></ script >
```

всередині якого буде написано весь код для генерації розмітки, стилів та логіки за допомогою JavaScript.

На жаль, такий підхід також призвів до цілої низки потенційних проблем.

По-перше, виникла проблема з пошуковою оптимізацією. Оскільки пошукові роботи Google читають і індексують веб-сайти, необхідно віддавати

інформацію про контент і розмітку з сервера, а при такому підході все генерується на клієнті. Тоді ще пошукові системи не були здатні обробити інформацію згенеровану таким чином. Все, що міг побачити робот це порожній кореневий HTML -тег.

Наразі ситуація дещо покращилася, оскільки багато пошукових роботів навчилися виконувати, необхідний для Client Side Рендерингу, Javascript. Проте результат такої індексації все одно бажає кращого.

По-друге, потенційною проблемою є продуктивність. Оскільки для відображення сторінки в браузері потрібно виконання великої кількості JavaScript , то програма може неабияк "гальмувати". Особливо це помітно на старих мобільних пристроях.

Для вирішення цих проблем розробники знову повернулися до ідеї створення розмітки на сервері.

## **2.4 Повернення до Server Side Rendering**

Однак, на відміну від старого підходу, коли розмітка генерувалася на сервері за допомогою серверних мов програмування, наприклад PHP , тепер у Server Side Rendering , як і в CSR , будуть використовуватися сучасні JavaScript бібліотеки, наприклад React .

Різниця тільки в тому, що програма генеруватиме розмітку за допомогою React не на клієнтській стороні, а на серверній.

Це рішення було спрямоване на виправлення існуючих проблем з продуктивністю та SEO (Search Engine Optimization). Щоразу, коли пошуковий робот запитує сторінку, він отримає необхідний контент – для візуалізації більше не потрібно виконання JavaScript на клієнті.

Тепер, коли у розробці згадують термін Server Side Rendering , то посилаються саме на цей сценарій.

Більше детально про Server Side Rendering (SSR).

Розглянемо плюси та мінуси SSR.

Плюси:

- SEO . Можливість настроїти пошукову оптимізацію.

- **First Contentful Paint.** Завдяки **Server Side Rendering** , сторінки вашого сайту швидше стають доступними для взаємодії. Маючи продуктивний сервер, **SSR** може дати вам чудову оцінку **First Contentful Paint** , Що покращує **UX** і, можливо, також **SEO** сторінки.

Насправді це всі плюси, тепер розглянемо мінуси:

- Уповільнюється час переходу між сторінками. Так, для початкового рендеру **SSR** швидше, ніж **CSR** , але далі при роботі з додатком вам доводиться малювати дані двічі, один раз на сервері та один раз на клієнті.
- Більш складна технологія. Написати програму тільки на **React** у зв'язці, наприклад, з **Redux** значно простіше, ніж додати до цього стеку ще й технологію для **SSR** . Відповідно збільшується кількість коду, складність розробки та з'являються додаткові бібліотеки.
- Кешування. Для часткового закриття першого мінуса та на додаток до другого, додається більш складне кешування даних.
- Вартість. На відміну від **CSR** тепер нам **100%** потрібен сервер, та ще й більш продуктивний, це коштує дорожче.

Виявляється, у **SSR** величезна кількість мінусів, то що робити? Розглянемо ще один підхід – **SSG**.

## 2.5 Static Site Generator

**SSG** – це новий підхід для веб-розробки, який дозволяє створювати веб-сайти, сторінки яких генеруються в статичні файли в момент складання. Тим самим, коли браузер робить запит, сервер не генерує розмітку, як у випадку **SSR**, а віддає вже готову.

Різниця в порівнянні з класичним **SSR** в тому, що замість використання **HTML** і **CSS** тепер використовуються сучасні інструменти, такі як **React** , **Vue** і **Angular** . Тобто додаток, написаний на сучасних інструментах, перетворюється на **HTML** , **CSS** і **JavaScript** за допомогою транспіляторів , збирачів пакетів і т.д.

Проблема в тому, що **SSG** не передбачає динамічну роботу з сервером, тобто ви не можете створити, наприклад, кошик інтернет-магазину за допомогою **SSG** , т.к. інформація, що відображається в кошику, повинна бути унікальна для

кожного користувача. У такому випадку ми можемо застосувати лише SSR або CSR. А статична генерація сайтів зручна, наприклад, для сторінки "Доставка" або картки товару, тобто. для інформації, яка ідентична кожному за користувача.

Щоб детальніше зрозуміти інструмент, розглянемо його плюси та мінуси.

Плюси :

- SEO. Як і у випадку з SSR , пошуковий робот отримує всю необхідну інформацію.
- Продуктивність. SSG має найбільшу продуктивність проти SSR і CSR , т.к. всі дані вже згенеровані під час складання.
- Дуже дешево. Використовуючи статичну генерацію сайтів, вам не потрібний сервер, достатньо завантажити код свого сайту на github і зібрати його, наприклад, за допомогою Netlify. У такому разі вам доведеться заплатити лише за доменне ім'я.
- Простота розробки. Якщо порівнювати з CSR, то SSG дещо складніше, т.к. потрібно розібратися в тонкощах, але загалом картинка слабо відрізняється від звичайної розробки.
- Безпека. У вашого статичного сайту немає сервера, а значить у зловмисників немає можливості отримати доступ до вашої бази даних або панелі адміністратора.

Мінуси:

- Немає панелі адміністратора. Плюс, як відсутність уразливостей, але мінус, тому що всі зміни на сайті вам доведеться робити у редакторі коду, потім робити пуш у репозиторій, а далі збирати та деплоїти ваш сайт.
- SSG запустити складніше, ніж Wordpress або іншу CMS. При початковому налаштуванні більшість CMS надає величезну кількість підказок для розгортання сайту. У разі роботи зі статичним генератором сайтів вам доведеться вивчати документацію.
- Відображення лише статичних даних. Ви не можете відобразити динамічні дані за допомогою SSG , відповідно у вас немає можливості зробити досить складну веб-додаток.

Добре, SSG звучить частково краще ніж SSR, але як тоді розробити складне веб-додаток з динамічними даними, використовуючи сучасні інструменти і не втративши в SEO ?

### **SSG + SSR як універсальний рендеринг**

На щастя, є метод під назвою універсальний рендеринг, за допомогою якого ви можете отримати найкращий з усіх підходів:

- Швидкі та плавні переходи між сторінками, як у CSR .
- Можливість роботи з SEO .
- Неймовірно швидкий First Contentful Paint .
- Роботу з динамічними даними .
- Зменшення витрат, пов'язаних із вмістом сервера.

Всі ці можливості надає такий фреймворк, як Next.JS.

## **2.6 Питання для самоконтролю**

1. Чим відрізняються підходи клієнтського рендерингу (CSR) та серверного рендерингу (SSR) сторінок?
2. Які JavaScript фреймворки забезпечують повноцінний CSR?
3. Які переваги і недоліки SSR?
4. Які переваги і недоліки CSR?
5. В чому відмінність SSG від SSR?

## РОЗДІЛ 3 ОГЛЯД МОВ ДЛЯ СЕРВЕРНИХ ЗАСТОСУНКІВ

### 3.1 Типи мов програмування

Мови програмування постійно розвиваються – коли одна відмирає, на зміну їй приходять нові. Якщо розвиватися в тій мові яка подобається, то звісно можна заробити багато грошей. Але краще покладатися на перевірені мови програмування, які зарекомендували себе протягом тривалого часу та які користуються великим попитом [7].

Кожна мова програмування відрізняється від інших. Хоча кожна мова програмування має свій унікальний синтаксис, спосіб її написання, виконання та компіляції може змінити все.

Багато розробників віддають перевагу роботі з певними типами мов. Також набагато легше перемикатися між схожими мовами, тому перша мова програмування, яку ви вивчаєте, має велике значення.

Мови програмування можуть бути практично будь-якими, але вони часто найкраще підходять для розробки програмного забезпечення, оскільки їх можна використовувати на різних платформах вони поєднуються. Більшість мов програмування призначені для розробки програмного забезпечення, різних програм, які ви завантажуйте і запускаєте на своїх пристроях.

Скриптові мови часто інтерпретуються, що означає, що їх код виконується на льоту, а не проходить процес компіляції в програмі. Такі мови веброботи часто є мовами сценаріїв.

Мови розмітки насправді не є мовами програмування, але вони використовуються для веброботи. Це читабельні теги, які використовуються для форматування документа.

Мови програмування спеціалізуються на створенні вебсторінок, як зовнішніх, так і внутрішніх.

Інтерфейсні (або клієнтські) мови модифікують вебсторінку в браузері користувача. Наприклад, якщо ви натискаєте на щось на вебсторінці та створюєте анімацію, це буде зроблено за допомогою інтерфейсного програмування, такого як CSS, HTML5 і JavaScript.



Внутрішні (або серверні) мови модифікують вебсторінку на рівні сервера або програми. Наприклад, коли ви відправляєте дані з форми або змінюєте щось у базі даних – це внутрішнє програмування.

Слід також зазначити, що багато людей просто використовують мови програмування для позначення всіх мов програмування в цілому. Мова сценаріїв – це спеціалізована мова програмування, але не всі мови програмування є мовами сценаріїв [7].

## 3.2 HTML та CSS

Хоча HTML і CSS технічно не є мовами програмування, сьогоднішні їхні більш довершені версії HTML5 і CSS3 є ідеальними відповідними точками, якщо ви хочете стати розробником сайтів чи вебінтерфейсів.

Поєднання цих двох (мов розмітки гіпертексту) формує блоки будь-якого вебсайту: HTML – це структурує та вміст сторінки, а CSS – це дизайнерське стилізування та модифікація структуру вебсторінки.

HTML та CSS – це чудові, можна навіть сказати базово необхідні початкові знання для будь-якого WEB розробника. У той час як HTML є легкою для вивчення мовою розмітки, CSS є більш складною, але також не важкою для вивчення [7].

Застосування: інтерфейсна веброзробка.

Переваги:

- Дуже легко освоїти, навіть для людей без досвіду програмування.
- Його висока популярність дозволяє легко знайти безкоштовні ресурси.
- Добре підтримується на всіх пристроях.

Недоліки:

- Ви не отримуєте особливо високої зарплати, тому що це обов'язкова умова для всіх вакансій веброзробників.
- Існують незначні, але все ж проблеми з кросбраузерністю.

### 3.3 Python

Сплеск популярності Python, здавалося, виник нізвідки, але вона охопила майже всі сфери розробки.

Зараз це друга за популярністю мова програмування на GitHub (після JavaScript).

Python може все: від серверної частини до програмного забезпечення для машинного навчання.

У ній є майже все, що можна побажати від мови програмування: універсальність, швидкість та ефективність. Крім того, вона надзвичайно легка у вивченні в порівнянні з іншими мовами програмування [7].

Застосування: WEB і розробка програмного забезпечення.

Переваги:

- Її можна використовувати майже будь-де, від веб-додатків до розробки програмного забезпечення та розробки ігор.
- Кросплатформенність.
- Висока популярність означає багато ресурсів і ще більше робочих місць.

Недоліки:

- Повільніша, ніж інші мови.
- З Python складніше перейти на інші мови.

### 3.4 JavaScript

Незважаючи на те, що HTML і CSS є обов'язковими для інтерфейсних веброзробників, JavaScript надзвичайно популярний. У той час як структури HTML і стилі CSS, JavaScript додає до вебсторінки розширену функціональність на стороні клієнта. На відміну від HTML і CSS, JavaScript є реальною мовою програмування та сценаріїв [7].

Застосування: мова програмування сценаріїв WEB-сторінок. Рідко використовується для розробки мобільного чи програмного забезпечення.

Переваги:

- Найшвидший і найпростіший спосіб програмування клієнтських сценаріїв, які виконуються у браузері.

- Дуже популярна мова програмування.
- Широка підтримка різноманітних програм.
- Вона практично працює на більшості сучасних вебсайтів.

Недоліки:

- Для тих, хто знає лише мови розмітки, навчання може бути важким.
- Має проблеми з безпекою та кросбраузерною.

### 3.5 PHP

PHP є мовою великих проєктів, наприклад найпопулярнішої системи управління сайтом як WordPress – мабуть, напевно всі чули про неї. Колись вона розділила спільноту розробників, оскільки була застарілою та повільною, і багато хто досі вважає, що її вивчення не варте часу.

Однак PHP повернулася в моду після PHP 5.x із рядом покращень швидкості та структури. За даними W3Techs, 79% перевірених вебсайтів використовують PHP.

Ясно одне: PHP це потужний спосіб програмування програм на стороні сервера, і її легко освоїти порівняно з іншими мовами сценаріїв.

Її популярність серед програмістів-початківців і велика кількість проєктів з відкритим кодом, таких як WordPress, означають, що тут також є багато навчальних ресурсів.

Існують десятки популярних фреймворків PHP, які можуть зробити вашу роботу з PHP ще простішою. З появою PHP 8.0, PHP намагається еволюціонувати від чистої серверної мови сценаріїв до мови програмування загального призначення [7].

Застосування: серверний веб-скрипт.

Переваги:

- Дуже легко навчитися.
- Добре зарекомендувала себе у веб-розробці та часто зустрічається на вебсайтах.
- Сучасні версії відносно швидкі.
- Легко знайти роботу розробника PHP.

Недоліки:

- Популярність падає порівняно з такими новими гарячими мовами, як Python і JavaScript.

### **3.6 Java**

Java виглядає як пращур усіх мов програмування, але насправді вона навіть не така стара, як C++.

Навіть якщо багато хто вважає її застарілою, вона все ще використовується в усьому світі та на всіх видах пристроїв.

Python ось-ось випередить її і загалом Java впаде в популярності, але Java точно не вмерла. Є тисячі вакансій розробників серверної частини Java, і попит залишається високим, що робить її надійним вибором.

Незважаючи на те, що Java є старішою мовою, популярність якої впала з роками, Java залишається головним претендентом на популярність [7].

Застосування: розробка програмного забезпечення/додатків, веб і мобільних пристроїв.

Переваги:

- Кросплатформенна та універсальна.
- Давня але ще популярна, незважаючи на свій вік.
- Більша безпека.

Недоліки:

- Важко вчитися.
- Популярність має тенденцію до спаду.

### **3.7 C#**

C# відома як сучасна, більш універсальна версія мови C++. З одного боку, C# набагато легше вивчити. Вона простіша і менш складна, її можна використовувати для створення різноманітних програм. Вона також набагато краще підходить для веб-розробки, ніж C++. Вона досить популярна для розробки ігор і знаходиться в середині списку найбільш високооплачуваних мов.

Що з цих двох обрати? Це залежить від того які завдання потрібно вирішувати. С++ краща, коли вам потрібна сира потужність. З С# простіша і легша працювати, як універсальне рішення [7].

Застосування: переважно розробка програмного забезпечення.

Переваги:

- Універсальна.

Недоліки:

- Відносно складна для вивчення.

### **3.8 Питання для самоконтролю**

1. Яка мова, на вашу думку, є найкращою для початківця в WEB-розробці?
2. Які мови застосовуються у frontend-розробці?
3. Які мови застосовуються у backend-розробці?
4. Які мови, на вашу думку, є взаємозамінними, частково або повністю?

## РОЗДІЛ 4 СЕРВІСНА АРХІТЕКТУРА

### 4.1 Основні принципи мікросервісної архітектури

**Мікросервісна архітектура** – це метод побудови програмного забезпечення, який передбачає розділення додатку на декілька незалежних, спеціалізованих сервісів. Цей підхід протиставляється традиційній монолітній архітектурі, де вся функціональність додатка знаходиться в одному великому блоку коду. Цей розділ розглядає основи побудови сучасного веб-додатку на базі мікросервісів та проводить порівняння сервісного та монолітного підходу (рис. 4).

Мікросервіси – це невеликі, автономні сервіси, які виконують специфічні функції додатка та спілкуються між собою за допомогою API (Application Programming Interface). Основні принципи мікросервісної архітектури включають:

- відповідальність за одну річ: кожен сервіс відповідає за одну функцію та забезпечує стабільність її виконання;
- гнучкість: сервіси можуть бути розроблені, розгорнуті та масштабовані незалежно один від одного;
- міжсервісна комунікація: сервіси спілкуються між собою за допомогою стандартизованих протоколів, таких як REST, gRPC або GraphQL.

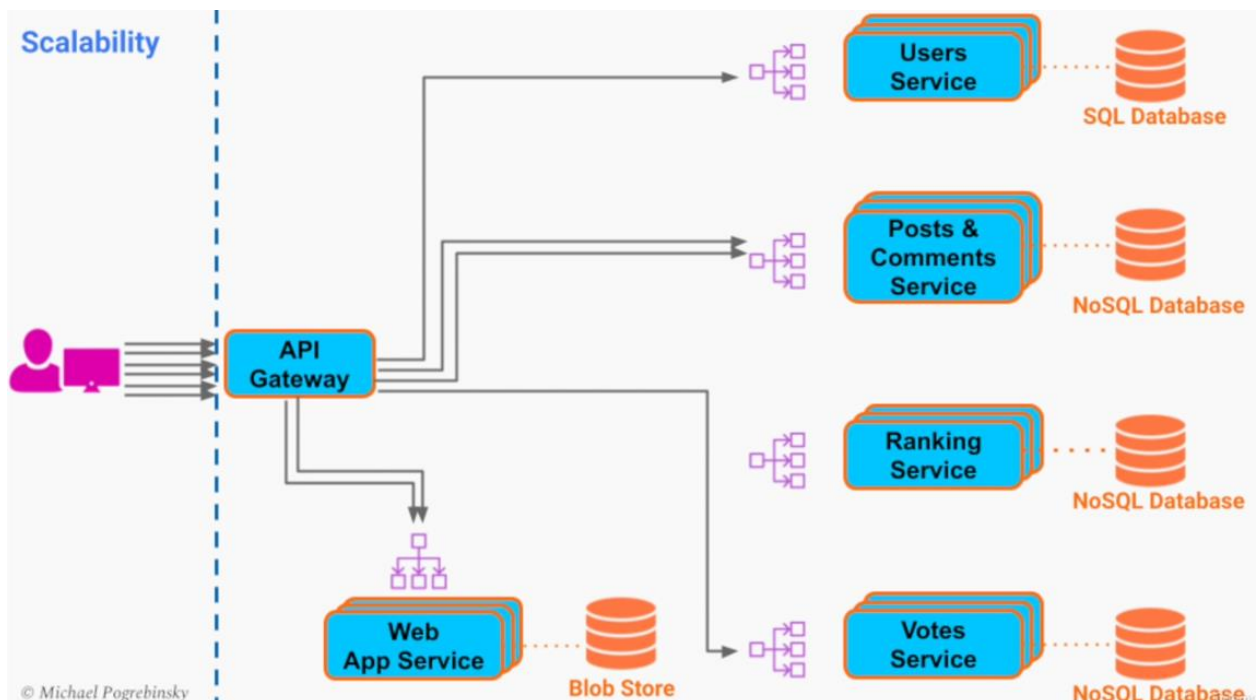


Рисунок 4 – Приклад мікросервісної архітектури застосунку

## 4.2 Переваги мікросервісної архітектури

Мікросервісна архітектура має декілька переваг порівняно з монолітним підходом:

- **Модульність:** кожен сервіс може розроблятися, тестуватися та розгортатися незалежно від інших сервісів. Це полегшує відлагодження, розвиток та розширення функціональності додатка.
- **Масштабованість:** сервіси можуть масштабуватися горизонтально, додаючи ресурси для окремих сервісів за потребами. Це дозволяє оптимізувати використання ресурсів та підвищити продуктивність системи.
- **Стійкість до відмов:** у разі відмови одного сервісу, інші можуть продовжувати функціонувати, забезпечуючи певний рівень стійкості системи.
- **Технологічна гнучкість:** різні сервіси можуть використовувати різні технології, мови програмування та бази даних, що дозволяє відібрати найбільш підходящі інструменти для кожного з них.

## 4.3 Монолітна архітектура

Монолітний підхід передбачає, що весь код додатка розміщується в одному блоку та ділить спільні ресурси (рис. 5). Основні характеристики монолітної архітектури:

- **Єдиний кодовий базис:** усі компоненти додатка зберігаються в одному проекті, що може ускладнювати розробку, тестування та розгортання.
- **Відсутність модульності:** компоненти додатка тісно пов'язані, що може ускладнити відлагодження та модифікацію коду.
- **Вертикальна масштабованість:** монолітні додатки можуть масштабуватися лише за допомогою додавання ресурсів до сервера, на якому вони розгорнуті.

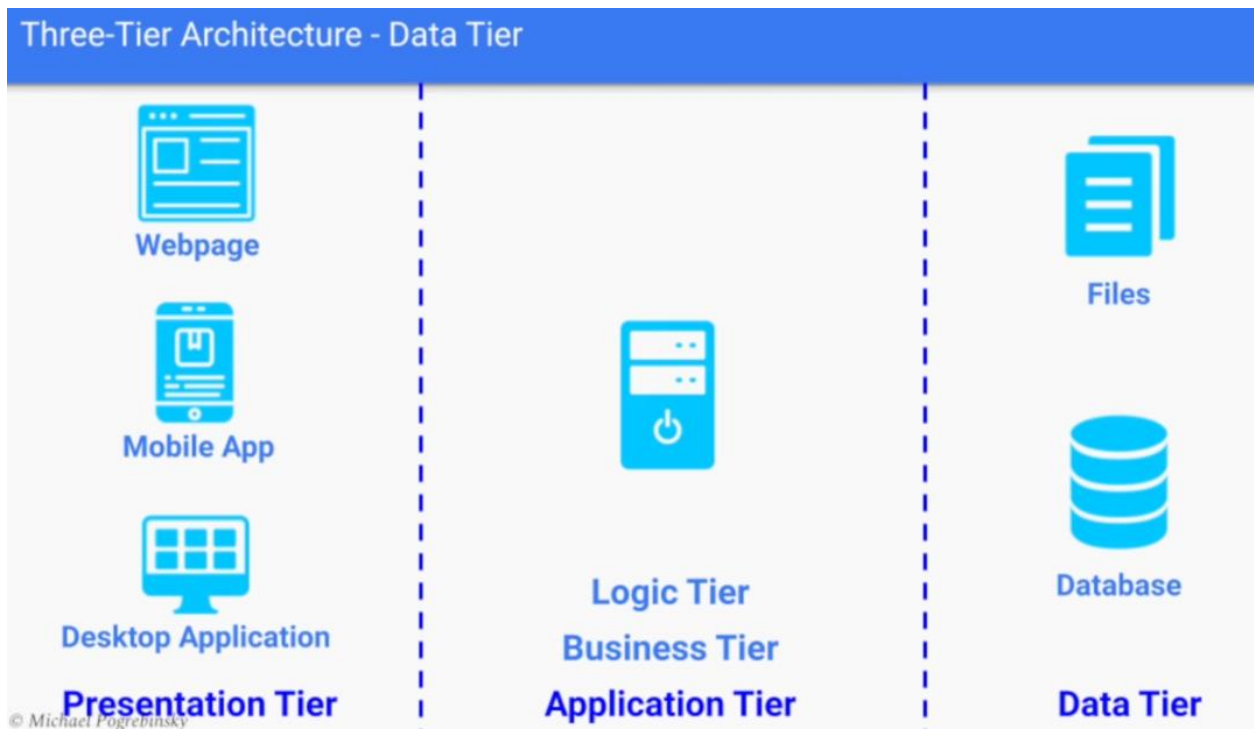


Рисунок 5 –Приклад монолітної архітектури застосунку

#### 4.4 Переваги монолітної архітектури

Незважаючи на обмеження, монолітна архітектура має деякі переваги:

- Простота розробки: монолітні додатки часто простіше розробляти та розгортати, оскільки вони мають єдиний кодовий базис та спільні ресурси.
- Простота комунікації між компонентами: у монолітних додатках компоненти спілкуються безпосередньо, що може прискорити обмін даними та зменшити складність взаємодії між ними.
- Менші витрати на інфраструктуру: монолітні додатки можуть вимагати менше ресурсів для розгортання, особливо на початкових етапах розвитку проекту.

#### 4.5 Порівняння сервісного та монолітного підходів

Вибір між мікросервісною та монолітною архітектурою залежить від потреб конкретного проекту. Мікросервіси можуть бути вигіднішими для великих, складних додатків з великою кількістю функціональності та потребами в масштабованості. Монолітна архітектура може бути підходящою для невеликих проектів з простою структурою та меншими вимогами до масштабування.



## **4.6 Перехід від моноліту до мікросервісів**

Перехід від монолітної архітектури до мікросервісної може бути виконаний поетапно. Це може включати в себе винесення окремих функцій або компонентів монолітного додатка в окремі сервіси, реорганізацію коду та впровадження API для комунікації між сервісами.

## **4.7 Підсумки до розділу**

Сервісна архітектура, зокрема мікросервісна архітектура, пропонує ряд переваг у порівнянні з монолітною архітектурою, зокрема модульність, масштабованість, стійкість до відмов та технологічну гнучкість. Однак монолітна архітектура може бути підходящою для менших проектів з простою структурою та меншими вимогами до масштабування.

При виборі архітектури для вашого проекту рекомендується оцінити потреби та масштаби проекту, складність функціональності, можливість розвитку та масштабування. Мікросервісна архітектура може виявитися вибором для великих та складних проектів, а монолітна архітектура – для невеликих та простих додатків.

У випадку потреби в переході від моноліту до мікросервісів, такий процес може бути проведений поетапно, з врахуванням аналізу поточної структури додатка та визначення стратегії переходу.

Загалом, вибір між мікросервісною та монолітною архітектурою залежить від специфіки кожного проекту, його розвитку та технічних вимог. Важливо ретельно аналізувати потреби проекту та враховувати переваги та недоліки кожного підходу, щоб прийняти оптимальне рішення.

## **4.8 Питання для самоконтролю**

1. Що таке сервісна архітектура, та які основні принципи її роботи?
2. У чому полягають переваги мікросервісної архітектури порівняно з монолітною архітектурою?
3. Які основні характеристики та переваги монолітної архітектури?

4. Які фактори слід враховувати при виборі між мікросервісною та монолітною архітектурою для конкретного проекту?

5. Які основні етапи переходу від монолітної архітектури до мікросервісної, та які аспекти слід враховувати під час такого переходу?

## РОЗДІЛ 5 ОСНОВИ МОВИ PHP

PHP — це скриптова мова програмування, створена для генерації HTML-сторінок на стороні вебсервера. Спочатку PHP розшифровувалася як **Personal Home Page**, але нині її офіційна назва **PHP: Hypertext Preprocessor**.

PHP є однією з найпоширеніших мов, яку використовують у сфері веброботи, її підтримує більшість хостинг-провайдерів. PHP інтерпретує вебсервер у HTML-код, який передається на сторону клієнта. На відміну від JavaScript, користувач не бачить PHP-коду, тому що браузер отримує готовий HTML-код. Це є перевагою з погляду безпеки, але погіршує інтерактивність сторінок [8].

Активна сторінка PHP є HTML-сторінкою, в текст якої вставлені інструкції мови PHP. Інструкції знаходяться всередині обмежувачів `<?php... ?>`. Файли сторінок PHP, зазвичай, мають розширення `php`, `phtm` або `phtml`. Під час оброблення сторінки, що містить код PHP, лексичного аналізу й інтерпретації піддається тільки вміст конструкції `<?php... ?>`. Решта всього текстового змісту (код розмітки) прямує у вихідний потік без зміни.

### 5.1 Створення змінної

Власне, код PHP складається з набору інструкцій. PHP створений не тільки для форматування статичного тексту. Для того, щоб обробляти різні дані були придумані змінні. Змінна - контейнер с даними. Кожна змінна містить певне значення.

**Змінні** зберігають окремі значення, які можна використовувати у виразах PHP. Для визначення змінних застосовується знак долара.

Синтаксис змінної складається з знака долара - \$ і "вільного" ідентифікатора якому присвоюється якесь значення [8].

Змінна створюється тоді, коли їй присвоюють якесь значення. Для присвоєння значення змінної використовують оператор присвоєння, який складається з знака рівності (Змінну можна вивести на екран за допомогою оператора `echo`).

Як правило, назви змінних починаються з маленької літери або символу підкреслення. Варто враховувати, що PHP є регістрозалежною мовою, а значить, змінні \$num і \$Num будуть представляти дві різні змінні.

Також при іменуванні змінних нам треба враховувати такі правила:

1) Імена змінних повинні починатися з алфавітного символу або підкреслення.

2) Імена змінних можуть містити лише символи: a–z, A–Z, 0–9, та знак підкреслення.

3) Імена змінних не повинні включати прогалини.

Після визначення змінної та надання їй значення ми можемо використовувати її у виразах PHP (наприклад, вивести значення на веб сторінку).

## 5.2 Константи в PHP

Коли не потрібно міняти задане значення для змінної, то має сенс створити константу і потім використовувати її в будь-якій частині скрипта. Для опису константи використовують функцію `define`, якій передається її ім'я і значення [8]. Константи, як і змінні, зберігають певне значення, однак на відміну від змінних константа може набути значення лише один раз, і далі ми можемо його змінити. Константи зазвичай визначаються для зберігання значень, які мають залишатися незмінними протягом усієї роботи скрипта.

Для визначення константи застосовується оператор `const`, у своїй назві константи знак долара \$ (на відміну змінних) не використовується. Зазвичай назви констант використовують великі символи, але це умовність. Після визначення константи ми можемо її використовувати так само, як і звичайну змінну. PHP дозволяє встановлювати значення констант на основі обчислюваних виразів.

Також для визначення константи може застосовуватися функція `define()`. Параметр `$name` передає назву константи, а параметр `$value` – її значення. Значення константи може бути типом `int`, `float`, `string`, `bool`, `null` або масивом.

*Магічні константи.*

Крім створюваних програмістом констант у PHP є ще кілька так званих "магічних" констант, які є частиною мови за замовчуванням [8] (див. таблицю 1):

Таблиця 1 – Деякі константи PHP

Назва константи	Опис
<code>__FILE__</code>	зберігає повний шлях та ім'я поточного файлу
<code>__LINE__</code>	зберігає поточний номер рядка, який обробляє інтерпретатор
<code>__DIR__</code>	зберігає каталог поточного файлу
<code>__FUNCTION__</code>	назва оброблюваної функції
<code>__CLASS__</code>	назва поточного класу
<code>__TRAIT__</code>	назва поточного трейту
<code>__METHOD__</code>	назва оброблюваного методу
<code>__NAMESPACE__</code>	назва поточного простору імен

Щоб перевірити, чи визначено константу, ми можемо використовувати функцію `bool defined (string $name)`. Якщо константа `$name` визначена, то функція повертатиме значення `true`.

### 5.3 Функції в PHP

Функція — це блок коду, який може бути іменований і викликаний повторно. Іноді функцію називають підпрограмою. Ми звикли, що звичайній змінній можна присвоїти число, рядок або масив, а потім отримати його назад, звернувшись до значення на ім'я змінної.

Функції є блоком інструкцій, які багаторазово можна викликати в різних частинах програми. Функції дозволяють розділяти програму на менші функціональні частини. Визначення функції починається з ключового слова `function`, за яким слідує ім'я функції [8]. Ім'я функції має починатися з алфавітного символу або підкреслення, за якими може йти будь-яка кількість алфавітно-цифрових символів або символів підкреслення. Після імені функції у дужках йде перелік параметрів. Навіть якщо параметрів функції немає, то просто

йдуть порожні дужки. Потім у фігурних дужках йде тіло функції, що містить набір інструкцій.

Функції бувають вбудовані та користувальницькі. *Вбудовані функції* за вас вже написали автори мови, і ви можете просто брати їх і використовувати. У PHP існують тисячі готових функцій на всі випадки життя, наприклад, `sort()` для сортування масивів, `print()` для виведення рядків на екран або функції для роботи з базами даних.

*Користувальницькі функції* програмісти пишуть самі — наприклад, щоб перевірити дані пасажирів за номером авіаквитка або визначити, чи зараз високосний рік. Ці функції зазвичай використовуються тільки всередині одного проєкту, але бувають винятки — і такі функції виносять до бібліотек.

*Аргументи функції* — це змінні, які ми передаємо у функцію обробки. Аргументів може бути кілька.

Правильне і зворотне — змінні, визначені всередині функції, не будуть доступні ззовні. Такі змінні називаються локальними, оскільки вони локальні стосовно функції. На відміну від аргументів, яких може бути кілька, повернути у зовнішній код функція може лише одне значення — за допомогою інструкції «`return`» (повернення). Значення, що повертається називають результатом роботи функції.

Функція складається з кількох частин:

імені функції, аргументів, що передаються у функцію, тіла функції, оператора `return`, який відповідає за повернення результату до сценарію.

Що потрібно запам'ятати: Функція — це фрагмент коду, якому дали ім'я. Функції потрібні, щоб не переписувати той самий код багато разів. У функцію можна передати багато змінних, але повернути щось одне. Змінні функції недоступні зовні, зовнішні змінні потрібно передавати через аргументи.

## 5.4 Посилання

Посилання в PHP дозволяють посилатися на область пам'яті, де розташовано значення змінної або параметра. Для створення посилання перед змінною вказується символ амперсанда & [8].

Що таке посилання на РНР? Посилання в РНР - це засіб доступу до вмісту однієї змінної під різними іменами. Вони не схожі на покажчики C; наприклад, ви не можете виконувати над ними адресну арифметику, вони не є реальними адресами пам'яті і т.д.

## 5.5 Масив

Масив – це набір даних, які об'єднані під одним ім'ям. Масив складається з кількох елементів, які мають певний індекс [8].

Масиви створюються з допомогою оператора присвоєння, як і змінна. Імена масивів починаються зі знака \$, після якого слідує довільний ідентифікатор, далі йдуть квадратні дужки: `$arr[0] = "php";`

Дана конструкція створює масив і надає його елементу з індексом 0 значення "php", після чого ми можемо звертатися до цього елемента як до звичайної змінної: `echo $arr[0]`. В результаті ми побачимо слово *php*.

## 5.6 Генератори

Генератори надають легкий спосіб реалізації простих ітераторів без використання додаткових ресурсів чи складнощів, пов'язаних із реалізацією класу, що реалізує інтерфейс Iterator. Генератор дозволяє вам писати код, що використовує `foreach` для перебору набору даних без необхідності створення масиву в пам'яті, що може призвести до перевищення ліміту пам'яті, або зажадає багато часу для його створення. Замість цього, ви можете написати функцію-генератор, яка, по суті, є звичайною функцією, за винятком того, що замість повернення єдиного значення, генератор може повертати (`yield`) стільки разів, скільки необхідно для генерації значень, що дозволяють перебрати вихідний набір даних. Наочним прикладом вищесказаного може бути використання функції `range()` як генератора. Стандартна функція `range()` генерує масив, що складається із значень, і повертає його, що може призвести до генерації величезних масивів даних. Наприклад, виклик `range(0, 1000000)` призведе до використання понад 100 МБ оперативної пам'яті. В якості альтернативи ми можемо створити генератор `xrange()`, який використовує пам'ять тільки для

створення об'єкта Iterator і збереження поточного стану, що вимагатиме не більше 1 кілобайта пам'яті [8].

Коли функція генератор викликається, вона поверне об'єкт вбудованого класу Generator. Цей об'єкт реалізує інтерфейс Iterator стане односпрямованим об'єктом ітератора і надасть методи, за допомогою яких можна керувати його станом, включаючи передачу в нього і повернення з нього значень.

## 5.7 Підхід до написання чистого коду

Немає універсального, істинного і єдиного шляху та рішення для написання оптимального і чистого коду. Є той, який найкраще підходить для вирішення конкретного завдання. При вирішенні завдання намагайтеся відтворити всі кейси, які можуть торкатися цього завдання і реалізуйте завдання з урахуванням всіх кейсів. Також при вирішенні задачі спробуйте піти від зворотного. Зрозумійте, які результати в решті-решт необхідно отримати і складіть на цій підставі алгоритм, за яким виконуватиметься завдання. Для самоперевірки можна спробувати відповісти на питання [9]:

- Які кейси можуть мати завдання?
- Чи все я врахував?
- Що може йти не так?
- Що можна поєднати?
- Чи є схожий функціонал?
- Що тут зайве?
- Як зробити простіше?
- Як зробити читабельніше?
- Як зробити зрозуміліше?

Як писати чистий та хороший код? Це схоже на написання книги. Спочатку ти робиш чернетку і потім "зачісуєш" її до того стану, в якому тобі було б приємно її читати.

Чистий код простий, виразний та спрямований на конкретне завдання. Чистий код читається легко як проза. Якщо це не так, його варто змінювати. Чистий код легко змінювати. Він повинен бути жорстко зав'язаний на купі



сутностей. Будь-яку сутність можна легко змінити. Чистий код набагато краще проходить перевірку. Якщо перевірка проходить із величезною кількістю коментарів, то він не чистий і його треба змінювати. Чистий код завжди виглядає так, ніби над ним дуже довго працювали. Які б шляхи для його покращення ти не шукав, ти все одно прийдеш до того, що цей код найкращий. Відповідно, чистий код – продуманий до всіх дрібниць. Правило бойскаута: «Залиш місце стоянки чистішим, ніж воно було до тебе» [9]. Це легко перекладається і програмування. Бачиш "брудний код"? Зроби його чистішим, поки вирішуєш своє завдання. Не варто захоплюватися цим і якщо "брудний код дуже брудний", варто виділити окреме завдання і час для його очищення. Будь-яка сутність має відповідати за один функціонал і лише за нього. І вона має виконувати його добре – Single Responsibility. Якщо сутність відповідає одночасно двом і більше діям, її функціонал треба розділяти. Код повинен читатись зверху вниз. У гарній та грамотній архітектурі внесення змін обходиться без значних витрат. Видаляй "мертвий код". "Мертвий код" – це код, який не буде викликаний за жодних умов або код, який ніде не використовується.

## **5.8 Найменування та розділення**

Використовуй зрозумілі і зручні імена для будь-яких сутностей. Вони повинні описувати, чому ця сутність існує, що вона робить і як використовується. Довжину імен змінних, методів варто намагатися робити не надто довгою. Ідеальною довжиною вважається 8 символів. Але якщо зміст ім'я не відображає зміст змінної, функціонал методу, призначення класу чи інтерфейсу, варто збільшити довжину. Довге та зрозуміле ім'я краще, ніж коротке, але незрозуміле. Не бійся витратити час на вибір кращого та зрозумілого імені. Ти виграєш у майбутньому під час роботи або читання цього коду. Якщо назва сутності не відповідає її функціоналу або за назвою не зрозуміло, що сутність робить, то її треба перейменувати на найзрозумілішу назву. Якщо цього зробити неможливо, то значить із її функціоналом щось не так і її треба рефакторити. Сутність, яка має назву "And", "With" — порушує Single Responsibility. Функціонал такої сутності варто поділяти. Незрозумілі тексти,

рядки варто виносити у змінні та давати їм зрозумілі назви. Назви методів повинні містити дієслово, яке описує, що цей метод робить. Потрібно уникати однакових найменувань для двох різних цілей [9].

Якщо сутність має схожу з іншою сутністю назву, то швидше за все їх функціонал дуже схожий і їх потрібно об'єднати. Якщо ні, їх назви потрібно змінювати, щоб вони не співпадали. Якщо ти подумки перейменовуєш сутність, коли читаєш код, щоб тобі було зрозуміліше розуміти її функціонал, то перейменуй її в цю назву. Виберіть одне слово для однієї концепції. Важко буде розуміти функціонал, коли в тебе є `fetch`, `retrieve` та `get` у назвах. Нехай краще скрізь буде `get`.

## 5.9 Функції

Функції мають бути короткими та компактними. Оптимальним розміром вважається 20 рядків і 150 символів в одному рядку, якщо не влізть, потрібно розділяти. Функція повинна виконувати лише одну операцію. Вона повинна виконувати її добре і нічого іншого вона не повинна робити. Якщо функція виконує лише ті дії, які знаходяться на одному рівні абстракції, функція виконує одну операцію. Щоб визначити, чи виконує функція більше однієї операції, спробуй витягти з неї іншу функцію, яка не буде простою переформулюванням реалізації. Будь-які умовні оператори з довгими виборами через `switch-case`, `if-else` повинні розділятися або об'єднуватися без дублювання, можливо на класи з реалізаціями, а вибір реалізації передати базовому класу, фабриці чи будь-яким іншим чином. `If`, `else`, `while` і т.д. повинні містити виклик однієї функції [9]. Так буде читабельніше, зрозуміліше та простіше. Якщо вхідних аргументів більше трьох, то варто задуматися, яким чином краще їх позбутися, наприклад, створити клас для цих аргументів. Що більше вхідних аргументів, то важче розуміється функція. Функція, в яку передається аргумент-прапор, від якого залежить робота функції свідчить, що функція виконує більше однієї операції. Такі функції слід розбити на дві та викликати їх рівнем вище. Функція, яка змінює вхідний аргумент, має віддавати посилання на змінений об'єкт, а не просто змінювати без повернення.

String transform(String text)

Якщо функція повинна змінювати вхідний аргумент, то нехай вона змінює стан свого об'єкта-власника. Якщо вхідний аргумент функції не повинен змінюватися (і використовується далі в коді), слід скопіювати значення аргументу і всередині функції працювати з копією. Замість return null краще використовувати порожній об'єкт Collection.empty() або null-об'єкт - EmptyObject(). Якщо є код, який повинен слідувати один за одним, передай результати першої функції в другу, щоб хтось не змінив послідовність викликів. Використовуй поліморфізм замість if/else чи switch/case чи when. Уникай негативних умов.

## 5.10 Коментарі

Не використовуй коментарі, якщо ти можеш використовувати функцію або змінну замість цього. Не коментуй поганий код – перепиши його. Не варто пояснювати, що відбувається у поганому коді, краще зробити його явним та зрозумілим. Коментарі можна використовувати при передачі будь-якої інформації, попередження про наслідки, але з пояснення того, як працює код. Використовуй TODO та FIXME у тих випадках, коли потрібно помітити, що код потребує доопрацювання, але зараз немає ресурсів на це [9].

Документуй код, який є складним, але чистим. Не залишай старий закоментований код. Ти можеш знайти його в історії коммітів, якщо потрібно. Коментарі мають бути короткими та зрозумілими. У коментарях з інформацією не має бути багато інформації. Все має бути стисло і змістовно.

## 5.11 Форматування та правила

Дотримуйся codestyle, прийнятий на проекті. Дотримуйся правил, прийнятих у команді. При дотриманні форматування та codestyle код читатиметься простіше і краще. Адже не даремно книгу віддають на редакцію, перед тим, як її видавати. Потрібно мати автоматичні засоби, які формуватимуть код за тебе. Файл із вихідним кодом має бути як газетна стаття. Є заголовок, короткий опис у вигляді параметрів та зміст у вигляді функцій.

Якщо це не так, то варто змінити форматування. Сутності, пов'язані один з одним, повинні бути поруч, наприклад, в одному пакеті, щоб було простіше здійснювати навігацію за кодом. Змінні (поля) класу повинні бути вгорі класу. Методи повинні знаходитися ближче до свого місця використання. Функції повинні знаходитись у порядку виклику. Якщо одна викликає іншу, то функція, що викликає, повинна перебувати над викликаною. З іншого боку, приватні функції нижчого рівня можуть бути внизу файлу і заважати розумінню коду високого рівня. Необхідно працювати з абстракціями, щоб реалізацію можна було легко змінити. Клієнт, який використовує функціонал, не повинен знати про деталі реалізації, він повинен знати, яку реалізацію в якому разі використовувати. Необхідно надавати API, з яким варто працювати та приховувати деталі реалізації, структуру. Так буде простіше працювати з сутностями та додавати нові види поведінок, функціоналу та реалізацій [9].

DTO - Data Transfer Object. Клас, який містить лише дані та ніякого функціоналу. Потрібний для того, щоб передавати якісь дані. Об'єкт такого класу має бути незмінним.

## **5.12 Класи**

Класи мають бути компактними. Ім'я класу має описувати його відповідальність. Функціонал класу повинен чітко відповідати та вписуватися в назву класу. Розділяй пов'язаність на маленькі класи. Жорсткої та рясної зв'язаності не повинно бути— це ускладнює підтримку та розвиток проекту [9].

Пам'ятай про Single Responsibility. Сутність повинна мати одну і лише одну причину зміни. Дотримуйся інкапсуляції. Ослаблення інкапсуляції завжди має бути останньою мірою. Зазвичай ми оголошуємо змінні та допоміжні функції приватними, але іноді їх потрібно оголошувати `protected` та мати змогу звернутися до неї з тесту. Якщо група функцій належить до певного функціоналу, то цю групу функцій можна виділити в окремий клас і використовувати його екземпляр.

### 5.13 Обробка помилок

Використовуйте Exceptions замість повернення кодів помилок. Обробка помилок – це одна операція. Якщо у функції є ключове слово try, то після блоків catch/finally нічого іншого у функції не повинно бути. Якщо в тебе є enum, який перераховує помилки, то його краще позбутися і замість нього використовувати винятки. Використовуйте unchecked exceptions, щоб явно вказати на місце в якому є проблеми. Такі помилки не потрібно відловлювати, натомість потрібно написати код так, щоб цієї помилки ніколи не було. Передавай достатню кількість інформації разом із появою виключення, щоб потім користувачі твого коду могли зрозуміти, що справді сталося. Замість умовних операторів із обробкою помилок краще викидати винятки та обробляти їх. Не передавай null будь-куди. Намагайся цього максимально уникнути. Обробка помилок – це окреме завдання і не відноситься до основної логіки програми [9].

### 5.14 Межі

Ми завжди використовуємо будь-які бібліотеки, які найчастіше дають нам занадто широкий функціонал або конфліктують із очікуваним функціоналом, що робить код бруднішим у його кінцевому використанні. Уникнути цього можна просто, застосувавши патерни типу Decorator, Adapter, Facade або інші. Бувають ситуації, коли тобі потрібно працювати з функціоналом, який знаходиться у розробці або поки що не адаптований для використання у коді продакшен. У цьому випадку варто уявити, чого ти чекаєш від бібліотеки/цього функціоналу і написати свій інтерфейс або створити сутність з якими ти працюватимеш у своєму проекті так, як тобі потрібно. Коли бібліотека доробиться та стане стабільною, ти адаптуєш її під свої готові структури та використати вже готовий функціонал.

### 5.15 Приклади коду PHP

#### Приклад № 1 Перший скрипт на PHP: hello.php

Створіть файл з ім'ям hello.php у кореневому каталозі веб-сервера (DOCUMENT\_ROOT) і запишіть у нього наступне:

```
<html>
<head>
<title>Тестуємо PHP</title>
</head>
<body>
<?php echo '<p>Привіт, світ!</p>'; ?>
</body>
</html>
```

Відкрийте файл у браузері, набравши ім'я вашого веб-сервера та /hello.php. При локальній розробці це посилання може бути чимось на зразок `http://localhost/hello.php` або `http://127.0.0.1/hello.php`, але це залежить від налаштувань вашого сервера. Якщо все налаштовано правильно, файл буде оброблений PHP і браузер виведе наступний текст:

```
<html>
  <head>
    <title> Тестуємо PHP </title>
  </head>
  <body>
    <p>Привіт, світ!</p>
  </body>
</html>
```

Ця програма надзвичайно проста, і для створення настільки простої сторінки навіть необов'язково використовувати PHP. Все, що вона робить, це виведення Hello World, використовуючи інструкцію PHP `echo`. Зверніть увагу, що файл не повинен бути виконуваним, або ще якось відрізнитися від інших файлів. Сервер знає, що цей файл повинен бути оброблений PHP, тому що файл має розширення `".php"`, про яке в налаштуваннях сервера сказано, що подібні файли повинні передаватися PHP. Розглядайте його як звичайний HTML-файл, якому пощастило отримати набір спеціальних тегів (доступних також і вам), здатних на купу цікавих речей.

Якщо у вас цей приклад не відображає нічого або виводить вікно завантаження, або якщо ви бачите весь цей файл у текстовому вигляді, то, швидше за все, ваш веб-сервер не має підтримки PHP або налаштований неправильно. Переконайтеся, що ви запитуєте файл на сервері через протокол `http`. Якщо ви просто відкриєте файл із вашої файлової системи, він не буде оброблений PHP. Мета прикладу – показати формат спеціальних тегів PHP. У цьому прикладі ми використовували `<? php` як тег, що відкривається, потім йшли

команди PHP, що завершуються закриваючим тегом `?>`. Таким чином можна будь-де "застрибувати" і "вистрибувати" з режиму PHP в HTML файлі.

### Приклад № 2 Отримання інформації про систему з PHP

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

### Приклад № 3 Виведення значення змінної (елемента масиву)

Визначимо, який браузер використовує той, хто дивиться зараз нашу сторінку. Для цього ми перевіримо рядок з ім'ям браузера, що надсилається нам у HTTP-запиті. Ця інформація зберігається у змінній. Змінні в PHP завжди передуються знаком долара. Змінна, що цікавить нас в даний момент, називається `$_SERVER['HTTP_USER_AGENT']`.

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];
?>
```

### Приклад виведення цієї програми:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

У PHP існує безліч типів змінних. У попередньому прикладі ми виводили елемент масиву.

`$_SERVER` – це одна із змінних, які надаються вам мовою PHP. Список таких змінних можна переглянути в розділі "Зарезервовані змінні" або переглянувши виведення функції `phpinfo()`, яка використовується в попереднього прикладі.

Всередині PHP-тегів можна поміщати кілька виразів і створювати маленькі блоки коду, що роблять більше, ніж простий виклик `echo`. Наприклад, якщо ви хочете додати перевірку для Internet Explorer.

### Приклад № 4 Приклад використання керуючих структур та функцій

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
    echo 'Ви використовуєте Internet Explorer.<br/>';
}
?>
```

### Приклад виведення цієї програми:

```
Ви використовуєте Internet Explorer.
```

Тут ми показали кілька нових елементів. По-перше, тут є конструкція `if`. Якщо ви знайомі з основами синтаксису мови C, ви вже помітили щось схоже. Якщо ж ви не знаєте C або подібної за синтаксисом мови, то найкращий варіант - взяти якусь вступну книжку по PHP і прочитати перші пару розділів.

Крім цього, тут є виклик функції `strpos()`. `strpos()` - вбудована в PHP функція, яка шукає один рядок в іншому. В даному випадку ми шукаємо рядок 'MSIE' (так звану голку - needle) в `$_SERVER['HTTP_USER_AGENT']` (у так званому "сіні" - haystack). Якщо "голка" знайдена всередині "сіна", функція повертає позицію "голки" щодо початку "сіна". Інакше вона повертає `false`. Якщо вона не поверне `false`, то умова в `if` виявиться істинною (`true`), і код у фігурних дужках (`{ }`) виконається. В іншому випадку код не виконується. Якщо ви не знаєте, як використовувати функції, можливо, вам варто прочитати сторінку посібника про те, як читати визначення функцій та розділ про функції PHP.

### Приклад № 5 Змішування режимів HTML та PHP

Продемонструємо, як можна входити в режим PHP коду і виходити з нього навіть прямо посередині блоку з кодом.

```
<?php
if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
?>
<h3>strpos() повернув не false</h3>
<p>Ви використовуєте Internet Explorer</p>
<?php
} else {
?>
<h3>strpos() вернул false</h3>
<p>Ви не використовуєте Internet Explorer</p>
<?php
}
?>
```

Приклад виведення цієї програми:

```
<h3>strpos() повернув не false</h3>
<p>Ви використовуєте Internet Explorer</p>
```

Замість використання команди PHP `echo` для виведення ми вийшли з режиму коду і просто послали вміст HTML. Важливий момент тут те, що логічна структура PHP коду при цьому не втрачається. Тільки одна HTML-частина буде



надіслана клієнту залежно від результату функції strpos() (іншими словами, залежно від того, знайдено рядок MSIE чи ні).

Одна з найголовніших переваг PHP – те, як він працює з формами HTML. Тут основним є те, що кожен елемент форми автоматично стає доступним вашим програмам на PHP.

### Приклад № 6 Найпростіша форма HTML

Приклад форми HTML:

```
<form action="action.php" method="post">
  <p>Ваше ім'я: <input type="text" name="name" /></p>
  <p>Ваш вік: <input type="text" name="age" /></p>
  <p><input type="submit" /></p>
</form>
```

У цій формі немає нічого особливого. Це звичайна форма HTML без спеціальних тегів. Коли користувач заповнить форму та натисне кнопку відправки, буде викликана сторінка action.php.

### Приклад № 7 Виводимо дані форми

У файлі action.php може бути наступне:

```
Вітаю, <?php echo htmlspecialchars($_POST['name']); ?>.
Вам <?php echo (int)$_POST['age']; ?> років.
```

Приклад виведення цієї програми:

```
Вітаю, Сергію. Вам тридцять років.
```

Якщо не брати до уваги фрагменти коду з htmlspecialchars() і (int), принцип роботи даного коду повинен бути простим і зрозумілим. htmlspecialchars() забезпечує правильне кодування "особливих" HTML-символів так, щоб шкідливий HTML або Javascript не був вставлений на вашу сторінку. Поле age, про яке нам відомо, що воно має бути число, ми можемо просто перетворити на int, що автоматично позбавить нас небажаних символів. PHP також може зробити це автоматично за допомогою модуля filter. Змінні \$\_POST['name'] та \$\_POST['age'] автоматично встановлені для вас засобами PHP. Раніше ми використовували суперглобальну змінну \$\_SERVER, тут ми так само використовуємо суперглобальну змінну \$\_POST, яка містить всі POST-дані. Зауважимо, що метод відправлення (method) нашої форми – POST. Якби ми

використовували метод GET, то інформація нашої форми була б у суперглобальній змінній \$\_GET. Крім цього, можна використовувати змінну \$\_REQUEST, якщо джерело даних не має значення. Ця змінна містить суміш GET, POST, COOKIE.

### Приклад № 8 Цілі числа

Цілі числа (int) можуть бути зазначені в десятковій (основа 10), шістнадцятковій (основа 16), вісімковій (основа 8) або двійковій (основа 2) системі числення. Для завдання негативних цілих (int) використовується оператор заперечення.

Для запису у вісімковій системі числення необхідно поставити перед числом 0(нуль). Починаючи з PHP 8.1.0, в вісімковій нотації також може передувати 0o або 0O. Для запису в шістнадцятковій системі числення необхідно поставити перед числом 0x. Для запису в двійковій системі числення необхідно поставити перед числом 0b.

Починаючи з PHP 7.4.0, цілі літерали можуть містити підкреслення (  ) між цифрами для кращої читаності літералів. Ці підкреслення видаляються сканером PHP.

```
<?php
$a = 1234; // десяткове число
$a = 0123; // вісімкове число (еквівалентно 83 в десятковій системі)
$a = 0o123; // вісімкове число (починаючи з PHP 8.1.0)
$a = 0x1A; // шістнадцяткове число (еквівалентно 26 в десятковій системі)
$a = 0b11111111; // двійкове число (еквівалентно 255 в десятковій системі)
$a = 1_234_567; // десяткове число (починаючи з PHP 7.4.0)
?>
```

Формально структура цілих чисел int прийнята в PHP 8.1.0 (раніше не допускалися вісімкові префікси 0o або 0O, а до PHP 7.4.0 не допускалося підкреслення):

десяткові:  $[1-9][0-9]^*(\_[0-9]^+)^*$   
          | 0

шістнадцяткові:  $0[xX][0-9a-fA-F]^+(\_[0-9a-fA-F]^+)^*$

вісімкові:  $0[oO]?[0-7]^+(\_[0-7]^+)^*$

```
двійкові: 0 [bV] [01]+( _[01]+) *
```

```
цілі: десяткові  
      | шістнадцяткові  
      | вісімкові  
      | двійкові
```

Розмір типу `int` залежить від платформи, хоча, як правило, максимальне значення дорівнює `2 147 483 647` (це в 32-розрядній системі). 64-бітові платформи зазвичай мають максимальне значення близько `9E18`. PHP не підтримує беззнакові цілі числа (`int`). Розмір `int` може бути визначений за допомогою константи `PHP_INT_SIZE`, максимальне значення – за допомогою константи `PHP_INT_MAX`, а за допомогою константи `PHP_INT_MIN` можна визначити мінімальне значення.

### Приклад № 9 Переповнення цілих на 32-розрядних системах

Якщо PHP виявив, що число перевищує розмір типу `int`, він інтерпретуватиме його як `float`. Аналогічно, якщо результат операції лежить за межами типу `int`, він буде перетворений на `float`.

```
<?php  
$large_number = 2147483647;  
var_dump($large_number);           // int(2147483647)  
  
$large_number = 2147483648;  
var_dump($large_number);           // float(2147483648)  
  
$million = 1000000;  
$large_number = 50000 * $million;  
var_dump($large_number);           // float(50000000000)  
?>
```

### Приклад № 10 Переповнення цілих на 64-розрядних системах

```
<?php  
$large_number = 9223372036854775807;  
var_dump($large_number);           // int(9223372036854775807)  
  
$large_number = 9223372036854775808;  
var_dump($large_number);           // float(9.2233720368548E+18)  
  
$million = 1000000;  
$large_number = 50000000000000 * $million;  
var_dump($large_number);           // float(5.0E+19)  
?> <?php  
$large_number = 9223372036854775807;  
var_dump($large_number);           // int(9223372036854775807)
```

```

$large_number = 9223372036854775808;
var_dump($large_number); // float(9.2233720368548E+18)

$million = 1000000;
$large_number = 50000000000000 * $million;
var_dump($large_number); // float(5.0E+19)
?>

```

У PHP немає оператора поділу цілих чисел (int), для цього використовуйте функцію явне приведення типу до (int). Результатом 1/2 буде float 0.5. Якщо привести значення до int, воно буде заокруглено вниз, тобто буде відкинута дрібна частина числа. Для більшого контролю за заокругленням використовуйте функцію round().

```

<?php
var_dump(25/7); // float(3.5714285714286)
var_dump((int) (25/7)); // int(3)
var_dump(round(25/7)); // float(4)
?>

```

Перетворення на ціле.

Для явного перетворення на int, використовуйте приведення (int) або (integer). Однак, у більшості випадків, у приведенні типу немає необхідності, оскільки значення буде автоматично перетворено, якщо оператор, функція або структура, що управляє, вимагає аргумент типу int. Значення також може бути перетворено на int за допомогою функції intval().

*З булевого типу.*

**false** перетворюється на 0(нуль), а **true**- на 1(одиницю).

*З чисел із плаваючою точкою.*

При перетворенні з float в int число буде округлено в бік нуля. Починаючи з PHP 8.1.0, при неявному перетворенні неінтегрального числа з плаваючою точкою (float) на ціле число (int), яке втрачає точність, видається повідомлення про застарілий (Deprecated) функціонал.

```

<?php
function foo($value): int {
    return $value;
}

```

```

var_dump(foo(8.1)); // "Deprecated: Implicit conversion from float
8.1 to int loses precision" починаючи з PHP 8.1.0
var_dump(foo(8.1)); // 8 до PHP 8.1.0
var_dump(foo(8.0)); // 8 в обох випадках

var_dump((int)8.1); // 8 в обох випадках
var_dump(intval(8.1)); // 8 в обох випадках
?>

```

Якщо число з плаваючою точкою перевищує розміри `int` (зазвичай  $\pm 2.15e+9 = 2^{31}$  на 32-бітових системах і  $\pm 9.22e+18 = 2^{63}$  на 64-бітових системах, результат буде невизначеним, оскільки `float` не має достатньої точності, щоб повернути правильний результат у вигляді цілого числа (`int`). Не буде виведено ні попередження, ні навіть зауваження!

#### Зауваження:

Значення `NaN` та `Infinity` при приведенні до `int` стають рівними нулю, замість невизначеного значення залежно від платформи.

#### **Приклад № 11** Приклад простого синтаксису

Якщо інтерпретатор зустрічає знак долара (`$`), він захоплює багато символів, скільки можливо, щоб сформувати правильне ім'я змінної. Якщо потрібно точно визначити кінець імені, укладайте ім'я змінної у фігурні дужки.

```

<?php
$juice = "apple";

echo "He drank some $juice juice.".PHP_EOL;

echo "He drank some juice made of $juices.";
// Некоректно. Символ 's' може використовуватися для імені
змінної, але змінна має ім'я $juice.

echo "He drank some juice made of ${juice}s.";
// Коректно. Строго вказаний кінець імені змінної за допомогою
дужок.

?>

```

#### Результат виконання цього прикладу:

```

He drank some apple juice.
He drank some juice made of.
He drank some juice made of apples.

```

Аналогічно можуть бути оброблені елементи масиву (`array`) або властивість об'єкта (`object`). У індексах масиву квадратна дужка (`[]`), що закриває, позначає

кінець визначення індексу. Для властивостей об'єкта застосовуються ті самі правила, як і простих змінних.

```
<?php
$juices = array("apple", "orange", "koolaid1" => "purple");

echo "He drank some $juices[0] juice.".PHP_EOL;
echo "He drank some $juices[1] juice.".PHP_EOL;
echo "He drank some $juices[koolaid1] juice.".PHP_EOL;

class people {
    public $john = "John Smith";
    public $jane = "Jane Smith";
    public $robert = "Robert Paulsen";

    public $smith = "Smith";
}

$people = new people();

echo "$people->john drank some $juices[0] juice.".PHP_EOL;
echo "$people->john then said hello to $people->jane.".PHP_EOL;
echo "$people->john's wife greeted $people->robert.".PHP_EOL;
echo "$people->robert greeted the two $people->smiths."; // Не
спрацює
?>
```

**Результат виконання цього прикладу:**

```
He drank some apple juice.
He drank some orange juice.
He drank some purple juice.
John Smith drank some apple juice.
John Smith then said hello to Jane Smith.
John Smith's wife greeted Robert Paulsen.
Robert Paulsen greeted the two.
```

**Приклад № 12 Негативні числові індекси**

```
<?php
$string = 'string';
echo "Символ з індексом -2 дорівнює $string[-2].", PHP_EOL;
$string[-3] = 'o';
echo "Зміна символу на позиції -3 на 'o' дає наступний рядок:
$string.", PHP_EOL;
?>
```

**Результат виконання цього прикладу:**

```
Символ з індексом -2 дорівнює n.
Зміна символу на позиції -3 на 'o' дає наступний рядок: strong
```

Для чогось складнішого, використовуйте складний синтаксис.

### Приклад № 13 Складний (фігурний) синтаксис

Синтаксис називається складним не тому, що важкий у розумінні, а тому, що дозволяє використовувати складні (комплексні) вирази.

Будь-яка скалярна змінна, елемент масиву або властивість об'єкта, що відображається в рядок, може бути представлена цим синтаксисом. Вираз записується так само, як і поза рядком, а потім заключаємо в фігурні дужки { }. Оскільки відкриваюча фігурна дужка { не може бути екранована, цей синтаксис буде розпізнаватись тільки коли знак долара \$ слідує безпосередньо за {. Використовуйте {\\$ для друку {\$.

```
<?php
// Показувати усі помилки
error_reporting(E_ALL);

$great = 'чудово';

// Не працює, виводить: Це { чудово}
echo "Це { $great}";

// Працює, виводить: Це чудово
echo "Це {$great}";

// Працює
echo "Цей квадрат шириною {$square->width}00 сантиметрів.";

// Працює, ключі, заключені в лапки, працюють тільки з синтаксисом фігурних дужок
echo "Це працює: {$arr['key']}";

// Працює
echo "Це працює: {$arr[4][3]}";

// Це не правильно з тієї ж причини, що і $foo[bar] поза
// рядка. Іншими словами, це буде працювати,
// але оскільки PHP спочатку шукає константу foo, це викличе
// помилку рівня E_NOTICE (невизначена константа).
echo "Це неправильно: {$arr[foo][3]}";

// Працює. При використанні багатовимірних масивів усередині
// рядків завжди використовуйте фігурні дужки
echo "Це працює: {$arr['foo'][3]}";

// Працює.
echo "Це працює: ". $arr['foo'][3];

echo "Це також працює: {$obj->values[3]->name}";

echo "Це значення змінної на ім'я $name: {${$name}}";

echo "Це значення змінної на ім'я, яке повертає функція getName():
```

```

{{getName()}}";

echo "Це значення змінної на ім'я, яке повертає \$object->getName():
{{\$object->getName()}}";

// Не працює, виводить: Це те, що повертає getName(): {getName()}
echo "Це те, що повертає getName(): {getName()}";

// Не працює, виводить: C:\folder\{fantastic}.txt
echo "C:\folder\{$great}.txt"

// Працює, виводить: C:\folder\fantastic.txt
echo "C:\\folder\\{$great}.txt"
?>

```

За допомогою цього синтаксису також доступ до властивостей об'єкта всередині рядків.

```

<?php
class foo {
    var $bar = 'I am bar.';
}

$foo = new foo();
$bar = 'bar';
$baz = array('foo', 'bar', 'baz', 'quux');
echo "{$foo->$bar}\n";
echo "{$foo->{$baz[1]}}\n";
?>

```

Результат виконання цього прикладу:

```

I am bar.
I am bar.

```

Зауваження:

Значення, до якого здійснюється доступ із функцій, викликів методів, статичних змінних класу та констант класу всередині `{}`, буде інтерпретуватися як ім'я змінної в області, в якій визначено рядок. Використання одинарних фігурних дужок (`{}`) не працюватиме для доступу до значень функцій, методів, констант класів або статичних змінних класу.

```

<?php
// Показувати усі помилки
error_reporting(E_ALL);

class beers {
    const softdrink = 'rootbeer';
    public static $ale = 'ipa';
}

$rootbeer = 'A & W';

```



```

$ipa = 'Alexander Keith\'s';

// Це працює, виводить: Я б хотів A & W
echo "Я б хотів ${beers::softdrink}}\n";

// Це теж працює, виводить: Я б хотів Alexander Keith's
echo "Я б хотів ${beers::$ale}}\n";
?>

```

### Приклад № 14 Простий масив

Насправді масив в PHP – це впорядковане відображення, яке встановлює відповідність між значенням і ключем. Цей тип оптимізований у кількох напрямках, тому ви можете використовувати його як власне масив, список (вектор), хеш-таблицю (що є реалізацією карти), словник, колекцію, стек, чергу. Оскільки значенням масиву може бути інший масив PHP, можна також створювати дерева та багатовимірні масиви.

Масив (тип array) може бути створений мовною конструкцією array(). Як параметри вона приймає будь-яку кількість розділених комами пар (ключ => значення, key => value).

```

array(
    key => value ,
    key2 => value2 ,
    key3 => value3 ,
    ...
)

```

Кома після останнього елемента масиву необов'язкова і може бути опущена. Зазвичай це робиться для однорядкових масивів, тобто array(1, 2) краще array(1, 2,). Для багаторядкових масивів з іншого боку зазвичай використовується завершальна кома, так як дозволяє легше додавати нові елементи в кінець масиву.

#### Зауваження:

Існує короткий синтаксис масиву, який замінює array() на [].

```

<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// Використання короткого синтаксису об'явлення масиву
$array = [

```

```

"foo" => "bar",
"bar" => "foo",
];
?>

```

key може бути або типу int , або типу string. value може бути будь-якого типу.

Додатково з ключем key будуть зроблені такі перетворення:

1) Рядки (string), що містять ціле число (int) (за винятком випадків, коли число випереджається знаком +) будуть перетворені до типу int. Наприклад, ключ зі значенням "8"буде збережений зі значенням 8. З іншого боку, значення "08"не буде перетворено, оскільки воно не є коректним десятковим цілим.

2) Числа з плаваючою точкою (float) також будуть перетворені до типу int , тобто дрібна частина буде відкинута. Наприклад, ключ зі значенням 8.7 буде збережений зі значенням 8.

3) Тип bool також перетворюються на тип int. Наприклад, ключ зі значенням true буде збережено зі значенням 1 і ключ зі значенням false буде збережено зі значенням 0.

4) Тип null буде перетворено на порожній рядок. Наприклад, ключ зі значенням null буде збережений зі значенням "".

5) Масиви (array) і об'єкти (object) не можуть використовуватися як ключі. При такому використанні генеруватиметься попередження: Неприпустимий тип зміщення (Illegal offset type).

Якщо кілька елементів в оголошенні масиву використовують однаковий ключ, лише останній буде використовуватися, а всі інші будуть перезаписані.

### Приклад № 15 Приклад перетворення типів та перезапису елементів

```

<?php
$array = array(
    1 => "a",
    "1" => "b",
    1.5 => "c",
    true => "d",
);
var_dump($array);
?>

```

Результат виконання цього прикладу:

```

array(1) {
    [1]=>

```

```
string(1) "d"
}
```

Оскільки всі ключі у наведеному прикладі перетворюються на 1, значення буде перезаписано на кожен новий елемент і залишиться тільки останнє присвоєне значення "d".

### Приклад № 16 Змішані ключі типів int та string

Масиви PHP можуть містити ключі типів int і string одночасно, так як PHP не робить відмінності між індексованими і асоціативними масивами.

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100 => -100,
    -100 => 100,
);
var_dump($array);
?>
```

Результат виконання цього прикладу:

```
array(4) {
  ["foo"]=>
  string(3) "bar"
  ["bar"]=>
  string(3) "foo"
  [100] =>
  int(-100)
  [-100] =>
  int(100)
}
```

Параметр `key` є необов'язковим. Якщо він не вказаний, PHP використовуватиме попереднє найбільше значення ключа типу `int`, збільшене на 1.

### Приклад № 17 Індексвані масиви без ключа

```
<?php
$array = array("foo", "bar", "hallo", "world");
var_dump($array);
?>
```

Результат виконання цього прикладу:

```
array(4) {
  [0] =>
  string(3) "foo"
  [1] =>
  string(3) "bar"
  [2] =>
```

```

string(5) "hallo"
[3]=>
string(5) "world"
}

```

### Приклад № 18 Ключі для деяких елементів

Можливо вказати ключ тільки для деяких елементів та пропустити для інших:

```

<?php
$array = array(
    "a",
    "b",
    6 => "c",
    "d",
);
var_dump($array);
?>

```

Результат виконання цього прикладу:

```

array(4) {
  [0]=>
  string(1) "a"
  [1]=>
  string(1) "b"
  [6]=>
  string(1) "c"
  [7] =>
  string(1) "d"
}

```

Як ви бачите останнє значення "d" було надано ключу 7. Це сталося тому, що найбільше значення ключа цілого типу перед цим було 6.

**Приклад № 19** Розширений приклад перетворення типів та перезапису елементів

```

<?php
$array = array(
    1 => 'a',
    '1' => 'b', // значення "b" перезапише значення "a"
    1.5 => 'c', // значення "c" перезапише значення "b"
    -1 => 'd',
    '01' => 'e', // оскільки це не цілочисельний рядок, вона НЕ
перезапише ключ для 1
    '1.5' => 'f', // оскільки це не цілочисельний рядок, вона НЕ
перезапише ключ для 1
    true => 'g', // значення "g" перезапише значення "c"
    false => 'h',
    '' => 'i',
    null => 'j', // значення "j" перезапише значення "i"
    'k', // значення "k" присвоєється ключу 2. Тому що найбільший

```

```

цілий ключ до цього був 1
 2 => '1', // значення "1" перезапише значення "k"
);
var_dump($array);
?>

```

Результат виконання цього прикладу:

```

array(7) {
  [1]=>
  string(1) "g"
  [-1]=>
  string(1) "d"
  ["01"]=>
  string(1) "e"
  ["1.5"]=>
  string(1) "f"
  [0]=>
  string(1) "h"
  [""]=>
  string(1) "j"
  [2]=>
  string(1) "l"
}

```

Цей приклад включає всі варіації перетворення ключів та перезапису елементів.

### Приклад № 20 Доступ до елементів масиву

Доступ до елементів масиву може бути здійснений за допомогою синтаксису `array[key]`.

```

<?php
$array = array(
  "foo" => "bar",
  42 => 24,
  "multi" => array(
    "dimensional" => array(
      "array" => "foo"
    )
  )
);

var_dump($array["foo"]);
var_dump($array[42]);
var_dump($array["multi"]["dimensional"]["array"]);
?>

```

Результат виконання цього прикладу:

```

string(3) "bar"
int(24)
string(3) "foo"

```

Зауваження:

До PHP 8.0.0 квадратні та фігурні дужки могли використовуватися взаємозамінно для доступу до елементів масиву (наприклад, у прикладі вище `$array[42]` і `$array{42}` робили те саме). Синтаксис фігурних дужок застарів у PHP 7.4.0 і більше не підтримується у PHP 8.0.0.

### Приклад № 21 Розіменування масиву

```
<?php
function getArray() {
    return array(1, 2, 3);
}

$secondElement = getArray()[1];
?>
```

Зауваження:

Спроба доступу до невизначеного ключа в масиві – це те саме, що й спроба доступу до будь-якої іншої невизначеної змінної: буде згенеровано помилку рівня `E_WARNING` (помилка рівня `E_NOTICE` до PHP 8.0.0), і результат буде `null`.

*Створення/модифікація за допомогою синтаксису квадратних дужок.*

Існуючий масив може бути змінений шляхом явної установки значень у ньому.

Це виконується присвоєнням значень масиву (`array`) із зазначенням у дужках ключа. Крім того, ключ можна опустити, в результаті вийде порожня пара дужок `[]`.

```
$arr[key] = $value ;
$arr[] = $value ;
// key може бути int або string
// value може бути будь-яким значенням будь-якого типу
```

Якщо масив `$arr` ще не існує або для нього встановлено значення `null` або `false`, він буде створений. Таким чином, це ще один спосіб визначити масив `array`. Однак такий спосіб застосовувати не рекомендується, тому що якщо змінна `$arr` вже містить певне значення (наприклад, значення типу `string` із змінної запиту), то це значення залишиться на місці і `[]` може означати доступ до символу в рядку. Краще ініціалізувати змінну шляхом явного надання значення.

Примітка:

Починаючи з PHP 7.1.0, використовуючи оператор "порожній індекс" на рядку, призведе до фатальної помилки. Раніше, у цьому випадку, рядок мовчки перетворювався на масив.

Для зміни певного значення просто надайте нове значення елементу, використовуючи його ключ. Якщо ви хочете видалити пару ключ/значення, вам необхідно використовувати функцію `unset()`.

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56; // У цьому місці скрипту це
            // те ж саме, що і $arr[13] = 56;

$arr["x"] = 42; // Додаємо до масиву новий
               // елемент із ключем "x"

unset($arr[5]); // Видаляємо елемент із масиву

unset($arr); // Видаляємо масив повністю
?>
```

Зауваження:

Як вже говорилося вище, якщо ключ не був вказаний, то буде взято максимальний з існуючих цілих (int) індексів, і новим ключем буде це максимальне значення (в крайньому випадку 0) плюс 1. Якщо цілих (int) індексів ще немає, то ключем буде 0(нуль).

Врахуйте, що максимальне значення ключа не обов'язково існує в масиві в даний момент. Воно могло просто існувати в масиві якийсь час, відколи він був переіндексований востаннє. Наступний приклад це ілюструє:

```
<?php
// Створюємо простий масив.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Тепер видаляємо кожен елемент, але сам масив залишаємо
недоторканим:
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Додаємо елемент (зверніть увагу, що новим ключем буде 5,
замість 0).
$array[] = 6;
```

```

print_r($array);

// Переіндексація:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>

```

Результат виконання цього прикладу:

```

Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
)
Array
(
)
Array
(
    [5] => 6
)
Array
(
    [0] => 6
    [1] => 7
)

```

## Приклад № 22 Деструктуризація масиву

Деструктуризація — поділ складної структури на прості частини. Масиви можуть бути деструктуровані за допомогою мовних конструкцій `[]` (починаючи з PHP 7.1.0) або `list()`. Ці конструкції можуть бути використані для деструктуризації масиву в окремі змінні.

```

<?php
$source_array = ['foo', 'bar', 'baz'];
[$foo, $bar, $baz] = $source_array;
echo $foo; // виведе "foo"
echo $bar; // виведе "bar"
echo $baz; // виведе "baz"
?>

```

Деструктуризація масиву може бути використана в конструкції `foreach` для деструктуризації багатовимірного масиву під час ітерації по ньому.

```

<?php
$source_array = [
    [1, 'John'],
    [2, 'Jane'],
];

```



```
foreach ($source_array as [$id, $name]) {  
    // логіка роботи з $id і $name  
}  
?>
```

Елементи масиву ігноруватимуться, якщо змінна не вказана.

Деструктуризація масиву завжди починається з індексу 0.

```
<?php  
$source_array = ['foo', 'bar', 'baz'];  
  
// Присвоювання елементу з індексом 2 змінної $baz  
[, , $baz] = $source_array;  
  
echo $baz; // виведе "baz"  
?>
```

Починаючи з PHP 7.1.0, асоціативні масиви можуть бути деструктуровані. Це дозволяє легше вибирати потрібний елемент у масивах з числовим індексом, оскільки індекс може бути явно вказаний.

```
<?php  
$source_array = ['foo' => 1, 'bar' => 2, 'baz' => 3];  
  
// Присвоювання елементу з індексом 'baz' змінної $three  
['baz' => $three] = $source_array;  
  
echo $three; // виведе 3  
  
$source_array = ['foo', 'bar', 'baz'];  
  
// Присвоювання елементу з індексом 2 змінної $baz  
[2 => $baz] = $source_array;  
  
echo $baz; // виведе "baz"  
?>
```

Деструктуризація масиву може бути використана для простої заміни двох змінних місць.

```
<?php  
$a = 1;  
$b = 2;  
[$b, $a] = [$a, $b];  
echo $a; // виведе 2  
echo $b; // виведе 1  
?>
```

*Що можна і не можна робити з масивами.*

Чому \$foo[bar] не так?

Завжди укладайте в лапки рядковий літерал в індексі асоціативного масиву. Наприклад, пишiть `$foo['bar']`, а чи не `$foo[bar]`. Але чому? Часто у старих скриптах можна зустріти наступний синтаксис:

```
<?php
$foo[bar] = 'ворог';
echo $foo[bar];
// і т.д.
?>
```

Це не так, хоч і працює. Причина в тому, що код містить невизначену константу (`bar`), а не рядок (`'bar'`- зверніть увагу на лапки). Це працює, тому що РНР автоматично перетворює "голий рядок" (не укладений у лапки рядок, який не відповідає жодному з відомих символів мови) у рядок зі значенням цього "голоого рядка". Наприклад, якщо константа з ім'ям `bar` не визначена, РНР замінить `bar` на рядок `'bar'` і використовує його.

#### Примітка:

Це не означає, що потрібно завжди укладати ключ у лапки. Немає необхідності укладати в лапки константи або змінні, оскільки це завадить РНР обробляти їх.

```
<?php
error_reporting(E_ALL);
ini_set('display_errors', true);
ini_set('html_errors', false);
// Простой массив:
$array = array(1, 2);
$count = count($array);
for ($i = 0; $i < $count; $i++) {
    echo "\nПеревіряємо $i: \n";
    echo "Погано: ". $array['$i']. "\n";
    echo "Добре: ". $array[$i]. "\n";
    echo "Погано: {$array['$i']}\n";
    echo "Добре: {$array[$i]}\n";
}
?>
```

#### Результат виконання цього прикладу:

```
Перевіряємо 0:
Notice: Undefined index: $i in /path/to/script.html on line 9
Погано:
Добре: 1
Notice: Undefined index: $i in /path/to/script.html on line 11
Погано:
Добре: 1
```

Перевіряємо 1:  
Notice: Undefined index: \$i in /path/to/script.html on line 9  
Погано:  
Добре: 2  
Notice: Undefined index: \$i in /path/to/script.html on line 11  
Погано:  
Добре: 2

Додаткові приклади, які демонструють цей факт:

```
<?php
// Показувати усі помилки
error_reporting(E_ALL);

$arr = array('fruit' => 'apple', 'veggie' => 'carrot');

// Верно
print $arr['fruit']; // apple
print $arr['veggie']; // carrot

// Неправильно. Це працює, але через невизначену константу з
// іменем fruit також викликає помилку PHP рівня E_NOTICE
//
// Notice: Use of undefined constant fruit - assumed 'fruit' in...
print $arr[fruit]; // apple

// Давайте визначимо константу, щоб продемонструвати, що
// відбувається. Ми надамо константі з ім'ям fruit значення
'veggie'.
define('fruit', 'veggie');

// Тепер зверніть увагу на різницю
print $arr['fruit']; // apple
print $arr[fruit]; // carrot

// Усередині рядка це нормально. Усередині рядків константи не
// розглядаються, так що помилки E_NOTICE тут не виникне
print "Hello $arr[fruit]"; // Hello apple

// З одним винятком: фігурні дужки навколо масивів усередині
// рядків дозволяють константам там перебувати
print "Hello {$arr[fruit]}"; // Hello carrot
print "Hello {$arr['fruit']}"; // Hello apple

// Це не буде працювати і викликає помилку обробки, таку як:
// Parse error: parse error, expecting T_STRING' or T_VARIABLE' or
T_NUM_STRING'
// Це, звичайно, також діє з суперглобальними змінними в рядках
print "Hello $arr['fruit']";
print "Hello $_GET['foo']";

// Ще одна можливість - конкатенація
```

```
print "Hello ". $arr['fruit']; // Hello apple
?>
```

Якщо ви переведете `error_reporting` в режим відображення помилок рівня `E_NOTICE` (наприклад, такий як `E_ALL`), ви відразу побачите ці помилки. За замовчуванням `error_reporting` не відображається.

Як зазначено в розділі синтаксис, усередині квадратних дужок (' [ ' i ' ]') має бути вираз. Це означає, що можна писати так:

```
<?php
echo $arr[somefunc($bar)];
?>
```

Це приклад використання значення, що повертається функцією, як індекс масиву. PHP відомі також і константи:

```
<?php
$error_descriptions[E_ERROR] = "Сталася фатальна помилка";
$error_descriptions[E_WARNING] = "PHP повідомляє про
попередження";
$error_descriptions[E_NOTICE] = "Це лише неофіційне зауваження";
?>
```

Зверніть увагу, що `E_ERROR` це такий же вірний ідентифікатор, як і `bar` в першому прикладі. Але останній приклад по суті еквівалентний такому запису:

```
<?php
$error_descriptions[1] = "Сталася фатальна помилка ";
$error_descriptions[2] = "PHP повідомляє про попередження ";
$error_descriptions[8] = "Це лише неофіційне зауваження ";
?>
```

оскільки `E_ERROR` відповідає 1 і т. д.

### Приклад № 23 Перетворення на масив

Для будь-якого з типів `int`, `float`, `string`, `bool` і `resource` перетворення значення в масив дає результатом масив з одним елементом (з індексом 0), що є скалярним значенням. Іншими словами, `(array)$scalarValue` – це точно те саме, що і `array($scalarValue)`.

Якщо ви перетворюєте на масив об'єкт (`object`), ви отримаєте як елементи масиву властивості (змінні-члени) цього об'єкта. Ключами будуть імена змінних-членів, з деякими примітними винятками: цілочисленні властивості стануть недоступними; до закритих полів класу (`private`) попереду буде дописано ім'я класу; до захищених полів класу (`protected`) попереду буде додано символ '\*'. Ці

додані значення з обох сторін мають NULL байти. Неініціалізовані типізовані властивості автоматично відкидаються.

```
<?php
class A {
    private $B;
    protected $C;
    public $D;
    function __construct()
    {
        $this->{1} = null;
    }
}
var_export((array) new A());
?>
```

Результат виконання цього прикладу:

```
array (
  '' => "\0". 'A'. "\0". 'B' => NULL,
  '*' => "\0". '*' . "\0". 'C' => NULL,
  'D' => NULL,
  1 => NULL,
)
```

Це може викликати дещо несподівану поведінку:

```
<?php
class A {
    private $A; // Це стане '\0A\0A'
}
class B extends A {
    private $A; // Це стане '\0B\0A'
    public $AA; // Це стане 'AA'
}
var_dump((array) new B());
?>
```

Результат виконання цього прикладу:

```
array(3) {
  ["BA"]=>
  NULL
  ["AA"]=>
  NULL
  ["AA"]=>
  NULL
}
```

Наведений вище код покаже 2 ключі з ім'ям 'AA', хоча один з них насправді має ім'я '\0A\0A'.

Якщо ви перетворюєте на масив значення null, ви отримаєте порожній масив.

**Приклад № 24** Просте розпакування масиву

Масив із префіксом "..." буде розпакований під час визначення масиву. Тільки масиви та об'єкти, які реалізують інтерфейс Traversable можуть бути розпаковані. Розпакування масиву за допомогою "..." доступне, починаючи з PHP 7.4.0.

Масив можна розпаковувати кілька разів і додавати звичайні елементи до або після оператора "...":

```
<?php
// Використання короткого синтаксису масиву.
// Також працює із синтаксисом array().
$arr1 = [1, 2, 3];
$arr2 = [...$arr1]; // [1, 2, 3]
$arr3 = [0,...$arr1]; // [0, 1, 2, 3]
$arr4 = [...$arr1,...$arr2, 111]; // [1, 2, 3, 1, 2, 3, 111]
$arr5 = [...$arr1,...$arr1]; // [1, 2, 3, 1, 2, 3]
function getArr() {
    return ['a', 'b'];
}
$arr6 = [...getArr(), 'c' => 'd']; // ['a', 'b', 'c' => 'd']
?>
```

### Приклад № 25 Розпакування масиву з дублюючим ключем

Розпакування масиву за допомогою оператора "..." слідує семантиці функції array\_merge(). Тобто пізніші рядкові ключі перезаписують раніші, а цілочисленні ключі перенумеровуються.

```
<?php
// символічний ключ
$arr1 = ["a" => 1];
$arr2 = ["a" => 2];
$arr3 = ["a" => 0,...$arr1,...$arr2];
var_dump($arr3); // ["a" => 2]
// цілочисельний ключ
$arr4 = [1, 2, 3];
$arr5 = [4, 5, 6];
$arr6 = [...$arr4,...$arr5];
var_dump($arr6); // [1, 2, 3, 4, 5, 6]
// Який [0 => 1, 1 => 2, 2 => 3, 3 => 4, 4 => 5, 5 => 6]
// де вихідні цілі ключі не були збережені.
?>
```

Зауваження:

До PHP 8.1 розпакування масиву з символічним ключем не підтримувалося:

```
<?php
$arr1 = [1, 2, 3];
$arr2 = ['a' => 4];
```

```

$sarr3 = [...$sarr1,...$sarr2];
// Помилка: неможливо розпакувати масив з символічними ключами example.php:5
$sarr4 = [1, 2, 3];
$sarr5 = [4, 5];
$sarr6 = [...$sarr4,...$sarr5]; // працює. [1, 2, 3, 4, 5]
?>

```

## Приклад № 26 Гнучкість масивів PHP

```

<?php
// Цей масив
$a = array('color' => 'red',
           'taste' => 'sweet',
           'shape' => 'round',
           'name' => 'apple',
           4 // ключем буде 0
          );

$b = array('a', 'b', 'c');

//...повністю відповідає
$a = array();
$a['color'] = 'red';
$a['taste'] = 'sweet';
$a['shape'] = 'round';
$a['name'] = 'apple';
$a[] = 4; // ключем буде 0

$b = array();
$b[] = 'a';
$b[] = 'b';
$b[] = 'c';

// після виконання цього коду, $a буде масивом
// array('color' => 'red', 'taste' => 'sweet', 'shape' => 'round',
// 'name' => 'apple', 0 => 4), а $b буде
// array(0 => 'a', 1 => 'b', 2 => 'c'), або просто array('a', 'b',
// 'c').
?>

```

## Приклад № 27 Використання array()

```

<?php
// Масив як мапа (властивостей)
$map = array('version' => 4,
            'OS' => 'Linux',
            'lang' => 'english',
            'short_tags' => true
           );

// виключно числові ключі
$array = array(7,
              8,
              0,
              156,
              -10
             );

```

```
// це те саме, що і array(0 => 7, 1 => 8,...)

$switching = array( 10, // ключ = 0
    5 => 6,
    3 => 7,
    'a' => 4,
    11, // ключ = 6 (максимальним числовим індексом було 5)
    '8' => 2, // ключ = 8 (число!)
    '02' => 77, // ключ = '02'
    0 => 12 // значення 10 буде перезаписано на 12
);

// пустий масив
$empty = array();
?>
```

### Приклад № 28 Колекція

```
<?php
$colors = array('red', 'blue', 'green', 'yellow');

foreach ($colors as $color) {
    echo "Вам подобається $color?\n";
}

?>
```

#### Результат виконання цього прикладу:

```
Вам подобається red?
Вам подобається blue?
Вам подобається green?
Вам подобається yellow?
```

### Приклад № 29 Зміна елемента в циклі

Зміна значень масиву безпосередньо можлива шляхом передачі за посиланням.

```
<?php
foreach ($colors as &$color) {
    $color = strtoupper($color);
}
unset($color); /* Це нужно для того, чтобы последующие записи в
$color не меняли последний элемент массива */

print_r($colors);
?>
```

#### Результат виконання цього прикладу:

```
Array
(
    [0] => RED
    [1] => BLUE
    [2] => GREEN
```



```
[3] => YELLOW
)
```

Наступний приклад створює масив, що починається з одиниці.

### Приклад № 30 Індекс, що починається з одиниці

```
<?php
$firstquarter = array(1 => 'Січень', 'Лютий', 'Березень');
print_r($firstquarter);
?>
```

Результат виконання цього прикладу:

```
Array
(
    [1] => 'Січень'
    [2] => 'Лютий'
    [3] => 'Березень'
)
```

### Приклад № 31 Заповнення масиву

```
<?php
// заповнюємо масив усіма елементами з директорії
$handle = opendir('.');
while (false !== ($file = readdir($handle))) {
    $files[] = $file;
}
closedir($handle);
?>
```

### Приклад № 32 Сортування масиву

Ви можете змінювати порядок елементів за допомогою різних функцій сортування. Для додаткової інформації дивіться розділ функції для роботи з масивами. Ви можете підрахувати кількість елементів у масиві за допомогою функції count().

```
<?php
sort($files);
print_r($files);
?>
```

### Приклад № 33 Рекурсивні та багатовимірні масиви

Оскільки значення масиву може бути будь-чим, ним також може бути інший масив. Таким чином ви можете створювати рекурсивні та багатовимірні масиви.

```
<?php
$fruits = array ("fruits" => array ("a" => "апельсин",
    "b" => "банан",
    "c" => "яблуко"
),
```

```

    "numbers" => array (1,
        2,
        3,
        4,
        5,
        6
    ),
    "holes" => array ("перша",
        5 => "друга",
        "третя"
    )
);

// Декілька прикладів доступу до значень попереднього масиву
echo $fruits["holes"][5]; // виведе "друга"
echo $fruits["fruits"]["a"]; // виведе "апельсин"
unset($fruits["holes"][0]); // видалить "перша"

// Створить новий багатовимірний масив
$juices["apple"]["green"] = "good";
?>

```

Зверніть увагу, що при присвоєнні масиву завжди відбувається копіювання значення. Щоб скопіювати масив за посиланням, потрібно використовувати оператор посилання.

```

<?php
$arr1 = array(2, 3);
$arr2 = $arr1;
$arr2[] = 4; // $arr2 змінився,
            // $arr1 все ще array(2, 3)
$arr3 = &$arr1;
$arr3[] = 4; // тепер $arr1 і $arr3 однакові

```

В дійсності можна наводити ще величезну кількість прикладів використання мови PHP, але тут ми розглядаємо основи роботи, тому обмежимося лише тим, що нам потрібно для початку роботи з PHP для загального розуміння.

Додаткову довідкову інформацію можна отримати на ресурсах:

<https://www.php.net/manual/en/index.php>

<https://www.w3schools.com/php/>

<https://w3schoolsua.github.io/php/index.html#gsc.tab=0>

## 5.16 Питання для самоконтролю

1. Сформулюйте переваги мови PHP для створення динамічних

2. сторінок.

Яким чином PHP-скрипти розміщуються на WEB-сторінках?

3. Сформулюйте відмінності в синтаксисі PHP і знайомих вам мов програмування.

4. Назвіть способи отримання доступу до даних із форм у PHP.

5. Які ви знаєте вбудовані константи PHP?

6. Які правила іменування змінних в PHP?

7. Що таке генератори в PHP?

8. Що собою являє масив в PHP?

9. Яким чином дані змінних можна вставляти в рядки?

10.Що відбувається в результаті деструктуризації масиву?

11. Що таке фігурний синтаксис?

## РОЗДІЛ 6 ПАКЕТНИЙ МЕНЕДЖЕР COMPOSER

Мова програмування PHP швидко розвивається з кожним роком. Щомісяця реєструють десятки, або навіть сотні бібліотек для роботи з PHP проектами. В PHP для керування бібліотеками використовують пакетний менеджер Composer. Це потужний і зручний інструмент, він легко справляється з усіма завданнями щодо встановлення та вирішення залежностей у бібліотеках. Цей інструмент дозволяє не тільки встановлювати сторонні пакети, але й оновлювати їх при виході нових версій. Також за допомогою Composer можна легко створювати пакети для бібліотек [10].

### 6.1 Інсталяція Composer

Composer можна встановити локально, в директорії проекту, або глобально. Встановивши Composer глобально, його можна використовувати у будь-якому проекті.

Детальну інструкцію зі скачування інсталятора можна прочитати за посиланням: <https://getcomposer.org/download/>. Після завантаження файлу установки можна приступати до інсталяції Composer [10].

### 6.2 Встановлення на Linux/Unix/macOS

#### Локально

Після правильного завантаження та встановлення інсталятора у директорії проекту з'явиться файл 'composer.phar'.

Щоб запустити Composer локально, достатньо виконати команду в терміналі, перебуваючи у директорії проекту [10].

```
php composer.phar
```

#### Глобально

Для того щоб зробити Composer глобальним і викликати його з будь-якої директорії, достатньо виконати команду в терміналі [10]:

```
mv composer.phar /usr/local/bin/composer
```

Тепер запустіть

```
Composer
```

### 6.3 Встановлення на Windows

Для того щоб встановити Composer глобально для Windows, потрібно виконати наступні кроки:

Створити новий `composer.bat` файл поруч з `composer.phar`:

Використовуючи `cmd.exe`:

```
C:\bin> echo @php "%~dp0composer.phar" %*>composer.bat
```

Використовуючи PowerShell:

```
PS C:\bin> Set-Content composer.bat '@php "%~dp0composer.phar" %*'
```

Додайте каталог до змінного середовища PATH, якщо це ще не зроблено. Для отримання інформації про зміну змінної PATH дивіться статтю за посиланням: <https://www.computerhope.com/issues/ch000549.htm>.

Закрийте поточний термінал. У новому вікні терміналу введіть наступну команду, щоб перевірити роботу Composer:

```
C:\Users\username>composer -V  
Composer version 2.0.12 2021-04-01 10:14:59
```

### 6.4 Синтаксис та опції Composer

Перше, що необхідно сказати, Composer — це консольна утиліта, вона не має графічного інтерфейсу, однак це не робить її гіршою. Її синтаксис досить простий, а подивитися всі опції та команди можна ввівши команду 'composer' у терміналі.

Список усіх опцій та команд можна розглянути нижче [10]:

Найкорисніші:

- **-h** — вивести довідку по утиліті
- **-q** — скорочений варіант виведення
- **-V** — показати версію утиліти
- **-n** — не ставити інтерактивні питання
- **-v, -vv, -vvv** — налаштування подробиці виводу
- **-d** — використовувати вказану робочу директорію

Список команд, що часто використовуються:

- **archive** — архівує поточний проект як бібліотеку для відправки в мережу
- **check-platform-reqs** — перевіряє, чи дотримані системні вимоги

- **create-project** — створює проект на основі пакета в зазначену директорію
- **depends** — виводить залежності пакету
- **dump-autoload** — оновлює систему автозавантаження класів
- **exec** — дозволяє виконувати скрипти із встановлених пакетів
- **init** — створює порожній проект у поточній папці
- **list** — виводить список доступних команд
- **outdated** — виводить список пакетів, для яких є оновлення
- **prohibits** — виводить назви пакетів, які заважають встановити вказаний пакет
- **search** — пошук пакетів у репозиторіях
- **self-update** — оновлення Composer до останньої версії, працює тільки при локальній установці
- **show** — інформація про пакет
- **update** — оновлює всі пакети до найактуальнішої версії

Встановлення проекту на основі пакету

Швидше за все, для майбутніх проектів, ви використовуватимете фреймворки, щоб мати базовий функціонал з коробки. У таких випадках вам знадобиться розгортати проект на основі існуючого пакета, Composer завантажить пакет з репозиторію і просто розпакує його в потрібну вам директорію.

Для цього використовуйте команду **create-project**, наприклад:

```
composer create-project laravel/laravel app_dir
```

де:

- **composer** — звернення до утиліти
- **create-project** — команда
- **laravel/laravel** — пакет фреймворку laravel
- **app\_dir** — директорія, в яку Composer розпакує вказаний пакет

Встановлення пакетів

Щоб інстальювати пакет за допомогою Composer, необхідно використовувати команду **require**. Утиліта встановить вказаний пакет і запише його у файл `composer.json`.

Наприклад, встановимо пакет для зручного виведення змінних:

```
composer require larapack/dd
```

Також можна вказати опцію `--dev` перед написанням назви пакета, щоб пакет був обов'язковим, але використовувався лише у розробці.

Усі доступні пакети можна переглянути в сервісі [Packagist](#).

## 6.5 Оновлення пакетів

Пакети необхідно оновлювати, щоб уникнути уразливостей у проектах та усунення старих проблем з боку підключених бібліотек [10].

Для оновлення всіх пакетів достатньо ввести команду **update**:

```
composer update
```

або

```
composer update -dev
```

## 6.6 Видалення пакетів

Для видалення пакета необхідно ввести команду **remove** та вказати потрібний пакет, також можна додати опцію `-dev`, якщо це пакет для розробників.

```
composer remove larapack/dd
```

Можна також видалити непотрібні пакети у файлі `composer.json` та запустити команду на оновлення.

## 6.7 Скидання автозавантаження

Composer з коробки може бути ролі автозавантажувача класів. Він використовує стандарт PSR-4. Іноді трапляється, що класи закешувалися, і новий клас не помітний у проекті.

У таких випадках необхідно виконати команду скидання:

```
composer dump-autoload
```

або

```
composer du
```

## **6.8 Питання для самоконтролю**

1. Що таке Composer?
2. Як Composer може бути встановленим в операційній системі?
3. Для якої мови програмування застосовується Composer?
4. Які команди Composer є найвживанішими?
5. Як додати залежність для проєкту за допомогою Composer?



## РОЗДІЛ 7 ФРЕЙМВОРК SYMFONY

### 7.1 Основні компоненти фреймворка Symfony

**Symfony** – це потужний фреймворк для розробки веб-додатків на мові програмування PHP. Він надає гнучкість та зручність для розробки сучасних веб-додатків, допомагає вирішувати ряд загальних проблем, забезпечує структуру та стандарти для розробки. В цьому розділі ми розглянемо основи фреймворка Symfony, його компоненти та структуру веб-додатку на основі Symfony.

#### **Bundle**

Bundle – це основний компонент Symfony, що представляє собою набір пов'язаних файлів (коду, шаблонів, стилів, конфігурації тощо), які разом імплементують певну функціональність. Bundle спрощує розробку, тестування та повторне використання коду, а також допомагає організувати структуру проекту.

#### **Контролери**

Контролери відповідають за обробку запитів від користувача, виконують бізнес-логіку та передають дані до відображення (шаблонів). Вони відіграють важливу роль у шаблоні проектування Model-View-Controller (MVC), на якому базується Symfony.

#### **Моделі та сутності**

Моделі та сутності представляють бізнес-логіку додатку та відповідають за взаємодію з базою даних. Вони слугують для створення, зберігання та обробки даних, а також їх перетворення у відповідні формати.

#### **Відображення (шаблони)**

Відображення, або шаблони, відповідають за відтворення даних, отриманих від контролерів, у візуальному вигляді. Вони можуть використовувати шаблонізатори, такі як Twig, для спрощення та підвищення створення візуального відображення даних.

#### **Маршрутизація**

Маршрутизація в Symfony відповідає за визначення шляхів (URL) для доступу до контролерів та передачу запитів від користувачів до відповідних

контролерів. Маршрутизація допомагає організувати структуру URL-адрес та забезпечує гнучкість у керуванні доступом до різних частин веб-додатку.

## **7.2 Структура веб-додатку на Symfony**

### **Каталоги**

Основні каталоги в структурі веб-додатку на Symfony:

- ``src``: містить основний код додатку, включаючи контролери, сутності, репозиторії та інші класи.
- ``templates``: містить шаблони відображення, які відповідають за візуальну частину додатку.
- ``config``: містить файли конфігурації додатку, такі як налаштування бази даних, маршрутизації, сервісів тощо.
- ``public``: містить відкриті ресурси, такі як стилі, скрипти, зображення та інші статичні файли.
- ``var``: містить файли журналів, кешу та інші тимчасові файли.
- ``vendor``: містить залежності та бібліотеки сторонніх розробників, які використовуються у проекті.

## **7.3 Процес запиту**

Коли користувач відправляє запит до веб-додатку на Symfony, наступний процес відбувається:

1. Запит обробляється фронт-контролером (зазвичай ``public/index.php``).
2. Маршрутизація визначає контролер, що відповідає за обробку запиту, на основі URL та HTTP-методу.
3. Контролер виконує бізнес-логіку, використовуючи моделі та сутності, та отримує дані для відображення.
4. Контролер передає отримані дані до відповідного шаблону відображення.
5. Шаблон відображення обробляє дані та генерує візуальний відтворення, яке буде відправлено користувачеві.
6. Відповідь з відображенням даних відправляється користувачеві.

## 7.4 Розгортання проєкту на Symfony

Розгортання проєкту на Symfony включає в себе налаштування середовища, встановлення залежностей та створення базової структури проєкту. В цьому розділі ми розглянемо кроки, необхідні для створення нового проєкту на Symfony з нуля.

### Вимоги до середовища

Для розгортання Symfony потрібно мати налаштоване середовище з встановленим PHP (версії 7.2.5 або вище) та Composer – менеджером залежностей для PHP. Також рекомендується встановити сервер баз даних (наприклад, MySQL або PostgreSQL) та веб-сервер (Apache або Nginx), якщо вони ще не встановлені.

### Встановлення Symfony CLI

Встановіть Symfony CLI (командний рядок) за допомогою наступної команди:

```
curl -sS https://get.symfony.com/cli/installer | bash
```

Після встановлення перемістіть виконавчий файл `symfony` до каталогу з виконавчими файлами, щоб використовувати його як глобальну команду:

```
mv ~/.symfony/bin/symfony /usr/local/bin/symfony
```

### Створення нового проєкту

Створіть новий проєкт Symfony, виконавши наступну команду:

```
symfony new my_project
```

Замініть `my\_project` назвою вашого проєкту. Команда створить новий каталог зі структурою Symfony та встановить необхідні залежності.

### Запуск веб-сервера

Перейдіть до каталогу нового проєкту:

```
cd my_project
```

Запустіть вбудований веб-сервер Symfony за допомогою команди:

```
symfony server:start
```

Це запустить сервер на порту 8000. Відкрийте веб-браузер та перейдіть за адресою `http://localhost:8000`, щоб перевірити, що ваш новий проєкт Symfony працює.

## **7.5 Підсумки до розділу**

Фреймворк Symfony надає потужний інструментарій для розробки веб-додатків на мові програмування PHP. Він включає в себе набір компонентів, таких як Bundle, контролери, моделі, шаблони відображення та маршрутизація, які разом допомагають створювати структуровані, гнучкі та легко розвиваємі веб-додатки. Структура веб-додатку на Symfony допомагає розробникам організувати код, ресурси та конфігурацію, сприяючи ефективній розробці та супроводженню проєктів.

## **7.6 Питання для самоконтролю**

1. Яка основна мета фреймворка Symfony?
2. Що таке Bundle в контексті Symfony, і яку роль він відіграє?
3. Які основні компоненти фреймворка Symfony та їх функції?
4. Яка структура каталогів веб-додатку на Symfony?
5. Які основні кроки процесу обробки запиту в веб-додатку на Symfony?

## РОЗДІЛ 8 WEB-СЕРВЕР НА NODE.JS

WEB-застосунки, написані за клієнтом-серверною архітектурою, працюють за наступною схемою. Клієнт запитує потрібний ресурс у сервера і сервер відправляє ресурс у відповідь. У цій схемі сервер, відповівши на запит, перериває з'єднання.

Така модель ефективна, оскільки кожен запит до сервера споживає ресурси (пам'ять, процесорний час і т.д.). Щоб обробляти кожен наступний запит від клієнта, сервер повинен завершити обробку попереднього.

Чи означає це, що сервер може обробляти лише один запит? Ні, це не так! Коли сервер отримує новий запит, він створює окремий **потік** для його обробки.

*Потік* – це час і ресурси, які CPU виділяє для виконання невеликого блоку інструкцій. З урахуванням сказаного сервер може обробляти кілька запитів одночасно, але тільки по одному на потік (рис. 6). Така модель також називається **thread-per-request model** [11].

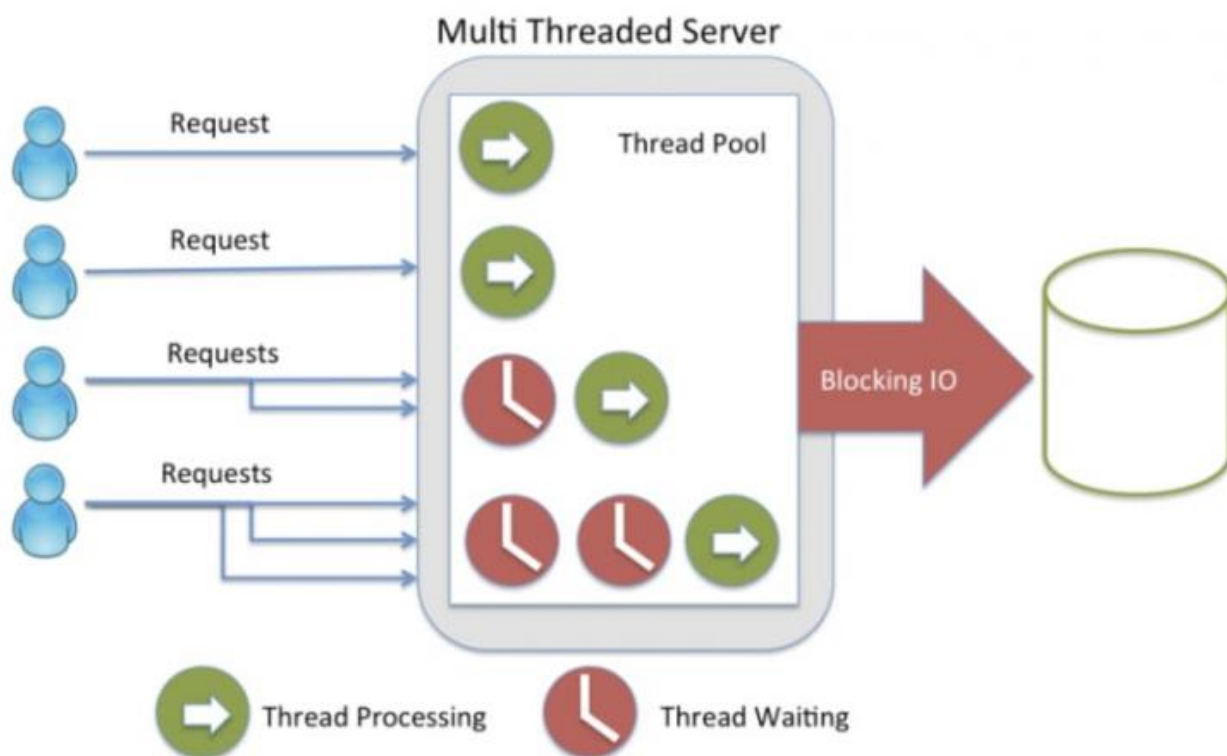


Рисунок 6 – Діаграма потоків обробки запитів сервером

Для обробки  $N$  запитів серверу потрібне  $N$  потоків. Якщо сервер отримує  $N+1$  запитів, тоді він повинен чекати доки один із потоків не стане доступним.

На малюнку вище, сервер може обробляти до 4 запитів (потоків) одночасно і коли він отримує наступні 3 запити, ці запити повинні чекати, доки будь-який з цих 4 потоків не стане доступним.

Один із способів позбутися обмежень – додати більше ресурсів (пам'яті, ядер процесора і т. д.) на сервер, але це не єдиний спосіб. І, звичайно, не забуваємо про технологічні обмеження.

## 8.1 Блокуючі введення/виведення

Обмежена кількість потоків на сервері не єдина проблема. Можливо, Вам стало цікаво, чому один потік не може обробляти кілька запитів одночасно? Все через **блокуючі операції введення/виведення** [11].

Припустимо, Ви розробляєте онлайн магазин і Вам потрібна сторінка, де користувач може переглядати список усіх товарів.

Користувач робить запит на <http://yourstore.com/products> і сервер формує HTML файл з усіма продуктами з бази даних у відповідь. Зовсім не складно, так?

Але що ж відбувається за лаштунками?

- Коли користувач робить запит на `/products` особливий метод або функція повинна виконатися, щоб обробити запит. Маленький фрагмент коду (ваш або фреймворка) аналізує URL-адресу запиту та шукає відповідний метод чи функцію. **Потік працює.** ✓
- Тепер потрібний метод чи функція виконуватимуться. **Потік працює.** ✓
- Так як Ви хороший розробник, Ви зберігаєте всі системні логи у файл, ну і звичайно ж, щоб бути впевненими, що роутер виконує потрібний метод/функцію - Ви так само логуете рядок "Method X executing!!". Але все це блокуючі операції введення/виводу **Потік чекає.** ✗
- Усі логи збережені та наступні рядки функції виконуються. **Потік працює знову.** ✓
- Час звертатися до бази даних і отримувати всі продукти - простий запит, начебто `SELECT * FROM products` виконує свою роботу, але вгадайте що? Так-так, це блокуюча операція введення/виводу. **Потік чекає.** ✗

- Ви отримали масив або список всіх продуктів, але переконайтеся, що ви все це залогували. **Потік чекає. ✗**
- Тепер у Вас є всі продукти і настав час рендерувати шаблон для майбутньої сторінки, але перед цим Вам потрібно їх прочитати. **Потік чекає. ✗**
- Шаблонізатор рендерингу робить свою роботу і надсилає відповідь клієнту. **Потік працює знову. ✓**
- Нарешті потік звільняється.

На скільки повільні операції введення/виводу? Це залежить від конкретної операції (табл. 2) [11].

Таблиця 2 – Ресурси процесора для блокуючих операцій

Операція	Кількість тактів CPU
CPU Registers	3
L1 Cache	8
L2 Cache	12
RAM	150
Disk	30 000 000
Network	250 000 000

Операції мережі та читання з диска надто повільні. Уявіть, скільки запитів або звернень до зовнішніх API ваша система могла б обробити за цей час.

Підбиваючи підсумки: операції введення/виводу змушують потік чекати й витрачати ресурси марно.

## 8.2 Проблема C10K

**C10k** (англ. *C10k*; *10k connections* – проблема 10 тисяч з'єднань)

У ранні 2000-і серверні та клієнтські машини були повільними. Проблема виникала при паралельній обробці 10 000 клієнтських з'єднань до однієї машини [11].

Але чому традиційна модель thread-per-request (потік на запит) не могла вирішити цієї проблеми? Що ж, давайте використовуємо трохи математики.

Нативна реалізація потоків виділяє більше 1 Мб пам'яті на потік, виходячи з цього – для 10 тисяч потоків потрібно 10 Гб оперативної пам'яті, і це лише для стека потоків. Так, і не забувайте, ми на початку 2000-х!

У наші дні серверні та клієнтські комп'ютери працюють швидше та ефективніше і майже будь-яка мова програмування або фреймворк справляються з цією проблемою. Але фактично проблема не вичерпана. Для 10 мільйонів клієнтських з'єднань до однієї машини проблема повертається знову (але тепер [C10M Problem](#)).

### 8.3 Програмна платформа Node.js

Node.js дійсно вирішує проблему C10K... але як?!

Серверний JavaScript не був чимось новим і незвичайним на початку 2000-х, на той момент вже існували продажі поверх JVM (java virtual machine) – RingoJS та AppEngineJS, що працювали на моделі thread-per-request.

Але якщо вони не змогли вирішити проблему, тоді як Node.js зміг? Все тому, що JavaScript **однопоточний**.

### 8.4 Node.js та цикл подій

Node.js це серверна платформа, що працює на механізмі Google Chrome-V8, який здатен компілювати JavaScript код в машинний код [11].

Node.js використовує модель, орієнтовану на події, і архітектуру **неблокуючого введення/виведення**, що робить його стрімким і ефективним. Це не фреймворк і не бібліотека, це середовище виконання JavaScript, програмна платформа.

Давайте напишемо маленький приклад:

```
// Importing native http module
const http = require('http');
```



```
// Creating a server instance where every call
// the message 'Hello World' is responded to the client
const server = http.createServer(function(request, response) {
  response.write('Hello World');
  response.end();
});

// Listening port 8080
server.listen(8080);
```

### *Non-blocking I/O*

Node.js використовує операції, що не блокують введення/виведення, що ж це означає:

- Головний потік не блокуватиметься операціями введення/виводу.
- Сервер продовжуватиме обслуговувати запити.
- Нам доведеться працювати з **асинхронним кодом**.

Давайте напишемо приклад, у якому на запит до /homeсервер у відповідь надсилає HTML сторінку, а для всіх інших запитів - 'Hello World'. Щоб відіслати HTML сторінку спочатку її потрібно прочитати з файлу.

home.html

```
<html>
  <body>
    <h1>This is home page</h1>
  </body>
</html>
```

index.js

```
const http = require('http');
const fs = require('fs');

const server = http.createServer(function(request, response) {
  if (request.url === '/home') {
    fs.readFile(`${__dirname}/home.html`, function(err, content) {
      if (!err) {
        response.setHeader('Content-Type', 'text/html');
        response.write(content);
      } else {
        response.statusCode = 500;
        response.write('An error has occurred');
      }
    });
  } else {
    response.write('Hello World');
    response.end();
  }
});
```

```
}  
});
```

```
server.listen(8080);
```

Якщо запитувана url-адреса /home, тоді використовується нативний модуль fs для читання файлу home.html.

Функції що потрапляють у http.createServer і fs.readFile як аргументи – **колбеки** (callbacks). Ці функції будуть виконані в одному з моментів у майбутньому (Перша, як тільки сервер отримає запит, а друга — коли файл буде прочитаний з диска і поміщений у буфер) [11].

Поки файл зчитується з диска, Node.js може обробляти інші запити і навіть зчитувати файл знову і все це в одному потоці. Давайте розберемося, як це працює.

## 8.5 Цикл подій

Цикл подій – це буквально нескінченний цикл всередині Node.js, і насправді один потік [11]. Фрагмент циклу подій Node.js наведений на рис. 7.

The image shows a snippet of C++ code from the Node.js source, specifically the 'StartNodeInstance' function. It features a 'do-while' loop that repeatedly runs the event loop. The code includes comments in green and function calls like 'uv\_run' and 'uv\_loop\_alive'. A yellow circle highlights the start of the loop, and a bracket on the left indicates the loop's structure. The file name 'node.cc' is visible in the top right corner.

```
// Entry point for new node instances, ... node.cc  
static void StartNodeInstance(void* arg) {  
    // ...  
    {  
        SealHandleScope seal(isolate);  
        bool more;  
        do {  
            v8::platform::PumpMessageLoop(default_platform, isolate);  
            more = uv_run(env->event_loop(), UV_RUN_ONCE);  
  
            if (more == false) {  
                v8::platform::PumpMessageLoop(default_platform, isolate);  
                EmitBeforeExit(env);  
  
                // Emit `beforeExit` if the loop became alive either after emitting  
                // event, or after running some callbacks.  
                more = uv_loop_alive(env->event_loop());  
                if (uv_run(env->event_loop(), UV_RUN_NOWAIT) != 0)  
                    more = true;  
            }  
        } while (more == true);  
    }  
    // ...  
}
```

Рисунок 7 – Фрагмент циклу подій Node.js

**Libuv** – C бібліотека, яка реалізує цей патерн і є частиною ядра Node.js.

Цикл подій має 6 фаз, кожне виконання всіх 6 фаз називають **tick**-ом (рис. 8).

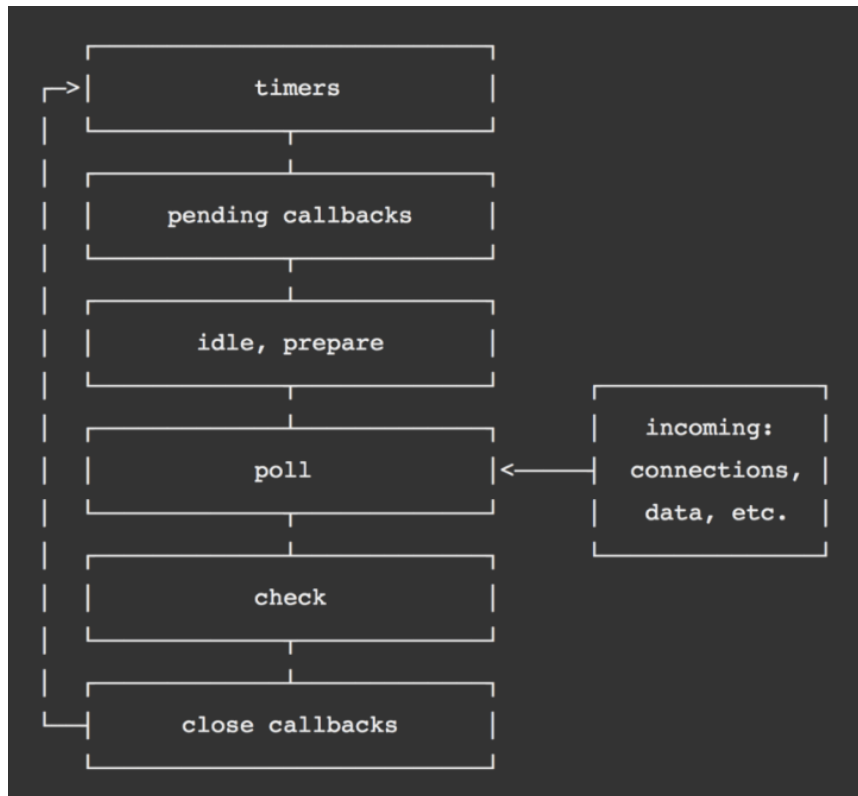


Рисунок 8 – Фази циклу подій

- **timers:** у цій фазі виконуються callback-функції, заплановані методами `setTimeout()` та `setInterval()`;
- **pending callbacks:** виконуються майже всі callback-функції, за винятком подій `close`, таймерів та `setImmediate()`;
- **idle, prepare:** використовується лише для внутрішніх цілей;
- **poll:** відповідальний за отримання нових подій вводу/виводу. Node.js може блокуватись на цьому етапі;
- **check:** callback-функції, викликані методом `setImmediate()`, виконуються на цьому етапі;
- **close callbacks :** наприклад, `socket.on('close',...)`;

Зверніть увагу

Коли циклу подій потрібно виконати операцію введення/виводу він використовує потік ОС з пулу потоку (thread pool), а коли завдання виконано, callback ставиться в чергу під час фази *pending callbacks* (рис. 9).

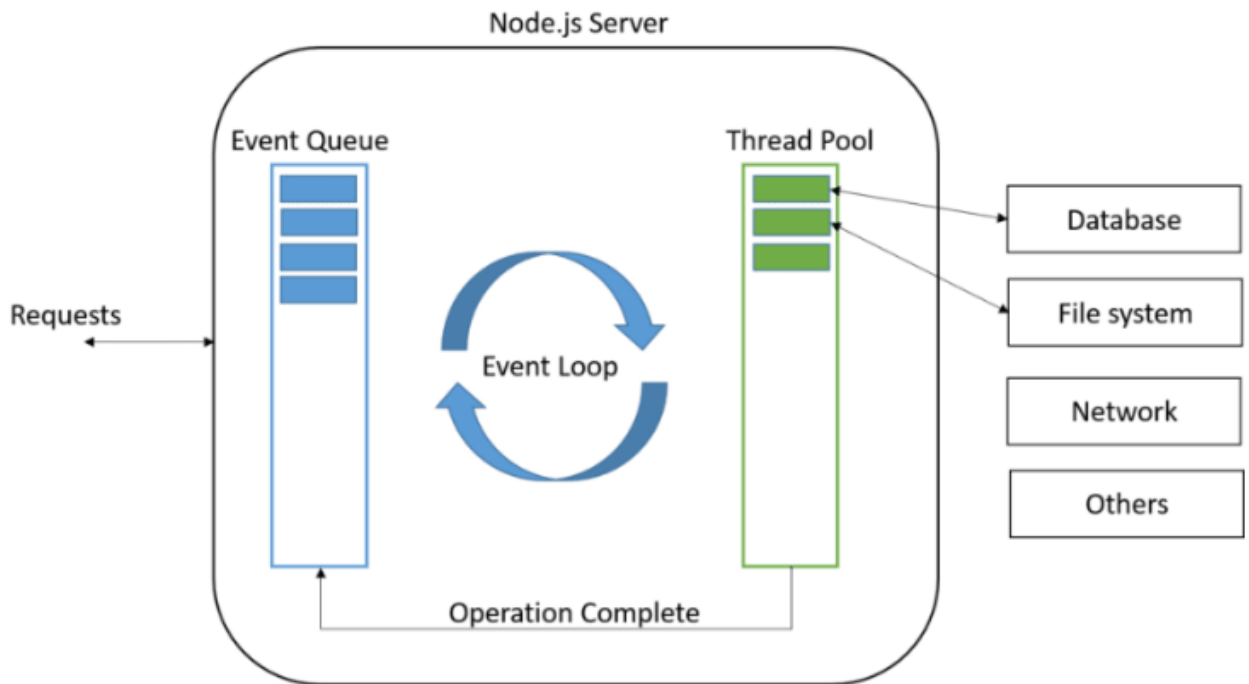


Рисунок 9 – Черга callback пулу потоку

## 8.6 Проблема CPU-смих завдань

Node.js здається ідеальним! Ви можете створювати все, що захочете. Давайте напишемо API для обчислень простих чисел.

Просте число – це ціле (натуральне) число більше одиниці і поділяється тільки на 1 і на себе (рис. 10).

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Рисунок 10 – Матриця простих чисел

Дано число N, API повинен обчислювати та повертати перші N простих чисел у список (або масив).

primes.js

```
function isPrime(n) {
  for(let i = 2, s = Math.sqrt(n); i <= s; i++) {
    if(n % i === 0) return false;
  }
  return n > 1;
}

function nthPrime(n) {
  let counter = n;
  let iterator = 2;
  let result = [];

  while(counter > 0) {
    isPrime(iterator) && result.push(iterator) && counter--;
    iterator++;
  }

  return result;
}

module.exports = { isPrime, nthPrime };
```

index.js

```
const http = require('http');
const url = require('url');
const primes = require('./primes');

const server = http.createServer(function (request, response) {
  const { pathname, query } = url.parse(request.url, true);

  if (pathname === '/primes') {
    const result = primes.nthPrime(query.n || 0);
    response.setHeader('Content-Type', 'application/json');
    response.write(JSON.stringify(result));
    response.end();
  } else {
    response.statusCode = 404;
    response.write('Not Found');
    response.end();
  }
});

server.listen(8080);
```

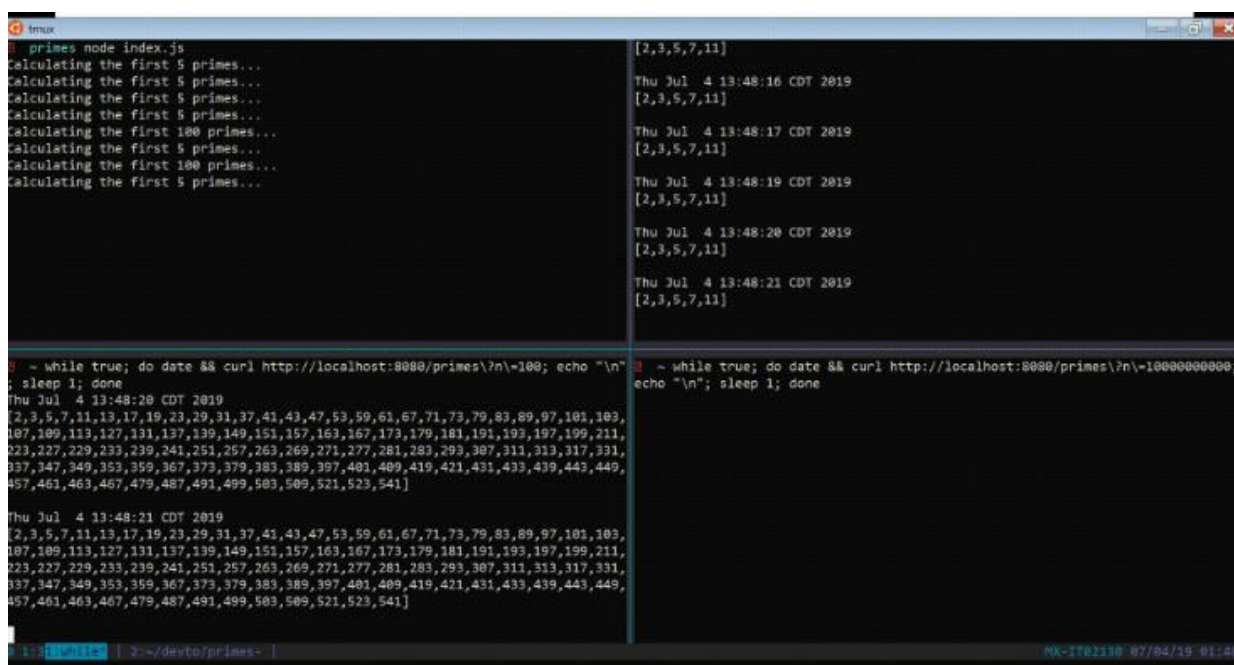
prime.js це реалізація необхідних обчислень: функція isPrime перевіряє чи число простим, а nthPrime повертає N таких чисел.

Файл `index.js` відповідає за створення сервера і використовує модуль `prime.js` для обробки кожного запиту на `/primes`. Число `N` прокидається через рядок запиту в URL-адресі.

Щоб отримати перші 20 простих чисел нам потрібно зробити запит на `http://localhost:8080/primes?n=20`.

Припустимо, до нас стукають 3 клієнти і намагаються отримати доступ до нашого API, що не блокується введенням/виводом (див. рис. 11):

- Перший запитує 5 простих чисел кожному секунду.
- Другий запитує 1000 простих чисел кожному секунду
- Третій запитує 10 000 000 000 простих чисел, але...



```
primes node index.js
calculating the first 5 primes...
calculating the first 5 primes...
calculating the first 5 primes...
calculating the first 5 primes...
calculating the first 100 primes...
calculating the first 5 primes...
calculating the first 100 primes...
calculating the first 5 primes...
[2,3,5,7,11]
Thu Jul 4 13:48:16 CDT 2019
[2,3,5,7,11]
Thu Jul 4 13:48:17 CDT 2019
[2,3,5,7,11]
Thu Jul 4 13:48:19 CDT 2019
[2,3,5,7,11]
Thu Jul 4 13:48:20 CDT 2019
[2,3,5,7,11]
Thu Jul 4 13:48:21 CDT 2019
[2,3,5,7,11]
~ while true; do date && curl http://localhost:8080/primes?n=100; echo "\n"; sleep 1; done
Thu Jul 4 13:48:20 CDT 2019
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541]
Thu Jul 4 13:48:21 CDT 2019
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541]
```

Рисунок 11 – Блокування головного потоку

Коли третій клієнт надсилає запит – головний потік блокується і це головна ознака проблеми **CPU-ємних завдань**. Коли головний потік зайнятий виконанням важкої задачі він стає недоступним для інших завдань.

Але як щодо `libuv`? Якщо Ви пам'ятаєте, ця бібліотека допомагає Node.js виконувати операції введення/виводу за допомогою потоків ОС уникаючи блокування головного потоку і Ви абсолютно праві, це вирішення нашої проблеми, але для того, щоб це стало можливим, наш модуль повинен бути написаний мовою C++, щоб `libuv` могла з ним працювати.

На щастя, починаючи з v10.5 до Node.js доданий нативний модуль **Worker Threads**.

## 8.7 Воркери та їх потоки

Воркери корисні для виконання CPU-ємних JavaScript операцій; не використовуйте їх для операцій вводу/виводу, вже вбудовані в Node.js механізми більш ефективно справляються з такими завданнями, ніж Worker thread.

### *Виправлення коду*

Настав час переписати наш код:

```
primes-workerthreads.js
```

```
const { workerData, parentPort } = require('worker_threads');
```

```
function isPrime(n) {  
  for(let i = 2, s = Math.sqrt(n); i <= s; i++)  
    if(n % i === 0) return false;  
  return n > 1;  
}
```

```
function nthPrime(n) {  
  let counter = n;  
  let iterator = 2;  
  let result = [];  
  
  while(counter > 0) {  
    isPrime(iterator) && result.push(iterator) && counter--;  
    iterator++;  
  }  
  
  return result;  
}
```

```
return result;  
}
```

```
parentPort.postMessage(nthPrime(workerData.n));
```

```
index-workerthreads.js
```

```
const http = require('http');  
const url = require('url');  
const { Worker } = require('worker_threads');
```

```
const server = http.createServer(function (request, response) {  
  const { pathname, query } = url.parse(request.url, true);
```

```
  if (pathname === '/primes') {  
    const worker = new Worker('./primes-workerthreads.js', { workerData: { n:  
      query.n || 0 } });
```

```
    worker.on('error', function () {  
      response.statusCode = 500;  
      response.write('Oops there was an error...');  
      response.end();  
    });
```

```
    let result;  
    worker.on('message', function (message) {
```

```

    result = message;
  });

worker.on('exit', function () {
  response.setHeader('Content-Type', 'application/json');
  response.write(JSON.stringify(result));
  response.end();
});
} else {
  response.statusCode = 404;
  response.write('Not Found');
  response.end();
}
});

server.listen(8080);

```

У файлі `index-workerthreads.js` при кожному запиті `/primes` створюється екземпляр класу `Worker` (з нативного модуля `worker_threads`) для вивантаження і виконання файлу `primes-workerthreads.js` в потік воркера. Коли список простих чисел прорахований і готовий, ініціюється подія `message` – результат потрапляє в головний потік через те, що у воркера не залишилося роботи, він також ініціює подію `exit`, дозволяючи основному потоку надсилати дані клієнту.

`primes-workerthreads.js` змінено небагато. Він імпортує `workerData` (це копія параметрів, переданих з основного потоку) і `parentPort` через який результат роботи воркера передається у головний потік.

Тепер давайте випробуємо наш приклад знову і подивимося, що станеться (див. рис. 12).

```

tmux
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 1000000000 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...
Calculating the first 5 primes...
Calculating the first 100 primes...

457,461,463,467,479,487,491,499,503,509,521,523,541]
Thu Jul 4 14:52:49 CDT 2019
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,
107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,
223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331,
337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,
457,461,463,467,479,487,491,499,503,509,521,523,541]

Thu Jul 4 14:52:50 CDT 2019
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,
107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,
223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331,
337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,
457,461,463,467,479,487,491,499,503,509,521,523,541]

Thu Jul 4 14:52:44 CDT 2019
[2,3,5,7,11]
Thu Jul 4 14:52:46 CDT 2019
[2,3,5,7,11]
Thu Jul 4 14:52:47 CDT 2019
[2,3,5,7,11]
Thu Jul 4 14:52:48 CDT 2019
[2,3,5,7,11]
Thu Jul 4 14:52:49 CDT 2019
[2,3,5,7,11]
Thu Jul 4 14:52:50 CDT 2019

~ while true; do date && curl http://localhost:8080/primes?\n~1000000000;
echo "\n"; sleep 1; done
Thu Jul 4 14:52:47 CDT 2019

```



## Рисунок 12 – Робота основного потоку

Основний потік більше не блокується

Тепер все працює як треба, але зловживати воркерами без жодних на те причин все ж таки не найкраща практика, створювати потоки не дешево задоволення. Обов'язково створіть пул потоків перед цим.

### 8.8 Директиви `Import` та `Export`

#### `Import`

Зазвичай ми маємо список того, що хочемо імпортувати, у фігурних дужках `import {...}`, наприклад:

```
import {sayHi, sayBye} from './say.js';
sayHi('John'); // Hello, John!
sayBye('John'); // Bye, John!
```

Але якщо імпортувати потрібно багато чого, ми можемо імпортувати все відразу як об'єкта, використовуючи `import * as <obj>`. Наприклад:

```
import * as say from './say.js';
say.sayHi('John');
say.sayBye('John');
```

На перший погляд, «імпортувати все» виглядає дуже зручно, не треба писати зайвого, навіть нам взагалі може знадобитися явно перераховувати список того, що потрібно імпортувати?

На це є кілька причин.

1. Сучасні інструменти складання (`webpack` та інші) збирають модулі разом і оптимізують їх, прискорюючи завантаження і видаляючи код, що не використовується.

Припустимо, ми додали до нашого проекту сторонню бібліотеку `say.js` з безліччю функцій:

```
export function sayHi() { ... }
export function sayBye() { ... }
export function becomeSilent() { ... }
```

Тепер, якщо з цієї бібліотеки у проєкті ми використовуємо лише одну функцію:

```
import {sayHi} from './say.js';
```

Тоді оптимізатор побачить, що інші функції не використовуються, і видалить інші із зібраного коду, тим самим роблячи код меншим. Це називається "tree-shaking".

2. Очевидно, перераховуючи те, що хочемо імпортувати, ми отримуємо більш короткі імена функцій: `sayHi()` замість `say.sayHi()`.
3. Явний перелік імпортів робить код зрозумілішим, дозволяє побачити, що саме і де використовується. Це спрощує підтримку та рефакторинг коду.

## Export

Ми можемо позначити будь-яке оголошення як експортоване, розмістивши `export` перед ним, будь то змінна, функція або клас.

Наприклад, всі наступні екпорти допустимі:

```
// экспорт массива
export let months = ['Jan', 'Feb', 'Mar', 'Apr', 'Aug', 'Sep',
'Oct', 'Nov', 'Dec'];

// экспорт константы
export const MODULES_BECAME_STANDARD_YEAR = 2015;

// экспорт класса
export class User {
  constructor(name) {
    this.name = name;
  }
}
```

*Експорт окремо від оголошення.*

Можна також написати `export` окремо. Тут ми спочатку оголошуємо, а потім екпортуємо:

```
function sayHi(user) {
  alert(`Hello, ${user}!`);
}

function sayBye(user) {
  alert(`Bye, ${user}!`);
}

export {sayHi, sayBye};
```

Або, технічно, ми також можемо розмістити `export` вище функції.

## 8.9 Cookie в node.js

Приклад використання Cookie в node.js:

```

const express = require('express'),
  app = express(),
  cookieParser = require('cookie-parser')

const host = '127.0.0.1'
const port = 7000

app.use(cookieParser('secret key'))

app.get('/get-cookie', (req, res) => {
  console.log('Cookie: ', req.cookies)
  res.send('Get Cookie')
})

app.get('/set-cookie', (req, res) => {
  res.cookie('token', '12345ABCDE')
  res.send('Set Cookie')
})

app.listen(port, host, () =>
  console.log(`Server listens http://${host}:${port}`)
)

```

## 8.10 Сесії в node.js

Приклад використання сесій в node.js:

```

const express = require("express");
const app = express();
const session = require("express-session");
const cookieParser = require("cookie-parser");
const PORT = 4000;

// Initialization
app.use(cookieParser());

app.use(session({
  secret: "amar",
  saveUninitialized: true,
  resave: true
}));

// User Object

const user = {
  name: "Amar",
  Roll_number: 43,
  Address: "Pune"
};

// Login page
app.get("/login", (req, res) => {
  req.session.user = user;
  req.session.save();
  return res.send("Your are logged in");
});

```

```
app.get("/user", (req, res) => {
  const sessionuser = req.session.user;
  res.send(sessionuser);
});

// Logout page
app.get("/logout", (req, res) => {
  req.session.destroy();
  res.send("Your are logged out ");
});

// Host
app.listen(PORT, () => console.log(`Server at ${PORT}`));
```

### **8.11 Питання для самоконтролю**

1. Які переваги багатопотокових WEB-серверів?
2. Наведіть приклади блокуючих операцій введення/виведення?
3. Що таке проблема C10K?
4. Що таке Node.js?
5. Що таке callback-функція?
6. Опишіть особливості роботи асинхронного коду?
7. Що таке цикл подій в Node.js?
8. В чому полягає проблема CPU-ємних завдань?
9. Для чого призначені так звані воркери?
10. Наведіть приклади застосування директив Import та Export.

## РОЗДІЛ 9 РОЗГОРТАННЯ ВЕБ-ДОДАТКІВ В МЕРЕЖІ

Розгортання веб-додатків в мережі вимагає від розробників знання технологій, що дозволяють створювати, тестувати та розгортати додатки в різних середовищах з мінімальними зусиллями. В цьому розділі ми розглянемо технології Docker та GitLab CI, які спрощують процес розгортання веб-додатків та сприяють ефективній командній роботі.

Найпростіші сайти, розроблені на мові PHP можна легко розмістити в мережі, використовуючи хостинг-провайдери, наприклад:

- <https://www.ukraine.com.ua/>
- <https://freehost.com.ua/>
- <https://hvesting.ua/>

Розгортання сайту полягає у підключенні доменного ім'я до вашого хостінгу та завантаженні файлів з вашого локального середовища на сервер за допомогою FTP протоколу.

### 9.1 Завантаження файлів на хостинг за допомогою FTP

File Transfer Protocol (FTP) – це стандартний протокол передачі файлів, який дозволяє користувачам обмінюватися файлами між комп'ютерами через мережу. FTP зазвичай використовується для завантаження та завантаження файлів на веб-сервери.

Процес завантаження файлів на хостинг за допомогою FTP:

1. Вибір FTP-клієнта: для роботи з FTP вам потрібен FTP-клієнт, такий як FileZilla, WinSCP або Cyberduck. Встановіть потрібний FTP-клієнт на свій комп'ютер.

2. З'єднання з FTP-сервером: відкрийте свій FTP-клієнт та введіть необхідні дані для підключення до вашого FTP-сервера (адреса сервера, ім'я користувача та пароль). Ці дані зазвичай надаються вашим хостинг-провайдером.

3. Завантаження файлів: після успішного підключення до FTP-сервера ви побачите структуру каталогів на сервері. Знайдіть каталог для вашого сайту

(наприклад, `public_html`) та перетягніть файли вашого веб-додатку з локальної машини в цей каталог. Зачекайте, поки файли будуть завантажені на сервер.

4. Перевірка результату: відкрийте веб-браузер та введіть адресу свого веб-сайту. Ви повинні побачити ваш веб-додаток, який відображається коректно. Якщо виникають проблеми, перевірте правильність завантаження файлів та налаштувань сервера.

### **Оновлення файлів на хостингу**

Для оновлення файлів вашого веб-додатку на хостингу просто повторіть процес завантаження, замінивши файли на сервері новими версіями. Використовуйте FTP-клієнт для цього.

Для запуску проєкту треба створити на хостингу базу даних та прописати логін та пароль до неї у конфігураційному файлі вашого сайту.

## **9.2 Docker**

Docker - це платформа для розробки, доставки та розгортання додатків у контейнерах. Контейнери дозволяють пакувати додаток разом з його залежностями та налаштуваннями, що забезпечує консистентність та ізоляцію середовищ.

### **Основні поняття Docker**

*Docker-образ (image)* – це відтворювана знімка додатку та його залежностей. Образи можна створювати вручну або за допомогою `Dockerfile`.

*Docker-контейнер (container)* – це запущений екземпляр `Docker`-образу, який можна використовувати для виконання додатку.

*Dockerfile* – це текстовий файл, що містить інструкції для створення `Docker`-образу.

*Docker Hub* – це публічний реєстр образів, де розробники можуть завантажувати та скачувати образи.

### **Робота з Docker**

Для інсталювання `Docker`: відвідайте офіційний сайт `Docker` (<https://www.docker.com/>) та завантажте відповідний інсталятор для вашої операційної системи.

Для створення Docker-образу: створіть Dockerfile у вашому проєкті та вкажіть інструкції для налаштування та встановлення залежностей.

Приклад Dockerfile для веб-додатку на PHP:

```
```\nFROM php:7.4-apache\nCOPY . /var/www/html\nRUN docker-php-ext-install mysqli\n```\n
```

Збірка Docker-образу: виконайте команду `docker build -t my-web-app` у каталозі з Dockerfile для збірки образу. Замініть my-web-app` на назву вашого додатку.`

Запуск Docker-контейнера: виконайте команду `docker run -p 8080:80 my-web-app` для запуску контейнера на порту 8080. Замініть my-web-app` на назву вашого додатку.`

Використання Docker Hub: створіть обліковий запис на Docker Hub (<https://hub.docker.com/>), авторизуйтеся в командному рядку (`docker login``) та використовуйте команди `docker push` та docker pull` для завантаження та скачування образів.`

### 9.3 GitLab CI

GitLab CI (Continuous Integration) – це система автоматизації збірки, тестування та розгортання коду, що інтегрована з системою керування версіями GitLab. GitLab CI дозволяє розробникам створювати пайплайни (pipelines) - послідовності робочих процесів, що виконуються автоматично після кожного коміту коду.

#### Основні поняття GitLab CI

*Пайплайн (pipeline)* – це послідовність задач (jobs), що виконуються автоматично після кожного коміту коду.

*Задача (job)* – це окрема дія, що виконується в рамках пайплайну (наприклад, збірка проєкту, тестування, розгортання).

*Раннер (runner)* – це агент, що виконує задачі на окремих машинах або у контейнерах.

`.gitlab-ci.yml` – це файл конфігурації, що містить визначення пайплайнів та задач для GitLab CI.

## Робота з GitLab CI

Створення `.gitlab-ci.yml`: у корені вашого репозиторію GitLab створіть файл `.gitlab-ci.yml` та додайте визначення пайплайнів та задач. Наприклад:

```
...
image: node:latest

stages:
  - build
  - test
  - deploy

build:
  stage: build
  script:
    - npm install
    - npm run build

test:
  stage: test
  script:
    - npm run test

deploy:
  stage: deploy
  script:
    - echo "Deploying to production server"
    - scp -r dist/* user@production-server:/path/to/webapp
...
```

У цьому прикладі ми використовуємо Node.js-додаток, визначаємо три етапи (build, test, deploy) та задачі, що відповідають кожному етапу.

Реєстрація GitLab Runner: для виконання задач потрібно зареєструвати GitLab Runner. Ви можете використовувати глобальні раннери, які надає GitLab, або встановити свій власний раннер на вашому сервері. Детальні інструкції з реєстрації раннера можна знайти в документації GitLab (<https://docs.gitlab.com/runner/register/>).

Моніторинг пайплайнів: після кожного коміту GitLab CI автоматично запускає пайплайн. Ви можете перевірити статус пайплайнів, перейшовши на вкладку "CI/CD" у вашому репозиторії GitLab.



Розгортання з Docker: якщо ваш додаток використовує Docker, ви можете додати кроки для публікації Docker-образів у вашому `.gitlab-ci.yml` файлі. Наприклад:

```
...
docker_publish:
  stage: deploy
  image: docker:latest
  services:
    - docker:dind
  script:
    - docker login -u "$DOCKER_HUB_USERNAME" -p "$DOCKER_HUB_PASSWORD"
    - docker build -t my-web-app:latest .
    - docker push my-web-app:latest
...
```

У цьому прикладі ми використовуємо сервіс Docker-in-Docker, авторизуємося на Docker Hub та завантажуюмо новий образ.

Технології Docker та GitLab CI дозволяють розробникам створювати консистентні та відтворювані середовища для веб-додатків, автоматизувати процес збірки, тестування та розгортання коду. Використання цих технологій сприяє швидкому розвитку проектів, полегшує командну роботу та забезпечує високу якість продукту.

## 9.4 Практичні поради та кращі практики

Поради при роботі з Docker:

- Використовуйте офіційні образи: якщо можливо, використовуйте офіційні Docker-образи для вашої технології, оскільки вони зазвичай відрізняються стабільністю та безпекою.
- Створюйте мінімальні образи: під час створення Docker-образів намагайтеся зменшити їх розмір, видаляючи непотрібні файли та встановлюючи тільки необхідні залежності.
- Використовуйте `.dockerignore`: створіть файл `.dockerignore`, щоб виключити файли та каталоги, які не повинні бути включені в Docker-образ (наприклад, файли розробки, кеш тощо).

- Використовуйте теги для версіонування: використовуйте теги Docker-образів для відслідковування версій вашого додатку. Це дозволить легко відкотити до попередньої версії, якщо виникнуть проблеми.

## **9.5 Поради при роботі з GitLab CI**

- Використовуйте кешування: для зменшення часу виконання пайплайнів використовуйте кешування ресурсів, таких як залежності, артефакти збірки тощо.
- Роздіть задачі на паралельні етапи: розділяйте задачі на паралельні етапи для прискорення процесу виконання пайплайнів. Наприклад, ви можете виконувати тести для різних компонентів вашого додатку паралельно.
- Використовуйте захищені змінні середовища: для зберігання та передачі паролів, ключів доступу та інших чутливих даних використовуйте захищені змінні середовища GitLab CI.
- Створюйте сповіщення про стан пайплайнів: налаштуйте сповіщення про стан пайплайнів для вашої команди, щоб швидко виявляти та вирішувати проблеми, що виникають під час збірки або розгортання вашого додатку.
- Регулярно оновлюйте раннери: слідкуйте за оновленнями GitLab Runner та оновлюйте його регулярно, щоб використовувати нові можливості та виправлення помилок.

## **9.6 Підсумки до розділу**

Використання технологій Docker та GitLab CI значно спрощує розгортання веб-додатків, автоматизацію збірки, тестування та розгортання коду. Вони допомагають командам розробників швидко ітерувати та забезпечувати якість продукту на високому рівні. Застосування кращих практик при використанні Docker та GitLab CI забезпечить ефективність, стабільність та безпеку вашого процесу розгортання.

## **9.7 Питання для самоконтролю**

1. Що таке Docker та які його переваги для розгортання веб-додатків?

2. Як створити Docker-образ для вашого веб-додатку?
3. Що таке GitLab CI та які його можливості для автоматизації збірки, тестування та розгортання коду?
4. Як налаштувати пайплайни GitLab CI для вашого веб-додатку?
5. Як завантажити сайт на звичайний хостинг?

## ПЕРЕЛІК ПОСИЛАНЬ

1. Технології розробки WEB-ресурсів [Електронний ресурс] : навчальний посібник / В. П. Молчанов, О. К. Пандорін. – Харків : ХНЕУ ім. С. Кузнеця, 2019. – 130 с. ISBN 978-966-676-754-0.
2. Chris Shiflett (2003). Http Developer's Handbook 1st Edition. Sams. p.400. ISBN 978-0672324543.
3. Micheal Full (2020). How the Internet Works & the Web Development Process. p. 30. ISBN 979-8641657073.
4. Ерік Фрімен. Head First. Патерни проєктування. ТОВ «Фабула», 2020. – 672с. <https://www.yakaboo.ua/ua/head-first-paterni-proektuvannja.html>
5. Ніколаєнко О.Ю. Використання НІТ у курсі “Технології створення Web-вузлів”//Вісник.-К: НПУ імені М.П. Драгоманова, 2012. - Випуск 3. - С.76-78.
6. Пуа Grigorik (2013). High Performance Browser Networking 1st Edition. O'Reilly Media. p. 398. ISBN 978-1449344764.
7. Огляд популярних мов програмування – Python, Java, PHP, C/C++ та інші. [www.best-work.com.ua](http://www.best-work.com.ua) [Електронний ресурс] – Режим доступу до ресурсу: <https://www.best-work.com.ua/programming-languages/> (дата звернення: 07.04.2023).
8. PHP Manual by Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy, Jakub Vrana and others [Електронний ресурс] – Режим доступу до ресурсу: <https://www.php.net/manual/en/index.php> (дата звернення: 07.04.2023).
9. Р.С. Мартін. Чиста архітектура: Пер. с англ.-К.: ТОВ «Фабула», 2019. - 368с. <https://www.yakaboo.ua/ua/chista-arhitektura.html>.
10. Composer частина 1. Навіщо його використовувати у PHP проєктах та як з ним працювати? <https://blog.ithillel.ua> [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.ithillel.ua/articles/composer-navishho-iogo-vikoristovuvati-u-php-proektax-ta-yak-z-nim-pracyuvati> (дата звернення: 07.04.2023).

11. Все, що вам потрібно знати про Node.js [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/webbdev/js-db3d35ffed7e> (дата звернення: 07.04.2023).