**Зміст**

1. [**PHP, опис та введення** 2](#_Toc153484678)
2. [**Symphony Framework, що це таке та основні характеристики** 2](#_Toc153484679)
3. [**Веб-сервер на Node.js** 3](#_Toc153484680)
4. [**Архітектури програмного забезпечення** 5](#_Toc153484681)
5. [**Принцип роботи веб-додатків** 6](#_Toc153484682)
6. [**Серверна частина та Клієнтська частина** 6](#_Toc153484683)
7. [**SOAP та REST** 7](#_Toc153484684)
8. [**Composer - що це і для чого використовують** 8](#_Toc153484685)
9. [**Впровадження веб-додатків у мережу** 9](#_Toc153484686)

**PHP, опис та введення**

PHP (Hypertext Preprocessor) - це серверна мова програмування, спеціально розроблена для веб-додатків, що надає широкі можливості обробки даних на стороні сервера та генерації веб-сторінок. PHP використовується для інтерактивного взаємодії з користувачем і взаємодії з базами даних.

Його основні характеристики включають інтерпретований характер, можливість вбудовування в HTML, різноманітні типи даних та змінні, а також умовні оператори та цикли для ефективної обробки даних. Окремо варто відзначити можливість роботи з формами та базами даних, що робить PHP потужним інструментом для розробників веб-додатків.

Окрім того, PHP підтримує об'єктно-орієнтоване програмування (ООП), що спрощує розподілення функціональності та даних в програмі. Робота з зображеннями, файлами і HTTP-запитами розширює можливості мови, надаючи розробникам широкий функціонал для створення високопродуктивних веб-додатків.

Завершуючи, PHP відкриває можливості для включення готових бібліотек і розширень, що дозволяє розробникам ефективно використовувати готові рішення та розширювати функціональні можливості мови. Його міжнародна підтримка різних кодувань і мов дозволяє створювати багатомовні веб-додатки для різних аудиторій.

**Symphony Framework, що це таке та основні характеристики**

Symfony - це високопродуктивний PHP-фреймворк для розробки веб-застосунків, який базується на принципах моделі-вигляд-контролер (Model-View-Controller, MVC). Фреймворк був розроблений SensioLabs і випущений під відкритою ліцензією MIT. Symfony забезпечує розширюваність, гнучкість та стандартизацію для розробки великих та складних веб-застосунків.

*Основні характеристики Symfony:*

1. Модульність та Компонентна Архітектура: Symfony побудований на основі компонентної архітектури, що дозволяє вам використовувати лише ті частини фреймворку, які вам потрібні. Це сприяє високій перевикористовуваності коду та полегшує тестування.
2. MVC Архітектура: Symfony використовує модель-вигляд-контролер, що розділяє логіку застосунку на три основні компоненти. Це полегшує організацію коду та робить застосунок більш масштабованим та обслуговуваним.
3. ORM (Object-Relational Mapping): Symfony має вбудовану підтримку для Doctrine, одного з найпопулярніших ORM для PHP. Це дозволяє працювати з базою даних за допомогою об'єктно-орієнтованого підходу, спрощуючи взаємодію з даними.
4. Twig - Сучасний Движок Шаблонів: Symfony використовує Twig як движок шаблонів. Twig надає простий та зрозумілий синтаксис для роботи з HTML-шаблонами, а також дозволяє використовувати успадкування шаблонів, макроси та інші зручні функції.
5. Конфігураційні Файли та Контейнер Об'єктів: Symfony використовує конфігураційні файли у форматі YAML або XML для налаштування застосунку. Контейнер об'єктів Symfony дозволяє вам керувати залежностями та ефективно взаємодіяти з різними компонентами застосунку.
6. Вбудовані Засоби для Тестування: Symfony враховує тестування як важливу частину розробки і надає зручні інструменти для написання автоматизованих тестів. Це сприяє створенню надійного та стабільного коду.

Symfony використовується для розробки різноманітних веб-застосунків, включаючи сайти, CMS (системи управління контентом), CRM (системи управління відносинами з клієнтами), API та інші великі та складні проекти. Фреймворк надає розробникам потужний інструментарій для швидкої та ефективної розробки веб-застосунків на мові PHP.

**Веб-сервер на Node.js**

***Програмна платформа Node.js***

Node.js - це виконавче середовище, яке забезпечує можливість виконання JavaScript коду на сервері. Основні характеристики та аспекти Node.js включають:

*Движок V8 від Google Chrome*

Node.js побудовано на движку V8, який розроблений Google для використання в їхньому веб-браузері Chrome. V8 відомий своєю високою швидкодією виконання JavaScript коду. Використовуючи V8, Node.js може виконувати код швидко та ефективно.

*Виконання JavaScript на сервері*

Однією з ключових переваг Node.js є можливість виконання JavaScript коду на сервері, що раніше було характерно тільки для браузерів. Це розширює можливості мови та дозволяє розробникам використовувати той самий мовний стек і для клієнтської, і для серверної частини додатків.

*Асинхронність та Неблокуючий І/O*

Однією з основних концепцій Node.js є асинхронність. Це означає, що він може обробляти багато запитів одночасно без чекання завершення кожного запиту. Також, Node.js використовує неблокуючий ввід/вивід (I/O), що дозволяє ефективно обробляти багато одночасних операцій вводу/виводу, таких як читання/запис до файлів або взаємодія з базою даних.

*Модульна система*

Node.js має потужну модульну систему, яка дозволяє розробникам легко організовувати свій код у вигляді модулів. Модулі можна використовувати для організації та використання функціоналу, а також для подальшого використання в інших проектах.

*Спільнота та Екосистема*

Node.js має широку та активну спільноту розробників, що призводить до розвитку численних сторонніх бібліотек та інструментів. Це створює багатий екосистем, який полегшує розробку, відлагодження та розгортання додатків.

Node.js є основним інструментом для розробки високопродуктивних веб-серверів та застосунків, і він надає розробникам широкі можливості для створення ефективних та масштабованих додатків.

***Node.js та цикл подій***

**Node.js** представляє виконавче середовище, побудоване на движку V8 JavaScript від Google Chrome. Однією з ключових його особливостей є використання \*\*асинхронного програмування\*\* та \*\*циклу подій\*\* для ефективної обробки багатьох запитів одночасно. Цей підхід ґрунтується на технології **non-blocking I/O**, що дозволяє обробляти багато операцій без очікування завершення кожної з них перед переходом до наступної.

У зв'язку з цим, коли Node.js отримує запит від клієнта, він викликає **зворотний виклик (callback)** і продовжує виконання інших завдань, не чекаючи завершення обробки даного запиту. Це дозволяє оптимально використовувати системні ресурси та забезпечує високу продуктивність в ситуаціях з великою кількістю одночасних з'єднань, що є актуальним для сучасних веб-систем.

Загалом, **Node.js** став популярним вибором для розробки веб-серверів, адже його асинхронна природа сприяє створенню швидких і масштабованих застосунків, забезпечуючи ефективну обробку багатьох паралельних операцій.

***Cookie та сесії в Node.js***

Node.js - це середовище виконання, побудоване на движку V8 JavaScript від Google Chrome. Воно дозволяє виконувати JavaScript на сервері, що робить його популярним для створення високопродуктивних веб-серверів. Однією з ключових особливостей є підтримка асинхронного програмування та циклу подій. Це дозволяє обробляти багато запитів одночасно, не блокуючи виконання інших задач.

Цикл подій Node.js дозволяє ефективно обробляти багато подій, використовуючи асинхронний підхід. При отриманні запиту від клієнта, Node.js викликає зворотний виклик (callback) та обробляє запит, не зупиняючи виконання інших завдань. Це робить його добре підходящим для завдань, які вимагають великої кількості одночасних операцій, таких як обробка багатьох з'єднань.

У веб-розробці зазвичай виникає потреба у збереженні стану між різними запитами. Для цього використовуються куки та сесії.

Куки - це невеликі файли, які сервер відправляє браузеру для зберігання інформації, яку браузер повертає з кожним наступним запитом.

Сесії - це механізм для зберігання даних на сервері для ідентифікації користувача між багатьма запитами. Бібліотека `express-session` в Node.js дозволяє ефективно керувати сесіями та забезпечити ідентифікацію користувачів та збереження стану між різними відвідуваннями.

**Архітектури програмного забезпечення**

Сервісна архітектура є стратегією розробки програмного забезпечення, яка акцентується на розбитті програмного додатка на відокремлені компоненти, що відомі як сервіси. Ці сервіси функціонують як автономні одиниці, не залежать один від одного та взаємодіють через мережеві протоколи. Основні принципи мікросервісної архітектури включають розгортання додатка в невеликі, самостійні сервіси, кожен з яких відповідає за конкретний фрагмент функціональності. Взаємодія між цими сервісами здійснюється через інтерфейси програмування застосунків (API), такі як HTTP або інші протоколи.

Мікросервісна архітектура має численні переваги, серед яких варто виділити масштабованість, незалежність розробки та впровадження, більшу гнучкість у виборі технологій для кожного сервісу, здатність до відновлення після збоїв та полегшений моніторинг. У порівнянні з традиційним підходом до розробки - монолітною архітектурою, яка об'єднує всі функції в одному блоку, мікросервіси забезпечують більшу гнучкість та можливість розвивати окремі компоненти незалежно.

Процес переходу від монолітної архітектури до мікросервісної включає аналіз потреб, розробку API для ефективної взаємодії сервісів та поступове впровадження нового функціоналу у вигляді окремих сервісів, поступово відокремлюючи їх від монолітного додатка. Цей підхід дозволяє системі органічно адаптуватися до нової, більш гнучкої та масштабованої архітектури.

**Принцип роботи веб-додатків**

**HTTP**-протокол використовується веб-додатками для обміну даними між клієнтом (зазвичай веб-браузером) та сервером. Цей протокол визначає правила передачі та отримання інформації через Інтернет.

*Клієнт і сервер*

Веб-додаток складається з двох ключових частин: клієнтської (front-end) та серверної (back-end). Клієнт відповідає за відображення та взаємодію з користувачем через веб-браузер, тоді як сервер обробляє запити, виконує бізнес-логіку та взаємодіє з базами даних і іншими послугами.

*Відправка HTTP-запитів*

Клієнт може надсилати HTTP-запити до сервера, запитуючи різні ресурси, такі як веб-сторінки, дані чи зображення. Запити включають тип (GET, POST, PUT, DELETE), URL та інші параметри.

*Обробка запитів на сервері*

Сервер отримує запити, обробляє їх і генерує відповіді. Цей процес може включати доступ до бази даних, обчислення, обробку даних та інші операції.

*Відправлення HTTP-відповідей*

Сервер надсилає HTTP-відповіді назад до клієнта. Відповіді можуть містити HTML-код для відображення сторінки, дані у форматі JSON або інші необхідні дані.

*Відображення сторінки*

Клієнтська частина (front-end) відповідає за графічне відображення веб-сторінок, обробку подій користувача і взаємодію з сервером, створюючи повноцінний веб-досвід..

**Серверна частина та Клієнтська частина**

Back-end представляє собою серверну частину веб-додатка, яка відповідає за обробку запитів, виконання бізнес-логіки, взаємодію з базами даних та надання даних для відображення на front-end. Основна функція back-end - це забезпечити коректну та ефективну роботу всього додатка.

Основні завдання back-end включають:

1. Обробка запитів: Приймання та обробка HTTP-запитів від клієнтів, надсилання запитів до баз даних, виконання розрахунків і логіки додатка.
2. Взаємодія з базою даних: Зберігання і отримання даних з баз даних, що дозволяє додатку зберігати та отримувати необхідну інформацію.
3. Бізнес-логіка: Виконання логіки, яка визначає, як додаток повинен відповідати на різні сценарії та запити.
4. Автентифікація та авторизація: Забезпечення безпеки та визначення доступу користувачів до різних ресурсів додатка.

Front-end відповідає за взаємодію з користувачем, відображення контенту та створення користувацького інтерфейсу на веб-сторінці.

Основні завдання front-end включають:

1. Відображення інтерфейсу: Розміщення та відображення графічних елементів, тексту та інших компонентів на веб-сторінці.
2. Взаємодія з користувачем: Обробка подій користувача, таких як кліки, введення тексту тощо.
3. Взаємодія з back-end: Відправлення запитів до back-end для отримання та відправлення даних, необхідних для відображення та оновлення сторінки.
4. Оптимізація продуктивності: Забезпечення ефективності та швидкодії веб-сторінки, адаптація для різних пристроїв та браузерів.

Важливо враховувати, що back-end та front-end взаємодіють, утворюючи єдиний функціональний веб-додаток для кінцевого користувача.

**SOAP та REST**

SOAP є протоколом обміну повідомленнями, спроектованим для обміну структурованими інформаційними повідомленнями в розподіленому середовищі. Основною його особливістю є використання XML для форматування повідомлень. В основі SOAP лежить ідея використання специфікацій для визначення структури повідомлень та правил обміну.

*Різниця між SOAP і REST:*

1. ПротоколSOAP використовує HTTP, SMTP, TCP, а також інші протоколи, тоді як REST зазвичай використовує тільки HTTP.
2. Формат повідомленьSOAP використовує XML для форматування повідомлень, в той час як REST може використовувати різні формати, такі як XML, JSON, HTML тощо.
3. Структура SOAP визначає чітку структуру повідомлень за допомогою XML-схем, тоді як REST розглядається як архітектурний стиль і не накладає таких обмежень на структуру повідомлень.

*Переваги SOAP:*

1. Стійкість до мережевих проблем: SOAP призначений для використання в розподілених системах і добре переноситься через різні мережеві протоколи.
2. Безпека: SOAP має вбудовані стандарти безпеки, такі як WS-Security, що надає високий рівень безпеки для обміну повідомленнями.

*Недоліки SOAP:*

1. ВажкістьSOAP-повідомлення мають великий обсяг через використання XML, що може призводити до зайвого навантаження на мережу.
2. ПовільністьЗавдяки великому обсягу інформації та високому рівню безпеки, SOAP може бути повільнішим у порівнянні з іншими протоколами, такими як REST.

REST є стилем архітектури для розробки веб-служб і використовує простий інтерфейс заснований на стандартних протоколах, зокрема HTTP. Він призначений для ефективного взаємодії між компонентами системи.

*Переваги REST:*

1. Простота: REST використовує простий та легко зрозумілий інтерфейс, що спрощує розробку та розуміння коду.
2. Швидкодія: Завдяки використанню легковагових форматів, таких як JSON, REST може бути ефективнішим щодо швидкодії порівняно з SOAP.

*Недоліки REST:*

1. Відсутність стандартів безпеки: REST не має вбудованих стандартів безпеки, і безпеку слід враховувати окремо в процесі розробки.

2. Обмежена функціональність: У порівнянні з SOAP, REST може бути менш потужним та обмеженим у функціональності, оскільки він не накладає жорстких обмежень на структуру повідомлень.

**Composer - що це і для чого використовують**

Composer є невід'ємною частиною PHP-розробки, забезпечуючи управління залежностями у проектах. Основною метою Composer є спрощення процесу встановлення та оновлення бібліотек і фреймворків, використовуваних у проектах. У файлі composer.json описуються всі залежності проекту, зокрема ключові елементи, такі як "require" (необхідні бібліотеки та їх версії), "require-dev" (залежності для розробки) та "autoload" (налаштування автозавантаження класів).

Щоб встановити Composer, потрібно завантажити composer.phar з офіційного сайту та розмістити його у каталозі /usr/local/bin/, щоб забезпечити глобальний доступ. Файл composer.json є конфігураційним файлом, в якому визначаються параметри проекту.

Команди Composer, такі як "install" та "update," використовуються для управління залежностями. "Composer update" дозволяє оновлювати залежності відповідно до вказаних версій у файлі composer.json. Видалення пакетів здійснюється за допомогою команди "remove," яка також оновлює файл автозавантаження класів.

Для перегенерації файлу автозавантаження класів використовується команда "dump-autoload." Це особливо корисно, якщо відбулися зміни у структурі або розташуванні класів, дозволяючи Composer знаходити та завантажувати їх з оновленого розташування.

**Впровадження веб-додатків у мережу**

В процесі розгортання веб-додатків в мережі існують різні стратегії, визначені великим обсягом різноманітних інструментів та підходів. Один із підходів - використання хостинг-провайдерів, таких як AWS, Google Cloud або Microsoft Azure. Обравши провайдера та зареєструвавшись, ви завантажуєте свій веб-додаток на сервер через SSH, FTP або інші зручні засоби. Це вимагає належної настройки сервера враховуючи потреби самого додатка.

У випадку використання FTP, встановлюєте FTP-клієнт, наприклад, FileZilla або WinSCP, на локальну машину. Після отримання FTP-доступу від хостинг-провайдера та введення необхідних даних (адреса сервера, ім'я користувача, пароль) у FTP-клієнт, ви зручно передаєте файли на сервер. Просто перетягніть файли з локальної папки до віддаленої папки на сервері. Після завантаження виконайте необхідні налаштування сервера через SSH або інші доступні засоби для забезпечення коректної роботи веб-додатка.

Однак існує й альтернативний підхід, який користується великою популярністю - використання Docker. Docker надає стандартизоване середовище, де весь веб-додаток та його залежності знаходяться в ізольованому контейнері. Завдяки Dockerfile, де описана конфігурація образу, та використанню команд типу `docker build` та `docker run`, процес створення та запуску відокремленого середовища стає швидким та зручним.

Розгортання через Docker дозволяє забезпечити високий рівень ізоляції, впорядкованість та легкість масштабування. Після встановлення Docker Engine та створення Docker-образу, ви можете легко запустити та управляти контейнерами, забезпечуючи консистентність середовища в різних умовах.

Цей процес є ключовим в арсеналі розробників та інженерів для розгортання високопродуктивних та масштабованих веб-додатків. Не залежно від обраного методу, розгортання веб-додатків в мережі вимагає від розробників ретельної підготовки, використання ефективних інструментів та впевненості у здатності їх додатків ефективно функціонувати в обраному середовищі.