**TUGAS**

**Kegiatan 1**

PHP Standards Recommendations

1. **PSR-1 (**[**Basic Coding Standard**](https://www.php-fig.org/psr/psr-1)**)**

Jika semua developer menggunakan standar yang sama, maka akan mudah menindahkan modul tanpa masalah.

1. setiap penulisan kode php harus diawali dengan tags <?php atau <?=
2. setiap penulisan method atau fungsi tidak mengikut sertakan konfigurasi yang bersifat merubah atau memodifikasi konfiruasi standar yang ada dalam php.ini
3. sebaiknya tidak langsung memanggil file menggunakan fungsi include
4. sebaiknya tidak menuliskan tags html atau javascript yang bersifat mempengaruhi output
5. sebaiknya tidak memodifikasi variable global
6. sebaiknya tidak menuliskan fungsi atau method tanpa memeriksa apakah fungsi itu tersedia atau tidak.
7. penggunaan namespaces dan clasess atau mengacu PSR-0 atau PSR-4

Contoh :

PenamaanConstanta

class ClassName

{

const VERSION = '1.0';

const DATE\_APPROVED = '2012-06-01';

}

1. **PSR – 2** [**Coding Style Guide**](https://www.php-fig.org/psr/psr-2)

Pada PSR yang kedua ini kita lebih difokuskan pada kerapian kode sehingga mudah dibaca baik oleh diri kita sendiri atau orang lain.

1. Mengikuti Aturan coding php PSR-1.
2. indent (penulisan yang menjorok kedalam layaknya paragraf) menggunakan 4 (empat) spasi bukan Tab.
3. Menulis kode jangan telalu panjang. usahakan batas maksimal sekitar 120 karakter atau lebih baik kurang dari 80 karakter.
4. Memberi garis baru (blank line) setelah mendeklarasikan namespace atau use (atau banyak use).
5. Penulisan Kurung buka “{” pada sebuah Classes dan Methods HARUS diberi garis baru (new line) dan  kurung tutup “}” juga diberikan satu garis baru.
6. Visibility (private, protected, public) HARUS ke semua Propertis dan method, abstract dan final HARUS dideklarasikan sebelum Visibility, dan static HARUS dideklarasikan sesudah Visibility.
7. Control Structure Keyword HARUS diberi spasi sebelum kurung buka “{“.
8. Kurung Buka “{” untuk Control Structure ditulis pada garis yang sama dan untuk penutup “}” ditulis disetelah body.
9. Setiap Arguments pada method HARUS diberi satu spasi setelah koma dan argumen yang memiliki nilai default SEBAIKNYA ditaruh di deret paling belakang.

Contoh :

Dekalarsi Namespace dan Use

<?php

namespace Vendor\Package;

use FooInterface;

use BarClass as Bar;

use OtherVendor\OtherPackage\BazClass;

// kode lain

1. **PSR -3** [**Logger Interface**](https://www.php-fig.org/psr/psr-3)

Peraturan PSR-3 untuk logging, dan secara khusus memaparkan sembilan metode untuk tata cara menulis log pada library framework. Hal ini membuat pemangkasan universal untuk semua kerangka, yang berarti akan mempermudah ketika ingin menerapkan logging dalam framework yang sebelumnya tidak mempunyai fungsi tersebut.

1. LoggerInterface memperlihatkan delapan metode untuk menulis log ke dalam RFC 5424 (debug, info, notice, warning, error, critical, alert, emergency).
2. Metode kesembilan (log), menerima tingkat log sebagai argumen pertama. Pemanggilan metode ini dengan salah satu dari tingkat konstanta log HARUS memiliki hasil yang sama seperti pemanggilan metode tingkatspesifik. Pemanggilan metode ini dengan tidak disertai konstanta level oleh HARUS mengeluarkan Psr\Log\InvalidArgumentException jika pelaksanaannya tidak berhasil. Pengguna SEBAIKNYA TIDAK menggunakan tingkat log kustom tanpa mengetahui pasti cara pengimplementasiannya.

Contoh :

{

“require”: {

“psr/log”: “dev-master”

}

}

1. **PSR-4** [**Autoloading Standard**](https://www.php-fig.org/psr/psr-4)

Acuan yang satu ini berkaitan dengan cara pemanggilan atau cara me-load sebuah class atau atauran penggunaan Namespaces (nanti akan kita bahas), aturan-aturan yang ada pada PSR-0 berkaitan dengan Autoloading, misalnya seperti

1. namespace dan class harus memiliki struktur \<Vendor>\(<Namespaces>)\\*ClassName
2. setiap namespaces harus memiliki nama Vendor
3. setiap namespaces bisa memiliki sub-namespaces tak terbatas
4. setiap namespaces haris dikonversi menjadi DIRECTORY\_SEPARATOR
5. setiap namespaces harus memiliki akhira file .php
6. setiap nama Vendor  harus berupa Alphabetic  kombinasi dari Huruf besar dan huruf kecil

Contoh :

<?php

function autoload($className)

{

    $className  = ltrim($className, '\\');

    $filename   = '';

    $namespace  = '';

    if($lastNsPos  = strpos($className, '\\')){

        $namespace = substr($className, 0, $lastNsPos);

        $className = substr($className, $lastNsPos + 1);

        $fileName  = str\_replace('\\', DIRECTORY\_SEPARATOR, $namespace) . DIRECTORY\_SEPARATOR;

    }

    $fileName .= str\_replace('\_', DIRECTORY\_SEPARATOR, $className) . '.php';

    require $fileName;

}

spl\_autoload\_register('autoload');

1. **PSR – 6** [**Caching Interface**](https://www.php-fig.org/psr/psr-6)

Tujuan

Tujuan dari PSR ini adalah untuk memungkinkan pengembang untuk membuat pustaka cache-aware it dapat diintegrasikan ke dalam framework dan sistem yang ada tanpa perlu pengembangan kustom.

Definisi

Memanggil libraries - libraries atau kode yang benar-benar membutuhkan cache jasa. libraries ini akan memanfaatkan layanan cache yang mengimplementasikan ini interfaces standar, tetapi tidak memiliki pengetahuan tentang itu implementasi layanan cache tersebut.

1. Implementing Library - libraries ini bertanggung jawab untuk menerapkan standar ini untuk menyediakan layanan cache ke libraries Panggilan.
2. Itu Menerapkan libraries HARUS menyediakan class yang mengimplementasikan

Cache \ CacheItemPoolInterface dan Cache \ CacheItem interfaces interfaces.

1. Menerapkan libraries HARUS mendukung pada fungsi TTL minimum seperti yang dijelaskan di bawah ini dengan granularitas keseluruhan.

TTL - The Time To Live (TTL) dari suatu item adalah jumlah waktu antara ketika barang itu disimpan dan dianggap basi. TTL biasanya didefinisikan oleh integer yang mewakili waktu dalam detik, atau objek DateInterval.

Kedaluwarsa - Waktu aktual saat item disetel menjadi basi. Ini adalah biasanya dihitung dengan menambahkan TTL ke waktu ketika suatu objek disimpan, tetapi mungkin juga secara eksplisit diatur dengan objek DateTime. Item dengan TTL 300 detik disimpan pada 1:30:00 akan memiliki waktu berakhir 1:35:00. Menerapkan libraries MUNGKIN kadaluarsa item sebelum Waktu Kedaluwarsa yang diminta, tetapi HARUS memperlakukan item sebagai kedaluwarsa setelah Waktu Kedaluwarsa tercapai. Jika libraries panggilan meminta sebuah item yang akan disimpan tetapi tidak menentukan waktu kedaluwarsa, atau menentukan nol waktu kedaluwarsa atau TTL, sebuah libraries Pelaksana MUNGKIN menggunakan default yang dikonfigurasi lamanya. Jika tidak ada durasi default yang ditetapkan, Pustaka Penerapan HARUS menafsirkannya sebagai permintaan untuk menyimpan item itu selamanya, atau selam mendukung implementasi yang mendasarinya.

Kunci - String setidaknya satu karakter yang secara unik mengidentifikasi a item cache. Menerapkan libraries HARUS mendukung kunci yang terdiri dari karakter A-Z, a-z, 0-9, \_, dan. dalam urutan apa pun dalam enkode UTF-8 dan a panjang hingga 64 karakter. Menerapkan libraries MUNGKIN mendukung tambahan karakter dan penyandian atau panjang yang lebih panjang, tetapi harus mendukung setidaknya itu minimum. libraries bertanggung jawab atas pelarian diri mereka sendiri dari string kunci sebagaimana mestinya, tetapi HARUS dapat mengembalikan string kunci asli yang tidak dimodifikasi. Karakter berikut disediakan untuk ekstensi masa depan dan TIDAK HARUS didukung oleh pelaksana pustaka: {} () / \ @:

Klik - Tembolok cache terjadi ketika Pustaka Memanggil meminta kunci Item by dan nilai yang cocok ditemukan untuk kunci itu, dan nilai itu belum kedaluwarsa, dan nilainya tidak valid karena beberapa alasan lain. Memanggil libraries HARUS membuat Pastikan untuk memverifikasi isHit () pada semua panggilan get ().

Miss - Cache miss adalah kebalikan dari cache hit. Cache miss terjadi ketika Pustaka Memanggil meminta item berdasarkan kunci dan nilai itu tidak ditemukan untuk itu

kunci, atau nilai ditemukan tetapi telah kedaluwarsa, atau nilainya tidak berlaku untuk sebagian Alasan lain. Nilai kedaluwarsa HARUS selalu dianggap sebagai cache miss.

Ditangguhkan - Penabung cache yang ditangguhkan menunjukkan bahwa item cache tidak boleh bertahan dengan segera di tepi kolam renang. Objek Pool MUNGKIN menunda penundaan item cache untuk mengambil keuntungan dari operasi kumpulan massal yang didukung oleh beberapa mesin penyimpanan. Sebuah Pool HARUS memastikan bahwa setiap item cache yang ditangguhkan pada akhirnya bertahan dan data tidak hilang, dan MUNGKIN tetap ada sebelum Memanggil libraries meminta agar mereka bertahan. Ketika Pustaka Memanggil memanggil komit () metode semua item yang ditangguhkan harus HARUS bertahan. libraries Implementasi MUNGKIN menggunakan logika apa pun yang sesuai untuk menentukan kapan harus bertahan ditangguhkan item, seperti destruktor objek, bertahan semua di save (), batas waktu atau pemeriksaan max-item atau logika lain yang sesuai. Meminta item cache itu telah ditangguhkan HARUS mengembalikan barang yang ditangguhkan tetapi belum-tetap.

<?php

namespace Psr\Cache;

/\*\*  
 \* CacheItemInterface defines an interface for interacting with objects inside a cache.  
 \*/  
interface CacheItemInterface  
{  
 /\*\*  
 \* Returns the key for the current cache item.  
 \*  
 \* The key is loaded by the Implementing Library, but should be available to  
 \* the higher level callers when needed.  
 \*  
 \* @return string  
 \* The key string for this cache item.  
 \*/public function getKey();

/\*\*  
 \* Retrieves the value of the item from the cache associated with this object's key.  
 \*  
 \* The value returned must be identical to the value originally stored by set().  
 \*  
 \* If isHit() returns false, this method MUST return null. Note that null  
 \* is a legitimate cached value, so the isHit() method SHOULD be used to  
 \* differentiate between "null value was found" and "no value was found."  
 \*  
 \* @return mixed  
 \* The value corresponding to this cache item's key, or null if not found.  
 \*/public function get();

/\*\*  
 \* Confirms if the cache item lookup resulted in a cache hit.  
 \*  
 \* Note: This method MUST NOT have a race condition between calling isHit()  
 \* and calling get().  
 \*  
 \* @return bool  
 \* True if the request resulted in a cache hit. False otherwise.  
 \*/public function isHit();

/\*\*  
 \* Sets the value represented by this cache item.  
 \*  
 \* The $value argument may be any item that can be serialized by PHP,  
 \* although the method of serialization is left up to the Implementing  
 \* Library.  
 \*  
 \* @param mixed $value  
 \* The serializable value to be stored.  
 \*  
 \* @return static  
 \* The invoked object.  
 \*/public function set($value);

/\*\*  
 \* Sets the expiration time for this cache item.  
 \*  
 \* @param \DateTimeInterface|null $expiration  
 \* The point in time after which the item MUST be considered expired.  
 \* If null is passed explicitly, a default value MAY be used. If none is set,  
 \* the value should be stored permanently or for as long as the  
 \* implementation allows.  
 \*  
 \* @return static  
 \* The called object.  
 \*/public function expiresAt($expiration);

/\*\*  
 \* Sets the expiration time for this cache item.  
 \*  
 \* @param int|\DateInterval|null $time  
 \* The period of time from the present after which the item MUST be considered  
 \* expired. An integer parameter is understood to be the time in seconds until  
 \* expiration. If null is passed explicitly, a default value MAY be used.  
 \* If none is set, the value should be stored permanently or for as long as the  
 \* implementation allows.  
 \*  
 \* @return static  
 \* The called object.  
 \*/public function expiresAfter($time);

}

<?php

namespace Psr\Cache;

/\*\*  
 \* CacheItemPoolInterface generates CacheItemInterface objects.  
 \*/  
interface CacheItemPoolInterface  
{  
 /\*\*  
 \* Returns a Cache Item representing the specified key.  
 \*  
 \* This method must always return a CacheItemInterface object, even in case of  
 \* a cache miss. It MUST NOT return null.  
 \*  
 \* @param string $key  
 \* The key for which to return the corresponding Cache Item.  
 \*  
 \* @throws InvalidArgumentException  
 \* If the $key string is not a legal value a \Psr\Cache\InvalidArgumentException  
 \* MUST be thrown.  
 \*  
 \* @return CacheItemInterface  
 \* The corresponding Cache Item.  
 \*/public function getItem($key);

/\*\*  
 \* Returns a traversable set of cache items.  
 \*  
 \* @param string[] $keys  
 \* An indexed array of keys of items to retrieve.  
 \*  
 \* @throws InvalidArgumentException  
 \* If any of the keys in $keys are not a legal value a \Psr\Cache\InvalidArgumentException  
 \* MUST be thrown.  
 \*  
 \* @return array|\Traversable  
 \* A traversable collection of Cache Items keyed by the cache keys of  
 \* each item. A Cache item will be returned for each key, even if that  
 \* key is not found. However, if no keys are specified then an empty  
 \* traversable MUST be returned instead.  
 \*/public function getItems(array $keys = array());

/\*\*  
 \* Confirms if the cache contains specified cache item.  
 \*  
 \* Note: This method MAY avoid retrieving the cached value for performance reasons.  
 \* This could result in a race condition with CacheItemInterface::get(). To avoid  
 \* such situation use CacheItemInterface::isHit() instead.  
 \*  
 \* @param string $key  
 \* The key for which to check existence.  
 \*  
 \* @throws InvalidArgumentException  
 \* If the $key string is not a legal value a \Psr\Cache\InvalidArgumentException  
 \* MUST be thrown.  
 \*  
 \* @return bool  
 \* True if item exists in the cache, false otherwise.  
 \*/public function hasItem($key);

/\*\*  
 \* Deletes all items in the pool.  
 \*  
 \* @return bool  
 \* True if the pool was successfully cleared. False if there was an error.  
 \*/public function clear();

/\*\*  
 \* Removes the item from the pool.  
 \*  
 \* @param string $key  
 \* The key to delete.  
 \*  
 \* @throws InvalidArgumentException  
 \* If the $key string is not a legal value a \Psr\Cache\InvalidArgumentException  
 \* MUST be thrown.  
 \*  
 \* @return bool  
 \* True if the item was successfully removed. False if there was an error.  
 \*/public function deleteItem($key);

/\*\*  
 \* Removes multiple items from the pool.  
 \*  
 \* @param string[] $keys  
 \* An array of keys that should be removed from the pool.

\* @throws InvalidArgumentException  
 \* If any of the keys in $keys are not a legal value a \Psr\Cache\InvalidArgumentException  
 \* MUST be thrown.  
 \*  
 \* @return bool  
 \* True if the items were successfully removed. False if there was an error.  
 \*/public function deleteItems(array $keys);

/\*\*  
 \* Persists a cache item immediately.  
 \*  
 \* @param CacheItemInterface $item  
 \* The cache item to save.  
 \*  
 \* @return bool  
 \* True if the item was successfully persisted. False if there was an error.  
 \*/public function save(CacheItemInterface $item);

/\*\*  
 \* Sets a cache item to be persisted later.  
 \*  
 \* @param CacheItemInterface $item  
 \* The cache item to save.  
 \*  
 \* @return bool  
 \* False if the item could not be queued or if a commit was attempted and failed. True otherwise.  
 \*/public function saveDeferred(CacheItemInterface $item);

/\*\*  
 \* Persists any deferred cache items.  
 \*  
 \* @return bool  
 \* True if all not-yet-saved items were successfully saved or there were none. False otherwise.  
 \*/public function commit();  
}

1. **PSR – 7** [**HTTP Message Interface**](https://www.php-fig.org/psr/psr-7)

Proposal ini menggambarkan interfaces umum untuk mewakili pesan HTTP. Pesan HTTP adalah permintaan dari klien ke server, atau tanggapan dari server ke klien. Spesifikasi ini mendefinisikan interfaces untuk pesan :

interface MessageInterface  
{  
 public function getProtocolVersion();  
 public function withProtocolVersion($version);  
 public function getHeaders();  
 public function hasHeader($name);  
 public function getHeader($name);  
 public function getHeaderLine($name);  
 public function withHeader($name, $value);  
 public function withAddedHeader($name, $value);  
 public function withoutHeader($name);  
 public function getBody();  
 public function withBody(StreamInterface $body);  
}

interface RequestInterface extends MessageInterface  
{  
 public function getRequestTarget();  
 public function withRequestTarget($requestTarget);  
 public function getMethod();  
 public function withMethod($method);  
 public function getUri();  
 public function withUri(UriInterface $uri, $preserveHost = false);  
}

interface ServerRequestInterface extends RequestInterface  
{  
 public function getServerParams();  
 public function getCookieParams();  
 public function withCookieParams(array $cookies);  
 public function getQueryParams();  
 public function withQueryParams(array $query);  
 public function getUploadedFiles();  
 public function withUploadedFiles(array $uploadedFiles);  
 public function getParsedBody();  
 public function withParsedBody($data);  
 public function getAttributes();  
 public function getAttribute($name, $default = null);  
 public function withAttribute($name, $value);  
 public function withoutAttribute($name);  
}

1. interface ResponseInterface extends MessageInterface  
   {  
    public function getStatusCode();  
    public function withStatus($code, $reasonPhrase = '');  
    public function getReasonPhrase();  
   }
2. **PSR – 11** [**Container Interface**](https://www.php-fig.org/psr/psr-11)

Dokumen ini menjelaskan interfaces umum untuk wadah injeksi ketergantungan.

Tujuan yang ditetapkan oleh Container Interface adalah untuk membakukan bagaimana framework dan pustaka memanfaatkan a

kontainer untuk mendapatkan objek dan parameter (disebut entri di sisa dokumen ini).

Kata pelaksana dalam dokumen ini harus ditafsirkan sebagai seseorang

mengimplementasikan ContainerInterface dalam libraries atau framework ketergantungan yang terkait dengan ketergantungan.

Pengguna kontainer injeksi ketergantungan (DIC) disebut sebagai pengguna.

1. Spesifikasi

1.1 Dasar-dasar

1.1.1 Pengidentifikasi masuk

Pengenal entri adalah string PHP-legal sekurang-kurangnya satu karakter yang secara unik mengidentifikasi item dalam penampung. Pengidentifikasi entri adalah string buram, jadi penelepon TIDAK HARUS berasumsi bahwa struktur string membawa makna semantik apa pun.

1.1.2 Membaca dari sebuah wadah

The Psr \ Container \ ContainerInterface mengekspos dua metode: dapatkan dan miliki.

dapatkan satu parameter wajib: pengenal entri, yang HARUS berupa string.

dapatkan dapat mengembalikan apa saja (nilai campuran), atau melempar NotFoundExceptionInterface jika identifier

tidak diketahui wadahnya. Dua panggilan berturut-turut untuk mendapatkan hal yang sama

identifier HARUS mengembalikan nilai yang sama. Namun, tergantung pada implementor

desain dan / atau konfigurasi pengguna, nilai yang berbeda mungkin dikembalikan, jadi

pengguna TIDAK HARUS mengandalkan untuk mendapatkan nilai yang sama pada 2 panggilan berturut-turut.

telah mengambil satu parameter unik: pengenal entri, yang HARUS berupa string.

HARUS mengembalikan true jika pengenal entri diketahui oleh penampung dan salah jika tidak.

Jika telah ($ id) mengembalikan false, dapatkan ($ id) HARUS melemparkan NotFoundExceptionInterface.

1.2 Pengecualian

Pengecualian langsung dilemparkan oleh kontainer HARUS menerapkan

Psr \ Container \ ContainerExceptionInterface.

Panggilan ke metode get dengan id yang tidak ada HARUS membuang a

Psr \ Container \ NotFoundExceptionInterface.

1.3 Penggunaan yang disarankan

Pengguna TIDAK HARUS melewatkan kontainer ke objek sehingga objek dapat mengambil dependensi sendiri.

Ini berarti wadah digunakan sebagai Penunjuk Lokasi

yang merupakan pola yang umumnya berkecil hati.

Silakan merujuk ke bagian 4 dokumen META untuk lebih jelasnya.

2. Paket

interfaces dan class yang dijelaskan serta pengecualian yang relevan disediakan sebagai bagian dari

paket psr / peti kemas.

Paket yang menyediakan implementasi kontainer PSR harus menyatakan bahwa mereka menyediakan psr / container-implementation 1.0.0.

Proyek yang membutuhkan implementasi harus membutuhkan psr / container-implementation 1.0.0.

3. interfaces

<?php

namespace Psr\Container;

/\*\*

\* Describes the interface of a container that exposes methods to read its entries.

\*/

interface ContainerInterface

{

/\*\*

\* Finds an entry of the container by its identifier and returns it.

\*

\* @param string $id Identifier of the entry to look for.

\*

\* @throws NotFoundExceptionInterface No entry was found for \*\*this\*\* identifier.

\* @throws ContainerExceptionInterface Error while retrieving the entry.

\*

\* @return mixed Entry.

\*/

public function get($id);

/\*\*

\* Returns true if the container can return an entry for the given identifier.

\* Returns false otherwise.

\*

\* `has($id)` returning true does not mean that `get($id)` will not throw an exception.

\* It does however mean that `get($id)` will not throw a `NotFoundExceptionInterface`.

\*

\* @param string $id Identifier of the entry to look for.

\*

\* @return bool

\*/

public function has($id);

}

1. **PSR – 13** [**Hypermedia Links**](https://www.php-fig.org/psr/psr-13)

Spesifikasi

1.1 Tautan dasar

Sebuah Hypermedia Link terdiri dari, minimal:

URI yang mewakili sumber sasaran yang dirujuk.

Hubungan yang mendefinisikan bagaimana sumber daya target berhubungan dengan sumber.

Berbagai atribut Link lainnya mungkin ada, tergantung pada format yang digunakan. Sebagai atribut tambahan

tidak terstandardisasi atau universal, spesifikasi ini tidak berusaha untuk membakukannya.

Untuk keperluan spesifikasi ini, definisi berikut berlaku.

Menerapkan Objek - Sebuah objek yang mengimplementasikan salah satu interfaces yang ditentukan oleh ini

spesifikasi.

Serializer - libraries atau sistem lain yang membutuhkan satu atau lebih objek Link dan menghasilkan

representasi serial itu dalam beberapa format yang ditentukan.

1.2 Atribut

Semua tautan MUNGKIN mencakup nol atau lebih atribut tambahan di luar URI dan hubungan.

Tidak ada registrasi resmi dari nilai-nilai yang diizinkan di sini, dan validitas nilai

tergantung pada konteks dan sering pada format serialisasi tertentu. Umumnya didukung

nilai termasuk ‘hreflang’, ‘title’, dan ‘type’.

Serializers MUNGKIN menghilangkan atribut pada objek tautan jika diperlukan untuk melakukannya oleh serialisasi

format. Namun, serializers HARUS menyandikan semua atribut yang disediakan untuk memungkinkan

memungkinkan ekstensi pengguna kecuali dicegah oleh definisi format serialisasi.

Beberapa atribut (biasanya hreflang) dapat muncul lebih dari sekali dalam konteksnya. Karena itu,

nilai atribut MUNGKIN menjadi larik nilai daripada nilai sederhana. Serializers MUNGKIN

menyandikan array tersebut dalam format apa pun yang sesuai untuk format serial (seperti itu

sebagai daftar yang dipisahkan dengan ruang, daftar yang dipisahkan koma, dll.). Jika atribut yang diberikan tidak

diizinkan untuk memiliki beberapa nilai dalam konteks tertentu, serializers HARUS gunakan yang pertama

nilai yang disediakan dan mengabaikan semua nilai selanjutnya.

Jika suatu nilai atribut benar boolean, serializers MUNGKIN menggunakan bentuk singkatan jika sesuai

dan didukung oleh format serialisasi. Sebagai contoh, HTML mengijinkan atribut untuk

tidak memiliki nilai ketika keberadaan atribut memiliki makna boolean. Aturan ini berlaku

jika dan hanya jika atribut itu benar boolean, bukan untuk nilai "kebenaran" lainnya

dalam PHP seperti integer 1.

Jika nilai atribut palsu boolean, serializers HARUS menghilangkan atribut sepenuhnya

kecuali melakukan hal itu mengubah makna semantik dari hasil. Aturan ini berlaku jika

dan hanya jika atributnya boolean false, bukan untuk nilai “falsey” lainnya di PHP

seperti integer 0.

1.3 Hubungan

Hubungan tautan didefinisikan sebagai string, dan merupakan kata kunci sederhana di

kasus hubungan yang didefinisikan secara umum atau URI mutlak dalam kasus a

hubungan pribadi.

Dalam hal kata kunci sederhana digunakan, itu HARUS cocok satu dari registri IANA di:

http://www.iana.org/assignments/link-relations/link-relations.xhtml

Opsional, registri microformats.org MUNGKIN digunakan, tetapi ini mungkin tidak valid

dalam setiap konteks:

http://microformats.org/wiki/existing-rel-values

Hubungan yang tidak didefinisikan dalam salah satu pendaftar di atas atau yang serupa

pendaftaran publik dianggap "pribadi", yaitu khusus untuk suatu hal tertentu

aplikasi atau use case. Hubungan seperti itu HARUS menggunakan URI mutlak.

1.4 Template Tautan

RFC 6570 mendefinisikan format untuk template URI, yaitu,

pola untuk URI yang diharapkan diisi dengan nilai-nilai yang diberikan oleh klien

alat. Beberapa format hypermedia mendukung tautan templated sementara yang lain tidak, dan mungkin

memiliki cara khusus untuk menunjukkan bahwa tautan adalah templat. Serializer untuk format

yang tidak mendukung Template URI HARUS mengabaikan Link templated yang dihadapinya.

1.5 penyedia Evolvable

Dalam beberapa kasus, Penyedia Tautan mungkin memerlukan kemampuan untuk memiliki tautan tambahan

ditambahkan ke dalamnya. Di sisi lain, penyedia tautan harus hanya-baca, dengan tautan

diturunkan saat runtime dari beberapa sumber data lainnya. Untuk alasan itu, penyedia yang dapat dimodifikasi

adalah interfaces sekunder yang dapat diterapkan secara opsional.

Selain itu, beberapa objek Penyedia Tautan, seperti objek Respons PSR-7, adalah

dengan desain yang tidak berubah. Itu berarti metode untuk menambahkan tautan ke mereka di tempat itu

tidak kompatibel. Oleh karena itu, metode tunggal EvolvableLinkProviderInterface

mensyaratkan bahwa objek baru dikembalikan, identik dengan aslinya tetapi dengan

objek tautan tambahan disertakan.

1.6 objek tautan Evolvable

Tautan objek dalam banyak hal adalah objek nilai. Dengan demikian, memungkinkan mereka berevolusi

dengan cara yang sama dengan objek nilai PSR-7 adalah opsi yang berguna. Untuk alasan itu,

tambahan EvolvableLinkInterface disertakan yang menyediakan metode untuk

menghasilkan instance objek baru dengan satu perubahan. Model yang sama digunakan oleh PSR-7

dan, berkat perilaku copy-on-write PHP, masih CPU dan memori yang efisien.

Tidak ada metode yang dapat berevolusi untuk templated, namun, sebagai nilai templated dari a

tautan hanya didasarkan pada nilai href. Ini TIDAK HARUS diatur secara independen, tetapi

berasal dari apakah atau tidak nilai href adalah template tautan RFC 6570.

2. Paket

interfaces dan class yang dijelaskan disediakan sebagai bagian dari

paket psr / link.

3. Interface

3.1 Psr\Link\LinkInterface

<?php

namespace Psr\Link;

/\*\*

\* A readable link object.

\*/

interface LinkInterface

{

/\*\*

\* Returns the target of the link.

\*

\* The target link must be one of:

\* - An absolute URI, as defined by RFC 5988.

\* - A relative URI, as defined by RFC 5988. The base of the relative link

\* is assumed to be known based on context by the client.

\* - A URI template as defined by RFC 6570.

\*

\* If a URI template is returned, isTemplated() MUST return True.

\*

\* @return string

\*/

public function getHref();

/\*\*

\* Returns whether or not this is a templated link.

\*

\* @return bool

\* True if this link object is templated, False otherwise.

\*/

public function isTemplated();

/\*\*

\* Returns the relationship type(s) of the link.

\*

\* This method returns 0 or more relationship types for a link, expressed

\* as an array of strings.

\*

\* @return string[]

\*/

public function getRels();

/\*\*

\* Returns a list of attributes that describe the target URI.

\*

\* @return array

\* A key-value list of attributes, where the key is a string and the value

\* is either a PHP primitive or an array of PHP strings. If no values are

\* found an empty array MUST be returned.

\*/

public function getAttributes();

}

1. **PSR – 15** [**HTTP Handlers**](https://www.php-fig.org/psr/psr-15)

1. Spesifikasi

1.1 Penangan Permintaan

Penangan permintaan adalah komponen individual yang memproses permintaan dan

menghasilkan respons, sebagaimana didefinisikan oleh PSR-7.

Penangan permintaan MUNGKIN melempar pengecualian jika kondisi permintaan mencegahnya

menghasilkan tanggapan. Jenis pengecualian tidak didefinisikan.

Permintaan penangan menggunakan standar ini HARUS menerapkan interfaces berikut:

Psr \ Http \ Server \ RequestHandlerInterface

1.2 Middleware

Komponen middleware adalah komponen individu yang berpartisipasi, sering bersama

dengan komponen middleware lainnya, dalam memproses permintaan yang masuk dan

penciptaan respon yang dihasilkan, sebagaimana didefinisikan oleh PSR-7.

Komponen middleware MUNGKIN menciptakan dan mengembalikan respons tanpa didelegasikan

penangan permintaan, jika kondisi yang mencukupi terpenuhi.

Middleware menggunakan standar ini HARUS menerapkan interfaces berikut:

Psr \ Http \ Server \ MiddlewareInterface

1.3 Menghasilkan Respons

Ini DIREKOMENDASIKAN bahwa setiap middleware atau permintaan handler yang menghasilkan respons

baik akan menyusun prototipe PSR-7 ResponseInterface atau pabrik yang mampu

membuat instance ResponseInterface untuk mencegah ketergantungan pada a

implementasi pesan HTTP tertentu.

1.4 Menangani Pengecualian

Direkomendasikan bahwa setiap aplikasi yang menggunakan middleware termasuk komponen

yang menangkap pengecualian dan mengubahnya menjadi respons. Middleware ini HARUS

menjadi komponen pertama yang dijalankan dan membungkus semua proses lebih lanjut untuk memastikan itu

respons selalu dihasilkan.

2. interfaces

2.1 Psr \ Http \ Server \ RequestHandlerInterface

interfaces berikut HARUS dilaksanakan oleh penangan permintaan.

namespace Psr\Http\Server;

use Psr\Http\Message\ResponseInterface;

use Psr\Http\Message\ServerRequestInterface;

/\*\*

\* Handles a server request and produces a response

\*

\* An HTTP request handler process an HTTP request in order to produce an

\* HTTP response.

\*/

interface RequestHandlerInterface

{

/\*\*

\* Handles a request and produces a response

\*

\* May call other collaborating code to generate the response.

\*/

public function handle(ServerRequestInterface $request): ResponseInterface;

}

1. **PSR – 16** [**Simple Cache**](https://www.php-fig.org/psr/psr-16)

1.1 Pendahuluan

Caching adalah cara umum untuk meningkatkan kinerja dari setiap proyek, pembuatan

caching libraries salah satu fitur paling umum dari banyak framework dan

libraries. Interoperabilitas pada level ini berarti libraries dapat menjatuhkannya

implementasi caching sendiri dan mudah bergantung pada yang diberikan kepada mereka oleh

framework, atau libraries cache khusus lainnya.

PSR-6 menyelesaikan masalah ini, tetapi dengan cara yang agak formal dan verbose

apa yang paling dibutuhkan kasus penggunaan sederhana. Pendekatan yang lebih sederhana ini bertujuan untuk membangun sebuah

interfaces streamline standar untuk kasus umum. Ini tidak bergantung pada

PSR-6 tetapi telah dirancang untuk membuat kompatibilitas dengan PSR-6 sebagai lugas

mungkin.

1,2 Definisi

Definisi untuk Memanggil libraries, Menerapkan libraries, TTL, Kedaluwarsa dan Kunci

disalin dari PSR-6 karena asumsi yang sama benar.

Memanggil libraries - libraries atau kode yang benar-benar membutuhkan cache

jasa. libraries ini akan memanfaatkan layanan cache yang mengimplementasikan ini

interfaces standar, tetapi tidak memiliki pengetahuan tentang itu

implementasi layanan cache tersebut.

Implementing Library - libraries ini bertanggung jawab untuk menerapkan

standar ini untuk menyediakan layanan cache ke libraries Panggilan. Itu

Menerapkan libraries HARUS menyediakan class yang mengimplementasikan interfaces Psr \ SimpleCache \ CacheInterface.

Menerapkan libraries HARUS mendukung pada fungsi TTL minimum seperti yang dijelaskan

di bawah ini dengan granularitas keseluruhan.

TTL - Waktu Untuk Langsung (TTL) dari suatu item adalah jumlah waktu antara

ketika barang itu disimpan dan dianggap basi. TTL biasanya didefinisikan

oleh integer yang mewakili waktu dalam detik, atau objek DateInterval.

Kedaluwarsa - Waktu aktual saat item disetel menjadi basi. Ini adalah

dihitung dengan menambahkan TTL ke waktu ketika suatu objek disimpan.

Item dengan TTL 300 detik yang disimpan pada pukul 1:30:00 akan memiliki waktu berakhir 1:35:00.

Menerapkan libraries MUNGKIN kadaluwarsa item sebelum Waktu Kedaluwarsa yang diminta,

tetapi HARUS memperlakukan item sebagai kedaluwarsa setelah Waktu Kedaluwarsa tercapai. Jika suatu panggilan

libraries meminta item untuk disimpan tetapi tidak menentukan waktu kedaluwarsa, atau

menentukan waktu kedaluwarsa null atau TTL, libraries Implementasi MUNGKIN menggunakan dikonfigurasi

durasi default. Jika tidak ada durasi default yang ditetapkan, libraries Implementing

HARUS mengartikan itu sebagai permintaan untuk menyimpan item itu selamanya, atau selama

mendukung implementasi yang mendasarinya.

Jika TTL negatif atau nol disediakan, item HARUS dihapus dari cache

jika ada, karena sudah kadaluwarsa.

Kunci - String setidaknya satu karakter yang secara unik mengidentifikasi a

item cache. Menerapkan libraries HARUS mendukung kunci yang terdiri dari

karakter A-Z, a-z, 0-9, \_, dan. dalam urutan apa pun dalam enkode UTF-8 dan a

panjang hingga 64 karakter. Menerapkan libraries MUNGKIN mendukung tambahan

karakter dan penyandian atau panjang yang lebih panjang, tetapi harus mendukung setidaknya itu

minimum. libraries bertanggung jawab atas pelarian diri mereka sendiri dari string kunci

sebagaimana mestinya, tetapi HARUS dapat mengembalikan string kunci asli yang tidak dimodifikasi.

Karakter berikut disediakan untuk ekstensi masa depan dan TIDAK HARUS

didukung oleh pelaksana pustaka: {} () / \ @:

Cache - Sebuah objek yang mengimplementasikan interfaces Psr \ SimpleCache \ CacheInterface.

Kesalahan Cache - Cache miss akan mengembalikan null dan karena itu mendeteksi

jika satu null yang disimpan tidak mungkin. Ini adalah penyimpangan utama dari PSR-6

asumsi.

1.3 Cache

Implementasi MUNGKIN menyediakan mekanisme bagi pengguna untuk menentukan TTL default

jika tidak ditentukan untuk item cache tertentu. Jika tidak ada standar yang ditetapkan pengguna

disediakan implementasi HARUS default ke nilai hukum maksimum yang diizinkan oleh

implementasi yang mendasarinya. Jika implementasi yang mendasarinya tidak

mendukung TTL, TTL yang ditentukan pengguna akan diabaikan secara diam-diam.

1.4 Data

Melaksanakan libraries HARUS mendukung semua tipe data PHP yang dapat diprogram, termasuk:

String - String karakter ukuran acak dalam pengkodean yang kompatibel dengan PHP.

Bilangan bulat - Semua bilangan bulat dari berbagai ukuran yang didukung oleh PHP, hingga 64-bit ditandatangani.

Floats - Semua nilai floating point yang ditandatangani.

Boolean - Benar dan Salah.

Null - Nilai nol (meskipun tidak dapat dibedakan dari a

cache miss ketika membacanya kembali).

Array - Array yang diindeks, asosiatif dan multidimensi dari kedalaman arbitrary.

Objek - Setiap objek yang mendukung serialisasi dan

deserialization sedemikian rupa sehingga $ o == unserialize (serialize ($ o)). Objek MUNGKIN

gunakan interfaces PHP Serializable, metode ajaib \_\_sleep () atau \_\_wakeup (),

atau fungsi bahasa yang serupa jika sesuai.

Semua data yang dikirimkan ke libraries Pelaksana HARUS dikembalikan persis seperti

lulus. Itu termasuk jenis variabel. Artinya, itu adalah kesalahan untuk kembali

(string) 5 jika (int) 5 adalah nilai yang disimpan. Menerapkan libraries MUNGKIN menggunakan PHP

serialize () / unserialize () berfungsi secara internal tetapi tidak diperlukan untuk melakukannya.

Kompatibilitas dengan mereka hanya digunakan sebagai dasar untuk nilai-nilai objek yang dapat diterima.

Jika tidak mungkin mengembalikan sav yang tepat

<?php

namespace Psr\SimpleCache;

interface CacheInterface

{

/\*\*

\* Fetches a value from the cache.

\*

\* @param string $key The unique key of this item in the cache.

\* @param mixed $default Default value to return if the key does not exist.

\*

\* @return mixed The value of the item from the cache, or $default in case of cache miss.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

public function get($key, $default = null);

/\*\*

\* Persists data in the cache, uniquely referenced by a key with an optional expiration TTL time.

\*

\* @param string $key The key of the item to store.

\* @param mixed $value The value of the item to store, must be serializable.

\* @param null|int|\DateInterval $ttl Optional. The TTL value of this item. If no value is sent and

\* the driver supports TTL then the library may set a default value

\* for it or let the driver take care of that.

\*

\* @return bool True on success and false on failure.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

public function set($key, $value, $ttl = null);

/\*\*

\* Delete an item from the cache by its unique key.

\*

\* @param string $key The unique cache key of the item to delete.

\*

\* @return bool True if the item was successfully removed. False if there was an error.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

public function delete($key);

/\*\*

\* Wipes clean the entire cache's keys.

\*

\* @return bool True on success and false on failure.

\*/

public function clear();

/\*\*

\* Obtains multiple cache items by their unique keys.

\*

\* @param iterable $keys A list of keys that can obtained in a single operation.

\* @param mixed $default Default value to return for keys that do not exist.

\*

\* @return iterable A list of key => value pairs. Cache keys that do not exist or are stale will have $default as value.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if $keys is neither an array nor a Traversable,

\* or if any of the $keys are not a legal value.

\*/

public function getMultiple($keys, $default = null);

/\*\*

\* Persists a set of key => value pairs in the cache, with an optional TTL.

\*

\* @param iterable $values A list of key => value pairs for a multiple-set operation.

\* @param null|int|\DateInterval $ttl Optional. The TTL value of this item. If no value is sent and

\* the driver supports TTL then the library may set a default value

\* for it or let the driver take care of that.

\*

\* @return bool True on success and false on failure.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if $values is neither an array nor a Traversable,

\* or if any of the $values are not a legal value.

\*/

public function setMultiple($values, $ttl = null);

/\*\*

\* Deletes multiple cache items in a single operation.

\*

\* @param iterable $keys A list of string-based keys to be deleted.

\*

\* @return bool True if the items were successfully removed. False if there was an error.

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if $keys is neither an array nor a Traversable,

\* or if any of the $keys are not a legal value.

\*/

public function deleteMultiple($keys);

/\*\*

\* Determines whether an item is present in the cache.

\*

\* NOTE: It is recommended that has() is only to be used for cache warming type purposes

\* and not to be used within your live applications operations for get/set, as this method

\* is subject to a race condition where your has() will return true and immediately after,

\* another script can remove it making the state of your app out of date.

\*

\* @param string $key The cache item key.

\*

\* @return bool

\*

\* @throws \Psr\SimpleCache\InvalidArgumentException

\* MUST be thrown if the $key string is not a legal value.

\*/

public function has($key);

1. **PSR – 17** [**HTTP Factories**](https://www.php-fig.org/psr/psr-17)

1. Spesifikasi

Pabrik HTTP adalah metode yang digunakan untuk objek HTTP baru, seperti yang didefinisikan oleh PSR-7,

dibuat. Pabrik HTTP HARUS mengimplementasikan interfaces ini untuk setiap jenis objek

yang disediakan oleh paket.

2. interfaces

interfaces berikut MUNGKIN diimplementasikan bersama dalam satu class atau

di class terpisah.

2.1 RequestFactoryInterface

Has the ability to create client requests.

namespace Psr\Http\Message;

use Psr\Http\Message\RequestInterface;

use Psr\Http\Message\UriInterface;

interface RequestFactoryInterface

{

/\*\*

\* Create a new request.

\*

\* @param string $method The HTTP method associated with the request.

\* @param UriInterface|string $uri The URI associated with the request.

\*/

public function createRequest(string $method, $uri): RequestInterface;

}

**Kegiatan 2**

1. Integer

Integer merupakan tipe bilangan bulat atau bilangan yang tidak memiliki nilai di belakang koma, contoh bilangan integer adalah 100, 1283, 9999 dst. tipe data integer tidak dapat di masukan karakter huruf dan juga tidak bisa di masukan nilai bilangan pecahan jika bilangan pecahan di masukan maka akan terpotong hanya bilangan yang ada di depan koma yang dapat masuk, contoh anda memiliki bilanga 555,23 lalu anda memasukan kedalam Tipe Data integer maka yang akan masuk hanya 555, contoh penulisan variabel yang memiliki Tipe Data Integer :

<?php

$a = 12345;

$b = 5000;

?>

1. Doubles/Float/Real

Tipe Data ini merupakan Tipe Data yang memuat bilangan pecahan dan bilangan bulat Float memiliki sedikit perbedaan dengan integer, perbedaanya hanya terdapat di koma jika integer tidak dapat menerima masukan koma maka Doubles/Float/Real kebalikanya Tipe Data ini dapat menerima koma. Namun yang perlu pembaca perhatikan pada saat nanti penilisanya komputer tidak mengenal "," dan koma itu sendiri di gantikan dengan "." (titik), Contoh penulisan tipe data ini adalah :

<?php

$a = 123.45;

$phi = 3.14;

?>

1. Boolean

Tipe data yang selanjutnya adalah Tipe Data Boolean yang hanya memiliki 2 nilai yaitu TRUE/FALSE atau BENAR/SALAH atau juga HIDUP/MATI, Tipe Data ini biasanya digunakan untuk menetukan sebuah kondisi. Contoh penulisan serta penggunaan Tipe Data Boolean di PHP :

<?php

if(TRUE){

echo "Im Codepolitan Rangers";

} else {

echo "im designer Comic";

}

?>

1. Null

Tipe Data yang satu ini hanya memiliki satu nilai yaitu NULL atau kosong, berikut merupakan contoh penggunaan Tipe Data NULL di PHP:

<?php

$a = null;

?>

1. String

Tipe Data yang selanjutnya adalah tipe data yang memuat segala karakter huruf, angka, simbol, Tipe Data ini memiliki ciri pada penulisan yang di apit oleh kutip. Berikut merupakan contoh penggunaan Tipe Data String di PHP:

<?php

$a = "Aku seorang Coders";

$b = "jumlah dari 5 di tambah 5 adalah 10";

?>

1. Array

Array (atau larik dalam bahasa indonesia) bukanlah tipe data dasar seperti integer atau boolen, Array adalah sebuah tipe data bentukan yang terdiri dari kumpulan tipe data lainnya. Menggunakan array akan memudahkan dalam membuat kelompok data, serta menghemat penulisan dan penggunaan variabel.

<?php

//pembuatan array

$nama = array(

1=>"Andri",

2=>"Joko",

3=>"Sukma",

4=>"Rina",

5=>"Sari");

//cara akses array

echo $nama[1]; //Andri

echo "<br />";

echo $nama[2]; //Joko

echo "<br />";

echo $nama[3]; //Sukma

?>

1. Object

Tipe Data selanjutnya adalah Tipe Data yang menyimpan data dan informasi tentang cara mengolah data tersebut. Berikut adalah Contoh penggunaan Object di PHP :

<?php

class Codepolitan {

function Codepolitan() {

$this->wijay = "Saya seorang Coder";

}

}

// mendefinisikan penggunaan objek

$soffi = new Codepolitan();

// mengeluarkan nilai objek

echo $soffi->wijay;

?>

1. Resource

Tipe data yang jika di artikan ke dalam bahasa Indonesia ber-arti "Sumber Daya" ini adalah tipe data spesial yang mereferensikan keterkaitan luar melalui fungsi, berbeda dengan tipe data lainya tipe data ini berbentuk sebuah fungsi. berikut adalah daftar beberapa fungsi yang masuk kedalam tipe data Resource PHP :

mysqli\_connect()

digunakan untuk menghubungkan ke database SQL.

ftp\_connect()

digunakan untuk menghubungkan dan membuka protokol FTP.

imagerotate()

digunakan untuk memutar suatu gambar.

Kegiatan 3 : Operator dan Kegunaannya

## I. Operator Perbandingan Pada PHP

Seperti namanya, operator perbandingan digunakan untuk membandingkan dua buah nilai pada PHP dan hasilnya berupa boelan true yang berarti benar atau false yang berarti salah.

Berikut ini daftar operator perbandingan pada PHP:

| **Operator** | **Keterangan** |
| --- | --- |
| $a == $b | TRUE jika nilai $a dan $b sama tanpa membedakan tipe data nya |
| $a === $b | TRUE jika nilai dan tipe data $a dan $b sama |
| $a != $b | TRUE jika nilai $a dan $b tidak sama tanpa membedakan tipe data nya |
| $a <> $b | TRUE jika nilai $a dan $b tidak sama tanpa membedakan tipe data nya |
| $a !== $b | TRUE jika nilai dan tipe data $a dan $b tidak sama |
| $a < $b | TRUE jika nilai $a lebih kecil daripada $b |
| $a > $b | TRUE jika nilai $a lebih besar daripada $b |
| $a <= $b | TRUE jika nilai $a lebih kecil atau sama dengan $b |
| $a >= $b | TRUE jika nilai $a lebih besar atau sama dengan $b |
| $a <=> $b | TRUE jika nilai $a lebih besar, sama dengan, atau lebih besar dari $b . Mulai tersedia pada PHP 7 |

Contoh:

<?phpif ($\_POST['password'] == '1234'){

echo 'Login sukses';}

Pada contoh diatas, jika user mengirim password dengan nilai 1234 maka akan muncul pesan Login sukses

### Type Juggling

Penting diperhatikan bahwa ketika PHP membandingkan string dengan number, maka PHP otomatis akan mengubah string tersebut menjadi number. Proses ini dinamakantype juggling. Hal ini tidak terjadi jika operator yang digunakan === atau !==

Hal ini terkadang menyebabkan hasil perbandingan tidak sesuai dengan yang diharapkan.

Contoh:

$usia = '5 Tahun';$batas = 5;if ($batas == $usia) {

echo 'Usia sesuai;} else {

echo 'Usia tidak sesuai';}// Browser akan mencetak output Usia sesuai;

Pada contoh diatas, php akan mengubah variabel $usia dari tipe data string menjadi menjadi number (integer) sehingga hasil perbandingan menghasilkan nilai true

Nah, agar hasil perbandingan sesuai dengan nilai dan tipe data nya, maka kita gunakan operator === atau !==

Jika contoh diatas kita ubah menjadi $batas === $usia, maka hasil yang kita peroleh adalah Usia tidak sesuai

0 = false

Selain type juggling, terdapat hal penting yang perlu diperhatikan ketika melakukan evaluasi suatu nilai, yaitu PHP akan menganggap 0 bernilai false.

Hal ini akan aneh jika kita mengecek apakah sebuah string mengandung kata tertentu menggunakan fungsi strpos()

Contoh:

$css = 'http://ggantara.com/includes/css/style.css';$site\_url = 'http://ggantara.com';if (strpos($css, $site\_url)) {

echo 'Domain sama';} else {

echo 'Domain tidak sama';}

Contoh diatas akan menghasilkan output Domain tidak sama, karena strpos($css, $site\_url) menghasilkan 0 (posisi teks berada di karakter ke-0), nah karena 0 bernilai false, maka PHP akan mengeksekusi statement else

Untuk itu kita perlu operator pembanding !==  sebagai berikut:

$css = 'http://ggantara.com/includes/css/style.css';$site\_url = 'http://ggantara.com';if (strpos($css, $site\_url) !== false) {

echo 'Domain sama';} else {

echo 'Domain tidak sama';}

## II. Operator Aritmatika Pada PHP

Selanjutnya mari kita bahas tentang operator aritmatika pada PHP.

Masih ingat pelajaran matematika SD? Operator ini persis seperti operator bilangan pada mata pelajaran tersebut.

Operator aritmatika terdiri dari:

| **Operator** | **Keterangan** |
| --- | --- |
| $a + $b | Penjumlahan |
| $a – $b | Pengurangan |
| $a \* $b | Perkalian |
| $a / $b | Pembagian |
| $a % $b | Modulus. Sisa bagi antara $a dan $b |
| $a \*\* $b | Exponentiation. Diperkenalkan mulai PHP 5.6 |

Operator modulus ( % ) biasanya digunakan untuk mengidentifikasi apakah suatu bilangan berupa bilangan ganjil atau genap, contoh:

<?php$bariske = 5;if ($bariske % 2) {

echo 'Ganjil';} else {

echo 'Genap';}// Hasil Ganjil

Pada praktiknya operator modulus digunakan untuk memberikan class tertentu pada baris yang ganjil atau genap, contoh:

<?phpfor ($i = 1; $i <= 10, $i++) {

if ($i % 2) {

echo '<div class="baris-ganjil"></div>';

} else {

echo '<div></div>';

}}

**Operator Prioritas**

Operator diatas memiliki prioritas tertentu dimana perkalian dan pembagian akan lebih diutamakan daripada penjumlahan dan pengurangan.

Misal

<?php$a = 2 + 10 / 2;echo $a; // Hasil 7

$b = 2 + 10 \* 2;echo $a; // Hasil 22

Jika kita ingin menjumlahkan angka 2 dengan 10 kemudian membaginya dengan 5, maka kita bisa membuat pengelompokan dengan memberikan tanda kurung.

<?php$a = (2 + 10) / 2;echo $a; // Hasil 6

Type Juggling

Seperti pada operator perbandingan, ketika menemukan string pada operasi aritmatika maka PHP akan melakukan type juggling, yaitu mengkonversi string menjadi number.

Contoh:

$usia = '7.5 Tahun';$max = 3 + $usia;echo $max; // 10.5

Hati-hati bahwa di bahasa pemrograman lain seperti Javascript, hal ini tidak berlaku, jika kita ingin menjumlahkan bilangan maka keduanya harus memiliki tipe data number.

## III. Operator Logika Pada PHP

Pembahasan selanjutnya adalah pembahasan tentang operator logika pada PHP.

Perbandingan operator logika digunakan untuk membandingkan dua buah bilai boelan (true atau false) dengan hasil nilai juga berupa boelan (true atau false)

Tabel berikut ini merupakan tabel daftar operator logika pada PHP

| **Operator** | **Keterangan** |
| --- | --- |
| $a AND $b | Bernilai TRUE jika $a dan $b bernilai TRUE |
| $a OR $b | Bernilai TRUE jika $a atau $b bernilai TRUE |
| $a && $b | Bernilai TRUE jika $a dan $b bernilai TRUE, $a memiliki prioritas |
| $a || $b | Bernilai TRUE jika $a atau $b bernilai TRUE, $a memiliki prioritas |
| $a XOR $b | Bernilai TRUE jika $a atau $b bernilai TRUE, tapi tidak keduanya |
| !$a | Not. Bernilai TRUE jika $a bernilai FALSE |

Contoh operator not:

<?phpif (!$\_POST['email']) {

echo 'Email harus diisi';}

**Note:**perlu diperhatikan bahwa PHP akan mengevaluasi operand mulai dari kiri ke kanan, hal ini akan berguna ketika kita mengevaluasi nilai dimana operator yang digunakan semuanya && atau AND

### Penggunaan &&, ||, AND, dan OR

Kita bebas menggunakan && atau AND, || atau OR, keduanya memiliki performa yang sama dan bersifat short-circuit artinya jika salah satu ekspresi telah memenuhi kondisi yang disyaratkan, maka ekspresi lain tidak dievaluasi, misal:

<?php$level = 1;if ($level == 1 || $level == 2) {

echo 'Selamat datang admin';}

Pada contoh diatas, $level == 2 tidak dievaluasi karena ekspresi 1 ($level == 1) sudah terpenuhi

Yang umum adalah AND dan OR digunakan untuk control flow seperti ketika membuat koneksi pada database. contoh:

$conn = mysqli\_connect('localhost', 'root', '', 'database') or die('Gagal terhubung dengan database');

Sedangkan && dan || untuk kondisi true dan false, seperti contoh sebelumnya:

$level = 1;if ($level == 1 || $level == 2) {

echo 'Selamat datang admin';}

Pada contoh diatas browser akan mencetak teks Selamat datang admin karena logika if bernilai true.

Logika if diatas dapat diibaratkan if (true || false) { .. } Nah, seperti disampaikan sebelumnya, pada logika OR, jika salah salah satu bernilai true, maka hasilnya akan true

### Operator Logika Dengan AND Semua

Pada logika yang berisi AND semua maka jika salah satu operand bernilai false, maka PHP akan langsung menghasilkan nilai false tanpa memeriksa operand lainnya

Contoh:

<?php$level = 2;if (check\_exists() && check\_status() && $level == 1) {

echo 'Selamat datang admin';}

Pada contoh diatas PHP akan mejalankan fungsi check\_exists() jika bernilai true maka akan mengecek nilai check\_status(), jika bernilai true maka akan mengecek nilai $level

Nah karena menjalankan fungsi membutuhkan proses tambahan, maka sebaiknya memperhatikan urutan operand, intinya, yang paling sederhana di letakkan di paling depan.

Pada contoh diatas, lebih optimal jika kita mengubah urutan operand menjadi:

<?php$level = 2;if ($level == 1 && check\_exists() && check\_status()) {

echo 'Selamat datang admin';}

Karena variabel level bernilai 2, maka operand pertama akan bernilai false, sehingga PHP tidak akan mengecek operand yang lain.

### Hati-Hati Penulisan Operator Assignment Pada Perbandingan

Terkadang operator pembanding yang kita tulis tidak pas, sehingga berubah menjadi operator assignment, yang akan selalu menghasilkan nilai true

Contoh:

$level = 1;if ($level = 2) {

echo 'Selamat datang admin';}

Pada contoh diatas, browser akan selalu mencetak output Selamat datang admin karena statement if selalu menghasilkan nilai true ($level = 2 seharusnya $level == 2)

### Pengelompokan Ekspresi

Ketika menggabungkan operator && dan || dalam satu statement, maka sebaiknya dikelompokkan sesuai dengan kriteria yang diinginkan, jika tidak maka akan memberikan hasil yang tidak sesuai yang diharapkan

Contoh misal hanya user dengan $level = 1 dan memiliki $hak\_akses = 1 atau $hak\_akses = 2 yang diijinkan masuk, jika kita tulis:

<?php$level = 0;$hak\_akses = 2;if ($level == 1 && $hak\_akses == 1 || $hak\_akses == 2) {

echo 'Login berhasil';}

Maka browser akan mencetak Login berhasil. Pada statemen diatas, user dengan level apapun yang memiliki $hak\_akses = 2 bisa masuk. Untuk itu, kita perlu mengelompokkannya menjadi:

<?php$level = 0;$hak\_akses = 2;if ($level == 1 && ( $hak\_akses == 1 || $hak\_akses == 2 )) {

echo 'Login berhasil';}

## IV. Operator Pada PHP – Increment dan Decrement

Operator ini digunakan untuk menambahkan atau mengurangkan nilai sebanyak 1 pada suatu variabel. Adapun jenis operator ini adalah sebagai berikut:

| **Operator** | **Nama** | **Keterangan** |
| --- | --- | --- |
| ++$a | Pre Increment | Tambahkan 1 ke $a, kemudian tampilkan nilai $a |
| $a++ | Post Increment | Tampilkan nilai $a, kemudian tambahkan 1 ke $a |
| –$a | Pre Decrement | Kurangkan 1 dari $a, kemudian tampilkan nilai $a |
| $a– | Post Decrement | Tampilkan $a, kemudian kurangkan 1 dari $a |

### Perbedaan pre increment dan post increment

Pada pre increment, nilai variabel akan ditambahkan 1 baru kemudian siap digunakan, sebaliknya, untuk  post increment, gunakan dulu nilai variabel kemudian baru ditambahkan dengan 1

Contoh:

<?php$nomor = 1;while($nomor <= 5) {

echo $nomor++;}

Contoh diatas, akan menghasilkan angka 12345

Contoh 2:

<?php$nomor = 1;while($nomor <= 5) {

echo ++$nomor;}

Contoh diatas akan menghasilkan 23456

## V. Operator Assignment Pada PHP

Sesuai namanya “Assignment” operator ini digunakan untuk memberikan nilai pada suatu variabel. Operator dasarnya adalah tanda sama dengan ( = )

| **Operator** | **Alias** | **Contoh** |
| --- | --- | --- |
| $a += $b | $a = $a + $b | $a += 2 |
| $a -= $b | $a = $a – $b | $a -= 2 |
| $a \*= $b | $a = $a \* $b | $a \*= 2 |
| $a /= $b | $a = $a / $b | $a /= 2 |
| $a %= $b | $a = $a % $b | $a %= 2 |

Dalam praktiknya, operator ini sering digunakan ketika menjumlahkan nilai pada suatu perulangan, seperti ketika menjumlahkan data hasil query database.

Contoh:

<?php$sql = 'SELECT \* FROM sales';$query = mysqli\_query($sql);$total = 0;while($row = mysqli\_fetch\_array($query)){

$total += $row['jml\_bayar'];}