SKRIPSI

EVALUASI PHP STANDARDS RECOMMENDATIONS PADA PROYEK SHARIF JUDGE

Nicholas Khrisna Sandyawan

NPM: 6181801060

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN

«tahun»

DAFTAR ISI

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pengembangan aplikasi berbasis web dengan bahasa PHP cukup populer di kalangan pengembang web. Banyak *framework* yang tersedia untuk memudahkan pengembangannya, salah satunya adalah CodeIgniter. Walaupun sudah tersedia *tools* yang membantu, masih ditemukan beberapa masalah seperti penulisan kode yang tidak konsisten karena tidak ada standar yang baku penulisannya. Hal ini membuat pengembangan aplikasi web menjadi rumit dan sulit dipelihara, terutama jika proyek tersebut melibatkan banyak pengembang.

PHP Standards Recommendations [1] (PSR) adalah kumpulan standar penulisan PHP yang dibuat oleh PHP Framework Interop Group. Pada saat skripsi ini dibuat, terdapat 14 bab yang sudah diterima (Accepted) untuk digunakan, 4 bab masih didiskusikan (Draft), 3 bab ditinggalkan (Abandoned), dan 2 bab sudah usang (Deprecated). Bab-bab standar yang sudah diterima dan digunakan antara lain:

- PSR-01: Basic Coding Standard
- PSR-03: Logger Interface
- PSR-04: Autoloading Standard
- PSR-06: Caching Interface
- PSR-07: HTTP Message Interface
- PSR-11: Container Interface
- PSR-12: Extended Coding Style Guide
- PSR-13: Hypermedia Links
- PSR-14: Event Dispatcher
- PSR-15: HTTP Handlers
- PSR-16: Simple Cache
- PSR-17: HTTP Factories
- PSR-18: HTTP Client
- PSR-20: Clock

SharIF Judge merupakan perangkat lunak berbasis web yang dapat digunakan untuk menilai kode program dalam bahasa C, C++, Java, dan Python. SharIF Judge [2] yang dibahas pada dokumen ini adalah *fork* dari Sharif Judge [3] yang dibuat oleh Mohammad Javad Naderi. Versi *fork* ini sudah dikembangkan sesuai kebutuhan jurusan Teknik Informatika UNPAR dalam proses penilaian di beberapa mata kuliah.

Pada skripsi ini, keseluruhan PHP pada SharIF Judge akan dilihat dan dievaluasi seberapa jauh

standar PSR yang sudah dipenuhi. Selanjutnya akan dibuat rekomendasi berdasarkan hasil evaluasi. Walaupun demikian, masih akan ditentukan strategi untuk melakukannya, misalnya menggunakan tools atau alat tertentu untuk membantu, bab-bab apa saja yang relevan untuk dievaluasi sesuai yang digunakan pada SharIF Judge, dan seberapa banyak bab yang harus diperiksa secara manual. Salah satu alat yang digunakan adalah PHP linter. PHP linter membantu proses pemeriksaan sesuai salah satu standar, yaitu Extended Coding Style Guide atau aturan penulisan PHP yang sudah diperbarui.

Perangkat lunak akhir yang akan dibuat memiliki fitur yang sama persis dengan SharIf Judge yang sudah ada. Perbedaannya terdapat pada struktur PHP yang sudah dievaluasi sehingga memenuhi PSR. Berikut adalah fitur-fitur dari dokumentasi SharIF Judge:

- Dapat diakses untuk empat role : *admin*, *head instructor*, *instructor*, dan *student*
- Dapat mendeteksi plagiarisme pada kode
- Pengaturan khusus untuk keterlambatan pengumpulan
- Menunjukkan antrean pengumpulan
- Hasil penilaian dapat diekspor dalam dokumen Excel
- Dapat melakukan penilaian ulang
- Terdapat Scoreboard dan Notifications
- Tersedia log untuk 24 jam

1.2 Rumusan Masalah

Rumusan masalah yang akan dibahas pada skripsi ini sebagai berikut:

- Seberapa jauh PSR yang sudah terpenuhi ada SharIF Judge?
- Bagaimana mengevaluasi kode PHP pada SharIF Judge sesuai PSR?
- Bagaimana memberikan rekomendasi perbaikan pada kode PHP SharIF Judge agar meningkatkan jumlah PSR yang terpenuhi?

1.3 Tujuan

Tujuan yang ingin dicapai dalam penulisan skripsi ini sebagai berikut:

- Mengetahui seberapa jauh PSR yang sudah terpenuhi pada SharIF Judge.
- Mengevaluasi kode PHP pada SharIF Judge sesuai PSR.
- Memberikan rekomendasi perbaikan pada kode PHP SharIF Judge agar meningkatkan jumlah PSR yang terpenuhi.

1.4 Batasan Masalah

1.5 Metodologi

Metode penelitian yang akan digunakan dalam skripsi ini adalah:

1. Memperlajari SharIF Judge saat ini
2. Melakukan studi literatur mengenai PSR dan PHP linter
3. Mengevaluasi PHP dari SharIF Judge sesuai dengan PSR

4. Menguji SharIF Judge yang sudah dievaluasi
5. Memberikan rekomendasi sesuai hasil evaluasi
6. Menulis dokumen skripsi

1.6 Sistematika Pembahasan

Untuk penulisan skripsi ini akan dibagi dalam lima bagian sebagai berikut:

Bab 1 Pendahuluan
Bab 2 Landasan Teori
Bab 3 Analisis
Bab 4 Perancangan
Bab 5 Implementasi
Bab 6 Kesimpulan

BAB 2

LANDASAN TEORI

2.1 SharIF Judge

SharIF Judge [2] (dengan huruf kapital ”IF”) merupakan perangkat lunak berbasis web yang digunakan untuk menilai kode program dalam bahasa C, C++, Java, dan Python. SharIF Judge yang dibahas dalam dokumen ini adalah versi fork dari Sharif Judge [3] (dengan huruf kecil ”if”) yang dibuat oleh Mohammad Javad Naderi. Versi fork ini dikembangkan sesuai dengan kebutuhan jurusan Teknik Informatika UNPAR dalam proses penilaian di beberapa mata kuliah. SharIF judge dibuat dengan PHP pada framework CodeIgniter dan BASH untuk backend.

2.1.1 Fitur

Berikut adalah fitur-fitur dari SharIF Judge.

1. Terdapat beberapa role pengguna, antara lain admin, head instructor, instructor, dan student.
2. Sandboxing (belum tersedia untuk Python)
3. Deteksi kecurangan (mendeteksi kemiripan kode) menggunakan Moss
4. Pengaturan untuk keterlambatan pengumpulan
5. Antrian pengiriman
6. Mengunduh hasil dalam bentuk file excel
7. Mengunduh kode yang dikirim dalam bentuk file zip
8. Metode "Output Comporison" dan "Tester Code" untuk memeriksa kebenaran dari hasil output
9. Menambahkan beberapa pengguna sekaligus
10. Deskripsi masalah (PDF/Markdown/HTML)
11. Penilaian ulang (rejudge)
12. Papan Nilai (Scoreboard) dan Notifikasi (Notifications)

2.1.2 Instalasi

Untuk menjalankan SharIF Judge, dibutuhkan sebuah server Linux dengan syarat sebagai berikut:

- Webserver menjalankan PHP versi 5.3 atau lebih baru
- Pengguna dapat menjalankan PHP melalui command line.
- Menggunakan database MySql atau PostgreSql.
- PHP harus diberikan akses untuk untuk menjalankan perintah menggunakan fungsi shell exec (pakai underscore).
- Tools yang digunakan untuk melakukan kompilasi dan menjalankan kode yang dikumpulkan.

- Perl lebih baik diinstal untuk alasan ketepatan waktu, batas memori dan memaksimalkan batas ukuran pada output kode yang dikirim.

2.2 PHP Standards Recommendations

PHP Standards Recommendations (PSR) adalah kumpulan rekomendasi yang dibuat oleh PHP Framework Interop Group (PHP-FIG) untuk membantu para pengembang PHP dalam menciptakan kode yang lebih mudah dibaca, dipahami, dan dipelihara. Dalam PSR, terdapat kata kunci prioritas yang diatur dalam dokumen RFC 2119 dari PSR sendiri. Dikarenakan sumber referensi berbahasa Inggris, maka ada beberapa penyesuaian yang diperlukan untuk dokumen ini [1].

- "MUST", "REQUIRED", "SHALL" akan ditulis sebagai "HARUS" dengan arti harus sesuai.
- "MUST NOT", "SHALL NOT" akan ditulis sebagai "DILARANG" atau "TIDAK" dengan arti dilarang atau tidak diperbolehkan.
- "SHOULD", "RECOMMENDED" akan ditulis sebagai "SEBAIKNYA" atau "DIREKOMENDASIKAN" dengan arti direkomendasikan untuk digunakan namun tetap memperhatikan keadaan dan kebutuhan.
- "SHOULD NOT", "NOT RECOMMENDED" akan ditulis sebagai "SEBAIKNYA TIDAK" atau "TIDAK DIREKOMENDASIKAN" dengan arti tidak direkomendasikan untuk digunakan namun tetap memperhatikan keadaan dan kebutuhan.
- "MAY", "OPTIONAL" akan ditulis "BOLEH" atau "BISA" dengan arti bersifat tidak wajib.

Setiap PSR memiliki status, antara lain Accepted, Draft, Abandoned, dan Deprecated. Saat dokumen ini dibuat, terdapat 23 bab PSR dengan rincian sebagai berikut.

2.2.1 Accepted

Status "Accepted" adalah bab-bab yang sudah diterima, disepakati, dan diawasi oleh tim kerja bersangkutan untuk digunakan oleh para pengembang.

PSR-01: Basic Coding Standard

- *File* HARUS(1) menggunakan *tag* `<?php` dan `<?=` dan TIDAK(2) menggunakan variasi *tag* lainnya.
- *File* HARUS(3) menggunakan UTF-8 tanpa BOM untuk kode PHP.
- *File* SEBAIKNYA(4) mendeklarasikan simbol (kelas, fungsi, konstanta, dan lain-lain) atau menyebabkan efek samping (misalnya menghasilkan output, mengubah pengaturan .ini, dan lain-lain), tetapi SEBAIKNYA TIDAK(5) keduanya.
- *Namespace* dan kelas HARUS(6) mengikuti PSR "autoloading": [PSR-0, PSR-4].
- Nama kelas HARUS(7) dideklarasikan di `StudlyCaps`.
- Kode yang ditulis untuk PHP 5.3 dan setelahnya HARUS(8) menggunakan *namespace* formal.
- Kode yang ditulis untuk PHP 5.2.x dan sebelumnya SEBAIKNYA(9) menggunakan konvensi *pseudo-namepsacing* dengan awalan `Vendor_` pada nama kelas.

[1] Beberapa kata kunci memiliki tingkatan yang sama berdasarkan RFC 2119 sehingga dikategorikan dan ditulis dalam satu baris. Misalnya, "MUST", "REQUIRED", dan "SHALL" berada dalam satu tingkatan prioritas yang sama sehingga hanya perlu diartikan ke satu kata untuk menunjukkan kategori tingkatannya.

- Konstanta kelas HARUS(10) dideklarasikan dalam huruf kapital dengan pemisah garis bawah.
- Konvensi penamaan apa pun SEBAIKNYA(11) diterapkan secara konsisten dalam lingkup yang masuk akal, baik itu tingkat *vendor*, tingkat *package*, tingkat *class*, atau tingkat *method*.
- Nama *method* HARUS(12) dideklarasikan dalam `camelCase`.

PSR-03: Logger Interface

- `Logger Interface` memiliki delapan *method* untuk menulis log ke delapan level RFC 5424 (debug, info, notice, warning, error, critical, alert, emergency). Terdapat satu *method* lainnya, `log`, menerima level log sebagai argumen pertama. Memanggil *method* ini dengan salah satu konstanta level log HARUS(1) memiliki hasil yang sama dengan memanggil *method* pada level yang spesifik. Memanggil *method* ini dengan level yang tidak ada pada spesifikasi ini HARUS(2) melempar `Psr\Log\InvalidArgumentException` jika implementasinya tidak tahu tentang level tersebut. Pengguna SEBAIKNYA JANGAN(3) menggunakan level versi berbeda tanpa tahu dengan pasti jika implementasi yang sekarang mendukung.
- Setiap *method* menerima string sebagai pesan, atau objek dengan *method* `__toString()`. *Implementor* BISA(4) memiliki *handling* khusus untuk objek yang diteruskan. Jika bukan demikian, maka *implementor* harus mengubahnya ke sebuah string.
- Pesan BISA(5) memiliki *placeholder* yang BISA(6) diganti oleh *implementor* dari *context array*.
- Nama *placeholder* harus sesuai dengan kunci dalam *context array*.
- Nama placeholder HARUS(7) dipisahkan dengan kurung kurawal buka tunggal `{` dan kurung kurawal tutup tunggal `}`. TIDAK BOLEH(8) ada spasi kosong antara pembatas dan nama *placeholder*.
- Nama *placeholder* SEBAIKNYA(9) hanya terdiri dari karakter `A-Z`, `a-z`, `0-9`, garis bawah `_`, dan titik `.`.
- *Implementor* BISA(10) menggunakan *placeholder* untuk menerapkan berbagai strategi *escaping* dan menerjemahkan log untuk ditampilkan. Pengguna SEBAIKNYA TIDAK(11) melakukan *pre-escape* nilai dalam *placeholder* karena mereka tidak tahu dalam konteks mana data akan ditampilkan.
- Setiap *method* menerima array sebagai data konteks. Ini dimaksudkan untuk menyimpan informasi asing yang tidak sesuai dengan string. Array dapat berisi apa saja. *Implementor* HARUS(12) memastikan mereka memperlakukan data konteks dengan kelonggaran sebanyak mungkin. Nilai yang diberikan dalam konteks TIDAK BOLEH(13) melempar *exception* atau menimbulkan *error*, peringatan, atau pemberitahuan PHP apa pun.
- Jika objek `Exception` diteruskan dalam data konteks, maka HARUS(14) berada dalam *key* `'exception'`. *Implementor* HARUS(15) tetap memastikan bahwa*key* `'exception'` adalah sebuah objek `Exception`, karena isinya BISA(16) mengandung apa saja.

PSR-04: Autoloading Standard

- Istilah "kelas" mengacu pada kelas, *interface*, *traits*, dan struktur lain yang serupa.
- Nama kelas yang memenuhi syarat memiliki bentuk `\<NamespaceName>(\<SubNamespaceNames>)*\<Clas`
  dengan spesifikasi berikut:

1. HARUS(1) memiliki nama *namespace* tingkat tertinggi, atau dikenal sebagai "vendor namespace".
2. BISA(2) memiliki satu atau lebih nama *sub-space*.
3. HARUS(3) memiliki nama kelas di akhir.
4. Garis bawah tidak memiliki makna khusus.
5. Karakter abjad BISA(4) berisi kombinasi dari huruf kapital dan huruf kecil.
6. Semua nama kelas harus direferensikan dengan cara yang *case-sensitive*.

- Implementasi *autoloader* TIDAK BOLEH(5) melempar *exception*, TIDAK BOLEH(6) memunculkan *error* dalam tingkat apa pun, dan SEBAIKNYA TIDAK(7) mengembalikan nilai.

PSR-06: Caching Interface

- *Library* pelaksana HARUS(1) menyediakan kelas yang mengimplementasikan *interface* `Cache\CacheItemPool` dan `Cache\CacheItemInterface`. Impementasinya HARUS(2) mendukung fungsionalitas TTL minimum dengan perincian detik yang penuh. Time To Live (TTL) suatu item adalah jumlah waktu antara waktu saat item tersebut disimpan dan ketika item dianggap kedaluwarsa. TTL biasanya didefinisikan dalam bentuk bilangan bulat (integer) yang mewakili waktu dalam detik, atau sebuah objek DateInterval.
- *Libarary* pelaksana BISA(3) mengakhiri masa berlaku item sebelum *Expiration Time* yang diminta, tetapi HARUS(4) mengakhiri masa berlaku item tersebut jika *Expiration Time* sudah tercapai. Jika suatu *libary* meminta item untuk disimpan tanpa menentukan waktu kedaluwarsa, atau ditentukan isinya null atau TTL, *library* pelaksana BISA(5) menggunakan durasi default yang sudah dikonfigurasi. Jika belum ada durasi defaut yang dikonfigurasi, maka *library* pelaksana HARUS(6) menafsirkannya sebagai permintaan untuk menyimpan item dalam cache selamanya, atau selama implementasi yang mendasarinya mendukung.
- *Key* adalah suatu string yang terdiri dari minimal satu karakter yang secara unik mengidentifikasi item yang di-*cache*. *Library* pelaksana HARUS(7) mendukung *key* yang terdiri dari karakter `A-Z`, `a-z`, `0-9`, `_`, dan `.` dengan urutan apa pun dalam pengkodean UTF-8 dan maksimum 64 karakter. *Library* pelaksana BISA(8) mendukung karakter tambahan dan pengkodean atau karakter yang lebih banyak, namun harus memenuhi syarat minimum di atas.
- Dalam mengimplementasikan *library* semuanya HARUS(9) mendukung semua tipe data PHP bersambung, termasuk: String, Integer, Float, Boolean, Null, Array, dan Object.

PSR-07: HTTP Message Interface

- Pesan HTTP dapat berupa permintaan dari klien ke server atau respons dari server ke klien.
- MUST
- SHOULD
- Selama konstruksi, implementasi HARUS berusaha menyetel header `Host` dari URI yang disediakan jika tidak ada header `Host` yang tersedia.
- Klien HTTP HARUS mengabaikan nilai dari `Uri::getPath()` dan `Uri::getQuery()`, sebaliknya menggunakan nilai yang dikembalikan oleh `getRequestTarget()`, yang secara default

menggabungkan kedua nilai ini.

- Klien yang memilih untuk tidak mengimplementasi 1 atau lebih dari 4 formulir *request-target*, HARUS tetap menggunakan `RequestTarget()`. Klien-klien ini HARUS menolak *request-target* yang tidak mereka dukung, dan DILARANG kembali ke nilai dari `getUri()`.

PSR-11: Container Interface

- *Entry identifier* adalah string legal PHP apa pun yang setidaknya terdiri dari satu karakter unik yang mengidentifikasi sebuah item dalam suatu *container*. *Entry identifier* adalah sebuah string buram, maka pemanggil SEBAIKNYA TIDAK berasumsi bahwa struktur string memiliki makna semantik apa pun.
- Dalam `Psr\Container\ContainerInterface` terdapat 2 metode: `get` dan `has`.
-
-
- Pengecualian yang diberikan secara langsung oleh *container* SEBAIKNYA mengimplementasi `Psr\Container\ContainerExceptionInterface`.
- Panggilan ke metode `get` dengan id yang tidak ada HARUS memunculkan `Psr\Container\NotFoundExcep`
- Pengguna SEBAIKNYA TIDAK meneruskan suatu *container* ke objek sehingga objek dapat mengambil sendiri dependensinya. Hal ini berarti *container* digunakan sebagai *Service Locator* yang merupakan pola yang umumnya tidak dianjurkan.

PSR-12: Extended Coding Style Guide

-

PSR-13: Hypermedia Links

-

PSR-14: Event Dispatcher

- Sebuah *Listener* BISA(1) melakukan beberapa *behavior* asinkron jika diinginkan.
- Sebuah *Dispatcher* bertanggung jawab untuk memastikan bahwa *Event* diteruskan ke semua *Listener* yang relevan, tetapi HARUS(2) menunggu sesuai *listener* yang bertanggung jawab ke *Listener Provider*.
- Sebuah *Listener Provider* bertanggung jawab untuk menentukan *Listener* apa yang relevan sesuai *Event* tertentu, tetapi TIDAK BOLEH(3) memanggil *Listener* itu sendiri.
- Objek *Event* BISA berubah jika kasus penggunaannya memanggil *Listener* yang memberikan informasi ke *Emitter*.
- Jika tidak ada komunikasi dua arah yang diperlukan, maka DIREKOMENDASIKAN(4) agar *Event* ditetapkan sebagai *mutable*; yaitu didefinisikan sedemikian rupa sehingga tidak memiliki *method* mutator.
- Pengimplementasi HARUS(5) berasumsi bahwa objek yang sama akan diteruskan ke semua *Listener*.

- DIREKOMENDASIKAN(6) tetapi TIDAK DIHARUSKAN bahwa objek *Event* mendukung serialisasi dan deserialisasi *lossless*; `$event == unserialize(serialize($event))` SEBAIKNYA(7) bernilai *true*.
- Objek BISA(8) memanfaatkan *interface* PHP `Serializable`, `__sleep()` atau `__wakeup()` *magic method*, atau fungsionalitas bahasa yang serupa jika dibutuhkan.
- Sebuah *Event* yang mengimplementasikan `StoppableEventInterface` HARUS(9) mengembalikan `true` dari `isPropagationStopped()` ketika *Event* apa pun yang direpresentasikannya telah selesai.
- Sebuah *Listener* HARUS(10) memiliki satu dan hanya satu parameter, yaitu *Event* yang diresponsnya.
- *Listener* SEBAIKNYA(11) menuliskan petunjuk bahwa parameter secara spesifik relevan untuk kasus penggunaannya; yaitu *Listener* BISA(12) menuliskan petunjuk terhadap sebuah *interface* untuk menunjukkan bahwa *interface* tersebut kompatibel dengan semua jenis *Event* yang mengimplementasikannya, atau dengan implementasi khusus dari antarmuka tersebut.

PSR-15: HTTP Handlers

- *Request handler* BOLEH(1) memberi pengecualian jika kondisi permintaan mencegahnya untuk membuat respons. Jenis pengecualiannya tidak ditentukan.
- Setiap *request handler* yang menggunakan standar ini HARUS(2) mengimplementasikan *interface* `Psr\Http\Server\RequestHandlerInterface`.
- Komponen *middleware* BOLEH(3) membuat dan mengembalikan respons tanpa mendelegasikan ke *request handler*, jika kondisi yang dibutuhkan sudah terpenuhi.
- *Middleware* yang menggunakan standar ini HARUS(4) mengimplementasikan *interface* `Psr\Http\Server\Mid`
- Setiap *middleware* atau *request handler* yang menghasilkan respons DIREKOMENDASIKAN(5) untuk membuat prototipe PSR-07 `ResponseInterface` atau pabrik yang mampu menghasilkan *instance* `ResponseInterface` untuk mencegah ketergantungan pada implementasi pesan HTTP tertentu.
- Setiap aplikasi yang menggunakan *middleware* DIREKOMENDASIKAN(6) untuk menyertakan komponen yang menangkap pengecualian dan mengubahnya menjadi respons. *Middleware* ini HARUS(7) menjadi komponen pertama yang dieksekusi dan mencakup semua pemrosesan lebih lanjut untuk memasukan bahwa respons selalu dibuat.
- Sebuah *Listener* SEBAIKNYA(8) memiliki kembalian `void`, dan SEBAIKNYA(9) menuliskan petunjuk yang mengembalikan secara eksplisit. Sebuah *Dispatcher* HARUS(10) mengabaikan nilai kembalian dari *Listener*.
- Sebuah *Listener* BISA(11) mendelegasikan tindakan ke kode lain. Hal ini termasuk *Listener* yang menjadi pembungkus sebuah objek yang menjalankan *business logic* yang sebenarnya.
- Sebuah Listener BISA(12) menyusun informasi dari *Event* untuk diproses nanti oleh proses sekunder, menggunakan cron, sebuah server antrean, atau dengan teknik serupa. Hal ini BISA(13) membuat serial objek *Event* itu sendiri untuk melakukannya; namun harus berhati-hati agar tidak semua objek *Event* dapat diserialkan dengan aman. Sebuah proses sekunder HARUS(14) berasumsi bahwa setiap perubahan yang dibuatnya ke suatu objek *Event* tidak akan menyebar ke *Listener* lain.

PSR-16: Simple Cache

- *Library* pelaksana

PSR-17: HTTP Factories

- HTTP *factory* adalah metode yang digunakan untuk membuat objek HTTP baru sesuai yang didefinisikan oleh PSR-07. Setiap HTTP *factory* HARUS(1) mengimplementasi semua *interface* berikut untuk setiap tipe objek yang disediakan oleh *package*. *Interface* berikut ini BOLEH(2) diimplementasikan bersama dalam satu kelas atau kelas terpisah.
    1. RequestFactoryInterface
    2. ResponseFactoryInterface
    3. ServerRequestFactoryInterface
    4. StreamFactoryInterface
    5. UploadFileFactoryInterface
    6. UriFactoryInterface

PSR-18: HTTP Client

- Klien BOLEH(1) mengirim permintaan HTTP yang diubah dari yang disediakan, misalnya melakukan *compress* pada badan pesan yang dikirim.
- Klien BOLEH(2) memilih untuk mengubah respons HTTP yang diterima sebelum mengembalikannya ke *library* pemanggil, misalnya melakukan *decompress* isi pesan yang masuk.
- Jika klien memilih untuk mengubah permintaan HTTP atau respons HTTP, klien HARUS(3) memastikan bahwa objek tetap konsisten secara internal. Misalnya, jika klien memilih untuk dekompresi isi pesan, maka klien juga HARUS(4) menghapus header `Content-Encoding` dan menyesuaikan header `Content-Length`.
-
- Klien HARUS(5) menyusun kembali respons HTTP 1xx multi-langkah secara mandiri sehingga apa yang dikembalikan ke *library* pemanggil adalah respons HTTP yang valid dengan kode status 200 atau di atasnya.
- Klien TIDAK BOLEH(6) memperlakukan permintaan atau respons HTTP yang dibuat dengan baik sebagai kondisi *error*. Sebagai contoh, kode status respons dalam rentang 400 dan 500 TIDAK BOLEH(7) menimbulkan pengecualian dan HARUS(8) dikembalikan ke *library* pemanggil seperti biasa.
- Klien HARUS(9) memberikan *instance* `Psr\Http\Client\ClientExceptionInterface` jika dan hanya jika tidak dapat mengirim permintaan HTTP sama sekali atau jika respons HTTP tidak dapat diuraikan menjadi objek respons PSR-07.
- Jika permintaan tidak dapat dikirim karena pesan permintaan bukan permintaan HTTP yang dibuat dengan baik atau kehilangan beberapa informasi penting (seperti Host atau Method), klien HARUS(10) memberikan *instance* `Psr\Http\Client\RequestExceptionInterface`.
- Jika permintaan tidak dapat dikirim karena kegagalan jaringan dalam bentuk apa pun, termasuk *timeout*, klien HARUS(11) memberi instance `Psr\Http\Client\NetworkExceptionInterface`.
- Klien MUNGKIN(12) memberikan pengecualian yang lebih spesifik daripada yang diten-

tukan di sini (misalnya `TimeOutException` atau `HostNotFoundException`), asalkan mereka mengimplementasikan antarmuka yang sesuai yang ditentukan di atas.

PSR-20: Clock

- Kembalian waktu HARUS(1) ditulis sebagai `\DateTimeImmutable`

2.2.2 Draft

Status "Draft" adalah bab-bab yang masih dalam tahap diskusi dan pengembangan lebih lanjut agar isinya layak untuk menjadi standar.

PSR-05: PHPDoc Standard

-

PSR-19: PHPDoc tags

-

PSR-21: Internationalization

-

PSR-22: Application Tracing

-

2.2.3 Abandoned

Status "Abandoned" adalah bab-bab yang tidak lagi dikerjakan ataupun dikembangkan. oleh tim kerja yang bersangkutan.

PSR-08: Huggable Interface

-

PSR-09: Security Advisories

-

PSR-10: Security Reporting Process

-

2.2.4 Deprecated

Status ”Deprecated” adalah bab-bab yang sudah pernah disetujui sebelumnya, namun dianggap sudah tidak relevan karena perubahan-perubahan seiring berjalannya waktu. Bab-bab ini tidak direkomendasikan untuk digunakan. Salah satu alasan lain adalah karena adanya bab baru yang lebih baik untuk menggantikannya.

PSR-00: Autoloading Standard

-

PSR-02: Coding Style Guide

-

2.3 PHP Linter

Lint awalnya merujuk pada tool yang digunakan untuk menganalisis suatu kode program dengan tujuan menemukan kesalahan pada bahasa C. Kemudian istilah ini menjadi sebutan untuk mendeskripsikan hal-hal yang berkaitan dengan pengecekan kode program. PHP linter adalah tool yang digunakan untuk menganalisis kode PHP sesuai dengan standar tertentu. PHP linter yang digunakan adalah yang dibuat oleh Brueggern. Linter ini berdasar pada standar PSR ke-2 dan ke-12, yaitu Coding Style Guide yang sudah usang (deprecated) dan Extended Coding Style Guide sebagai penggantinya (accepted).

2.3.1 Syarat Instalasi

Sebelum menginstal linter, perlu dilakukan penginstalan Composer terlebih dahulu. Composer adalah alat untuk mengelola dependency pada PHP.

2.3.2 Instalasi

Berikut adalah langkah instalasi PHP linter.

1. Pada bagian root project, buka file composer.json.
2. Pada bagian "Repositories", tambahkan kode berikut.

Kode 2.1: kode kode

```
{
    "repositories": [
    {
        "type": "vcs",
        "url": "git@github.com:brueggern/php-linter.git"
    }
    ]
}
```

3. Install composer package. `composer require brueggern/php-linter`
4. Tambahkan script berikut untuk menjalankan linting/fixing. "app" dapat diganti dengan nama file atau folder yang akan di-lint.

Kode 2.2: kode kode

```
{
    "scripts": {
        "lint": "php-linter␣app",
        "lint:fix": "php-linter␣--fix␣app"
    }
}
```

2.3.3 Penggunaan

Untuk menjalankan linter perintah yang digunakan adalah: `composer run lint`

Untuk memperbaiki error secara otomatis, perintah yang digunakan adalah:
`composer run lint:fix`

DAFTAR REFERENSI

[1] (2019) *PHP Standards Recommendations*.

[2] Commit 02ce9a0 (2019) *SharIF-Judge*. Fakultas Teknologi Informasi dan Sains Universitas Katolik Parahyangan. Bandung, Indonesia.

[3] Commit 642120b (2015) *Sharif-Judge*.