Evaluasi PHP Standards Recommendations pada Proyek SharIF Judge

Nicholas Khrisna Sandyawan–6181801060

1 Data Tugas Akhir

Pembimbing utama/tunggal: **Pascal Alfadian Nugroho**
Pembimbing pendamping: **-**
Kode Topik : **PAN5401**
Topik ini sudah dikerjakan selama : **2** semester
Pengambilan pertama kali topik ini pada : Semester **54 - Genap 22/23**
Pengambilan pertama kali topik ini di kuliah : **Skripsi 1**
Tipe Laporan : **B** - Dokumen untuk reviewer pada presentasi dan **review Tugas Akhir 1**

2 Latar Belakang

Pengembangan aplikasi berbasis web dengan bahasa PHP cukup populer di kalangan pengembang web. Bahkan tersedia banyak *framework* yang dapat digunakan untuk memudahkan pengembangannya. Walaupun demikian, masih cukup sering dijumpai masalah dalam pengembangan. Salah satunya adalah penulisan kode program yang tidak konsisten karena belum ditentukan aturan atau standar penulisan tertentu. Hal ini membuat suatu proyek aplikasi web menjadi rumit dan sulit dipelihara, terutama jika melibatkan banyak pengembang.

Salah satu proyek aplikasi berbasis web yang digunakan di jurusan Informatika UNPAR adalah SharIF Judge. SharIF Judge merupakan aplikasi berbasis web yang dapat digunakan untuk menilai kode program dalam bahasa C, C++, Java, dan Python. Aplikasi ini ditulis dengan *framework* PHP CodeIgniter dan bagian *backend* dibuat dengan BASH. SharIF Judge yang dibahas pada tugas akhir ini adalah *fork* dari Sharif Judge yang dibuat oleh Mohammad Javad Naderi. Versi *fork* ini sudah dikembangkan sesuai kebutuhan Informatika UNPAR untuk digunakan dalam proses penilaian pada beberapa mata kuliah. Tidak menutup kemungkinan akan ada perbaikan atau penambahan fitur seiring berjalannya waktu. Hal ini dapat melibatkan lebih dari satu orang. Akan lebih baik jika ditentukan suatu aturan atau standar dalam pengembangannya.

PHP Standards Recommendation (PSR) adalah kumpulan standar penulisan PHP yang dibuat oleh PHP Framework Interop Group. Standar-standar ini berisi aturan dan rekomendasi penulisan kode program PHP yang dapat membantu pengembangan aplikasi agar lebih konsisten. Pada saat dokumen ini dibuat, terdapat 14 bab PSR yang berlaku untuk digunakan (*Accepted*) 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

Standar yang digunakan hanya standar dari bab-bab yang berstatus "*Accepted*". Bab dengan status lain yaitu "*Draft*", "*Abandoned*", dan "*Deprecated*" tidak akan digunakan sehingga tidak dicantumkan dalam dokumen tugas akhir.

Pada tugas akhir ini, kode-kode program PHP pada SharIF Judge akan diperiksa dan dievaluasi seberapa patuh SharIF Judge terhadap PSR berdasarkan jumlah standar yang sudah dipenuhi. Pemeriksaan PSR-12 dilakukan dengan bantuan *tools* PHP CS Fixer sementara pemeriksaan PSR lainnya dilakukan secara manual. Setelah itu, kode-kode yang belum memenuhi akan diberikan rekomendasi sesuai PSR berdasarkan hasil evaluasi tersebut.

3 Rumusan Masalah

- Bagaimana tingkat kepatuhan kode PHP pada SharIF Judge terhadap PSR?
- Rekomendasi perbaikan apa yang dapat diberikan pada kode PHP SharIF Judge untuk meningkatkan jumlah aturan PSR yang terpenuhi?

4 Tujuan

- Mengukur tingkat kepatuhan kode PHP pada SharIF Judge terhadap PSR.
- Memberikan perbaikan pada kode PHP SharIF Judge sesuai PSR agar meningkatkan jumlah standar yang terpenuhi.

5 Detail Perkembangan Pengerjaan Tugas Akhir

Detail bagian pekerjaan tugas akhir sesuai dengan rencana kerja/laporan perkembangan terkahir :

1. **Mempelajari SharIF Judge.**
   Status : Ada sejak rencana kerja tugas akhir.
   Hasil : Studi tentang SharIF Judge sejauh ini adalah membedah *file-file* PHP yang menjadi lingkup evaluasi. Semua *file* yang berasal dari *framework* atau *library* tidak termasuk ke dalamnya. Hanya *file-file* dalam folder `application/` dan memiliki tanda "SharIF Judge" di dalamnya yang menjadi lingkup evaluasi. Diambil dari dokumentasinya, berikut adalah langkah-langkah instalasi SharIF Judge:
   (a) Mengunduh versi terbaru dari SharIF Judge dan *unpack* hasil unduhan di direktori *public html*.
   (b) Memindahkan folder `system` dan `application` ke luar direktori *public*, dan masukkan *path* lengkap di *file* `index.php`.

```
$system_path = '/home/mohammad/secret/system';
$application_folder = '/home/mohammad/secret/application';
```

(c) Membuat sebuah *database* Mysql atau PostgreSql untuk SharIF Judge. Jangan menginstall paket koneksi *database* untuk C/C++, Java, atau Python.

(d) Mengatur koneksi *database* di *file* `application/config/database.php` dan simpan dengan nama `database.php`. Pengguna dapat menggunakan awalan untuk nama tabel.

```
/*  Enter database connection settings here:  */
'dbdriver' => 'postgre',    // database driver (mysqli, postgre)
'hostname' => 'localhost',  // database host
'username' => ‘,           // database username
'password' => ‘,           // database password
'database' => ‘,           // database name
'dbprefix' => 'shj_',       // table prefix
/*********************************************/
```

(e) Mengatur server `RADIUS` dan server *mail* pada *file application/config/secrets.example.php* dan simpan dengan nama `secrets.php`.

(f) Membuat direktori `application/cache/Twig` agar dapat ditulis oleh PHP.

(g) Membuka halaman utama SharIF Judge pada *web browser* dan mengikuti proses instalasi berikutnya.

(h) *Log in* menggunakan akun admin.

(i) Memindahkan folder `tester` dan `assigments` ke luar direktori *public* lalu simpan *path* lengkap di halaman Settings. Dua folder tersebut harus dapat ditulis oleh PHP. File-file yang diunggah akan disimpan di folder `assigments` sehingga tidak dapat diakses publik.

Eksperimen menjalankan SharIF Judge secara lokal sedang dalam tahap percobaan karena masih belum berhasil.

Selain itu, ditemukan folder `psr`. Di dalamnya terdapat folder `container` yang berisi *method-method* rekomendasi dari PSR-11: Container Interface, `log` yang berisi *method-method* rekomendasi dari PSR-03: Logger Interface, dan `simple-cache` yang berisi *method-method* rekomendasi dari PSR-16: Common Interface for Caching Libraries. Masih dipelajari lebih lanjut apakah *file-file* ini memang diimplementasikan pada SharIF Judge sehingga rekomendasi-rekomendasi dari ketiga PSR tersebut sebenarnya sudah terpenuhi atau belum.

2. **Melakukan studi literatur mengenai PSR dan PHP CS Fixer.**
Status : Ada sejak rencana kerja rencana tugas akhir namun ada perubahan dari PHP Linter menjadi PHP CS Fixer.
Hasil : Berikut adalah rincian hasilnya.

(a) Penulisan rekomendasi-rekomendasi standar PSR ini diterjemahkan dari bahasa Inggris ke bahasa Indonesia. Salah satu tantangan dalam menerjemahkan PSR ini adalah tidak setiap aturan ditulis dalam bentuk *list* yang mudah untuk dibaca. Banyak di antaranya merupakan bagian dari suatu paragraf yang berkelanjutan dengan konteksnya masing-masing sehingga penjelasan atau deskripsinya perlu dicantumkan pula dalam Landasan Teori agar tidak ada rekomendasi yang bermakna ambigu atau tidak jelas. Untuk saat ini, semua PSR dicantumkan dalam Landasan Teori kecuali dua bab yaitu PSR-07 dan PSR-12. Hal ini dikarenakan jumlah aturan yang harus diterjemahkan dan ditulis kembali cukup banyak. Untuk sementara, hal ini dicantumkan dalam Batasan Masalah. Rencananya kedua bab ini akan ditambahkan setelah selesai dilakukan pemeriksaan dan evaluasi kepatuhan.

(b) Ada beberapa kata kunci yang harus ditaati dalam standar PSR berdasarkan dokumen Request For Comments (RFC) 2119, yaitu "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", dan "MAY". Ada juga beberapa kata kunci lain yang disertakan namun sangat jarang digunakan, yaitu "REQUIRED", "SHALL", "SHALL NOT", "RECOMMENDED", dan "OPTIONAL". Setiap kata kunci ditulis dengan huruf kapital dan memiliki tingkat prioritas yang berbeda dalam hal keharusan pemenuhannya. Beberapa kata kunci memiliki tingkat yang sama, sebagai contoh "MUST", "SHALL", dan "REQUIRED" memiliki tingkat yang sama yaitu harus dipenuhi sesuai rekomendasi PSR. Maka dari itu, setiap kata kunci dikelompokkan berdasarkan tingkatan tersebut dan diterjemahkan ke bahasa Indonesia agar penulisan dokumen tugas akhir menjadi konsisten dan tidak mengubah makna dari standar yang asli. Misalnya, kata kunci "MUST" dan setingkatnya diterjemahkan menjadi "HARUS". Begitu pula dengan lawan katanya "MUST NOT" diterjemahkan menjadi "TIDAK BOLEH".

(c) Setiap standar diberikan kode tertentu agar dapat digunakan sebagai acuan dari bab-bab lain dalam dokumen tugas akhir. Kode tersebut ditulis dengan format PSR-XXYY, di mana XX adalah nomor bab dan YY adalah nomor urut standar dalam bab tersebut. Sebagai contoh PSR-0402 berarti mengacu pada standar PSR nomor 2 dari bab PSR-04. Urutan tersebut didasarkan pada standar mana yang ditulis terlebih dahulu. Kode-kode ini tidak ada pada dokumentasi asli PSR dan hanya dibuat untuk kepentingan tugas akhir ini. Berikut adalah bab-bab dalam PSR berstatus *Accepted* yang sudah ditulis:

- PSR-01: Basic Coding Standard
  - *File* (PSR-0101) HARUS menggunakan *tag* `<?php` dan `<?=` dan (PSR-0102)TIDAK menggunakan variasi *tag* lainnya.
  - *File* (PSR-0103) HARUS menggunakan UTF-8 tanpa Byte Order Mark (BOM) untuk kode PHP.
  - *File* (PSR-0104) SEBAIKNYA mendeklarasikan simbol (kelas, fungsi, konstanta, dan lain-lain) atau menyebabkan efek samping (misalnya menghasilkan output, mengubah pengaturan .ini, dan lain-lain), tetapi (PSR-0105) SEBAIKNYA TIDAK keduanya.
  - *Namespace* dan kelas (PSR-0106) HARUS mengikuti PSR "autoloading": [PSR-4].
  - Nama kelas (PSR-0107) HARUS dideklarasikan di `StudlyCaps`.
  - Kode yang ditulis untuk PHP 5.3 dan setelahnya (PSR-0108) HARUS menggunakan *namespace* formal.
  - Kode yang ditulis untuk PHP 5.2.x dan sebelumnya (PSR-0109) SEBAIKNYA menggunakan konvensi *pseudo-namepsacing* dengan awalan `Vendor_` pada nama kelas.
  - Konstanta kelas (PSR-0110) HARUS dideklarasikan dalam huruf kapital dengan pemisah garis bawah.
  - Konvensi penamaan apa pun (PSR-0111) SEBAIKNYA diterapkan secara konsisten dalam lingkup yang masuk akal, baik itu tingkat *vendor*, tingkat *package*, tingkat *class*, atau tingkat *method*.
  - Nama *method* (PSR-0112) HARUS 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 (PSR-0301) HARUS memiliki hasil yang sama dengan memanggil *method* pada level yang spesifik. Memanggil *method* ini dengan level yang tidak ada pada spesifikasi ini (PSR-0302) HARUS melempar `Psr\Log\InvalidArgumentException`

jika implementasinya tidak tahu tentang level tersebut. Pengguna (PSR-0303) SEBAIKNYA JANGAN 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* (PSR-0304) BISA memiliki *handling* khusus untuk objek yang diteruskan. Jika bukan demikian, maka *implementor* harus mengubahnya ke sebuah string.
- Pesan (PSR-0305) BISA memiliki *placeholder* yang (PSR-0306) BISA diganti oleh *implementor* dari *context array*.
- Nama *placeholder* harus sesuai dengan kunci dalam *context array*.
- Nama placeholder (PSR-0307) HARUS dipisahkan dengan kurung kurawal buka tunggal (`{`)dan kurung kurawal tutup tunggal (`}`). (PSR-0308) TIDAK BOLEH ada spasi kosong antara pembatas dan nama *placeholder*.
- Nama *placeholder* (PSR-0309) SEBAIKNYA hanya terdiri dari karakter `A-Z`, `a-z`, `0-9`, garis bawah (`_`), dan titik (`.`).
- *Implementor* (PSR-0310) BISA menggunakan *placeholder* untuk menerapkan berbagai strategi *escaping* dan menerjemahkan log untuk ditampilkan. Pengguna (PSR-0311) SEBAIKNYA TIDAK 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* (PSR-0312) HARUS memastikan mereka memperlakukan data konteks dengan kelonggaran sebanyak mungkin. Nilai yang diberikan dalam konteks (PSR-0313) TIDAK BOLEH melempar *exception* atau menimbulkan *error*, peringatan, atau pemberitahuan PHP apa pun.
- Jika objek `Exception` diteruskan dalam data konteks, maka (PSR-0314) HARUS berada dalam *key* `'exception'`. *Implementor* (PSR-0315) HARUS tetap memastikan bahwa *key* `'exception'` adalah sebuah objek `Exception`, karena isinya (PSR-0316) BISA 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>(\<SubNamespace Names>)*\<ClassName>` dengan spesifikasi berikut:
    i. (PSR-0401) HARUS memiliki nama *namespace* tingkat tertinggi, atau dikenal sebagai "vendor namespace".
    ii. (PSR-0402) BISA memiliki satu atau lebih nama *sub-space*.
    iii. (PSR-0403) HARUS memiliki nama kelas di akhir.
    iv. Garis bawah tidak memiliki makna khusus.
    v. Karakter abjad (PSR-0404) BISA berisi kombinasi dari huruf kapital dan huruf kecil.
    vi. Semua nama kelas harus direferensikan dengan cara yang *case-sensitive*.
  - Implementasi *autoloader* (PSR-0405) TIDAK BOLEH melempar *exception*, (PSR-0406) TIDAK BOLEH memunculkan *error* dalam tingkat apa pun, dan (PSR-0407) SEBAIKNYA TIDAK mengembalikan nilai.
- PSR-06: Caching Interface
  - *Library* pelaksana (PSR-0601) HARUS menyediakan kelas yang mengimplementasikan *interface* `Cache\CacheItemPoolInterface` dan `Cache\CacheItemInterface`. Impementasinya (PSR-0602) HARUS 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 (PSR-0603) BISA mengakhiri masa berlaku item sebelum *Expiration Time* yang diminta, tetapi (PSR-0604) HARUS 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 (PSR-0605) BISA menggunakan durasi default yang sudah dikonfigurasi. Jika belum ada durasi defaut yang dikonfigurasi, maka *library* pelaksana (PSR-0606) HARUS 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 (PSR-0607) HARUS 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 (PSR-0608) BISA mendukung karakter tambahan dan pengkodean atau karakter yang lebih banyak, namun harus memenuhi syarat minimum di atas.
- Dalam mengimplementasikan *library* semuanya (PSR-0609) HARUS mendukung semua tipe data PHP bersambung, termasuk:
  - String
  - Integer
  - Float
  - Boolean
  - Null
  - Array
  - Object - Setiap Object (PSR-0610) BISA memanfaatkan antarmuka Serializable PHP, metode `__sleep()` atau `__wakeup()`, atau fungsi serupa lain jika diperlukan.
- Semua data yang diteruskan ke *library* Pelaksana (PSR-0611) HARUS dikembalikan sama persis seperti data yang diteruskan tersebut.
- *Library* Pelaksana (PSR-0612) BISA menggunakan fungsi PHP `serialize()`/`unserialize()` secara internal tetapi tidak diwajibkan.
- Jika tidak dimungkinkan untuk mengembalikan *value* yang tersimpan sama persis untuk alasan apa pun, *library* pelaksana harus memberikan respons dengan kehilangan *cache*, bukan data yang rusak (*corrupted*).
- Pool merepresentasikan kumpulan *item* dalam sistem *caching*. Pool adalah repositori logis dari *item-item* di dalamnya. Semua *item* yang dapat di-*cache* diambil dari Pool sebagai objek Item, dan semua interaksi yang terjadi antar objek yang di-*cache* terjadi melalui Pool. Item merepresentasikan satu pasangan *key*/*value* dalam suatu Pool. *Key* adalah penanda unik untuk suatu *item* dan (PSR-0613) HARUS *immutable* (tidak dapat diubah). *Value* BISA diubah setiap waktu.
- Meskipun *caching* sering kali merupakan bagian penting dari kinerja aplikasi, *caching* tidak boleh memengaruhi fungsionalitas aplikasi. Oleh karena itu, kesalahan dalam sistem *cache* (PSR-0614) TIDAK BOLEH mengakibatkan kegagalan aplikasi. Untuk alasan tersebut, *Library* Pelaksana (PSR-0615) TIDAK BOLEH melempar *exception* selain yang ditentukan oleh antarmuka, dan (PSR-0616) HARUS menangkap *error* atau *exception* apa

pun yang dipicu oleh penyimpanan data yang mendasarinya (dan tidak membiarkannya menggelembung).
  - *Library* Pelaksana (PSR-0617) HARUS mencatat *error* tersebut atau melaporkannya ke administrator sebagaimana mestinya.
  - Jika *Library* Pemanggil meminta agar satu atau lebih Item dihapus, atau pool dibersihkan, maka (PSR-0618) TIDAK BOLEH dianggap sebagai kondisi *error* jika *key* yang ditentukan tidak ada. Kondisi pasca pun sama (*key* tidak ada, atau pool kosong), sehingga tidak ada kondisi error.
- PSR-07: HTTP Message Interface Standar PSR-07 tidak ditulis pada dokumen ini sebagaimana telah dicantumkan pada Batasan Masalah di Bab 1 Pendahuluan.
- 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 (PSR-1101) 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* (PSR-1102) SEBAIKNYA mengimplementasi `Psr\Container\ContainerExceptionInterface`.
  - Panggilan ke metode `get` dengan id yang tidak ada (PSR-1103) HARUS memunculkan `Psr\Container\NotFoundExceptionInterface`.
  - Pengguna (PSR-1104) 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 Standar PSR-12 tidak ditulis pada dokumen ini sebagaimana telah dicantumkan pada Batasan Masalah di Bab 1 Pendahuluan.
- PSR-13: Hypermedia Links *Link* Hypermedia paling sedikit terdiri dari:
  i. URI yang merepresentasikan *resource* target yang direferensikan.
  ii. Suatu hubungan yang mendefinisikan bagaimana *resource* target berhubungan dengan asal sumbernya.

  Berbagai atribut *Link* lainnya mungkin ada, tergantung pada format yang digunakan. Dikarenakan atribut tambahan tidak terstandarisasi dengan baik atau bersifat universal, spesifikasi ini tidak berupaya untuk membuatnya standar.
  Untuk keperluan spesifikasi ini, definisi berikut berlaku.
  i. Implementing Object - Objek yang mengimplementasikan salah satu antarmuka yang ditentukan oleh spesifikasi ini.
  ii. Serializer - Sebuah *library* atau sistem lain yang mengambil satu atau lebih objek Link dan membuat representasi serial dalam beberapa format yang ditentukan.
  - Semua tautan (PSR-1301) BOLEH menyertakan nol atau lebih atribut tambahan di luar URI dan hubungannya.
  - Serializer (PSR-1302) BOLEH menghilangkan atribut pada objek *link* jika diperlukan oleh format serialisasi. Namun, serializer (PSR-1303) HARUS menyandikan (*encode*) semua atribut yang disediakan untuk memungkinkan ekstensi pengguna kecuali dicegah oleh definisi format serialisasi.
- PSR-14: Event Dispatcher
  - Sebuah *Listener* (PSR-1401) BISA melakukan beberapa *behavior* asinkron jika diinginkan.

  - Sebuah *Dispatcher* bertanggung jawab untuk memastikan bahwa *Event* diteruskan ke semua *Listener* yang relevan, tetapi (PSR-1402) HARUS menunggu sesuai *listener* yang bertanggung jawab ke *Listener Provider*.
  - Sebuah *Listener Provider* bertanggung jawab untuk menentukan *Listener* apa yang relevan sesuai *Event* tertentu, tetapi (PSR-1403) TIDAK BOLEH 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 (PSR-1404) DIREKOMENDASIKAN agar *Event* ditetapkan sebagai *mutable*; yaitu didefinisikan sedemikian rupa sehingga tidak memiliki *method* mutator.
  - Pengimplementasi (PSR-1405) HARUS berasumsi bahwa objek yang sama akan diteruskan ke semua *Listener*.
  - (PSR-1406) DIREKOMENDASIKAN tetapi (PSR-1407) TIDAK DIHARUSKAN bahwa objek *Event* mendukung serialisasi dan deserialisasi *lossless*;
    `$event == unserialize(serialize($event))` (PSR-1408) SEBAIKNYA bernilai *true*.
  - Objek (PSR-1409) BISA memanfaatkan *interface* PHP `Serializable`, `__sleep()` atau `__wakeup()` *magic method*, atau fungsionalitas bahasa yang serupa jika dibutuhkan.
  - Sebuah *Event* yang mengimplementasikan `StoppableEventInterface` (PSR-1410) HARUS mengembalikan `true` dari `isPropagationStopped()` ketika *Event* apa pun yang direpresentasikannya telah selesai.
  - Sebuah *Listener* (PSR-1411) HARUS memiliki satu dan hanya satu parameter, yaitu *Event* yang diresponsnya.
  - *Listener* (PSR-1412) SEBAIKNYA menuliskan petunjuk bahwa parameter secara spesifik relevan untuk kasus penggunaannya; yaitu *Listener* (PSR-1413) BISA 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* (PSR-1501) BOLEH memberi pengecualian jika kondisi permintaan mencegahnya untuk membuat respons. Jenis pengecualiannya tidak ditentukan.
  - Setiap *request handler* yang menggunakan standar ini (PSR-1502) HARUS mengimplementasikan *interface* `Psr\Http\Server\RequestHandlerInterface`.
  - Komponen *middleware* (PSR-1503) BOLEH membuat dan mengembalikan respons tanpa mendelegasikan ke *request handler*, jika kondisi yang dibutuhkan sudah terpenuhi.
  - *Middleware* yang menggunakan standar ini (PSR-1504) HARUS mengimplementasikan *interface* `Psr\Http\Server\MiddlewareInterface`.
  - Setiap *middleware* atau *request handler* yang menghasilkan respons (PSR-1505) DIREKOMENDASIKAN 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* (PSR-1506) DIREKOMENDASIKAN untuk menyertakan komponen yang menangkap pengecualian dan mengubahnya menjadi respons. *Middleware* ini (PSR-1507) HARUS menjadi komponen pertama yang dieksekusi dan mencakup semua pemrosesan lebih lanjut untuk memasukan bahwa respons selalu dibuat.

  - Sebuah *Listener* (PSR-1508) SEBAIKNYA memiliki kembalian `void`, dan (PSR-1509) SEBAIKNYA menuliskan petunjuk yang mengembalikan secara eksplisit. Sebuah *Dispatcher* (PSR-1510) HARUS mengabaikan nilai kembalian dari *Listener*.
  - Sebuah *Listener* (PSR-1511) BISA mendelegasikan tindakan ke kode lain. Hal ini termasuk *Listener* yang menjadi pembungkus sebuah objek yang menjalankan *business logic* yang sebenarnya.
  - Sebuah Listener (PSR-1512) BISA menyusun informasi dari *Event* untuk diproses nanti oleh proses sekunder, menggunakan cron, sebuah server antrean, atau dengan teknik serupa. Hal ini (PSR-1513) BISA membuat serial objek *Event* itu sendiri untuk melakukannya; namun harus berhati-hati agar tidak semua objek *Event* dapat diserialkan dengan aman. Sebuah proses sekunder (PSR-1514) HARUS berasumsi bahwa setiap perubahan yang dibuatnya ke suatu objek *Event* tidak akan menyebar ke *Listener* lain.

- PSR-16: Simple Cache
  - *Library* pelaksana HARUS menyediakan kelas yang mengimplementasikan antarmuka `Psr\SimpleCache\CacheInterface interface`. *Library* pelaksana HARUS mendukung fungsionalitas TTL minimum seperti yang disebutkan pada PSR-06.
- PSR-17: HTTP Factories
  - HTTP *factory* adalah metode yang digunakan untuk membuat objek HTTP baru sesuai yang didefinisikan oleh PSR-07. Setiap HTTP *factory* (PSR-1701) HARUS mengimplementasi semua *interface* berikut untuk setiap tipe objek yang disediakan oleh *package*. *Interface* berikut ini (PSR-1701) BOLEH diimplementasikan bersama dalam satu kelas atau kelas terpisah.
    i. RequestFactoryInterface
    ii. ResponseFactoryInterface
    iii. ServerRequestFactoryInterface
    iv. StreamFactoryInterface
    v. UploadFileFactoryInterface
    vi. UriFactoryInterface
- PSR-18: HTTP Client
  - Klien (PSR-1801) BOLEH mengirim permintaan HTTP yang diubah dari yang disediakan, misalnya melakukan *compress* pada badan pesan yang dikirim.
  - Klien (PSR-1802) BOLEH 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 (PSR-1803) HARUS memastikan bahwa objek tetap konsisten secara internal. Misalnya, jika klien memilih untuk dekompresi isi pesan, maka klien juga (PSR-1804) HARUS menghapus header `Content-Encoding` dan menyesuaikan header `Content-Length`.
  - Klien (PSR-1805) HARUS 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 (PSR-1806) TIDAK BOLEH memperlakukan permintaan atau respons HTTP yang dibuat dengan baik sebagai kondisi *error*. Sebagai contoh, kode status respons dalam rentang 400 dan 500 (PSR-1807) TIDAK BOLEH menimbulkan pengecualian dan (PSR-1808) HARUS dikembalikan ke *library* pemanggil seperti biasa.

  - Klien (PSR-1809) HARUS 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 (PSR-1810) HARUS memberikan *instance* `Psr\Http\Client\RequestExceptionInterface`.
  - Jika permintaan tidak dapat dikirim karena kegagalan jaringan dalam bentuk apa pun, termasuk *timeout*, klien (PSR-1811) HARUS memberi instance `Psr\Http\Client\NetworkExceptionInterface`.
  - Klien (PSR-1812) MUNGKIN memberikan pengecualian yang lebih spesifik daripada yang ditentukan di sini (misalnya `TimeOutException` atau `HostNotFoundException`), asalkan mereka mengimplementasikan antarmuka yang sesuai yang ditentukan di atas.
- PSR-20: Clock
  - Kembalian waktu (PSR-2001) HARUS ditulis sebagai `\DateTimeImmutable`

(d) PHP Linter pada dasarnya digunakan untuk membantu melakukan pengecekan penulisan kode PHP secara keseluruhan dengan standar tertentu. Pada rencana kerja sebelumnya, PHP Linter yang digunakan adalah yang dibuat oleh pengguna GitHub dengan nama Brueggern. *Linter* ini dapat memeriksa berdasarkan PSR-02 (*Deprecated*) dan PSR-12. Fiturnya dapat menunjukkan kesalahan penulisan dan memberikan rekomendasi perbaikannya. Satu fitur lainnya yaitu perintah untuk memperbaiki kesalahan-kesalahan tersebut secara otomatis. Namun setelah dipelajari lebih lanjut, *linter* tersebut menggunakan PHP Coding Standards Fixer (PHP CS Fixer) yang dimodifikasi menjadi lebih sederhana untuk digunakan. Pada dokumentasinya, PHP CS Fixer memiliki *rule set* untuk PSR-01, PSR-02 (*Deprecated*), dan PSR-12. PHP CS Fixer dapat diatur untuk memberikan tanda bagian mana saja dari kode yang perlu diperbaiki dan juga memberikan perbaikan secara otomatis tergantung perintah yang diberikan. Dengan PHP CS Fixer, proses pemeriksaan menjadi lebih fleksibel karena perintah yang diberikan lebih beragam dibandingkan PHP Linter sebelumnya yang hanya memiliki dua perintah utama. Maka dari itu, *tool* yang digunakan adalah PHP CS Fixer. PHP Coding Standards Fixer (PHP CS Fixer) adalah sebuah *tool* yang dapat memperbaiki kode program agar mengikuti standar tertentu, misalnya standar dari PSR-1, PSR-2, dan lainnya, atau standar berbasis komunitas lain seperti Symfony. Dari semua standar yang tersedia hanya satu standar yang akan ditentukan sebagai *ruleset* yaitu PSR-12.

Pada dasarnya, PHP CS Fixer tidak perlu terintegrasi dengan SharIF Judge untuk menjalankan fungsinya. Salah satu cara untuk menginstalnya adalah dengan mengunduh `php-cs-fixer-v3.phar` dari dokumentasi PHP CS Fixer dan menjalankannya dengan perintah dan *flag* tertentu. Perintah dan *flag* yang digunakan adalah:

```
php php-cs-fixer-v3.phar fix path/to/code.php --rules=@PSR12 --dry-run --
diff
```

di mana

- `--rules=@PSR12` untuk menentukan standar yang akan digunakan yaitu PSR-12
- `--dry-run` untuk menjalankan *fixer* tanpa melakukan perubahan pada *file*
- `--diff` untuk membuat *fixer* mengembalikan semua perubahan dalam format `udiff`

3. **Mengevaluasi tingkat kepatuhan SharIF Judge terhadap PSR.**
   Status : Ada sejak rencana kerja tugas akhir.

Hasil : Proses pemeriksaan dilakukan dengan cara mengecek terlebih dahulu menggunakan PHP CS Fixer lalu setelah itu diperiksa secara manual sesuai rekomendasi-rekomendasi PSR yang relevan. Dalam suatu *file*, tidak semua rekomendasi akan relevan untuk digunakan karena belum tentu semua jenis perintah, aturan penulisan, dan fitur PHP yang diatur dalam PSR diimplementasikan dalam *file* tersebut. Berikut adalah contoh keluaran dari PHP CS Fixer pada potongan kode `application/controllers/Dashboard.php` dalam bentuk *diff* yang isinya menampilkan kode asli (bertanda "-") dan rekomendasi perbaikannya (bertanda "+").

```
@@ -4,66 +4,70 @@
* @file Dashboard.php
* @author Mohammad Javad Naderi <mjnaderi@gmail.com>
*/
-defined('BASEPATH') OR exit('No direct script access allowed');
+defined('BASEPATH') or exit('No direct script access allowed');

class Dashboard extends CI_Controller
{
+    public function __construct()
+    {
+        parent::__construct();
+        if (! $this->db->table_exists('sessions')) {
+            redirect('install');
+        }
+        if (! $this->session->userdata('logged_in')) { // if not logged in
+            redirect('login');
+        }
+        $this->load->model('notifications_model')->helper('text');
+    }


-  public function __construct()
-  {
-   parent::__construct();
-   if ( ! $this->db->table_exists('sessions'))
-    redirect('install');
-   if ( ! $this->session->userdata('logged_in')) // if not logged in
-    redirect('login');
-   $this->load->model('notifications_model')->helper('text');
-  }
```

Jika diperiksa dengan seksama, kode asli (bertanda "-") memiliki jarak indentasi satu kali *tab* dari sisi kiri awal penulisan yang mana seharusnya dimulai dengan empat spasi untuk setiap level indentasi seperti yang ada dalam PSR-12: "Kode HARUS menggunakan indentasi sebanyak 4 spasi untuk setiap level indentasi, dan TIDAK BOLEH menggunakan *tab-tab* untuk indentasi". Hal ini mungkin tidak terlihat secara sekilas oleh mata sehingga harus diperiksa kembali setiap indentasinya. Hasil rekomendasi perbaikan yang diberikan keluaran (bertanda "+") di atas menampilkan indentasi sepanjang empat spasi dan kode aslinya (bertanda "-") sepanjang satu *tab*. Hal ini akan lebih jelas jika diperiksa secara langsung melalui terminal. Sejauh ini, semua *file* sudah diperiksa dengan PHP CS Fixer dan hasil keluarannya lebih lengkap ada di Lampiran dokumen Tugas Akhir karena terlalu panjang jika dicantumkan semuanya di Progress Report. Secara umum, standar yang belum terpenuhi sama dengan contoh di atas yaitu penggunaan *tab* yang seharusnya menggunakan empat spasi.

4. **Mengimplementasikan rekomendasi PSR sesuai hasil evaluasi.**
   Status : Ada sejak rencana kerja tugas akhir.
   Hasil : Proses implementasi berjalan beriringan dengan pemeriksaan. Dengan bantuan PHP CS Fixer, hasil keluaran dalam contoh potongan kode sebelumnya dapat diimplementasikan secara langsung untuk memenuhi PSR-12. Perintah yang diberikan dapat diubah untuk melakukan *fix* secara langsung. Dikarenakan sejauh ini baru dilakukan pemeriksaan menyeluruh dengan *fixer*, belum ada contoh rekomendasi perbaikan berdasarkan PSR yang lain selain yang ditentukan dalam PHP CS Fixer. Pemeriksaan secara manual sedang dalam tahap pengerjaan sambil mengecek standar-standar PSR yang relevan untuk setiap *file* PHP.

5. **Menguji SharIF Judge yang sudah dievaluasi dan diperbaiki.**
   Status : Ada sejak rencana kerja tugas akhir.
   Hasil : Belum terlaksana karena SharIF Judge belum berhasil dijalankan dan evaluasi belum selesai dilakukan.

6. **Menulis dokumen tugas akhir.**
   Status : Ada sejak rencana kerja tugas akhir.
   Hasil : Dokumen tugas akhir yang sudah dibuat adalah Bab 1 Pendahuluan dan sebagian besar Bab 2 Landasan Teori, terutama PSR dan PHP CS Fixer. Tahap analisis, evaluasi, implementasi, dan pengujian (Bab 3 dan 4) sedang dalam proses sesuai yang dipaparkan pada poin-poin sebelumnya.

6 Pencapaian Rencana Kerja

Langkah-langkah kerja yang berhasil diselesaikan dalam Tugas Akhir 1 ini adalah sebagai berikut:

1. Membuat dokumen tugas akhir Bab 1 Pendahuluan dan sebagian besar Bab 2 Landasan Teori.
2. Melakukan studi literasi PSR dan PHP CS Fixer.
3. Menentukan strategi untuk mengevaluasi keseluruhan PHP SharIF Judge.
4. Mengevaluasi sebagian PHP pada SharIF Judge, khususnya yang dapat dilakukan dengan bantuan PHP CS Fixer.

Bandung, 20/12/2023

Nicholas Khrisna Sandyawan

Menyetujui,

Nama: Pascal Alfadian Nugroho
Pembimbing Tunggal