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
2023

DAFTAR ISI

Daftar Isi iii

Daftar Gambar v

1 Pendahuluan 1
- 1.1 Latar Belakang . . . 1
- 1.2 Rumusan Masalah . . . 2
- 1.3 Tujuan . . . 2
- 1.4 Batasan Masalah . . . 2
- 1.5 Metodologi . . . 2
- 1.6 Sistematika Pembahasan . . . 3

2 Landasan Teori 5
- 2.1 SharIF Judge . . . 5
  - 2.1.1 Fitur . . . 5
  - 2.1.2 Instalasi . . . 5
- 2.2 PHP Standards Recommendations . . . 7
  - 2.2.1 Accepted . . . 7
  - 2.2.2 Draft . . . 14
  - 2.2.3 Abandoned . . . 14
  - 2.2.4 Deprecated . . . 15
- 2.3 PHP Coding Standards Fixer . . . 15

3 Analisis 17
- 3.1 Metodologi Pemeriksaan . . . 17
- 3.2 Tingkat Kepatuhan SharIF Judge terhadap PSR . . . 17
  - 3.2.1 Pemeriksaan dengan PHP CS Fixer . . . 18
  - 3.2.2 Pemeriksaan Tanpa PHP CS Fixer . . . 21

4 Implementasi dan Pengujian 23
- 4.1 Implementasi . . . 23
- 4.2 Pengujian . . . 23

5 Kesimpulan dan Saran 25
- 5.1 Kesimpulan . . . 25
- 5.2 Saran . . . 25

Daftar Referensi 27

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 Interoperability 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, *file-file* PHP pada SharIF Judge dievaluasi dan diukur tingkat kepatuhannya

terhadap aturan PSR serta dibuat rekomendasi perbaikannya. Walaupun demikian, pada akhirnya ditentukan strategi untuk melakukannya, yaitu menggunakan *tools* atau alat tertentu untuk membantu dan membatasi jumlah *file* PHP yang diperiksa karena sebagian besar harus dilakukan secara manual. Salah satu alat yang digunakan adalah PHP Coding Standards Fixer (PHP CS Fixer). PHP CS Fixer membantu proses pemeriksaan sesuai salah satu standar, yaitu Extended Coding Style Guide atau aturan penulisan PHP yang sudah diperbarui.

Perangkat lunak akhir yang dibuat memiliki fitur yang sama persis dengan SharIf Judge yang sudah ada. Perbedaannya terdapat pada struktur penulisan PHP yang sudah dievaluasi dan diperbaiki sehingga memenuhi aturan 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:

- 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?

1.3 Tujuan

Tujuan yang ingin dicapai dalam penulisan skripsi ini sebagai berikut:

- Mengukur tingkat kepatuhan kode PHP pada SharIF Judge terhadap PSR.
- Membuat rekomendasi perbaikan pada kode PHP SharIF Judge agar meningkatkan jumlah PSR yang terpenuhi.

1.4 Batasan Masalah

- PSR-07 dan PSR-12 tidak dituliskan aturannya pada Landasan Teori dikarenakan jumlah aturan yang sangat banyak dan akan sangat memakan waktu. Evaluasi PSR-12 dilakukan pemeriksaannya secara otomatis dengan bantuan PHP CS Fixer.

1.5 Metodologi

Metode penelitian yang akan digunakan dalam skripsi ini adalah:

1. Mempelajari SharIF Judge saat ini

2. Melakukan studi literatur mengenai PSR dan PHP CS Fixer
3. Mengevaluasi tingkat kepatuhan PHP dari SharIF Judge terhadap PSR
4. Memberikan rekomendasi sesuai hasil evaluasi
5. Menguji SharIF Judge yang sudah dievaluasi dan diperbaiki
6. Menulis dokumen skripsi

1.6 Sistematika Pembahasan

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

1. Bab 1 Pendahuluan
   Bab ini berisi latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan.
2. Bab 2 Landasan Teori
   Bab ini berisi dasar-dasar teori yang digunakan sebagai acuan dalam pembuatan skripsi, antara lain SharIF Judge, PSR, dan PHP CS Fixer.
3. Bab 3 Analisis
   Bab ini berisi analisis tingkat kepatuhan SharIF Judge dan rekomendasi perbaikannya berdasarkan aturan PSR.
4. Bab 4 Implementasi dan Pengujian
   Bab ini berisi implementasi dari rekomendasi perbaikan dan menguji jalannya aplikasi yang sama dengan kode PHP yang sudah diperbaiki.
5. Bab 5 Kesimpulan dan Saran
   Bab ini berisi kesimpulan dari hasil evaluasi dan saran yang dapat digunakan untuk penelitian lebih lanjut.

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

Berikut adalah persyaratan dan langkah-langkah instalasi yang perlu dilakukan untuk menjalankan SharIF Judge.

Persyaratan

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*. Pada Ubuntu, pengguna perlu menginstal paket `php5-cli`.
- Menggunakan *database* MySql atau PostgreSql.
- PHP harus diberikan akses untuk menjalankan perintah menggunakan fungsi `shell_exec`. Contoh perintahnya adalah sebagai berikut:

```
1 echo shell_exec("php -v");
2
```

- *Tools* yang digunakan untuk melakukan kompilasi dan menjalankan kode yang dikumpulkan (perintah `gcc`, `g++`, `javac`, `java`, `python2`, `python3`).
- Perl lebih baik diinstal untuk alasan ketepatan waktu, batas memori dan memaksimalkan batas ukuran pada output kode yang dikirim.

Instalasi

Berikut adalah langkah-langkah instalasi SharIF Judge.

1. Mengunduh versi terbaru dari SharIF Judge dan *unpack* hasil unduhan di direktori *public html*.
2. Memindahkan folder `system` dan `application` ke luar direktori *public*, dan masukkan *path* lengkap di *file* `index.php`.

```
1 $system_path = '/home/mohammad/secret/system';
2 $application_folder = '/home/mohammad/secret/application';
3
```

3. Membuat sebuah *database* Mysql atau PostgreSql untuk SharIF Judge. Jangan menginstall paket koneksi *database* untuk C/C++, Java, atau Python.
4. Mengatur koneksi *database* di *file* `application/config/database.php` dan simpan dengan nama `database.php`. Pengguna dapat menggunakan awalan untuk nama tabel.

```
1 /*  Enter database connection settings here:  */
2 'dbdriver' => 'postgre',    // database driver (mysqli, postgre)
3 'hostname' => 'localhost',  // database host
4 'username' => ‘,           // database username
5 'password' => ‘,           // database password
6 'database' => ‘,           // database name
7 'dbprefix' => 'shj_',       // table prefix
8 /*********************************************/
9
```

5. Mengatur server `RADIUS` dan server *mail* pada *file application/config/secrets.example.php* dan simpan dengan nama `secrets.php`.
6. Membuat direktori `application/cache/Twig` agar dapat ditulis oleh PHP.
7. Membuka halaman utama SharIF Judge pada *web browser* dan mengikuti proses instalasi berikutnya.

8. *Log in* menggunakan akun admin.
9. 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.

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. Walaupun tidak resmi, PSR secara *de facto* diakui oleh banyak pengembang sebagai standar penulisan PHP, terutama PSR-01: Basic Coding Standard. Dalam berbagai proyek aplikasi yang menggunakan PHP setidaknya penulisannya mengikuti PSR-01, misalnya penggunaan *tag* `<?php ?>` dan `<?= ?>`. Selain itu, kredibilitas PSR didukung juga oleh orang-orang yang pernah atau masih terlibat dalam grup PHP-FIG seperti Taylor Otwell selaku pendiri Laravel, Jordi Boggiano salah satu pendiri Composer, Cees-Jan Kiewiet salah satu pendiri ReactPHP, dan lain-lain. Dalam PSR, terdapat kata kunci prioritas yang diatur dalam dokumen RFC 2119. RFC 2119 [4] berisi tentang penggunaan kata kunci yang menunjukkan spesifikasi *requirement*-nya. Misalnya beberapa kode harus diaplikasikan sesuai standar, sedangkan beberapa lainnya bersifat opsional. 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". Aturan-aturan PSR yang digunakan hanyalah yang berstatus "Accepted". Saat dokumen ini dibuat, terdapat 23 bab PSR dengan rincian sebagai berikut.

2.2.1 Accepted

Status "Accepted" adalah bab-bab aturan yang sudah diterima, disepakati, dan diawasi oleh tim kerja bersangkutan untuk digunakan oleh para pengembang. Untuk mempermudah sitasi dalam proses evaluasi, setiap aturan PSR ditandai dengan kode PSR-XXYY, di mana XX menunjukkan bab PSR dan YY menunjukkan urutan aturan dalam bab tersebut.

[1] Beberapa kata kunci memiliki tingkatan yang setara sehingga dikategorikan dan ditulis dalam satu poin. Misalnya, "MUST", "REQUIRED", dan "SHALL" berada dalam satu tingkatan yang setara sesuai deskripsi pada RFC 2119. Pemilihan kata "HARUS", "DILARANG", dan sebagainya digunakan untuk penyesuaian dalam bahasa Indonesia agar struktur kalimat tidak rancu.

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>(\<SubNamespaceNames>)*\<Clas`
  dengan spesifikasi berikut:
  1. (PSR-0401) HARUS memiliki nama *namespace* tingkat tertinggi, atau dikenal sebagai "vendor namespace".
  2. (PSR-0402) BISA memiliki satu atau lebih nama *sub-space*.
  3. (PSR-0403) HARUS memiliki nama kelas di akhir.
  4. Garis bawah tidak memiliki makna khusus.
  5. Karakter abjad (PSR-0404) BISA berisi kombinasi dari huruf kapital dan huruf kecil.
  6. 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:

1. URI yang merepresentasikan *resource* target yang direferensikan.
2. 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.

1. Implementing Object - Objek yang mengimplementasikan salah satu antarmuka yang ditentukan oleh spesifikasi ini.
2. 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\Ca`
  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.
  1. RequestFactoryInterface
  2. ResponseFactoryInterface
  3. ServerRequestFactoryInterface
  4. StreamFactoryInterface
  5. UploadFileFactoryInterface
  6. 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`

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 Coding Standards Fixer

PHP Coding Standards Fixer (PHP CS Fixer) [5] 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. Berikut adalah perintah dan *flag* yang dapat digunakan.

Perintah

- Perintah `fix` akan memperbaiki masalah standar penulisan kode sebanyak mungkin. Jika ada *file config* perintah yang dijalankan adalah:

```
1 php php-cs-fixer-v3.phar fix
2
```

  Jika tidak ada *file config*, perintah berikut dapat dijalankan untuk memperbaiki *file-file* yang *non-hidden*, *non-vendor*, atau yang memiliki *ruleset* bawaan @PSR12:

```
1 php php-cs-fixer-v3.phar fix .
2
```

  Selain itu, perintah ini dapat dijalankan untuk *file-file* pada *path* tertentu dengan perintah:

```
1 php php-cs-fixer-v3.phar fix /path/to/dir
2 php php-cs-fixer-v3.phar fix /path/to/file
3
```

- Perintah `check` adalah versi singkat dari `fix --dry-run` dan menyediakan semua opsi dan argumen sebagai perintah `fix`. Perbedaannya adalah perintah `check` tidak akan menerapkan perubahan apa pun, tetapi hanya mencetak hasil analisis.
- Perintah `list-files` akan mencantumkan semua file yang perlu diperbaiki.

```
php php-cs-fixer-v3.phar list-files
```

Flag

- *Flag* `--rules` digunakan untuk menentukan *rules* yang akan diterapkan dalam *project*:

```
php php-cs-fixer-v3.phar fix /path/to/project --rules=@PSR12
```

  Secara *default*, *rules* yang digunakan adalah `PSR12`. Jika *flag* `--rules` digunakan, maka *rules* dari *file config* diabaikan.
- *Flag* `--dry-run` akan menjalankan *fixer* tanpa megubah apa pun.
- *Flag* `--diff` dapat digunakan untuk memberikan *output* yang berisi semua perubahan yang dibuat dalam format `udiff`.

Sebagai contoh, perintah dan flag yang digunakan untuk memeriksa *Dashboard.php* adalah:

```
php php-cs-fixer-v3.phar fix application/controllers/Dashboard.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`

BAB 3

ANALISIS

Bab ini membahas tentang analisis terkait evaluasi yang akan dilakukan pada SharIF Judge.

3.1 Metodologi Pemeriksaan

Sebelum menentukan metodologi yang akan digunakan, ada beberapa hal yang perlu dipertimbangkan terlebih dahulu. Hal ini akan memengaruhi cara serta hasil akhir pemeriksaan dan evaluasi. Hal-hal tersebut antara lain sebegai berikut.

- Jumlah *file* PHP dari SharIF Judge yang diperiksa berjumlah 34. Jumlah ini didasarkan pada jumlah *file* yang memiliki tanda "SharIF Judge" di dalamnya. *File-file* yang dibuat oleh *framework* atau *library* lainnya tidak diperiksa. Walaupun ada kemungkinan bahwa telah dilakukan perubahan atau tambahan kode pada *file-file* yang dibuat *framework*, pemeriksaan tetap tidak dilakukan karena akan cukup sulit dan memakan waktu yang sangat banyak untuk mencarinya satu per satu.
- Saat dokumen ini dibuat, hanya tersedia *tools* yang dapat membantu memeriksa penulisan kode sesuai PSR-12. Selain dari itu pemeriksaan harus dilakukan secara manual.
- Beberapa aturan PSR mungkin tidak relevan dengan *file* PHP yang diperiksa. Sebagai contoh, tidak semua *file* PHP menggunakan fungsi yang berkaitan dengan waktu dan tanggal dalam kodenya sehingga PSR-20: Clock tidak relevan dalam pemeriksan *file-file* tersebut.

Dengan pertimbangan-pertimbangan tersebut, maka metodologi pemeriksaan yang dilakukan adalah sebagai berikut.

1. Melakukan pemeriksaan secara otomatis sesuai rekomendasi PSR-12 dengan bantuan PHP CS Fixer.
2. Melakukan pemeriksaan pemenuhan rekomendasi PSR lainnya secara manual tanpa PHP CS Fixer.

3.2 Tingkat Kepatuhan SharIF Judge terhadap PSR

Jumlah *file* PHP yang diperiksa adalah 34, di luar *file-file* bawaan CodeIgniter. Setiap *file* tersebut diperiksa sesuai kebutuhannya karena tidak semua rekomendasi PSR relevan terhadap kode program. Dalam prosesnya, pemeriksaan dibantu dengan PHP CS Fixer.

3.2.1 Pemeriksaan dengan PHP CS Fixer

PHP CS Fixer menyediakan fitur untuk menampilkan kode program yang diperiksa beserta dengan rekomendasi perubahannya sesuai dengan *ruleset* yang sudah ditentukan sebelumnya yaitu PSR-12. Maka dari itu, PHP CS Fixer sudah memeriksa dan memberikan rekomendasi perbaikannya sekaligus. Berikut adalah salah satu contoh ketika melakukan pemeriksaan pada *Dashboard.php*. Setelah menjalankan perintah dan *flag* yang ditentukan, keluaran yang dihasilkan dapat dilihat pada Kode 3.1.

Kode 3.1: Contoh penggunaan PHP CS Fixer pada *Dashboard.php*

```
@@ -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');
-   }
+    // --------------------------------------------------------------------

```

```
34
35 -    // ------------------------------------------------------------------------
36 +     public function index()
37 +     {
38 +         $data = array(
39 +             'all_assignments' => $this->assignment_model->all_assignments(),
40 +             'week_start' => $this->settings_model->get_setting('week_start'),
41 +             'wp' => $this->user->get_widget_positions(),
42 +             'notifications' => $this->notifications_model->
10     get_latest_notifications()
43 +         );
44
45 +         // detecting errors:
46 +         $data['errors'] = array();
47 +         if($this->user->level === 3) {
48 +             $path = $this->settings_model->get_setting('assignments_root');
49 +             if (! file_exists($path)) {
50 +                 array_push($data['errors'], 'The path to folder "assignments" is
19      not set correctly. Move this folder somewhere not publicly accessible, and
20     set its full path in Settings.');
51 +             } elseif (! is_writable($path)) {
52 +                 array_push($data['errors'], 'The folder <code>"'.$path.'"</code>
23      is not writable by PHP. Make it writable. But make sure that this folder is
24     only accessible by you. Codes will be saved in this folder!');
53 +             }
54
55 -    public function index()
56 -    {
57 -        $data = array(
58 -            'all_assignments'=>$this->assignment_model->all_assignments(),
59 -            'week_start'=>$this->settings_model->get_setting('week_start'),
60 -            'wp'=>$this->user->get_widget_positions(),
61 -            'notifications' => $this->notifications_model->
34     get_latest_notifications()
62 -        );
63 +             $path = $this->settings_model->get_setting('tester_path');
64 +             if (! file_exists($path)) {
65 +                 array_push($data['errors'], 'The path to folder "tester" is not
39     set correctly. Move this folder somewhere not publicly accessible, and set its
40      full path in Settings.');
66 +             } elseif (! is_writable($path)) {
67 +                 array_push($data['errors'], 'The folder <code>"'.$path.'"</code>
```

```
     is not writable by PHP. Make it writable. But make sure that this folder is
    only accessible by you.');
+           }
+       }

-       // detecting errors:
-       $data['errors'] = array();
-       if($this->user->level === 3){
-           $path = $this->settings_model->get_setting('assignments_root');
-           if ( ! file_exists($path))
-               array_push($data['errors'], 'The path to folder "assignments" is
    not set correctly. Move this folder somewhere not publicly accessible, and set
     its full path in Settings.');
-           elseif ( ! is_writable($path))
-               array_push($data['errors'], 'The folder <code>"'.$path.'"</code>
    is not writable by PHP. Make it writable. But make sure that this folder is
    only accessible by you. Codes will be saved in this folder!');
+        $this->twig->display('pages/dashboard.twig', $data);
+    }

-           $path = $this->settings_model->get_setting('tester_path');
-           if ( ! file_exists($path))
-               array_push($data['errors'], 'The path to folder "tester" is not
    set correctly. Move this folder somewhere not publicly accessible, and set its
     full path in Settings.');
-           elseif ( ! is_writable($path))
-               array_push($data['errors'], 'The folder <code>"'.$path.'"</code>
    is not writable by PHP. Make it writable. But make sure that this folder is
    only accessible by you.');
-       }

-       $this->twig->display('pages/dashboard.twig', $data);
-   }
+    // --------------------------------------------------------------------

+    /**
+     * Used by ajax request, for saving the user's Dashboard widget positions
+     */
+    public function widget_positions()
+    {
+        if (! $this->input->is_ajax_request()) {
+            show_404();
```

```
100 100 +         }
101 101 +         if ($this->input->post('positions') !== null) {
102 102 +             $this->user->save_widget_positions($this->input->post('positions'));
103 103 +         }
104 104 +     }
105 105
106 106 -     // ------------------------------------------------------------------------
107 107 -
108 108 -     /**
109 109 -      * Used by ajax request, for saving the user's Dashboard widget positions
110 110 -      */
111 111 -     public function widget_positions()
112 112 -     {
113 113 -         if ( ! $this->input->is_ajax_request() )
114 114 -             show_404();
115 115 -         if ($this->input->post('positions') !== NULL)
116 116 -             $this->user->save_widget_positions($this->input->post('positions'));
117 117 -     }
118 118 -
119 119 -}
120 120 \ No newline at end of file
121 121 +}
123
```

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.

3.2.2 Pemeriksaan Tanpa PHP CS Fixer

Setelah pemeriksaan dengan bantuan PHP CS Fixer, selanjutnya dilakukan pemeriksaan sesuai rekomendasi PSR selain PSR-12 pada setiap *file* PHP secara manual. Setiap hasil pemeriksaan akan ditandai sebagai "terpenuhi" jika rekomendasi PSR sudah terpenuhi, "belum terpenuhi" jika rekomendasi PSR belum terpenuhi, dan "tidak relevan" jika rekomendasi dari PSR tersebut tidak relevan dengan kode program yang diperiksa.

PSR-01

- Assignments.php
- Dashboard.php
- Halloffame.php
- Install.php
- Login.php
- Logs.php
- Moss.php
- Notifications.php
- Problems.php
- Profile.php
- Queue.php
- Queueprocess.php
- Rejudge.php
- Scoreboard.php
- Settings.php
- Submissions.php
- Submit.php
- Users.php
- User.php
-

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Bagian ini membahas perbaikan apa saja yang dilakukan untuk memenuhi rekomendasi-rekomendasi PSR yang belum terpenuhi sesuai hasil analisis di subbab 3.2. Setiap perubahan ditampilkan dalam format *diff* dengan jarak indentasi yang diubah untuk memperbaiki penempatan dan menghemat ruang.

4.2 Pengujian

Bagian ini berisi skenario dan hasil yang didapatkan dari setiap skenario pengujian. Tujuan dari pengujian ini adalah untuk memastikan bahwa perbaikan yang dilakukan pada subbab 4.1 terhadap kode program tidak mengubah jalannya aplikasi SharIF Judge sesuai fitur-fitur yang sudah ada.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

5.2 Saran

DAFTAR REFERENSI

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

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

[3] Version 1.4.1 (2015) *Sharif-Judge*. Mohammad Javad Naderi. Tehran, Iran.

[4] Bradner, S. (1997) Key words for use in rfcs to indicate requirement levels. RFC 2119. RFC Editor, http://www.rfc-editor.org.

[5] Version 3.30.0 (2023) *PHP-Coding-Standards-Fixer*. Dariusz Rumiński and Julián Gutiérre and HypeMC and SpacePossum.