BAB II
LANDASAN TEORI

Pada bab ini membahas tentang landasan teori yang berkaitan dengan sistem yang akan dibuat. Landasan teori ini membahas tentang *class diagram, website, framework,* laravel dan HMVC. Semua pembahasan tersebut berguna dalam menunjang sistem yang akan dibuat, sehingga sistem dapat berjalan sesuai yang diharapkan.

2.1 *Class Diagram*

Class diagram menggambarkan struktur statis dari kelas dalam sistem anda dan menggambarkan atribut, operasi dan hubungan antara kelas. *Class diagram* membantu dalam memvisualisasikan struktur kelas-kelas dari suatu sistem dan merupakan tipe diagram yang paling banyak dipakai. Selama tahap desain, *class diagram* berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem yang dibuat.

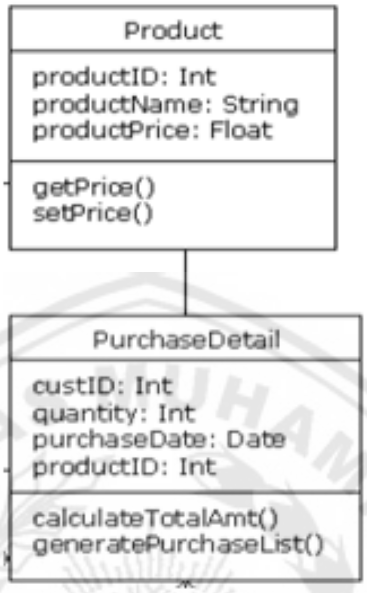
Class diagram memiliki 3 area pokok (utama) yaitu: nama, atribut dan operasi. Nama berfungsi untuk member identitas pada sebuah *class*, atribut fungsinya adalah untuk member karakteristik pada data yang dimiliki suatu objek didalam kelas, sedangkan operasi fungsinya adalah memberikan sebuah fungsi ke sebuah objek. Dalam mendefinisikan metode yang ada didalam *class diagram* harus diperhatikan yang namanya *Cohesion* dan *Coupling*, *Cohesion* adalah ukuran keterkaitan sebuah instruksi disebuah metode, *Coupling* adalah ukuran keterkaitan antar metode. Didalam *class diagram* terdapat hubungan antar *class* secara konseptual, yang disebut relasi antar *class*.

2.1.1 Macam-macam Relasi

UML disediakan macam-macam relasi antar *class*, diantaranya: Asosiasi (Hubungan statis antar kelas), Agregasi (hubungan dari keseluruhan objek), Generalisasi (relasi beberapa subkelas ke super kelas), Dependensi (keterhubungan tiap kelas).

2.1.1.1 Asosiasi / *Association*

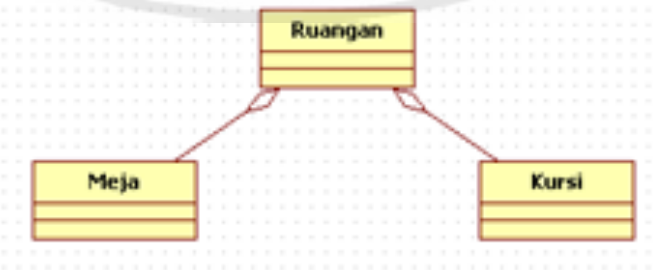
Asosiasi yaitu hubungan statis antar kelas. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.



Gambar 2.1 *Relasi Asosiasi*

2.1.1.2 Agregasi / *Agregation*

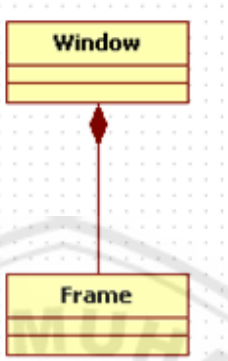
Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas"). Agregasi merupakan hubungan antara satu *object* dengan *object* lainnya dimana *object* satu dengan *object* lainnya sebenarnya terpisah namun disatukan, sehingga tidak terjadi kebergantungan (*Object* lain bisa terbentuk walaupun object penampungnya belum terbentuk).



Gambar 2.2 *Relasi Agregasi*

2.1.1.3 Composisi / *Compostion*

Composisi adalah aggregasi dengan ikatan yang lebih kuat. Di dalam *composition aggregation*, siklus hidup part *class* sangat bergantung pada *whole class* sehingga bila objek instance dari *whole class* dihapus maka *object instance* dari bagian *class* juga akan terhapus.



Gambar 2.3 *Relasi Composisi*

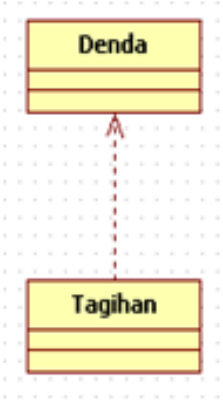
2.1.1.4 Generalisasi / *Generalization*

Generalisasi yaitu hubungan pewarisan (*inheritance*) antar unsur dalam *class diagram*. Pewarisan memungkinkan suatu kelas mewarisi semua atribut, operasi, relasi dari kelas yang berada dalam hirarki pewarisannya

Gambar 2.4 *Relasi Generalisasi*

2.1.1.5 Kebergantungan / *Dependency*

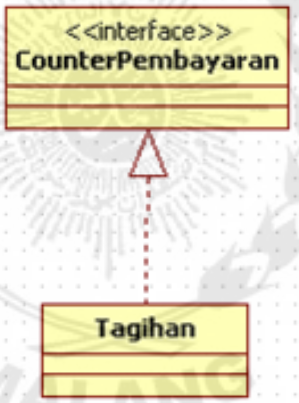
Kebergantungan yaitu hubungan antar *class* di mana sebuah *class* memiliki ketergantungan pada *class* lainnya tetapi tidak sebaliknya.



Gambar 2.5 *Relasi Kebergantungan*

2.1.1.6 Realisasi / *Realization*

Realisasi adalah hubungan antar *class* dimana sebuah *class* memiliki keharusan untuk mengikuti aturan yang ditetapkan *class* lainnya. Biasanya realisasi digunakan untuk menspesifikasikan hubungan antara sebuah *interface* dengan *class* yang mengimplementasikan *interface* tersebut.



Gambar 2.6 *Relasi Relasi*

2.1.2 Elemen-elemen *Class Diagram*

Digambarkan dengan bujur sangkar yang memiliki tiga ruangan yaitu: Nama, Atribut dan *Method* (fungsi).

2.1.3 *Class Diagram* Sebagai acuan kerja

Website builder ini mengacu pada *class diagram* yang sudah ada untuk membuat 2 komponen utama pada aplikasi yaitu *table management* dan *module management. Table management* adalah fitur yang disediakan *website builder*

untuk membuat atau mengatur tabel-tabel yang digunakan untuk menyimpan data didalam sistem, sedangkan *module management* adalah fitur yang digunakan untuk membuat *controller* yang mengelola antara *database* dengan tampilan aplikasi. *Output* dari *table management* dan *module management* merupakan satu kesatuan yang nantinya akan digunakan untuk membuat sebuah *website*.

2.2 *Website*

2.2.1 Pengertian *Website*

Website merupakan sebuah layanan yang menampilkan sebuah halaman yang berisi informasi dalam bentuk digital. *Website* menggunakan bahasa HTML (*Hypertext Markup Language*) dan berjalan pada protokol HTTP atau *Hypertext Transfer Protocol*.[3]

2.2.2 Kategori *Website*

Secara umum, situs web digolongkan menjadi 3 jenis yaitu: *Website* statis, *Website* dinamis, *Website* interaktif.

2.2.2.1 *Website* statis

Website statis adalah *website* yang mempunyai halaman tetap. Artinya untuk melakukan perubahan pada suatu halaman dilakukan secara manual dengan mengubah baris kode yang menjadi struktur dari *website* tersebut. Contoh umum mengenai *website* statis adalah *website* profil perusahaan atau organisasi.

2.2.2.2 *Website* dinamis

Website dinamis merupakan *website* yang secara struktur diperuntukan untuk update sesering mungkin. Halaman *website* dinamis terdapat dua bagian yaitu bagian depan (*frontend*) dan halaman admin (*backend*) untuk mengubah isi dan tampilan dari *website*. Contoh umum mengenai *website* dinamis adalah *website* berita atau *website* portal yang di dalamnya terdapat fasilitas berita, polling dan sebagainya.

2.2.2.3 *Website* interaktif

Website interaktif adalah *website* dengan adanya komunikasi antara pengguna dengan komponen yang terdapat di dalam komputer. Komunikasi dapat

melalui *keyboard*, *mouse*, atau alat *input* lainnya. Dalam hal ini pengguna dapat memilih apa yang akan dikerjakan selanjutnya, bertanya dan mendapatkan jawaban yang mempengaruhi komputer untuk mengerjakan fungsi selanjutnya. contoh umum mengenai *website* interaktif adalah blog dan forum atau komunitas.

2.3 *Framework*

Framework adalah kumpulan perintah atau fungsi dasar yang membentuk aturan-aturan tertentu dan saling berinteraksi satu sama lain sehingga dalam pembuatan aplikasi website kita harus mengikuti aturan dari *framework* tersebut[4]. *Framework* juga dapat diartikan sebagai kumpulan kode program (terutama *class* dan *function*) yang dapat membantu *programmer* dalam menangani berbagai masalah-masalah dalam pemrograman seperti koneksi ke *database*, pemanggilan variabel, *file*, dll. Sehingga *programmer* lebih fokus dan lebih cepat dalam membangun sebuah aplikasi. Didalam *framework* terdapat komponen yang re-*useable function*, *library, helper, configuration*, sehingga *programmer* tidak harus membuat baris kode yang sama untuk tugas yang sama.

Framework kelebihan utamanya bukan dari seberapa banyak *library* yang di sediakan, meskipun hal itu tentunya akan sangat membantu proses *development*. Kelebihan yang bisa kita ambil dari *framework* adalah kerangka kerja dari *framework* tersebut dalam menyelesaikan modul-modul yang dikembangkan sehinga mengeluarkan sebuah metode pekerjaan yang lebih efisien, lebih rapi dan lebih bersifat general.

2.3.1 Alasan menggunakan *framework*

Dibawah ini adalah alasan-alasan mengapa menggunakan sebuah *framework* didalam membuat *website* menjadi sangat penting.

- Mempercepat dan mempermudah pembangunan sebuah aplikasi.
- Relatif memudahkan dalam proses *maintenance* karena sudah ada pola tertentu dalam sebuah *framework*.
- Pada umumnya framework menyediakan fasilitas-fasilitas yang umum dipakai sehingga kita tidak membutuhkan membangun kerangka dari awal. misalnya validasi, ORM (*object relational model*), *pagination*, *multiple database, scaffolding*, pengaturan *session*, *error handling*, dll.

- Lebih bebas dalam pengembangan jika dibandingkan CMS (*Content Management System*).

2.4 Laravel *framework*

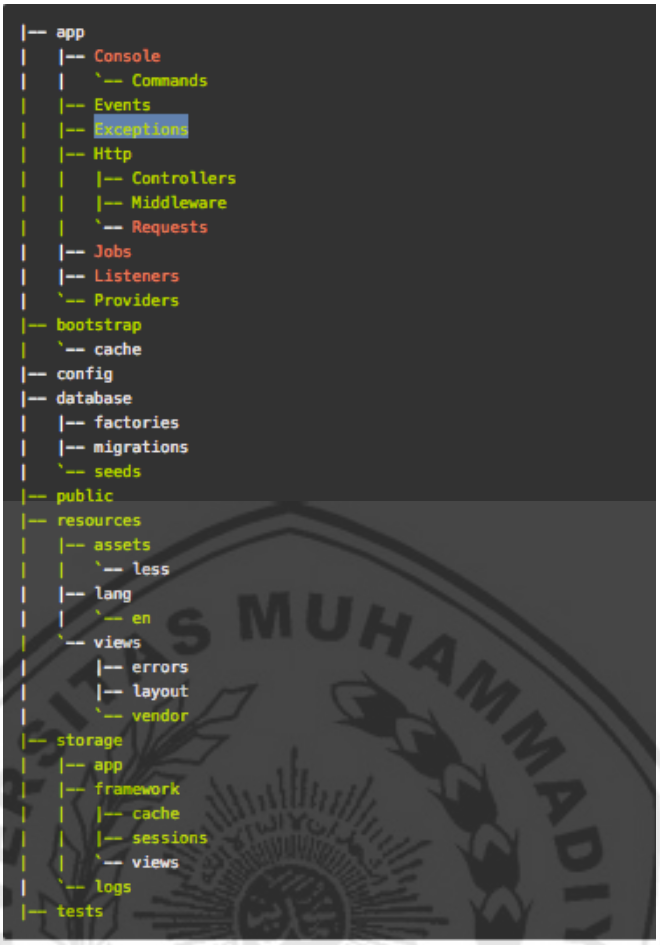
Laravel adalah sebuah *Framework* PHP yang bersifat *opensource* yang ditulis oleh Taylor Otwell dengan lisensi dibawah MIT *License*. Laravel dibuat untuk membatu para *programmer* khususnya dalam membuat sebuah *website* dengan sintak yang sederhana, elegan, ekspresif dan menyenangkan.[6] Seperti yang ditulis dalam websitenya :

Laravel is a clean and classy framework for PHP web development. Freeing you from spaghetti code, it helps you create wonderful applications, using simple, expressive syntax. Development should be a creative experience that you enjoy, not something that is painful. Enjoy the fresh air!

Laravel adalah aplikasi *website* dengan sintak yang ekspresif dan elegan. Dengan Laravel, tugas-tugas umum *programmer* dapat dikurangi pada sebagian besar proyek-proyek *website* seperti *routing*, *session* dan *caching*. Disamping itu, laravel berusaha menggabungkan pengalaman-pengalaman *development* dalam bahasa lain, seperti Ruby on Rails, ASP.NET, MVC dan Sinatra.

2.4.1 Struktur direktori Laravel

Laravel mempunyai standar untuk stuktur folder, setiap folder memiliki peran masing-masing yang berguna untuk memilah atau mengelompokkan *file* agar memudahkan proses *development*. Dibawah ini adalah struktur direktori dari *framework* Laravel dan berikut penjelasannya :



Gambar 2.7 *Struktur direktori laravel*

2.4.1.1 Direktori app/Http

Direktori ini merupakan direktori yang dibuat secara khusus untuk menyimpan seluruh file-file yang berkaitan dengan proses *request* dan *response* Http. Dikretori ini memiliki tiga buah sub direktori yang diantaranya adalah "*Controllers*", "*Middleware*" dan "*Requests*". Berikut ini adalah penjelasan mengenai fungsi dari ketiga buah sub direktori tersebut:

- **app/Http/Controllers**: Direktori ini digunakan untuk menyimpan seluruh *class Controller* yang kita buat seperti misalnya ProductController.php, SalesController.php, dll.
- **app/Http/Middleware**: Direktori ini digunakan untuk menyimpan seluruh *class* yang berhubungan dengna *middleware* PHP. Secara umum *middleware* adalah sebuah *class* yang akan dieksekusi sebelum HTTP *request* yang masuk diberikan kepada *Controller*. Tujuan dari *class Middleware* adalah untuk melakukan filter seperti misalnya menolak akses dari user yang belum login.

- **app/Http/Requests**: Direktori ini hanya berisikan sebuah *class* yaitu Request.php yang dapat digunakan untuk mendapatkan data dari *form request* yang dikirim oleh *web browser*. Selain itu direktori ini juga ditujukan untuk menyimpan *class validator* yang kita buat baik yang dibuat secara manual ataupun dengan menggunakan perintah "php artisan make:request".

2.4.1.2 Direktori database/migrations

Direktori ini berisikan *file-file migrations* yang digenerate oleh laravel pada saat kita menjalankan perintah "php artisan make:migration". fitur *migration* sendiri sangat berguna untuk melakukan perubahan pada *database* baik itu penambahan tabel, penambahan kolom, menghapus kolom, menghapus tabel serta melakukan *roll-back* setiap perubahan *database* yang kita buat. Fitur *migration* ini akan sangat terasa manfaatnya terutama pada saat kita mengerjakan sebuah *project* di dalam sebuah tim dan banyak struktur *database* yang berubah seiring perkembangan *project*.

2.4.1.3 Direktori database/seeds

Direktori ini berisikan *file-file database seeds* yang dibuat oleh laravel pada saat kita menjalankan perintah "php artisan make:seeder". fitur *seeding* di laravel sendiri sangat berguna apabila kita ingin melakukan inisialisasi data (data awalan) pada tabel yang kita buat.

2.4.1.4 Direktori *public*

Pada dasarnya laravel memisahkan antara direktori *public* dan *private*. direktori *public* adalah direktori dimana seluruh *resource* aplikasi dapat diakses melalui *web browser* seperti misalnya gambar, JavaScript dan CSS. Sedangkan direktori *private* sendiri berisikan seluruh kode PHP yang telah kita buat ataupun yang merupakan bawaan dari *framework* laravel itu sendiri. Umumnya, dalam melakukan proses *deployment* laravel yang *secure*, hanya direktori *public* ini sajalah yang diletakkan di dalam direktori public_html pada *web server* sedangkan direktori lainnya diletakkan di luar direktori public_html.

2.4.1.5 Direktori resources

Direktori ini memiliki tiga buah sub direktori yaitu “assets”, “lang” dan “views”. Berikut ini adalah penjelas singkat terkait fungsi dari masing-masing sub direktori tersebut:

- **assets**: Sejak rilis versi 5, laravel memiliki sebuah fitur yang bernama laravel *elixir*. Fitur ini ditujukan untuk membantu para pengguna laravel untuk meng-*compile file* less, saas dan coffescript yang mereka buat. Nah, direktori ini ditujukan untuk menyimpan *resources* tersebut yang nantinya akan secara otomatis di-*compile* oleh laravel dengan menggunakan *gulp* dan dipindahkan ke dalam direktori *public*. Selain itu kita juga dapat menyimpan *resources* berupa *image* atau berkas-berkas lain yang nantinya akan dipindahkan oleh laravel kedalam direktori *public* dengan cara yang sama.
- **lang**: Secara default laravel sudah memiliki *support* terhadap implementasi *localization* yang dapat membantu para pengguna *framework* untuk menciptakan aplikasi *web* yang multi bahasa. Direktori ini menyimpan seluruh definisi bahasa yang telah kita buat.
- **views**: Direktori ini digunakan untuk menyimpan seluruh *file* html / *template blade* yang kita buat.

2.4.1.6 Direktori tests

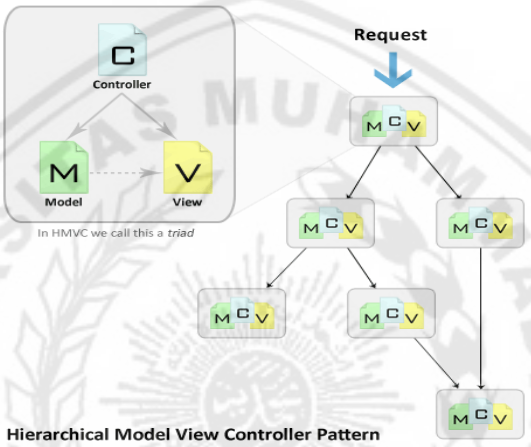
Laravel merupakan sebuah *framework* yang didesain dengan *mindset testable framework*. Oleh karena itu, secara *default* laravel sudah menyediakan *library-library* yang dibutuhkan untuk dapat melakukan *unit testing* seperti PHPUnit dan Mockery. Nah, direktori ini berfungsi untuk menyimpan seluruh *file test* yang dibuat untuk kemudian dijalankan oleh PHPUnit.

2.5 HMVC

HMVC atau *Hierarchical Model View Controller* adalah suatu pola MVC tetapi berupa hirarki dimana dalam implementasinya MVC tersimpan di dalam modul-modul tertentu sehingga setiap modul memiliki *model*, *view*, dan *controller* sendiri [2]. Pada MVC, sebuah sistem dibagi menjadi tiga komponen utama, yaitu: *Model, Controller, dan View*.

Model merupakan komponen yang menangani data pada sistem. *Model* memiliki fungsi untuk memanipulasi data dan dapat mengakses basisdata. *View* adalah komponen yang menangani tampilan visual sistem. *Controller* berisi alur proses sistem dan menangani input dari pengguna sistem. *Controller* dapat memanggil *Model* dan *View* dalam menjalankan fungsinya.

Pada HMVC, sistem dibagi menjadi komponen-komponen yang disebut MVC *triad*. Setiap MVC *triad* adalah komponen yang menggunakan pola MVC. Setiap MVC *triad* saling terkait melalui *Controller* dari masing-masing MVC *triad* tersebut.



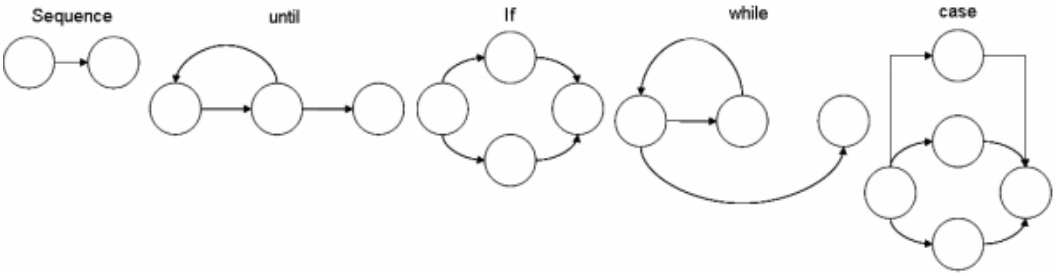
Gambar 2.8 *Struktur HMVC*

2.6 Whitebox Testing

Merupakan metode perancangan *test case* yang menggunakan struktur kontrol dari perancangan prosedural untuk mendapatkan *test case*. *White box testing* bertujuan untuk mengetahui mekanisme kerumitan dalam kode program dengan berfokus pada aliran kontrol atau aliran data dari program[7].

2.6.1 Notasi Diagram Alir (Path Graph Notation)

Notasi yang digunakan untuk menggambarkan jalur eksekusi adalah notasi diagram alir atau grafik program, yang menggunakan notasi lingkaran (simpul atau *node*) dan anak panah (*link* atau *edge*). Notasi ini menggambarkan aliran control logika yang digunakan dalam suatu bahasa pemrograman.



Gambar 2.8 *Notasi diagram alir*

2.6.2 *Cyclomatic Complexity*

Cyclomatic Complexity merupakan suatu sistem pengukuran yang menyediakan ukuran kuantitatif dari kompleksitas logika suatu program. Pada *Basic Path Testing*, hasil dari *cyclomatic complexity* digunakan untuk menentukan banyaknya *independent paths*. *Independent path* adalah sebuah kondisi pada program yang menghubungkan *node* awal dengan *node* akhir.

Terdapat 2 persamaan yang digunakan, yaitu:

V(G) = E - N + 2 atau V(G) = P + 1

Keterangan:

V(G) = *Cyclomatic complexity* untuk *flow graph* G

E = Jumlah *edge*(panah)

N = Jumlah *node*(lingkaran)

P = Jumlah *predicate node*

2.7 ISO-9126

International Organization for Standarization (ISO) dalam ISO Standard 9126 telah mengusulkan beberapa karakteristik untuk melakukan pengujian terhadap kualitas sebuah perangkat lunak. ISO-9126 [8] mengidentifikasikan enam karakteristik sebuah perangkat lunak dikatakan berkualitas yaitu: *functionality, reliability, usability, efficiency, maintability*, dan *portability* yang ditunjukkan pada Tabel 2.1.

Tabel 2.1 Karakteristik dan Sub-karakteristik ISO 9126

Katakteristik	Sub-Karakteristik
Functionality	*Suitability, Accuracy, Interoperability, Security, Compliance*
Reliability	*Maturity, Fault Tolerance, Recoverability*
Usability	*Understandability, Learnability, Operability, Attractiveness*
Efficiency	*Time behavior, Resource Utilization*
Maintainability	*Analyzability, Changeability, Stability, Testability*
Portability	*Adaptability, Installability, Co-existence, Replacability*

2.7.1 *Functionality*

Kemampuan perangkat lunak untuk menyediakan fungsi sesuai kebutuhan pengguna, ketika digunakan dalam kondisi tertentu. Aspek *functionality* diuji ahli pemrograman (*programmer/ developer*) dengan menggunakan kuisioner sesuai dengan fungsi pada *user requirement list*. Sehingga, dapat diketahui fungsi fungsi yang berjalan dan tidak berjalan (*error*).

2.7.1.1 *Suitability*

Kemampuan perangkat lunak untuk menyediakan serangkaian fungsi yang sesuai untuk tugas-tugas tertentu dan tujuan pengguna.

2.7.1.2 *Accuracy*

Kemampuan perangkat lunak dalam memberikan hasil yang presisi dan benar sesuai dengan kebutuhan.

2.7.1.3 *Security*

Kemampuan perangkat lunak untuk mencegah akses yang tidak diinginkan, menghadapi penyusup (*hacker*) maupun otorisasi dalam modifikasi data.

2.7.1.4 *Interoperability*

Kemampuan perangkat lunak untuk berinteraksi dengan satu atau lebih sistem tertentu.

2.7.1.5 *Compliance*

Kemampuan perangkat lunak dalam memenuhi standar dan kebutuhan sesuai peraturan yang berlaku.

2.7.2 *Reliability*

Kemampuan perangkat lunak untuk mempertahankan tingkat kinerja tertentu, ketika digunakan dalam kondisi tertentu. Aspek *reliability* diuji menggunakan pengujian *stress testing*. *Stress testing* adalah salah satu jenis pengujian sistem (*system testing*). *Stress testing* menjalankan sebuah sistem dengan sumber daya jumlah, frekuensi atau volume yang abnormal. Pengujian ini dilakukan oleh peneliti dengan menggunakan *software Web Application Load*, *Stress* and *Performance Testing* yang meliputi beberapa parameter pada *error report* yang ada dalam *software* tersebut: *Failed Session, Failed Hits, dan Failed Pages.*

2.7.2.1 *Maturity*

Kemampuan perangkat lunak untuk menghindari kegagalan sebagai akibat dari kesalahan dalam perangkat lunak.

2.7.2.2 *Fault Tolerance*

Kemampuan perangkat lunak untuk mempertahankan kinerjanya jika terjadi kesalahan perangkat lunak.

2.7.2.3 *Recoverability*

Kemampuan perangkat lunak untuk membangun kembali tingkat kinerja ketika terjadi kegagalan sistem, termasuk data dan koneksi jaringan.

2.7.3 *Usability*

Kemampuan perangkat lunak untuk dipahami, dipelajari, digunakan, dan menarik bagi pengguna, ketika digunakan dalam kondisi tertentu. Aspek *usability* diukur menggunakan kuisioner. Pengujian ini dilakukan dengan menggunakan instrumen *Usefulness*, *Satisfaction*, *and Ease of use* (USE) *Questionnaire* yang dikembangkan oleh *STC Usability and User Experience Community*.

2.7.3.1 *Understandibility*

Kemampuan perangkat lunak dalam kemudahan untuk dipahami.

2.7.3.2 *Learnability*

Kemampuan perangkat lunak dalam kemudahan untuk dipelajari.

2.7.3.3 *Operability*

Kemampuan perangkat lunak dalam kemudahan untuk dioperasikan.

2.7.3.4 *Attractiveness*

Kemampuan perangkat lunak dalam menarik pengguna.

2.7.4 *Efficiency*

Kemampuan perangkat lunak untuk memberikan kinerja yang sesuai dan relatif terhadap jumlah sumber daya yang digunakan pada saat keadaan tersebut. Pengujian ini menggunakan alat ukur YSlow yang dikembangkan oleh Yahoo Developer Network dan Page Speed yang dikembangkan oleh Google Developer untuk mengukur performa efisiensi sebuah halaman website. Performa yang akan diukur adalah besarnya bytes data dokumen, jumlah HTTP request, minifikasi, kompresi GZIP, dan score / grade akhir.

2.7.4.1 *Time behavior*

Kemampuan perangkat lunak dalam memberikan respon dan waktu pengolahan yang sesuai saat melakukan fungsinya.

2.7.4.2 *Resource behavior*

Kemampuan perangkat lunak dalam menggunakan sumber daya yang dimilikinya ketika melakukan fungsi yang ditentukan.

2.7.5 *Maintainability*

Kemampuan perangkat lunak untuk dimodifikasi. Modifikasi meliputi koreksi, perbaikan atau adaptasi terhadap perubahan lingkungan, persyaratan, dan spesifikasi fungsional.

2.7.5.1 *Analyzability*

Kemampuan perangkat lunak dalam mendiagnosis kekurangan atau penyebab kegagalan.

2.7.5.2 *Changeability*

Kemampuan perangkat lunak untuk dimodifikasi tertentu.

2.7.5.3 *Stability*

Kemampuan perangkat lunak untuk meminimalkan efek tak terduga dari modifikasi perangkat lunak.

2.7.5.4 *Testability*

Kemampuan perangkat lunak untuk dimodifikasi dan divalidasi perangkat lunak lain.

2.7.6 *Portability*

Kemampuan perangkat lunak untuk ditransfer dari satu lingkungan ke lingkungan lain.

2.7.6.1 *Adaptability*

Kemampuan perangkat lunak untuk diadaptasikan pada lingkungan yang berbeda-beda.

2.7.6.2 *Instalability*

Kemampuan perangkat lunak untuk diinstal dalam lingkungan yang berbeda-beda.

2.7.6.3 *Coexistence*

Kemampuan perangkat lunak untuk berdampingan dengan perangkat lunak lainnya dalam satu lingkungan dengan berbagi sumber daya.

2.7.6.4 *Replaceability*

Kemampuan perangkat lunak untuk digunakan sebagai sebagai pengganti perangkat lunak lainnya.