BAB II

LANDASAN TEORI

2.1 Penelitian Terdahulu

Penelitian terdahulu yang dilakukan oleh Adam Mukharil Bachtiar dan Arifin Bardansyah dengan judul "Pembangunan Class Library untuk Domain Product Management di Aplikasi M-Commerce pada Android" yang melakukan pembangunan *class library* untuk digunakan saat pengembangan aplikasi *M-Commerce* berbasis Android [3]. Penelitian tersebut menggunakan konsep *reuse* dalam pengembangan perangkat lunak untuk menggunakan ulang fungsional sistem. Selanjutnya adalah penelitian yang dilakukan oleh Mohammad Abdul Iman Syah dengan judul "Pembangunan Class Library Untuk Domain Aplikasi Informasi Penjualan Dan Penyewaan Properti Pada Platform Android" yang juga memanfaatkan konsep *reuse* [4].

Penelitian ini memiliki persamaan dengan penelitian terdahulu, yaitu membangun solusi yang *reusable* untuk digunakan saat pengembangan perangkat lunak. Adapun perbedaan dalam penelitian ini dengan penelitian terdahulu adalah penelitan ini berfokus pada pembangunan *code library* untuk pengembangan perangkat lunak berbasis web, sedangkan penelitan terdahulu berfokus untuk aplikasi berbasis Android.

2.2 Design System

Design System adalah seperangkat pola yang saling berhubungan dan *shared practices* yang diatur secara koheren untuk mempermudah proses mendesain dan mengembangkan produk digital [1]. *Design System* terdiri dari *style*

guide, pattern library dan *code library* yang menjadi standar untuk desainer dan *developer* dalam mengembangkan perangkat lunak. Dengan menggunakan *Design System,* perangkat lunak yang dibangun akan memakan waktu lebih cepat dan memiliki identitas yang konsisten karena menggunakan *reusable solution* yang sudah didefinisikan sebagai sebuah standar.

Berikut ini adalah penjelasan *Style Guide dan Pattern Library*.

1. *Style Guide*

   Style guide adalah seperangkat standar untuk penulisan dan desain dokumen. *Style guide* biasanya berisi panduan penggunaaan *font*, warna, dan tata letak halaman. Penggunaan *style guide* memberikan keseragaman dalam penerapan elemen grafis dan *formatting* [5].

2. *Pattern Library*

   Pattern Library adalah alat untuk menangkap, mengumpulkan, dan membagikan desain *pattern* dan *guidelines* untuk penggunaannya. Berbeda dengan *style guide* yang berfokus *style* seperti tipografi dan warna, *pattern library* mencakup lebih kumpulan pola yang lebih luas [1].

2.3 Code Library

Seperti yang dijelaskan sebelumnya bahwa *design system* terdiri dari beberapa bagian yang salah satunya adalah *code library. Code library* adalah kumpulan kode yang telah ditulis sebelumnya yang dapat digunakan oleh pengguna untuk mengoptimalkan tugas yang dikerjakan [6]. Pada saat membangun perangkat lunak berbasis web, *code library* untuk *design system* berisi *reusable component* yang merupakan kombinasi dari HTML, CSS, dan JavaScript. Detail implementasi

dari HTML, CSS dan JavaScript akan diabstraksi dan dapat digunakan menggunakan *Application Programming Interface* (API) dari *component* yang sudah didefinisikan.

Pengelolaan *code library* dilakukan secara terpusat, sehingga bisa membuatnya konsisten dan *reusable* di banyak perangkat lunak yang menggunakannya. *Code library* dapat menghemat waktu, biaya, dan dapat meningkatkan keandalan perangkat lunak yang dihasilkan. Manfaat paling besar yang didapatkan dengan menggunakan *code library* adalah dapat mengurangi jumlah waktu dan energi yang dibutuhkan *developer* pada saat membangun perangkat lunak [7].

2.4 Application Programming Interface

Application Programming Interface (API) adalah spesifikasi interaksi yang bisa digunakan untuk berinteraksi dengan komponen perangkat lunak. Jadi API merupakan perantara yang memungkinkan 2 perangkat lunak yang berbeda bisa berkomunikasi satu sama lain [8]. Agar API mudah digunakan dan dipelihara, maka *developer* dan *designer* API harus memiliki pemahaman yang baik mengenai *usability* dari API. Minhaz Zibran mendefinisikan 22 faktor [9] yang mempengaruhi *usability* API yaitu sebagai berikut:

1. *Complexity*

   Peningkatan ukuran dan *complexity* fitur, konsep, dan arsitektur yang terekspos akan mengurangi *usability* dari API.

2. *Naming*

   Kaidah dalam penamaan untuk fungsi atau variabel, nama yang deskriptif lebih disarankan daripada singkatan.

3. *Caller's perspective*

   Bagaimana pengguna dapat memanggil fungsi atau fitur harus jelas dan intuitif agar mudah digunakan.

4. *Documentation*

   Dokumentasi yang lengkap, jelas, dan terkini serta contoh penggunaan meningkatkan *usability.*

5. *Consistency*

   Konsistensi dalam desain mengikuti kaidah yang umum meningkatkan *usability.*

6. *Conceptual correctness*

   Ketepatan konseptual dalam desain dan penamaan fitur penting untuk *usability.*

7. *Parameter and return*

   Jumlah dan tipe parameter dari fungsi dan tipe nilai yang dikembalikan mempengaruhi *usability.* Jumlah parameter yang terlalu banyak akan menurunkan tingkat *usability.*

8. *Constructor parameter*

   Constructor tanpa parameter akan lebih mudah digunakan daripada yang memiliki parameter, terutama untuk programmer pemula dan menengah.

9. *Factory pattern vs constructor*

   Programmer secara alami mengharapkan konstruktor untuk membuat *instance* objek, bukan *factory methods*. Membuat *instance* objek melalui *factory methods* terkadang menimbulkan kesulitan.

10. *Data types*

    Tipe data harus dipilih dengan benar untuk menghindari *type-casting* yang tidak perlu, konsumsi sumber daya, dan hilangnya presisi.

11. *Use of attributes*

    Dispersi dan ketergantungan fungsional atribut. Implementasi fungsionalitas yang kohesif dapat meningkatkan kegunaan.

12. *Concurrency*

    Implementasi yang tepat dari konkurensi dan eksposur elemen yang bisa berubah. Eksposur yang tidak perlu dari elemen yang bisa berubah dapat meningkatkan masalah keamanan thread dan meningkatkan perangkap penyalahgunaan.

13. *Error handling*

    Mekanisme pencegahan kesalahan dengan penyembunyian informasi, serta penanganan kondisi kesalahan yang tepat melalui informasi diagnosis dan mekanisme pemulihan.

14. *Leftovers for client*

    Ketersediaan implementasi yang dibutuhkan pengguna dapat mengurangi *overhead* pengguna.

15. *Multiple ways to do one thing*

    Ketersediaan berbagai cara (misalnya, beberapa metode yang menawarkan fungsionalitas yang sama) untuk melakukan hal yang sama dapat membingungkan pengguna dalam memilih dari alternatif.

16. *Reference chain*

    Rantai panjang panggilan metode atau hierarki pewarisan sulit dilacak, dan mengurangi *usability*.

17. *Implementation vs interface dependency*

    Dependensi *interface* antar komponen memberikan lebih banyak fleksibilitas dan karenanya direkomendasikan daripada dependensi implementasi.

18. *Memory management*

    Manajemen memori (alokasi dan dealokasi memori) yang diserahkan kepada pengguna mengurangi *usability* dari API.

19. *Technical mismatch*

    Kompatibilitas dengan platform dan teknologi lain di lingkungan fungsional penting untuk *usability*.

20. *API change*

    Backward compatibility diperlukan untuk kegunaan, sementara penghentian fitur umum dapat mengejutkan pengguna.

21. *API aging*

    API aging terjadi ketika platform target berubah tetapi API gagal mengimbangi evolusi platform, dan akibatnya menjadi API yang tidak dapat digunakan.

22. *Code intelligibility*

    Readability dari *client code* mempengaruhi pemeliharaan.

2.5 Framework

Framework adalah suatu kerangka kerja yang digunakan untuk mengimplementasi standar yang sudah ditetapkan untuk suatu *environment* tertentu [7]. *Design system* yang dibangun dan memiliki *code library* didalamnya akan menjadi *framework* yang nantinya akan digunakan oleh *developer* untuk mengembangkan perangkat lunak.

2.6 User Interface

Agar perangkat lunak mudah digunakan oleh pengguna. Maka dibutuhkan *User Interface* yang menjadi media komunikasi interaktif antara pengguna dan perangkat lunak [10]. Jika *User Interface* yang dibangun sulit untuk digunakan, hal ini akan membuat pengguna kesulitan dan melakukan kesalahan saat menggunakannya, sehingga mempersulit pengguna mencapai tujuannya [10]. Jadi sangat penting untuk membangun *User Interface* perangkat lunak yang konsisten dan mudah digunakan.

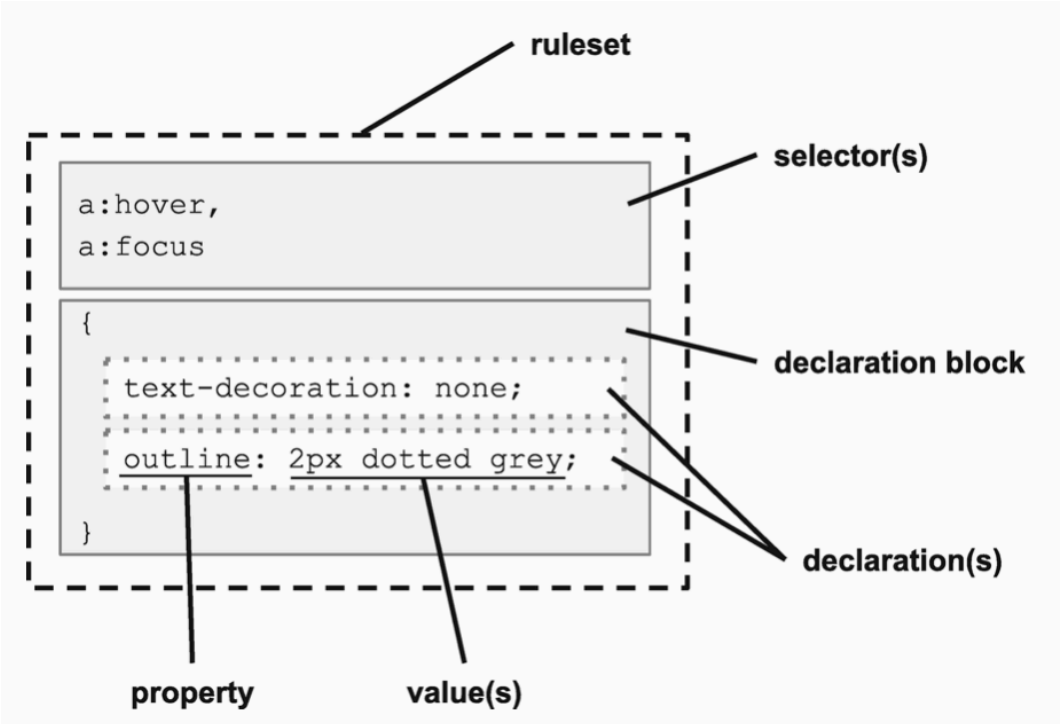
2.7 Web

Menurut Rohi Abdulloh (2015) web adalah: “Sekumpulan halaman yang terdiri dari beberapa halaman yang berisi informasi dalam bentuk data digital baik

berupa teks, gambar, video, audio, dan animasi lainnya yang disediakan melalui jalur koneksi internet" [11].

2.8 CSS (Cascading Style Sheets)

CSS adalah teknologi web yang memungkinkan tata letak, tema dan *style* diterapkan pada dokumen web. Dalam kebanyakan kasus, dokumen yang dimaksud adalah file *Hypertext Markup Language* (HTML) dan proses penerjemahannya dilakukan oleh web *browser.* CSS merupakan bahasa yang deklaratif dan bukan imperatif, yang berarti alih-alih memberi tahu web *browser* cara untuk menerapkan menerapkan *styles* ke halaman, dengan CSS yang dilakuakan adalah memberi tahu web *browser styles* apa yang harus diterapkan dan dimana menerapkannya. Hal tersebut bisa dilakukan dengan menggunakan deklarasi yang disebut *rules yang* bisa dilihat pada **Gambar 2.1.**



Gambar 2.1 CSS *Ruleset*

(Sumber: Architecting CSS: The Programmer's Guide to Effective Style Sheets [12, p. 21])

Setiap *declaration* terdiri dari kombinasi dari *property* dan *value.* Pada saat penelitian ini ditulis, CSS memiliki lebih dari 563 *properties.* Setiap *property* harus didukung oleh *user agent* (biasanya *web browser*) agar bisa di implementasikan [12]. Untuk menentukan elemen apa yang akan menerapkan *property* dari CSS, maka dibuthkan *selector*. Berikut ini adalah beberapa jenis *selector* di CSS.

1. *Universal Selector*

   Karakter "*" dalam CSS adalah *universal selector* yang akan berdampak ke semua elemen di dalam halaman. Contoh penggunaannya adalah sebagai berikut:

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

2. *Type Selector*

   Type selector adalah penggunaan *tag* HTML sebagai *selector* di CSS. Semua *tag* di HTML adalah *selector* yang valid. Berikut ini adalah contoh penggunaan *type selector* pada tag *paragraph*:

```
p {
    color: #212121;
    line-height: 1.5;
}
```

3. *Class Selector*

   Untuk memilih elemen menggunakan *class* yang diterapkan, bisa dengan menggunakan tanda titik diikuti dengan nama *class* yang dimaksud. Satu *tag* HTML bisa memiliki lebih dari 1 *class*. Berikut ini contoh penggunaan *class selector*:

```
<style>
    .card {
        background-color: #ccc;
        border-radius: 4px;
        padding: 16px;
    }
    .card-image {
        width: 100%;
    }
    .button {
        background-color: skyblue;
    }
    .button-secondary {
        background-color: slategray;
    }
</style>

<div class="card">
    <h2>Test</h2>
    <img src="" class="card-image">
    <button class="button">Lihat</button>
    <button class="button button-secondary">Hapus</button>
</div>
```

4. *ID Selector*

   Berbeda dengan *class selector* yang bisa digunakan lebih dari 1 pada suatu elemen, ID merupakan sebuah *identifier* yang hanya boleh ada 1 dalam suatu halaman dan tidak boleh ada duplikasi. Contoh penggunaan ID adalah sebagai berikut:

```
#header {
    background-color: #3399ff;
}
```

5. *Attribute Selector*

   Attribute selector berguna untuk menargetkan elemen berdasarkan *attribute* yang dimilikinya. *Selector* ini menggunakan kurung sikut yang berisi *attribute* yang sesuai dimiliki elemen yang ditargetkan. Berikut ini adalah contoh penggunaan *attribute selector*:

```
<style>
    input[type="password"] {
        color: blue;
    }
</style>

<input type="password">
```

2.9 SASS (*Syntactically Awesome Style Sheets*)

SASS adalah CSS *Preprocessor* sumber terbuka yang mudah digunakan yang membantu mengurangi tantangan pengulangan dan pemeliharaan CSS tradisional. SASS memungkinkan untuk menskalakan *styles* saat mengerjakan proyek pengembangan web besar, membuatnya lebih cepat dan efisien untuk menulis gaya yang dapat digunakan kembali dari awal untuk proyek yang lebih kecil. SASS akan dikompilasi menjadi Cascading Style Sheets (CSS) sehingga bisa dibaca oleh *browser* [13].

Berikut ini adalah fitur-fitur yang dimiliki oleh SASS:

1. *Variables*
2. *Nesting*

3. *Partials*
4. *Modules*
5. *Mixins*
6. *Extend/Inheritance*
7. *Operators*

2.10 ITCSS (*Inverted Triangle* CSS)

Ada beberapa pendekatan yang bisa digunakan untuk mengatur arsitektur CSS, beberapa diantaranya yang cukup sering digunakan adalah sebagai berikut:

Tabel 2.1 CSS Metodologi

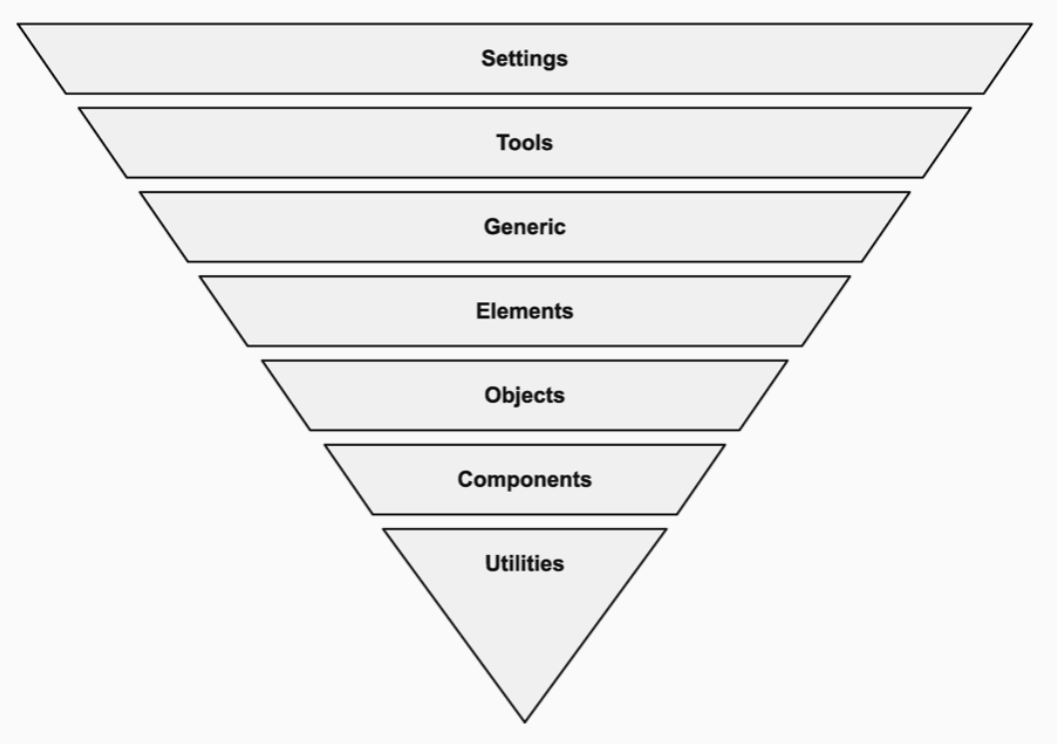
Tahun	Pencipta	Metodologi
2008	Nicole Sullivan	OOCSS
2009	Yandex	BEM
2012	Jonathan Snook	SMACSS
2015	Harry Roberts	ITCSS

OOCSS memiliki prinsip untuk memisahkan *structure* dari *skin* dan memisahkan struktur dari konten. SMACSS membagi aturan kedalam 5 kategori yaitu *base, layout, module, state,* dan *theme.* ITCSS membagi *style sheets* kedalam 7 *layers* yaitu *settings, tools, generic, elements, objects, components* dan *utilities.* Dan BEM membagi komponen kedalam 3 bagian yaitu *block, element,* dan *modifier* [12].

Keempat metodologi tersebut memiliki kesamaan yaitu penggunaan *class* sebagai *selector* di dalam HTML dan CSS, dan penggunaan *class* di dalam HTML

hanya digunakan untuk CSS dan JavaScript *binding.* Banyak profesional CSS yang mencampur dan mengkombinasikan beberapa metodologi tersebut untuk menyesuaikan dengan kebutuhan, dan hal ini bisa dilakukan selama konsisten dengan metodologi yang dipilih [12].

Penelitian ini menggunakan ITCSS yang dikembangkan oleh Harry Roberts dengan tujuan untuk mengatasi *selector specificity* memanfaatkan prioritas alami dari CSS karena sesuai untuk kebutuhan pembangunan berdasarkan *style guide* dan *pattern library* yang ada dalam *design system* saat ini. Dengan ITCSS, *style sheets* dibagi kedalam beberapa *layer* berdasarkan tujuannya.



Gambar 2.2 Diagram ITCSS

(Sumber: Architecting CSS: The Programmer's Guide to Effective Style Sheets [12, p. 403])

Dasar dari segitiga (di bagian atas diagram) mewakili aturan terluas dan paling tidak spesifik, dan bagian paling bawah dari diagram merupakan bagian yang memiliki aturan paling eksplisit dan *specificity* paling tinggi. Berikut ini adalah penjelasan lebih rinci untuk setiap *layer*:

1. *Settings*

   Settings dapat digunakan dengan memanfaatkan *preprocessors, layer* ini digunakan untuk mendefinisikan variabel *preprocessors* seperti warna dan *font.* Hasil dari *layer* ini tidak akan dihasilkan menjadi CSS apapun.

2. *Tools*

   Layer ini berisi *mixins* ataupun fungsi untuk penggunaan kembali lebih lanjut. Sama seperti *layer settings, layer* ini juga dapat digunakan dengan memanfaatkan *preprocessors* dan tidak akan dihasilkan menjadi CSS apapun.

3. *Generic*

   Layer ini berisi *normalization* dan *reset styles* bawaan CSS dari *browser* dan variabel CSS. Bagian ini memang diperuntukan untuk memiliki jangkauan yang luas dan tidak terlalu spesifik.

4. *Elements*

   Style bawaan untuk standar HTML elemen.

5. *Objects*

   Layer ini merupakan lapisan pertama yang menggunakan *class-based selector* dan diperuntukan untuk penggunaan *object* seperti pembangunan *layout.*

6. *Components*

   Layer ini merupakan lapisan yang paling besar daripada lapisan yang lain. Di sini berisi *styles* untuk semua komponen beserta dengan variannya.

7. *Utilities*

   Berisi *helper* class yang mungkin dibutuhkan untuk menimpali *styles* apapun yang sudah didefinisikan pada *layer* sebelumnya. Hal itu lah yang menyebabkan *layer* ini diperbolehkan untuk memiliki *specificity* yang tinggi dan penggunaan properti *!important* dari CSS.

2.11 BEM (*Block Element Modifier*)

Berbeda dengan ITCSS yang memiliki fokus untuk mengatur struktur dari CSS, BEM adalah suatu metodologi yang memiliki tujuan untuk mengatur penamaan dari suatu komponen yang dibangun [14]. BEM merupakan singkatan dari *Block, Element*, dan *Modifier*. Penamaan dengan BEM akan mengikuti pola *Block__Element—Modifier.* Penelitian ini menggunakan BEM sebagai *naming convetion* pada *layer components* didalam ITCSS.

Berikut ini adalah penjelasan dari setiap bagian tersebut:

1. *Block*

   Block di dalam BEM mengacu pada konsep *reusable component*, dengan kata lain *Block* akan diisi dengan nama suatu komponen.

2. *Element*

   Element merupakan bagian dari *Block.* Meskipun sama-sama berguna untuk mengatur suatu *item* di HTML, perbedaan dari *Element* dan *Block* adalah

Element tidak bisa berdiri sendiri dan bergantung kepada *Block* yang menjadi *parent*-nya. Sedangkan *Block* tidak bergantung kepada apapun.

3. *Modifier*

   Modifier berguna sebagai pengubah untuk *Block* ataupun *Element.*

Contoh Penggunaan BEM

```
<div class="toggle toggle--simple">
    <div class="toggle__control toggle__control--active">
        <h2 class="toggle__title">Title 1</h2>
    </div>

    <div class="toggle__details toggle__details--active">
        ...
    </div>
    ...
</div>
```

2.12 Usability Metrics

Usability metrics adalah standar untuk mengukur performa pengguna dalam mengerjakan serangkaian tugas tertentu [15]. Tugas yang diuji adalah skenario penggunaan dari sistem yang bisa dikerjakan oleh partisipan. Salah satu *usability metrics* yang bisa diukur adalah *success rate* [16]. *Success rate* adalah persentase tugas yang bisa diselesaikan pengguna dengan benar. Untuk mengukur *success rate* bisa menggunakan rumus berikut ini:

$$Success\ Rate = \frac{Total\ Success + (Total\ Partially\ Succes * 0{,}5)}{Total\ Tasks} \times 100$$

Pada penelitian ini pengujian dilakukan dengan memberikan 4 tugas kepada 5 *developer* karena menurut penelitian Jakob Nielsen dan Tom Landauer hasil terbaik dari pengujian bisa didapatkan dengan menguji ke tidak lebih dari 5 pengguna [17]. Selain dari *success rate,* dilakukan juga pengukuran efisiensi waktu dengan membandingkan waktu yang dihabiskan ketika menggunakan *code library* dan tanpa *code library* saat menyelesaikan tugas.