BAB II

TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Karya tulis ini dibuat dengan mengacu pada karya tulis yang telah dibuat, dijabarkan pada Tabel 2.1:

Tabel 2.1. Perbandingan Penelitian

Parameter / Penulis	Judul Penelitian	Bahasa Pemrograman	Platform
Paul Christersson Frend (2016)	*Unit testing the user interface strategy for automatic testing of web UI's in React*	Javascript	Web
Thomas Nagele (2015)	*Client-side performance profiling of JavaScript for web applications*	Javascript	Web
Muhammad Aris Ganiardi (2014)	*jQuery* sebagai komponen usabilitas antarmuka aplikasi web	Javascript	Web
Alexander, Svensson (2014)	*Speed Performance Comparison of JavaScript MVC Frameworks*	Javascript	Web
Nisha Gogna (2014)	*Study of Browser Based Automated Test Tools WATIR and Selenium*	Javascript	Web

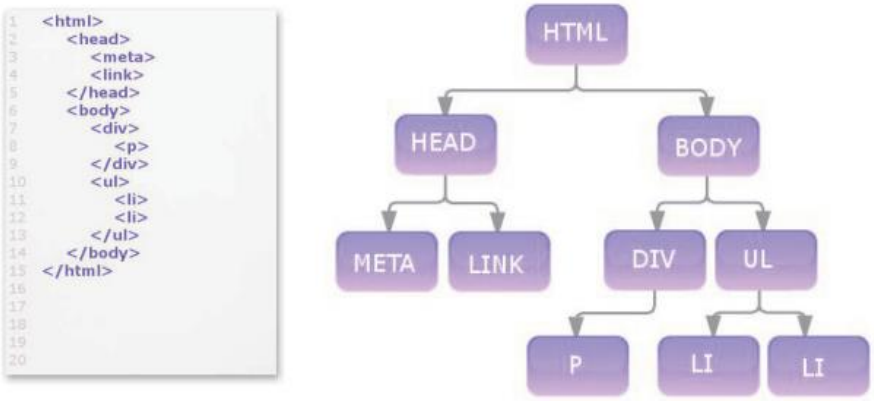
diusulkan: Cahyo Wibowo Widianarko	Studi perbandingan website dengan view framework (vue js) dengan *vanilla js*.	Javascript	Web

Pada tabel baris 4 diatas, pengujian di spesifikasikan menggunakan berbagai macam framework javascript antara lain Angular, Backbone, Aurelia, Ember, Knockout dan Vue tanpa membahas perbandinganya dengan vanila javascript dan jQuery. Kemudian pada baris 5, materi penelitannya ialah automasi dengan menggunakan Selenium, sementara pada penelitian kali ini automasi dilakukan dengan menggunakan puppeteer untuk melakukan automasi pada website sekaligus mengukur performa dengan menggunakan chrome *performance tools.*

2.2 Dasar Teori

2.2.1 DOM

DOM / *Document Object Model* adalah antarmuka programming untuk HTML, XML dan SVG dokumen. DOM menyediakan struktur dokumen yang berorientasi objek dan setiap elemen yang menyusun DOM memiliki property dan nilai nya masing masing. Saat pertama kali web browser mengunjungi sebuah halaman, web browser akan memparsing text tersebut menjadi DOM.



Gambar 2.1 Contoh HTML dokumen dikonversi ke Document Object Model

Gambar diatas merupakan sebuah pohon DOM dan dari gambar diatas dapat dilihat bahwa objek HTML / tag HTML membungkus dua *element* yaitu head dan body. Gambar bagian kanan diatas merupakan contoh dari penyederhanaan DOM *interface* dari script pada gambar bagian kiri.

Tiap node pada pohon DOM merepresentasikan element atau atribute dan dengan menggunakan DOM *API* kita dapat menambah, menghapus atau mengupdate node pada pohon DOM tersebut. Setiap perubahan yang dilakukan oleh DOM akan membuat kerja web browser semakin tinggi dan resource atau memory komputer semakin menurun. Setiap perubahan DOM satu dengan yang lainya tentu membutuhkan waktu yang berbeda beda.

Sampai pada hari ini umumnya sebuah halaman web menggunakan javascript untuk memanipulasi DOM. Semakin banyak sebuah halaman website melakukan manipulasi pada DOM, maka semakin tinggi *resource* yang

dibutuhkan oleh web browser untuk menampilkan kembali hasil perubahan dari DOM yang dimanipulasi.

2.2.2 CSS Object Model (CSSOM)

Saat web browser mengkonstruksi DOM, web browser juga akan melakukan pencarian element link (tag link, biasanya terdapat pada head) yang akan mereferensikan external css. Jika tidak ditemukan css maka web browser akan menggunakan *stylesheet* standar yang tersedia pada web browser masing masing.

Setelah *stylesheet* dimuat pada sebuah halaman, web browser akan mengubah *rule* yang tersedia pada *stylesheet* tersebut dan menjadikanya struktur seperti DOM namun hanya untuk css. Pada saat ini web browser akan mengkalkulasi *rule* dari *stylesheet* tersebut untuk di kalkulasi dan di aplikasikan ke DOM.

2.2.3 Pohon Rendering

Pohon rendering adalah gabungan dari DOM dan CSSOM yang berisi segala sesuatu yang akan ditampilkan pada halaman web. Pohon rendering memberi tahu web browser bagaimana tata letak halaman dan cara melukisnya.

Jika kita melakukan sesuatu di halaman web seperti menggulir halaman, mengubah DOM dengan JavaScript, animasi CSS atau pada dasarnya apa pun yang kita lakukan pada halaman web pohon Rendering akan diperbarui sekitar

enam puluh kali setiap detik. Waktu render setiap frame pada setiap browser dapat berbeda beda.

2.2.4 Perenderan Web Browser

Perenderan browser web terjadi setiap kali kita berinteraksi dengan situs web dan setiap web browser memiliki mesin sendiri untuk menghitung setiap frame yang direnderer.

Mesin browser web (terkadang disebut layout engine atau rendering engine) sangat kompleks dan terminologi yang digunakan oleh setiap web browser dapat berbeda beda tetapi dapat diartikan dengan hal yang sama. Misal web browser seperti google chrome dan safari menggunkan WebKit, Mozilla firefox dan sea monkey menggunakan gecko dll. Semua web browser rata rata berusaha untuk dapat melakukan *frame rendering* hingga kecepatan 60fps agar dapat terlihat halus oleh para pengguna web.

Perenderan browser melakukan langkah langkah yang berbeda seperti mengatur tata letak dan mengaplikasikan *stylesheet* pada DOM agar halaman web dapat dimuat lebih cepat. Misal kita akan melihat text terlebih dahulu pada halaman web sebelum gambar ditampilkan.

2.2.5 Performa rendering

Setiap perangkat seperti smartphone dan komputer rata rata memiliki *refresh rate* hingga 60fps dimana web browser juga akan melakukan hal animasi transisi misal dengan css, *page scrolling* dll. Setiap frame yang direnderer setidaknya harus selesai dalam waktu 16ms (1 detik / 60). Namun browser membutuhkan waktu 6 ms untuk berjaga jaga sehingga browser hanya memiliki waktu kurang lebih 10 ms untuk setiap frame. Jika waktunya lebih dari itu maka tampilan website akan menjadi tidak halus / *laggy* dan tentu saja dapat menurunkan kenyamanan user yang sedang melakukan browsing.

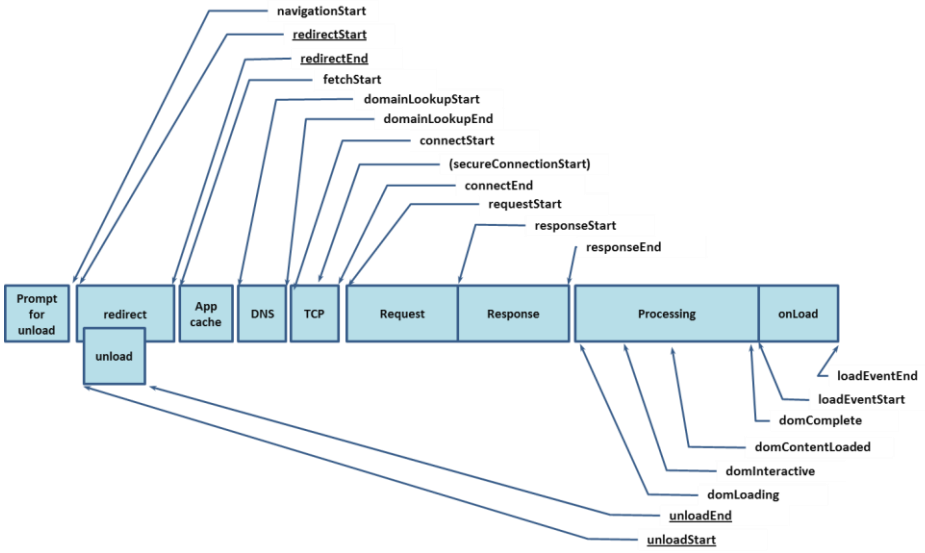
Setiap kali kita berinteraksi dengan halaman web seperti menggulir halaman, animasi css, mengeksekusi javascript yang memanipulasi DOM, web browser akan berusaha menyelesaikan perubahan yang terjadi pada halaman web dalam waktu 10ms per frame nya. Jadi intinya setiap kali kita melakukan "perubahan visual" pada halaman website, frame baru akan dibangun.

2.2.6 Navigation Timing API

Navigation timing API menyediakan data yang dapat digunakan untuk mengukur kinerja website. Output yang dihasilkan sama seperti saat kita menjalankan perintah window.performance.timing pada console browser.

Navigation timing API adalah standar pengukuran agnostik browser yang dikelola oleh W3C dan umumnya tersedia pada seluruh web browser.

Navigation timing API memiliki banyak *properties* yang berasal dari pengukuran dengan menggunakan window.performance.timing.



Gambar 2.2 Macam *properties* window.performance.timing

2.2.7 Vue js

Vue js ialah *progressive framework* yang digunakan untuk membangun tampilan halaman website. Berbeda dengan framework lainya yang menggunakan prinsip MVC (*Model View Controller*) Vue js hanya difokuskan untuk membangun tampilan / hanya bekerja pada *view layer*. Dalam hal lainnya Vue js juga dapat membantu programmer untuk membuat web dengan aplikasi single page yang canggih dan dapat di kombinasikan dengan library browser lainnya.

Selain itu Vue js menggunakan template sintaks bebrbasis HTML yang memungkinkan pengguna untuk mendeklarasikan data / state kedalam DOM. Semua template Vue js adalah HTML yang valid yang dapat diuraikan oleh browser sesuai spesifikasi dan parser HTML.

2.2.8 Chrome DevTools

Chrome DevTools adalah seperangkat alat untuk *developer web* yang terdapat pada browser google chrome. Chrome DevTools dapat membantu *developer* untuk mendiagnosis masalah yang ada pada sebuah website dengan cepat dan dapat membantu *developer* untuk membangun situs web yang lebih baik. Beberapa fungsi yang dapat dilakukan oleh Chrome DevTools antara lain.

1. Melihat dan mengganti *stylesheet* halaman website
2. *Debugging* Javascript
3. Melihat pesan / notifikasi dan menjalankan Javascript pada console.log
4. Menganalisa Performa website saat halaman sedang ditampilkan

2.2.9 Puppeteer

Puppeteer adalah sebuah *library* node yang menyediakan fungsi *API* untuk dapat mengontrol browser *chromium* lewat *DevTools Protocol*. Selain itu *Pupeteer* dapat juga di konfigurasikan untuk menjalankan mode *non-headless.* Berikut ini kegunaan dari Puppeteer*:*

1. Melakukan *Screenshoot* & membuat PDF berdasarkan tampilan pada website
2. Melakukan *crawling* pada *SPA* website untuk menghasilkan *Prerendered content*

3. Melakukan *Scraping* pada sebuah konten website

4. Melakukan otomasi pada element website, melakukan *UI testing* dll

5. Merekam *Runtime Performance Analysis* untuk membantu diagnosa *issue* pada browser.

2.2.10 First meaningful paint

First Meaningfull paint adalah waktu yang dibutuhkan browser untuk "melukis" konten utama yang muncul pada halaman website. Definisi konten utama pada halaman website dapat berbeda beda, misal untuk artikel konten utama itu akan menjadi judul, teks konten (teks harus explisit terlihat, tidak menunggu waktu *loading* font), sementara komponen seperti spinner, icon dan komponen tersier lainya tidak dihitung.