SKRIPSI

MENINGKATKAN KEAMANAN APLIKASI WEB DENGAN MELAKUKAN PENYIMPANAN JSON WEB TOKEN PADA VARIABEL LOKAL

Disusun dan diajukan oleh:

MUHAMMAD BISHRAM YASHIR ALFARIZI AMINUDDIN
D121 17 1506

PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
GOWA
2023

LEMBAR PENGESAHAN SKRIPSI

MENINGKATKAN KEAMANAN APLIKASI WEB DENGAN MELAKUKAN PENYIMPANAN JSON WEB TOKEN PADA VARIABEL LOKAL

Disusun dan diajukan oleh

MUHAMMAD BISHRAM YASHIR ALFARIZI AMINUDDIN
D121171506

Telah dipertahankan di hadapan Panitia Ujian yang dibentuk dalam rangka Penyelesaian Studi Program Sarjana Program Studi Teknik Informatika Fakultas Teknik Universitas Hasanuddin pada tanggal 29 November 2023 dan dinyatakan telah memenuhi syarat kelulusan.

Menyetujui,

Pembimbing Utama,	Pembimbing Pendamping,
Dr.Eng. Ady Wahyudi Paundu, ST.,MT.	Iqra' Aswad, S.T., M.T
Nip. 197503132009121003	Nip. 199011282019043001

Ketua Program Studi,

Prof. Dr. Ir. Indrabayu, S.T., M.T., M.Bus.Sys., IPM, ASEAN. Eng.
Nip. 19750716 200212 1 004

PERNYATAAN KEASLIAN

Yang bertanda tangan dibawah ini ;

Nama	:	Muhammad Bishram Yashir Alfarizi Aminuddin
NIM	:	D121171506
Program Studi	:	Teknik Informatika
Jenjang	:	S1

Menyatakan dengan ini bahwa karya tulisan saya berjudul

Meningkatkan Keamanan Aplikasi Web dengan Melakukan Penyimpanan JSON Web Token pada Variabel Lokal

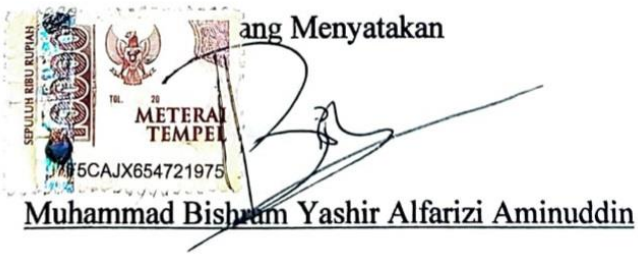
Adalah karya tulisan saya sendiri dan bukan merupakan pengambilan alihan tulisan orang lain dan bahwa skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri.

Semua informasi yang ditulis dalam skripsi yang berasal dari penulis lain telah diberi penghargaan, yakni dengan mengutip sumber dan tahun penerbitannya. Oleh karena itu semua tulisan dalam skripsi ini sepenuhnya menjadi tanggung jawab penulis. Apabila ada pihak manapun yang merasa ada kesamaan judul dan atau hasil temuan dalam skripsi ini, maka penulis siap untuk diklarifikasi dan mempertanggungjawabkan segala resiko.

Segala data dan informasi yang diperoleh selama proses pembuatan skripsi, yang akan dipublikasi oleh Penulis di masa depan harus mendapat persetujuan dari Dosen Pembimbing.

Apabila dikemudian hari terbukti atau dapat dibuktikan bahwa sebagian atau keseluruhan isi skripsi ini hasil karya orang lain, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Gowa, 29 November 2023



ABSTRAK

MUHAMMAD BISHRAM YASHIR ALFARIZI AMINUDDIN. *Meningkatkan Keamanan Aplikasi Web dengan Melakukan Penyimpanan JSON Web Token pada Variabel Lokal* (dibimbing oleh Dr. Eng. Ady Wahyudi Paundu, S.T., M.T. dan Iqra Aswad, S.T., M.T.)

JSON *Web Token* (JWT) merupakan sebuah JSON *Object* berbentuk *string* yang digunakan untuk transmisi informasi antar dua atau lebih pihak secara *secure*. Di rana pemograman *web*, JWT sering di gunakan untuk otorisasi pengguna. Agar dapat digunakan kembali, JWT ini biasanya disimpan di *local storage* atau *cookie* karena kemudahan dalam implementasinya. Walaupun mudah digunakan, ketika *website* terkena serangan *Cross-Site Scripting* (XSS) atau *Cross-Site Request Forgery* (CSRF), data yang disimpan di penyimpanan ini dapat ditemukan dan digunakan untuk melakukan aksi yang pemilik akun tidak inginkan. Sebagai solusi, *local variable* dapat digunakan. Namun penyimpanan ini tidak sempurna, ketika pengguna *website* membuka sesi baru, data yang disimpan akan hilang. Maka dari itu, penelitian ini akan mencoba untuk membuat penyimpanan *local variable* yang dapat tetap tersimpan ketika pengguna membuka sesi baru. Untuk mencapai hal tersebut, penulis memanfaatkan *local storage* sebagai tempat penyimpanan data sementara, ketika pengguna kembali membuka sesi di *website*, data dari *local storage* akan dikembalikan ke *local variable*. Agar aman dari serangan XSS, perpindahan data ini dilakukan sebelum *element* pada *website* ditampilkan. Jika dibandingkan penyimpanan *local storage* atau *cookie*, data yang disimpan di *local variable* akan tetap aman ketika terjadi serangan XSS atau CSRF. Untuk perbandingan kinerja implementasi penyimpanan, *local variable* lebih lambat jika dibandingkan dengan penyimpanan *local storage* dan *cookie*. Sedangkan untuk performa dari penyimpanan sendiri, *local variable* merupakan penyimpanan yang tercepat, diikuti dengan *local storage*, dan yang terakhir *cookie*.

Kata Kunci: *Local Variable*, *Local Storage*, *Cookie*, JWT, XSS, CSRF

ABSTRACT

MUHAMMAD BISHRAM YASHIR ALFARIZI AMINUDDIN. *Improving Web Application Security by Storing JSON Web Token in Local Variable* (supervised by Dr. Eng. Ady Wahyudi Paundu, S.T., M.T. and Iqra Aswad, S.T., M.T.)

JSON Web Token (JWT) is a JSON Object in string format that is used to transmit information securely between two or more parties. In web programming, JWT is often used for user authorization. For it to be reusable, JWT is typically stored in local storage or cookies due to its ease of implementation. Although it's easy to use, when a website is attacked with Cross-Site Scripting (XSS) or Cross-Site Request Forgery (CSRF), data that is stored in this storage is exposed and can be used for nefarious intent. As a solution, local variables can be used. However, this solution is not perfect, when a user opens a new session, the stored data will be lost. That's why this research will aim to create a local variable storage that can persist data even when a user opens a new session. To achieve this, the author utilizes local storage as temporary storage. When the user reopens a session, data from local storage will be restored to local variable. To make it safe from XSS, this data restoration is performed before any elements is rendered on a website. Compared to local storage or cookie, data stored in local variable will remain secure in the event of an XSS or CSRF attack. For performance comparison of storage implementation, local variable is slower when compared to local storage and cookie. Meanwhile, in terms of storage performance, local variables are the fastest, followed by local storage, and lastly, cookie.

Keywords: Local Variable, Local Storage, Cookie, JWT, XSS, CSRF

DAFTAR ISI

LEMBAR PENGESAHAN SKRIPSI i
PERNYATAAN KEASLIAN ii
ABSTRAK iii
ABSTRACT iv
DAFTAR ISI v
DAFTAR GAMBAR vi
DAFTAR TABEL vii
DAFTAR SINGKATAN DAN ARTI SIMBOL viii
DAFTAR LAMPIRAN ix
KATA PENGANTAR x
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 3
1.3 Tujuan Penelitian/Perancangan 3
1.4 Manfaat Penelitian/Perancangan 3
1.5 Ruang Lingkup/Asumsi perancangan 4
BAB II TINJAUAN PUSTAKA 5
2.1 *Local Variable* 5
2.2 *Local Storage* 6
2.3 Cookie 8
2.4 *Cross-Site Scripting* 8
2.5 *Cross-Site Request Forgery* 9
2.6 *Same-Origin Policy* 11
2.7 JSON *Web Token* 11
2.8 *Session Credential* 12
2.9 ReactJS 15
2.10 *User Experience* 18
2.11 NextJS 19
2.12 MongoDB 19
BAB 3 METODE PENELITIAN/PERANCANGAN 20
3.1 Tahapan Penelitian 20
3.2 Waktu dan Lokasi Penelitian 21
3.3 Instrumen Penelitian 21
3.4 Perancangan Sistem 22
3.5 Pengujian Sistem 30
BAB 4. HASIL DAN PEMBAHASAN 40
4.1 Hasil Uji Coba Keamaman 40
4.2 Hasil Kinerja Implementasi Sistem 44
4.3 Hasil Kinerja Penyimpanan 46
BAB 5. KESIMPULAN DAN SARAN 49
5.1 Kesimpulan 49
5.2 Saran 49
DAFTAR PUSTAKA 51

DAFTAR GAMBAR

Gambar 1 Struktur *variable* JS 5
Gambar 2 Contoh serangan XSS (Sivanesan dkk., 2018) 9
Gambar 3 Ilustrasi terjadinya serangan CSRF (Siddiqui dan Verma, 2011) 10
Gambar 4 Contoh JWT (Peyrott, 2018) 12
Gambar 5 Cara mengambil *access token* baru menggunakan *refresh token* (Arias dan Bellen, 2021) 14
Gambar 6 Contoh kode JSX (Thanh, 2021) 15
Gambar 7 React *hook flow* diagram (West, 2019) 17
Gambar 8 Tahapan penelitian 20
Gambar 9 Lokasi penelitian Fakultas Teknik Universitas Hasanuddin 21
Gambar 10 *Flowchart* memuat ulang atau menutup aplikasi 22
Gambar 11 *Flowchart* membuka ulang aplikasi 23
Gambar 12 Skenario pengguna *login* membuka *tab* baru 25
Gambar 13 Ilustrasi cara kerja penyimpanan *local variable* pada *website* 26
Gambar 14 Cara kerja CSRF *token* 28
Gambar 15 Proses *request* tanpa CSRF *token* 29
Gambar 16 UI halaman serangan CSRF 29
Gambar 17 Halaman uji coba kecepatan simpan *local variable* 30
Gambar 18 Skrip menghapus semua *quotes* pengguna 31
Gambar 19 *Activity diagram* uji coba *reflected* XSS 32
Gambar 20 Hasil *submit* skrip *reflected* XSS 33
Gambar 21 *Activity diagram* uji coba *stored* XSS 34
Gambar 22 Hasil *submit* skrip *stored* XSS 35
Gambar 23 *Activity diagram* uji coba CSRF 36
Gambar 24 Hasil *submit* skrip CSRF 36
Gambar 25 *Component* serangan CSRF 37
Gambar 26 Form untuk melakukan uji coba keamanan 40
Gambar 27 *Quote* oleh pengguna “mbishram” 41
Gambar 28 Skrip serangan *reflected* XSS 42
Gambar 29 Halaman setelah pengguna menekan link pada skrip *reflected* XSS 42
Gambar 30 Skrip serangan *stored* XSS 43
Gambar 31. Skrip serangan CSRF 44
Gambar 32 Grafik perbandingan uji coba kinerja implementasi sistem 46
Gambar 33 Grafik perbandingan uji coba kinerja penyimpanan 48

DAFTAR TABEL

Tabel 1. Contoh aturan SOP (Saiedian dan Broyle, 2011) 11
Tabel 2. Batas ukuran data *header* HTTP *request* (Den, 2020)..... 12
Tabel 3. Perbandingan pengguna *guest* dan *user* 27
Tabel 4. *Endpoint website* quote.me 27
Tabel 5. Hasil uji coba keamanan sistem 40
Tabel 6. Hasil uji coba performa implementasi sistem 44
Tabel 7. Hasil uji coba performa penyimpanan 47

DAFTAR SINGKATAN DAN ARTI SIMBOL

Lambang/Singkatan	Arti dan Keterangan
JWT	JSON *Web Token*
CSRF	*Cross-Site Scripting*
XSS	*Cross-Site Request Forgery*
UX	*User Experience*
HMAC	*Hash-Based Message Authentication Codes*
JS	JavaScript
SOP	*Same-Origin Policy*
API	*Application Programming Interface*
SPA	*Single-Page Application*
UI	*User Interface*
DOM	*Document Object Model*
HTML	*HyperText Markup Language*
CSS	*Cascading Style Sheets*

DAFTAR LAMPIRAN

Lampiran 1 UI *website* uji coba 55
Lampiran 2 UI *website* skrip 57
Lampiran 3 Hasil form uji coba keamanan 59
Lampiran 4 Contoh hasil uji coba kinerja implementasi sistem 62
Lampiran 5 Contoh hasil uji coba kinerja penyimpanan 63
Lampiran 6 Halaman *package* npmjs persist-local-variable 64
Lampiran 7 Contoh kasus asli *stored* XSS tanpa dan menggunakan *local variable* 65
Lampiran 8 Contoh kasus asli *reflected* XSS tanpa dan menggunakan *local variable* 66
Lampiran 9 Contoh kasus asli CSRF tanpa dan menggunakan *local variable* 67
Lampiran 10 Lembar perbaikan skripsi 68

KATA PENGANTAR

Segala puji dan syukur di panjatkan ke hadirat Allah SWT, yang telah memberi rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan tugas akhir dengan judul "Meningkatkan Keamanan Aplikasi Web Dengan Melakukan Penyimpanan JSON Web Token Pada Variabel Lokal" sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 Departemen Teknik Informatika, Fakultas Teknik, Universitas Hasanuddin.

Penulis menyadari bahwa tanpa bantuan dan bimbingan dari berbagai pihak, penulisan serta punyusunan tugas akhir ini tidak dapat terselesaikan. Oleh karena itu, penulis ingin menyampaikan banyak terima kasih kepada:

1. Kedua Orang tua dan saudara-saudara penulis, yang selalu memberikan dukungan, doa, dan semangat;
2. Bapak Dr.Eng. Ady Wahyudi Paundu, ST., M.T, selaku pembimbing I dan Bapak Iqra Aswad, ST., M.T., selaku pembimbing II yang senantiasa menyediakan waktu, tenaga, pikiran, dan perhatian yang luar biasa untuk mengarahkan penulis dalam penyusunan tugas akhir;
3. Bapak Dr. Indrabayu, S.T., M.T., M.Bus.Sys. selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas bimbingannya selama masa perkuliahan penulis;
4. Semua teman-teman yang telah memberikan begitu banyak bantuan, semangat, inspirasi, ilmu, dan hiburan selama penulisan dan penyusunan tugas akhir;
5. Segenap Staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu penulis;
6. Seluruh teman-teman angkatan 2017 Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin;
7. Teman-teman yang telah membantu melakukan uji coba keamanan sistem yang penulis telah buat;
8. Orang-orang berpengaruh lainnya yang tanpa sadar telah menjadi inspirasi penulis.

Penulis berharap semoga Allah SWT berkenan membalas segala kebaikan dari semua pihak yang telah banyak membantu penulis menyelesaikan tugas akhir ini. Semoga tugas akhir ini dapat memberikan manfaat bagi para membaca. Aamiin.

Makassar, September 2023

Penulis

BAB I
PENDAHULUAN

1.1 Latar Belakang

JSON Web Token (JWT) merupakan sebuah JSON *Object* berbentuk *string* yang digunakan untuk transmisi informasi antar dua atau lebih pihak (RuangKoding, 2022). JWT terdiri dari 3 bagian yang dipisah dengan titik (.), ketiga bagian tersebut adalah *header*, *payload*, dan *signature*.

Di rana pemograman web, JWT paling sering digunakan untuk otorisasi pengguna. Setelah pengguna *login* ke dalam *website*, JWT akan dikembalikan dan sistem akan menyimpan JWT tersebut. JWT yang telah disimpan lalu dapat digunakan oleh sistem untuk otorisasi pengguna yang mengakses *private resource* dari sistem.

Tempat yang paling sering digunakan untuk menyimpan JWT adalah *local storage* atau *cookie*. Kedua metode penyimpanan ini paling sering digunakan karena kemudahan dalam implementasinya. Ketika data disimpan di *local storage* atau *cookie*, data akan tetap ada walaupun pengguna menutup atau memuat ulang *website*. Properti inilah yang membuat *user experience* (UX) dari sebuah *website* menjadi lebih baik. Ketika pengguna memuat ulang *website*, mereka dapat langsung lanjut mengerjakan urusan mereka tanpa perlu *login* ulang terlebih dahulu.

Walaupun mudah untuk digunakan, kedua metode penyimpanan ini memiliki beberapa kelemahan. Jika sebuah *website* yang rentan terhadap serangan *Cross-Site Scripting* (XSS) atau *Cross-Site Request Forgery* (CSRF) menyimpan JWT di *local storage* atau *cookie*, JWT tersebut dapat digunakan oleh aktor jahat untuk melakukan aksi yang pemilik akun tidak inginkan. Sebagai contoh, di sebuah *website* perbankan, jika aktor jahat mendapatkan JWT dari pengguna, dia dapat melakukan transfer uang ke rekening miliknya sendiri, tanpa sepengetahuan pemilik akun sebenarnya.

Satu contoh lagi, pada tahun 2020, platform populer TikTok ditemukan memiliki dua celah keamanan yang membuat penyerang dapat mengambil alih akun pengguna ketika pengguna tersebut *login* menggunakan aplikasi pihak ketiga. Celah keamanan ini ditemukan oleh seorang *bug bounty hunter* asal Jerman

bernama Muhammed Taskiran. Ia menemukan serangan *reflected* XSS pada parameter URL TikTok yang belum di sanitasi, serangan ini dapat menyebabkan eksfiltrasi data TikTok. Dia juga menemukan bahwa API TikTok rentan terhadap serangan CSRF, serangan ini dapat membuat penyerang dapat mengganti password pengguna yang *login* ke TikTok menggunakan aplikasi pihak ketiga (Gatlan, 2020). Artikel ilmiah serupa telah ditulis juga oleh Bulgakova dkk. (2020) yang membuat aplikasi yang menyimpan *access token* di *local variable*. Untuk membuat *access token* tidak hilang pada saat muat ulang, *refresh token* digunakan untuk mengambil *access token* yang baru setelah pengguna menutup atau memuat ulang *website*. *Refresh token* ini disimpan di *http-only cookie* agar ketika pengguna menutup atau memuat ulang *website*, token tersebut tidak akan hilang. Dengan meyimpan token di *local variable, access token* dapat aman terhadap serangan XSS dan CSRF.

Artikel ilmiah lainnya juga di lakukan oleh Calzavara dkk. (2018) yang menggunakan *cookie* dan *web storage* untuk menyimpan *session credentials* yang telah dienkripsi menggunakan *Hash-Based Message Authentication Codes* (HMAC) dan kunci HMAC dari *session credentials* tersebut. Mereka mengusulkan dua strategi, pada strategi pertama, *session credential* disimpan di *web storage* dan kunci HMAC disimpan di dalam *cookie*. *Cookie* tersebut terlindungin menggunakan atribut *HttpOnly* dan *Secure*. Untuk strategi kedua, *session credential* akan disimpan di dalam *cookie* dan kunci HMAC disimpan di *web storage*. Sama seperti strategi pertama, *cookie* akan diberi atribut *HttpOnly* dan *Secure*. Dengan menggunakan *cookie* dan *web storage, website* akan lebih aman terhadap serangan CSRF.

Maka dari itu, di tugas akhir ini, penulis akan mencoba untuk menyimpan JWT di *local variable* agar aktor jahat yang mencoba melakukan serangan XSS dan CSRF ke *website* tidak dapat menemukan dan menggunakan JWT tersebut. Untuk membuat *local variable* menjadi tetap tersimpan pada saat *website* ditutup atau dimuat ulang oleh pengguna, JWT akan di pindahkan dari *local variable* ke *local storage*. Ketika aplikasi dibuka kembali, JWT tersebut akan dikembalikan ke *local variable* dan di hapus dari *local storage*. Untuk memastikan metode ini aman dari serangan XSS, JWT dipindahkan dan dihapus sebelum *browser* menampilkan

element apa pun. Harapnya metode ini dapat menjadi salah satu metode yang di gunakan untuk menyimpan *token* demi autentikas pengguna.

1.2 Rumusan Masalah

Berdasarkan latar belakang, maka rumusan masalah pada tugas akhir ini adalah sebagai berikut:

1. Bagaimana cara membuat JWT di *local variable* menjadi tetap tersimpan walaupun pengguna menutup atau memuat ulang *website*?
2. Apakah metode penyimpanan ini dapat mencegah *leak* ketika terjadi serangan XSS dan CSRF?
3. Bagaimana perbandingan kecepatan buka ulang *website* dan performa dari penyimpanan ketika menggunakan metode penyimpanan *local variable* dan metode penyimpanan *local storage dan cookie*?

1.3 Tujuan Penelitian/Perancangan

Tujuan yang ingin dicapai dari penelitian ini adalah sebagai berikut:

1. Membuat JWT pada *local variable* menjadi tetap tersimpan walaupun pengguna menutup atau memuat ulang *website*.
2. Menguji ketahanan metode penyimpanan di *local variable* terhadap serangan XSS dan CSRF.
3. Membandingkan kecepatan buka ulang *website* dan performa dari penyimpanan ketika menggunakan metode penyimpanan *local variable* dan metode penyimpanan *local storage dan cookie*.

1.4 Manfaat Penelitian/Perancangan

Adapun manfaat yang dapat diperoleh dari penelitian ini adalah sebagai berikut:

1. Manfaat ilmiah, yaitu untuk memberikan sumbangan pemikiran atau menambah informasi mengenai metode penyimpanan JWT pada *website*.
2. Manfaat praktis, yaitu untuk memberikan alternatif metode peyimpanan JWT pada *website* yang aman terhadap serangan XSS dan CSRF.

1.5 Ruang Lingkup/Asumsi perancangan

Ruang lingkup dari penelitian ini adalah sebagai berikut:

1. Serangan yang akan diuji antara lain adalah *Reflected* XSS, *Stored* XSS, dan CSRF *Attack*.
2. Penyerang di asumsikan tidak memiliki akses langsung ke komputer korban.
3. Sistem dibuat rentan terhadap serangan XSS dan CSRF, berarti *element* akan tidak disanitasi dan JWT yang disimpan di *local storage* akan disimpan juga di *cookie* dengan atribut *sameSite* bernilai *None*.
4. Jenis *local variable* yang diuji merupakan *module scope*.
5. Sistem telah ter-*compiled* dan ter-*minified* sebelum pengujian dilakukan.
6. Sistem akan diuji menggunakan *browser* Google Chrome.

BAB II
TINJAUAN PUSTAKA

2.1 *Local Variable*

Yang penulis maksud dengan *local variable* adalah *variable* yang biasa digunakan di aplikasi *website*, yakni *variable* JavaScript (JS) yang memiliki cakupan lokal. Untuk sistem yang penulis buat, *variable* memiliki cakupan lokal terhadap *module* JS (*Module scope*). Untuk definisi v*ariable* sendiri adalah merupakan sebuah media untuk menyimpan nilai/data (Shodiq, 2012).

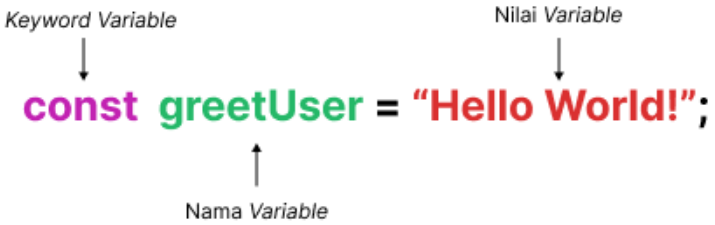
Untuk membuat *variable* di JS, terdapat 3 *keywords* yang dapat digunakan, *keywords* tersebut adalah (skyridetim, 2021).

var, digunakan untuk membuat *variable* global. Itu berarti, *variable* tersebut dapat diakses dan diubah dimana saja.

let, digunakan untuk membuat *variable* lokal. Yaitu, *variable* yang hanya dapat diakses dan diubah pada *scope variable* dibuat.

const, sama dengan *let*, tetapi, *variable* yang menggunakan *keyword* ini tidak dapat diubah.

Keywords inilah yang nantinya digunakan bersama dengan nama dan nilai *variable* untuk membuat *variable* utuh. Untuk lebih jelasnya, berikut struktur *variable* JS.



Gambar 1 Struktur *variable* JS

Sama seperti *variable* di bahasa pemograman lainnya, *variable* di JS disimpan di dua tempat, memori *stack* dan *heap*. *Stack* merupakan bagian memori yang kontinu yang berisi konteks lokal untuk setiap fungsi yang sedang dieksekusi. Sedangkan *heap*, merupakan bagian memori yang jauh lebih besar dari *stack* yang menyimpan data yang dialokasikan secara dinamis (Bahmutov, 2014).

Seperti yang dikatakan sebelumnya, *local variable* pada sistem yang penulis buat merupakan *module scope*. Tapi apa itu *scope*, dan ada berapa jenis *scope* yang dapat digunakan di JS? Berikut penjelasannya.

2.1.1 *Scope* di JS

Scope merupakan aturan yang mengatur ketersediaan dari sebuah *variable*. *Variable* hanya dapat diakses di dalam *scope* itu sendiri. Jika dicoba untuk diakses di luar *scope*, *variable* tersebut tidak akan didapat. Di JS, seperti yang disebutkan sebelumnya, *scope* dibuat menggunakan *keyword let* dan *const*, *keyword* var tidak termasuk kedalam *keyword* untuk membuat *scope*. Ketika menggunakan *var*, *variable* dapat diakses di luar *scope variable* tersebut dibuat

Scope juga memiliki banyak jenis cakupan, cakupan tersebut antara lain (Pavlutin, 2020).

1. *Function scope* merupakan *scope* yang mencakupi isi deklarasi dari sebuah *function*. Jadi jika *variable* dibuat di dalam sebuah *function*, *variable* tersebut hanya dapat diakses di dalam *function variable* tersebut dibuat.
2. *Module scope* merupakan *scope* yang mencakupi seluruh *module*. Module itu sendiri merupakan fitur dari ES2015. *Module* berdiri sendiri, itu berarti, semua *variable* yang tidak di-*export* keluar dari *module*, tidak akan dapat diakses di luar *module* tersebut.
3. *Global scope* merupakan *scope* paling luar. *Scope* ini dapat diakses dari semua *scope* yang lebih dalam (*local scope*). Di *browser*, yang dimaksud dengan *global scope* adalah skrip JS yang dimuat pada *tag* <*script*>.

Untuk memperjelas poin kedua, ketika penyerang mencoba *import* pada saat serangan XSS, dia tidak dapat menemukan apa-apa karena sistem telah ter-*compile* dan ter-*minified* sebelumnya. Untuk serangan CSRF sendiri, karena *cookie* tidak dipasang, serangan tersebut tidak mungkin terjadi.

2.2 *Local Storage*

Local storage merupakan bagian dari HTML5 *web storage* API. Dia sendiri merupakan penyimpanan lokal berbentuk pasangan *key-value pair*. Data yang disimpan di *local storage* bersifat permanen, itu berarti, data akan tetap ada

walaupun browser ditutup atau dimuat ulang oleh pengguna. Untuk ukuran *local storage* sendiri, dia dapat menampung data hingga 5 MB (Jiang dkk., 2015). Hal inilah yang membuat *local storage* sering digunakan untuk menyimpan *access token* guna otorisasi pengguna. Untuk tempat *local storage* disimpan sendiri, dia disimpan pada sebuah *file SQLite* pada subfolder di direktori profil *user* (Obaseki, 2023). Untuk lokasi spesifiknya tergantung dengan *browser* yang digunakan pengguna.

Cara implementasinya juga mudah, berikut beberapa metode yang dapat digunakan (MDN, 2022).

1. *setItem(key, value)*, digunakan untuk menambah data, *key* merupakan indikator/nama data dan *value* adalah nilai data.
2. *getItem(key)*, digunakan untuk membaca data, *key* adalah indikator/nama data yang ingin dibaca.
3. *removeItem(key)*, digunakan untuk menghapus data, *key* merupakan indikator/nama data yang akan hapus.
4. *clear()*, digunakan untuk menghapus semua data dari *local storage*.

Meski mudah digunakan, *local storage* memiliki beberapa kelemahan, kelemahan tersebut adalah (Jemel dan Serhrouchni, 2014).

1. Data disimpan di perangkat pengguna, jadi data dapat diakses oleh pengguna lain menggunakan perangkat yang sama.
2. Data yang berkaitan dengan domain tertentu dapat ditemukan menggunakan serangan *DNS Spoofing*.
3. Website yang memiliki *hostname* yang sama (Sebagai contoh, *website* yang dibuat menggunakan geocities.com) memiliki objek *local storage* yang sama.
4. Dengan XSS, penyerang dapat mengambil atau mengubah data yang tidak dienkripsi.

Untuk memperjelas poin keempat, penulis akan memberikan contoh. Dengan XSS, penyerang dapat melakukan *request* ke *endpoint* yang dilindungi, dengan langsung mengambil *session credential* dari *local storage*.

2.3 Cookie

HTTP *cookie* atau biasa disingkat *cookie* merupakan sebuah paket teks yang dikirim oleh *server* ke *web browser* dan nantinya akan *web browser* kirim kembali ke *server* tiap kali *request* dilakukan ke *server* tersebut. *Cookie* ini biasanya digunakan untuk autentikasi, monitor, dan menjaga informasi tentang pengguna *website*. Sebagai contoh, *cookie* digunakan untuk menyimpan preferensi ke *website* atau isi dari keranjang pada *website e-commerce* (Guo dan Zhou, 2008). Sama seperti *local storage*, *cookie* disimpan di *file* pada *hard drive* atau *browser*, hal ini tergantung pada *browser* atau *operating system* yang digunakan (Croft dan McNally, 2023).

Untuk membuat *cookie* lebih aman, terdapat beberapa *flags HttpOnly* dan *Secure* dapat digunakan. *Flags ini* sendiri berarti (Kwon dkk., 2020)

1. *HttpOnly* memastikan agar *cookie* tidak dapat di akses oleh *client*, misalnya menggunakan JS.
2. *Secure* berarti *cookie* hanya bisa dikirim melalui koneksi yang terenkripsi, misalnya menggunakan protokol TLS.

Meski *flag* tersebut membuat *cookie* menjadi lebih aman, tapi hal tersebut tidak cukup, *cookie* akan masih tetap dapat terkena serangan CSRF. Walaupun *flag Secret* telah ditambahkan, *cookie* akan tetap dikirim tiap request (KirstenS, 2021).

2.4 *Cross-Site Scripting*

Cross-Site Scripting (XSS) merupakan ancaman di *application-layer* yang muncul dari kelemahan bahasa skrip *client-side* seperti HTML atau JS. Tujuan utama serangan XSS adalah untuk meyuntikkan kode berbahaya di *website* demi melewati akses kontrol keamanan dan mengambil data atau melakukan *hijacking* sesi pengguna. Hal ini dilakukan dengan cara memanipulasi dan menyuntik kode berbahaya ke *website* demi mengubah kode berkonteks HTML menjadi kode berkonteks skrip. Penyuntikan kode ini nantinya akan dijalankan oleh *browser* dan akan menjadi *output* dari *website*.

Serangan XSS dapat dibagi menjadi tiga kategori, kategori tersebut adalah (Stasinopoulos dkk., 2015).

1. *Reflected* XSS merupakan serangan yang terjadi ketika pengguna menekan *link*, lebih tepatnya, ketika skrip XSS dimunculkan kembali pada *output* aplikasi.

Tipe serangan ini dapat terjadi dengan pesan *error*, *submit* pencarian, *preview* komentar, dan lain sebagainya.

2. *Stored* XSS merupakan serangan yang tetap tersimpan dengan cara disimpan ke *storage* aplikasi dan dimunculkan tiap kali pengguna mengunjungi *website* atau menampilkan data tersebut.
3. *DOM-Based* XSS merupakan serangan yang mengendalikan *Document Object Model* (DOM) dari sebuah *website*. Serangan *DOM-Based* XSS dapat berbentuk *reflected* maupun *stored*. Penyerang dapat menjalankan skrip JS dengan menargetkan kelemahan di kode HTML dan berinteraksi dengan DOM dari *website*.

```
<img
src="http://whatattacker.com/cookies.php?id="+localStorage
.getItem('SessionID')+"'>");
</img>
```

Gambar 2 Contoh serangan XSS (Sivanesan dkk., 2018)

Untuk menguji serangan XSS, terdapat beberapa hal yang dapat dilakukan. Berikut penjelasannya (OWASP, n.d.-b).

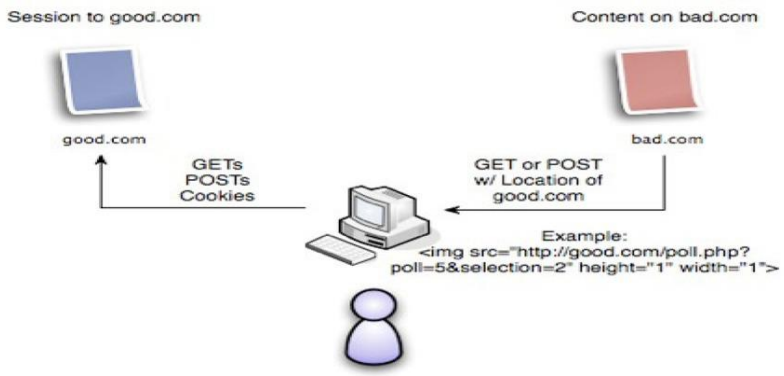
1. *Black-Box Testing*, terdapat tiga tahapan pada uji coba ini, deteksi *input*, menganalisa *input*, dan yang terakhir, periksa dampak serangan.
2. *Bypass XSS Filters*, serangan XSS dapat dicegah dengan melakukan sanitasi *input* dari *website*. Tetapi, bisa saja sanitasi tersebut tidak lengkap. Uji coba ini bertujuan untuk mencari celah pada *input* yang telah disanitasi.
3. *Gray-Box Testing*, hampir sama dengan *Black-Box Testing* tetapi penguji mengetahui beberapa hal tentang aplikasi. Misalnya, *input* pengguna, validasi *input*, dan lain sebagainya.

2.5 *Cross-Site Request Forgery*

Cross-Site Request Forgery (CSRF) merupakan serangan yang memaksa *browser* pengguna untuk melakukan aksi di *website* yang pengguna percaya tanpa sepengetahuan pengguna tersebut. Jenis serangan ini juga biasa di sebut *session riding*.

Serangan CSRF terdiri dari tiga bagian, pengguna, *website* terpercaya, dan *website* penyerang. Seumpama pengguna memiliki sesi aktif di *website* terpercaya, lalu dia mengunjungi *website* penyerang. *Website* penyerang tersebut kemudian menyuntikkan *request* ke *website* terpercaya menggunakan sesi pengguna sebelumnya.

Sebagai contoh, misalnya terdapat pengguna yang *login* ke *website* A. Setelah itu, dia mengunjungi *website* B tanpa keluar terlebih dahulu. Tetap, di *website* B, penyerang telah mengunduh *link* berbahaya yang dapat mengirim *request* ke *website* A yang membutuhkan sesi yang valid. Ketika pengguna mengunjungi *website* B, HTTP *request* akan dikirim ke *website* A bersama dengan sesi valid sebelumnya untuk melakukan aksi di *website* A (Siddiqui dan Verma, 2011). Hal ini dapat terjadi karena, seperti yang disebutkan sebelumnya, *cookie* akan tetap dikirim pada setiap request.



Gambar 3 Ilustrasi terjadinya serangan CSRF (Siddiqui dan Verma, 2011)

Serangan CSRF dapat terjadi karena beberapa faktor sebagai berikut (OWASP, n.d.-a).

1. Perilaku *browser* tentang penanganan informasi mengenai sesi seperti *cookie* dan informasi autentikasi HTTP.
2. Pengetahuan penyerang tentang URL, *request*, atau fungsionalitas dari aplikasi *web* yang valid.
3. Sesi dari aplikasi hanya menggunakan informasi yang ada di *browser*.
4. *Tag* HTML yang dapat melakukan akses ke *resource* HTTP[S]. Misalnya *tag img*.

2.6 *Same-Origin Policy*

Same-Origin Policy (SOP) merupakan salah satu upaya keamanan paling pertama yang *web browser* terapkan. SOP ada sejak era Netscape Navigator 2.0. SOP ini melarang data untuk dibagi antara *origin* yang berbeda, *origin* adalah *host* yang unik (misalnya *website*), *port*, atau protokol aplikasi. Sebagai contoh, SOP mencegah *website* satu untuk mengakses isi atau properti dari *website* lainnya. Dengan begitu, SOP membuat pengguna dapat mengunjungi *website* tidak aman dan membuat *website* tersebut tidak dapat memanipulasi data atau sesi dari *website* terpercaya.

Misalnya, pengguna membuka *website* "http://example.com/index.htm", aturan SOP akan menolak serta menerima skrip dari *website* berikut (Saiedian dan Broyle, 2011).

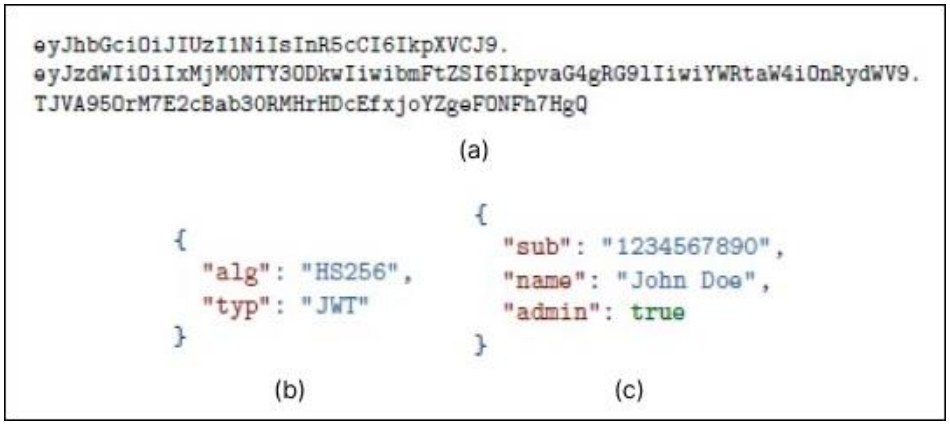
Tabel 1. Contoh aturan SOP (Saiedian dan Broyle, 2011)

Link	**Port**	**SOP**
http://example.com/about.htm	80	Terima
https://example.com/doc.html	334	Tolak
http://google.com/search.php	80	Tolak
http://dev.example.com/more.htm	80	Tolak

2.7 JSON *Web Token*

JSON *Web Token* (JWT) merupakan sebuah JSON *Object* berbentuk *string* yang digunakan untuk transmisi informasi antar dua atau lebih pihak (RuangKoding, 2022). JWT terdiri dari tiga bagian, *header*, *payload*, dan data *signature/encryption*. Bagian dua pertama merupakan *object* JSON dengan struktur yang tetap. Bagian ketiga tergantung dengan algoritma yang digunakan untuk *signing* atau enkripsi. Jika JWT tidak dienkripsi, bagian tersebut dihilangkan. Beberapa fungsi JWT antara lain, autentikasi, otorisasi, dan lain sebagainya (Peyrott, 2018).

Untuk lebih jelas, berikut contoh (a) *string* JWT dan hasil (b) *decode header* dan (b) *payload*-nya.



Gambar 4 Contoh JWT (Peyrott, 2018)

Untuk batas ukuran dari JWT sendiri, RFC7519 (Jones dkk., 2015) tidak menyebutkan apa-apa tentang JWT memiliki batas. Namun, karena JWT biasanya digunakan dalam konteks otorisasi *request*, batas JWT adalah batas ukuran data *header* HTTP *request* yang server bisa terima ketika *request* dilakukan. Berikut beberapa batas ukuran data *header* HTTP *request* dari beberapa *server* paling populer (Den, 2020).

Tabel 2. Batas ukuran data *header* HTTP *request* (Den, 2020)

Web Server	***Size Limit***
Apache	8KB
Nginx	4KB-8KB
IIS	8KB-16KB
Tomcat	8KB-48KB

2.8 *Session Credential*

Yang penulis maksud dengan *session credential* adalah sebuah *token* yang digunakan untuk kebutuhan sesi pengguna. Untuk sistem yang penulis buat sendiri, terdapat tiga jenis *token* yang digunakan, *token* tersebut antara lain.

2.8.1 Access Token

Access token merupakan hal yang aplikasi gunakan untuk melakukan *request* API atas nama pengguna aplikasi. *Access token* mewakili otorisasi aplikasi untuk mengakses bagian dari data pengguna.

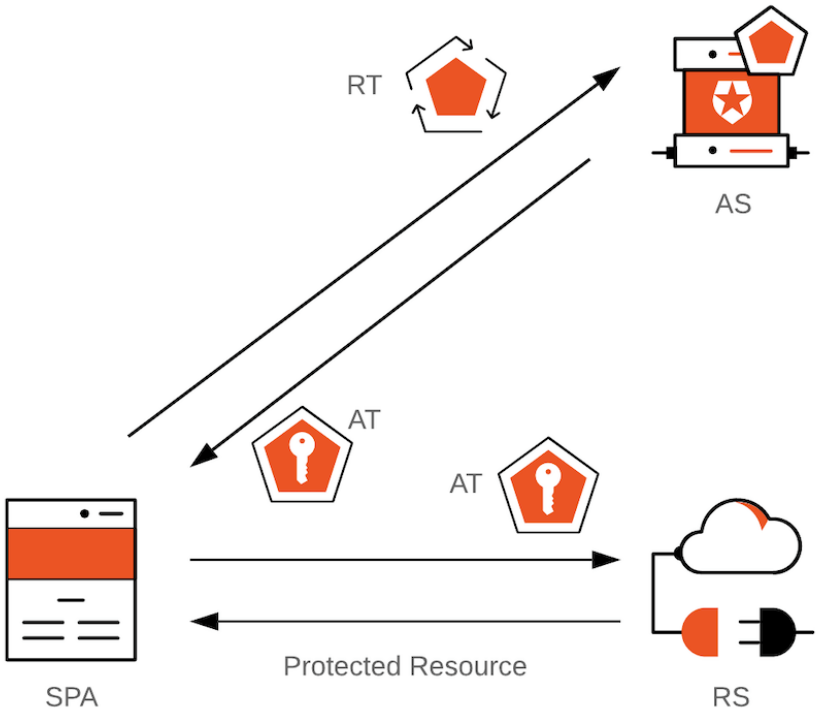
Access token tidak memiliki format tertentu, yang aplikasi cuma tau, *access token* itu merupakan *opaque string* dan aplikasi akan menggunakan apapun *string* tersebut untuk menjalankan sebuah *request*. *Server* tujuan *request* dikirim perlu tahu arti dari *string* tersebut, tetapi, aplikasi tidak perlu tahu.

Access token perlu dijaga kerahasiaannya pada saat pengiriman maupun pada saat disimpan. Yang boleh melihat *token* tersebut hanya aplikasi itu sendiri, *server* untuk otorisasi, dan *server resource* disimpan (Aaron Parecki, n.d.). Salah satu strategi untuk mengurangi risiko bocornya *access token,* dengan membuat *token* yang memiliki jangka valid yang pendek, misalnya, hanya hitungan jam atau hari (Arias dan Bellen, 2021)

2.8.2 Refresh Token

Seperti yang disebutkan sebelumnya, untuk menjaga keamanan aplikasi, *access token* memiliki jangka valid yang pendek. Setelah *access token* habis masa berlakunya, *refresh token* dapat digunakan untuk memperbaharui (*refresh*) *access token*. Jadi, *refresh token* merupakan kredensial yang digunakan aplikasi *client* untuk mendapatkan *access token* baru tanpa perlu meminta pengguna untuk *login* kembail (Arias dan Bellen, 2021).

Untuk lebih jelasnya, berikut ilustrasi yang menjelaskan cara kerja pengambilan ulang *access token* menggunakan *refresh token*.



Gambar 5 Cara mengambil *access token* baru menggunakan *refresh token* (Arias dan Bellen, 2021)

Singkatan pada diagram diatas antara lain, *Single-Page Application* (SPA), *Authorization Server* (AS), *Resource Server* (RS), *Access Token* (AT), dan *Refresh Token* (RT)

2.8.3 CSRF Token

CSRF *token* merupakan nilai yang unik, rahasia, dan tidak bisa ditebak yang dibuat oleh aplikasi *server-side* dan dikirim ke *client*. Nantinya *client* akan menggunakan *token* tersebut pada *request* yang dia buat dan aplikasi *server-side* akan memvalidasi *token* yang dikirim, jika tidak valid *request* akan ditolak.

CSRF *token* dapat mencegah serangan CSRF dengan membuat penyerang tidak dapat membuat *request* HTTP yang valid. Karena penyerang tidak dapat memprediksi CSRF *token* dari pengguna, penyerang tidak bisa mendapatkan semua parameter yang dibutuhkan untuk melakukkan *request* (PortSwigger, n.d.).

2.9 ReactJS

ReactJS atau React merupakan *library user interface* (UI) yang dibuat oleh Facebook (Meta) untuk memfasilitasi pembuatan komponen UI yang interaktif, *stateful*, dan dapat digunakan kembali. *Library* ini paling baik digunakan untuk pembuatan UI yang kompleks dengan performa yang tinggi.

React menggunakan *virtual Document Object Model* (DOM), dimana dia dapat di-*render* di *client-side* maupun *server-side*. *Virtual* DOM merupakan *node tree* yang berisi semua *element* dan *attribute*-nya, dan juga *content* sebagai *object* dan *property*-nya. *Virtual* DOM me-*render* sesuai dengan perubahan yang terjadi pada *state*. Dia berusaha melakukan perubahan DOM yang paling sedikit untuk membuat komponen tetap ter-*update* (Kumar dan Singh, 2016).

React memiliki banyak fitur, fitur tersebut antara lain.

2.9.1 JSX

JSX merupakan perpanjangan sintaksi JS yang membuat *developers* React untuk menulis kode HTML dan JS di *file* yang sama. Berikut contoh kode dari JSX (Thanh, 2021).

```
function App () {
                const greeting = 'Hello Function Component!';
          return <h1>{greeting}</h1>;
}
export default App;
```

Gambar 6 Contoh kode JSX (Thanh, 2021)

2.9.2 *Component*

Component merupakan konsep utama dari React. Dia merupakan fondasi dasar saat membangun UI dari sebuah *website*. *Component* di React itu sendiri merupakan sebuah *element* yang dapat digunakan kembali, dia terdiri dari *markup*, CSS, dan JS. Sama seperti *tag* HTML, *component* React dapat digabung, diatur, dan ditumpuk untuk membuat sebuah halaman *website* (Meta, n.d.-b). Terdapat dua jenis *component*, *class component* dan *function component* (Thanh, 2021).

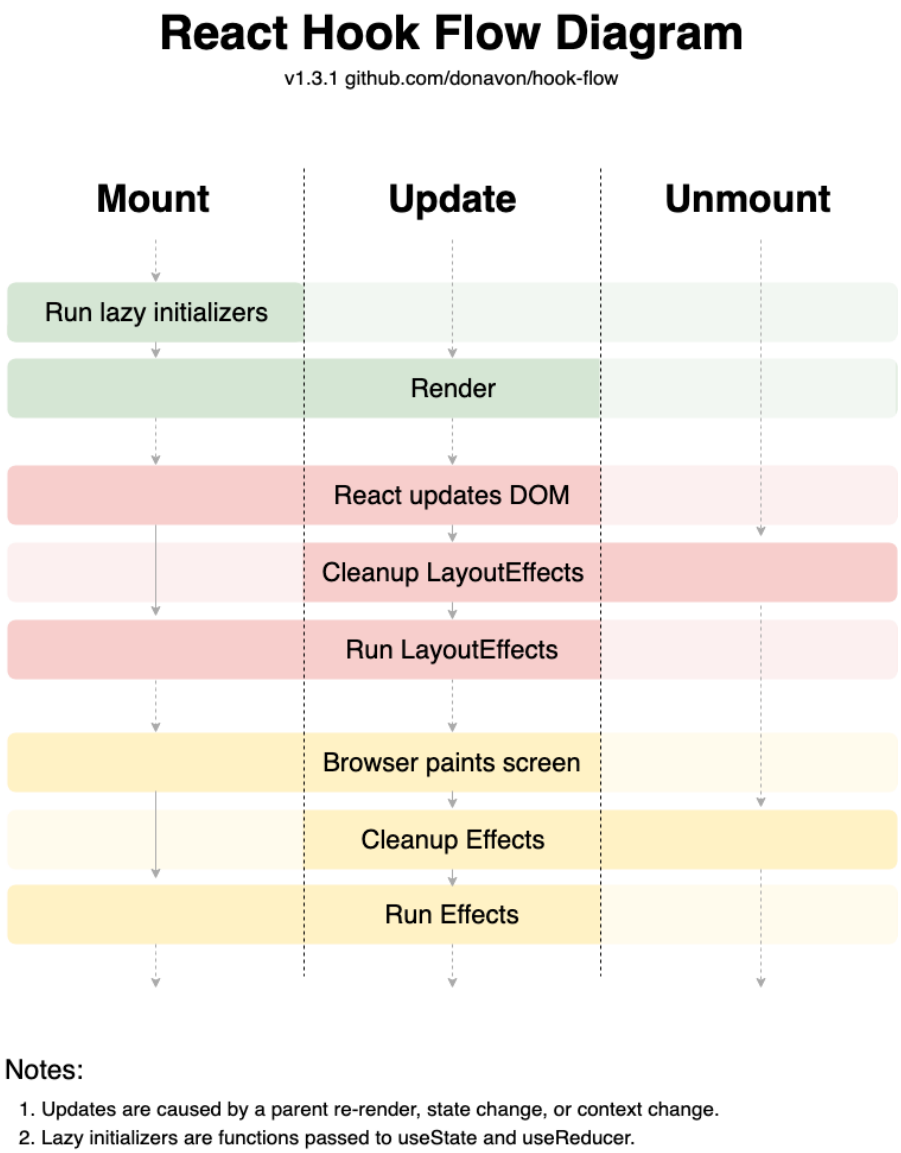
2.9.3 *State*

Component biasanya butuh mengganti apa yang tampil di layar ketika pengguna berinteraksi dengan *website*. Misalnya, mengetik ke sebuah *form* harusnya mengubah isi *input form* tersebut atau menekan tombol "*Next*" pada gambar di *carousel* harunya mengubah gambar yang ditampilkan. *Component* perlu "mengingat" interaksi yang telah dilakukan, baik itu nilai dari *input form* ataupun gambar yang sekarang ditampil. Di React, penyimpanan spesifik dari *component* inilah yang disebut dengan *state* (Meta, n.d.-a).

2.9.4 *Hook*

Hook merupakan fitur yang awalnya diperkenalkan pada React 16.8. Fitur tersebut membuat *developer* dapat menggunakan *state* atau fitur React lainnya pada *function component* ketimbang hanya dapat menggunakan fitur tersebut pada *class component*. *Hook* ini dibuat khusus untuk mengatur *state* dan juga *lifecycle* dari *function component* (Thanh, 2021). Maka dari itu, *hook* ini tidak dapat digunakan pada *class component*.

Seperti disinggung sebelunya, *hook* digunakan untuk mengatur *lifecycle* dari sebuah *function component*, berikut React *hook flow diagram* untuk memperjelas konsep *lifecycle* pada *function component*.



Gambar 7 React *hook flow* diagram (West, 2019)

Semua *component* memiliki tiga fase, fase tersebut adalah sebagai berikut (Pachipulusu, 2020).

1. *Mount*, fase ini terjadi ketika *component* pertama kali dipasang (*mounted*), pada fase ini, step paling pertama adalah "*Run lazy initializers*". *Lazy initializers* merupakan *function* yang dioper ke *useState* dan *useReducer*. *Function* tadi hanya akan dijalankan pada fase ini. Setelah itu, "*Render*", di step ini, *hook useState* dan hal lainnya ada. Step ketiga, "*React update DOM*", memperbaharui DOM itu tidak sama dengan menampilkan *element* ke *browser*. Setelah itu, "*Run Layout Effect*", dimana *useLayoutEffect* akan dijalankan. Lalu

"*Browser paints the screen*", di step inilah baru *element* akan ditampilkan oleh *browser*. Dan yang terakhir, "*Run Effects*", *useEffect* akan dijalankan.

2. *Update*, fase ini terjadi ketika *component* diperbaharui (*update*). *Update* dapat terjadi ketika *parent* dari *component* terkena *render* ulang, *state* dari *component* berubah, atau *context* berubah. Pada fase ini, yang paling pertama terjadi adalah "*Render*". Setelah itu, "*React updates* DOM". Selanjutnya, "*Cleanup Layout Effects*", yang dimaksud *cleanup* di React itu bertujuan untuk merapikan kode, seperti mencegah terjadinya *memory leak* dan lain sebagainya. Selanjutnya, "*Run Layout Effects*". Lalu, "*Browser paints the screen*". Setelah itu, "*Cleanup Effects*", sama seperti *cleanup* pada *useLayoutEffect*, tapi untuk *useEffect*. Dan yang terakhir, "*Run Effect*".
3. *Unmount*, fase ini terjadi ketika *component* dilepas (*unmount*) dari halaman *website*. Yang pertama dilakukan adalah "*Cleanup Layout Effects*". Setelah itu "*Cleanup Effects*"

2.10 *User Experience*

Persyaratan pertama dari *user experience* yang baik adalah memenuhi kebutuhan dari pengguna tanpa menganggu pengguna tersebut. Persyaratan selanjutnya adalah kesederhanaan dan keanggunan yang membuat sebuah produk menyenangkan untuk dimiliki dan digunakan. Agar dapat mencapai kualitas *user experience* yang tinggi, harus terdapat sebuah harmonisasi antara berbagai bidang, antara lain *engineering*, pemasaran, desain *graphic* dan *industrial*, dan desain *interface* (Norman dan Nielsen, n.d.).

Di *website*, terdapat tiga *limit* mengenai *response time* yang perlu diperhatikan, *limit* tersebut sebagai berikut (Nielsen, 1993).

1. 0.1 detik, pengguna merasa sistem beraksi secara instan. Tidak perlu ada *feedback* dari sistem.
2. 1 detik, tidak ada interuspi terhadap *flow* pengguna, namun pengguna tersebut merasa terdapat sebuah *delay*. Biasanya, sistem tidak perlu memberikan *feedback.*
3. 10 detik, merupakan batas untuk menjaga perhatian pengguna. Sistem perlu memberikan *feedback* kepada pengguna.

2.11 NextJS

NextJS merupakan *framework* berbasis React yang di buat oleh Vercel. NextJS memiliki banyak fitur yang membuat *developer* dapat membuat aplikasi *full-stack* dan *web-application* yang dapat digunakan di lingkungan *production*. Beberapa fitur tersebut antara lain *image optimization, internationalization, hybrid static site generation* dan *server-side rendering, typescript support, file-system routing,* API *routes, built-in* CSS *support,* dan *code-splitting* dan *bundling* (Pinnis, 2022).

2.12 MongoDB

MongoDB merupakan *datastore* berbasis NoSQL yang *open source* dan didukung secara komersial oleh 10gen (MongoDB, Inc.). Walaupun MongoDB merupakan *non-relational database,* MongoDB memiliki banyak fitur seperti *relational database*, fitur tersebut antara lain, *sorting, secondary indexing, range queries*, dan *nested document indexing*, operasi seperti *create*, *insert, read, update*, dan *remove*, dan juga *manual indexing*, *indexing on embedded documents*, dan *index location-based data*.

Di MongoDB, data disimpan di sebuah *collection* bernama *document*, di mana *document* ini berfungsi untuk memberikan struktur ke data. Setiap *document* memiliki *key* yang unik bernama “ObjectId”, *key* ini digunakan untuk identifikasi data. *Document* MongoDB diserialisasi menjadi objek JSON dan disimpan secara internal menggunakan *binary encoding* dari JSON bernama BSON (Makris dkk., 2021).