**ABSTRAK**

Teknologi *web* saat ini berkembang cukup pesat dengan hadirnya *web* modern, bahkan *web*site dapat tetap berfungsi tanpa adanya jaringan internet dengan menggunakan teknologi *service worker.* *Service worker* merupakan salah satu API (*Application Programming Interface*) *Javascript* yang memungkinkan pengembang dalam melakukan pemrograman *cache* dan melakukan *load asset data*, melakukan managemen *push notification*, dan yang lainnya. Dengan kemampuan *service worker* menjadikan sebuah *website* dapat difungsikan walaupun koneksi jaringan yang lemah ataupun tidak ada koneksi internet sekalipun (*offline).* *Service worker* mampu mengatasi masalah *website* yang tidak harus selalu bergantung pada koneksi jaringan. Selain itu *service worker* juga memiliki fitur *boost performance* sehingga *web* yang terintegrasi *service worker* mampu membuat waktu akses *web* lebih cepat. Dengan memadukan teknologi *service worker* dan *React JS* sebuah sistem dapat dibuat dengan menggunakan pendekatan *offline first* dan memiliki kemampuan yang cepat. Sebagai studi kasus teknologi ini akan di manfaatkan untuk menunjang proses bisnis dari IKM (Industri Kecil Menengah) yang rata-rata berada pada sektor pedesaan yang minim terhadap jaringan internet. Proses analisis penelitian ini, akan membandingkan aplikasi *web* yang terintegrasi service worker dan *web*site konvensional dengan skenario pengujian tertentu. Analisis yang dilakukan menggunakan indikator waktu rata – rata, proses *browser*, *throughput,* dan penggunaan CPU. Hasil penelitian menunjukkan bahwa *web* yang terintegrasi *service worker* memberikan nilai waktu proses *browser* yang lebih cepat dibandingkan dengan *web* konvensional. Untuk hasil pengujian *throughput* *web service worker* juga memberikan nilai hasil yang lebih besar dibandingkan dengan *web* konvensional. Dengan demikian, diharapkan *web service worker* dapat diimplementasikan bukan hanya pada IKM, tetapi juga pada sistem *web* yang memerlukan kehandalan.

**Kata Kunci:** *web*,*service worker;offline;ReactJS;*

**KATA PENGANTAR**

**DAFTAR ISI**

[ABSTRAK i](#_Toc534194448)

[KATA PENGANTAR ii](#_Toc534194449)

[DAFTAR ISI iii](#_Toc534194450)

[DAFTAR GAMBAR vi](#_Toc534194451)

[DAFTAR TABEL i](#_Toc534194452)

[BAB I PENDAHULUAN 1](#_Toc534194453)

[1.1. Latar Belakang 1](#_Toc534194454)

[1.2. Rumusan Masalah 3](#_Toc534194455)

[1.3. Tujuan Penelitian 3](#_Toc534194456)

[1.4. Manfaat Penelitian 4](#_Toc534194457)

[1.5. Batasan Masalah 4](#_Toc534194458)

[1.6. Sistematika Penulisan 5](#_Toc534194459)

[BAB II TINJAUAN PUSTAKA 6](#_Toc534194460)

[2.1 Pengertian *Website* 6](#_Toc534194461)

[2.2 Modern *website* 7](#_Toc534194462)

[2.3 Teknologi *Offline first app* 11](#_Toc534194463)

[2.3.1 *Single-page Application* 11](#_Toc534194464)

[2.3.2 *Offline First* 13](#_Toc534194465)

[*2.3.3* *Web Caching* 15](#_Toc534194466)

[2.3.4 *Service worker* 18](#_Toc534194467)

[2.4 Bahasa Pemrograman 21](#_Toc534194468)

[2.4.1 . *Java script* 22](#_Toc534194469)

[2.4.2 . Node JS 23](#_Toc534194470)

[2.4.3 . React js 25](#_Toc534194471)

[2.4.3.1. JSX 25](#_Toc534194472)

[2.4.3.2. *Stateful Components* 26](#_Toc534194473)

[2.4.3.3. *Virtual Document Object Model* 26](#_Toc534194474)

[2.5 Basis data 27](#_Toc534194475)

[2.5.1 SQL Basis data 30](#_Toc534194476)

[2.5.2 NoSQL Basis data 31](#_Toc534194477)

[2.5.2.1 IndexedDB 32](#_Toc534194478)

[2.5.2.2 RXDB 34](#_Toc534194479)

[2.5.2.3 Couch DB 34](#_Toc534194480)

[2.5.2.4 Pouch DB 34](#_Toc534194481)

[BAB III METODOLOGI PENELITIAN 36](#_Toc534194482)

[3.1. Lokasi dan Waktu Penelitian 36](#_Toc534194483)

[3.2. Instrumen Penelitian 36](#_Toc534194484)

[3.3. Prosedur Penelitian 37](#_Toc534194485)

[3.4. Tahap Persiapan 39](#_Toc534194486)

[3.5. Gambaran Umum Sistem 39](#_Toc534194487)

[3.5.1. *System Activity* 43](#_Toc534194488)

[3.6. Hasil Pembuatan Sistem 53](#_Toc534194489)

[3.7. Skenario Pengujian 57](#_Toc534194490)

[3.7.1 *Respon Time* 57](#_Toc534194491)

[3.7.2 *Throughput* 57](#_Toc534194492)

[3.7.3 Proses *Browser* 58](#_Toc534194493)

[3.7.4 Prosesor 59](#_Toc534194494)

[BAB IV HASIL DAN PEMBAHASAN 61](#_Toc534194495)

[4.1. Pengujian *Respon Time* 61](#_Toc534194496)

[4.2. *Throughput* 63](#_Toc534194497)

[4.3. Proses *Browser* 65](#_Toc534194498)

[4.3.1 . Waktu akses secara *online* 65](#_Toc534194499)

[4.3.2 Waktu akses secara *offline* 69](#_Toc534194500)

[*4.3.3* Skenario *online* *input data* 71](#_Toc534194501)

[4.4. Performansi CPU 73](#_Toc534194502)

[BAB V PENUTUP 76](#_Toc534194503)

[5.1 KESIMPULAN 76](#_Toc534194504)

[5.2 SARAN 77](#_Toc534194505)

[DAFTAR PUSTAKA 79](#_Toc534194506)

**DAFTAR GAMBAR**

[Gambar 3. 1. Diagram Tahapan Penelitian 37](#_Toc534194623)

[Gambar 3. 2. Proses Manajemen Pemeliharaan Mesin IKM 40](#_Toc534194624)

[Gambar 3. 3. Blok Diagram Sistem 42](#_Toc534194625)

[Gambar 3. 3. Diagram activity web konvensional . 44](#_Toc534194626)

[Gambar 3. 4. Diagram activity web dengan service worker 44](#_Toc534194627)

[Gambar 3. 5. Diagram activity input data pada web konvensional 45](#_Toc534194628)

[Gambar 3. 6. Diagram activity input data pada web dengan service worker 46](#_Toc534194629)

[Gambar 3. 7. Diagram activity update data pada web dengan service worker 48](#_Toc534194630)

[Gambar 3. 8. Proses hapus data pada web dengan menggunakan service worker 49](#_Toc534194631)

[Gambar 3. 8. Hardware Environtment 49](file:///C:\Users\AlRiefqy\OneDrive\Documents\Tugas%20Akhir%20Kodong\SKRIPSI_Al%20Riefqy.docx#_Toc534194632)

[Gambar 3. 9. Use case diagram 52](#_Toc534194633)

[Gambar 3.10. Halaman utama sistem 53](#_Toc534194634)

[Gambar 3. 11. Service worker pada aplikasi web 54](#_Toc534194635)

[Gambar 3. 12. Halaman Mesin 55](#_Toc534194636)

[Gambar 3. 13. Basis data lokal indexedDB 56](#_Toc534194637)

[Gambar 3. 14. Basis data server couch db 56](#_Toc534194638)

[Gambar 3. 15. Network Panel Chrome yang digunakan mengukur waktu akses. 59](#_Toc534194639)

[Gambar 4. 1 Grafik hasil pengujian waktu rata – rata 62](#_Toc534194640)

[Gambar 4. 2 Tabel Hasil Pengujian Throughput pada web Online 63](#_Toc534194641)

[Gambar 4. 3 Grafik hasil throughput 64](#_Toc534194642)

[Gambar 4. 4. Grafik prowses browser untuk waktu load 67](#_Toc534194643)

[Gambar 4. 5. Grafik proses browser DOMContentLoaded 67](#_Toc534194644)

[Gambar 4. 6. Grafik proses browser web service worker 70](#_Toc534194645)

[Gambar 4.7. Grafik waktu load input data 73](#_Toc534194646)

[Gambar 4. 8 Grafik perbandingan performansi CPU 74](#_Toc534194647)

**DAFTAR TABEL**

[Tabel 3. 1. Tabel format basis data 55](#_Toc534194702)

[Tabel 4. 1. Hasil Pengujian Average Secara *Online* 56](#_Toc532554001)

[Tabel 4. 2 Tabel hasil pengujian akses pertama kali tanpa cache 61](#_Toc532554002)

[Tabel 4. 3 Tabel pengujian *online* dengan cache 62](#_Toc532554003)

[Tabel 4. 4 Tabel pengujian offline dengan cache 65](#_Toc532554004)

[Tabel 4. 5 Tabel pengujian *online* input data 67](#_Toc532554005)

[Tabel 4. 6 Tabel pengujian performansi CPU 69](#_Toc532554006)

**BAB I  
PENDAHULUAN**

* 1. **Latar Belakang**

Kontribusi Industri Kecil dan Menengah (IKM) berperan penting dalam mendongkrak pertumbuhan manufaktur dan perekonomian di Tanah Air. Kementrian Perindustrian Republik Indonesia mencatat, jumlah unit usaha IKM di dalam negeri terus mengalami peningkatan setiap tahun. Misalnya, pada tahun 2013, sebanyak 3,43 juta IKM, naik menjadi 3,52 juta IKM pada tahun 2014. Kemudian, mampu mencapai 3,68 juta IKM di tahun 2015, dan bertambah lagi hingga 4,41 juta tahun 2016. Pada triwulan II tahun 2017, jumlah IKM berada di angka 4,59 juta unit usaha [1].

*Maintenance* adalah suatu kegiatan yang dilakukan secara berulang-ulang dengan tujuan agar peralatan selalu memiliki kondisi yang sama dengan keadaan awalnya [2]. Kurangnya modal dan ketidak-efektifan dalam melaksanakan pemeliharaan dan perawatan mesin adalah masalah utama yang ada di IKM. Sistem pengambilan keputusan berbasis web dengan konsep lama yang sudah dibuat sebelumnya menggunakan *PHP MySQL* terkendala pada konektivitas yang lambat dan *bandwith* internet yang rendah.

Oleh karena itu, perlu dibuat sistem yang handal dalam konektivitas jaringan tersebut. Sistem yang akan dibuat juga menerapkan PWA dengan service worker dan framework vue.js pada konsep *Decision Making Grid* (DMG) untuk pengambilan keputusan terhadap proses pemeliharaan mesin di IKM.

*Progressive Web Applications* mulai diperkenalkan oleh *Google* pada tahun 2015 [3]. *Progressive Web Applications* merupakan sebuah aplikasi *web* yang dilengkapi dengan teknologi yang membuatnya dapat menjadi seperti aplikasi *Native* di perangkat *smartphone*, namun fungsinya berjalan melalui *browser* yang terpasang diperangkat *smartphone*. *Progressive Web Applications* dapat ditambahkan dan dibuka langsung melalui *Home Screen*, pada saat dibuka *Progressive Web Application* akan langsung memuat konten-konten yang ada di dalamnya tanpa memperhatikan koneksi jaringan yang dimiliki pada saat itu. Hal ini disebabkan karena adanya bantuan dari *Service Workers*, sebuah *Java Script Workers*, yang dapat melakukan *caching* pada konten untuk memberikan akses secara *offline* [4].

Teknologi dibalik *progressive web apps* adalah *service worker*. Menurut *google developer* salah satu karakteristik dari *progressive web apps* adalah *connectivity independent* dimana  *service workers* membantu web untuk dapat diakses secara *offline* atau pada kualitas koneksi jaringan yang rendah[5].

Vue.js adalah suatu librari *Javascript* yang digunakan untuk membangun antar muka sebuah website yang interaktif. Library dari Vue difokuskan hanya pada view layer dan sangat mudah diimplementasikan dan diintegrasikan dengan library lain ataupun juga dengan project yang sudah ada sebelumnya. [6]

Sistem ini diharapkan tidak hanya mampu mendukung proses pengambilan keputusan dan tetapi juga mendorong IKM untuk menggunakan website dalam proses produksinya.

Berdasarkan uraian diatas penulis kemudian mengangkat sebuah penelitian dengan judul “**Sistem Pendukung Keputusan Pemeliharaan Mesin Untuk Industri Kecil Dan Menengah Menggunakan *Progressive Web Application* dengan *Vue.js* ”** .

* 1. **Rumusan Masalah**

1. Bagaimana cara membuat sistem pendukung keputusan untuk pemeliharaan mesin pada industri kecil dan menengah mengunakan PWA dengan service worker dan framework vue.js?.
2. Bagaimana mengetahui kinerja PWA untuk proses sistem pengambilan keputusan pada IKM?.
   1. **Tujuan Penelitian**
3. Membuat Website Pengambilan keputusan pemeliharaan mesin di IKM dengan memanfaatkan teknologi PWA dan *framework* Vue.js .
4. Membuat sistem yang handal dalam konektifitas jaringan dengan memanfaatkan teknologi *service worker* dari Vue.js.
5. Menerapkan konsep PWA yang dibuat dengan *framework* Vue.js di IKM.
   1. **Manfaat Penelitian**
6. Teknologi *Progressive Web Apps* dapat menjadi pilihan alternatif baru dalam pembuatan website pada IKM.
7. Penelitian ini dapat digunakan untuk menambah pengetahuan dan sebagai referensi mengenai kinerja *website* dengan konsep *Progressive Web App.*
8. Bagi institusi pendidikan, dapat digunakan sebagai referensi dalam pengembangan penelitian topik terkait untuk mempelajari pemrograman berbasis web *Progressive Web Apps*.
   1. **Batasan Masalah**
9. *Website* yang menerapkanteknologi *Progressive Web Apps* dibuat menggunakan *framework website* *HTML5* yaitu *Vue-JS.*
10. Penelitian fokus pada Industri Kecil dan Menengah.
11. Metode pengambilan keputusan yang digunakan adalah DMG aplikasi ketika diakses oleh banyak *request* dan kecepatan pengaksesan *website* yang dibuat.
12. Parameter yang digunakan untuk pengujian kinerja dari *Progressive Web Apps*  adalah *response time* dan browser proses.
13. Perangkat yang digunakan dalam membuat sistem adalah Laptop Asus X455L dengan memori RAM 6GB dan prosesor Intel core i3.
    1. **Sistematika Penulisan**

**BAB I PENDAHULUAN :** Bab ini berisi latar belakang masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, sistematika penulisan.

**BAB II LANDASAN TEORI :** Pada bab ini akan dijelaskan teori-teori yang menunjang percobaan yang dilakukan.

**BAB III METODOLOGI PENELITIAN :** Bab ini berisi analisis kebutuhan sistem, perancangan sistem, dan skenario pengujian.

**BAB IV HASIL DAN PEMBAHASAN :** Bab ini berisi hasil penelitian dan pembahasan penjabaran dari penelitian yang dilakukan.

**BAB V PENUTUP :** Bab ini berisi kesimpulan dari hasil penelitian dan saran

**BAB II  
TINJAUAN PUSTAKA**

1. **Pengertian *Website***

Sebuah situs web (sering pula disingkat menjadi situs saja, *website* atau site ) adalah sebutan bagi sekelompok halaman web ( *web page* ), yang umumnya merupakan bagian dari suatu nama domain ( *domain name* ) atau subdomain di *World Wide Web* *(WWW*) di Internet. Sebuah web page adalah dokumen yang ditulis dalam format HTML (*Hyper Text Markup Language*), yang hampir selalu bisa diakses melalui HTTP, yaitu protokol yang menyampaikan informasi dari server website untuk ditampilkan kepada para pemakai melalui *web browser* baik yang bersifat statis maupun dinamis yang membentuk satu rangkaian bangunan yang saling terkait dimana masing-masing dihubungkan dengan jaringan-jaringan halaman (*hyperlink*) (Ali Zaki, 2009). [1]

Penemu *website* adalah Sir Timothy John “Tim” Berners -Lee, sedangkan *website* yang tersambung dengan jaringan, pertama kali muncul pada tahun 1991. Maksud dari Tim ketika membuat *website* adalah untuk mempermudah tukar- menukar dan memperbarui informasi kepada sesama peneliti di tempat bekerja. Pada tanggal 30 April 1993, CERN (tempat dimana Tim bekerja) menginformasikan bawah WWW dapat digunakan secara gratis oleh semua orang yang dapat diakses melalui sebuah *software* yang disebut *browser,* seperti *internet exploer*, *mozilla firefox, opera* dan lain-lain.(Rahmat Hidayat, 2010, p2) [2]

* 1. ***Progressive Web Apps* (PWA)**

*Progressive Web Apps* (PWAs) merupakan aplikasi berbasis web yang sama seperti halaman *web* ataupun situs *web* biasa hanya saja menawarkan fungsionalitas yang berbeda seperti dapat bekerja secara *offline*, dapat memberikan *push notification* dan akses *hardware* yang tersedia untuk aplikasi *native*. PWA ini sendiri dikembangkan oleh *Google*. *Progressive Web Apps* menggunakan beberapa teknologi diantaranya *Hypertext Transfer Protocol Secure* (HTTPS), *Manifest* dan *Service Workers*.[3]

Pada Mei 2016 Google memperkenalkan Progressive web apps. Progressive web apps dirancang oleh Frances Berriman dan Google Chrome Engineer Alex Russel. Teknologi dibalik progressive web apps adalah service worker. Menurut google developer karakteristik dari progressive web apps adalah sebagai berikut :

* ***Progressive*** – dapat digunakan oleh semua pengguna, terlepas browser apa yang digunakan karena aplikasi dikembangkan secara progressive.
* ***Responsive*** – dapat digunakan pada semua perangkat mulai dari desktop, tablet, smartphone dan lainnya.
* ***Connectivity*** independent – memiliki service workers untuk dapat diakses secara offline atau pada kualitas koneksi jaringan yang rendah.
* ***App-like*** – terasa seperti aplikasi karena model aplikasi shell memisahkan fungsi dari konten aplikasi.
* ***Fresh***– selalu *up-to-date* dikarenakan update proses dari service worker.
* ***Safe*** – dilayani vie HTTPS untuk mencegah pengintaian dan untuk memastikan konten tidak dirusak.
* ***Discoverable*** – yaitu diidentifikasi sebagai "aplikasi" berkat cakupan W3C dan ruang lingkup pendaftaran *service worker*, yang memungkinkan mesin pencari menemukannya.
* ***Re-engageable*** – yaitu membuat keterlibatan ulang menjadi mudah dengan adanya fitur seperti *push notification*.
* ***Installable*** – memungkinkan user untuk menambahkan aplikasi ke home screen tanpa melalui appstore.
* ***Linkable*** – dapat berbagi dengan mudah melalui URL dan tidak memerlukan installasi.

*Progressive Web Apps* memiliki tiga kebutuhan yang harus dipenuhi, yaitu :

* Harus dilayani melalui *Hypertext Transfer Protocol Secure* (HTTPS).
* Memiliki *web app manifest.*
* Memiliki setidaknya satu *service worker*.
  + 1. ***Hybrid application***

*Hybrid application* adalah aplikasi yang memadukan elemen dari *native application* dan *web application*. Melalui *hybrid application*, pembuat aplikasi dimungkinkan untuk membuat sebuah aplikasi yang berjalan pada lebih dari satu sistem operasi dan mengakses fasilitas asli (*native*) dari sebuah *device*, seperti kamera, *geolocation*, kalender, dan lain-lain.

Definisi *hybrid application* berdasarkan dokumentasi yang ditulis oleh Appcelerator *online* adalah sebuah aplikasi dimana sebagian atau seluruh antarmuka dan logikanya ditulis dalam HTML, CSS dan JavaScript yang dijalankan dalam sebuah “*native wrapper*” seperti misalnya Titanium *Web*View ataupun PhoneGap container. *Hybrid application* memiliki akses terbatas untuk mengakses piranti keras perangkat walaupun cara akses dapat bermacam-macam, tergantung dari sistem operasi *mobile* dan kerangka kerja yang digunakan.

Terdapat beberapa kelebihan dari *hybrid application* dibandingkan dengan *native application*, yaitu:

1) *Faster time to market* (waktu yang lebih cepat untuk memasarkan)

2) *Inexpensive cross-platform development cycle* (siklus pengembangan cross-*platform* yang tidak mahal)

3) *Abundant human resource* (sumber daya manusia yang melimpah)

4) *Cost of maintenance* (biaya perawatan)

5) *Approval process* (proses persetujuan)

6) *Hybrid apps are the future* (*hybrid aplication* adalah masa depan)

Terdapat pula beberapa kelebihan dari *hybrid application* dibandingkan dengan *web application*, yaitu:

1) *Access to device capabilities* (akses ke kemampuan perangkat)

2) *Unavailable new platform features* (fitur *platform* baru yang tidak tersedia)

3) *Distribution through app stores* (distribusi melalui *app store*)

4) *Offline access and execution* (akses dan eksekusi secara *offline*)

* + 1. ***Single-page Application***

Secara tradisional, *web* memiliki banyak tampilan halaman. Kebanyakan halaman *web* secara konsisten hanya sebuah halaman statis yang berisi teks dan gambar. Ini memungkinkan untuk melakukan koneksi antara halaman *web* melalui *hyperlink*. Ketika pengguna membuka halaman *web* yang lain melalui *hyperlink* maka *browser* akan melakukan *refresh* seluruh tampilan halaman *web* dengan konten yang baru dari halaman *web* yang baru. Kemudian muncul skrip sisi server seperti PHP yang memungkinkan untuk membuat halaman *web* yang dinamis berdasarkan dengan interaksi dari pengguna.

Kemudian paradigma baru dimulai sejak sekitar tahun 2005 ketika model pemrograman *asynchronous Java script* dan XML (AJAX) menjadi daya tarik baru. AJAX memungkinkan halaman *web* untuk melakukan 12

permintaan HTTP ke *web* server, dan melakukan pembaharuan hanya pada bagian halaman *web* yang berubah. Revolusi lain dimulai pada sekitar tahun 2006 ketika dua kelompok yaitu *World Wide Web Consortium (W3C)* dan *Web Hypertext Application Technology Working Group (WHATWG)*, mulai bekerja bersama untuk sebuah standar *Hypertext Markup Language (HTML)* yang baru yaitu HTML 5. Standar HTML 5 yang baru menambahkan berbagai macam fitur kedalam sebuah perangkat *web* yang memungkinkan pengembang untuk membangun sebuah *web* yang lebih kompleks dan lebih kaya akan fitur (Vanhala 2017).

Evolusi *web* mengantarkan kepada teknologi baru dalam membangun sebuah aplikasi *web*. *Single Page Application* (SPA) adalah aplikasi *web* yang menggunakan halaman HTML yang tidak melakukan *refresh* pada saat digunakan. Sebagai gantinya SPA menggunakan *Java script* untuk setiap interaksi pengguna dan mengandalkan AJAX ketika ingin melakukan komunikasi dengan server. SPA memberikan pengalaman user yang lebih baik, waktu respon yang lebih cepat dari *web* tradisional sejak interaksi server terjadi secara *asynchronous* pada sisi belakang aplikasi, dan memerlukan data yang sedikit dalam melakukan transaksi (Vanhala 2017).

* + 1. **Web app Manifest**

*Web app manifest* merupakan *file JSON* yang memberitahukan *browser* tentang aplikasi *web* yang dijalankan seperti bagaimana harusnya berperilaku ketika dilakukan instalasi pada perangkat selular atau *desktop*, ini diperlukan untuk memunculkan permintaan “Tambahkan ke Layar Beranda”.

* + 1. **Lighthouse**

Lighthouse adalah aplikasi *open source* yang dibangun oleh Google untuk menguji konsep PWA dengan aspek-aspeknya, *performance*, *accessibility* dan *best practices*. Lighthouse dapat dijalankan dari *browser* pengguna berupa ekstensi pada Google Chrome. Pengujian menggunakan Lighthouse hanya bisa dijalankan untuk satu pengujian pada satu *browser client*. Untuk melakukan pengujian menggunakan Lighthouse pada banyak *client* digunakan layanan *third-party* Calibre.

* + 1. **Service Workers**

*Service worker* adalah salah satu jenis dari *web worker*, yaitu script yang berjalan di belakang *browser* pengguna. *Service worker* pada dasarnya adalah berkas JavaScript yang berjalan pada *thread* yang berbeda dengan *main thread browser*, menangani *network request*, *caching*, mengembali-kan *resource* dari *cache*, dan bisa mengirimkan *push message*. Aset web dapat disimpan sebagai *cache* lokal, sehingga dengan jaringan internet yang kurang memadai pun, pengguna tetap mendapat pengalaman yang baik. Aplikasi dapat tetap menjalankan halaman web yang sudah di-*cache* atau memberikan status koneksi tanpa *browser* menampilkan tulisan eror karena ketiadaan koneksi internet.[4]

Langkah-langkah dalam melakukan konfigurasi dasar *service worker* sebagai berikut:

* Daftarkan *service worker* melalui URL (*Uniform Resource Locator*) fungsi *serviceWorkerContainer*.register().
* Jika berhasil, *service worker* dijalankan di *ServiceWorkerGlobalScope*.
* *Service worker* telah siap untuk memproses *event*.
* Instalasi *service worker* dicoba ketika *service worker* mengontrol halaman yang diakses setelah dan sebelumnya. *Event install* akan selalu dikirim pertama kali ke *service worker*.
* Ketika handler oninstall selesai, *service worker* dipasang.
* Proses aktivasi. Ketika *service worker* terpasang, selanjutnya akan menerima *event activate*. Penggunaan utama dari *onactivate* ini adalah untuk membersihkan sumber daya yang digunakan sebelumnya.
* *Service control* sekarang dapat mengontrol halaman, tapi hanya dibuka setelah *register*() telah sukses seperti dokumen mulai aktif dengan atau tanpa *service worker*dan menjaganya selama masih digunakan. Jadi dokumen harus dimuat ulang agar benar-benar terkontrol[5].
  + 1. **JSON**

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemprograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran data. JSON terbuat dari dua struktur:

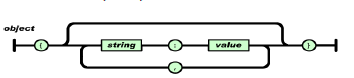
1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.

2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini

dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

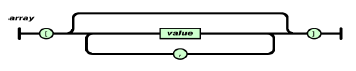
Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut:

1. **Objek** adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma).



Gambar 1. JSON Object

2. **Larik** adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [ (kurung kotak buka) dan diakhiri dengan ] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).



Gambar 2 JSON Array

* + 1. ***Chace***

*Cache interface* menyediakan mekanisme penyimpanan untuk pasangan objek *Request* dan *Response* mau disimpan ke dalam *cache*, contohnya sebagai bagian dari daur hidup *service worker*. Perlu diketahui bahwa *cache interface* terbuka terhadap halaman web dan juga *workers*. *Cache* tidak harus selalu digunakan bersamaan dengan *service worker* walaupun *cache* tercantum di dalam spesifikasinya . *Cache* digambarkan sebagai sebuah *array* berisi objek *Request* yang bertindak sebagai pasangan untuk responsnya yang disimpan di dalam *browser* .

* 1. **Bahasa Pemrograman**

“Program adalah algoritma yang ditulis dalam bahasa komputer”. Pemrograman adalah proses mengimplementasikan urutan langkah untuk menyelesaikan suatu masalah dengan menggunakan bahasa pemrograman. Penulisan program biasanya menggunakan menggunakan program editor yang telah disediakan oleh bahasa pemrograman yang dipilih.

* Langkah-Langkah Pembuatan Program Dalam menyusun suatu program yang besar dan kompleks dibutuhkan beberapa tahapan yang sistematis dan terpadu, yaitu sebagai berikut: Mendifinisikan masalah, Analisis kebutuhan, Desain algoritma, Pengkodean, Bahasa pemrograman, Testing dan debugging, Dokumentasi, Pemeliharaan.
* Menentukan modul-modul program. Penentuan modul utama dan modul ritun sangat berguna untuk mempermudah penanganan jika terjadi kesalahan.
* Penyusunan algoritma program. Algoritma dibuat dengan tujuan untuk menyelesaikan masalah dengan menuliskan langkah-langkah pemecahan masalah yang ada.

Bahasa pemrograman merupakan prosedur penulisan. Ada tiga *record* dalam penulisan bahasa pemograman.

* *Syntax* adalah aturan penulisan bahasa tersebut (tata bahasa).
* *Semantic* adalah arti atau maksud yang terkandung di dalam *statement* tersebut.
* Kebenaran logika adalah berhubungan dengan benar tidaknya urutan *statement.* 
  + 1. **JavaScript**

JavaScript adalah bahasa scripting kecil, ringan, berorientasi objek yang ditempelkan pada kode HTML dan di proses di sisi *client*. JavaScript digunakan dalam pembuatan website agar lebih interaktif dengan memberikan kemampuan tambahan terhadap HTML melalui eksekusi perintah di sisi browser. JavaScript dapat merespon perintah user dengan cepat dan menjadikan halaman web menjadi responsif. JavaScript memiliki struktur sederhana, kodenya dapat disisipkan pada dokumen HTML atau berdiri sebagai satu kesatuan aplikasi. Struktur penulisan JavaScript adalah sebagai berikut.

<script language = “javascript”><!--Penulisan kode javascript//--></script>

* + 1. **jQuery**

jQuery adalah JavaScript library yang dirancang untuk meringkas kode-kode JavaScript, sehingga dapat menyederhanakan penulisan skrip program, sesuai dengan slogan “write less, do more” . jQuery pertama kali dirilis oleh John Resig pada tahun 2006, pada perkembangannya jQuery tidak hanya sebagai framework JavaScript, namun memiliki kelebihan antara lain.

* Kemudahan mengakses dan memanipulasi elemen-elemen HTML.
* Memanipulasi CSS.
* Penanganan eventHTML.
* Efek-efek JavaScript dan animasi.
* Memodifikasi elemen HTML DOM.

Sintak dasar jQuery $(selector).action(), tanda $ untuk mendefinisikan jQuery, jQuery selector digunakan untuk mendapatkan elemen HTML, action adalah tindakan yang dilakukan jQuery pada elemen ()

Contoh penggunaan jQuery untuk menyembunyikan elemen dengan id “test” sebagai berikut.

$(“#test”).hide()

Semua metode jQuery berada di dalam fungsi document.ready() yaitu perintah inisialisasi yang menunjukan dokumen telah siap ditampilkan dan sekaligus menjalankan perintah yang terdapat didalam fungsi.

* + 1. **Vuejs**

VueJs adalah javascript framework yang dikembangkan untuk membangun antarmuka suatu software. VueJs telah menyediakan berbagai macam fungsi javascript yang telah dimodifikasi sehingga programmer dapat lebih mudah untuk membangun software, tentunya dengan aturan-aturan tertentu. Selain memudahkan dalam pengembangan, VueJs juga memberi kemudahan kepada pengguna software dalam menggunakan software itu sendiri dengan realtime response, di mana VueJs meminimalkan waktu antara aksi user dengan respons perangkat lunak. VueJs pertama kali dirilis pada Februari 2014 oleh Evan You setelah bekerja di Google menggunakan AngularJS di beberapa proyek.

**2.3.3.1 Keunggulan Vue.js**

* Menggunakan file yang sangat kecil

Keberhasilan Javascript tergantung dengan ukurannya. Semakin kecil akan semakin baik dan semakin banyak yang menggunakan. Framework ini hanya memakan 18–21KB dibandingkan framework lainnya yang lebih besar.

* Mudah dipahami

Dengan struktur yang sederhana, VueJS mudah dipahami. Dengan begitu, akan lebih sedikit waktu yang digunakan untuk memahami framework ini

* Integerasi yang sederhana

Salah satu faktor yang membuat VueJS banyak digunakan adalah integrasi yang sederhana. Integrasi yang mudah ke aplikasi yang sudah ada membuat developer tidak perlu repot-repot.

* “Two-way communication”

Dengan menggunakan MVVM architecture membuat lebih mudah mengurus HTML blocks. Hal ini seperti yang dilakukan oleh Angular.

* Dokumentasi yang detail

Dokumentasi di VueJS sangat komprehensif, bahkan orang yang mengetahui sedikit tentang Javascript dan HTML akan bisa men-develop web page nya sendiri

* Fleksibilitas

VueJS dapat ditulis dengan file HTML ataupun File JavaScript yang memudahkan developer framework lainnya untuk memahami framework VueJS.

* 1. **Basis Data**

Basis data adalah kumpulan data yang saling berelasi. Data sendiri merupakan fakta mengenai objek, orang dan lain-lain. Data dinyatakan dengan nilai(angka, deretan karakter, atau simbol).

Basis data bertujuan untuk mengatur data sehingga diperoleh kemudahan, ketepatan dan kecepatan dalam pengembalian kembali. Untuk mencapai tujuannya, syarat sebuah basis data yang baik adalah sebagai berikut: Tidak adanya redudansi dan inkonsistensi data, Kesulitan pengaksesan data, *Multiple user.*

* + 1. **Basis Data NoSQL**

Istilah NoSQL diciptakan oleh Carlo Strozzi pada tahun 1998 dan mengacu pada database non-relasional, pada tahun 2009 Eric Evans memperkenalkan kembali istilah NoSQL. Baru-baru ini, istilah ini memiliki makna lain, yaitu "Not Only SQL", istilah yang lebih baik dari sebelumnya yang lebih dikenal dengan ―anti relasional, NoSQL mengakomodasi tanda yang tidak terstruktur, kehadirannya bukan untuk menggantikan SQL namun kedua teknologi ini dapat saling berdapampingan. Perbedaan utama kedua database ini adalah SQL memiliki skema yang kaku sementara database NoSQL menawarkan desain yang fleksibel yang dapat diubah tanpa downtime atau gangguan layanan. NoSQL juga dirancang untuk menyimpan data yang didistribusikan untuk kebutuhan data dalam skala besar; misalnya Facebook memiliki 500 juta pengguna dan Twitter terakumulasi terabyte data, database NoSQL telah memiliki popularitas yang tinggi, sehingga database ini diklaim lebih baik dari database SQL, Database NoSQL dimotivasi oleh kesederhanaan, horizontal scaling dan kontrol yang lebih dalam kesediaan data.

NoSQL menjadi solusi dalam penanganan data dalam jumlah besar yang berkembang pesat saat ini. Data ini biasanya non-terstruktur, kompleks dan tidak cocok digunakan dalam model relasional. Contoh data yang bisa kita yang bisa kita rasakan adalah data yang berasal dari smartphone yang mencatat lokasi broadcast setiap saat, video dan kamera bahkan halaman halaman website yang berisi banyak informasi serta dokumen .

Basis data NoSQL mempunyai karakteristik BASE (*Basically, Available, Soft state* and *Eventual Consistency*) yang merupakan kebalikan dari ACID pada basis data SQL. Setelah transaksi yang konsisten, keadaan (*state*) yang didapat adalah keadaan sementara (*soft state*) bukan keadaan tetap (*solid state*). Fokus utama dari BASE adalah ketersediaan permanen. Karakteristik berikutnya adalah CAP (*Consistency, Availability dan Partition*) yang mempunyai tiga prinsip utama:

* Data tersedia pada semua mesin harus sama di semua aspek dan *updatem* harus dilakukan terhadap semua mesin.
* Data harus tersedia secara permanen dan harus dapat diakses setiap saat.
* Pada saat terjadi kegagalan mesin atau kesalahan lainnya, basis data tetap

bekerja dengan baik tanpa ada pekerjaan yang berhenti.

Klasifikasi model data NoSQL terdiri dari: Key Value Stores, Document Stores, Column Family Stores and Graph Database.

* Key Value Stores- Key value stores memiliki kesamaan dengan pemetaan data atau direktori dimana data ditangani oleh sebuah kunci unik, dalam pelaksanaannya sangat sederhana yaitu berdasarkan atribut kunci saja.
* Document Stores- Document Stores merupakan data dalam bentuk dokumen JSON, lebih fleksibel. Secara konsep data ini memiliki jenis : JSON, BSON, XML and BLOBs.
* Column Family Store- Column Family Stores juga dikenal dengan data yang berorientasi kolom. Media penyimpanan ini terinpirasi oleh Googles Bigtable. Data disimpan dalam kelompok sel dalam sebuah kolom, dan kolom tersusun dari group kolom. Secara teori jumlahmya tidak terbatas.
* Graph Database – Database graph efesien dalam penyimpanan data dalam bentuk grafik atau gambar.
  + 1. **Basis Data SQL**

Basis data SQL mempunyai karakteristik ACID (*Atomic, Consistent, Isolation* dan *Durability*) . *Atomic* adalah apabila ada sebuah transaksi yang terdiri dari dua atau lebih komponen informasi, semua komponennya harus disimpan atau semua komponennya tidak disimpan. Tidak ada sebagian saja komponen yang disimpan atau tidak disimpan. *Consistent* adalah data yang disimpan tidak boleh melanggar integritas basis data. Perubahan data yang mengalami gangguan dibatalkan untuk memastikan basis data berada dalam kondisi sama seperti sebelum ada perubahan. *Isolation* adalah sebuah transaksi tidak dipengaruhi oleh transaksi lain yang sedang berjalan. Hal ini untuk mencegah terjadinya benturan data antar transaksi. *Durability* adalah apabila transaksi sudah disimpan di basis data secara permanen, untuk seterusnya transaksi tersebut ada di basis data meskipun terjadi kegagalan sistem.

Basis data SQL memiliki kelebihan yaitu dapat memproses query yang kompleks dalam waktu yang relatif lebih singkat dan mendukung konsistensi . Selain itu basis data SQL dapat melakukan *subquery*, *joins* dan *grouping/aggregation* . Basis data SQL dirancang untuk berjalan pada satu server sehingga apabila basis data memerlukan tempat penyimpanan data yang besar maka diperlukan server dengan ukuran besar. Basis data SQL mempunyai kekurangan dalam mendistribusikan data dan beban secara merata terhadap sebanyak mungkin server .

* 1. **Firebase**

Firebase Adalah *Backend as a Service* (BaaS) yang saat ini dimiliki oleh Google. Firebase merupakaan solusi yang ditawarkan oleh Google untuk mempermudah pengembangan aplikasi *mobile*. Dua fitur menarik dari Firebase adalah Firebase Remote Config dan Firebase Real Time Database. Selain itu juga terdapat fitur pendukung untuk aplikasi yang memerlukan *push notification* yaitu Firebase Notification Console.

Firebase Database merupakan penyimpanan basis data non-SQL yang memungkinan untuk menyimpan beberapa tipe data. Tipe data itu antara lain String, Long, dan Boolean. Data pada Firebase Database disimpan sebagai objek JSON *tree*. Tidak seperti basis data SQL, tidak ada tabel dan baris pada basis data non-SQL. Ketika ada penambahan data, data tersebut akan menjadi *node* pada struktur JSON. *Node* merupakaan simpul yang berisi data dan bisa memiliki cabang-cabang berupa *node* lainnya yang berisi data pula. Proses pengisian suatu data ke Firebase Database dikenal dengan istilah *push*.

Selain Firebase Database, Firebase menyediakan beberapa layanan lainnya yang juga dimanfaatkan dalam pengembangan aplikasi ini. Layanan tersebut antara lain Firebase Authentication, Storage, dan Cloud Messaging. Pada pengembangan aplikasi, layanan lainnya yang digunakan pada pengembangan aplikasi adalah Firebase Storage. Layaknya sebuah penyimpanan awan, Firebase Storage memungkin pengembang untuk mengunggah atau mengunduh sebuah berkas.

**2.5.1 Firebase Authentication**

Sebagian besar aplikasi tentu membutuhkan autentikasi untuk mengetahui identitas pengguna. Mengetahui identitas pengguna memungkinkan aplikasi untuk secara aman menyimpan data pengguna di *cloud* dan memberikan pengalaman pribadi yang sama di semua perangkat pengguna. Firebase menyediakan layanan backend, SDK yang mudah digunakan dan pustaka siap pakai untuk mengautentikasi pengguna ke aplikasi. Saat ini Firebase mendukung autentikasi menggunakan sandi, sosial media popular seperti Google, Facebook, Twitter dan lainnya.

**2.5.2 Firebase Realtime Database**

Firebase Realtime Database merupakan *cloud database*. Data disimpan dalam format JSON dan disinkronkan secara *realtime* ke setiap klien yang terhubung. Ketika membangun aplikasi *hybrid* lintas platform, seperti Android dan iOS maka semua klien berbagi satu *instance* Realtime Database dan secara otomatis menerima pembaruan dengan data tertentu. Firebase Realtime Database adalah basis data NoSQL dan karena itu memiliki optimalisasi dan fungsionalitas yang berbeda dibandingkan dengan basis data relasional. Membuat database Firebase bisa melalui import file JSON ke konsol Firebase, atau dapat juga dibuat langsung melalui halaman konsol Realtime Database secara manual.

**2.5.3 Firebase Storage**

Firebase Storage digunakan untuk menyimpan data seperti gambar, audio dan video. Sebagian aplikasi pasti menggunakan file gambar, audio atau video. Adanya Firebase Storage sangat memudahkan proses unggahan dan unduhan untuk aplikasi. Beberapa kelebihan utama dari Firebase Storage adalah sebagai berikut:

* *Strong*

Firebase Storage dapat melakukan unggahan dan unduhan file di semua kualitas jaringan internet. Ia dapat berhenti atau melanjutkan, sehingga menghemat waktu dan *badwidth* pengguna.

* *Secure*

Firebase Storage terintegrasi dengan Firebase Authentication untuk menyediakan autentikasi yang mudah dan intuitif untuk pengembang. Model keamanan dapat diatur berdasarkan nama file, ukuran, tipe konten dan metadata lainnya.

* *Scalable*

Firebase Storage didukung oleh Google Cloud Storage hingga skala petabyte. Ini sangat penting, ketika aplikasi yang dari prototipe kemudian berkembang viral dengan jutaan pengguna.

**BAB III  
METODOLOGI PENELITIAN**

1. **Lokasi dan Waktu Penelitian**

Proses penelitian dilakukan sejak bulan Februari 2019. Dan lokasi penelitian dilakukan di Laboratorium *Cloud Computing and Internet Engineering* Departemen Informatika Fakultas Teknik Universitas Hasanuddin Gowa.

1. **Instrumen Penelitian**

Instrumen yang digunakan pada penelitian ini terdiri dari Hardware dan Software.

1. *Hardware*

*Hardware* atau perangkat keras yang dibutuhkan untuk mengimplementasikan rancangan sistem adalah :

1. Komputer Server, Komputer server digunakan untuk menempatkan program dan basis data sistem *web* pemeliharaan mesin industri kecil menengah.
2. Komputer Klien, digunakan untuk melakukan pengaksesan program yang berada pada komputer server melalui jaringan.
3. Modem / Router.
4. Printer.
5. *Software*

Software atau perangkat lunak yang dibutuhkan untuk mengimplementasikan rancangan sistem adalah :

1. Sistem Operasi Windows 7
2. *Web* *Browser* Google Chrome
3. Sublime Text 3
4. Firebase
5. Node JS
6. Library Vue JS
7. Apache Jmeter
8. Material UI
9. IndexedDB
10. **Prosedur Penelitian**

Tahapan pada penelitian ditunjukkan pada gambar 3.1

Studi Literatur

Persiapan Instrumen Pembuatan Sistem

Desain dan Perancangan sistem

Implementasi Desain dan Perancangan Sistem

Sinkronisasi data *offline*

Pemasangan *Service worker* pada sistem

Pengambilan Data

Performansi Sistem

Analisis Data

Gambar 3. 1. Diagram Tahapan Penelitian

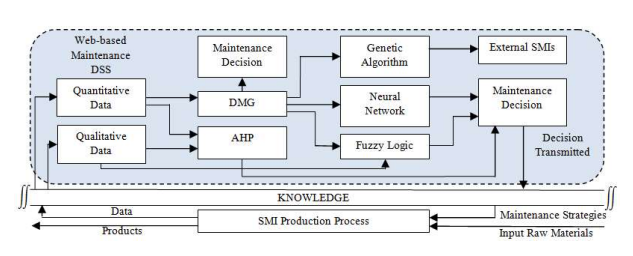
Tahapan secara garis besar dijelaskan sebagai berikut.:

1. Pada studi literatur dilakukan pencarian penelitian terkait dengan *service worker* dan *offline first app* dan juga penelitian sebelumnya yang berkaitan dengan topik penelitian yang dapat menunjang kegiatan penelitian.
2. Pada tahap ini, dilakukan desain rancangan sistem yang akan dibuat mulai dari perancangan alur kerja sistem, dan perancangan basis data sistem.
3. Pada tahap ini dilakukan persiapan berbagai jenis instrument yang dibutuhkan dalam penelitian, berupa software maupun hardware. Segala macam software yang dibutuhkan disiapkan dengan cara memasang atau meng-*install* di perangkat komputer yang akan digunakan untuk membangun sistem.
4. Pada tahap ini, desain sistem yang telah dirancang kemudian di implementasikan dan dibangun menggunakan instrument yang sudah disiapkan berupa perangkat komputer, komputer server , Node JS dan basis data CouchDB
5. Pada tahap ini dilakukan pengimplementasian *offline first app* pada sistem menggunakan *service worker* sehingga sistem yang dibuat dapat di akses dalam kondisi *offline* sekalipun.
6. Pada tahap ini, kemudian sistem dibuat agar mampu melakukan sinkronisasi data secara berrala antara basis data server CouchDB dengan basis data IndexedDB. Sehingga manajemen basis data tetap dapat dilakukan dalam kondisi *offline*.
7. Pada tahap ini, setelah sistem telah dibangun maka dilakukan pengambilan data performansi berupa waktu akses, paket lost, dan penggunaan sumber daya menggunakan jmeter dengan menggunakan skema terentu.
8. Setelah dilakukan pengambilan data, maka data yang diperoleh dari berbagai skema kemudian dilakukan analisis. Analisis dilakukan dengan membandingkan dua buah sistem yang menggunakan teknologi *offline first app* dan *service worker* dengan aplikasi *web* biasa.
9. **Tahap Persiapan**

Tahap persiapan merupakan tahapan awal dalam melakukan penelitian, pada tahap ini penulis melakukan studi literartur dengan mencari penelitian terkait dan jurnal yang bersangkutan denga subjek penilitan dengan tujuan untuk melengkapi literatur mengenai penelitian ini. Penulis juga melakukan identifikasi masalah pada penelitian ini, membaca dan memahami kelebihan dan kekurangan dari penelitian yang sudah dilakukan sebelumnya. Dan juga penulis menyiapkan alat dan bahan yang diperlukan dalam penelitian ini untuk mempersiapkan menuju ke tahap selanjutnya. Selain itu pada tahap ini dilakukukan desain perancangan sistem yang akan dibuat untuk dilakukan analisis dan menyiapkan berbagai jenis intrumentasi baik berupa software maupun hardware untuk membangun sistem yang akan dibuat.

1. **Gambaran Umum Sistem**

Sistem yang dibangun adalah sistem *web* pemeliharaan mesin kecil industri menengah (IKM). *Web* ini dibangun dengan tujuan untuk membantu pengguna dalam melakukan pengambilan keputusan pada proses perawatan dan pemeliharaan mesin pada IKM.

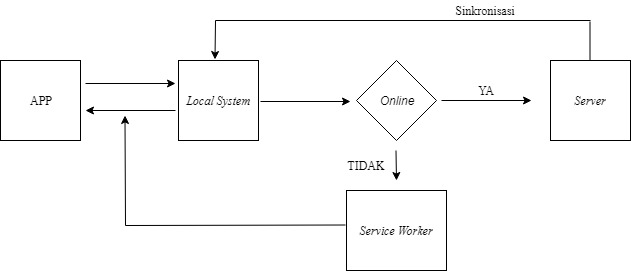
Sistem yang dibangun akan memberikan kemampuan dalam melakukan analisis keputusan pemeliharaan mesin pada IKM. sistem yang dibangun menerapkan teknologi DSS (*Desicion Support System*) untuk melakukan manajemen pemeliharaan mesin dengan model CMMS (*Computerized Maintenance Management System*), menerapkan konsep AI (*Artificial Intelligence* dan teknologi berbasis web. 

Gambar 3. 2. Proses Manajemen Pemeliharaan Mesin IKM

Gambar 3.2 Adalah penggambaran proses manajemen pemeliharaan mesin dengan menggunakan integrasi model pengambilan keputusan. Secara kuantitatif dan kualitatif data dari IKM dkan dicatat dan disimpan, dilakukan ekstraksi dan dianalisis menjadi keputusan pemeliharaan yang tepat membuat model yaitu DMG (*Decision Making Grid*) dan AHP (*Analytic Hierarchy Process*). DMG dan AHP digunakan oleh DSS untuk melakukan penyederhanaan proses analisis dan mengurangi waktu yang diperlukan dalam membuat keputusan perawatan. Hasil keputusan perawatan kemudian dapat dicoba yang memberikan saran perawatan yang lebih baik yang dapat dilakukan oleh IKM.

Kemudian digunakan algoritma *fuzzy logic*, algoritma genetika dan NNBP (*Neural Network Back Propagation*) yang disematkan untuk meningkatkan metode DMG yang memiliki beberapa kelemahan pada tahap implementasi pada IKM. Algoritma *Fuzzy Logic* kemudian dikombinasikan oleh DMG. Hasilnya model dapat mengirimkan data kuantitatif yang telah diinput untuk memberikan keputusan yang lebih baik. Kemudian algoritma genetika diterapkan oleh DMG untuk memberikan kepercayaan terhadap IKM terkait atau IKM lainnya yang sama untuk memanfaatkan DSS walaupun tidak memiliki data. Metode NNBP digunakan untuk melakukan prediksi keputusan terbaik yang mungkin diambil dalam melakukan manajemen pemeliharaan mesin IKM berdasarkan metode DMG. Namun dalam penelitian ini sistem yang dibangun hanya mencakup bagian melakukan pemasukan data kuantitatif atau kualitatif untuk keperluan analisis performansi sistem .yaitu dengan membuat sebuah sistem yang dapat menambahkan data mesin sederhana.

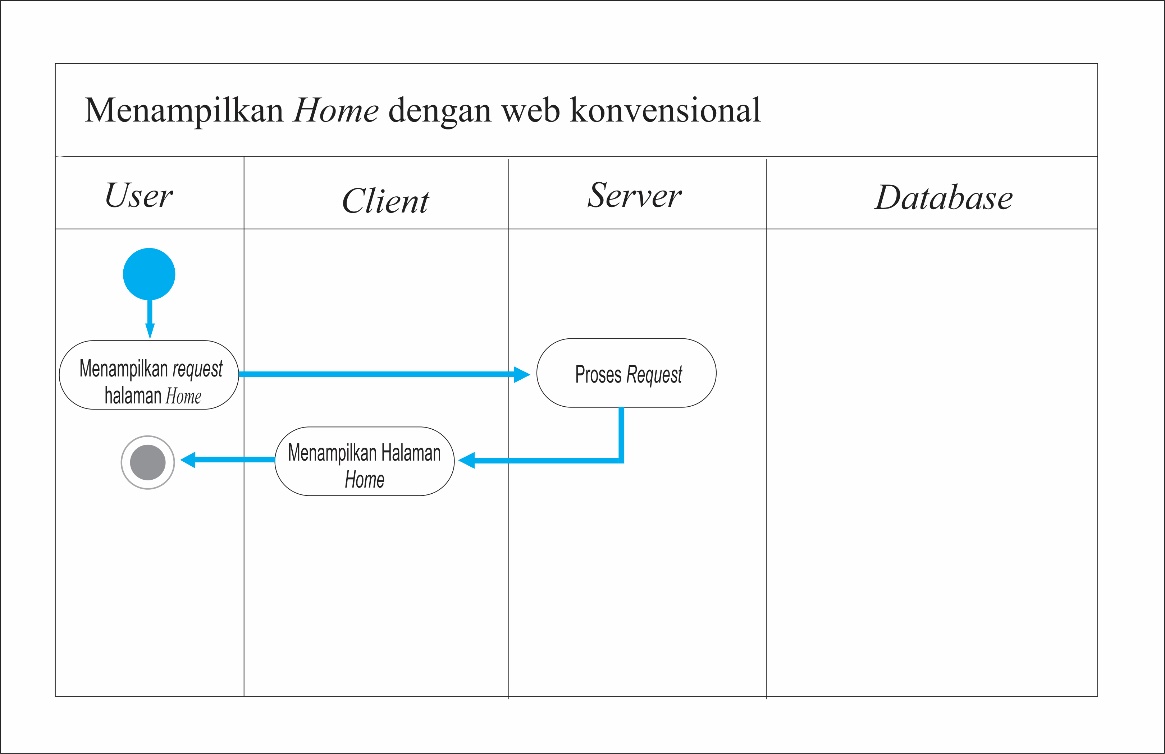
DSS yang memanfaatkan teknologi web memiliki masalah klasik yang sama yaitu tidak dapat di jalankan jika terputus dari server. Sehingga digunakan teknologi web dengan menggunakan model *offline first app* dengan memanfaatkan *service worker.* Sistem *offline first app* akan dibuat menggunakan library reactJS basis data lokal IndexedDB dan server basis data couchDB. sistem akan dibuat untuk mampu bekerja secara *offline* maupun *online*. Dengan menggunakan basis data lokal indexedDB untuk menangani manajemen basis data secara lokal atau *offline*, dan CouchDB untuk menangani manajemen basis data secara *online*. Kedua basis data ini akan melakukan sinkronisasi secara berkala selama koneksi internet ada.

Gambar 3. 3. Blok Diagram Sistem

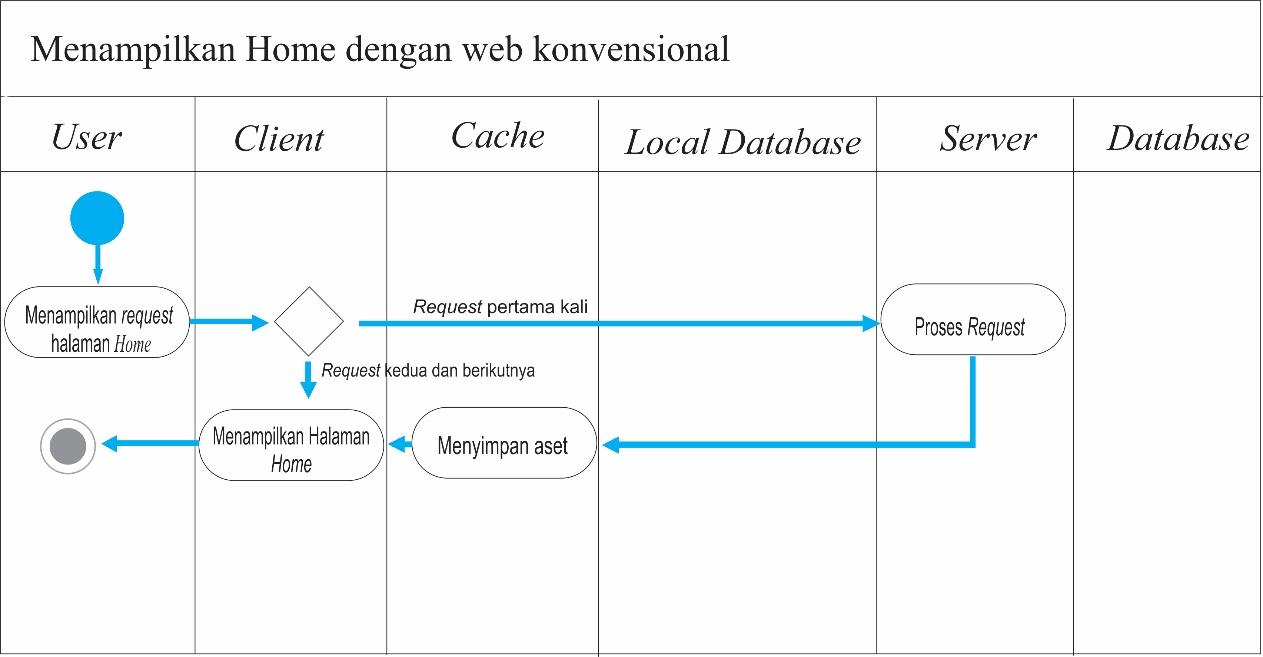
Sistem yang dibuat memiliki alur seperti pada gambar 3.2. Aplikasi akan mengirimkan *request* namun *request* hanya akan diterima oleh lokal sistem yaitu sistem yang berjalan pada sisi *client* dengan menggunakan media penyimpanan local indexedDB untuk pertama kali. selanjutnya sistem akan melakukan pengecekan koneksi internet, ketika koneksi internet tersedia maka *request* kemudian akan diteruskan menuju server basis data yaitu CouchDB yang di simpan pada server terpusat. Kemudian akan dilakukan sinkronisasi antara data yang ada pada lokal basis data dengan data yang ada pada server basis data selama koneksi internet tersedia, kemudian hasil *request* akan dikembalikan menuju aplikasi. Namun apabila dalam kasus koneksi internet tidak tersedia maka kemudian *service worker* berperan penting pada kondisi ini. *Service worker* akan aktif dan akan memberikan nilai kembalian berupa asset data yang sebelumnya telah di *cache* pada saat sistem pertama kali dijalankan dan hanya mampu mengakses lokal basis data dan dikembalikan menuju aplikasi.

1. ***System Activity***

*System Activity* dijelaskan melalui *Activity* Diagram. Activity diagram mengandung diagram use case entities seperti halaman *home*, halaman mesin, dan halaman pemeliharaan. Konsep pertama menggambarkan sistem apabila dibuat dengan cara yang biasa tanpa menggunakan *Service worker*, dan konsep kedua dibuat dengan menggunakan *Service worker.*



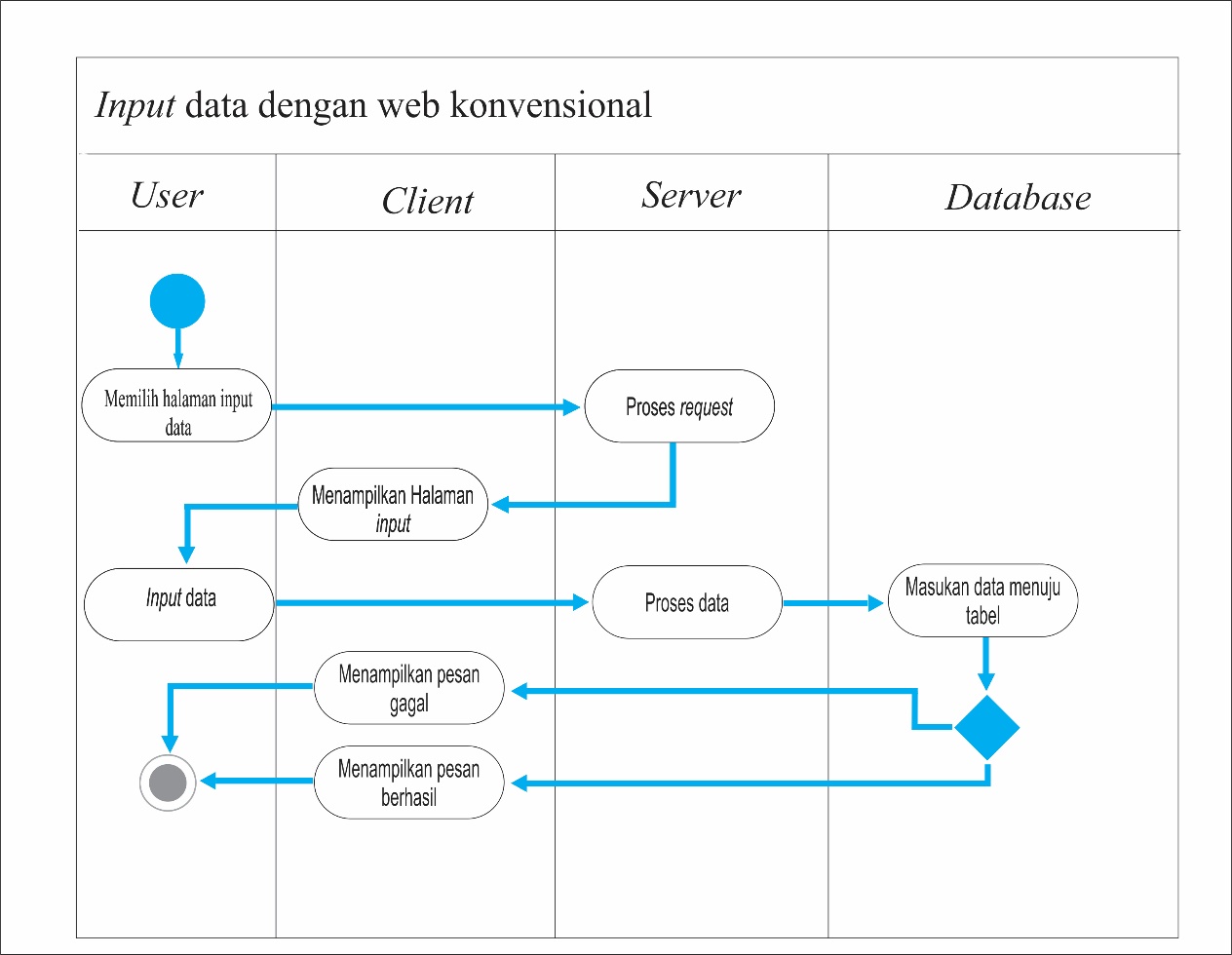
Gambar 3. 3. Diagram activity web konvensional .



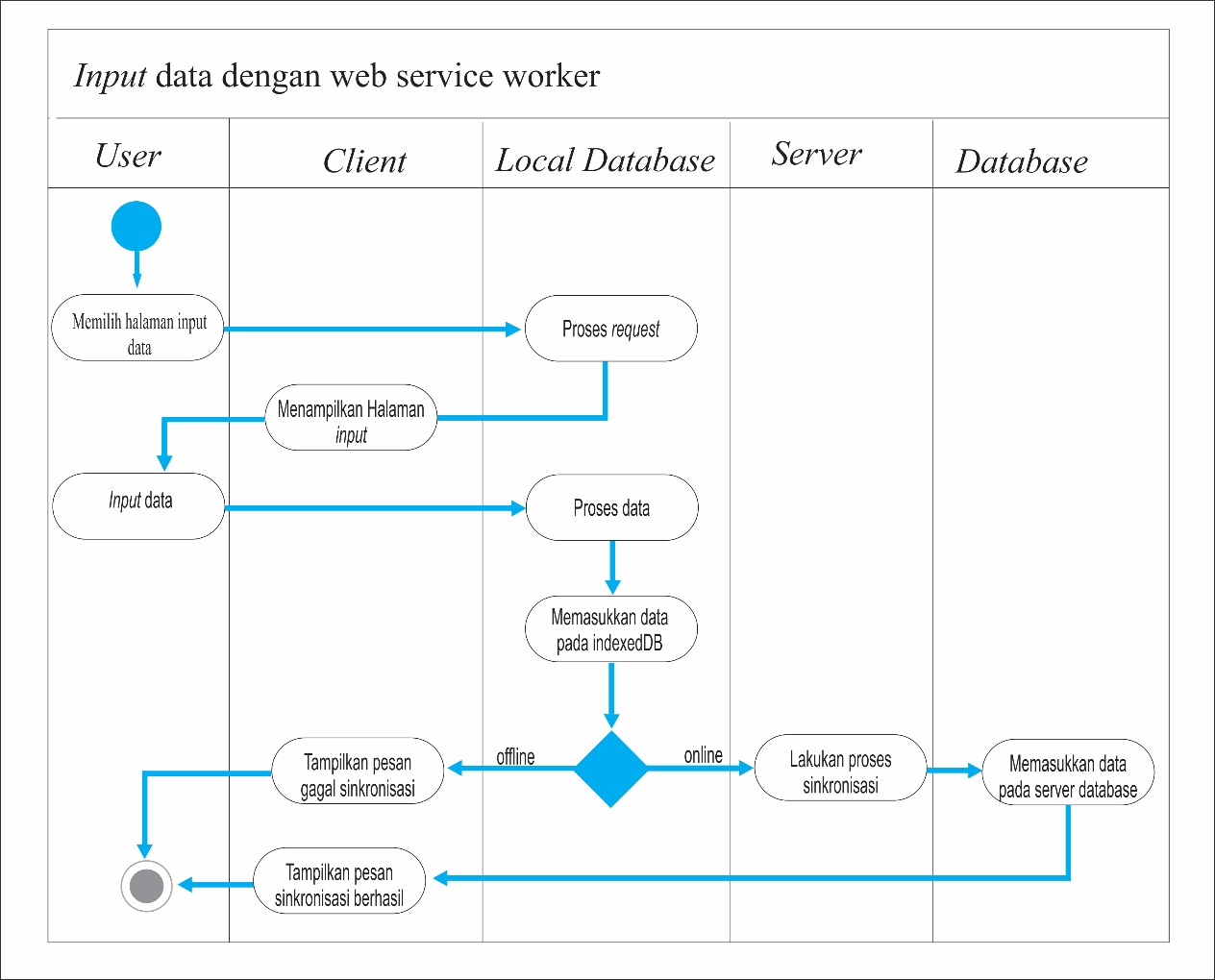
Gambar 3. 4. Diagram activity web dengan service worker

Pada gambar 3.4 *request* yang dilakukan oleh user kemudian akan dikirim dengan kondisi apabila *request* dilakukan pertama kali maka *request* akan menuju ke server dan akan menyimpan asset menuju *lokal* basis data dan *cache* kemudian menampilkan halaman *request* kembali ke pengguna. Namun ketika *request* sudah dilakukan sebelumnya maka *request* akan langsung dikembalikan ke pengguna dan menampilkan hasil *request*.

Selain itu pada gambar 3.5 dan 3.6 menunjukkan diagram activity sistem pada saat melakukan masukan data menuju basis data.



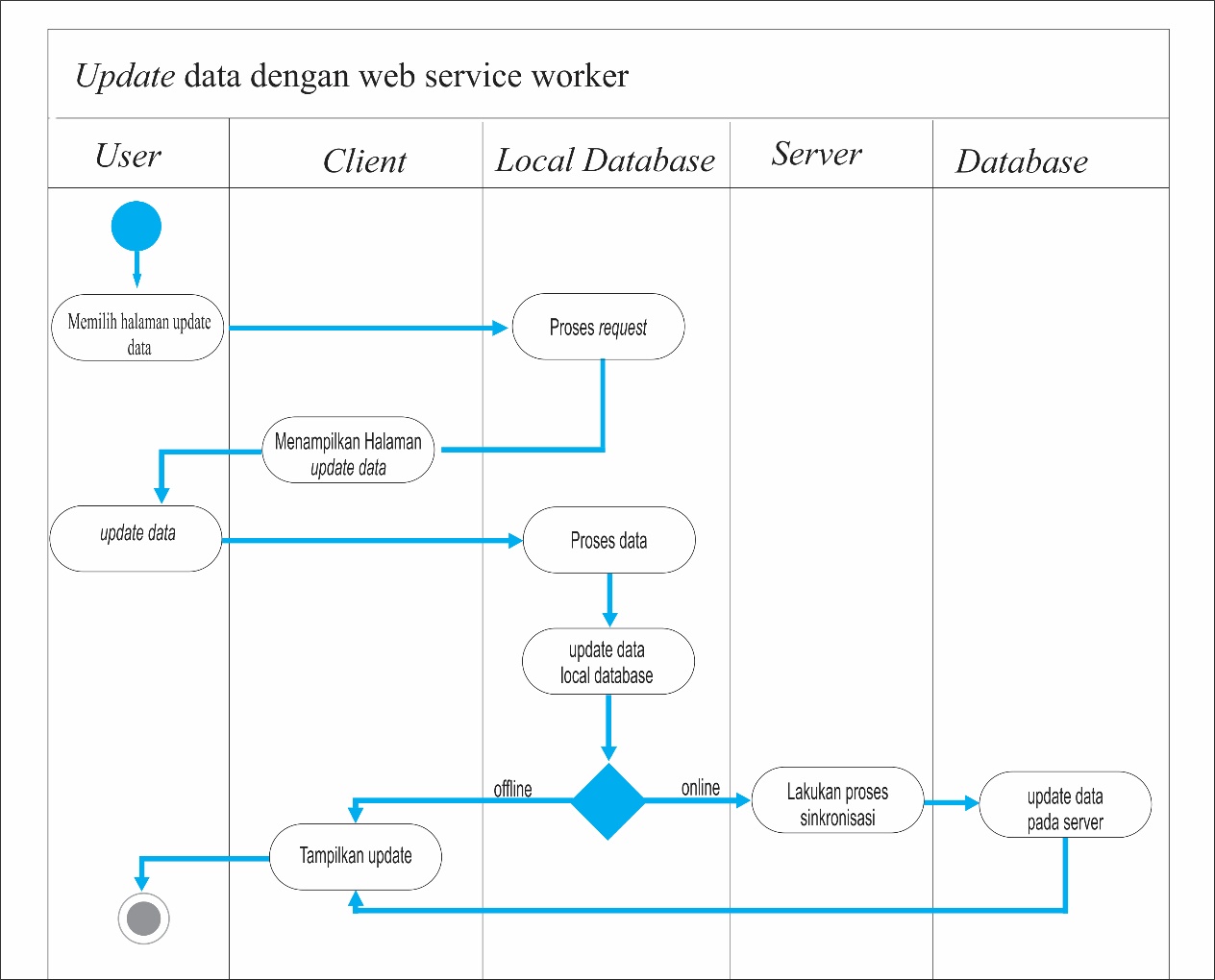
Gambar 3. 5. Diagram activity input data pada web konvensional

Pada *web* konvensional, data dimasukkan menuju server basis data secara langsung dan diproses melalui server. Pengguna terlebih dahulu melakukan *request* halaman *input* data, kemudian server mengirim halaman *input* dari server menuju *client*. Kemudian pengguna mampu memasukkan data dan disimpan menuju basis data server. 

Gambar 3. 6. Diagram activity input data pada web dengan service worker

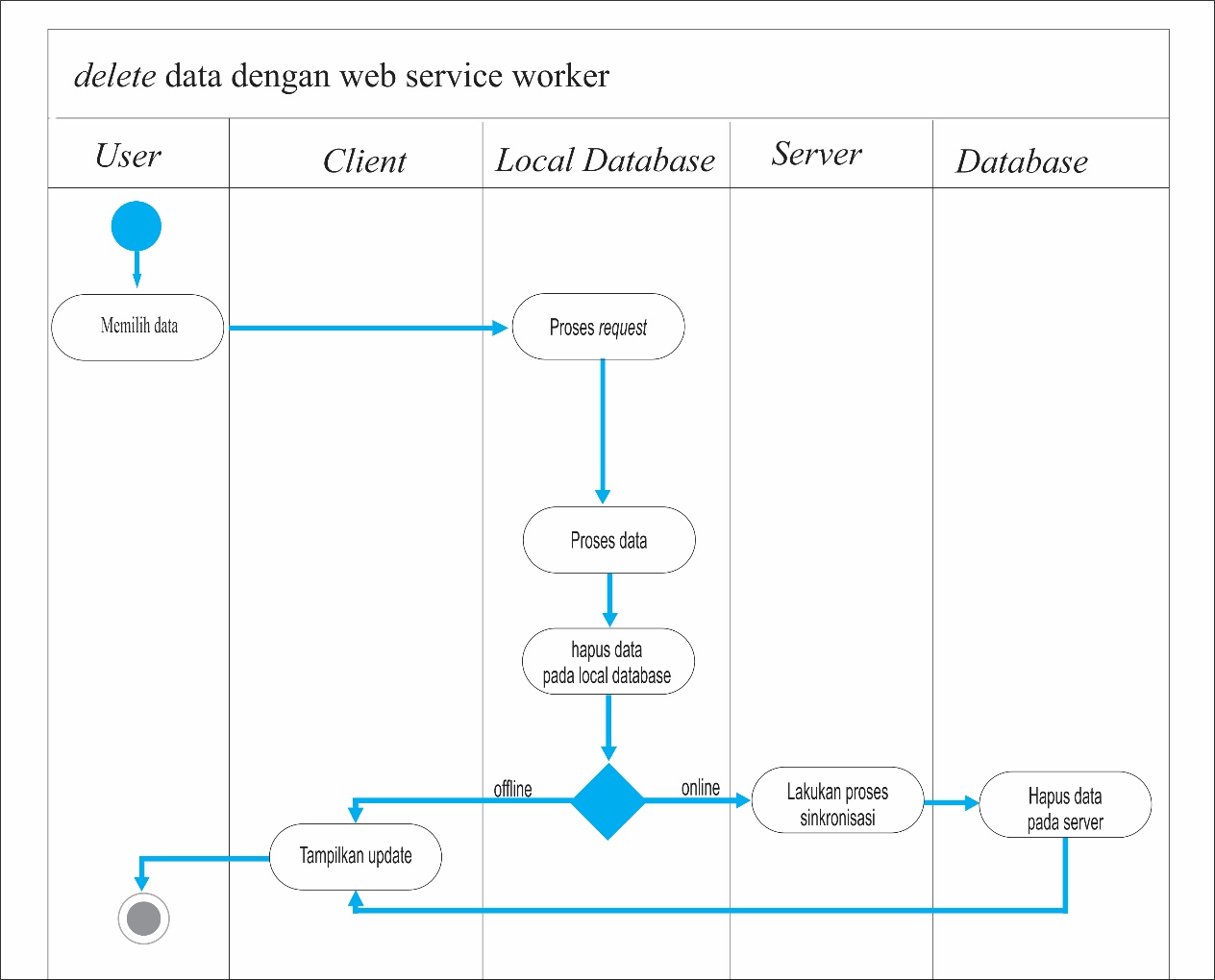
Diagram activity yang ditunjukkan pada gambar 3.6 menunjukkan alur sistem pemasukan data pada *web* dengan menggunakan *service worker* dan menggunakan metode *offline first app*. Data yang dimasukkan pertama kali menuju lokal basis data kemudian akan dilakukan proses pengecekan, ketika dalam kondisi *online* maka akan dilakukan proses sinkronisasi data ke server. Kemudian sistem akan menampilkan pesan bahwa sistem berhasil melakukan sinkronisasi, sebaliknya jika sistem *offline* maka akan ditampilkan pesan gagal sinkronisasi namun data tetap di tampilkan yang diambil melalui lokal basis data.

Untuk proses pembaharuan data pada web dengan menggunakan *service worker* data yang mengalami pembaharuan terlebih dahulu akan diproses pada sisi *client* dan akan merubah data yang sebelumnya pada basis data lokal. Ketika sistem dalam kondisi *online* maka secara otomatis akan dilakukan proses sinkronisasi data menuju server basis data. Begitu pula sebaliknya ketika dalam kondisi *offline* maka proses pembaharuan data akan selesai. Sistem kemudian akan melakukan sinkronisasi apabila sistem tehubung dengan internet.

Proses pembaharuan data ditunjukkan melalui diagram *activity* pada gambar 3.7. 

Gambar 3. 7. Diagram activity update data pada web dengan service worker

Sedangkan ketika sistem melakukan hapus data maka sistem terlebih dahulu akan melakukannya pada sisi *client* dan menghilangkan data pada basis data lokal. Apabila koneksi internet tersedia maka sistem secara lansung akan melakukan sinkronisasi data pada server basis data. Namun apabila sistem dalam kondisi *offline* maka sistem akan selesai kemudian menghentikan proses dan menunggu sampai tersedianya koneksi internet untuk melakukan sinkronisasi data. Proses hapus data ditunjukkan melalui diagram *activity* pada gambar 3.8



Gambar 3. 8. Proses hapus data pada web dengan menggunakan service worker

***Hardware Environtment***

Dibawah ini adalah *hardware environtment* yang akan digunakan dalam proses penelitian :



Gambar 3. 8. Hardware Environtment

Berdasarkan gambar 3.3, dapat dilihat bahwa terdapat sebuah PC yang terhubung dengan router kemudian terhubung ke sebuah server. Sistem akan disimpan pada server terpusat bersama dengan basis data couchDB. router jenis Asus *wireless AC3100 Gigabit* digunakan untuk mejadi perantara antara PC dan server. Perancangan Sistem

Perancangan sistem *Web* Pemeliharaan Mesin Industri Kecil Menengah ,terdiri dari tiga tahap perancangan yaitu Perancangan Sistem *web*, Pemasangan *service worker*, pembuatan basis data lokal dan server.

1. Perancangan Sistem *Web*

Tahap perancangan sistem *web* dilakukan dengan menggunakan :

* + - 1. Node JS ( sebagai library)
      2. Library ReactJS (sebagai *front-end*)
      3. Material UI (digunakan untuk mebangun tampilan sistem)
      4. RxDB (Sebagai back-end)

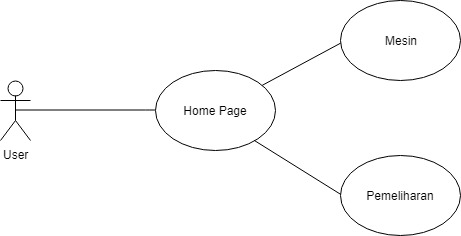
Sistem dibangun dengan membuat tampilan sistem dinamis dengan menggunakan ReactJS dan Material UI untuk membuat tampilan sistem lebih bagus.

1. Pemasangan *Service worker*

*Service worker* berperan untuk membuat sistem yang dibuat memiliki akses yang lebih cepat dan dapat melakukan manajemen *cache*, sehingga sistem mampu diakses walaupun tidak memiliki koneksi internet dan dapat menerapkan teknologi *offline first app*. Pemasangan *service worker* dilakukan dengan melakukan register *service worker* pada sistem yang dibuat.

1. Pembuatan basis data lokal dan server

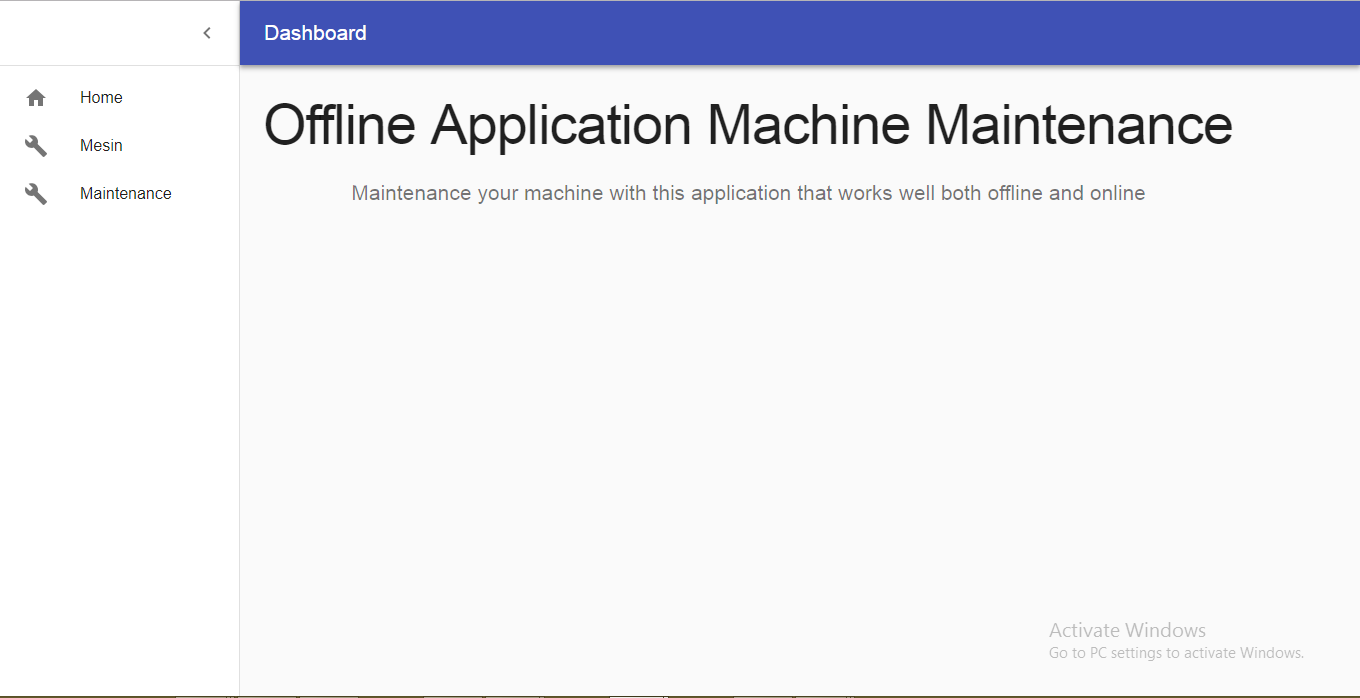
Basis data digunakan untuk melakukan penyimpanan data masukan dari pengguna. Sistem menggunakan dua jenis basis data yaitu lokal basis data dan server basis data. Lokal basis data yaitu indexedDB digunakan sebagai media penyimpanan utama, sehingga sistem dapat mendukung teknologi *offline first app*. Sedangkan server basis data menggunakan couchDB yang akan melakukan sinkronisasi secara berkala setiap kali sistem memiliki koneksi internet.

Sistem pemeliharaan mesin industry kecil menengah memiliki dua fitur yaitu manajemen mesin, dan manajemen pemeliharaan mesin. Scenario dari sistem yang dibuat ditunjukkan menggunakan use case diagram pada gambar 3.3 . Skenario yang ada menjelaskan bagaimana pengguna dalam menggunakan sistem. 

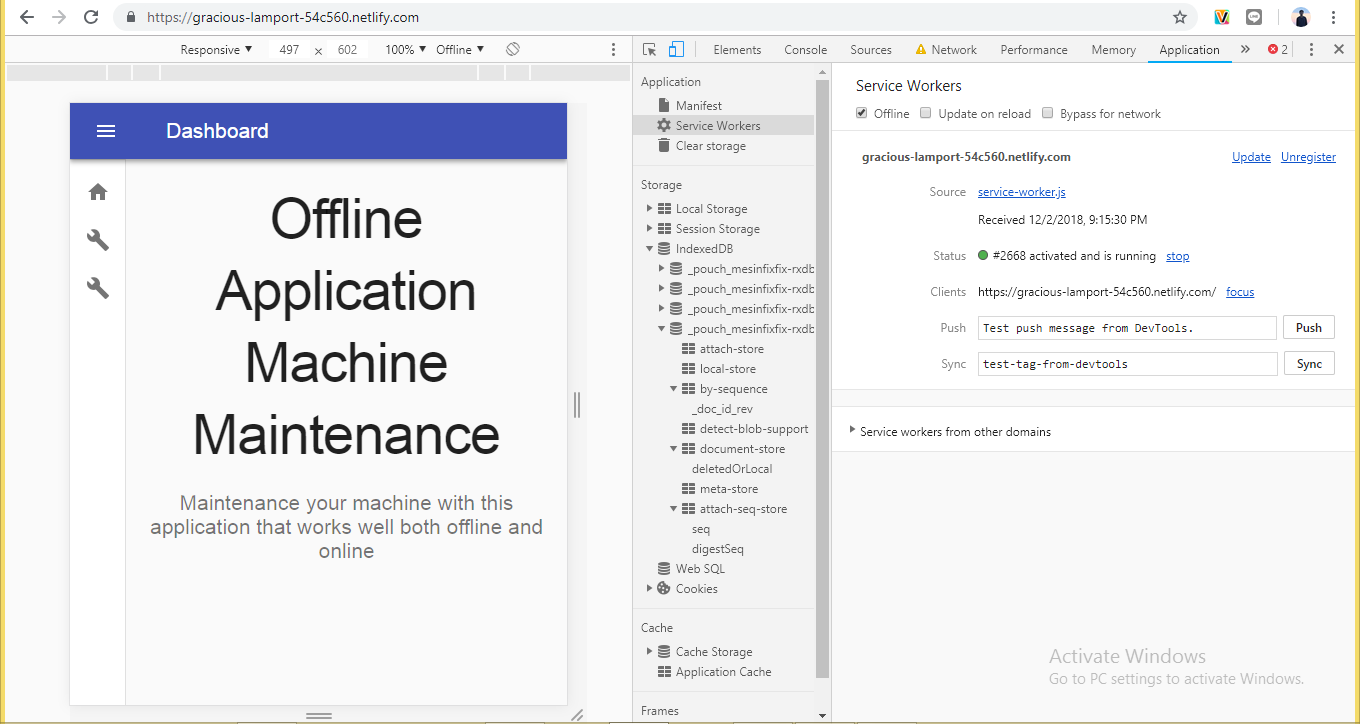
Gambar 3. 9. Use case diagram

Use case dari halaman home berinteraks langsung dengan pengguna. Halaman didesain untuk menampilkan link keseluruh halaman *web* yang lain. Skenario Pengujian

1. **Hasil Pembuatan Sistem**

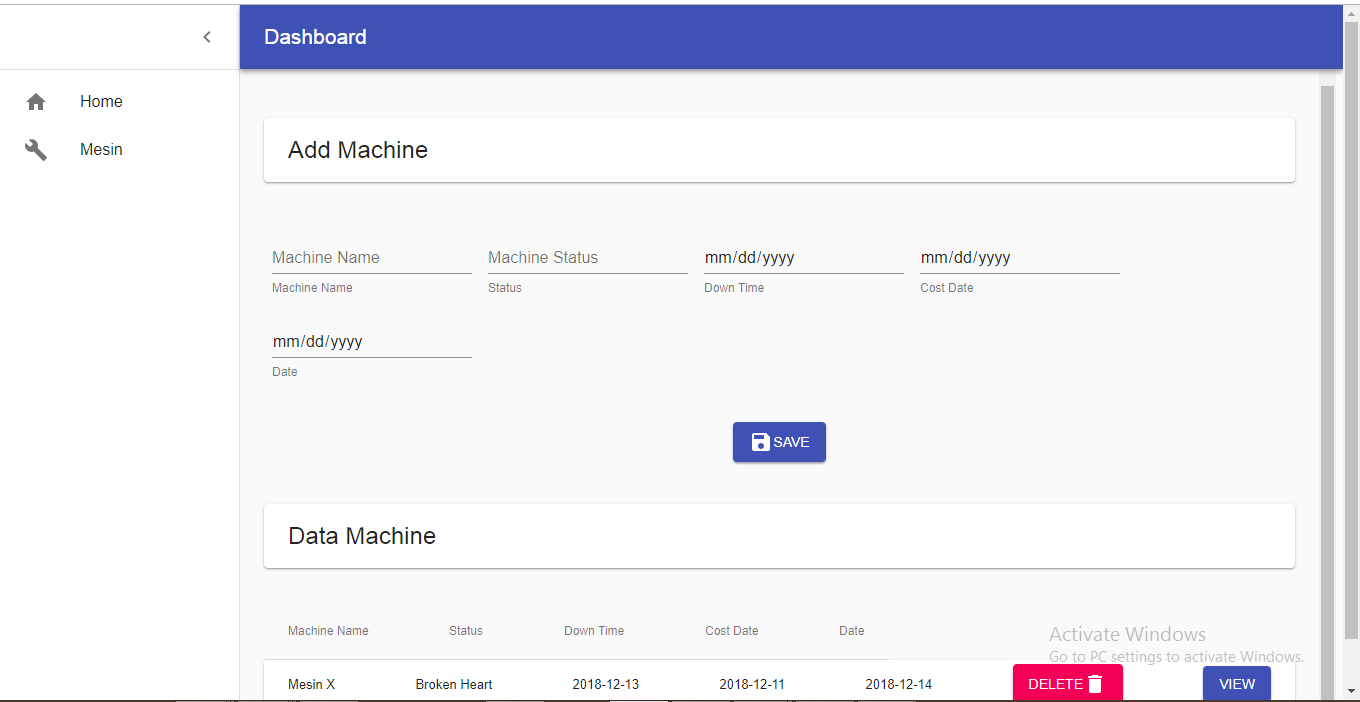
Setelah semua perancangan sistem telah dilakukan maka selanjutnya adalah membangun sistem yang akan dianalisis. Sistem dibangun dengan menggunakan *library reactJS* sebagai sisi client dan menggunakan RxDB untuk melakukan transfer data menuju basis data lokal maupun server basis data. 

Gambar 3.10. Halaman utama sistem



Gambar 3. 11. Service worker pada aplikasi web

Gambar 3.9 menunjukkan halaman utama sistem yang telah dibuat menggunakan library *reactJS* dan telah terintegrasi dengan *service worker* yag ditunjukkan pada gambar 3.10.Sistem dibuat dengan menerapkan teknologi *single page application* (SPA) dan memiliki dua halaman (Mesin dan *Maintenance*) dengan fitur untuk melakukan penambahan data, pembaharuan data, dan penghapusan data. Fitur yang ada bekerja dengan *online* maupun *offline*, ada atau tidaknya koneksi internet aplikasi akan tetap dapat bekerja. Sehingga memasukkan data, melakukan pembaharuan data, dan menghapus data dapat dilakukan tanpa memerlukan koneksi internet sekalipun.



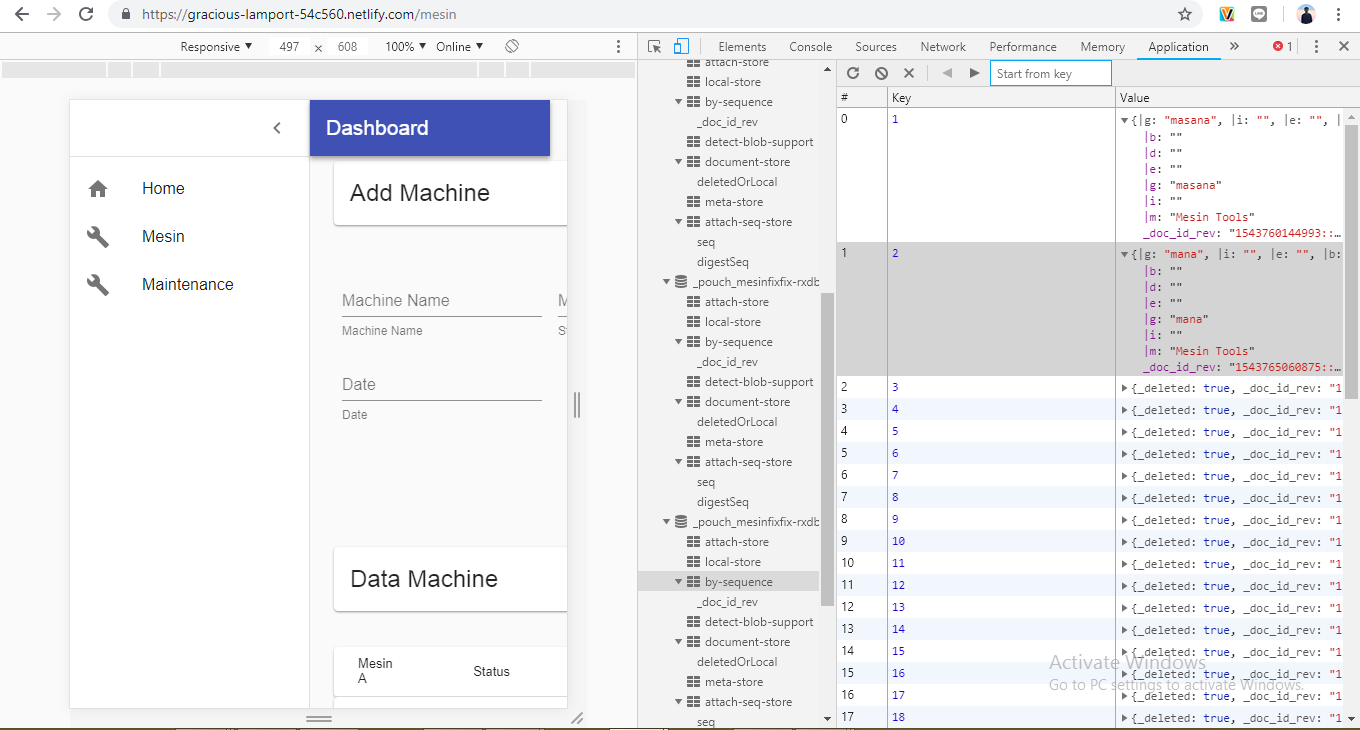
Gambar 3. 12. Halaman Mesin

Sistem yang dibangun menggunakan format basis data nosql sesuai dengan tabel 3.1.

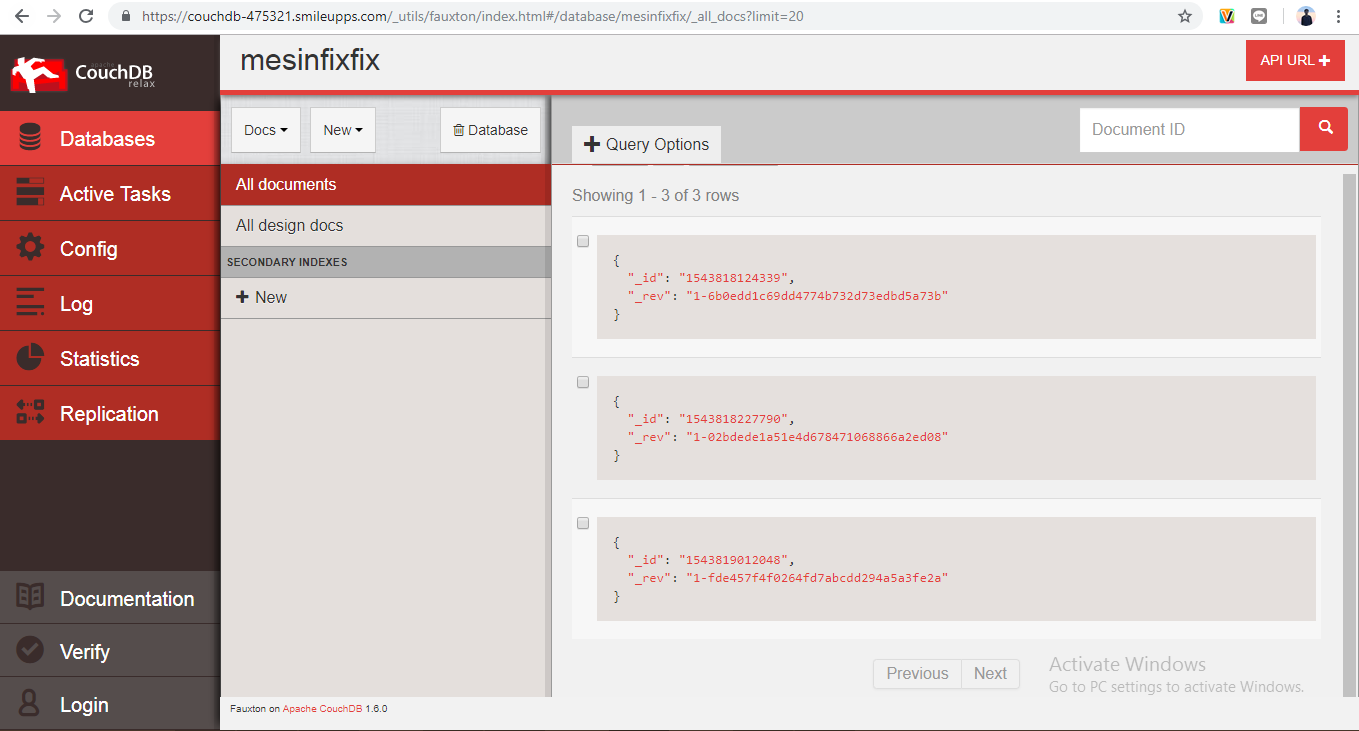
Tabel 3. 1. Tabel format basis data

|  |  |  |  |
| --- | --- | --- | --- |
| Nama Elemen Data | Deskripsi | Sumber | Jenis Field |
| id | Pengidentifikasi yang unik untuk setiap mesin | Dibuat secara otomatis oleh sistem | string |
| nama\_mesin | Nama mesin yang di masukkan | Di masukkan oleh *user* | string |
| status | Status mesin yang dimasukkan | Di masukkan oleh *user* | string |
| down\_time | Tanggal dan waktu mesin mengalami kerusakan | Di masukkan oleh *user* | string |
| cost | Biaya yang digunakan untuk melakukan perbaikan mesin | Di masukkan oleh *user* | string |
| date | Waktu data dimasukkan | Dibuat secara otomatis oleh sistem | string |

Basis data yang digunakan untuk lokal adalah basis data indexedDB dengan menerapkan *document objek* dan couch db untuk basis data server yang keduanya akan melakukan sinkronisasi secara berkala selama jaringan internet tersedia.



Gambar 3. 13. Basis data lokal indexedDB



Gambar 3. 14. Basis data server couch db

1. **Skenario Pengujian**

Pengujian dilakukan dengan menggunakan parameter sebagai berikut :

* + 1. ***Respon Time***

*Respon time* dari kedua sistem yaitu *web* konvensional dan *web service worker* akan dihitung. Aplikasi apache jmeter akan digunakan dalam melakukan pengujian respon time. Jmeter akan memberikan nilai interval waktu pada saat *request* dilakukan dan pada saat respon diterima.

Pengujian respon time akan dilakukan dengan memberikan banyak *request* sekaligus dari *client* menuju server masing-masing *thread* sebanyak sepuluh kali kemudian dilakukan perhitungan nilai rata – rata respon time atau *average time.*

*Average time* adala nilai jumlah dari serangkaian waktu antara *request* dibagi jumlah *request* dari *client* ke server.

: Jumlah *request* dari *client* ke server.

: *counter number* ( = 1,2, …., )

: interval waktu tempuh anatara *client* ke server setiap *request*

(3. 1)

* + 1. ***Throughput***

*Throughput* adalah salah satu indikator yang dapat digunakan untuk melakukan analisa performansi. Throughput merupakan jumlah total transaksi yang dapat dilakukan dalam waktu tertentu atau TPS (*transaction per second*).

Dalam melakukan pengujian *throughput* diberikan *thread* sampai dengan 450 *thread*. Setiap *thread* dilakukan pengujian sebanyak sepuluh kali secara berulang dan mengambil nilai *throughput* masing – masing percobaan. Kemudian dilakukan perhitungan terhadap nilai rata – rata dari hasil *throughput* setiap percobaanuntuk setiap *thread.*

Adapun cara perhitungan *throughput* dilakukan dengan menggunakan persamaan berikut.

Throughput = ( 3. 2)

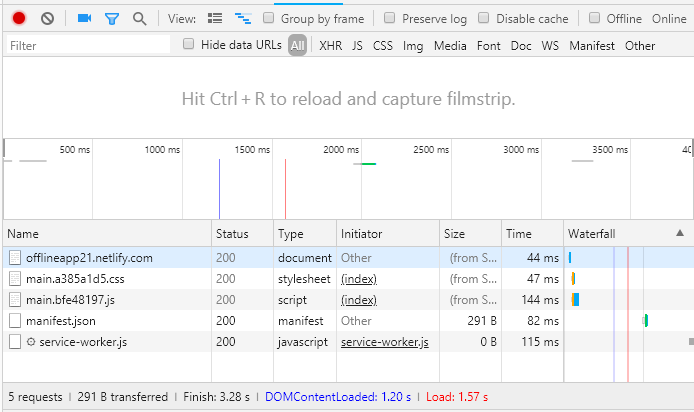
Persamaan 3.3 merupakan persamaan yang digunakan untuk menghitung throughput untuk satu *request* saja. Untuk *request* yang berulang atau lebih dari satu maka digunakan persamaan berikut.

Throughput = (3. 3)

* + 1. **Proses *Browser***

Dengan menggunakan *dev-tools* google chrome kita mampu mengukur waktu load sumber daya *website*. Chrome akan memberikan daftar sumber daya yang di akses dari server seperti *Java script*,HTML, dan CSS kemudian akan memberikan waktu akses masing masing sumber daya dengan model grafik *timeline*. Waktu yang digunakan untuk melakukan penarikan data dari server menuju *browser* akan muncul dalam bentuk angka yang disebut *DOMContentLoaded*. Kemudian waktu yang digunakan untuk halaman *web* berhasil melakukan pengambilan data yang ada secara keseluruhan dari server menuju *browser* juga akan ditampilakn dalam bentuk angka yang disebut dengan *Load.*

Untuk proses browser pengujian dilakukan dengan mengukur waktu *load* untuk *web* dengan *service* worker dan *web* konvensional dengan melakukan akses masing masing *web* sebanyak sepuluh kali akses kemudian mengambil nilai rata – rata waktu akses.



Gambar 3. 15. Network Panel Chrome yang digunakan mengukur waktu akses.

* + 1. **Prosesor**

Performansi prosesor dari server akan dilakukan pengujian dengan menggunakan *windows performance* *monitor* dalam bentuk persen (%). Angka yang dihasilkan adalah angka waktu yang digunakan prosesor dalam melakukan eksekusi *request* kemudian mengurangi nilai yang ada dengan 100%. Pengujian prosesor dilakukan dengan melakukan akses kepada server dengan beberapa jumlah *thread* sekaligus dengan menggunakan Jmeter.

Proses pengujian dilakukan dengan menggunakan beberapa thread pada apache jmeter dengan interval thread antara satu sampai seribu sesuai dengan kondisi pengujian yang dilakukan.

# **BAB IV HASIL DAN PEMBAHASAN**

1. **Pengujian *Respon Time***

Hasil pengujian pada *web* biasa dan *web* *offline first app* secara *online* dengan menggunakan software apache Jmeter akan diperoleh hasil untuk masing masing percobaan dengan waktu rata – rata yang ditunjukkan pada tabel 4.1.

Tabel 4. 1. Hasil Pengujian Average Secara Online

|  |  |  |  |
| --- | --- | --- | --- |
| Pengujian | Jumlah *Thread* | *Web* Konvensional (detik) | *Web* *offline first app*  (detik) |
| 1 | 1 | 131 | 26.2 |
| 2 | 50 | 1261.552 | 3701.342 |
| 3 | 100 | 3152.708 | 3714.128 |
| 4 | 150 | 2208.533 | 4904.058 |
| 5 | 200 | 2219.373 | 2890.123 |
| 6 | 250 | 2668.27 | 1954.679 |
| 7 | 300 | 2949.699 | 3402.078 |
| 8 | 350 | 5769.697 | 3900.643 |
| 9 | 400 | 6690.479 | 2744.229 |
| 10 | 450 | 5454.258 | 1856.495 |
| 11 | 600 | 3364.905 | 3849.139 |
| 12 | 700 | 8352.743 | 7597.123 |
| 13 | 800 | 13561.76 | 5033.896 |
| 14 | 900 | 15407.66 | 1784.489 |
| 15 | 1000 | 15037.41 | 1179.45 |

Sedangkan untuk grafik yang diperoleh berdasarkan data yang ada ditunjukkan pada gambar 4.1.

Gambar 4. 1 Grafik hasil pengujian waktu rata – rata

Berdasarkan grafik yang diperoleh dari masing masing *web* untuk waktu rata – rata, menunjukkan bahwa waktu akses rata rata pada sebuah *web* konvensional untuk satu *thread* menghasilkan angka yang lebih besar dibandingkan dengan *web service worker*. Hal ini dikarenakan *web service worker* memiliki waktu akses yang cepat.

Untuk *web service worker* dengan *thread* 1000 menghasilkan waktu rata – rata yang lebih rendah dibandingkan dengan *web* konvensional. Hal ini membuktikan bahwa *web service worker* dapat mengatasi jumlah *request* yang besar secara bersamaan namun tetap memiliki waktu akses yang rendah. Pada *web* konvensional pada saat pengujian dengan menggunakan 900 *thread* diberikan, waktu akses rata – rata yang diperoleh kemudian menurun. Hal ini disebabkan karena banyaknya *request* yang akhirnya gagal dilakukan dalam waktu yang bersamaan. Berbanding terbalik dengan *web* yang menggunakan *service worker* yang tetap memiliki waktu akses rata – rata yang lebih rendah dan tetap mampu menangani *request* yang banyak tanpa ada *request* yang gagal.

1. ***Throughput***

Hasil Pengujian dari *throughput* pada *web* konvensional dan *web* *offline first app* ditunjukkan pada tabel 4.2 dan grafik pada gambar :

Gambar 4. 2 Tabel Hasil Pengujian Throughput pada web Online

|  |  |  |  |
| --- | --- | --- | --- |
| Pengujian | Jumlah *Thread* | *Web* Konvensional (/sec) | *Web* *offline first app* (/sec) |
| 1 | 1 | 0.007651 | 37,6 |
| 2 | 50 | 19.09 | 35.45 |
| 3 | 100 | 31.9 | 61.56 |
| 4 | 150 | 32.7 | 62.7 |
| 5 | 200 | 82 | 106.35 |
| 6 | 250 | 58.94 | 148.3 |
| 7 | 300 | 63.13333 | 126.59 |
| 8 | 350 | 43.74 | 115.16 |
| 9 | 400 | 42.74 | 178.0767 |
| 10 | 450 | 54.07 | 195.39 |

Gambar 4. 3 Grafik hasil throughput

Dari grafik yang diperoleh dari hasil pengujian untuk *throughput* menunjukkan bahwa untuk *website* konvensional memiliki throughput yang lebih rendah dibandingkan dengan *website* dengan menggunakan *service worker.* Hal ini menunjukkan bahwa *website* dengan menggunakan *service worker* mampu menghasilkan throughput yang lebih tinggi, sehingga mampu melakukan proses *request* yang lebih besar dalam satuan waktu.

1. **Proses *Browser***
   * 1. **. Waktu akses secara *online***

Hasil Pengujian dari pemrosesan *browser* pada *web* konvensional dan *web* *offline first app* ditunjukkan dengan tabel dan grafik. dengan menggunakan dua parameter yaitu *DOMContentLoaded* dan *Load*. pengujian dilakukan dengan melakukan akses *web* tanpa *cache* sebanyak sepuluh kali kemudian mengambil nilai rata – rata waktu akses *browser* untuk masing masing *web*.

Tabel 4. 2 Tabel hasil pengujian akses pertama kali tanpa cache

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Percobaan | *Web* konvensional | | *Web service worker* | |
| *DomContentLoaded* (detik) | *Load* (detik) | *DomContentLoaded* (detik) | *Load* (detik) |
| 1 | 1.11 | 1.904 | 3.05 | 2.607 |
| 2 | 1.25 | 0.567 | 3.03 | 0.452 |
| 3 | 1.62 | 0.679 | 2.83 | 0.463 |
| 4 | 1.19 | 0.665 | 2.05 | 0.548 |
| 5 | 0.506 | 0.558 | 2.26 | 0.441 |
| 6 | 0.494 | 0.698 | 2.14 | 0.484 |
| 7 | 0.458 | 0.634 | 2.76 | 0.496 |
| 8 | 0.555 | 0.672 | 2.85 | 0.459 |
| 9 | 0.508 | 0.553 | 2.92 | 0.497 |
| 10 | 0.568 | 0.723 | 2.12 | 0.445 |

Setelah dilakukan percobaan untuk melakukan akses web pertama kali sebanyak sepuluh percobaan, kemudian dilakukan load web dengan membiarkan cache yang ada pada setiap browser sebanyak sepuluh kali.

Tabel 4. 3 Tabel pengujian online dengan cache

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Percobaan | *Web* konvensional | | *Web service worker* | |
| *DomContentLoaded* | *Load* | *DomContentLoaded* | *Load* |
| 1 | 0.492 | 0.567 | 0.433 | 0.452 |
| 2 | 0.578 | 0.679 | 0.445 | 0.463 |
| 3 | 0.574 | 0.665 | 0.537 | 0.548 |
| 4 | 0.52 | 0.558 | 0.428 | 0.441 |
| 5 | 0.624 | 0.698 | 0.474 | 0.484 |
| 6 | 0.554 | 0.634 | 0.478 | 0.496 |
| 7 | 0.588 | 0.672 | 0.449 | 0.459 |
| 8 | 0.497 | 0.553 | 0.482 | 0.497 |
| 9 | 0.642 | 0.723 | 0.435 | 0.445 |
| 10 | 0.59 | 0.654 | 0.458 | 0.469 |

Gambar 4. 4. Grafik prowses browser untuk waktu load

Gambar 4. 5. Grafik proses browser DOMContentLoaded

Dari data yang diperoleh untuk waktu rata – rata load pada saat *website* pertama kali di akses akan menghasilkan waktu yang tinggi, baik *web* konvensional maupun *web service worker*. Namun untuk akses berikutnya akan memberikan nilai yang secara signifikan sangat rendah dibandingkan dengan waktu akses pertama kali. Ini disebabkan karena *browser* telah melakukan *cache* pada saat akses pertama kali, sehingga untuk akses berikutnya data akan di ambil melalui penyimpanan *cache*.

Untuk waktu akses load pertama kali *web service worker* menghasilkan nilai waktu load yang tinggi dibandingkan dengan *web* konvensional. Hal ini dikarenakan *web service worker* menggunakan metode *single page application* sehingga semua asset dan sumber daya yang ada akan diakses dan digunakan ketika diakses pertama kali, seperti akses data dari penyimpanan basis data sehingga untuk akses ke halaman lain waktu akses yang digunakan hanya membutuhkan waktu yang sangat rendah dan untuk akses *web* berikutnya akan menghasilkan waktu yang sangat rendah sesuai dengan grafik yang ditunjukkan pada gambar 4.4. Sehingga berbeda dengan *web* konvensional ketika melakukan perpindahan halaman yang membutuhkan waktu yang cukup lebih lama dibandingkan *web* *service worker* walaupun waktu akses load pertama kali yang di peroleh lebih cepat dibandingkan dengan *web* *service worker*.

Walaupun menggunakan penyimpanan *cache* *browser* *web service worker* tetap memiliki waktu akses load yang lebih rendah untuk akses kedua dan berikutnya dibandingkan dengan *web* konvensional. Hal ini menandakan bahwa waktu akses *browser* untuk *web service worker* yang telah melakukan *cache* memiliki waktu akses yang lebih cepat dibandingkan dengan *web* konvensional.

* + 1. **Waktu akses secara *offline***

Untuk waktu akses secara *offline* dilakukan pengambilan data dimulai pada percobaan ke dua. Hal ini dikarenakan proses pertama dilakukan secara *online* untuk melakukan penyimpanan asset. Untuk proses pengambilan data waktu akses pertama dilakukan sebanyak sepuluh kali dan mengambil nilai rata – rata. Untuk pengaksesan secara *offline* pada *website* dengan menggunakan *service worker* diperoleh hasil *load* dan *DOMContentLoaded* pada tabel 4.7

Tabel 4. 4 Tabel pengujian offline dengan cache

|  |  |  |
| --- | --- | --- |
| No | *Web service worker* | |
| *DomContentLoaded* (detik) | *Load* (detik) |
| 1 | 1.25 | 1.82 |
| 2 | 1.02 | 1.46 |
| 3 | 1.06 | 1.47 |
| 4 | 1.03 | 1.41 |
| 5 | 1.04 | 1.6 |
| 6 | 1 | 1.4 |
| 7 | 1.1 | 1.55 |
| 8 | 1.12 | 1.54 |
| 9 | 0.935 | 1.27 |
| 10 | 1.12 | 1.53 |

Gambar 4. 6. Grafik proses browser web service worker

Berdasarkan percobaan perbandingan waktu akses *load*  pada *web service worker* pada kondisi *online* dan *offline* diperoleh hasil grafik yang ditunjukkan pada gambar 4.6. Untuk kondisi *online* *web service worker* untuk akses pertama kali menghasilkan waktu *load* yang tinggi. Pada proses *load* pertama semua file *asset* akan disimpan pada penyimpanan *cache* sehingga proses ini menggunakan waktu yang lama untuk proses *load* yang dilakukan. Proses berikutnya akan mendukung sistem tetap dapat diakses walaupun jaringan internet tidak tersedia. Namun untuk *load* berikutnya waktu yang digunakan akan lebih cepat, hal ini dikarenakan sistem akan melakukan penarikan data asset dari penyimpanan *cache* tidak pada penyimpanan server sehingga waktu yang diperlukan untuk mengirim *request* dan waktu umpan balik lebih cepat.

Untuk waktu akses *web* dalam kondisi offline waktu yang digunakan untuk melakukan *load data* akan lebih besar dibandingkan dengan pada saat kondisi *online*. Kemungkinan hal ini disebabkan karena *web service worker* dalam kondisi *offline* akan menjalankan fungsi - fungsi perintah *offline* yang ada pada *service worker* sehingga waktu akses yang digunakan akan lebih lama dibandingkan pada saat kondisi *online*.

* + 1. **Skenario *online* *input data***

Untuk skenario *input* data secara *online* digunakan dua indikator yang sama dengan waktu akses yaitu load dan *DOMContentLoaded,* sehingga diperoleh hasil waktu yang dibutuhkan aplikasi untuk melakukan komunikasi dan pemasukan data ke server basis data.

Tabel 4. 5 Tabel pengujian online input data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | *Web* konvensional | | *Web service worker* | |
| *DomContentLoaded* (detik) | *Load* | *DomContentLoaded* | *Load* |
| 1 | 0.493 | 0.909 | 0.727 | 0.923 |
| 2 | 0.145 | 0.268 | 0.727 | 0.923 |
| 3 | 0.328 | 0.625 | 0.727 | 0.923 |
| 4 | 0.514 | 0.937 | 0.727 | 0.923 |
| 5 | 0.516 | 0.967 | 0.727 | 0.923 |
| 6 | 0.537 | 1 | 0.727 | 0.923 |
| 7 | 0.521 | 0.996 | 0.727 | 0.923 |
| 8 | 0.489 | 0.922 | 0.727 | 0.923 |
| 9 | 0.528 | 0.00105 | 0.727 | 0.923 |
| 10 | 0.486 | 0.98 | 0.727 | 0.923 |

Gambar 4.7. Grafik waktu load input data

Dari hasil data yang diperoleh, waktu rata rata untuk *input* data pada *web* konvensional yaitu 0.76 detik. Sedangkan untuk waktu rata – rata dari waktu *input* pada *web* *service worker* yaitu 0.73 detik. Pada *web* *service worker* yang telah dilakukan percobaan *input* data sebanyak sepuluh kali, semua data yang diperoleh menunjukkan waktu yang sama yaitu 0.727 detik. Hal ini dikarenakan ketika *web service worker* melakukan proses pemasukan data, waktu Load yang ada tidak mengalami perubahan. Ini disebabkan karena fitur dari *library reactJS* yang melakukan perubahan DOM terntentu, tidak melakukan perubahan keseluruhan DOM yang ada.

1. **Performansi CPU**

Untuk pengujian performansi CPU, dilakukan percobaan dengan menggunakan tool apache jmeter untuk melakukan skenario pengujian untuk setiap jumlah *thread* yang ada. Pengujian dilakukan dengan menggunakan dua buah komputer, komputer pertama digunakan untuk melakukan pengujian dengan menggunakan apache jmeter dan komputer kedua digunakan sebagai server. Pada komputer kedua ditampilkan performansi CPU yang ada ketika skenario setiap *thread* dijalankan pada komputer pertama. Hasil pengujian ditampilkan pada tabel 4.10.

Tabel 4. 6 Tabel pengujian performansi CPU

|  |  |  |  |
| --- | --- | --- | --- |
| Pengujian | Jumlah *Thread* | *Web* Konvensional (%) | *Web* *offline first app* (%) |
| 1 | 1 | 19,26,28,31 | 12,13,16,21 |
| 2 | 50 | 14,21,19,26 | 3,7,17,9 |
| 3 | 100 | 12,18,20,30 | 4,9,11,12 |
| 4 | 150 | 15,20,27,33 | 4,11,15,12 |
| 5 | 200 | 16,28,33,38 | 4,8,19,8 |
| 6 | 250 | 14,19,33,41 | 4,11,14,6 |
| 7 | 300 | 14,17,33,47 | 4,11,22,20 |
| 8 | 350 | 13,17,33,51 | 4,7,11,19 |
| 9 | 400 | 14,27,32,53 | 4,7,11,7 |
| 10 | 450 | 12,21,53,60 | 3,11,24,14 |
| 11 | 600 | 6,13,33,63 | 5,15,27,40 |
| 12 | 700 | 13,25,57,99 | 4,15,20,31 |

Gambar 4. 8 Grafik perbandingan performansi CPU

Dari grafik pada gambar 4.8 menunjukkan bahwa performansi CPU untuk web konvensional memiliki persentase yang lebih tinggi dibandingkan dengan web service worker. Untuk web konvensional semakin besar jumlah thread yang diberikan maka semakin besar pula persentase penggunaan CPU total. Berbeda dengan web service worker jumlah thread yang diberikan tidak memengaruhi penggunaan performansi CPU. Keunggulan dari web service worker bahwa performansi CPU untuk jumlah thread terbesar yang diberikan diperoleh persentase 31%, berbanding terbalik dengan web konvensional dimana hasil persentase untuk 700 thread menunjukkan sampai dengan angka 90% penggunaan CPU. Hal ini tentunya menjadi keunggulan dari penggunaan service worker pada sebuah website. Web service worker dapat mengatasi banyak jumlah request (thread) tanpa menaikkan performansi CPU server secara tinggi, sehingga dapat meringankan beban server yang digunakan. Hal ini kemungkinan dipengaruhi oleh kemampuan service worker yang dapat melakukan intercept dan menangani permintaan jaringan.

# **BAB V PENUTUP**

1. **KESIMPULAN**

Dalam analisis performansi antara *web* konvensional dan *web* *offline* *first app* terintegrasi *service worker* dengan menggunakan tool apache jmeter dan *developer tools google chrome,* hasil menunjukkan bahwa :

* + - *Web service worker* akan memberikan waktu akses rata – rata yang lebih rendah dibandingkan dengan *web* konvensional. Dengan jumlah *thread* total yang diberikan 450 pada *web* konvensional menunjukkan waktu akses yang lebih tinggi dibandingkan *web* *service worker*. Sehingga dapat dikatakan bahwa *web service worker* dapat melayani *request* yang banyak secara bersamaan dengan cepat.
    - Untuk hasil pengujian *throughput, web service worker* menghasilkan nilai rata - rata throughput yang tinggi dibandingkan dengan *web* konvensional. Hal ini menunjukkan bahwa *web service worker* dapat melakukan transfer file yang lebih besar dalam satuan waktu.
    - Untuk pemrosesan *browser*, *web service worker* ketika melakukan akases pertama kali akan menghasilkan waktu *load* yang lebih lama dibandingkan dengan *web* konvensional. Akan tetapi ketika akses berikutnya dilakukan *web service worker* akan menghasilkan waktu *load* yang lebih cepat dibandingkan dengan *web* konvensional. Ini disebabkan oleh *boost performance* dan manajemen *cache* yang ada pada *service worker*.
    - Untuk pengujian *input* data *web service worker* tidak mengalami perbedaan waktu *load*. Berbeda dengan *web* konvensional yang akan mengalami perubahan waktu *load*. Ini disebabkan karena fitur yang ada pada *library* *reactJS* yang melakukan render DOM tertentu saja yang mengalami perubahan konten. Berbeda dengan *web* konvensional yang akan melakukan render untuk seluruh halaman *web*.
    - Pengaruh performansi CPU pada saat *web service worker* diakses menunjukkan hasil persentase yang stabil dan sangat rendah dibandingkan dengan *web* konvensional. Hal ini disebabkan karena kemampuan *service worker* yang dapat membantu dalam menangani manajemen *request* yang masuk ke server.
    - *Service worker* memiliki keunggulan dalam segi performansi juga mampu bekerja dengan baik dalam kondisi *offline*, sehingga sistem yang dibangun mampu bekerja tanpa harus bergantung pada kondisi jaringan yang cepat.
    - *Web* dengan menggunakan *service worker* memberikan performansi yang lebih baik dibandingkan dengan web konvensional, sehingga data yang diperoleh dari hasil pengujian sesuai dengan hasil yang diharapkan.

1. **SARAN**

Adapun saran untuk penelitian selanjutnya :

* + - Perlu dilakukan penelitian lebih lanjut terhadap perangkat dan *software* yang berberda.
    - Penelitian berikutnya mampu melakukan analisis performansi dengan indikator yang berbeda seperti *availability, fault tolerance,* realibilitas dan yang lainnya.
    - Penelitian berikutnya mampu membangun sistem yang sempurna dengan menggunakan metode *offline first app* dan terintegrasi *service worker*.

# **DAFTAR PUSTAKA**

Adi, Laurensius, Rizky Januar Akbar, and Wijayanti Nurul Khotimah. 2017. “Platform E-Learning Untuk Pembelajaran Pemrograman *Web* Menggunakan Konsep Progressive *Web* Apps.” 6(2):2–6.

Anonym. "What is JSX". Diperoleh 17 Oktober 2018, dari https://www.reactenlightenment.com/react-jsx/5.1.html.

Batubara, Febrin Aulia. 2015. “PERANCANGAN *WEBSITE* PADA PT . RATU ENIM PALEMBANG.” 15–27.

Basques Kayce. (2018, 22 Agustus). "Measure Resource Loading Times". Diperoleh 20 Oktober 2018, dari https://developers.google.com/*web*/tools/chrome-devtools/network-performance/resource-loading

Butkiewicz, Michael. 2011. “Understanding *Website* Complexity : Measurements , Metrics , and Implications Categories and Subject Descriptors.”

Colanus, Ivo, Rally Drajana, and Feature Selection. 2017. “METODE SUPPORT VECTOR MACHINE DAN FORWARD SELECTION PREDIKSI PEMBAYARAN PEMBELIAN BAHAN BAKU.” 9:116–23.

Fajar Ridwan. (2016, 9 Mei). "Damien Katz Bapak database NoSQL CouchDB". Diperoleh 17 Oktober 2018, dari [https://www.codepolitan.com/damien-katz-bapak-basis data-nosql-couchdb](https://www.codepolitan.com/damien-katz-bapak-database-nosql-couchdb)

Haidar Dzacko. 2007. “1 . BASIS DATA ( BASIS DATA ).”

Hannonen, Daria. 2017. “Development of *Website* Solution for Association to Assist Young Professionals.” (November).

Khuat, Tung. 2018. “Developing a Frontend Application Using ReactJS and Redux.”

Mozahhebi, Massih. 2013. “Comparison of IndexedDB and SQLite Based on Developers ’ Concerns.”

Muhammad Agung Rizkyana, R.Sandhika Galih Amalga. 2014. “RANCANGAN ARSITEKTUR APLIKASI PENGUMPULAN TUGAS DENGAN *PUSH NOTIFICATION* REAL-TIME MENGGUNAKAN.” 2014(semnasIF):70–75.

No, Vol and Apri Junaidi. 2016. “Studi Perbandingan Performansi Antara MongoDB Dan MySQL Dalam Lingkungan Big Data.” 2(1):460–65.

Nylas.(2016) "Electron - RXDB". Diperoleh 17 Oktober 2018, dari <https://github.com/nylas/electron-RxDB>.

Permana, Endang Cahya. 2016. “Penulisan Fungsi Pada *Java script*.” 1–27.

Tamire, Workneh Tefera. 2016. “HTML5 and Its Capability to Develop *Offline* *Web* Applications.” (April).

Vanhala, Janne. 2017. “Implementing an *Offline* First *Web* Application.”