**ANALISIS PERFORMANSI *SERVICE WORKER***

**DENGAN *LIBRARY REACT JS***

**STUDI KASUS : WEB PEMELIHARAAN MESIN PADA INDUSTRI KECIL MENENGAH (IKM)**



**TUGAS AKHIR**

*Disusun dalam rangka memenuhi salah satu persyaratan*

*Untuk menyelesaikan program Strata-1 Departemen Teknik Informatika*

*Fakultas Teknik Universitas Hasanuddin*

*Makassar*

**Disusun Oleh :**

**AL RIEFQY DASMITO**

**D421 14 502**

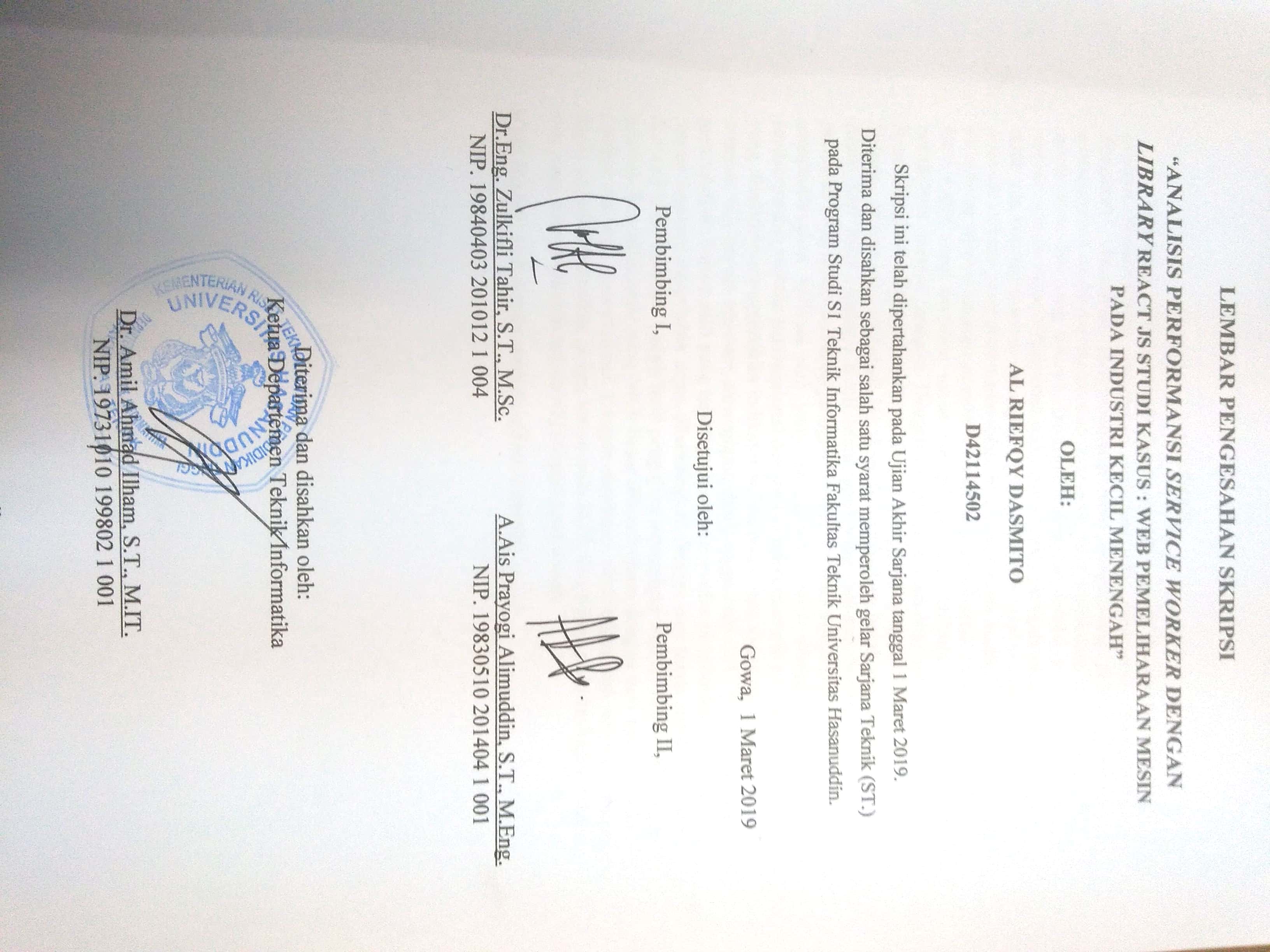
**DEPARTEMEN TEKNIK INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS HASANUDDIN**

**MAKASSAR**

**2019**

****

**ABSTRAK**

Teknologi *web* saat ini berkembang cukup pesat dengan hadirnya *web* modern. *Service worker* merupakan salah satu API (*Application Programming Interface*) *Javascript* yang memungkinkan pengembang dalam melakukan pemrograman *cache* dan melakukan *load asset data*, melakukan manajemen *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* mampu membuat waktu akses *web* lebih cepat karena fungsi *fetch event* yang berperan untuk mengakses *cache* terlebih dahulu kemudian melakukan *request* jaringan. Dengan memadukan teknologi *service worker* dan *React JS* sebuah sistem dapat dibuat dengan menggunakan pendekatan *offline first*. 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* menggunakan *javascript* dan *web*site konvensional dengan beberapa skenario pengujian. Analisis yang dilakukan menggunakan indikator waktu rata – rata akses pada jaringan 3G maupun 4G, 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* tanpa *service worker* dan *web* konvensional. Untuk hasil pengujian *throughput* *web service worker* juga memberikan nilai hasil yang lebih besar dibandingkan dengan *web* tanpa *service worker* dan *web* konvensional. Dengan demikian, diharapkan *web service worker* dapat diimplementasikan bukan hanya pada IKM, tetapi juga pada berbagai sistem *web* yang memerlukan kehandalan jaringan yang lemah.

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

**KATA PENGANTAR**

Assalamu’alaikum Warahmatullahi Wabarakatuh.

Segala puji dan syukur kami panjatkan ke hadirat Allah S.W.T Tuhan Yang Maha Esa yang dengan limpahan rahmat dan hidayah-Nya sehingga tugas akhir dengan judul “*Analisis Performansi Service worker dengan Library React JS, Studi Kasus : Web Pemeliharaan Mesin pada Industri Kecil Menengah (IKM)”* ini dapat diselesaikan sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 pada Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

Dalam penyusunan penelitian ini disajikan hasil penelitian terkait judul yang telah diangkat dan telah melalui proses pencarian dari berbagai sumber baik jurnal penelitian, prosiding pada seminar-seminar nasional/internasional, buku maupun dari situs-situs di internet.

Penulis menyadari bahwa tanpa bantuan dan bimbingan dari berbagai pihak, mulai dari masa perkuliahan sampai dengan masa penyusunan tugas akhir, sangatlah sulit untuk menyelesaikan tugas akhir ini. Oleh karena itu, pada kesempatan ini penulis menyampaikan ucapan terima kasih sedalam-dalamnya kepada:

1. Tuhan Yang Maha Esa atas semua berkat, karunia serta pertolongan-Nya yang tiada batas, yang telah diberikan kepada penulis disetiap langkah dalam pembuatan program hingga penulisan laporan skripsi ini.
2. Kedua orang tua penulis serta kedua saudara penulis, serta keluarga yang senantiasa memberikan kekuatan, inspirasi, motivasi, bimbingan moral, materi, kepercayaan dan kasih sayang yang tidak terbatas kepada penulis.
3. Bapak Dr.Eng Zulkifli Tahir, S.T.,M.Sc., selaku pembimbing 1 yang telah banyak memberi bimbingan, inspirasi, motivasi, dan masukan yang bermanfaat kepada penulis.
4. Bapak A.Ais Prayogi Alimuddin, S.T., M.Eng., selaku pembimbing II yang telah banyak memberi keyakinan, perhatian, bimbingan, motivasi, dan masukan yang bermanfaat kepada penulis.
5. Bapak  Dr. Ir Zahir Zainuddin, M.Sc. dan Bapak Adnan, S.T., M.T., Ph.D., selaku dosen penguji yang telah memberikan saran sehingga laporan skripsi ini menjadi lebih baik.
6. Bapak Dr. Amil Ahmad Ilham, S.T., M.IT., selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas bantuan dan bimbingannya selama masa perkuliahan penulis.
7. Bapak Dr.Eng. Muhammad Niswar, S.T., M.IT., selaku kepala LAB CCIE yang telah memberikan bimbingan selama masa pengerjaan tugas akhir penulis.
8. Bapak Robert dan Bapak Zainuddin serta segenap staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu kelancaran penyelesaian tugas akhir penulis.
9. Segenap keluarga LAB CCIE Universitas Hasanuddin yang telah memberikan begitu banyak bantuan selama penelitian, pengambilan data dan diskusi *progress* penyusunan tugas akhir serta memberikan semangat di masa-masa sulit.
10. Para sahabat dekat penulis terutama Tiwi Nur Safitri,S.T, A. Khairil Fajri, Muh Zulfadli A. Suyuti, Gian Aron Angelo, Aryandi, S.T., Inka G. Mallisa, Fitriani Idrus, Winda Astiyanti, Armiani Putri, Rizka Irianti, David Reinhart, Zulfahmi, Fachrial yang telah memberikan doa, bantuan dan dukungan sejak masa awal perkuliahan.
11. Teman-teman Adyaksa Yakip, Muh Nur Alamsyah, Hermawan Safrin, Syarif Hidayatullah, Abdillah Satari Rahim, Rahmat Firman, Fadel Pratama, Muhammad Shadiq, Ahmad Setiadi.
12. Seluruh teman-teman RECTIFIER’14 atas semua bantuan dan semangat yang diberikan selama ini.
13. Serta seluruh pihak yang tidak sempat disebutkan satu persatu yang telah banyak meluangkan tenaga, waktu, dan pikiran selama penyusunan laporan tugas akhir ini.

Akhir kata, penulis berharap semoga Tuhan Yang Maha Esa berkenan membalas segala kebaikan dari semua pihak yang telah banyak membantu. Semoga tugas akhir ini dapat memberikan manfaat bagi pengembangan ilmu selanjutnya. Amin.

Wassalam

Gowa, 1 Maret 2019

Penulis

**DAFTAR ISI**

[HALAMAN JUDUL i](#_Toc2387014)

[LEMBAR PENGESAHAN ii](#_Toc2387015)

[ABSTRAK ii](#_Toc2387016)

[KATA PENGANTAR ii](#_Toc2387017)

[DAFTAR ISI ii](#_Toc2387018)

[DAFTAR TABEL ii](#_Toc2387019)

[BAB I PENDAHULUAN 2](#_Toc2387020)

[1.1. Latar Belakang 2](#_Toc2387021)

[1.2. Rumusan Masalah 2](#_Toc2387022)

[1.3. Tujuan Penelitian 2](#_Toc2387023)

[1.4. Manfaat Penelitian 2](#_Toc2387024)

[1.5. Batasan Masalah 2](#_Toc2387025)

[1.6. Sistematika Penulisan 2](#_Toc2387026)

[BAB II TINJAUAN PUSTAKA 2](#_Toc2387027)

[2.1. Pengertian *Website* 2](#_Toc2387028)

[2.2. Modern *website* 2](#_Toc2387029)

[2.3. Teknologi *Offline first app* 2](#_Toc2387030)

[2.3.1. *Single-page Application* 2](#_Toc2387031)

[2.3.2. *Offline First* 2](#_Toc2387032)

[2.3.3. *Web Caching* 2](#_Toc2387033)

[2.3.4. *Service worker*](#_Toc2387034) 17

[2.4. Bahasa Pemrograman 2](#_Toc2387035)

[2.4.1. *Java script* 2](#_Toc2387036)

[2.4.2. Node JS 2](#_Toc2387037)

[2.4.3. React js 2](#_Toc2387038)

[2.4.3.1. JSX 2](#_Toc2387039)

[2.4.3.2. *Stateful Components* 2](#_Toc2387040)

[2.4.3.3. *Virtual Document Object Model* 2](#_Toc2387041)

[2.5. Basis data 2](#_Toc2387042)

[2.5.1. SQL Basis data 2](#_Toc2387043)

[2.5.2. NoSQL Basis data 2](#_Toc2387044)

[2.5.2.1. IndexedDB 2](#_Toc2387045)

[2.5.2.2. RXDB 2](#_Toc2387046)

[2.5.2.3. Couch DB 2](#_Toc2387047)

[2.5.2.4. Pouch DB 2](#_Toc2387048)

[BAB III METODOLOGI PENELITIAN 2](#_Toc2387049)

[3.1. Lokasi dan Waktu Penelitian 2](#_Toc2387050)

[3.2. Instrumen Penelitian 2](#_Toc2387051)

[3.3. Prosedur Penelitian 2](#_Toc2387052)

[3.4. Tahap Persiapan 2](#_Toc2387053)

[3.5. Gambaran Umum Sistem 2](#_Toc2387054)

[3.5.1. *System Activity* dan Pengujian *Black box* 2](#_Toc2387055)

[3.5.2. *Hardware environtment* **Error! Bookmark not defined.**](#_Toc2387056)49

[3.6. Hasil Pembuatan Sistem 2](#_Toc2387057)

[3.7. Skenario Pengujian 2](#_Toc2387058)

[3.7.1. *Respon Time* 2](#_Toc2387059)

[3.7.2*. Throughput* 2](#_Toc2387060)

[3.7.3. Proses *Browser* 2](#_Toc2387061)

[3.7.4. Prosesor 2](#_Toc2387062)

[BAB IV HASIL DAN PEMBAHASAN 2](#_Toc2387063)

[4.1. Pengujian *Respon Time* 2](#_Toc2387064)

[4.1.1. Kondisi jaringan 3G 2](#_Toc2387065)

[4.1.2. Kondisi Jaringan 4G 2](#_Toc2387066)

[4.2. *Throughput* 2](#_Toc2387067)

[4.3. Proses *Browser* 2](#_Toc2387068)

[4.3.1. Waktu akses secara *online* 2](#_Toc2387069)

[4.3.2. Waktu akses secara *offline* 2](#_Toc2387070)

[4.3.3. Skenario *online* *input data* 2](#_Toc2387071)

[4.4. Performansi CPU 2](#_Toc2387072)

[4.5*.*Pengujian *black box sistem* 2](#_Toc2387073)

[BAB V PENUTUP 2](#_Toc2387074)

[5.1. KESIMPULAN 2](#_Toc2387075)

[5.2. SARAN 2](#_Toc2387076)

[DAFTAR PUSTAKA 2](#_Toc2387077)

**DAFTAR GAMBAR**

[Gambar 3. 1. Diagram Tahapan Penelitian 2](#_Toc534552862)

[Gambar 3. 2. Proses Manajemen Pemeliharaan Mesin IKM 2](#_Toc534552863)

[Gambar 3. 3. Blok Diagram Sistem 2](#_Toc534552864)

[Gambar 3. 4. Diagram *activity* *web* konvensional . 2](#_Toc534552865)

[Gambar 3. 5. Diagram *activity* *web* dengan *service worker* 2](#_Toc534552866)

[Gambar 3. 6. Diagram *activity* *input* data pada *web* konvensional 2](#_Toc534552867)

[Gambar 3. 7. Diagram *activity* *input* data pada *web* dengan *service worker* 2](#_Toc534552868)

[Gambar 3. 8. Diagram *activity* pembaharuan data pada *web* *service worker* 2](#_Toc534552869)

[Gambar 3. 9. Proses hapus data pada *web* dengan menggunakan *service worker* 2](#_Toc534552870)

[Gambar 3. 10. *Hardware environtment* 2](file:///C:\Users\AlRiefqy\OneDrive\Documents\Tugas%20Akhir%20Kodong\SKRIPSI_Al%20Riefqy.docx#_Toc534552871)

[Gambar 3. 11. *Use case* diagram 2](#_Toc534552872)

[Gambar 3.12. Halaman utama sistem 2](#_Toc534552873)

[Gambar 3. 13. *Service worker* pada aplikasi *web* 2](#_Toc534552874)

[Gambar 3. 14. Halaman Mesin 2](#_Toc534552875)

[Gambar 3. 15. Basis data lokal indexedDB 2](#_Toc534552876)

[Gambar 3. 16. Basis data server couch db 2](#_Toc534552877)

[Gambar 3. 17. *Network Panel Chrome* yang digunakan mengukur waktu akses. 2](#_Toc534552878)

[Gambar 4. 1 Grafik hasil pengujian waktu rata – rata 2](#_Toc1759234)

[Gambar 4. 2. Grafik Waktu Rata - Rata pada Jaringan 4G 2](#_Toc1759235)

[Gambar 4. 3 Grafik hasil throughput 2](#_Toc1759236)

[Gambar 4. 4. Grafik prowses browser untuk waktu *load* 2](#_Toc1759237)

[Gambar 4. 5. Grafik proses browser *DomContentLoaded* 2](#_Toc1759238)

[Gambar 4. 6. Grafik proses browser *web* *service worker* 2](#_Toc1759239)

[Gambar 4.7. Grafik waktu *load* *input* data 2](#_Toc1759240)

[Gambar 4. 8 Grafik perbandingan performansi CPU 2](#_Toc1759241)

**DAFTAR TABEL**

[Tabel 3. 1. Tabel format basis data 2](#_Toc534194702)

[Tabel 4. 1. Hasil Pengujian *Average* Secara *Online* pada Jaringan 3G 2](#_Toc1766392)

[Tabel 4. 2.Tabel Pengujian Waktu Rata - Rata pada Jaringan 4G 2](#_Toc1766393)

[Tabel 4. 3.Tabel hasil pengujian throughput pada *web* *online* 2](#_Toc1766394)

[Tabel 4. 4. Tabel hasil pengujian akses pertama kali tanpa *cache* 2](#_Toc1766395)

[Tabel 4. 5. Tabel pengujian *online* dengan *cache* 2](#_Toc1766396)

[Tabel 4. 6. Tabel pengujian *offline* dengan *cache* 2](#_Toc1766397)

[Tabel 4. 7. Tabel pengujian *online* *input* data 2](#_Toc1766398)

[Tabel 4. 8. Tabel pengujian performansi CPU 2](#_Toc1766399)

[Tabel 4. 9. Tabel hasil pengujian *black box* *input* data 2](#_Toc1766400)

[Tabel 4. 10. Hasil pengujian *black box* update data 2](#_Toc1766401)

[Tabel 4. 11. Hasil pengujian *black box* delete data 2](#_Toc1766402)

**BAB I  
PENDAHULUAN**

* 1. **Latar Belakang**

*Service worker* merupakan salah satu APIs *Java script* yang memungkinkan pengembang dalam melakukan pemrograman *cache* dan melakukan *load asset data*, melakukan managemen *push notification*, dan yang lainnya.  *Service worker* adalah sebuah *script* yang bekerja pada *browser* *background* tanpa interaksi dari pengguna. *Service worker* memiliki kehandalan dalam hal koneksi internet yang tidak stabil ataupun lambat sehingga membuat sistem yang ada tidak terlalu bergantung pada koneksi internet yang cepat. Selain itu disatu sisi *service worker* disebut sebagai *performance booster* karena menghemat jaringan dan menyediakan *user experience* yang lebih baik. *Service worker* diimplementasikan pada *Java script* yang berjalan pada *background* *website* utama dan berjalan secara paralel dengan *website* utama sehingga kerja dari *service worker* tidak terlalu terlihat secara langsung. *Service worker* pada dasarnya adalah sebuah *script* atau *Java script* file yang berjalan dibelakang *website* utama dan membantu dalam pengembangan aplikasi *web* *offline*.

Dengan kemampuan *service worker* menjadikan sebuah *website* dapat difungsikan walaupun koneksi jaringan yang lemah ataupun tidak ada koneksi internet sekalipun mampu mengatasi masalah *website* yang tidak harus selalu bergantung pada koneksi jaringan. Hal tersebut dapat dilakukan dengan menerapkan pengembangan teknologi *offline first web application.*  *Offline first* adalah sebuah pendekatan dalam pengembangan perangkat lunak dimana *developer* membangun fitur atau aplikasi yang ada akan bekerja dengan atau tanpa koneksi internet. Dengan menggunakan pendekatan ini data yang ada akan di simpan secara lokal dan secara berkala akan melakukan sinkronisasi ketika koneksi internet tersedia.

Aplikasi *Web*

*Service worker*

*Offline*

*Online*

Lokal Storage / *Cache*

Respon

Respon

*Web* server

Telah disebutkan sebelumnya bahwa *service worker* adalah API *Java script* sehingga *service worker* didukung oleh berbagai *library Java script* dan *library* yang ada. Contohnya adalah *React JS* yang mendukung *service worker* dalam melakukan pengembangan aplikasi *website* dengan pendekatan *offline first*.

*React JS* adalah sebuah *Java script library* untuk membangun *user interface* yang didesain oleh *facebook* untuk membuat sebuah *website* yang cepat namun minim kode. *React JS* hanya menangani semua hal yang berkaitan dengan tampilan. Telah disebutkan bahwa *React JS* cepat dan efisien, hal ini disebabkan karena *React JS* menciptakan *Virtual DOM (Document Object Model)* dimana setiap perubahan yang terjadi pada setiap laman *website* hanya akan berubah pada DOM yang hanya mengalami perubahan saja sehingga tidak mengubah seluruh halaman *website*.

Dengan memadukan teknologi *service worker* dan *React JS* sebuah sistem dapat dibuat dengan menggunakan pendekatan *offline first* dan memiliki kemampuan yang cepat. Teknologi ini dapat dimanfaatkan untuk menunjang proes bisnis dari IKM yang ada, yang rata-rata berada pada sektor pedesaan yang minim terhadap jaringan internet. Dengan semakin baiknya proses pada industri kecil menengah di desa – desa yang ada di Indonesia nantinya akan mendorong penyerapan tenaga kerja secara merata dan meluas disegala sektor ekonomi yang ada. IKM dituntut untuk mengikuti perkembangan teknologi yang ada agar dapat meningkatkan kinerja dan mampu bertahan dalam persaingan ekonomi terkini.

Sebuah sistem yang berbasis *online* adalah satu dari banyak cara yang dapat digunakan IKM untuk bertahan. Namun dalam menerapakan sebuah sistem berbasis *online* pada IKM yang mayoritas berada pada pedesaan memiliki kesulitan dalam meraih koneksi jaringan yang handal, sehingga dibutuhkan sebuah sistem yang mampu bekerja dengan kondisi jaringan yang lemah atau tidak ada sama sekali sehingga *service worker* merupakan salah satu teknologi yang dapat mengatasi masalah koneksi jaringan yang ada.

Berdasarkan uraian diatas penulis kemudian mengangkat sebuah penelitian dengan judul “**Analisis performansi *service worker* dengan *library React JS*. Studi kasus : *web* pemeliharaan mesin pada Industri Kecil Menengah (IKM)”** .

* 1. **Rumusan Masalah**

1. Bagaimana meningkatkan performansi *web* IKM sektor pedesaan yang mempunyai kemampuan internet terbatas?
2. Bagaimana membuat *website* pemeliharaan mesin industri kecil menengah (IKM)dengan menggunakan teknologi *service worker*?
3. Bagaimana mengintegrasikan *service worker* dengan teknologi *React JS*?
   1. **Tujuan Penelitian**
4. Mengetahui cara mengatasi performa *website* dengan kondisi jaringan dan bandwith yang lemah.
5. Mengetahui cara membuat *website* pemeliharaan mesin pada industri kecil menengah (IKM) dengan menggunakan teknologi *service worker*
6. Mengetahui bagaimana menggunakan teknologi *service worker* dengan teknologi *React JS library*  dalam pembuatan *website*.
   1. **Manfaat Penelitian**
7. Dapat mengetahui cara mengatasi performa *website* dalam kondisi jaringan yang lemah
8. Dapat membuat sebuah *website* pemeliharaan mesin industri kecil menengah (IKM) dengan menggunakan teknologi *service worker*.
9. Dapat mengetahui cara integrasi *service worker* dan teknologi React JS.
   1. **Batasan Masalah**
10. Pembuatan *website* menggunakan *library React JS.*
11. *Website* yang dibangun hanya mampu memasukkan data kedalam basis data, memperbaharui data, menghapus data dengan menggunakan dua fitur utama yaitu melakukan pemasukan data mesin dan data pemeliharaan mesin.
12. Basis data yang digunakan adalah indexedDB sebagai *local storage* dan CouchDB sebagai *remote* basis data*.*
13. Analisis *service worker* pada *website* dengan menggunakan *library React JS* merupakan analisis yang lebih menekankan kepada bagaimana penggunaan aplikasi ketika diakses oleh banyak *request* dan kecepatan pengaksesan *website* yang dibuat.
14. Software yang digunakan untuk men
15. guji performa *website* yang dibuat adalah apache Jmeter
    1. **Sistematika Penulisan**

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

**BAB II TINJAUAN PUSTAKA :** 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***

*Website* atau situs dapat diartikan sebagai kumpulan halaman-halaman yang digunakan untuk menampilkan informasi teks, gambar diam atau gerak, animasi, suara, dan atau gabungan dari semuanya baik yang bersifat statis maupun dinamis yang membentuk satu rangkaian bangunan yang saling terkait, yang masing-masing dihubungkan dengan jaringan- jaringan halaman. Hubungan antara satu halaman *web* dengan halaman *web* yang lainnya disebut *hyperlink*, sedangkan teks yang dijadikan media penghubung disebut *hypertext* (Batubara 2015) .

Dalam beberapa dekade, *website* telah menjadi lebih besar dan kompleks. Pada awalnya *website* digunakan hanya untuk mempermudah tukar menukar dan melakukan perbaruan informasi kepada sesama peneliti yang dilakukan oleh Sir Timothy John. namun dalam perkembangannya *website* dapat melakukan manajemen konten seperti video dan gambar (Butkiewicz 2011).

Secara teknis, *web* adalah sebuah sistem dimana informasi dalam bentuk teks, gambar, suara, dan lain-lain yang tersimpan dalam sebuah internet *web* server dipresentasikan dalam bentuk *hypertext*. Informasi di *web* dalam bentuk teks umumnya ditulis dalam format HTML. Informasi lainnya disajikan dalam bentuk grafis (dalam format GIF, JPG, PNG), suara (dalam format AU, WAV), dan objek multimedia lainnya (seperti MIDI, Shockwave, Quicktime Movie, 3D World). *Web* dapat diakses oleh perangkat lunak *web client* yang secara populer disebut sebagai *browser*. *Browser* membaca halaman-halaman yang tersimpan dalam *webserver* melalui protokol yang disebut HTTP (*Hypertext Transfer Protocol*). Sebagai dokumen *hypertext*, dokumen-dokumen di *web* dapat memiliki *link* dengan dokumen lain, baik yang tersimpan dalam *web* server yang sama maupun di *web* server lainnya. Link memudahkan para pengakses *web* berpindah dari satu halaman ke halaman lainnya, dan "berkelana" dari satu server ke server lain. Kegiatan penelusuran halaman *web* ini biasa diistilahkan sebagai *browsing*, ada juga yang menyebutnya sebagai *surfing* (berselancar). Seiring dengan semakin berkembangnya jaringan internet di seluruh dunia, maka jumlah situs *web* yang tersedia juga semakin meningkat. Hingga saat ini, jumlah halaman *web* yang bisa diakses melalui internet telah mencapai angka miliaran. Untuk memudahkan penelusuran halaman *web*, terutama untuk menemukan halaman yang memuat topik-topik yang spesifik, maka para pengakses *web* dapat menggunakan suatu mesin pencari (*search engine*). Penelusuran berdasarkan *search engine* dilakukan berdasarkan kata kunci (*keyword*) yang kemudian akan dicocokkan oleh *search engine* dengan basis data miliknya (Batubara 2015).

1. **Modern *website***

Saat ini, ada banyak cara dalam melakukan pengembangan dan pembuatan aplikasi *web*. Namun pengembangan *website* dimulai dengan memahami bagaimana arsitektur *web* yang akan digunakan, *web* statik atau dinamis dan juga penentuan pengembangan *tool* dan *service* yang digunakan.

Sebuah *website* statis terdiri atas beberapa halaman HTML, CSS yang secara bersamaan saling terhubung oleh *hyperlinks*. *Website* dinamis memiliki lokasi penyimpanan konten pada basis data dan ditampilkan berdasarkan permintaan penguna. Namun HTML dan CSS juga dapat digunakan dalam *website* dinamis ketika memiliki *java script* yang mengandung pemrograman *back-end*. *Website* modern dibangun menggunakan pemrograman *front-end* yang dieksekusi oleh *browser* dan berjalan pada sisi *client* yaitu HTML,CSS, dan pemrograman *back-end* yang dieksekusi pada sisi server yaitu *java script,* PHP, Python dan yang lainnya yang sering digunakan oleh pengembang *web*. Pemrograman *back-end*  bekerja pada sisi belakang *website* yang tidak dilihat oleh pengguna dan bekerja berdampingan dalam melakukan akses basis data, sedangkan pemrograman *front-end* berhubungan dengan apa yang pengguna dapat lihat (Hannonen 2017).

Sisi server

*File*

*Client-side*

*Web server*

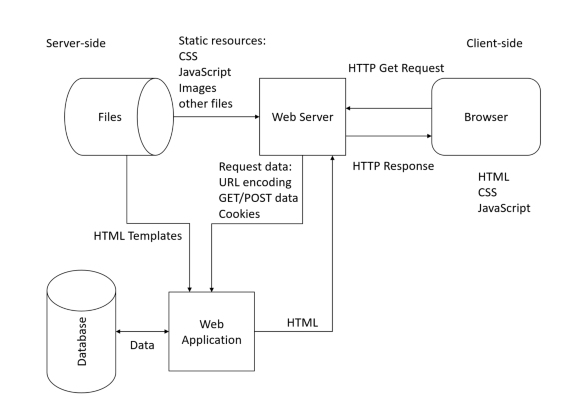
*browser*

Respon HTTP

HTTP mendapatkan

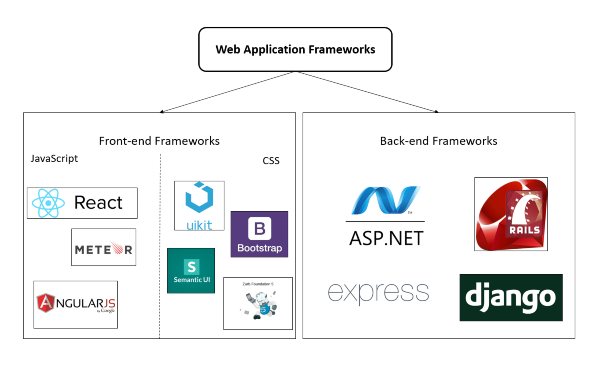
permintaan

Gambar 2.1 : Arsitektur *Website* statis (Hannonen 2017).

 Pada *website* statis ketika pengguna ingin melakukan perpindahan halaman maka *browser* mengirimkan permintaan HTTP “GET” yang menunjukkan URL. Server kemudian mengirimkan dokumen permintaan dari sistem file menuju *browser* pengguna.

Gambar 2.2 : Arsitektur *web* dinamis (Hannonen 2017).

Dari kedua gambar yaitu gambar 2.1 dan gambar 2.2, masing masing arsitektur memiliki kekurangan dan kelebihan. *Website* statis memiliki kelebihan yaitu sederhana dan cepat dalam melakukan proses pengembangan. Namun kekurangannya adalah pengembang harus melakukan perubahan kode baik HTML maupun CSS setiap konten mengalami perubahan. Berbeda dengan *website* dinamis dimana memiliki kemudahan dalam melakukan perubahan konten karena bekerja dengan data yang dinamis. Namun dalam mengembangkan arsitektur *web* dinamis dibutuhkan lebih dari satu pengembang untuk memastikan *web* yang dibuat berjalan dengan semestinya dan memiliki desain yang menarik.

 Seiring perkembangannya, hari ini pengembangan *web* memiliki cara pengembangan yang lebih mudah dan disukai oleh pengembang, yaitu dengan menggunakan *framework* dalam melakukan pengembangan *web*. *Framework* memudahkan para pengembang karena kemampuannya menyediakan manajemen *session,* penyimpanan data, dan melakukan *templating* sistem. *Framework* juga memudahkan dalam melakukan manajemen pengerjaan *web* yaitu dengan memisahkan bentuk pengerjaan dari sisi *front-end* dan *back-end.*

Gambar 2.3 : Contoh aplikasi *web* *Framework* (Hannonen 2017).

*Framework* sisi server adalah *Framework back-end* yang memudahkan pengembang dalam melakukan penulisan dan pemeliharaan aplikasi *web*. *Framework* menyediakan pustaka yang sederhana yang sering digunakan dalam membangun sebuah sistem *web*, termasuk URL *routing* , interaksi dengan basis data, mendukung manajemen *session* maupun otorisasi pengguna (*user authorization*), manajemen masukan (HTML, JSON, XML), dan peningkatan keamanan terhadap serangan *web* (Hannonen 2017).

Sedangkan *front-end Framework* sangat membantu dalam melakukan desain proses dan masukan dari berbagai fitur yang ada, termasuk fitur kanvas, menu, dan model yang ada. *Framework front-end* dibagi menjadi dua yaitu *Java script Framework* (Angular,ReactJS,MeterJS) dan CSS *Framework*  (Bootstrap, Foundation, SemanticUI, UIkit, MaterialUI) (Hannonen 2017).

1. **Teknologi *Offline first app***
2. ***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 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. ***Offline First***

Aplikasi *web* biasanya bekerja dengan kondisi yang memerlukan jaringan internet. Oleh karena itu, ketika pengguna melakukan permintaan ke server untuk melakukan interaksi ada proses yang harus di tempuh oleh permintaan untuk bisa sampai ke server. Hal ini bisa menjadi masalah ketika proses yang ditempuh mengalami masalah jaringan sehingga membutuhkan waktu yang lama untuk server menerima respon dari pengguna.

Dalam membuat aplikasi *web* *offline* maka arsitektur yang dibuat harus menggunakan logika yang dapat bekerja walaupun koneksi internet tidak tersedia, tidak seperti aplikasi *web* yang standar. Jenis aplikasi ini menyimpan sumber daya yang penting dari aplikasi seperti *cache* dan secara terus menerus membawa informasi dari server ketika dilakukan permintaan dari *browser* (Tamire 2016).

Kemampuan aplikasi bekerja secara *offline* sangat berharga terutama ketika tidak tersedianya koneksi internet namun tetap memerlukan koneksi agar aplikasi dapat bekerja. Untuk membuat performa aplikasi *offline* bekerja dengan sukses maka perlu di lakukan pengecekan,pembaharuan dan membuat sumber daya menjadi aman yang tersimpan pada perangkat pengguna.

Penyimpanan server

Penyimpanan data lokal

Sisi klien Aplikasi

sinkronisasi

*read*

*write*

*write*

*read*

Gambar 2.4 : Elemen inti aplikasi *web* *offline* (Tamire 2016)

Sesuai dengan gambar 2.4, aplikasi *web* *offline* menggunakan penyimpanan lokal ketika jaringan internet tidak tersedia. Sebaliknya, ketika koneksi internet tersedia, data akan dapat dibaca dan di buat langsung dari server.

*Offline first* adalah paradigma desain terbaru dalam membuat sebuah aplikasi *web*. Teknologi ini digunakan pertama kali pada tahun 2012 oleh Lambert dalam artikelnya berjudul “*Offline First – A better HTML5 User Experience”.* Dalam artikelnya, Lambert menjelaskan bahwa *offlline* adalah fitur yang penting dan harus dipertimbangkan pada awal proses pengembangan sistem. Lambert menjabarkan ada tiga panduan untuk melakukan pengembangan aplikasi *offline first* dari segi teknis. (1) semua logika aplikasi yang biasanya berada pada sisi server dipindahkan pada sisi klien. Server hanya digunakan sebagai media penyimpanan dan berkomunikasi dengan menggunakan *Java script Object Nation* (JSON) antara klien dan server. (2) Membuat sebuah kode layer yang melakukan interaksi dengan sisi server API (*application programming interface*). (3) Membuat kode pemrograman yang melakukan permintaan data dari server dan melakukan *caching* di penyimpanan *browser*  (Lambert 2012).

Feyerke memberikan teori yang hampir sama dengan Lambert dalam artikel nya. (1) Logika aplikasi berjalan pada *browser*. (2) Kedua, untuk memaksimalkan pengalaman *offline* pengguna, klien harus melakukan penyimpanan data pada *browser* dan mampu melakukan sinkronisasi dengan server (Feyerke 2013).

1. ***Web Caching***

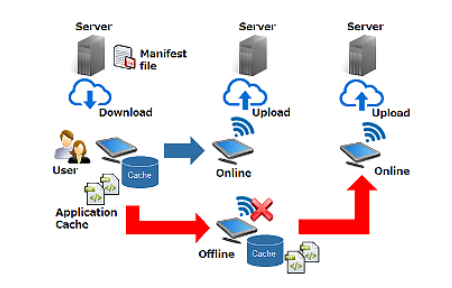
Ketika menggunakan aplikasi *web* atau melakukan pencarian informasi dari sebuah *web*, dua hal yang penting untuk diperhatikan yaitu kecepatan dan efisiensi. Setiap pengguna akan merasa puas ketika bekerja dengan kecepatan aplikasi yang tinggi. Apalagi ketika mendesain aplikasi *web*, dengan menerapkan teknologi *caching* akan mampu meningkatkan waktu akses dan menurunkan lalu lintas jaringan.

Akibatnya, *web* *cache* memungkinkan untuk meningkatkan kinerja dan skalabilitas aplikasi sehingga dengan menggunakan metode ini dapat mengoptimalkan kinerja aplikasi. Selain itu, *cache* mampu menyimpan Salinan halaman aplikasi *web* pada sistem penyimpanan lokal pengguna sehingga nanti dapat digunakan ketika jaringan internet tidak tersedia (Tamire 2016).

*Web* *cache* mampu melakukan penyimpanan *Cascading Style sheet* (CSS), *Java script*, gambar, HTML dan sumber daya yang lain pada *web* *browser* sehingga mampu menurunkan waktu akses data dari *web* server. Disisi lain ketika permintaan dikirm ke server, maka akan banyak sumber daya dikirim kembali kepada pengguna meminta halaman yang sama untuk kedua kalinya yang berarti akan membuat server menjadi sibuk dan membuat lalu lintas *web* menjadi tinggi.

Kebanyakan *web* *browser* akan melakukan *cache* pada halaman *web* ketika pertama kali diakses oleh pengguna. Namun melakukan *cache* halaman *web* memiliki beberapa keterbatasan. Yang pertama adalah pengembang *web* harus menentukan halaman *web* mana yang akan dilakukan *cache* dalam kondisi tidak adanya koneksi internet. Masalah kedua adalah *caching* halaman *web* otomatis tidak menjamin untuk melakukan *load* *cache* yang sudah disimpan ketika tidak tersedianya koneksi internet.

Hasilnya, untuk mengatasi masalah tersebut HTML5 memiliki API *offline* *cache*. Dengan menggunakan teknologi ini pembuat *web* dengan mudah mampu menentukan sumber daya apa yang akan dilakukan *cache* dan tersedia ketika aplikasi berjalan secara *offline*. Selain itu terdapat sebuah API *Java script* yang mampu melakukan managemen *cache* terhadap halaman *web*, sehingga pengembang *web* mampu mengatur dan melakukan managemen *cache* (Tamire 2016).

Aplikasi HTML 5 *cache* adalah teknologi yang sangat penting untuk mendukung aplikasi berjalan secara *offline*. Hasilnya dengan menggunakan teknologi ini membuat *website* untuk bekerja dengan baik pada kasus tidak tersedianya koneksi .

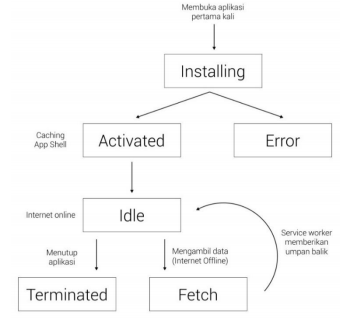
Gambar 2.5 : Level tinggi Arsitektur HTML (Tamire 2016)

Sesuai dengan gambar 2.5, ketika permintaan dibuat untuk melakukan pada akses *single page*, maka kemudian sistem melakukan penguduhan setiap data yang berada pada daftar manifest aplikasi. Setelah semua manifest di simpan maka aplikasi akan bekerja dengan atau tanpa koneksi internet (Tamire 2016).

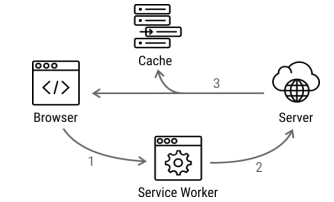
1. ***Service worker***

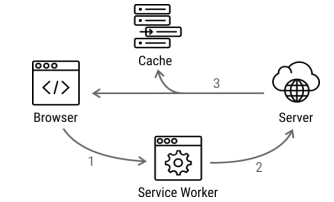
*Service worker* adalah salah satu jenis dari *web* worker, yaitu script yang berjalan di belakang *browser* pengguna. *Service worker* pada dasarnya adalah berkas *Java script* yang berjalan pada *thread* yang berbeda dengan main *thread* *browser*, menangani network *request*, *caching*, mengembalikan 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 pesan kesalahan karena ketiadaan koneksi internet. Untuk memasang *service worker*, kita butuh melakukan registrasi dengan menggunakan *Java script* yang ada di halaman *web*. Setelah registrasi, *browser* akan memulai tahap pemasangan *service worker*. Setelah aktif, *service worker* akan menangani semua halaman di bawah scope dimana *service worker* di-install, lalu akan ada 2 kemungkinan *state*: terminated untuk menghemat memori, atau menangani operan data ketika ada permintaan dari halaman *web*.

*Service worker* bisa menangani berbagai jenis *request*, tetapi yang bisa disimpan kedalam *cache* hanya semua *request* jenis GET. Pada *service worker*, bisa aplikasikan strategi *caching* sesuai keinginan kita, namun tidak ada satu strategi terbaik untuk *caching* konten dinamis, dan ada banyak situasi yang bisa mempengaruhi strategi *caching* (Adi, Akbar, and Khotimah 2017).



Gambar 2.6 : Daur hidup *service worker* (Adi et al. 2017)

*Service worker* bekerja sebagai pengatur *event fetch* dari *browser*, lalu *service worker* memutuskan apakah *request* akan diteruskan ke server atau ke *cache* berdasarkan kondisi jaringan *online* atau *offline*. Dari banyak jenis strategi *caching* pada *service worker* akan digunakan jenis *network first*, *cache* *fallback*. Pada strategi ini pertama-tama *service worker* mengecek apakah network memberikan respons, dan jika berhasil, kembalikan data sekarang ke halaman. Jika gagal, *service worker* mengembalikan data dari *cache*. Strategi ini dipakai ketika membutuhkan data yang selalu baru seperti respons API, tapi butuh menampilkan sesuatu ketika network tidak bisa dicapai (Adi et al. 2017).

Gambar 2.6 : *Service worker* dalam kondisi *offline* (Adi et al. 2017)*.*

Gambar 2.7 : *Service worker* dalam kondisi *onnline* (Adi et al. 2017)

*Service worker* akan meneruskan setiap *request* berjenis GET dari halaman *web* ke server dalam kondisi *online*, lalu menduplikasi respons server dan disimpan ke dalam *cache* di *browser*, lalu respons server diteruskan kembali ke halaman *web* seperti digambarkan pada Gambar 2. Setiap halaman yang pernah dikunjungi oleh pengguna secara otomatis akan tersedia dalam versi *offline*-nya, hanya saja pengguna tidak bisa mengirim data ke server. Ketika kondisi *offline* atau tidak ada umpan balik dari server, maka akan dikembalikan halaman dari *cache*. Halaman bisa ditampilkan apabila halaman tersebut sudah ada di dalam *cache* sebelumnya seperti digambarkan pada Gambar 2.7 (Adi et al. 2017).

Dengan menggunakan *service worker* maka berbagai jenis fitur dapat digunakan yaitu , melakukan *intretrcept* *request* dan *Responese* , melakukan *push notification,* dan *background sync. Intercept request dan response* bekerja melalui *service worker*, sehingga setiap *request* yang ada dapat dilakukan *intercept* dan membuat skenario pengaksesan *web*. Sedangkan fitur *push notification* dapat memunculkan notifikasi melalui *service worker* walaupun halaman *web* tertutup. Dan *background* sinkronisasi, ada atau tidaknya jaringan internet maka *service worker* tetap dapat melakukan permintaan ke server dan melakukan sinkronisasi secara berkala.

1. **Bahasa Pemrograman**

Bahasa pemrograman, atau sering diistilahkan juga dengan Bahasa komputer, adalah instruksi standar untuk memerintah komputer. Bahasa pemrograman ini merupakan suatu himpunan dari aturan sintaks dan semantik yang dipakai untuk mendefiniskan program komputer. Adapun Bahasa pemrograman yang digunakan dalam pengerjaan sistem ini adalah *java script* dengan menggunakan *Framework Node Js, React Js, dan Material UI*.

* + 1. **. *Java script***

*Java script* adalah bahasa pemrograman yang popular. *Java script* adalah bahasa pemrograman yang digunakan untuk HTML dan *web*, untuk Server, PC, laptop, tablet dan lebih banyak lagi. Kode pemrograman *Java script* dapat disisipkan kedalam halaman HTML Pada awalnya, *Java script* mulai diperkenalkan di *browser* Netscape Navigator 2. Namun waktu itu namanya bukan *java script*, namun *LiveScript*. Mengingat pada waktu itu teknologi Java sedang panas-panasnya atau sedang tren, maka pihak Netscape memutuskan untuk mengganti namanya menjadi *Java script*, yang sepertinya nama tersebut lebih *marketible* dibandingkan *LiveScript*. Selanjutnya pihak Microsoft (rival Netscape) pun mulai ikut-ikutan memfasilitasi *web* *browser* buatannya, ‘Internet Explorer’, supaya bisa mendukung *Java script*. Namun mungkin karena gengsi, pihak Microsoft memberi nama bahasa yang lain, yaitu *Java script*. Mulai saat itu, Netscape dan Microsoft mulai berlomba-lomba mengembangkan bahasa tersebut dalam versi yang berlainan. Oleh sebab persaingan itulah terkadang suatu *Java script* mungkin bisa bekerja dengan baik di *browser* Netscape, tapi tidak demqikian halnya di IE, begitu pula sebaliknya (Permana 2016).

Pada awalnya, *Java script* mulai diperkenalkan di *browser* Netscape Navigator 2. Namun waktu itu namanya bukan *Java script*, namun *LiveScript*. Mengingat pada waktu itu teknologi Java sedang panas-panasnya atau sedang tren, maka pihak Netscape memutuskan untuk mengganti namanya menjadi *Java script*, yang sepertinya nama tersebut lebih marketible dibandingkan LiveScript. Selanjutnya pihak Microsoft (rival Netscape) pun mulai ikut-ikutan memfasilitasi *web* *browser* buatannya, ‘Internet Explorer’, supaya bisa mendukung *Java script*. Namun mungkin karena gengsi, pihak Microsoft memberi nama bahasa yang lain, yaitu Jscript. Mulai saat itu, Netscape dan Microsoft mulai berlomba-lomba mengembangkan bahasa tersebut dalam versi yang berlainan. Oleh sebab persaingan itulah terkadang suatu *Java script* mungkin bisa bekerja dengan baik di *browser* Netscape, tapi tidak demikian halnya di IE, begitu pula sebaliknya (Permana 2016).

Ada dua jenis bagaimana *Java script* dibuat, pertama *Java script* ditulis dalam file yang terpisah dengan HTML, kedua *Java script* ditulis dalam HTML. *Java script* yang ditulis diluar HTML disebut Eksternal *Java script* dengan ektensi file .js. Dalam HTML, penulisan script diawali dengan. Script yang akan dijalankan harus diletakkan diantara <script> dan </script> tag <script> memiliki beberapa atribut, namun yang terpenting adalah atribut language dan type. Karena *Java script* bukna satu – satunya bahasa scripting, maka sangatlah perlu untuk memberitahukan kepada *browser* bahwa bahasa script yang digunakan adalah *Java script* dan selanjutnya *browser* akan menjalankan modul pendukung *Java script* untuk memprosesnya (Permana 2016).

* + 1. . **Node** **JS**

Node JS merupakan platform server yang dibangun menggunakan *Java script* dan berjalan di dalam interpreter Chrome *Java script* runtime. Dibuat untuk pengembangan perangkat lunak berbasis *web* dengan cepat, aplikasi jaringan yang scalable. Node.js menggunakan event-driven, model non-blocking I/O yang membuatnya menjadi ringan dan efisien. Sangat baik digunakan untuk aplikasi waktu-nyata yang digunakan diberbagai perangkat (Muhammad Agung Rizkyana 2014) .

**Node.js** adalah perangkat lunak yang didesain untuk mengembangkan aplikasi berbasis *web* dan ditulis dalam sintaks bahasa pemrograman *Java script*. Bila selama ini *Java script* dikenal sebagai bahasa pemrograman yang berjalan di sisi *client* / *browser* saja, maka Node.js ada untuk melengkapi peran *Java script* sehingga bisa juga berlaku sebagai bahasa pemrograman yang berjalan di sisi server, seperti halnya PHP, Ruby, Perl, dan sebagainya. Node.js dapat berjalan di sistem operasi Windows, Mac OS X dan Linux tanpa perlu ada perubahan kode program. Node.js memiliki pustaka server HTTP sendiri sehingga memungkinkan untuk menjalankan server *web* tanpa menggunakan program server *web* seperti Apache atau Nginx.

Node JS dibuat dengan *engine* yang sama dengan *browser* *chrome* yang dikembangkan oleh google yang bersifat *open source* dimana Node JS memiliki keuntungan pada sifatnya yang *non blocking*. Sebagai contoh misalnya pada bahasa program biasa bila terdapat sebuah fungsi A yang berjalan, maka umumnya fungsi A harus selesai terlebih dahulu kemudian menjalankan fungsi B. tetapi berbeda dengan Node JS yang sifatnya parallel, atinya Node JS akan mengerjakan hal – hal yang sama dan ketika sudah selesai dengan fungsi yang ada maka dapat ditambahkan sebuah fungsi callback. *Callback* ini yang akan melihat apakah tugas A sudah selesai dikerjakan dan tidak akan menunggu A selesai tetapi juga akan sambil menjalankan tugas B.

Jadi pada program yang lain sebuah proses akan diselesaikan terlebih dahulu baru dapat ke proses berikutnya. Tapi pada Node JS berbeda karena dia berbasis *Java script* maka dia tidak akan menunggu proses A itu selesai tetapi semua proses akan dijalankan satu-satu lalu akan menjalankan proses berikutnya yang kita sudah tentukan. Dengan kata lain Node JS bisa **mengatasi *multiple request* secara bersamaan** dan prosesnya itu tidak langsung diselesaikan tetapi menjalankan beberapa proses terlebih dahulu.

* + 1. . **React js**

ReactJS, juga di kenal dengan nama React atau ReactJS, adalah pustaka *java script* bersifat *open source* digunakan untuk membangun antar muka pengguna. ReactJS digunakan untuk mengatasi tampilan pada aplikasi berhalaman tunggal (*single page application)* dan pengembangan aplikasi mobile (Khuat 2018) .

React dikembangkan oleh facebook dengan berusaha untuk menyediakan kecepatan, kesederhanaan, dan skalabilitas. Beberapa fitur utama react adalah JSX, komponen *state*ful, dan Virtual Document Object Model.

1. **JSX**

JSX adalah ekstensi sintaks *Java script* yang terlihat mirip XML. JSX sintaks HTML atau XML yang digunakan pada React JS dan memperluas ECMAScript sehingga XML/HTML dapat bekerja pada kode *Java script* atau React. Pada dasarnya, JSX dapat diperlakukan seperti ekspresi pada *Java script*, artinya elemen JSX dapat disimpan dalam variabel,function,object, array, dan lain – lain seperti pada umumnya *Java script* (Anonym n.d.) .

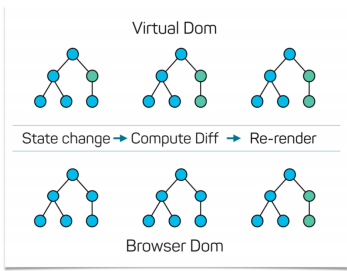
1. ***Stateful Components***

React memungkinkan pengguna untuk membagi antar muka pengguna menjadi independen, dan dapat digunakan berkali kali dengan melakukan pemanggilan komponen React. Komponen React meggunakan implementasi metode render yang mengambil dan menerima masukan dan mengembalikan hasil masukan menjadi sebuah tampilan antar muka. Setiap komponen memiliki beberapa metode *lifecycle* yang dapat mengganti untuk melakukan eksekusi kode pada waktu tertentu selama proses berlangsung. Pemanggilan method dapat dilakukan menggunakan API (*Aplication Programming Interface*) React.

*State* adalah objek *Java script* sederhana yang digunakan untuk merekam dan memberi reaksi sesuai perintah pengguna. Setiap kelas berbasis komponen mendefinisikan objek *state*nya sendiri. Kapanpun komponen *state* berubah, komponen, dan semua komponen turunan, secepat mungkin akan dilakukan *re-render*. *State* dapat menyimpan sebuah nilai dan dapat dikirim melalui antar komponen turunan dengan menggunakan *props.*

1. ***Virtual Document Object Model***

HTML DOM (*Document Object Model)* awalnya digunakan pada halamam statis yang tidak bekerja pada halaman antarmuka yang bersifat dinamis. Ketika DOM mengalami pembaharuan, DOM akan melakukan pembaharuan setiap *node* tanpa melakukan *reload* halaman *web*. Ini sering digunakan pada *single page application* yang memiliki ratusan halaman yang dihasilkan secara dinamis melalui aksi pengguna. Pada halaman dinamis, HTML DOM harus melakukan pengecekan perubahan setiap *node* data pada interval reguler. Hal ini dapat mengurangi performansi aplikasi. Sehinga virtual hadir sebagai solusi terhadap masalah tersebut (Khuat 2018).

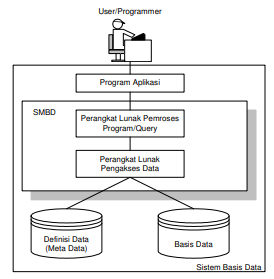


Gambar 2.8 : Perbedaan Virtual DOM dan *Browser* DOM dalam beberapa tahap perubahan (Khuat 2018)

1. **Basis data**

Data merupakan fakta mengenai suatu objek seperti manusia, benda, peristiwa, konsep, keadaan dan sebagainya yang dapat dicatat dan mempunyai arti secara implisit. Data dapat dinyatakan dalam bentuk angka, karakter atau simbol, sehingga bila data dikumpulkan dan saling berhubungan maka dikenal dengan istilah basis data (basis data). Sedangkan menurut George Tsu-der Chou basis data merupakan kumpulan informasi bermanfaat yang diorganisasikan ke dalam aturan yang khusus. Informasi ini adalah data yang telah diorganisasikan ke dalam bentuk yang sesuai dengan kebutuhan seseorang . Menurut *Encyclopedia of Computer Science and Engineer*, para ilmuwan di bidang informasi menerima definisi standar informasi yaitu data yang digunakan dalam pengambilan keputusan. Definisi lain dari basis data menurut Fabbri dan Schwab adalah sistem berkas terpadu yang dirancang terutama untuk meminimalkan duplikasi data. Menurut Ramez Elmasri mendefinisikan basis data lebih dibatasi pada arti implisit yang khusus, yaitu (Haidar Dzacko 2007):

* 1. Basis data merupakan penyajian suatu aspek dari dunia nyata (real world).
  2. Basis data merupakan kumpulan data dari berbagai sumber yang secara logika mempunyai arti implisit. Sehingga data yang terkumpul secara acak dan tanpa mempunyai arti, tidak dapat disebut basis data.
  3. Basis data perlu dirancang, dibangun dan data dikumpulkan untuk suatu tujuan. Basis data dapat digunakan oleh beberapa user dan beberapa aplikasi yang sesuai dengan kepentingan user.

Dari beberapa definisi-definisi tersebut, dapat dikatakan bahwa basis data mempunyai berbagai sumber data dalam pengumpulan data, bervariasi derajat interaksi kejadian dari dunia nyata, dirancang dan dibangun agar dapat digunakan oleh beberapa pengguna untuk berbagai kepentingan.

Gambar 2.9 : Konsep Sistem Basis data (Haidar Dzacko 2007)

C. J. Date menyatakan bahwa sistem basis data dapat dianggap sebagai tempat untuk sekumpulan berkas data yang terkomputerisasi dengan tujuan untuk memelihara informasi dan membuat informasi tersebut tersedia saat dibutuhkan (Haidar Dzacko 2007).

Basis data yang membutuhkan sebuah media untuk melakukan penyimpanan dan pengelolaan data yang ada yang diolah oleh program sql maupun nosql. Keduanya adalah media yang digunakan untuk melakukan pengolahan basis data yang digunakan sesuai dengan kebutuhan pengembang.

1. **SQL Basis data**

SQL atau *Structured Query Language* adalah sebuah bahasa yang digunakan untuk mengakses data dalam basis data relasional (Wikipedia,2018). SQL merupakan sebuah bahasa komputer yang mengikuti standar ANSI (American Nasional Standard Institute) yang digunakan dalam manajemen basis data relasional. Dengan SQL, kita dapat mengakses basis data, menjalankan query untuk mengambil data dari basis data, menambahkan data ke basis data, menghapus data di dalam basis data, dan mengubah data di dalam basis data. Saat ini hampir semua server basis data yang ada mendukung SQL untuk melakukan manajemen datanya.

Sejarah SQL dimulai dari artikel seorang peneliti dari IBM bernama EF Codd yang membahas tentang ide pembuatan basis data relasional pada bulan Juni 1970. Artikel ini juga membahas kemungkinan pembuatan bahasa standar untuk mengakses data dalam basis data tersebut. Bahasa tersebut kemudian diberi nama SEQUEL (Structured English Query Language). Setelah terbitnya artikel tersebut, IBM mengadakan proyek pembuatan basis data relasional berbasis bahasa SEQUEL. Akan tetapi, karena permasalahan hukum mengenai penamaan SEQUEL, IBM pun mengubahnya menjadi SQL. Implementasi basis data relasional dikenal dengan System/R. Di akhir tahun 1970-an, muncul perusahaan bernama Oracle yang membuat server basis data populer yang bernama sama dengan nama perusahaannya. Dengan naiknya kepopuleran Oracle, maka SQL juga ikut populer sehingga saat ini menjadi standar *de facto* bahasa dalam manajemen basis data (Haidar Dzacko 2007).

1. **NoSQL Basis data**

Istilah NoSQL diciptakan oleh Carlo Strozzi pada tahun 1998 dan mengacu pada basis data 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 basis data ini adalah SQL memiliki skema yang kaku sementara basis data 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, basis data NoSQL telah memiliki popularitas yang tinggi, sehingga basis data ini diklaim lebih baik dari basis data SQL, Basis data 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 (No dan Junaidi 2016).

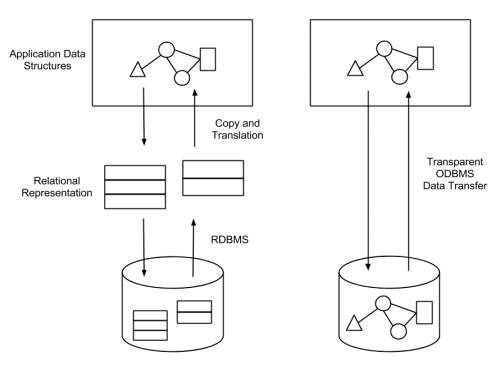
Teknologi NoSQL memiliki keunggulan sendiri dalam melakukan manajeman basis data sehingga penggunaan jenis basis data ini lebih diminati oleh pengembang. Ada banyak jenis basis data NoSQL yang sering digunakan oleh pengembang sesuai dengan kebutuhan, seperti IndexedDB, RXDB, PouchDB, dan CouchDB. keempat basis data tersebut memiliki keunggulan yaitu mampu menjadi basis data yang memungkinkan pengembang untuk membuat sebuah media penyimpanan *offline* bagi sistem yang bekerja, sehinga cocok digunakan pada sebuah aplikasi yang bekerja secara *online* maupun *offline*.

* + 1. **IndexedDB**

IndexedDB adalah sebuah API untuk penyimpanan lokal sisi klien, yang berarti teknologi ini mampu meyimpan data terstruktur. Pada tahun 2009, Oracle menawarkan IndexedDB sebagai penyimpanan lokal baru pada *web* *browser* standar. Pada jenis basis data ini, data disimpan sebagai *key value* *pair* tapi *key* dapat diartikn sebagai property dari objek yang disimpan dalam nilai (Mozahhebi 2013).

Umumnya IndexedDB tidak memiliki batas kapasitas, tetapi setiap *browser* memiliki aturan sendiri dalam kapasitas. Sebagai contoh, Firefox tidak menekankan pada kapasitas, namun ketika data lebih dari 50MB maka aka nada permintaan izin untuk melakukan penyimpanan data. Begitu pula dengan Google Chrome yang memungkinkan aplikasi untuk menggunkan 20% dari penyimpanan lokal, walaupun akan tidak mungkin untuk melakukan penyimpanan lebih (Mozahhebi 2013).

IndexedDB sebagai *object oriented* basis data NoSQL, yang membedakannya dengan kebanyakan basis data tradisional. Basis data NoSQL memiliki skema, dimana menyimpan dan mengembalikan data tidak menggunakan *structured query language* (*SQL)*. Selain itu IndexedDB bukan merupakan basis data relasional (RDBMS), dimana tabel memiliki baris dan kolom. Sebenarnya, IndexedDB adalah basis data berbasi objek yang mengolah data dalam bentuk objek (Mozahhebi 2013).

Gambar 2.10 : Perbandingan *Relasional* basis datadan

basis data *object* (Mozahhebi 2013)

IndexedDB API dapat digunakan melalui *Java script* dalam hal melakukan menyimpan,mengambil, menghapus, dan melakukan pembaharuan data. API ini menggunakan DOM *events* untuk menggunakan operasi basis data. Semua hirarki objek seperti JSON mendukung API ini.

* + 1. **RXDB**

RXDB adalah *object store* yang memiliki kinerja yang cepat yang dibangun diatas SQLIite (Nylas,2016) . Rxdb digunakan sebagai media untuk melakukan komunikasi ke sisi server dari sebuah aplikasi *website*. Segala bentuk transaksi data dari user menuju basis data akan ditangani oleh Rxdb.

* + 1. **Couch DB**

CouchDB merupakan salah satu basis data [NoSQL](https://www.codepolitan.com/tag/nosql/) berbasis dokumen yang masuk dalam pembinaan Apache Foundation. Bersama dengan Cassandra, Hadoop, Tomcat, Lucene, [CouchDB](https://www.codepolitan.com/tag/couchdb) tinggal serumah dalam dunia open source. CouchDB dibangun menggunakan bahasa pemrograman Erlang yang mengandalkan pada reliabilitas dan konkurensi. CouchDB juga menjadi salah satu basis dalam pengembangan IBM Cloudant, sebuah solusi basis data berbasis cloud yang ditawarkan IBM kepada enterprise (Ridwan Fajar,2016).

* + 1. **Pouch DB**

PouchDB adalah basis data bersifat *open source* dibangun menggunakan *Java script* dan merupakan pengembangan dan model dari CouchDB. dengan menggunakan API ini kita mampu mebangun dan mengembangkan sebuah aplikasi yang mampu bekerja secara *online* maupun *offline*. Teknologi ini menggunakan *Web*SQL dan IndexedDB untuk melakukan penyimpanan data.

Pouch DB bekerja ketika aplikasi berjalan *offline* , data akan disimpan secara lokal menggunakan *Web*SQL dan IndexedDB pada *browser*. Dan ketika aplikasi kembali *online*, data akan di sinkronisasi menggunakan CouchDB (Colanus, Drajana, and Selection 2017).

**BAB III  
METODOLOGI PENELITIAN**

1. **Lokasi dan Waktu Penelitian**

Proses penelitian dilakukan sejak bulan Mei 2018. Dan loka si 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 8
2. *Web* *Browser* Google Chrome
3. Sublime Text 3
4. CouchDB
5. Node JS
6. Library React JS
7. Apache Jmeter
8. Material UI
9. IndexedDB
10. **Prosedur Penelitian**

Tahapan pada penelitian ditunjukkan pada gambar 3.1

Studi Literatur

Analisis Data

Desain dan Perancangan sistem

Pengambilan Data

Performansi Sistem

Implementasi Desain dan Perancangan Sistem

Persiapan Instrumen Pembuatan Sistem

Sinkronisasi data *offline*

Pemasangan *Service worker* pada sistem

Gambar 3. 1. Diagram Tahapan Penelitian

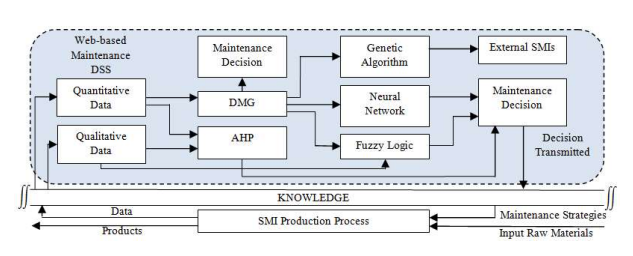
Tahapan secara garis besar dijelaskan sebagai berikut.:

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, proses *browser* , dan penggunaan sumber daya menggunakan jmeter dengan menggunakan skema tertentu.
8. Setelah dilakukan pengambilan data, maka data yang diperoleh dari berbagai skema kemudian dilakukan analisis. Analisis dilakukan dengan membandingkan sistem yang menggunakan teknologi *service worker,* dan tanpa menggunakan *service worker* serta membandingkan 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. 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.

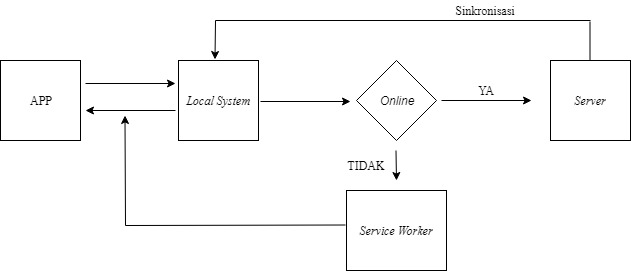
Sistem yang dibangun akan memberikan kemampuan dalam melakukan analisis keputusan pemeliharaan mesin pada IKM. sistem yang dibangun menerapkan teknologi DSS (*Desicion Support System*) untuk melakukan manajemen pemeliharaan mesin dengan model CMMS (*Computerized Maintenance Management System*), menerapkan 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 di*input* 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.

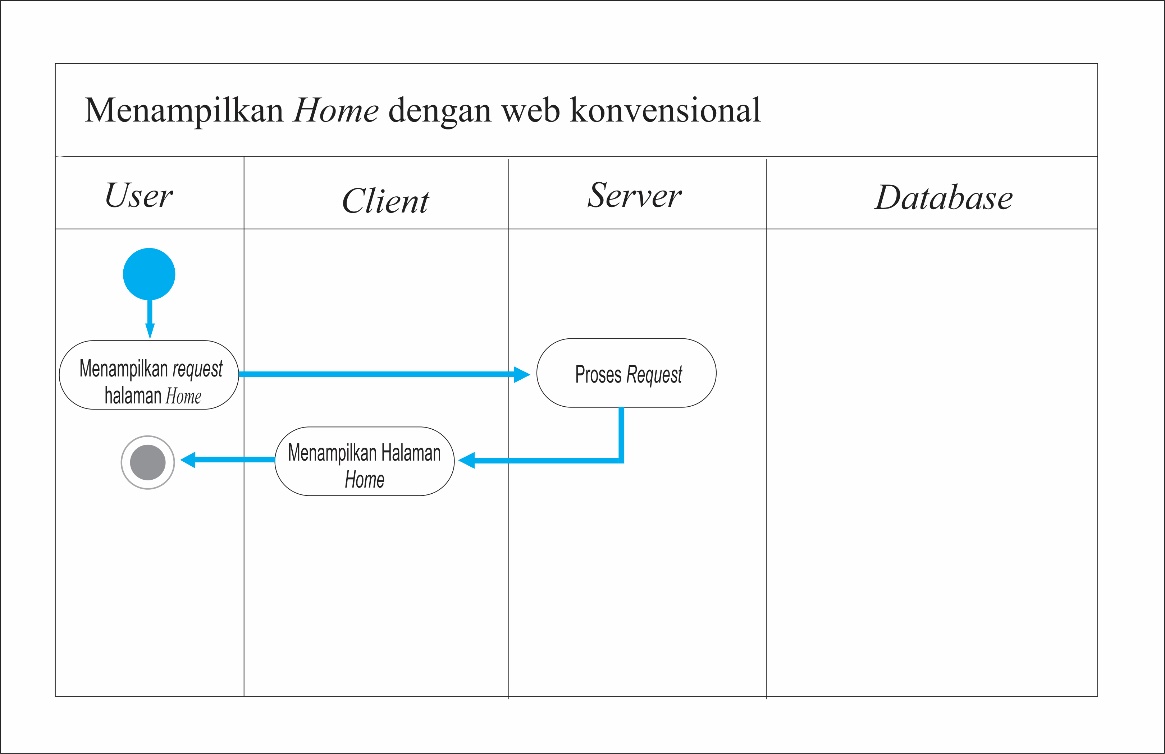
DSS yang memanfaatkan teknologi *web* memiliki masalah klasik yang sama yaitu tidak dapat di jalankan jika terputus dari server. Sehingga digunakan teknologi *web* dengan menggunakan model *offline first app* dengan memanfaatkan *service worker.* Sistem *offline first app* akan dibuat menggunakan library 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

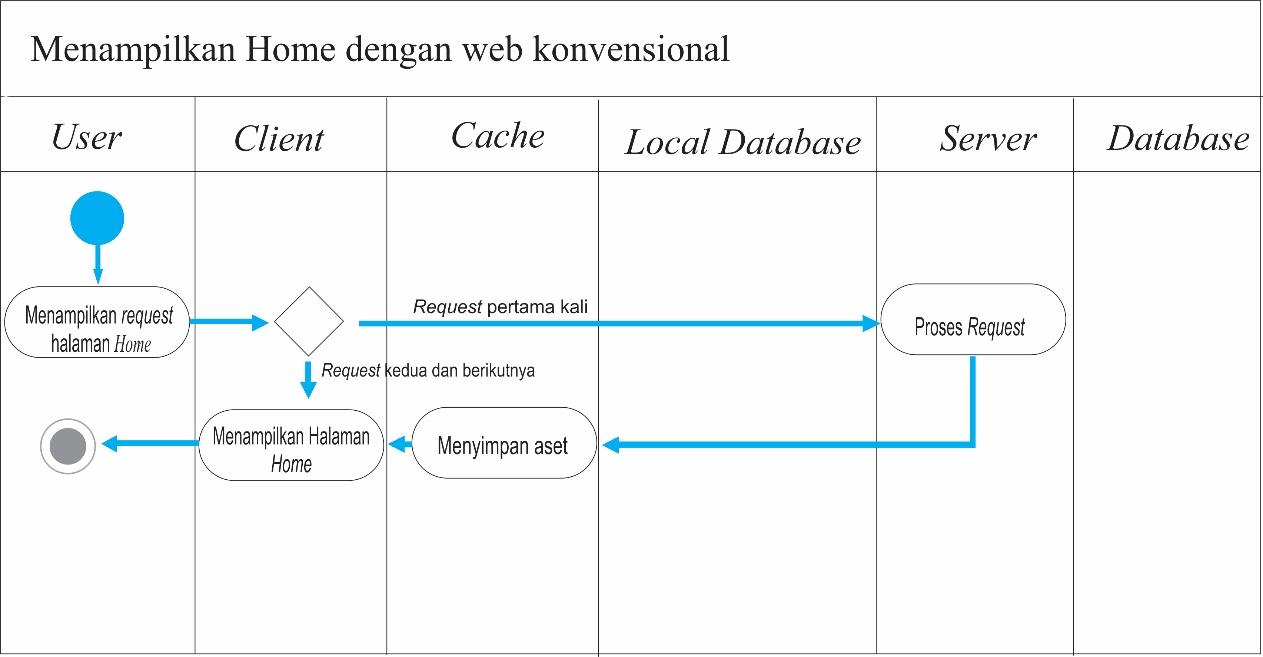
Sistem yang dibuat memiliki alur seperti pada gambar 3.2. Aplikasi akan mengirimkan *request* namun *request* hanya akan diterima oleh lokal sistem yaitu sistem yang berjalan pada sisi *client* dengan menggunakan media penyimpanan local indexedDB untuk pertama kali. selanjutnya sistem akan melakukan 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* dan Pengujian *Black box***

*System Activity* dijelaskan melalui *Activity* Diagram. *Activity* diagram mengandung diagram *use case* *entities* seperti halaman *home*, halaman mesin, dan halaman pemeliharaan. Selain itu sistem yang dibangun akan dilakukan pengujian dan di bandingkan dengan *diagram activity* yang dibuat.



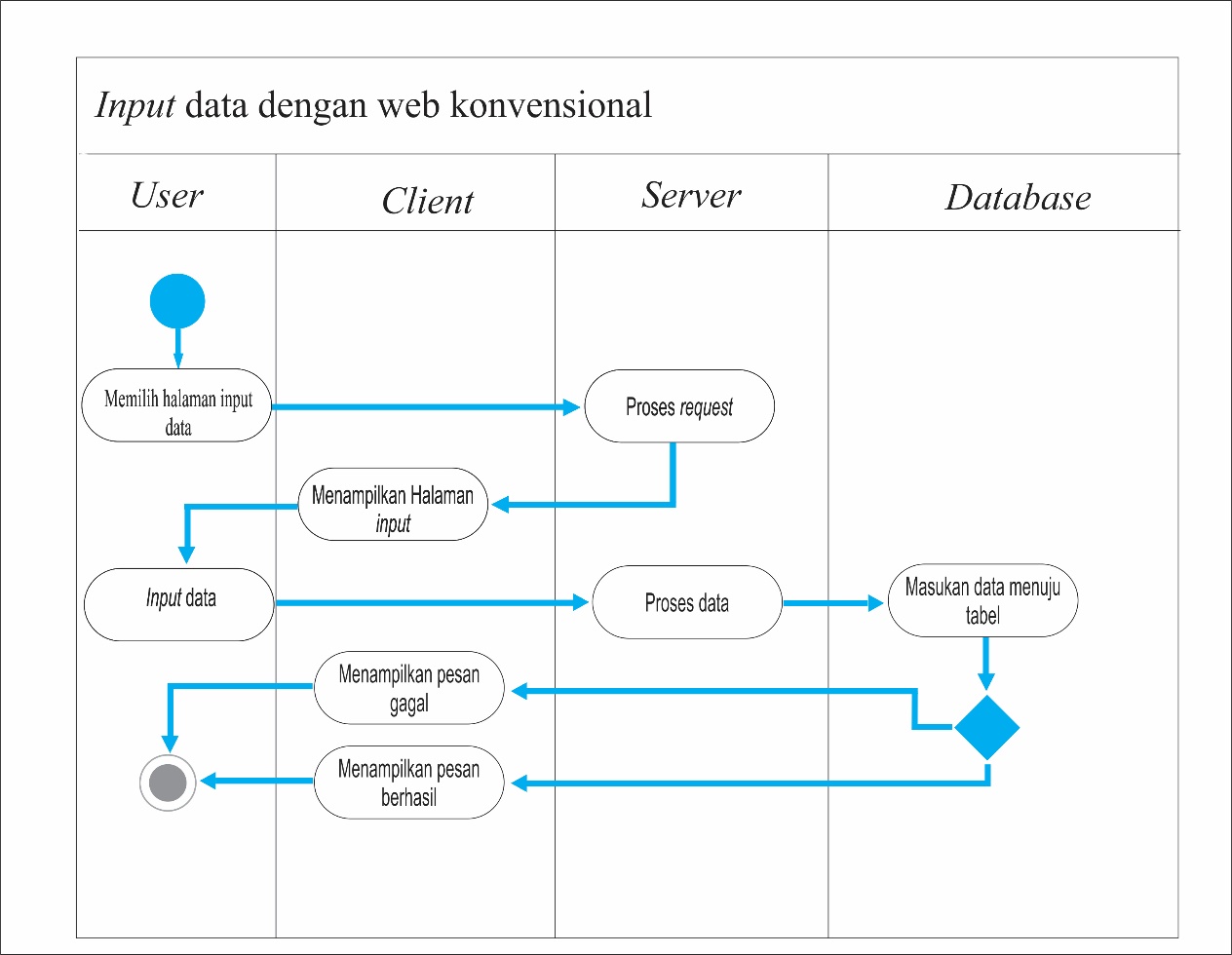
Gambar 3. 4. Diagram activity web konvensional .



Gambar 3. 5. Diagram activity web dengan service worker

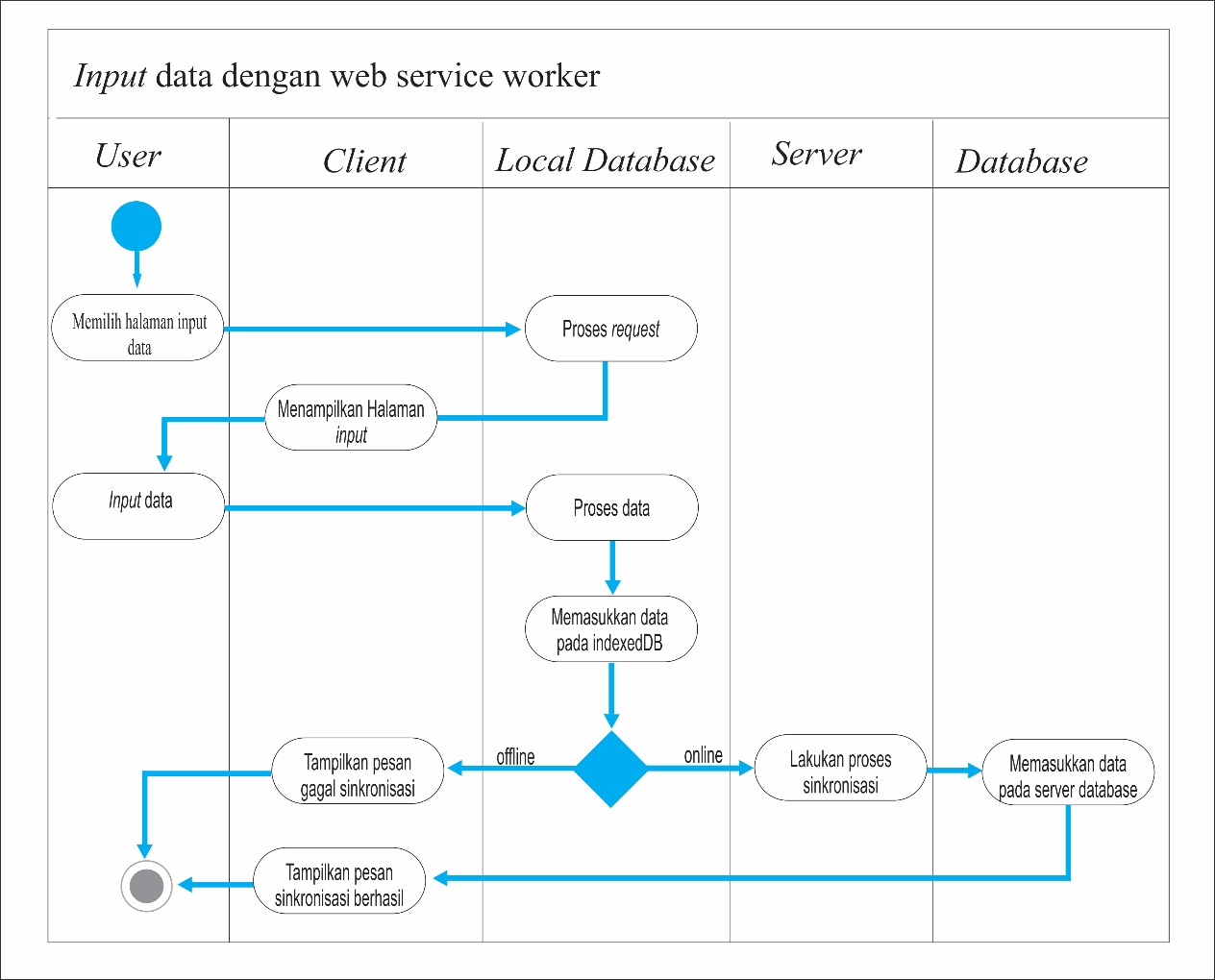
Pada gambar 3.4 *request* yang dilakukan oleh user kemudian akan dikirim dengan kondisi apabila *request* dilakukan pertama kali maka *request* akan menuju ke server dan akan menyimpan asset menuju *lokal* basis data dan *cache* kemudian menampilkan halaman *request* kembali ke pengguna. Namun ketika *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. 6. 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.



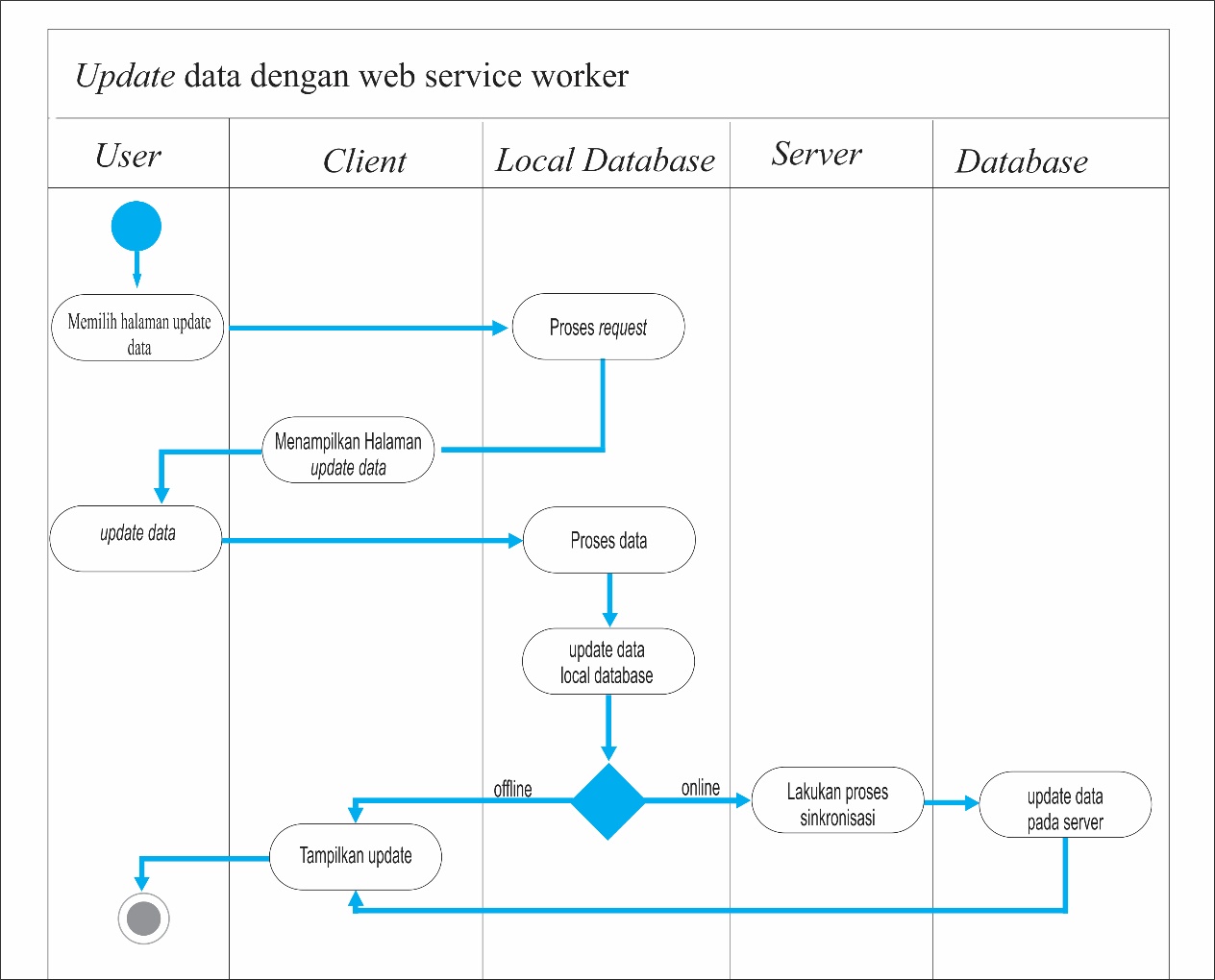
Gambar 3. 7. Diagram activity input data pada web dengan service worker

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.

Adapun penjabaran proses *input* data yang digambarkan pada gambar 3.7 dijelaskan pada tabel 3.1.

Tabel 3. 1. Tabel rancangan input data

|  |  |  |  |
| --- | --- | --- | --- |
| No | Aksi | Masukan | Keluaran |
| 1 | *User* memilih halaman *input* data mesin yang akan melakukan proses *request* menuju lokal sistem untuk pertama kali dan akan mengirim umpan balik menuju *client*. | Masukan berupa klik kursor | Memunculkan halaman *input* data |
| 2 | *User* memasukkan data *inputan* berupa data mesin atau pemeliharaan mesin yang kemudian akan diproses oleh sistem dan dikirim menuju lokal *database* *indexedDB.* | Masukan berupa *text* atau karakter pada *form text* dan klik pada tombol *submit* data | Memunculkan data yang telah dimasukkan pada *database* lokal ataupun *database server* pada halaman mesin. |

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

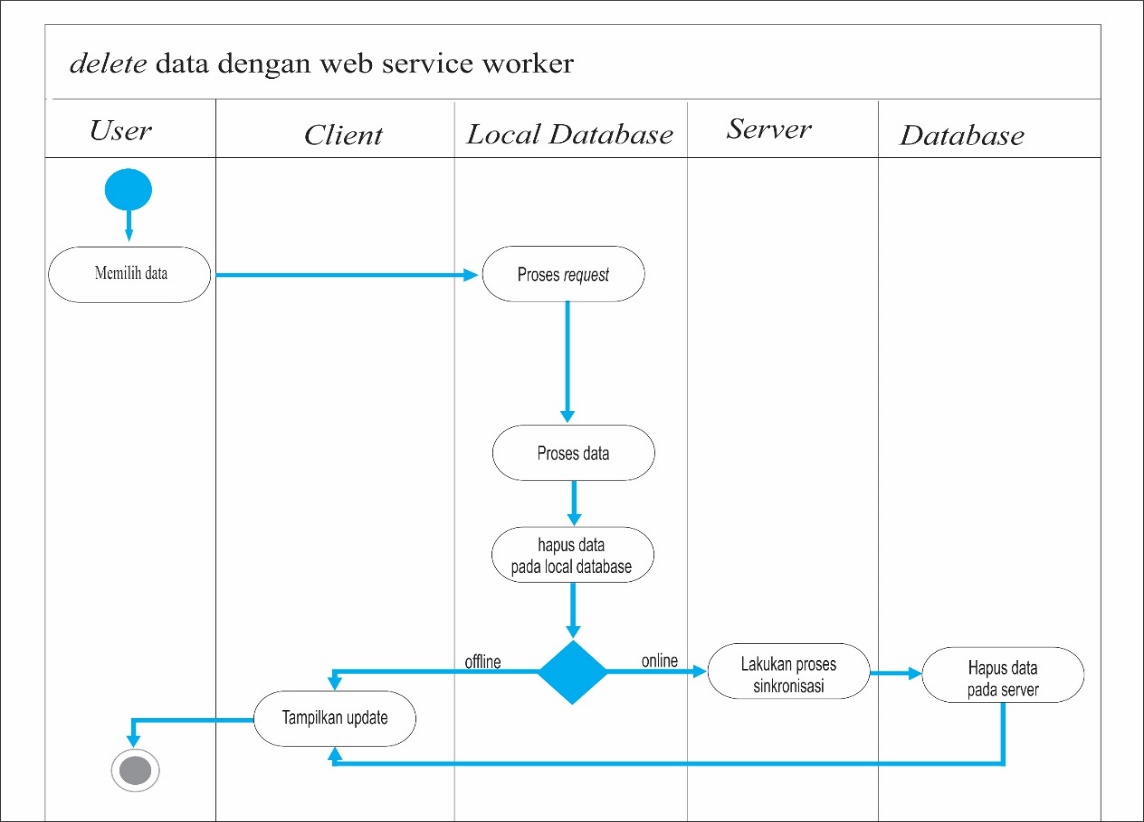
Gambar 3. 8. Diagram activity pembaharuan data data pada web dengan service worker

Adapun penjabaran proses *update* data yang digambarkan pada gambar 3.8 dijelaskan pada tabel 3.2.

Tabel 3. 2. Tabel perancangan update data

|  |  |  |  |
| --- | --- | --- | --- |
| Percobaan | Aksi | Masukan | Keluaran |
| 1 | *User* memilih halaman *update* data mesin. *Request* kemudian akan diterima oleh sistem lokal dan akan menampilkan umpan balik. | Masukan berupa klik kursor | Memunculkan halaman *update* data |
| 2 | *User* melakukan pembaharuan data data mesin yang sudah ada atau pemeliharaan mesin yang kemudian akan diproses oleh sistem dan dikirim menuju lokal *database* *indexedDB* untuk pembaharuan data*.* | Masukan berupa *text* atau karakter pada *form text* dan klik pada tombol *submit* data | Memunculkan data yang telah diperbaharui pada *database* lokal ataupun *database server* pada halaman pembaharuan data mesin. |

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. 9. Proses hapus data pada web dengan menggunakan service worker

Adapun penjabaran proses *hapus* data yang digambarkan pada gambar 3.9 dijelaskan pada tabel 3.3.

Tabel 3. 3. Tabel pengujian black box hapus data

|  |  |  |  |
| --- | --- | --- | --- |
| No | Aksi | Masukan | Keluaran |
| 1 | *User* memilih data yang ingin di hapus. | Masukan berupa klik kursor | Memunculkan halaman mesin dengan data yang telah terhapus |

1. ***Hardware environtment***

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



Gambar 3. 10. 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)

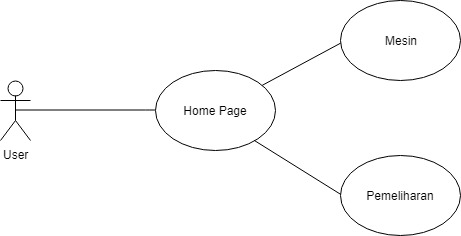
Sistem dibangun dengan membuat tampilan sistem dinamis dengan menggunakan ReactJS dan Material UI untuk membuat tampilan sistem lebih bagus.

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

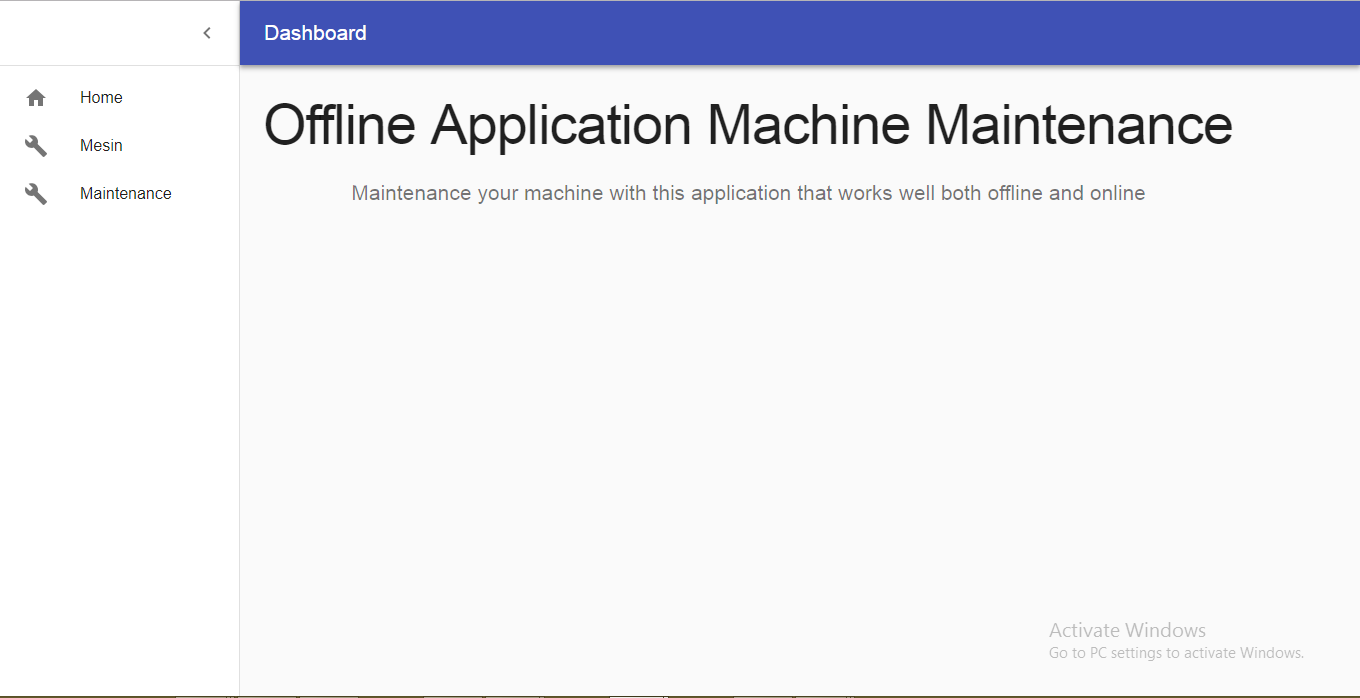
Basis data digunakan untuk melakukan penyimpanan data masukan dari pengguna. Sistem menggunakan dua jenis basis data yaitu lokal basis data dan server basis data. Lokal basis data yaitu indexedDB digunakan sebagai media penyimpanan utama, sehingga sistem dapat mendukung teknologi *offline first app*.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

Sistem pemeliharaan mesin industri 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. 

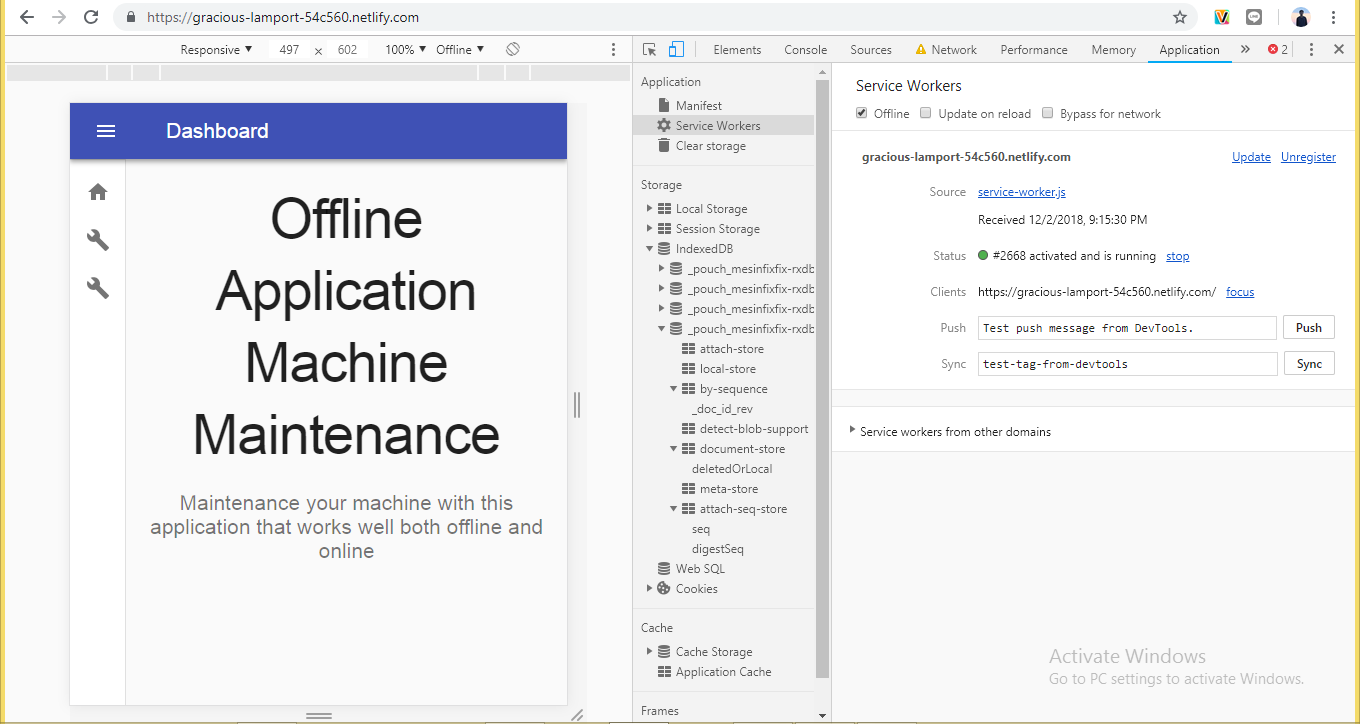
Gambar 3. 11. Use case diagram

*Use case* dari halaman *home* berinteraks langsung dengan pengguna. Halaman didesain untuk menampilkan *hyperlink* keseluruh halaman *web* yang lain yaitu halaman mesin dan halaman pemeliharaan mesin.

1. **Hasil Pembuatan Sistem**

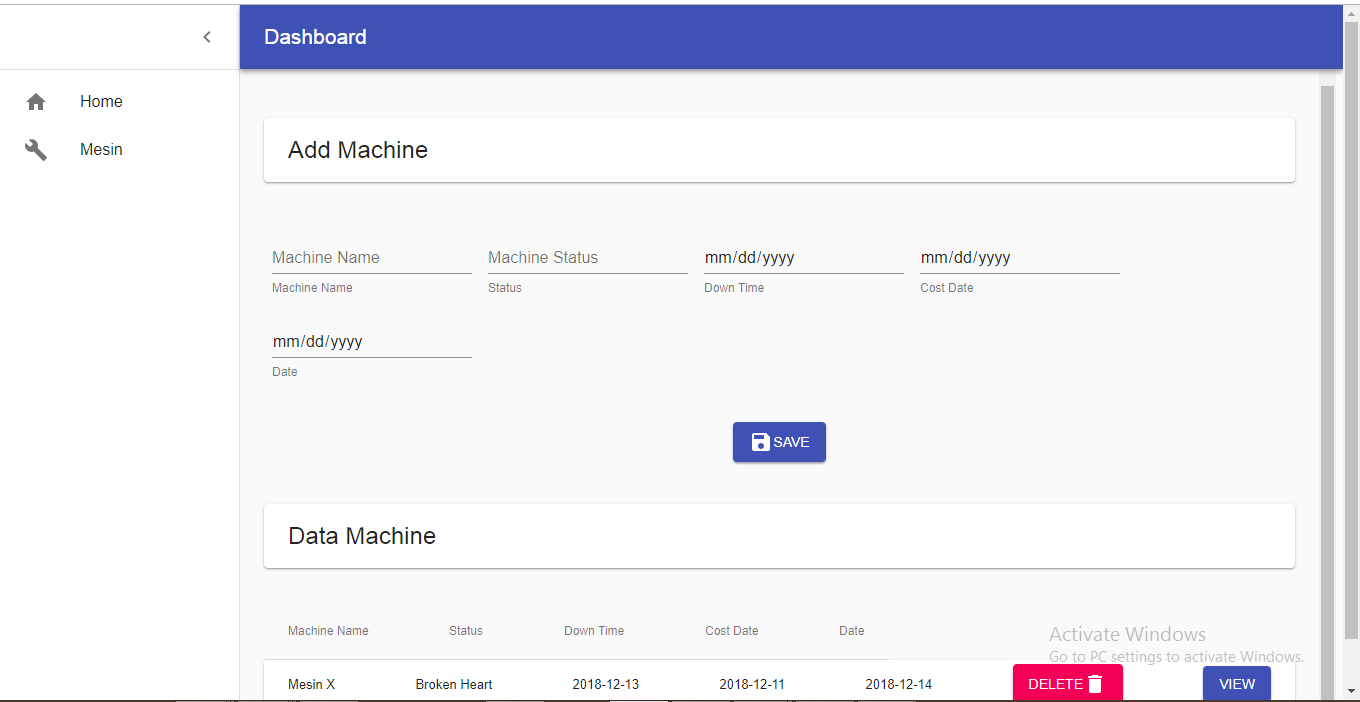
Setelah semua perancangan sistem telah dilakukan maka selanjutnya adalah membangun sistem yang akan dianalisis. Sistem dibangun dengan menggunakan *library* ReactJS sebagai sisi *client* dan menggunakan RxDB untuk melakukan transfer data menuju basis data lokal maupun server basis data. 

Gambar 3.12. Halaman utama sistem



Gambar 3. 13. Service worker pada aplikasi web

Gambar 3.9 menunjukkan halaman utama sistem yang telah dibuat menggunakan library ReactJS dan telah terintegrasi dengan *service worker* yang 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.



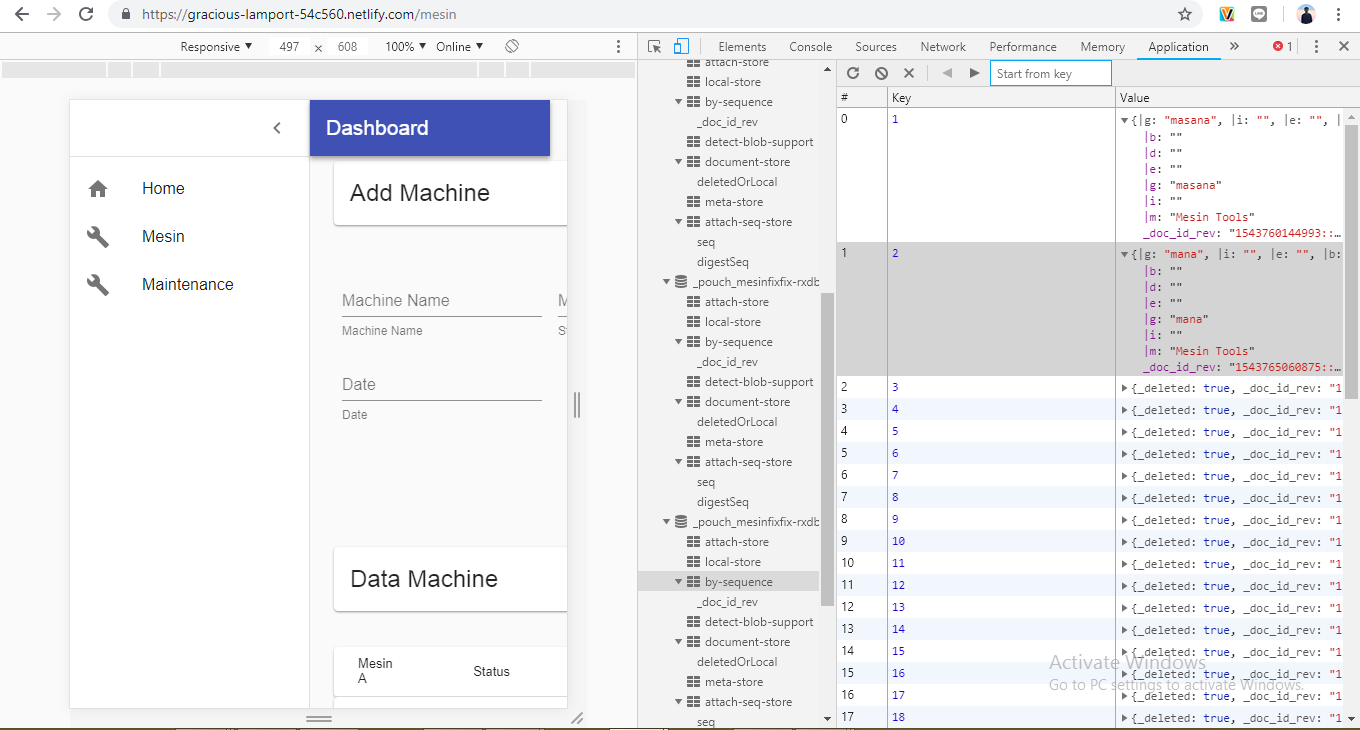
Gambar 3. 14. Halaman Mesin

Sistem yang dibangun menggunakan format basis data nosql sesuai dengan tabel 3.4.

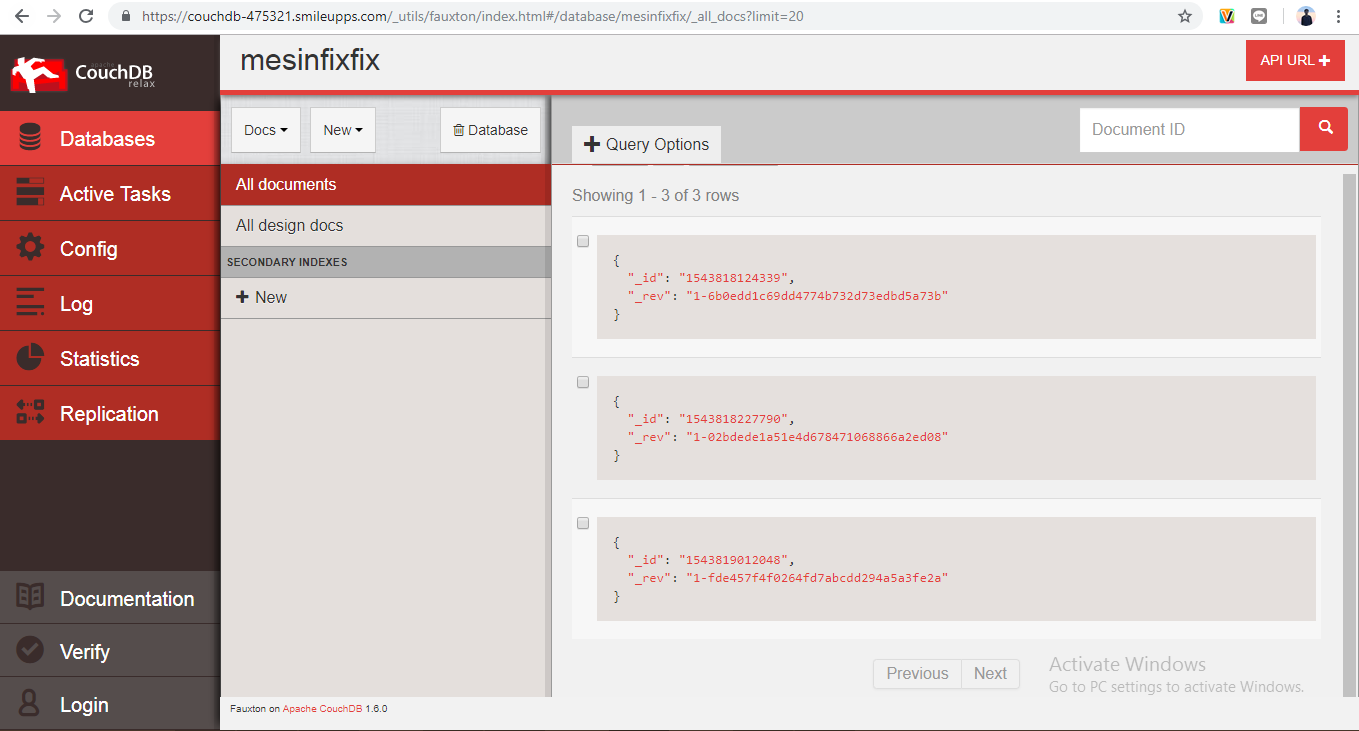
Tabel 3.4. Tabel format basis data

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |
| Nama Elemen Data | Deskripsi | Sumber | Jenis Field |
| 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. 15. Basis data lokal indexedDB



Gambar 3. 16. Basis data server couch db

1. **Skenario Pengujian**

Pengujian dilakukan dengan menggunakan parameter sebagai berikut :

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

*Respon time* dari *web* dengan menggunakan *react JS* dan *web* konvensional akan dihitung dengan melakukan simulasi masing - masing pada kondisi jaringan 3G dan 4G. 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 untuk setiap kondisi jaringan 3G dan 4G pada masing masing *web* untuk 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 1000 *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 percobaandari setiap *thread.*

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

Throughput = ( 3. 2)

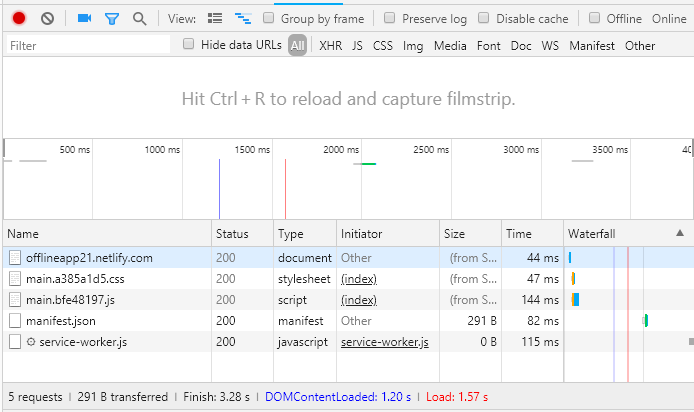
Persamaan 3.3 merupakan persamaan yang digunakan untuk menghitung throughput untuk satu *request* saja. Untuk *request* yang berulang atau lebih dari satu maka digunakan persamaan berikut.

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*, *web tanpa service worker* dan *web* konvensional dengan melakukan akses masing masing *web* sebanyak sepuluh kali akses kemudian mengambil nilai rata – rata waktu akses.



Gambar 3. 17. 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 persentase awal prosesor dan angka persentase prosesor saat melakukan eksekusi *request* kemudian mengurangi persentase awal prosesor dengan persentase prosesor saat melakukan eksekusi *request.* 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***
   1. **Kondisi jaringan 3G**

Hasil pengujian pada *web* *javascript* dan *web* konvensional secara *online* pada kondisi jarigan 3G 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 pada Jaringan 3G

|  |  |  |  |
| --- | --- | --- | --- |
| Pengujian | Jumlah *Thread* | *Web* *java script service worker*  (ms) | *Web* Konvensional (ms) |
| 1 | 1 | 54 | 77 |
| 2 | 50 | 64 | 113 |
| 3 | 100 | 139 | 190 |
| 4 | 150 | 261 | 298 |
| 5 | 200 | 282 | 319 |
| 6 | 250 | 342 | 390 |
| 7 | 300 | 408 | 447 |
| 8 | 350 | 422 | 509 |
| 9 | 400 | 452 | 974 |
| 10 | 450 | 513 | 1057 |
| 11 | 600 | 662 | 1204 |

|  |  |  |  |
| --- | --- | --- | --- |
| Pengujian | Jumlah *Thread* | *Web* *java script service worker*  (ms) | *Web* Konvensional (ms) |
| 12 | 700 | 991 | 1550 |
| 13 | 800 | 1251 | 1807 |
| 14 | 900 | 1771 | 1900 |
| 15 | 1000 | 2015 | 3929 |

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

Gambar 4. 1 Grafik hasil pengujian waktu rata – rata

* 1. **Kondisi Jaringan 4G**

Hasil pengujian pada *web javascript* dan *web* konvensional secara *online* pada kondisi jarigan 4G dengan menggunakan software apache Jmeter akan diperoleh hasil untuk masing - masing percobaan dengan waktu rata – rata yang ditunjukkan pada tabel 4.2.

Tabel 4. 2.Tabel Pengujian Waktu Rata - Rata pada Jaringan 4G

|  |  |  |  |
| --- | --- | --- | --- |
| Pengujian | Jumlah *Thread* | *Web* *java script service worker*  (ms) | *Web* Konvensional (ms) |
| 1 | 1 | 39 | 55 |
| 2 | 50 | 56 | 96 |
| 3 | 100 | 102 | 134 |
| 4 | 150 | 180 | 182 |
| 5 | 200 | 219 | 279 |
| 6 | 250 | 230 | 295 |
| 7 | 300 | 329 | 330 |
| 8 | 350 | 385 | 405 |
| 9 | 400 | 392 | 821 |
| 10 | 450 | 468 | 861 |
| 11 | 600 | 587 | 1049 |
| 12 | 700 | 885 | 1169 |
| 13 | 800 | 1091 | 1281 |
| 14 | 900 | 1278 | 1768 |
| 15 | 1000 | 1314 | 2176 |

Sedangkan untuk grafik yang diperoleh berdasarkan data yang ada ditunjukkan pada gambar 4.2.

Gambar 4. 2. Grafik Waktu Rata - Rata pada Jaringan 4G

Berdasarkan grafik yang diperoleh dari masing masing *web* dan setiap jaringan untuk waktu rata – rata pada kondisi jaringan 4G, menunjukkan bahwa waktu akses rata rata pada sebuah *web* PHP untuk satu *thread* menghasilkan angka yang lebih besar dibandingkan dengan *web java script*. Hal ini dikarenakan *web reactJS* memiliki fitur virtual DOM. Selain itu *web* *reactJS* yang menggunakan *Node JS* sebagai bahasa pemrograman sisi server yang memiliki *input* *output* *non blocking* yang dengan mudah melayani beberapa *event* secara bersamaan sehingga pengembang dapat dengan mudah membuat server lebih skalabel yang memaksimalkan penggunaan satu CPU.

*Web* dengan PHPmenggunakan cara *multi threaded execution* dalam proses *request* ke server dimana satu *host* untuk satu *thread* proses, sehingga cara ini akan memakan waktu lama dan akan membebankan server dibandingkan dengan *web react JS*.

1. ***Throughput***

Hasil Pengujian dari *throughput* pada *web* konvensional dan *web* *offline first app* ditunjukkan pada tabel 4.2 dan grafik pada gambar 4.3 :

Tabel 4. 3.Tabel hasil pengujian throughput pada web online

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pengujian | Jumlah *Thread* | *Web* dengan *service worker* (/sec) | *Web* tanpa *service worker* (/sec) | *Web* Konvensional (/sec) |
| 1 | 1 | 380.8 | 366.83 | 263.71 |
| 2 | 50 | 596.65 | 220.63 | 200.02 |
| 3 | 100 | 388.84 | 387.89 | 333.77 |
| 4 | 150 | 458.01 | 456.76 | 356.18 |
| 5 | 200 | 617.18 | 481 | 359.09 |
| 6 | 250 | 670 | 620 | 416.63 |
| 7 | 300 | 637 | 542 | 360.62 |
| 8 | 350 | 579 | 430 | 344 |
| 9 | 400 | 620 | 407 | 323.11 |
| 10 | 450 | 207 | 133.4 | 164.5 |

Gambar 4. 3 Grafik hasil throughput

Dari grafik yang diperoleh dari hasil pengujian untuk *throughput* menunjukkan bahwa untuk *website* tanpa *service worker* 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. 4. Tabel hasil pengujian akses pertama kali tanpa cache

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Percobaan | *Web* dengan *service worker* | | *Web* tanpa *service worker* | | *Web* konvensional | |
| *DomContentLoaded* (detik) | *Load* (detik) | *DomContentLoaded* (detik) | *Load* (detik) | *DomContentLoaded* (detik) | *Load* (detik) |
| 1 | 3.05 | 3.05 | 3.94 | 3.97 | 3.01 | 4.82 |
| 2 | 3.03 | 3.03 | 4.19 | 4.22 | 3.14 | 4.7 |
| 3 | 2.83 | 2.84 | 3.54 | 3.57 | 2.76 | 4.18 |
| 4 | 2.05 | 2.05 | 3.28 | 3.31 | 2.32 | 4.25 |
| 5 | 2.26 | 2.27 | 3.06 | 3.09 | 2.53 | 4.43 |
| 6 | 2.14 | 2.15 | 3.71 | 3.92 | 2.38 | 4.69 |
| 7 | 2.76 | 2.77 | 3.64 | 3.87 | 2.6 | 3.34 |
| 8 | 2.85 | 2.85 | 3.28 | 3.31 | 2.82 | 3.51 |
| 9 | 2.92 | 2.93 | 2.93 | 2.96 | 2.81 | 3.77 |
| 10 | 2.12 | 2.13 | 2.87 | 2.9 | 2.3 | 5.04 |

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. 5. Tabel pengujian online dengan cache

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Percobaan | *Web* dengan *service worker* | | *Web* tanpa *service worker* | | *Web* konvensional | |
| *DomContentLoaded* (detik) | *Load* (detik) | *DomContentLoaded* (detik) | *Load* (detik) | *DomContentLoaded* (detik) | *Load* (detik) |
| 1 | 0.433 | 2.607 | 3.444 | 3.512 | 0.492 | 1.904 |
| 2 | 0.445 | 0.452 | 3.36 | 3.37 | 0.578 | 0.567 |
| Percobaan | *Web* dengan *service worker* | | *Web* tanpa *service worker* | | *Web* konvensional | |
| *DomContentLoaded* (detik) | *Load* (detik) | *DomContentLoaded* (detik) | *Load* (detik) | *DomContentLoaded* (detik) | *Load* (detik) |
| 3 | 0.537 | 0.463 | 3.22 | 3.23 | 0.574 | 0.679 |
| 4 | 0.428 | 0.548 | 2.84 | 2.84 | 0.52 | 0.665 |
| 5 | 0.474 | 0.441 | 2.37 | 2.38 | 0.624 | 0.558 |
| 6 | 0.478 | 0.484 | 2.15 | 2.16 | 0.554 | 0.698 |
| 7 | 0.449 | 0.496 | 2.13 | 2.13 | 0.588 | 0.634 |
| 8 | 0.482 | 0.459 | 2.03 | 2.03 | 0.497 | 0.672 |
| 9 | 0.435 | 0.497 | 1.95 | 1.96 | 0.642 | 0.553 |
| 10 | 0.458 | 0.445 | 1.92 | 1.93 | 0.59 | 0.723 |

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 untuk masig – masing *web*. 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*.

Dibandingkan dengan *web* *service worker* dan *web* konvensional, *web tanpa service worker dengan menggunakan java script*memiliki waktu akses yang lebih tinggi. Hal ini disebabkan karena *web* tanpa *service worker* dibuat dengan menggunakan java script tanpa *service worker* yang memiliki banyak fungsi java script sehingga membutuhkan waktu yang lebih lama dibandingkan dengan *web* konvensional.

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. 6. Tabel pengujian offline dengan cache

|  |  |  |
| --- | --- | --- |
| No | *Web service worker* | |
| *DomContentLoaded* (ms) | *Load* (ms) |
| 1 | 171 | 338 |
| 2 | 158 | 317 |
| 3 | 159 | 298 |
| 4 | 147 | 286 |
| 5 | 156 | 284 |
| 6 | 149 | 300 |
| 7 | 150 | 304 |
| 8 | 146 | 307 |
| 9 | 151 | 305 |
| 10 | 144 | 299 |

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 tanpa jaringan.

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. 7. Tabel pengujian online input data

|  |  |  |  |
| --- | --- | --- | --- |
| Percobaan | *Web* dengan *service worker* | *Web* tanpa *service worker* | *Web* konvensional |
| 1 | 1.12 | 0.818 | 0.866 |
| 2 | 1.14 | 0.711 | 0.795 |
| 3 | 1.08 | 0.723 | 0.733 |
| 4 | 1.02 | 0.753 | 0.748 |
| 5 | 1.01 | 0.745 | 0.749 |
| 6 | 1.06 | 0.762 | 0.768 |
| 7 | 0.995 | 0.735 | 0.677 |
| 8 | 0.975 | 0.709 | 0.664 |
| 9 | 0.933 | 0.757 | 0.681 |
| 10 | 0.944 | 0.74 | 0.688 |

Gambar 4.7. Grafik waktu load input data

Dari hasil data yang diperoleh, waktu rata rata untuk *input* data pada *web* konvensional yaitu 0.7369 detik. Sedangkan untuk waktu rata – rata dari waktu *input* pada *web* tanpa *service worker* yaitu 0.7453 detik dan pada *web* *service worker* yang telah dilakukan percobaan *input* data sebanyak sepuluh kali diperoleh hasil yaitu 1.277 detik. Waktu yang digunakan *web* *service worker* untuk melakukan pemasukan data lebih besar dibandingkan dengan *web* lainnya.Hal ini dikarenakan ketika *web service worker* melakukan proses pemasukan data akan menjalankan fungsi singkronisasi sekaligus yaitu melakukan pemasukan data pada basis data local dan kemudian melakukan pemasukan data pada database server.

1. **Performansi CPU**

Untuk pengujian performansi CPU, dilakukan percobaan dengan menggunakan tool apache jmeter untuk melakukan skenario pengujian dengan 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. 8. Tabel pengujian performansi CPU

|  |  |  |  |
| --- | --- | --- | --- |
| Percobaan | *Web* dengan *service worker* (%) | *Web* tanpa *service worker* (%) | *Web* konvensional (%) |
| 1 | 0.7 | 1.1 | 1.7 |
| 2 | 2.8 | 5.2 | 6.1 |
| 3 | 5.5 | 7 | 12 |
| 4 | 6.5 | 7.2 | 13.6 |
| 5 | 7.7 | 8.3 | 14.1 |
| 6 | 9.6 | 10.2 | 24 |
| 7 | 10 | 11.3 | 26 |
| 8 | 13.3 | 14.1 | 26.2 |
| 9 | 13.8 | 17 | 27.1 |
| 10 | 16.5 | 19.4 | 38.5 |

Gambar 4. 8 Grafik perbandingan performansi CPU

Dari grafik pada gambar 4.8 menunjukkan bahwa performansi CPU untuk web konvensional PHP dan web tanpa service worker dengan menggunakan java script memiliki persentase yang lebih tinggi dibandingkan dengan web service worker. Hal ini disebabkan karena service worker mampu meringankan beban kerja dari server dengan adanya fungsi fetch event sehingga service worker mampu melakukan manajemen request data ke server. Apabila data yang di request berada pada memori cache browser maka request tidak akan sampai ke server sehingga beban kerja server lebih ringan karena tidak terjadi interaksi antara server dan client. Berbeda dengan web tanpa service worker yang dibuat menggunakan PHP dan javascript yang tidak memiliki fitur fetch event sehingga beban kerja server akan lebih besar.

1. **Pengujian *Black box Sistem***
2. Pengujian *input* data

Pada alur proses memasukkan data telah dilakukan pengujian *black box* sesuai dengan rancangan diagram *activity* pada gambar 3.6 yang ditunjukkan pada tabel 4.7.

Tabel 4. 9. Tabel hasil pengujian black box input data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Aksi | Masukan | Keluaran | Hasil Pengujian *black box* |
| 1 | *User* memilih halaman *input* data mesin yang akan melakukan proses *request* menuju lokal sistem untuk pertama kali dan akan mengirim umpan balik menuju *client*. | Masukan berupa klik kursor | Memunculkan halaman *input* data | Pengujian berhasil. Sistem menampilka halaman *input* data |
| 2 | *User* memasukkan data *inputan* berupa data mesin yang kemudian akan diproses oleh sistem dan dikirim menuju lokal *database* *indexedDB.* | Masukan berupa *text* atau karakter pada *form text* dan klik pada tombol *submit* data | Memunculkan data yang telah dimasukkan pada *database* lokal ataupun *database server* pada halaman mesin. | Pengujian berhasil. Sistem menampilkan halaman mesin dengan data yang telah di masukkan. |

1. Pengujian *update* data

Pengujian *black box* pada proses *update data* dilakukanuntuk melihat kesesuaian sistem yang dibangun dengan hasil yang diharapkan. Hasil pengujian ditunjukkan pada tabel 4.8.

Tabel 4. 10. Hasil pengujian black box update data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Percobaan | Aksi | Masukan | Keluaran | Hasil Pengujian *black box* |
| 1 | *User* memilih halaman *update* data mesin. *Request* kemudian akan diterima oleh sistem lokal dan akan menampilkan umpan balik. | Masukan berupa klik kursor | Memunculkan halaman *update* data | Pengujian berhasil. Sistem menampilkan halaman *update* data |
| 2 | *User* melakukan pembaharuan data data mesin yang sudah ada atau pemeliharaan mesin yang kemudian akan diproses oleh sistem dan dikirim menuju lokal *database* *indexedDB* untuk pembaharuan data*.* | Masukan berupa *text* atau karakter pada *form text* dan klik pada tombol *submit* data | Memunculkan data yang telah diperbaharui pada *database* lokal ataupun *database server* pada halaman pembaharuan data mesin. | Pengujian berhasil. Sistem menampilkan halaman pembaruan data mesin dengan data yang telah di perbaharui. |

1. Pengujian *delete* data

Pengujian *black box* pada proses *delete data* dilakukanuntuk melihat kesesuaian sistem yang dibangun dengan hasil yang diharapkan. Hasil pengujian ditunjukkan pada tabel 3.2.

Tabel 4. 11. Hasil pengujian black box delete data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Aksi | Masukan | Keluaran | Hasil Pengujian *black box* |
| 1 | *User* memilih data yang ingin di hapus. | Masukan berupa klik kursor | Memunculkan halaman mesin dengan data yang telah terhapus | Pengujian berhasil. Sistem menampilkan halaman mesin dan data yang dihapus tidak muncul |

# **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 react JS* akan memberikan waktu akses rata – rata yang lebih rendah dibandingkan dengan *web* konvensional. Dengan jumlah *thread* total yang diberikan 1000 pada *web* konvensional menunjukkan waktu akses yang lebih tinggi dibandingkan *web* *react JS*. Sehingga dapat dikatakan bahwa *web* dengan menggunakan *react JS* 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 dan *web* tanpa *service worker* dengan menggunakan *javascript*. Hal ini menunjukkan bahwa *web service worker* dapat melakukan transfer file yang lebih besar dalam satuan waktu dibandingkan dengan *web* konvensional dan *web* tanpa *service worker* dengan menggunakan *java script*.
    - 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 dan *web* tanpa *service worker* dengan menggunakan *javascript*. Ini disebabkan oleh *boost performance* dan manajemen *cache* yang ada pada *service worker*.
* Waktu yang digunakan *web* *service worker* untuk melakukan pemasukan data lebih besar dibandingkan dengan *web* lainnya.Hal ini dikarenakan ketika *web service worker* melakukan proses pemasukan data akan menjalankan fungsi singkronisasi sekaligus yaitu melakukan pemasukan data pada basis data local dan kemudian melakukan pemasukan data pada database server.
  + - Pengaruh performansi CPU pada saat *web service worker* diakses menunjukkan hasil persentase yang stabil dan lebih rendah dibandingkan dengan *web* konvensional dan *web* tanpa *service worker* dengan menggunakan *service worker*. 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 dan *web* tanpa *service worker* dengan *java script*, sehingga data yang diperoleh dari hasil pengujian sesuai dengan hasil yang diharapkan.
    - Hasil yang didapatkan dari proses pengujian menggunakan metode *black box* memberikan keluaran yang diharapkan pada setiap proses *input,update, dan delete* data.

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. n.d. “Www.reactenlightenment.com.”

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

Butkiewicz, Michael. 2011. “Understanding *Web*site 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.

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

Hannonen, Daria. 2017. “Development of *Web*site 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.

Permana, Endang Cahya. 2016. “Penulisan Fungsi Pada Javascript.” 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.”

**LAMPIRAN**

1. ***Source code service worker***

*// In production, we register a service worker to serve assets from local cache.*

*// This lets the app load faster on subsequent visits in production, and gives*

*// it offline capabilities. However, it also means that developers (and users)*

*// will only see deployed updates on the "N+1" visit to a page, since previously*

*// cached resources are updated in the background.*

*// To learn more about the benefits of this model, read https://goo.gl/KwvDNy.*

*// This link also includes instructions on opting out of this behavior.*

*const isLocalhost = Boolean(*

*window.location.hostname === 'localhost' ||*

*// [::1] is the IPv6 localhost address.*

*window.location.hostname === '[::1]' ||*

*// 127.0.0.1/8 is considered localhost for IPv4.*

*window.location.hostname.match(*

*/^127(?:\.(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}$/*

*)*

*);*

*export default function register() {*

*if (process.env.NODE\_ENV === 'production' && 'serviceWorker' in navigator) {*

*// The URL constructor is available in all browsers that support SW.*

*const publicUrl = new URL(process.env.PUBLIC\_URL, window.location);*

*if (publicUrl.origin !== window.location.origin) {*

*// Our service worker won't work if PUBLIC\_URL is on a different origin*

*// from what our page is served on. This might happen if a CDN is used to*

*// serve assets; see https://github.com/facebookincubator/create-react-app/issues/2374*

*return;*

*}*

*window.addEventListener('load', () => {*

*const swUrl = `${process.env.PUBLIC\_URL}/service-worker.js`;*

*if (isLocalhost) {*

*// This is running on localhost. Lets check if a service worker still exists or not.*

*checkValidServiceWorker(swUrl);*

*// Add some additional logging to localhost, pointing developers to the*

*// service worker/PWA documentation.*

*navigator.serviceWorker.ready.then(() => {*

*console.log(*

*'This web app is being served cache-first by a service ' +*

*'worker. To learn more, visit https://goo.gl/SC7cgQ'*

*);*

*});*

*} else {*

*// Is not local host. Just register service worker*

*registerValidSW(swUrl);*

*}*

*});*

*}*

*}*

*function registerValidSW(swUrl) {*

*navigator.serviceWorker*

*.register(swUrl)*

*.then(registration => {*

*registration.onupdatefound = () => {*

*const installingWorker = registration.installing;*

*installingWorker.onstatechange = () => {*

*if (installingWorker.state === 'installed') {*

*if (navigator.serviceWorker.controller) {*

*// At this point, the old content will have been purged and*

*// the fresh content will have been added to the cache.*

*// It's the perfect time to display a "New content is*

*// available; please refresh." message in your web app.*

*console.log('New content is available; please refresh.');*

*} else {*

*// At this point, everything has been precached.*

*// It's the perfect time to display a*

*// "Content is cached for offline use." message.*

*console.log('Content is cached for offline use.');*

*}*

*}*

*};*

*};*

*})*

*.catch(error => {*

*console.error('Error during service worker registration:', error);*

*});*

*}*

*function checkValidServiceWorker(swUrl) {*

*// Check if the service worker can be found. If it can't reload the page.*

*fetch(swUrl)*

*.then(response => {*

*// Ensure service worker exists, and that we really are getting a JS file.*

*if (*

*response.status === 404 ||*

*response.headers.get('content-type').indexOf('javascript') === -1*

*) {*

*// No service worker found. Probably a different app. Reload the page.*

*navigator.serviceWorker.ready.then(registration => {*

*registration.unregister().then(() => {*

*window.location.reload();*

*});*

*});*

*} else {*

*// Service worker found. Proceed as normal.*

*registerValidSW(swUrl);*

*}*

*})*

*.catch(() => {*

*console.log(*

*'No internet connection found. App is running in offline mode.'*

*);*

*});*

*}*

*export function unregister() {*

*if ('serviceWorker' in navigator) {*

*navigator.serviceWorker.ready.then(registration => {*

*registration.unregister();*

*});*

*}*

*}*

1. ***Source code* Mesin.js**

import React from 'react';

import PropTypes from 'prop-types';

import classNames from 'classnames';

import { withStyles } from '@material-ui/core/styles';

import CssBaseline from '@material-ui/core/CssBaseline';

import Drawer from '@material-ui/core/Drawer';

import AppBar from '@material-ui/core/AppBar';

import Toolbar from '@material-ui/core/Toolbar';

import List from '@material-ui/core/List';

import Typography from '@material-ui/core/Typography';

import Divider from '@material-ui/core/Divider';

import IconButton from '@material-ui/core/IconButton';

import Badge from '@material-ui/core/Badge';

import MenuIcon from '@material-ui/icons/Menu';

import ChevronLeftIcon from '@material-ui/icons/ChevronLeft';

import NotificationsIcon from '@material-ui/icons/Notifications';

import { mainListItems, secondaryListItems } from '@material-ui/core/ListItem';

import ListItem from '@material-ui/core/ListItem';

import ListItemText from '@material-ui/core/ListItemText';

import Avatar from '@material-ui/core/Avatar';

import ImageIcon from '@material-ui/icons/Image';

import TextField from '@material-ui/core/TextField';

import WorkIcon from '@material-ui/icons/Work';

import {Link} from 'react-*router*-dom';

import Button from '@material-ui/core/Button';

import Grid from '@material-ui/core/Grid';

import ListItemIcon from '@material-ui/core/ListItemIcon';

import Paper from '@material-ui/core/Paper';

//import ListItemText from '@material-ui/core/ListItemText';

import ListSubheader from '@material-ui/core/ListSubheader';

import AssignmentIcon from '@material-ui/icons/Assignment';

import \* as moment from 'moment';

import FormControlLabel from '@material-ui/core/FormControlLabel';

import Checkbox from '@material-ui/core/Checkbox';

import \* as RxDB from 'rxdb';

import {QueryChangeDetector} from 'rxdb';

import { skema } from './Schema';

import { Browser*Router*, Route} from 'react-*router*-dom';

import { ToastContainer, toast } from 'react-toastify';

import Icon from '@material-ui/core/Icon';

import Visibility from '@material-ui/icons/Visibility';

import VisibilityOff from '@material-ui/icons/VisibilityOff';

import *Input*Adornment from '@material-ui/core/*Input*Adornment';

import SaveIcon from '@material-ui/icons/Save';

import 'react-toastify/dist/ReactToastify.css';

import Table from '@material-ui/core/Table';

import TableBody from '@material-ui/core/TableBody';

import TableCell from '@material-ui/core/TableCell';

import TableHead from '@material-ui/core/TableHead';

import TableRow from '@material-ui/core/TableRow';

import View from './View';

import Edit from './Edit';

import Home from './Home';

const drawerWidth = 240;

QueryChangeDetector.enable(); // kita membutuhkan query change detector untuk meningkatkan performa query

QueryChangeDetector.enableDebugging(); //karena RxDb yang akan meminta request data ketika terjadi perubhan maka dia akan memberatkan query makanya pake kode ini

RxDB.plugin(require('pouchdb-adapter-idb')); // inisialisasi database local using rxdb

RxDB.plugin(require('pouchdb-adapter-http'));// inisialisasi remote db

const syncURL = 'https://98855c8d.ngrok.io/'; // remote db

const dbName = 'mesinfixfix'; // name local db

const styles = theme => ({

root: {

display: 'flex',

flexWrap: 'wrap',

...theme.mixins.gutters(),

paddingTop: theme.spacing.unit \* 2,

paddingBottom: theme.spacing.unit \* 2,

width: '100%',

marginTop: theme.spacing.unit \* 3,

overflow:'hidden !important',

},

table: {

minWidth: 700,

},

container: {

display: 'flex',

flexWrap: 'wrap',

},

margin: {

margin: theme.spacing.unit,

},

textField: {

flexBasis: 200,

},

dense: {

marginTop: 16,

},

menu: {

width: 200,

},

toolbar: {

paddingRight: 24, // keep right padding when drawer closed

},

toolbarIcon: {

display: 'flex',

alignItems: 'center',

justifyContent: 'flex-end',

padding: '0 8px',

...theme.mixins.toolbar,

},

appBar: {

zIndex: theme.zIndex.drawer + 1,

transition: theme.transitions.create(['width', 'margin'], {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.leavingScreen,

}),

},

appBarShift: {

marginLeft: drawerWidth,

width: `calc(100% - ${drawerWidth}px)`,

transition: theme.transitions.create(['width', 'margin'], {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.enteringScreen,

}),

},

menuButton: {

marginLeft: 12,

marginRight: 36,

},

menuButtonHidden: {

display: 'none',

},

title: {

flexGrow: 1,

},

drawerPaper: {

position: 'relative',

whiteSpace: 'nowrap',

width: drawerWidth,

transition: theme.transitions.create('width', {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.enteringScreen,

}),

},

drawerPaperClose: {

overflowX: 'hidden',

transition: theme.transitions.create('width', {

easing: theme.transitions.easing.sharp,

duration: theme.transitions.duration.leavingScreen,

}),

width: theme.spacing.unit \* 7,

[theme.breakpoints.up('sm')]: {

width: theme.spacing.unit \* 9,

},

},

appBarSpacer: theme.mixins.toolbar,

content: {

flexGrow: 1,

padding: theme.spacing.unit \* 3,

height: '100vh',

overflow: 'auto',

},

chartContainer: {

marginLeft: -22,

},

tableContainer: {

height: 320,

},

});

let id = 0;

function createData(name, calories, fat, carbs, protein) {

id += 1;

return { id, name, calories, fat, carbs, protein };

}

const rows = [

createData('Frozen yoghurt', 159, 6.0, 24, 4.0),

createData('Ice cream sandwich', 237, 9.0, 37, 4.3),

createData('Eclair', 262, 16.0, 24, 6.0),

createData('Cupcake', 305, 3.7, 67, 4.3),

createData('Gingerbread', 356, 16.0, 49, 3.9),

];

class Mesin extends React.Component {

state = {

open: true,

};

constructor(props) {

super(props);

this.state = {

nama\_mesin: '',

status: '',

downTime:'',

costDate:'',

data : '',

contoh:[],

mesin: [],

};

this.subs = [];

this.addMessage = this.addMessage.bind(this);

// this.DeleteMessage = this.DeleteMessage.bind(this);

this.handleMessageChange = this.handleMessageChange.bind(this);

//console.log("ini "+ window.location.href)

}

// 1) buat method pembuatan db dengan cara awit

handleChange = prop => event => {

this.setState({ [prop]: event.target.value });

};

async createDatabase() {

// password must have at least 8 characters dan ini fungsinya untuk enkripsi data yang masuk ke dalam koleksi data

const db = await RxDB.create(

{name: dbName, adapter: 'idb', password: '12345678', ignoreDuplicate: true}

);

console.dir(db);

// show who's the leader in page's title

db.waitForLeadership().then(() => {

document.title = 'Home ' + document.title;

});

// leader ellection algorithm, dia membuat satu tab hanya me manage remote db

// create collection

const mesinCollection = await db.collection({

name: 'mesin',

schema: skema

});

// set up replication

const replicationState = mesinCollection.sync({ remote: syncURL + dbName + '/' });

this.subs.push(

replicationState.change$.subscribe(change => {

toast('Replication change');

console.dir(change)

})

);

this.subs.push(

replicationState.docs$.subscribe(docData => console.dir(docData))

);

this.subs.push(

replicationState.active$.subscribe(active => toast(`Replication active: ${active}`))

);

this.subs.push(

replicationState.complete$.subscribe(completed => toast(`Replication completed: ${completed}`))

);

this.subs.push(

replicationState.error$.subscribe(error => {

toast('Replication Error');

console.dir(error)

})

);

return db;

}

async componentDidMount() {

this.db = await this.createDatabase();

// Subscribe to query to get all messages

const sub =

this.db.mesin.find().where('tipe').eq('Mesin Tools').sort({id: 1}).$.subscribe(mesin => {

if (!mesin)

return;

toast('Re*load*ing Data');

this.setState({mesin: mesin});

});

this.subs.push(sub);

}

componentWillUnmount() {

// Unsubscribe from all subscriptions

this.subs.forEach(sub => sub.unsubscribe());

}

render() {

const {data,status,contoh,mesin,downTime,costDate,date} = this.state;

return (

<div>

<div>

<Link to="/mesin">Mesin</Link>

<Link to="/maintenance">Maintenance</Link>

</div>

<div>

<p>{this.addData()}</p>

</div>

</div>

);

}

addData = () => {

const { classes } = this.props;

const {data,status,contoh,mesin,downTime,costDate,date} = this.state;

return (

<main className={classes.content}>

<div className={classes.appBarSpacer} />

<Paper className={classes.root} elevation={1}>

<Typography variant="headline" component="h3">

Tambah Data Mesin

</Typography>

</Paper>

<br />

<br />

<form className={classes.container} noValidate autoComplete="off">

<TextField

id="outlined-adornment-weight"

className={classNames(classes.margin, classes.textField)}

variant="outlined"

name="nama\_mesin"

label="Nama Mesin"

value={this.state.nama\_mesin}

onChange={(e)=>this.handleMessageChange(e)}

helperText="Nama Mesin"

*Input*Props={{

endAdornment: <*Input*Adornment position="end"></*Input*Adornment>,

}}

/>

<TextField

id="outlined-adornment-weight"

className={classNames(classes.margin, classes.textField)}

variant="outlined"

name="status"

label="Status Mesin"

value={status}

onChange={(e)=>this.handleMessageChange(e)}

helperText="Status"

*Input*Props={{

endAdornment: <*Input*Adornment position="end"></*Input*Adornment>,

}}

/>

<TextField

id="outlined-adornment-weight"

className={classNames(classes.margin, classes.textField)}

variant="outlined"

name="downTime"

type="date"

label=" "

value={downTime}

onChange={(e)=>this.handleMessageChange(e)}

helperText="Down Time"

*Input*Props={{

endAdornment: <*Input*Adornment position="end"></*Input*Adornment>,

}}

/>

<TextField

id="outlined-adornment-weight"

className={classNames(classes.margin, classes.textField)}

variant="outlined"

name="costDate"

label="Cost"

value={costDate}

onChange={(e)=>this.handleMessageChange(e)}

helperText="Down Time"

*Input*Props={{

endAdornment: <*Input*Adornment position="end"></*Input*Adornment>,

}}

/>

</form>

<br />

<br/>

<Button onClick={()=>this.addMessage()} variant="contained" color="primary" >

<SaveIcon className={classNames(classes.leftIcon, classes.iconSmall)} />

Save

</Button>

<br />

<br/>

<Paper className={classes.root} elevation={1}>

<Typography variant="headline" component="h3">

Data Mesin

</Typography>

</Paper>

<br />

<br />

<TableHead>

<TableRow>

<TableCell>Machine Name</TableCell>

<TableCell align="right">Status Mesin</TableCell>

<TableCell align="right">Down Time</TableCell>

<TableCell align="right">Cost (g)</TableCell>

<TableCell align="right">Date</TableCell>

</TableRow>

</TableHead>

{this.renderMessages()}

</main>

)

}

renderMessages = () =>{

return this.state.mesin.map(({id, nama\_mesin,status,downTime,costDate,date}) => {

const waktu*Input* = moment(id, 'x').fromNow();

return (

<Paper>

<TableBody>

<TableRow>

<TableCell component="th" scope="row">

{nama\_mesin}

</TableCell>

<TableCell align="right">

{status}

</TableCell>

<TableCell numeric>

{downTime}

</TableCell>

<TableCell numeric>

{costDate}

</TableCell>

<TableCell numeric>

{date}

</TableCell>

<TableCell numeric>

<Link to={`/mesin/view/${id}`}><Button variant="contained" color="primary">

View

</Button></Link>

</TableCell>

<TableCell numeric>

<Button variant="contained" color="secondary" onClick={()=>{this.DeleteMessage(id)}}>Delete</Button>

</TableCell>

</TableRow>

</TableBody>

</Paper>

);

});

}

handleMessageChange(e) {

this.setState({[e.target.name]: e.target.value});

}

async addMessage() {

var today = new Date();

var dd = today.getDate();

var mm = today.getMonth() + 1; //January is 0!

var yyyy = today.getFullYear();

if (dd < 10) {

dd = '0' + dd;

}

if (mm < 10) {

mm = '0' + mm;

}

const hari = mm + '/' + dd + '/' + yyyy;

const id = Date.now().toString();

const {nama\_mesin,status,downTime,costDate} = this.state

const mesin = nama\_mesin;

const tipe = 'Mesin Tools';

const date = hari.toString();

const dataMesin = {id,nama\_mesin,status,downTime,costDate,date,tipe};

await this.db.mesin.insert(dataMesin);

this.setState({nama\_mesin: ''});

this.setState({status:''});

this.setState({downTime: ''});

this.setState({costDate:''});

}

DeleteMessage = async (id) =>{

await this.db.mesin.findOne().where('\_id').eq(id)

.remove();

// this.db.messages.find().exec() // <- find all documents

//.then(documents => console.log(id));

// console.log(this.state.messages.map);

}

editData = async (id) => {

//this.setState({newMessage: ''});

const query = this.db.mesin.find().where('\_id').gt(id);

// await query.update({

// $inc: {

// me: 1 // increases age of every found document by 1

// }

console.log(query);

//});

}

//---------------------------------------------------------------------------------------------

getData = async (id) => {

//await this.db.messages.findOne().where('\_id').eq(id).exec().then(doc => console.log(doc));;

await this.db.mesin.findOne(id).exec().then(doc => {

console.log(doc.\_data)

this.setState({contoh : doc.\_data})

setTimeout(()=>{

console.log(this.state.contoh)

},2000)

});

}

//---------------------------------------------------------------------------------------------

sgetData = async (id) =>{

// await this.db.messages.findOne().where('\_id').eq(id).exec().then(doc => console.log(doc));;

await this.db.mesin.findOne(id).exec().then(doc => console.log(doc.\_data)) ;

}

updateData = async (id) => {

this.getData(id);

console.log(id)

}

dataContoh = () => {

const {data} = this.state

data === [] ? <p>Kosong</p> : <p>Ada</p>

}

}

Mesin.propTypes = {

classes: PropTypes.object.isRequired,

};

export default withStyles(styles)(Mesin);