Pemanfaatan Progressive Web Apps Pada Web Akuntansi

Grace Levina Dewi[1*], Suhatati Tjandra[2], Ricardo[3]

[1, 2, 3]Program Studi Informatika, Institut Sains dan Teknologi Terpadu Surabaya, Jawa Timur
Email: [1*]gracelevina@stts.edu

(Naskah masuk: 12 Feb 2020, direvisi: 13 Apr 2020, diterima: 5 Mei 2020)

Abstrak

Software untuk mengelola data transaksi menjadi laporan keuangan sangat banyak dipakai di berbagai sektor bisnis. *Software* yang cepat, akurat, reliabilitas tinggi serta mudah diimplementasikan menjadi faktor pemilihan *software*. Oleh karena itu, dikembangkan *software* pengelola transaksi dengan menggunakan PWA (*Progressive Web Apps*). Alasan memanfaatkan PWA karena dapat memenuhi kebutuhan *software* yang dibutuhkan oleh pengguna. Secara garis besar, kelebihan PWA ini terletak pada bagian notifikasi, dapat diakses secara *offline*, memiliki performa yang bagus, sistem keamanan yang bagus (karena harus menggunakan HTTPS), dan dapat diterapkan pada web. Penerapan PWA ini juga sama seperti memakai aplikasi di *smartphone*. Uji coba dalam pengembangan ini dilakukan dengan dua metode, yaitu *Performance Testing* dan *Functionality Testing*. Hasil uji coba menyatakan bahwa web akuntansi dengan PWA mempunyai kecepatan yang hampir sama dengan *native* tetapi *User Experience* PWA lebih baik dibandingkan aplikasi *native*. Hasil dari *Functionality Testing* menunjukan *Web App* dapat berjalan dengan baik sesuai fungsi setiap fitur.

Kata Kunci: *Progressive Web Apps*, Porto, SAP, *website*, Quasar.

The Use of Progressive Web Apps on Accounting Web

Abstract

Software for managing transaction data into financial statements is widely used in various business sectors. Software which is fast, accurate, high reliability and easy to implement is a software selection factor. Therefore, transaction management software was developed using PWA *(Progressive Web Apps). The reason to use* PWA *is because able to meet the software requirements needed by the users. Broadly speaking, the advantages of this* PWA *lies on the notification section, can be accessed offline, has good performance, a good security system (because it must use* HTTPS*), and can be applied to the web. The application of* PWA *is also the same as using an application on a smartphone. Testings in this development are carried out by two methods, namely Performance Testing and Functionality Testing. The testing results state that the Web App with* PWA *has almost the same speed as native but the* PWA *User Experience is better than native application. The results of Functionality Testing show the Web App can run properly according to the function of each feature.*

Keywords: *Progressive Web Apps, Porto, SAP, Website, Quasar.*

I. PENDAHULUAN

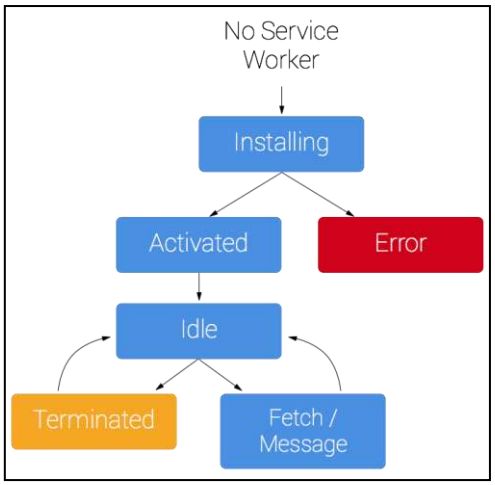
Mengetahui kesehatan perusahaan merupakan hal yang vital bagi setiap perusahaan. Laporan kesehatan perusahaan dapat digunakan sebagai indikasi untuk mengambil keputusan yang penting. Laporan ini dibuat dengan mengorganisir data transaksi dan aktivitas keuangan yang terjadi pada perusahaan tersebut. Sistem informasi akuntansi merupakan sistem yang digunakan untuk mengorganisir data keuangan dan transaksi untuk membuat laporan keuangan yang digunakan untuk indikasi kesehatan keuangan perusahaan tersebut [1].

Sistem informasi akuntansi tidak cukup untuk menyelesaikan masalah reliabilitas data. Data yang dihasilkan harus dikirim ke *user* dengan cepat serta akurat. Oleh karena itu *Progressive Web Apps* diimplementasikan pada sistem informasi akuntansi untuk meningkatkan kinerja serta reliabilitas data. *Progressive Web Apps* adalah teknologi *Web App* yang dikembangkan oleh Google. PWA adalah teknologi *caching* data dan *file* yang akan diminta oleh sistem.

Progressive Web Apps (PWA) adalah aplikasi web yang memuat seperti halaman web tetapi dapat menawarkan fungsionalitas pengguna seperti bekerja *offline*, notifikasi, dan akses perangkat keras. PWAs mempunyai karakteristik dapat diandalkan, cepat, dan menarik [2]. Berikut adalah karakteristik dari PWA [3]:

1) *Progressive* – kompatibel untuk semua *browser* dan *device*.
2) *Responsive* – dapat beradaptasi pada setiap jenis layar.
3) *Conectivity Independent* – dapat berjalan pada saat *offline* atau jaringan tidak stabil.
4) *App Like – Web App* serperti *Native App*.
5) *Fresh* – selalu menyediakan data yang terbaru.
6) *Safe* – mengunakan koneksi yang terenkripsi (HTTPS).
7) *Discoverable* – mudah dicari di *search engine*.
8) *Re-engageable* – mudah untuk membuka *Web App* lewat notifikasi.
9) *Installable* – dapat diinstal ke *mobile device*.
10) *Linkable* – mudah membagikan *web app* melalui *link*.

Service worker adalah sebuah *script* yang *browser* jalankan di *background* terpisah dari *web page* sehingga tidak memerlukan interaksi *web page* atau *user*. Sekarang *service worker* sudah mendukung penggunaan fitur seperti *push notification* dan *background sync*. *Service worker* merupakan *JavaScript worker*, sehingga tidak bisa mengakses DOM (*Document Object Model*) secara langsung.



Gambar 1. Daur Hidup *Service Worker*

Gambar 1 adalah daur hidup dari *service worker*. *Service worker* punya daur hidup yang sepenuhnya terpisah dari *web page*. Proses instalasi *service worker* ke situs perlu untuk melakukan pendaftaran terlebih dahulu di *JavaScript* yang ada di halaman web. Setelah proses pendaftaran berhasil maka *browser* akan melakukan proses instalasi *service worker* di *background*.

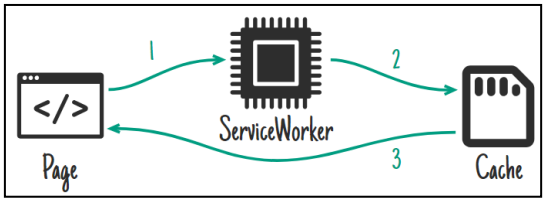
Saat melalukan instalasi biasanya akan ada beberapa aset statik yang perlu diletakkan pada *cache* (data yang sifatnya sementara disimpan pada internal *smartphone*). Jika semua *file* sudah selesai disimpan pada *cache* maka *service worker* berhasil diinstalasi. Tetapi jika ada *file* yang gagal disimpan pada *cache* maka *service worker* tidak akan dapat ter-*install* di *cache* dan tidak akan aktif. *File* yang gagal dimasukan ke *cache* akan diproses kembali di *cache* saat dikunjungi lagi.

Ketika langkah instalasi berhasil, maka langkah pengaktifan akan dijalankan dan pada langkah ini semua *cache* lama bisa ditangani kembali. Setelah langkah pengaktifan selesai maka *service worker* sudah bisa mengendalikan semua *page* yang berada di dalam cakupan. Tapi hal ini tidak akan bisa dilakukan ketika *service worker* baru pertama kali terdaftar sehingga membutuhkan *load page* ulang. Saat *service worker* berjalan akan ada dua kondisi yang akan dilalui yaitu kondisi berhenti untuk menyimpan memori atau kondisi untuk menangani *fetch* dan *message event* yang terjadi ketika *network request* atau *message* dibuat dari *page*.

Workbox adalah library *service worker* yang dikembangkan oleh Google sebagai *boilerplate best practice*. Library ini berfungsi untuk memudahkan dan mempercepat pembuatan PWA untuk aplikasi web. Beberapa fitur yang ada dalam *workbox* antara lain: *precaching*, *runtime caching*, strategi, *request routing*, *background sync*, *debugging service worker*, kode yang fleksibel, dan sederhana.

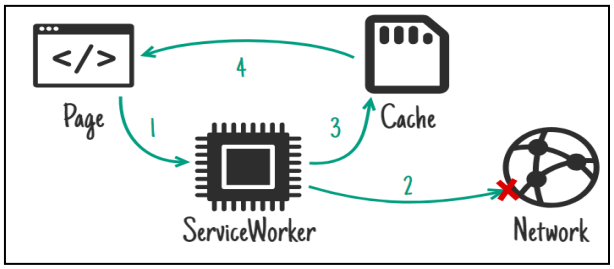
Workbox merupakan penerus dari *sw-precache dan sw-toolbox*. *Sw-precache* hanya library untuk *precaching*, sedangkan *sw-toolbox* adalah *tool* untuk membuat *runtime caching*. Setelah pembuatan dua *tool* ini selesai, Google mengembangkan dan mengabungkan keduanya menjadi satu. *Tool* gabungan tersebut bernama *workbox*. Selain fitur yang ada di *sw-toolbox* dan *sw-precache*, *workbox* juga menyediakan *custom injection service worker* dan *background sync*. Fitur terpenting dalam *workbox* adalah *precaching route* mengunakan suatu strategi. Ada lima strategi yang disediakan oleh *workbox*. Strategi tersebut antara lain:

- *Cache First*
  Strategi *cache first* sangat baik digunakan untuk *resource web* statik yang jarang untuk diganti seperti *file* CSS dan JS. Strategi ini akan menyimpan *resource* tersebut ke *cache* setelah pertama kali di-*request*. Ketika *request* itu dilakukan lagi, *service worker* akan mengembalikan data dari *cache* terlebih dahulu sebelum melakukan *request* ke *server* yang sebenarnya. Gambar 2 menunjukkan strategi *cache first*.



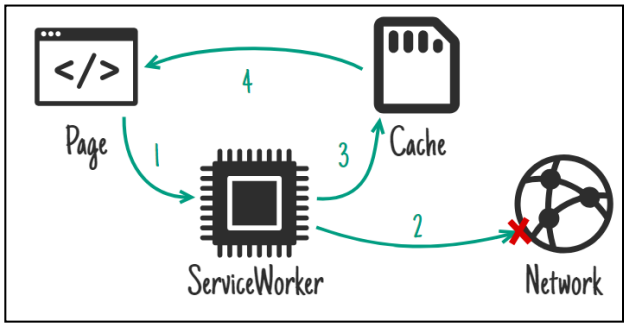
Gambar 2. Strategi *Cache First*

- *Cache Only*
  Strategi *cache only* baik digunakan untuk *plugin* yang ada di *workbox*. Strategi ini lebih baik digunakan ke *resource* yang tidak mungkin untuk diganti ke depannya. Gambar 3 menunjukkan strategi *cache only*.



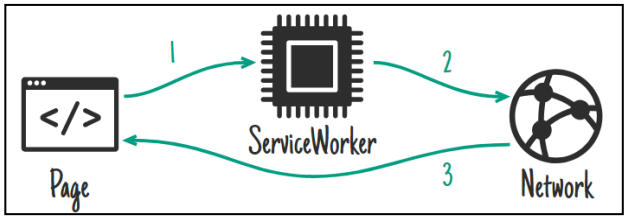
Gambar 3. Strategi *Cache Only*

- *Network First*
  Strategi *Network First* baik digunakan untuk aplikasi *realtime* yang membutuhkan data dengan realibilitas tinggi. *Service worker* akan memberi perintah *request* ke *server* sesungguhnya. Jika *request* berhasil, simpan *request signature* dan *data response* ke *cache* serta mengembalikan *request* ke *user*. Jika *request* gagal, cek apakah *request signature* ada di *cache*. Jika ada di *cache* maka kembalikan *response* dari *cache* ke *user*. Gambar 4 menunjukkan strategi *network first*.



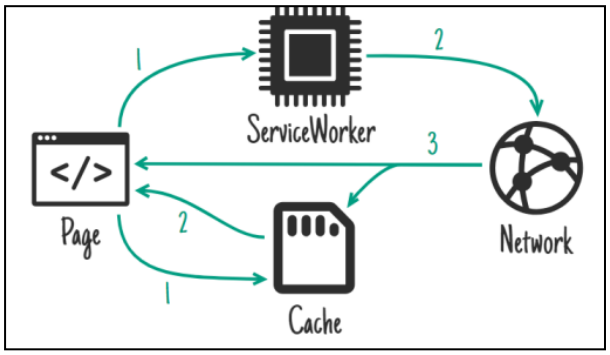
Gambar 4. Strategi *Network First*

- *Network Only*
  Strategi *Network Only* tidak ada bedanya dengan *request* normal ke *server* dan dikembalikan ke *user*. Strategi ini tidak mendapatkan *cache response*. Gambar 5 menunjukkan strategi dari *network only*.



Gambar 5. Strategi *Network Only*

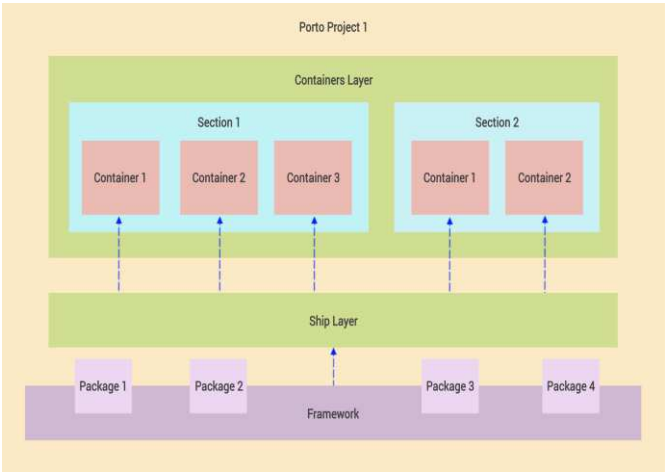
- *Stale While Revalidate*
  Strategi *Stale While Revalidate* adalah strategi yang baik digunakan untuk *App* yang butuh menampilkan data ke *user* dengan latensi yang sangat rendah. Strategi ini mempunyai banyak kelebihan dibandingkan strategi yang lain. Keuntungan memakai strategi ini sangat jelas, Strategi ini sangat cepat dan *responsive*. Kerugian untuk mengimplementasikan strategi ini adalah lebih kompleks dibandingkan dengan strategi yang lain. Gambar 6 menunjukkan *strategi stale while revalidate*.



Gambar 6. Strategi *Stale While Revalidate*

Selanjutnya pembahasan mengenai *degin pattern. Design pattern* sangat dibutuhkan dalam pengembangan *software* ini. Tanpa adanya prosedur dan arsitektur yang baik maka lebih sulit untuk mendokumentasikan semua fitur yang ada dalam penelitian. Sistem pada penelitian ini memakai *Porto SAP design pattern* sebagai arsitektur sistem yang diterapkan. *Porto SAP* adalah pola arsitektur perangkat lunak modern, yang dirancang untuk membantu pengembang mengatur kode mereka dengan cara yang mudah dikelola [4]. *Design Pattern* sangat membantu untuk proyek-proyek jangka panjang dan besar, karena mereka cenderung memiliki kompleksitas yang lebih tinggi dengan waktu.

Gambar 7 adalah arsitektur *design pattern Porto SAP*. Arsitektur dibagi menjadi tiga bagian yaitu *framework layer*, *ship layer*, dan *container layer*. *Framework layer* adalah *layer* yang menyediakan *package default* yang dibutuhkan untuk menjalankan sistem. *Ship layer* terdiri atas kelas *parent* dan fungsi *helper* yang di-*share* pada semua *container*. *Container layer* adalah *layer* dimana logika bisnis akan diimplementasikan. Gambar 7 menunjukkan *design pattern porto*.



Gambar 7. *Design Pattern Porto*

Setelah membahas tentang *desaign pattern,* dibahas tentang *web service* yang digunakan. *Web service* adalah suatu sistem perangkat lunak yang dirancang untuk

mendukung interoperabilitas dan interaksi antar sistem pada suatu jaringan. *Web service* digunakan sebagai suatu fasilitas yang disediakan oleh suatu *website* untuk menyediakan layanan (dalam bentuk informasi) kepada sistem lain, sehingga sistem lain dapat berinteraksi dengan sistem tersebut melalui layanan-layanan (*service*) yang disediakan oleh suatu sistem yang menyediakan *web service* [5].

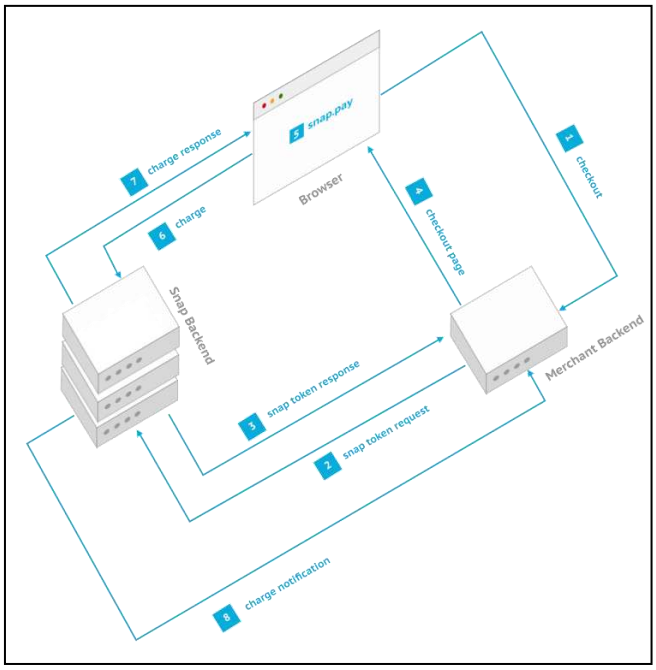
Web Service terbagi menjadi dua arsitektur, yaitu: REST (*Representational State Transfer*) dan SOAP (*Simple Object Access Protocol*). REST lebih mudah diimplementasikan serta lebih hemat *bandwith*. Hal ini dibuktikan dengan bagaimana SOAP dan REST merepresentasikan datanya. SOAP mengunakan format XML (*Extended Markup Language*), sedangkan REST mengunakan format JSON (*JavaScript Object Notation*). Selain itu untuk dokumentasi SOAP ada strukturnya sendiri sehingga untuk mengimplementasikannya lebih sulit. Penelitian akan mengunakan arsitekur REST.

Front end framework yang diterapkan pada penelitian ini menggunakan *Quasar*. *Quasar* merupakan *front-end framework* berbasis Vue.js yang berlisensi MIT yang bersifat *open source* [6]. *Framework* ini ditujukan untuk pengembangan aplikasi berbasis *Android* maupun iOS yang bersifat *hybrid* dengan menggunakan HTML, CSS, dan *JavaScript* [7]. Meskipun aplikasi berbasis *mobile* ini dibuat dengan menggunakan HTML, CSS, dan *JavaScript*, tampilan dari aplikasi tidak kalah baiknya dengan aplikasi *mobile* yang dibuat secara *native*. Jadi dapat dikatakan, *Quasar* merupakan sebuah *framework website* yang sudah menyediakan fitur tampilan *responsive* untuk aplikasi *mobile*, sehingga seolah-olah seperti sebuah aplikasi *mobile* yang dibuat secara *native*.

Quasar bisa digunakan untuk pembuatan aplikasi pada *Android* (*Cordova*), iOS (*Cordova*), *Web Apps*, dan PWAs. *Quasar* juga sudah didukung dengan bantuan *framework JavaScript* yang dikhususkan untuk pembuatan *front-end* yaitu *Vue.js* sehingga dapat memperindah tampilan dari desain aplikasi yang telah dibuat dengan *Quasar*.

Metode pembayaran menggunakan *mobile payment*. *Mobile payment* sebagai pendukung metode pembayaran pada penelitian ini menggunkan *Midtrans*. *Midtrans* adalah perusahaan yang menyediakan *third party payment gateway* [8]. API (*Application Programming Interface*) yang disediakan oleh *Midtrans* mempunyai banyak metode pembayaran seperti kartu kredit, *virtual account*, *internet banking*, bayar melalui Indomaret, dan lain-lain. API ini digunakan untuk menerima pembayaran *feature website* yang akan diimplementasikan pada penelitian.

Penelitian akan mengunakan SNAP API yang disediakan oleh *Midtrans* untuk menerima pembayaran. SNAP merupakan portal pembayaran yang memungkinkan *merchant* untuk menggunakan sistem pembayaran *Midtrans* dengan memunculkan halaman pembayaran *Midtrans* langsung di halaman pembayaran. Pemasangan API dari Midtrans mudah dan tidak dikenakan tagihan bulanan, cocok untuk bisnis skala kecil dan menengah. Gambar 8 adalah bagaimana SNAP API bekerja.



Gambar 8. Arsitektur SNAP API

II. TAHAPAN PENGEMBANGAN SISTEM

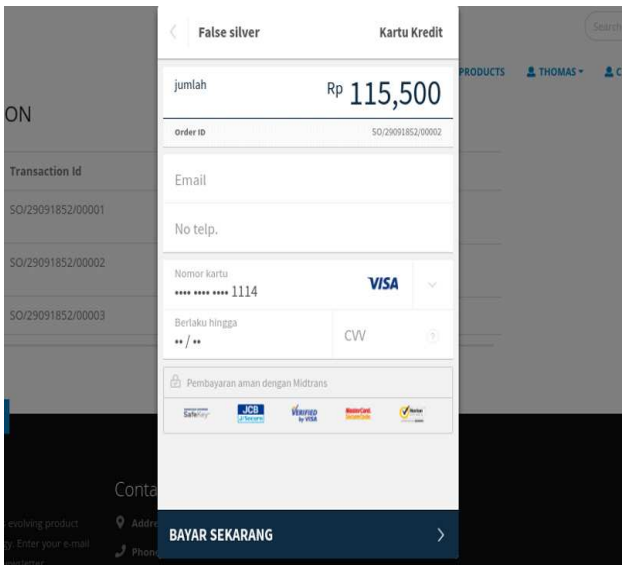
Penelitian ini membahas mengenai langkah-langkah untuk melakukan pengembangan sistem. Pengembangan ini sangat dibutuhkan untuk mempersiapkan pengerjaan penelitian ini. Langkah-langkah yang dilakukan, sebagai berikut:

- Menganalisis kebutuhan sistem.
  Proses analisis yang dilakukan dengan mempelajari *software* akuntansi yang sudah ada.
- Mengumpulkan *plugin* dan *library* yang dibutuhkan.
  Plugin dan *library* digunakan untuk memudahkan implementasi arsitektur yang sudah didesain.
- Membuat *back end server*.
  Back end server adalah API *Server* yang berbasiskan *framework Laravel* dengan *wrapper Apiato*.
- Mendesain *database*.
  Database dibuat dengan menentukan tabel apa saja yang dibutuhkan, serta relasi antar tabel.
- Membuat *database* serta tabelnya dan isi *default* tabel.
  Setelah desain selesai dibuat, desain tersebut diimplementasikan serta ditambah data *default* yang ada supaya sistem dapat berjalan
- Mendesain tampilan aplikasi *mobile* dan menghubungkan dengan API.
  Desain *interface* dengan mengunakan *front end framework* berbasis *Vue.js*. Pemanggilan API mengunakan *JavaScript Library* dengan nama *Axios*.
- Melakukan uji coba.
  Uji coba dilakukan untuk memastikan bahwa *Web App* yang dibuat dapat digunakan dan tidak ada kendala dalam mengelola sistem.

III. SPESIFIKASI KEBUTUHAN

Pada bagian ini dibahas mengenai spesifikasi kebutuhan yang diperlukan dalam melakukan pembuatan penelitian ini. Kebutuhan yang diperlukan dari segi akuntansi, fitur masing-masing *role*, dan pemanfaatan dari *Progressive Web Apps*. Spesifikasi kebutuhan dalam penelitian, yaitu:

- Proses akuntansi, yang merupakan proses utama. Proses ini terdiri dari enam proses utama yaitu validasi dan pembersihan, cek hak akses, modul jurnal, modul COA (*chart of account*), dan modul *template*.
- Fitur *user* dan *role* yaitu fitur yang menyediakan pengaturan hak akses.
- Fitur akses *mobile*, yang membuat *user* dapat mengakses sistem dari *smartphone* atau dari *tablet*. Fitur ini diimplementasikan dengan menggunakan CSS *Library Boostrap* untuk *website* dan *Quasar* yang sudah *responsive* secara desain.
- *Progressive Web Apps* adalah fitur yang diimplementasikan untuk meningkatkan kinerja serta reliabilitas data. Tampilan *responsive* merupakan hasil dari *framework Quasar* dengan *Vue.js*.
- Fitur *Push Notification* menunjukkan bahwa proses *service worker* bekerja pada proses yang berbeda dengan *browser*. Hal ini dibuktikan dengan notifikasi muncul walaupun *browser* atau *Web App* telah ditutup. Notifikasi dapat di buat saat suatu *event* terjadi seperti stok habis atau ada pembayaran yang belum dibayar.
- Fitur pembayaran dari *customer*. Fitur ini mengimplementasikan *Midtrans Payment Gateway* Fitur ini dapat terjadi setelah *customer* melakukan *checkout* pada keranjang belanja. Fitur ini membutuhkan *client key* dan *secret key* dari *dashboard Midtrans* untuk bisa menerima pembayaran dari *customer*. *Key* dari *Midtrans* dapat dimasukan pada halaman menu *settings*. Gambar 9 merupakan tampilan dari *Midtrans Payment Gateway*.

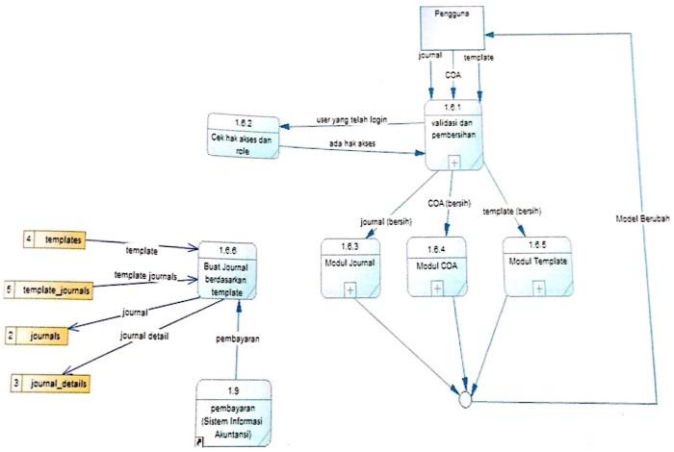


Gambar 9. *Midtrans Payment*

- *Background Sync* adalah fitur *Progressive Web Apps* yang akan menyimpan *request* gagal karena ganguan konektivitas. *Request* yang gagal akan disimpan pada *indexDB* dan saat *browser* merasa terkoneksi ke *internet request* tersebut akan dikirim kembali untuk dijalankan.

IV. DESAIN SISTEM

Modul Jurnal mengambil data dari *template* dan membuat jurnal entri untuk membukukan transaksi sesuai dengan *template* yang telah diatur sesuai dengan sistem debit dan kredit. Modul COA merupakan proses yang mengelola *chart of account*. Pada Gambar 10 menunjukkan DFD (*Data Flow Diagram*) pada proses akuntansi.



Gambar 10. DFD Proses Akuntansi

V. UJI COBA

Uji Coba dilakukan dengan dua metode, yaitu *Functionality Testing* dan *Performance Testing*. *Functionality Testing* adalah metode uji coba dengan mencoba fungsi yang dibuat satu per satu (*Black Box Testing*). *Performance Testing* adalah *benchmark* yang mengunakan *tool Lighthouse* dan *GTMetrix*.

A. *Functionality Testing*

Functionality Testing merupakan jenis pengujian *black-box* yang mendasarkan uji kasusnya pada spesifikasi komponen perangkat lunak yang diuji. Fungsi diuji dengan memberi masukan dan memeriksa *output*, dan struktur program internal jarang dipertimbangkan (tidak seperti pengujian *white-box*). Pengujian fungsional biasanya menggambarkan apa yang dilakukan sistem.

Functionality Testing dilakukan dengan menginputkan data ke *web app* dan melihat hasil dari inputan tersebut apakah *output* sesuai dengan harapan. *Functionality Testing* hanya dilakukan pada fitur yang penting pada penelitian ini, yaitu fitur PWA. *Functionality Testing* dilakukan dengan beberapa perangkat yaitu *Redmi Pro 3* (*Android*), dan *notebook* HP. Spesifikasi perangkat tersebut dapat dilihat pada Tabel 1.

Tabel 1. Spesifikasi Perangkat *Testing*

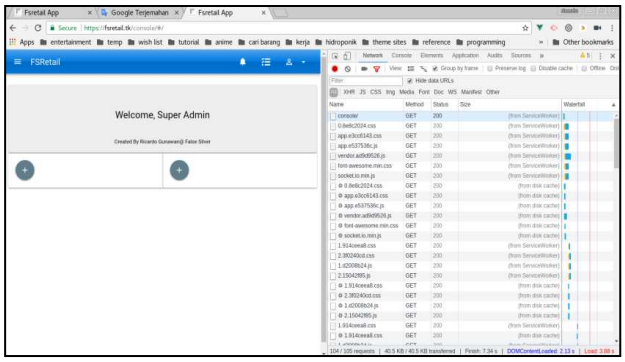
Nama	Layar	OS	Browser
Redmi 3 Pro	5” (720x1.280) pixel	Android 5.1 (Lollipop)	Chrome
HP14-g008AU	14” diagonal HD (1.366x768) piksel	Kali Linux Rolling	Chrome
Ipad 3	97” (24.64 cm)	IOS	Chrome

Uji coba juga dilakukan pada fitur PWA adalah fitur utama dari penelitian ini. PWA menyediakan banyak fitur antara lain *file caching*, *runtime caching*, *push notification*, dan *background sync*. Fitur *Background Sync* sudah diimplementasikan tetapi fitur ini tidak berjalan sesuai yang diharapkan. Berikut adalah penjelasan hasil uji coba dari fitur PWA:

1. *Progressive*

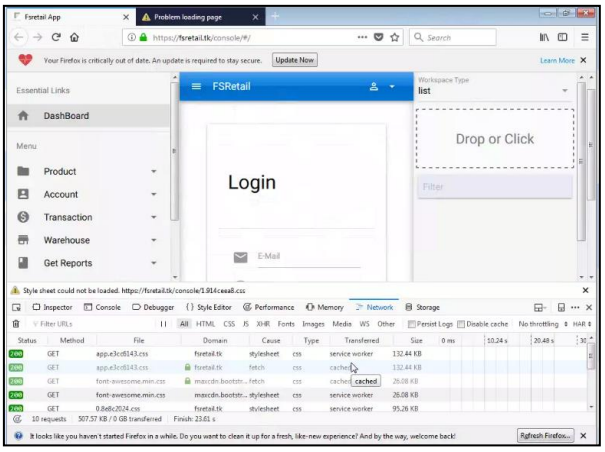
Uji coba ini dilakukan untuk memastikan PWA yang telah dibuat kompatibel dengan semua *browser* yang ada. Uji coba ini menggunakan beberapa *browser*, yaitu: *Chrome*, *Opera*, *Firefox*, *Safari*, dan *Microsoft Edge*. Pengujian ini dapat dicek pada bagian *inspect element* pada halaman *web*. Berikut uji coba kompatibel PWA dengan mengunakan berbagai *browser*:

- *Chrome*
  Chrome adalah *browser* buatan *Google*. Uji coba dilakukan dengan *Chrome* versi 64.0. Gambar 11 adalah bukti bahwa *Google Chrome* dapat menjalan Fitur PWA dengan baik.



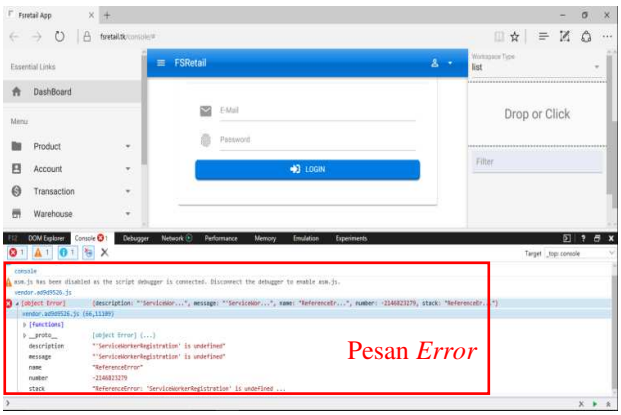
Gambar 11. Uji Coba *Browser Chrome*

- *Firefox*
  Firefox yang digunakan untuk uji coba adalah *browser Firefox* versi 63.0. Gambar 12 adalah bukti bahwa PWA dapat berjalan dengan baik pada *Firefox*.



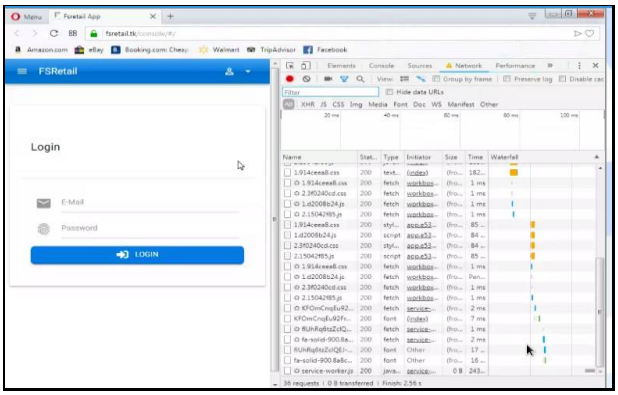
Gambar 12. Uji Coba *Browser Firefox*

- *Microsoft Edge*
  Browser yang digunakan adalah *browser Edge* 11. *Microsoft Edge* tidak mendukung *Service Worker*. Gambar 13 adalah bukti bahwa *browser Microsoft Edge* tidak kompatibel dengan PWA. Terdapat pesan error di bagian bawah (bewarna merah).



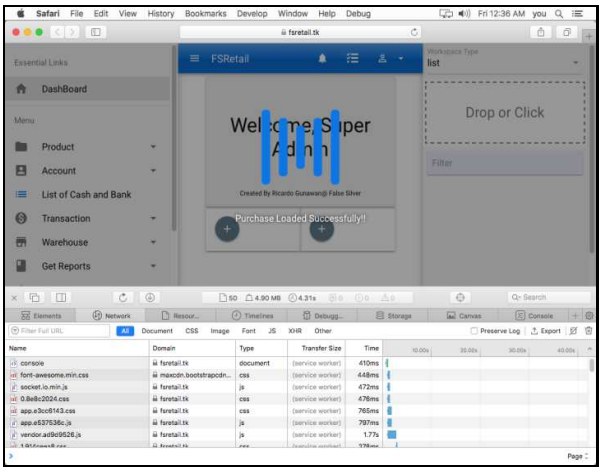
Gambar 13. Uji Coba *Browser Microsoft Edge*

- *Opera*
  Opera adalah *browser* yang berbasiskan *Chromium*. Gambar 14 adalah bukti bahwa *browser* ini kompatibel dengan PWA. Uji coba mengunakan *browser Opera* versi 55.



Gambar 14. Uji Coba *Browser Opera*

- *Safari*
  Uji coba dilakukan dengan menggunakan *browser Safari* versi 11. Gambar 15 adalah bukti bahwa *browser Safari* kompatibel dengan PWA.

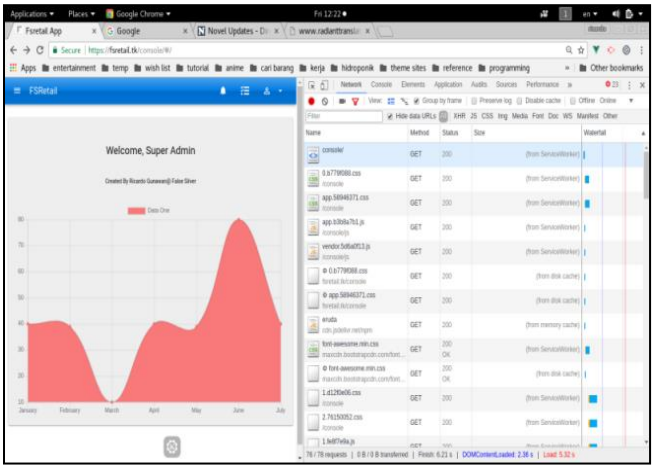


Gambar 15. Uji Coba *Browser Safari*

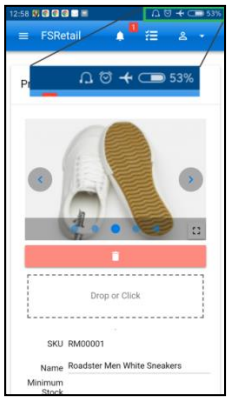
2. *File Caching*

Fitur *file caching* adalah fitur untuk menyimpan *file* statik (*file* yang jarang ada perubahan) pada *client*. Fitur ini diujicobakan dengan cara mematikan koneksi internet pada perangkat yang diuji dan akses halaman konsol (menggunakan *library workbook* yang sudah memanfatatkan kelima strategi yang sudah dijelaskan sebelumya). Halaman tersebut seharusnya dapat dimuat di *browser* karena *file* statik seperti CSS dan JS sudah ada di *client*. Hanya saja operasi transaksi tidak dapat dilakukan. Fitur untuk menyimpan *request* gagal dijelaskan pada fitur *background sync*.

Gambar 16 dan Gambar 17 adalah hasil uji coba dari fitur *file caching*. Gambar 16 adalah gambar uji coba pada laptop yang sudah diputuskan koneksi internet. Dapat dilihat *file statik* seperti CSS dan JS telah dikembalikan ke *browser* sebagai *request* berhasil walaupun tidak ada koneksi internet. Gambar 17 adalah uji coba pada *smartphone Xiaomi Redmi*. Fitur ini berjalan dengan lancar pada uji coba ini.



Gambar 16. Uji Coba *File Caching* Pada *Laptop*

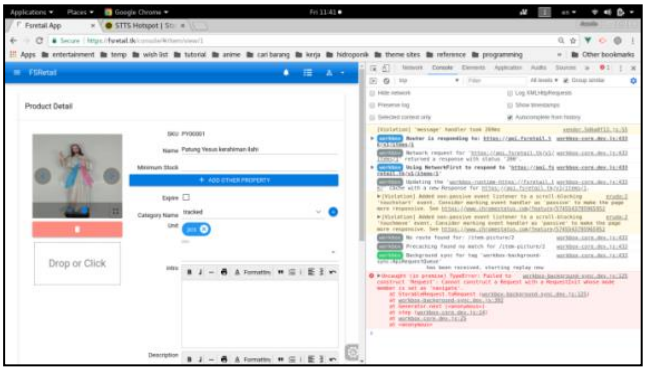


Gambar 17. Uji Coba *File Caching* Pada *Redmi 3 Pro*

3. *Runtime Caching*

Fitur *Runtime Caching* adalah fitur untuk mengingat *request* yang pernah dilakukan oleh pengguna. Fitur ini hanya menyimpan *request* yang bersifat dinamis dan *request* yang sederhana (*GET* dan *POST*). *Request* yang disimpan akan dikembalikan ke pengguna ketika koneksi internet putus.

Fitur ini akan diujicobakan dengan melakukan *request* ke *server backend* untuk melihat detail dari sebuah *item*. *Request* tersebut mempunyai URL '/item/{id}' ini bersifat dinamis karena parameter ID dapat dimasukan dengan ID yang ada di *server*. Gambar 18 dan Gambar 19 adalah hasil uji coba untuk fitur *Runtime Caching*. Gambar 18 menunjukkan uji coba yang dilakukan pada *laptop* dan Gambar 19 adalah uji coba *runtime caching* pada *Redmi 3 Pro*.



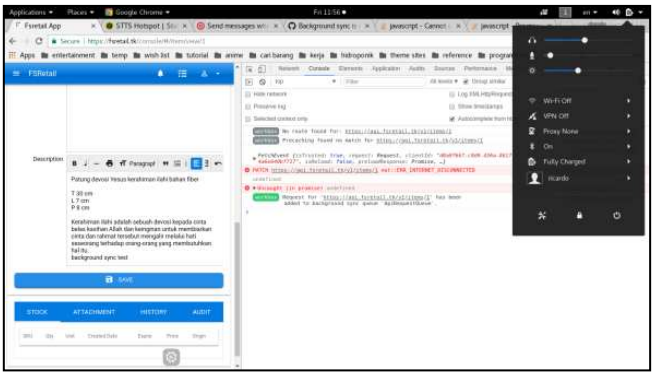
Gambar 18. Uji Coba *File Caching* Pada *Laptop*

Gambar 19. Uji Coba *File Caching* Pada *Redmi 3 Pro*

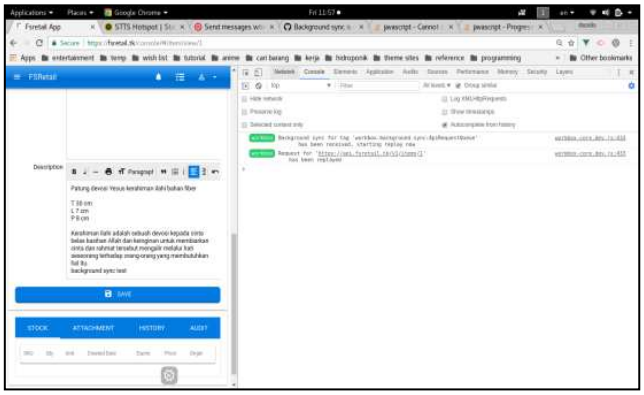
4. *Background Sync*

Background Sync adalah fitur PWA yang akan mengingat *request* yang dilakukan oleh pengguna saat tidak ada akses ke internet. *Request* tersebut akan dikirim ulang ketika *browser* merasa koneksi internet sudah dipulihkan. *Browser* akan memberikan *trigger event* yang bernama *'sync'* ketika perangkat telah terhubung dengan internet. *Request* yang gagal akan di kirim ulang ke *backend server*.

Gambar 20 dan Gambar 21 adalah hasil uji coba yang dilakukan pada laptop HP. Gambar 20 adalah proses uji coba dengan memutuskan koneksi ke internet dan melakukan *request* saat tidak ada koneksi. Gambar 21 adalah pengiriman *request* yang gagal dengan fitur *Background Sync*. Dapat dilihat Gambar 21 data sudah diperbarui pada *server*.



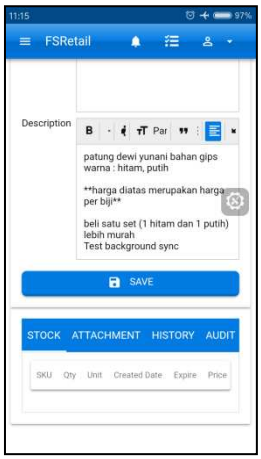
Gambar 20. Uji Coba *Background Sync*



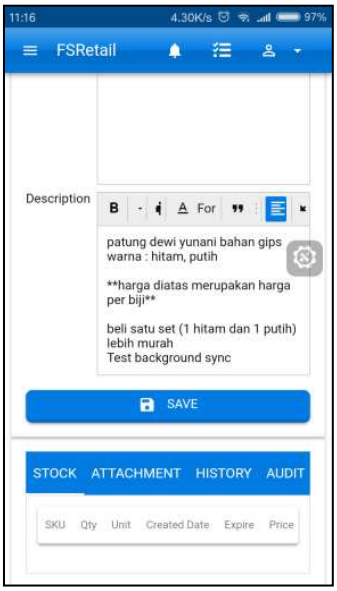
Gambar 21. Uji Coba *Background Sync* Data Baru

Fitur *Background Sync* tidak berjalan sesuai ekspetasi. Hal ini diakibatkan oleh *Background Sync* tidak mendukung *request* CORS (*Cross Origin Resource Sharing*). Saat *request* gagal terkirim *request* akan tersimpan pada *queue background sync* untuk dijalankan ulang ketika perangkat *online* kembali. *Request* CORS akan mengirim dua *request*, yaitu: *options request* dan *request* asli. *Options request* digunakan untuk memvalidasi apakah *server* memperbolehkan CORS dan jika boleh maka *request* asli akan dikirim. Hal ini menyebabkan jika *request* gagal *queue* akan menyimpan *request* yang pertama yaitu *options request* bukan *request* yang asli. Hal ini membuat fitur *Background Sync* kadang berjalan tidak sesuai dengan ekspetasi.

Gambar 22 dan Gambar 23 adalah hasil uji coba *Background Sync* menggunakan *Redmi 3 Pro*. Uji Coba tidak dapat dilakukan dengan melihat *Console Developmnet* pada perangkat *mobile* karena *Console* tidak dapat ditunjukan. Oleh karena itu uji coba dilihat mengunakan apakah data sudah berubah ketika pengguna *online*. Gambar 22 adalah gambar penambahan *string 'Test background sync'* pada sebuah deskripsi produk. Perangkat pada Gambar 22 dalam kondisi *offline*. Gambar 23 menunjukkan perangkat sudah dikoneksikan dengan internet serta halaman sudah di *load* ulang. Penambahan *string 'Test background sync'* pada deskripsi produk telah tersimpan ke *database* dengan adanya fitur *Background Sync*.



Gambar 22. Redmi 3 Pro Kondisi *Offline*

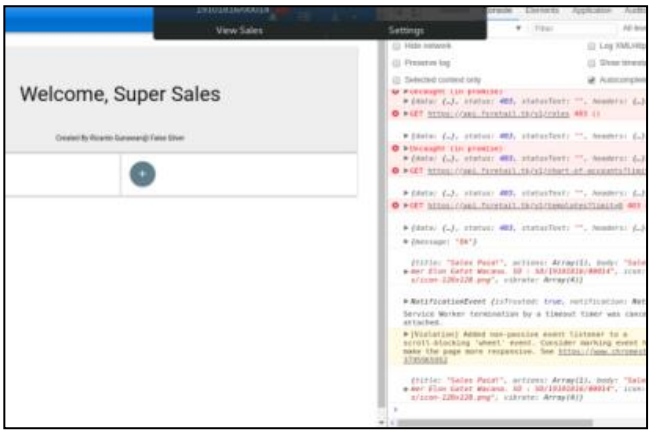


Gambar 23. Redmi 3 Pro Kondisi *Online*

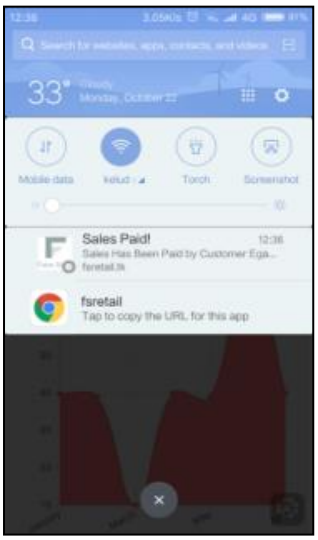
5. *Push Notification*

Push Notification adalah fitur untuk mengirim notifikasi ke perangkat *user*. Notifikasi akan terkirim saat *customer* membayar barang belanjaan, ada pembelian yang jatuh tempo untuk dibayar dan ada penjualan yang jatuh tempo. *Push notification* diujicobakan secara manual dengan

menggunakan *trigger event* tersebut dengan mengunakan *PHP console tinker*. Gambar 24 dan 25 adalah hasil uji coba dari fitur *Push Notification*. Gambar 24 adalah uji coba pada *laptop* HP yang menunjukan notifikasi pada bagian atas layar. Gambar 25 adalah hasil uji coba pada *Redmi 3 Pro*. Notifikasi sudah masuk dalam notifikasi list dan ada getaran untuk menotifikasi pengguna bahwa ada notifikasi.



Gambar 24. Uji Coba *Push Notification* Pada *Laptop*



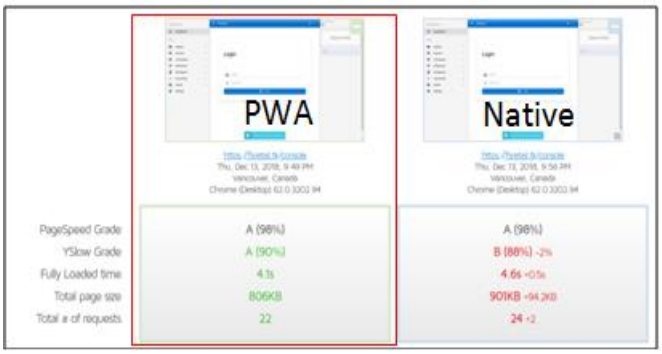
Gambar 25. Hasil *Push Notification* Pada *Redmi 3 Pro*

B. *Performance Testing*

Uji Coba ini akan mencoba bagaimana kinerja yang dilakukan dengan dua *benchmark* yaitu *Lighthouse* dan *GTMetrix*. *Light House* adalah *software open source* untuk meningkatkan kualitas aplikasi web dengan menilai *source code* dari aplikasi web tersebut [9]. *Light House* mejalankan serangkaian pengujian terhadap web dan digunakan untuk *benchmark* apakah *web app* sudah mengunakan *Best Pratice* dan *Service worker* sudah berjalan dengan baik. Gambar 26 adalah hasil *benchmark* dari *Lighthouse*. Nilai yang ditujukkan adalah nilai PWA (pemanfaatan *Progressive Web Apps*) sebesar 91%, *Accessibility* (kemudahan pengaksesan website) 89%, dan *Best Practice* (efisiensi penggunaan website) sebesar 94%.

Gambar 26. *Benchmark Light House*

GTMetrix adalah *performance benchmark* untuk mengukur bagaimana kecepatan *load* dari suatu *website*, yang memungkinkan pemantauan *URL* setiap jam dari lokasi yang berbeda sehingga bisa didapat gambaran lengkap mengenai kinerja web [10]. Gambar 27 adalah hasil *Benchmark* perbandingan antara *GTMetrix* menggunakan PWA dan *native*. Dari hasil *banchmark* kecepatan *load time* tidak terlalu signifikan, yaitu: 0,5 detik (didapatkan dari selisih waktu pengujian PWA yang memakan waktu sebesar 4,1 detik dengan pengujian *Native* yang memakan waktu sebesar 4,6 detik).



Gambar 27. *Benchmark GTMetrix*

VI. KESIMPULAN

Dari hasil uji coba aplikasi yang dikembangkan, terdapat tiga kesimpulan yang bisa diambil dari hasil uji coba. Pertama, *Progressive Web Apps* adalah teknologi internet yang digunakan untuk memecahkan masalah realibilitas data. Hal ini dapat dibuktikan dengan fitur *file caching*, *runtime caching*, dan *background sync*. *Progressive Web Apps* merupakan fitur yang harus digunakan untuk generasi *Web App* selanjutnya, karena dapat membantu kecepatan *load* dari website.

Kedua, pada ujicoba hasil *benchmark* dari *Lighthouse* menunjukkan nilai PWA (pemanfaatan *Progressive Web Apps*) sebesar 91%, *Accessibility* (kemudahan pengaksesan website) 89%, dan *Best Practice* (efisiensi penggunaan website) sebesar 94%. Kesimpulan terakhir, fitur *Background Sync* tidak berjalan sesuai ekspektasi. Hal ini telah diujicobakan dengan *request* sebelum dan setelah *offline*. Hal ini dapat diatasi dengan menggunakan CORS (*Cross Origin Resource Sharing*) yang ada dari *Quasar Framework*.

REFERENSI

[1] Porter, D. (2001). *Financial Accounting*. Virginia: Tidewater Community College.

[2] Adi, L. (2017). Platform E-Learning untuk Pembelajaran Pemrograman Web Menggunakan Konsep Progressive Web Apps. *Surabaya: Jurnal Teknik ITS*, Vol. 6(2), pp. A579-A583.

[3] Tandel, S. (2018). Impact of Progressive Web Apps on Web App Development. *International Journal of Innovative Research in Science, Engineering and Technology*, Vol. 7(9), pp. 9439-9444.

[4] Zalt, M. (2016). *Porto SAP*. Diakses dari https://github.com/Mahmoudz/Porto.

[5] Christensen, J.H. (2009). Using RESTful web-services and cloud computing to create next generation mobile applications. *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications*, pp. 627-634.

[6] Gore, A. (2017). *Full-Stack Vue.js 2 and Laravel 5*. Brimingham: Packt Publishing.

[7] Stoenescu, R. (2015). *Quasar Framework*. Diakses dari http://quasar-framework.org/.

[8] Midtrans. (2012). Midtrans. Diakses dari https://midtrans.com.

[9] Google Developers. (2017). *Lighthouse*. Diakses dari https://developers.google.com/web/tools/lighthouse/.

[10] Singh, T. (2014). *Performance Testing of Any Website "Gtmetrix Tool"*. Diakses dari https://gtmetrix.com/