BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

2.1. Tinjauan Pustaka

Penelitian dari Afif Rizki Kurniawan pada tahun 2018, membahas tentang aplikasi lowongan pekerjaan yang menggunakan teknologi *Service Worker* pada penerapan *Proggresive Web apps*. Tujuan dari penelitian ini adalah penerapan *Progressive Web apps* pada aplikasi lowongan pekerjaan yang dapat membantu pencari pekerjaan untuk mendapatkan informasi yang mudah dan cepat serta membantu para pencari kerja di Lingkup STMIK AKAKOM untuk melihat informasi pekerjaan walaupun konektifitas terganggu atau *offline*.

Penelitian dari Nur Najmi Wicaksana pada tahun 2019, membahas tentang aplikasi *monitoring* untuk service laptop yang menggunakan teknologi *Service Worker* pada penerapan *Progressive Web apps*. Tujuan dari penelitian ini adalah membuat sebuah aplikasi *Progressive Web apps* yang dapat digunakan untuk melihat status terkini pengerjaan di Bengkel OS.

Penelitian dari Deviana Wulandari pada tahun 2019, membahas tentang aplikasi pelacakan pengiriman barang dengan penerapan *Proggresive Web apps*. Tujuan dari penelitian ini adalah untuk merancang dan membangun aplikasi berbasis *web* yang dapat melakukan pelacakan status barang dan juga digunakan untuk pelaporan pengiriman barang secara komputerisasi dengan menggunakan teknologi *Progressive Web apps* (PWA)

Penelitian dari Gharizi Matiini, Rahmat Setiyadi, Adi Setiawan, dan M. Ramli pada tahun 2021, membahas tentang pengembangan aplikasi *E-learning* Bahasa Inggris dengan teknologi *Proggresive Web apps*. Tujuan dari penelitian ini adalah menghasilkan suatu aplikasi yang dapat berjalan pada sistem operasi *Android* dan *iOS* berupa PWA dengan memanfaatkan website Pusat Bahasa ITI sebagai media pembelajaran Bahasa Inggris dan melakukan evaluasi terhadap *English Grammar Online Course* yang telah diujicobakan.

Penelitian dari Fiko Hanif Putra Ramadan pada tahun 2021, membahas tentang aplikasi penjualan bawang putih menggunakan penerapan *Progressive Web apps*. Tujuan dari penelitian ini adalah membuat sebuah aplikasi penjualan bawang putih yang digunakan untuk kelompok tani dalam mengelola bawang putih petani untuk dijual melalui aplikasi berbasis web.

Dalam penelitian ini akan dilakukan pengujian dari dua aplikasi web katalog restoran yang sebelumnya telah dikembangkan secara tradisional (biasa) dan yang dikembangkan menggunakan *Progressive Web app* (PWA) serta kedua aplikasi telah dibuat dengan struktur yang sama yakni implementasi DOM, REST API, dan sama-sama bersifat *responsive*. Tujuan dari penelitian ini adalah melakukan untuk membuktikan bahwa implementasi dari *Progressive Web app* pada aplikasi web menjadi solusi dari permasalahan yang ada pada pengembangan *web app* secara tradisional (biasa) khususnya *web app* katalog restoran dari segi *Web Vitals*, performa, dan aksesbilitas aplikasi web sehingga memberikan pengalaman pengguna yang baik bagi pengunjung.

Tabel 2. 1 Tinjauan Pustaka

Penulis	Studi Kasus	Metode	Interface
Afif Rizki Kurniawan (2018)	Akakom Career Center	Progressive Web apps	Web
Nur Najmi Wicaksana (2019)	Service Laptop Bengkel OS	Progressive Web apps	Web
Deviana Wulandari (2019)	Jasa Pengiriman PT.ORIFLAME	Progressive Web apps	Web
Antonius Angga Kurniawan (2020)	Performa Perangkat Mobile	Progressive Web apps	Android dan iOS
Fiko Hanif Putra Ramadan (2021)	Penjualan Bawang Putih	Progressive Web apps	Web
Gharizi Matiini, Rahmat Setiyadi, Adi Setiawan, dan M. Ramli (2021)	E-Learning Bahasa Inggris (KELAS ENGLISH GRAMMAR ONLINE COURSE)	Progressive Web apps	Android dan iOS
Yang diteliti	Pengujian aplikasi web Katalog restoran	Progressive Web apps	Web

2.2. Dasar Teori

2.2.1. Proggresive *Web app*s

Istilah *Progressive Web apps* pertama diperkenalkan oleh Alex Russel,salah seorang *Google Chrome Engineer*, dan Frances Berriman, designer yang bekerja untuk Google pada tahun 2015. Google mendefinisikan bahwa *Progressive Web apps* menggunakan kemampuan *web modern* untuk memberikan pengalaman pengguna seperti aplikasi. Mereka berkembang dari halaman di *tab browser* menjadi aplikasi tingkat atas yang *imersif*, menjaga gesekan web yang rendah setiap saat.

Tujuan hadirnya PWA adalah memungkinkan *web developer* mengubah web yang sudah ada agar bisa berperilaku layaknya aplikasi *mobile native* tanpa banyak perubahan atau menambah programmer khusus. Tujuan ini bisa dicapai berkat kumpulan teknologi *web*. Ia bisa memanfaatkan kelebihan-kelebihan web

dan aplikasi *mobile native* sekaligus. Kumpulan teknologi ini memungkinkan aplikasi *web* tradisional mampu diakses tanpa koneksi internet (*offline*), bisa dipasang di homescreen seperti aplikasi *native*, bisa melakukan sinkronisasi data ke *server*, bisa mengirim *push notification* dan lain-lain. Berikut detail dari karakteristik PWA:

1. Progresif - Bekerja untuk setiap pengguna. Tak peduli apapun browsernya, tidak masalah. PWA dibangun dengan peningkatan progresif sebagai intinya.
2. Responsif - Mampu menyesuaikan dengan berbagai perangkat. Baik itu *desktop*, seluler, tablet, atau yang lainnya.
3. Konektivitas independen - *Service Worker* membantu meningkatkan proses load time ketika internet memiliki kualitas rendah, bahkan dapat diakses dalam keadaan *offline*.
4. Seperti Aplikasi *Native* - *Experience* yang diberikan tak kalah dengan aplikasi *native*, karena PWA dibangun dengan struktur *Application Shell*.
5. Aman - PWA mewajibkan web untuk berjalan pada HTTPS. Ini tentu membuat *web* aman dari berbagai ancaman.
6. Dapat ditemukan - Terdefinisi sebagai “Aplikasi” berkat *Web app Manifest* dan *service worker*. Dan dapat mudah ditemukan oleh search engine seperti Google.

7. Re-engageable - Dengan fitur pemberitahuan seperti *push notification*, dapat mengajak (*engaged)* kembali pengguna untuk menggunakan aplikasi.
8. Dapat dipasang - Memungkinkan pengguna untuk "memasang" di layar beranda tanpa melalui *application store*.
9. Bisa ditautkan - Dapat dengan mudah dibagikan melalui URL.

Dalam membangun PWA ada beberapa komponen yang harus digunakan. Berikut adalah penjelasan ringkas dari beberapa komponen PWA yang akan kita pelajari. Ada beberapa komponen wajib dan ada juga komponen *opsional*. Komponen wajib adalah komponen yang selalu digunakan setiap kali membuat PWA, sedangkan komponen opsional adalah komponen yang tidak mempengaruhi kinerja PWA namun dapat digunakan untuk memperkaya fitur PWA. Berikut komponen wajib yang dimaksud:

1. *Application Shell*
3. *Web app Manifest*
4. *Service Worker*
5. *Cache API*
6. *Fetch API*

2.2.2. Web Tradisional

Web tradisional adalah sebuah aplikasi web yang dikembangkan menggunakan teknologi web yang lebih lama seperti HTML, CSS, dan JavaScript. Aplikasi web tradisional dapat diakses melalui browser web dan ditampilkan di halaman yang di-load secara keseluruhan. Halaman web tradisional umumnya tidak

memiliki interaksi yang kompleks dan tidak dapat digunakan offline. Pengalaman pengguna dari web tradisional lebih sederhana dibandingkan dengan aplikasi web modern seperti PWA (Progressive *Web app*) atau WEB *Native*, karena web tradisional tidak memiliki fitur seperti push notification, offline access, dan dapat diinstal seperti aplikasi *native*.

2.2.3. JavaScript

JavaScript adalah bahasa pemrograman tingkat tinggi yang pada awalnya dikembangkan untuk membuat website menjadi lebih "hidup". Bersama dengan HTML dan CSS, JavaScript menjadi bahasa pemrograman paling populer untuk mengembangkan aplikasi berbasis web. Bahasa ini mampu memberikan *logic* ke dalam *website*, sehingga website tersebut memiliki fungsionalitas tambahan dan lebih interaktif.

Awalnya JavaScript dibuat supaya dapat berjalan di lingkungan *browser* dan membuat *website* menjadi lebih interaktif. Namun, saat ini Anda sebagai developer dapat menggunakan bahasa pemrograman JavaScript di berbagai lingkungan pengembangan. Sehingga, tidak hanya sebatas *browser* atau *client*, namun JavaScript juga bisa berjalan di server menggunakan Node.js.

JavaScript termasuk ke dalam kategori *scripting language*. Di mana salah satu ciri-ciri utama dari bahasa scripting adalah kode tidak perlu dikompilasi agar bisa dijalankan. *Scripting language* menggunakan *interpreter* untuk menerjemahkan kode atau perintah yang kita tulis supaya dimengerti oleh mesin.

2.2.4. Service Worker

Service Worker merupakan sebuah berkas JavaScript yang diproses oleh browser di background. Berkas ini dieksekusi secara terpisah tak seperti berkas JavaScript biasa yang membentuk *website*. *Service Worker* menjadi gerbang bagi berbagai fitur *browse*r yang tidak memerlukan tampilan atau interaksi dengan pengguna. Kemampuan utama dari *Service Worker* adalah mengambil alih seluruh urusan *request* pada *browser*.

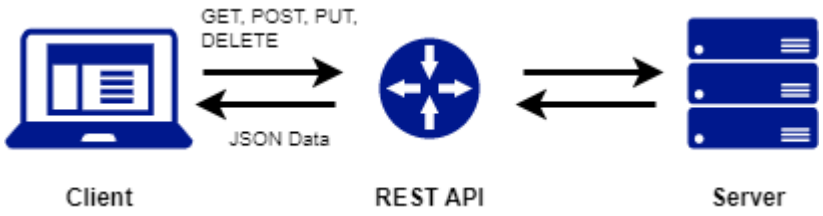
Dengan adanya *Service Worker* sebagai perantara (*proxy*) kita dapat menerapkan sebuah logika sebelum *request* yang dilakukan *browser* benar-benar dikirimkan ke *server*, begitu pula sebaliknya. Contohnya, kita dapat menyimpan response yang didapat dari *web server* ke dalam *Cache API* sebelum datanya ditampilkan pada jendela *browser*, contoh lain kita bisa mengevaluasi sebuah *request* apakah sudah terdapat pada *cache* atau belum, sebelum dikirimkan ke *server*. Dengan *service worker*, kita bisa melakukan apapun terhadap *request* dan *response* yang terjadi pada *browser*. Karena *Service Worker* memegang posisi penting dalam jalur transaksi data, *Service Worker* hanya dapat dijalankan pada protokol HTTPS. Pemaksaan ini bertujuan untuk meningkatkan kemanan aplikasi.

Bila tidak seperti itu, sangat rentan terhadap serangan yang disebut "*man-in-the-middle*". Namun selama proses pengembangan, *Service Worker* dapat berjalan pada *localhost*.

2.2.5. REST API

REST API merupakan salah satu dari desain arsitektur yang terdapat di dalam API itu sendiri. Dan cara kerja dari RESTful API yaitu REST *client* akan

melakukan akses pada data atau *resource* pada REST *server* di mana masing-masing resource. Data atau *resource* tersebut akan dibedakan oleh sebuah *global* ID atau URIs (*Universal Resource Identifiers*)



Gambar 2. 1 Model REST API

Jadi, nantinya data yang diberikan oleh REST *server* itu dapat berupa format *text*, JSON atau XML. Dan saat ini format yang paling populer dan paling banyak digunakan adalah format JSON.

Adapun metode HTTP yang secara umum dipakai dalam REST API adalah:

1. GET, berfungsi untuk membaca data atau *resource* dari REST *server*.
2. POST, berfungsi untuk membuat sebuah data atau *resource* baru di REST *server*.
3. PUT, berfungsi untuk memperbaharui data atau *resource* di REST *server*.
4. DELETE, berfungsi untuk menghapus data atau *resource* dari REST *server*.
5. OPTIONS, berfungsi untuk mendapatkan operasi yang disupport pada *resource* dari REST *server*.

2.2.6. Node.JS

Node.js adalah runtime environment untuk JavaScript yang bersifat *open-source* dan *cross-platform*. Dengan Node.js kita dapat menjalankan kode JavaScript

di mana pun, tidak hanya terbatas pada lingkungan *browser*. Node.js menjalankan V8 JavaScript *engine* (yang juga merupakan inti dari Google Chrome) di luar browser. Ini memungkinkan Node.js memiliki performa yang tinggi.

Node.js juga menyediakan banyak *library* atau *module* JavaScript yang membantu menyederhanakan pengembangan aplikasi *web*. Berikut ini adalah beberapa fitur penting dari Node.js yang menjadikannya pilihan utama dalam pengembangan aplikasi:

1. *Asynchronous & Event-driven*

   Semua API dari Node.js bersifat *asynchronous*, artinya tidak memblokir proses lain sembari menunggu satu proses selesai. *Server* Node.js akan melanjutkan ke ke pemanggilan API berikutnya lalu memanfaatkan mekanisme *event notification* untuk mendapatkan respon dari panggilan API sebelumnya.

2. *Very Fast*

   Eksekusi kode dengan Node.js sangat cepat karena berjalan pada V8 JavaScript Engine dari Google Chrome.

3. *Single Threaded but Highly Scalable*

   Node.js menggunakan model *single thread* dengan *event looping*. Mekanisme ini membantu server untuk merespon secara *asynchronous* dan menjadikan *server* lebih *scalable* dibandingkan *server* tradisional yang menggunakan banyak *thread* untuk menangani permintaan.

Node.js dirancang untuk aplikasi dengan proses *I/O* yang intensif seperti *network server* atau *backend* API. Pemrograman dengan *multi threading* relatif lebih berat dan sulit untuk dilakukan. Jika kita ingin membuat web server yang

bisa menangani ratusan *request* bersamaan, menggunakan ratusan *thread* akan membutuhkan memori yang besar. Oleh karena itu, karakteristik Node yang *asynchronous* dan *single thread* dirancang untuk memungkinkan implementasi *server* yang dapat menangani banyak *request* pada waktu yang sama.

2.2.7. Postman

Postman adalah sebuah aplikasi yang digunakan untuk melakukan tes dan debugging pada API (Application Programming Interface). Fitur-fitur yang disediakan oleh Postman seperti pengiriman request HTTP, pembuatan dokumentasi, dan pembuatan skrip automatis dapat membantu developer dalam proses pengembangan aplikasi yang menggunakan API.

2.2.8. Lighthouse

Lighthouse adalah sebuah alat analisis yang bisa dijalankan sebagai ekstensi browser atau dari *command line* yang digunakan untuk mengevaluasi kualitas website dari sisi teknis maupun *user experience*. Alat ini dapat memeriksa hal-hal seperti kecepatan loading, kompatibilitas mobile, aksesibilitas, dan optimisasi SEO. Lighthouse dikembangkan oleh Google dan dapat digunakan pada browser seperti Google Chrome, Firefox dan Brave.

2.2.9. WebPageTest

WebPageTest adalah sebuah alat analisis website yang digunakan untuk menguji kecepatan dan kinerja website. Alat ini dapat digunakan untuk menguji website dari berbagai lokasi geografis dan perangkat, termasuk desktop dan mobile. WebPageTest menyediakan berbagai macam ukuran yang dapat digunakan untuk mengevaluasi kinerja website, seperti waktu *loading*, ukuran halaman, dan jumlah

request. Selain itu, alat ini juga menyediakan visualisasi data yang jelas yang dapat digunakan untuk mengidentifikasi dan mengatasi masalah yang mungkin ada pada website.

2.2.10. Web Vitals

Web Vitals adalah sekumpulan ukuran kinerja website yang diakui penting oleh Google untuk menentukan kualitas user experience dari sebuah website. *Web Vitals* mencakup tiga kategori utama, yaitu:

1. *Speed*: Ukuran yang mengukur kecepatan loading website, seperti *Time to First Byte* (TTFB) dan Speed Index.
2. Interaktivitas: Ukuran yang mengukur kinerja website dari sisi interaktivitas, seperti *First Input Delay* (FID) dan *Total Blocking Time* (TBT).
3. *Visual Stability*: Ukuran yang mengukur stabilitas visual website, seperti *Cumulative Layout Shift* (CLS)

Web Vitals adalah ukuran kinerja yang dapat digunakan untuk mengevaluasi website secara keseluruhan, di mana ukuran ini dapat digunakan untuk mengevaluasi website dari sisi kecepatan, interaktivitas, dan stabilitas visual. Dengan mengoptimalkan *Web Vitals*, website akan lebih baik dalam memberikan pengalaman pengguna yang baik bagi pengunjung.