Perancangan *Web*site *Text Scanner* Untuk Konversi *Handwritten* Ke Teks Digital Dengan Menggunakan *Optical Character Recognition*

Design a Website text scanner for Handwitten to Digital Text Conversion Using Optical Character Recognition

1st Hafizh Ghiyats Ash Shiddiq
Fakultas Ilmu Terapan
Universitas Telkom
Bandung, Indonesia
hafizhh@student.telkomuniversity.ac.id

2nd Muhammad Iqbal
Fakultas Ilmu Terapan
Universitas Telkom
Bandung, Indonesia
miqbal@telkomuniversity.ac.id

3rd Aris Hartaman
Fakultas Ilmu Terapan
Universitas Telkom
Bandung, Indonesia
arishartaman@telkomuniversity.ac.id

Abstrak — Pesatnya perkembangan teknologi saat ini menuntut kita untuk mengembangkan cara yang lebih efisien dan efektif dalam mengkonversi tulisan tangan maupun teks dokumen dalam bentuk *hardfile* menjadi teks digital. Oleh karena itu, dibutuhkan sebuah *platfrom* yang menggunakan teknologi *Optical Character Recognition* (OCR) untuk melakukan scan tulisan tangan maupun teks dokumen dalam bentuk *hardfile* dan mengubahnya menjadi teks digital yang dapat diedit dan disimpan Dengan demikian, dibuatkannya *web*site yang dapat melakukan scan tulisan tangan dan mengubahnya menjadi teks digital. *Web*site ini dirancang untuk memudahkan pengguna dalam mengkonversi tulisan tangan ke dalam format digital tanpa perlu melakukan proses manual. Pengguna hanya perlu mengunggah gambar yang terdapat tulisan tangan tersebut ke dalam *web*site. OCR akan melakukan pemindaian pada gambar dan mengubahnya menjadi teks digital. Dalam perancangan *web*site ini, diperhatikannya faktor-faktor penting seperti kecepatan dan akurasi pengenalan karakter oleh OCR. *Web*site ini juga dilengkapi dengan fitur pengeditan teks sehingga pengguna dapat mengoreksi kesalahan pengenalan karakter oleh OCR. Perancangan ini melibatkan pengujian *text recognition* oleh tesseract.js sebanyak 10 kali untuk gambar tulisan tangan dan 10 kali untuk dokumen *hardfile* dan pengujian performa *web*site. Hasil pengujian menunjukkan bahwa tingkat akurasi rata-rata pengujian OCR untuk dokumen *hardfile* adalah 99,69% dan 91,27% untuk gambar tulisan tangan.

***Kata kunci — website*, OCR, *optical character recognition*, teks digital, tesseract.js**

Abstrak — The rapid advancement of technology today demands us to develop more efficient and effective ways to convert handwritten text into digital text. Therefore, a platform is needed that utilizes Optical Character Recognition (OCR) technology to scan handwritten text and convert it into editable and savable digital text. Consequently, a website has been created that can scan handwritten text and convert it into digital text. This website is designed to facilitate users in converting handwritten text into a digital format without the need for manual processes. Users only need to upload an image containing the handwritten text to the website. This The OCR will scan the image and convert it into digital text. In designing this website, important factors such as the speed and accuracy of character recognition by the OCR are taken into account. The website is also equipped with a text editing feature so that users can correct character recognition errors made by the OCR. This design involved testing text recognition by tesseract.js 10 times for handwritten images and 10 times for hardfile documents and website performance testing. The test results show that the average accuracy level of OCR testing for hardfile documents is 99.69% and 91.27% for handwritten images.

Kata kunci — website, OCR, optical character recognition, digital text, tesseract.js

I. PENDAHULUAN

Perkembangan teknologi dan inovasi telah memberikan kemajuan pada berbagai aspek kehidupan, termasuk dalam bidang digitalisasi dokumen. Salah satu inovasi yang cukup dominan adalah teknologi teks *recognition* untuk tulisan tangan maupun teks dokumen dalam bentuk *hardfile*. Teknologi pengenalan teks, atau yang lebih dikenal dengan *Optical Character Recognition* (OCR) [1], telah menjadi bagian integral dalam proses digitalisasi dokumen. OCR

memungkinkan kita untuk mengubah dokumen *hardfile* atau tulisan tangan menjadi teks digital yang dapat diedit dan disimpan.[2].

Namun, penting juga untuk mempertimbangkan kemudahan penggunaan dan aksesibilitas teknologi ini. Salah satu solusi yang mungkin adalah dengan mengintegrasikan teknologi OCR ke dalam sebuah *platfrom online*, seperti *web*site, yang dapat diakses oleh pengguna kapan saja dan dimana saja.[3] *Web*site ini dirancang untuk kemudahan pengguna dalam mengkonversi tulisan tangan ataupun *hardfile* dokumen lainnya ke dalam format digital tanpa perlu melakukan proses manual. Pengguna hanya perlu mengunggah gambar yang terdapat teks ataupun tulisan.[4]

Dalam perancangan *web*site ini, diperhatikan beberapa faktor penting seperti kecepatan dan akurasi pengenalan karakter oleh OCR.[5] Selain itu, *web*site ini juga dilengkapi dengan fitur pengeditan teks sehingga pengguna dapat mengoreksi kesalahan pengenalan karakter yang dilakukan oleh OCR. Pada implementasinya, ada beberapa kebutuhan yang dapat diproses oleh teknologi OCR, salah satu contohnya adalah laporan keuangan Usaha Mikro Kecil Menengah (UMKM) yang ditulis dengan menggunakan tangan.

Pada perancangan sebelumnya, implementasi OCR ini tidak mengintegrasikan teknologi OCR kedalam sebuah *platfrom online* seperti *web*site. Karena itu, dengan adanya *web*site dan kemajuan teknologi serta inovasi ini, diharapkan dapat mempercepat dan membuat proses digitalisasi dokumen menjadi lebih efisien dan efektif, terutama untuk dokumen yang ditulis tangan. *Web*site ini tidak hanya dapat membantu mengurangi kesalahan yang mungkin terjadi saat melakukan pengetikan manual, tetapi juga membantu individu dan organisasi dalam mengelola informasi dan data mereka dengan lebih baik.

I. KAJIAN TEORI

A. *Optical Character Recognition* (OCR)

Optical Character Recognition (OCR) merupakan suatu sistem yang digunakan untuk mengkonversi karakter huruf, angka, simbol dan karakter pada suatu gambar dalam berbagai bahasa yang digunakan agar dapat dibaca oleh mesin. Input pada OCR umumnya berupa tulisan tangan atau karakter dari cetakan mesin.[1]

B. *Image* Preprocessing

Untuk keperluan pemrosesan gambar metode pra-pemrosesan gambar dimaksudkan untuk meningkatkan kualitas gambar. Tujuan utama dari pra-pemrosesan adalah menghilangkan *noise*, menghapus distorsi, dan menonjolkan atribut lain yang penting untuk pemrosesan gambar.

GAMBAR 2.1 GAMBAR SETELAH DILAKUKAN *IMAGE* PREPORCESSING

C. TesseractJs

Tesseract.js adalah sebuah *library* Javascript yang dapat mengenali kata-kata dalam gambar. Ini adalah *port* javascript murni dari mesin OCR Tesseract yang populer. *Library* ini mendukung lebih dari 100 bahasa, orientasi teks otomatis, dan deteksi skrip, serta menyediakan antarmuka sederhana untuk membaca kotak pembatas paragraf, kata, dan karakter. Tesseract.js melibatkan *port web*assembly dari Mesin OCR Tesseract. Ini bekerja di browser menggunakan *web*pack, esm, atau tag script biasa dengan CDN dan di server dengan Node.js. Fungsi utama Tesseract.js (*mis. recognize, detect*) mengambil parameter gambar.

D. NodeJs

Node.js adalah *platform* yang memungkinkan pengembang untuk menulis kode JavaScript yang dapat dijalankan di server, bukan hanya di *browser*. Ini dibangun di atas mesin JavaScript V8 milik Chrome dan bersifat open-source. Node.js memungkinkan pembuatan aplikasi *web* skala besar yang dapat menangani banyak koneksi dan operasi I/O secara simultan. Selain itu, Node.js juga dilengkapi dengan npm (*Node Package Manager*), sebuah manajer paket yang memudahkan penginstalan dan manajemen *library* yang dibutuhkan oleh aplikasi dengan Node.js, JavaScript dapat digunakan untuk berbagai jenis aplikasi, mulai dari aplikasi *web* hingga utilitas baris perintah.[9]

E. ReactJs

React.js adalah sebuah perpustakaan JavaScript yang memungkinkan pembuatan antarmuka pengguna yang interaktif dan dinamis. Dibangun di atas mesin JavaScript V8 milik Chrome, React.js memungkinkan pengembang untuk membangun aplikasi *web* dengan menggunakan komponen-komponen yang dapat digunakan kembali. Dengan React.js, pengembang dapat membangun antarmuka pengguna dari komponen-komponen individu yang dapat digabungkan menjadi layar, halaman, dan aplikasi lengkap. Komponen-komponen ini dapat dibuat oleh pengembang sendiri atau oleh orang lain, dan dapat digabungkan untuk menciptakan antarmuka pengguna yang kompleks. [9]

F. Tailwind CSS

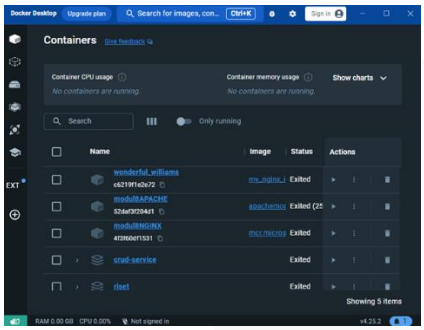
Tailwind CSS adalah sebuah kerangka kerja CSS (*Cascading Style Sheets*) yang memungkinkan pembuatan situs *web* dengan cepat dan efisien. Ini menggunakan pendekatan berbasis utilitas, di mana Anda dapat membangun desain langsung dalam markup Anda dengan menggabungkan berbagai kelas seperti flex, pt-4, text-center, dan rotate-90. Proses kerja Tailwind CSS melibatkan pemindaian semua *file* HTML (*Hypertext Markup Language*), komponen JavaScript, dan template untuk mencari nama kelas.[10]

G. MongoDB

MongoDB adalah *platform database* NoSQL yang memungkinkan pengembang untuk membangun aplikasi dengan menggunakan database modern terkemuka. Dengan MongoDB, pengembang dapat mengirim dan mengulangi data dengan kecepatan yang lebih tinggi berkat model data dokumen yang fleksibel dan antarmuka kueri yang seragam untuk setiap kasus penggunaan. Platform ini juga memungkinkan pengembang untuk memenuhi *Service Level Agreement* (SLA) mereka di lingkungan apa pun, baik untuk pelanggan pertama mereka atau untuk jutaan pengguna di seluruh dunia. MongoDB menawarkan fitur yang memastikan ketersediaan tinggi, melindungi integritas data, dan memenuhi standar keamanan dan kepatuhan untuk beban kerja yang kritis.

H. Docker

Docker adalah sebuah aplikasi *open-source* yang berfungsi untuk mengembangkan, mendistribusi, serta menjalankan *software application*. Desain arsitektur docker memudahkan untuk mendistribusi serta mengembangkan aplikasi secara lebih cepat karena docker memiliki sifat *lightweight containerization* yang dilengkapi dengan beragam komponen serta fitur sehingga mampu memudahkan para developer untuk mengembangkan dan memantau kinerja aplikasi yang diciptakan.[11]



GAMBAR 2.2 DOCKER

I. Visual Studio Code

Visual Studio Code, atau VS Code, adalah editor kode yang berjalan di desktop Anda dan tersedia untuk Windows, macOS, dan Linux. Meski ringan, VS Code sangat kuat dan mendukung berbagai bahasa pemrograman seperti JavaScript, TypeScript, Python, C#, Java, Go, Ruby, dan lainnya. Fitur IntelliSense pada VS Code membantu dalam mendeteksi potongan kode yang belum selesai dan secara otomatis membuat sintaks variabel umum dan deklarasi variabel. Jika bahasa pemrograman yang ingin digunakan tidak didukung, pengguna dapat mengunduh dan menggunakan ekstensi. Ekstensi ini tidak mempengaruhi kinerja editor karena berjalan sebagai proses yang terpisah.[9]

J. *Microservices*

Arsitektur *microservices* adalah pendekatan dalam pengembangan aplikasi di mana aplikasi dibagi menjadi serangkaian layanan kecil yang berjalan secara independen dan berkomunikasi melalui API (*Application Programming Interface*).[12] Setiap layanan ini dirancang untuk melakukan tugas tertentu dan dapat berjalan pada prosesnya sendiri, memungkinkan aplikasi untuk lebih tahan terhadap kerusakan dan memfasilitasi proses *agile*.

II. PERANCANGAN DAN PENGEMBANGAN

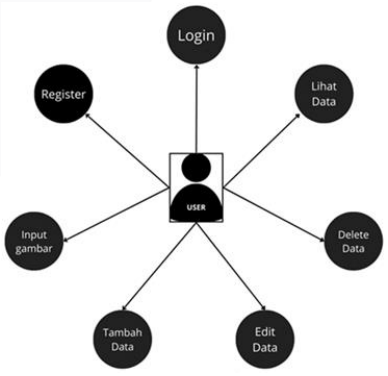
A. Sistem *Web*site Text Scanner

*Web*site text scanner sebagai sistem digitalisasi dokumen menggunakan OCR. Dokumen yang sudah di *scan* dan menjadi gambar digital, lalu di input kedalam *web*site, gambar akan memalui proses ke tahap selanjutnya yaitu *image preprocessing*. Setelah itu, OCR akan melakukan pengenalan karakter dan proses penerjemahan lalu hasilnya akan ditampilkan dalam bentuk teks digital lalu di *submit* kedalam *database* ataupun dapat disimpan di *local storage* dan riwayat teks digital dapat dilihat didalam *history web*site. Berikut merupakan ilustrasi pada Sistem *Web*site *Text scanner* yang dapat dilihat pada Gambar 3.1 dibawah ini

GAMBAR 3.1 GAMBARAN SISTEM *WEB*SITE *TEXT SCANNER*

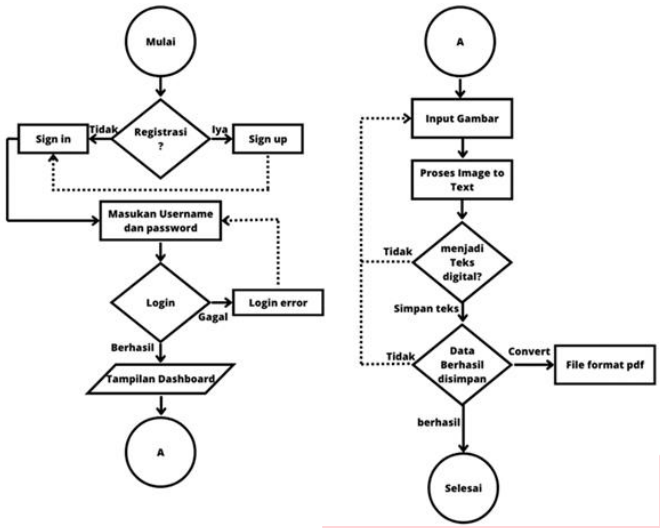
B. *Use Case* Diagram

Pembuatan diagram *use case* berdasarkan data yang telah dikumpulkan. Diagram use case adalah salah satu jenis diagram UML (*Unified Modelling Language*) yang menggambarkan interaksi antara sistem dan aktor. Dalam perancangan *web*site *text scanner*, terdapat sistem untuk pengguna yang dapat dilihat pada Gambar 3.2.



GAMBAR 3.*2 USE CASE* DIAGRAM

C. Alur kerja Sistem

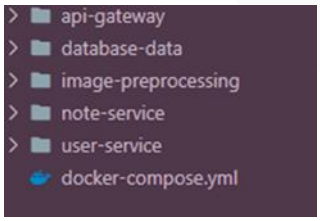


GAMBAR 3.3 ALUR KERJA SISTEM

Pada Gambar 3.3 menunjukkan alur kerja sistem dimana pada proses diatas dimulai dengan langkah pertama pengguna melakukan login atau registrasi, namun apabila belum terdaftar pengguna akan diminta untuk melakukan registrasi terlebih dahulu, registrasi mencakup dalam pembuatan akun. Jika pengguna sudah registrasi, pengguna akan diminta untuk memasukan *username* dan *password*, jika *login* berhasil pengguna akan diarahkan ke *dashboard*, jika gagal pengguna diarahkan untuk mencoba lagi, dari *dashboard* pengguna dapat memasukan gambar untuk diproses menjadi teks, jika gambar tidak bisa diproses maka pengguna akan diminta untuk mencoba lagi, jika gambar berhasil diproses menjadi teks maka hasilnya akan disimpan dan apabila perlu hasilnya bisa dikonversi menjadi *file* PDF.

D. Koding dan Pengembangan

Tahap penulisan kode dan pengembangan, pada tahap ini *web*site *text scanner* berbasis *microservices* mulai diimplementasikan, Proses pengkodean dimulai dengan pembuatan *user service* terlebih dahulu lalu dilanjutkan dengan *note service*, *image preprocessing*, *Api gateway* dan inisiasi *backend*, yang menghasilkan kerangka kerja atau struktur direktori seperti yang ditunjukkan pada Gambar 3.4



GAMBAR 3.4 STRUKTUR DIREKTORI *BACKEND*

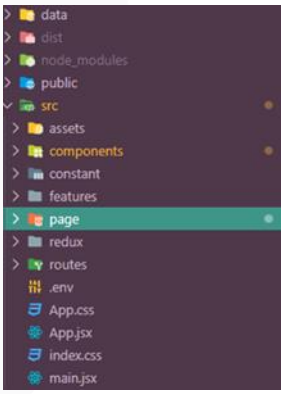
salah satu contoh kodingan yaitu pada *folder* "api-gatway", didalam *folder* tersebut terdapat *file* 'index.js' yang berfungsi sebagai titik akses tunggal ke beberapa layanan lainnya. Kode sumber dari dokumen 'index.js' dapat dilihat pada Gambar 3.5 berikut.

GAMBAR 3.5 *SOURCE CODE INDEX.JS API GATEWAY*

```
const express = require("express");
const cors = require("cors");
const { createProxyMiddleware } = require("http-proxy-middleware");
const app = express();
const port = process.env.PORT;
app.use(cors());
app.use("/user", createProxyMiddleware({ target: "http://user-service:8081", changeOrigin: true }));
app.use("/note", createProxyMiddleware({ target: "http://note-service:8082", changeOrigin: true }));
app.use("/preprocess", createProxyMiddleware({ target: "http://image-preprocessing:8084", changeOrigin: true }));
app.listen(port, () => {
  console.log(`API Gateway is running on port ${port}`);
});
```

Pada *source code* Gambar 5 Kode ini mendefinisikan sebuah *API gateway* menggunakan Node.js dan Express.js. *API Gateway* ini berfungsi sebagai titik akses tunggal ke beberapa layanan lainnya. Menggunakan *middleware proxy* HTTP, permintaan ke "/user", "/note", dan "/preprocess" diteruskan ke layanan yang sesuai. Layanan ini berjalan pada *port* yang ditentukan dalam variabel lingkungan *PORT*. CORS diaktifkan untuk mengizinkan permintaan lintas asal.

Setelah selesai membangun server *API* atau bagian *backend*, langkah berikutnya dalam eksekusi kode adalah mengembangkan bagian *frontend*. Anda dapat merujuk pada Gambar 6 berikut untuk melihat struktur *folder* atau kerangka kerja yang digunakan.



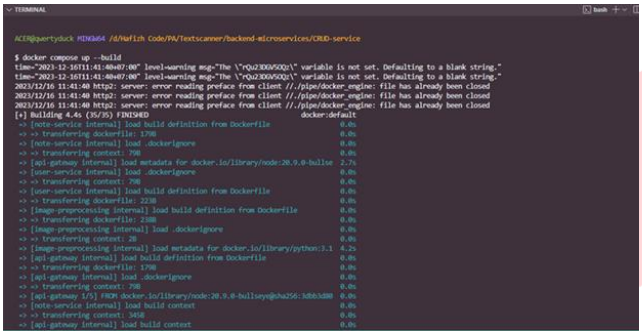
GAMBAR 3.6 STRUKTUR DIREKTORI FRONTEND

Gambar 6 menunjukkan struktur *file* dan *folder* pada kerangka kerja *frontend*. Mirip dengan sisi *backend*, *frontend* juga memiliki beberapa *file* dan *folder* utama serta konfigurasi. Untuk mengakses *API* yang telah dibuat di sisi *backend*, *frontend* memiliki *folder* redux yang berfungsi untuk mengirimkan *request* ke *endpoint* di sisi *backend*.. Redux sendiri adalah sebuah perpustakaan JavaScript sumber terbuka yang digunakan untuk mengatur *state* dalam aplikasi *web*. Redux beroperasi dengan membuat sebuah *store* yang berisi semua *state* dalam aplikasi *web*. Saat ada perubahan *state*, Redux akan membuat sebuah aksi yang akan diproses oleh *reducer* sehingga *store* akan berubah sesuai dengan perubahan *state*.

Selanjutnya yang terakhir, men-*deploy* seluruh komponen *web*site kedalam docker. Proses ini melibatkan pembuatan

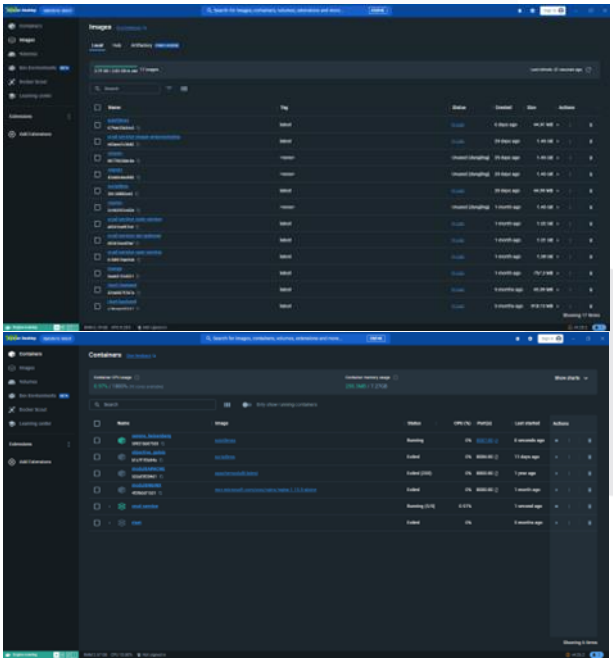
Dockerfile dan Docker Compose yang mendefinisikan lingkungan *runtime* aplikasi. Setelah Dockerfile dan Docker Compose dibuat, kita dapat menjalankan perintah ‘docker composer up --build’ untuk membuat *image* docker bagian *backend* dan perintah ‘docker build --no-cache -t scriptlens .’ Untuk membuat *image* docker bagian *frontend*, *command* ini dapat dilihat pada Gambar 3.7 dan Gambar 3.8 berikut.

GAMBAR 3.7 MEMBUAT *IMAGE* DOCKER UNTUK FRONTEND



GAMBAR 3.8 MEMBUAT *IMAGE* DOCKER UNTUK *BACKEND*

Dapat dilihat pada gambar 3.7 dan gambar 3.8. Sistem sedang membuat *image* docker untuk setiap *service* pada *backend* dan seluruh komponen serta dependensi pada *frontend*, dapat dilihat hasil dari pembuatan *image* docker pada Gambar 3.9.

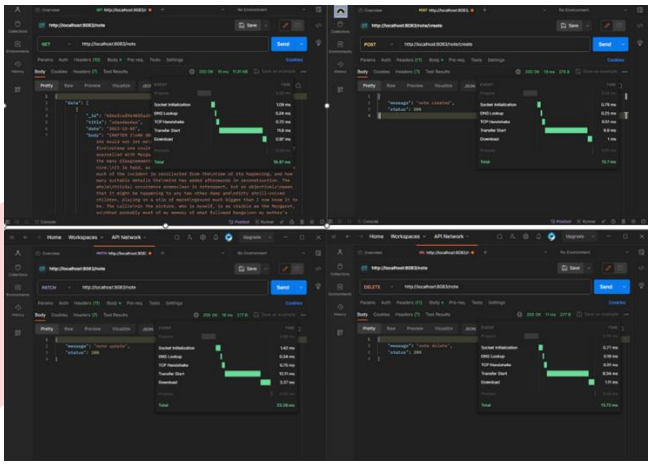


GAMBAR 3.9 DOCKER *IMAGE* DAN DOCKER CONTAINER.

III. HASIL PENGUJIAN DAN ANALISIS

A. Pengujian dan Anallisis Respones Times

Pengujian ini bertujuan utuk meneliti respon waktu setiap methode pada *backend* yang menggunakan arsitektur *microservices*. Pengujian ini menggunakan Postman, Pengujian setiap *methode* dapat dilihat pada Gambar 4.1 dan Tabel 4.1.



GAMBAR 4.1 PERFORMA GET, CREATE, UPDATE, DELETE

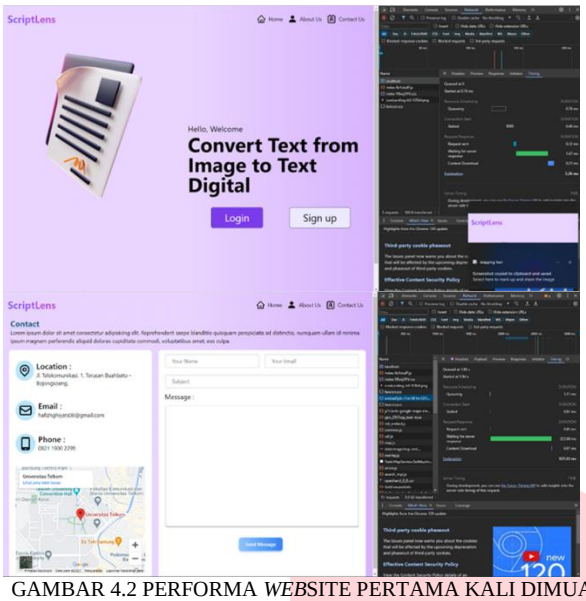
Methode	Total *respones time*
Get	18,47 ms
Post	15,7 ms
Patch	23,8 ms
Delete	13,72 ms

TABEL 4.1 HASIL PENGUJIAN RESPONES TIME

Gambar 10 dan Tabel 1 menunjukkan catatan waktu performa operasi CRUD data melalui API, yang diuji menggunakan aplikasi Postman. Waktu yang dibutuhkan untuk operasi create data adalah 15,7 ms, read data 18,47 ms, update data 23,8 ms, dan delete data 13,72 ms. Dari hasil pengujian ini, dapat disimpulkan bahwa kecepatan yang dihasilkan dari implementasi arsitektur microservice sangat cepat, tidak sampai 1 detik, jauh lebih cepat dibandingkan waktu pemuatan web yang dianggap tidak normal oleh Kinsta, yaitu di atas 4 detik[9]. Meskipun ini hanya sebagian dari proses pemuatan data website, namun tetap berpengaruh signifikan terhadap durasi dan kecepatan pemuatan website secara keseluruhan.

B. Pengujian dan Analisis Performa *Web*site

Untuk mendapatkan pemahaman yang mendalam tentang kinerja *website*, penulis melakukan inspeksi menggunakan *developer tools* yang ada di *browser*, khususnya pada *tab* performa dan *network*. Inspeksi ini difokuskan pada transaksi data CRUD pada *website scanner* teks dan SPA (*Single Page Application*) saat terjadi perpindahan halaman, berdasarkan permintaan dan *respons* yang dikirimkan ke server. Berikut adalah hasil dari evaluasi performa tersebut:

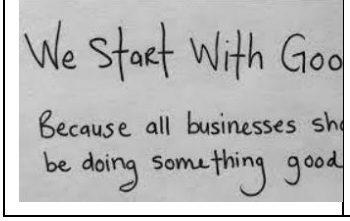


GAMBAR 4.2 PERFORMA *WEB*SITE PERTAMA KALI DIMUAT DAN PERPINDAHAN HALALAMAN BERBASIS SPA

Gambar 4.2 menggambarkan perbedaan yang terjadi saat website pertama kali dimuat dengan semua elemen yang ada, dibandingkan dengan perpindahan halaman dalam website berbasis SPA. Pada saat pertama kali dimuat, website menghasilkan 5 permintaan ke server dengan waktu pemuatan 3,26 ms untuk mendapatkan semua sumber daya dan data yang diperlukan. Namun, ketika terjadi perpindahan dari halaman Home ke halaman Contact, tidak ada pemuatan ulang secara keseluruhan. Sebaliknya, hanya data yang dibutuhkan saja yang dimuat, seperti yang ditunjukkan pada Gambar 4.2. Dalam hal ini, data yang dibutuhkan adalah data halaman Contact. Akibatnya, jumlah permintaan data ke server meningkat menjadi 46 permintaan, sehingga totalnya menjadi 51 permintaan. Namun, pada SPA, hanya 46 permintaan tersebut yang dimuat ulang, sementara 5 lainnya sudah dimuat sejak awal. Meski jumlah permintaan yang dimuat setelah perpindahan halaman cukup banyak, waktu pemuatan hanya 327,33 ms. Ini masih sangat cepat, mengingat halaman yang dimuat terdiri dari 46 permintaan.

C. Pengujian *Image Pre-processing*

Untuk menguji kinerja dari *image pre-processing* penulis melakukan pengujian terhadap gambar yang akan diinputkan yaitu satu gambar tulisan tangan dan gambar yang terdapat pada dokumen *hardfile*. Hasil pengujian dapat dilihat pada tabel 4.2.

Sebelum dilakukan *image* pre-processing	sesudah dilakukan *image* pre-processing
	

TABEL 4.2 PENGUJIAN *IMAGE* PRE-PROCESSING

Dari tabel 4.2 pengujian *image pre-processing*, proses *pre-processing* telah meningkatkan kualitas dan kejelasan teks dalam gambar. Teks menjadi lebih mudah dibaca dan jelas setelah dilakukan *image pre-processing*. Sebagai contoh, frase bahasa Inggris di bagian atas "We Start With Good Because all businesses should be doing something good." tampak lebih jelas dan mudah dibaca setelah pre-processing. Demikian juga dengan beberapa baris teks di bawahnya. Jadi, secara umum, *image pre-processing* telah berhasil meningkatkan legibilitas teks dalam gambar

D. Pengujian dan Analisis OCR

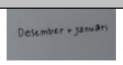
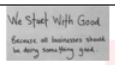
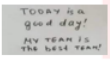
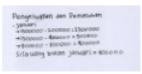
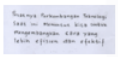
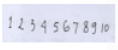
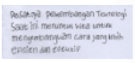
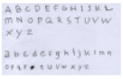
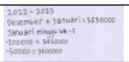
image Pada pengukuran tingkat akurasi OCR dan *execution time*, penulis melakukan pengujian dengan cara mencoba fitur OCR pada *web*site dengan sampel 10 tulisan tangan, 10 dokumen *hardfile* dan 5 dokumen dengan *background* hitam. Berikut rumus untuk mencari presentase *error*.

$$Percent\ error\ (\%) = \left|\frac{a-b}{a}\right| X\ 100\%$$

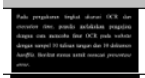
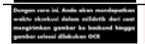
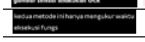
Keterangan : a: jumlah karakter ; b: jumlah karakter tidak *error*.

Pengujian ke	OCR		Ukuran *file*	execution time	Selisih	Error Rate (%)	Akurasi (100%-Error rate)
	Jumlah karakter	Jumlah karakter tidak error					
1	2106	2095	100KB	5,8 s	11	0,5	99,5%
2	1844	1843	423KB	3,5 s	1	0,05	99,95%
3	1289	1284	73,9KB	4,5 s	5	0,3	99,7%
4	1590	1585	78KB	4,6 s	5	0,3	99,7%
5	1804	1800	939KB	8,6 s	4	0,2	99,8%
6	1829	1821	893KB	5,1 s	8	0,4	99,6%
7	2022	2009	562KB	6.4 s	13	0.6	99,2%
8	1858	1851	783KB	6,04s	7	0,3	99,7%
9	2376	2369	857KB	7,5 s	7	0,2	99,8%
10	1744	1743	884KB	5.7 ss	1	0,05	99,95%
Rata-rata					**6**	**0,26**	**99,69%**

TABEL 4.3 PENGUJIAN TINGKAT AKURASI OCR DAN EXECUTION TIME DOKUMEN *HARDFILE*

Pengujian	OCR		Ukuran *file*	execution time	Selisih	Error Rate (%)	Akurasi (100%-Error rate)
	Jumlah karakter	Jumlah karakter tidak error					
	19	19	17,6KB	0,6 s	0	0	100%
	73	71	56,4KB	0,6 s	2	2,7	97,3,%
	47	46	41KB	0,7 s	1	2,1	97,9%
	146	143	32,5KB	0,9 s	3	2,05	97,95%
	116	107	1,44MB	1,5 s	9	7,7	92,3%
	11	10	58,3KB	0,6 s	1	9,9	90,1%
	110	17	202KB	1,5 s	17	15,45	84,55%
	11	7	62,8KB	0,7 s	4	36,3	63,7%
	53	50	254KB	1,5 s	3	6	94%
	98	93	205KB	1,4 s	5	5,1	94,9%
Rata-rata					**4,5**	**8,73**	**91,27%**

TABEL 4.4 PENGUJIAN TINGKAT AKURASI OCR DAN EXECUTION TIME DOKUMEN TULISAN TANGAN

Pengujian	OCR		Ukuran file	execution time	Selisih	Error Rate (%)	Akurasi (100%-Error rate)
	Jumlah karakter	Jumlah karakter tidak error					
	231	231	67KB	2,4 s	0	0	100%
	6	5	35KB	0,6 s	1	16,6	83,4%
	19	19	73,9KB	4,5 s	0	0	100%
	146	145	48KB	1,7 s	1	0,6	99,4%
	53	53	20KB	0,8s	0	0	100%
Rata-rata					**0.4**	**3,44**	**96,56%**

TABEL 4.5 PENGUJIAN GAMBAR BACKGROUND HITAM

Berdasarkan hasil pengujian OCR, dapat kita ketahui bahwa tingkat akurasi pembacaan karakter yang ditunjukan pada tabel diatas terutama tabel 4.1 yang mana merupakan perbandingan antara hitungan jumlah karakter dan jumlah karakter yang tidak error dengan hitungan manual menghasilkan presentase yang sangat baik yaitu dengan rata-rata presentase 99,69% dan untuk kecepatan eksekusi gambar relatif cepat mengingat jumlah karakter terbanyak pada gambar yang diuji adalah 2376 karakter dengan kecepatan eksekusinaya adalah 7,6 detik.

Namun dapat dilihat pada tabel 4.2 untuk pembacaan karakter tulisan tangan walaupun rata-rata presentase menghasilkan 91,27% diperlukan lagi pemodelan OCR dan proses *image* preprocessing yang lebih canggih, dikarenakan ada beberapa pengujian yang mendapati pembacaan OCR yang akurasinya kurang baik. Selain itu, menurut Docsumo juga menyebutkan bahwa untuk kasus yang kompleks yang melibatkan teks tulisan tangan dengan konten yang sangat heterogen dan di luar kosakata, nilai CER (*Character Error Rate*) sebesar sekitar 20% (yaitu akurasi 80%) dapat dianggap memuaskan[9]

Berdasarkan tabel 4.5 pengujian OCR pada gambar dengan latar belakang hitam, dapat disimpulkan bahwa OCR memiliki akurasi yang cukup tinggi dalam mengenali teks pada gambar dengan latar belakang hitam. Sebagian besar hasil pengujian menunjukkan tingkat kesalahan yang rendah dan akurasi di atas 83%. Tabel tersebut menampilkan hasil pengujian pada beberapa gambar spesifik, seperti "RESUME" dan "SORRY WE'RE CLOSED", dengan akurasi pengenalan karakter yang bervariasi dari 83,4% hingga 100%. Rata-rata, tingkat kesalahan pengujian adalah 3,44% dan akurasi adalah 96,56%. Ini menunjukkan bahwa meskipun ada beberapa kasus di mana OCR tidak sempurna, secara umum, performanya cukup baik.

Melihat hasil tersebut, jelas bahwa ada ruang untuk peningkatan, terutama dalam hal pembacaan karakter tulisan tangan. Untuk mencapai ini, beberapa strategi dapat diadopsi. Misalnya, penggunaan teknik pemrosesan gambar yang lebih canggih dapat membantu dalam meningkatkan kualitas gambar sebelum diteruskan ke mesin OCR.

IV. KESIMPULAN

Berdasarkan analisis dari hasil pengujian respones times dari *request methode* (*get, create, update,* dan *delete*) arsitektur *microservices* berbasis docker, kecepatan yang didapat ini terbilang sangat cepat bahkan tidak sampai menghabiskan waktu 1 detik meski pengujiannya hanya pada bagian *backend* dengan menggunakan postman. Performa *web*site secara menyeluruh yang telah dilakukan pada *web*site *scanner* teks dan SPA saat berpindah antara halaman. waktu yang dibutuhkan untuk memuat semua itu adalah 327,33 ms. Hal ini masih terbilang sangat cepat mengingat pemuatan halaman yang cukup banyak setelah perpindahan halaman. analisis dari hasil pengujian OCR dapat disimpulkan bahwa teknologi OCR telah menunjukkan kinerja yang sangat baik dalam pengujian, dengan tingkat akurasi rata-rata 99,69% dan kecepatan eksekusi yang cepat. Namun, ada tantangan dalam membaca karakter tulisan tangan, di mana tingkat akurasi rata-rata adalah 91,27% dan itupun terdapat salah satu pengujian yang mendapati presentase akurasi dibawah 80%. Meskipun demikian, tingkat akurasi rata – rata ini masih dianggap memuaskan untuk kasus yang kompleks.

Pengembangan lebih lanjut guna meningkatkan performa *web*site dan OCR tersebut dapat menggunakan arsitektur pengembangan *web*site yang lebih baik seperti *microfrontend* dan Untuk meningkatkan akurasi lebih lanjut, peningkatan pada *image preprocessing* diperlukan, serta penggunaan teknik pemrosesan gambar yang lebih canggih. Kesimpulannya, meskipun teknologi OCR telah menunjukkan hasil yang mengesankan, masih ada ruang untuk peningkatan, terutama dalam hal pembacaan karakter tulisan tangan

REFERENSI

[1] F. O. Rahmalisty, “Perancangan Language Translator *Image* To Text Menggunakan Metode *Optical Character Recognition* Berbasis Pengolahan Citra Language Translator *Image* To Text Design Using Optical Character Tecognition Method Based On *Image* Processing.”2023

[2] H. Nindya Murwato, S. Aulia, and A. Novianti, “PERANCANGAN TRANSLATOR *IMAGE* TO TEXT DENGAN MENGGUNAKAN METODE *OPTICAL CHARACTER RECOGNITION* BERBASIS MATLAB *IMAGE* TO TEXT TRANSLATOR DESIGN USING MATLAB BASED *OPTICAL CHARACTER RECOGNITION* METHOD.” 2020

[3] H. D. Aldi , “Pemanfaatan Teknologi *Optical Character Recognition* pada Pembacaan Kartu Tanda Penduduk Artikel Ilmiah,” 2019.

[4] J. Memon, M. Sami, R. A. Khan, and M. Uddin, “Handwritten *Optical Character Recognition* (OCR): A Comprehensive Systematic Literature Review (SLR),” *IEEE Access*, vol. 8. Institute of Electrical and Electronics Engineers Inc., pp. 142642–142668, 2020. doi: 10.1109/ACCESS.2020.3012542.

[5] Setiawan A, “Implementasi *Optical Character Recognition* (OCR) pada Mesin Penerjemah Bahasa Indonesia ke Bahasa Inggris,” *Jurnal Sistem dan Teknologi Informasi (JUSTIN)*, vol. 5, pp. 135–141, 2017.

[6] A. N. Rahmawati, S. A. Wibowo, and U. Sunarya, “ANALISIS SISTEM *OPTICAL CHARACTER RECOGNITION* (OCR) PADA DOKUMEN DIGITAL MENGGUNAKAN METODE TESSERACT PERFOMANCE ANALYSIS OF *OPTICAL CHARACTER RECOGNITION* (OCR) SYSTEM ON DIGITAL DOCUMENTS USING TESSERACT METHOD.” 2021

[7] J. Stastny, J. Šastný, and M. Minaík, “A Brief Introduction to *Image* Pre-Processing for Object Recognition,” 2007. [Online]. Available: http://www.fme.vutbr.cz/

[8] S. Supriadi, “Simple Handwriting Calculator Application Using *Optical Character Recognition* (OCR),” *Buletin Ilmiah Sarjana Teknik Elektro*, vol. 2, no. 3, p. 163, Jan. 2021, doi: 10.12928/biste.v2i3.3348.

[9] Nasution, “IMPLEMENTASI MONGO DB, EXPRESS JS, REACT JS DAN NODE JS (MERN) PADA PENGEMBANGAN APLIKASI FORMULIR, KUIS, DAN SURVEI ONLINE” 2022.

[10] F. Rifandi, Tri Viqi Adriansyah, and Rina Kurniawati, “*Website* Gallery Development Using Tailwind CSS Framework,” *Jurnal E-Komtek (Elektro-Komputer-Teknik)*, vol. 6, no. 2, pp. 205–214, Dec. 2022, doi: 10.37339/e-komtek.v6i2.937.

[11] M. Fihri, R. M. Negara, and D. D. Sanjoyo, “IMPLEMENTASI & ANALISIS PERFORMANSI LAYANAN *WEB* PADA PLATFORM BERBASIS DOCKER IMPLEMENTATION & ANALYSIS OF *WEB* SERVICE PERFORMANCE BASED ON DOCKER PLATFORM.”

[12] T. A. Diajukan, M. Salah, S. Persyaratan, and M. Derajat, “IMPLEMENTASI ARSITEKTUR MICROSERVICE PADA APLIKASI *WEB* PENGAJARAN AGAMA ISLAM HOME PESANTREN,” 2020.