Implementasi Single Page Application Sebagai Sistem Pembayaran Rumah Kos Dengan Vue.Js

Jerry Yan Krismanto[1], Agus Prihanto[2]

[1,2] Jurusan Teknik Informatika, Fakultas Teknik, Universitas Negeri Surabaya
[1]jerry.20028@umhs.unesa.ac.id
[2]agusprihanto@unesa.ac.id

***Abstrak*— Pada zaman ini, bermacam-macam teknologi terus ada dan diperbaharui hari demi hari. Salah satunya adalah perkembangan *website*, perkembangan teknologi dalam pembangunan *website* menjadi faktor yang sangat mendukung dunia bisnis seperti sistem pembayaran rumah kos. *Single Page Application* adalah salah satu hasil perkembangan teknologi *website* yang memungkinkan seluruh konten *website* ditampilkan dalam satu halaman. Tujuan dari penelitian ini adalah menjelaskan dan mengimplementasikan *Single Page Application* pada sistem pembayaran rumah kos dengan basis *website*, menguji hasil implementasi tersebut, dan menganalisa hasil dari pengujian tersebut.**

Hasil pengujian menunjukkan bahwa *website* memiliki waktu respon yang cepat terhadap *input* dari pengguna dengan waktu ±0,02 detik. Hasil analisa yang didapat adalah bahwa *website SPA* memiliki waktu respon yang cepat karena semua komponen sudah dimuat di awal, dan oleh karena itu juga waktu load awal *website SPA* lebih lambat daripada waktu responnya.

***Kata Kunci*— *Single Page Application*, Performa, *Website*, Pembayaran Kos, *Frontend*.**

I. PENDAHULUAN

Dalam era digital ini, dimana hampir semua kebutuhan dan informasi dapat diakses menggunakan gawai, adanya sistem yang mudah digunakan akan sangat sangat membantu bisnis manapun, tidak terkecuali rumah kos [2]. Namun, akses informasi untuk penghuni seringkali terbatas dengan penghuni harus menemui pemilik kos untuk mengakses suatu informasi.

Pada saat ini memang sudah ada beberapa sistem yang dibuat menggunakan website, namun teknologi yang digunakan masih bersifat dasar, yaitu hanya menggunakan *HTML*, *CSS*, *Javascript* atau mungkin *PHP* [1], [5], [6]. Seperti pada penelitian berjudul "Management System For an Apartment" buatan Manoj Kumar Athota. Penelitian ini bertujuan untuk merancang sebuah *platform online* berbasis *web* untuk penyewa dan pemilik apartemen atau rumah sewa yang memungkinkan kedua belah pihak saling mendapat kemudahan dalam melakukan transaksi dan interaksi [4], [3]. *Website* yang dirancang memungkinkan calon penyewa untuk memilih dan menyewa ruangan dengan mudah. Pembuatan website dalam penelitian ini menggunakan *Visual Basic* sebagai *IDE* dan *HTML*, *CSS*, *jQuery*, dan *Javascript* sebagai bagian dari *tech stack* nya. sistem yang dibuat menggunakan teknologi tersebut memang sudah dapat menyelesaikan masalah akses informasi tersebut. Namun saat ini sebagian besar *website* dibuat menggunakan *framework* atau *library* [8], salah satu yang populer adalah *Vue.js*, *library* ini menghasilkan *website* dengan format *SPA* (*Single Page Application*) [16].

Dengan menggunakan *Vue.js* kita dapat menerapkan *Reactive Rendering* pada *website* yang kita buat, dengan demikian diharapkan sistem yang dibangun akan memiliki performa yang sesuai standar.

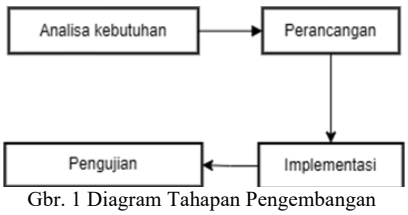
Namun, standar performa *website* yang bagus tentu berbeda-beda untuk masing-masing individu dan kelompok. Oleh karena itu, diperlukan sebuah standar yang dapat mengukur *parameter* yang dapat digunakan untuk pengujian performa.

Core Web Vitals adalah *subset* metrik dari *Web Vitals* yang dapat diterapkan pada semua halaman *web*. Masing-masing *Core Web Vitals* mewakili aspek yang berbeda dari pengalaman pengguna yang dapat digunakan untuk mengukur performa dan aksesibilitas. Ada banyak alat yang menerapkan *Core Web Vitals* sebagai metrik pengukurannya. Salah satunya adalah *Google Lighthouse* [9]. *Google Lighthouse* adalah alat yang dapat diaplikasikan pada halaman *web* apapun, yang kemudian akan memberi hasil berupa pengukuran performa dan aksesibilitas *web*. *Core Web Vitals* dan *Google Lighthouse* memiliki pengelola yang sama yaitu *Google*. Artinya standar yang digunakan pada pengukuran parameter dari masing-masing metrik adalah besutan dari *Google* juga.

Pada penelitian ini, peneliti akan melakukan implementasi penggunaan *Vue.js* untuk mengaplikasikan *SPA* pada sistem pembayaran rumah kos. Peneliti kemudian akan menguji performa dari *website* menggunakan *Google Lighthouse*.

II. METODOLOGI PENELITIAN

Penerapan tahap pengembangan ini dapat menjadi pendekatan yang efektif untuk pengembangan *website*. Hal ini dapat membantu memastikan bahwa semua aspek penelitian terdefinisi dengan jelas dan bahwa sistem memenuhi persyaratan pengguna.



Gbr. 1 Diagram Tahapan Pengembangan

A. *Analisis Kebutuhan*

Untuk merancang sebuah sistem yang optimal dan untuk memastikan *website* memiliki performa yang baik. Dibutuhkan alat-alat yang dapat mendukung pembuatan *website* [15]. Tabel I Mencantumkan hasil analisis kebuthan dari sistem yang akan dibuat.

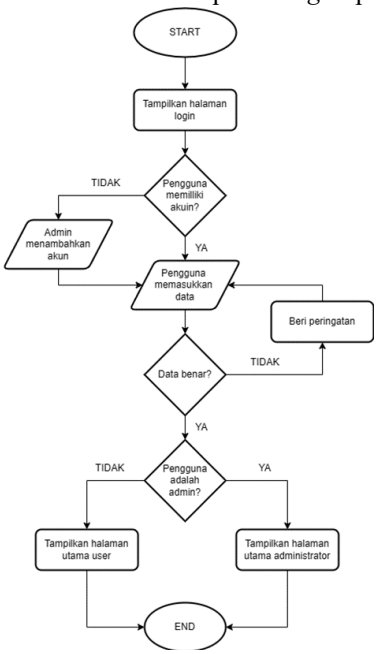
TABEL I ANALISIS KEBUTUHAN

Kebutuhan Fungsional	Kebutuhan Non Fungsional
Aplikasi harus dapat menampilkan data yang sesuai dengan kebutuhan pengguna	Aplikasi harus dapat me*render* konten halaman tanpa harus memuat ulang keseluruhan halaman
Aplikasi harus dapat menjalankan perintah yang sesuai dengan input pengguna, seperti pembayaran atau pengecekan status.	Aplikasi harus dapat bereaksi terhadap perubahan konten dalam halaman secara otomatis dan tanpa muat ulang.

B. Perancangan Sistem

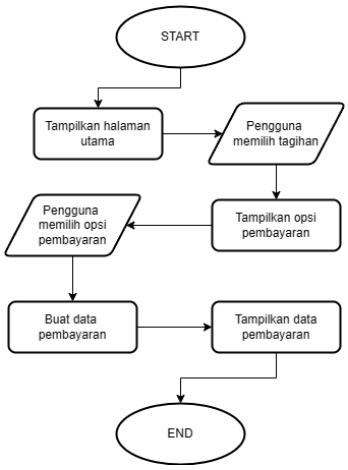
1) Flowchart

Flowchart disusun sebagai visualisasi alur kerja dan dapat digunakan untuk membuat pandangan yang terstruktur tentang bagaimana sebuah aplikasi berfungsi, dan berperan penting dalam memfasilitasi pengembangan, analisis, dan pemeliharaan aplikasi dengan lebih efisien. Berikut adalah *flowchart* yang memvisualisasikan proses *log in* penghuni kos.



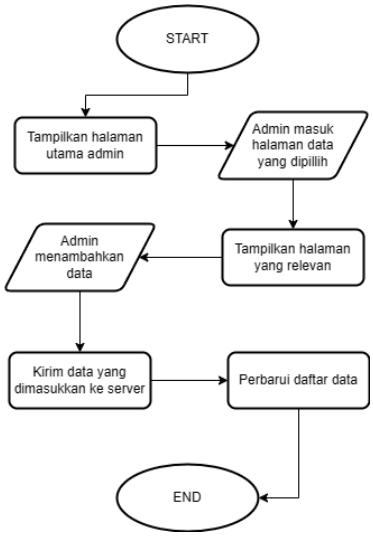
Gbr. 2 *Flowchart* Login Pengguna

Gbr. 2 adalah *flowchart* yang memvisualisasikan proses *log in* pengguna



Gbr. 3 *Flowchart* Pembayaran Tagihan

Gbr. 3 adalah *flowchart* yang memvisualisasikan proses pembayaran tagihan penghuni kos. *flowchart* ini mengasumsikan bahwa pengguna sudah memiliki akun dan merupakan penghuni kos.



Gbr. 4 *Flowchart* Menambah Data

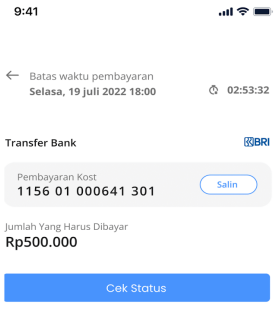
Gbr. 4 adalah *flowchart* yang menjelaskan alur menambahkan data bagi pengguna dengan *role administrator*.

2) UI Design

Tahap ini merupakan proses pembuatan *mockup* desain antarmuka yang nantinya akan menjadi referensi pembuatan antarmuka aplikasi. Pembuatan *mockup* dilakukan menggunakan *figma*.

Gbr. 5 Desain Halaman Utama Penghuni

Gbr. 5 menunjukkan desain dari halaman utama penghuni. Halaman ini memuat sebagian besar informasi yang dibutuhkan pengguna.



Gbr. 6 Desain Halaman Pembayaran

Gbr. 6 menunjukkan halaman detail pembayaran, halaman tersebut akan menampilkan informasi yang dibutuhkan pengguna untuk menyelesaikan pembayaran.

C. *Implementasi*

Tahap implementasi adalah tahap dimana semua rancangan yang telah dibuat direalisasikan.

D. *Pengujian*

Pengujian dilakukan untuk mengidentifikasi aspek-aspek dalam performa *website*, termasuk kecepatan muat halaman yang terdiri dari beberapa parameter, Seperti *total blocking time,* yang merupakan waktu yang diperlukan oleh *website* dari pertama kali komponen ditampilkan sampai halaman menjadi interaktif sepenuhnya, dan *interaction to next paint* yang merupakan waktu yang diperlukan *website* untuk merespon *input* dari *user*. Proses pengujian melibatkan menjalankan *Google Lighthouse* pada halaman *website* dengan variasi konten dan fungsionalitas. Pengukuran dilakukan terhadap berbagai metrik performa yang disediakan oleh alat tersebut.

1) Parameter dan Metrik Pengujian

Untuk memastikan hasil performa dari *Single Page Application*, maka diperlukan parameter pengujian untuk mengukur aspek-aspek performa dari sistem. Setiap parameter akan menampilkan hasil atau metrik yang sudah ditetapkan sebagai berikut:

TABEL II PENGUJIAN PERFORMA SPA MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor
1	*First Contentful Paint (FCP)*	
2	*Total Blocking Time (TBT)*	
3	*Speed Index (SI)*	
4	*Largest Contentful Paint (LCP)*	
5	*Cumulative Layout Shift (CLS)*	

TABEL III PENGUJIAN PERFORMA SPA MENGGUNAKAN GOOGLE LIGHTHOUSE (TIMESPAN)

No	Parameter	Skor
1	*Total Blocking Time (TBT)*	
2	*Cumulative Layout Shift (CLS)*	
3	*Interaction to Next Paint (INP)*	

2) Keterangan Parameter Pengujian

Untuk mempermudah pemahaman mengenai metrik dari setiap parameter. Peneliti menyajikan keterangan dalam tabel berikut:

TABEL IV KETERANGAN PARAMETER PENGUJIAN

Parameter	Keterangan
First Contentful Paint (FCP)	Waktu yang dibutuhkan untuk me*render* elemen pertama pada *website*.
Total Blocking Time (TBI)	Total waktu yang dibutuhkan dari *FCP* hingga *website* interaktif sepenuhnya
Speed Index (SI)	Seberapa cepat sebagian besar elemen dan konten memenuhi halaman
Largest Contentful Paint (LCP)	Waktu yang dibutuhkan untuk merender elemen yang paling besar/berat dalam *website*
Cumulative Layout Shift (CLS)	Total pergerakan elemen dan konten pada *viewport* saat proses *render* berlangsung
Interaction to Next Paint (INP)	Mengukur seberapa cepat *website* merespon *input* dari pengguna

3) Interpretasi Skor Pengujian

Skor pengujian yang akan didapatkan akan mendapatkan interpretasinya sendiri sesuai dengan standar yang digunakan oleh *Google Lighthouse* yaitu *Core Web Vitals*. Berikut masing-masing standar dari parameter yang ditetapkan *Core Web Vitals*. Standar ini didapat dari laman resmi *Core Web Vitals* dan *Chrome for Developers*.

a. First Contentful Paint

Gbr. 7 Standar parameter *FCP*

Gambar diatas menunjukkan bahwa nilai *FCP* yang baik adalah dibawah 1,8 detik, dan nilai *FCP* yang buruk adalah 3,0 detik keatas. [9]

b. Total Blocking Time

TBT time (in milliseconds)	Color-coding
0–200	Green (fast)
200-600	Orange (moderate)
Over 600	Red (slow)

Gbr. 8 Standar Parameter *TBT*

Gambar diatas menunjukkan bahwa nilai *TBT* yang baik adalah dibawah 0,2 detik, dan nilai *TBT* yang buruk adalah 0,6 detik keatas. [10]

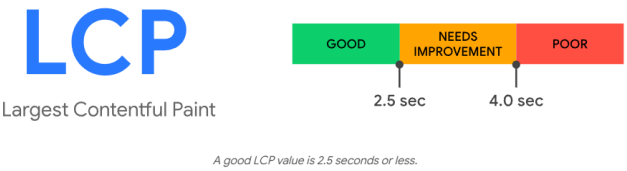
c. Speed Index

Speed Index (in seconds)	Color-coding
0–3.4	Green (fast)
3.4–5.8	Orange (moderate)
Over 5.8	Red (slow)

Gbr. 9 Standar Pengujian *SI*

Gambar diatas menunjukkan bahwa nilai *SI* yang baik adalah dibawah 3,4 detik, dan nilai *SI* yang buruk adalah 5,8 detik keatas. [11]

d. Largest Contentful Paint



Gbr. 10 Standar Pengujian *LCP*

Gambar diatas menunjukkan bahwa nilai *LCP* yang baik adalah dibawah 2,5 detik, dan nilai *LCP* yang buruk adalah 4,0 detik keatas. [12]

e. Cumulative Layout Shift

Gbr. 11 Standar Pengujian *CLS*

Gambar diatas menunjukkan bahwa nilai *CLS* yang baik adalah dibawah 0,1, dan nilai *CLS* yang buruk adalah 0,25 keatas.[13]

f. Interaction to Next Paint

Gbr. 12 Standar Pengujian *INP*

Gambar diatas menunjukkan bahwa nilai *INP* yang baik adalah dibawah 0,2 detik, dan nilai *INP* yang buruk adalah 0,5 detik keatas. [14]

III. Hasil dan Pembahasan

A. Implementasi Aplikasi

Peneliti mengimplementasikan sistem *frontend* yang dapat menjembatani interaksi pengguna dengan *server*.

1) Inisialisasi Vue.js

Peneliti menggunakan *NPM* untuk menginisialisasi *library Vue.js* sebagai sebuah proyek. Hal ini peneliti lakukan dengan menjalankan perintah berikut.

```
npm create vue@latest
```

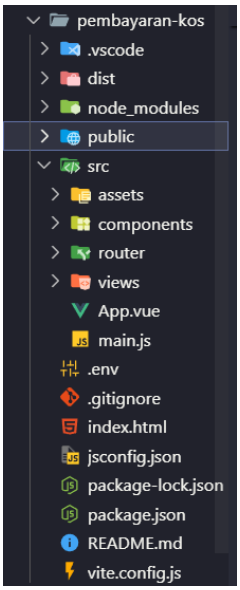
Perintah tersebut akan menambahkan berkas-berkas *library* pengembangan *Vue.js*. Langkah selanjutnya adalah menginstal *dependency* dari *Vue.js*. Hal ini dapat dilakukan menggunakan *NPM* dengan menjalankan perintah berikut.

```
npm install
```

Perintah tersebut akan menginstal seluruh *dependency* yang tertulis pada berkas package.json.

2) Struktur Folder

Setelah proses inisialisasi selesai, akan ada beberapa folder yang sudah secara otomatis terstruktur oleh *Vue.js*, folder-folder ini memiliki nama yang merepresentasikan fungsinya masing-masing.



Gbr. 13 Struktur Folder

Terdapat empat folder utama yang tersimpan dalam folder src. diantaranya assets yang berisi file *CSS* dan Gambar yang dibutuhkan dalam *website*, components yang berisi file komponen individu yang akan di *render* dalam *website*. *router* yang berisi file *Javascript* untuk menangani *routing*, dan views yang berisi komponen view yang akan memuat komponen individu dalam folder components.

3) Menyiapkan variabel .env

Peneliti menyimpan variabel lingkungan dalam file. env, file ini berisi variabel yang tidak akan berubah dan bersifat global, dalam kasus ini variabel tersebut adalah *url API* di sistem *backend*.

```
VITE_API_URL=https://kost-payment-backend-lngftw6qaa-et.a.run.app/api
```

Variabel tersebut akan mempermudah pemanggilan *API* pada komponen yang nantinya dibuat.

4) Menyusun App.vue

Halaman utama dari website berkaitan dengan *routing* pada aplikasi. pertama *Vue.js* akan mengakses file App.vue yang berisi perintah yang menampilkan hasil *render* dari *route* "/" atau yang juga disebut *home*.

```
<script setup>
import { RouterLink, RouterView } from "vue-router";
</script>
<template>
  <RouterView />
</template>
```

5) Membuat router

Router berfungsi untuk mengelola navigasi dalam *SPA* sehingga pengguna dapat mengganti komponen yang ditampilkan sesuai kebutuhan tanpa berpindah halaman. Pada *Vue.js*, *router* yang digunakan dalam pengembangan adalah *Vue Router*.

```
import { createRouter, createWebHistory } from "vue-
router";
import HomeView from "../views/HomeView.vue";


const router = createRouter({
  history: createWebHistory(import.meta.env.BASE_URL
  routes: [
    {
      path: "/",
      name: "home",
      component: HomeView,
      beforeEnter: (to, from, next) => {
        const isLogin =
localStorage.getItem("username", "password");
        if (!isLogin) {
          next("/login");
        } else {
          next();
        }
      },
    },
    {
      path: "/login",
      name: "login",
      component: () =>
import("../views/LoginView.vue"),
    },
. . .
})
```

Router juga dapat berfungsi sebagai kontrol navigasi saat ada pengguna dengan level akses yang berbeda, *router* dapat mencegah pengguna yang belum *login* mengakses halaman utama dan mencegah pengguna biasa mengakses halaman administrator. terdapat beberapa *route* yang dibuat dalam aplikasi ini, diantaranya:

- Home
- Login
- Payment Options
- Payment
- Bills
- Profile
- Admin
- Rooms
- Users

Masing-masing *route* akan merender komponen yang ada sesuai dengan kebutuhannya. Perlu diketahui bahwa satu komponen yang sama dapat digunakan oleh *route* yang berbeda.

6) Membuat Komponen

Komponen merupakan faktor utama dari segala *SPA*. dengan adanya komponen, maka kita dapat menggunakan kode yang sudah dibuat dalam berbagai bagian pada *website*. Hal ini membuat *browser* tidak perlu memuat ulang komponen yang sebelumnya sudah ditampilkan. Ada dua tipe komponen dalam aplikasi ini. pertama adalah komponen individu, dan kedua adalah komponen views komponen *views* akan berisi kode yang menyusun komponen yang bersifat tetap dan unik dalam sebuah *route*, artinya komponen ini tidak akan digunakan atau di render pada route lainnya, komponen route juga memuat beberapa komponen individu yang bisa saja juga digunakan pada komponen lainnya.

```
<template>
  <div>
    <HelloUser                                    />
    <RentMainCard
      :total-bills="totalBills"
      :message="'Total                      Tagihan'"
      @click="pushBills"                          />
    <p>Riwayat                          pembayaran</p>
    <RentCardList
      v-for="bill      in      bills.slice(0,      3)"
      :key="bill.id"
      :month="bill.month"
      :amount="bill.amount"
      :status="bill.status"
      :id="bill.id"
      @push-payment="pushPayment"
      @click="handleCardClick(bill.status,
bill.invoice,              bill.id)"              />
  </div>
</template>
```

Kode diatas menunjukkan potongan kode utama yang menyusun sebuah komponen *view*. Komponen tersebut selain memuat kode komponennya sendiri, juga memanggil komponen individu yang sudah ada. Berbeda dengan komponen individu yang hanya berisi kode untuk komponen itu sendiri dan tidak memanggil komponen lainnya.

```
<template>
  <div class="rent-cards">
    <img src="./icons/blue-calendar.png" alt="" />
    <div>
      <p class="month-text">{{ bill.month }}</p>
      <p class="price-text">{{ bill.amount }}</p>
    </div>
    <div class="button-status">
      <p class="paid" v-if="bill.status ==
'paid'">Terbayar</p>
```

```
      <p class="pending" v-else-if="bill.status ==
'pending'">Pending</p>
      <button v-else
@click="pushPayment">Bayar</button>
    </div>
  </div>
</template>
```

Potongan kode diatas menunjukkan potongan kode utama yang menyusun sebuah komponen individu, isi dari file hanya kode yang menyusun komponen tersebut dan tidak ada kode yang memanggil komponen lainnya.

7) Integrasi API

Supaya komponen yang dibuat dapat menampilkan data yang sesuai, maka diperlukan pengambilan data dari *server*. pengambilan data ini dapat dilakukan melalui *API*, terdapat beberapa metode yang dapat digunakan untuk mengirim permintaan pada *server*, metode yang peneliti gunakan pada proyek ini adalah "*fetch*". Umumnya, kode yang digunakan untuk mengambil data hanya akan digunakan pada komponen *views*, data yang diterima kemudian akan digunakan pada komponen itu sendiri dan dibagikan juga pada komponen individu yang dipanggil.

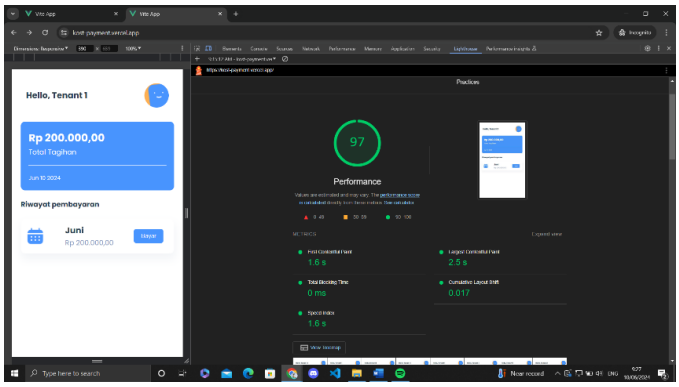
```
const fetchTotalBills = () => {
  const email = localStorage.getItem("email");
  const password = localStorage.getItem("password");
  const url = `${apiUrl}/total-bills`;
  const getOptions = {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      email: email,
      password: password,
      "ngrok-skip-browser-warning": "Yes",
    },
  };

 fetch(url, getOptions)
    .then((response) => response.json())
    .then((data) => {
      console.log(data);
      totalBills.value = data.data.total;
    })
    .catch((error) => console.error("Error:", error));
};
```

Data yang didapat dari pemanggilan tersebut akan ditampilkan pada komponen yang sudah dibuat. Karena komponen *Vue.js* bersifat reaktif, maka segala perubahan data dapat dilacak dan tampilan akan berubah sesuai perubahan data.

B. Pembahasan Hasil

1) Pengujian

Tahap serangkaian pengujian pada *website* yang sudah di *host* di *Vercel*. Pengujian dilakukan pada *browser Google Chrome* mode *incognito* dengan metode *Navigation* dan *Timespan* sebanyak 5 kali pengujian untuk masing-masing metode. Hasil pengujian kemudian akan digunakan untuk mencari nilai *mean* untuk setiap metode pengujian.

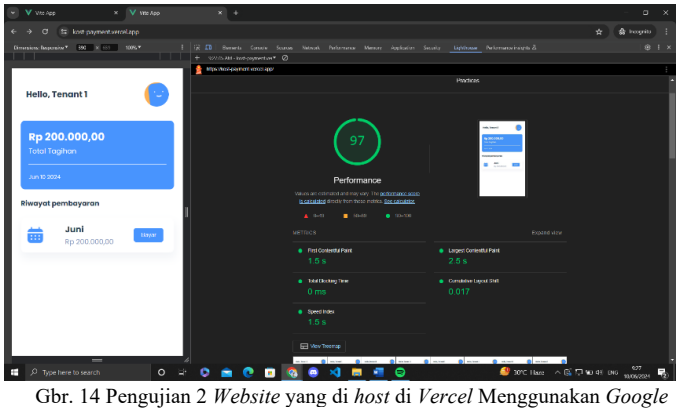


Gbr. 13 Pengujian 1 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Navigation*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL V PENGUJIAN 1 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor
1	*First Contentful Paint (FCP)*	1,6s
2	*Total Blocking Time (TBT)*	0ms
3	*Speed Index (SI)*	1,6s
4	*Largest Contentful Paint (LCP)*	2,5s
5	*Cumulative Layout Shift (CLS)*	0,017

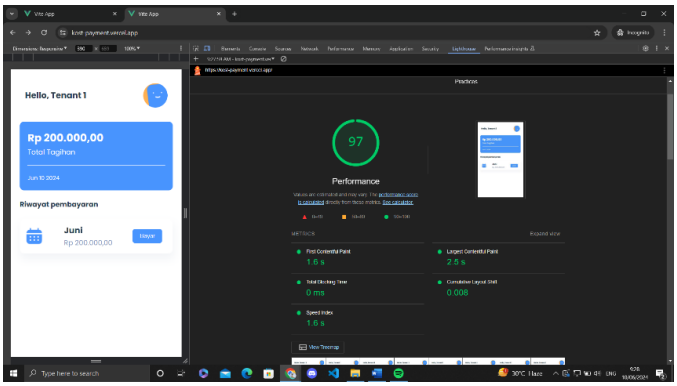


Gbr. 14 Pengujian 2 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Navigation*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL VI PENGUJIAN 2 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor
1	*First Contentful Paint (FCP)*	1,5s
2	*Total Blocking Time (TBT)*	0ms
3	*Speed Index (SI)*	1,5s
4	*Largest Contentful Paint (LCP)*	2,5s
5	*Cumulative Layout Shift (CLS)*	0,017

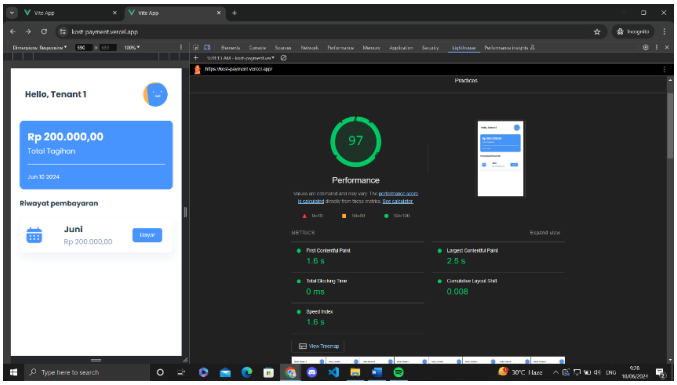


Gbr. 15 Pengujian 3 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Navigation*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL VII PENGUJIAN 3 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor
1	*First Contentful Paint (FCP)*	1,6s
2	*Total Blocking Time (TBT)*	0ms
3	*Speed Index (SI)*	1,6s
4	*Largest Contentful Paint (LCP)*	2,5s
5	*Cumulative Layout Shift (CLS)*	0,008

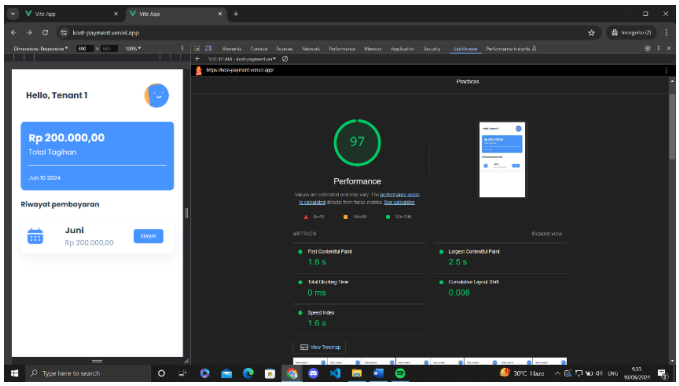


Gbr. 16 Pengujian 4 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Navigation*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL VIII PENGUJIAN 4 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor
1	*First Contentful Paint (FCP)*	1,6s
2	*Total Blocking Time (TBT)*	0ms
3	*Speed Index (SI)*	1,6s
4	*Largest Contentful Paint (LCP)*	2,5s
5	*Cumulative Layout Shift (CLS)*	0,008



Gbr. 17 Pengujian 5 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Navigation*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL IX PENGUJIAN 5 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor
1	*First Contentful Paint (FCP)*	1,6s
2	*Total Blocking Time (TBT)*	0ms
3	*Speed Index (SI)*	1,6s
4	*Largest Contentful Paint (LCP)*	2,5s
5	*Cumulative Layout Shift (CLS)*	0,008

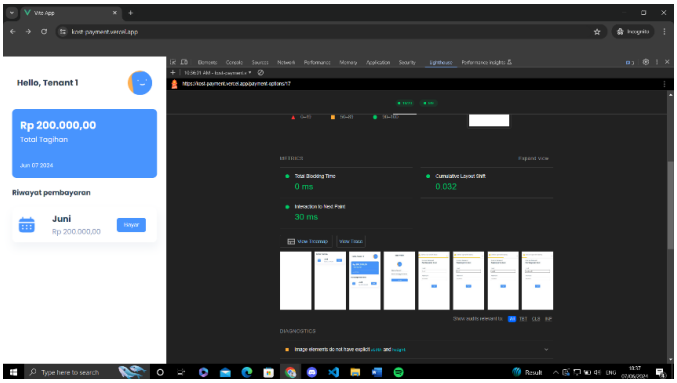
Berdasarkan tabel-tabel diatas, terlihat hasil dari pengujian implementasi *Single Page Application* pada *website* pembayaran kos yang di *host* pada *Vercel* menggunakan *Google Lighthouse* dengan mode *Navigation*, mode ini menguji *website* dalam *state default*-nya. Dari lima hasil pengujian diatas, peneliti dapat menarik nilai *mean* sebagai berikut.

TABEL X NILAI MEAN PENGUJIAN WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor	Keterangan
1	*First Contentful Paint (FCP)*	1,58s	Waktu rata-rata yang diperlukan untuk merender konten pertama dari *website* adalah 1,58 detik
2	*Total Blocking Time (TBT)*	0ms	Waktu rata-rata yang dibutuhkan dari *FCP*

No	Parameter	Skor	Keterangan
			hingga *website* sudah interaktif adalah 0 detik
3	*Speed Index (SI)*	1,58s	Waktu rata-rata yang dibutuhkan hingga konten-konten dalam *website* terlihat adalah 1,58 detik
4	*Largest Contentful Paint (LCP)*	2,5s	Waktu rata-rata yang dibutuhkan untuk merender konten terbesar adalah 2,5 detik
5	*Cumulative Layout Shift (CLS)*	0,0	Nilai rata-rata pergerakan elemen dan konten dalam halaman *web* adalah sebesar 0,0

Berdasarkan tabel diatas. Waktu yang dibutuhkan untuk menampilkan konten pertama kepada pengguna adalah 1,58 detik, hal tersebut dapat kita lihat pada parameter *First Content Paint* (*FCP*), kemudian waktu yang dibutuhkan dari *FCP* hingga *website* menjadi *interaktif* adalah 0 detik seperti yang dapat kita lihat pada parameter *Total Blocking Time* (*TBT*). Waktu yang dibutuhkan hingga konten-konten ditampilkan di *website* adalah 1,58 detik, hal ini dapat kita lihat pada parameter *Speed Index* (*SI*). Kemudian waktu untuk merender konten paling besar dapat kita lihat pada parameter *Largest Contentful Paint* (*LCP*), dimana waktu yang dibutuhkan adalah 2,5s.

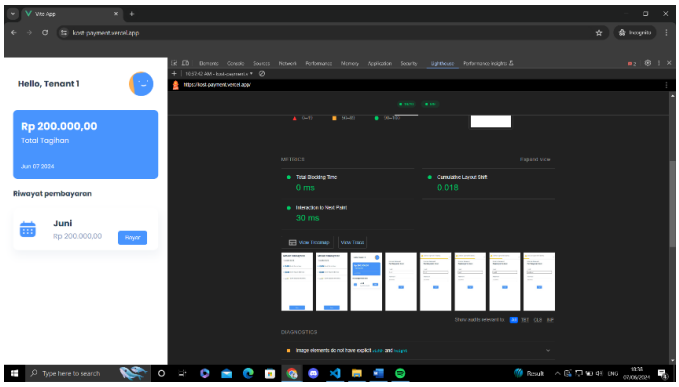


Gbr. 18 Pengujian 1 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Timespan*)

Hasil dari pengujian di atas disajikan dalam tabel berikut.

TABEL XI PENGUJIAN 1 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (TIMESPAN)

No	Parameter	Skor
1	*Total Blocking Time (TBT)*	0ms
2	*Cumulative Layout Shift (CLS)*	0,032
3	*Interaction to Next Paint (INP)*	30ms

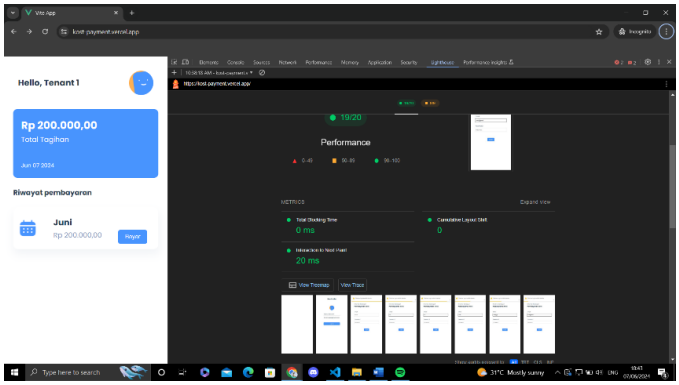


Gbr. 19 Pengujian 2 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Timespan*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL PENGUJIAN 2 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (TIMESPAN)

No	Parameter	Skor
1	*Total Blocking Time (TBT)*	0ms
2	*Cumulative Layout Shift (CLS)*	0,018
3	*Interaction to Next Paint (INP)*	30ms

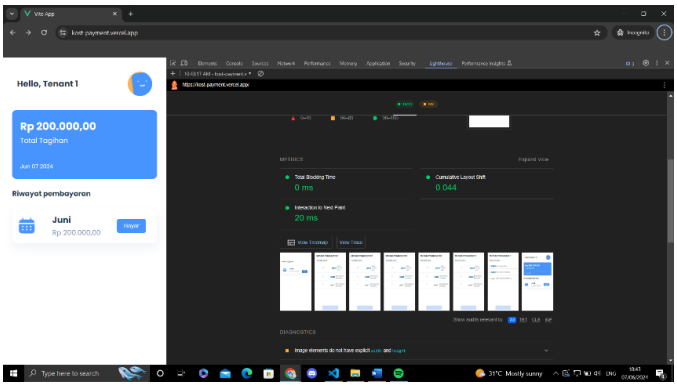


Gbr. 20 Pengujian 3 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Timespan*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL XII PENGUJIAN 3 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (TIMESPAN)

No	Parameter	Skor
1	*Total Blocking Time (TBT)*	0ms
2	*Cumulative Layout Shift (CLS)*	0
3	*Interaction to Next Paint (INP)*	20ms

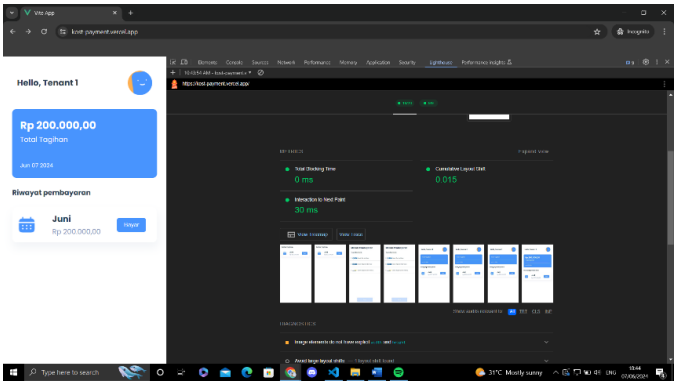


Gambar 21 Pengujian 4 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Timespan*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL XIII PENGUJIAN 4 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (TIMESPAN)

No	Parameter	Skor
1	*Total Blocking Time (TBT)*	0ms
2	*Cumulative Layout Shift (CLS)*	0,044
3	*Interaction to Next Paint (INP)*	20ms



Gbr. XIV Pengujian 5 *Website* yang di *host* di *Vercel* Menggunakan *Google Lighthouse* (*Timespan*)

Hasil dari pengujian diatas disajikan dalam tabel berikut.

TABEL 4. 11. PENGUJIAN 5 WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (TIMESPAN)

No	Parameter	Skor
1	*Total Blocking Time (TBT)*	0ms
2	*Cumulative Layout Shift (CLS)*	0,015
3	*Interaction to Next Paint (INP)*	30ms

Berdasarkan serangkaian pengujian diatas, terlihat hasil dari pengujian implementasi *Single Page Application* pada *website* pembayaran kos yang di *host* pada *Vercel* menggunakan *Google Lighthouse* dengan mode *Timespan*, mode ini menguji performa *website* saat pengguna berinteraksi dan memberi *input* pada *website*. Dari lima pengujian tersebut, peneliti dapat menarik nilai mean sebagai berikut.

TABEL XV NILAI MEAN PENGUJIAN WEBSITE YANG DI HOST DI VERCEL MENGGUNAKAN GOOGLE LIGHTHOUSE (NAVIGATION)

No	Parameter	Skor	Keterangan
1	*Total Blocking Time (TBT)*	0ms	Waktu rata-rata yang dibutuhkan dari *FCP* hingga *website* sudah interaktif adalah 0 detik
2	*Cumulative Layout Shift (CLS)*	0,021	Nilai rata-rata Pergerakan elemen dan konten dalam halaman *web* adalah sebesar 0,021
3	*Interaction to Next Paint (INP)*	26ms	Waktu yang dibutuhkan *website* untuk merespon *input* dari pengguna adalah 26 *milisecond* atau 0,026 detik

Waktu yang dibutuhkan dari *FCP* hingga website menjadi interaktif adalah 0 detik seperti yang dapat kita lihat pada parameter *Total Blocking Time* (*TBT*). Kemudian waktu yang dibutuhkan oleh *website* untuk merespon *input* dari *user* adalah 0,026 detik seperti yang terlihat pada parameter *Interaction to Next Paint* (*INP*).

2) Analisis Karakteristik Aplikasi

Karakteristik dari *SPA* ini peneliti dapatkan berdasarkan dari penelitian ini dan juga dari beberapa dokumentasi dan forum diskusi seperti *medium* dan *reddit*.

a. Kecepatan dan Responsivitas

Salah satu keuntungan paling signifikan dari *SPA* adalah kecepatan dan responsivitasnya. *SPA* memuat konten secara dinamis, yang berarti bahwa ketika pengguna berinteraksi dengan aplikasi, hanya konten yang diperlukan yang diambil dari *server*. Hal ini akan menyebabkan respon *website* terhadap interaksi user dan kecepatan *render website* setelah semua komponen selesai dimuat menjadi cepat.

b. Waktu Loading Awal

Karena *SPA* perlu memuat seluruh komponen aplikasinya untuk memastikan semua komponen dapat di *render* secara dinamis, maka waktu loading awal dari *website SPA* akan memakan sebagian besar atau keseluruhan dari waktu muat. Setelah muat awal selesai, seluruh komponen dalam *website* dapat di *load* dengan waktu yang cepat.

IV. KESIMPULAN

Berdasarkan hasil penelitian, pengujian, dan pembahasan yang telah dilakukan peneliti, maka dapat ditarik suatu kesimpulan sebagai berikut:

1. Dalam penelitian ini, hasil yang peneliti dapatkan menunjukkan bahwa *library Javascript Vue.js* sangat mendukung implementasi sistem pembayaran kos menggunakan *SPA*. dengan menggunakan *routing* yang tersedia dalam *Vue Router, website* dapat menampilkan konten yang berbeda-beda tanpa perlu berpindah halaman. Dalam *SPA*, seluruh komponen yang dibutuhkan *website* akan dimuat di awal, sehingga *website* dapat dengan bebas menampilkan komponen sesuai input pengguna dan path dari *route*. *Vue.js* juga memiliki fitur *reactive rendering* yang memeungkinkan nilai-nilai dalam elemen atau konten berubah tanpa perlu memuat ulang halaman.
2. Pada tahap pengujian yang dilakukan peneliti, *website SPA* sistem pembayaran rumah kos dapat berjalan dengan sebagaimana mestinya saat di *host* menggunakan *Vercel*. Peneliti kemudian menguji *website* yang telah di *host* tersebut menggunakan *Google Lighthouse*.
3. Hasil pengujian yang didapat oleh peneliti menunjukkan bahwa performa dari *website SPA* sistem pembayaran kos yang di *host* di *Vercel* memiliki metrik dengan standar yang dikategorikan cepat sesuai standar *Core Web Vitals*, dimana setiap metrik memiliki skor yang berada pada kategori "*Good*" atau area berwarna hijau.

REFERENSI

[1] Arimbi, Y. D., Kartinah, D., Nila, A., & Della, W. (2022). Rancangan Sistem Informasi Kost Putri Malika Berbasis Website Menggunakan Framework Laravel dan MySQL. Jurnal Ilmiah Multidisiplin, 1(3), 93–103. https://www.teamstart.my.id/.

[2] Hidayat, I. S., Setiawan, E., Efendi, Y., & Ihsan, T. (2023). Pengembangan Sistem Manajemen Kamar Kost Berbasis Web di Ikebana Kost Palembang.

[3] Kumar Athota, M. (2022). Management System for an Apartment. https://opus.govst.edu/capstones

[4] Maulana, I., & Ginanjar, R. (2017). Sistem Informasi Manajemen Kost Berbasis Web. Information System Application, 11(19).

[5] Paul, J. (2022). The Rental Zone (House Renting Website). In International Research Journal of Modernization in Engineering Technology and Science www.irjmets.com @International Research Journal of Modernization in Engineering. www.irjmets.com

[6] Peter Gommans, H., Mwenda Njiru, G., Nguka Owange, A., Proffessional, S., & Kenya Limited, T. (2014). Rental House Management System. International Journal of Scientific and Research Publications, 4(11). www.ijsrp.org

[7] Rafiq Misyam, M., & Selamat, N. (2021). House Rental Management System. Applied Information Technology And Computer Science, 2(2), 1745–1754. https://doi.org/10.30880/aitcs.2021.02.02.114

[8] Rathore, K., Syed, A., Patel, R., Tech, B., & Professor, E.-A. (2021). Rental House Management System. In International Research Journal of Modernization in Engineering Technology and Science www.irjmets.com @International Research Journal of Modernization in Engineering. www.irjmets.com

[9] web.dev (2024, March 12) *Interaction to Next Paint (INP)*

[10] web.dev (2024, February 19) *Largest Contentful Paint (LCP)*

[11] web.dev (2023, December 6) *First Contentful Paint (FCP)*

[12] web.dev (2023, November 17) *Total Blocking Time (TBT)*

[13] web.dev (2023, April 12) *Cumulative Layout Shift (CLS)*

[14] developer.chrome.com (2019, May 2) *Speed Index (SI)*

[15] Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[16] VAISHAK (2023, November 1) *The Pros and Cons of Single-Page Applications (SPAs)*