JURNAL ILMU KOMPUTER
KHARISMA .TECH

Jurnal KHARISMA Tech | ISSN:1907-2317 | e-ISSN: 2810-0344
https://jurnal.kharisma.ac.id/kharismatech
published by: Pusat Penelitian STMIK KHARISMA Makassar

Volume: 20 no. 01 – Maret 2025- hlm. 29-43

PERANCANGAN APPLICATION PROGRAMMING INTERFACE PADA WEBSITE SAHEB MENGGUNAKAN ARSITEKTUR REST

Oleh:

Vic Bradley Sanjaya[1], Syaiful Rahman[2*], Hendra Surasa[3]
[1,2] Teknik Informatika, STMIK Kharisma Makassar
e-mail: [1]vicbradley_21@kharisma.ac.id, [2]syaifulrahman@kharisma.ac.id,
[3]hendrasurasa@kharisma.ac.id

Abstrak: *Saheb adalah aplikasi petshop berbasis web yang dibangun menggunakan framework Next.js dan Firebase, Saheb menyediakan penjualan produk hewan peliharaan dan layanan konsultasi dengan dokter hewan secara online. Namun, Saheb menghadapi kendala dalam integrasi dengan aplikasi pihak ketiga dan efisiensi sistem yang terpengaruh oleh ketergantungan pada Firebase. Penelitian ini bertujuan merancang REST API menggunakan Express.js untuk meningkatkan modularitas, integrasi, dan kinerja aplikasi. Metode penelitian meliputi studi literatur, perancangan REST API dengan prinsip layered architecture, dan implementasi autentikasi menggunakan JWT. Evaluasi menunjukkan bahwa implementasi REST API mengurangi penggunaan sumber daya Firebase sebesar 57,66% dan meningkatkan performa website secara signifikan, dengan penurunan rata-rata waktu Speed Index, TBT, dan LCP. Integrasi REST API dengan aplikasi pihak ketiga berhasil memfasilitasi interaksi tanpa koneksi langsung ke Firebase. Hasil penelitian menyimpulkan bahwa penerapan REST API meningkatkan efisiensi, skalabilitas, dan kinerja aplikasi Saheb, serta membuka peluang untuk pengembangan lebih lanjut.*

Kata kunci: *REST API, Express.js, Firebase, JWT, petshop online, performa website*

Abstract: *Saheb is a web-based pet shop application built with Next.js and Firebase, offering pet product sales and online veterinary consultation services. However, Saheb faces challenges in integrating with third-party applications and system efficiency due to its reliance on Firebase. This study aims to design a REST API using Express.js to enhance modularity, integration, and application performance. The research methodology includes literature review, REST API design using layered architecture principles, and JWT authentication implementation. Evaluation shows that the REST API implementation reduced Firebase resource usage by 57.66% and significantly improved website performance, with average reductions in Speed Index, TBT, and LCP times. The integration of the REST API with third-party applications successfully facilitates interaction without direct Firebase connections. The study concludes that the implementation of REST API improves the efficiency, scalability, and performance of the Saheb application, and opens up opportunities for further development.*

Keywords: *REST API, Express.js, Firebase, JWT, online pet shop, website performance*

1. PENDAHULUAN

Saheb adalah aplikasi petshop berbasis web yang dibangun menggunakan *framework* Next.js dan memanfaatkan Firebase sebagai *Backend as a Service* (BaaS). Saheb dapat diakses melalui URL https://saheb-mu.vercel.app, dimana Saheb menawarkan dua fitur

* Corresponding author : Syaiful Rahman (syaifulrahman@kharisma.ac.id)

Diterima : *September, 2024* ***Disetujui :*** *November, 2024* ***Dipublikasikan :*** *April, 2025*

utama: penjualan beragam produk untuk hewan peliharaan, mulai dari makanan, perlengkapan, mainan, hingga obat-obatan, serta layanan konsultasi dengan dokter hewan secara *online* melalui fitur *chat*. Walaupun *website* Saheb sudah dapat diakses dengan baik, terdapat beberapa masalah yang perlu diatasi untuk meningkatkan efektivitasnya. Pertama, belum adanya antarmuka yang memungkinkan integrasi dengan aplikasi pihak ketiga, sehingga membatasi potensi ekspansi aplikasi. Kedua, kompleksitas kode semakin meningkat seiring dengan penambahan fitur, karena pengelolaan *Front End* dan *Back End* aplikasi berada dalam satu basis kode yang sama. Ketergantungan pada layanan BaaS yaitu Firebase juga menimbulkan tantangan dalam hal performa dan penggunaan sumber daya yang dapat menyebabkan peningkatan biaya operasi dan penurunan performa aplikasi.

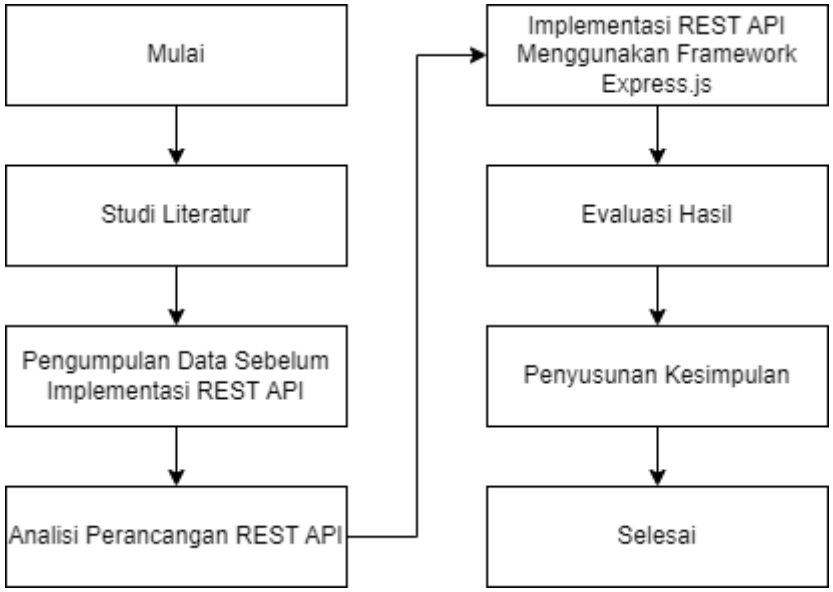
Untuk mengatasi masalah tersebut, peneliti bertujuan untuk merancang *Application Programming Interface* (API). API adalah sekumpulan aturan yang memungkinkan perangkat lunak berkomunikasi satu sama lain [1], [2], [3]. Peneliti akan mengembangkan API menggunakan arsitektur REST (*Representational State Transfer*), gaya arsitektur perangkat lunak yang memanfaatkan fitur HTTP yang diusulkan oleh Roy Fielding [4]. Peneliti memilih arsitektur REST karena kesederhanaan, kemudahan penggunaan, dan dukungan komunitas yang luas. Pengembangan API ini dapat secara efektif menyelesaikan tiga masalah utama Saheb. Pertama, dengan API memungkinkan *website* Saheb untuk melakukan integrasi dengan sistem informasi lainnya. Kedua, API dapat menciptakan lapisan abstraksi yang efisien antara *Front End* dan *Back End* aplikasi, sehingga memungkinkan modularitas yang lebih baik. Ketiga, API memungkinkan penggunaan layanan *Back End* yang lebih optimal (*caching*, *load balancing*, *pagination*) sehingga mengurangi ketergantungan pada layanan BaaS. Peneliti memilih untuk mengembangkan REST API menggunakan Express.js. Express.js adalah *framework* web berbasis Node.js yang populer untuk membangun Sistem *Back End* karena menyediakan kemudahannya dalam menangani rute-rute API, mengelola permintaan HTTP, dan mengoptimalkan pengolahan data [5].

Sebelum implementasi, peneliti melakukan kajian terhadap beberapa penelitian terdahulu yang berkaitan dengan penggunaan REST API. Dalam penelitian dengan judul “Integrasi Sistem Informasi Pemantauan Kualitas Lingkungan Air dan Udara Menggunakan REST API dan Web Service”, berhasil mengimplementasikan REST API dan *Web Service* untuk mengintegrasikan tiga sistem informasi pemantauan kualitas lingkungan yang berbeda. Hasil pengujian menunjukkan bahwa sistem beroperasi dengan stabil dan akurat [6]. Demikian pula, penelitian lain yang berjudul “Pengembangan Aplikasi E-learning Sukabaca Menggunakan Framework Express.js dan MongoDB” berhasil memanfaatkan REST API untuk mengintegrasikan aplikasi dengan basis data MongoDB [7]. Dengan dasar dari penelitian-penelitian tersebut, peneliti berharap untuk menerapkan konsep serupa guna meningkatkan efisiensi dan skalabilitas *website* Saheb. Peneliti juga bermaksud melakukan penelitian lebih lanjut terkait kemampuan REST API dalam beradaptasi dengan layanan BaaS seperti Firebase. Adanya implementasi ini diharapkan mampu memberikan wawasan baru tentang fleksibilitas REST API dalam mengelola berbagai layanan BaaS.

2. METODE PENELITIAN

2.1. Tahapan Penelitian

Gambar 1 menunjukkan tahapan penelitian diawali dengan studi literatur untuk memahami konsep REST API dan peran Express.js dalam pembangunan API. Data awal tentang penggunaan Firebase dan performa website Saheb dikumpulkan untuk evaluasi. Pada tahap perancangan, struktur dan endpoint REST API dikembangkan menggunakan Express.js. Implementasi meliputi pengembangan, pengujian, dan deployment di lingkungan pengembangan, diikuti evaluasi kinerja dan keamanan. Hasil penelitian menilai peningkatan efisiensi dan kinerja website Saheb, serta memberikan saran pengembangan lanjutan.



Gambar 1. Tahapan Penelitian

2.2. Metode Pengumpulan Data

Penelitian ini menggunakan sumber data primer, yaitu data penggunaan sumber daya Firebase serta data hasil pengukuran performa *website* sebelum implementasi REST API. Data yang dikumpulkan dalam penelitian ini bersifat kuantitatif, memungkinkan peneliti untuk melakukan analisis statistik.

1. Data Penggunaan Sumber Daya Firebases

Pengukuran penggunaan sumber daya Firebase pada aplikasi web Saheb dilakukan dengan mengamati jumlah *document reads* (jumlah dokumen yang diakses atau dibaca) dari Firebase. Data ini penting untuk memahami seberapa sering dan intensif aplikasi mengakses data dari *database*. Proses pengumpulan data dilakukan dengan fokus mengakses halaman produk dimana ada total 50 data produk dan halaman konsultasi dengan 4 data konsultan sebanyak 5 kali. Pengukuran ini dilakukan hanya satu kali dan hanya dengan satu pengguna untuk memastikan konsistensi data selama pengukuran. Gambar 2 menunjukkan hasil pengukuran dengan jumlah *document reads* yang tercatat adalah 274.

Gambar 2. Data Penggunaan Sumber Daya Firebase Sebelum Implementasi REST API

2. Data Performa *Website*

Data performa halaman produk pada *website* Saheb akan diukur menggunakan *tools* pengukuran performa *website* yaitu PageSpeed Insights, GTmetrix, dan WebPage Test. Peneliti memilih halaman produk sebagai objek pengujian karena halaman produk merupakan salah satu halaman paling penting dan paling sering dikunjungi oleh pengguna. *Tools* pengukuran performa akan mengevaluasi skor kinerja berdasarkan metrik: *Speed Index* (kecepatan tampilan konten), *First Contentful Paint* (FCP, waktu tampil konten pertama dari DOM), *Total Blocking Time* (TBT, waktu halaman terblokir oleh tugas JavaScript), *Largest Contentful Paint* (LCP, waktu tampil konten terbesar di *viewport*), dan *Cumulative Layout Shift* (CLS, stabilitas visual halaman). Setelah pengumpulan data, hasil dari setiap metrik dirata-ratakan untuk menghitung performa awal setiap metrik sebelum implementasi REST API. Tabel 1 menyajikan hasil pengukuran performa dari ketiga *tools* tersebut.

Tabel 1 : Pengukuran Performa Sebelum Implementasi REST API dan Optimasi

Tools Pengukuran	**Speed Index**	**FCP**	**TBT**	**LCP**	**CLS**
PageSpeed Insights	2.4 s	0.3 s	350 ms	4.5 s	1
GTmetrix	3.4 s	0.409 s	70 ms	6.1 s	0
WebPage Test	12.3 s	3.748 s	1474 ms	14.6 s	0
Rata - Rata	**6.03 s**	**1.4856 s**	**631.3 ms**	**8.4 s**	**0.33**

Tabel 2 : Indikator Pengukuran Kualitas Metrik

Metrik	**Good**	**Need Improvement**	**Poor**
Speed Index	≤ 3.4 s	> 3.4 s dan ≤ 5.8 s	> 5.8 s
FCP	≤ 1.8 s	> 1.8 s dan ≤ 3 s	> 3 s
TBT	≤ 200 ms	> 200 ms dan ≤ 600 ms	> 600 ms
LCP	≤ 2.5 s	> 2.5 s dan ≤ 4 s	> 4 s
CLS	≤ 0.1	> 0.1 dan ≤ 0.25	> 0.25

Tabel 2 digunakan sebagai acuan untuk menentukan kualitas setiap metrik. Indikator pengukuran kualitas metrik yang digunakan dalam penelitian ini didapatkan dari jurnal ilmiah dan dokumentasi dari Google [8]. Berdasarkan tabel indikator pengukuran kualitas metrik, performa *website* Saheb saat ini berada dalam kategori *"Poor"* untuk semua metrik yang diukur, kecuali metrik FCP.

3. HASIL DAN PEMBAHASAN

3.1. Implementasi REST API Menggunakan Layered Architecture

Layered architecture adalah pendekatan desain perangkat lunak yang memisahkan aplikasi ke dalam beberapa lapisan dengan tanggung jawab spesifik untuk meningkatkan modularitas dan maintainability. Pada arsitektur REST API Saheb, sistem terdiri dari tiga lapisan utama: *Presentation Layer*, *Business Layer*, dan *Database Layer* [9].

1. *Presentation Layer (Controller)*

```
import express from "express";
import { getAllProduct, getProductById, getProductsByKeyword } from "./product.services.js";

const productController = express.Router();

productController.get("/", async (req, res) => {
  try {
    const page = parseInt(req.query.page) || 1;
    const limit = 20;

    const { products, totalItems, totalPages } = await getAllProduct(page, limit);

    res.status(200).send({ products, totalItems, totalPages });
  } catch (error) {
    res.status(400).send(error.message);
  }
});

productController.get("/search/:query", async (req, res) => {
  try {
    const searchedProducts = await getProductsByKeyword(req.params.query);

    res.status(200).send(searchedProducts);
  } catch (error) {
    res.status(400).send(error.message);
  }
});

productController.get("/:id", async (req, res) => {
  try {
    const productId = req.params.id;
    const product = await getProductById(productId);

    res.json(product);
  } catch (error) {
    res.status(400).send(error.message);
  }
});

export default productController;
```

Gambar 3. Sub Kode *Presentation Layer* (*Controller*)

Presentation Layer (*Controller*) berfungsi untuk menangani siklus permintaan dan respons dari pengguna serta menyediakan berbagai fungsi untuk *endpoint* yang terkait. Pada Gambar 3, sub kode tersebut mengimplementasikan fungsi-fungsi yang menangani rute yang berhubungan dengan produk, contohnya fungsi untuk mendapatkan daftar produk, pencarian produk, dan mendapatkan detail produk berdasarkan ID. Setiap permintaan kemudian diteruskan ke *Business Layer* yang bertanggung jawab atas logika bisnis lebih lanjut.

2. *Business Layer (Services)*

```
import { findAllProduct, findProductById, findProductsByKeyword } from "./product.repository.js"

export const getAllProduct = async (page, limit) => {
  const { products, totalItems } = await findAllProduct(page, limit);
  const totalPages = Math.ceil(totalItems / limit);

  return { products, totalItems, totalPages };
}

export const getProductsByKeyword = async (query) => {
  const searchedProducts = await findProductsByKeyword(query);

  return searchedProducts;
}

export const getProductById = async (productId) => {
  const product = await findProductById(productId);

  return product;
}
```

Gambar 4. Sub Kode *Business Layer (Services)*

Business Layer (*Services*) berperan sebagai perantara antara *Presentation Layer* dan *Database Layer*. Gambar 4 menunjukkan contoh implementasi logika bisnis terkait produk, seperti pengambilan data dengan fitur *pagination*, pencarian produk berdasarkan kata kunci, dan pengambilan detail produk.

3. Business Layer (Services)

```
import { collection, getDocs, doc, getDoc, limit, query, startAfter  } from "firebase/firestore";
import { db } from "./../db/firebase.js";
import MiniSearch from 'minisearch';

export const findAllProduct = async (page, limitValue) => {
  const productCollection = collection(db, "products");

  // Hitung total item
  const totalSnapshot = await getDocs(productCollection);
  const totalItems = totalSnapshot.size;

  // Tentukan offset
  const offset = (page - 1) * limitValue;

  // Buat query untuk paginasi
  let productQuery = query(productCollection, limit(limitValue));
  if (offset > 0) {
    const lastVisibleDoc = totalSnapshot.docs[offset - 1];
    productQuery = query(productCollection, startAfter(lastVisibleDoc), limit(limitValue));
  }

  const snapshot = await getDocs(productQuery);
  const products = snapshot.docs.map((doc) => ({
    ...doc.data(),
    id: doc.id,
  }));

  return { products, totalItems };
};

export const findProductsByKeyword = async (query) => {···
};

export const findProductById = async (productId) => {···
};
```

Gambar 5. Sub Kode *Database Layer* (Repository)

Database Layer (*Repository*) bertugas mengakses dan mengelola data dari *database* (Firebase). Contohnya pada Gambar 5 disediakan berbagai operasi untuk pengambilan data produk pada *collection products* di Firebase, pencarian produk menggunakan *library* pencarian *full-text MiniSearch*, dan pengambilan detail produk berdasarkan ID produk di Firebase. Pemisahan yang jelas antara ketiga lapisan ini meningkatkan modularitas dan *maintainability* dari aplikasi.

3.2. Implementasi Sistem Autentikasi Menggunakan JWT

JWT adalah standar terbuka yang digunakan untuk mentransfer informasi secara aman antara dua pihak dalam format JSON [10]. Sistem autentikasi REST API Saheb akan menggunakan JWT karena JWT memungkinkan autentikasi yang aman, efisien, dan meningkatkan skalabilitas tanpa memerlukan penyimpanan sesi di server.

1. Proses Pembuatan JWT pada Saat Pengguna *Login*

```
userController.post("/login", async (req, res) => {
  try {
    const isUserAuth = await checkUserAuth(req.body);

    if (!isUserAuth) {
      return res.status(400).send("Invalid Auth");
    }

    const accessToken = jwt.sign({ userId: req.body.userId }, config.jwtSecret);

    res.status(200).send(accessToken);
  } catch (error) {
    res.status(400).send(error.message);
  }
});
```

Gambar 6. Sub Kode Fungsi Untuk Membuat Token JWT

Gambar 6 menunjukkan bagaimana proses pembuatan JWT pada sistem REST API Saheb. Proses dimulai ketika pengguna mengirim permintaan *POST* ke *endpoint "/login"* dengan kredensial milik pengguna. Server memvalidasi kredensial melalui fungsi *"checkUserAuth"*. Jika tidak valid, respons status 401 dengan pesan *"Invalid Auth"* dikirim. Jika valid, server menghasilkan JWT dengan *"jwt.sign()"*, memuat *payload* seperti *userId*, dan ditandatangani dengan kunci rahasia. Token dikirim kembali ke klien dengan status 200 untuk digunakan pada permintaan berikutnya, disertakan di *header* dalam format *"Authorization: Bearer <token>"*. Klien biasanya menyimpan token di *local storage* atau *session storage*.

2. *Middleware* untuk Memverifikasi Token JWT

```
import jwt from "jsonwebtoken";
import config from "../../config.js";

export const verifyToken = (req, res, next) => {
  const authHeader = req.headers["authorization"];

  if (!authHeader) {
    return res.status(403).send({ message: "No token provided." });
  }

  const token = authHeader.split(" ")[1];

  jwt.verify(token, config.jwtSecret, (err, decoded) => {
    if (err) {
      return res.status(401).send({ message: "Unauthorized!" });
    }

    req.userId = decoded.userId;

    next();
  });
};
```

Gambar 7. Middleware *verifyToken*

Sub Kode diatas menunjukkan bagaimana *Middleware verifyToken* digunakan untuk memverifikasi token JWT pada *endpoint* yang memerlukan izin. Prosesnya dimulai dengan memeriksa keberadaan *header Authorization* jika tidak ada, server mengembalikan respons status 400 dengan pesan *"No token provided."* Token kemudian diekstraksi dari *header Authorization* dengan format *"Bearer {token}."* Selanjutnya, token diverifikasi menggunakan *"jwt.verify"* dengan kunci rahasia pada *file* konfigurasi aplikasi. Jika verifikasi berhasil, *userId* dari token akan disimpan pada objek *req*, dan permintaan dilanjutkan ke rute berikutnya. Jika token tidak valid, server mengirim respons status 401 dengan pesan *"Unauthorized"*.

3.3 Implementasi REST API Pada Aplikasi Pihak Ketiga

Pada subbab ini akan diberikan contoh bagaimana cara melakukan permintaan ke REST API Saheb apabila aplikasi tidak terhubung langsung dengan Firebase Saheb. Penjelasan ini bertujuan untuk menggambarkan bagaimana aplikasi pihak ketiga dapat melakukan interaksi dengan REST API Saheb.

1. Ilustrasi Penggunaan *API Key* pada Aplikasi Pihak Ketiga

```
const handleSubmit = async (event) => {
  event.preventDefault();

  const data = {
    transactionId,
    recipientName,
    shipmentProof,
  };

  try {
    const response = await fetch(`https://saheb-api.vercel.app/transactions/${transactionId}`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "x-api-key": process.env.NEXT_PUBLIC_API_KEY,
      },
      body: JSON.stringify(data),
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.statusText}`);
    }

    const result = await response.json();

    toast.success("Transaksi berhasil diselesaikan");
    onTransactionComplete();
  } catch (error) {
    toast.error(`Error: ${error.message}`);
  }
};
```

Gambar 8. Sub Kode Untuk Melakukan permintaan *POST* ke REST API Saheb

Fungsi *handleSubmit* merupakan fungsi yang berada pada aplikasi diluar sistem Saheb, sehingga tidak terhubung langsung dengan *Database* milik Saheb. Fungsi ini berperan sebagai ilustrasi umum dari bagaimana aplikasi pihak ketiga dapat berinteraksi dengan REST API Saheb, di mana kunci utama keberhasilan akses adalah penggunaan *API key* yang valid pada *header* permintaan spesifiknya pada atribut *x-api-key*. Tanpa *API key* yang valid, permintaan yang dikirimkan oleh aplikasi pihak ketiga tidak akan dapat diproses.

2. *Middleware* untuk Memverifikasi *API Key*

```
import { collection, where, query, getDocs } from "firebase/firestore";
import { db } from "../db/firebase.js";

export const verifyAPIKey = async (req, res, next) => {
  try {
    const apiKey = req.headers["x-api-key"];

    if (!apiKey) return res.status(401).json({ message: "No API Key Provided" });

    const q = query(collection(db, "integratedApps"), where("apiKey", "==", apiKey));
    const querySnapshot = await getDocs(q);

    if (querySnapshot.empty) return res.status(401).json({ message: "Invalid API Key" });

    next();
  } catch (error) {
    return res.status(500).json({ message: "Internal server error", error: error.message });
  }
};
```

Gambar 9. Middleware verifiyAPIKey

Middleware verifyAPIKey digunakan untuk memverifikasi *API Key* sebelum memproses data lebih lanjut. Pertama, fungsi akan memeriksa apakah *API Key* disertakan dalam *header* permintaan. Jika tidak ada, server akan mengembalikan status 401 dengan pesan *"No API Key Provided"*. Jika ada, *middleware* melakukan *query* pada koleksi *integratedApps* pada Firebase untuk mencari dokumen yang memiliki *API Key* sesuai. Jika tidak ditemukan dokumen yang sesuai, artinya *API Key* yang dikirimkan tidak valid dan tidak terdaftar di sistem. Hal ini menandakan bahwa permintaan berasal dari sumber yang tidak diotorisasi, sehingga perlu ditolak untuk menjaga keamanan data dan mencegah akses ilegal. Dalam situasi ini, server akan mengembalikan status 401 dengan pesan *"Invalid API Key"* sebagai respon standar untuk permintaan yang tidak terautentikasi. Jika valid, *middleware* memanggil metode "*next()"* untuk melanjutkan ke *middleware* berikutnya atau *handler request* berikutnya.

3. Penggunaan *Middleware verifyAPIKey*

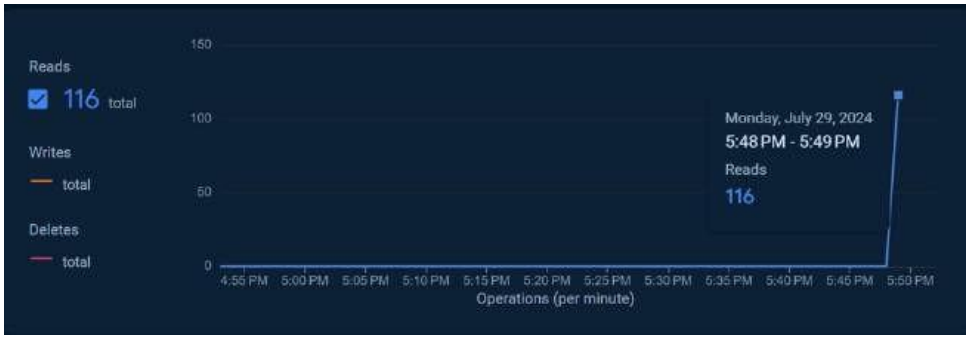
```
transactionController.post("/:userId", verifyAPIKey, async (req, res) => {
  try {
    const transaction = await editTransactionById(req.body);

    res.status(200).json(transaction);
  } catch (error) {
    res.status(400).send(error.message);
  }
});
```

Gambar 10. Contoh Penggunaan *middleware verifyAPIKey* pada Rute yang Memerlukan Autentikasi dengan *API Key*

Subkode diatas menunjukkan contoh penggunaan *middleware verifyAPIKey* pada sebuah *route handler* yang menangani permintaan *POST* pada *endpoint "transactions/:userId". Middleware verifyAPIKey* memastikan bahwa hanya permintaan dengan *API key* yang valid yang dapat melakukan operasi pada endpoint ini, memberikan lapisan keamanan tambahan untuk menghindari akses yang tidak sah.

3.4. Evaluasi Penggunaan Sumber Daya Firebase



Gambar 11. Data Penggunaan Sumber Daya Firebase Setelah Implementasi REST API dan Optimasi

Setelah implementasi REST API pada aplikasi web Saheb, terjadi penurunan jumlah *document reads* pada Firebase dari 274 menjadi 116, atau sebesar 57,66%. Pengurangan ini disebabkan oleh pengelolaan dan penyajian data yang lebih efisien melalui REST API, yang memungkinkan akses data secara lebih selektif dan terstruktur. Hasilnya, performa aplikasi meningkat dalam hal responsivitas dan kecepatan akses data, serta mengurangi biaya penggunaan sumber daya Firebase. Evaluasi ini menunjukkan bahwa penerapan REST API efektif dalam mengoptimalkan penggunaan *database* pada aplikasi web Saheb.

3.5. Evaluasi Performa Website

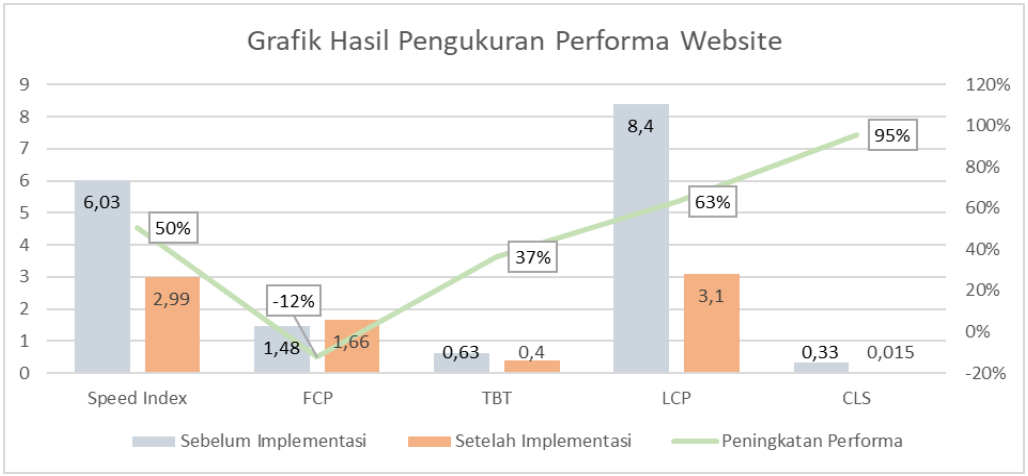
Setelah implementasi REST API pada aplikasi web Saheb, dilakukan pengukuran ulang pada performa *website*. Berikut hasil pengukuran performa setelah implementasi:

Tabel 3 : Pengukuran Performa Website Setelah Implementasi REST API dan Optimasi

Tools Pengukuran	**Speed Index**	**FCP**	**TBT**	**LCP**	**CLS**
PageSpeed Insights	1.7 s	0.3 s	160 ms	1.2 s	0
GTmetrix	1.7 s	0.86 s	0 ms	1.9 s	0.01
WebPageTest	5.59 s	3.826 s	1064 ms	6.22 s	0.035
Rata - Rata	**2.99 s**	**1.66 s**	**408 ms**	**3.1 s**	**0.015**

Tabel 4 : Pengukuran Performa Website Sebelum dan Setelah Implementasi REST API

Metric	**Rata-rata Sebelum Implementasi**	**Status**	**Rata-rata Setelah Implementasi**	**Status**
Speed Index	6.03 s	Poor	2.99 s	Good
FCP	1.4856 s	Good	1.66 s	Good
TBT	631.3 ms	Poor	408 ms	Need Improvement
LCP	8.4 s	Poor	3.1 s	Need Improvement
CLS	0.33	Poor	0.015	Good



Gambar 12. Grafik Perbandingan Performa *Website* Sebelum dan Setelah Implementasi REST API

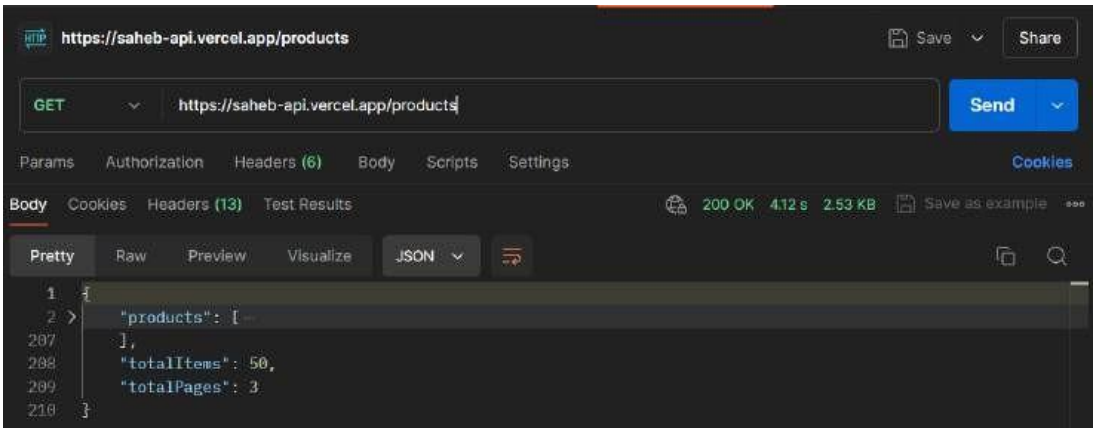
Setelah implementasi REST API dan optimasi, semua metrik performa menunjukkan peningkatan yang signifikan. Tabel 3 menampilkan pengukuran performa setelah implementasi menggunakan *tools* pengukuran yang sama, dengan hasil yang menunjukkan penurunan waktu pemuatan pada metrik utama seperti *Speed Index*, FCP, TBT, LCP, dan CLS, sehingga meningkatkan responsivitas halaman. Tabel 4 membandingkan performa sebelum dan sesudah implementasi, menegaskan bahwa semua metrik mengalami peningkatan signifikan, meskipun TBT dan LCP masih memerlukan optimasi lebih lanjut. Gambar 12 menggambarkan secara visual perbandingan performa sebelum dan sesudah implementasi REST API, memperlihatkan perubahan signifikan pada setiap metrik utama.

3.6. Evaluasi Integrasi REST API dengan Aplikasi Pihak Ketiga

Evaluasi ini bertujuan untuk menguji efektivitas dan kinerja REST API yang dikembangkan dalam menghubungkan Saheb dengan aplikasi pihak ketiga. Pengujian akan dilakukan menggunakan dua alat: sebuah *website* demo yang dapat diakses melalui URL https://demo-saheb-api.vercel.app/ dan Postman. Pengujian ini mencakup tiga jenis rute akses yang disediakan oleh REST API Saheb, yaitu: rute publik, rute yang memerlukan autentikasi JWT, dan rute yang memerlukan *API Key*.

1. Rute Publik

Gambar 13. Hasil Pengujian Rute Publik pada *Website* Demo

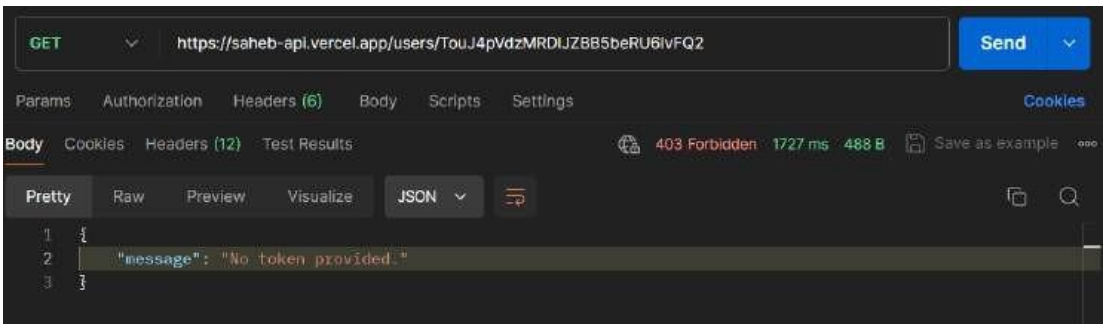


Gambar 14. Hasil Pengujian Rute Publik pada Postman

Rute publik adalah rute pada API yang dapat diakses oleh semua pengguna tanpa autentikasi. Rute ini umumnya digunakan untuk menyediakan informasi umum yang tidak memerlukan pembatasan akses, seperti data yang tidak bersifat sensitif atau layanan umum. *Endpoint* yang diuji adalah https://saheb-api.vercel.app/products, yang dirancang untuk menyediakan data produk dalam format JSON. Gambar 13 menunjukkan bagaimana *website* demo (sistem terpisah dari *website* Saheb) dapat mengakses data produk Saheb dengan baik. Gambar 14 juga menunjukkan API dapat merespon dengan baik melalui *platform testing* seperti Postman.

2. Rute yang Memerlukan Autentikasi Menggunakan JWT

Gambar 15. Hasil Pengujian Rute yang Memerlukan Autentikasi Menggunakan JWT pada *Website* Demo

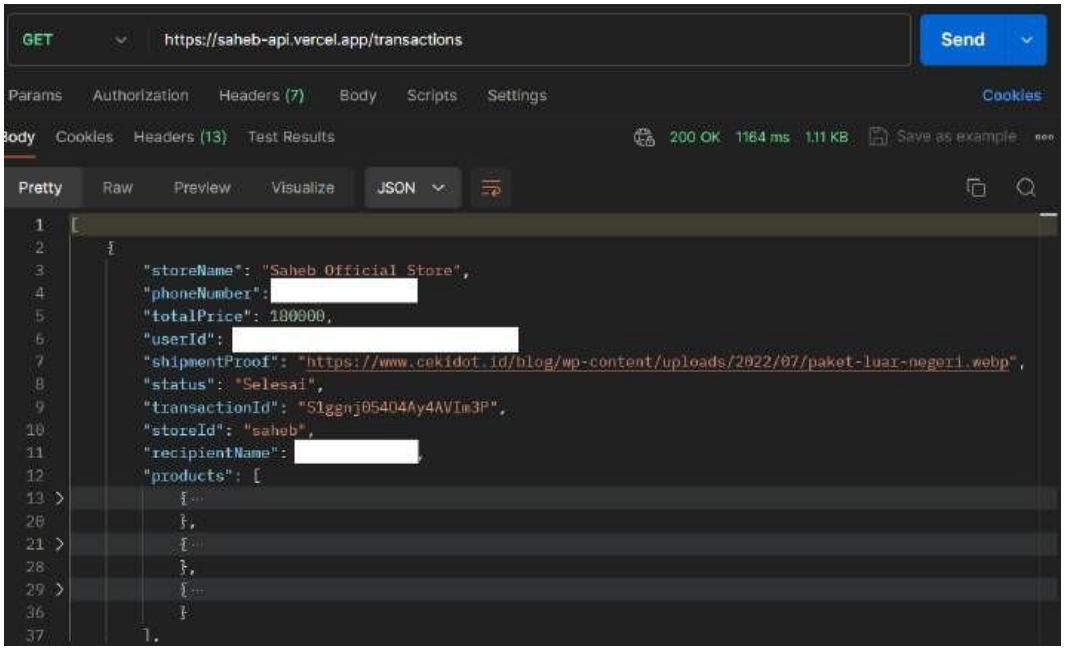


Gambar 16. Hasil Pengujian Rute yang Memerlukan Autentikasi Menggunakan JWT pada Postman

Gambar 15 dan 16 menunjukkan bagaimana *website* demo dan Postman merespon permintaan ke REST API Saheb pada rute yang diproteksi (rute yang mengandung informasi sensitif) gagal karena pengguna tidak terautentikasi melalui token JWT. *Website* demo dan Postman mencoba mengakses *endpoint* untuk mendapatkan data pengguna, tetapi gagal karena tidak adanya token autentikasi yang valid. Sistem mengembalikan respon dengan kode status 403 *"Forbidden"*, menunjukkan bahwa pengguna tidak memiliki izin untuk mengakses sumber daya yang diminta tanpa token autentikasi yang valid. Pengujian ini berhasil menunjukkan bahwa hanya pengguna yang sah dan terverifikasi yang dapat mengakses data dan layanan yang dilindungi.

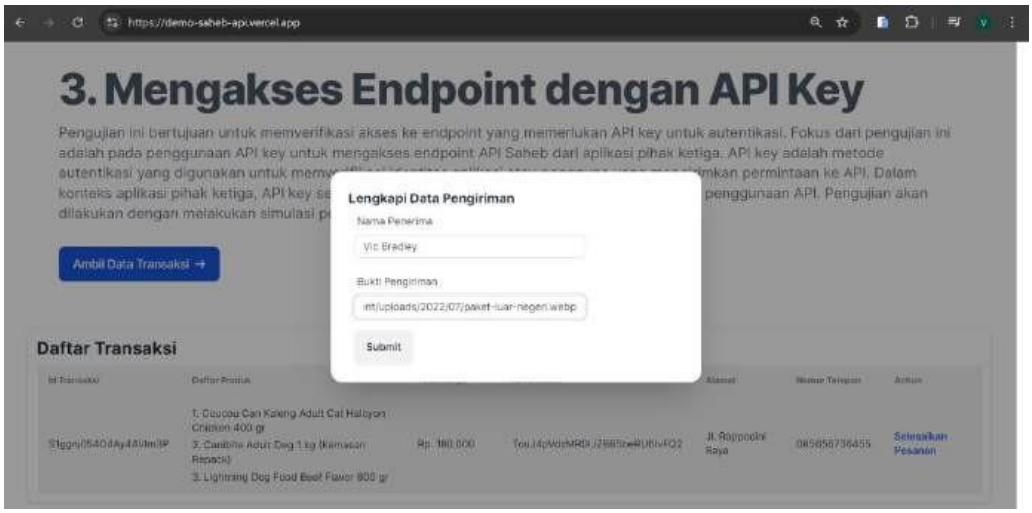
3. Rute yang Memerlukan *API Key*

Gambar 17. Hasil Pengujian Rute yang Memerlukan *API Key* pada *Website* Demo

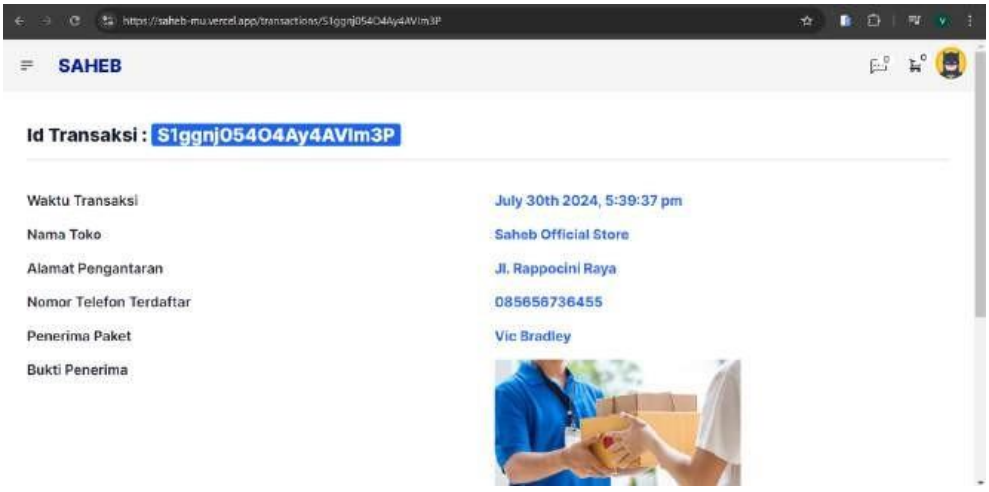


Gambar 18. Hasil Pengujian Rute yang Memerlukan *API Key* pada Postman

Rute ini dirancang untuk aplikasi pihak ketiga yang memerlukan akses khusus ke data atau layanan tertentu dari Saheb. Setiap aplikasi pihak ketiga harus memiliki *API Key* yang valid untuk mengakses rute ini. Gambar 17 dan 18 menunjukkan bahwa keduanya dapat menampilkan daftar transaksi yang memerlukan pengiriman meskipun tidak terhubung langsung dengan Firebase yang dimiliki oleh Saheb. Hal ini dimungkinkan karena menyertakan *API Key* pada *header* permintaan. Dengan adanya *API Key* yang valid, permintaan dari *website* demo dapat diotorisasi dan data dapat ditampilkan, begitu juga pada Postman data diterima dalam bentuk JSON.



Gambar 19. Pengujian Pengiriman Data Transaksi dari *Website* Demo



Gambar 20. Hasil Pengiriman Data Transaksi pada *Website* Saheb

Gambar 19 menunjukkan proses website demo Saheb melakukan permintaan ke REST API Saheb untuk menyelesaikan transaksi. Setelah permintaan diproses, detail transaksi akan dicatat dan ditampilkan di sistem utama Saheb, seperti yang ditampilkan pada gambar 20. Hal ini menunjukkan bagaimana data yang dikirim melalui *website* demo dapat diintegrasikan dan ditampilkan pada *website* Saheb, sehingga memberikan pengalaman yang konsisten bagi pengguna.

4. Kesimpulan

Penelitian ini berhasil merancang dan mengimplementasikan REST API pada website Saheb menggunakan Express.js, yang meningkatkan modularitas, efisiensi, dan keamanan aplikasi. Penggunaan sumber daya Firebase berkurang sebesar 57,66% setelah implementasi, menunjukkan pengelolaan data yang lebih efisien. Performa website juga mengalami peningkatan signifikan, meskipun beberapa metrik masih perlu dioptimalkan. Integrasi REST API dengan aplikasi pihak ketiga berjalan sukses, memungkinkan ekspansi dan kolaborasi lebih lanjut. Saran untuk pengembangan berikutnya adalah fokus pada optimasi metrik performa dan peningkatan skalabilitas sistem.

DAFTAR PUSTAKA

[1] F. Tias Dewantoro and A. Fira Waluyo, "Penerapan Rest Api Dalam Perancangan Aplikasi Reservasi Perawatan dan Penitipan Hewan Berbasis Android," *Media Online)*, vol. 4, no. 2, pp. 1011–1020, 2023, doi: 10.30865/klik.v4i2.1262.

[2] I. Gita Anugrah and M. Aldi Rifai Imam Fakhruddin, "Development Authentication and Authorization Systems of Multi Information Systems Based REst API and Auth Token,"

[3] N. A. Rakhmawati, S. H. Suryawan, M. A. Furqon, and D. Hermansyah, "Indonesia's Public Application Programming Interface (API)," *Jurnal Penelitian Pos dan Informatika*, vol. 9, no. 2, pp. 85–96, Dec. 2019, doi: 10.17933/jppi.v9i2.167.

[4] R. T. Fielding *et al.*, "Reflections on the REST architectural style and 'principled design of the modern web architecture' (impact paper award," in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, Aug. 2017, pp. 4–14. doi: 10.1145/3106237.3121282.

[5] R. Handoyo, L. W. Santoso, and A. Setiawan, "Real-Time BPMN Website Menggunakan Teknologi MERN Stack," 2019. [Online]. Available: https://www.mongodb.com/blog/post/the-modern-

[6] M. A. Salim, H. D. Pusat Teknologi Lingkungan, B. Pengkajian dan Penerapan Teknologi Gedung, K. Puspiptek Serpong, dan T. Selatan, "INTEGRASI SISTEM INFORMASI PEMANTAUAN KUALITAS LINGKUNGAN AIR DAN UDARA MENGGUNAKAN REST API DAN WEB SERVICE," Integrasi Sistem Informasi… JRL, vol. 14, no. 2, hlm. 183–192, 2021.

[7] G. Faqih Sucipto dan A. Soeharso, "Pengembangan Aplikasi E-learning Sukabaca Menggunakan Framework Express.js dan MongoDB."

[8] K. Król and W. Sroka, "Internet in the Middle of Nowhere: Performance of Geoportals in Rural Areas According to Core Web Vitals," *ISPRS Int J Geoinf*, vol. 12, no. 12, Dec. 2023, doi: 10.3390/ijgi12120484.

[9] M. (W. M. Richards, *Software architecture patterns : understanding common architecture patterns and when to use them*. 2015.

[10] U. Budi, A., and A. Sujarwo, "Penerapan JSON Web Token sebagai Strategi Pengamanan Data pada Aplikasi MultiMasjid," 2023.