e-ISSN2774-5155
p-ISSN 2774-5147

***Jurnal Sosial dan Teknologi (SOSTECH)***
Volume 4, Number 6, *Juni* 2024
p-ISSN **2774-5147** ; e-ISSN **2774-5155**

# Perancangan Dasbor yang *Secure Scalable* dan *Reusable* dengan Microservices Case Study Di PT. XYZ

**Shofyan[1], Sani Muhamad Isa**
Universitas Bina Nusantara, Indonesia
Email: pencari.angin1@gmail.com, agus.bandiyono@stan.ac.id

## Abstrak

Era digital, perusahaan menghadapi tantangan dalam mengembangkan dan mengelola dashboard monitoring produk digital, terutama terkait pengeluaran tinggi dan kurangnya kualitas akibat pembangunan ulang untuk skalabilitas dan keamanan. Solusi yang diusulkan adalah menggunakan arsitektur mikroservice, meskipun kompleksitas dan isu keamanan harus diperhatikan. Penelitian ini fokus pada pengembangan dashboard skalabel, dengan perhatian pada keamanan data dan manajemen pengguna, memberikan panduan praktis bagi organisasi untuk implementasi dashboard yang terstandarisasi, aman, dan efisien dalam arsitektur mikroservice, berpotensi meningkatkan efisiensi operasional dan melindungi data sensitif. Tujuan penelitian ini adalah Penelitian ini menekankan pentingnya keamanan data pribadi yang sensitif. Hasil penelitian ini memberikan wawasan tentang strategi keamanan yang dapat membantu melindungi data pribadi dalam lingkungan *microservice*. Metode yang digunakan penelitian ini adalah studi literatur, analisis teknis, desain arsitektur, pengembangan, pengujian, sedangkan untuk hasil dari penelitian ini adalah menghasilkan penelitian yang berfokus pada keamanan, skalabilitas, dan kegunaan kembali (reusability) telah membawa manfaat yang signifikan. Kesimpulan yang didapat dari penelitian ini adalah dapat menerapkan prinsip-prinsip keamanan OWASP, dasbor dapat dijamin keamanannya terhadap berbagai serangan yang umum terjadi. Penggunaan arsitektur microservices memungkinkan dasbor untuk menjadi lebih skalabel, memungkinkan penyesuaian dengan beban kerja yang berubah-ubah tanpa mengorbankan performa. Selain itu, penerapan *framework* komponen *backend* dan *frontend* yang dapat digunakan kembali telah meningkatkan efisiensi pengembangan dan kualitas kode secara keseluruhan.

**Kata Kunci : Arsitektur Mikroservice, Pengembangan Dashboard, Keamanan Data, Manajemen Pengguna Skalabilitas Sistem.**

## Abstract

*In the digital era, companies face challenges in developing and managing digital product monitoring dashboards, especially related to high expenditure and lack of quality due to rebuilding for scalability and security. The proposed solution is to use a microservices architecture, although complexity and security issues must be taken into account. This research focuses on the development of scalable dashboards, with attention to data security and user management, providing practical guidance for organizations for the implementation of standardized, secure, and efficient dashboards in microservices architectures, potentially improving operational efficiency and protecting sensitive data. The purpose of this research is that this research emphasizes the importance of the security of sensitive personal data. The results of this study provide insight into security strategies that can help protect personal data in a microservice environment. The methods used in this study are literature study, technical analysis, architectural design, development, testing, while the results of this study are to produce research that focuses on security, scalability, and reusability has brought significant benefits. The conclusion obtained from this study is that by applying OWASP security principles, the dashboard can be guaranteed to be safe against various common attacks. The use of microservices architecture allows dashboards to become more scalable, allowing for adaptation to changing workloads without sacrificing performance. In addition, the implementation of reusable backend and frontend component frameworks has improved development efficiency and overall code quality.*

***Keywords: Microservice Architecture, Dashboard Development, Data Security, User Management, System Scalability.e data, and enhance user management and access control.***

e-ISSN2774-5155
p-ISSN 2774-5147

## PENDAHULUAN

PT. XYZ adalah perusahaan yang memiliki banyak anak perusahaan, setiap anak perusahaan memiliki produk digital yang harus dijual. Salah satu layanan pendukung yang selalu dibuat pada setiap produk digital tersebut ialah dasbor monitoring baik untuk memantau penjualan ataupun, memantau proses bisnis internal produk digital tersebut, atau pun untuk pengguna akhir.

Dasbor dibuat selalu dari awal tanpa menggunakan solusi dari anak perusahaan lain sehingga hal ini menimbulkan pengeluaran untuk pembuatan layanan hal yang sama, pada beberapa kasus kualitas dari dasbor kurang baik karena tidak mempertimbangkan skalabilitas dan keamanan. Jadi bila suatu saat produk digital tersebut memiliki jumlah pengguna yang banyak maka layanan dasbor tersebut harus dibangun ulang (revamp) agar memenuhi aspek skalabilitas dan keamanan (Singh, Raj, and Ravichandra 2022).

Dalam upaya untuk mengatasi tantangan ini, penggunaan arsitektur *microservice* telah menjadi pendekatan solusi (Blinowski, Ojdowska, and Przybyłek 2022); (Tapia et al. 2020). Keuntungan utama dari pendekatan ini terletak pada kemampuannya untuk meningkatkan skalabilitas sistem(Pratama and Susetyo 2024); (Sabila, Rahayu, and Sumarni 2024).

Meskipun arsitektur *microservice* membawa sejumlah keuntungan yang signifikan, terdapat pula tantangan yang muncul seiring dengan penerapannya (Flora 2020). Penelitian terkait Singh, Raj, and Ravichandra (2022) telah menyoroti kompleksitas dan isu keamanan yang dapat muncul dalam manajemen aplikasi yang terdiri dari banyak layanan dengan arsitektur mikroservice. Oleh karena itu, dalam mengadopsi arsitektur *microservice*, perlu memperhatikan berbagai aspek ini untuk memastikan keberlanjutan dan keamanan sistem (Laigner et al. 2021).

Bersamaan dengan kebutuhan akan skalabilitas, aspek keamanan juga menjadi faktor kritis dalam pengelolaan dasbor, terutama jika melibatkan informasi sensitif dan identifikasi pribadi. Dalam lingkungan yang terdistribusi seperti arsitektur *microservice*, tantangan baru muncul terkait dengan manajemen pengguna dan pengendalian akses yang perlu diatasi dengan cermat (Yarygina and Bagge 2018); (Hannousse and Yahiouche 2021).

Penelitian ini merespon kompleksitas ini dengan mengarahkan fokusnya pada pengembangan dasbor yang tidak hanya memiliki tingkat skalabilitas yang tinggi, tetapi juga menjaga keamanan data secara optimal. Dalam konteks ini, aspek-aspek tertentu seperti manajemen pengguna dan implementasi pengendalian akses berbasis peran atau *Role Based Access Control* (RBAC) (Ferraiolo & Kuhn, 2009) menjadi pusat perhatian (Raj and Sinha 2015). Oleh karena itu, pemahaman mendalam tentang bagaimana merancang dan mengimplementasikan dasbor yang terstandarisasi, aman, dan efisien dalam arsitektur mikroservice menjadi krusial untuk memenuhi tantangan ini (Khalid et al. 2020).

Melalui penelitian ini menggambarkan proses desain, pengembangan, dan implementasi standar dasbor dalam lingkungan arsitektur *microservice*. Hasil penelitian ini memberikan panduan praktis bagi organisasi yang mengadopsi arsitektur microservice untuk mengimplementasikan dasbor yang terstandarisasi, aman, dan efisien. Kesimpulan penelitian ini memberikan wawasan berharga tentang praktik terbaik dalam standarisasi dasbor untuk pengelolaan mikro layanan di era teknologi informasi yang terus berkembang.

Peningkatan kompleksitas aplikasi modern telah mendorong penggunaan arsitektur *Microservices* sebagai pendekatan untuk mengatasi tantangan pengembangan perangkat lunak (Mateus-Coelho, Cruz-Cunha, and Ferreira 2021). Namun, dengan keuntungan-keuntungan tersebut juga datang tantangan-tantangan baru terkait keamanan dan efisiensi. Sejumlah penelitian telah dilakukan untuk

e-ISSN2774-5155
p-ISSN 2774-5147

menginvestigasi solusi-solusi yang memperkuat keamanan dan meningkatkan kinerja sistem berbasis *Microservices* (Kalubowila et al. 2021). Rangkuman berikut memberikan gambaran luas tentang penelitian-penelitian terkait yang membahas berbagai aspek keamanan, performa, dan aplikasi Microservices dalam berbagai konteks, mulai dari *cloud computing* hingga analisis data pertanian berikut in:

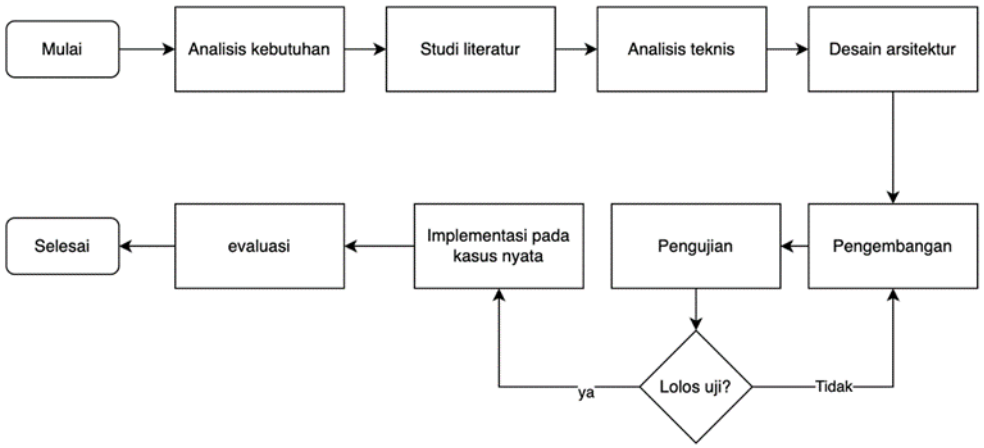
Penelitian flora dengan judul *Improving the Security of Microservice Systems by Detecting and Tolerating Intrusions* menyatakan bahwa Keamanan *microservice*: Artikel ini membahas peningkatan adopsi arsitektur layanan mikro dan masalah keamanan yang menyertainya, khususnya dalam sistem yang penting bagi bisnis dan penelitian Hannousse and Yahiouche (2021) dengan judul *Securing Microservices and Microservice Architectures: A Systematic Mapping Study* menyatakan bahwa Akses tanpa izin, kebocoran data sensitif, dan berkompromi pada individual microsevice yaitu layanan yang dapat terpengaruh atau disusupi, sehingga dapat berdampak pada keseluruhan sistem merupakan ancaman yang paling banyak diperhatikan dan diatasi menurut studi kontemporer. Hasil penelitian juga mengungkapkan bahwa audit, pengendalian akses, dan solusi berbasis pencegahan adalah mekanisme keamanan yang paling banyak diusulkan.

Tujuan penelitian ini adalah membuat dasbor yang dapat digunakan kembali, membuat dasbor yang memperhatikan keamanan, membuat dasbor yang memiliki tingkat skalabilitas tinggi. Dengan diadakannya penelitian ini, manfaat yang didapatkan dijabarkan bahwa penelitian ini tidak hanya menyajikan teori, tetapi juga memberikan panduan praktis bagi organisasi yang mengadopsi arsitektur microservice. Panduan ini mencakup desain, pengembangan, dan implementasi standar dasbor yang terintegrasi dan efisien. Ini bermanfaat bagi praktisi di lapangan yang ingin meningkatkan manajemen aplikasi mereka. Penelitian ini menekankan pentingnya keamanan data pribadi yang sensitif. Hasil penelitian ini memberikan wawasan tentang strategi keamanan yang dapat membantu melindungi data pribadi dalam lingkungan *microservice*. Melalui studi kasus, penelitian ini menguji dampak dari standar dasbor terhadap efisiensi pengelolaan sistem. Hal ini berpotensi membantu organisasi meningkatkan efisiensi operasional mereka, menghemat waktu dan sumber daya.

## METODE PENELITIAN

Pada Penelitian ini kami menjalani serangkaian tahapan yang terinci untuk memastikan bahwa setiap aspek dari perancangan dasbor ini diperhatikan dengan cermat. Tahapan-tahapan tersebut meliputi analisis kebutuhan, studi literatur, analisis teknis, desain arsitektur, pengembangan, pengujian, implementasi pada kasus nyata, dan evaluasi. Setiap tahapan ini akan membimbing kami dalam menyusun dasbor yang tidak hanya memenuhi kebutuhan bisnis PT. XYZ, tetapi juga mengikuti praktik terbaik dalam keamanan, skalabilitas, dan *reuseability* dalam konteks penggunaan *microservices*.

e-ISSN2774-5155
p-ISSN 2774-5147




**Gambar 1. Tahapan penelitian**

Penelitian ini melibatkan 9 tahapan yang terstruktur dengan baik. Tahap pertama dimulai dengan menetapkan tujuan, membentuk tim, dan mengidentifikasi sumber daya yang diperlukan. Dilanjutkan dengan analisis kebutuhan, studi literatur, dan analisis teknis yang mendalam. Desain arsitektur dan pengembangan menjadi tahapan berikutnya sebelum masuk ke pengujian dan implementasi pada kasus nyata. Tahap terakhir adalah evaluasi yang mencakup penilaian efektivitas dasbor, identifikasi area perbaikan, dan dokumentasi temuan serta rekomendasi. penelitian ini mencakup seluruh proses dari konsepsi hingga evaluasi yang komprehensif.

Activity diagram user registration menggambarkan proses pendaftaran pengguna pada aplikasi, dimulai dengan pengisian alamat email, dilanjutkan dengan verifikasi email, dan pengisian informasi penting seperti nama, nomor telepon, dan password. Proses ini dirancang untuk memastikan pengguna memberikan informasi dengan mudah. User login meminta pengguna untuk memasukkan email dan password, dengan validasi kredensial sebelum akses ke halaman utama. Proses assign role dan assign resource memberikan peran dan sumber daya kepada pengguna sesuai dengan kebutuhan dan hak akses. Forgot password memungkinkan pengguna untuk mereset password melalui email verifikasi. Ini semua merupakan langkah-langkah dalam mengelola akses dan informasi pengguna dengan efisien.

## HASIL DAN PEMBAHASAN

### Keamanan *(Secure)*

Penerapan arsitektur microservices dalam perancangan dasbor PT. XYZ telah memberikan hasil yang signifikan dalam aspek keamanan sistem. Dengan memisahkan fungsionalitas ke dalam layanan-layanan kecil yang independen, kami dapat mengisolasi setiap bagian sistem secara lebih efektif. Hal ini meminimalkan risiko serangan terhadap seluruh sistem karena kerentanan pada satu layanan tidak akan secara otomatis mengancam keseluruhan infrastruktur (Krämer, Frese, and Kuijper 2019).

Selain itu, penggunaan teknologi-teknologi keamanan terkini seperti otentikasi berbasis token dan enkripsi data dalam komunikasi antar layanan telah menjaga keamanan informasi yang diproses oleh dasbor (Yu et al. 2019). Penggunaan mekanisme otentikasi yang kuat dan manajemen hak akses yang terpusat juga meningkatkan lapisan keamanan dalam aplikasi.

Dari 10 daftar teratas celah keamanan OWASP ada 6 point yang dapat diimplementasikan untuk mengamankan dasbor ini berikut penjelasannya

### Injeksi SQL *(SQL Injection)*

Injeksi SQL adalah jenis serangan keamanan yang memanfaatkan celah pada aplikasi yang berinteraksi dengan basis data menggunakan SQL (*Structured Query Language*) (Pardosi et al. 2024). Serangan ini terjadi ketika seorang penyerang menyisipkan atau "menyuntikkan" kode SQL berbahaya

e-ISSN2774-5155
p-ISSN 2774-5147

ke dalam input pengguna yang kemudian dieksekusi oleh aplikasi. Teknik ini dapat digunakan untuk melewati mekanisme otentikasi, mengakses, memodifikasi, atau menghapus data tanpa izin. Injeksi SQL biasanya terjadi karena kurangnya validasi input dan penggunaan *query* SQL dinamis yang rentan terhadap manipulasi.

Dampak dari injeksi SQL bisa sangat merugikan, mulai dari pencurian data pribadi hingga kerusakan total basis data. Serangan ini sering digunakan untuk mendapatkan akses tidak sah ke informasi sensitif seperti *detail login*, nomor kartu kredit, dan data penting lainnya. Untuk mencegah injeksi SQL, penting bagi pengembang aplikasi untuk menerapkan praktik keamanan yang baik seperti *parameterized query*, penggunaan ORM (*Object-Relational Mapping*), dan sanitasi input yang tepat. Langkah-langkah ini membantu memastikan bahwa input dari pengguna tidak bisa digunakan untuk memanipulasi perintah SQL yang dijalankan oleh aplikasi (Alshuqayran, Ali, and Evans 2016).

**1. Implementasi *Parameterized Query* untuk Mencegah Injeksi SQL.**

*Parameterized query* adalah teknik dalam pemrograman database yang digunakan untuk meningkatkan keamanan dan mencegah serangan injeksi SQL dengan memisahkan kode SQL dari data input pengguna. Dalam *parameterized query*, perintah SQL ditulis dengan *placeholder* untuk nilai parameter, dan nilai tersebut diberikan secara terpisah saat query dieksekusi. Ini memastikan bahwa data input tidak dapat mengubah struktur perintah SQL yang telah ditentukan. Dengan memisahkan data dan kode, *parameterized query* menjamin bahwa input pengguna diperlakukan sebagai data biasa, bukan sebagai bagian dari perintah SQL, sehingga menjaga integritas dan keamanan database.

Implementasi *paramerized query* dilakukan dengan dua cara, cara pertama, pada query langsung diberi placeholder. Sedangkan cara kedua menggunakan fungsi "*processArrayAndJoin*". Fungsi *processArrayAndJoin* ini bertujuan untuk mengambil nilai dari objek data berdasarkan kunci *key*, mengolahnya menjadi *array*, dan membuat sebuah *string query* yang sesuai dengan panjang *array* tersebut. Ini sangat berguna dalam konteks persiapan *query* SQL dinamis dimana *placeholder* (?) diperlukan untuk setiap elemen dalam *array*.

**2. Validasi Input Pengguna secara Ketat untuk Mencegah Karakter Berbahaya.**

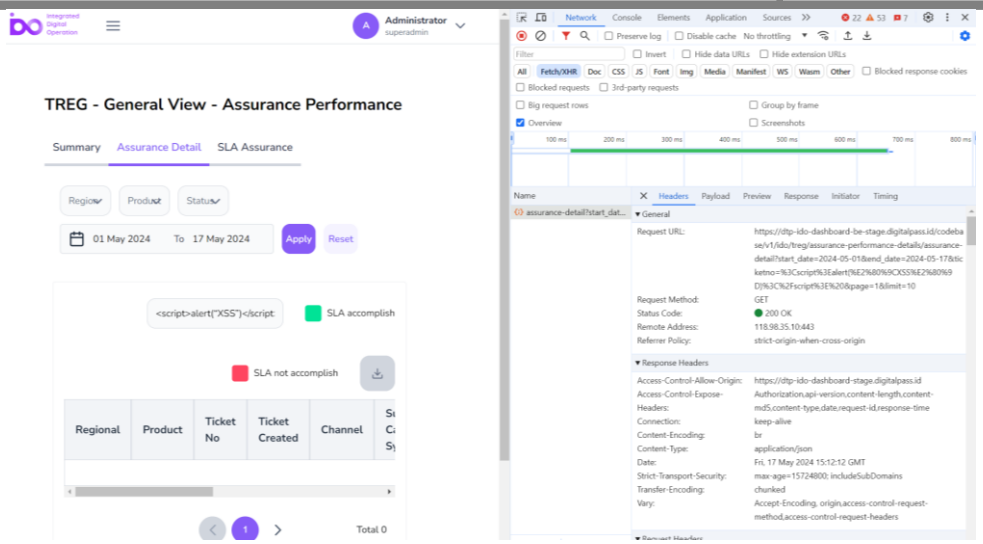
Validasi input adalah proses memeriksa data yang dimasukkan oleh pengguna untuk memastikan bahwa data tersebut sesuai dengan kriteria atau aturan tertentu sebelum diproses lebih lanjut. Tujuannya adalah untuk memastikan bahwa input aman, akurat, dan dalam format yang diharapkan, sehingga mencegah kesalahan atau potensi serangan keamanan seperti SQL *Injection*.

Pada implementasinya, digunakan *library joi* untuk melakukan validasi. Contohnya untuk variable product, dilakukan validasi untuk memastikan input dari user berupa string dan hanya dapat menggunakan huruf saja. Sebagai contoh, diberikan parameter product dengan value "Antares" dan API memberikan response 200. Sedangkan pada parameter *product* diberi nilai (') mendapatkan *response false* atau gagal.

***Cross-Site Scripting* (XSS)**

*Cross Site Scripting* (XSS) adalah jenis serangan keamanan pada aplikasi web dimana penyerang memasukan skrip berbahaya ke dalam konten atau input yang dilihat oleh pengguna. Skrip berbahaya ini digunakan untuk mencuri informasi sensitif, seperti cookie, atau mengubah tampilan halaman web. XSS biasanya terjadi ketika aplikasi web tidak memvalidasi atau menyaring input pengguna dengan benar sebelum menampilkannya kembali kepada pengguna.

e-ISSN2774-5155
p-ISSN 2774-5147




**Gambar 2. Percobaan Cross Site Scripting**

Pada Gambar 6 Percobaan *Cross Site Scripting*, *scripting* dilakukan pada dasbor untuk memastikan keamanannya. Pengujian ini melibatkan memasukkan *payload* skrip berbahaya ke dalam input pengguna dan memantau apakah skrip tersebut dieksekusi dalam *konteks browser*. Skrip yang diberikan pada contoh diatas adalah skrip alert yang akan menampilkan pop up apabila dasbor tidak mencegah XSS. Selain itu juga bisa dilakukan percobaan menggunakan *skrip image* dan *iframe* yang biasa digunakan dalam percobaan XSS. Skrip berbahaya lainnya bisa saja berupa skrip yang mencoba mengirim token login ke server lain, yang dapat mengakibatkan pencurian identitas atau akses tidak sah ke akun pengguna. Hasil pengujian menunjukkan bahwa browser tidak mengeksekusi skrip tersebut, menandakan bahwa mekanisme sanitasi input telah diterapkan dengan baik.

Dasbor dikembangkan menggunakan *framework React* pada bagian *Front End* (Aitlmoudden et al. 2023). Salah satu mekanisme utama yang digunakan oleh React untuk melindungi aplikasi web dari XSS adalah melakukan mitigasi berupa escaping otomatis pada nilai-nilai yang dimasukkan ke dalam JSX. JSX adalah ekstensi sintaks yang memungkinkan penggunaan sintaks serupa HTML dalam Javascript (Sulistyorini, Sova, and Ramadhan 2022). Saat nilai-nilai ditampilkan di dalam JSX, React akan otomatis mengonversi karakter-karakter yang berpotensi berbahaya menjadi entitas HTML yang aman, sebagai contoh:

a. < menjadi <
b. > menjadi >
c. " menjadi "
d. ' menjadi '
e. / menjadi /

Ketika pengguna memasukkan nilai-nilai tersebut, *React* DOM secara otomatis melakukan escaping pada setiap nilai yang dimasukkan sebelum melakukan rendering. Proses escaping ini memastikan bahwa karakter-karakter khusus diubah menjadi sintaks HTML yang aman. Dengan demikian, setiap skrip berbahaya yang dimasukkan oleh pengguna akan ditampilkan sebagai teks biasa dan tidak akan dieksekusi oleh browser. Hal ini secara efektif mencegah serangan XSS yang sering memanfaatkan kelemahan dalam penanganan input pengguna.

***Broken Authentication***

*Broken Authentication* adalah kerentanan keamanan di mana mekanisme otentikasi suatu aplikasi tidak memadai, memungkinkan penyerang untuk mendapatkan akses tidak sah ke akun pengguna (Sya'bani 2023). Masalah ini sering terjadi karena implementasi otentikasi yang salah atau tidak aman, seperti penggunaan kata sandi yang lemah, tidak adanya penguncian akun setelah sejumlah upaya login yang gagal, atau pengelolaan sesi yang tidak aman. Untuk mencegahnya dilakukan upaya penceegahan seperti berikut:

e-ISSN2774-5155
p-ISSN 2774-5147

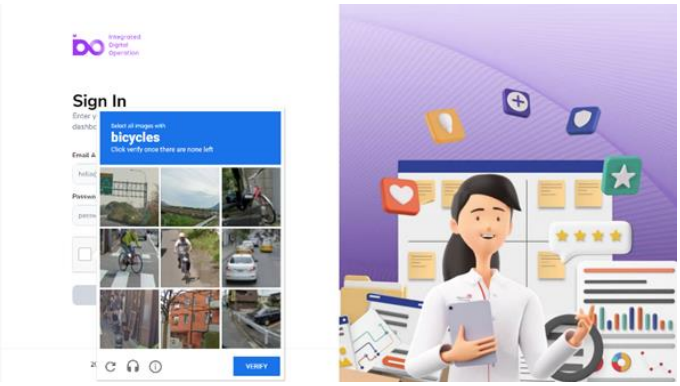
### 1. Kebijakan Kata Sandi Kuat

Kebijakan kata sandi kuat adalah langkah penting dalam keamanan siber yang memastikan setiap pengguna membuat dan menggunakan kata sandi yang sulit ditebak dan cukup kompleks untuk mencegah serangan *brute force* dan akses tidak sah. Kebijakan ini biasanya mencakup persyaratan seperti panjang minimal kata sandi, kombinasi huruf besar dan kecil, angka, serta karakter khusus. Dengan menerapkan kebijakan kata sandi yang kuat, organisasi dapat meningkatkan perlindungan terhadap data sensitif dan mengurangi risiko kebocoran informasi akibat kata sandi yang lemah. Pada Gambar 4.7, ketika pengguna mendaftarkan kata sandinya, sistem akan memeriksa apakah kata sandi tersebut sudah sesuai dengan standar yang telah ditetapkan. Standar ini meliputi panjang minimal kata sandi yaitu 8 karakter, serta harus mengandung huruf kapital, huruf kecil, angka, dan simbol.

```
const register = joi.object({
  email: joi.string().email().required(),
  name: joi.string().required(),
  password: joi.string().required().regex(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$/),
  phone: joi.string().required()
    .pattern(/^(0|\+62)\d{9,}$/)
    .message('Phone number must start with "0" or "+62" and have at least 10 digits' ),
});
```

**Gambar 3. Validasi Password Pengguna**

### 2. Penggunaan Captcha



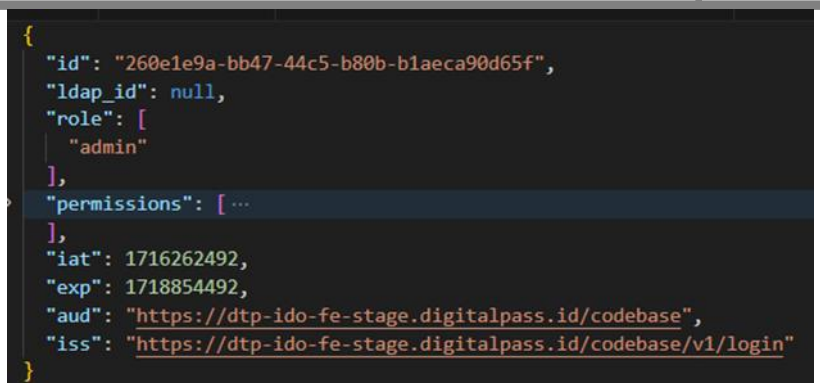

**Gambar 4. *Captcha* pada Dasbor**

*Captcha* adalah mekanisme keamanan yang digunakan untuk membedakan antara pengguna manusia dan bot otomatis (Wandapranata and Hansun 2017). Seperti yang bisa dilihat pada Gambar 8. Captcha pada dasbor biasanya menampilkan tugas sederhana yang mudah dilakukan oleh manusia tetapi sulit untuk dipecahkan oleh komputer, seperti mengenali teks dalam gambar yang terdistorsi, mengidentifikasi objek dalam gambar, atau menyelesaikan puzzle sederhana. Tujuan utama captcha adalah untuk mencegah penyalahgunaan layanan online oleh bot, seperti spam, *brute force attacks*, dan pencurian data.

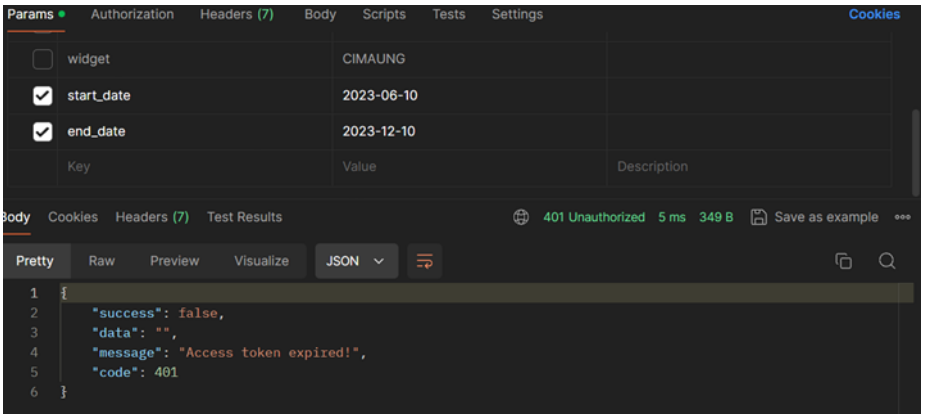
### 3. Penggunaan JWT untuk Meningkatkan Keamanan Otentikasi

JWT (JSON *Web Token*) adalah standar terbuka yang digunakan untuk berbagi informasi aman antara klien dan server dalam bentuk token (Pudoli, Yulianawati, and Suwandi 2023). JWT terdiri dari tiga bagian: *header*, *payload*, dan *signature*. *Header* berisi informasi tentang algoritma enkripsi yang digunakan, payload berisi klaim atau data yang ingin dibagikan, dan signature adalah hasil enkripsi dari *header* dan *payload* yang memastikan token tidak dapat diubah tanpa deteksi. JWT sering digunakan dalam otentikasi dan otorisasi untuk memastikan identitas pengguna dan mengamankan komunikasi antara aplikasi web dan server tanpa perlu menyimpan sesi di server. Pada Gambar 9 dapat dilihat role dan permission yang dimiliki oleh user. Dengan begitu akses yang dimiliki pengguna dapat dibatasin. Selain itu token juga dapat *expired* untuk mengurangi kemungkinan token di gunakan oleh pihak lain. Seperti pada Gambar 10, ketika token yang dimiliki pengguna expired, maka pengguna tidak dapat mengakses API tersebut lagi.

e-ISSN2774-5155
p-ISSN 2774-5147




**Gambar 5. Payload dari JWT Token**




**Gambar 6. Contoh Response Expired Token**

### *4. Rate Limiter*

*Rate limiter* adalah mekanisme yang digunakan untuk mengontrol jumlah permintaan yang dapat dilakukan oleh klien ke server dalam jangka waktu tertentu (Setiawan 2018). Dengan menetapkan batas maksimum permintaan dalam periode waktu yang ditentukan, *rate limiter* membantu mencegah penyalahgunaan seperti serangan *Denial of Service* (DoS) dan brute force, serta memastikan penggunaan sumber daya yang adil dan stabilitas aplikasi. Ketika batas tercapai, permintaan tambahan dapat dibatasi atau ditolak, sering kali dengan mengirimkan respons seperti HTTP 429. Gambar 7 menujukan contoh konfigurasi *rate limiter* dengan menggunakan plugins yang telah di sediakan oleh *restify*. pada konfigurasi ini diperlukan 3 parameter yaitu burst, rate¸ dan IP. Burst menentukan jumlah maksimum permintaan yang diizinkan dalam jangka waktu yang sangat singkat (burst window) sebelum rate limiting yang lebih ketat diterapkan. Rate menentukan jumlah permintaan yang diizinkan per detik secara berkelanjutan. Setelah batas burst tercapai, permintaan akan diatur berdasarkan rate ini. IP menentukan apakah pembatasan harus diterapkan berdasarkan alamat IP klien. Jika true, maka rate limiting akan diterapkan per alamat IP.

```
const throttleParams = Object.freeze({ burst: 30, rate: 10, ip: true });
this.server.post('/codebase/v1/user/request-token', asyncMiddleware(basicAuth.isAuthenticated),
  restify.plugins.throttle(throttleParams), userHandler.requestToken);
```

**Gambar 7. Konfigurasi Rate Limiter**

### *Broken Authorization*

*Broken authorization* adalah kerentanan keamanan di mana mekanisme otorisasi aplikasi tidak berfungsi dengan benar, memungkinkan pengguna mendapatkan akses ke data atau tindakan yang seharusnya dibatasi. Masalah ini sering muncul ketika aplikasi gagal memverifikasi apakah pengguna memiliki izin yang cukup untuk melakukan tindakan tertentu. Contoh umum termasuk pengguna biasa yang dapat mengakses data administratif, atau pengguna yang dapat mengubah data milik pengguna lain. Kerentanan ini dapat muncul akibat pengaturan izin yang tidak benar, logika otorisasi yang lemah, atau kegagalan untuk memvalidasi kontrol akses secara konsisten di seluruh aplikasi (Cobb 2023).

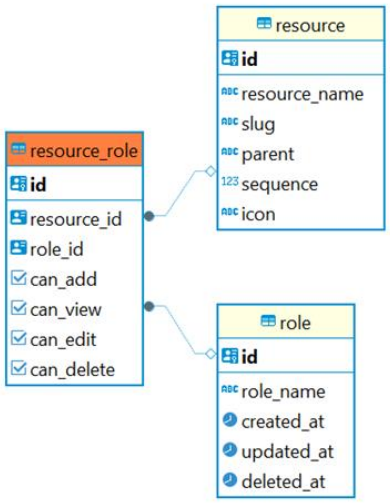
e-ISSN2774-5155
p-ISSN 2774-5147

Dampak dari broken authorization bisa sangat merugikan, termasuk kebocoran data pribadi, manipulasi data sensitif, dan kerusakan integritas sistem. Misalnya, dalam sebuah aplikasi perbankan, kegagalan otorisasi bisa memungkinkan seorang pengguna biasa melihat atau mengubah saldo rekening orang lain. Untuk mencegah broken authorization, penting untuk menerapkan kontrol akses yang ketat dan terpusat, memastikan bahwa setiap permintaan diverifikasi untuk izin yang tepat sebelum diproses. Pengujian keamanan rutin, termasuk pengujian penetrasi dan audit kode, juga penting untuk mengidentifikasi dan memperbaiki kerentanan otorisasi.

**1. Kontrol Akses Berbasis Peran (RBAC)**

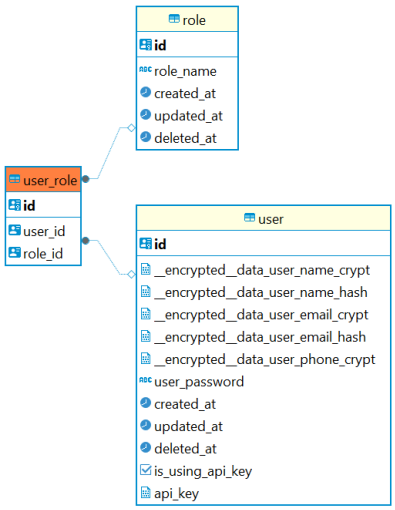
Implementasi Kontrol Akses Berbasis Peran (*Role-Based Access Control*/RBAC) pada Gambar 12 ER Diagram *Resource Role* dan pada Gambar 13 ER Diagram User Role menghasilkan sejumlah manfaat signifikan dalam pengelolaan akses dan keamanan sistem. Hasil utama dari implementasi RBAC ialah peningkatan keamanan, RBAC membatasi akses pengguna berdasarkan peran mereka dalam organisasi, yang mengurangi risiko akses tidak sah dan pelanggaran keamanan contohnya prinsip least privilege, pengguna hanya mendapatkan akses yang diperlukan untuk menjalankan tugas mereka, mengurangi kemungkinan penyalahgunaan akses. Contoh lainnya pencegahan kesalahan manusia, dengan peran yang telah ditentukan, kemungkinan kesalahan dalam pemberian akses secara manual berkurang.

RBAC mempermudah pengelolaan hak akses dengan mengelompokkan izin ke dalam peran, yang kemudian dapat dengan mudah diberikan kepada pengguna. Manajemen Terpusat: Administrator dapat mengelola akses dari satu tempat tanpa harus mengubah izin individu setiap kali ada perubahan tugas atau posisi. Penambahan dan Penghapusan Pengguna yang Mudah, menambah atau menghapus pengguna cukup dengan memberikan atau mencabut peran mereka.




**Gambar 8. ER Diagram *Resource Role***

e-ISSN2774-5155
p-ISSN 2774-5147



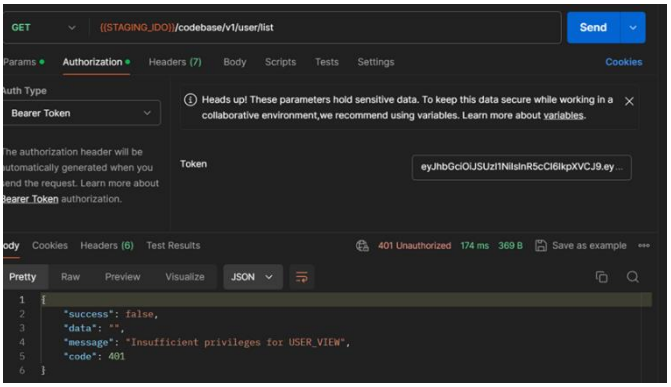

**Gambar 9. ER Diagram User Role**

## 2. Validasi Izin pada Setiap Permintaan

Pada setiap API yang telah dibuat, akan diberikan *middleware* untuk melihat apakah pengguna yang meakses api tersebut memiliki izin yang dibutuhkan. Implementasi yang dilakukan seperti pada Gambar 14. setiap API dibagi dalam 2 jenis, yaitu berdasarkan resourcenya dan jenis aksesnya. Jika pengguna tidak memiliki izin, maka Ketika mengakses API tersebut akan ditolak seperti pada Gambar 15.

```
server.get('/codebase/v1/resource/:id', asyncMiddleware(jwtAuth.verifyToken),
  roleHelper.checkRole('RESOURCE_VIEW'), resourceHandler.getResource);
server.get('/codebase/v1/resource/list', asyncMiddleware(jwtAuth.verifyToken),
  roleHelper.checkRole('RESOURCE_VIEW'), resourceHandler.listResource);
server.post('/codebase/v1/resource', asyncMiddleware(jwtAuth.verifyToken),
  roleHelper.checkRole('RESOURCE_ADD'), resourceHandler.createResource);
server.put('/codebase/v1/resource', asyncMiddleware(jwtAuth.verifyToken),
  roleHelper.checkRole('RESOURCE_EDIT'), resourceHandler.updateResource);
server.del('/codebase/v1/resource/:id', asyncMiddleware(jwtAuth.verifyToken),
  roleHelper.checkRole('RESOURCE_DELETE'), resourceHandler.deleteResource);
```

**Gambar 10. Implementasi Validasi Izin pada setiap API**




**Gambar 11. *Response* Jika Pengguna Tidak Memiliki Izin**

## Sensitive Data Exposure

OWASP *Sensitive* Data *Exposure* mengacu pada kelemahan dalam perlindungan data sensitif dari akses atau penyadapan yang tidak sah. Data sensitif ini bisa mencakup informasi pribadi (PII), informasi kesehatan yang dilindungi (PHI), data keuangan, kredensial login, dan data penting lainnya. Kelemahan ini bisa terjadi karena kurangnya enkripsi, manajemen kunci yang buruk, atau konfigurasi yang tidak aman.

e-ISSN2774-5155
p-ISSN 2774-5147

**Penggunaan Enkripsi yang Kuat**

Hasil implementasi algoritma enkripsi AES-GCM, hash SHA-256, dan penggunaan pepper menghasilkan peningkatan yang signifikan dalam hal keamanan data. Berikut adalah ringkasan manfaat dari implementasi ini:

**1. Keamanan Data yang Tinggi**

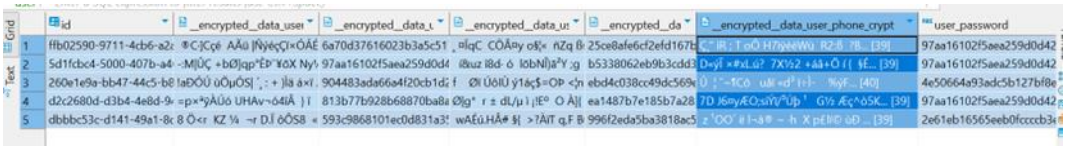
AES-GCM (Advanced Encryption Standard in Galois/Counter Mode) menyediakan enkripsi yang kuat dengan autentikasi bawaan, memastikan bahwa data yang dienkripsi tidak hanya dirahasiakan tetapi juga integritasnya terjaga Enkripsi dan Integritas: AES-GCM memastikan bahwa data terenkripsi tidak dapat diubah tanpa deteksi, memberikan keamanan yang kuat untuk data sensitif.

**2. Hashing yang Aman dengan SHA-256 dan Pepper**

SHA-256 menghasilkan hash yang kuat dan tahan terhadap serangan *brute-force*. Dengan menambahkan pepper (nilai rahasia tambahan yang disimpan terpisah dari data yang di-hash), keamanan hash ditingkatkan diantaranya hash yang kuat: SHA-256 menghasilkan hash yang sulit dipatahkan, membuatnya ideal untuk menyimpan *password* dan data sensitif lainnya dengan keamanan tambahan dengan *pepper*: Penggunaan pepper membuat hash lebih tahan terhadap serangan yang mencoba menggunakan tabel precomputed seperti *rainbow tables*.

**3. Perlindungan dari Serangan**

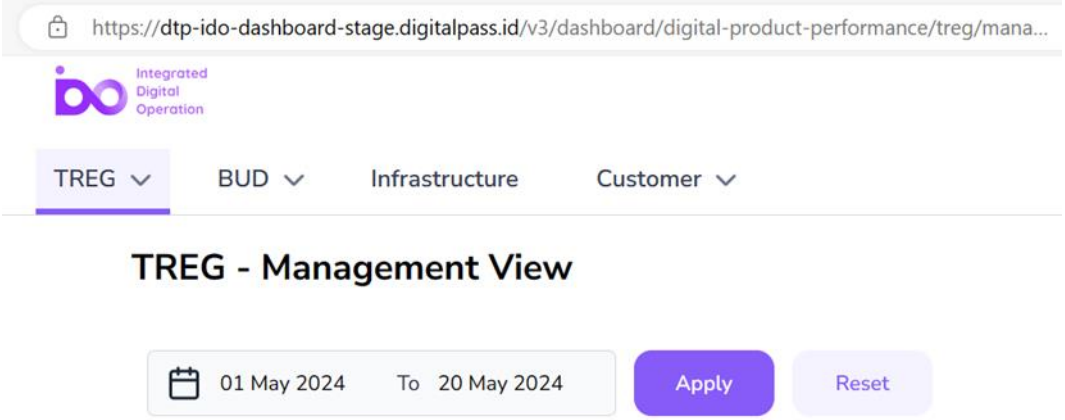
Kombinasi AES-GCM dan SHA-256 dengan pepper memberikan perlindungan yang komprehensif terhadap berbagai jenis serangan diantaranya perlindungan terhadap Serangan *Brute-force* dan Rainbow *Table*: SHA-256 dan pepper membuat hash sulit untuk diprediksi atau dibalik. Autentikasi Data dan Non-Repudiation: AES-GCM menyediakan mekanisme autentikasi yang memastikan data tidak dapat dimodifikasi tanpa deteksi. Impelementasi *encryption* diilustrasikan pada




**Gambar 12. Implementasi encrypt dan hash data.**

**Implementasi HTTPS**

Banyak browser modern memberikan indikator visual yang jelas ketika situs tidak aman, seperti Peringatan Tidak Aman, Situs yang tidak menggunakan HTTPS mungkin menampilkan peringatan "Tidak Aman" di bilah alamat, yang bisa mengurangi kepercayaan dan interaksi pengguna. Secara keseluruhan, implementasi HTTPS memberikan banyak manfaat yang signifikan bagi keamanan, kepercayaan, dan kinerja situs web. Dengan mengadopsi HTTPS seperti diilustrasikan pada Gambar 17 Implementasi HTTPS, pemilik situs web dapat memastikan bahwa data pengguna terlindungi dan memberikan pengalaman yang lebih baik bagi pengunjung.

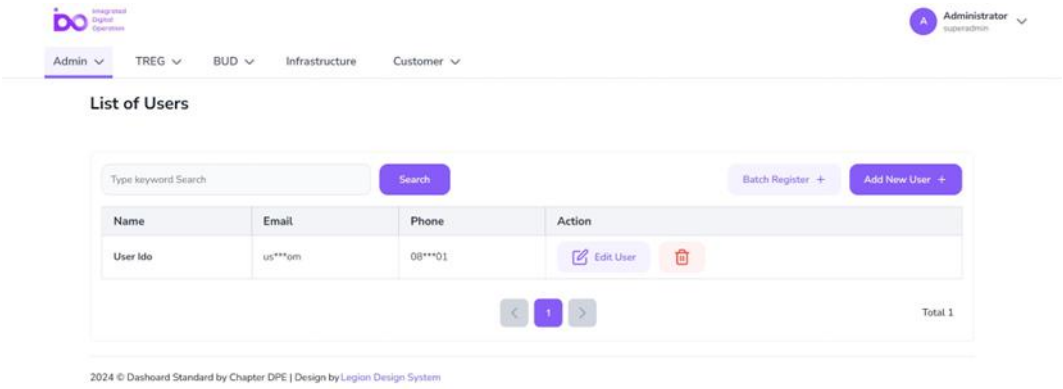



**Gambar 13. Implementasi HTTPS**

e-ISSN2774-5155
p-ISSN 2774-5147

***Masking* data**

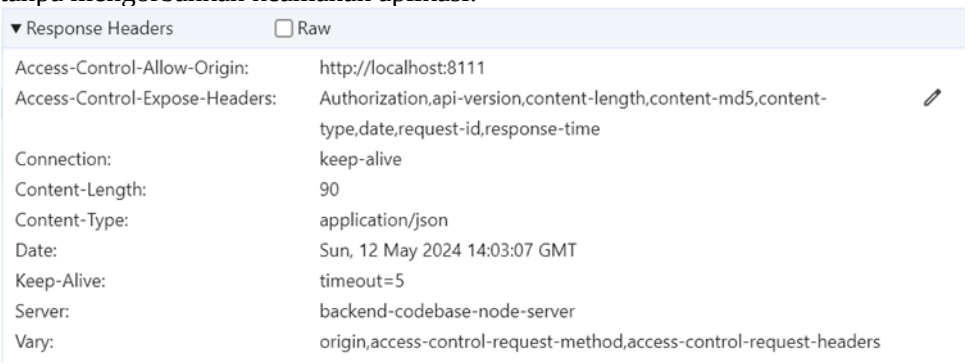
Hasil implementasi data masking menunjukkan peningkatan signifikan dalam keamanan dan privasi data pengguna. Email disamarkan sehingga hanya dapat dilihat oleh pengguna yang melakukan tindakan eksplisit untuk membukanya, melindungi data sensitif dari akses yang tidak sah. Pendekatan ini membantu memenuhi persyaratan kepatuhan terhadap privasi data, memastikan bahwa data email tetap terlindungi dan hanya dapat diakses secara terkontrol, sehingga meningkatkan keamanan dan privasi data pengguna secara keseluruhan. Masking data diilustrasikan pada Gambar 18 *Masking* data email pada daftar halaman user.




**Gambar 14. *Masking* data email pada daftar halaman user**

***Cross Origin Resource Sharing* (CORS)**

Implementasi CORS berperan penting dalam memastikan akses yang aman dan terkontrol dari domain yang berbeda. CORS memungkinkan dasbor untuk mengambil dan menampilkan data secara real-time tanpa batasan akses lintas domain. Dengan mengaktifkan CORS, dasbor dapat mengakses data dengan aman dari sumber daya yang berbeda, memastikan ketersediaan informasi yang akurat dan terbaru kepada pengguna. Dasbor dapat menyajikan informasi yang lebih lengkap dan berguna kepada pengguna tanpa mengorbankan keamanan aplikasi.




**Gambar 15. *Response Header* pada Jaringan *Browser***

Pada Gambar 19 *Response Header* pada Jaringan Browser pada jaringan browser terdapat header bernama *Access-Control-Allow-Origin* dalam respon dari server pada jaringan. Dapat dikatakan bahwa dasbor telah mengimplementasikan CORS pada server backend untuk mengizinkan akses lintas domain ke sumber daya dalam dasbor. Implementasi CORS ini menjadi krusial dalam memungkinkan dashbord untuk mengambil data dari sumber eksternal secara aman dan efisien, meningkatkan fungsionalitas dan fleksibilitas aplikasi secara keseluruhan.

**Penelusuran Komponen dengan Kerentanan yang Diketahui**

Implementasi hasil scan untuk "*Components with Known Vulnerabilities*" (Komponen dengan Kerentanan yang Diketahui) menghasilkan beberapa tindakan penting yang meningkatkan keamanan

e-ISSN2774-5155
p-ISSN 2774-5147

aplikasi dan sistem. Berikut adalah hasil dan langkah-langkah utama dari implementasi ini:

**1. Identifikasi Kerentanan**

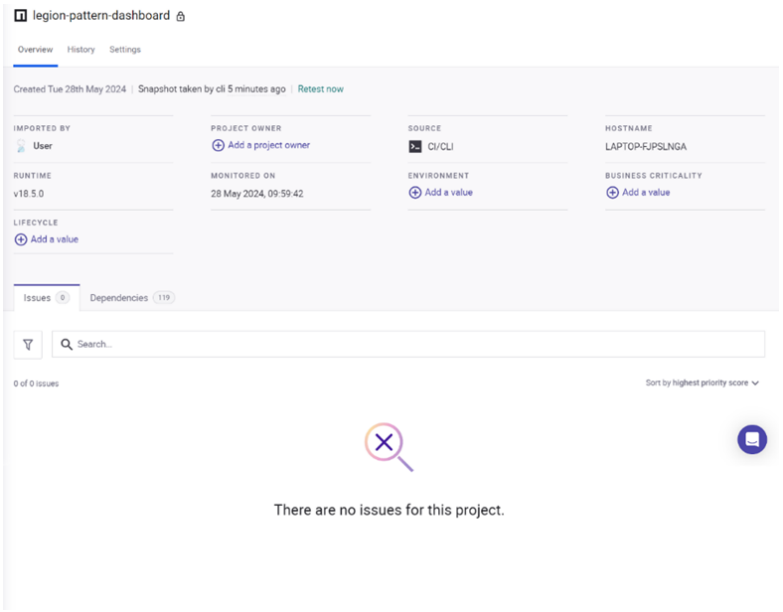
Hasil scan mengidentifikasi komponen perangkat lunak yang memiliki kerentanan keamanan yang diketahui, seperti pustaka atau framework yang digunakan dalam aplikasi Pemetaan Komponen Rentan Daftar semua komponen dan versinya yang memiliki kerentanan dikenal dari database seperti CVE (*Common Vulnerabilities and Exposures*).

**2. Penilaian Risiko**

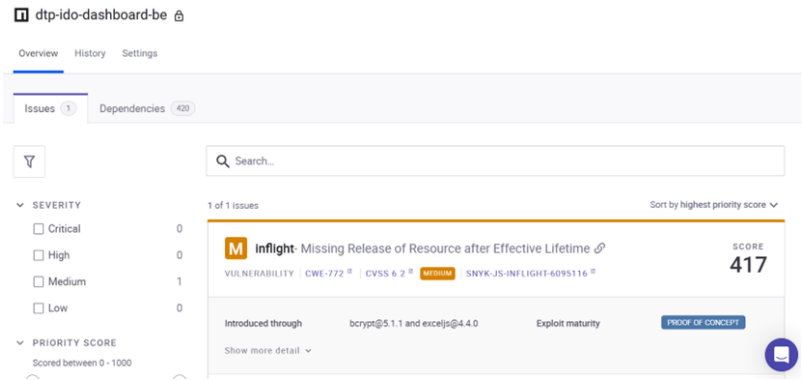
Setelah mengidentifikasi komponen yang rentan, langkah selanjutnya adalah menilai risiko yang ditimbulkan oleh kerentanan ini terhadap aplikasi Evaluasi Dampak: Menentukan dampak potensial dari kerentanan pada keamanan, kinerja, dan keandalan sistem. Prioritas Kerentanan: Mengurutkan kerentanan berdasarkan tingkat keparahan dan potensi dampaknya terhadap bisnis.

**3. Perbarui dan Patch Komponen**

Mengambil tindakan untuk memperbarui atau memperbaiki komponen yang rentan dengan versi terbaru yang sudah ditambal, Pembaruan Komponen: Mengganti komponen rentan dengan versi terbaru yang tidak memiliki kerentanan yang diketahui. Penerapan *Patch*, Menerapkan patch keamanan yang dirilis oleh pengembang atau vendor komponen. Untuk *frontend* diilustrasikan pada Gambar 20 *Front End Dependency Check* dan pada backend di Gambar 21 *Back End Dependency Check*.




**Gambar 16. Front End Dependency Check**




**Gambar 17. *Back End Dependency Check***

Pastikan untuk menggunakan komponen *backend* dan *frontend* yang terjamin keamanannya. Ini

e-ISSN2774-5155
p-ISSN 2774-5147

dapat dilakukan dengan memilih *library* dari npmjs atau menggunakan *library* internal perusahaan yang telah terbukti keamanannya. Selain itu, pastikan untuk secara berkala melakukan pembaruan komponen guna memperbaiki kerentanan yang mungkin muncul. Selalu gunakan library versi terbaru untuk memastikan bahwa Anda mendapatkan semua pembaruan keamanan yang diperlukan. Terakhir, lakukan pengujian keamanan secara berkala untuk mendeteksi dan mengatasi kerentanan yang mungkin ada dalam aplikasi Anda. Dengan mengikuti langkah-langkah ini, Anda dapat memastikan bahwa aplikasi Anda tetap aman dan terhindar dari ancaman keamanan yang mungkin timbul.

**Skalabilitas (Scalable)**

Arsitektur *microservices* telah membuktikan kemampuannya dalam mendukung skalabilitas aplikasi. Dengan membagi aplikasi menjadi sejumlah layanan yang mandiri, kami dapat dengan mudah menyesuaikan ukuran setiap layanan sesuai dengan kebutuhan beban kerja saat ini. Hal ini memungkinkan kami untuk menangani lonjakan trafik atau permintaan tanpa mengorbankan performa keseluruhan sistem.

Selain itu, penggunaan teknologi teknologi seperti Kubernetes untuk manajemen kontainer dan penskalaan otomatis telah mempermudah dalam mengelola infrastruktur secara dinamis. Dengan demikian, dasbor PT. XYZ dapat dengan mudah berkembang seiring dengan pertumbuhan perusahaan tanpa mengalami kendala dalam hal performa atau ketersediaan.

**Teknik yang ditempuh antara lain:**

a. Membagi aplikasi web menjadi microservices kecil dan independen.
b. Setiap microservice memiliki tanggung jawab spesifik dan terisolasi dari microservices lain.
c. Komunikasi antar microservice menggunakan API RESTful.
d. Implementasi Docker dan Kubernetes untuk containerisasi dan orkestrasi microservices
e. Implementasi *autoscaling* untuk secara otomatis menambahkan atau menghapus *instance microservice* berdasarkan beban.
f. Implementasi load balancing untuk mendistribusikan lalu lintas secara merata antar *instance microservice*.
g. Implementasi *caching* untuk meningkatkan performa dan mengurangi beban database.

**Load Testing**

Pengujian dilakukan secara bertahap, dimulai dari 10 hingga 100 pengguna virtual secara simultan, dengan durasi pengujian selama 180 detik, untuk menentukan rata-rata dan TPS maksimum yang dapat dicapai selama pengujian berlangsung. Dari test ini di harapkan *service* tersebut dapat menangani minimal 10 transaksi per detiknya. Perangkat lunak yang digunakan ialah Apache Jmeter 5.4.3* dengan konfigurasi perangkat keras sebagaimana ditunjukan pada Tabel 1 Speksifikasi perangkat keras. Pengujian dilakukan secara bertahap mulai dari 10 hingga 100 virtual *concurrent user* dengan rentang waktu test selama 180 detik untuk mengetahui rata-rata dan max tps yang didapatkan pada saat pengujian dijalankan. Hasil *load testing* pada Tabel 4.2 Hasil *load testing* Pada jumlah pengguna bersamaan virtual yang rendah (10 dan 25), sistem memiliki tingkat keberhasilan yang tinggi dengan tingkat kesalahan yang relatif rendah. Pada jumlah pengguna bersamaan virtual yang lebih tinggi (50), tingkat keberhasilan meningkat dan kesalahan menurun dibandingkan dengan pengujian sebelumnya. Pada jumlah pengguna bersamaan virtual yang sangat tinggi (100), sistem mengalami kegagalan yang signifikan dengan tingkat kesalahan yang sangat tinggi, terutama kesalahan 502 dan 503, menunjukkan bahwa sistem tidak mampu menangani beban yang sangat tinggi. Analisis ini menunjukkan bahwa sistem memiliki batasan pada jumlah pengguna bersamaan yang dapat ditangani dengan baik, dan kinerja menurun secara drastis pada beban yang sangat tinggi.

**Tabel 1 Speksifikasi perangkat keras**

| Lingkungan tes | Spesifikasi perangkat keras |
|---|---|
| *Test Server Local* | Windows 11 Pro CPU : *6 Cores Memory* : 32GB *Bandwith*: 100 Mbps |
| *Test Server Cloud* | Alibaba Cloud Server Distributed ( 1 Master 4 Slave) CPU : 8 *Cores Memory* : 16GB *Disk :* 60GB OS : Debian |

e-ISSN2774-5155
p-ISSN 2774-5147

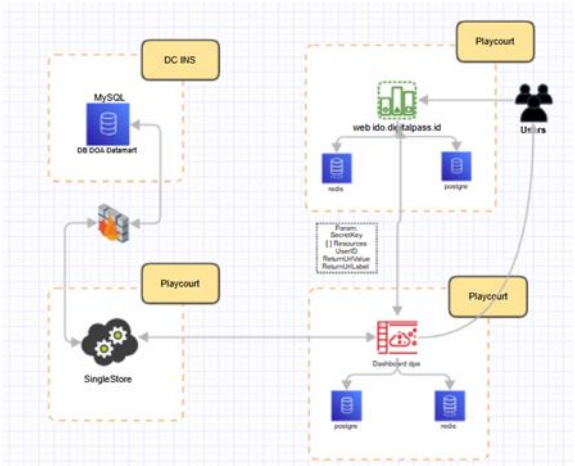
**Tabel 12 Hasil load testing**

| *Test No* | *Virtual CCU* | *Test Time (In Second)* | *Succes Rate (%)* | *Error Rate (%)* | *Max TPS* | *Total Request* | *Errors* | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | 401 | 404 | 502 | 503 |
| 1 | 10 | 180 | 96.37 | 3,63 | 40 | 6644 | 1,14 | 2,48 | 0 | 0 |
| 2 | 25 | 180 | 94,62 | 5,38 | 79 | 12243 | 1,14 | 4,25 | 0 | 0 |
| 3 | 50 | 180 | 97,71 | 2,29 | 60 | 7001 | 1,14 | 1,14 | 0 | 0 |
| 4 | 100 | 180 | 5,14 | 94,86 | 60 | 169112 | 0,09 | 0,04 | 53,69 | 41,03 |

Berdasarkan pengujian yang dilakukan, *service* yang diujikan sudah memenuhi kriteria dengan beberapa detil sebagai berikut:

1. *Service* yang diujikan sudah dapat menangani beban hingga 50 VCU dengan maksimum 70 transaksi perdetik
2. *Service* yang diujikan belum dapat menangani hingga 100 VCU karena sudah muncul error 50x (*Bad Gateway*).

**Arsitektur microservice IDO dashbor**




**Gambar 19. Arsitektur Microservice IDO Dashbor**

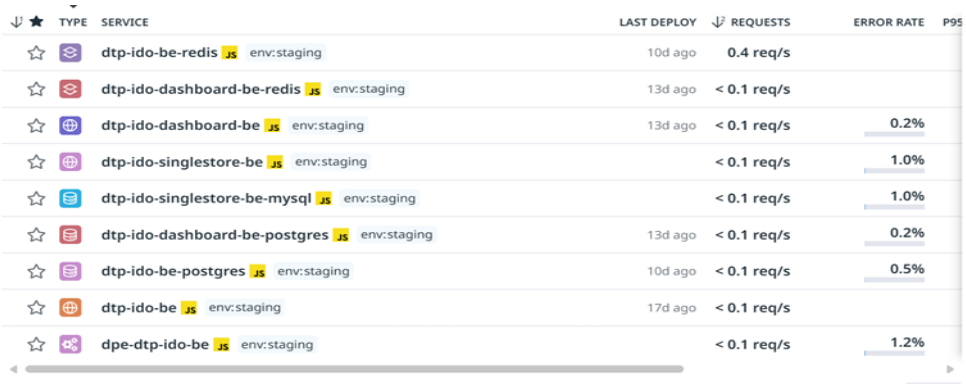
Pada Gambar 22 Arsitektur Microservice IDO Dashbor menggambarkan arsitektur sistem untuk aplikasi web yang memiliki dua komponen utama: web ido.digitalpass.id dan Dashboard dpe. Pengguna mengakses sistem melalui web ido.digitalpass.id, di mana data pengguna diatur dan disimpan dalam basis data PostgreSQL. Redis digunakan sebagai caching untuk mempercepat akses data pengguna. Ketika pengguna ingin melihat dashboard, mereka akan diarahkan (redirect) ke Dashboard dpe. Di sini, Dashboard dpe mengatur izin akses pengguna menggunakan PostgreSQL untuk menyimpan informasi otorisasi, sementara Redis digunakan untuk caching.

Data yang ditampilkan di dashboard berasal dari MySQL, yang dihosting sebagai DB DOA Datamart di DC INS. Layanan *backend* (*SingleStore*) yang berada di *Playcourt* digunakan untuk melakukan query data dari MySQL dan menyajikannya di dashboard. Komunikasi data antara komponen, terutama antara DC INS dan Playcourt, melewati firewall untuk memastikan keamanan dan integritas data. Struktur ini memastikan bahwa sistem dapat menangani permintaan pengguna secara efisien dan aman, dengan pemisahan yang jelas antara pengelolaan data pengguna dan data dashboard, serta penggunaan caching untuk meningkatkan kinerja.

Pada Gambamr 23 Jumlah Service dasbor ido, semua layanan yang berhubungan dengan IDO ditampilkan. Layanan-layanan ini dibagi menjadi dua bagian utama: layanan IDO dan layanan dashboard. Untuk dashboard, terdapat dua layanan utama yaitu "*dtp-ido-dashboard-be*" dan "*dtp-ido-singlestore-be*". Layanan "*dtp-ido-dashboard-be*" bertanggung jawab mengatur izin akses pengguna ke

e-ISSN2774-5155
p-ISSN 2774-5147

setiap halaman dashboard dan melakukan permintaan data ke "*dtp-ido-singlestore-be*", kemudian menyampaikan data tersebut ke frontend. Sementara itu, "*dtp-ido-singlestore-be*" adalah layanan yang melakukan query dan pengolahan data dari sumber data yang dimiliki oleh IDO.



| TYPE | SERVICE | LAST DEPLOY | REQUESTS | ERROR RATE | P95 |
|---|---|---|---|---|---|
| | dtp-ido-be-redis JS env:staging | 10d ago | 0.4 req/s | | |
| | dtp-ido-dashboard-be-redis JS env:staging | 13d ago | < 0.1 req/s | | |
| | dtp-ido-dashboard-be JS env:staging | 13d ago | < 0.1 req/s | 0.2% | |
| | dtp-ido-singlestore-be JS env:staging | | < 0.1 req/s | 1.0% | |
| | dtp-ido-singlestore-be-mysql JS env:staging | | < 0.1 req/s | 1.0% | |
| | dtp-ido-dashboard-be-postgres JS env:staging | 13d ago | < 0.1 req/s | 0.2% | |
| | dtp-ido-be-postgres JS env:staging | 10d ago | < 0.1 req/s | 0.5% | |
| | dtp-ido-be JS env:staging | 17d ago | < 0.1 req/s | | |
| | dpe-dtp-ido-be JS env:staging | | < 0.1 req/s | 1.2% | |

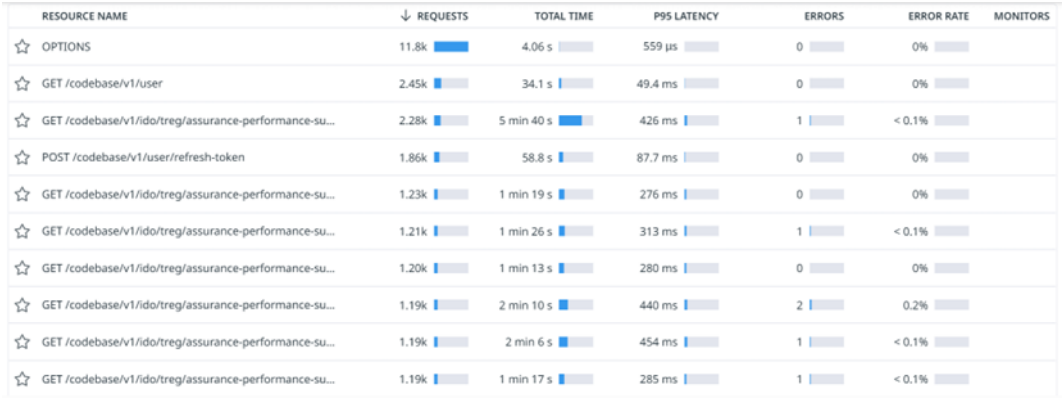
**Gambar 20. Jumlah Service Dasbor Ido**

**List request**

Gambar 4.24 Contoh list request menunjukkan metrik kinerja berbagai permintaan (requests) ke API dalam sebuah sistem. Kolom "RESOURCE NAME" menampilkan nama endpoint yang diminta, seperti GET dan POST ke "/codebase/v1/ido/user" dan endpoint lainnya yang terkait dengan "assurance-performance-summaries". Terdapat total jumlah permintaan yang diajukan ke setiap endpoint.

Kolom "TOTAL TIME" menunjukkan total waktu yang dihabiskan untuk memproses semua permintaan ke setiap endpoint. Misalnya, permintaan GET ke "/codebase/v1/ido/treg/assurance-performance-summaries" memiliki total waktu pemrosesan antara 1 menit 13 detik hingga 2 menit 10 detik. Kolom "P95 LATENCY" menunjukkan latensi di persentil ke-95, yang berarti 95% dari permintaan diproses dalam waktu kurang dari nilai tersebut.

Kolom "ERRORS" dan "ERROR RATE" mengindikasikan jumlah kesalahan dan persentase kesalahan dari total permintaan untuk setiap endpoint. Sebagian besar endpoint menunjukkan kesalahan yang sangat sedikit atau bahkan tidak ada, dengan error rate yang sangat rendah, menunjukkan performa yang baik.



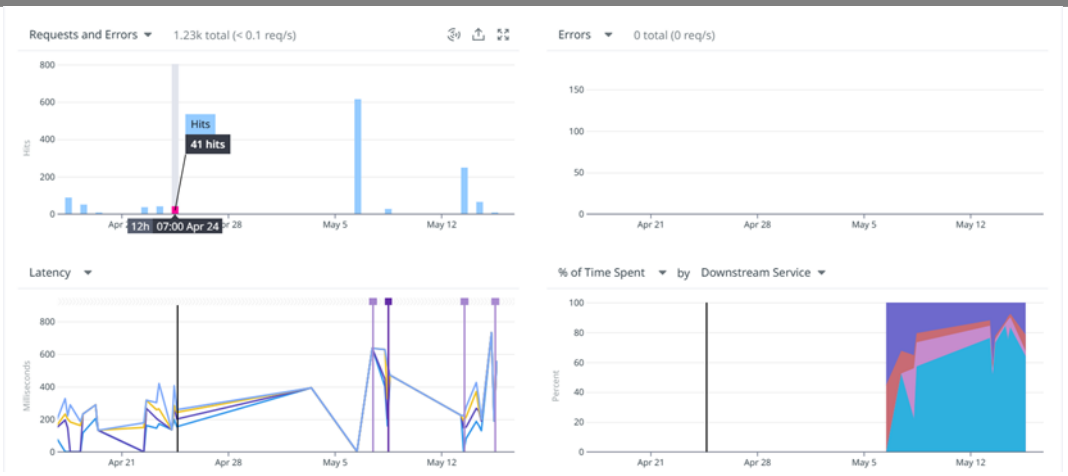
| RESOURCE NAME | REQUESTS | TOTAL TIME | P95 LATENCY | ERRORS | ERROR RATE | MONITORS |
|---|---|---|---|---|---|---|
| OPTIONS | 11.8k | 4.06 s | 559 µs | 0 | 0% | |
| GET /codebase/v1/user | 2.45k | 34.1 s | 49.4 ms | 0 | 0% | |
| GET /codebase/v1/ido/treg/assurance-performance-su... | 2.28k | 5 min 40 s | 426 ms | 1 | < 0.1% | |
| POST /codebase/v1/user/refresh-token | 1.86k | 58.8 s | 87.7 ms | 0 | 0% | |
| GET /codebase/v1/ido/treg/assurance-performance-su... | 1.23k | 1 min 19 s | 276 ms | 0 | 0% | |
| GET /codebase/v1/ido/treg/assurance-performance-su... | 1.21k | 1 min 26 s | 313 ms | 1 | < 0.1% | |
| GET /codebase/v1/ido/treg/assurance-performance-su... | 1.20k | 1 min 13 s | 280 ms | 0 | 0% | |
| GET /codebase/v1/ido/treg/assurance-performance-su... | 1.19k | 2 min 10 s | 440 ms | 2 | 0.2% | |
| GET /codebase/v1/ido/treg/assurance-performance-su... | 1.19k | 2 min 6 s | 454 ms | 1 | < 0.1% | |
| GET /codebase/v1/ido/treg/assurance-performance-su... | 1.19k | 1 min 17 s | 285 ms | 1 | < 0.1% | |

**Gambar 21. Contoh List Request**

**Statistic request**

Jika memilih salah satu resource name yang ada pada Gambar 4.24 Contoh list request maka datadog akan menampilkan chart seperti pada Gambar 4.25 Statistic Request. Disini developer mendapatkan gambaran statistik lebih baik untuk menentukan kualitas dari API atau service tersebut.

e-ISSN2774-5155
p-ISSN 2774-5147

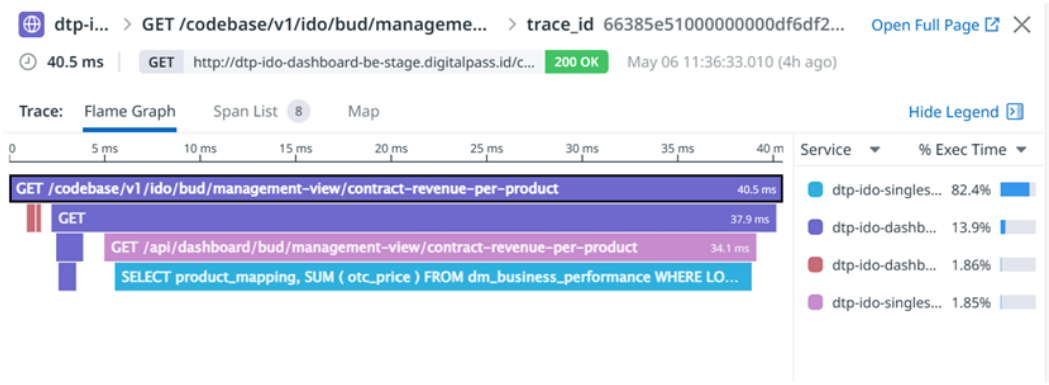



**Gambar 22. Statistic Request**

**Trace detail request**

Datadog memiliki fitur tracing yang memungkinkan pengguna untuk melacak dan memantau performa aplikasi mereka, terutama aplikasi yang kompleks dan terdistribusi. Tracing membantu mengidentifikasi bottleneck, error, dan latensi dalam aplikasi dengan menyediakan visualisasi perjalanan suatu request melalui berbagai layanan dan komponen aplikasi.

Sebagai contoh, Gambar23 Trace Detail Request menunjukkan proses yang terjadi ketika API contract-revenue-per-product dijalankan. Terlihat bahwa API contract-revenue-per-product yang berada di backend dashboard akan memanggil API contract-revenue-per-product yang berada di backend singlestore. Ketika API singlestore berjalan, API tersebut akan melakukan permintaan data ke MySQL. Sebagian besar waktu digunakan untuk memproses data di MySQL (Bünz et al. 2021).




**Gambar 23. *Trace Detail Request***

***Docker Compose***

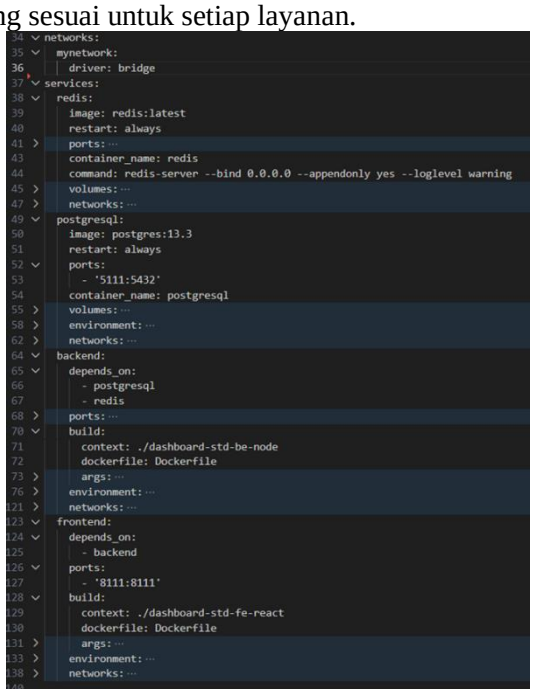
Docker Compose adalah alat yang digunakan untuk mendefinisikan dan menjalankan aplikasi multi-container Docker (Isron and Putra 2021). Dengan Docker Compose, kita bisa mendefinisikan layanan-layanan yang diperlukan dalam sebuah aplikasi dalam satu file docker-compose.yml. File ini mendeskripsikan bagaimana setiap container dikonfigurasi, bagaimana mereka saling berhubungan, dan bagaimana mereka diatur ulang ketika ada perubahan.

Gambar 27 Konfigurasi Docker Compose menunjukan layanan apa saja yang dibutuhkan untuk menajalankan aplikasi. Terdapat 4 layanan utama yaitu redis, postgresql, backend dan frontend. Setiap layanan diatur dengan menggunakan image yang spesifik, pengaturan ulang otomatis (restart), serta pemetaan port antara container dan host. Misalnya, layanan redis menggunakan image redis:latest dan menjalankan perintah khusus saat memulai, sementara layanan postgresql menggunakan image postgres:13.3 dengan pemetaan port 5432 ke 5111 di host.

Gambar 24 Stats Docker Container menunjukkan keluaran dari perintah docker stats, yang mengindikasikan bahwa empat container sedang berjalan dengan penggunaan sumber daya yang

e-ISSN2774-5155
p-ISSN 2774-5147

bervariasi. Semua container berjalan dalam batas memori yang diizinkan, dengan tidak ada tanda-tanda penggunaan sumber daya yang berlebihan. Ini menunjukkan bahwa sistem berjalan efisien dan stabil, dengan alokasi sumber daya yang sesuai untuk setiap layanan.




**Gambar 24. Konfigurasi *Docker Compose***

| CONTAINER ID | NAME | CPU % | MEM USAGE / LIMIT | MEM % | NET I/O | BLOCK I/O | PIDS |
|---|---|---|---|---|---|---|---|
| 4fa03ef6fbe2 | dashboardstd-frontend-1 | 0.00% | 68.02MiB / 7.663GiB | 0.87% | 280kB / 10.5kB | 0B / 0B | 23 |
| 5be187d99e97 | dashboardstd-backend-1 | 3.66% | 65.27MiB / 7.663GiB | 0.83% | 1.54kB / 710B | 0B / 0B | 22 |
| 26d6a08dd36b | redis | 0.25% | 11.26MiB / 7.663GiB | 0.14% | 1.48kB / 158B | 0B / 0B | 5 |
| 56b03827b828 | postgresql | 0.05% | 29.81MiB / 7.663GiB | 0.38% | 1.65kB / 527B | 0B / 0B | 7 |

**Gambar 25. *Stats Docker Container***

**Digunakan Kembali *(Reusable)***

Pendekatan pengembangan yang berorientasi pada layanan (service-oriented) memungkinkan kami untuk merancang komponen-komponen dasbor yang dapat digunakan kembali dalam berbagai konteks. Setiap layanan dibangun dengan fokus pada satu fungsi atau fitur tertentu, sehingga memungkinkan untuk digunakan kembali dalam proyek-proyek yang berbeda.

Hal ini telah membawa dampak positif dalam efisiensi pengembangan, karena pengembang dapat memanfaatkan kembali layanan-layanan yang sudah ada tanpa perlu membangun kembali dari awal. Selain itu, hal ini juga memungkinkan untuk meningkatkan konsistensi dan kualitas kode secara keseluruhan, karena setiap layanan hanya perlu dikelola dan diperbarui secara terpisah (Shabut, Lwin, and Hossain 2016).

## KESIMPULAN

Merancang dasbor untuk PT. XYZ, pendekatan yang berfokus pada keamanan, skalabilitas, dan kegunaan kembali (reusability) telah membawa manfaat yang signifikan. Dengan menerapkan prinsip-prinsip keamanan OWASP, dasbor dapat dijamin keamanannya terhadap berbagai serangan yang umum terjadi. Penggunaan arsitektur microservices memungkinkan dasbor untuk menjadi lebih skalabel, memungkinkan penyesuaian dengan beban kerja yang berubah-ubah tanpa mengorbankan performa. Selain itu, penerapan *framework* komponen *backend* dan *frontend* yang dapat digunakan kembali telah meningkatkan efisiensi pengembangan dan kualitas kode secara keseluruhan.

e-ISSN2774-5155
p-ISSN 2774-5147

## DAFTAR PUSTAKA

Aitlmoudden, Othmane, Mohamed Housni, Nisrine Safeh, and Abdelwahed Namir. 2023. "A Microservices-Based Framework for Scalable Data Analysis in Agriculture with IoT Integration." *International Journal of Interactive Mobile Technologies* 17(19).

Alshuqayran, Nuha, Nour Ali, and Roger Evans. 2016. "A Systematic Mapping Study in Microservice Architecture." Pp. 44–51 in *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*. IEEE.

Blinowski, Grzegorz, Anna Ojdowska, and Adam Przybyłek. 2022. "Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation." *IEEE Access* 10:20357–74.

Bünz, Benedikt, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. 2021. "Proof-Carrying Data without Succinct Arguments." Pp. 681–710 in *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part I 41*. Springer.

Cobb, Charles G. 2023. *The Project Manager's Guide to Mastering Agile: Principles and Practices for an Adaptive Approach*. John Wiley & Sons.

Flora, José. 2020. "Improving the Security of Microservice Systems by Detecting and Tolerating Intrusions." Pp. 131–34 in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE.

Hannousse, Abdelhakim, and Salima Yahiouche. 2021. "Securing Microservices and Microservice Architectures: A Systematic Mapping Study." *Computer Science Review* 41:100415.

Isron, Muhammad Hussein, and Ricky Eka Putra. 2021. "Implementasi Virtual Server Berbasis Container Pada Sistem Informasi Geografis Cagar Budaya Mojokerto." *Journal of Informatics and Computer Science (JINACS)* 3(02):143–54.

Kalubowila, D. C., S. M. Athukorala, B. A. S. Tharaka, H. W. Y. R. Samarasekara, Udara Srimath S. Samaratunge Arachchilage, and Dharshana Kasthurirathna. 2021. "Optimization of Microservices Security." Pp. 49–54 in *2021 3rd International Conference on Advancements in Computing (ICAC)*. IEEE.

Khalid, Ayesha, Shariq Aziz Butt, Tauseef Jamal, and Saikat Gochhait. 2020. "Agile Scrum Issues at Large-Scale Distributed Projects: Scrum Project Development at Large." *International Journal of Software Innovation (IJSI)* 8(2):85–94.

Krämer, Michel, Sven Frese, and Arjan Kuijper. 2019. "Implementing Secure Applications in Smart City Clouds Using Microservices." *Future Generation Computer Systems* 99:308–20.

Laigner, Rodrigo, Yongluan Zhou, Marcos Antonio Vaz Salles, Yijian Liu, and Marcos Kalinowski. 2021. "Data Management in Microservices: State of the Practice, Challenges, and Research Directions." *ArXiv Preprint ArXiv:2103.00170*.

Mateus-Coelho, Nuno, Manuela Cruz-Cunha, and Luis Gonzaga Ferreira. 2021. "Security in Microservices Architectures." *Procedia Computer Science* 181:1225–36.

Pardosi, Victor Benny Alexsius, S. Kom, Abdul Karim, M. Ti, Rozali Ilham, M. Kom, Hayadi Hamuda, Fernando V Dotulong, Afif Zuhri Arfianto, and Selly Septiani. 2024. *Sistem Keamanan Komputer*. CV Rey Media Grafika.

Pratama, Rizky Nandang, and Yeremia Alfa Susetyo. 2024. "Implementasi Python API Dengan Framework Flask Sebagai Cloud Run Service Untuk Proses Update Di PT. XYZ." *Kesatria: Jurnal Penerapan Sistem Informasi (Komputer Dan Manajemen)* 5(2):669–76.

Pudoli, Ahmad, Yulianawati Yulianawati, and Iqbal Suwandi. 2023. "Implementasi Web Service Restful Dengan Autentikasi JSon Web Token Berbasis Web Dan Android." *Academic Journal of Computer Science Research* 5(2):95–103.

Raj, Prixit, and Parul Sinha. 2015. "Project Management in Era of Agile and Devops Methodlogies." Pp. 1024–33 in *International Conference on Applied Sciences*. Vol. 9.

Sabila, Khoiru, Siti Rahayu, and Titin Sumarni. 2024. "Peningkatan Efisiensi Penggunaan Sumber Daya Jaringan Melalui Teknik Load Balancing." *CEMERLANG: Jurnal Manajemen Dan*

e-ISSN2774-5155
p-ISSN 2774-5147

*Ekonomi Bisnis* 4(3):31–41.
Setiawan, Andhika Ricky. 2018. "LKP: Optimalisasi Manajemen Jaringan Menggunakan Mikrotik Pada PT. Angkasa Pura I (Persero) Juanda Surabaya."
Shabut, Antesar M., Khin T. Lwin, and M. Alamgir Hossain. 2016. "Cyber Attacks, Countermeasures, and Protection Schemes—A State of the Art Survey." Pp. 37–44 in *2016 10th International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*. IEEE.
Singh, Amandeep, Vinay Raj, and Sadam Ravichandra. 2022. "Integration of Attribute-Based Access Control in Microservices Architecture." Pp. 681–90 in *ICT Systems and Sustainability: Proceedings of ICT4SD 2021, Volume 1*. Springer.
Sulistyorini, Tri, Erma Sova, and Rafli Ramadhan. 2022. "Pemantauan Kasus Penyebaran Covid-19 Berbasis Website Menggunakan Framework React Js Dan Api." *Jurnal Ilmiah Multidisiplin* 1(04):1–13.
Sya'bani, Fadila Ahmad. 2023. "Hardening Sistem Informasi Sekawan V2 Menggunakan Framework Owasp."
Tapia, Freddy, Miguel Ángel Mora, Walter Fuertes, Hernán Aules, Edwin Flores, and Theofilos Toulkeridis. 2020. "From Monolithic Systems to Microservices: A Comparative Study of Performance." *Applied Sciences* 10(17):5797.
Wandapranata, Jessika, and Seng Hansun. 2017. "Pengembangan Modul Autentikasi Captcha Berbasis Gambar Dengan Algoritma Flood Fill." *Journal of Information Systems Engineering and Bussiness Intelligence* 3(1):1–7.
Yarygina, Tetiana, and Anya Helene Bagge. 2018. "Overcoming Security Challenges in Microservice Architectures." Pp. 11–20 in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. IEEE.
Yu, Dongjin, Yike Jin, Yuqun Zhang, and Xi Zheng. 2019. "A Survey on Security Issues in Services Communication of Microservices-enabled Fog Applications." *Concurrency and Computation: Practice and Experience* 31(22):e4436.