LAPORAN PROYEK SISTEM TERINTEGRASI STUDI KASUS: SECURE MESSAGING PLATFORM (CHAT & ENCRYPTION MICROSERVICES)

II3160 - TEKNOLOGI SISTEM TERINTEGRASI

Disusun Oleh:

Mudzaki Kaarzaqiel Hakim / 18223024

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2026

Daftar Isi

CHAT CORE API SERVICE

A. DESKRIPSI LAYANAN

Chat Core API adalah mikroservis utama yang bertindak sebagai orkestrator dalam ekosistem *Secure Messaging Platform*. Dibangun menggunakan lingkungan eksekusi Node.js dengan kerangka kerja Express.js, layanan ini bertanggung jawab penuh atas logika bisnis aplikasi, manajemen identitas pengguna, serta validasi aturan pertemanan (*friendship logic*).

Peran krusial dari layanan ini adalah sebagai jembatan (*gateway*) antara antarmuka pengguna (*Frontend*) dan layanan keamanan (*Encryption Service*). Chat Core API menerapkan arsitektur *Event-Driven Non-Blocking I/O*, yang memungkinkannya menangani ribuan permintaan konkuren seperti pengiriman pesan dan *polling* data secara efisien tanpa membebani *thread* utama CPU, sebuah karakteristik yang sangat vital saat dijalankan pada perangkat dengan sumber daya terbatas seperti *Set Top Box* (STB).

B. DATASET DAN PERSISTENSI

Mengingat keterbatasan siklus tulis (*write-cycle*) pada penyimpanan *flash* di perangkat STB, penggunaan sistem manajemen basis data relasional (RDBMS) konvensional seperti MySQL atau PostgreSQL dinilai terlalu berat (*resource-heavy*). Oleh karena itu, layanan ini mengimplementasikan LowDB, sebuah adaptor basis data berbasis JSON yang ringan (*lightweight*) dan *serverless*.

Struktur Penyimpanan Data: Data disimpan dalam satu berkas db.json yang terstruktur secara hirarkis. Skema data dirancang dengan pendekatan denormalisasi parsial untuk mengoptimalkan kecepatan pembacaan (*read performance*).

1. Entitas Users (users): Menyimpan profil pengguna, kredensial keamanan, dan daftar kontak.

```
{
  "id": "uuid-v4",
  "username": "user1",
  "password": "$2b$10$hashed_password...", // Disimpan dalam bentuk Hash Bcrypt
  "contacts": ["uuid-friend1", "uuid-friend2"] // Array ID teman (Mutual)
```

```
}
```

Penjelasan*:* Password tidak pernah disimpan dalam bentuk teks asli (*plaintext*). Daftar kontak disimpan langsung dalam objek user untuk menghindari operasi *JOIN* yang mahal.

2. Entitas Chats (chats): Menyimpan riwayat percakapan. Penting dicatat bahwa atribut content pada entitas ini wajib berisi *ciphertext* (teks terenkripsi).

```
{
  "id": "uuid-chat",
  "senderId": "uuid-sender",
  "receiverId": "uuid-receiver",
  "content": "a8f9c2...[ENCRYPTED]",
  "timestamp": 1704445566
}
```

Penjelasan*:* LowDB memungkinkan operasi I/O atomik, memastikan bahwa data pesan hanya tertulis ke *disk* setelah proses enkripsi dari layanan Go berhasil dikonfirmasi.

C. ENDPOINT API

Layanan ini menyediakan antarmuka RESTful API yang dikonsumsi oleh aplikasi Frontend. Berikut adalah spesifikasi *endpoint* utama:

Endpoint	**Method**	**Deskripsi**	**Request (Body/Params/Query)**	**Response (Success/Error)**	**Autentikasi**
/auth/register	POST	Mendaftarkan pengguna baru	{ "username", "password" }	**200:** Result (authService.register) **400:** { "error": "..."	**Publik**

				}	
/auth/login	POST	Login pengguna	{ "username", "password" }	**200:** { "id": "...", ... } **401:** { "error": "..." }	**Publik**
/auth/change-password	POST	Mengubah password	{ "userId", "newPassword" }	**200:** Result (authService.changePassword) **404:** { "error": "..." }	**Wajib**
/friends/add	POST	Menambahkan teman	{ "myId", "friendUsername" }	**200:** { "message": "...", "contact": {...} } **400:** { "error": "..." }	**Wajib**

/friends/:userId	GET	Mengambil daftar teman	**URL Param:** userId	**200:** [{ "id": "...", "username": "..." }, ...] **500:** { "error": "..." }	**Wajib**
/chat/send	POST	Mengirim pesan	{ "senderId", "receiverId", "message" }	**200:** { "id", "senderId", "receiverId", "content", "timestamp" } **500:** { "error": "..." }	**Wajib**
/chat/history	GET	Mengambil riwayat chat	**Query:** ?myId=...&friendId=...	**200:** [{ "id", "senderId", "receiverId", "content", "timestamp" }, ...] **500:** { "error": "..."	**Wajib**

				}	

D. MEKANISME AUTENTIKASI DAN OTORISASI

Sistem keamanan akses pada Chat Core API dirancang berlapis untuk melindungi integritas data pengguna:

1. Password Hashing (Bcrypt): Saat registrasi, *password* pengguna di-*hash* menggunakan algoritma Bcrypt dengan *salt rounds* 10. Ini adalah standar industri untuk memperlambat serangan *brute-force* atau *rainbow table*. Verifikasi *login* dilakukan dengan membandingkan *hash* yang tersimpan di LowDB dengan input pengguna.
2. Validasi Sesi Klien (Client-Side Session): Untuk mengurangi penggunaan memori RAM pada STB, sistem tidak menggunakan *server-side session storage* (seperti Redis). Sebagai gantinya, setelah login sukses, server mengembalikan objek profil pengguna yang kemudian disimpan secara aman di *LocalStorage* browser. Setiap *request* sensitif (seperti kirim pesan) mewajibkan validasi ID pengguna yang sah.
3. Otorisasi Logika Bisnis: Sistem menerapkan aturan *Mutual Friendship Authorization*. Pengguna A hanya diizinkan mengirim pesan ke Pengguna B jika dan hanya jika ID Pengguna B terdapat dalam daftar contacts Pengguna A. Validasi ini mencegah pengiriman pesan *spam* dari pengguna yang tidak dikenal.

E. DOKUMENTASI API

Dokumentasi teknis untuk integrasi Frontend disediakan dalam format JSON standar. Karena sifat proyek ini sebagai sistem tertutup (*closed system*), dokumentasi difokuskan pada struktur *payload* permintaan dan tanggapan.

Contoh Respons Sukses (Login):

```
{
  "id": "550e8400-e29b-41d4-a716-446655440000",
  "username": "hakim",
  "contacts": [...]
}
```

Contoh Respons Sukses (Kirim Pesan):

```
{
  "message": "Berhasil kirim pesan",
```

```
  "data": {
    "senderId": "...",
    "receiverId": "...",
    "timestamp": 1704445566
  }
}
```

F. KONFIGURASI API DI STB

Mengingat perangkat STB memiliki arsitektur ARM dan memori terbatas, API ini dikonfigurasi dengan strategi optimasi khusus:

1. Kontainerisasi Ringan (Docker Alpine): Layanan dikemas menggunakan *base image* node:lts-alpine. Versi Alpine Linux dipilih karena ukurannya yang sangat kecil (< 50MB), meminimalkan jejak penyimpanan (*storage footprint*) pada STB dibandingkan image OS standar.
2. Manajemen Memori Node.js: Konfigurasi --max-old-space-size=512 diterapkan pada proses Node.js untuk membatasi penggunaan *heap memory* maksimal 512MB. Hal ini mencegah aplikasi memonopoli RAM yang dapat menyebabkan sistem operasi STB menjadi tidak responsif (*hang*).
3. Sistem menerapkan strategi keamanan berlapis melalui isolasi jaringan dan penggunaan reverse proxy untuk mengatur eksposur layanan. Layanan Chat Core dijalankan pada port 8080 di localhost STB dan menjadi satu-satunya komponen yang dapat diakses dari internet publik. Akses publik tersebut tidak dilakukan secara langsung, melainkan melalui mekanisme tunneling Cloudflare yang memetakan domain aman https://hakim.tugastst.my.id/api ke port lokal Chat Core. Sebaliknya, Encryption Service dan basis data LowDB dikonfigurasi agar sepenuhnya terisolasi dari jaringan publik dan hanya dapat diakses melalui jaringan internal Docker (internal bridge network) oleh Chat Core. Dengan arsitektur ini, sistem memastikan bahwa tidak ada jalur alternatif bagi penyerang untuk mengakses API enkripsi maupun file database secara langsung dari internet, karena satu-satunya pintu masuk yang terbuka hanyalah API Chat Core yang telah dilindungi oleh mekanisme autentikasi.

Layanan Terintegrasi - chat services

A. DESKRIPSI SISTEM

a. Tinjauan Umum Sistem (*System Overview*)

"Secure Messaging Platform" adalah sistem komunikasi instan berbasis web yang dirancang dengan pendekatan *Security-First Microservices*. Berbeda dengan aplikasi *chat* konvensional yang bersifat monolitik, sistem ini mendekonstruksi fungsi komunikasi menjadi dua entitas otonom: Communication Core (pengelola logika pertukaran pesan) dan Cryptography Engine (pengelola logika keamanan).

Sistem ini tidak hanya bertujuan mengirimkan pesan dari A ke B, tetapi bertindak sebagai *Secure Pipeline*. Setiap pesan yang masuk ke sistem diperlakukan sebagai *untrusted payload* yang wajib melalui proses sanitasi dan enkripsi oleh layanan terpisah sebelum diizinkan menyentuh lapisan persistensi (database). Arsitektur ini menjamin bahwa basis data utama hanya menyimpan *ciphertext* (teks sandi) yang tidak dapat dibaca (unintelligible), bahkan oleh administrator sistem sekalipun, menjaga prinsip kerahasiaan (*confidentiality*) dan integritas (*integrity*) data secara absolut.

b. Karakteristik & Kendala Desain (*Design Constraints*)

Sistem ini dibangun di atas kendala infrastruktur yang ketat untuk mensimulasikan lingkungan *edge computing* nyata:

1. Hardware Constraint: Dijalankan pada *Set Top Box* (STB) dengan RAM < 2GB. Hal ini memaksa penggunaan *runtime* yang efisien (Golang untuk enkripsi) dan strategi manajemen memori yang agresif pada Node.js.
2. Network Constraint: Menggantikan protokol WebSocket yang berat (stateful) dengan mekanisme *HTTP Short-Polling* yang dioptimalkan untuk mengurangi beban *socket* terbuka pada server STB.

3. Storage Constraint: Menggunakan basis data berbasis file (LowDB/JSON) dengan strategi denormalisasi untuk meminimalkan operasi I/O disk yang lambat pada penyimpanan flash STB.

c. Fungsionalitas Utama (*Key Functionalities*)

Sistem menawarkan serangkaian fitur yang diorkestrasi melalui antarmuka RESTful API:

1. Manajemen Identitas Terdesentralisasi (Decoupled Identity)

- Registrasi & Login: Pengguna mendaftar dengan *Username* unik. Sistem menggunakan *Bcrypt Hashing* (Salt Rounds 10) untuk mengamankan kredensial.
- Session Handling: Alih-alih menggunakan sesi server yang memakan memori, sistem menggunakan pendekatan *Client-Side Session* di mana identitas pengguna disimpan secara lokal di browser (*LocalStorage*) dan divalidasi ulang setiap kali melakukan *request* sensitif.

2. Mekanisme Berteman Mutual (*Mutual Friendship Logic*)

- Add by Username: Sistem menyederhanakan proses penambahan kontak dengan menggunakan *Username* sebagai pengenal publik, menyembunyikan UUID internal yang kompleks.
- Atomic Friendship: Saat Pengguna A menambahkan Pengguna B, sistem secara otomatis mengeksekusi transaksi ganda: memperbarui daftar kontak A dan daftar kontak B secara bersamaan. Ini mencegah inkonsistensi data (seperti hubungan satu arah).

3. Enkripsi Pesan Transparan (*Transparent Encryption Workflow*)

- On-the-fly Encryption: Saat tombol "Kirim" ditekan, pesan tidak langsung disimpan. *Chat Core Service* bertindak sebagai *proxy* yang meneruskan pesan mentah (*plaintext*) ke *Encryption Service*.
- AES-GCM Standardization: Pesan dienkripsi menggunakan algoritma AES-256-GCM yang menghasilkan *Ciphertext* + *Nonce*. Nonce ini unik untuk setiap pesan, menjamin bahwa dua pesan dengan isi "Halo" yang sama akan menghasilkan string terenkripsi yang berbeda total (mencegah *pattern analysis attack*).
- Seamless Decryption: Saat pesan diambil (*fetch*), sistem secara otomatis mendekripsi konten di *backend* sebelum dikirim ke *frontend*, sehingga pengguna akhir menerima pesan yang bisa dibaca tanpa perlu mengelola kunci dekripsi secara manual.

4. Sinkronisasi Pesan Semu-Realtime (*Pseudo-Realtime Sync*)

- Adaptive Polling: Frontend melakukan *fetching* data pesan secara berkala (interval 3-5 detik) untuk memperbarui tampilan obrolan.

- Delta Updates: API dirancang untuk hanya mengirimkan data jika terjadi perubahan status, mengurangi penggunaan *bandwidth* jaringan yang tidak perlu.

B. ARSITEKTUR SISTEM

Sistem *Secure Messaging Platform* dibangun menggunakan pola arsitektur Microservices yang dikombinasikan dengan pola Backend for Frontend (BFF). Dalam arsitektur ini, tidak ada satu komponen "raksasa" yang menangani semua tugas. Sebaliknya, tanggung jawab dibagi menjadi unit-unit kecil yang independen: unit untuk tatap muka (*Frontend*), unit untuk logika bisnis (*Chat Core*), dan unit untuk keamanan (*Encryption Service*).

Sistem ini didesain untuk berjalan secara terisolasi di dalam Docker Containers, memungkinkan portabilitas tinggi dan efisiensi sumber daya saat dijalankan pada perangkat *Set Top Box* (STB).

a. Komponen Utama

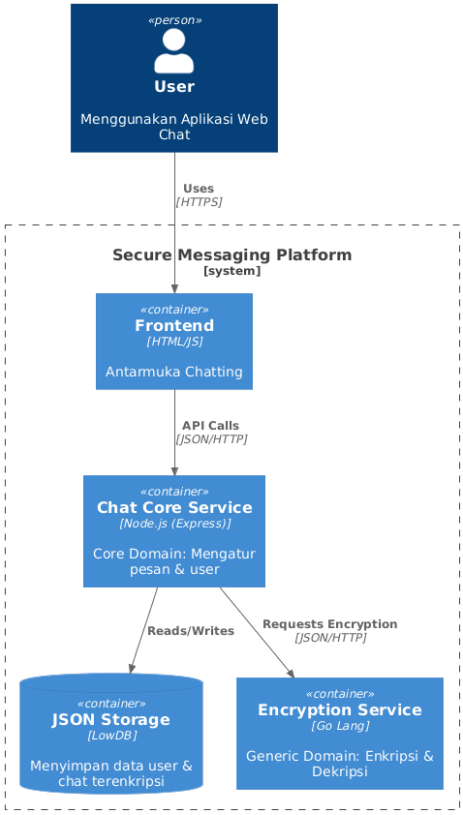
Arsitektur sistem terdiri dari empat komponen utama yang berinteraksi melalui protokol standar HTTP/JSON:

1. Client Application (Frontend Layer)
    - Teknologi: HTML5, CSS3, Vanilla JavaScript.
    - Peran: Bertindak sebagai antarmuka visual bagi pengguna. Aplikasi ini bersifat *Stateless* dan *Client-Side Rendered*. Tugas utamanya adalah menangkap input pengguna, melakukan *polling* data ke server, dan me-render pesan chat.
    - Lokasi: Dihosting pada layanan CDN (Vercel) untuk akses publik.
2. Chat Core Service (Integration Layer)
    - Teknologi: Node.js, Express.js.
    - Peran: Bertindak sebagai orkestrator atau "otak" sistem. Komponen ini mengatur siapa yang boleh menghubungi siapa (*Access Control*), menyimpan riwayat pesan ke database, dan meneruskan permintaan enkripsi ke layanan keamanan. Layanan ini adalah satu-satunya pintu gerbang (*gateway*) yang dapat diakses oleh Frontend.
3. Encryption Service (Security Layer)
    - Teknologi: Go (Golang).
    - Peran: Bertindak sebagai "brankas matematika" (*generic domain*). Layanan ini terisolasi dari internet luar dan hanya menerima perintah dari *Chat Core Service* melalui jaringan internal Docker. Tugasnya murni mengubah *Plaintext* menjadi *Ciphertext* dan sebaliknya.
4. Persistence Layer (Storage)
    - Teknologi: LowDB (JSON File).

- Peran: Menyimpan struktur data relasional (User & Chats) dalam format dokumen JSON. Dipilih karena *overhead* yang sangat kecil dibandingkan database SQL konvensional, cocok untuk I/O terbatas pada STB.

b. Diagram Kontainer (C4 Container Diagram)

Berikut adalah visualisasi arsitektur sistem menggunakan standar C4 Model yang menggambarkan interaksi antar kontainer:



c. Alur Data

Untuk memvisualisasikan bagaimana prinsip *Security-by-Design* diterapkan pada level transmisi data, diagram urutan (*Sequence Diagram*) di bawah ini memetakan interaksi temporal antara empat entitas utama: Pengguna (*User*), Antarmuka Klien (*Frontend*), Layanan Inti (*Chat Core*), dan Layanan Kriptografi (*Encryption Service*).

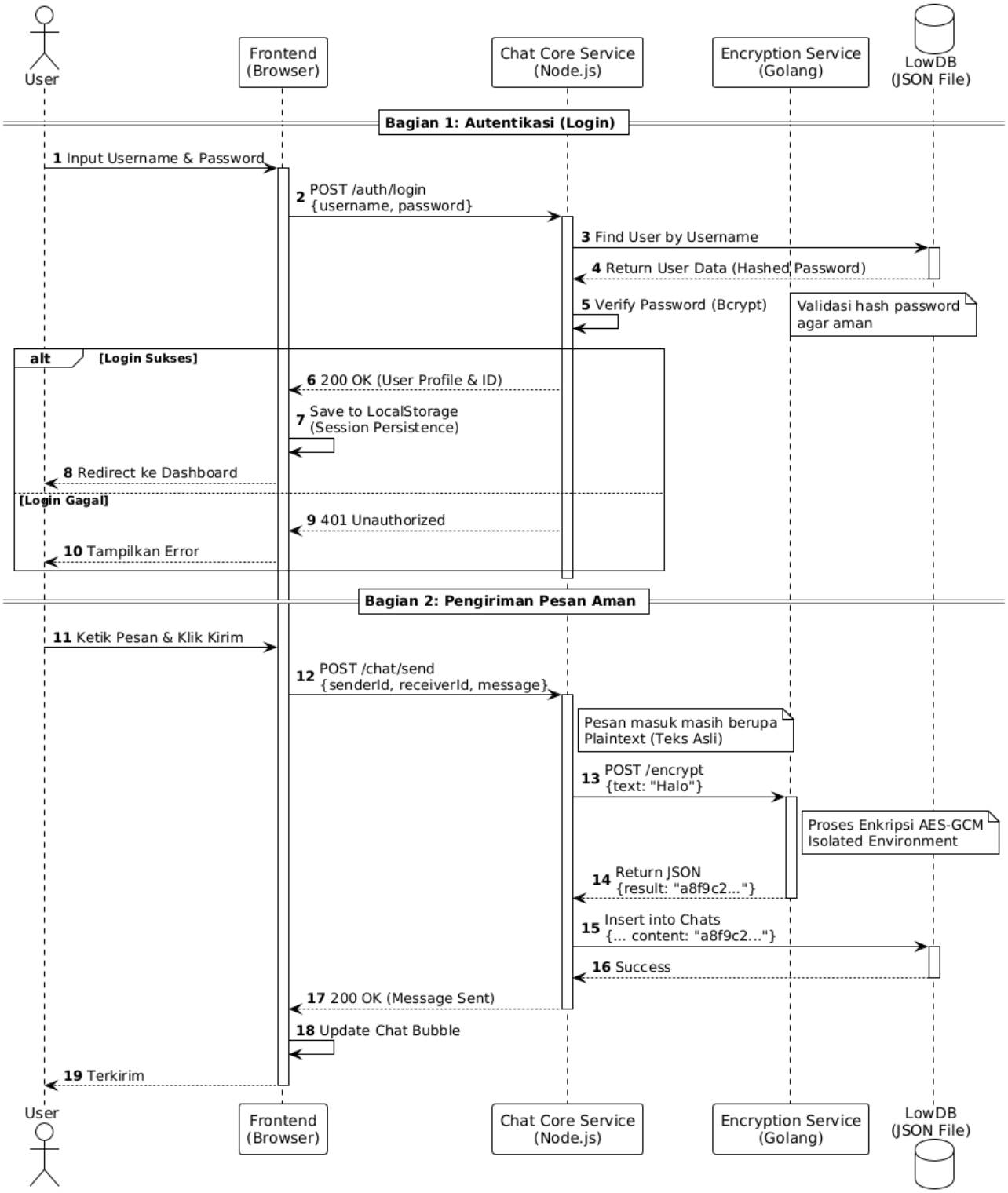
Alur data dirancang dengan mekanisme Dual-Path Processing:

1. Jalur Penulisan Aman (*Secure Write Path*): Menggambarkan perjalanan pesan dari input pengguna hingga persistensi database. Krusial untuk dicatat bahwa *Chat Core Service* bertindak sebagai *gatekeeper* yang menahan data *plaintext* dalam memori volatil (*RAM*)

dan hanya meneruskan data ke *disk storage* (LowDB) setelah menerima konfirmasi *ciphertext* dari *Encryption Service*. Ini menjamin bahwa data tidak pernah menyentuh *hard drive* dalam bentuk yang dapat dibaca.

2. Jalur Pembacaan Transparan (*Transparent Read Path*): Menggambarkan bagaimana *Frontend* melakukan *polling* data terenkripsi, yang kemudian didekripsi secara *on-the-fly* oleh *backend* sebelum dikirimkan ke klien.

Diagram ini juga menegaskan batasan jaringan (*network boundary*), di mana komunikasi ke *Encryption Service* terjadi secara internal dalam jaringan Docker tertutup, tidak terekspos ke internet publik.


User
Frontend (Browser)
Chat Core Service (Node.js)
Encryption Service (Golang)
LowDB (JSON File)
Bagian 1: Autentikasi (Login)
1 Input Username & Password
2 POST /auth/login {username, password}
3 Find User by Username
4 Return User Data (Hashed Password)
5 Verify Password (Bcrypt)
Validasi hash password agar aman
alt
[Login Sukses]
6 200 OK (User Profile & ID)
7 Save to LocalStorage (Session Persistence)
8 Redirect ke Dashboard
[Login Gagal]
9 401 Unauthorized
10 Tampilkan Error
Bagian 2: Pengiriman Pesan Aman
11 Ketik Pesan & Klik Kirim
12 POST /chat/send {senderId, receiverId, message}
Pesan masuk masih berupa Plaintext (Teks Asli)
13 POST /encrypt {text: "Halo"}
Proses Enkripsi AES-GCM Isolated Environment
14 Return JSON {result: "a8f9c2..."}
15 Insert into Chats {... content: "a8f9c2..."}
16 Success
17 200 OK (Message Sent)
18 Update Chat Bubble
19 Terkirim
User
Frontend (Browser)
Chat Core Service (Node.js)
Encryption Service (Golang)
LowDB (JSON File)

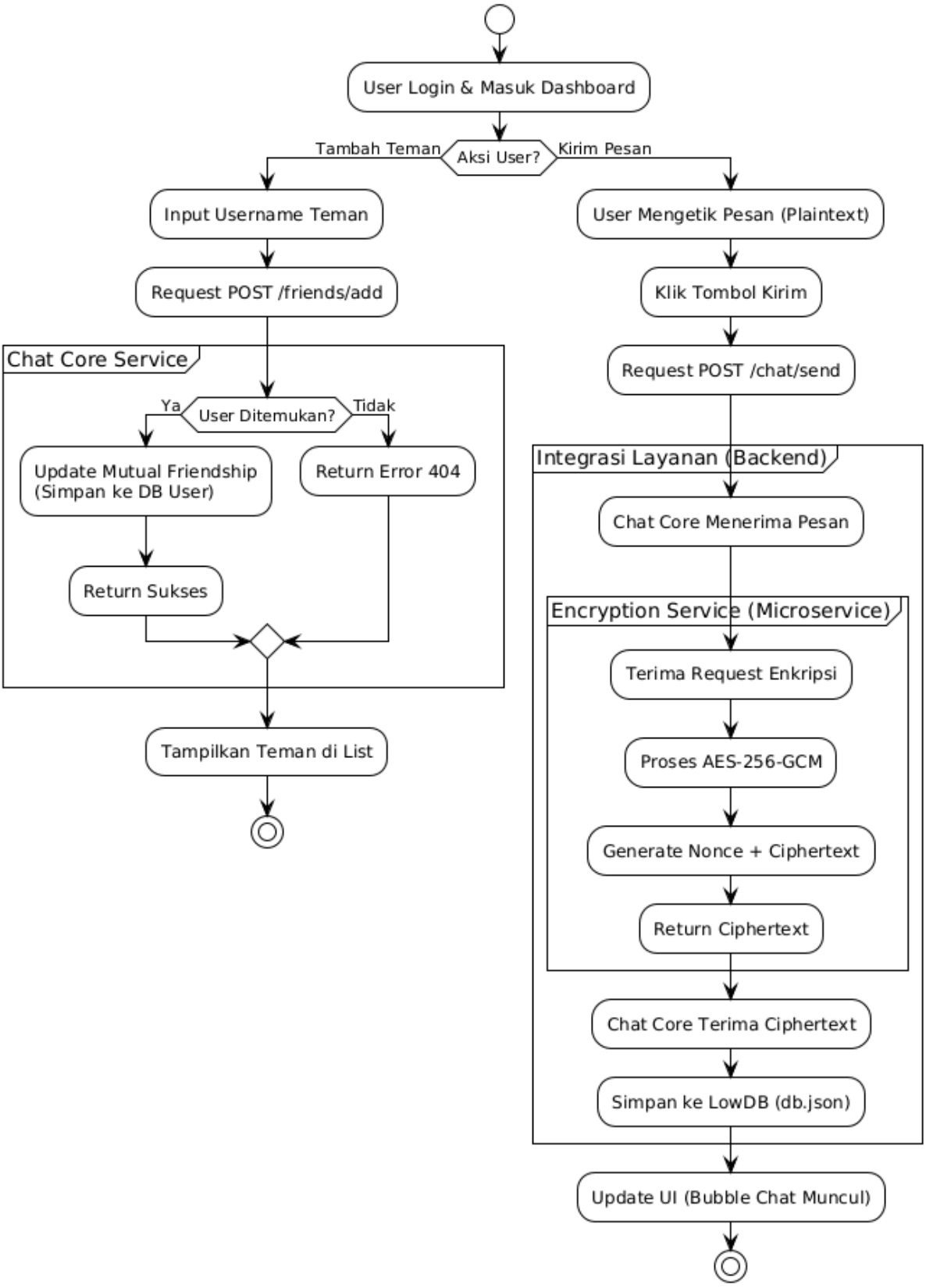
C. ALUR PROSES SISTEM TERINTEGRASI

Kompleksitas logika bisnis sistem tidak hanya terletak pada pengamanan data, tetapi juga pada orkestrasi fitur sosial (pertemanan) dan manajemen sesi. Diagram alur (*Flowchart*) berikut merepresentasikan algoritma sistem secara *end-to-end*, dimulai dari inisiasi sesi pengguna hingga eksekusi fitur inti.

Alur proses dibagi menjadi dua cabang eksekusi utama berdasarkan interaksi pengguna:

1. Logika Mutasi Pertemanan (*Friendship Mutation Logic*): Cabang kiri diagram menunjukkan mekanisme penambahan kontak. Sistem melakukan validasi keberadaan pengguna target dan menerapkan pembaruan atomik (*atomic update*) pada dokumen JSON kedua belah pihak. Jika pengguna target tidak ditemukan, sistem mengembalikan kode error 404 tanpa mengubah *state* database, menjaga konsistensi data.
2. Logika Pipa Enkripsi (*Encryption Pipeline Logic*): Cabang kanan diagram mendetailkan integrasi mikroservis. Setiap permintaan pengiriman pesan memicu sub-rutin enkripsi yang memanggil layanan Golang. Diagram memperlihatkan bahwa proses ini bersifat memblokir (*blocking*); sistem tidak akan memperbarui antarmuka pengguna (*UI update*) sebelum siklus enkripsi-simpan selesai sepenuhnya, memberikan kepastian status pengiriman (*delivery acknowledgement*) kepada pengguna.

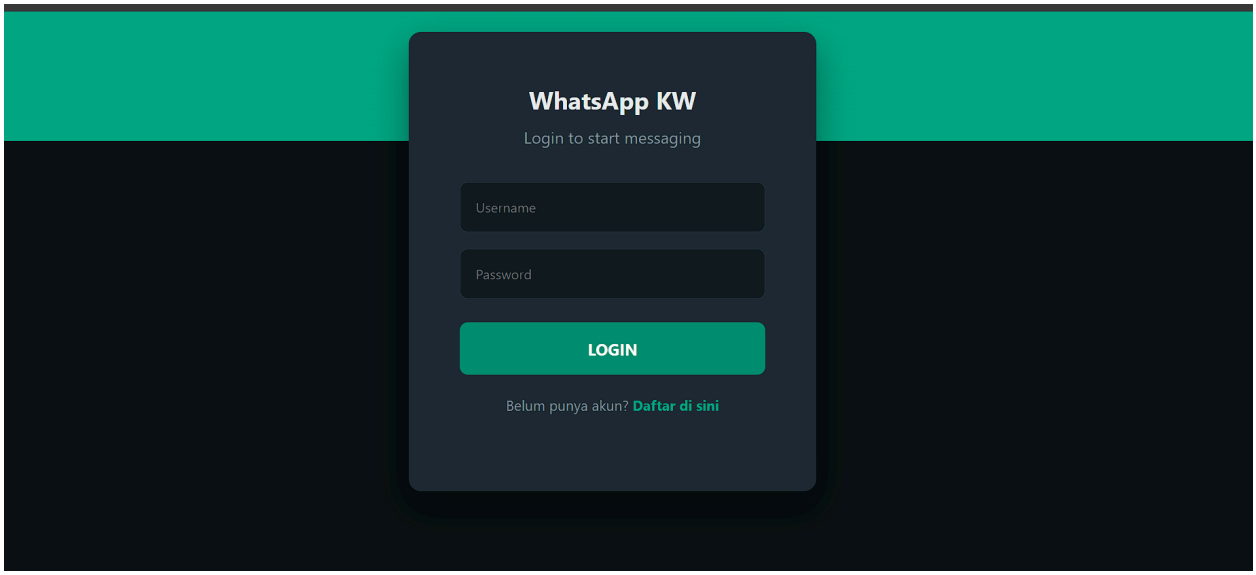
Alur Proses Secure Messaging Platform



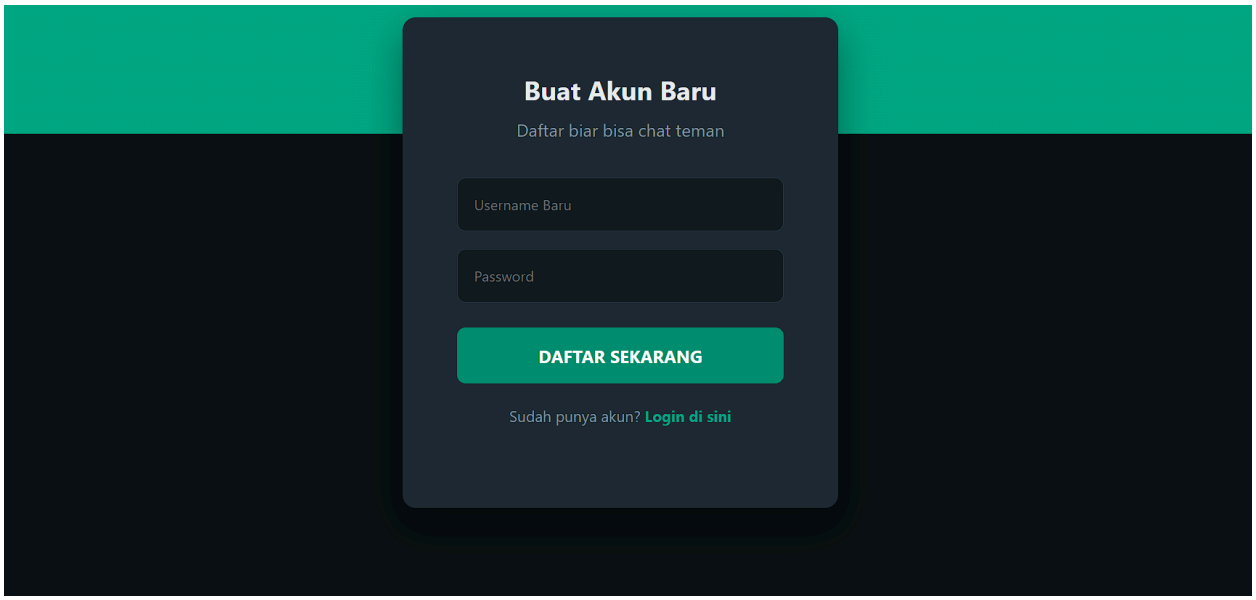
D. DOKUMENTASI SISTEM TERINTEGRASI

1. Bukti Antarmuka Pengguna (Frontend)

Halaman Login & Register :

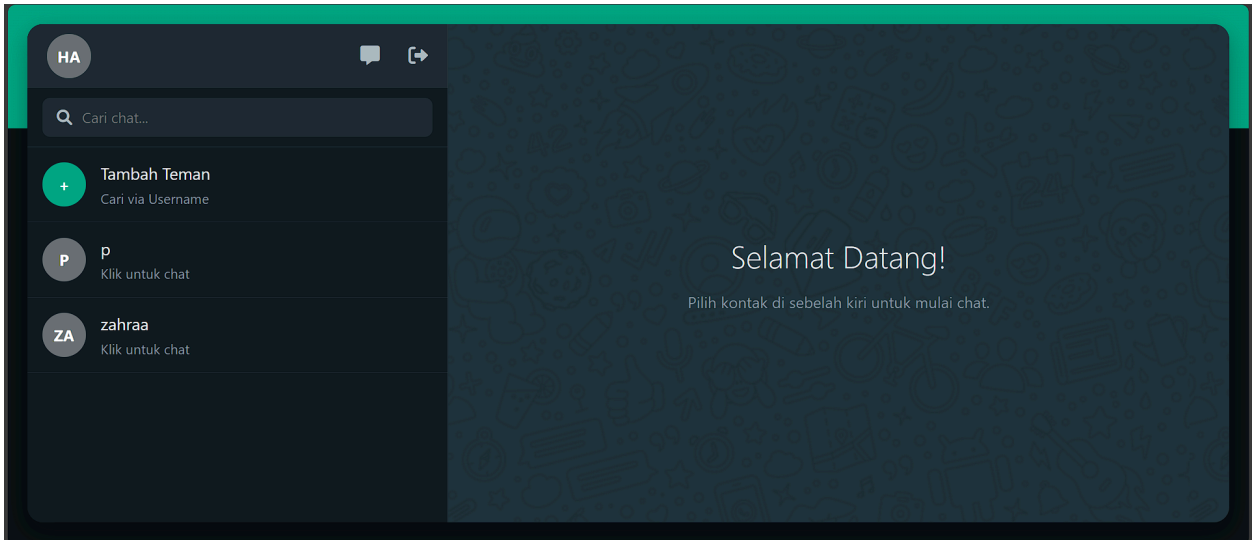


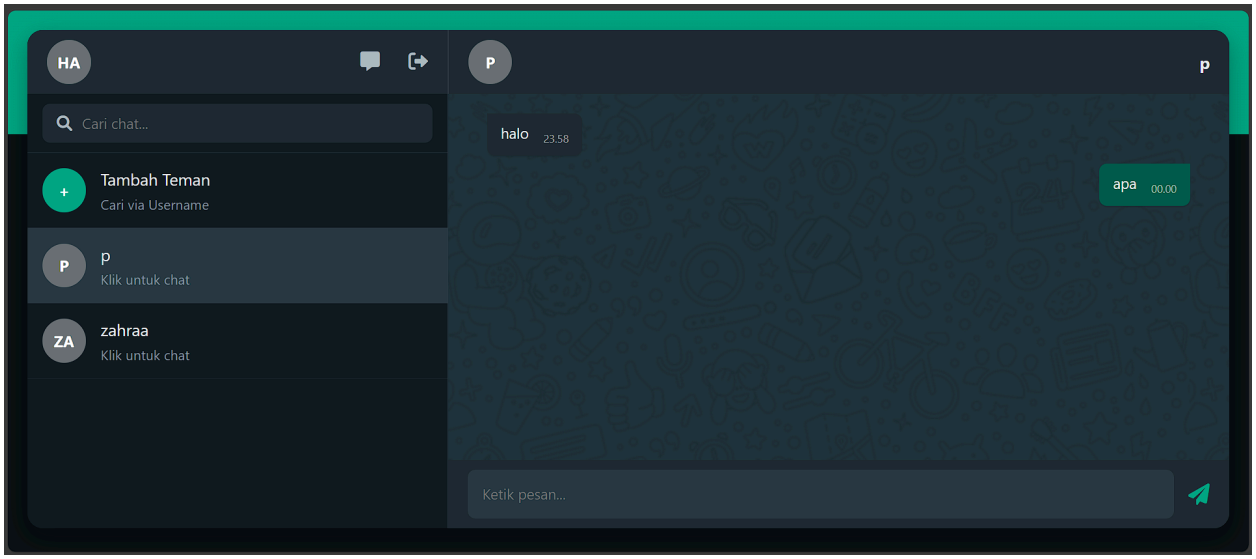
Login



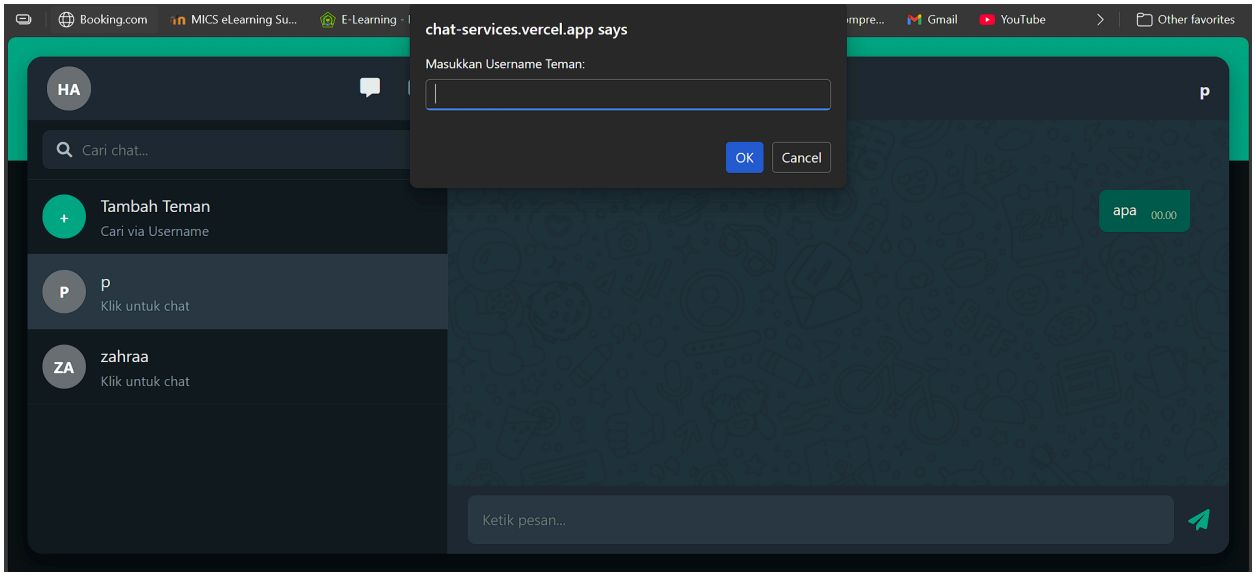
Register

Dashboard Chat Utama :

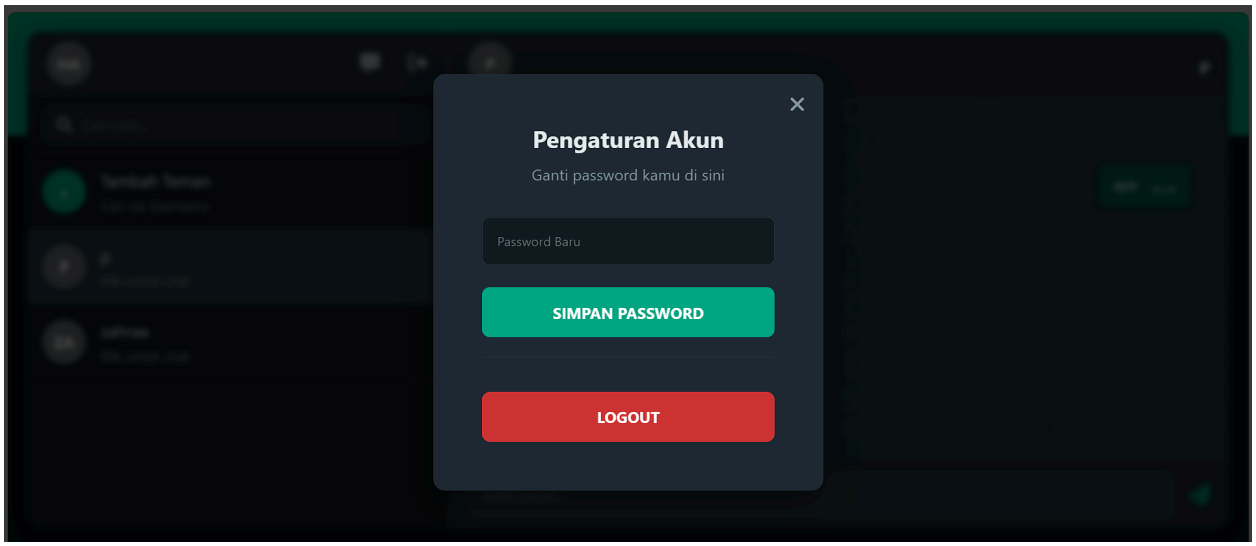




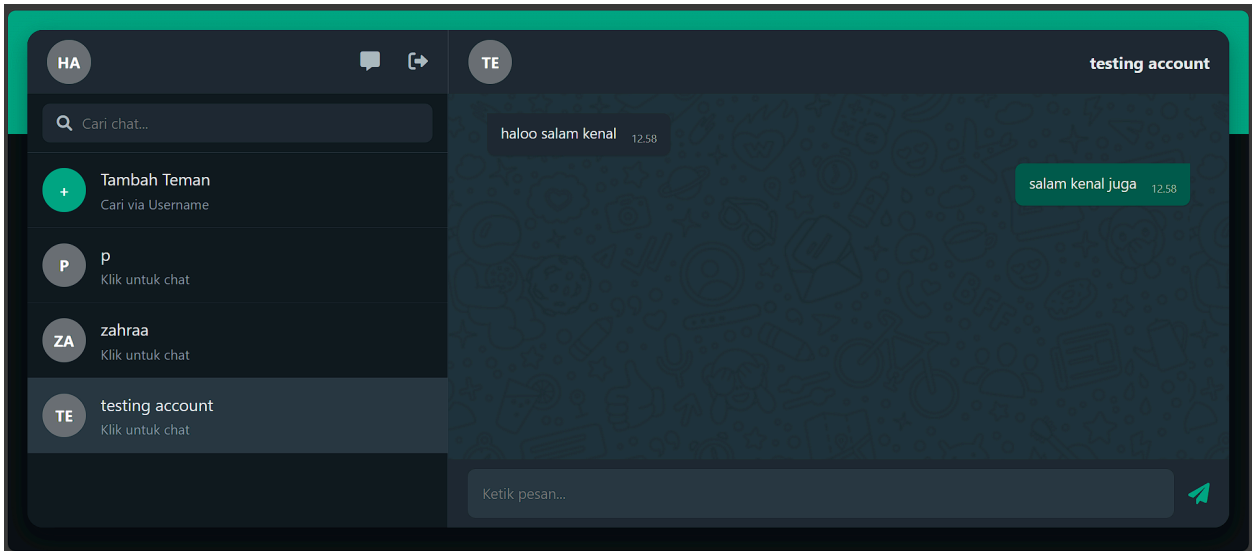
Tampilan Tambah Teman :

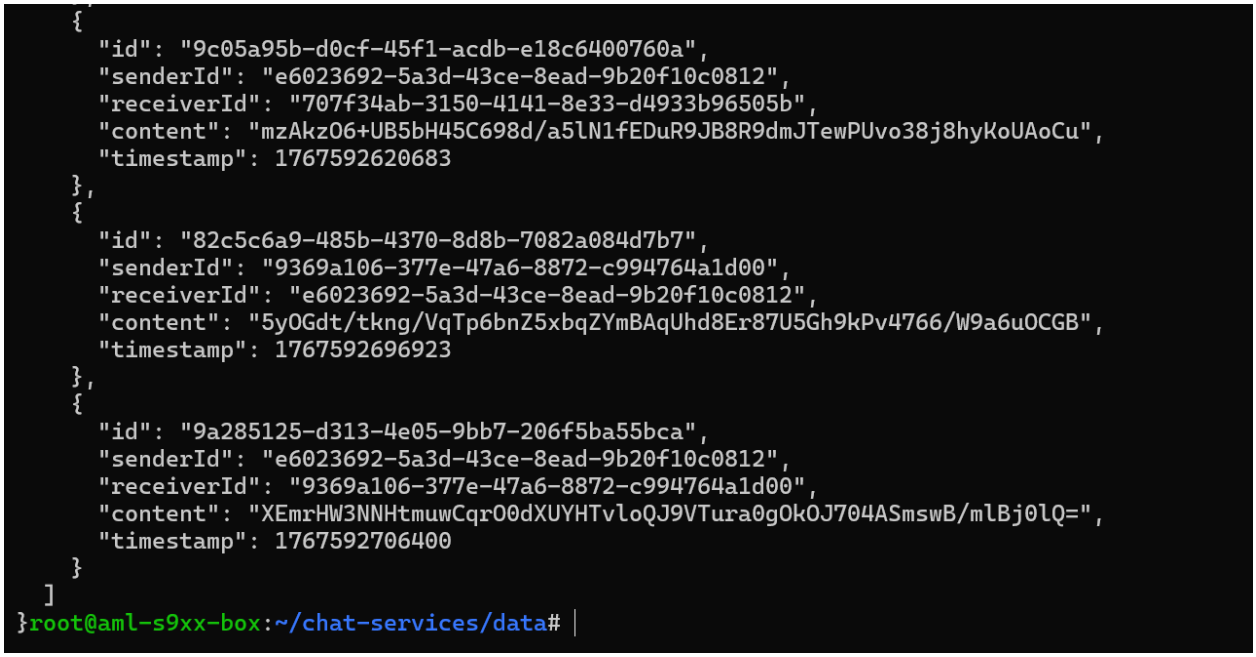


Tampilan Ubah Password



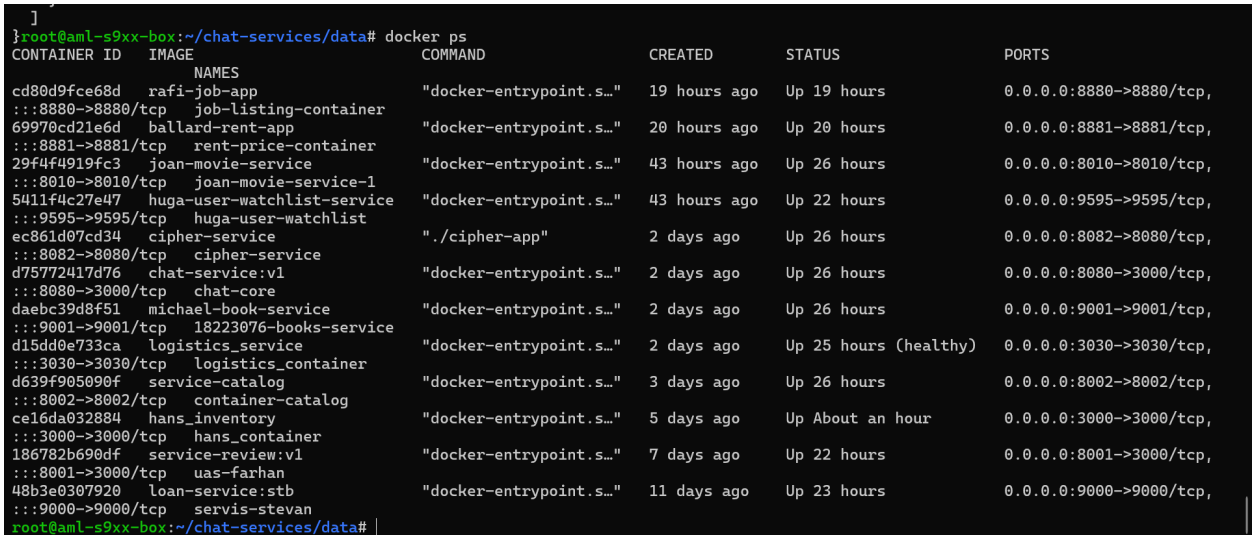
2. Bukti Keamanan Data





Pembuktian mekanisme enkripsi *Data-at-Rest*. Pesan pada antarmuka pengguna (atas) tersimpan sebagai *ciphertext* heksadesimal pada basis data (bawah), membuktikan isolasi data.

3. Bukti Integrasi & Arsitektur



```
  ]
}root@aml-s9xx-box:~/chat-services/data# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED        STATUS                    PORTS
                  NAMES
cd80d9fce68d   rafi-job-app                   "docker-entrypoint.s…"   19 hours ago   Up 19 hours               0.0.0.0:8880->8880/tcp,
:::8880->8880/tcp   job-listing-container
69970cd21e6d   ballard-rent-app               "docker-entrypoint.s…"   20 hours ago   Up 20 hours               0.0.0.0:8881->8881/tcp,
:::8881->8881/tcp   rent-price-container
29f4f4919fc3   joan-movie-service             "docker-entrypoint.s…"   43 hours ago   Up 26 hours               0.0.0.0:8010->8010/tcp,
:::8010->8010/tcp   joan-movie-service-1
5411f4c27e47   huga-user-watchlist-service    "docker-entrypoint.s…"   43 hours ago   Up 22 hours               0.0.0.0:9595->9595/tcp,
:::9595->9595/tcp   huga-user-watchlist
ec861d07cd34   cipher-service                 "./cipher-app"           2 days ago     Up 26 hours               0.0.0.0:8082->8080/tcp,
:::8082->8080/tcp   cipher-service
d75772417d76   chat-service:v1                "docker-entrypoint.s…"   2 days ago     Up 26 hours               0.0.0.0:8080->3000/tcp,
:::8080->3000/tcp   chat-core
daebc39d8f51   michael-book-service           "docker-entrypoint.s…"   2 days ago     Up 26 hours               0.0.0.0:9001->9001/tcp,
:::9001->9001/tcp   18223076-books-service
d15dd0e733ca   logistics_service              "docker-entrypoint.s…"   2 days ago     Up 25 hours (healthy)     0.0.0.0:3030->3030/tcp,
:::3030->3030/tcp   logistics_container
d639f905090f   service-catalog                "docker-entrypoint.s…"   3 days ago     Up 26 hours               0.0.0.0:8002->8002/tcp,
:::8002->8002/tcp   container-catalog
ce16da032884   hans_inventory                 "docker-entrypoint.s…"   5 days ago     Up About an hour          0.0.0.0:3000->3000/tcp,
:::3000->3000/tcp   hans_container
186782b690df   service-review:v1              "docker-entrypoint.s…"   7 days ago     Up 22 hours               0.0.0.0:8001->3000/tcp,
:::8001->3000/tcp   uas-farhan
48b3e0307920   loan-service:stb               "docker-entrypoint.s…"   11 days ago    Up 23 hours               0.0.0.0:9000->9000/tcp,
:::9000->9000/tcp   servis-stevan
root@aml-s9xx-box:~/chat-services/data#
```

Status kontainer Docker pada lingkungan STB, menunjukkan layanan chat-service dan cipher-service berjalan secara terisolasi namun terhubung dalam satu jaringan virtual.

LAMPIRAN

Link layanan terintegrasi : https://chat-services.vercel.app/

API Message services : http://hakim.tugastst.my.id/

API docs Message Services : https://hakim.tugastst.my.id/api-docs/

API cipher services : http://radit.tugastst.my.id/

src API Message services : hakimmudzaki/chat-services

Src frontend : hakimmudzaki/chat-services/tree/main/chat-frontend

Link youtube : https://www.youtube.com/watch?v=PN2mIK4VamY

Link folder dokumen laporan 1 dan 2 : UAS TST