BAB III

METODE PELAKSANAAN MBKM KEWIRAUSAHAAN

3.1 Produksi

Proses produksi dalam program MBKM Kewirausahaan yang dilaksanakan oleh tim pengembang HELLOPET! berfokus pada pengembangan solusi digital berbasis teknologi untuk sektor perawatan dan kesehatan hewan peliharaan. Tidak seperti proses produksi konvensional yang berkaitan dengan manufaktur atau pembuatan produk fisik, produksi dalam konteks ini lebih menitikberatkan pada aktivitas pengembangan sistem informasi berbasis *web*, mulai dari tahap perancangan hingga implementasi dan pengujian.

Platform HELLOPET! dikembangkan sebagai sarana layanan daring yang memungkinkan para pecinta hewan untuk melakukan konsultasi online dengan dokter hewan, melakukan pemesanan layanan seperti grooming dan penitipan, serta mengakses berbagai konten edukatif seputar perawatan hewan. Proses produksi yang dijalankan bertujuan untuk membentuk ekosistem digital yang menyatukan *pet owner*, penyedia layanan hewan, dan dokter hewan dalam satu wadah terintegrasi.

Secara teknis, tahapan produksi dibagi menjadi beberapa elemen utama yang mencakup: perancangan arsitektur sistem, pengembangan *backend* dan *frontend*, integrasi layanan *real-time*, penerapan keamanan sistem, serta pengujian dan penyempurnaan produk. Pengembangan dilakukan menggunakan stack teknologi modern, yaitu Node.js dan Express.js di sisi *backend*, Vue.js untuk *frontend*, dan PostgreSQL sebagai sistem manajemen basis data.

3.1.1. Produksi Digital

Proses produksi digital platform HELLOPET! melibatkan sejumlah tahapan yang saling terintegrasi secara sistematis dalam pengembangan perangkat lunak berbasis web modern. Setiap tahapan dirancang untuk memastikan platform dapat beroperasi dengan optimal, aman, serta

memberikan pengalaman pengguna yang responsif dan relevan terhadap kebutuhan pasar. Rangkaian proses ini mencakup perancangan arsitektur sistem, pengembangan *backend*, penerapan komunikasi *real-time*, integrasi *frontend* dengan *backend*, hingga implementasi aspek keamanan sistem secara menyeluruh.

a. Perancangan Arsitektur Sistem Aplikasi

Perancangan arsitektur sistem merupakan tahap awal yang sangat krusial dalam proses pengembangan platform. Kegiatan ini dimulai dengan melakukan identifikasi dan analisis mendalam terhadap kebutuhan sistem, baik dari sisi fungsionalitas (fungsi yang harus dijalankan sistem) maupun non-fungsionalitas (seperti keamanan, performa, dan skalabilitas). Proses ini dilakukan melalui pendekatan teknik pemodelan sistem yang menghasilkan berbagai artefak visual dan dokumentatif.

Beberapa artefak teknis yang dikembangkan antara lain:

1. ***Use Case* Diagram**, yang menggambarkan relasi antara pengguna (user) dan fungsionalitas sistem;
2. ***Activity* Diagram**, untuk menunjukkan alur aktivitas dari berbagai proses dalam sistem secara runtut;
3. ***Entity Relationship* Diagram (ERD)**, yang digunakan untuk mendesain struktur basis data relasional agar efisien dan mudah dipelihara.

Arsitektur sistem HELLOPET! didesain dengan pendekatan *modular* dan *scalable*, yang artinya sistem dibangun dari komponen-komponen kecil yang bisa dikembangkan secara independen dan dapat diintegrasikan kembali menjadi satu kesatuan. Pendekatan ini memudahkan penambahan fitur di masa mendatang dan memungkinkan integrasi dengan sistem atau layanan pihak ketiga, seperti *payment gateway*, layanan API eksternal, atau chatbot layanan pelanggan.

b. Pengembangan Minimum Viable Product (MVP)

Tahapan berikutnya adalah pengembangan sisi server (*backend*) menggunakan bahasa pemrograman **JavaScript** yang dijalankan dalam lingkungan runtime **Node.js**, serta menggunakan framework **Express.js** yang ringan dan fleksibel. Penggunaan teknologi ini memungkinkan proses pembangunan *server* dilakukan secara cepat, efisien, dan mudah dikembangkan karena berbasis *event-driven architecture* yang cocok untuk aplikasi berskala besar dan *real-time* seperti HELLOPET!.

Fungsi utama dari *backend* ini adalah menyediakan berbagai *endpoint* **RESTful API** yang melayani proses-proses penting seperti:

1. Registrasi dan autentikasi pengguna;
2. Konsultasi online antara pengguna dan mitra dokter;

Setiap endpoint menggunakan format data **JSON (JavaScript *Object Notation*)** untuk memastikan interoperabilitas tinggi dengan *frontend* dan sistem eksternal lainnya. Struktur logika pemrograman *backend* juga memanfaatkan arsitektur *Model-Controller-Service* agar mudah dibaca dan dikelola oleh tim pengembang.

c. Integrasi Sistem Konsultasi

Fitur *real-time* menjadi keunggulan utama dalam platform HELLOPET!, terutama dalam layanan konsultasi kesehatan hewan secara daring. Untuk mewujudkan fitur komunikasi langsung antara pengguna dan dokter hewan, dikembangkan sistem WebSocket menggunakan pustaka **Socket.IO**, yang memungkinkan komunikasi dua arah secara instan tanpa perlu memuat ulang halaman.

Beberapa fitur *real-time* yang dibangun dalam sistem ini meliputi:

- ***Live Chat***: Pengguna dan dokter dapat bertukar pesan secara langsung melalui antarmuka obrolan;
- ***Typing Indicator***: Notifikasi yang menunjukkan ketika pengguna sedang mengetik pesan;

- **Status Koneksi**: Sistem dapat mendeteksi apakah pengguna atau dokter sedang *online* atau *offline*;
- **Manajemen Socket**: Menggunakan struktur data *userSocketMap* untuk memetakan ID pengguna dengan socket aktif, dan *typingUsers* untuk melacak siapa yang sedang mengetik.

Sistem ini berjalan secara dinamis dan dirancang agar tetap responsif dalam menangani banyak koneksi secara bersamaan, mendukung kualitas layanan yang interaktif dan profesional.

d. Integrasi API di Frontend Menggunakan Vue.js

Sisi antarmuka pengguna atau *frontend* dikembangkan dengan menggunakan Vue.js, framework JavaScript progresif yang mendukung pengembangan antarmuka pengguna yang reaktif, dinamis, dan modular. Komponen *frontend* dirancang agar mudah dinavigasi dan ramah pengguna, memungkinkan *pet owners* untuk melakukan pemesanan, berkonsultasi, atau menjelajahi layanan dengan mudah dan cepat.

Integrasi data dari *backend* dilakukan menggunakan pustaka Axios, yang digunakan untuk melakukan permintaan HTTP ke *endpoint* API. Data yang diperoleh dari backend akan diproses dan ditampilkan dalam bentuk antarmuka visual yang terstruktur dengan baik. Selain itu, sistem state management pada frontend menggunakan Pinia, yang membantu menyimpan status pengguna seperti *login session*, detail profil, serta data layanan yang sedang diakses. Pendekatan ini memungkinkan pengalaman pengguna yang *seamless* dan konsisten, serta memudahkan dalam menjaga integritas data antar komponen aplikasi.

e. Pengujian dan Evaluasi

Keamanan merupakan aspek yang sangat penting dalam pengembangan aplikasi digital, terlebih ketika aplikasi menangani data pribadi pengguna, termasuk informasi identitas, data transaksi,

dan riwayat konsultasi. Untuk itu, diterapkan berbagai praktik keamanan *backend* yang telah terbukti efektif dalam melindungi sistem dari berbagai ancaman siber.

Beberapa langkah implementatif yang digunakan antara lain:

- ***Hashing password* menggunakan *bcrypt***: Setiap kata sandi yang dimasukkan pengguna akan dienkripsi terlebih dahulu menggunakan algoritma hashing *bcrypt* sebelum disimpan ke *database*, sehingga tidak dapat dibaca oleh pihak tidak berwenang;
- **Autentikasi menggunakan JWT (*JSON Web Token*)**: Sistem otorisasi berbasis token digunakan untuk mengamankan akses ke *endpoint* privat;
- ***Middleware* autentikasi**: *Endpoint* yang bersifat sensitif hanya dapat diakses oleh pengguna yang telah berhasil *login* dan membawa token valid;
- **Input *validation***: Sistem secara aktif memfilter dan memvalidasi input dari pengguna untuk mencegah serangan seperti SQL *Injection atau Cross-Site Scripting* (XSS);
- ***Rate limiting***: Digunakan untuk membatasi jumlah permintaan ke server dalam periode tertentu guna mencegah serangan DDoS atau *brute-force*.

Seluruh tahapan produksi digital ini berjalan dalam kerangka kerja yang kolaboratif, berorientasi pada pengguna, dan berlandaskan pada prinsip-prinsip pengembangan perangkat lunak modern yang adaptif dan tangguh terhadap perubahan.

3.1.2. Pengemasan Layanan

Sebagai platform layanan digital yang bergerak di bidang perawatan dan kesehatan hewan peliharaan, HELLOPET! menerapkan strategi pengemasan layanan yang tidak hanya berfokus pada fungsi, tetapi juga pada pengalaman pengguna secara keseluruhan. Teknik pengemasan ini dirancang untuk menciptakan kesan profesional, ramah, dan terpercaya

yang mampu menarik dan mempertahankan pengguna melalui antarmuka dan interaksi yang intuitif. Pendekatan ini mencakup beberapa aspek utama berikut:

a. **Branding dan Identitas Visual**

   HELLOPET! mengedepankan identitas visual yang konsisten, profesional, dan bersahabat, mencerminkan nilai empati terhadap hewan peliharaan serta rasa aman bagi pemiliknya. Logo dirancang dengan karakter yang ramah dan mudah dikenali, sementara palet warna yang digunakan cenderung lembut dan menenangkan, menciptakan suasana yang nyaman selama pengguna berinteraksi dengan aplikasi. Semua elemen desain UI (*User Interface*), termasuk ikonografi, tipografi, dan ilustrasi, dikurasi secara konsisten untuk memperkuat brand image HELLOPET! sebagai solusi layanan hewan modern dan terpercaya.

b. **Pengelompokan dan Presentasi Layanan**

   Untuk meningkatkan keterbacaan dan kemudahan navigasi dalam aplikasi, layanan pada HELLOPET! dikategorikan dengan jelas berdasarkan jenisnya, seperti layanan grooming dan konsultasi dokter. Setiap layanan dilengkapi dengan deskripsi detail, informasi harga, lokasi mitra penyedia jasa, serta ulasan dari pengguna lain sebagai bentuk transparansi dan validasi sosial. Penyajian ini memungkinkan pengguna untuk mengambil keputusan secara cepat dan tepat berdasarkan kebutuhan serta preferensi mereka.

c. **Konten Informasi dan Edukasi**

   Selain berperan sebagai platform layanan, HELLOPET! juga berfungsi sebagai media edukasi bagi pemilik hewan. Aplikasi menyediakan berbagai artikel singkat, tips perawatan, dan informasi kesehatan hewan yang dikurasi langsung oleh mitra dokter hewan. Konten ini disusun untuk meningkatkan kesadaran pengguna akan pentingnya perawatan hewan yang baik, sekaligus memperkuat citra platform sebagai sumber informasi yang kredibel dan bermanfaat.

3.1.3. Teknologi yang Digunakan

Dalam membangun platform digital HELLOPET!, pemilihan teknologi dilakukan dengan pertimbangan mendalam terhadap performa, skalabilitas, kemudahan integrasi, serta efisiensi pengembangan. Seluruh elemen teknologi yang digunakan berasal dari ekosistem modern dan berbasis *open-source*, sehingga memudahkan proses pengembangan, pemeliharaan, dan pengembangan lanjutan di masa depan. Berikut adalah daftar teknologi yang digunakan beserta fungsinya masing-masing:

Tabel 3. 1 Teknologi yang digunakan

Komponen	Teknologi	Fungsi dan Alasan Penggunaan
Frontend	Vue.js	Pembangunan antarmuka dinamis, ringan, dan modular
Backend	Node.js + Express.js	Pembuatan *REST* API yang cepat dan efisien
Database	PostgreSQL	Penyimpanan data relasional yang konsisten dan stabil
WebSocket	Socket.IO	Komunikasi *real-time* untuk fitur *chat* konsultasi
Payment Gateway	Midtrans	Pembayaran aman dan mendukung metode lokal
Messaging	WhatsApp API	Pemesanan layanan grooming

3.2 Penetapan Harga

Penetapan harga pada platform HELLOPET! dilakukan secara strategis dan berbasis data, dengan mempertimbangkan berbagai variabel penting guna memastikan keseimbangan antara keterjangkauan bagi pengguna, daya saing di pasar, dan keberlanjutan finansial bagi mitra penyedia layanan. Pendekatan yang

digunakan bersifat holistik, menggabungkan analisis terhadap harga pasar, segmentasi dan daya beli pengguna, nilai tambah dari layanan berbasis teknologi, serta struktur biaya dan margin keuntungan yang realistis. Selain itu, fleksibilitas juga diberikan kepada mitra dalam menentukan harga dasar layanan mereka. Strategi penetapan harga ini dirancang untuk menciptakan fondasi yang kuat sejak tahap awal peluncuran (*Minimum Viable Product*) dan dapat disesuaikan secara dinamis seiring pertumbuhan platform. Dalam implementasinya, HELLOPET! mengadopsi tiga pendekatan utama, yaitu benchmarking terhadap harga kompetitor yang telah memiliki basis pelanggan di wilayah Jabodetabek, survei preferensi dan daya beli pengguna potensial, serta penerapan model pembagian pendapatan yang adil melalui sistem komisi per transaksi. Pendekatan ini tidak hanya memberikan keunggulan kompetitif dari sisi harga, tetapi juga membangun ekosistem layanan yang berkelanjutan bagi seluruh pihak yang terlibat.

3.2.1. Metode Benchmarking Harga Kompetitor

HELLOPET! melakukan proses benchmarking secara sistematis terhadap platform kompetitor yang telah memiliki pangsa pasar di wilayah Jabodetabek, seperti Pet Backer, Pawsh Pet Care, dan PetsKita. Analisis ini bertujuan untuk memahami standar harga pasar dan menemukan celah kompetitif yang dapat dimanfaatkan untuk menarik pengguna awal (*early adopters*).

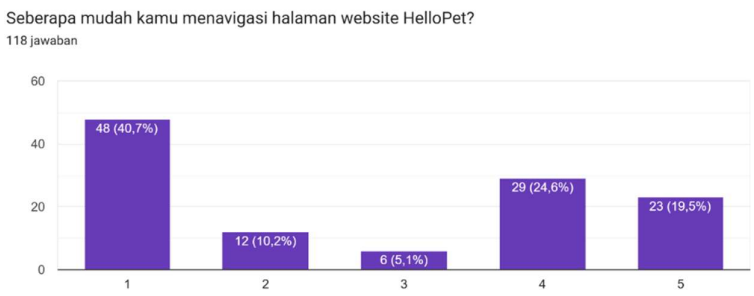
Tabel 3. 2 Benchmarking Harga Kompetitor

Jenis Layanan	**HELLOPET! (Target Harga)**	**Pet Backer (Rata-rata)**	**PetsKita (Rata-rata)**
Grooming	Rp85.000 – Rp120.000	Rp95.000 – Rp130.000	Rp100.000 – Rp140.000
Penitipan Harian	Rp60.000 – Rp90.000	Rp70.000 – Rp100.000	Rp80.000 – Rp110.000
Konsultasi Online	Rp30.000 – Rp50.000	Rp50.000 – Rp80.000	Rp60.000 – Rp90.000
Jenis Layanan	HELLOPET! (Target Harga)	Pet Backer (Rata-rata)	PetsKita (Rata-rata)

3.2.2. Survei Preferensi dan Daya Beli Pengguna Potensial

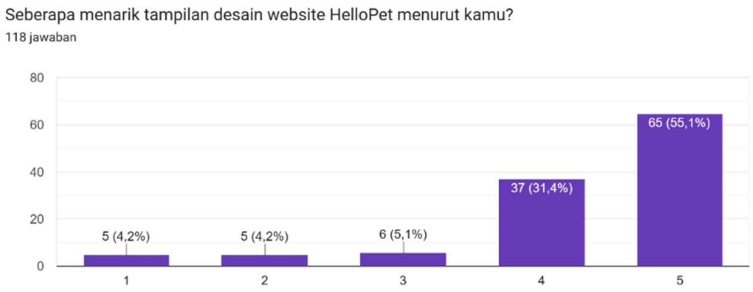
Selain analisis pasar eksternal, HELLOPET! juga melakukan validasi harga melalui survei daring terhadap 118 responden yang merupakan pemilik hewan peliharaan berusia 20–35 tahun di kawasan Jabodetabek. Survei ini bertujuan untuk mengukur persepsi nilai, kesediaan membayar, dan preferensi layanan dari segmen pengguna sasaran. Temuan utama dari survei adalah sebagai berikut:

a Sebanyak 48% responden menyatakan bahwa navigasi pada halaman *website* HELLOPET! sangat mudah digunakan, sehingga menunjukkan bahwa struktur menu dan alur interaksi telah sesuai dengan ekspektasi pengguna umum.



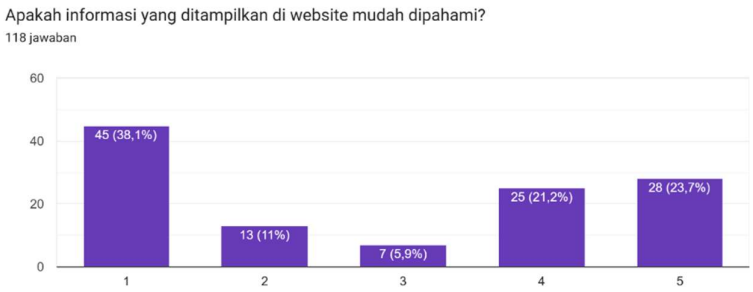
Gambar 3. 1 Hasil Responden Navigasi

b Sebanyak 65% responden menilai tampilan desain *website* HELLOPET! sangat menarik, yang mengindikasikan bahwa pemilihan elemen visual, seperti warna, *layout*, dan tipografi, telah berhasil membangun kesan profesional dan ramah pengguna.



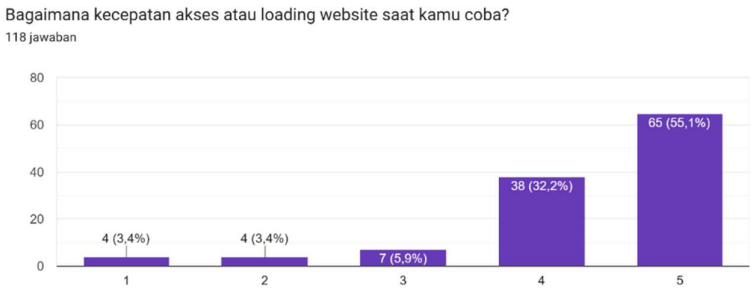
Gambar 3. 2 Hasil Responden Tampilan

c Sebanyak 45% responden merasa bahwa informasi yang disajikan pada website HELLOPET! sangat mudah dipahami, menunjukkan efektivitas dalam penyampaian konten dan penyusunan bahasa yang komunikatif.



Gambar 3. 3 Hasil Responden Informasi

d Sebanyak 65% responden merasa sangat cepat saat mengakses website HELLOPET!, menandakan performa *website* dari sisi teknis cukup optimal dan mendukung kenyamanan pengguna.



Gambar 3. 4 Hasil Responden Kecepatan

e Sebanyak 83,1% responden menyatakan ketertarikan untuk menggunakan *website* HELLOPET! secara langsung, yang menunjukkan bahwa secara umum, platform ini telah memenuhi harapan dan kebutuhan utama dari target pengguna.

Gambar 3. 5 Hasil Responden Ketertarikan pada HELLOPET!

Berdasarkan temuan tersebut, dapat disimpulkan bahwa HELLOPET! memiliki peluang besar untuk diterima oleh pasar, terutama dari kalangan pengguna yang menghargai kecepatan akses, tampilan antarmuka yang menarik, kemudahan navigasi, dan penyampaian informasi yang jelas. Hasil validasi ini juga memperkuat arah pengembangan produk selanjutnya, baik dari sisi fungsionalitas maupun strategi komunikasi yang lebih tepat sasaran. Data ini menunjukkan adanya peluang yang kuat untuk menargetkan segmen ekonomi menengah yang mengedepankan efisiensi biaya dan kemudahan akses terhadap layanan digital.

3.2.3. Struktur Harga Mitra dan Model Pendapatan Platform

Untuk menjaga fleksibilitas dan kolaborasi yang adil, HELLOPET! menerapkan model komisi per transaksi, di mana mitra tetap memiliki otonomi dalam menetapkan harga dasar layanan mereka. Struktur harga dibagi menjadi tiga lapisan:

- **Harga Dasar oleh Mitra**

  Mitra menentukan harga layanan berdasarkan standar operasional dan biaya internal mereka masing-masing.

- **Markup Platform**

  HELLOPET! menambahkan margin 10–15% dari harga dasar sebagai biaya layanan platform. Margin ini mencakup pengembangan sistem digital, pemeliharaan server, biaya pemrosesan pembayaran, dan kegiatan pemasaran.

- **Harga Akhir kepada Konsumen**

  Pengguna membayar harga final melalui sistem pembayaran terintegrasi. Selanjutnya, HELLOPET! melakukan pembayaran kepada mitra secara berkala melalui sistem *payout* mingguan yang transparan dan terotomatisasi.

3.3 Promosi Target Pasar

Dalam proses pengembangan dan ekspansi sebuah platform digital, khususnya yang bergerak dalam sektor layanan kesehatan dan perawatan hewan seperti HELLOPET!, penentuan target pasar serta strategi promosi merupakan elemen yang sangat esensial dan strategis. Keberhasilan produk digital bukan hanya bergantung pada kecanggihan teknologi yang digunakan atau keunikan fitur yang ditawarkan, tetapi juga pada bagaimana produk tersebut diposisikan, diperkenalkan, dan disampaikan kepada konsumen yang tepat. Dengan kata lain, memahami siapa yang menjadi pengguna utama dan bagaimana cara menjangkau mereka secara efektif menjadi bagian yang tidak terpisahkan dari keberhasilan bisnis secara menyeluruh.

Sebagai sebuah platform layanan berbasis digital yang dirancang untuk

memenuhi kebutuhan pecinta hewan peliharaan, HELLOPET! menghadirkan solusi modern yang terintegrasi untuk konsultasi dengan dokter hewan secara daring (*online*), akses ke layanan grooming terpercaya, edukasi kesehatan hewan, serta fitur interaktif lain yang dapat memperkuat ikatan antara pengguna dan hewan peliharaan mereka. Untuk itu, pendekatan pemasaran yang digunakan tidak hanya berfokus pada perluasan jangkauan audiens semata, tetapi juga pada penciptaan nilai tambah, keterlibatan emosional (emotional engagement), serta kenyamanan dalam setiap interaksi pengguna dengan platform.

A. Target Pasar HELLOPET!

Penetapan target pasar dari HELLOPET! dilakukan melalui pendekatan kuantitatif dan kualitatif, dimulai dari survei internal yang melibatkan 120 responden dari kalangan pemilik hewan peliharaan di wilayah urban, khususnya Jabodetabek. Survei ini bertujuan untuk menggali karakteristik pengguna potensial, mulai dari latar belakang demografis, gaya hidup, kebiasaan penggunaan teknologi, hingga preferensi terhadap layanan kesehatan dan perawatan hewan. Hasil dari survei ini kemudian dikombinasikan dengan data eksternal seperti laporan dari Badan Pusat Statistik (BPS) dan studi industri dari Statista pada tahun 2023. Berdasarkan hasil analisis yang komprehensif tersebut, maka disusunlah segmentasi pasar utama HELLOPET! sebagai berikut:

Tabel 3. 3 Segmentasi Pasar

Kriteria Segmentasi	Deskripsi Terperinci
Demografis	Laki-laki dan perempuan dengan rentang usia antara 20 hingga 35 tahun, berasal dari kalangan mahasiswa, pekerja kantoran, maupun wirausaha yang memiliki daya beli menengah ke atas. Mereka cenderung memiliki gaya hidup yang aktif, dinamis, dan modern.
Geografis	Wilayah dengan kepadatan penduduk tinggi dan infrastruktur digital yang sudah mapan seperti Jakarta, Depok,

Kriteria Segmentasi	Deskripsi Terperinci
	Tangerang, dan Bekasi, yang dikenal memiliki jumlah pemilik hewan peliharaan yang tinggi.
Psikografis	Individu yang mencintai hewan peliharaan (terutama kucing dan anjing), memiliki tingkat empati yang tinggi terhadap hewan, serta bersedia mengalokasikan waktu dan dana untuk kesehatan, kenyamanan, serta kesejahteraan hewan mereka.
Perilaku Konsumen	Terbiasa menggunakan layanan berbasis digital dalam kehidupan sehari-hari, seperti aplikasi pesan makanan, transportasi online, serta layanan kesehatan digital. Mereka juga aktif di media sosial, cepat merespons tren baru, dan cenderung mencari kemudahan serta efisiensi dalam menggunakan layanan.

Menurut data BPS, lebih dari 62% rumah tangga urban di Indonesia memiliki hewan peliharaan, dan mayoritas dari mereka mengandalkan informasi serta layanan digital untuk mengelola perawatan hewan tersebut. Selain itu, laporan dari Statista menunjukkan bahwa industri perawatan hewan di Indonesia diproyeksikan akan mencapai nilai pasar sebesar *USD* 1,1 miliar pada tahun 2025, menjadikan Indonesia sebagai salah satu pasar *pet-care* terbesar dan paling berkembang di kawasan Asia Tenggara. Temuan ini menunjukkan bahwa segmen pasar yang disasar oleh HELLOPET! tidak hanya relevan, tetapi juga memiliki potensi pertumbuhan yang sangat besar dan menjanjikan.

B. Strategi Promosi

Agar mampu menjangkau target pasar yang telah ditetapkan secara tepat sasaran dan efisien, HELLOPET! merancang strategi promosi

komprehensif yang terdiri dari pendekatan digital, kemitraan strategis, insentif loyalitas, serta keterlibatan dalam komunitas offline. Strategi promosi ini disusun secara sistematis dan berbasis data dengan mempertimbangkan perilaku konsumen digital masa kini yang mengutamakan kenyamanan, kecepatan, dan personalisasi dalam berinteraksi dengan layanan digital. Berikut ini adalah rincian dari strategi yang diterapkan:

1. **Pemasaran Digital (Digital Marketing)**

   a) **Optimalisasi Media Sosial (Instagram, TikTok, YouTube *Shorts*)**

   Menggunakan platform media sosial populer untuk membangun keterlibatan emosional dengan audiens melalui konten edukatif, hiburan, tips perawatan hewan, hingga konten interaktif berbentuk video testimonial dan *behind the scene*. Target utamanya adalah menciptakan koneksi langsung antara pengguna dengan nilai yang diusung oleh *HELLOPET!*, serta mencapai lebih dari 10.000 impresi setiap bulannya.

   b) **SEO dan Google Ads**

   Optimalisasi mesin pencari dilakukan melalui penerapan teknik *Search Engine Optimization* (SEO) dan kampanye Google *Ads* untuk kata kunci strategis seperti "dokter hewan online", "grooming kucing terbaik", dan "pet care digital Jakarta". Strategi ini bertujuan untuk meningkatkan lalu lintas organik ke situs web *HELLOPET!* serta memperbesar peluang konversi pengguna baru.

   c) **Meta Ads (Facebook & Instagram Ads)**

   Penerapan iklan berbayar dilakukan dengan menyasar audiens yang memiliki minat terhadap hewan peliharaan dalam radius 30 km dari pusat kota. Target audiens disegmentasi lebih lanjut berdasarkan usia, minat, dan

kebiasaan digital mereka untuk menghasilkan hasil promosi yang lebih presisi.

2. **Kemitraan dan Kolaborasi**
    a) **Kolaborasi dengan Pet Influencer dan Micro-KOL** Menggandeng para influencer pecinta hewan dengan jumlah pengikut aktif 5.000–50.000 yang memiliki tingkat *engagement* tinggi untuk mempromosikan *HELLOPET!* melalui testimoni dan konten edukatif yang bersifat organik. Strategi ini terbukti mampu meningkatkan kepercayaan calon pengguna dan memperluas jangkauan audiens secara natural.
    b) **Kemitraan Strategis dengan Pet Shop dan Klinik Hewan** Menyediakan materi promosi fisik berupa *leaflet*, stiker *QR code*, dan banner digital yang diletakkan di area kasir dan ruang tunggu klinik hewan. Tujuannya adalah untuk menjaring pengguna potensial yang sedang berada dalam konteks layanan perawatan hewan.
3. **Program Referral dan Insentif Loyalitas**
    a) Penerapan sistem referral di mana pengguna aktif yang berhasil mengajak pengguna baru akan memperoleh diskon 10% untuk layanan berikutnya.
    b) Implementasi *loyalty program* berbasis poin, di mana setiap transaksi menghasilkan poin yang bisa dikumpulkan dan ditukarkan dengan diskon atau layanan gratis. Program ini dirancang untuk meningkatkan retensi pengguna jangka panjang.
4. **Keterlibatan Komunitas dan Kegiatan Offline**
    a) Berpartisipasi dalam kegiatan komunitas pecinta hewan seperti *event* vaksinasi massal, pameran adopsi hewan, dan kompetisi *pet fashion show*. Dalam acara tersebut, HELLOPET! membuka booth promosi, menyediakan

souvenir, serta mengedukasi masyarakat mengenai manfaat layanan digital pet-care.

C. Evaluasi dan Monitoring Promosi

Seluruh aktivitas promosi yang telah dilaksanakan akan terus dipantau dan dievaluasi melalui pendekatan berbasis data. Evaluasi dilakukan menggunakan indikator-indikator kinerja utama (*Key Performance Indicators* / KPI) yang telah ditentukan, untuk menilai efektivitas strategi serta mengetahui aspek mana yang perlu ditingkatkan. Adapun indikator yang digunakan antara lain sebagai berikut:

Tabel 3. 4 Evaluasi KPI

Indikator Evaluasi	Target yang Ditetapkan
Jumlah Pengguna Baru	Minimal 500 pengguna aktif dalam kurun waktu 3 bulan pertama
Engagement Rate Konten Sosial	Minimal 6% di platform Instagram dan TikTok
Cost per Acquisition (CPA)	Maksimal Rp15.000 untuk setiap pengguna baru
Tingkat Retensi Pengguna Bulanan	Minimal 60% pengguna melakukan transaksi ulang di bulan berikutnya

Untuk mengukur dan memantau seluruh aktivitas ini, digunakan berbagai alat bantu analitik seperti Google *Analytics* untuk memantau perilaku pengguna di situs web, Meta Business Suite untuk mengevaluasi performa iklan dan konten di media sosial, serta dashboard internal milik HELLOPET! yang mencatat data transaksi dan penggunaan layanan. Evaluasi ini dilakukan secara berkala guna memastikan bahwa setiap aktivitas promosi yang dilakukan memberikan kontribusi nyata terhadap pertumbuhan pengguna, loyalitas pelanggan, dan pencapaian tujuan bisnis secara keseluruhan.

3.4 Tahapan Pekerjaan yang Dilakukan Dalam MBKM Kewirausahaan

Pelaksanaan kerja dalam program MBKM Kewirausahaan untuk pengembangan platform digital HELLOPET! dilakukan secara sistematis, terstruktur, dan bertahap guna memastikan setiap aspek pengembangan produk dapat berjalan dengan efektif dan efisien. Sebagai *Chief Technology Officer* (CTO), penulis bertanggung jawab penuh dalam perancangan dan implementasi aspek teknologi, yang mencakup sisi *backend*, *frontend*, *database*, hingga integrasi sistem. Alur kerja pengembangan platform HELLOPET! dibagi menjadi lima fase utama yang saling terintegrasi, yaitu: Perencanaan, Pengembangan, Integrasi, Keamanan dan Pengujian, serta Evaluasi.

1) Tahap Perencanaan

   Fase ini menjadi pondasi awal yang sangat penting dalam membangun sistem HELLOPET!. Kegiatan dalam tahap ini mencakup proses identifikasi kebutuhan pengguna (user requirement) serta kebutuhan bisnis dari sisi penyedia layanan. Seluruh hasil analisis kemudian diterjemahkan ke dalam bentuk desain teknis yang konkret, seperti pemilihan arsitektur sistem, penentuan alur kerja aplikasi (*flow diagram*), perancangan struktur *database* menggunakan PostgreSQL, hingga pemilihan tumpukan teknologi (*technology stack*) seperti Node.js untuk *backend* dan Vue.js untuk *frontend*. Fase ini juga mencakup perancangan fitur utama seperti konsultasi online, integrasi pembayaran Midtrans dan *dashboard* dokter.

2) Tahap Pengembangan

   Tahapan ini melibatkan proses penulisan kode (*coding*) dan implementasi logika bisnis. Pengembangan dilakukan dengan pendekatan *modular* dan *scalable*, dimulai dari sisi *backend* menggunakan Node.js dan Express.js. Fungsi-fungsi penting seperti autentikasi pengguna (login, register), pengelolaan data dokter dan pengguna, manajemen sesi konsultasi, hingga integrasi dengan

WebSocket untuk komunikasi *real-time* dibangun secara bertahap. Backend dirancang mengikuti prinsip RESTful API dan didukung oleh struktur *folder* yang rapi agar memudahkan pemeliharaan jangka panjang. Sementara itu, *frontend* dikembangkan menggunakan framework Vue.js untuk menciptakan antarmuka pengguna yang interaktif, responsif, dan mudah digunakan.

3) Tahap Integrasi

Setelah *backend* dan *frontend* dikembangkan secara terpisah, tahap integrasi menjadi sangat penting untuk menghubungkan keduanya. Pada tahap ini, setiap endpoint API yang telah dibuat di sisi backend mulai digunakan dan diuji pada bagian *frontend*. Proses ini memastikan bahwa semua data dan logika bisnis dapat ditampilkan dan dikontrol langsung dari antarmuka pengguna. Contoh fitur yang diintegrasikan meliputi: tampilan daftar dokter, halaman konsultasi, sistem pembayaran Midtrans, serta sistem login dan register.

4) Tahap Keamanan dan Pengujian

Keamanan sistem menjadi prioritas yang tidak bisa diabaikan dalam platform layanan berbasis teknologi. Pada fase ini dilakukan penerapan beberapa lapisan proteksi *backend*, seperti *middleware* autentikasi dan otorisasi, pengaturan CORS, validasi input dari pengguna, serta implementasi *middleware custom* originCheck yang membatasi akses hanya dari domain resmi HELLOPET!. Selain itu, konfigurasi sensitif juga dikelola menggunakan *file* .env agar data rahasia seperti API *key* tidak terekspos. Seluruh sistem kemudian diuji dengan berbagai skenario penggunaan untuk menemukan dan memperbaiki *bug*, memastikan keamanan data, serta menghindari potensi celah yang dapat dimanfaatkan pihak tidak bertanggung jawab.

5) Tahap Evaluasi

Fase terakhir dalam alur pekerjaan ini adalah evaluasi menyeluruh terhadap semua proses yang telah dilakukan. Evaluasi dilakukan terhadap performa teknis sistem, kestabilan API, serta pengalaman pengguna dari sisi *frontend*. Selain itu, dilakukan pula dokumentasi teknis serta penyusunan laporan sebagai bentuk pertanggungjawaban atas seluruh aktivitas MBKM. Evaluasi ini juga menjadi landasan penting untuk pengembangan lanjutan dan skalabilitas sistem HELLOPET! di masa depan.

Seluruh pekerjaan yang dilakukan selama masa program MBKM, baik dari tahap konseptual hingga tahap implementasi dan evaluasi, direkam secara sistematis dan dijabarkan dalam tabel berikut. Tabel ini merinci pekerjaan mingguan serta proyek utama yang dikerjakan selama program berlangsung.

Tabel 3. 5 Detail Pekerjaan yang Dilakukan Dalam MBKM Cluster Kewirausahaan

No.	Minggu	Proyek	Keterangan
1	1	Perancangan Arsitektur Sistem Aplikasi	Merancang struktur arsitektur teknis HELLOPET!, termasuk alur kerja sistem, pemetaan layanan *backend-frontend*, dan desain *database* PostgreSQL.
2	2	Pengembangan *Backend* API (Node.js + Express)	Membangun RESTful API untuk fitur autentikasi, pengelolaan data dokter, layanan konsultasi, pembayaran, dan histori interaksi dengan menggunakan Node.js dan Express.js.
3	4 - 5	Implementasi WebSocket (*Chat Real-time*)	Mengembangkan fitur komunikasi langsung antara user dan dokter menggunakan WebSocket (socket.io) yang dihubungkan dengan sistem *backend* dan *database*.
4	5 - 8	Penerapan Midtrans *Payment Gateway*	Mengintegrasikan Midtrans sebagai sistem pembayaran digital untuk pengguna yang ingin berkonsultasi

No.	Minggu	Proyek	Keterangan
			dengan dokter, lengkap dengan *callback* dan verifikasi pembayaran.
5	8 - 10	Integrasi API ke *Frontend* (Vue.js)	Menghubungkan *frontend* HELLOPET! yang dibuat dengan Vue.js ke REST API yang telah dibangun, termasuk penanganan data dinamis dan tampilan interaktif.
6	10 - 11	Pengembangan Statistik & *Dashboard* Dokter	Membuat API dan *frontend* untuk visualisasi data statistik, seperti jumlah konsultasi dan *profit* bulanan/tahunan dokter dalam bentuk grafik.
7	11 - 12	Penerapan Keamanan *Backend* (*Middleware*)	Membuat *middleware* custom untuk pemeriksaan origin (originCheck), konfigurasi CORS, pengamanan *endpoint* API, dan penggunaan *file* .env untuk variabel sensitif.

Pelaksanaan kegiatan dalam program MBKM Kewirausahaan pada proyek HELLOPET! dilakukan secara sistematis dan bertahap, mencerminkan pendekatan profesional dalam membangun sebuah produk digital. Seluruh tahapan yang dilalui, mulai dari perencanaan sistem, pengembangan backend dan frontend, integrasi API, penerapan keamanan, hingga proses *deployment* dan evaluasi akhir, menunjukkan bahwa peran CTO sangat krusial dalam memastikan sistem berjalan dengan stabil, aman, dan sesuai kebutuhan pengguna.

Melalui tahapan-tahapan tersebut, berbagai tantangan teknis berhasil diselesaikan dengan pendekatan terstruktur. Penggunaan teknologi modern seperti Node.js, Vue.js, PostgreSQL, serta penerapan CI/CD dan integrasi Midtrans sebagai sistem pembayaran, menjadi bagian penting dalam membentuk platform HELLOPET! sebagai layanan digital yang andal. Selain itu, perhatian terhadap keamanan seperti validasi origin, *middleware* otorisasi, serta penggunaan file konfigurasi .env, turut memperkuat integritas sistem yang dibangun.

3.5 Uraian Pelaksanaan Kerja Dalam MBKM Kewirausahaan

Pada bagian ini, penulis akan menguraikan secara mendalam dan komprehensif mengenai lingkup pekerjaan serta tanggung jawab strategis yang dipegang dalam perannya sebagai *Chief Technology Officer* (CTO) selama mengikuti program MBKM (Merdeka Belajar Kampus Merdeka) dalam cluster Kewirausahaan. Peran ini menempatkan penulis pada posisi sentral dalam proses realisasi visi bisnis startup HELLOPET! dari sebuah konsep menjadi sebuah produk digital yang fungsional dan siap pasar. HELLOPET! sendiri merupakan sebuah platform digital inovatif yang dirancang sebagai jembatan teknologi antara komunitas pemilik hewan peliharaan (*pet lovers*) dengan para profesional di bidang kesehatan dan perawatan hewan. Tujuan utamanya adalah menyediakan akses mudah untuk sesi konsultasi daring bersama dokter hewan terverifikasi dan menyajikan direktori layanan *pet grooming* yang terkurasi.

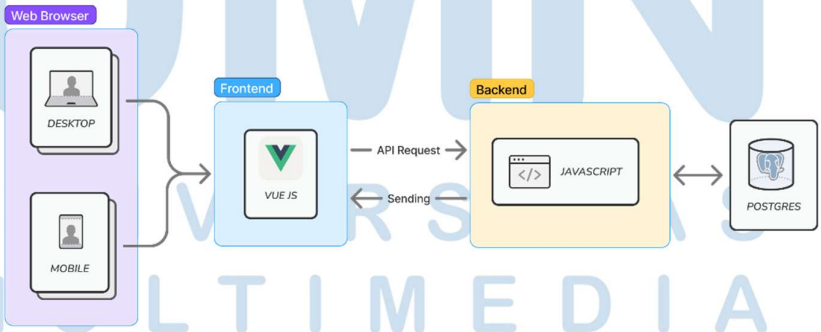
Tanggung jawab utama penulis sebagai CTO mencakup spektrum penuh dari siklus hidup pengembangan produk teknologi, yang dimulai dari fase konseptual hingga implementasi akhir. Secara rinci, tanggung jawab tersebut meliputi perancangan arsitektur sistem yang kokoh, di mana penulis bertugas merancang fondasi teknis platform secara keseluruhan untuk memastikan sistem bersifat *scalable* (dapat berkembang untuk menampung lebih banyak pengguna di masa depan) dan *maintainable* (mudah untuk dipelihara dan dikembangkan lebih lanjut). Selanjutnya adalah pemilihan teknologi secara strategis, yang melibatkan riset dan pengambilan keputusan krusial dalam menentukan tumpukan teknologi (*technology stack*) yang paling sesuai. Keputusan jatuh pada Node.js untuk pengembangan sisi server (*backend*) karena sifatnya yang *asynchronous* dan sangat efisien dalam menangani operasi I/O dan koneksi *real-time* yang vital untuk fitur konsultasi. Untuk sisi klien (*frontend*), Vue.js dipilih karena kerangka kerjanya yang reaktif dan progresif, memungkinkan pembuatan antarmuka pengguna yang dinamis dan interaktif. Sementara itu, PostgreSQL diadopsi sebagai sistem manajemen basis data karena reputasinya yang unggul dalam menjaga integritas data relasional dan kemampuannya menangani kueri yang kompleks.

Selain perencanaan strategis, tanggung jawab juga mencakup eksekusi teknis seperti pengembangan *backend* dan *frontend*, pengelolaan basis data dari perancangan skema hingga optimisasi, serta yang tidak kalah penting, implementasi lapisan keamanan berlapis untuk melindungi data sensitif pengguna dan integritas *platform*. Seluruh rangkaian pekerjaan teknis ini diarahkan pada satu tujuan utama: membangun dan meluncurkan sebuah platform digital yang tidak hanya fungsional, tetapi juga andal (*reliable*), aman (*secure*), dan efisien (*efficient*), sehingga mampu memberikan nilai tambah nyata bagi para penggunanya dan membangun fondasi kepercayaan yang kuat di pasar.

3.5.1 Proses Pelaksanaan

3.5.1.1. Perancangan Arsitektur Sistem Aplikasi

Proyek pertama ini merupakan pilar fundamental yang menopang keseluruhan struktur teknis dari platform HELLOPET!. Sebagai tahap paling krusial, proyek ini tidak hanya meletakkan dasar bagi pengembangan fitur, tetapi juga menentukan skalabilitas, keamanan, dan kemudahan pemeliharaan aplikasi di masa depan. Proses ini diawali dengan analisis kebutuhan secara mendalam, membedah apa saja fitur yang harus ada (kebutuhan fungsional, seperti pendaftaran pengguna, konsultasi, dan pencarian *grooming*) serta bagaimana platform seharusnya beroperasi (kebutuhan *non-fungsional*, seperti harus cepat, aman, dan dapat diandalkan).



Gambar 3. 6 Desain *Arsitektur Monolithic*-API *Based*

Langkah awal adalah merancang arsitektur sistem seperti pada gambar 3.5.1.1. Pilihan jatuh pada pendekatan *Monolithic*-API *Based*. Dalam arsitektur ini, seluruh logika bisnis, pemrosesan data, dan manajemen pengguna dibangun dalam satu basis kode (*codebase*) di sisi *backend* (Node.js). Namun, *backend* ini tidak menghasilkan tampilan visual secara langsung. Sebaliknya, ia berfungsi murni sebagai penyedia API (*Application Programming Interface*) yang mengekspos data dan fungsionalitas melalui serangkaian *endpoint* RESTful. Di sisi lain, *frontend* (Vue.js) dibangun sebagai aplikasi yang sepenuhnya terpisah (*decoupled*) dan bertugas sebagai konsumen API. Keputusan strategis untuk memisahkan *backend* dan *frontend* ini diambil dengan beberapa pertimbangan utama, yaitu:

1. **Fleksibilitas Pengembangan**

   Tim atau pengembang dapat bekerja secara paralel. Tim *frontend* bisa mengembangkan antarmuka menggunakan data tiruan (*mock data*) tanpa harus menunggu backend selesai, dan sebaliknya.

2. **Kemudahan Pemeliharaan**

   Ketika terjadi masalah, lebih mudah untuk mengisolasinya apakah bersumber dari logika di *backend* atau dari tampilan di *frontend*.

3. **Skalabilitas**

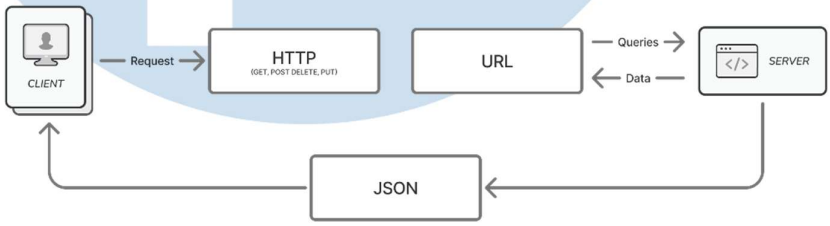
   Arsitektur ini membuka kemungkinan untuk membuat *applikasi Android/iOS* di masa yang dapat menggunakan API yang sama tanpa perlu mengubah logika di *backend*.

3.5.1.2. Pengembangan *Backend* Menggunakan Node.js

Pada proyek ini, dilakukan proses pengembangan *backend* sebagai komponen inti yang mendukung keseluruhan fungsi utama

dalam aplikasi HELLOPET!. *Backend* berperan sebagai pusat kendali dari setiap proses yang berjalan di dalam sistem, termasuk pengelolaan data, pelaksanaan logika bisnis, serta menjembatani komunikasi antara antarmuka pengguna (*frontend*) dan sistem basis data (*database*) di sisi server.

Arsitektur backend dibangun menggunakan kombinasi Node.js dan Express.js. Node.js dipilih karena kemampuannya dalam menangani banyak permintaan secara *asynchronous* dan *non-blocking*, menjadikannya sangat sesuai untuk kebutuhan aplikasi *real-time* seperti layanan konsultasi daring di HELLOPET!. Sementara itu, Express.js digunakan sebagai *framework* utama karena struktur yang fleksibel dan dukungan pustaka yang luas melalui NPM, sehingga proses pengembangan dapat dilakukan secara lebih cepat dan terorganisir.



Gambar 3. 7 Desain RESTful API

Backend HELLOPET! dikembangkan dengan menerapkan prinsip RESTful API (*Representational State Transfer*) sebagai fondasi utama dalam merancang sistem komunikasi antara *frontend* dan *backend*. Pendekatan ini dipilih karena RESTful API memberikan struktur yang konsisten, mudah dikembangkan, serta mampu menangani berbagai jenis permintaan secara efisien dan terorganisir. API dalam platform ini berperan sebagai jembatan utama yang menghubungkan sistem *frontend* dengan basis data serta berbagai layanan bisnis yang berjalan di sisi *server*. Melalui API inilah, seluruh interaksi penting antara pengguna (*Pet Lover*), dokter hewan, dan sistem dijalankan.

RESTful API HELLOPET! menangani berbagai kebutuhan inti dalam sistem, di antaranya termasuk: proses autentikasi pengguna (seperti registrasi, *login*, dan *logout*), manajemen data dokter hewan, pengaturan jadwal konsultasi, pengolahan transaksi pembayaran, serta pencatatan *history* sesi konsultasi antara pengguna dengan dokter. Semua ini dirancang dengan prinsip *modular* dan *reusable*, sehingga tiap bagian dapat dikembangkan secara independen dan mudah untuk dikontrol.

Salah satu pekerjaan awal dalam pengembangan *backend* adalah pembuatan dan pengelolaan rute API (*routes*). Setiap rute API dibuat dengan struktur yang sistematis dan mudah dibaca, dengan pendekatan segmentasi berdasarkan fungsi. Contohnya, *endpoint* /*auth* digunakan untuk menangani seluruh permintaan terkait autentikasi dan otorisasi pengguna (seperti *login*, *register*, dan *token refresh*), sedangkan /*roles* digunakan untuk mengelola peran pengguna seperti dokter dan *client*. Endpoint lainnya seperti /*messages*, /*consultations*, /*payments*, dan /*statistics* disusun untuk mengatur pengiriman pesan, penjadwalan konsultasi, transaksi, serta analisis performa layanan.

Setiap rute ini dihubungkan ke *controller* masing-masing yang bertugas menjalankan logika bisnis secara spesifik. Misalnya, ketika permintaan masuk ke *endpoint* /auth/*login*, permintaan tersebut akan diarahkan ke *controller* authController.login, yang kemudian akan melakukan validasi data pengguna, pengecekan *database*, dan menghasilkan token autentikasi jika valid. Struktur pemisahan ini membantu menjaga keteraturan kode, meningkatkan keterbacaan, serta memudahkan *debugging* dan pengembangan lanjutan.

Gambar 3. 8 Desain RESTful API

Pada Gambar 3.5.1.2.1 ditampilkan desain arsitektur RESTful API HELLOPET! yang menggambarkan hubungan antara *client*, *server*, dan basis data. Setiap *endpoint* yang ditampilkan dalam diagram memiliki tugas dan tanggung jawab yang spesifik, mulai dari mengambil data, menyimpan data baru, memperbarui data yang sudah ada, hingga menghapus data sesuai dengan kebutuhan pengguna. Diagram tersebut juga merepresentasikan bagaimana setiap endpoint memiliki *child* atau *sub-routes* yang akan menangani aksi-aksi tertentu secara granular. Misalnya, *endpoint* /messages akan memiliki child API seperti POST /*messages*/*send*, GET /*messages*/:*sessionId*, dan lain sebagainya, yang masing-masing dirancang untuk tugas spesifik dalam sistem perpesanan.

Dengan pendekatan RESTful ini, pengembangan backend HELLOPET! menjadi lebih terstruktur, fleksibel, dan *scalable* yang memungkinkan fitur-fitur baru ditambahkan di masa mendatang tanpa mengganggu fungsionalitas yang sudah ada. Berikut ini merupakan penjelasan lengkap mengenai *child* API yang telah dikembangkan pada sistem ini:

1) **API *Authentication*:** Dalam proses pengembangan *backend* HELLOPET!, salah satu komponen paling krusial yang dibangun adalah sistem API *Authentication*, yang berfungsi sebagai mekanisme utama untuk mengatur proses identifikasi

(*authentication*) dan pemberian izin akses (*authorization*) kepada setiap pengguna platform. Sistem ini dirancang untuk memastikan bahwa hanya pengguna yang telah terverifikasi yang dapat mengakses fitur-fitur inti, seperti konsultasi online dengan dokter hewan, melihat histori konsultasi, hingga melakukan pembayaran melalui sistem. Oleh karena itu, API *Authentication* menjadi lapisan keamanan pertama (*first line of defense*) dalam melindungi data pengguna dan mencegah potensi akses ilegal ke dalam sistem.

Proses autentikasi ini dikembangkan menggunakan pendekatan *modern* berbasis token, dengan memanfaatkan *JSON Web Token* (JWT) sebagai media utama untuk otorisasi pengguna setelah berhasil login. Dengan metode ini, sistem tidak perlu menyimpan sesi secara terus-menerus di *server*, karena token JWT berfungsi sebagai bukti autentikasi yang dapat dibawa pengguna dalam setiap permintaan (*request*) yang dikirimkan ke server, menjadikan sistem lebih ringan, efisien, dan aman.

API *Authentication* dalam HELLOPET! dibagi menjadi beberapa *endpoint* utama yang masing-masing menangani tahapan penting dalam proses autentikasi:

a. ***Login*****:** Setelah pengguna memiliki akun, mereka dapat menggunakan *endpoint* ini untuk masuk ke sistem. Proses login akan memverifikasi kredensial (*email* dan *password*) dan jika berhasil, akan mengembalikan JSON Web Token (JWT) yang digunakan sebagai tanda autentikasi saat mengakses fitur lainnya.

```
const { email, password } = req.body;

    if (!email || !password) {
      return res
        .status(400)
        .json({ message: "Email and password are required" });
    }

    let user = await User.findOne({ where: { email } });

    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }
```

Gambar 3. 9 Validasi Input dan Pencarian Data

Pada tahap awal, sistem menerima input berupa *email* dan *password* dari pengguna melalui *body* permintaan HTTP. Kemudian dilakukan validasi apakah kedua data tersebut tersedia. Jika salah satu kosong, maka *server* langsung merespons dengan status 400 *Bad Request*. Jika sudah lengkap, sistem akan mencari data pengguna di basis data berdasarkan alamat *email*. Jika pengguna tidak ditemukan, akan dikembalikan pesan kesalahan "*User not found*".

```
const match = await bcrypt.compare(password, user.password);
    if (!match) {
      return res.status(401).json({ message: "Invalid credentials" });
    }
```

Gambar 3. 10 Verifikasi *Password*

Setelah data pengguna ditemukan, sistem melakukan verifikasi apakah kata sandi yang dimasukkan sesuai dengan yang tersimpan di *database*. Proses ini menggunakan *bcrypt*, sebuah *library* untuk enkripsi. Jika

hasil perbandingan tidak cocok, maka akan dikembalikan status 401 *Unauthorized* karena kredensial tidak valid.

```
const token = jwt.sign(
      {
        id: user.id, username: user.username, email: user.email, roles: user.roles,
      },
      process.env.JWT_SECRET,{ expiresIn: "1h" }
    );
```

Gambar 3. 11 Pembuatan Token JWT

Jika *password* cocok, maka sistem akan membuat JSON Web Token (JWT) yang berisi informasi penting tentang pengguna (ID, *username*, *email*, dan *roles*). Token ini ditandatangani menggunakan JWT_*SECRET* yang tersimpan di variabel lingkungan dan memiliki masa aktif selama satu jam.

```
res.cookie("accessToken", token, {
      httpOnly: false,
      secure: process.env.NODE_ENV === "production",
      sameSite: process.env.NODE_ENV === "production" ? "none" : "lax",
      maxAge: 3600000, // 1 jam
      domain:
        process.env.NODE_ENV === "production" ? ".hellopet.site" : undefined,
    });
```

Gambar 3. 12 Menyimpan Token dalam Cookie

Token yang telah dibuat kemudian disimpan dalam *cookie* untuk dikirim kembali oleh *browser* pada setiap permintaan selanjutnya. *Cookie* diatur agar aman saat berada di lingkungan produksi, dengan masa aktif yang sama seperti token (1 jam). Fitur ini mendukung keamanan dan kenyamanan pengguna saat berpindah halaman tanpa harus login ulang.

```
return res.status(200).json({
      status: true,
      message: "Login successful",
      data: {
        user: {
          id: user.id,
          username: user.username,
          roles: user.roles,
          phone_number: user.phone_number,
        },
        token: token,
      },
    });
```

Gambar 3. 13 Respons Keberhasilan Login

Jika semua langkah berhasil, sistem akan memberikan *respons* sukses (HTTP 200 *OK*) yang berisi data pengguna serta token yang akan digunakan untuk autentikasi pada permintaan berikutnya.

b. ***Register*:** *Endpoint* ini digunakan oleh pengguna baru untuk membuat akun di platform HELLOPET!. Data yang dikirim meliputi nama lengkap, *email*, *password*, serta informasi tambahan jika dibutuhkan. *Password* disimpan dengan sistem *hashing* menggunakan algoritma *bcrypt* untuk memastikan keamanan.

```
const { username, email, password, phone_number } = req.body;

    if (!username || !email || !password || !phone_number) {
      return res.status(400).json({ message: "All fields are required" });
    }
```

Gambar 3. 14 Validasi Data Masukan Pengguna

Bagian ini bertanggung jawab untuk memastikan bahwa seluruh data penting yang dibutuhkan untuk proses registrasi telah disediakan oleh pengguna. Jika salah satu data tidak ada, maka *server* akan langsung mengembalikan *respons* 400 *Bad Request* dengan pesan bahwa semua kolom harus diisi. Hal ini merupakan bentuk input *validation* dasar.

```
const existing = await User.findOne({ where: { email } });
    if (existing) {
      return res.status(400).json({ message: "Email already used" });
    }
```

Gambar 3. 15 Pemeriksaan Apakah *Email* Sudah Terdaftar

Sebelum membuat akun baru, sistem akan memeriksa apakah *email* yang didaftarkan telah digunakan oleh pengguna lain di *database*. Hal ini dilakukan dengan metode *findOne*() dari ORM *Sequelize* yang digunakan untuk mengakses *model User*. Jika *email* ditemukan, proses registrasi akan dihentikan untuk mencegah duplikasi akun.

```
const saltRounds = 10;
const hashed = await bcrypt.hash(password, saltRounds);
```

Gambar 3. 16 Enkripsi Kata Sandi

Untuk menjaga keamanan data pengguna, terutama kata sandi, sistem tidak menyimpannya secara langsung dalam bentuk teks asli *(plain text*). Sebagai gantinya, digunakan algoritma *hashing bcrypt* dengan jumlah *salt rounds* sebanyak 10. Ini menghasilkan *string* acak yang aman dan tidak bisa dikembalikan ke bentuk aslinya, sebagai bagian dari praktik keamanan digital *modern*.

Gambar 3. 17 Pembuatan Akun Baru di *Database*

Setelah semua validasi dan proses *hashing* selesai, sistem akan membuat data pengguna baru ke dalam tabel *User*. Semua data pengguna disimpan termasuk hasil enkripsi *password* dan *role default* sebagai "*user*". Proses ini menggunakan fungsi create() dari *Sequelize*.

Gambar 3. 18 Respon Keberhasilan Registrasi

Jika semua proses berhasil dijalankan tanpa kendala, *server* akan mengirimkan respon dengan status 201 *Created* dan pesan bahwa registrasi telah berhasil

dilakukan. Ini menjadi penanda bahwa akun pengguna telah resmi dibuat dan disimpan dalam sistem.

c. **Logout:** *Endpoint* ini digunakan untuk menghapus data JWT pada *cookie* pada sisi *server* dan *client*, sehingga user tidak punya akses untuk menggunakan jasa konsultasi HELLOPET!.

```
export const logout = async (req, res) => {
  try {
    res.clearCookie("accessToken", {
      httpOnly: false,
      secure: process.env.NODE_ENV === "production",
      sameSite: process.env.NODE_ENV === "production" ? "none" : "lax",
      domain:
        process.env.NODE_ENV === "production" ? ".hellopet.site" : undefined,
    });

    return res.status(200).json({
      status: true,
      message: "Logout successful",
    });
  } catch (error) {
    console.error("Error during logout:", error);
    return res.status(500).json({
      status: false,
      message: "Internal server error",
    });
  }
};
```

Gambar 3. 19 Fungsi *Logout*

Untuk mengelola proses verifikasi token ini, digunakan middleware khusus. *Middleware* ini secara otomatis dijalankan sebelum endpoint dijalankan. *Middleware* akan membaca token dari *header Authorization*, memverifikasi keasliannya menggunakan kunci rahasia (*secret key*), dan hanya akan melanjutkan permintaan jika token valid. Jika tidak valid atau tidak ditemukan, *middleware* akan menolak akses dengan respons yang sesuai, misalnya status 401 *Unauthorized*.

Selain autentikasi, *middleware* juga dimanfaatkan untuk validasi data *input*, pengecekan peran pengguna (misalnya apakah pengguna adalah dokter atau pelanggan), dan untuk

penanganan *error* secara terpusat. Dengan pendekatan ini, setiap permintaan yang masuk ke API dapat dikontrol dan dijaga keamanannya tanpa harus mengulang kode yang sama di banyak tempat.

Semua rute terkait autentikasi ini diatur dalam *endpoint* khusus /*auth*, yang dipisahkan ke dalam *controller* sendiri agar lebih *modular* dan mudah dalam pemeliharaan kode. *Middleware-middleware* tersebut diletakkan di *folder* tersendiri dan dipanggil sesuai kebutuhan pada tiap *rute*, memberikan fleksibilitas sekaligus menjaga keamanan.

Gambar 3. 20 *Endpoint* API *Authentication*

Dengan adanya sistem API *Authentication* yang terintegrasi dengan JWT serta dukungan *middleware* yang kuat, platform HELLOPET! mampu memberikan pengalaman yang aman dan terpercaya kepada seluruh penggunanya. Tidak hanya itu, sistem ini juga mendukung skalabilitas dan efisiensi, memungkinkan ratusan hingga ribuan pengguna untuk mengakses sistem secara bersamaan tanpa mengorbankan kinerja maupun keamanan data pribadi mereka.

2) **API Roles:** Pada pengembangan *backend* HELLOPET!, API *Roles* berperan penting dalam pengelolaan dan pengambilan data berdasarkan peran pengguna dalam sistem, khususnya peran sebagai dokter. *Endpoint* ini memungkinkan sistem untuk

membedakan dan menampilkan data pengguna yang memiliki role sebagai dokter. Data ini nantinya digunakan dalam berbagai fitur *frontend*, salah satunya adalah fitur *list* dokter, di mana pengguna (*pet owners*) dapat melihat informasi lengkap mengenai dokter-dokter yang tersedia untuk melakukan konsultasi.

Informasi yang ditampilkan mencakup nama dokter, spesialisasi, rating, pengalaman, serta jadwal ketersediaan konsultasi. Dengan adanya API ini, proses pemfilteran dan pemilahan data menjadi lebih efisien dan akurat, karena sistem hanya akan menampilkan pengguna yang benar-benar memiliki *role* sebagai dokter.

Gambar 3. 21 *Endpoint* API *Roles*

Fungsi getDokter merupakan sebuah fungsi yang digunakan untuk mengambil daftar data dokter dari *database* yang telah terhubung dengan sistem *backend*. Fungsi ini akan melakukan pemanggilan data pada tabel *user* atau tabel khusus dokter, kemudian memfilter berdasarkan *role* tertentu agar hanya data yang berperan sebagai dokter yang dikembalikan. Data yang berhasil diambil akan digunakan untuk menampilkan informasi dokter pada fitur *List* Dokter di sisi *frontend*, seperti nama, harga, pengalaman, dan rating dokter. Fungsi ini menjadi bagian penting dalam proses integrasi antara *database* dan tampilan antarmuka pengguna.

```
export const getDokter = async (req, res) => {
  try {
    const users = await User.findAll({
      where: {
        roles: "dokter",
      },
      attributes: [
        "id", "username", "email", "picture",
        "price", "workExperience", "rate",
      ],
    });

    if (!users || users.length === 0) {
      return res.status(404).json({ message: "No users found" });
    }

    res.status(200).json(users);
  } catch (error) {
    console.error("Error fetching users:", error.message);
    res.status(500).json({ error: "Internal server error" });
  }
};
```

Gambar 3. 22 Fungsi Mendapatkan Data Dokter

3) **API Messages Client:** API *Messages Client* merupakan bagian penting dari infrastruktur komunikasi dua arah yang disediakan oleh *platform* HELLOPET! antara pengguna (*client*/*pet owner*) dan dokter. API ini dikembangkan untuk menangani berbagai permintaan dari sisi pengguna, khususnya dalam konteks berkonsultasi secara daring. Fungsi utamanya meliputi pengiriman pesan, pengambilan histori pesan berdasarkan sesi konsultasi tertentu, serta menampilkan kembali percakapan yang pernah dilakukan antara pengguna dengan dokter.

Gambar 3. 23 *Endpoint* API *Message Client*

Sistem ini tidak hanya mendukung komunikasi *real-time* menggunakan teknologi *WebSocket*, tetapi juga mendukung mode *non-real-time*, di mana pengguna masih dapat membaca kembali pesan yang telah dikirim dan diterima meskipun sesi konsultasi telah berakhir. Semua data pesan disimpan dalam *database* secara terstruktur, lengkap dengan *metadata* seperti waktu pengiriman dan identitas pengirim. Dengan demikian, pengalaman berkonsultasi menjadi lebih nyaman, aman, dan dapat dilacak kembali jika dibutuhkan.

Berdasarkan *endpoint* tersebut digunakan untuk mengambil seluruh data pesan yang dikirim dan diterima dalam satu sesi konsultasi antara pengguna dan dokter. Proses pengambilan data ini didasarkan pada parameter yang spesifik, seperti ID sesi konsultasi atau kombinasi ID pengguna dan ID dokter, sehingga sistem dapat secara tepat menelusuri dan menampilkan hanya pesan-pesan yang relevan dengan sesi tersebut. Fungsi ini sangat penting dalam mendukung kelangsungan komunikasi yang terstruktur karena memungkinkan pengguna untuk meninjau kembali seluruh isi percakapan sebelumnya kapan pun dibutuhkan, baik untuk keperluan pengingat, dokumentasi, ataupun rujukan lanjutan pada sesi konsultasi berikutnya.

Selain itu, data yang ditampilkan oleh *endpoint* ini disusun berdasarkan dari *conversation* yang baru, sehingga *chat*

terdahulu tidak akan ditampilan. Hal ini memberikan ruang keamanan dan *privacy* pada user agar menampilkan pesan yang *relevant* atau sesuai berdasarkan dari hari konsultasi.

```
export const getMessages = async (req, res) => {
  try {
    const { receiverId, senderId } = req.query;

    if (!senderId || !receiverId) {
      return res
        .status(400)
        .json({ error: "senderId and receiverId are required" });
    }
    const messages = await Message.findAll({
      where: {
        conversationId: conversation.id,
      },
      include: [
        {
          model: User,
          as: "sender",
          attributes: ["id", "username", "picture"],
        },
        {
          model: User,
          as: "receiver",
          attributes: ["id", "username", "picture"],
        },
      ],
      order: [["createdAt", "ASC"]],
    });

    res.status(200).json(messages);
  } catch (error) {
    console.error("Error in getMessages controller: ", error);
    res.status(500).json({ error: "Internal server error" });
  }
};
```

Gambar 3. 24 *Endpoint* API *Message Client*

4) API Messages

API *Messages* merupakan komponen inti dari sistem komunikasi HELLOPET! yang memungkinkan terjadinya interaksi langsung antara pengguna dan dokter selama sesi konsultasi berlangsung. API ini bertanggung jawab dalam mengatur seluruh alur pertukaran pesan mulai dari pengiriman, penyimpanan, mengakhiri pesan. Peran API ini sangat krusial

karena menjembatani komunikasi dua arah secara digital dalam waktu nyata.

Setiap pesan yang dikirimkan melalui platform HELLOPET! akan terlebih dahulu diproses oleh *middleware* autentikasi, yang bertugas memverifikasi apakah pengirim dan penerima adalah pihak yang sah dan telah memiliki izin untuk mengakses sesi konsultasi tersebut. Ini merupakan langkah penting untuk menjamin privasi, keamanan, serta kerahasiaan informasi antar pengguna dan dokter.

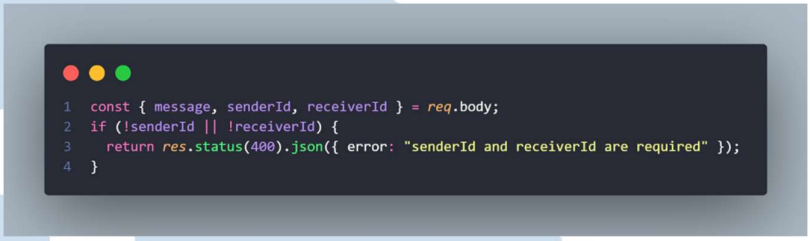
Gambar 3. 25 *Endpoint* API *Message*

Pengembangan API *Messages* juga diintegrasikan secara langsung dengan *WebSocket*, khususnya menggunakan teknologi Socket.io, untuk mengakomodasi kebutuhan komunikasi *real-time*. Dengan demikian, baik dokter maupun pengguna dapat berkonsultasi seolah-olah sedang melakukan percakapan langsung tanpa mengalami jeda waktu atau keterlambatan pesan (latency), yang sangat penting dalam penyampaian informasi medis secara cepat dan akurat. API Messages terdiri dari dua endpoint utama, yaitu:

a. Send Message

Fungsi sendMessage merupakan bagian dari *backend* HELLOPET! yang digunakan untuk mendukung fitur komunikasi dua arah secara *real-time* antara pengguna (*pet owner*) dengan dokter hewan. Fungsi ini mengelola pengiriman pesan berbasis API dan juga memastikan

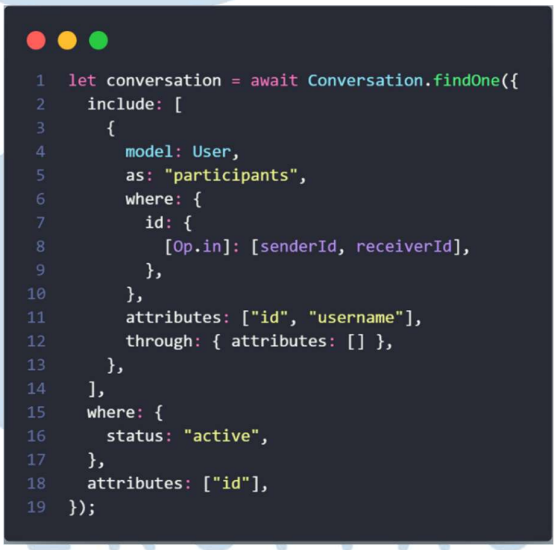
bahwa pesan langsung diteruskan secara instan menggunakan WebSocket (Socket.IO).



```
const { message, senderId, receiverId } = req.body;
if (!senderId || !receiverId) {
  return res.status(400).json({ error: "senderId and receiverId are required" });
}
```

Gambar 3. 26 Validasi Data

Langkah pertama dalam fungsi ini adalah memvalidasi apakah pengirim (senderId) dan penerima (receiverId) telah ditentukan dalam *body request*. Jika salah satu tidak ada, maka *server* akan mengembalikan status *error* 400 dengan pesan kesalahan yang sesuai. Ini penting untuk mencegah terjadinya kesalahan saat memproses percakapan.

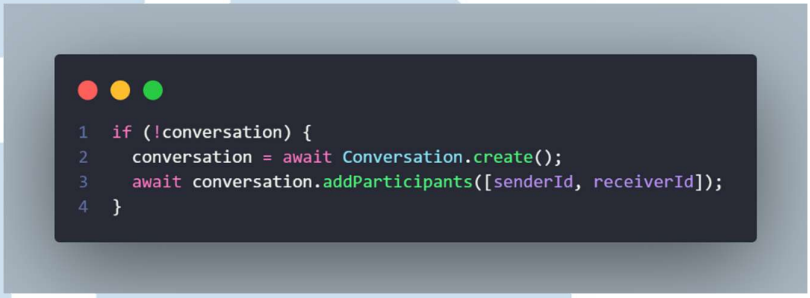


```
let conversation = await Conversation.findOne({
  include: [
    {
      model: User,
      as: "participants",
      where: {
        id: {
          [Op.in]: [senderId, receiverId],
        },
      },
      attributes: ["id", "username"],
      through: { attributes: [] },
    },
  ],
  where: {
    status: "active",
  },
  attributes: ["id"],
});
```

Gambar 3. 27 Validasi Data

Kode ini memeriksa apakah sudah ada percakapan aktif antara kedua pengguna (pengirim dan penerima). Proses ini menggunakan relasi *many-to-many* antara

model *Conversation* dan *User* melalui tabel pivot. Jika tidak ditemukan, maka sistem akan membuat sesi percakapan baru.



```
if (!conversation) {
  conversation = await Conversation.create();
  await conversation.addParticipants([senderId, receiverId]);
}
```

Gambar 3. 28 Membuat Percakapan Jika Belum Ada

Jika belum ada percakapan antara dua pihak tersebut, maka akan dibuat sesi *conversation* baru. Setelah itu, kedua ID pengguna ditambahkan ke sesi tersebut sebagai peserta percakapan. Ini menjaga struktur agar setiap pesan memiliki konteks historis dalam sebuah sesi.

```
const newMessage = await Message.create({ senderId, receiverId, message, conversationId: conversation.id,
});
```

Gambar 3. 29 Menyimpan Pesan Baru ke *Database*

Pesan yang dikirim oleh pengguna kemudian disimpan ke *database*. Informasi penting yang direkam meliputi:

1. ID pengirim (senderId)
2. ID penerima (receiverId)
3. Isi pesan (message) ID percakapan (conversationId) Dengan penyimpanan ini, setiap percakapan dapat ditelusuri kembali, memungkinkan *log* histori *chat*.

Gambar 3. 30 23 Pengiriman Pesan *Real-Time* via Socket.IO

Setelah pesan disimpan, fungsi ini mengecek apakah penerima sedang online dan terhubung ke *socket*. Jika ya, maka pesan dikirim secara *instan* ke penerima menggunakan *WebSocket* (dalam hal ini melalui Socket.IO). Hal ini memungkinkan pengalaman percakapan *real-time* seperti aplikasi *chat modern* pada umumnya.

Gambar 3. 31 Mengembalikan Respons

Jika seluruh proses berhasil, *server* akan mengirimkan respons sukses dengan data pesan baru yang telah dibuat, menggunakan status HTTP 201 (*Created*).

b. End Message

Endpoint ini digunakan untuk menandai bahwa sesi konsultasi telah berakhir. Dokter ataupun pengguna dapat menginisiasi penutupan sesi, dan sistem akan mencatat status konsultasi menjadi "selesai." Setelah sesi ditutup, pengguna tidak dapat lagi mengirimkan pesan baru kecuali membuka sesi baru. *Endpoint* ini juga membantu sistem dalam mengelola arsip percakapan, menghentikan aliran data aktif, serta menjaga penggunaan sumber daya secara optimal.

```
export const endConversation = async (req, res) => {
  const { id } = req.params;

  try {
    const conversation = await Conversation.findByPk(id);

    if (!conversation) {
      return res.status(404).json({ message: "Percakapan tidak ditemukan." });
    }

    if (conversation.status === "ended") {
      return res.status(400).json({ message: "Percakapan sudah berakhir." });
    }

    conversation.status = "ended";
    await conversation.save();

    return res
      .status(200)
      .json({ message: "Percakapan berhasil diakhiri.", conversation });
  } catch (error) {
    console.error("Gagal mengakhiri percakapan:", error);
    return res.status(500).json({ message: "Terjadi kesalahan server." });
  }
};
```

Gambar 3. 32 Fungsi *End Conversation*

5) API Payment

API *Payment* merupakan bagian penting dari sistem HELLOPET! yang menangani seluruh proses transaksi *digital* pengguna, khususnya saat pengguna ingin memulai sesi konsultasi dengan dokter. Fungsi utama dari API ini adalah memastikan bahwa setiap proses pembayaran berjalan dengan aman, efisien, dan tervalidasi sebelum akses ke fitur konsultasi diberikan kepada pengguna. Dengan kata lain, API ini berperan sebagai gerbang utama antara proses pemesanan layanan dan validasi hak akses pengguna berdasarkan status transaksi.

```
export const payConsulting = async (req, res) => {
  try {
    const { id } = req.body;
    const patientId = req.user.id;

    const doctor = await User.findByPk(id);
    const amount = doctor.price;
    const transaction = await db.Transaction.create({
      amount,
      status: "pending",
      doctorId: doctor.id,
      patientId: patientId,
    });
    const parameter = {
      transaction_details: {
        order_id: transaction.id,
        gross_amount: Number(transaction.amount),
      },
      item_details: [
        {
          id: doctor.id,
          price: Number(transaction.amount),
          quantity: 1,
          name: `Konsultasi dengan Dr. ${doctor.username}`,
        },
      ],
      customer_details: {
        first_name: req.user.username,
        email: req.user.email,
        phone: req.user.phone_number,
      },
    };
    const midtransResponse = await snap.createTransactionToken(parameter);
    res.status(201).json({
      token: midtransResponse,
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Gagal memproses pembayaran" });
  }
};
```

Gambar 3. 33 Fungsi Pembayaran

Dalam pengembangannya, sistem API *Payment* diintegrasikan secara langsung dengan *Midtrans*, sebuah penyedia layanan pembayaran *digital* (*payment gateway*) yang telah terpercaya di Indonesia. *Midtrans* memungkinkan pengguna HELLOPET! untuk melakukan pembayaran dengan berbagai metode, seperti *transfer bank*, *e-wallet* (OVO, GoPay, DANA), kartu kredit, hingga gerai ritel. Integrasi ini dilakukan melalui skema *server-to-server* menggunakan Snap API dan

Core API *Midtrans*, yang memungkinkan *backend* aplikasi untuk membuat dan mengelola transaksi secara otomatis.

Gambar 3. 34 Endpoint API Payment

6) API *Statistic*

API *Statistic* merupakan komponen *backend* yang dikembangkan untuk menyajikan data analitik dan performa layanan konsultasi dokter secara visual dan terstruktur. API ini dirancang khusus untuk membantu dokter dalam memahami perkembangan kinerja mereka dari waktu ke waktu, melalui penyajian data berupa jumlah sesi konsultasi serta akumulasi profit (pendapatan) yang diperoleh dalam kurun waktu satu tahun, dengan perincian setiap bulan.

Gambar 3. 35 *Endpoint* API Statistik

API ini berperan penting dalam menyajikan informasi berbasis data yang dapat dijadikan dasar evaluasi dan pengambilan keputusan oleh dokter. Dengan memanfaatkan metode agregasi data dari basis data, sistem mampu menghitung secara otomatis:

a) Jumlah konsultasi yang dilakukan setiap bulan, disusun secara kronologis dari Januari hingga Desember, memungkinkan dokter melihat pola lonjakan atau penurunan aktivitas konsultasi dalam satu tahun.

```
export const getMonthlyConversationsByDoctor = async (req, res) => {
  const doctorId = req.user.id;
  try {
    const thisYear = new Date().getFullYear();
    const startOfYear = new Date(thisYear, 0, 1);
    const endOfYear = new Date(thisYear, 11, 31, 23, 59, 59);

    const conversations = await db.Conversation.findAll({
      where: {
        createdAt: {
          [Op.between]: [startOfYear, endOfYear],
        },
      },
      include: [
        {
          association: "participants",
          where: {
            id: doctorId,
            roles: "dokter",
          },
          attributes: [],
          through: { attributes: [] },
        },
      ],
      attributes: ["createdAt"],
    });

    const monthlyCounts = Array(12).fill(0);

    conversations.forEach((conv) => {
      const monthIndex = new Date(conv.createdAt).getMonth();
      monthlyCounts[monthIndex]++;
    });

    res.json({
      labels: moment.monthsShort(),
      data: monthlyCounts,
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Internal server error" });
  }
};
```

Gambar 3. 36 Fungsi Jumlah Konsultasi Yang Terjadi Setiap Bulan

b) Profit atau pendapatan bulanan, dihitung berdasarkan total pembayaran yang diterima dari sesi konsultasi yang berhasil dan valid. Informasi ini disusun dalam bentuk data numerik yang kemudian divisualisasikan dalam bentuk *line chart* pada *dashboard* dokter.

```
export const getMonthlyProfitByDoctor = async (req, res) => {
  try {
    const doctorId = req.user.id;
    let monthExtract;
    monthExtract = [
      Sequelize.literal('EXTRACT(MONTH FROM "createdAt")'),
      "month",
    ];
    const monthlyProfits = await Transaction.findAll({
      attributes: [
        monthExtract,
        [Sequelize.fn("SUM", Sequelize.col("amount")), "total_profit"],
      ],
      where: {
        doctorId: doctorId,
        status: "pending",
        createdAt: {
          [Op.between]: [
            new Date(`${currentYear}-01-01`),
            new Date(`${currentYear}-12-31`),
          ],
        },
      },
      group: ["month"],
      order: [["month", "ASC"]],
    });
    const formattedData = Array(12)
      .fill(0)
      .map((_, index) => {
        const monthNumber = index + 1;
        const monthData = monthlyProfits.find((item) => {
          const itemMonth = Number(item.dataValues.month);
          return itemMonth === monthNumber;
        });

        return {
          month: monthNames[index],
          profit: monthData ? parseFloat(monthData.dataValues.total_profit) : 0,
        };
      });
    return res.status(200).json({
      success: true,
      data: formattedData,
      year: currentYear,
    });
  } catch (error) {
    console.error("Error fetching monthly profits:", error);
    return res.status(500).json({
      success: false,
      message: "Failed to fetch monthly profits",
      error: error.message,
    });
  }
};
```

Gambar 3. 37 Fungsi Pendapatan Bulanan

Endpoint ini bekerja dengan mengakses data dari tabel transaksi dan tabel konsultasi yang telah terverifikasi pembayarannya. Data yang terkumpul kemudian diolah dalam backend menggunakan fungsi agregasi waktu (*group by* bulan dan tahun), dan hasilnya dikirimkan ke *frontend* untuk divisualisasikan dalam bentuk grafik interaktif yang mudah dipahami.

3.5.1.3. Membuat Websocket di *Backend*

Fitur konsultasi daring secara *real-time* merupakan inti utama dari layanan inovatif yang ditawarkan oleh *platform* HELLOPET!, yang dirancang khusus untuk memfasilitasi komunikasi langsung antara pengguna (*Pet Lover*) dengan dokter hewan profesional secara efisien, cepat, dan intuitif. Fitur ini menjadi nilai jual utama dari HELLOPET! karena memungkinkan pengguna mendapatkan layanan konsultasi kesehatan hewan tanpa harus datang langsung ke klinik, serta memperkuat pengalaman digital dalam dunia *pet care* yang semakin berkembang. Dalam implementasinya, sistem ini tidak hanya ditujukan untuk menyediakan saluran komunikasi, tetapi juga untuk memastikan bahwa interaksi berlangsung secara natural, cepat, dan tidak terganggu oleh delay jaringan atau sistem polling yang lambat.

Berbeda dari pendekatan tradisional menggunakan protokol HTTP yang bersifat *stateless* dan hanya mendukung komunikasi satu arah berbasis pola *request-response*, fitur konsultasi ini membutuhkan koneksi yang lebih canggih yakni koneksi yang dapat tetap terbuka secara terus-menerus dan mendukung komunikasi dua arah (bidirectional) antara klien dan *server* secara simultan. Hal ini sangat penting agar setiap pesan yang dikirim dari salah satu pihak, baik dari pengguna maupun dokter, dapat langsung diterima dan ditampilkan kepada pihak lainnya secara *real-time*, tanpa harus menunggu permintaan pengambilan data secara manual. Dengan kata lain,

pengalaman yang ingin dibangun di dalam HELLOPET! adalah komunikasi yang terasa live sebagaimana layaknya aplikasi perpesanan *modern* seperti WhatsApp atau Telegram, namun dengan konteks konsultasi medis hewan.

Untuk memenuhi kebutuhan ini, teknologi WebSocket menjadi pilihan utama karena menawarkan solusi komunikasi *stateful* yang memungkinkan satu koneksi tetap terbuka untuk jangka waktu panjang dan dapat digunakan secara bersamaan oleh kedua pihak (klien dan *server*) untuk saling bertukar data. Di sisi *backend*, WebSocket diimplementasikan melalui pustaka socket.io, yang sangat kompatibel dengan ekosistem Node.js dan menawarkan banyak fitur unggulan seperti *fallback* ke *long polling* untuk jaringan yang tidak mendukung WebSocket secara *native*, serta antarmuka API berbasis *event-driven* yang sangat intuitif. Dengan socket.io, setiap koneksi pengguna dapat dipantau, dikelola, dan direspons dengan efisien, menciptakan arsitektur komunikasi yang sangat responsif dan terukur.

Dalam penerapannya, server *backend* HELLOPET! dikonfigurasi untuk secara aktif "mendengarkan" koneksi WebSocket yang masuk, serta mampu merespons berbagai peristiwa komunikasi (*events*) yang terjadi dalam sesi konsultasi, seperti pengiriman pesan, status pengguna sedang mengetik, tanda pesan telah dibaca, hingga pengelolaan pengguna yang keluar dari sesi. Hal ini menciptakan sebuah sistem komunikasi dua arah yang bersifat dinamis dan *ss*, di mana server tidak hanya bereaksi terhadap permintaan, tetapi juga secara aktif mengirimkan informasi baru ke klien ketika ada perubahan status dalam sistem. Berikut adalah implementasi penggunaan websocket pada sisi backend, yaitu:

1) **Event “typing”**

   Dalam sebuah sistem komunikasi daring, khususnya pada platform digital yang berfokus pada layanan konsultasi

seperti HELLOPET!, keberadaan fitur indikator pengetikan (typing indicator) memegang peran yang sangat penting dalam menciptakan pengalaman pengguna yang lebih interaktif, humanis, dan menyerupai percakapan secara langsung (face-to-face). Sistem ini dirancang sedemikian rupa agar terasa seintuitif dan sealami mungkin, mirip dengan dinamika interaksi verbal sehari-hari di dunia nyata. Indikator pengetikan menjadi elemen visual yang tidak hanya menambah dimensi emosional dan psikologis dalam berkomunikasi, tetapi juga mampu meningkatkan keterlibatan (engagement) dan kenyamanan pengguna selama sesi konsultasi berlangsung.

Fitur event typing di HELLOPET! menjadi salah satu bentuk dukungan nyata terhadap kebutuhan komunikasi real-time yang tidak hanya instan, tetapi juga responsif secara sosial. Dengan adanya indikator ini, pengguna—baik itu pemilik hewan (pet lover) maupun dokter hewan—dapat memperoleh informasi kontekstual secara langsung bahwa lawan bicara mereka sedang dalam proses menyusun pesan. Indikasi ini secara tidak langsung mengurangi kecemasan atau ketidakpastian yang kerap muncul saat terjadi jeda dalam komunikasi, sehingga pengguna tidak terburu-buru mengirimkan pesan lanjutan yang mungkin tidak perlu. Hal ini sangat krusial dalam konteks konsultasi medis daring, di mana ketepatan dan kejelasan komunikasi dapat berdampak langsung terhadap kualitas pemahaman terhadap gejala atau kondisi kesehatan hewan yang dibahas.

Secara teknis, implementasi fitur ini memanfaatkan pendekatan event-driven menggunakan pustaka Socket.IO pada sisi backend HELLOPET!, yang terintegrasi dengan Node.js. Ketika seorang pengguna—baik dokter maupun

pemilik hewan—mulai mengetik pada perangkat mereka, maka frontend akan segera mengirimkan event bernama typing menuju backend. Data yang dikirim dalam event ini memuat tiga komponen penting: senderId (ID pengguna yang sedang mengetik), receiverId (ID lawan bicara yang sedang dalam sesi yang sama), serta isTyping, sebuah flag Boolean yang menunjukkan apakah pengguna sedang aktif mengetik (true) atau telah berhenti (false).

Setelah event ini diterima di sisi backend, server akan memprosesnya dan menyimpan status pengetikan pengguna dalam objek lokal bernama typingUsers. Objek ini bersifat sementara (non-persisten) dan berfungsi sebagai penyimpan data status real-time yang mencatat apakah seorang pengguna saat ini sedang mengetik, beserta waktu terakhir aktivitas tersebut terjadi melalui properti timestamp. Informasi ini sangat berguna untuk keperluan pelacakan status pengetikan dengan logika tambahan, seperti penghapusan otomatis jika pengguna tidak lagi aktif setelah jangka waktu tertentu.

Langkah selanjutnya, sistem akan memeriksa apakah penerima pesan dalam keadaan online melalui pemetaan socket ID yang disimpan dalam objek userSocketMap. Jika penerima terdeteksi aktif, maka server akan segera mengirimkan event userTyping ke socket tujuan, sehingga lawan bicara dapat menerima dan menampilkan indikator pengetikan secara visual di antarmuka pengguna (frontend). Contohnya, dokter hewan akan melihat teks bertuliskan "Pengguna sedang mengetik..." di layar mereka, dan sebaliknya, pengguna akan diberi tahu saat dokter sedang menyusun pesan balasan.

Dengan implementasi ini, HELLOPET! berhasil menghadirkan fitur yang tidak hanya meningkatkan aspek teknis dari sisi komunikasi, tetapi juga secara signifikan menyempurnakan pengalaman pengguna (user experience). Fitur ini memperkuat kesan bahwa pengguna sedang berbicara langsung dengan seorang profesional yang responsif dan hadir secara real-time, meskipun interaksi tersebut sepenuhnya dilakukan secara daring.

```
socket.on("typing", (data) => {
    const { senderId, receiverId, isTyping } = data;
    if (isTyping) {
      typingUsers[senderId] = {
        isTyping: true,
        timestamp: new Date(),
      };
    } else {
      if (typingUsers[senderId]) {
        typingUsers[senderId].isTyping = false;
      }
    }
    const receiverSocketId = userSocketMap[receiverId];
    if (receiverSocketId) {
      io.to(receiverSocketId).emit("userTyping", {
        userId: senderId,
        isTyping,
      });
    }
  });
```

Gambar 3. 38 *Typing Socket*

2) Event "sendMessage"

Dalam pengembangan fitur komunikasi real-time pada platform HELLOPET!, salah satu komponen utama yang diimplementasikan adalah mekanisme pengiriman pesan secara langsung antara pengguna dan dokter hewan menggunakan Socket.IO. Salah satu event penting yang bertanggung jawab dalam proses ini adalah event

sendMessage. Event ini berfungsi sebagai jembatan utama dalam aliran komunikasi teks, memastikan bahwa setiap pesan yang dikirim oleh pengguna dapat segera diteruskan dan diterima oleh pihak yang dituju tanpa jeda waktu yang signifikan.

```
socket.on("sendMessage", (message) => {
    console.log("Received message to send:", message);
    const receiverSocketId = userSocketMap[message.receiverId];

    if (receiverSocketId) {
      console.log(
        `Sending to receiver ${message.receiverId} with socket ${receiverSocketId}`
      );
      io.to(receiverSocketId).emit("newMessage", message);
    } else {
      console.log(`Receiver ${message.receiverId} is not online`);
    }
  });
```

Gambar 3. 39 *Send Message Socket*

Secara teknis, proses ini dimulai ketika client (baik itu pengguna atau dokter hewan) mengirimkan pesan baru melalui socket yang telah terhubung sebelumnya. Pesan tersebut diproses melalui event listener socket.on("sendMessage", callback), di mana server backend menerima payload atau objek message yang berisi informasi penting seperti receiverId (ID pengguna yang akan menerima pesan), senderId, isi pesan (text), timestamp, dan atribut tambahan lainnya yang dibutuhkan untuk proses pencatatan atau tampilan frontend.

Setelah server menerima objek message, langkah pertama yang dilakukan adalah mencatat aktivitas tersebut melalui log console.log("Received message to send:", message) untuk memudahkan debugging dan pemantauan aktivitas pesan di server.

Selanjutnya, server akan mencari tahu apakah penerima pesan sedang dalam keadaan online atau tidak. Hal ini dilakukan dengan memeriksa keberadaan receiverId dalam struktur data sementara bernama userSocketMap, yaitu sebuah objek yang berfungsi sebagai pemetaan antara user ID dan socket ID mereka yang aktif. Jika pengguna sedang online, maka receiverSocketId akan bernilai valid dan server dapat langsung mengirimkan pesan.

Jika penerima terdeteksi aktif secara real-time (artinya receiverSocketId tersedia), maka server akan menggunakan fungsi io.to(receiverSocketId).emit("newMessage", message) untuk menyampaikan pesan tersebut langsung ke pengguna yang bersangkutan. Event newMessage yang dipancarkan akan segera diterima di sisi klien penerima, yang kemudian akan memicu pembaruan antarmuka pengguna (*frontend*) seperti penambahan bubble chat baru di layar tanpa perlu memuat ulang halaman.

Sebaliknya, apabila penerima sedang offline (yakni receiverSocketId tidak ditemukan dalam userSocketMap), maka *server* akan mencatat kondisi ini dengan log console.log(Receiver ${message.receiverId} is not online). Meskipun pesan tidak dapat diteruskan secara langsung dalam kondisi ini, sistem backend biasanya akan tetap menyimpan pesan tersebut dalam *database* agar dapat dimuat kembali ketika penerima kembali online. Dengan pendekatan ini, platform HELLOPET! mampu menjamin bahwa komunikasi tetap berjalan mulus baik dalam kondisi sinkron (online bersamaan) maupun asinkron (offline dan online tidak bersamaan).

Fungsi sendMessage ini menjadi tulang punggung dari arsitektur komunikasi real-time HELLOPET!, yang tidak

hanya meningkatkan efisiensi interaksi antara pengguna dan dokter hewan, tetapi juga menambah nilai kenyamanan dan keandalan dari layanan konsultasi daring yang disediakan oleh platform.

3) Event "markAsRead"

Dalam sistem komunikasi real-time HELLOPET!, keberadaan fitur pelacakan status pesan — khususnya status "sudah dibaca" — merupakan bagian penting dalam membangun pengalaman pengguna yang transparan dan informatif. Fitur ini mengadopsi konsep yang lazim ditemukan pada platform chatting modern, seperti WhatsApp atau Telegram, yang memberi indikator kepada pengirim bahwa pesan mereka telah dilihat oleh penerima. Untuk mewujudkan hal ini secara teknis, backend HELLOPET! mengimplementasikan event markAsRead yang diproses menggunakan teknologi Socket.IO.

```
socket.on("markAsRead", (data) => {
    const { senderId, receiverId } = data;

    const senderSocketId = userSocketMap[receiverId];
    if (senderSocketId) {
      io.to(senderSocketId).emit("messagesRead", {
        userId: senderId,
      });
    }
  });
```

Gambar 3. 40 *Mark As Read Socket*

Event markAsRead dipicu ketika pengguna, dalam hal ini baik Pet Lover maupun Dokter Hewan, membuka atau melihat pesan yang diterima dalam sesi konsultasi mereka. Pada sisi frontend, saat pengguna membuka layar obrolan

dan membaca pesan yang sebelumnya belum dilihat, event ini dikirimkan ke server melalui WebSocket. Payload dari event ini berbentuk objek data yang berisi dua parameter penting: senderId dan receiverId. senderId mengacu pada ID pengguna yang sebelumnya mengirim pesan, sedangkan receiverId adalah ID pengguna yang baru saja membuka atau membaca pesan tersebut.

Setelah event diterima oleh server melalui listener socket.on("markAsRead", ...), langkah pertama yang dilakukan adalah mengambil senderSocketId dari struktur data userSocketMap, yaitu sebuah objek yang menyimpan pemetaan antara user ID dan socket ID mereka yang saat ini sedang aktif. Dalam konteks ini, receiverId digunakan untuk mendapatkan socket ID dari pengirim awal pesan, karena informasi status "dibaca" akan dikirimkan kembali kepadanya.

Jika pengirim pesan (yang berperan sebagai penerima status markAsRead) sedang online dan socket-nya aktif, maka server akan menggunakan metode io.to(senderSocketId).emit("messagesRead", { userId: senderId }) untuk mengirimkan notifikasi secara real-time. Event messagesRead ini memberitahu pengirim pesan bahwa pesan mereka telah dibaca oleh pihak tujuan. Parameter userId: senderId dalam payload notifikasi memberi konteks kepada klien bahwa pesan yang dikirim oleh pengguna dengan ID tersebut telah dibuka oleh lawan bicara mereka.

Namun, apabila socket ID dari pengirim tidak ditemukan (yang berarti pengirim sedang offline atau koneksinya terputus), maka event ini tidak dapat dipancarkan secara langsung. Dalam skenario tersebut, sistem umumnya akan

menyimpan status baca dalam basis data, dan status ini akan ditampilkan kepada pengirim begitu mereka kembali online atau membuka aplikasi kembali.

Implementasi event markAsRead ini tidak hanya meningkatkan kualitas interaksi antar pengguna, tetapi juga menumbuhkan kepercayaan dalam komunikasi daring. Dengan adanya indikator bahwa pesan telah dibaca, pengguna tidak merasa ragu apakah pesan mereka sudah diterima atau belum. Hal ini sangat krusial dalam konteks konsultasi kesehatan hewan yang sensitif terhadap waktu dan kejelasan informasi. Keberadaan fitur ini menjadikan HELLOPET! semakin menyerupai platform komunikasi profesional yang siap menangani kebutuhan konsultasi real-time dengan standar tinggi.

4) Event "disconnect"

Dalam sistem komunikasi real-time berbasis WebSocket seperti yang diimplementasikan pada platform HELLOPET!, menjaga konsistensi status koneksi pengguna merupakan salah satu aspek krusial untuk menjamin pengalaman pengguna yang mulus dan efisien. Salah satu bagian penting dalam hal ini adalah penanganan event disconnect, yang secara otomatis dipicu ketika pengguna kehilangan koneksi WebSocket mereka — baik karena menutup aplikasi, berpindah jaringan, keluar dari aplikasi, atau mengalami gangguan koneksi.

```
socket.on("disconnect", () => {
    console.log("User disconnected:", socket.id);

    if (userId && userId !== "undefined") {
      delete userSocketMap[userId];
      delete typingUsers[userId];

      io.emit("getOnlineUsers", Object.keys(userSocketMap));
    }
  });
```

Gambar 3. 41 *Disconnect Socket*

Kode socket.on("disconnect", ...) bertanggung jawab untuk mengelola logika ketika koneksi dari sisi klien terputus. Saat koneksi terputus, server secara langsung menerima pemberitahuan melalui event ini dan menjalankan prosedur tertentu untuk menjaga agar status sistem tetap valid dan bersih dari data yang tidak relevan.

Langkah pertama yang dilakukan saat event ini dipicu adalah mencetak log ke konsol menggunakan console.log("User disconnected:", socket.id);. Ini berguna untuk keperluan debugging dan monitoring selama pengembangan atau produksi, karena membantu developer mengidentifikasi siapa saja pengguna yang keluar atau kehilangan koneksi.

Setelah itu, sistem melakukan validasi terhadap nilai userId — yakni ID pengguna yang dikaitkan dengan socket yang terputus. Pemeriksaan if (userId && userId !== "undefined") memastikan bahwa hanya pengguna yang valid dan teridentifikasi dengan benar di sistem yang akan diproses lebih lanjut. Jika nilai userId sah, maka dua tindakan utama dilakukan:

1. **Penghapusan dari userSocketMap**

Objek userSocketMap merupakan peta data sementara yang menyimpan hubungan antara user ID dan socket ID. Hubungan ini penting agar sistem dapat mengetahui socket mana yang terhubung ke user tertentu dan memungkinkan komunikasi dua arah secara spesifik. Ketika pengguna terputus, data miliknya harus dihapus dari objek ini dengan perintah delete userSocketMap[userId];. Hal ini bertujuan untuk mencegah terjadinya mis-routing pesan atau pengiriman ke socket yang sudah tidak aktif.

2. **Penghapusan dari typingUsers**

   Selain koneksi socket, sistem HELLOPET! juga memiliki fitur indikator pengetikan real-time yang bergantung pada status yang disimpan dalam objek typingUsers. Jika pengguna sedang mengetik pada saat koneksi terputus, maka informasi tersebut harus dihapus agar tidak terjadi false positive yang menampilkan bahwa pengguna masih mengetik meskipun mereka sudah offline. Maka, baris delete typingUsers[userId]; digunakan untuk membersihkan informasi status pengetikan pengguna dari sistem.

Setelah proses pembersihan data pengguna selesai, sistem kemudian memancarkan event getOnlineUsers menggunakan io.emit(...) ke seluruh klien yang masih aktif. Payload dari event ini berisi daftar terbaru user ID yang masih online, yaitu hasil dari Object.keys(userSocketMap). Dengan demikian, semua pengguna lain akan langsung mendapatkan pembaruan daftar pengguna aktif secara real-time, dan dapat menyesuaikan tampilan antarmuka mereka

misalnya, dengan menyembunyikan indikator "online" pada pengguna yang baru saja terputus.

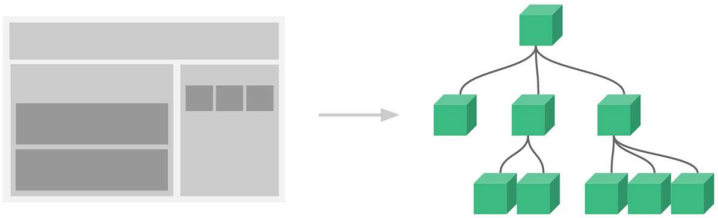
Fungsi disconnect ini bukan hanya menjaga integritas data dan komunikasi, tetapi juga menjadi pondasi penting dalam membangun sistem komunikasi yang skalabel dan andal. Dalam konteks aplikasi HELLOPET! yang menghubungkan pemilik hewan dengan dokter hewan secara daring, fitur ini memastikan bahwa hanya pengguna yang benar-benar aktif yang dianggap tersedia untuk konsultasi, dan mencegah miskomunikasi akibat status yang tidak sinkron.

3.5.1.4. Mengintegrasi Fitur Utama di *Frontend*

Dalam tahap ini, fokus utama dari proses pengembangan aplikasi diarahkan pada pembuatan dan penyempurnaan antarmuka pengguna (*user interface*/UI) yang tidak hanya menarik secara visual, namun juga intuitif, interaktif, dan mudah dioperasikan oleh berbagai jenis pengguna, termasuk pengguna awam. Antarmuka ini dirancang agar mampu meningkatkan pengalaman pengguna (*user experience*/UX) secara keseluruhan, sehingga interaksi dengan sistem terasa lebih alami, efisien, dan menyenangkan. Untuk mencapai tujuan tersebut, dipilihlah Vue.js, sebuah framework JavaScript progresif yang telah banyak digunakan dalam pengembangan frontend aplikasi modern, sebagai teknologi inti.

Vue.js dipilih karena menawarkan kombinasi ideal antara kesederhanaan dan fleksibilitas, yang sangat cocok untuk membangun aplikasi berbasis *Single Page Application* (SPA). Konsep SPA memungkinkan aplikasi hanya memuat satu halaman HTML utama di awal, kemudian memperbarui konten secara dinamis di dalam halaman yang sama sesuai dengan navigasi pengguna, tanpa perlu melakukan pemuatan ulang (*reload*) secara penuh. Hal ini berdampak langsung pada peningkatan kecepatan interaksi pengguna, pengurangan beban

jaringan, serta pengalaman yang lebih mulus dan konsisten antar halaman.



Gambar 3. 42 Konsep Komponen

Salah satu kekuatan utama Vue.js adalah pendekatannya yang berbasis arsitektur komponen (component-based architecture). Dengan sistem ini, setiap bagian dari antarmuka mulai dari tombol, formulir, kartu informasi, hingga halaman penuh dapat dipisahkan ke dalam komponen independen yang memiliki logika, tampilan, dan gaya tersendiri. Misalnya, pada aplikasi HELLOPET!, halaman login dibuat sebagai Login.vue, halaman utama pengguna sebagai Dashboard.vue, layanan konsultasi dokter sebagai Chat.vue, dan layanan grooming sebagai ServiceList.vue. Dengan struktur ini, pengembangan menjadi lebih terorganisir, pemeliharaan kode lebih mudah dilakukan, serta potensi penggunaan ulang (reusability) meningkat drastis karena komponen bisa dipakai kembali di berbagai konteks.

Vue.js juga memiliki mekanisme reactive data-binding, yaitu kemampuan menghubungkan data dan tampilan secara otomatis. Ketika data berubah, tampilan akan secara langsung menyesuaikan tanpa perlu menulis kode tambahan untuk memperbarui DOM. Ini menjadikan proses pembangunan antarmuka lebih efisien dan mengurangi kemungkinan terjadinya bug.

Dalam implementasinya, proyek HELLOPET! juga memanfaatkan berbagai ekosistem pendukung Vue, seperti **Vue Router**

dan **Pinia**. Vue Router digunakan untuk mengatur navigasi antar halaman seperti login, registrasi, dashboard pengguna, hingga halaman konsultasi, semuanya dilakukan secara real-time tanpa perpindahan fisik halaman, yang menjadi ciri khas dari SPA. Sementara itu, **Pinia** digunakan untuk mengelola data global, seperti status login pengguna, data layanan yang tersedia, dan riwayat konsultasi, sehingga data dapat diakses dengan mudah di berbagai komponen tanpa harus meneruskannya secara manual. Fitur-fitur utama yang berhasil dikembangkan menggunakan Vue.js pada sisi frontend HELLOPET! meliputi:

1. **Formulir login dan registrasi** yang dilengkapi dengan validasi data pengguna, sehingga pengguna dapat masuk atau mendaftar akun dengan aman, cepat, dan minim kesalahan input.
2. **Sistem chat interaktif berbasis WebSocket**, yang memungkinkan komunikasi langsung antara pengguna dan dokter hewan berlangsung secara dua arah, cepat, dan mendekati waktu nyata (real-time), sehingga meningkatkan efektivitas konsultasi.

Selain dari sisi teknis, desain UI juga dikembangkan dengan prinsip **mobile-first dan responsive design**, memastikan tampilan aplikasi tetap optimal di berbagai ukuran layar, termasuk smartphone, tablet, dan desktop. Desain ini sangat penting karena mayoritas pengguna aplikasi diperkirakan mengakses layanan melalui perangkat seluler.

Secara keseluruhan, pemanfaatan Vue.js dalam pengembangan frontend HELLOPET! terbukti memberikan dampak positif dalam hal efisiensi pengembangan, kualitas antarmuka, serta kenyamanan penggunaan. Arsitektur modular, performa tinggi, dan ekosistem yang matang menjadikan Vue.js pilihan yang tepat dalam membangun aplikasi web

modern yang kompleks namun tetap mudah dikelola dan dikembangkan di masa depan.

a) Fitur Authentication & Authorization

Fitur Authentication & Authorization merupakan bagian fundamental dalam sistem aplikasi web HELLOPET! yang bertujuan untuk membatasi akses hanya kepada pengguna yang berwenang. Autentikasi memastikan bahwa hanya pengguna yang memiliki kredensial valid yang dapat masuk ke dalam sistem, sementara otorisasi menentukan apa yang dapat diakses oleh masing-masing pengguna berdasarkan peran atau role-nya (contohnya: pengguna biasa vs. dokter hewan).

Untuk membangun fitur ini secara efisien di sisi frontend, digunakan Pinia sebagai state management library resmi untuk Vue 3. Pinia menggantikan Pinia dan menyediakan cara yang lebih ringan, deklaratif, dan modular untuk mengelola state aplikasi secara global, termasuk data autentikasi pengguna dan token akses.

1) Login

Fitur Login merupakan bagian penting dari sistem keamanan dan autentikasi pada platform ini. Melalui proses login, pengguna yang sudah memiliki akun dapat mengakses layanan dan fitur yang tersedia sesuai dengan peran mereka, seperti pengguna umum maupun dokter. Pada tahap ini, pengguna diminta untuk memasukkan alamat email dan kata sandi yang telah terdaftar sebelumnya.

Jika informasi yang dimasukkan valid dan cocok dengan data yang tersimpan di *database*, maka sistem akan mengizinkan akses dengan memberikan token autentikasi sebagai tanda bahwa pengguna telah berhasil masuk. Selain itu, proses login ini dilengkapi dengan umpan balik visual

menggunakan library SweetAlert2 untuk memberikan notifikasi sukses atau gagal kepada pengguna secara interaktif dan informatif. Proses login ini juga secara otomatis memeriksa peran (role) dari pengguna, dan mengarahkan mereka ke halaman yang sesuai. Misalnya, jika pengguna berperan sebagai dokter, maka sistem akan mengalihkan ke halaman dashboard dokter, sedangkan untuk pengguna umum akan diarahkan ke halaman beranda.

Gambar 3. 43 Fungsi submitForm

Mendeklarasikan fungsi submitForm sebagai fungsi asynchronous. Artinya, fungsi ini akan menangani operasi yang berjalan secara asynchronous seperti request ke API.

Gambar 3. 44 Mengirim data melalui API

Fungsi mencoba memanggil metode login dari store Pinia (auth) dengan payload yang berisi email dan password dari

objek form. await digunakan karena auth.login() merupakan operasi asynchronous (biasanya mengirim HTTP POST ke endpoint login). Nilai yang dikembalikan (res) akan berupa true jika login berhasil, atau melempar error jika gagal.

Gambar 3. 45 *Authorization*

Setelah pengguna menutup notifikasi, dilakukan pengalihan halaman (routing) berdasarkan peran pengguna. Jika peran pengguna (roles) adalah 'dokter', maka pengguna diarahkan ke dashboard dokter dengan ID spesifik. Jika bukan, diarahkan ke halaman utama ('/').

2) Register

Fitur Register atau pendaftaran digunakan oleh pengguna baru untuk membuat akun dan mengakses layanan yang disediakan oleh platform. Dalam proses ini, pengguna harus mengisi sejumlah informasi penting seperti nama lengkap, alamat email yang valid, nomor telepon, serta kata sandi yang akan digunakan untuk login ke sistem. Semua data yang dimasukkan akan melalui proses validasi untuk memastikan bahwa tidak ada data yang kosong atau tidak sesuai format.

Setelah validasi berhasil, data tersebut akan dikirim ke server, dan kata sandi pengguna akan diubah menjadi bentuk hash menggunakan algoritma bcrypt sebelum disimpan ke

dalam *database*, untuk menjamin keamanan data sensitif. Jika proses registrasi berhasil, pengguna akan mendapatkan notifikasi berupa pesan sukses dan dapat langsung login ke dalam sistem menggunakan kredensial yang telah dibuat.

Fitur ini juga telah didesain agar mudah digunakan dan memberikan *feedback* interaktif kepada pengguna jika terjadi kesalahan selama proses pendaftaran, seperti alamat email yang sudah digunakan sebelumnya. Dengan sistem registrasi ini, platform dapat memastikan bahwa hanya pengguna terverifikasi yang dapat mengakses layanan lebih lanjut, termasuk fitur komunikasi dan transaksi.

Gambar 3. 46 Fungsi submitForm

Mendeklarasikan fungsi submitForm sebagai fungsi asynchronous. Artinya, fungsi ini akan menangani operasi yang berjalan secara asynchronous seperti request ke API.

Gambar 3. 47 Memasukan data kedalam API register

Fungsi ini memanggil method register() dari Pinia store auth. Objek yang dikirim berupa data dari form:

i. username → nama pengguna
ii. email → alamat email
iii. password → kata sandi
iv. phone_number → nomor telepon

Semua data tersebut berasal dari form reaktif (form) yang sebelumnya diisi oleh pengguna.Hasil pemanggilan fungsi register() disimpan dalam variabel success. Jika pendaftaran berhasil, success akan bernilai true.

Gambar 3. 48 Notifikasi Berhasil Register

Jika nilai success bernilai true, maka sistem akan menampilkan pop-up notifikasi sukses menggunakan SweetAlert2. Notifikasi tersebut akan menampilkan ikon berbentuk centang hijau yang menunjukkan keberhasilan, dengan judul "Berhasil" dan pesan yang menyatakan "Pendaftaran berhasil!". Setelah pengguna menutup notifikasi tersebut, sistem secara otomatis akan mengarahkan pengguna ke halaman login menggunakan metode router.push() yang tersedia dalam Vue Router. Proses ini bertujuan untuk memastikan bahwa pengguna langsung diarahkan ke tahap berikutnya, yaitu proses masuk ke dalam sistem setelah berhasil melakukan pendaftaran.

b) Real Time Konsultasi

Fitur Real Time Konsultasi merupakan salah satu fitur utama yang disediakan dalam platform HELLOPET!, yang memungkinkan pengguna dan dokter hewan untuk saling berkomunikasi secara langsung dan interaktif melalui sistem percakapan yang berlangsung secara waktu nyata (real-time). Dengan adanya fitur ini, pengguna tidak perlu lagi menunggu

balasan dalam waktu yang lama karena pesan akan dikirim dan diterima secara instan oleh kedua belah pihak. Sistem ini menggunakan teknologi WebSocket, yang memungkinkan terjadinya koneksi dua arah secara simultan antara klien (pengguna) dan server, sehingga setiap perubahan atau kiriman pesan dapat ditampilkan secara langsung tanpa perlu melakukan refresh halaman.

Fitur ini sangat membantu dalam mempercepat proses konsultasi, terutama saat hewan peliharaan mengalami kondisi darurat atau gejala yang memerlukan penanganan segera. Pengguna cukup mengetikkan pesan, mengirim gambar, atau menyampaikan pertanyaan, dan dokter akan langsung merespons sesuai dengan waktu yang tersedia. Seluruh riwayat percakapan juga disimpan di sistem, sehingga pengguna bisa melihat kembali saran atau diagnosis yang telah diberikan.

Gambar 3. 49 Fungsi Mengirim Pesan

Mendeklarasikan fungsi **sendMessage** sebagai fungsi asynchronous. Artinya, fungsi ini akan menangani operasi yang berjalan secara asynchronous seperti request ke API

```
const senderId = authStore.user.id
const receiverId = this.selectedUser.id
```

Gambar 3. 50 Menyimpan Data Sementara

Baris kode ini bertujuan untuk mengidentifikasi siapa yang mengirim pesan dan siapa yang menerimanya, yang merupakan hal fundamental dalam sistem komunikasi dua arah seperti chat.

```
const tempId = `temp-${Date.now()}`
const tempMessage = {
  id: tempId,
  senderId,
  receiverId,
  message: text,
  createdAt: new Date().toISOString(),
  status: 'sending',
}
```

Gambar 3. 51 Mengirim Pesan ke *Client*

Sebelum pesan benar-benar dikirim ke server, sistem terlebih dahulu menambahkan pesan tersebut ke antarmuka pengguna (UI) secara langsung dengan status "sending". Tujuannya adalah agar pengguna merasa bahwa proses pengiriman pesan berjalan cepat dan responsif, meskipun sebenarnya pesan masih dalam proses dikirim ke server di latar belakang. Untuk membedakan pesan sementara ini dari pesan yang sudah benar-benar

tersimpan di server, digunakan sebuah tempId (ID sementara) yang dihasilkan berdasarkan waktu saat itu, misalnya dengan format temp-<timestamp>. ID sementara ini sangat penting karena nantinya ketika server membalas dengan pesan versi resminya (yang sudah disimpan di *database*), sistem dapat mencocokkan dan menghapus pesan yang menggunakan tempId tersebut, lalu menggantinya dengan pesan dari server yang memiliki ID sebenarnya dan status telah berubah menjadi "sent".

Gambar 3. 52 Menyimpan Pesan Menggunakan API

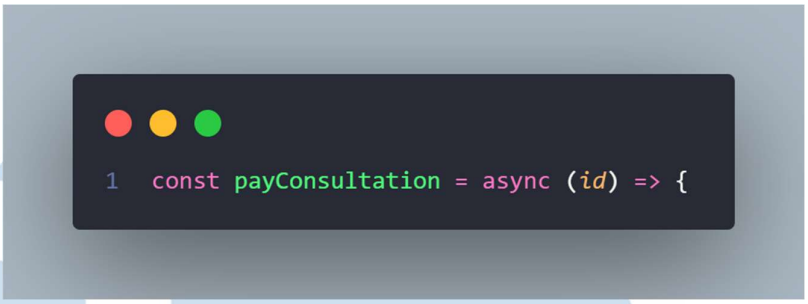
Setelah pesan ditampilkan secara sementara di UI dengan status "sending", sistem kemudian mengirim data pesan tersebut ke backend menggunakan endpoint REST, yaitu dengan melakukan permintaan POST ke URL /api/messages/client/send/${receiverId}. Permintaan ini membawa informasi penting seperti senderId, receiverId, dan isi pesan (message) yang ingin dikirim. Backend kemudian akan memproses pesan ini, menyimpannya ke dalam basis data, dan mengembalikan respons (res.data) yang berisi data lengkap pesan yang telah disimpan, termasuk conversationId yang menandakan identitas percakapan antara dua pengguna. Data inilah yang kemudian digunakan untuk menggantikan pesan sementara sebelumnya, sehingga pesan di UI kini mencerminkan data yang telah benar-benar dikonfirmasi dan dicatat oleh server. Proses ini memastikan konsistensi antara client dan server dalam sistem percakapan real-time.

c) Payment Gateway

Payment Gateway adalah sebuah layanan yang memungkinkan sistem untuk memproses transaksi pembayaran secara online antara pengguna (pelanggan) dan penyedia layanan (merchant atau aplikasi). Teknologi ini bertindak sebagai jembatan penghubung antara aplikasi dan sistem keuangan, seperti bank, dompet digital, atau penyedia kartu kredit, sehingga transaksi dapat dilakukan secara aman, cepat, dan efisien.

Dalam konteks pengembangan website HELLOPET!, fitur Payment Gateway menjadi komponen vital yang memungkinkan pengguna untuk melakukan pembayaran secara langsung di dalam platform, terutama untuk layanan berbayar seperti konsultasi dengan dokter hewan. Integrasi Payment Gateway memastikan bahwa proses transaksi dapat berlangsung tanpa hambatan dan minim risiko, baik dari sisi pengguna maupun pengelola platform.

Salah satu solusi yang digunakan dalam proyek ini adalah Midtrans, yang merupakan salah satu penyedia layanan Payment Gateway paling populer di Indonesia. Midtrans menyediakan layanan Snap UI, yang memudahkan dalam integrasi antarmuka pembayaran tanpa harus membangun sistem dari nol. Snap memungkinkan pengguna untuk memilih metode pembayaran yang diinginkan seperti transfer bank, e-wallet (GoPay, ShopeePay, dll), kartu kredit, atau pembayaran gerai ritel dalam satu tampilan yang aman dan terpercaya.



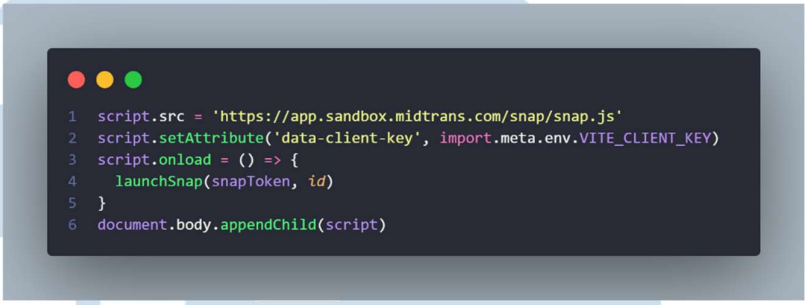
Gambar 3. 53 Fungsi Payment

Mendefinisikan fungsi asynchronous payConsultation dengan parameter id, yang kemungkinan besar merupakan ID dari jadwal konsultasi atau invoice yang ingin dibayar oleh pengguna.

Gambar 3. 54 Menyimpan Pembayaran

Fungsi payConsultation bertugas untuk mengirim permintaan pembayaran ke backend melalui metode POST ke endpoint /api/payment/consulting, dengan menyertakan data berupa id dari layanan konsultasi yang ingin dibayar. Permintaan ini menggunakan konfigurasi withCredentials: true, yang berarti bahwa cookie pengguna—termasuk token otorisasi sesi—akan dikirim bersama permintaan untuk memastikan bahwa hanya pengguna yang telah terautentikasi yang dapat melakukan transaksi. Setelah backend memproses permintaan tersebut, server akan membalas dengan token transaksi dari Midtrans Snap, yang kemudian digunakan oleh frontend untuk memunculkan antarmuka pembayaran dari Midtrans secara otomatis. Token ini menjadi kunci untuk meluncurkan jendela

pembayaran sehingga pengguna dapat langsung menyelesaikan transaksi melalui metode yang mereka pilih, seperti transfer bank, e-wallet, atau kartu kredit.



Gambar 3. 55 Membuat UI Payment Midtrans

Script Snap dari Midtrans dimuat secara dinamis ke dalam halaman dengan cara membuat elemen `<script>` baru menggunakan JavaScript. Proses pemuatan ini menggunakan URL sandbox milik Midtrans, yang berfungsi untuk keperluan pengujian atau testing dan bukan untuk transaksi di lingkungan produksi. Dalam elemen `<script>` tersebut, disisipkan juga *client key* yang diambil dari environment variable bernama `VITE_CLIENT_KEY`, sebagai identifikasi aplikasi kepada sistem Midtrans. Setelah script berhasil dimuat sepenuhnya, fungsi `launchSnap()` akan dipanggil secara otomatis untuk menampilkan antarmuka pembayaran Snap UI ke pengguna dan memulai proses transaksi. Seluruh elemen script ini kemudian ditambahkan ke dalam struktur DOM agar dapat dijalankan oleh browser dengan benar.

3.5.1.5. Penerapan Security Backend

Setelah seluruh fungsionalitas inti dan sistem autentikasi pengguna berhasil diimplementasikan, tahap berikutnya dalam pengembangan sistem HELLOPET! adalah penerapan strategi keamanan pada tingkat aplikasi. Hal ini menjadi sangat penting mengingat sistem HELLOPET! menangani data sensitif seperti

informasi pengguna, detail transaksi, dan rekam jejak konsultasi medis antara pengguna dengan dokter hewan. Oleh karena itu, keamanan backend tidak hanya dijadikan fitur tambahan, tetapi menjadi bagian fundamental dari arsitektur aplikasi.

Proyek ini bertujuan untuk membangun lapisan perlindungan yang kokoh terhadap backend RESTful API dengan fokus pada pencegahan akses tidak sah, pengendalian sumber akses (origin), pengamanan variabel rahasia (secret key, credential), serta perlindungan terhadap potensi serangan seperti Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), dan exposure credential. Berikut proses implementasi yang dilakukan di website HELLOPET!, yaitu:

3) **Konfigurasi CORS (Cross-Origin Resource Sharing)**

Untuk memastikan keamanan dan integritas komunikasi antara frontend dan backend HELLOPET!, diterapkan mekanisme Cross-Origin Resource Sharing (CORS) sebagai salah satu lapisan pertahanan utama dalam arsitektur backend. CORS adalah kebijakan keamanan yang diimplementasikan oleh browser modern untuk mencegah aplikasi web dari domain tertentu mengakses resource yang berada di domain berbeda tanpa izin eksplisit dari server tujuan.

Dalam konteks sistem HELLOPET!, di mana frontend dan backend dapat berjalan di domain atau port yang berbeda, penerapan CORS menjadi sangat penting. Misalnya, frontend berjalan pada https://hellopet.id, sedangkan backend berjalan di https://api.hellopet.id atau bahkan dalam tahap pengembangan di http://localhost:3000. Tanpa konfigurasi CORS yang tepat, browser secara default akan memblokir permintaan lintas domain tersebut, dan interaksi

antara UI dan server akan gagal. Penerapan CORS pada sistem HELLOPET! bertujuan untuk:

1) **Mencegah akses tidak sah** dari situs atau aplikasi pihak ketiga yang mencoba melakukan request ke server API secara langsung.
2) **Mengizinkan komunikasi resmi** antara frontend HELLOPET! dan backend, baik dalam pengambilan data (GET), pengiriman data (POST), maupun permintaan update dan penghapusan (PUT, DELETE).
3) **Meningkatkan keamanan komunikasi lintas origin** dengan membatasi hanya origin yang terpercaya yang dapat mengakses resource backend.
4) **Menghindari eksploitasi API** oleh klien atau bot dari luar sistem yang mencoba mengakses data atau melakukan manipulasi transaksi tanpa izin.

Untuk mengimplementasikan CORS, digunakan middleware cors dari ekosistem NPM yang terintegrasi dengan framework Express.js. Middleware ini secara default akan menyisipkan header HTTP seperti Access-Control-Allow-Origin, Access-Control-Allow-Methods, dan Access-Control-Allow-Credentials, yang menjadi indikator bagi browser apakah permintaan dari origin tertentu boleh dilanjutkan atau harus diblokir.

Konfigurasi middleware CORS pada HELLOPET! disusun secara dinamis menggunakan variabel lingkungan (.env) untuk memungkinkan fleksibilitas antar environment (development, staging, production). Berikut adalah contoh pengaturan CORS di dalam file konfigurasi backend:

```
app.use(
  cors({
    origin: [
      "https://www.hellopet.site",
      "https://hellopet.site",
      "http://localhost:5173",
    ],
    methods: ["GET", "POST", "PATCH", "PUT", "DELETE"],
    allowedHeaders: ["Content-Type", "Authorization"],
    credentials: true,
  })
);
```

Gambar 3. 56 Mengimplementasi CORS

Langkah ini secara efektif mencegah situs web jahat lain memanggil API HELLOPET! dari browser pengguna, melindungi dari serangan seperti Cross-Site Request Forgery (CSRF) dan pencurian data.

4) Middleware Akses API

Sebagai bagian penting dari pengamanan sistem backend HELLOPET!, diterapkan sebuah middleware khusus bernama originCheck, yang berfungsi untuk memverifikasi asal (origin) dari setiap permintaan HTTP yang masuk ke server. Middleware ini dirancang untuk mencegah permintaan yang berasal dari domain yang tidak sah atau mencurigakan, yang dapat berpotensi membahayakan data pengguna maupun kestabilan sistem secara keseluruhan. Fungsi keamanan ini menjadi sangat krusial terutama dalam arsitektur aplikasi modern yang memisahkan antara frontend dan backend pada domain yang berbeda, seperti https://hellopet.site dan https://api.hellopet.site.

Middleware originCheck dikembangkan menggunakan JavaScript pada lingkungan Node.js, dan diekspor sebagai

middleware standar dalam format Express.js. Fungsinya dimulai dengan mendeklarasikan sebuah array bernama allowedOrigins, yang berisi daftar origin yang secara eksplisit diperbolehkan untuk mengakses backend HELLOPET!, yaitu https://hellopet.site dan https://www.hellopet.site. Daftar ini dibuat secara ketat untuk mencegah serangan dari domain luar yang mungkin mencoba menyisipkan permintaan melalui metode seperti Cross-Site Scripting (XSS) atau Cross-Origin Resource Sharing (CORS) exploit.

```
export const originCheck = (req, res, next) => {
  const allowedOrigins = ["https://hellopet.site", "https://www.hellopet.site"];
  const origin = req.headers.origin || req.headers.referer;
  if (
    !origin ||
    !allowedOrigins.some((allowed) => origin.startsWith(allowed))
  ) {
    console.log("Blocked origin:", origin);
    return res.status(403).json({ message: "Access forbidden" });
  }
  console.log("Received headers:", {
    origin: req.headers.origin,
    referer: req.headers.referer,
    host: req.headers.host,
  });
  next();
};
```

Gambar 3. 57 Mengimplementasi Middleware Origin

Saat sebuah permintaan HTTP diterima, middleware akan mencoba membaca nilai origin dari header permintaan (req.headers.origin). Jika nilai tersebut tidak tersedia, maka middleware akan mencoba menggunakan nilai referer sebagai alternatif identifikasi asal permintaan. Setelah itu, dilakukan pengecekan apakah origin atau referer tersebut sesuai dengan salah satu domain yang ada di dalam array allowedOrigins. Proses ini dilakukan dengan metode .startsWith() agar dapat mencakup subdomain atau variasi struktur URL secara fleksibel, namun tetap aman.

Jika origin dari permintaan tersebut tidak masuk ke dalam daftar yang diperbolehkan, maka permintaan langsung ditolak dengan mengembalikan HTTP response 403 Forbidden beserta pesan "Access forbidden". Di sisi lain, jika permintaan berasal dari origin yang sah, maka proses akan diteruskan ke middleware atau controller berikutnya menggunakan next().

Untuk keperluan debugging dan audit keamanan, middleware ini juga mencetak ke log informasi penting terkait permintaan yang diterima, termasuk nilai origin, referer, dan host. Informasi ini sangat berguna dalam proses pelacakan dan pengujian, terutama saat terjadi percobaan akses ilegal dari domain yang tidak dikenal.

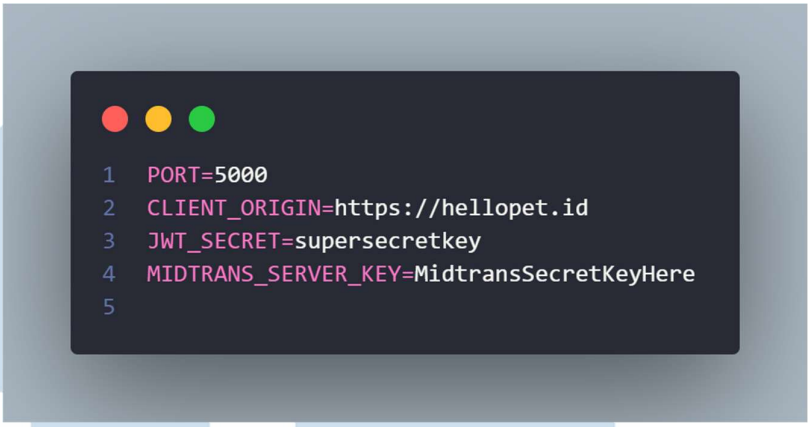
Middleware originCheck bekerja secara sinergis dengan middleware CORS standar seperti cors() dari package cors di Node.js. Di sisi konfigurasi global, cors() digunakan untuk mengatur metode HTTP yang diperbolehkan, pengiriman kredensial, serta pengaturan headers tambahan. Sedangkan originCheck memberikan lapisan keamanan tambahan dengan secara eksplisit menolak permintaan dari domain yang tidak dikenali, bahkan sebelum CORS diproses sepenuhnya.

Penerapan originCheck ini menjadi salah satu bukti komitmen pengembangan sistem HELLOPET! terhadap prinsip keamanan by design. Dengan menerapkan filter akses berbasis origin, sistem tidak hanya mencegah penyalahgunaan API oleh pihak ketiga, tetapi juga memberikan kontrol granular terhadap siapa yang dapat mengakses layanan backend. Hal ini menjadi sangat penting untuk menjaga kepercayaan pengguna dan memastikan bahwa data konsultasi, informasi dokter, hingga histori

percakapan tidak terekspos kepada pihak yang tidak berwenang.

5) **Pengelolaan Kredensial Sensitif dengan File .ENV**

Dalam pengembangan aplikasi backend HELLOPET!, aspek keamanan dan fleksibilitas konfigurasi menjadi prioritas utama. Salah satu pendekatan penting yang diterapkan untuk memenuhi kebutuhan tersebut adalah dengan menggunakan file konfigurasi lingkungan .env (environment file). File .env berfungsi sebagai tempat penyimpanan seluruh informasi sensitif dan variabel konfigurasi penting yang digunakan oleh aplikasi, seperti URL *database*, nomor port server, secret key untuk autentikasi JWT (JSON Web Token), API key Midtrans untuk pemrosesan pembayaran, serta daftar whitelist origin yang diizinkan mengakses layanan backend. Penggunaan file .env memiliki beberapa keunggulan krusial. Pertama, praktik ini memungkinkan pemisahan konfigurasi dari kode utama aplikasi, sehingga memudahkan manajemen lingkungan (environment management). Misalnya, konfigurasi pada tahap development dapat berbeda dari konfigurasi pada lingkungan staging maupun production, tanpa perlu mengubah kode program secara langsung. Hal ini selaras dengan prinsip separation of concerns, yaitu pemisahan antara logika bisnis dengan konfigurasi operasional.



Gambar 3. 58 Mengimplementasi .ENV File

Kedua, file .env mendukung praktik keamanan modern dengan memastikan bahwa informasi rahasia tidak tersebar ke luar secara tidak sengaja. Untuk itu, file .env secara eksplisit dimasukkan ke dalam .gitignore, sehingga tidak akan ikut tersimpan dalam repositori Git, baik dalam versi publik maupun pribadi. Langkah ini sangat penting untuk mencegah kebocoran data seperti kunci API pembayaran (seperti Midtrans) yang jika jatuh ke tangan pihak tidak bertanggung jawab, dapat disalahgunakan untuk tujuan yang merugikan.

Ketiga, pendekatan ini mendukung prinsip secure deployment practices, di mana setiap lingkungan (server staging, production, atau bahkan CI/CD pipeline) memiliki file .env masing-masing yang dapat diatur secara independen. Hal ini juga mempermudah proses deployment otomatis karena sistem CI/CD dapat membaca konfigurasi dari file .env tanpa perlu menyimpan kredensial secara hardcoded dalam kode.

Secara teknis, semua variabel yang didefinisikan di file .env akan diakses melalui process.env di dalam kode Node.js. Contohnya, untuk mengakses kunci JWT, cukup

menggunakan process.env.JWT_SECRET. Hal ini membuat kode lebih bersih, aman, dan mudah untuk diuji dalam berbagai skenario.

Secara keseluruhan, penerapan file .env pada pengembangan backend HELLOPET! bukan hanya langkah praktis, tetapi juga bagian integral dari strategi keamanan aplikasi. Praktik ini memungkinkan kontrol yang lebih baik terhadap variabel lingkungan yang sensitif, menjaga kerahasiaan kredensial, serta memastikan bahwa proses deployment ke berbagai server tetap konsisten, aman, dan terstandarisasi.

3.5.2 Kendala yang Ditemukan

Selama proses pengembangan platform HELLOPET!, beberapa kendala teknis dan non-teknis ditemukan, antara lain:

1. **Kesulitan dalam Integrasi Fitur *Real-Time***: Implementasi WebSocket untuk fitur konsultasi *real-time* awalnya menemui tantangan, terutama dalam mengelola koneksi yang stabil dan memastikan pesan terkirim dan diterima secara konsisten antar pengguna yang berbeda.
2. **Manajemen *State* Aplikasi yang Kompleks:** Seiring bertambahnya fitur, pengelolaan *state* (data aplikasi) di sisi *frontend* Vue.js menjadi rumit. Data seperti status login pengguna, informasi profil, dan pesan obrolan harus konsisten di berbagai komponen yang berbeda.
3. **Konfigurasi Keamanan Lintas Domain (CORS):** Memahami dan mengonfigurasi CORS dengan benar pada awalnya menjadi kendala. Kesalahan konfigurasi menyebabkan *frontend* tidak dapat berkomunikasi dengan *backend* meskipun keduanya berjalan normal secara terpisah.

3.5.3 Solusi atas Kendala yang Ditemukan

Untuk mengatasi kendala-kendala yang ditemukan, beberapa solusi diterapkan sebagai berikut:

1. **Solusi Integrasi *Real-Time*:** Kendala WebSocket diatasi dengan melakukan riset mendalam pada dokumentasi pustaka socket.io dan mempelajari studi kasus implementasinya. Solusinya adalah dengan membuat prototipe kecil terisolasi terlebih dahulu untuk memahami alur kerja event-emitting dan broadcasting sebelum mengintegrasikannya ke dalam proyek utama.
2. ***Solusi Manajemen State*:** Untuk mengatasi kompleksitas state pada Vue.js, solusi yang diterapkan adalah mengadopsi pustaka manajemen state terpusat, yaitu Pinia. Dengan Pinia, state aplikasi dipecah menjadi modul-modul yang logis (misalnya authStore, chatStore), sehingga lebih mudah dikelola, dilacak, dan diakses dari komponen manapun.
3. **Solusi Konfigurasi CORS:** Kendala CORS diselesaikan dengan menggunakan pustaka cors untuk Express.js dan membaca dokumentasinya dengan saksama. Solusinya adalah dengan mengonfigurasi whitelist secara eksplisit, yaitu hanya mengizinkan domain tempat aplikasi frontend di-hosting, sambil tetap memblokir semua domain lainnya. Pengujian dilakukan secara intensif menggunakan alat seperti Postman.