BAB V
PENUTUP

5.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan mengenai analisis perbandingan performa *WebSocket* menggunakan *Pusher* dan *Laravel Reverb* pada implementasi *Live Cam website* muncak.id, dapat diambil beberapa kesimpulan sebagai berikut:

a. Implementasi Fitur *Live Cam* dengan *Pusher*

   Implementasi fitur *Live Cam* menggunakan *Pusher* sebagai layanan *cloudbased WebSocket* telah berhasil dilakukan dengan baik. Sistem mampu menangani *streaming* video *real-time*, komunikasi *chat*, pembaruan jumlah *viewer*, dan klasifikasi AI kondisi jalur secara otomatis. *Pusher* menyediakan infrastruktur *WebSocket* yang matang dengan kemampuan *auto-scaling* otomatis, kemudahan integrasi melalui API yang simpel, dan dokumentasi yang lengkap. Proses *deployment* tidak memerlukan konfigurasi server *WebSocket* yang kompleks karena seluruh infrastruktur komunikasi *real-time* dikelola oleh *Pusher*.

b. Implementasi Fitur *Live Cam* dengan *Laravel Reverb*

   Implementasi fitur *Live Cam* menggunakan *Laravel Reverb* sebagai *server WebSocket self-hosted* telah berhasil dilakukan dengan fungsionalitas yang identik dengan implementasi *Pusher*. *Laravel Reverb* menyediakan integrasi *native* dengan ekosistem *Laravel* melalui sistem *broadcasting* yang sudah terintegrasi. Proses *deployment* memerlukan konfigurasi tambahan seperti instalasi *Supervisor* untuk menjalankan *Reverb server* sebagai *daemon* dan

pengaturan Redis sebagai *backing storage*. Meskipun memerlukan pengelolaan *server* secara mandiri, *Reverb* memberikan kontrol penuh terhadap infrastruktur komunikasi *real-time* tanpa ketergantungan pada layanan pihak ketiga.

c. Perbandingan Performa Kedua Implementasi *WebSocket*

Hasil pengujian performa menunjukkan perbedaan karakteristik yang signifikan antara kedua implementasi sebagaimana ditunjukkan pada Tabel 33.

Tabel 33. Perbandingan Performa *Pusher* dan *Laravel Reverb*

Aspek Performa	***Pusher***	***Laravel Reverb***	**Keterangan**
Latensi *WebSocket*	162,4 ms (median)	2.186,8 ms (median)	Pusher 13× lebih cepat
Latensi HTTP	252,2 ms (median)	11.050,8 ms (median)	Pusher 44× lebih cepat
Tingkat Keberhasilan	100% (0 error)	99,01% (0,99% error)	Pusher lebih stabil
Keberhasilan Koneksi	100%	71,93% (gagal 28,07%)	Pusher lebih andal
Penggunaan CPU	199,70% (rata-rata)	303,27% (rata-rata)	Pusher 52% lebih efisien
Penggunaan Memori	11.164,90 MB	10.495,04 MB	Reverb 6% lebih hemat
Throughput	15 request/detik	23 request/detik	Reverb 53% lebih tinggi
Koneksi Bersamaan	1.394 koneksi	2.114 koneksi	Reverb 52% lebih banyak
Auto-scaling	Ya (otomatis)	Tidak (manual)	Pusher unggul skalabilitas

Berdasarkan tabel di atas, *Pusher* unggul dalam aspek latensi, stabilitas, keandalan, dan efisiensi CPU dengan kemampuan *auto-scaling* otomatis. Sementara itu, *Laravel Reverb* unggul dalam *throughput*, kapasitas koneksi bersamaan, dan efisiensi memori, namun mengalami kegagalan koneksi

signifikan (28,07%) pada lonjakan trafik mendadak karena ketiadaan mekanisme *auto-scaling*.

d. Implementasi Fitur Klasifikasi Tingkat Keramaian dengan API *Artificial Intelligence*

Implementasi fitur klasifikasi tingkat keramaian menggunakan API *artificial intelligence* telah berhasil diintegrasikan pada sistem *Live Cam*. Sistem mampu melakukan *capture frame* dari *video stream* setiap 30 menit, mengirim gambar ke AI *Model* API, dan menerima hasil klasifikasi kondisi jalur yang mencakup cuaca (cerah, berawan, hujan), tingkat keramaian (sepi, sedang, ramai), dan visibilitas (jelas, kabut sedang, kabut tebal) lengkap dengan *confidence score*. Hasil klasifikasi disimpan ke basis data dan di-*broadcast* secara *real-time* kepada seluruh *viewer* melalui *WebSocket*, sehingga pendaki dapat memperoleh informasi kondisi jalur secara objektif tanpa perlu interpretasi manual dari *streaming* video. Fitur ini memberikan nilai tambah signifikan pada sistem *Live Cam* dengan menyediakan informasi kuantitatif yang dapat membantu pendaki dalam pengambilan keputusan.

e. Rekomendasi Solusi *WebSocket* Optimal

Berdasarkan hasil analisis performa, rekomendasi penggunaan disesuaikan dengan karakteristik dan kebutuhan aplikasi:

1. *Pusher* direkomendasikan untuk: Aplikasi yang memprioritaskan latensi rendah dan stabilitas tinggi, aplikasi dengan trafik fluktuatif yang memerlukan *auto-scaling*, aplikasi dengan *budget* untuk layanan *cloudbased*,

dan *startup* atau proyek yang ingin fokus pada pengembangan fitur tanpa mengurus kompleksitas infrastruktur *server*.

2. *Laravel Reverb* direkomendasikan untuk: Aplikasi dengan beban stabil dan dapat diprediksi, aplikasi yang memerlukan efisiensi memori dan *throughput* tinggi, aplikasi dengan *budget* terbatas dan tim teknis yang mampu mengelola *server*, serta aplikasi yang memerlukan kontrol penuh terhadap infrastruktur dan *data sovereignty*.
3. Strategi Optimisasi *Reverb*: Untuk implementasi *production*, *Laravel Reverb* memerlukan strategi tambahan seperti *load balancing* menggunakan Nginx atau HAProxy, *horizontal scaling* dengan multiple *instance*, optimisasi konfigurasi *server* (batas koneksi maksimum, parameter Redis, *PHPFPM*/*Octane*), dan implementasi *monitoring* aktif dengan mekanisme *auto-restart*.

Secara keseluruhan, penelitian ini membuktikan bahwa pemilihan teknologi *WebSocket* harus disesuaikan dengan prioritas performa, karakteristik beban aplikasi, ketersediaan tim teknis, dan *budget* yang tersedia. *Pusher* merupakan solusi *production-ready* untuk aplikasi yang memerlukan keandalan maksimal dan kemudahan pengelolaan, sementara *Laravel Reverb* menjadi alternatif *costeffective* untuk aplikasi dengan beban yang dapat diprediksi dan tim yang mampu melakukan optimisasi infrastruktur.

5.2 Saran

Berdasarkan hasil penelitian dan kesimpulan yang telah dipaparkan, terdapat beberapa saran untuk pengembangan lebih lanjut:

a. Implementasi FrankenPHP untuk Peningkatan Performa

Disarankan untuk mengimplementasikan FrankenPHP sebagai pengganti PHP-FPM tradisional pada *deployment production*. FrankenPHP merupakan *application server* modern yang dibangun menggunakan Caddy *web server* dan mendukung fitur-fitur seperti *worker mode*, *early hints*, dan HTTP/3. Dengan *worker mode*, aplikasi Laravel dapat tetap berada di memori sehingga mengurangi *overhead bootstrap* pada setiap *request*. Implementasi FrankenPHP diperkirakan dapat meningkatkan *throughput* hingga 3-4 kali lipat dibanding PHP-FPM, yang sangat bermanfaat untuk menangani koneksi bersamaan pada fitur *live streaming*. Selain itu, FrankenPHP juga mendukung HTTP/3 yang dapat meningkatkan performa koneksi WebSocket, terutama pada kondisi jaringan yang tidak stabil. Integrasi dengan Laravel Octane dan Laravel Reverb dapat memaksimalkan keunggulan arsitektur *event-driven* untuk menangani ribuan koneksi *real-time* secara bersamaan.

b. Pengujian Multiresolusi untuk *Streaming* Adaptif

Sistem saat ini menggunakan *auto bitrate adjustment* yang menyesuaikan kualitas *stream* berdasarkan kondisi *bandwidth*. Untuk meningkatkan kualitas pengalaman pengguna, disarankan melakukan pengujian performa pada berbagai resolusi *streaming* yang telah ditentukan yaitu 360p (640×360, *bitrate* ~500 Kbps), 720p (1280×720, *bitrate* ~2500 Kbps), dan 1080p (1920×1080,

bitrate ~5000 Kbps). Pengujian ini penting untuk memvalidasi bahwa sistem dapat menangani beban yang berbeda pada setiap tingkat resolusi dan memberikan data konkret mengenai *resource utilization* (CPU, memori, *bandwidth*) pada masing-masing resolusi. Hasil pengujian dapat digunakan sebagai acuan untuk implementasi *Adaptive Bitrate Streaming* (ABR) yang lebih terstruktur di masa mendatang, di mana sistem secara otomatis dapat memilih resolusi optimal berdasarkan kondisi jaringan pengguna. Pengujian pada setiap resolusi juga perlu mencakup skenario *stress testing* dengan jumlah *viewer* yang bervariasi untuk mengidentifikasi *bottleneck* pada setiap tingkat kualitas *streaming*.

c. Optimisasi Implementasi Laravel Reverb

Untuk meningkatkan performa Laravel Reverb dalam menangani lonjakan trafik mendadak, disarankan untuk mengimplementasikan arsitektur *horizontal scaling* dengan beberapa *instance* Reverb *server* yang di-*load balance* menggunakan Nginx atau HAProxy. Implementasi *connection pooling* dan optimisasi konfigurasi Redis sebagai *message broker* antarinstans dapat meningkatkan kemampuan sistem dalam menangani koneksi bersamaan. Selain itu, perlu dilakukan *fine-tuning* pada parameter PHP-FPM atau Laravel Octane untuk memaksimalkan penggunaan sumber daya server. Konfigurasi *worker process*, *max connections per worker*, dan *timeout settings* perlu disesuaikan berdasarkan karakteristik beban kerja *streaming real-time* yang memiliki pola koneksi *long-lived*.

d. Penelitian Lanjutan tentang Skalabilitas

Penelitian lebih lanjut perlu dilakukan untuk menguji skalabilitas kedua solusi pada skala yang lebih besar dengan simulasi ribuan hingga puluhan ribu koneksi bersamaan. Pengujian dalam kondisi jaringan yang berbeda (*latency*, *bandwidth*, *packet loss*) juga perlu dilakukan untuk memahami perilaku sistem dalam kondisi jaringan yang tidak ideal. Studi komparasi dengan solusi WebSocket lainnya seperti Socket.IO, Ably, atau Centrifugo dapat memberikan perspektif yang lebih komprehensif. Penelitian tentang *trade-off* antara biaya operasional, kompleksitas implementasi, dan performa pada berbagai skenario penggunaan juga dapat memberikan panduan bagi organisasi dalam memilih solusi yang paling sesuai dengan kebutuhan mereka.

e. Pengembangan Fitur Klasifikasi AI

Fitur klasifikasi AI dapat dikembangkan lebih lanjut dengan menambahkan kategori analisis seperti deteksi cuaca ekstrem, prediksi kondisi jalur berdasarkan *historical data*, dan integrasi dengan sensor IoT untuk data lingkungan yang lebih akurat. Implementasi model AI *on-device* menggunakan TensorFlow.js atau ONNX *Runtime* dapat mengurangi ketergantungan pada API eksternal dan meningkatkan kecepatan klasifikasi. Sistem juga dapat dikembangkan untuk memberikan notifikasi *real-time* kepada pendaki ketika terdeteksi kondisi berbahaya. Penggunaan model *lightweight* seperti MobileNet atau EfficientNet dapat mengoptimalkan performa klasifikasi pada perangkat dengan sumber daya terbatas. Selain itu, implementasi *feedback loop* untuk meningkatkan akurasi model berdasarkan data aktual dari lapangan dapat meningkatkan reliabilitas sistem peringatan dini.

f. Implementasi Fitur Tambahan

Pengembangan fitur-fitur tambahan seperti *recording* dan *playback* siaran untuk arsip, sistem moderasi *chat* berbasis AI untuk menyaring konten tidak pantas, fitur *multicamera streaming* dari berbagai sudut pandang, dan integrasi dengan sistem *booking* atau reservasi pos pendakian dapat meningkatkan nilai tambah platform. Implementasi fitur *offline mode* dengan *Progressive Web App* (PWA) juga dapat meningkatkan aksesibilitas bagi pendaki di area dengan koneksi internet terbatas. Fitur *collaborative streaming* yang memungkinkan beberapa *broadcaster* melakukan siaran bersama dalam satu *session* dapat menciptakan pengalaman yang lebih interaktif. Integrasi dengan platform media sosial untuk *cross-posting* dan *sharing* juga dapat meningkatkan jangkauan konten.

g. *Monitoring* dan *Analytics* yang Lebih Komprehensif

Implementasi sistem *monitoring real-time* yang lebih canggih menggunakan *tools* seperti Grafana, Prometheus, atau New Relic dapat memberikan visibilitas yang lebih baik terhadap performa sistem. *Dashboard analytics* yang menampilkan metrik seperti *viewer retention rate*, *peak concurrent viewers*, *average watch time*, dan *chat engagement* dapat membantu *broadcaster* memahami perilaku *viewer* dan meningkatkan kualitas siaran. *Error tracking* dan *alerting system* yang terintegrasi dengan Slack atau Discord juga penting untuk respons cepat terhadap insiden. Implementasi *distributed tracing* menggunakan *tools* seperti Jaeger atau Zipkin dapat membantu mengidentifikasi *bottleneck* pada arsitektur *microservices*. *Log aggregation* dengan ELK *Stack*

(Elasticsearch, Logstash, Kibana) dapat memudahkan *debugging* dan analisis pola *error*.

h. Optimisasi Biaya Operasional

Untuk implementasi dengan Pusher, perlu dilakukan analisis biaya operasional secara detail dan eksplorasi paket berlangganan yang sesuai dengan proyeksi pertumbuhan pengguna. Implementasi *caching strategy* yang efektif dapat mengurangi jumlah *message* yang dikirim dan menghemat biaya. Untuk implementasi dengan Reverb, optimisasi penggunaan sumber daya server dan implementasi *auto-scaling* infrastruktur menggunakan Kubernetes atau Docker Swarm dapat meningkatkan efisiensi biaya jangka panjang. Analisis *cost per concurrent connection* dan *cost per message* dapat membantu dalam membuat keputusan strategis tentang kapan harus beralih dari *managed service* ke *selfhosted solution*. Implementasi *resource pooling* dan optimisasi *database query* juga dapat mengurangi kebutuhan infrastruktur.

i. Peningkatan Keamanan Sistem

Implementasi mekanisme keamanan tambahan seperti *rate limiting* yang lebih ketat, *IP whitelisting* untuk *broadcaster*, enkripsi *end-to-end* untuk *chat messages*, dan sistem *authentication token* dengan *expiry time* yang lebih pendek perlu dipertimbangkan. *Audit log* untuk semua aktivitas penting dan implementasi *Web Application Firewall* (WAF) juga dapat meningkatkan keamanan sistem secara keseluruhan. *Penetration testing* berkala untuk mengidentifikasi kerentanan keamanan dan implementasi *security headers*

seperti *Content Security Policy* (CSP) dan *HTTP Strict Transport Security* (HSTS) dapat melindungi sistem dari serangan *common web vulnerabilities*. Implementasi *multi-factor authentication* (MFA) untuk akun *broadcaster* dan admin juga dapat meningkatkan keamanan akses.

j. Dokumentasi dan *Knowledge Transfer*

Dokumentasi teknis yang komprehensif mencakup arsitektur sistem, konfigurasi server, prosedur *deployment*, *troubleshooting guide*, dan *disaster recovery plan* perlu disusun dan dipelihara secara berkala. Program pelatihan untuk tim *support* dan *operations* tentang pengelolaan sistem *Live Cam* juga penting untuk memastikan keberlanjutan operasional platform. Dokumentasi API yang lengkap dengan contoh penggunaan dan *best practices* dapat memudahkan integrasi dengan sistem eksternal. Pembuatan *runbook* untuk *incident response* dan *standard operating procedures* (SOP) untuk *maintenance* rutin dapat meningkatkan efisiensi operasional.

k. Kolaborasi dengan Komunitas Pendaki

Melibatkan komunitas pendaki dalam pengembangan fitur melalui *user feedback* dan *beta testing* dapat memastikan bahwa platform memenuhi kebutuhan nyata pengguna. Program *content creator* untuk *broadcaster* yang aktif dan sistem reputasi untuk *viewer* yang berkontribusi positif dapat meningkatkan *engagement* dan kualitas konten pada platform. Implementasi forum diskusi atau *community board* dapat memfasilitasi *knowledge sharing* antarpengguna. *Partnership* dengan organisasi pendaki, taman nasional, dan komunitas pecinta alam dapat meningkatkan kredibilitas dan adopsi platform.

l. Riset Pengalaman Pengguna (*User Experience*)

Penelitian kualitatif dan kuantitatif tentang pengalaman pengguna (*User Experience*/UX) dalam menggunakan fitur *Live Cam* perlu dilakukan untuk mengidentifikasi area perbaikan dalam hal *usability*, aksesibilitas, dan kepuasan pengguna. *A/B testing* untuk berbagai elemen antarmuka dan analisis *user journey* dapat memberikan *insights* berharga untuk pengembangan produk. Implementasi *accessibility standards* seperti WCAG 2.1 untuk memastikan platform dapat digunakan oleh pengguna dengan disabilitas juga perlu dipertimbangkan. *Usability testing* dengan target pengguna yang beragam dapat mengidentifikasi *friction points* dalam *user flow* dan memberikan data empiris untuk *design decisions*.