BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan rangkaian penelitian dan implementasi yang dilakukan dalam tugas akhir berjudul "Implementasi Sistem Ekspedisi Berbasis Microservice Menggunakan Pendekatan *Domain Driven Design* dan *Clean Architecture*", diperoleh sejumlah temuan yang secara umum memberikan jawaban terhadap rumusan masalah sekaligus menggambarkan capaian dari proses perancangan dan pengembangan sistem.

1. Penelitian ini menunjukkan bahwa sistem ekspedisi yang kompleks dan dinamis dapat dirancang secara modular dan terstruktur menggunakan pendekatan arsitektur *microservice*. Layanan-layanan seperti *user*, *shipment*, *courier*, *locations*, dan *cargo* berhasil dipecah ke dalam *microservice* yang terpisah dan otonom, sehingga memungkinkan pengembangan dan pengelolaan sistem yang lebih fleksibel, terukur, dan terdistribusi. Pemisahan ini juga memungkinkan koordinasi yang lebih baik antar tim serta pengembangan berkelanjutan tanpa saling bergantung.
2. Penerapan *Domain-Driven Design* (DDD) terbukti efektif dalam memetakan domain logistik ke dalam *bounded context* yang jelas. Melalui proses eksplorasi domain dan identifikasi konteks, setiap layanan dikembangkan berdasarkan tanggung jawab bisnis yang spesifik. Hasilnya, struktur *microservice* tidak hanya terbagi berdasarkan teknis, tetapi juga mencerminkan pembagian tanggung jawab secara konseptual sesuai dengan pemahaman mendalam terhadap proses bisnis. Hal ini berhasil mengurangi overlap logika bisnis antar layanan dan meningkatkan konsistensi model domain.
3. Implementasi prinsip *Clean Architecture* pada masing-masing layanan memberikan struktur internal yang kuat dan terpisah dengan jelas antara lapisan input/output (*adapter*), *business logic* (*use case*), dan data access (*repository*). Pendekatan ini memberikan kemudahan dalam proses

pengujian, pemeliharaan, serta pengembangan lanjutan. Terbukti bahwa perubahan pada satu lapisan tidak menimbulkan gangguan pada lapisan lain, sehingga meningkatkan maintainability sistem secara keseluruhan.

4. Dari sisi komunikasi layanan, penelitian ini menghasilkan rancangan integrasi menggunakan kombinasi *REST API* dan *gRPC*, dengan pertimbangan efisiensi dan kebutuhan interaksi. *REST API* digunakan untuk komunikasi dengan layanan eksternal, sedangkan *gRPC* digunakan sebagai protokol utama antar *microservice* internal, yang terbukti memberikan kinerja tinggi dan latency rendah. Sementara itu, *API Gateway* berbasis *GraphQL* memainkan peran sentral sebagai titik akses utama *client* ke seluruh sistem. Selain melakukan autentikasi dan manajemen akses, *API Gateway* juga berhasil berfungsi sebagai *response aggregator*, menyederhanakan respons dari berbagai layanan menjadi satu kesatuan yang utuh bagi *client*, sehingga meningkatkan efisiensi integrasi.

Penelitian ini tidak ditujukan untuk menghasilkan sebuah produk logistik yang sepenuhnya siap digunakan secara komersial, melainkan difokuskan pada eksplorasi dan demonstrasi penerapan pendekatan arsitektural modern dalam konteks nyata sistem ekspedisi. Hasil yang diperoleh diharapkan dapat menjadi kontribusi awal yang bermanfaat bagi pengembang, peneliti, maupun praktisi yang ingin mempelajari dan mengadopsi konsep *Domain-Driven Design* dan *Clean Architecture* dalam pengembangan sistem berbasis microservice secara lebih terarah dan kontekstual.

5.2 Saran

Peneliti menyadari bahwa meskipun sistem yang dibangun dalam penelitian ini telah menunjukkan implementasi yang fungsional dan sesuai dengan pendekatan arsitektural yang ditetapkan, masih terdapat banyak ruang untuk pengembangan lebih lanjut. Pengembangan lanjutan ini tidak hanya bertujuan untuk menyempurnakan sistem dari sisi teknis, tetapi juga untuk memperluas penerapan arsitektur agar lebih relevan terhadap skenario dunia nyata, terutama dalam konteks sistem berskala besar dan berorientasi produksi. Oleh karena itu, beberapa saran yang dapat menjadi pertimbangan untuk penelitian atau pengembangan selanjutnya

adalah sebagai berikut:

1. Integrasi *observability tools* seperti *OpenTelemetry* dan *Jaeger* disarankan untuk meningkatkan kemampuan *monitoring*, *tracing*, dan analisis kinerja antar layanan *microservice*. Hal ini penting untuk memberikan visibilitas menyeluruh terhadap alur komunikasi dan memudahkan proses *debugging* serta deteksi *bottleneck* dalam sistem.
2. Menambahkan *instance* untuk masing-masing layanan *microservice* serta mengatur distribusinya menggunakan *load balancer*, sehingga sistem dapat lebih tangguh dan responsif dalam menangani beban yang lebih tinggi, sekaligus mendukung ketersediaan dan skalabilitas layanan.
3. Melakukan pengujian performa secara intensif dalam skenario beban tinggi, guna mengevaluasi ketahanan arsitektur yang diimplementasikan. Pengujian ini dapat membantu mengidentifikasi titik-titik kritis dalam sistem dan menjadi dasar dalam melakukan optimasi performa untuk keperluan produksi.