BAB IV
KESIMPULAN

4.1.Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan evaluasi arsitektur backend yang scalable dengan Express.js dan Prisma ORM pada platform Volunteerin.id yang telah diuraikan pada bab-bab sebelumnya, dapat ditarik beberapa kesimpulan sebagai berikut:

1. Perancangan Arsitektur Backend yang Scalable Arsitektur backend Volunteerin.id berhasil dirancang dengan menerapkan *layered architecture pattern* yang memisahkan komponen sistem menjadi lapisan *routes*, *middleware*, *controller*, *config*, dan *models*. Pemisahan tanggung jawab (*separation of concerns*) ini memungkinkan setiap lapisan dikembangkan secara independen dan memfasilitasi skalabilitas horizontal melalui penambahan instance server di belakang *load balancer*. Penerapan prinsip *stateless design* dengan JWT authentication memastikan setiap request bersifat mandiri tanpa ketergantungan pada *server-side session*, sehingga sistem siap untuk di-*scale* sesuai peningkatan beban pengguna.
2. Implementasi RESTful API dengan Express.js RESTful API Volunteerin.id berhasil diimplementasikan menggunakan Express.js dengan konsistensi penggunaan HTTP methods (GET, POST, PUT, DELETE), *resource-based URL naming*, dan response format JSON standar. Struktur routing modular dengan pemisahan endpoint berdasarkan *concern* (authentication, event management, partner management, admin management) memudahkan pengembangan dan pemeliharaan sistem. Implementasi *middleware chain* untuk autentikasi, otorisasi, validasi input, dan *error handling* terpusat memastikan setiap request melewati tahapan keamanan yang ketat sebelum mencapai *business logic*. Total 13 endpoint utama berhasil diimplementasikan untuk mendukung keseluruhan kebutuhan fungsional platform.

3. Penerapan Prisma ORM untuk Pengelolaan Basis Data Prisma ORM terbukti efektif dalam mengelola basis data PostgreSQL secara terstruktur dan *type-safe*. Definisi skema database melalui Prisma Schema Language yang deklaratif memudahkan pemodelan 8 entitas utama (User, PartnerProfile, Event, Form, FormResponse, Participant, Category, Notification) beserta relasi kompleks di antaranya (*one-to-one*, *one-to-many*, *many-to-many*). Sistem migrasi otomatis Prisma Migrate memungkinkan evolusi skema database secara bertahap tanpa *downtime* layanan. Prisma Client menghasilkan query yang aman dari SQL injection dan menyediakan *autocomplete* serta *type checking* yang mengurangi *runtime error* hingga meningkatkan produktivitas pengembangan.
4. Implementasi Autentikasi dan Otorisasi Multi-Role Mekanisme autentikasi berbasis JWT berhasil diimplementasikan untuk mendukung tiga peran pengguna (VOLUNTEER, PARTNER, ADMIN) dengan *Role-Based Access Control* (RBAC). Sistem keamanan mencakup *password hashing* dengan bcrypt (salt rounds 10), email verification dengan token expiry 30 menit, dan *access token* JWT dengan expiry 24 jam. Middleware autentikasi dan otorisasi (*isAuthenticate*, *isPartner*, *isAdmin*) memastikan setiap endpoint hanya dapat diakses oleh role yang berwenang. Fitur *forgot password* dan *reset password* dengan JWT token memberikan mekanisme pemulihan akun yang aman. Evaluasi fungsionalitas menunjukkan *success rate* 100% untuk semua *happy path scenarios* dan *edge cases* seperti *unauthorized access* ter-*handle* dengan baik.
5. Evaluasi Kinerja Backend Evaluasi performa melalui *load testing* dengan tool k6 menunjukkan bahwa backend Volunteerin.id mampu menangani hingga 100 *concurrent users* dengan rata-rata *response time* di bawah 300ms untuk mayoritas endpoint, memenuhi target *non-functional requirement* yang ditetapkan. P95 (95% request) diselesaikan dalam waktu kurang dari 550ms. *Error rate* sangat rendah (< 0.5%), dengan error yang terjadi mayoritas disebabkan oleh validasi bisnis (*expected errors*) seperti quota penuh atau duplikasi pendaftaran, bukan error sistem. Prisma *connection*

pooling terbukti efektif dalam menangani *concurrent database queries* tanpa menimbulkan "*too many connections*" error. Sistem mencapai *bottleneck* pada 200 *concurrent users* dengan CPU utilization 89%, mengindikasikan bahwa untuk *production deployment* dengan *expected load* lebih tinggi diperlukan deployment multi-instance dengan PM2 cluster mode atau Kubernetes horizontal pod autoscaling.

Secara keseluruhan, penelitian ini berhasil membuktikan bahwa arsitektur backend yang scalable dapat dirancang dan diimplementasikan dengan efektif menggunakan Express.js dan Prisma ORM untuk mendukung platform manajemen relawan digital. Kombinasi teknologi Node.js, Express.js, Prisma ORM, dan PostgreSQL memberikan fondasi yang kuat untuk sistem yang dapat berkembang seiring peningkatan jumlah pengguna dan kompleksitas fitur.

4.2.Saran

Berdasarkan hasil evaluasi dan keterbatasan yang ditemukan selama penelitian, beberapa saran untuk pengembangan lebih lanjut adalah sebagai berikut:

1. Implementasi Caching Mechanism Untuk meningkatkan performa dan mengurangi beban database, disarankan mengimplementasikan Redis sebagai *caching layer* untuk data yang jarang berubah seperti kategori event, benefit, dan list event populer. *Cache-aside pattern* dengan TTL (Time-To-Live) 5-10 menit dapat mengurangi *database queries* hingga 60-70% untuk endpoint yang sering diakses. Implementasi cache invalidation strategy juga perlu dipertimbangkan untuk menjaga konsistensi data saat terjadi update.
2. Penambahan Rate Limiting dan Security Enhancement Implementasi *rate limiting middleware* (misalnya menggunakan express-rate-limit) sangat direkomendasikan untuk mencegah abuse API dan serangan DDoS. Batasan yang disarankan adalah 100 request per 15 menit untuk public endpoints dan 1000 request per jam untuk authenticated endpoints. Selain itu, penambahan *security headers* tambahan, implementasi HTTPS enforcement, dan periodic security audit dengan tools seperti OWASP ZAP dapat meningkatkan keamanan sistem secara keseluruhan.

3. Optimasi Location-Based Search dengan PostGIS Untuk meningkatkan performa pencarian event berbasis lokasi pada dataset besar (10,000+ events), disarankan mengimplementasikan PostgreSQL PostGIS extension yang menyediakan native geospatial indexing dan queries. Alternatif lain adalah menggunakan Elasticsearch dengan geo-queries untuk offload calculation ke database layer. Pendekatan ini dapat mengurangi response time location-based search hingga 80-90% dibandingkan perhitungan Haversine di application layer.
4. Implementasi Automated Testing dan CI/CD Pipeline Untuk meningkatkan kualitas kode dan mempercepat development cycle, sangat disarankan mengimplementasikan automated testing dengan Jest untuk unit test dan Supertest untuk integration test. Target code coverage minimal 70-80% untuk critical business logic. Selain itu, setup CI/CD pipeline dengan GitHub Actions atau GitLab CI untuk automated testing, code linting, security scanning, dan deployment dapat mengurangi human error dan mempercepat release cycle.
5. Centralized Logging dan Monitoring System Untuk production deployment, implementasi centralized logging dengan ELK Stack (Elasticsearch, Logstash, Kibana) atau alternatif seperti Loki + Grafana sangat penting untuk memudahkan debugging dan troubleshooting pada distributed system. Penambahan application performance monitoring (APM) dengan tools seperti New Relic, Datadog, atau Prometheus + Grafana dapat memberikan visibility real-time terhadap performa sistem, bottleneck, dan error rate.
6. Database Optimization dan Read Replicas Untuk mendukung skalabilitas lebih tinggi, disarankan mengimplementasikan PostgreSQL read replicas untuk memisahkan read queries dari write queries. Pendekatan ini dapat meningkatkan throughput database hingga 3-5x lipat. Selain itu, penambahan database indexing strategy yang lebih komprehensif (indexing pada kolom yang sering di-query seperti event.startAt, event.endAt,

user.email) dan periodic query optimization dapat meningkatkan performa database secara signifikan.

7. Real-Time Notification dengan WebSocket Untuk meningkatkan user experience, disarankan mengganti polling-based notification dengan real-time push notification menggunakan WebSocket (Socket.io atau Server-Sent Events). Pendekatan ini lebih efisien dalam penggunaan bandwidth dan memberikan notifikasi instan kepada user tanpa perlu client melakukan request berkala. Implementasi dapat dimulai dari fitur notification untuk status approval pendaftar event yang paling membutuhkan real-time update.
8. API Documentation dengan Swagger/OpenAPI Untuk memudahkan kolaborasi dengan frontend developer dan external API consumers, sangat disarankan membuat comprehensive API documentation menggunakan Swagger/OpenAPI specification. Tools seperti swagger-jsdoc dapat generate documentation langsung dari code comments, atau swagger-ui-express untuk interactive API documentation yang dapat langsung di-testing dari browser. Documentation yang baik mengurangi miscommunication dan mempercepat integration process.
9. Microservices Architecture untuk Scale Ekstrem Untuk future development dengan expected scale yang sangat besar (100,000+ concurrent users), dapat dipertimbangkan evolusi arsitektur dari monolithic ke microservices. Misalnya, memisahkan event management service, authentication service, dan notification service menjadi independent services dengan API gateway (Kong, AWS API Gateway) di depan. Pendekatan ini memungkinkan setiap service di-scale secara independen sesuai beban masing-masing dan meningkatkan fault tolerance sistem.

Saran-saran di atas dapat diimplementasikan secara bertahap sesuai prioritas dan kebutuhan production deployment. Implementasi seluruh saran akan menghasilkan sistem backend yang tidak hanya scalable, tetapi juga maintainable, secure, dan production-ready untuk melayani ribuan hingga ratusan ribu pengguna secara bersamaan.