Implementasi NestJS dan Prisma pada pengembangan Backend Monolitik pada Aplikasi Web Antria

Proposal Tugas Akhir

Kelas MK Penulisan Proposal (CII4A2)

1302204044
Muhammad Rovino Sanjaya

Program Studi Sarjana Rekayasa Perangkat Lunak

Fakultas Informatika

Universitas Telkom

Bandung

2024

Lembar Persetujuan

Implementasi NestJS dan Prisma pada pengembangan Backend Monolitik pada Aplikasi Web Antria

Implementation of Monolithic Backend Development with NestJS and Prisma for Antria's Web Application

NIM: 1302204044
Muhammad Rovino Sanjaya

Proposal ini diajukan sebagai usulan pembuatan tugas akhir pada
Program Studi Sarjana Rekayasa Perangkat Lunak
Fakultas Informatika Universitas Telkom

Bandung, 9 Desember 2024
Menyetujui

Calon Pembimbing 1	Calon Pembimbing 2
(Dr. Mira Kania Sabariah, S.T., M.T.)	(Monterico Adrian, S.T., M.T.)
NIP: 14770011	NIP: 20870024

Abstrak

Populasi penduduk yang tinggi di Indonesia mengakibatkan antrian panjang dalam berbagai pelayanan publik atau pelayanan konsumen. Pelanggan harus datang ke tempat untuk mengambil antrian dan menunggu gilirannya, semakin panjang antrian, semakin lama waktu tunggu yang dibutuhkan. Hal ini tidak efisien dan menyebabkan banyak waktu terbuang hanya untuk menunggu antrian. Jika nomor antrian bisa didapatkan secara online dan dapat dipantau secara online, maka dapat mengurangi waktu yang terbuang. Dengan membuat aplikasi antrian virtual, diharapkan dapat memperpendek antrian secara fisik, dengan cara antri secara virtual, melihat antrian yang sedang berjalan, dan booking tempat. Pada Pengembangan aplikasi ini, framework yang digunakan adalah NestJS dan PrismaJS dengan menerapkan RESTful API, Object Relational Mapping, dan menghindari Anti-Pattern. Framework NestJS mendukung pembuatan aplikasi ber-arsitektur monolitik dan *microservice*. Setelah aplikasi dibangun di arsitektur monolitik, aplikasi dapat dengan mudah dimigrasikan ke *microservice* saat penggunaan aplikasi sudah hampir mendekati batas muat pengguna.

Kata Kunci: NestJS, PrismaJS, Anti-Pattern, Antrian, Backend, REST

Daftar Isi

Bab I

Pendahuluan

1.1 Latar Belakang

Meningkatnya populasi di indonesia mengakibatkan banyak pelanggan yang mengantri untuk mendapatkan layanan di bank, restoran, rumah sakit, dan tempat penyedia jasa lainnya. Mengantri merupakan kegiatan yang membosankan dan menguras waktu. Panjangnya antrian juga mampu berdampak pada mutu pelayanan di suatu tempat. Pelanggan yang harus menunggu lama berpotensi beralih ke pesaing, atau jika ada urusan lain yang lebih penting, maka pelanggan akan keluar dari tempat antrian, meninggalkan antriannya [8][4][17]. Solusi yang ada pada bank, kantor pos, dan rumah sakit saat ini menggunakan *ticketting* nomor antrian secara manual, dimana antrian yang sedang dilayani ditampilkan di layar pada ruang tunggu. Hal ini kurang efektif karena pelanggan harus berada di ruang tunggu[4].

Perkembangan teknologi yang cepat mengakibatkan penggunaan perangkat pintar atau *smartphone* merupakan hal lumrah, banyak bermunculan aplikasi antrian virtual seperti Antrique, Qiwee, ExaQue dimana pengguna dapat mengantri dari jarak jauh melalui aplikasi maupun *website*. Para pengguna aplikasi tersebut dapat melakukan hal lain saat mengantri sebelum gilirannya. Namun, aplikasi-aplikasi tersebut memiliki banyak kelemahan seperti *UI/UX* yang tidak bagus, sering *crash* dan *freeze*, tidak ada estimasi waktu antrian, dan masih belum ada yang berfokus ke sektor *food and beverage*.

Oleh karena itu, perlunya dikembangkan sebuah aplikasi yang memiliki fitur yang sama atau lebih dengan menutup kekurangan pada aplikasi tersebut. Pengembangan aplikasi yang direncanakan menggunakan arsitektur monolitik karena mudahnya untuk dibuat dan di-*deploy* secara cepat untuk di iterasikan. Namun, arsitektur monolitik memiliki kelemahan seperti sulitnya untuk di-*maintenance*, *scale*, dan reliabilitas nya. Oleh karena itu, perlu diperhatikan bagaimana cakupan aplikasi kedepannya dan perlunya migrasi ke arsitektur *microservice* [6] [7].

Dalam pengembangan aplikasi web, pemilihan bahasa pemrograman untuk digunakan di *backend* sangatlah penting karena dapat mempengaruhi performa aplikasi yang dibangun. Dalam pemilihan bahasa pemrograman *backend*,

banyak pilihan yang tersedia seperti PHP, Python, Ruby, PERL, dan banyak lagi. NodeJs merupakan *tools* yang memungkinkan bahasa JavaScript dapat dijalankan pada sisi *backend*. Dalam sisi performa, NodeJs lebih unggul dibanding PHP dan Python dalam sisi kecepatan melayani *request* dari client [18] [11].

NestJs merupakan *framework backend* dari Nodejs yang menggunakan bahasa typescript, dan bisa digunakan untuk pengembangan arsitektur berbasis *microservice* dan monolitik, jadi jika aplikasi dikembangkan pada arsitektur monolitik dapat dengan mudah dimigrasikan ke *microservice*. NestJs juga bisa digunakan bersamaan dengan *framework* PrismaJs untuk mengelola database [10]. PrismaJs merupakan *framework* Object Relational Mapping (ORM) [13], digunakan untuk mempercepat, dan mempermudah pengembangan aplikasi yang database-nya memiliki relasi yang kompleks dan sulit di-*maintenance* jika menggunakan Structured Query Language (SQL) [19].

Implementasi Application Programming Interface (API) yang digunakan adalah Representational State Transfer (RESTful) API, RESTful API adalah arsitektur untuk mempermudah komunikasi client-server agar efektif untuk transaksi data. Namun, pada implementasi RESTful API, ada beberapa hal yang perlu diperhatikan seperti keamanan saat transaksi atau komunikasi [3]. Keamanan yang lemah dapat mengakibatkan hacker dapat dengan mudah melakukan *request tampering*, mengambil data pengguna, dan dapat membocorkan data keuangan mitra. *Design pattern* juga perlu diperhatikan dalam penggunaan bahasa untuk API *endpoint* nya agar tidak terjadi *anti pattern*. *Anti pattern* terjadi saat penamaan API tidak sesuai dengan fungsi, atau ada fungsi sejenis tapi penamaannya berbeda jauh. Dengan menghindari *anti pattern*, dapat berakibat ke aplikasi yang lebih mudah di-*sustain* dan di-*maintain* [1] [2].

Berdasarkan uraian di atas, penelitian ini akan membuat sebuah *backend* aplikasi antrian dengan menggunakan arsitektur monolitik dengan *framework* NestJs dan PrismaJs sebagai *framework* nya. Setelah fitur-fitur aplikasi dibuat, perlu dilakukan unit testing untuk memvalidasi kodingan yang telah ditulis. Hal ini bertujuan untuk meminimalisir bug dan mencegah terjadinya regresi saat fitur baru ditambahkan [15].

1.2 Perumusan Masalah

Aplikasi antria memerlukan backend developer untuk mengimplementasikan fungsi fungsi API dan manajemen database nya. Maka dapat dirumuskan permasalahan sebagai berikut:

1. Bagaimana meningkatkan *sustainability* dan *maintainability* pada penggunaan manajemen database.

2. Bagaimana merancang API yang bebas dari *anti pattern*.

3. Bagaimana merancang sistem keamanan pada API untuk melayani *request*.

1.3 Tujuan

Tujuan dari pengerjaan Tugas Akhir ini yaitu:

1. Mengimplementasikan Prisma ORM untuk meningkatkan *sustainability* dan *maintainability* pada manajemen database.

2. Membuat dokumentasi API yang dapat dengan mudah di *sustain* dan di *maintain*.

3. Mengamankan data pengguna dengan menambahkan *anti request tampering* pada setiap header request.

1.4 Batasan Masalah

1. Hanya berfokus kepada implementasi database menggunakan Prisma ORM.

2. Berfokus ke bagaimana membuat *endpoint* API yang tidak menimbulkan *anti pattern*.

3. Implementasi keamanan pada saat penanganan *request* menggunakan JSON Web Token (JWT).

1.5 Rencana Kegiatan

Rencana kegiatan yang akan dilakukan adalah sebagai berikut : Studi Literatur, Pengumpulan data, Perancangan database dengan Prisma, Implementasi RESTful API pada NestJS, Analisa hasil unit testing, dan Penulisan lapoaran.

1.6 Jadwal Kegiatan

Tabel 1.1: Jadwal kegiatan proposal tugas akhir

No	**Kegiatan**	**Bulan ke-**																					
		1				**2**				**3**				**4**				**5**				**6**			
1	Studi Literatur																						
2	Pengumpulan Data																						
3	Perancangan database dengan prisma																						
4	Implementasi RESTful API pada NestJS																						
5	Analisa hasil unit testing																						
6	Penulisan Laporan																						

Bab II

Kajian Pustaka

2.1 NodeJs

NodeJs adalah *runtime* javascript yang basisnya dibangun dari V8 JavaScript Engine. NodeJs berjalan dalam bentuk *event-driven*, dan menggunakan model *non blocking I/O*. meskipun menggunakan *event-driven* untuk melayani *request*, NodeJs dapat melayani jutaan koneksi dalam waktu bersamaan secara *asynchronous* [16].

2.2 NestJs

NestJs merupakan *framework* untuk Nodejs yang dikembangkan oleh Kamil Myśliwiec yang bertujuan untuk membuat aplikasi NodeJs yang efektif dan *scalable*. NestJs mendukung penggunaan bahasa typescript dan javascript. NestJs juga menggabungkan komponen-komponen dari Functional Programming, Object Oriented Programming, dan Functional Reactive Programming [12] [10].

2.3 Object Relational Mapping

Object Relational Mapping (ORM) adalah sebuah teknologi yang memetakan tabel database ke dalam objek, biasanya dipakai dalam bahasa yang berbasis Object Oriented Programming. Dengan menggunakan ORM, developer dapat berfokus ke *business logic* tanpa mengkhawatirkan penggunaan akses database yang rumit [9].

2.4 PrismaJs

PrismaJs adalah ORM Open Source, biasanya digunakan sebagai alternatif dari menggunakan Structured Query Language (SQL) secara langsung. PrismaJs mendukung penggunaan database MySQL, PostgreSQL, SQLite, SQL Server, CockroachDB, dan MongoDB. PrismaJs digunakan untuk mempermudah pengembangan database yang memiliki relasi yang kompleks dan besar, dengan cara memberikan API yang *type-safe* untuk *query* database nya dan mengembalikan hasil *query* dalam bentuk JavaScript Object Notation (JSON) [13].

2.5 Arsitektur Monolitik

Arsitektur Monolitik adalah arsitektur sebuah *software* dimana beberapa fungsi komponen yang berbeda seperti fungsi otorisasi, *business logic*, notifikasi, dan pembayaran. Semua fungsi tersebut berada dalam satu program dan *platform* yang sama. Arsitekru monolitik mudah untuk dikembangkan dan di-*deploy*. Namun, sulit untuk di-*maintenance* dan di-*scale* [6].

2.6 JSON Web Token

JSON Web Token (JWT) adalah sebuah token berbentuk *string* json yang dapat digunakan untuk melakukan otorisasi. Ukuran JWT tergolong kecil jadi dapat dengan cepat ditransfer antar client dan server. JWT menggunakan algoritma HMAC atau RSA untuk mengenkripsi *digital signature* yang digunakan. JWT memiliki 3 bagian pada *string* nya yang dipisahkan menggunakan ".", bagian ini berupa *header*, *payload*, dan *signature* [14].

2.7 Anti Pattern

Anti Pattern terjadi jika pembuatan nama sebuah objek tidak konsisten dengan yang lain. Objek disini dapat berupa *endpoint* API, nama *variable*, nama fungsi, dan nama lain yang penggunaannya bersifat publik. Terjadinya *anti pattern* dapat mengakibatkan sulitnya untuk memahami suatu dokumentasi dan kodingan aplikasi [1] [2].

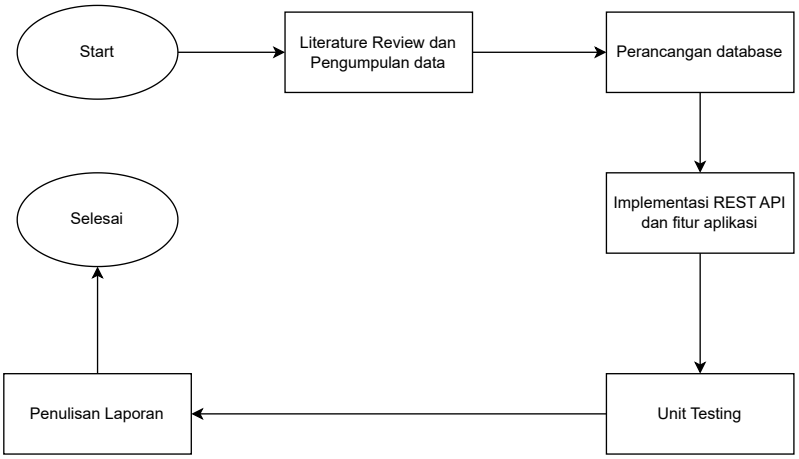
2.8 RESTful API

Representational State Transfer (RESTful) Application Programming Interface (API) adalah arsitektur untuk mempermudah komunikasi client-server agar efektif untuk transaksi data. Tipe data yang paling sering digunakan untuk transaksi client server adalah JSON. Karakteristik RESTful meliputi : Client-Server, Stateless, Layered Architecture, Caching, Code on Demand, dan Uniform Interface [5].

Bab III

Perancangan Sistem

3.1 Alur Perancangan



Gambar 3.1: Alur Perancangan

Gambar 3.1 menunjukkan alur perancangan sistem *backend* yang dibuat. Terdapat 5 tahapan dalam perancangan sistem yaitu :

3.1.1 *Literature Review* dan Pengumpulan data

Pada tahap ini, dilakukan pengumpulan data melalui Software Requirement Specification (SRS) yang telah dibuat oleh System Analyst (SA) serta membaca literatur ilmiah mengenai pengembangan *backend* web.

3.1.2 Perancangan database

Pada tahap ini dilakukan perancangan dan implementasi ERD menggunakan PrismaJs sebagai *framework* Object Relational Mapping.

3.1.3 Implementasi REST API dan fitur aplikasi

Pada tahap ini dilakukan sesi koding untuk mengimplementasikan berbagai macam fitur aplikasi berdasarkan SRS yang telah dibuat dan membuat API

endpoint yang menghindari *anti pattern*.

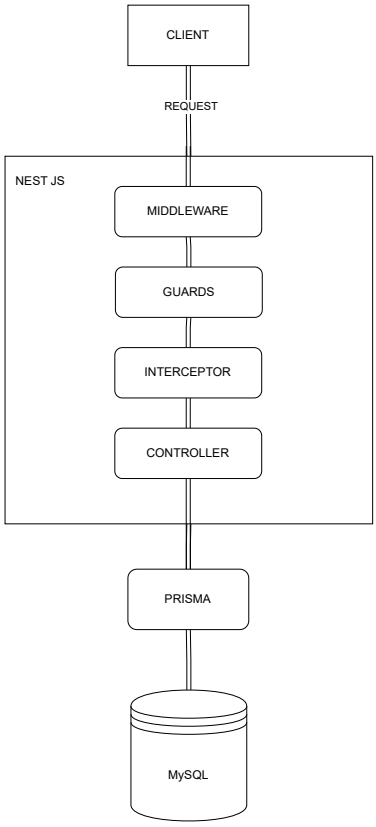
3.1.4 Unit Testing

Pada tahap unit testing, dilakukan pengujian logic dan fungsi API, sampai tidak terdapat error lagi, maka dilanjut ke penulisan proposal.

3.1.5 Penulisan laporan

Pada tahap ini, akan disusun semua tahapan pekerjaan, hasil analisis dan pembahasan terhadap semua yang telah dilakukan, diamati, dan dihasilkan dalam membuat dan mengimplementasikan *backend* aplikasi.

3.2 Desain Sistem



Gambar 3.2: Desain Sistem

Implementasi dan perancangan RESTful API, Database, dan fungsi bisnis dikembangkan menggunakan *framework* NestJs. Pada NestJs terdapat beberapa komponen seperti: Middleware, Guards, Interceptor, Controller, dan Service. Service yang dipakai adalah PrismaJs untuk menghubungkan NestJs ke *Database System*. Gambar 3.2 merupakan *Request Lifecycle* yang men-

jelaskan management API atau bagaimana alur *request* ditangani dari awal sampai akhir.

3.2.1 Middleware

Pada Middleware, fungsi akan dipanggil sebelum masuk ke routing. fungsi middleware dapat mengakses data *request* dan *response*. beberapa fungsi Middleware seperti *logger*, dan cek notifikasi.

3.2.2 Guard

Pada Guard, *request* akan dicek *authenticity*, untuk mengetahui validitas dari *request* tersebut. Tahap ini juga akan dicek keamanan session menggunakan JWT dan CSRF.

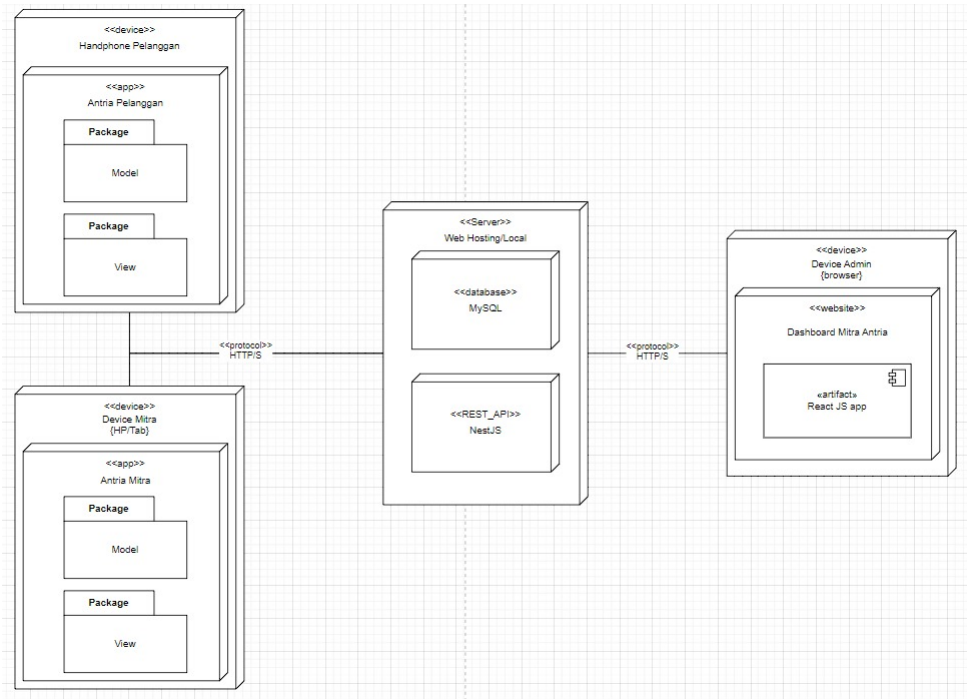
3.2.3 Interceptor

Setelah melalui Guard, Request akan masuk ke Interceptor. Dimana jika suatu *request* mempunyai suatu karakteristik yang ditentukan, maka akan menjalankan fungsi tambahan. Interceptor terjadi ketika *request* datang *(pre)*, dan response *(post)*.

3.2.4 Controller

Setelah melewati Interceptor, fungsi di Controller akan dijalankan. Jika pada Controller tersebut perlu data dari database maka akan turun ke service PrismaJs untuk melakukan database call menggunakan ORM [10].

3.3 Deployment



Gambar 3.3: Deployment Diagram

Pada diagram 3.3 menjelaskan bagaimana server backend berkomunikasi dengan Aplikasi lain melalui protokol HTTPS.

Daftar Pustaka

[1] Aghajani, E., Nagy, C., Bavota, G., and Lanza, M. A large-scale empirical study on linguistic antipatterns affecting apis. In *Soft System Stakeholder Analysis Methodology* (11 2018), Institute of Electrical and Electronics Engineers Inc., pp. 25–35.

[2] Alshraiedeh, F. S., and Katuk, N. A uri parsing technique and algorithm for anti-pattern detection in restful web services. *International Journal of Web Information Systems 17* (1 2021), 1–17.

[3] Beer, M. I., and Hassan, M. F. Adaptive security architecture for protecting restful web services in enterprise computing environment. *Service Oriented Computing and Applications 12* (6 2018), 111–121.

[4] Ghazal, M., Hamouda, R., and Ali, S. A smart mobile system for the real-time tracking and management of service queues. *International Journal of Computing and Digital Systems 5* (7 2016), 305–313.

[5] Giessler, P., Gebhart, M., Sarancin, D., Steinegger, R., and Abeck, S. Best practices for the design of restful web services. In *International Conferences of Software Advances (ICSEA)* (2015), pp. 392–397.

[6] Gos, K., and Zabierowski, W. The comparison of microservice and monolithic architecture. In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)* (2020), IEEE, pp. 150–153.

[7] Jatkiewicz, P., and Okrój, S. Differences in performance, scalability, and cost of using microservice and monolithic architecture. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing* (2023), pp. 1038–1041.

[8] Khong, Y. L., Ooi, B. C., Tan, K. E., Ibrahim, S. A. B., and Tee, P. L. E-queue mobile application. In *SHS Web of Conferences* (2017), vol. 33, EDP Sciences, p. 00033.

[9] Lorenz, M., Rudolph, J.-P., Hesse, G., Uflacker, M., and Plattner, H. Object-relational mapping revisited-a quantitative study on the impact of database technology on o/r mapping strategies. *Proceedings of the 50th Hawaii International Conference on System Sciences* (2017).

[10] Mysliwiec, K. Nestjs documentation.

[11] Odeniran, Q., Wimmer, H., and Rebman, C. M. Node.js or php? determining the better website server backend scripting language. *Issues in Information Systems 24* (2023), 328–341.

[12] Pham, A. D. Developing back-end of a web application with nestjs framework: Case: Integrify oy's student management system. *Theseus* (2020).

[13] Prisma Data, I. Prismajs documentation.

[14] Rahmatullo, A., Aldya, A. P., and Arifin, M. N. Stateless authentication with json web tokens using rsa-512 algorithm. *JURNAL INFOTEL 11*, 2 (2019), 36–42.

[15] Runeson, P. A survey of unit testing practices. *IEEE software 23*, 4 (2006), 22–29.

[16] Shah, H., and Soomro, T. R. Node. js challenges in implementation. *Global Journal of Computer Science and Technology 17*, 2 (2017), 73–83.

[17] Uddin, M. N., Rashid, M., Mostafa, M., Salam, S., Nithe, N., and Ahmed, S. Z. Automated queue management system, 2016.

[18] William, G., Anthony, R., and Purnama, J. Development of node-js based backend system with multiple storefronts for batik online store. *ACM International Conference Proceeding Series* (2020). Cited by: 0.

[19] Zmaranda, D., Pop-Fele, L.-L., Győrödi, C., Győrödi, R., and Pecherle, G. Performance comparison of crud methods using net object relational mappers: A case study, 2020.

Lampiran