KERJA PRAKTIK - EF234603

ANALISIS ARSITEKTUR DAN ALUR SISTEM BACKEND PADA APLIKASI TRAVEL DIVER "BLUEMEET"

PT. KERTAS DIGITAL INDONESIA

Periode: 3 Februari 2025 - 27 Juni 2025

Oleh:

Aloysius deDeo Darmo Susanto 5025221174

Pembimbing Jurusan

Moch. Nafkhan Alzamzami, S.T, M.T.

Pembimbing Lapangan

Michelle Agatha

Depertemen Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya 2025

KERJA PRAKTIK – EF234603

ANALISIS ARSITEKTUR DAN ALUR SISTEM BACKEND PADA APLIKASI TRAVEL DIVER “BLUEMEET”

PT. KERTAS DIGITAL INDONESIA

Periode: 3 Februari 2025 - 27 Juni 2025

Oleh:

Aloysius deDeo Darmo Susanto 5025221174

Pembimbing Jurusan

Moch. Nafkhan Alzamzami, S.T, M.T.

Pembimbing Lapangan

Michelle Agatha

Departemen Informatika

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya 2025

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

LEMBAR PENGESAHAN

KERJA PRAKTIK

ANALISIS ARSITEKTUR DAN ALUR SISTEM BACKEND PADA APLIKASI TRAVEL DIVER "BLUEMEET"

Disusun Oleh:

Aloysius deDeo Darmo Susanto 5025221174

Disetujui oleh Pembimbing Kerja Praktik:

1. Moch. Nafkhan Alzamzami, S.T, M.T.

NIP 199911222024061001

(Dosen Pembimbing Departemen)

2. Michelle Agatha

(Pembimbing Lapangan)

ABSTRAK

Abstrak

PT Kertas Digital Indonesia merupakan perusahaan yang bergerak di bidang teknologi digital dan telah mengembangkan aplikasi "Bluemeet" untuk memfasilitasi para penyelam dalam merencanakan perjalanan. Kerja praktik ini bertujuan untuk menganalisis arsitektur, teknologi, dan alur data pada sisi *backend* aplikasi Bluemeet. Metode yang digunakan adalah observasi dan analisis sistem yang sudah ada (*existing system*). Analisis mencakup identifikasi teknologi yang digunakan, pemodelan arsitektur sistem, analisis alur data untuk fitur-fitur utama seperti registrasi pengguna, pemesanan trip, dan manajemen profil. Hasil dari analisis ini berupa dokumentasi arsitektur sistem, evaluasi terhadap potensi pengembangan, serta identifikasi kemungkinan adanya *bottleneck* atau celah perbaikan pada sistem *backend*. Laporan ini diharapkan dapat memberikan gambaran utuh mengenai sistem *backend* Bluemeet dan menjadi masukan yang berharga bagi perusahaan.

Kata kunci: *Backend*, Analisis Sistem, Arsitektur Perangkat Lunak, Aplikasi Travel, Bluemeet.

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan kegiatan Kerja Praktik di PT Kertas Digital Indonesia serta menyusun laporan ini dengan baik.

Laporan ini disusun berdasarkan hasil analisis dan observasi yang dilakukan pada aplikasi "Bluemeet", sebuah platform travel untuk komunitas penyelam di Indonesia. Meskipun dalam pelaksanaannya penulis tidak terlibat langsung dalam proses pengembangan, penulis mendapatkan kesempatan untuk menganalisis sistem dari sisi *backend* yang sudah berjalan.

Penulis ingin mengucapkan terima kasih kepada semua pihak yang telah membantu, terutama kepada:

1. Kedua orang tua penulis.
2. Bapak Moch. Nafkhan Alzamzami, S.T, M.T., selaku Dosen Pembimbing.
3. Ibu Michelle Agatha selaku Pembimbing Lapangan di PT Kertas Digital Indonesia.
4. Seluruh tim PT Kertas Digital Indonesia yang telah memberikan kesempatan.

Penulis menyadari laporan ini masih jauh dari sempurna. Oleh karena itu, kritik dan saran yang membangun sangat diharapkan.

Surabaya, 4 Juli 2025

Aloysius deDeo Darmo Susanto

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi digital telah mengubah berbagai sektor industri di Indonesia, tidak terkecuali sektor pariwisata. Sebagai negara maritim dengan kekayaan bawah laut yang luar biasa, Indonesia memiliki potensi besar dalam wisata selam (diving). Besarnya komunitas penyelam dan penyedia jasa wisata selam menciptakan kebutuhan akan sebuah platform digital yang mampu menghubungkan keduanya secara efisien, aman, dan terpercaya.

Menjawab tantangan tersebut, PT Kertas Digital Indonesia, sebuah perusahaan yang berfokus pada pengembangan solusi digital, menciptakan aplikasi "Bluemeet". Aplikasi ini dirancang sebagai ekosistem digital bagi para penyelam di seluruh Indonesia, memfasilitasi pencarian destinasi, pemesanan paket perjalanan (*trip*), hingga interaksi antar anggota komunitas.

Keberhasilan sebuah aplikasi modern seperti Bluemeet sangat bergantung pada keandalan dan skalabilitas sistem *backend*-nya. Sistem *backend* bertanggung jawab atas pengelolaan data pengguna, transaksi, jadwal perjalanan, serta logika bisnis inti lainnya. Arsitektur *backend* yang baik akan memastikan aplikasi dapat berjalan dengan cepat, aman, dan mampu menangani pertumbuhan jumlah pengguna di masa depan.

Dalam rangka memenuhi salah satu mata kuliah wajib program studi, penulis mendapatkan kesempatan untuk melaksanakan kerja praktik di PT Kertas Digital Indonesia. Pada kesempatan ini, penulis berfokus pada analisis arsitektur dan teknologi yang digunakan pada sisi *backend* aplikasi Bluemeet. Meskipun tidak terlibat langsung dalam penulisan kode, kegiatan analisis sistem yang sudah berjalan ini memberikan kesempatan berharga untuk membedah dan memahami implementasi arsitektur perangkat lunak modern (NestJS) dalam kasus nyata di dunia industri.

1.2 Rumusan Masalah

Berdasarkan latar belakang, maka rumusan masalahnya adalah:

1. Bagaimana arsitektur sistem *backend* yang diterapkan pada aplikasi Bluemeet?
2. Teknologi apa saja yang digunakan dalam pengembangan sisi *backend* aplikasi Bluemeet?
3. Bagaimana alur data pada fitur-fitur kunci seperti registrasi pengguna, pencarian trip, dan proses pemesanan?
4. Apa saja potensi pengembangan atau perbaikan yang dapat diidentifikasi dari analisis sistem *backend* yang ada?

1.3 Tujuan Kerja Praktik

Tujuan dari kerja praktik ini adalah untuk menyelesaikan kewajiban kuliah kerja praktik di Institut Teknologi Sepuluh Nopember dengan bobot empat SKS. Selain itu, tujuan lainnya adalah untuk menganalisis cara kerja dan sistematika sistem backend pada aplikasi Bluemeet.

1.4 Manfaat Kerja Praktik

Bagi mahasiswa, kerja praktik ini memberikan manfaat esensial berupa pengalaman praktis di dunia industri, kesempatan untuk menerapkan pengetahuan teoretis ke dalam studi kasus

nyata, serta mendapatkan wawasan mendalam mengenai arsitektur *backend* modern yang relevan dengan kebutuhan industri. Secara bersamaan, perusahaan juga memperoleh keuntungan signifikan dengan adanya dokumen analisis sistem yang objektif sebagai bahan evaluasi internal, serta masukan dan ide-ide baru yang potensial untuk pengembangan produk di masa mendatang, sekaligus membina hubungan baik dengan institusi pendidikan untuk rekrutmen talenta.

1.5 Lokasi dan Waktu Pelaksanaan

Kerja praktik ini dilaksanakan pada waktu dan tempat sebagai berikut:

Lokasi : Online

Waktu : 27 Januari – 27 Juni 2025

Hari Kerja : Senin – Jumat

Jam Kerja : Fleksibel (minimal 12 jam per minggu)

1.6 Metodologi

Metodologi yang digunakan dalam pelaksanaan kerja praktik dan penyusunan laporan ini adalah pendekatan kualitatif dengan metode studi kasus. Tahapan yang dilakukan meliputi beberapa langkah sistematis sebagai berikut:

1. **Studi Literatur:** Tahap awal yang dilakukan adalah mengumpulkan landasan teori yang relevan dengan objek analisis. Studi ini mencakup pemahaman mendalam mengenai konsep-konsep inti seperti arsitektur perangkat lunak, pola desain modular pada *framework* NestJS, implementasi TypeScript, penggunaan *Object-Relational Mapping* (ORM) dengan TypeORM, desain RESTful API, serta penggunaan diagram UML untuk pemodelan sistem. Sumber literatur berasal dari dokumentasi resmi, buku, artikel ilmiah, dan publikasi online yang kredibel.

2. **Analisis Sistem (*Existing System*):** Pada tahap ini, dilakukan analisis mendalam terhadap struktur kode sumber (*source code*) dari *backend* aplikasi Bluemeet yang telah disediakan. Fokus utama analisis adalah untuk mengidentifikasi arsitektur yang diterapkan, teknologi yang digunakan, serta pola-pola desain yang ada. Analisis ini mencakup pembedahan struktur direktori, modul, *controller*, *service*, dan entitas untuk memahami bagaimana setiap komponen saling berinteraksi.

3. **Pemodelan Sistem:** Untuk memvisualisasikan hasil analisis dan mempermudah pemahaman, dilakukan pemodelan sistem menggunakan notasi standar. Ini termasuk pembuatan diagram arsitektur untuk menggambarkan komponen-komponen utama sistem dan hubungannya, serta *Sequence Diagram* untuk menggambarkan alur interaksi dan data pada fitur-fitur kunci seperti registrasi pengguna atau pemesanan perjalanan.

4. **Dokumentasi dan Pelaporan:** Tahap akhir adalah menyusun seluruh hasil analisis, pemodelan, dan pembahasan ke dalam sebuah laporan kerja praktik yang terstruktur. Laporan ini mendokumentasikan semua temuan secara sistematis, mulai dari latar belakang, landasan teori, analisis sistem, hingga kesimpulan dan saran yang dapat diberikan kepada perusahaan.

1.7 Sistematika Laporan

1.7.1 BAB I: PENDAHULUAN

Menguraikan latar belakang masalah, tujuan dan manfaat kerja praktik, serta metodologi dan sistematika penulisan laporan.

1.7.2 BAB II: PROFIL PERUSAHAAN

Menyajikan informasi mengenai PT Kertas Digital Indonesia, mencakup sejarah, visi misi, dan struktur organisasi perusahaan.

1.7.3 BAB III: TINJAUAN PUSTAKA

Menjelaskan landasan teori mengenai teknologi-teknologi utama yang digunakan dalam pengembangan *backend* Bluemeet, seperti Node.js, NestJS, dan TypeORM.

1.7.4 BAB IV: ANALISIS DAN PERANCANGAN SISTEM

Berisi analisis mendalam terhadap arsitektur, teknologi, dan alur data pada fitur-fitur inti aplikasi Bluemeet, disertai dengan contoh kode yang relevan.

1.7.5 BAB V: EVALUASI DAN PEMBAHASAN

Menyajikan evaluasi kritis terhadap sistem berdasarkan hasil analisis di bab sebelumnya, mencakup aspek arsitektur, keamanan, kinerja, serta potensi pengembangan.

1.7.6 BAB VI: KESIMPULAN DAN SARAN

Merangkum seluruh hasil analisis dan evaluasi, serta memberikan saran yang dapat bermanfaat bagi perusahaan dan pengembangan ilmu pengetahuan.

BAB 2 PROFIL PERUSAHAAN

2.1 Sejarah singkat

PT Kertas Digital Indonesia (KDI) didirikan pada tahun 2019 sebagai perusahaan *software house* yang berfokus pada pengembangan solusi digital. KDI memulai perjalanannya dengan mengembangkan sistem inventaris dan Point of Sale (PoS) untuk salah satu perusahaan ritel multinasional di Indonesia.

Pada tahun 2020, KDI beradaptasi dengan cepat terhadap tantangan pandemi COVID-19 dengan menyelenggarakan lebih dari 15 pameran virtual serta mendukung berbagai perusahaan dalam menjaga keterlibatan dengan pelanggan mereka secara daring.

Memasuki tahun 2021, KDI mulai melakukan riset mendalam mengenai tokenisasi aset dan teknologi Distributed Ledger (DLT), termasuk di dalamnya teknologi blockchain. Hasil dari riset ini mendorong KDI untuk mengerjakan dua proyek berbasis blockchain pada tahun 2022.

Sebagai langkah strategis di tahun 2023, KDI meluncurkan tecHeros, sebuah inisiatif yang bertujuan untuk memperkuat pengembangan produk dan kapasitas engineering lokal di Indonesia, sekaligus mendorong pertumbuhan talenta digital dalam negeri.

2.2 Logo Perusahaan

Gambar 2. 1 Logo Perusahaan

2.3 Visi Misi Perusahaan

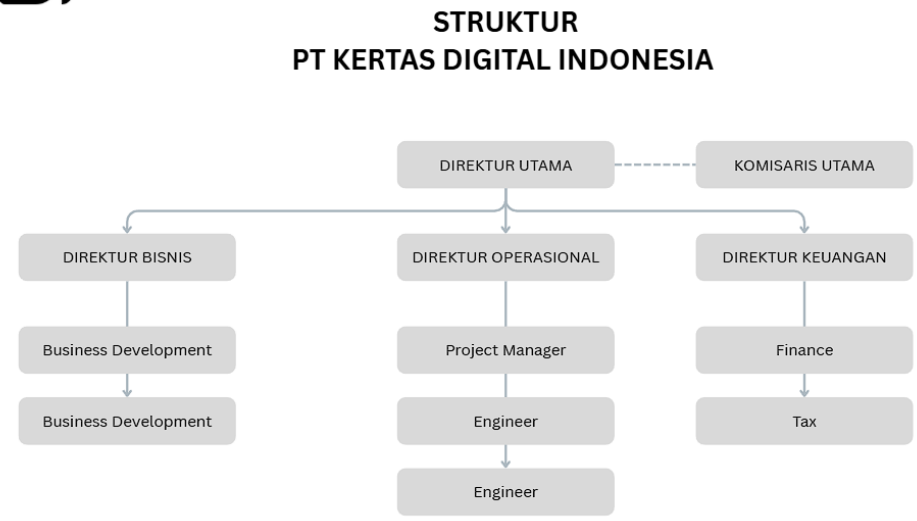
2.3.1 Visi

Perusahaan teknologi yang mengedepankan inovasi strategis dan solusi terdesentralisasi yang andal, kuat, dan ramah lingkungan untuk masa depan digital yang berkelanjutan.

2.3.2 Misi

1. Mengembangkan sistem teknologi terdesentralisasi dan terdistribusi yang aman, efisien, dan dapat diandalkan.
2. Mengintegrasikan pemikiran strategis dan kreatif dalam setiap solusi teknologi yang ditawarkan.
3. Berkontribusi pada pembangunan infrastruktur digital yang berkelanjutan dan ramah lingkungan di Indonesia.
4. Memberdayakan talenta lokal dan mendorong kolaborasi untuk menciptakan dampak positif dalam transformasi digital nasional.

2.4 Struktur Organisasi



Gambar 2. 2 Struktur Organisasi

BAB 3 TINJAUAN PUSTAKA

3.1 Node.js

Node.js adalah sebuah *runtime environment* untuk JavaScript yang bersifat *open-source*, lintas platform, dan dibangun di atas mesin JavaScript V8 milik Google Chrome. Node.js memungkinkan eksekusi kode JavaScript di sisi server, membebaskannya dari ketergantungan pada peramban web. Arsitektur Node.js didasarkan pada model I/O (*Input/Output*) yang bersifat *non-blocking* dan *event-driven*. Konsep ini diimplementasikan melalui *event loop*, sebuah mekanisme yang berjalan pada satu *thread* utama untuk menangani semua permintaan yang masuk. Ketika sebuah operasi yang memakan waktu (seperti membaca file atau melakukan kueri basis data) diminta, Node.js mendelegasikannya ke *worker thread* di latar belakang dan segera melanjutkan untuk menangani permintaan lain tanpa menunggu operasi tersebut selesai. Setelah operasi selesai, sebuah *event* akan dipicu dan *callback function* yang sesuai akan dimasukkan ke dalam antrian untuk dieksekusi oleh *event loop* (Node.js Foundation, 2024). Model ini membuat Node.js sangat efisien dan berkinerja tinggi untuk aplikasi yang intensif I/O, seperti API, aplikasi *real-time*, dan *microservices*. Ekosistem Node.js juga didukung oleh Node Package Manager (NPM), manajer paket terbesar di dunia, yang menyediakan akses ke ribuan pustaka dan alat bantu untuk mempercepat proses pengembangan.

3.2 TypeScript

TypeScript adalah bahasa pemrograman yang dikembangkan oleh Microsoft sebagai superset sintaksis dari JavaScript. Ini berarti bahwa setiap kode JavaScript yang valid secara sintaksis juga merupakan kode TypeScript yang valid. Fitur utama yang membedakan TypeScript adalah penambahan sistem tipe statis (*static type system*) (Microsoft, 2024). Dalam JavaScript standar, tipe data bersifat dinamis dan hanya diperiksa saat kode dijalankan (*runtime*), yang dapat menyebabkan kesalahan yang tidak terduga. Sebaliknya, TypeScript memungkinkan pengembang untuk mendefinisikan tipe data untuk variabel, parameter fungsi, dan nilai kembalian secara eksplisit. Proses ini, yang disebut *type-checking*, dilakukan pada saat kompilasi (*compile-time*). Manfaat utama dari penggunaan TypeScript antara lain:

- **Early debugging**: Kesalahan terkait tipe, seperti salah eja nama properti atau memberikan tipe data yang salah ke sebuah fungsi, dapat dideteksi sebelum aplikasi dijalankan.
- **Readability dan Maintenance**: Deklarasi tipe yang eksplisit berfungsi sebagai dokumentasi, membuat kode lebih mudah dipahami oleh pengembang lain atau oleh diri sendiri di masa depan.
- **Dukungan Alat Pengembangan yang Lebih Baik**: Editor kode seperti Visual Studio Code dapat memanfaatkan informasi tipe untuk menyediakan fitur *autocompletion* yang akurat, navigasi kode, dan *refactoring* yang aman. Untuk dapat dijalankan oleh lingkungan JavaScript seperti Node.js atau *browser*, kode TypeScript harus terlebih dahulu dikompilasi (atau *transpiled*) menjadi kode JavaScript standar.

3.3 NestJS

NestJS adalah sebuah kerangka kerja (framework) Node.js yang progresif untuk membangun aplikasi sisi server yang efisien, andal, dan dapat diskalakan. NestJS sepenuhnya

ditulis dengan TypeScript dan mengadopsi prinsip-prinsip pengembangan perangkat lunak yang matang seperti *Object-Oriented Programming* (OOP), *Functional Programming* (FP), dan *Functional Reactive Programming* (FRP) (NestJS Documentation, 2024). Arsitekturnya yang modular dan terstruktur sangat terinspirasi oleh Angular, mempromosikan organisasi kode yang rapi dan pemisahan tanggung jawab (*Separation of Concerns*). Komponen arsitektur fundamental dalam NestJS meliputi:

- **Modules**: Merupakan unit organisasi dasar yang mengelompokkan fungsionalitas terkait. Setiap aplikasi NestJS memiliki setidaknya satu modul akar (*root module*), dan fitur-fitur yang lebih besar dipecah menjadi modul-modul fitur.
- **Controllers**: Bertugas menerima permintaan HTTP yang masuk dari klien dan mengembalikan respons. Mereka mendefinisikan *routing* aplikasi dan merupakan jembatan antara klien dan logika bisnis.
- **Providers**: Konsep luas yang mencakup berbagai komponen NestJS seperti *Services*, *Repositories*, *Factories*, dan *Helpers*. Komponen yang paling umum adalah *Service*, yang dirancang untuk menampung logika bisnis yang kompleks dan dapat digunakan kembali. *Providers* dapat diinjeksikan sebagai dependensi ke komponen lain melalui mekanisme *Dependency Injection* (DI) bawaan NestJS, yang memfasilitasi pembuatan kode yang longgar keterikatannya (*loosely coupled*) dan mudah diuji.
- **Pipes, Guards, dan Interceptors**: Ini adalah mekanisme *middleware* yang kuat. *Pipes* digunakan untuk transformasi dan validasi data. *Guards* digunakan untuk otorisasi, menentukan apakah sebuah permintaan diizinkan untuk melanjutkan. *Interceptors* memungkinkan pengembang untuk mengikat logika tambahan sebelum atau sesudah eksekusi sebuah fungsi, berguna untuk *caching*, *logging*, atau mengubah format respons

3.4 TypeORM (Object-Relational Mapping)

TypeORM adalah sebuah pustaka *Object-Relational Mapping* (ORM) canggih yang ditulis sepenuhnya dalam TypeScript dan dapat berjalan di berbagai platform JavaScript, termasuk Node.js. Tujuan utama ORM seperti TypeORM adalah untuk menjembatani model pemrograman berorientasi objek dengan model basis data relasional yang berbasis tabel (TypeORM Documentation, 2024). Ini memungkinkan pengembang untuk berinteraksi dengan basis data menggunakan objek dan kelas TypeScript, alih-alih menulis kueri SQL mentah. Konsep utama dalam TypeORM meliputi:

- **Entities**: Kelas TypeScript yang dihiasi dengan dekorator Entity(). Setiap *entity* merepresentasikan sebuah tabel dalam basis data, dan properti di dalam kelas tersebut (dihiasi dengan Column(), PrimaryGeneratedColumn(), dll.) merepresentasikan kolom-kolom tabel.
- **Repositories**: Pola desain yang menyediakan abstraksi untuk mengakses data dari *entity* tertentu. TypeORM menyediakan *repository* bawaan untuk setiap *entity* yang berisi metode-metode standar untuk operasi CRUD (Create, Read, Update, Delete) seperti find(), findOne(), save(), dan remove().
- **Relations**: TypeORM memungkinkan definisi relasi antar *entity* (seperti OneToOne, OneToMany, ManyToOne, ManyToMany) langsung di dalam kelas *entity*, yang secara otomatis akan diterjemahkan menjadi relasi *foreign key* di basis data. Dengan

menggunakan TypeORM, kode menjadi lebih deklaratif, aman dari serangan *SQL injection* pada operasi dasar, dan lebih mudah untuk berpindah antar sistem basis data yang berbeda (misalnya dari PostgreSQL ke MySQL) dengan perubahan konfigurasi minimal.

3.5 RESTful API (Representational State Transfer)

REST (Representational State Transfer) adalah gaya arsitektur perangkat lunak yang mendefinisikan serangkaian batasan untuk membuat layanan web yang dapat diskalakan. Sebuah API yang mematuhi batasan-batasan ini disebut sebagai RESTful API. Diciptakan oleh Roy Fielding dalam disertasinya pada tahun 2000, REST bukanlah standar atau protokol, melainkan sebuah pendekatan desain (Fielding, 2000). Prinsip-prinsip utama yang mendefinisikan arsitektur REST adalah:

- **Client-Server Architecture**: Memisahkan secara tegas antara klien (yang menangani antarmuka pengguna) dan server (yang menangani penyimpanan dan logika data). Pemisahan ini memungkinkan keduanya untuk berevolusi secara independen.
- **Stateless**: Setiap permintaan dari klien ke server harus berisi semua informasi yang diperlukan untuk memahaminya. Server tidak menyimpan konteks atau sesi klien di antara permintaan. Seluruh status sesi disimpan di sisi klien.
- **Cacheable**: Respons dari server harus dapat didefinisikan sebagai *cacheable* atau *non-cacheable*. Hal ini memungkinkan klien atau perantara untuk menyimpan respons dalam *cache* guna meningkatkan kinerja.
- **Uniform Interface**: Ini adalah batasan fundamental yang menyederhanakan dan memisahkan arsitektur. Ini terdiri dari empat sub-batasan: identifikasi sumber daya (melalui URI), manipulasi sumber daya melalui representasi (misalnya, JSON), pesan yang mendeskripsikan diri sendiri, dan *hypermedia as the engine of application state* (HATEOAS). Dalam implementasi praktis, RESTful API menggunakan metode HTTP standar untuk operasi pada sumber daya: GET untuk mengambil data, POST untuk membuat data baru, PUT atau PATCH untuk memperbarui data, dan DELETE untuk menghapus data.

3.6 Migrasi Database (Database Migration)

Migrasi basis data adalah sebuah pendekatan sistematis untuk mengelola dan melacak perubahan pada skema basis data secara inkremental dan terprogram. Dalam pengembangan perangkat lunak modern, di mana beberapa pengembang bekerja secara bersamaan dan aplikasi di-*deploy* ke berbagai lingkungan (pengembangan, pengujian, produksi), menjaga konsistensi skema basis data menjadi tantangan. Migrasi mengatasi masalah ini dengan memperlakukan skema basis data sebagai bagian dari kode sumber aplikasi (TypeORM Documentation, 2024). Setiap perubahan pada skema—seperti membuat tabel, menambahkan kolom, membuat indeks, atau mengubah tipe data—ditulis dalam sebuah file migrasi. File ini biasanya berisi dua fungsi: up() dan down().

- Fungsi up() berisi logika untuk menerapkan perubahan (misalnya, CREATE TABLE users).

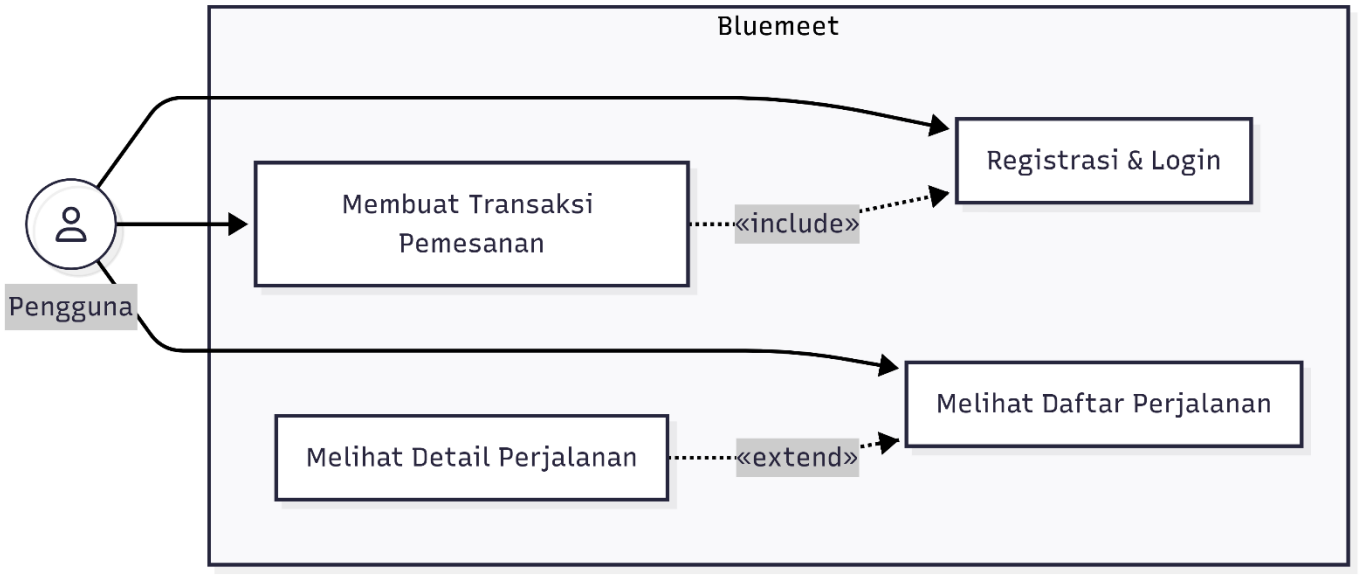
- Fungsi down() berisi logika untuk membatalkan atau mengembalikan perubahan tersebut (misalnya, DROP TABLE users). Alur kerja migrasi memungkinkan pengembang untuk menerapkan perubahan skema secara berurutan dan terjamin, serta membatalkan perubahan jika terjadi masalah. Alat bantu seperti TypeORM menyediakan antarmuka baris perintah (CLI) untuk secara otomatis menghasilkan dan menjalankan file-file migrasi ini. Hal ini memastikan bahwa semua lingkungan memiliki struktur basis data yang identik dan sinkron dengan versi kode aplikasi yang sedang berjalan.

BAB 4 ANALISIS DAN PERANCANGAN SISTEM

Bab ini berisi analisis mendalam terhadap sistem *backend* aplikasi Bluemeet. Analisis dimulai dari gambaran umum, kebutuhan fungsional, dan arsitektur sistem secara keseluruhan. Kemudian, dilanjutkan dengan analisis rinci untuk tiga fitur utama yang menjadi alur inti pengguna: Manajemen Pengguna, Penemuan Perjalanan, dan Proses Transaksi.

4.1 Gambaran Umum Sistem Aplikasi Bluemeet

Sistem *backend* Bluemeet dirancang sebagai sebuah RESTful API yang berfungsi sebagai penyedia layanan data dan logika bisnis untuk aplikasi klien (web atau mobile). Dibangun dengan arsitektur modular menggunakan NestJS, sistem ini memisahkan setiap domain bisnis utama (seperti pengguna, perjalanan, dan transaksi) ke dalam modul-modul independen. Setiap modul memiliki tanggung jawab yang jelas, mulai dari menerima permintaan HTTP, menjalankan logika bisnis, hingga berinteraksi dengan basis data. Arsitektur ini memungkinkan pengembangan yang terstruktur, pemeliharaan yang lebih mudah, dan skalabilitas di masa depan. Titik masuk utama bagi pengguna adalah melalui modul otentikasi, yang mengatur proses registrasi dan login sebelum pengguna dapat mengakses fitur-fitur inti lainnya seperti melihat daftar perjalanan dan melakukan pemesanan.



Gambar 4. 1 Use Case Pengguna

4.2 Analisis Kebutuhan

Berdasarkan analisis kode sumber, berikut adalah kebutuhan fungsional yang diidentifikasi untuk fitur-fitur inti yang akan dianalisis:

4.2.1 Kebutuhan Fungsional Manajemen Pengguna

- Sistem harus dapat mendaftarkan pengguna (lokal & Google).
- Sistem harus dapat mengautentikasi pengguna dan memberikan token.
- Sistem harus mengelola verifikasi email dan proses lupa password melalui OTP.

4.2.2 Kebutuhan Fungsional Penemuan Perjalanan

- Sistem harus dapat menampilkan daftar semua perjalanan (trip) yang tersedia.

- Sistem harus menyediakan fungsionalitas pencarian dan filter berdasarkan kota.
- Sistem harus mendukung paginasi untuk menampilkan data dalam jumlah besar.
- Sistem harus dapat menampilkan halaman detail untuk satu perjalanan spesifik.

4.2.3 Kebutuhan Fungsional Proses Transaksi

- Sistem harus dapat membuat catatan transaksi baru untuk sebuah perjalanan.
- Sistem harus dapat mengelola status transaksi (misal: PENDING, SUCCESS, FAILED).
- Sistem harus dapat mencatat *payload* permintaan dan respons dari interaksi dengan sistem eksternal (misalnya, *payment gateway*).

4.3 Analisis Arsitektur dan Teknologi

Arsitektur *backend* Bluemeet mengikuti pola desain standar dari NestJS yang sangat terstruktur dan modular. Teknologi utama yang digunakan adalah Node.js, TypeScript, NestJS, dan TypeORM. Analisis kode lebih lanjut mengidentifikasi penggunaan pustaka spesifik seperti bcrypt untuk enkripsi password, @nestjs/jwt untuk otentikasi berbasis token, @nestjs/cache-manager untuk manajemen OTP, dan nestjs-typeorm-paginate untuk paginasi data.

4.4 Analisis Rinci Alur Sistem per Fitur

Bagian ini akan membedah alur sistem untuk setiap fitur utama yang dipilih, disertai dengan contoh potongan kode yang relevan untuk mendukung analisis.

4.4.1 **Fitur 1: Manajemen Pengguna dan Otentikasi**

4.4.1.1 Deskripsi Fitur

Fitur ini merupakan gerbang masuk bagi pengguna ke dalam aplikasi Bluemeet. Fitur ini bertanggung jawab untuk mengelola identitas pengguna, proses pendaftaran, dan mekanisme login yang aman. Modul user menangani semua fungsionalitas yang terkait dengan pengguna, termasuk pendaftaran akun baru melalui dua metode: pendaftaran lokal yang menggunakan email dan password dengan verifikasi OTP (One-Time Password), dan pendaftaran/login cepat menggunakan akun Google. Modul ini juga mengelola proses otentikasi untuk pengguna yang kembali dan menyediakan fungsionalitas untuk pemulihan akun seperti lupa password.

4.4.1.2 Analisis Komponen Kode

- **user.controller.ts**: Berfungsi sebagai lapisan terluar yang menerima permintaan HTTP dari klien. File ini mendefinisikan *endpoints* API seperti /user/register dan /user/login. Controller ini kemudian mendelegasikan tugas ke UserService untuk diproses lebih lanjut.
- **user.service.ts**: Merupakan inti dari logika bisnis modul. Di sinilah proses validasi OTP, pengecekan email, enkripsi password, perbandingan password saat login, dan pembuatan JWT terjadi. Service ini berinteraksi langsung dengan *repository* basis data.
- **entities/**: Direktori ini berisi definisi struktur data yang akan dipetakan ke tabel di basis data menggunakan TypeORM. user.entity.ts mendefinisikan tabel users yang berisi data profil pengguna, sementara account.entity.ts (tidak ditampilkan, namun direferensikan) kemungkinan besar berisi data kredensial seperti email dan password.

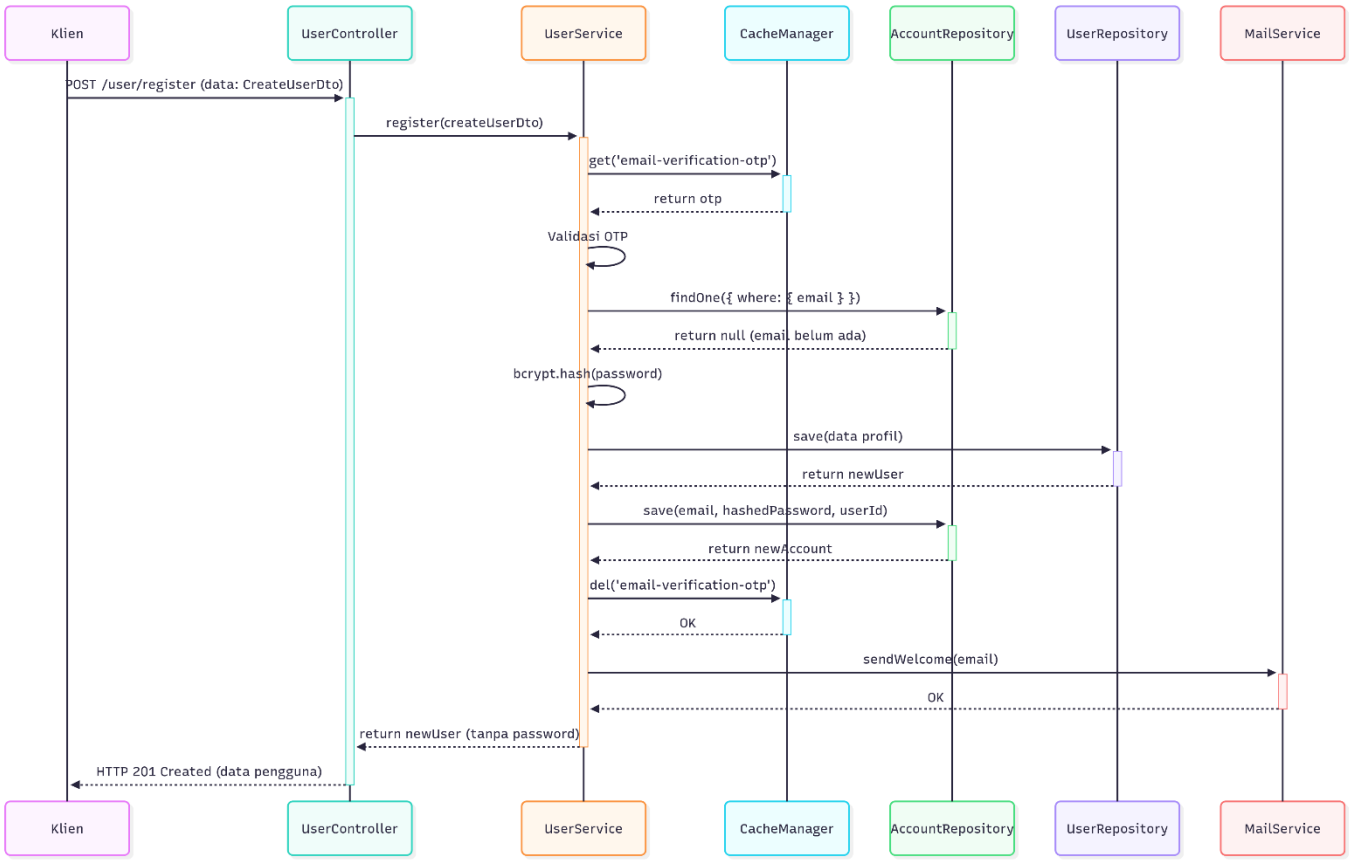
- **dto/ (Data Transfer Object)**: Direktori ini berisi kelas-kelas seperti create-user.dto.ts yang mendefinisikan "bentuk" data yang diharapkan dari klien. Kelas ini digunakan

oleh NestJS bersama class-validator untuk secara otomatis memvalidasi data yang masuk sebelum diproses oleh *controller*.

4.4.1.3 Analisis Alur Data

Alur Registrasi Pengguna Lokal Alur ini dimulai ketika pengguna mengirimkan data pendaftaran ke *endpoint* /user/register.

1. Klien -> *UserController*: Klien mengirim permintaan POST ke /user/register dengan body yang berisi data sesuai *CreateUserDto* (firstName, email, password, otp, dll).
2. *UserController -> UserService*: *UserController* memanggil metode register() pada *UserService* dan meneruskan *createUserDto* sebagai argumen.
3. Validasi OTP: *UserService* pertama-tama memeriksa validitas OTP yang dikirimkan. Ia mengambil OTP yang tersimpan di cache dengan kunci unik berdasarkan email pengguna. Jika OTP tidak ada atau tidak cocok, proses gagal dan mengembalikan pesan kesalahan.
4. Pengecekan Email: *UserService* melakukan kueri ke basis data melalui *accountRepository* untuk memeriksa apakah email tersebut sudah terdaftar. Jika sudah, proses dihentikan.
5. Enkripsi Password: *UserService* menggunakan *bcrypt.hash()* untuk mengenkripsi password yang diberikan pengguna.
6. Membuat Entitas Pengguna: *UserService* memanggil *userRepository.save()* untuk membuat entri baru di tabel users dengan data profil (nama, no. telp, dll).
7. Membuat Entitas Akun: Setelah pengguna berhasil dibuat, *UserService* memanggil *accountRepository.save()* untuk membuat entri baru di tabel accounts, menyimpan email, password yang sudah di-hash, dan menghubungkannya dengan ID pengguna yang baru saja dibuat.
8. Pembersihan & Notifikasi: *UserService* menghapus OTP dari cache dan memanggil *MailService* untuk mengirim email selamat datang.
9. *UserService -> UserController*: *UserService* mengembalikan data pengguna yang baru dibuat (tanpa password).
10. *UserController* -> Klien: *UserController* mengirimkan respons HTTP 201 (Created) beserta data pengguna ke klien.

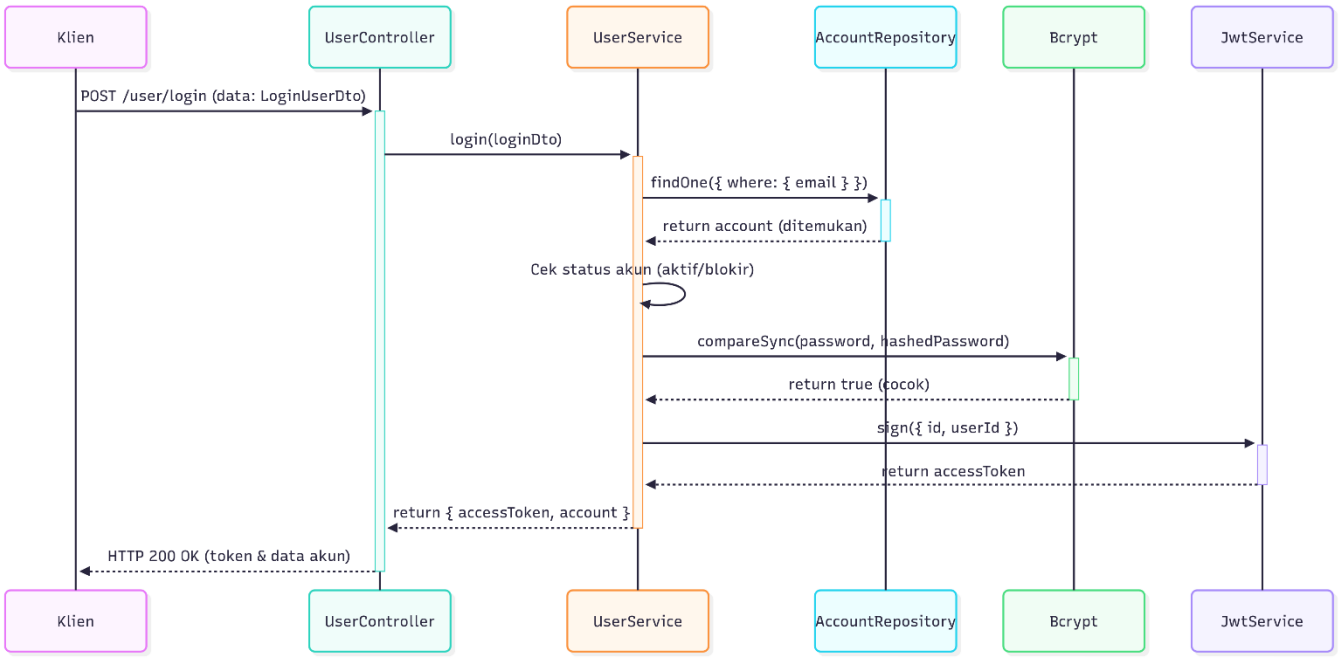


Gambar 4. 2 Sequence Diagram Registrasi Pengguna

Alur Login Pengguna Lokal Alur ini dimulai ketika pengguna mengirimkan kredensial ke *endpoint* /user/login.

1. Klien -> *UserController*: Klien mengirim permintaan POST ke /user/login dengan body berisi email dan password.
2. *UserController -> UserService*: *UserController* memanggil metode login() pada *UserService*.
3. Pencarian Akun: *UserService* menggunakan *accountRepository.findOne()* untuk mencari akun berdasarkan email yang diberikan. Jika tidak ditemukan, proses gagal.
4. Pengecekan Status Akun: Sistem memeriksa status akun (misalnya, active, inactive, block). Jika akun tidak aktif atau diblokir, proses dihentikan.
5. Verifikasi Password: *UserService* menggunakan *bcrypt.compareSync()* untuk membandingkan password yang dikirim klien dengan hash password yang tersimpan di basis data untuk akun tersebut. Jika tidak cocok, proses gagal.
6. Pembuatan Token: Jika verifikasi berhasil, UserService menggunakan *jwtService.sign()* untuk membuat JWT. Payload token berisi informasi identifikasi unik seperti id akun dan userId.
7. *UserService -> UserController*: *UserService* mengembalikan objek yang berisi accessToken (JWT) dan data akun (tanpa password).

8. *UserController* -> Klien: *UserController* mengirimkan respons HTTP 200 (OK) beserta token dan data akun ke klien. Klien kemudian harus menyimpan token ini untuk digunakan pada permintaan selanjutnya.



Gambar 4. 3 Sequence Diagram Login Pengguna

4.4.1.4 Desain Antarmuka API

Metode HTTP	URI Endpoint	Deskripsi	DTO yang Digunakan
POST	/user/login	Mengautentikasi pengguna dan mengembalikan JWT.	LoginUserDto
POST	/user/register	Mendaftarkan pengguna baru.	CreateUserDto
POST	/user/email-verification	Mengirim OTP ke email pengguna untuk verifikasi sebelum registrasi.	EmailVerification
POST	/user/forgot-password	Mengirim OTP ke email pengguna untuk proses lupa password.	EmailVerification
POST	/user/change-password	Mengubah password pengguna setelah verifikasi OTP lupa password.	ChangePasswordDto
POST	/user/check-user	Memeriksa apakah seorang pengguna sudah terdaftar atau belum.	LoginUserDto

Tabel 4. 1 Antarmuka API Manajemen User

4.4.1.5 Contoh Source Code

- Validasi Data Input (create-user.dto.ts):

```ts
// create-user.dto.ts
import { IsArray, IsDate, IsDateString, IsEmail, IsEnum, IsNotEmpty,
IsNumber, IsNumberString, IsOptional, IsPhoneNumber, IsUrl, MaxLength,
Min, MinLength, ValidateIf, ValidateNested } from "class-validator";

export class CreateUserDto {
    // ...
    @IsEmail()
    @IsNotEmpty()
    @ValidateIf(o => o.accountType !== 'google')
    email: string;

    @MinLength(6)
    @ValidateIf(o => o.accountType !== 'google')
    @IsNotEmpty()
    password: string;
    // ...
}
```

Kode 4. 1 Validasi Data Input

- Logika Enkripsi Password (user.service.ts):

```ts
// user.service.ts
import * as bcrypt from 'bcrypt';

// ... (di dalam metode register)
const hashPass = await bcrypt.hash(registerDto.password, 10);
const newAccount = await this.accountRepository.save({
  email: registerDto.email.toLowerCase(),
  password: hashPass,
  // ...
});
```

Kode 4. 2 Logika Enkripsi Password

4.4.2 **Fitur 2: Penemuan dan Detail Perjalanan**

4.4.2.1 Deskripsi Fitur

Fitur ini merupakan inti dari katalog produk aplikasi Bluemeet, yang memungkinkan pengguna untuk menjelajahi dan menemukan perjalanan selam yang mereka minati. Fitur ini menyediakan fungsionalitas bagi pengguna untuk melihat daftar semua perjalanan yang tersedia dengan paginasi, melakukan pencarian berdasarkan nama, memfilter berdasarkan lokasi (kota), dan melihat informasi rinci dari setiap perjalanan. Fitur ini melibatkan beberapa modul seperti trip, city, category, dan images yang saling berelasi untuk menyajikan data yang komprehensif.

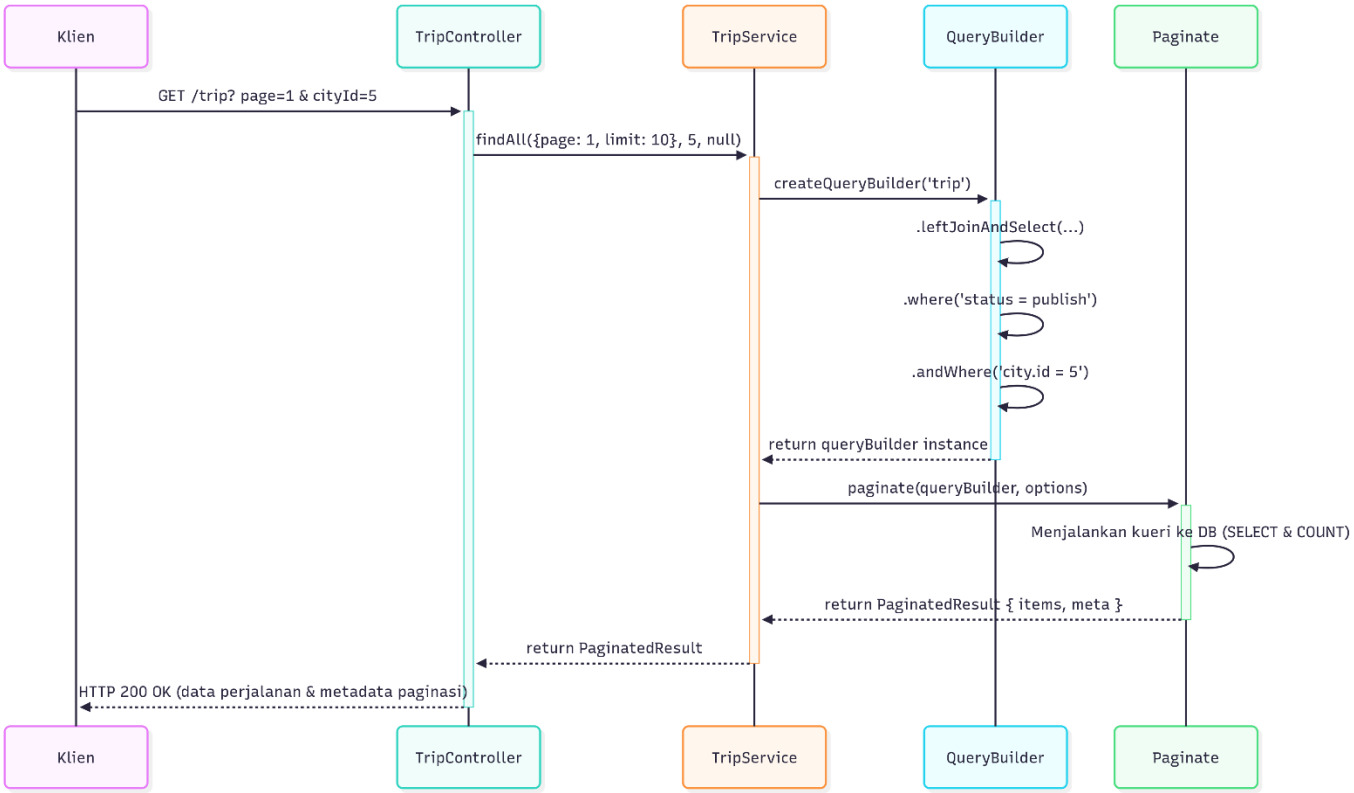
4.4.2.2 Analisis Komponen Kode

- **trip.controller.ts**: Berfungsi sebagai gerbang untuk semua permintaan terkait perjalanan. Controller ini mendefinisikan *endpoint* utama seperti GET /trip untuk mendapatkan daftar perjalanan dengan berbagai parameter kueri (page, limit, cityId, search) dan GET /trip/:slug untuk mendapatkan detail satu perjalanan. Penggunaan dekorator @Query memungkinkan pengambilan parameter dari URL secara dinamis.
- **trip.service.ts**: Merupakan pusat logika untuk fitur ini. Metode findAll() menggunakan createQueryBuilder dari TypeORM untuk membangun kueri yang dinamis dan kompleks. Metode ini secara efisien menggabungkan (LEFT JOIN) tabel trip dengan tabel-tabel relasionalnya (city, category, images, merchant) untuk mengambil semua data yang diperlukan dalam satu kueri, menghindari masalah N+1. Selain itu, ia menerapkan logika filter dan pencarian secara kondisional sebelum menggunakan pustaka nestjs-typeorm-paginate untuk membagi hasilnya ke dalam halaman-halaman. Metode findOne() bertanggung jawab untuk mengambil satu entitas perjalanan beserta relasinya.
- **entities/*.entity.ts**: Entitas seperti trip.entity.ts dan category.entity.ts mendefinisikan struktur tabel di basis data. Berdasarkan kueri di TripService, dapat disimpulkan bahwa entitas Trip memiliki relasi Many-to-One dengan Category, City, dan Merchant, serta relasi One-to-Many dengan Images.

4.4.2.3 Analisis Alur Data

1. **Klien -> TripController**: Klien mengirim permintaan GET ke /trip dengan parameter kueri opsional, misalnya /trip?page=1&limit=10&cityId=5.
2. **TripController -> TripService**: Controller menerima parameter kueri dan memanggil metode findAll() pada TripService, meneruskan opsi paginasi dan filter.
3. **Pembangunan Kueri**: TripService menggunakan createQueryBuilder untuk memulai kueri pada tabel trip. Ia menambahkan klausa leftJoinAndSelect untuk mengambil data dari tabel-tabel terkait.
4. **Penerapan Filter**: Jika parameter cityId atau search ada, klausa andWhere() ditambahkan ke kueri untuk memfilter hasil.
5. **Eksekusi Kueri & Paginasi**: Pustaka paginate dieksekusi, yang akan secara otomatis menambahkan klausa LIMIT dan OFFSET ke kueri SQL dan menjalankannya ke basis data. Pustaka ini juga melakukan kueri hitungan total untuk metadata paginasi.
6. **TripService -> TripController**: TripService mengembalikan objek paginasi yang berisi data perjalanan untuk halaman saat ini (items) dan informasi meta (totalItems, totalPages, dll.).

7. **TripController -> Klien**: Controller mengirimkan objek paginasi tersebut sebagai respons HTTP 200 (OK).



Gambar 4. 4 Sequence Diagram Perjalanan

4.4.2.4 Desain Antarmuka API

Metode HTTP	URI Endpoint	Deskripsi
GET	/trip	Mengambil daftar perjalanan dengan paginasi, filter, dan pencarian.
GET	/trip/top-location	Mengambil daftar 4 lokasi teratas berdasarkan jumlah perjalanan.
GET	/trip/:slug	Mengambil informasi detail dari satu perjalanan.

Tabel 4. 2 Antarmuka API Perjalanan

4.4.2.5 Contoh Source Code

- Logika Query

```
// trip.service.ts
import { paginate } from 'nestjs-typeorm-paginate';

findAll(options: IPaginationOptions, cityId: number, search: string) {
  const queryBuilder = this.tripRepository.createQueryBuilder('trip');
  queryBuilder
    .leftJoinAndSelect('trip.city', 'city')
    .leftJoinAndSelect('trip.category', 'category')
    .leftJoinAndSelect('trip.images', 'images')
    .where('trip.status = :status', { status: 'publish' });

  if (cityId) {
    queryBuilder.andWhere('trip.city.id = :cityId', { cityId });
  }

  // ... (logika pencarian)

  return paginate(queryBuilder, options);
}
```

Kode 4. 3 Logika Query

4.4.3 **Fitur 3: Proses Pemesanan (Transaksi)**

4.4.3.1 Deskripsi Fitur

Fitur ini merupakan puncak dari alur pengguna, di mana pengguna melakukan pemesanan untuk perjalanan yang telah mereka pilih. Fitur ini sangat krusial karena melibatkan pencatatan data finansial dan sering kali berintegrasi dengan sistem eksternal. Modul ini dirancang untuk dapat diperluas guna menangani berbagai jenis transaksi dan terintegrasi dengan layanan pembayaran eksternal.

4.4.3.2 Analisis Komponen Kode

- **transaction.controller.ts**: Mengekspos *endpoint* POST /transaction. Controller ini mendelegasikan tugas ke PaymentService, menunjukkan bahwa ini adalah desain yang baik karena proses pembayaran yang rumit ditangani oleh layanan khusus, terpisah dari manajemen data transaksi dasar.
- **transaction.service.ts**: Adanya file transaction.service.ts (meski masih kosong) dan payment.service.ts di dalam modul transaction menunjukkan bahwa arsitek sistem telah merencanakan pemisahan tanggung jawab secara jelas. Sistem dirancang untuk memisahkan logika murni manajemen data transaksi, yang seharusnya berada di TransactionService, dari logika proses pembayaran yang lebih kompleks, yang ditangani oleh PaymentService. Ini merupakan praktik desain yang baik untuk menjaga agar struktur kode tetap terorganisir, mudah dipelihara, dan scalable. Meskipun pada kode yang dianalisis layanan TransactionService masih berupa placeholder, peran utamanya dalam arsitektur penuh adalah untuk menangani operasi CRUD murni terhadap entitas Transaction dan TransactionItem. Layanan ini juga akan digunakan oleh PaymentService, misalnya untuk membuat

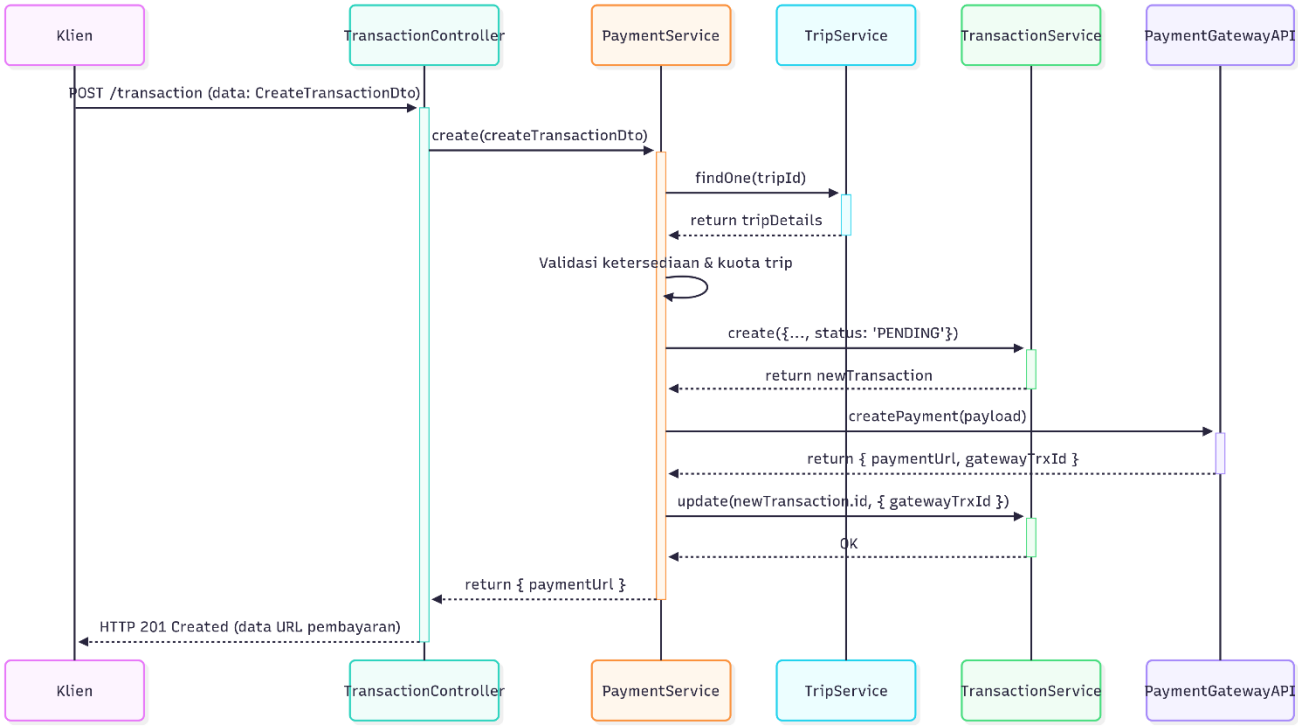
catatan transaksi awal dengan status PENDING, lalu memperbaruinya menjadi SUCCESS setelah proses pembayaran berhasil dikonfirmasi.

- **transaction.entity.ts**: Entitas ini dirancang untuk menjadi catatan audit yang detail dan robust. Kolom penting seperti trxId (ID internal), trxIdPaymentGateway (ID eksternal), status (PENDING, SUCCESS, FAILED), serta requestPayload dan responsePayload menunjukkan bahwa sistem ini dirancang untuk berinteraksi dengan *payment gateway* dan menyimpan log komunikasi yang lengkap untuk keperluan *debugging* dan audit.
- **dto/create-transaction.dto.ts**: Meskipun file create-transaction.dto.ts saat ini kosong, dapat diinferensikan bahwa untuk memulai sebuah transaksi, DTO ini setidaknya akan memerlukan properti seperti tripId: number dan quantity: number untuk mengidentifikasi perjalanan mana yang dipesan dan berapa banyak."

4.4.3.3 Analisis Alur Data (Hipotesis)

Karena logika bisnis inti tidak ada dalam TransactionService, alur data berikut diinferensikan berdasarkan struktur yang ada dan praktik umum dalam sistem pemesanan.

1. **Klien -> TransactionController**: Pengguna yang telah login mengirim permintaan POST ke /transaction dengan *body* yang berisi data pemesanan (misalnya, ID perjalanan dan jumlah peserta).
2. **TransactionController -> PaymentService**: Controller memanggil metode create() pada PaymentService, meneruskan data pemesanan.
3. **Logika di PaymentService**:
    a. **Validasi**: PaymentService akan memanggil TripService untuk memvalidasi ketersediaan perjalanan (misalnya, memeriksa kuota).
    b. **Membuat Catatan Transaksi**: PaymentService memanggil TransactionService untuk membuat entri baru di tabel transactions dengan status PENDING. Data seperti ID pengguna, detail item, dan total harga disimpan.
    c. **Interaksi dengan Payment Gateway**: PaymentService mempersiapkan *payload* dan melakukan permintaan ke API *payment gateway* eksternal (misalnya, Midtrans, Xendit).
    d. **Menyimpan Respons Gateway**: PaymentService menerima respons dari *gateway* (yang mungkin berisi URL pembayaran atau kode QR) dan memperbarui catatan transaksi dengan trxIdPaymentGateway dan responsePayload.
4. **PaymentService -> TransactionController**: PaymentService mengembalikan data yang relevan ke controller, seperti URL pembayaran.
5. **TransactionController -> Klien**: Controller mengirimkan respons HTTP 201 (Created) beserta URL pembayaran ke klien. Klien kemudian diarahkan ke halaman pembayaran.
6. **Alur Notifikasi (Webhook)**: Setelah pengguna menyelesaikan pembayaran, *payment gateway* akan mengirim notifikasi *server-to-server* (webhook) ke sebuah *endpoint* khusus di *backend* Bluemeet (misalnya, POST /payment/notification). *Endpoint* ini akan memvalidasi notifikasi, lalu memanggil PaymentService untuk memperbarui status transaksi di basis data menjadi SUCCESS atau FAILED.



Gambar 4. 5 Sequence Diagram Transaksi

4.4.3.4 Desain Antarmuka API

Metode HTTP	URI Endpoint	Deskripsi
POST	/transaction	Memulai proses pembuatan transaksi dan pembayaran baru.

Tabel 4. 3 Antarmuka API Transaksi

4.4.3.5 Contoh Source Code

- Struktur Entity Transaksi (transaction.entity.ts):

```ts
// transaction.entity.ts
import { User } from "src/modules/user/entities/user.entity";
import { PrimaryGeneratedColumn, Column, CreateDateColumn, UpdateDateColumn,
DeleteDateColumn, Entity, ManyToOne, OneToMany } from "typeorm";

@Entity({ name: 'transactions' })
export class Transaction {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({ nullable: false, type: 'enum', enum: ['SUCCESS', 'FAILED', 'PENDING'] })
    status: string;

    @Column({ nullable: true, type: 'text' })
    responsePayload: string;

    @ManyToOne(() => User, (user) => user.transactions)
    user: User;
}
```

Kode 4. 5 Struktur Entity Transaksi

- Delegasi ke Payment Service (transaction.controller.ts):

```ts
// transaction.controller.ts
import { Controller, Post, Body } from '@nestjs/common';
import { PaymentService } from './services/payment.service';
import { CreateTransactionDto } from './dto/create-transaction.dto';

@Controller('transaction')
export class TransactionController {
  constructor(
    private readonly transactionService: TransactionService,
    private readonly paymentService: PaymentService,
  ) {}

  @Post()
  create(@Body() createTransactionDto: CreateTransactionDto) {
    return this.paymentService.create(createTransactionDto);
  }
}
```

Kode 4. 4 Delegasi ke Payment Service

BAB 5 EVALUASI

Bab ini menyajikan evaluasi kritis terhadap sistem *backend* aplikasi Bluemeet berdasarkan hasil analisis yang telah dipaparkan di Bab 4. Evaluasi ini mencakup aspek arsitektur, keamanan, kinerja, serta mengidentifikasi potensi pengembangan dan area diskusi untuk masa mendatang.

5.1 Evaluasi Arsitektur dan Desain Sistem

Pemilihan teknologi stack yang terdiri dari NestJS, TypeScript, dan TypeORM merupakan keputusan yang sangat tepat untuk aplikasi seperti Bluemeet.

1. Keunggulan
   - **Arsitektur Modular**: Penggunaan modul pada NestJS (misalnya user, trip, transaction) mendorong penerapan *Separation of Concerns*. Hal ini membuat kode lebih terorganisir, mudah dipahami, dan memudahkan tim untuk bekerja secara paralel pada fitur yang berbeda tanpa menimbulkan konflik.
   - **Kualitas Kode dan *Maintenance***: TypeScript dengan sistem tipe statisnya secara signifikan mengurangi potensi *bug* saat *runtime* dan meningkatkan keterbacaan kode. Kombinasi dengan DTO (*Data Transfer Object*) dan class-validator menciptakan lapisan validasi yang kuat dan deklaratif pada gerbang masuk API.
   - **Pemisahan Tanggung Jawab yang Jelas**: Pola desain yang memisahkan Controller (penanganan HTTP), Service (logika bisnis), dan Repository (akses data) sangat efektif. Contohnya terlihat jelas pada delegasi proses pembayaran dari TransactionController ke PaymentService, yang menunjukkan desain matang untuk memisahkan alur kerja yang kompleks.
2. Area Diskusi

   Kompleksitas Abstraksi: Meskipun sangat kuat, tingkat abstraksi yang tinggi pada NestJS dan TypeORM dapat menjadi kurva belajar yang curam bagi pengembang baru. Namun, untuk proyek jangka panjang, investasi dalam mempelajari arsitektur ini akan sangat terbayarkan dalam hal pemeliharaan dan skalabilitas.

5.2 Evaluasi Keamanan

Keamanan adalah aspek krusial untuk aplikasi yang menangani data pengguna dan transaksi. Sistem Bluemeet telah menerapkan fondasi keamanan yang baik.

1. Keunggulan
   - **Enkripsi Password**: Implementasi bcrypt untuk melakukan *hashing* pada password pengguna adalah praktik standar industri yang sangat baik. Ini memastikan bahwa password asli tidak pernah disimpan dalam bentuk teks biasa.
   - **Otentikasi Berbasis Token**: Penggunaan JWT (@nestjs/jwt) untuk mengelola sesi pengguna adalah metode yang modern dan *stateless*, cocok untuk arsitektur RESTful API dan memudahkan skalabilitas horizontal.
   - **Validasi Input**: Penggunaan DTO dengan class-validator secara otomatis menolak permintaan dengan data yang tidak valid, menjadi lapisan pertahanan pertama terhadap serangan seperti *injection*.
2. Potensi Peningkatan

- **Otorisasi Berbasis Peran (RBAC)**: Saat ini, sistem telah memiliki fondasi otentikasi (siapa Anda). Langkah selanjutnya yang dapat dipertimbangkan adalah implementasi otorisasi (apa yang boleh Anda lakukan). Dengan menggunakan *Guards* dari NestJS, sistem dapat dengan mudah menerapkan aturan hak akses, misalnya membedakan antara pengguna biasa, pemilik *merchant*, dan admin sistem.
- **Keamanan JWT Lanjutan**: Untuk meningkatkan keamanan token, dapat dipertimbangkan implementasi mekanisme *refresh token*. Ini memungkinkan accessToken memiliki masa berlaku yang sangat singkat (misalnya 15 menit), sementara refreshToken yang disimpan dengan aman dapat digunakan untuk meminta accessToken baru tanpa mengharuskan pengguna login kembali.

5.3 Evaluasi Kinerja dan Skalabilitas

1. Keunggulan
   - **Paginasi**: Implementasi paginasi pada *endpoint* GET /trip menggunakan nestjs-typeorm-paginate adalah langkah yang sangat baik untuk skalabilitas. Ini mencegah server mengirimkan ribuan data sekaligus, yang dapat menyebabkan penggunaan memori yang tinggi dan waktu respons yang lama.
   - **Query yang Efisien**: Penggunaan leftJoinAndSelect pada createQueryBuilder di TripService menunjukkan pemahaman untuk menghindari masalah performa N+1, di mana data terkait diambil dalam satu kueri efisien ke basis data.
2. Potensi Peningkatan
   - **Strategi Caching**: Untuk data yang jarang berubah namun sering diakses (misalnya daftar kategori, daftar kota, atau bahkan detail perjalanan yang sangat populer), implementasi mekanisme *caching* menggunakan @nestjs/cache-manager yang sudah terpasang dapat secara drastis mengurangi beban pada basis data dan mempercepat waktu respons.
   - **Optimasi Indeks Basis Data**: Performa kueri pada trip yang menggunakan filter cityId dan pencarian name akan meningkat secara signifikan jika terdapat indeks pada kolom-kolom tersebut di tabel basis data. Analisis kueri yang sering digunakan dapat menjadi dasar untuk strategi pengindeksan yang efektif.

BAB 6 KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan analisis yang telah dilakukan pada sistem *backend* aplikasi Bluemeet, dapat disimpulkan bahwa sistem ini dibangun di atas fondasi arsitektur yang modern, terstruktur, dan selaras dengan praktik terbaik industri perangkat lunak. Pemilihan teknologi *stack* yang terdiri dari NestJS, TypeScript, dan TypeORM memberikan keunggulan signifikan dalam hal *readability*, *maintenance*, dan *scalability*. Arsitektur modular yang memisahkan setiap domain bisnis secara jelas, ditambah dengan penerapan pola desain *Controller-Service-Repository*, menciptakan sistem yang kokoh dan mudah untuk dikembangkan secara berkelanjutan oleh tim.

Dari sisi fungsionalitas, alur kerja pada fitur-fitur inti seperti manajemen pengguna dan penemuan perjalanan telah diimplementasikan secara efisien dan aman. Sistem telah menerapkan standar keamanan esensial seperti enkripsi password dan otentikasi berbasis token, serta teknik optimasi kinerja seperti paginasi dan kueri yang efisien. Secara keseluruhan, sistem *backend* Bluemeet tidak hanya fungsional, tetapi juga memiliki fondasi yang kuat dan matang, yang membuatnya siap untuk menghadapi tantangan pengembangan fitur baru dan peningkatan jumlah pengguna di masa mendatang.

6.2 Saran

Berdasarkan analisis terhadap arsitektur dan kode yang ada, berikut adalah beberapa area yang dapat menjadi fokus untuk pengembangan dan penyempurnaan sistem di masa depan. Rekomendasi ini bersifat sebagai bahan diskusi.

1. **Implementasi *Logging* dan *Monitoring* Terpusat**: Untuk meningkatkan *observability* sistem, direkomendasikan untuk mengintegrasikan sistem *logging* terpusat. Library seperti Pino atau Winston dapat digunakan untuk menghasilkan *log* terstruktur dalam format JSON, yang kemudian dapat dikirim ke platform agregasi *log* seperti ELK Stack atau Datadog.
2. **Pengembangan Fitur Notifikasi *Real-time***: Untuk meningkatkan *user engagement* dan memberikan instan *feedback* (misalnya, saat status transaksi berubah dari PENDING menjadi SUCCESS), sistem dapat dikembangkan untuk mendukung notifikasi *real-time*.

DAFTAR PUSTAKA

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*.

Microsoft. (2024). *TypeScript Documentation*. Diakses dari https://www.typescriptlang.org/docs/

NestJS Documentation. (2024). *Introduction*. Diakses dari https://docs.nestjs.com/

Node.js Foundation. (2024). *About Node.js*. Diakses dari https://nodejs.org/en/about

OmniTI. (2012). *JSend: A Specification for a Simple JSON-based Format for Application-level Communication*. Diakses dari https://github.com/omniti-labs/jsend

Richards, M. (2015). *Software Architecture Patterns*. O'Reilly Media.

TypeORM Documentation. (2024). *What is TypeORM?*. Diakses dari https://typeorm.io/

LAMPIRAN

1. Persetujuan Kerja Praktik

DEPARTEMEN TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SEPULUH NOPEMBER

FORMULIR PENGAJUAN KERJA PRAKTIK

Nama	: Aloysius deDeo Darmo Susanto
NRP	: 5025221174
SKS Ditempuh (min 90 sks)	: 101
Alamat Surat	: Pondok Jati AK-50, Buduran, Kab. Sidoarjo, Jawa Timur
Nomor Telepon	: 0822-3377-9731

Nama	: _____
NRP	: _____
SKS Ditempuh (min 90 sks)	: _____
Alamat Surat	: _____
Nomor Telepon	: _____

Akan Melakukan Kegiatan Kerja Praktik di:

Nama instansi / perusahaan	: PT Kertas Digital Indonesia (Kdigital.id)
Alamat perusahaan	: CIBIS NINE BUILDING 11th Floor, CIBIS PARK, CILANDAK KKO, JAKARTA SELATAN, 12560
Nomor telepon	: +62 815 900 1927
Nama wakil instansi/perusahaan	: Danial Daniarta
Jabatan/divisi wakil instansi	: Direktur Bisnis
Rencana tanggal pelaksanaan	: 27 Januari 2025 – 27 April 2025
Pekerjaan/deskripsi	: WEB2 Developer

Surabaya, 24 Januari 2025

Pemohon 1,

(Aloysius deDeo Darmo Susanto)

Pemohon 2,

(_____)

Menyetujui,
Koordinator KP

Hadziq Fabroyir, S.Kom., Ph.D.

*) Tiap kelompok mengisi 1 lembar formulir, untuk persetujuan dikirimkan ke hadziq@its.ac.id
cc ke inka.nd@its.ac.id dengan subject "**Pengajuan Kerja Praktik**" dalam format PDF

2. Source Code

- User.service.ts

  https://gist.github.com/aloyyys/c1cf56520ded31aec03b3a93fb947503

- User.entity.ts

```
// import{
//   ...
// }

@Entity({ name: 'users' })
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column({ nullable: false })
  firstName: string;

  @Column({ nullable: true })
  lastName: string;

  @ManyToOne(() => Country, (country) => country.user, { nullable: true })
  country: Country;

  @ManyToOne(() => City, (city) => city.user, { nullable: true })
  domicile: City;

  @Column({ nullable: true })
  phoneNumber: string;

  @Column({ nullable: true })
  birthDate: Date;

  @Column({ nullable: true })
  logDives: number;

  @Column({ nullable: true })
  referenceBy: string;

  @Column({
    nullable: true,
  })
  typeDiver: string;

  @CreateDateColumn({ type: 'timestamptz' })
```

```
  createdAt!: Date;

  @UpdateDateColumn({ type: 'timestamptz' })
  updatedAt!: Date;

  @DeleteDateColumn({ type: 'timestamptz' })
  deletedAt!: Date;

  @OneToMany(() => Account, (account) => account.user)
  account: Account[];

  @OneToMany(() => TripReviews, (tripReviews) => tripReviews.user)
  review: TripReviews[];

  @OneToMany(() => Transaction, (transaction) => transaction.user)
  transactions: Transaction[];

  @OneToMany(() => TripParticipant, (tripParticipant) => tripParticipant.user)
  participants: TripParticipant[];

  @OneToMany(() => UserCertificate, (userCertificate) => userCertificate.user,
{ nullable: true })
  certificate: UserCertificate[];
}
```

- Trip.service.ts

```
// import{
//   ...
// }

@Injectable()
export class TripService {
  constructor(
    @InjectRepository(Trip)
    private readonly tripRepository: Repository<Trip>,
    @InjectRepository(TripReviews)
    private readonly tripReviewRepository: Repository<TripReviews>,
    @InjectRepository(City)
    private readonly cityRepository: Repository<City>,
  ) { }

  findAll(options: IPaginationOptions, cityId: number, search: string) {
    const queryBuilder = this.tripRepository.createQueryBuilder('trip');
    queryBuilder
      .leftJoinAndSelect('trip.city', 'city')
      .leftJoinAndSelect('trip.repeatDays', 'repeatDays')
      .leftJoinAndSelect('trip.category', 'category')
      .leftJoinAndSelect('trip.images', 'images')
```

```
      .leftJoinAndSelect('trip.merchant', 'merchant')
      .where('trip.status = :status', {
        status: 'publish',
      });

    if (cityId) {

      queryBuilder.andWhere('trip.city.id = :cityId', {
        cityId,
      });
    }

    if (search) {
      queryBuilder.andWhere('trip.name LIKE :search', {
        search: `%${search}%`,
      });
    }

    queryBuilder.orderBy('trip.id', 'DESC');
    return paginate(queryBuilder, options);
  }

  findOne(slug: string) {
    return this.tripRepository.findOne({
      relations: ['category', 'images', 'city', 'repeatDays', 'language',
'catalog', 'additionalGear'],
      where: {
        slug,
        status: 'publish',
      },
    });
  }

  async findAllTopLocation() {
    const queryBuilder = this.cityRepository.createQueryBuilder('city');
    queryBuilder.addSelect((qb) => {
      return qb.select('COUNT(trip.id)', 'count').from(Trip, 'trip')
      .where('trip.city.id = city.id')
      .andWhere('trip.status = :status', { status: 'publish' });
    }, 'count')
    queryBuilder.take(4);
    queryBuilder.orderBy('count', 'DESC');
    return queryBuilder.getRawMany();
  }
}
```

- Transaction.controller.ts

```
// import{
//   ...
// }

@Controller('transaction')
export class TransactionController {
  constructor(
    private readonly transactionService: TransactionService,
    private readonly paymentService: PaymentService,
  ) {}

  @Post()
  create(@Body() createTransactionDto: CreateTransactionDto) {
    return this.paymentService.create(createTransactionDto);
  }
}
```

- Transaction.entity.ts

```
// import{
//   ...
// }

@Entity({ name: 'transactions' })
export class Transaction {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({ nullable: false })
    trxId: string;

    @Column({ nullable: true })
    trxIdPaymentGateway: string;

    @Column({ nullable: false, type: 'enum', enum: ['Trip', 'Event',
'Ecommerce'], default: 'Trip' })
    type: string;

    @Column({ nullable: false, type: 'enum', enum: ['SUCCESS', 'FAILED',
'PENDING'] })
    status: string;

    @Column({ nullable: true, type: 'text' })
    requestPayload: string;

    @Column({ nullable: true, type: 'text' })
```

```
    responsePayload: string;

    @CreateDateColumn({ type: 'timestamptz' })
    createdAt!: Date;

    @UpdateDateColumn({ type: 'timestamptz' })
    updatedAt!: Date;

    @DeleteDateColumn({ type: 'timestamptz' })
    deletedAt!: Date;

    @ManyToOne(() => User, (user) => user.transactions)
    user: User;

    @OneToMany(() => TransactionItem, (transactionItem) =>
transactionItem.transaction)
    transactionItems: TransactionItem[];
}
```

BIODATA PENULIS

Nama	: Aloysius deDeo Darmo Susanto
Tempat, Tanggal Lahir	: Sidoarjo, 25 April 2004
Jenis Kelamin	: Laki-laki
Telepon	: +62 82233779731
Email	: darmoaloysius@gmail.com

AKADEMIS

Kuliah	: Departemen Teknik Informatika – FTEIC , ITS
Angkatan	: 2022
Semester	: 6 (Enam)