BAB III

PELAKSANAAN KERJA

3.1 Kedudukan dan Koordinasi

3.1.1 Kedudukan

Pelaksanaan program *Career Acceleration Program* bertempat di Dinas *Corporate Strategy & Digital Transformation* (TD) PT GMF AeroAsia Tbk. Unit ini memegang peranan krusial sebagai pusat inovasi dan digitalisasi bagi seluruh proses bisnis MRO perusahaan. Dinas TD bertanggung jawab langsung kepada Direktur Utama (*Chief Executive Officer*), yang menegaskan bahwa setiap inisiatif teknologi informasi merupakan prioritas strategis korporasi.

Dalam struktur internal Dinas TD, terdapat pembagian fungsi yang jelas guna mendukung efektivitas pengelolaan sumber daya manusia:

a. LCU (*Learning Centre Unit*)

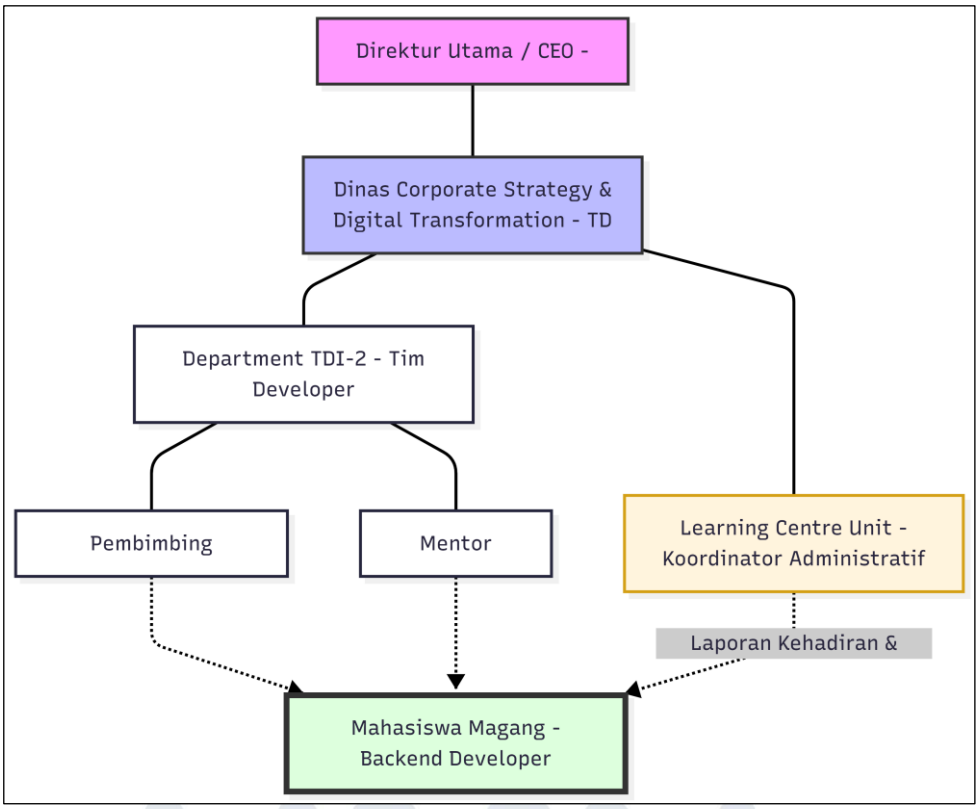
   Berfungsi sebagai unit pengelola administratif yang menjembatani hubungan antara mahasiswa magang dengan kebijakan korporat. LCU bertanggung jawab atas proses orientasi, pemantauan kehadiran harian, serta pemenuhan standar jam kerja yang telah ditetapkan oleh universitas dan perusahaan.

b. Unit TDI-2 (*Digitalization & System Development*):

   Merupakan unit teknis di mana mahasiswa magang ditempatkan secara operasional. Unit ini bertanggung jawab atas pengembangan, integrasi, dan pemeliharaan perangkat lunak internal. Fokus utama di unit ini adalah transformasi proses bisnis manual ke dalam ekosistem digital berbasis web dan aplikasi.

Kedudukan fungsional mahasiswa magang adalah sebagai *Backend Developer Intern* yang bertugas melakukan perancangan layanan API menggunakan *framework* NestJS dan pengelolaan basis data PostgreSQL.

Posisi ini berada di bawah supervisi Pembimbing Lapangan terkait manajemen beban kerja dan Mentor Teknis terkait kualitas kode serta arsitektur sistem seperti yang terlihat pada Gambar 3.1.



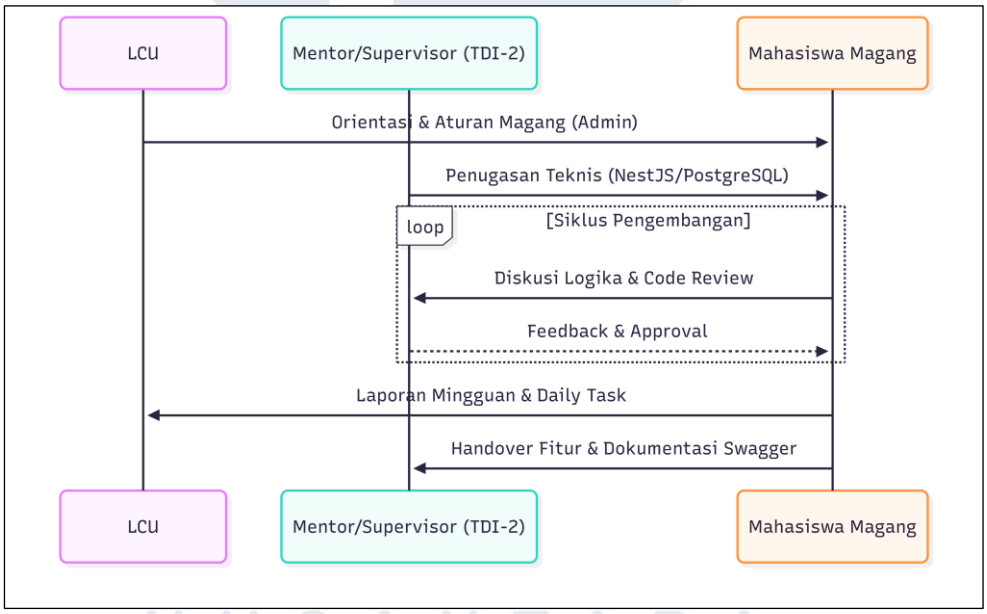
Gambar 3.1 Struktur Organisasi Unit Kerja

3.1.2 Koordinasi

Alur koordinasi pada Gambar 3.2 dirancang untuk memastikan setiap tahapan pengembangan sistem terdokumentasi dengan baik dan selaras dengan kebutuhan unit bisnis. Koordinasi dilakukan melalui dua jalur utama:

1. **Koordinasi Administratif** Dilakukan bersama pihak LCU untuk memastikan seluruh kewajiban administratif terpenuhi, termasuk pengisian *Daily Task* sebagai bukti aktivitas harian. Koordinasi ini memastikan bahwa mahasiswa magang memahami budaya kerja serta regulasi keselamatan kerja yang berlaku di lingkungan GMF AeroAsia.

2. **Koordinasi Teknis dan Operasional** Koordinasi ini dilakukan secara intensif di unit TDI-2 menggunakan beberapa kanal komunikasi profesional:
    a. ***Briefing* Rutin**: Dilakukan untuk menentukan prioritas fitur yang akan dikembangkan, seperti modul *Bizcase* atau optimalisasi tabel *Master*.
    b. **Diskusi Teknis dan *Code Review***: Dilakukan melalui pertemuan tatap muka maupun melalui fitur komentar pada repositori (Git/Bitbucket). Mentor teknis akan memberikan evaluasi terhadap efisiensi logika, penanganan *error*, serta keamanan API.
    c. **Validasi Dokumentasi**: Setiap *endpoint* API yang telah dibangun dikoordinasikan melalui platform Swagger. Hal ini dilakukan untuk memastikan tim *frontend* atau unit lain dapat mengonsumsi data API secara akurat sesuai kontrak data yang disepakati.



Gambar 3.2 Bagan Alur Koordinasi Pekerjaan

3.2 Tugas yang Dilakukan

Seluruh aktivitas pengerjaan selama masa magang didokumentasikan untuk memantau perkembangan proyek dan pemenuhan target kompetensi. Berikut adalah rincian tugas yang telah diselesaikan:

Tabel 3.1 Detail Pekerjaan yang Dilakukan

No.	Proyek	Uraian / Keterangan
1	**Proyek 1:** Pengembangan Arsitektur Data Terintegrasi	Inisiasi arsitektur modular NestJS, perancangan skema basis data modul Bizcase, serta pemetaan relasi entitas menggunakan Prisma ORM.
2	**Proyek 2:** Implementasi Sistem Validasi Keamanan melalui *Custom Decorator* (No-HTML)	Pengembangan lapisan keamanan input menggunakan *Regular Expression* (Regex) untuk mencegah serangan *Cross-Site Scripting* (XSS) pada sistem.
3	**Proyek 3:** Optimalisasi Dokumentasi API Interaktif dengan Swagger UI	Penyusunan dokumentasi teknis OpenAPI dan standardisasi kontrak data antar-modul guna mendukung kolaborasi tim pengembang.
4	**Proyek 4:** Pengembangan Logika Bisnis Modul Bizcase (*Financial Mapping*)	Implementasi layanan API untuk pengolahan data operasional dan finansial serta transformasi data dari DTO ke dalam basis data PostgreSQL.
5	**Proyek 5:** Implementasi *Database Transaction* dan *Audit Trail* (Update V3)	Penerapan prisma.$transaction untuk menjamin konsistensi data atomik serta otomatisasi pencatatan jejak perubahan data pada sistem versi 3 (V3).
6	**Proyek 6:** Analisis Hasil Pengujian Fungsional (*Black-box Testing*)	Validasi fungsionalitas seluruh *endpoint* API menggunakan Swagger UI dan Postman untuk memastikan integritas data serta penanganan *error*.

3.3 Uraian Pelaksanaan Kerja

Bagian ini memaparkan rincian teknis mengenai pengerjaan proyek pengembangan *backend* sistem manajemen proyek internal di PT GMF AeroAsia Tbk. Fokus pengerjaan dilakukan menggunakan *tech stack* utama berupa NestJS sebagai *framework* aplikasi, Prisma sebagai *Object-Relational Mapping* (ORM), dan PostgreSQL sebagai sistem manajemen basis data.

Setiap proyek dijalankan mengikuti siklus pengembangan perangkat lunak yang sistematis, mulai dari analisis kebutuhan, perancangan skema, hingga tahap pengujian fungsional.

3.3.1 Proses Pelaksanaan

Berikut adalah rincian enam proyek utama yang mencakup proses perancangan, implementasi, hingga tahap pengujian fungsional sistem. Tahap pengembangan sistem dilakukan secara menyeluruh mulai dari tingkat basis data. Fokus utama pengerjaan meliputi perancangan dan pembuatan modul-modul *database* untuk kebutuhan *Bizcase*, yang mencakup pendefinisian

skema relasional yang masif pada PostgreSQL melalui Prisma ORM. Selain pembangunan struktur data, dilakukan pula pendefinisian sistem validasi (*validator*) dari tahap awal pengembangan. Hal ini diwujudkan melalui pembuatan *custom decorator* @NoHtml untuk menjamin integritas dan keamanan setiap data yang masuk ke dalam sistem, memastikan bahwa seluruh *input* telah melewati proses filter keamanan sebelum disimpan ke dalam basis data.

3.3.1.1 Tahap Orientasi Teknis dan Pembelajaran Mandiri

Fase orientasi teknis dan adaptasi dijalankan pada awal masa pelaksanaan *Career Acceleration Program* di unit TDI-2 PT GMF AeroAsia Tbk sebagai persiapan sebelum terlibat langsung dalam pengerjaan proyek sistem manajemen proyek internal. Pada tahap ini, pengarahan diberikan oleh Pembimbing Lapangan untuk melakukan pembelajaran mandiri melalui materi tutorial pada platform YouTube. Materi yang dipelajari secara spesifik mencakup konsep-konsep dasar pengembangan *backend* serta *tech stack* yang relevan dengan kebutuhan operasional perusahaan.

Sebagai bentuk validasi terhadap hasil pembelajaran mandiri tersebut, sesi evaluasi berkala dilaksanakan bersama mentor. Sesi ini dilakukan melalui tanya jawab interaktif guna mendiskusikan tingkat pemahaman terkait materi video yang telah ditonton. Proses tersebut bertujuan untuk memastikan kesiapan landasan teoretis yang kuat sebelum dilakukan implementasi logika pemrograman yang lebih kompleks pada modul Bizcase maupun pengembangan *Project* versi 3 (V3).

3.3.1.2 Proyek 1: Pengembangan Arsitektur Data Terintegrasi

Proyek pertama ini merupakan fondasi paling krusial dalam pengembangan sistem manajemen proyek internal di unit TDI-2 PT GMF AeroAsia Tbk. Pengembangan ini difokuskan pada perancangan dan implementasi arsitektur data untuk modul *Bizcase*, yang

merupakan inti dari logika bisnis sistem untuk mengelola kelayakan proyek dari sisi teknis dan finansial.

1. **Analisis Relasi Data Analisis dan Perancangan Skema Basis Data (Prisma ORM)**

   Mahasiswa magang melakukan perancangan skema basis data yang melibatkan relasi *one-to-many* antara tabel master dengan sub-entitas pada modul *Bizcase*. Sub-entitas ini mencakup rincian teknis seperti *Bizcase Efficiency, Bizcase Migration,* serta pemetaan kebutuhan infrastruktur (*Infra Needs*). Perancangan ini sangat krusial agar setiap data finansial dan efisiensi yang diinput dapat terelasi secara akurat ke entitas utama proyek.

   Tahap awal pengerjaan dimulai dengan perancangan skema basis data relasional menggunakan Prisma ORM. Mahasiswa magang melakukan pemetaan kebutuhan bisnis ke dalam bentuk model objek pada file schema.prisma. Model Bizcase dirancang untuk memiliki struktur data yang sangat kompleks karena harus mampu mengakomodasi berbagai parameter proyek yang bersifat dinamis.

   Dalam perancangannya, model ini dilengkapi dengan atribut id sebagai *primary key* berbasis *autoincrement* serta uniqueId yang menggunakan tipe data String dengan generator gen_random_uuid() untuk keamanan akses API. Selain itu, diimplementasikan kolom audit otomatis seperti createdAt dan updatedAt guna memenuhi standar kepatuhan (*compliance*) perusahaan. Cuplikan kode skema basis data tersebut dapat dilihat pada Gambar 3.3 di bawah ini.

```prisma
model Bizcase {
  id                        Int       @id @default(autoincrement())
  uniqueId                  String    @unique @map("unique_id") @default(dbgenerated("gen_random_uuid()")) @db.Uuid
  projectTypeId             Int       @unique @map("project_type_id")
  bsSummary                 String    @map("bs_summary") @db.VarChar(255)
  sFlowProcess              String?   @map("s_flow_process") @db.VarChar(255)
  sUseCase                  String?   @map("s_use_case") @db.VarChar(255)
  sIntegrationDiagram       String?   @map("s_integration_diagram") @db.VarChar(255)
  sModuleApp                String?   @map("s_module_app") @db.VarChar(255)
  sMockup                   String?   @map("s_mockup") @db.VarChar(255)
  sTechnicalSpecification   String?   @map("s_technical_specification") @db.VarChar(255)
  sDataCenter               String?   @map("s_data_center") @db.VarChar(255)
  sNetworkDesign            String?   @map("s_network_design") @db.VarChar(255)
  sDevicePeripheralDesign   String?   @map("s_device_peripheral_design") @db.VarChar(255)
  sScopeOfWork              String    @map("s_scope_of_work") @db.VarChar(255)
  ssServiceDescription      String    @map("ss_service_description") @db.VarChar(255)
  ssLevel                   String    @map("ss_level") @db.VarChar(255)
  ssAvailability            String    @map("ss_availability") @db.VarChar(255)
  ssSecurityDesign          String    @map("ss_security_design") @db.VarChar(255)
  cRevenue                  String?   @map("c_revenue") @db.VarChar(255)
  cOperatingCost            String?   @map("c_operating_cost") @db.VarChar(255)
  cBreakEventPoint          String?   @map("c_break_event_point") @db.VarChar(255)
  epImplementationDetail    String    @map("ep_implementation_detail") @db.VarChar(255)
  epImplementedBy           String    @map("ep_implemented_by") @db.VarChar(255)
  epManageServiceDetail     String    @map("ep_manage_service_detail") @db.VarChar(255)
  epManagedServiceBy        String    @map("ep_managed_service_by") @db.VarChar(255)
  epDuration                Int       @map("ep_duration")
  epNotes                   String?   @map("ep_notes") @db.VarChar(255)

  createdAt                 DateTime  @default(now()) @map("created_at") @db.Timestamp(6)
  updatedAt                 DateTime  @updatedAt @map("updated_at") @db.Timestamp(6)
  createdBy                 String    @map("created_by") @db.VarChar(50)
  updatedBy                 String?   @map("updated_by") @db.VarChar(50)

  // Relations
  bcInfraNeeds          BcInfraNeeds[]
  bcServiceCategory     BcServiceCategory[]
  bcUpgradePatching     BcUpgradePatching[]
  bcCostManageService   BcCostManageService[]
  bcCostImplementation  BcCostImplementation[]
  bcRisk                BcRisk[]
  bcTestingScope        BcTestingScope[]
  bcDataMigration       BcDataMigration[]
  bcActivity            BcActivity[]
  bcCostEfficiency      BcCostEfficiency[]

  projectType           ProjectType @relation(fields: [projectTypeId], references: [id])

  @@map("bizcase")
}
```

Gambar 3.3 schema.prisma model Bizcase

2. **Arsitektur Relasi Entitas dan Sub-Entitas**

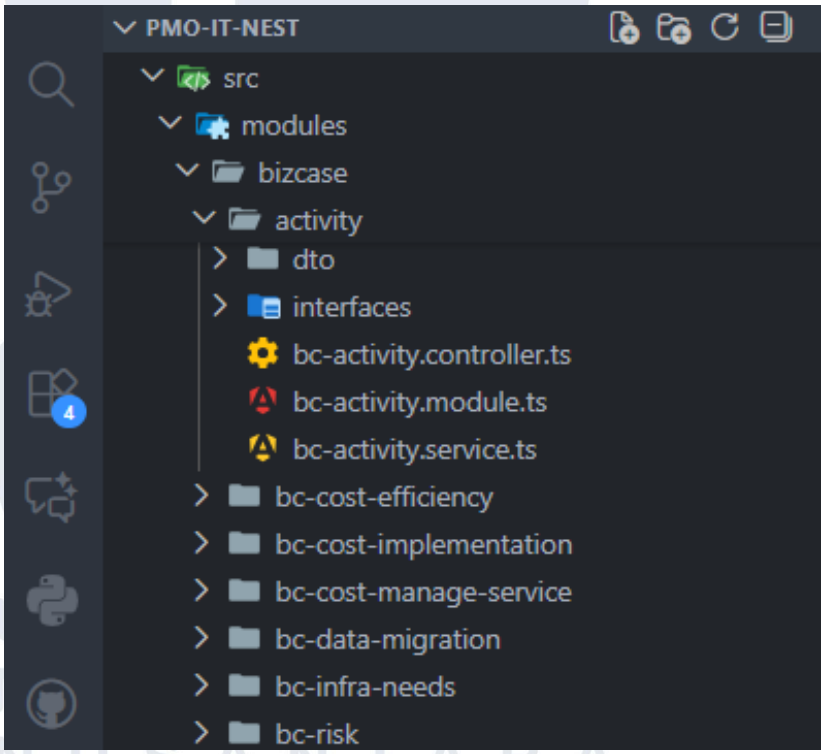
Karakteristik utama dari proyek ini adalah penanganan relasi *one-to-many* yang sangat masif antara entitas Bizcase dengan berbagai sub-entitas operasionalnya. Berdasarkan rancangan teknis pada Gambar 3.3, model Bizcase bertindak sebagai entitas induk yang menaungi berbagai sub-entitas seperti bcInfraNeeds untuk kebutuhan infrastruktur, bcRisk untuk analisis risiko, hingga bcCostEfficiency untuk pengolahan data efisiensi biaya.

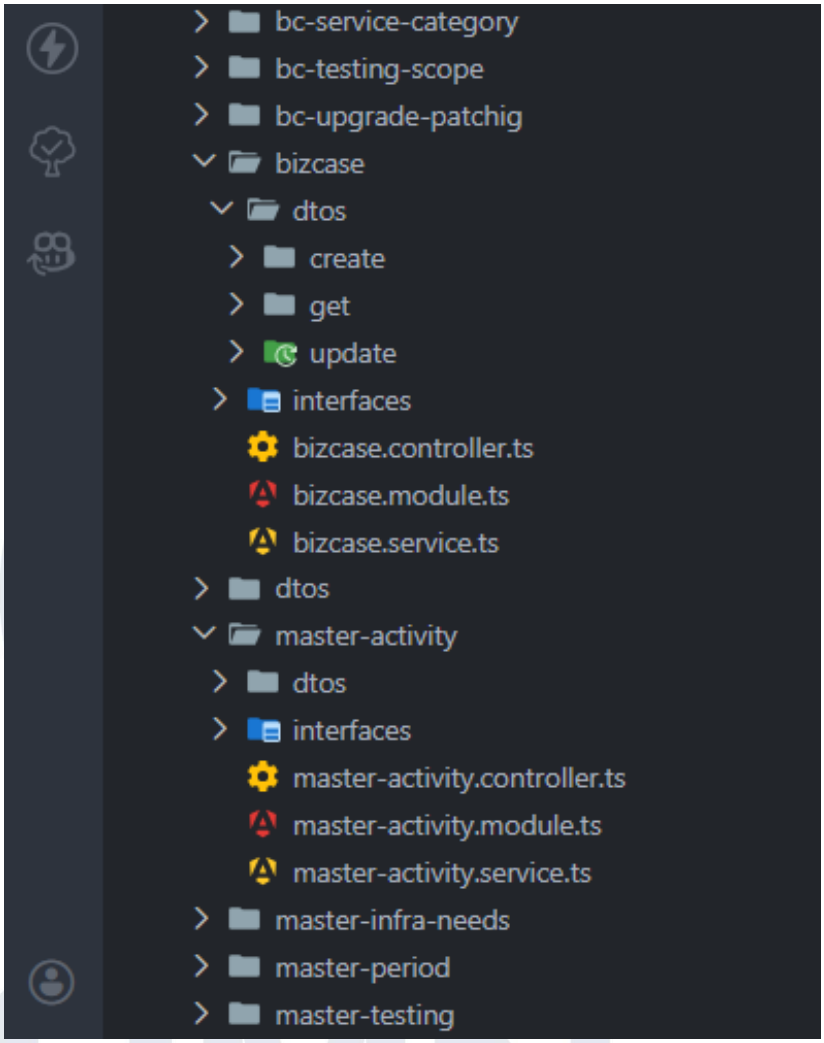
Setiap relasi didefinisikan secara eksplisit menggunakan tipe data array (misal: BcInfraNeeds[]), yang memungkinkan Prisma

Client untuk melakukan *nested query* secara efisien. Integrasi ini memastikan bahwa saat pengguna melakukan pembaruan pada data utama, seluruh data pada entitas anak tetap sinkron dan terjaga integritasnya.

3. **Implementasi Struktur Folder Modular pada NestJS**

Guna mendukung skalabilitas sistem, mahasiswa magang menerapkan pola organisasi kode berbasis modular. Di dalam direktori src/modules/bizcase, kode program dipecah menjadi beberapa sub-modul yang merepresentasikan setiap sub-entitas yang ada pada skema basis data. Struktur folder modular ini secara visual dipaparkan dalam gambar berikut.





Gambar 3.4 Struktur Folder Modular *src/modules/bizcase*

Penerapan pola pada Gambar 3.4 dirancang agar setiap komponen fungsional memiliki tanggung jawab tunggal (*Single Responsibility Principle*). Sebagai contoh, modul activity di dalam bizcase memiliki sub-direktori dtos untuk validasi data masukan, interfaces untuk pendefinisian kontrak data, serta file *controller*, *module*, dan *service* yang berdiri sendiri. Pola ini mempermudah tim pengembang di TDI-2 dalam melakukan *debugging* dan pemeliharaan kode tanpa mengganggu fungsionalitas modul lainnya.

4. **Manajemen Dependensi melalui Registry Module**

Tahap akhir dari Proyek 1 adalah melakukan registrasi seluruh komponen ke dalam BizcaseModule. File bizcase.module.ts berperan sebagai pengatur lalu lintas dependensi menggunakan teknik *Dependency Injection*.

Gambar 3.5 *bizcase.module.ts*

Berdasarkan implementasi pada Gambar 3.5, BizcaseService dan PrismaService didaftarkan di dalam *array providers* agar dapat diinjeksikan dan digunakan di seluruh lingkup modul Bizcase. Selain itu, BizcaseController didaftarkan pada *array controllers* untuk mengekspos *endpoint* API ke publik, sementara BizcaseService dimasukkan ke dalam *array exports* agar fungsinya dapat diakses oleh modul lain di luar lingkup *Bizcase* jika diperlukan integrasi lintas modul di masa mendatang.

3.3.1.3 Proyek 2: Implementasi Sistem Validasi Keamanan melalui *Custom Decorator (No-HTML)*

Proyek kedua difokuskan pada penguatan lapisan keamanan aplikasi dari sisi *input* pengguna. Dalam lingkungan industri penerbangan seperti PT GMF AeroAsia Tbk, integritas data adalah hal yang mutlak. Ancaman serangan siber seperti *Cross-Site Scripting* (XSS) menjadi perhatian utama, di mana penyerang dapat mencoba memasukkan skrip berbahaya melalui kolom *input* teks yang kemudian dapat tereksekusi di sisi peramban pengguna lain.

1. **Analisis Risiko dan Perancangan Keamanan**

Pada modul *Bizcase* dan *Master*, terdapat banyak kolom bertipe *string* yang memungkinkan pengguna memasukkan deskripsi panjang. Tanpa validasi yang ketat, kolom-kolom ini rentan disalahgunakan untuk menyisipkan tag HTML atau tag <script>. Meskipun *library* validasi standar menyediakan pengecekan tipe data, diperlukan logika tambahan yang spesifik untuk mendeteksi pola karakter yang menyerupai struktur HTML.

Oleh karena itu, dirancang sebuah *Custom Decorator* bernama @NoHtml. Keputusan menggunakan *custom decorator* diambil agar logika validasi ini bersifat *reusable* (dapat digunakan kembali) di seluruh DTO (*Data Transfer Object*) dalam aplikasi tanpa perlu menulis ulang logika pengecekan di setiap fungsi *service*.

2. **Implementasi Logika Validator (Custom Decorator)**

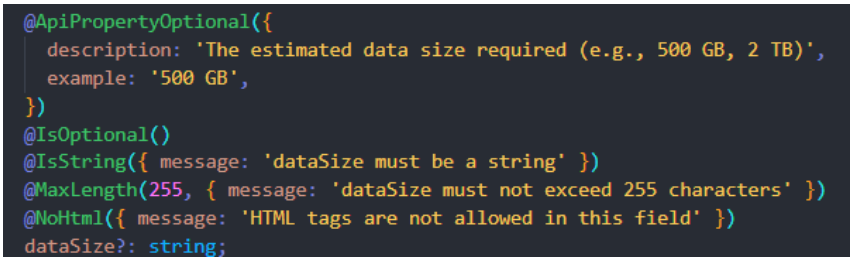
Implementasi dilakukan dengan memanfaatkan fungsi registerDecorator dari pustaka class-validator. Mahasiswa magang menyusun fungsi NoHtml yang menerima parameter ValidationOptions. Inti dari validator ini terletak pada penggunaan *Regular Expression* (Regex) untuk mendeteksi keberadaan tag skrip.

Gambar 3.6 *no-html.decorator.ts*

Berdasarkan Gambar 3.6, logika validate akan mengembalikan nilai true jika *input* bukan merupakan *string* atau jika *input* tidak mengandung pola yang didefinisikan dalam scriptPattern. Jika pola terdeteksi, maka validator akan mengembalikan pesan kesalahan standar, yaitu *"HTML tags are not allowed in this field"*, yang juga dapat dikustomisasi melalui properti defaultMessage.

3. **Penerapan pada *Data Transfer Object* (DTO)**

Setelah decorator berhasil dibangun, tahap selanjutnya adalah mengimplementasikannya pada kontrak data sistem. Salah satu contoh penerapannya adalah pada modul *Bizcase Infra Needs*, khususnya pada file *bc_infra_need.dto.ts*. Dekorator @NoHtml diletakkan bersamaan dengan dekorator validasi lainnya untuk memastikan pemeriksaan berlapis.



Gambar 3.7 Penerapan *Decorator* pada *bc_infra_need.dto.ts*

Pada Gambar 3.7, atribut dataSize memiliki aturan validasi yang sangat ketat. Selain harus berupa *string* dengan panjang maksimal 255 karakter, atribut ini wajib melewati pengecekan @NoHtml. Jika seorang pengguna mencoba memasukkan nilai seperti 500 GB *<script>alert(1)</script>,* maka sistem secara otomatis akan menolak permintaan tersebut sebelum data mencapai lapisan *service* atau basis data, sehingga keamanan sistem tetap terjaga secara preventif.

4. **Integrasi dengan NestJS ValidationPipe**

Seluruh sistem validasi ini terintegrasi secara otomatis dengan ValidationPipe global pada NestJS. Hal ini memastikan bahwa setiap kali terjadi permintaan HTTP (*POST* atau *PUT*), sistem akan memvalidasi *payload* berdasarkan dekorator yang ada pada DTO terkait. Pendekatan ini tidak hanya meningkatkan keamanan, tetapi juga memastikan pesan kesalahan yang dikembalikan ke pengguna (melalui *response body*) bersifat informatif dan konsisten sesuai dengan standar API yang dikembangkan di TDI-2.

3.3.1.4 Proyek 3: Optimalisasi Dokumentasi API Interaktif dengan Swagger UI

Proyek ketiga berfokus pada penyediaan infrastruktur dokumentasi teknis yang komprehensif untuk seluruh layanan API di dalam sistem manajemen proyek. Dalam lingkungan kerja yang

kolaboratif seperti pada unit TDI-2, sinkronisasi antar-pengembang sangat bergantung pada ketersediaan kontrak data yang jelas dan mudah diakses.

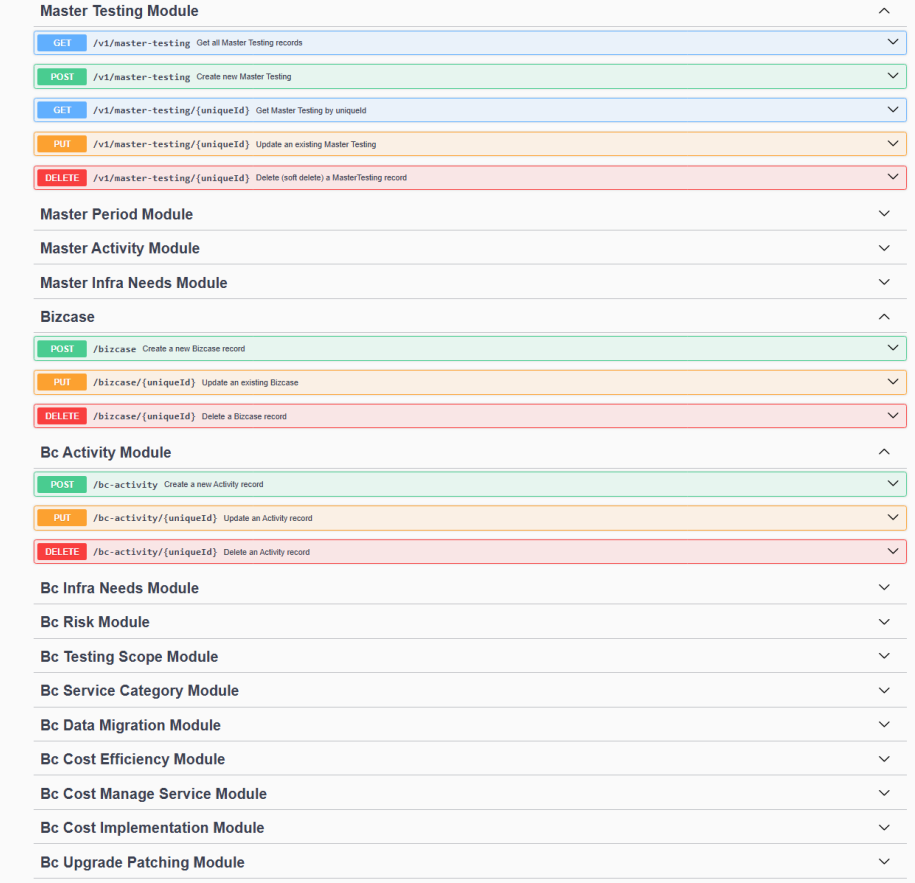
1. **Analisis Kebutuhan Kolaborasi dan Standardisasi**

   Sebelum adanya dokumentasi interaktif, proses integrasi antara bagian *backend* dan *frontend* sering kali terhambat oleh perbedaan pemahaman mengenai struktur data dan parameter *request*. Oleh karena itu, diimplementasikan standar OpenAPI menggunakan Swagger UI. Strategi ini memungkinkan sistem untuk menghasilkan dokumentasi secara otomatis dari kode sumber, sehingga dokumentasi selalu sinkron dengan versi aplikasi terbaru yang sedang dikembangkan.

2. **Implementasi Dekorator dan Pemetaan Modul**

   Proses integrasi dilakukan dengan menyematkan berbagai dekorator Swagger pada lapisan *Controller* dan *Data Transfer Object* (DTO). Mahasiswa magang mengelompokkan setiap modul fungsional menggunakan dekorator @ApiTags agar tampilan dokumentasi menjadi terorganisir dengan baik.

   Penyusunan dilakukan secara mendetail dengan memberikan informasi mengenai tujuan setiap *endpoint* melalui @ApiOperation, serta mendefinisikan berbagai kemungkinan respon HTTP (seperti 200 *OK*, 201 *Created*, 400 *Bad Request*, hingga 404 *Not Found*) melalui dekorator @ApiResponse. Hasil dari kategorisasi modul tersebut secara visual dipaparkan dalam gambar 3.8 berikut.

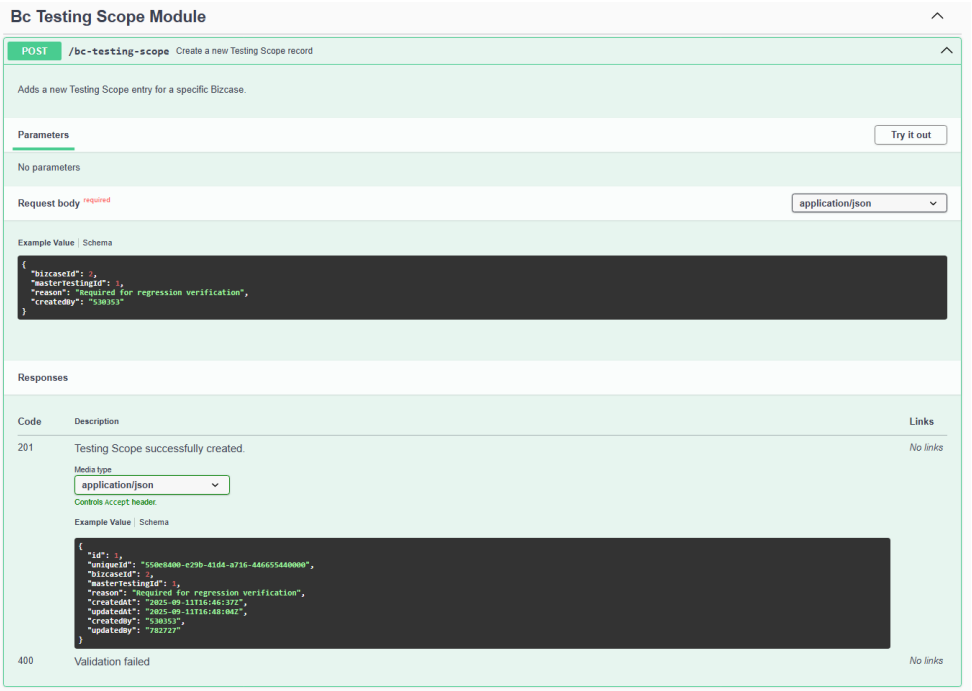


Gambar 3.8 Dashboard Utama Swagger UI

3. Visualisasi Kontrak Data dan Pengujian Fungsional (Sandbox)

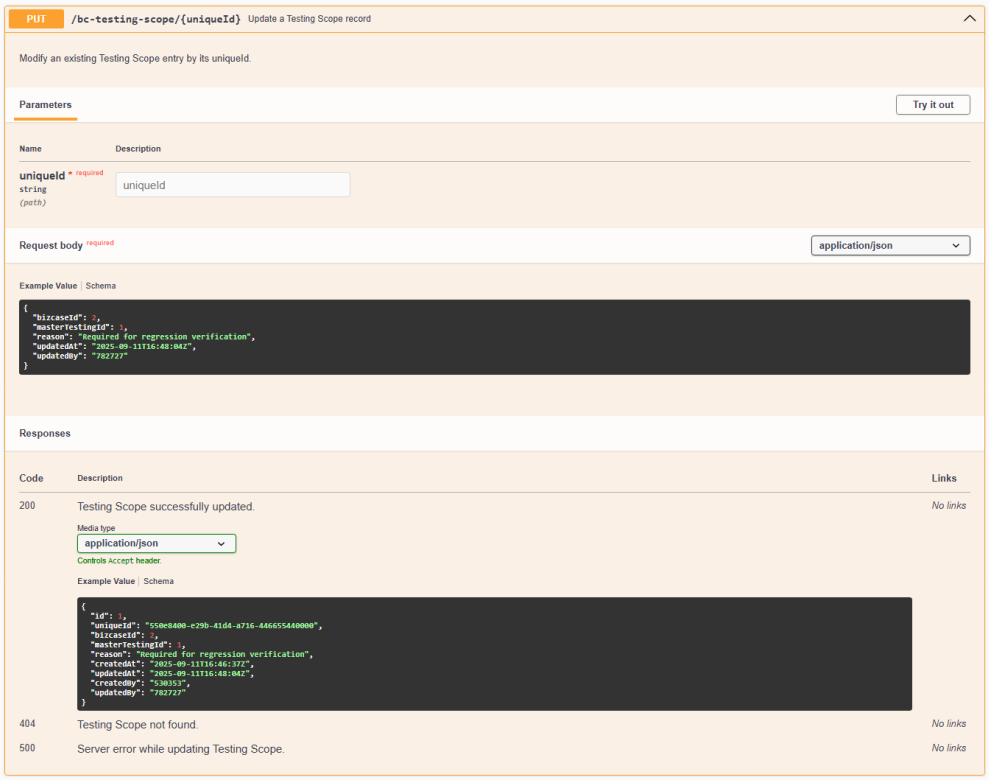
Salah satu keunggulan utama dari implementasi ini adalah fitur pengujian langsung (*sandbox*) yang tersedia di dalam portal Swagger. Pengembang *frontend* dapat melihat struktur *request body* yang dibutuhkan serta contoh data (*Example Value*) tanpa perlu membuka kode program.

Sebagai contoh, pada modul *Bc Testing Scope* dalam Gambar3.9, disediakan dokumentasi lengkap untuk operasi pembuatan data baru (*POST*). Dokumentasi ini merinci atribut apa saja yang wajib dikirimkan, seperti bizcaseId, masterTestingId, dan reason.

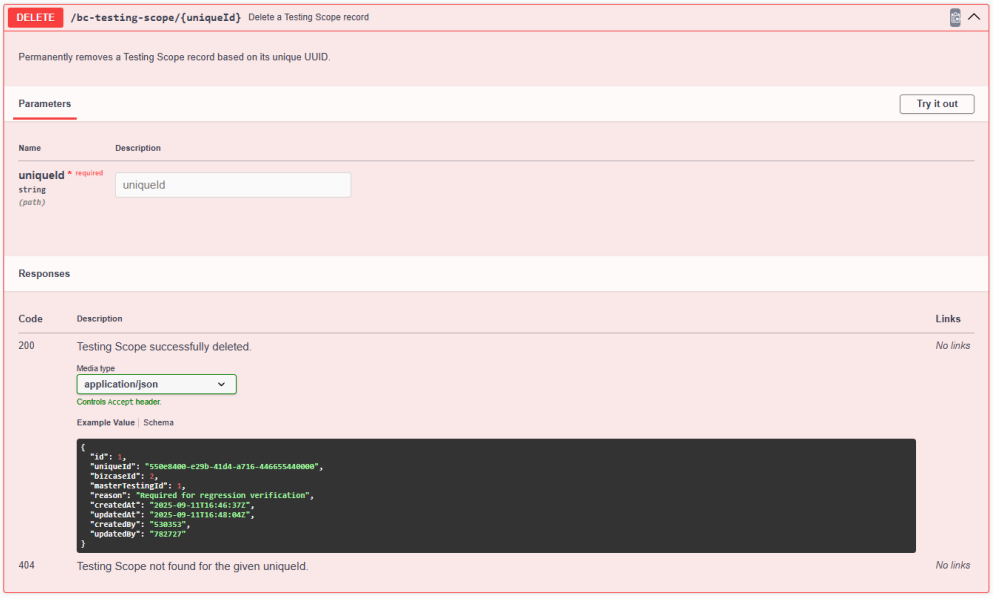


Gambar 3.9 API POST untuk Modul *Bc Testing Scope*

Selain operasi pembuatan data, portal Swagger juga mendokumentasikan proses pembaruan (*PUT*) dan penghapusan data (*DELETE*) secara spesifik menggunakan parameter uniqueId berbasis UUID. Hal ini memastikan bahwa pengembang lain memahami cara melakukan manipulasi data pada catatan tertentu secara tepat dan dapat dilihat pada Gambar 3.10 dan Gambar 3.11.



Gambar 3.10 Dokumentasi API PUT untuk Modul Bc Testing Scope



Gambar 3.11 Dokumentasi API DELETE untuk Modul Bc Testing Scope

4. **Validasi Respon dan Penanganan Error**

Swagger UI juga dimanfaatkan untuk memvalidasi apakah penanganan kesalahan (*error handling*) sudah bekerja sesuai ekspektasi. Berdasarkan Gambar 3.10, portal dokumentasi tidak hanya menampilkan respon sukses, tetapi juga skema respon jika data tidak ditemukan (404 *Not Found*) atau jika validasi gagal (400 *Validation failed*). Dengan adanya informasi ini, tim *frontend* dapat membangun logika penanganan *error* pada antarmuka pengguna secara lebih presisi, sehingga meningkatkan kualitas pengalaman pengguna secara keseluruhan di sistem GMF AeroAsia.

3.3.1.5 Proyek 4: Pengembangan Logika Bisnis Modul Bizcase (Financial Mapping)

Proyek keempat merupakan inti dari fungsionalitas sistem manajemen proyek internal di PT GMF AeroAsia Tbk. Pada tahap ini, mahasiswa magang bertanggung jawab mengembangkan logika bisnis yang kompleks untuk menangani data operasional dan finansial pada modul *Bizcase*. Pengerjaan dilakukan dengan memisahkan tanggung jawab antara *Entry Point* (Controller) dan Logika Bisnis (Service) sesuai dengan arsitektur NestJS guna memastikan kode mudah dirawat (*maintainable*) dan diuji.

1. **Implementasi Arsitektur API pada Lapisan *Controller***

Pengerjaan diawali dengan menyediakan *endpoint* API melalui BizcaseController. Lapisan ini bertugas mengatur jalur lalu lintas permintaan HTTP serta melakukan pengemasan data (*data wrapping*) agar sesuai dengan standar kontrak API yang telah disepakati bersama tim pengembang lain.

Mahasiswa magang mengimplementasikan berbagai dekorator Swagger untuk meningkatkan kualitas dokumentasi

teknis. Sebagai contoh, dekorator @ApiOperation digunakan untuk memberikan ringkasan fungsi secara eksplisit kepada pengguna API, sementara @ApiBody dengan referensi BizcaseCreateDTO menjamin bahwa *payload* data yang dikirimkan oleh sistem *frontend* telah melewati proses validasi tipe data sebelum masuk ke tahap pemrosesan logika.

```
@ApiTags('Bizcase')
@Controller('bizcase')
export class BizcaseController {
  constructor(private readonly bizcaseService: BizcaseService) {}
 @Post()
  @HttpCode(HttpStatus.CREATED)
  @ApiOperation({
    summary: 'Create a new Bizcase record',
    description:
      'Add a new Bizcase entry with required project and case details.',
  })
  @ApiBody({ type: BizcaseCreateDTO })
  @ApiResponse({
    status: 201,
    description: 'Bizcase successfully created.',
    type: BizcaseDTO,
  })
  @ApiBadRequestResponse({
    description: 'Invalid request payload or missing required fields.',
  })
  @ApiInternalServerErrorResponse({
    description: 'Server error during Bizcase creation.',
  })
  async createBizcase(@Body() body: BizcaseCreateDTO): Promise<BizcaseDTO> {
    return await this.bizcaseService.createBizcase(undefined, body);
  }

```

Gambar 3.12 Cuplikan Kode bizcase.controller.ts untuk Method POST

Berdasarkan Gambar 3.12, terlihat bahwa setiap fungsi dalam *controller* bersifat asinkronus (async) yang mengembalikan objek Promise. Hal ini sangat penting untuk menjaga performa aplikasi agar tetap responsif, terutama saat menangani operasi basis data yang masif. Penggunaan status kode HttpStatus.CREATED (201) pada *endpoint* pembuatan data baru memberikan respon yang standar secara industri, yang menandakan bahwa catatan *Bizcase* berhasil dibentuk di dalam sistem.

2. **Pengembangan Logika Bisnis dan Transformasi Data pada Lapisan *Service***

Seluruh inti dari pemrosesan data diletakkan di dalam BizcaseService. Lapisan ini mengelola integrasi antara model Bizcase utama dengan berbagai sub-entitas teknisnya. Dalam proses pengembangan, mahasiswa magang merancang fungsi yang mampu melakukan transformasi data secara dinamis dari format *Data Transfer Object* (DTO) ke dalam skema basis data PostgreSQL melalui Prisma ORM.

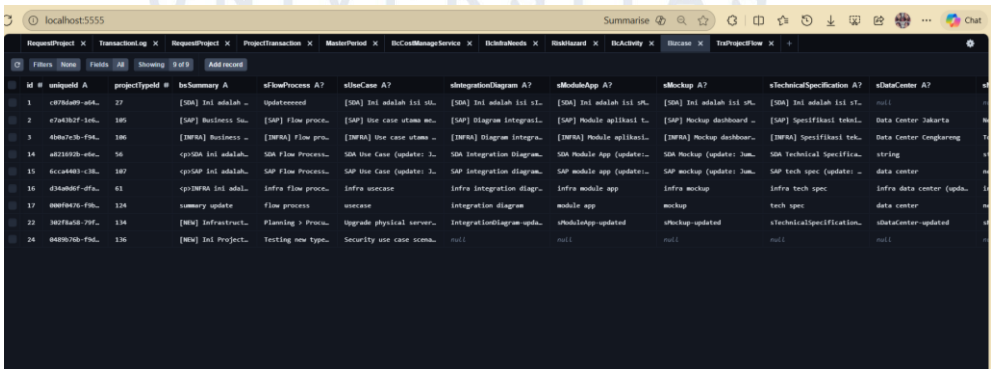
Salah satu aspek yang paling teknis dalam pengerjaan ini adalah penanganan data *nested* (bertingkat). Saat sebuah catatan *Bizcase* dibuat, *service* harus memastikan bahwa identitas unik (uniqueId) berbasis UUID dihasilkan secara otomatis melalui fungsi gen_random_uuid() di tingkat basis data guna memitigasi risiko keamanan akses data. Selain itu, dilakukan pemetaan terhadap variabel operasional seperti sFlowProcess dan sUseCase agar data tersimpan dengan integritas referensial yang kuat. Dokumentasi dapat dilihat pada Gambar 3.13.

```typescript
@Injectable()
export class BizcaseService implements BizcaseServiceInterface {
  constructor(private readonly prisma: PrismaService) {}

  // Get all Bizcase records.
  async getAllBizcase(): Promise<BizcaseDTO[]> {
    try {
      const result = await this.prisma.bizcase.findMany({
        orderBy: { createdAt: 'desc' },
      });
      return result as BizcaseDTO[];
    } catch (error) {
      throw new BadRequestException(
        `Failed to fetch bizcases: ${error.message}`,
      );
    }
  }

  // Find a single Bizcase by its uniqueId (UUID).
  async findBizcase(uniqueId: string): Promise<BizcaseDTO> {
    const bizcase = await this.prisma.bizcase.findUnique({
      where: { uniqueId },
    });

    if (!bizcase) {
      throw new NotFoundException(
        `Bizcase with uniqueId '${uniqueId}' not found`,
      );
    }

    return bizcase as BizcaseDTO;
  }
```

Gambar 3.13 Cuplikan Logika Pemrosesan Data pada bizcase.service.ts

3. Verifikasi Integritas Data Melalui Prisma Studio

Setelah logika bisnis berhasil diimplementasikan, tahap selanjutnya adalah melakukan verifikasi data secara langsung pada tingkat fisik basis data. Mahasiswa magang menggunakan Prisma Studio sebagai alat inspeksi data *real-time* guna memastikan bahwa setiap atribut finansial dan operasional telah tersimpan pada kolom yang tepat di PostgreSQL.



Gambar 3.14 Data Modul Bizcase pada Prisma Studio

Melalui Gambar 3.14, dapat divalidasi bahwa sistem secara otomatis mengelola kolom audit seperti createdAt dan updatedAt. Terdapat pula kolom projectTypeId yang menunjukkan keberhasilan integrasi relasi antar-tabel yang sebelumnya telah dirancang pada tahap inisiasi proyek. Keberadaan data pada kolom operasional seperti bsSummary dan sModuleApp menunjukkan bahwa alur koordinasi data dari pengguna hingga ke basis data telah berjalan sesuai dengan spesifikasi teknis yang ditetapkan oleh unit TDI-2 PT GMF AeroAsia Tbk.

3.3.1.6 Proyek 5: Implementasi Database Transaction dan Audit Trail (Update V3)

Proyek kelima berfokus pada penguatan keandalan sistem saat melakukan manipulasi data yang kompleks pada modul *Project* dan *Bizcase* versi 3 (V3). Pada tahap ini, mahasiswa magang mengimplementasikan mekanisme *Database Transaction* untuk menjamin konsistensi data serta sistem *Audit Trail* guna mencatat setiap aktivitas perubahan data secara transparan.

1. **Urgensi dan Mekanisme *Database Transaction***

   Dalam sistem manajemen proyek di PT GMF AeroAsia Tbk, satu aksi pembaruan (*update*) sering kali melibatkan perubahan pada beberapa tabel yang saling berelasi secara bersamaan. Tanpa mekanisme transaksi, terdapat risiko di mana salah satu tabel berhasil diperbarui namun tabel lainnya gagal akibat kendala teknis, yang akan menyebabkan ketidakkonsistenan data (*data anomaly*).

   Untuk memitigasi risiko tersebut, mahasiswa magang menerapkan fitur prisma.$transaction. Mekanisme ini memastikan prinsip **Atomisitas**, di mana serangkaian operasi basis data dianggap sebagai satu kesatuan tunggal; jika salah satu operasi

gagal, maka seluruh rangkaian operasi akan dibatalkan (*rollback*) dan basis data kembali ke kondisi semula.

```
async updateProjectV3(
  data: ProjectV3UpdateDTO,
  uniqueId: string,
): Promise<RequestProject> {
  console.log('updateProjectV3() called for', uniqueId);

  let updatedUniqueId: string | null = null;

  updatedUniqueId = await this.prisma.$transaction(
    async (trx: Prisma.TransactionClient) => {
      // find project
      const existingProject = await trx.requestProject.findUnique({
        where: { uniqueId },
        include: { projectScale: true },
      });

      return updatedProject.uniqueId;
    },
    { timeout: 30000 },
  );

  if (!updatedUniqueId) throw new Error('Update did not complete');

  // return
  return await this.findProjectV3Internal(this.prisma, updatedUniqueId);
}
```

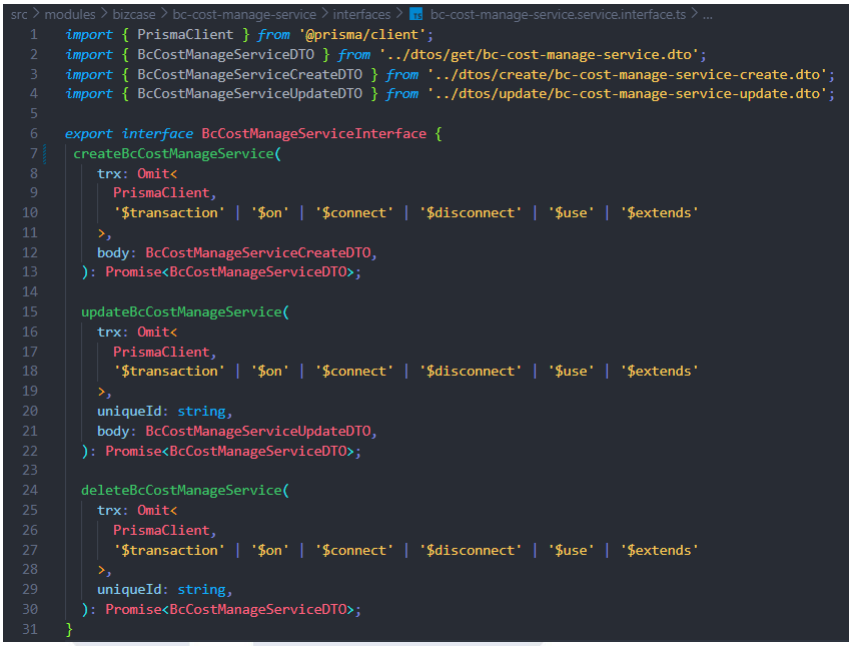
Gambar 3.15 Implementasi $transaction pada project.service.ts

Berdasarkan Gambar 3.15, terlihat bahwa sebelum melakukan pembuatan data transaksi proyek yang baru, sistem terlebih dahulu menghapus data transaksi lama di dalam blok transaksi yang sama. Hal ini memastikan tidak terjadi duplikasi data atau sisa data lama yang tidak valid di dalam PostgreSQL.

2. **Standarisasi *Transaction Client* pada *Interface***

Guna mendukung keterhubungan antar-layanan (*cross-service communication*), mahasiswa magang merancang *interface* layanan yang mendukung parameter transaksi opsional (trx). Teknik ini memungkinkan sebuah modul untuk membagikan *instance* transaksi yang sama ke modul lain, sehingga seluruh

operasi di berbagai modul tetap berada dalam satu siklus transaksi yang sinkron.



Gambar 3.16 Pendefinisian Interface dengan Parameter trx

Penerapan pada Gambar 3.16 menunjukkan profesionalisme koding, di mana mahasiswa magang membatasi akses PrismaClient menggunakan fungsi Omit agar fungsi-fungsi sistemik seperti $connect atau $disconnect tidak dapat dipanggil secara tidak sengaja di tengah proses transaksi, yang dapat menyebabkan pemutusan koneksi secara mendadak.

3. **Implementasi Otomatisasi *Audit Trail***

Selain aspek transaksi, Proyek 5 juga mencakup implementasi sistem jejak audit (*audit trail*). Hal ini diwujudkan melalui pendefinisian kolom audit pada skema basis data yang secara otomatis mencatat identitas pengguna dan waktu perubahan.

Setiap entitas, termasuk entitas *Bizcase*, dilengkapi dengan atribut updatedAt yang menggunakan dekorator @updatedAt

serta atribut updatedBy yang menyimpan ID pengguna yang melakukan perubahan terakhir. Implementasi ini sangat krusial bagi GMF AeroAsia untuk kebutuhan kepatuhan (*compliance*) dan transparansi operasional, sehingga setiap perubahan pada parameter proyek dapat dilacak kembali jika terjadi anomali di masa depan.

4. **Validasi Pengambilan Data Kompleks (*Find Logic*)**

Untuk memastikan seluruh data yang telah diproses secara aman melalui transaksi dapat disajikan kembali dengan akurat, mahasiswa magang mengembangkan fungsi pengambilan data (find) yang mendalam. Fungsi ini memanfaatkan fitur include pada Prisma untuk mengambil data dari berbagai tabel relasi dalam satu kali kueri (*single fetch*).

```
async findProjectV3(uniqueId: string) {
    return await this.prisma.$transaction(async (trx) => {
      const project = await trx.requestProject.findUnique({
        where: { uniqueId },
        include: {
          masterRequestType: {
            include: { masterStatus: true },
          },
          document: {
            include: { documentType: true },
          },
          progressLog: true,
          projectTransaction: {
            include: { transactionLog: true },
          },
          referenceProcedure: true,
          riskHazard: true,
          remarkLog: {
            orderBy: { createdAt: 'asc' },
          },
          // nested: projectScale -> projectType -> bizcase -> all sub tables
          projectScale: {
            include: {
              projectType: {
                include: {
                  masterProjectType: true,
                  assignment: true,
                  fillByIt: {
                    include: { fillByItDetail: true },
                  },
                  bizcase: {
                    include: {
                      projectType: {
                        include: { masterProjectType: true },
                      },
                      bcInfraNeeds: {
                        include: { masterInfraNeeds: true },
                        orderBy: { id: 'asc' },
                      },
                      bcServiceCategory: true,
                      bcUpgradePatching: {
                        include: { masterPeriod: true },
                      },
                      bcDataMigration: true,
                      bcTestingScope: {
                        include: { masterTesting: true },
```

Gambar 3.17 Cuplikan Fungsi findAll dengan Nested Include

Melalui implementasi pada Gambar 3.17, sistem dapat menyajikan informasi proyek secara utuh, mulai dari detail tipe proyek hingga rincian *business case* terkait, tanpa perlu melakukan pemanggilan API berulang kali. Integrasi antara mekanisme transaksi yang aman dan pengambilan data yang efisien ini menjadi standar kualitas pengembangan aplikasi yang diterapkan selama masa magang di TDI-2.

3.3.1.7 Proyek 6: Analisis Hasil Pengujian Fungsional (Black-box Testing)

Setelah seluruh proses implementasi logika bisnis dan integrasi basis data pada modul manajemen proyek (V3) selesai dilakukan, tahap selanjutnya adalah pengujian fungsional sistem. Tahap ini bertujuan untuk memastikan bahwa setiap endpoint API yang dikembangkan telah berfungsi sesuai dengan spesifikasi kebutuhan sistem dan kontrak data (*data contract*) yang telah ditetapkan. Metode pengujian yang digunakan pada proyek ini adalah

Black-box Testing, yaitu metode pengujian yang berfokus pada kesesuaian input dan output sistem tanpa meninjau struktur kode internal secara langsung.

Pengujian fungsional ini difokuskan pada validasi perilaku sistem dari sudut pandang pengguna (*consumer perspective*), khususnya pada modul Project dan Bizcase versi 3 (V3), guna memastikan bahwa seluruh fitur yang disediakan dapat berjalan secara konsisten, aman, dan sesuai dengan kebutuhan operasional perusahaan.

1. **Metodologi dan Perangkat Pengujian**

   Pengujian dilakukan untuk memverifikasi bahwa seluruh fitur backend API telah memenuhi kebutuhan pengguna serta mampu menangani skenario penggunaan yang kompleks. Proses pengujian dilakukan melalui dua perangkat utama, yaitu:

   a) **Swagger UI**: Digunakan untuk pengujian interaktif dan validasi dokumentasi OpenAPI secara *real-time*.

   b) **Postman**: Digunakan untuk pengujian logika kueri yang lebih mendalam, terutama pada operasi pengambilan data (*GET*) dan pembaruan data (*PUT*) yang melibatkan transaksi basis data yang kompleks.

   Kombinasi kedua perangkat tersebut memungkinkan proses pengujian dilakukan secara menyeluruh, baik dari sisi dokumentasi API maupun validasi perilaku sistem dalam berbagai skenario pengujian.

2. **Pengujian Interaktif melalui Swagger UI**

   Swagger UI digunakan sebagai tahap awal pengujian untuk memverifikasi kesesuaian skema *request* dan *response* secara visual. Pada pengujian modul **Bc Risk**, dilakukan simulasi

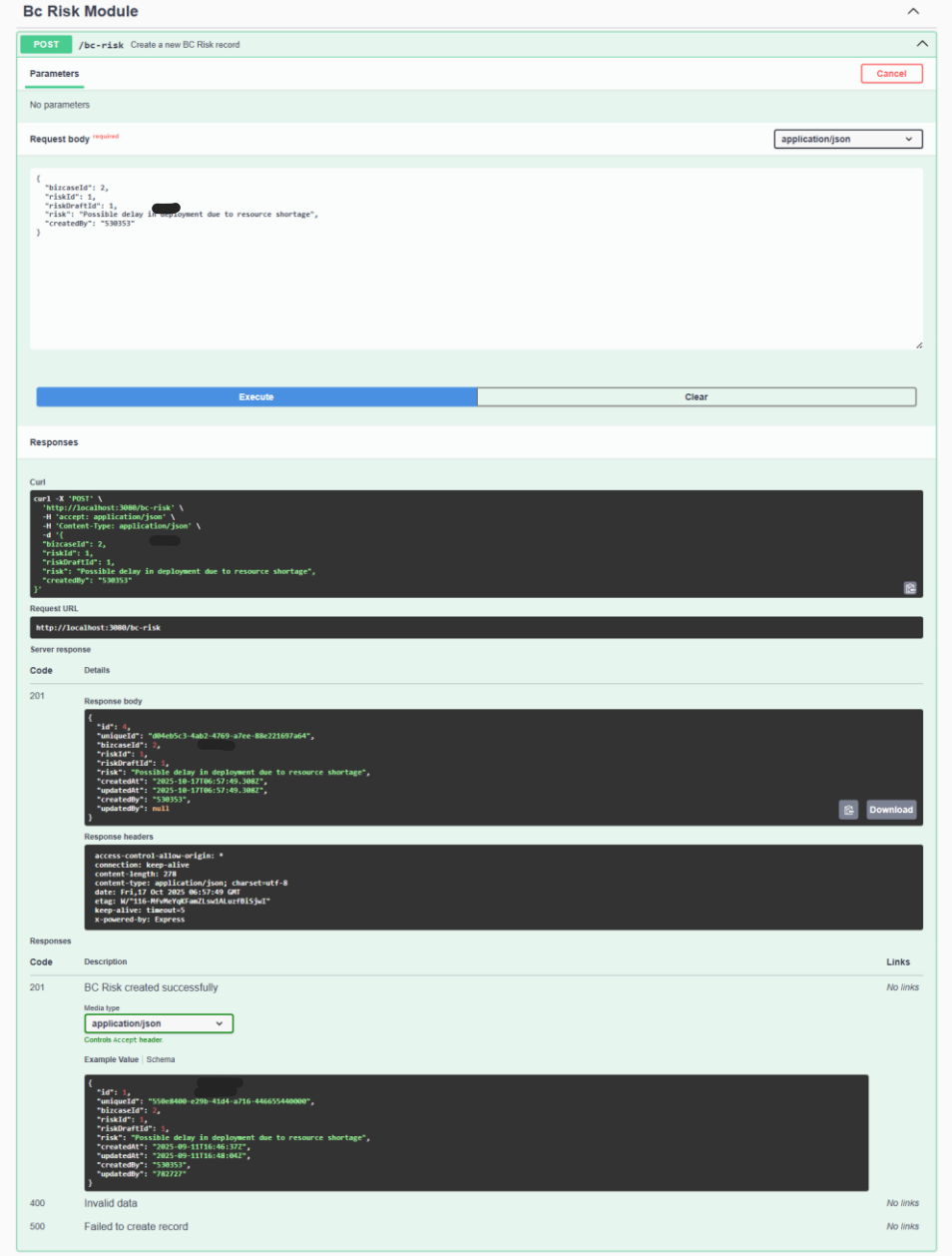
penginputan data risiko proyek untuk memvalidasi efektivitas Data Transfer Object (DTO) serta logika layanan (*service logic*) yang telah diimplementasikan.

a) **Skenario Pengujian POST**:

Penulis mengirimkan *payload* JSON yang berisi atribut bizcaseId dan deskripsi risiko teknis ke endpoint terkait. Permintaan tersebut diproses melalui *BizcaseService* sesuai dengan alur logika bisnis yang telah dirancang.

b) **Hasil Eksekusi**:

Berdasarkan hasil pengujian pada Gambar 3.18, sistem memberikan respons dengan status kode **201 Created**, yang menandakan bahwa data berhasil disimpan. Pada *response body*, sistem secara otomatis menghasilkan uniqueId berbasis UUID serta mengisi kolom audit createdAt sesuai dengan skema basis data yang telah dirancang pada Proyek 1. Hasil ini menunjukkan bahwa mekanisme validasi DTO dan proses penyimpanan data telah berjalan dengan baik.



Gambar 3.18 Screenshot Hasil Pengujian API POST pada Modul Bc Risk di Swagger

3. Validasi Logika Get dan Put (Postman)

Selain pengujian melalui Swagger UI, dilakukan pengujian lanjutan menggunakan Postman untuk memvalidasi logika pengambilan data (*GET*) dan pembaruan data (*PUT*). Pengujian menggunakan Postman memberikan fleksibilitas dalam

pengaturan *environment variables* serta kemudahan dalam memantau struktur respons dan waktu eksekusi permintaan.

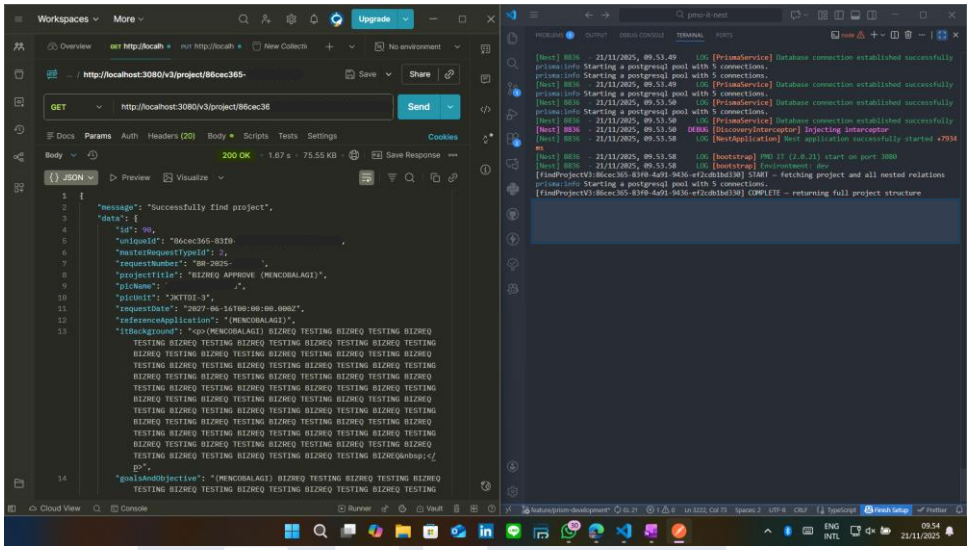
a) **Validasi Logika Pengambilan Data (GET – Find Logic)**

Pengujian menggunakan Postman difokuskan pada validasi fungsionalitas pengambilan data masif pada modul proyek versi 3 (V3). Berdasarkan cuplikan kode pada file project.service.ts, fungsi findProjectV3 dirancang untuk menarik data secara atomik dan menyeluruh menggunakan blok transaksi.

Fungsi ini memiliki kompleksitas tinggi karena menggunakan fitur include Prisma untuk mengambil belasan tabel relasi dalam satu kali kueri (*single fetch*). Relasi yang ditarik mencakup:

a. **Data Administrasi**: masterRequestType, document, dan progressLog.
b. **Data Transaksi & Risiko**: projectTransaction dan riskHazard.
c. **Data Bizcase Terintegrasi**: Objek bizcase ditarik secara mendalam (*deep nested*) mencakup bcInfraNeeds, bcTestingScope, bcCostEfficiency, hingga bcRisk.

Pengujian melalui Postman membuktikan bahwa respon JSON yang dihasilkan tetap akurat dan cepat meskipun volume data relasi yang ditarik sangat besar. Hal ini memastikan tim *frontend* mendapatkan data yang lengkap tanpa perlu melakukan pemanggilan API secara berulang kali (*n+1 query problem*) seperti yang terlihat pada Gambar 3.19.



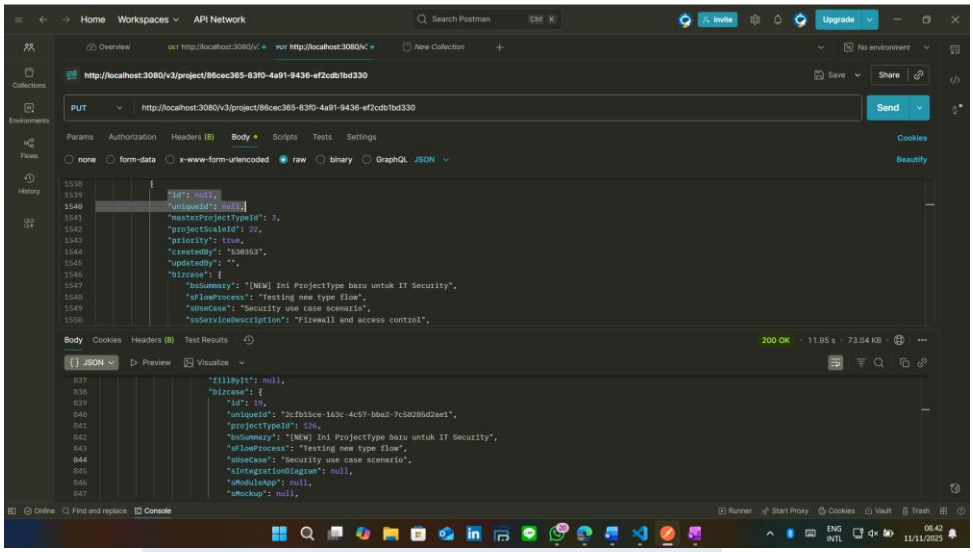
Gambar 3. 19 Respon JSON GET Project V3 pada Postman

b) **Validasi Logika Pembaruan Data (PUT – Update Logic)**

Pengujian pembaruan data difokuskan pada penggunaan uniqueId berbasis UUID (*Universally Unique Identifier*) untuk menjamin keamanan dan keakuratan data. Berbeda dengan identitas numerik (*autoincrement*), penggunaan UUID yang dihasilkan melalui fungsi gen_random_uuid() pada tingkat basis data bertujuan untuk memitigasi risiko keamanan berupa eksploitasi enumerasi ID oleh pihak yang tidak berwenang.

Dalam pengujian ini, penulis melakukan simulasi pembaruan data pada modul *Project* dan *Bizcase* V3 melalui *endpoint* PUT. Secara teknis, proses ini mengeksekusi metode updateProjectV3 pada lapisan *service* yang mengimplementasikan mekanisme prisma.$transaction. Penggunaan transaksi ini sangat krusial karena satu aksi pembaruan melibatkan manipulasi pada beberapa tabel relasi sekaligus, seperti tabel projectTransaction yang memerlukan proses pembersihan

data lama (*deleteMany*) sebelum memasukkan data baru (*createMany*) dalam satu siklus atomik.



Gambar 3.20 Pengujian PUT (updateProjectV3) pada Postman

Berdasarkan hasil pengujian pada Gambar 3.20, sistem memberikan respons dengan status kode **200 OK**. Pada bagian *response body*, terlihat bahwa kolom updatedAt telah diperbarui secara otomatis oleh Prisma ORM sesuai dengan stempel waktu eksekusi permintaan. Hasil ini memvalidasi beberapa aspek teknis sebagai berikut:

a. **Integritas Transaksi**:

Mekanisme prisma.$transaction berhasil menjaga konsistensi data, di mana seluruh perubahan pada entitas induk dan anak tersimpan secara utuh tanpa adanya data yang korup.

b. **Keamanan Akses**:

Penggunaan uniqueId sebagai parameter pencarian pada kueri findUnique terbukti akurat dalam mengidentifikasi catatan spesifik yang akan diperbarui.

c. **Otomatisasi Audit**:

Kolom updatedAt dan updatedBy berhasil mencatat jejak perubahan (*audit trail*), yang merupakan persyaratan fungsional penting dalam sistem manajemen proyek di PT GMF AeroAsia Tbk.

3.3.2 Kendala yang Ditemukan

Selama pelaksanaan program magang di unit TDI-2 PT GMF AeroAsia Tbk, ditemukan berbagai tantangan teknis maupun operasional yang menuntut kemampuan analisis mendalam dalam proses pengembangan sistem manajemen proyek versi 3 (V3). Kendala-kendala ini muncul seiring dengan kompleksitas *tech stack* yang digunakan serta standar integritas data yang sangat ketat di industri penerbangan. Berikut adalah penjabaran mendalam mengenai kendala-kendala tersebut:

1. **Kompleksitas Pemetaan Relasi Data pada Skema Basis Data Eksisting**

   Tantangan utama yang dihadapi adalah memahami dan memetakan relasi antar-tabel pada skema basis data PostgreSQL yang sudah sangat luas. Mengingat model *Bizcase* memiliki belasan sub-entitas yang saling bergantung seperti bcInfraNeeds, bcRisk, dan bcCostEfficiency, penentuan strategi relasi pada Prisma ORM menjadi sangat kompleks.

   Kesalahan dalam pendefinisian kunci tamu (*foreign key*) atau tipe relasi (seperti *one-to-one* vs *one-to-many*) berpotensi menyebabkan kegagalan saat proses pengambilan data (*querying*) atau ketidakkonsistenan data saat dilakukan penghapusan catatan induk. Hal ini memerlukan ketelitian ekstra karena setiap data finansial harus terhubung secara akurat dengan ID proyek terkait.

2. **Kurva Pembelajaran Arsitektur Modular NestJS yang Ketat**

   Implementasi arsitektur modular pada NestJS menuntut pemahaman mendalam mengenai konsep *Dependency Injection* dan manajemen *Providers*. Kendala muncul saat harus mengatur keterhubungan antar-

modul, misalnya bagaimana modul Project dapat mengakses fungsi pada BizcaseService tanpa menyebabkan *circular dependency* (ketergantungan melingkar) yang dapat mengakibatkan aplikasi gagal dijalankan.

Selain itu, penggunaan dekorator yang sangat intensif pada lapisan *Controller* dan DTO memerlukan waktu adaptasi tambahan guna memastikan seluruh metadata API terkonfigurasi dengan benar sesuai standar dokumentasi OpenAPI yang diinginkan perusahaan.

3. **Penanganan *Edge Cases* pada Logika Validasi Keamanan (XSS)**

   Dalam mengembangkan dekorator kustom @NoHtml untuk keamanan, kendala ditemukan pada tahap perancangan ekspresi reguler (*regex*) yang mampu mendeteksi tag berbahaya secara akurat tanpa mengganggu *input* data yang valid.

   Tantangannya adalah membedakan antara *input* teks normal yang mungkin mengandung karakter khusus (seperti simbol < atau > untuk perbandingan data teknis) dengan tag HTML asli yang bersifat instruksional.

   Ketidaktepatan dalam penyusunan *regex* ini dapat menyebabkan *false positive*, di mana sistem menolak data yang sah, yang pada akhirnya dapat menghambat pengalaman pengguna saat menginput data operasional di GMF AeroAsia.

4. **Sinkronisasi Logika Bisnis MRO yang Dinamis dan Masif**

   Prosestransformasi proses bisnis manual ke dalam ekosistem digital pada modul *Bizcase* memerlukan pemahaman mendalam terhadap logika industri MRO (*Maintenance, Repair, and Overhaul*). Kendala ditemukan saat melakukan penyelarasan perhitungan efisiensi finansial dan pemetaan data migrasi dari sistem versi sebelumnya (V2) ke versi 3.

   Banyaknya variabel finansial seperti estimasi pendapatan (cRevenue) dan biaya operasional (cOperatingCost) yang bersifat opsional namun tetap harus terintegrasi dalam laporan manajerial

menuntut logika pengkondisian yang rumit pada lapisan *service* agar tidak terjadi *error* saat manipulasi data dilakukan.

5. **Manajemen Transaksi pada Operasi Data Masif dan Bertingkat**

   Saat mengembangkan fitur pembaruan proyek yang melibatkan penghapusan data lama dan pembuatan data transaksi baru secara simultan, tantangan teknis muncul dalam menjaga atomisitas transaksi. Kesulitan dialami saat melakukan *debugging* pada blok kode prisma.$transaction ketika terjadi kegagalan di salah satu sub-proses, seperti saat proses deleteMany berhasil namun createMany gagal.

   Tanpa penanganan kesalahan (*error handling*) yang presisi dan penggunaan *Transaction Client* (trx) yang benar, kegagalan di tengah proses dapat meninggalkan data "sampah" (*junk data*) yang merusak integritas basis data PostgreSQL secara keseluruhan.

Selain aspek teknis dalam pengembangan perangkat lunak, terdapat pula kendala non-teknis yang dihadapi selama menjalankan program *Career Acceleration Program* di lingkungan PT GMF AeroAsia Tbk. Kendala ini berkaitan erat dengan penyesuaian diri terhadap budaya kerja industri penerbangan yang memiliki standar disiplin dan regulasi yang sangat tinggi.

1. **Adaptasi Budaya Kerja dan Disiplin Waktu Industri Penerbangan**

   Sebagai mahasiswa, transisi menuju lingkungan kerja profesional dengan jadwal yang sangat ketat menjadi tantangan tersendiri. Penulis diwajibkan mengikuti jam operasional kantor secara penuh (*Work from Office*) mulai pukul 07.00 WIB hingga 16.00 WIB.

   Standar disiplin yang diterapkan di GMF AeroAsia sangat tinggi, di mana ketepatan waktu memulai dan mengakhiri pekerjaan merupakan bagian dari penilaian profesionalisme. Hal ini menuntut manajemen waktu yang sangat baik untuk menjaga produktivitas dalam durasi 9 jam kerja per hari di tengah tekanan proyek pengembangan sistem yang dinamis.

2. **Prosedur Keamanan dan Regulasi Area Terbatas**

Bekerja di kawasan Bandara Internasional Soekarno-Hatta mengharuskan setiap personel mematuhi regulasi keamanan yang sangat ketat. Penulis harus melewati berbagai tahapan administratif yang kompleks, mulai dari proses *Security Clearance* secara daring hingga pengurusan kartu identitas akses (*Pass Intern*).

Kendala muncul saat proses koordinasi akses area terbatas ini memerlukan waktu dan kepatuhan prosedur yang birokratis, yang mana hal ini merupakan pengalaman baru bagi penulis dalam lingkungan kerja skala *enterprise*.

3. **Sinkronisasi Dokumentasi Administratif dan Tugas Harian**

Kewajiban administratif untuk mendokumentasikan setiap aktivitas harian melalui formulir *Daily Task* (PRO-STEP 03) menuntut ketelitian dalam pencatatan. Kendala dirasakan saat penulis harus merinci setiap progres teknis pengembangan *backend* ke dalam bahasa administratif yang dapat dipahami oleh pihak universitas maupun pihak *Learning Centre Unit* (LCU) GMF. Sering kali terdapat penumpukan laporan administratif yang harus diselesaikan di sela-sela fokus pengerjaan logika koding yang kompleks.

4. **Komunikasi Lintas Fungsi dan Pemahaman Proses Bisnis MRO**

Dalam lingkungan TDI-2, penulis tidak hanya berkomunikasi dengan sesama pengembang, tetapi juga harus memahami kebutuhan dari berbagai unit bisnis terkait. Kendala komunikasi sering kali muncul saat mencoba menerjemahkan proses bisnis perawatan pesawat (MRO) yang sangat teknis dan spesifik ke dalam logika pemrograman yang efisien. Diperlukan kemampuan komunikasi yang adaptif untuk menjembatani perbedaan terminologi antara kebutuhan operasional di hangar dengan keterbatasan teknis di sisi pengembangan sistem digital.

3.3.3 Solusi atas Kendala yang Ditemukan

Sebagai respons terhadap berbagai tantangan teknis maupun non-teknis yang dihadapi, mahasiswa magang melakukan serangkaian langkah

solutif yang sistematis. Implementasi solusi ini tidak hanya bertujuan untuk menyelesaikan hambatan sesaat, tetapi juga untuk memastikan bahwa arsitektur sistem yang dibangun memiliki kualitas standar industri. Berikut adalah rincian solusi yang diimplementasikan:

1. **Analisis Skema Relasional dan Optimalisasi Pemetaan Prisma**

   Untuk mengatasi kompleksitas relasi data, dilakukan analisis mendalam terhadap file schema.prisma guna memastikan setiap relasi *one-to-many* antara entitas Bizcase dan sub-entitasnya terdefinisi dengan akurat. Solusi teknis yang diambil adalah dengan memanfaatkan dekorator @relation untuk mengatur *field* dan *references* secara eksplisit, guna menjaga integritas referensial data. Selain itu, dilakukan validasi data secara berkala menggunakan **Prisma Studio** untuk memantau apakah data relasional telah masuk ke tabel yang tepat di PostgreSQL.

2. **Penerapan Arsitektur Modular dan Manajemen Dependensi**

   Hambatan pada kurva pembelajaran NestJS diatasi dengan menerapkan pemisahan tanggung jawab yang ketat (*Separation of Concerns*) melalui pembuatan modul-modul terpisah. Solusi ini mencakup pengorganisasian folder modul fungsional seperti master-activity, bc-infra-needs, dan bc-risk di bawah direktori src/modules. Manajemen dependensi diatur secara efisien melalui BizcaseModule yang mendaftarkan BizcaseService dan PrismaService sebagai *providers*, sehingga memudahkan pemeliharaan kode.

3. **Pengembangan dan Pengujian Unit *Custom Decorator* Keamanan**

   Solusi terhadap risiko keamanan XSS dilakukan dengan mengembangkan kustom dekorator @NoHtml berbasis *Regular Expression* (Regex) yang presisi. Keberhasilan solusi ini divalidasi dengan menerapkannya pada berbagai DTO, seperti pada atribut dataSize di modul *Bizcase Infra Needs*. Dengan integrasi ini, sistem secara otomatis menolak permintaan yang mengandung tag skrip berbahaya dan memberikan respon kesalahan yang informatif.

4. **Pemanfaatan Swagger UI untuk Koordinasi dan Validasi API**

   Kendala komunikasi teknis diatasi dengan optimalisasi dokumentasi interaktif menggunakan Swagger UI. Portal Swagger digunakan sebagai media validasi kontrak data antara tim *backend* dan *frontend*, di mana setiap *endpoint* seperti proses pembuatan (*POST*) atau penghapusan (*DELETE*) data dapat diuji secara langsung di lingkungan *sandbox*.

5. **Implementasi Transaksi Atomik untuk Konsistensi Data**

   Tantangan pada manajemen transaksi diatasi dengan mengimplementasikan fungsi prisma.$transaction pada operasi data bertingkat, terutama pada fitur pembaruan proyek. Solusi inovatif yang diterapkan adalah penggunaan parameter transaksi (trx) yang dioperasikan antar-layanan melalui *interface* yang terstandarisasi. Jika terjadi kesalahan pada salah satu proses, sistem secara otomatis akan memicu instruksi *rollback*, sehingga menjamin integritas basis data PostgreSQL.

6. **Manajemen Waktu dan Kedisiplinan Mandiri**

   Guna beradaptasi dengan budaya kerja WFO yang ketat (07.00 - 16.00 WIB), mahasiswa magang menerapkan strategi manajemen waktu yang disiplin. Hal ini mencakup perencanaan jadwal keberangkatan lebih awal untuk memastikan kehadiran tepat waktu serta penyusunan prioritas tugas harian guna menjaga produktivitas selama 9 jam kerja di lingkungan GMF AeroAsia.

7. **Proaktif dalam Prosedur Keamanan dan Administrasi**

   Kendala birokrasi keamanan diatasi dengan bersikap proaktif dalam mengikuti setiap tahapan *Security Clearance* dan pengurusan akses area terbatas. Pemenuhan kewajiban administratif, seperti pengisian formulir *Daily Task* (PRO-STEP 03), dilakukan secara rutin setiap sore hari sebelum jam kerja berakhir untuk memastikan seluruh progres teknis terdokumentasi dengan akurat dan tepat waktu.

8. **Peningkatan Komunikasi Lintas Fungsi**

Untuk memahami proses bisnis MRO yang kompleks, mahasiswa magang melakukan diskusi rutin dan sesi tanya jawab dengan Mentor Teknis serta tim pengembang senior di unit TDI-2. Komunikasi ini bertujuan untuk menyelaraskan pemahaman teknis dengan kebutuhan operasional di lapangan, sehingga logika sistem yang dikembangkan, seperti modul *Bizcase*, benar-benar relevan dengan proses bisnis nyata di perusahaan.