BAB III
PELAKSANAAN KERJA PROFESI

3.1 Bidang Kerja

Dalam pelaksanaan kerja profesi di PT. Enerren Technologies yang bertempat di Gedung Graha Kapital 1 Lt. 1 Suite 101, Jl. Kemang Raya No. 4, Kemang, Jakarta Selatan – 12730. Praktikan ditempatkan pada Departemen *Information Technology* (IT) di divisi IT yang menjalankan fungsi :

1. Menyediakan layanan dan pengembangan IT dan komunikasi.
2. Menyediakan perangkat keras beserta layanannya seperti *server*, *repository* dan perangkat jaringan sesuai kebutuhan pelanggan dan internal perusahaan.
3. Memastikan ketersediaan, kesesuaian dan keamanan infrastruktur dan semua sistem IT agar dapat berjalan dengan lancar dengan melakukan perawatan secara berkala.
4. Melakukan *sharing knowledge* per bulan antar anggota divisi IT.

Dalam melaksanakan kerja profesi, Praktikan menjabat sebagai Manajer Proyek IT yang membawahi staf *Team Leader*, Programmer, IT QA, *Web Designer* dan UI/UX *Designer*. Praktikan melapor langsung kepada Manajer IT yaitu bapak Hari Fajri sebagai Kepala Divisi IT sekaligus pembimbing kerja selama masa kerja profesi.

Adapun deskripsi pekerjaan atau *role* yang harus dilakukan praktikan pada divisi IT diantaranya :

1. Memastikan proyek selesai tepat waktu dan sesuai anggaran.
2. Melakukan proses manajemen resiko untuk mengurangi resiko perusahaan.
3. Mengelola Sumber Daya Manusia (SDM) IT, termasuk efisiensi dan efektivitas penggunaan SDM.
4. Mengukur kinerja proyek.

5. Berkomunikasi secara efektif dengan pelanggan untuk memastikan dan menjaga kepuasan pelanggan.
6. Berkoordinasi dengan vendor eksternal.
7. Memastikan tercapainya sasaran biaya, mutu, waktu, K3 dan lingkungan dalam setiap proyek yang dikerjakan.

Dalam menjalani profesi yang dijalankan oleh praktikan sebagai mahasiswa SIF harus dapat mengaplikasikan pekerjaan sesuai dengan mata perkuliahan yang telah praktikan pelajari di UPJ. Seorang manajer proyek tidak hanya berhubungan dengan internal divisi saja melainkan harus bisa melakukan hubungan kerja yang profesional dengan lintas fungsional kerja seperti departemen *Sales & Marketing*, departemen *Finance* dan lain sebagainya, selain itu praktikan dituntut untuk dapat berkomunikasi dengan baik terhadap pelanggan maupun vendor eksternal perusahaan.

Tantangan yang dihadapi pun sangat besar, karena selain harus dapat memastikan proyek berjalan sesuai target dan perencanaan, seorang manajer proyek juga diharuskan untuk dapat secara disiplin meningkatkan pengetahuan di bidang teknologi serta meminimalisir terjadinya kesenjangan antara teknologi yang digunakan perusahaan dengan yang saat ini tengah berkembang di masyarakat. Manajer proyek juga harus dapat mempertanggungjawabkan pelaporan *Key Performance Index* (KPI) dari divisi IT agar dalam kondisi yang baik, karena di PT. Enerren Technologies, KPI inti terletak pada departemen dan divisi IT.

3.2 Pelaksanaan Kerja

Dalam melaksanakan kegiatan kerja profesi, Praktikan berkolaborasi dengan semua karyawan divisi IT. Berikut merupakan tugas dan tanggung jawab praktikan :

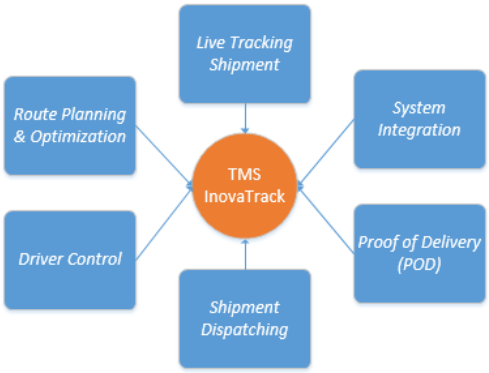
a. Melakukan presentasi produk dan negosiasi dengan pelanggan.
b. Melakukan perhitungan estimasi kebutuhan Sumber Daya Manusia (SDM), estimasi waktu pengerjaan dan biaya yang dikeluarkan dalam proyek.
c. Melakukan analisis sistem dan mengajukan arsitektur solusi yang akan diberikan dalam proyek.

d. Melakukan *daily stand-up meeting* dengan tim proyek, rapat laporan mingguan dan bulanan serta KPI dengan manajemen dan direksi.
e. Melakukan pengawasan dan pelaporan jalannya proyek, *coaching* tim IT, melakukan *sharing knowledge* dan melakukan eskalasi jika terdapat suatu hambatan yang layak mendapatkan eskalasi manajemen dan direksi.
f. Melakukan rapat *project progress update* kepada pelanggan.

Pada pelaksanaan kerja profesi ini, Praktikan mengajukan arsitektur yang dapat digunakan sebagai referensi perusahaan guna meningkatkan kinerja dan fleksibilitas dari program Sistem Informasi Geografis (SIG). Adapun langkah-langkah yang dilakukan praktikan antara lain sebagai berikut :

a. Melakukan analisis terhadap aplikasi InovaTrack *Transportation Management System* (TMS) yang berjalan saat ini.

InovaTrack TMS merupakan aplikasi berbasis *web* dan *mobile* yang berfokus untuk memantau pengiriman barang untuk setiap kendaraan yang terpasang GPS *Tracking*.



Gambar 3.1 Modul TMS InovaTrack

TMS memiliki beberapa fungsi :

1. *Optimisasi route*

setiap rute pengiriman telah mendapatkan optimisasi rute menggunakan algoritma rute dari Google.

2. *Live Tracking Shipment*

Setelah sistem mendapatkan pengiriman barang, maka sistem akan otomatis menyediakan pemantauan langsung diatas peta yang dapat diakses menggunakan *web* dan *mobile*.

3. *Driver Control*

Setiap pengemudi dinilai kualitas mengemudinya dengan menggunakan sensor rem mendadak, gas mendadak dan banting stir yang didapatkan dari alat pelacak GPS.

4. Dispatching (Pengiriman)

TMS menyediakan modul untuk melakukan draft pengiriman, penjadwalan menggunakan kendaraan yang terpasang GPS serta pemantauannya.

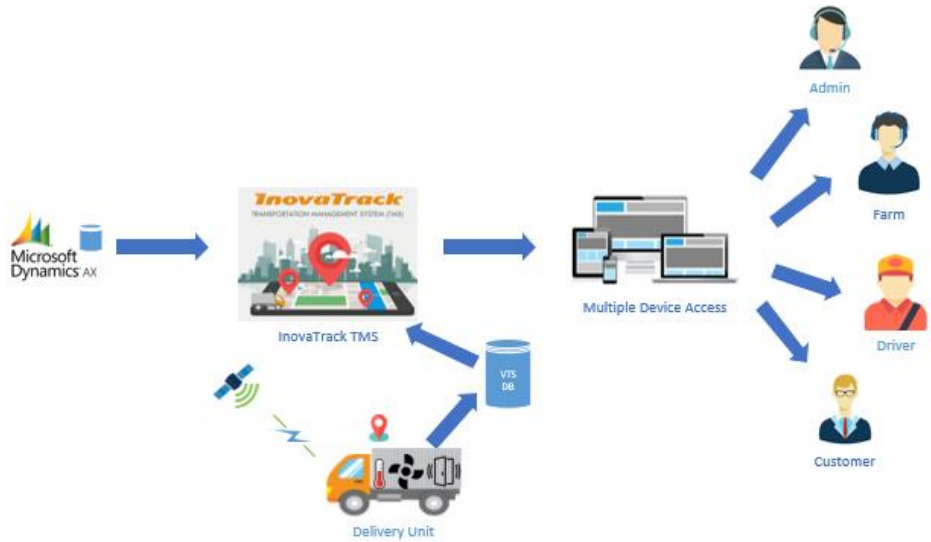
5. Integrasi Sistem

TMS InovaTrack dapat terhubung dengan aplikasi lain milik pelanggan seperti SAP, Microsoft Dynamics dan aplikasi *custom* milik perusahaan pelanggan.

6. *Proof of Delivery* (POD) Pengiriman

Proof of Delivery (POD) adalah verifikasi bahwa penerima barang sesuai penanggung jawabnya dan barang sampai di tujuan dengan kondisi yang baik.

Berikut alur dari sistematis TMS :

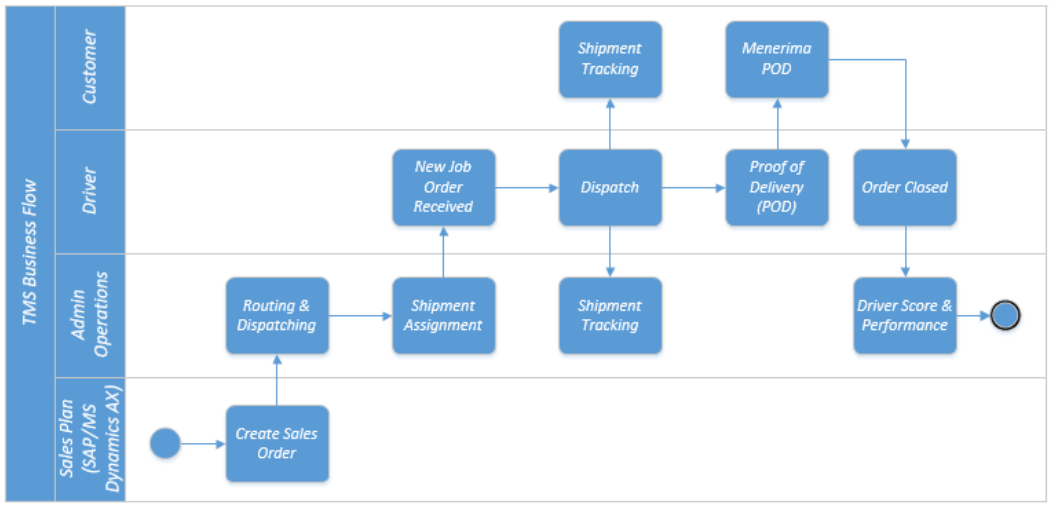


Gambar 3.2 topologi TMS

Topologi TMS (SRS PT. Sierad Produce, Tbk, 2019)

Berikut penjelasan mengenai began topologi diatas :

1. Microsoft Dynamics AX atau SAP atau aplikasi internal milik perusahaan pelanggan dapat dihubungkan dan mengirimkan daftar pengiriman barang yang akan dilakukan oleh TMS.
2. TMS menerima data menggunakan *Application Interface Program* (API) dalam format JSON untuk diolah di dalam *database* TMS sebagai jadwal pengiriman barang.
3. Aplikasi InovaTrack TMS akan mengolah data tersebut dan selanjutnya terhubung dengan alat pelacak GPS yang dipasang di setiap kendaraan.
4. Selanjutnya aplikasi akan dapat diakses melalui beragam perangkat seperti komputer, *laptop* ataupun ponsel pintar.
5. Setiap pengguna memiliki akses masing-masing.



Gambar 3.3 Alur aplikasi TMS

Berikut penjelasan dari diagram diatas :

1. Pengguna melakukan pembuatan *order* pengiriman barang, *order* ini dapat dilakukan secara manual atau otomatis dengan menghubungkan aplikasi dengan sistem *Enterprise Resource Planning* (ERP) seperti SAP atau Microsoft Dynamics.

2. Ketika pelaksanaan pengiriman barang, sistem akan menerima posisi kendaraan dan sensor-sensor yang aktif akan mengirimkan status saat itu secara *realtime*. Jika terjadi kondisi tidak normal seperti melebihi batas kecepatan yang ada, gas mendadak, rem mendadak, banting stir atau suhu mencapai batas maksimal dan atau terlalu rendah serta pintu belakang boks terbuka saat pengiriman barang, maka sistem akan mengirimkan peringatan kepada setiap pengguna yang berhak mendapatkan peringatan.
3. Data yang telah diolah dapat diakses oleh setiap pengguna sesuai dengan hak aksesnya untuk memantau kendaraan, baik yang menggunakan *web* maupun yang berubah *mobile*.
4. Ketika sampai di lokasi tujuan maka supir akan mendapatkan *Proof of Delivery* (POD) berupa foto penerima, tanda tangan digital dan foto-foto barang. Hasil foto dokumentasi ini dapat dilihat oleh pengguna yang berhak.

Dari analisis praktikan, aplikasi InovaTrack TMS memiliki struktur program sebagai berikut :

Tabel 3.1 Arsitektur Web InovaTrack TMS

Framework	.NET 4.5.2
Bahasa Pemrograman	C#.NET
Arsitektur	*Component-based monolith*
Pola Desain	*Model-View-ViewModel* (MVVM)
Plugins	Bootstrap, JQuery
Database	Microsoft SQL Server 2017
Jenis Server	*Azure Cloud Server*

Sumber : SRS PT. Sierad Produce Indonesia, Tbk.

Berikut struktur program aplikasi ponsel pintar TMS :

Tabel 3.2 Arsitektur Mobile InovaTrack TMS

Framework	Flutter
Bahasa Pemrograman	Dart
Arsitektur	*Component-based monolith*

Pola Desain	*Model-View-ViewModel* (MVVM)
Plugins	Bootstrap, JQuery
Database	Microsoft SQL Server 2017
Komunikasi	RESTFul API (JSON)

Sumber : SRS PT. Sierad Produce Indonesia, Tbk.

Dalam hal arsitektur, TMS menggunakan arsitektur monolitik yang hanya memiliki satu struktur *Model View Controller* (MVC) saja.

b. Mengamati kekurangan dari aplikasi TMS yang berjalan saat ini.

Berikut ini merupakan hasil analisis dari aplikasi TMS yang berjalan saat ini :

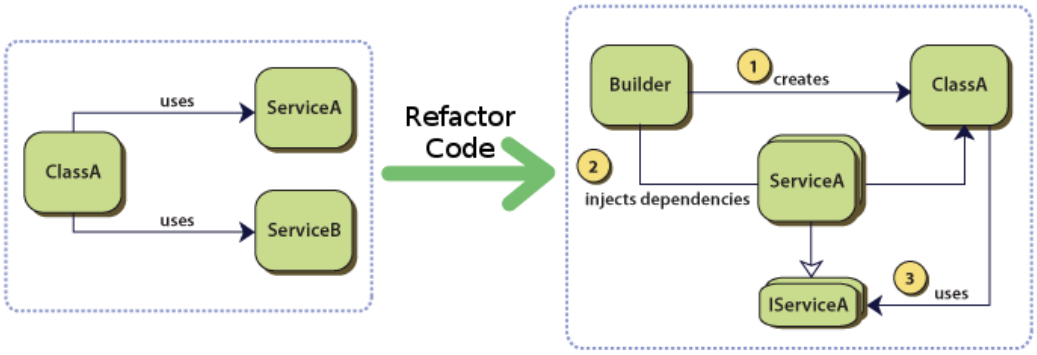
1. Kelebihan
   i. Arsitektur sederhana, dalam arsitektur monolitik maka arsitektur yang disajikan akan lebih sederhana.
   ii. Mudah dalam mempelajarinya, staf programmer yang baru bergabung dengan perusahaan dapat dengan mudah mempelajarinya.
   iii. Mudah dalam melakukan publikasi ke *server*. Ketika melakukan publikasi ke *server*, tinggal melakukan salin semua berkas ke *server*.
2. Kekurangan
   i. Kurang memaksimalkan konsep *reusability* dalam pemrograman berorientasi objek.
   ii. Bila terjadi perubahan kecil di salah satu sub class aplikasi, maka programmer harus melakukan perombakan terhadap keseluruhan sistem.
   iii. Bila aplikasi telah menjadi besar, maka akan lebih sulit melakukan pemeliharaan karena kembali ke poin diatas, ketika akan mengubah suatu proses maka harus diubah dan dilakukan kompilasi terhadap keseluruhan solusi program.
   iv. Sangat terbatas, kurang cocok untuk aplikasi yang sudah terintegrasi dengan berbagai macam aplikasi lain.

v. Bila memiliki pelanggan dengan proses bisnis yang berbeda-beda maka akan lebih sulit mengimplementasikannya di dalam satu *repository* proyek.

c. Membuat rancangan transformasi dari sistem berjalan saat ini ke yang baru dengan membuat arsitektur *Dependency Injection* (DI).

Dalam perjalanannya, sistem arsitektur Inovatrack TMS telah dilakukan perubahan arsitektur oleh praktikan. Konsep *Dependency Injection* (DI) yang diterapkan memang belum menjadi hasil pencapaian yang diharapkan yaitu *microservices*, Transformasi ke *microservices* harus dilakukan banyak transisi perubahan dari monolitik dengan melakukan pemecahan modul dan fungsi ke dalam *sub project*.

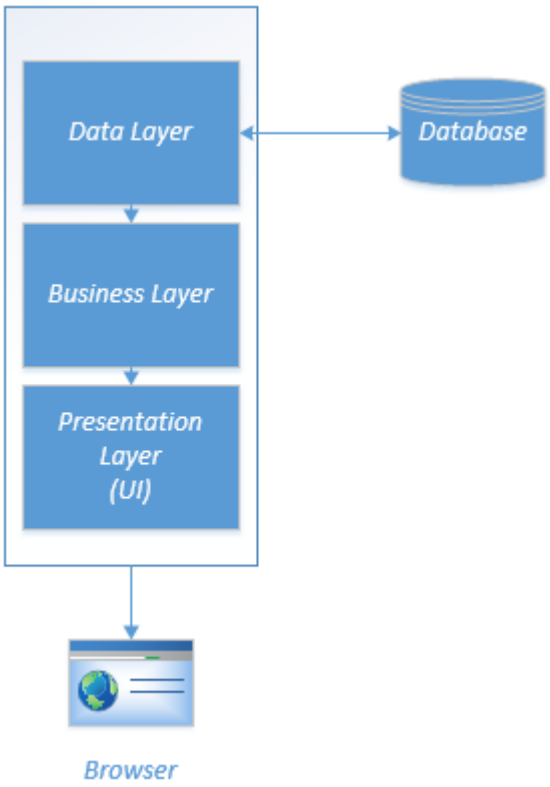
DI adalah sebuah *design pattern* program yang berorientasi masa depan. DI menitikberatkan pada penulisan kode yang *loosely coupled*, *highly modular* dimana ketergantungan antar *class* tidaklah tinggi, sehingga kita dapat mengubah sebuah *class* tanpa diliputi rasa khawatir *class* tersebut membuat *break class-class* lain yang bergantung pada *class* yang kita ubah tadi. (Wibowo, 2017).



Gambar 3.4 ilustrasi *refactor code* menjadi DI
Code Refactor (Devodia, 2018)

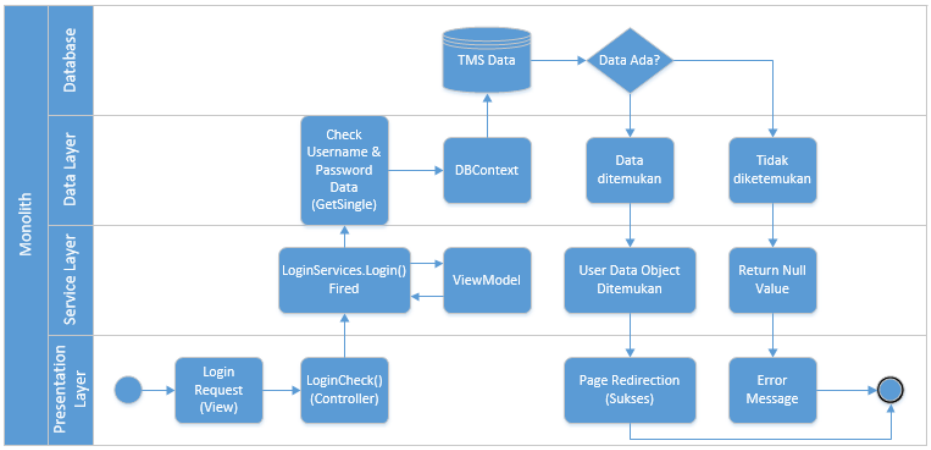
Pada dasarnya DI adalah melakukan injeksi fungsional dari konstruktor kepada *class* program tertentu sesuai dengan fungsinya, hal ini akan memaksimalkan penggunaan konsep *reusability* dari *class* dan programmer dapat melakukan perubahan pada *class* yang diperlukan tanpa harus mengkhawatirkan akan mengakibatkan *class* lain terjadi *break*.

d. Analisis arsitektur *microservices* terhadap TMS.



Gambar 3.5 Arsitektur Monolitik Berbasis Komponen

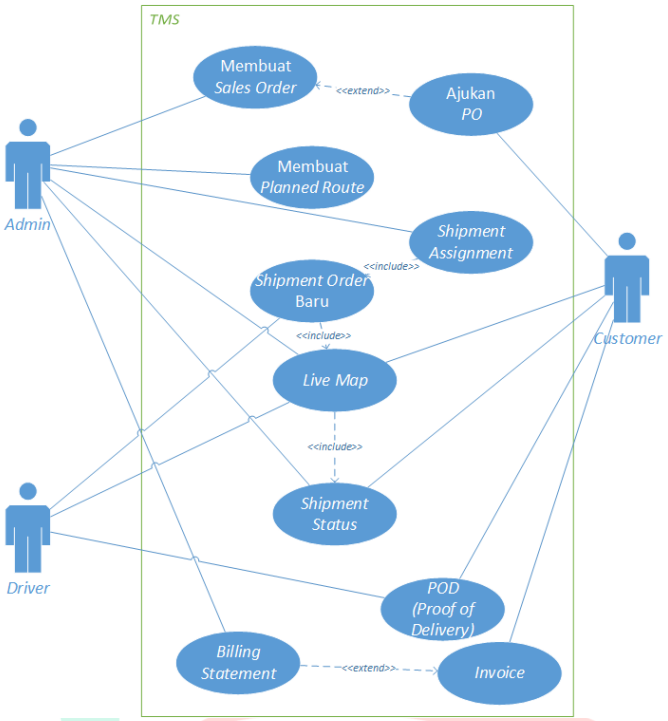
TMS Sebelumnya menggunakan arsitektur monolitik berbasis komponen dengan alur sebagaimana tampil diatas. Setiap bagian proses dipisahkan oleh *layer* yang berbeda sesuai dengan tugas masing-masing lapisan. Request dari user yang diakses melalui lapisan (*layer*) antarmuka (*Presentation Layer*) diproses oleh *Business Layer* yang terdiri atas logika bisnis yang memproses semua proses bisnis seperti menambah, mengurangi, menampilkan dan menghitung. Selanjutnya data dipersiapkan oleh *Data Layer* yang berhubungan langsung dengan *database*.



Gambar 3.6 Contoh Alur Request dan Response data arsitektur monolitik TMS (berbasis komponen)

Pada contoh alur diatas, TMS dalam melakukan *request* suatu fungsional masih dilakukan secara berurutan hingga pada akhirnya lapisan data yang melakukan akses langsung dengan *database*. Kelemahan proses ini adalah lapisan data tidak dapat digunakan kembali untuk aplikasi selain dari aplikasi *web* TMS.

Sedangkan, pada TMS, terdapat beberapa fitur fungsional yang beririsan dan memiliki fungsi yang sama dan membutuhkan peningkatan *reusability* yang tinggi. Selain itu, fitur-fitur yang dibuat juga dapat dikonsumsi oleh pelanggan agar dapat diintegrasikan dengan sistem informasi internal mereka.



Gambar 3.7 *Unified Model Language* (UML) Diagram yang menggambarkan hubungan fungsional sistem dengan penggunanya.

Berdasarkan UML diatas dapat diambil kesimpulan bahwa terdapat beberapa fitur yang dapat dijadikan fungsi yang memiliki sifat *Object Oriented Programming* (OOP) yaitu *reusability* adalah :

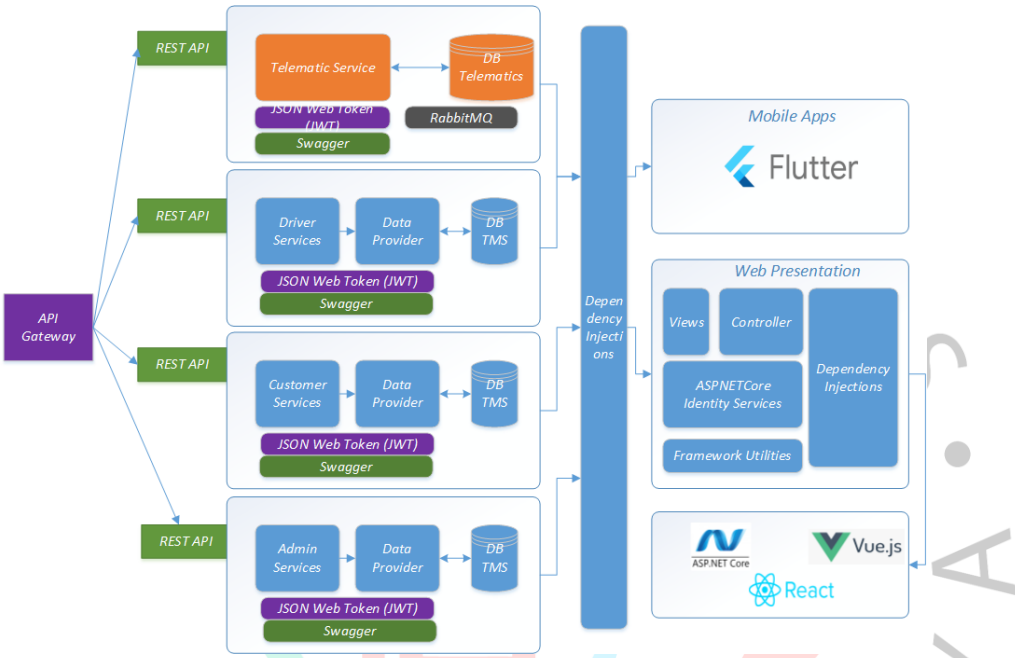
1. *Live Map*, yaitu fitur untuk menampilkan posisi terkini kendaraan diatas peta.

   Pada fitur ini terbagi lagi sub fungsionalnya yang mengambil dari proses telematika dari GPS Tracking diantaranya :

   a. Detail informasi kendaraan dan supir yang membawanya.
   b. Status posisi kendaraan.
   c. Status semua sensor yang aktif seperti sensor pintu belakang terbuka/tertutup, sensor kipas, sensor suhu, sensor pada kabin dan lain sebagainya.
   d. Status supir dalam cara membawa kendaraannya diantaranya sensor rem mendadak (*Harsh Breaking*), gas mendadak (*Harsh Acceleration*) dan banting stir (*Harsh Cornering*).
2. *Shipment Status,* yaitu fitur untuk menampilkan riwayat antar.
3. *Billing Statement,* yaitu fitur penagihan pemesanan.

e. Desain arsitektur *microservices* terhadap TMS.

Berdasarkan fungsional modul pada TMS, maka rancangan desain arsitektur adalah berikut ini :



Gambar 3.8 Pengajuan desain arsitektur *Microservices*

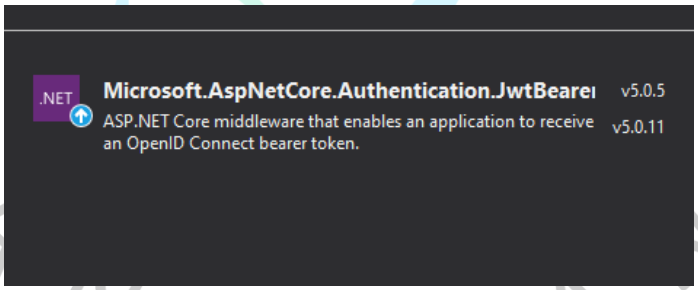
Aplikasi TMS yang besar dipecah menjadi beberapa bagian sub fungsional bisnis yaitu *Telematic Service*, *Driver Service*, *Customer Service* dan *Admin Service* yang mana masing-masing sub fungsional terhubung oleh *Representational State Transfer* (REST) API. Arsitektur ini memungkinkan setiap sub fungsional untuk memiliki beberapa database atau dalam satu *database* tergantung dari kebutuhan dan keputusan manajemen perusahaan.

Dalam rancangan yang akan diajukan, *service* dan *database telematics* yang menangani koneksi dan pengolahan data yang berasal dari alat *GPS Tracking* berdiri sendiri. *Database telematics* ini memiliki fisik dan skema database SQL Server yang terpisah dari aplikasi logistik *end-user* milik PT. Enerren Technologies seperti TMS, *PeopleTransport*, *MilestoneDelivery* dan lain sebagainya. Dalam prakteknya aplikasi

telematics ini merupakan aplikasi yang berhubungan dengan aplikasi logistik *end-user* lainnya dengan menggunakan REST API.

Setiap sub fungsional akan berkomunikasi menggunakan REST API sehingga memungkinkan penggunaan beragam bahasa pemrograman misalkan untuk *web* menggunakan .NET Core 5 atau React JS sementara untuk *mobile* menggunakan Flutter. Selain itu, modul fungsional yang dikembangkan dengan menggunakan konsep ini akan membantu memaksimalkan *reusability* antar aplikasi contohnya seperti modul *Shipment Tracking* yang digunakan tak hanya di aplikasi TMS melainkan juga di aplikasi *Digital Logistics* (Digilog) dan *Warehouse Management System* (WMS).

Untuk enkripsi komunikasi antar program, praktikan menggunakan JSON *Web* Token (JWT) yaitu verifikasi token pada aplikasi berbasis *web* untuk mengirimkan data sehingga dapat berbagi data oleh dua aplikasi atau bahkan lebih. Dalam penerapannya di otentikasi atau pun otorisasi, biasanya data ini berupa data yang sifatnya unik bagi *user*, seperti: *email*, id/uuid, dan juga data yang berkaitan dengan otorisasi seperti *role*, karena data tersebut akan digunakan sebagai tanda pengenal si pengirim token. (Faqih, 2020).



Gambar 3.9 Nuget *Package* penerapan JWT pada Bahasa pemrograman C#.NET

Untuk dapat menggunakan JWT pada arsitektur, maka pada *layer services* harus menambahkan paket Microsoft.AspnetCore.Authentication.JWTBearer. Selanjutnya pada logika bisnis yang menjalankan autentikasi, tambahkan *syntax* untuk mengaktifkan masa pakai JWT beserta tokennya dan metode enkripsinya. Dalam hal ini, praktikan menerapkan konfigurasi awal JWT menggunakan

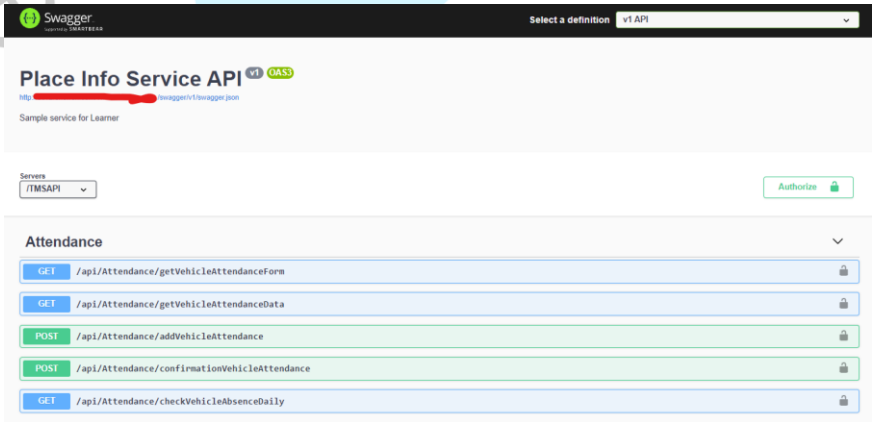
HmacSha256Signature. Microsoft SecurityAlgorithm juga menyediakan opsi enkripsi lainnya.

```
// authentication successful so generate jwt token
var tokenHandler = new JwtSecurityTokenHandler();
var key = Encoding.ASCII.GetBytes(appSettings.Secret);
var expiredTime = DateTime.Now.AddDays(7);
var tokenDescriptor = new SecurityTokenDescriptor
{
    Subject = new ClaimsIdentity(new Claim[]
    {
    new Claim(ClaimTypes.Sid, driver.driver_id.ToString()),
    new Claim(ClaimTypes.Actor, [redacted]),
    new Claim(ClaimTypes.MobilePhone, driver.phone_number ?? ""),
    new Claim(ClaimTypes.SerialNumber, DeviceId ?? ""),
    }),
    Expires = expiredTime,
    SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
};
var token = tokenHandler.CreateToken(tokenDescriptor);
```

Gambar 3.10 Penerapan JWT pada modul autentikasi pengguna

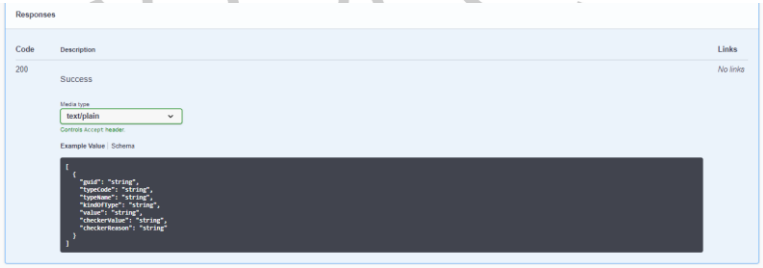
Selanjutnya, token akan selalu aktif sampai masa berlakunya berakhir, praktikan memberi masa aktif selama 7 hari. Selama token ini aktif, maka pengguna ketika menggunakan aplikasi tidak perlu melakukan login ulang meskipun arsitektur aplikasi yang berjalan di belakang layar menggunakan beragam segmentasi. Maka dengan demikian, *services* tersebut dapat menyediakan akses lintas *services* fungsional lainnya dengan *front-end* berbasis multi bahasa pemrograman.

Untuk dokumentasi akses API, praktikan menggunakan Swagger UI, yaitu sebuah *library open source* yang dibuat oleh SMARTBEAR untuk memvisualisasikan fungsional API tanpa harus melakukan *coding* terhadap fungsi tersebut, dokumentasi tersebut secara otomatis akan tersedia pada akses *web* dan dapat digunakan dan dicoba langsung secara visual.



Gambar 3.11 Penerapan Swagger sebagai visualisasi API

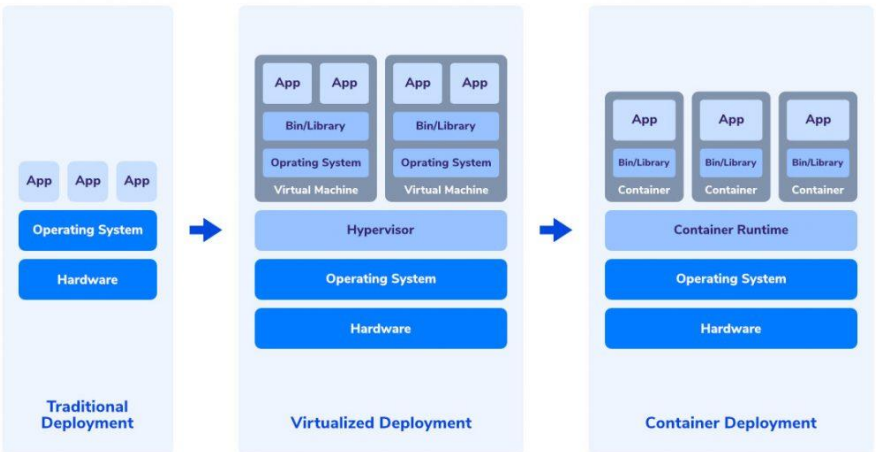
Dengan menggunakan swagger, programmer akan dengan mudah dapat melakukan pengetesan input dan output dari *method* API yang dibuat. Hal ini akan mempersingkat waktu pengetesan dan perbaikan jika ditemukan kesalahan logika. Selain itu, hasil berupa visual akan dapat langsung ditunjukkan kepada pihak IT pelanggan untuk melakukan validasi hasil yang sesuai.



Gambar 3.12 Output respons menggunakan Swagger.

f. Pengembangan desain lebih lanjut.

Praktikan menyadari bahwa desain arsitektur *Microservices* yang ditawarkan belum sempurna hingga 100%, karena desain *Microservices* dapat dikatakan sempurna jika diterapkan dengan menggunakan *container* seperti Kubernetes. Kubernetes adalah platform *open source* untuk mengelola kumpulan kontainer dalam suatu *cluster server*. Platform ini pertama kali dikembangkan oleh Google dan kini dikelola oleh *Cloud Native Computing Foundation* (CNCF) sebagai platform manajemen kontainer yang cukup populer. (Salsabila, 2021).



Gambar 3.13 Transformasi arsitektur *microservices*
Microservices Container (Niagahoster,2021)

Kelebihan melakukan transformasi dari *modular dependency injection services* ke *container* adalah melakukan *scale up* dan *balancing* performa. Hal ini dikarenakan setiap *container* tersebut memiliki *environment* masing-masing dengan sumber daya, CPU dan *file* sendiri, hal ini akan mengurangi *downtime* ketika terjadi komunikasi antar aplikasi *microservices*.

3.3 Kendala Yang Dihadapi

Dalam perancangan awal, Praktikan melakukan pendekatan menggunakan metode *Code-First* (CF) *Entity Framework database* dengan menggunakan *Framework* .NET Core 5. Metode CF ini secara otomatis akan membandingkan skema *class* yang ada di pemrograman dan dibandingkan secara otomatis dengan yang ada di *database*. Akibatnya jika ada ketidaksesuaian, sistem akan melakukan *override* terhadap *database* dan dapat mengakibatkan semua skema data rusak secara keseluruhan.

Dalam prakteknya, metode CF ini dengan konfigurasi otomatis melakukan perbandingan database dan akan menjalankan *migration script*, maka akan mengakibatkan *database* akan menjalankan proses *drop-create* yang jika terjadi perbedaan skema maka akan menghapus database yg ada, termasuk produksi dan tentu sangat fatal. Dalam alur pemrograman yang ada di PT. Enerren Technologies, agak kurang cocok menggunakan metode ini karena dinamika perubahan skema *database* yang cepat dan kadang dilakukan langsung pada *database* tanpa memerhatikan *class object* terlebih dahulu. maka praktikan perlu melakukan beberapa perubahan konfigurasi yang agak berbeda dengan umumnya metode CF yang dilakukan pada *Framework* Microsoft .NET.

Selain mengenai CF, terdapat kendala lainnya, yaitu pada saat terjadi request melalui *Application Program Interface* (API), dapat dipastikan terjadi request data dengan jumlah yang sangat banyak, misalkan pada saat mengambil data posisi kendaraan maka sistem akan melakukan pengambilan data lebih dari 1000 baris dalam sehari untuk satu kendaraan. Hal ini dapat mengakibatkan *load* data yang sangat besar dan menyebabkan terjadinya *hang* atau *timeout*.

Microservices memang memiliki banyak manfaat namun untuk melakukan implementasinya tergolong sulit dan memerlukan banyak pengetahuan mengenai arsitektur desain aplikasi, selain itu *Microservices* merupakan bentuk implementasi terdistribusi yang mana dapat mengakibatkan kegagalan jaringan (*network failed*) ketika salah melakukan implementasinya.

3.4 Cara Mengatasi Kendala

Dalam mengatasi permasalahan yang ada mengenai *Code-First* (CF) *Entity Framework*. Maka, perlu dilakukan beberapa perubahan pada saat deklarasi *class database context*. Pada level *class initialize migration* yang ada pada sub-project TMS.DataProvider maka perlu dilakukan perubahan dari yang tadinya menggunakan *properties* *DropCreateDatabaseAlways<namaContext>* atau *DropCreateDatabaseIfModelChanges<namaContext>* menjadi *DbContextOptions* saja tanpa mengaktifkan Initializernya.

Sedangkan pada permasalahan pengambilan data dalam jumlah yang sangat besar yang berakibat terjadinya *hang* atau *timeout*, kita dapat melakukan penerapan *lazy loading,* menurut Ishaniaggarwal *lazy loading* adalah sebuah teknik *on demand loading* yang mana ketika melakukan penarikan data, disesuaikan dengan kebutuhannya dan dipecah-pecah menjadi per proses bagian. Tujuan dari metode pengambilan data ini adalah untuk mengurangi penggunaan *resources* yang berlebihan. (Ishaniaggarwal, 2018).

Dalam penanganan mengatasi kendala disebabkan oleh implementasi *Microservices* yang dapat mengakibatkan kegagalan, maka dapat dilakukan beberapa cara yaitu :

1. Melakukan *sharing knowledge* secara berkala dan intensif diantara tim developer. Hal ini dimaksudkan untuk mengurangi kesenjangan antara pengetahuan *developer* yang satu dengan lainnya dan memperkecil *learning curve*.
2. Dalam mengatasi implementasi yang kurang baik dalam hal jaringan, selain menggunakan *load balancer* pada *server* dapat juga dengan membatasi penggunaan resources yang berlebihan diantaranya dengan

memperkecil penggunaan objek *blob* pada *database*, melakukan *resize* untuk arsip yang diunggah serta memakai terapan *lazy loading*.

3.5 Pembelajaran Yang Diperoleh dari Kerja Profesi

Praktikan mendapatkan kesempatan untuk mendapatkan banyak manfaat dari Kerja Profesi ini, Salah satu yang paling utama adalah praktikan mendapatkan pengalaman dan bimbingan langsung dari pendamping kerja untuk membuatkan desain arsitektur yang tepat untuk Sistem Informasi Geografis (SIG) yang merupakan bisnis utama dari perusahaan PT. Enerren Technologies. Selain itu, praktikan juga mendapatkan kesempatan untuk mengaplikasikan teori-teori yang selama ini diperoleh selama berkuliah di prodi Sistem Informasi (SIF) Universitas Pembangunan Jaya (UPJ). Dalam proses membuat rancangan arsitektur *Microservices*, praktikan mendapatkan pengalaman-pengalaman berharga dalam rangka menyelesaikan permasalahan dan kendala selama proses pembuatan arsitektur tersebut yang mana turut mengasah kemampuan berpikir logis dan sikap pengambilan keputusan yang cepat dalam menyelesaikan permasalan-permasalahan yang timbul selama proses pembuatan desain arsitektur sistem.