BAB V
PENUTUP

5.1. Kesimpulan

Berdasarkan hasil analisis dan perancangan pada aplikasi *Transportation Management System* (TMS) menggunakan *Dependency Injection* (DI), dapat diambil beberapa kesimpulan, yaitu :

1. Untuk mencegah terjadinya *human error* ketika proses pembaruan *database* pada aplikasi TMS, dapat dilakukan dengan menggunakan pembaruan *database* Code-First. Dengan menggunakan CF, maka pembaruan dapat dilakukan secara otomatis melalui perintah *migration* yang akan mengurangi kemungkinan terjadinya *human error*. Namun, aplikasi dengan metode Code-First (CF) harus sangat berhati-hati ketika menerapkan perintah migrasi *database*, jika salah penggunaan maka sangat fatal akibatnya karena bisa menghapus seluruh isi *database* yang dituju. Maka, diperlukan kehati-hatian dan pengetahuan mengenai strategi migrasi *database* tersebut.
2. Aplikasi TMS yang *tightly-coupled* dapat diatasi dengan cara penerapan aplikasi dengan metode *Dependency Injection* (DI), hal ini membuat aplikasi jauh lebih fleksibel dan *loosely coupled* bila dibandingkan dengan ketika menggunakan pemodelan arsitektur aplikasi monolitik yang bersifat *tightly coupled*. Selain itu, aplikasi ini dibagi atas komponen-komponen yang lebih kecil dan membuat pengembangan dan pemeliharaan aplikasi menjadi lebih mudah
3. Proses pembaruan versi database berurutan dan seragam pada aplikasi TMS dapat dilakukan dengan menerapkan konsep *database Code-First* (CF), maka aplikasi akan bersifat lebih suportif dalam mengakomodir *update database* pada klien meskipun tersebar di berbagai lokasi *server*.

5.2. Saran

Aplikasi dengan pemodelan *Dependency Injection* (DI) membutuhkan alur pembelajaran yang lumayan panjang pada masa awal-awal adaptasi ke *framework* yang baru. Maka, diperlukan sesi pembelajaran yang banyak dan akan terlihat bahwa *mindset* pengembangan dari yang sebelumnya menggunakan pemodelan

aplikasi monolitik secara penuh ke transisi menggunakan komponen-komponen yang dapat disusun secara *modular*.

Dalam penerapan aplikasi TMS, diperlukan pembagian modul yang lebih matang dan lebih independen. Pada pengembangan aplikasi versi yang tertuang masih terdapat beberapa komponen yang sebenarnya masih dapat dibuat versi independennya contohnya seperti komponen untuk modul *service*, masih terjadi *tight coupled* dengan modul domain data utama. Akan jauh lebih baik jika modul *service* dapat berdiri sendiri dan digunakan pada domain dan DB *context* yang lain. Sebab, tidak semua aplikasi turunan dari TMS akan menggunakan *service* tersebut, dengan demikian maka aplikasi akan jauh lebih fleksibel.

Selain itu, pada versi aplikasi perancangan saat ini, masih belum dapat menggunakan berbagai macam bahasa pemrograman sebagai *presentation tier* sebagaimana yang diharapkan sebagai salah satu tujuan dari perancangan yang diajukan. Hendaknya di kemudian hari, perancangan *framework* aplikasi akan bisa lebih suportif terhadap beberapa bahasa pemrograman terutama pada *presentation tier*. Adapun *application/service tier* tetap menggunakan perancangan yang ada saat ini.