BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Dalam berlangsungnya kegiatan magang di PT Cranium Royal Aditama, posisi yang diisi adalah *Fullstack Developer Intern* yang dibimbing oleh Bapak Sugito yang merupakan *Vice President Engineering* di perusahaan tersebut sekaligus sebagai *Team Leader* di proyek pembuatan ERP.

3.2 Tugas yang Dilakukan

Tugas utama yang diberikan adalah merancang sebuah sistem ERP sebagai produk utama dari Cranium Indonesia. Arsitektur yang digunakan adalah *modular monolith architecture* dengan bahasa Java Spring Boot sebagai *backend* dan juga React Js sebagai *frontend*. Tugas pertama yang diberikan adalah mempelajari *tech stack* yang diperlukan dalam pengembangan sistem ERP. Elemen-elemen tersebut meliputi *framework Spring Boot*, sistem ERP, serta arsitektur *modular monolith*.

Setelah waktu pembelajaran selesai, diberikan tugas kedua yaitu mendesain ER diagram dan juga ER *description* berdasarkan modul yang dikerjakan. Maka dalam kasus ini, penulis bersama tim mengerjakan ER diagram dan ER *description* dari modul purchasing, inventory dan juga production. Dalam tahap ini, pengembang juga diharapkan untuk membuat *database migration* untuk setiap modul yang dikerjakan.

Tugas selanjutnya adalah mulai mengerjakan modul sesuai dengan tim yang telah dibagi. Tahap ini diawali dengan pembagian submodul dalam tim sehingga satu orang mendapatkan tiga sampai empat submodul. Dalam tahap ini submodul yang dikerjakan diharapkan dapat melakukan CRUD (*Create*, *Read*, *Update* dan *Delete*) termasuk dengan unit test masing-masing proses.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu ke-	**Aktivitas yang dikerjakan**
1	Mempelajari tentang sistem ERP. Memasang hal yang dibutuhkan dalam proyek ERP dan mempelajari konsep dasar *Spring Boot* dan Jakarta Persistence API
2	Mengimplementasikan Spring Boot dan Jakarta Persistence API dalam sebuah proyek serta mempelajari sistem ERP yang lama
3	Melanjutkan pembelajaran Spring khususnya di bagian keamanan dan mempelajari arsitektur yang digunakan dalam proyek ERP
4	Melakukan clonning submodul dari *template* yang diberikan pembimbing dan membuat *method* CRUD serta menambahkan autorisasi di setiap *method*
5	Menambahkan *field* di submodul dan juga validasi di dalam DTO
6	Melakukan clonning di *template* yang bar diberikan oleh pembimbing dan membuat *method* CRUDnya serta memulai membuat ER diagram dan ER description modul kelompok
7	Menyelesaikan ER diagram dan ER description, membuat unit test di submodul
8	Merevisi ER diagram dan ER description sesuai dengan schema database dari sistem ERP yang lama yang diberikan oleh pembimbing
9	Mulai mengerjakan submodul dan membuat method CRUD untuk header dan juga detail
10	Membuat unit test submodul
11	Menyelesaikan unit test submodul dan memperbaiki error yang ada
12	Mulai mengerjakan submodul selanjutnya dan membuat *method* CRUD untuk submodul tersebut
13	Memperbaiki error di contract test dan melanjutkan pekerjaan ke submodul item withdraw
14	Membuat *method* CRUD dan membuat unit test submodul item withdraw
15	Menyelesaikan unit test item withdraw dan melanjutkan pekerjaan ke submodul item receipt

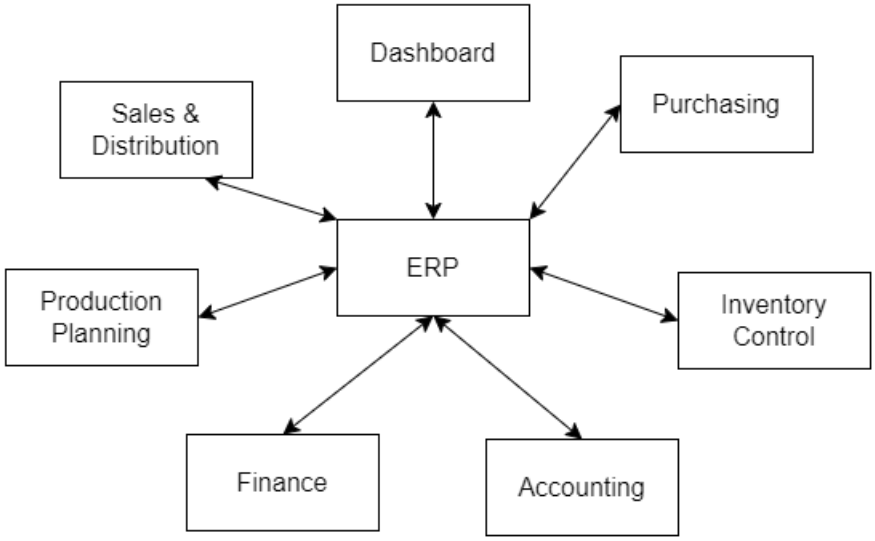
Continued on next page

Tabel 3.1 – dilanjutkan dari tabel sebelumnya

Minggu ke-	Aktivitas yang dikerjakan
16	Menyelesaikan *method* CRUD item receipt dan membuat unit test item receipt
17	Menyelesaikan unit test item receipt dan melanjutkan pekerjaan ke submodul production order item receipt
18	Membuat *method* CRUD production order item receipt dan revisi unit test modul purchasing
19	Menyelesaikan revisi dan membantu tim master membuat submodul warehouse dan supplier
20	Menyelesaikan submodul warehouse dan supplier. Lalu, melanjutkan pekerjaan ke submodul salesman
21	Menyelesaikan *method CRUD* dan unit test submodul salesman dan menambahkan authiry serta scope di user

3.3.1 Pengenalan Proyek ERP

Pengenalan proyek ERP dilakukan untuk membantu pengembang mengetahui alur jalannya sistem ERP secara garis besar. Pengenalan mengenai sistem ERP yang telah dikembangkan sebelumnya dilakukan dengan adanya bantuan dari Bapak Erpan selaku *project manager* di pengembangan sistem ERP sebelumnya dan juga sebuah *website* demo untuk sistem ERP tersebut untuk mempermudah pembelajaran.



Gambar 3.1. Gambaran besar modul-modul ERP

Gambar 3.1 merupakan gambaran besar dari modul-modul dari sistem ERP lama yang dijelaskan oleh Bapak Erpan. Dalam pengembangan sistem ERP baru ada beberapa modul yang diubah, dihilangkan maupun ditambahkan.

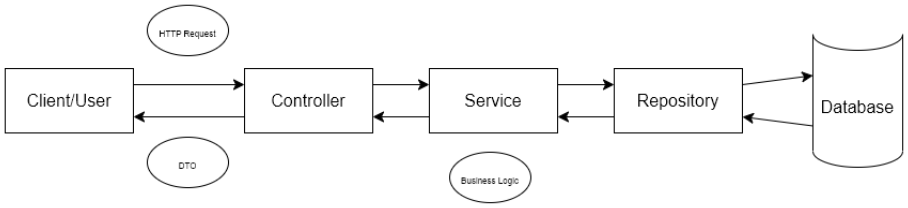
3.3.2 Pembelajaran Tech Stack

Tech stack dalam proyek ERP yang dikerjakan meliputi Java Spring Boot sebagai *framework* dan PostgreSQL sebagai *database managemement system*. Bapak Sugito sebagai *supervisor* memberikan waktu sebanyak dua minggu untuk mempelajari Java Spring Boot dan PostgreSQL secara mandiri. Pembelajaran mandiri ini dilakukan dengan referensi dari internet. Dimulai dari konsep dasar dari Java Spring Boot itu sendiri sampai dengan percobaan mandiri penggunaan Java Spring Boot dan PostgresSQL dilakukan supaya terbiasa dengan *tech stack* yang digunakan.

3.3.3 Pengenalan Arsitektur

Bapak Sugito juga membantu pengembang untuk mempelajari arsitektur *modular monolith*. Pada awalnya, Bapak Sugito memberikan gambaran umum bagaimana bentuk dan juga jalannya arsitektur yang digunakan. Seiring berjalannya waktu, pembimbing juga memberikan sebuah *template* yang menggunakan arsitektur *modular monolith*, lalu pengembang diminta untuk mencoba *template*

tersebut dan membuat modul baru berdasarkan *template* tersebut. Modul yang baru dibuat tersebut diharapkan dapat melakukan proses CRUD (*Create*, *Read*, *Update* dan *Delete*). Pada saat ini, dilakukan juga pembagian tim berdasarkan modul-modul yang akan dibuat. Tim penulis mendapatkan modul purchasing, inventory dan juga production.



Gambar 3.2. Alur kerja arsitektur

Gambar 3.2 merupakan alur kerja arsitektur dalam pengembangan sistem ERP.

A. DTO

DTO atau *Data Transfer Object* merupakan sebuah *object* yang berisi data dalam setiap proses. DTO digunakan saat transfer *object* antara *client* dan *server*. Dalam pengembangan sistem ERP ini terdapat tiga macam DTO yang digunakan. Pertama adalah response DTO yang merupakan respons yang dapat dilihat *user* ketika melakukan HTTP request. Selanjutnya adalah create DTO yang merupakan object yang dibuat oleh pengguna saat hendak membuat data baru atau melakukan method create. Lalu, update DTO yang merupakan *object* yang dibuat *user* saat melakukan perubahan kepada data yang sudah ada di database, Update DTO memiliki *field* ”version” yang berguna untuk memastikan bahwa data yang sedang diubah di sisi klien sama dengan yang ada di dalam database.

B. Controller

Controller dalam arsitektur ini berhubungan dengan langsung karena controller yang melakukan proses terhadap API request dari *user*. Selain itu, controller juga yang memberikan respons kepada *user* saat request selesai.

C. Service

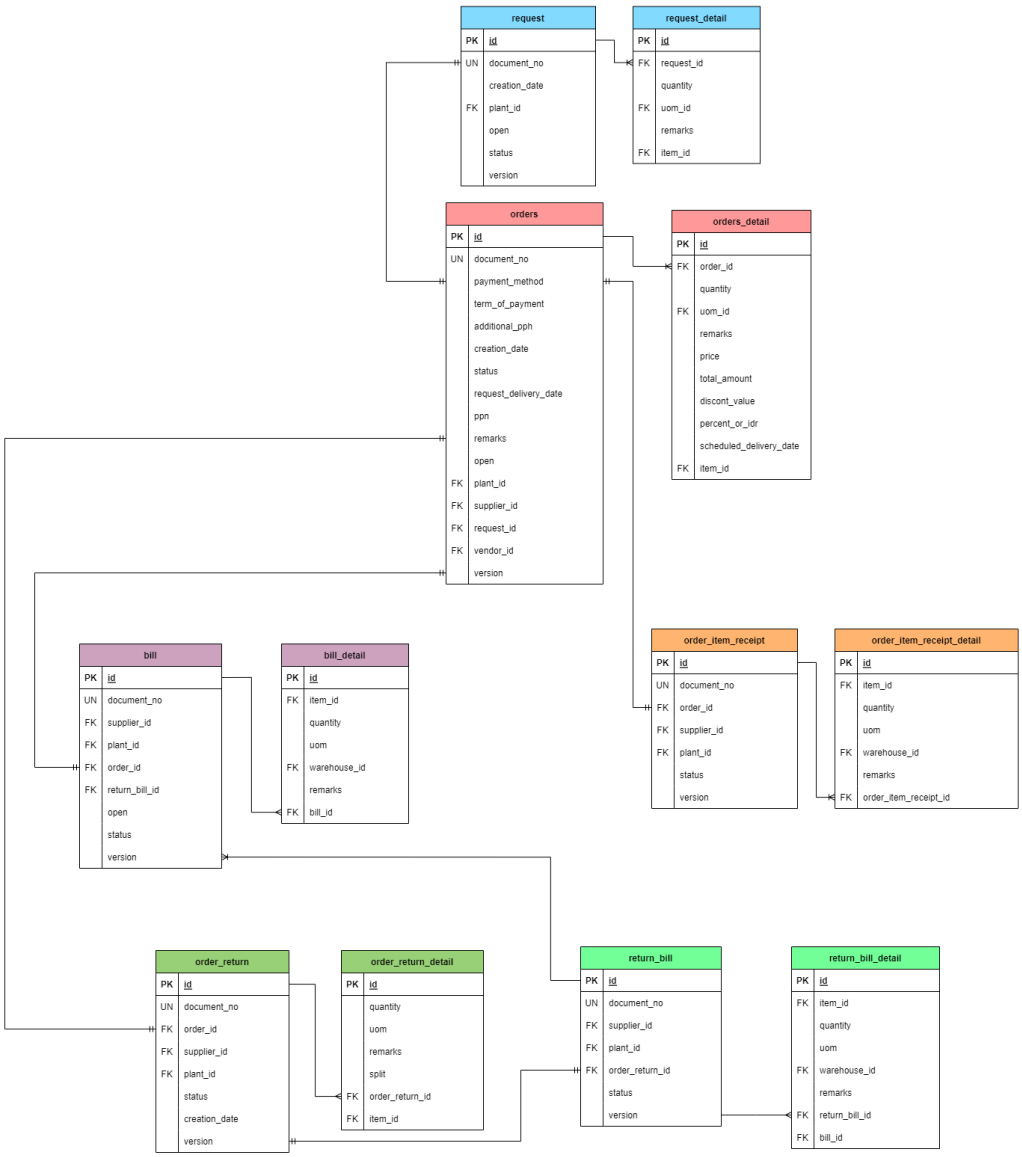
Service merupakan sebuah class yang berisikan *business logic* atau pun algoritme lain yang mengolah data.

D. Repository

Repository merupakan sebuah *interface* yang berisikan berbagai *method* yang berhubungan dengan database. Di service, *method* yang ada di repository ini akan dipanggil saat melakukan hal yang berkaitan dengan database.

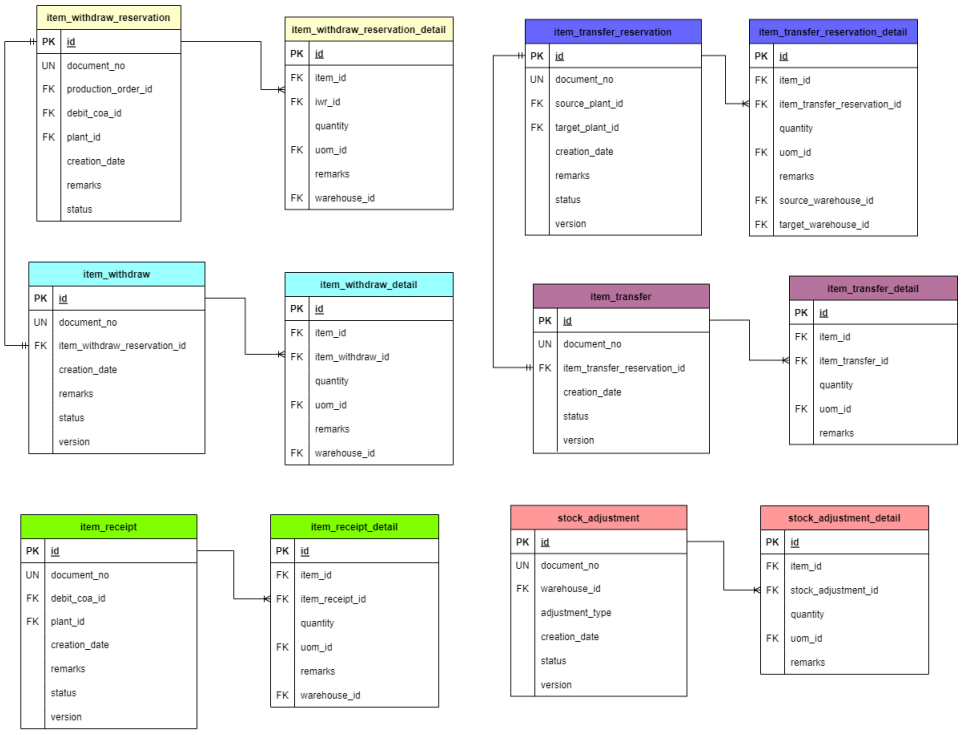
3.3.4 Entity Relation Diagram

Salah satu tugas awal yang diberikan adalah membuat ERD atau *Entity Relation Diagram* dari modul yang dikerjakan. Dalam hal ini ERD yang dikerjakan bersama tim adalah ERD dari modul purchasing, inventory dan production.



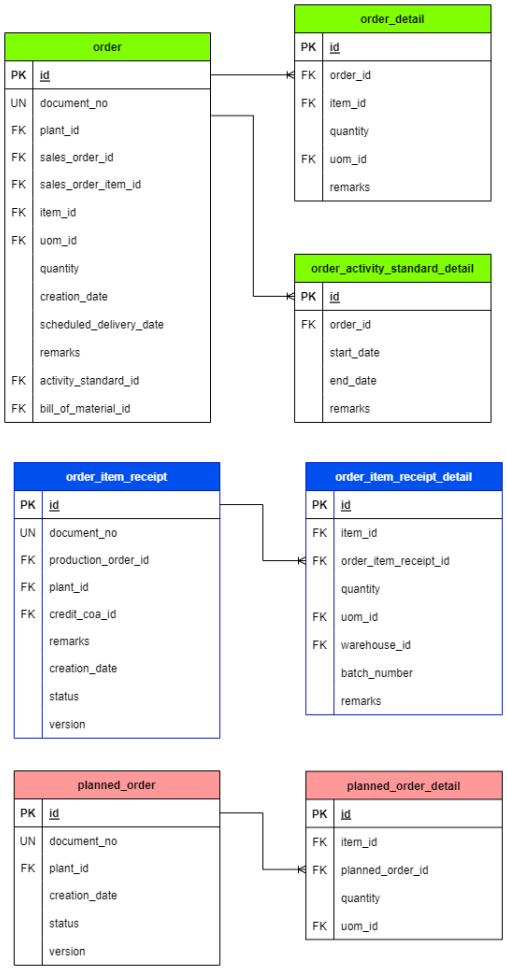
Gambar 3.3. ERD modul purchasing

Gambar 3.3 dari ERD modul purchasing. Setiap submodul dari modul purchasing terdiri dua bagian, yaitu tabel utama atau *header* dan juga tabel detail. Setiap tabel utama dari sebuah submodul memiliki relasi *one to many* dengan tabel detailnya. Yang berarti satu tabel *header* dapat memiliki banyak detail di dalamnya. Di dalam modul purchasing sendiri terdapat submodul, yaitu purchase request, purchase order, purchase order item receipt, purchase bill, purchase order return, dan juga purchase return bill.



Gambar 3.4. ERD modul inventory

Gambar 3.4 adalah gambar dari ERD modul inventory yang dibuat oleh tim penulis. Sama seperti modul purchasing, modul inventory juga memiliki tabel utama atau *header* dan juga tabel detail untuk setiap submodulnya. Submodul dari inventory sendiri terdiri dari item withdraw reservation, item withdraw, item transfer reservation, item transfer, item receipt, dan stock adjustment

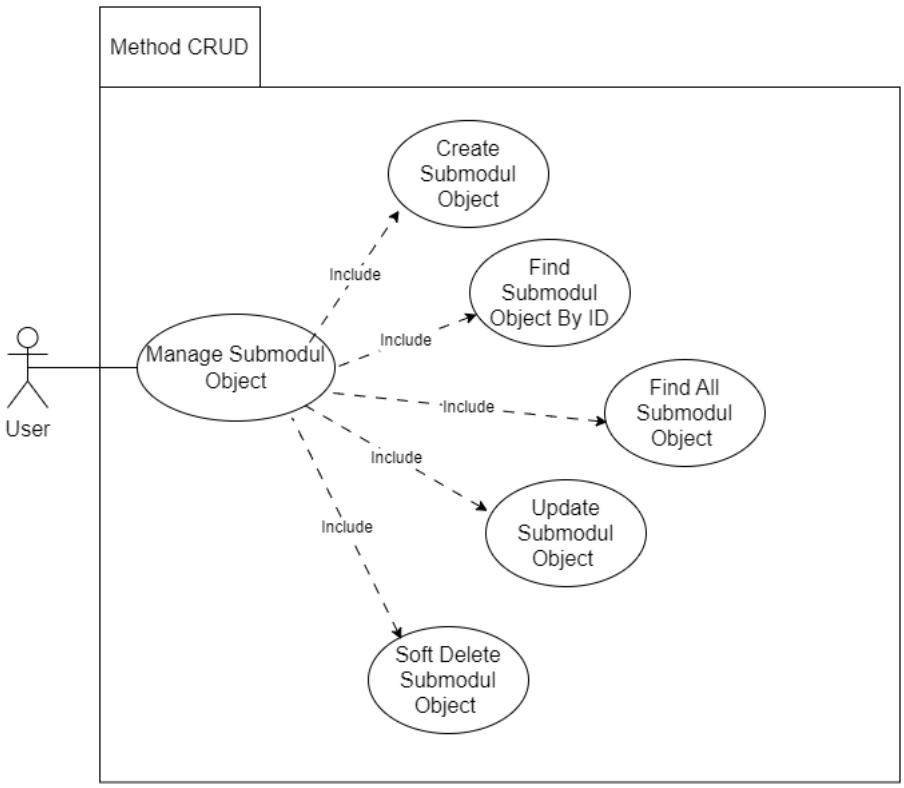


Gambar 3.5. ERD modul production

Gambar 3.5 di atas merupakan gambar dari ERD modul production. Sama seperti dua modul sebelumnya, modul production ini juga memiliki tabel *header* dan juga tabel detail untuk setiap submodulnya. Submodul dari modul ini meliputi production order, production order item receipt, dan planned order.

3.3.5 Use Case Diagram

Use case diagram merupakan sebuah diagram untuk menunjukkan jalannya sebuah sistem dengan menggunakan sebuah diagram.

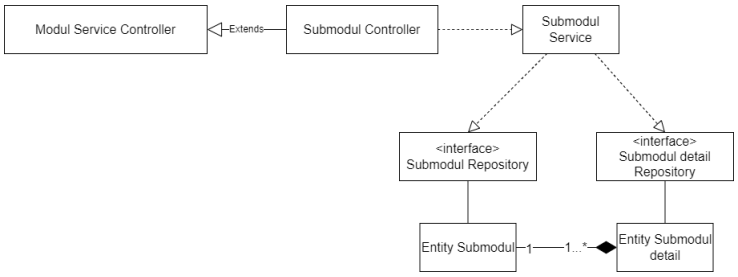


Gambar 3.6. *Use Case Diagram* ERP

Yang dimaksud dengan *submodul object* pada Gambar 3.6 adalah data dari setiap submodul yang ada dalam sistem ERP. Pada Gambar 3.6 tersebut terlihat bahwa *user* atau pengguna dapat mengelola data dari submodul yang meliputi *create* atau membuat data, *find by id* atau mengambil data sesuai dengan id tertentu, *find all* atau mengambil semua data submodul, *update* atau mengubah isi data submodul dan juga *soft delete* atau membuat sebuah data tidak valid tanpa menghapusnya dari database.

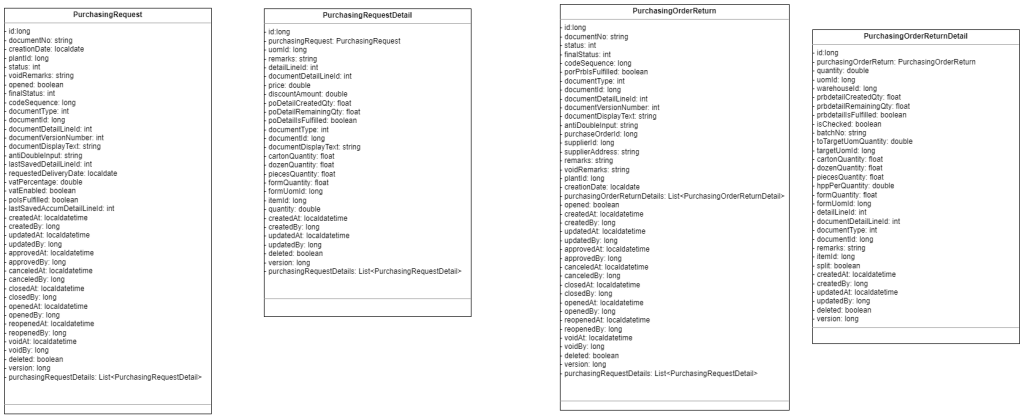
3.3.6 Class Diagram

Gambar 3.7 merupakan representasi relasi antar class yang ada di sistem ERP dengan sebuah *class diagram*.

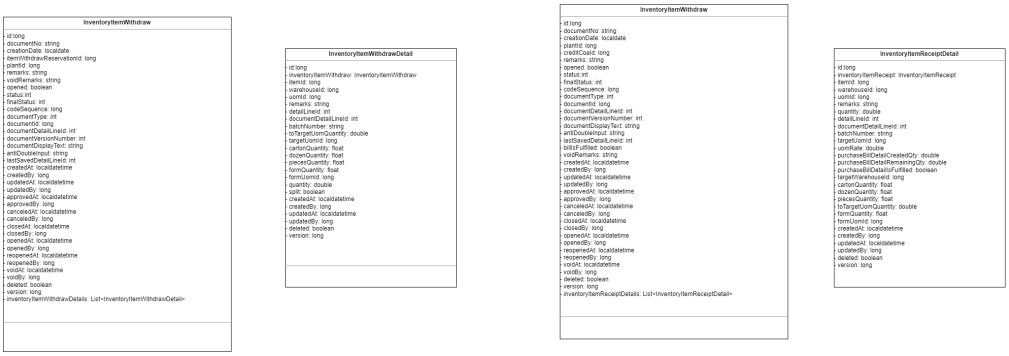


Gambar 3.7. Relasi *Class Diagram*

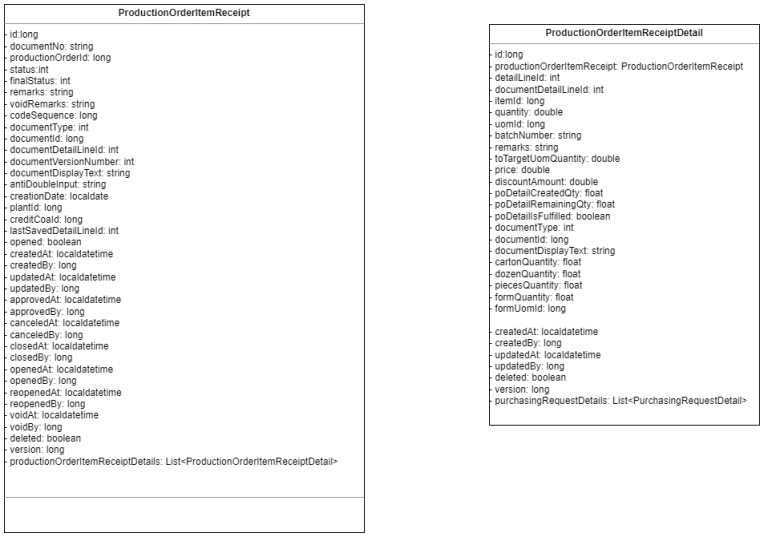
Class diagram pada Gambar 3.7 menampilkan relasi antar class yang ada di setiap submodul secara umum. Submodul yang termasuk adalah purchase request, purchase order retuen, item withdraw, item receipt dan juga production order item receipt. Semua controller dari tiap submodul berhubungan dengan Modul Service Controller dengan jenis hubungan *generalization* atau *inheritance*. Entity header dengan entity detail memiliki relasi *one to many* yang digambarkan dengan simbol *composition*. Hubungan antara controller dengan service dan juga hubungan antara service ke repository disimbolkan dengan *dependency*.



Gambar 3.8. *Class Diagram entity* modul purchasing



Gambar 3.9. *Class Diagram entity* modul inventory

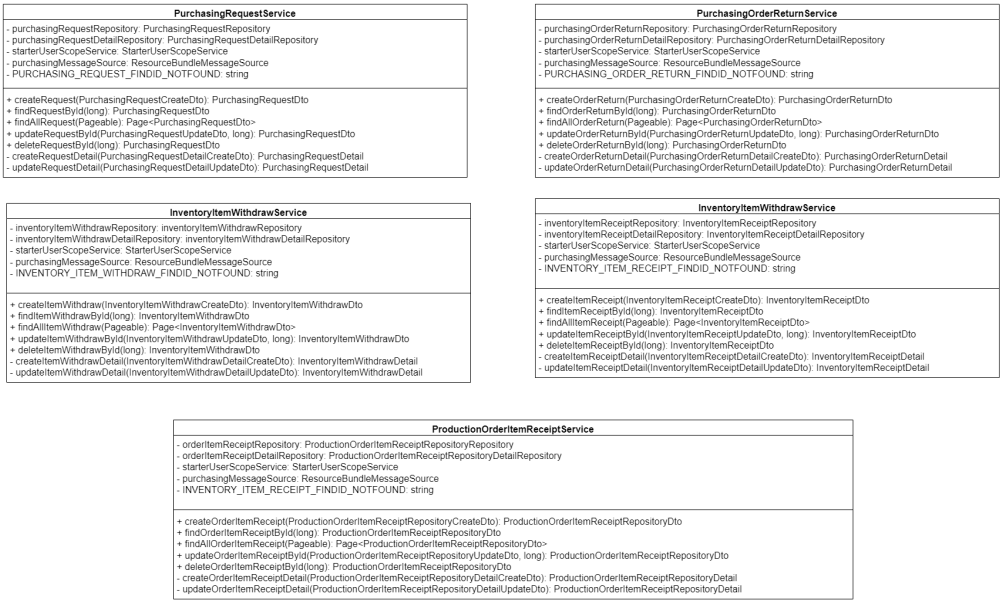


Gambar 3.10. *Class Diagram entity* modul production

Gambar 3.8, 3.9, dan 3.10 menunjukkan *class diagram* untuk class entity pada submodul yang dikerjakan.

Gambar 3.11. *Class Diagram* controller

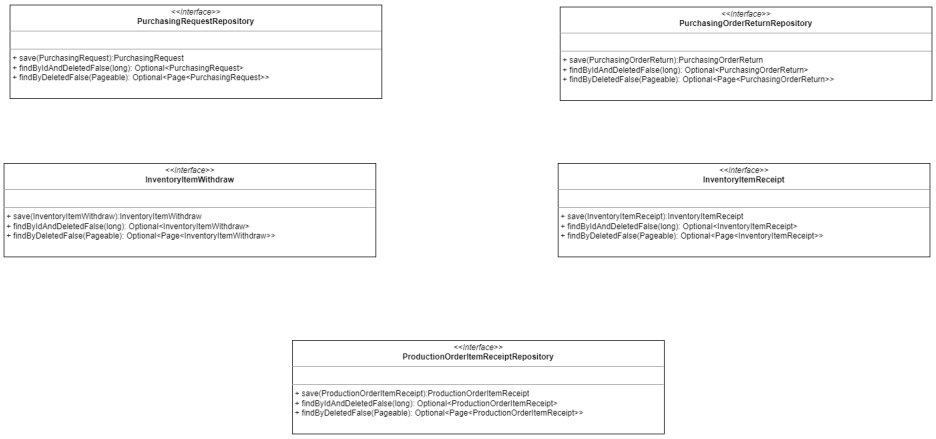
Gambar 3.11 menunjukkan class diagram dari controller pada setiap submodul yang dikerjakan. Di dalam controller terdapat service dari submodul yang dipanggil serta method-method yang akan dipanggil saat user melakukan *hit* terhdapat API.



Gambar 3.12. *Class Diagram* service

Gambar 3.12 menunjukkan class diagram dari service pada setiap submodul yang dikerjakan. Dalam setiap class service terdapat pemanggilan repository, lalu pemanggilan starter user dan juga ResourceMessageSource serta messagenya itu sendiri untuk melakukan *message handle*. Adapun method-method di dalam service yang meliputi proses CRUD dan *find all* yang juga berisi *business logic*.

Method-method ini nantinya akan dipanggil oleh controller saat user melakukan *hit* terhadap API. Selain itu, ada juga method pembantu yang bersifat *private* yang akan dipanggil oleh method utama saat menjalankan fungsinya.



Gambar 3.13. *Class Diagram* repository

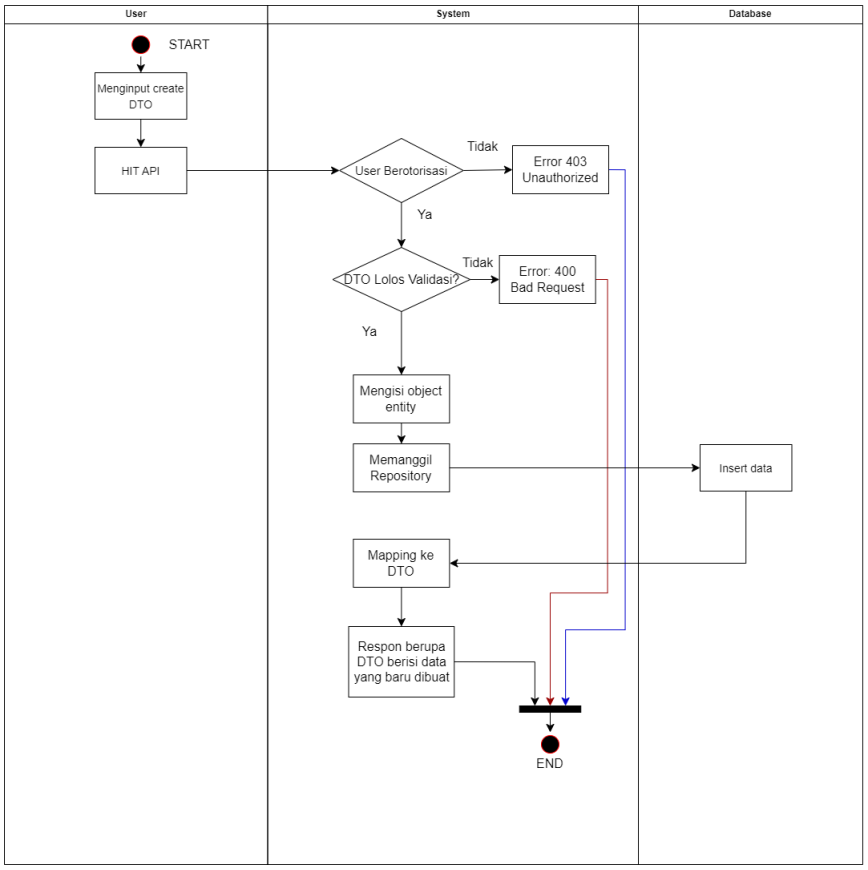
Gambar 3.13 adalah class diagram dari repository tiap submodul yang dikerjakan. Repository bertipe interface yang mana method-method yang ada di dalamnya akan dipanggil oleh service saat service ingin berhubungan dengan database.

3.3.7 Pembuatan API Method CRUD

Pengerjaan yang dimaksud merupakan pembuatan *method create, read, update, delete*, dan juga *find all* dari setiap submodul yang telah dibagi setiap tim. Dalam hal ini, penulis mendapat bagian untuk mengerjakan submodul purchase request, purchase order return dari modul purchasing. Lalu item withdraw, item receipt dari modul inventory dan production order item receipt dari modul production.

A. Create

Method create dalam submodul digunakan *user* untuk menyimpan data ke database. Jika digambarkan dalam sebuah *activity diagram*, *method create* akan terlihat seperti gambar 3.14.



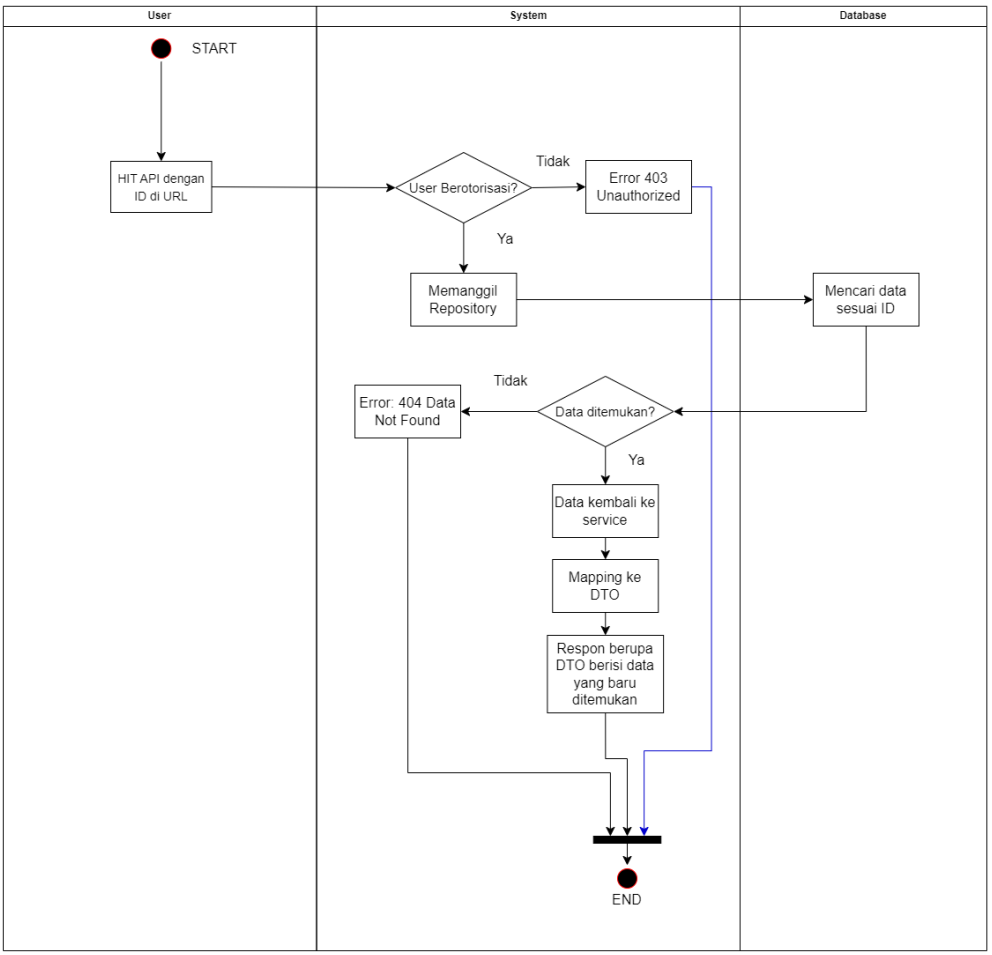
Gambar 3.14. *Activity diagram method create*

Method create akan diawali dengan *user* memasukkan data berupa create DTO dalam bentuk JSON, lalu API *layer* atau controller akan dipanggil dan controller akan memanggil service yang berisi *business logic*. Namun, sebelum masuk ke service layer akan dilakukan validasi. Validasi ini dapat berbeda di setiap submodulnya. Namun, secara sederhana validasi ini dilakukan supaya tidak ada data yang *null* atau kosong dalam *create* DTO tersebut. Jika create DTO yang di-*input* pengguna tidak lolos validasi, sistem akan mengeluarkan error Bad Request dengan kode 400. Namun, jika create DTO tersebut tervalidasi, data akan masuk ke service dan di dalam service ini, data dari *create* DTO yang juga ada di dalam *entity* dari sebuah submodul akan digunakan untuk mengisi *object entity*, lalu data yang tidak ada dalam *object entity* akan dilengkapi dan diolah di service. Setelah selesai mengolah data, service akan memanggil repository untuk menyimpan data yang telah diolah. Repository akan menyimpan data ke database. Lalu, repository akan mengembalikan data berupa *object entity* yang telah disimpan di database ke service. Setelah itu, service akan melakukan *mapping* terhadap *object*

entity tersebut menjadi DTO yang menjadi response DTO. *Response* DTO tersebut dikembalikan ke *user* kembali dalam bentuk JSON.

B. Read

Method read dalam submodul digunakan *user* untuk membaca data yang ada dalam database sesuai dengan id yang ada dalam URL yang digunakan oleh *user*. Jika digambarkan dalam *activity diagram*, maka alur method read dapat dilihat di gambar 3.15.



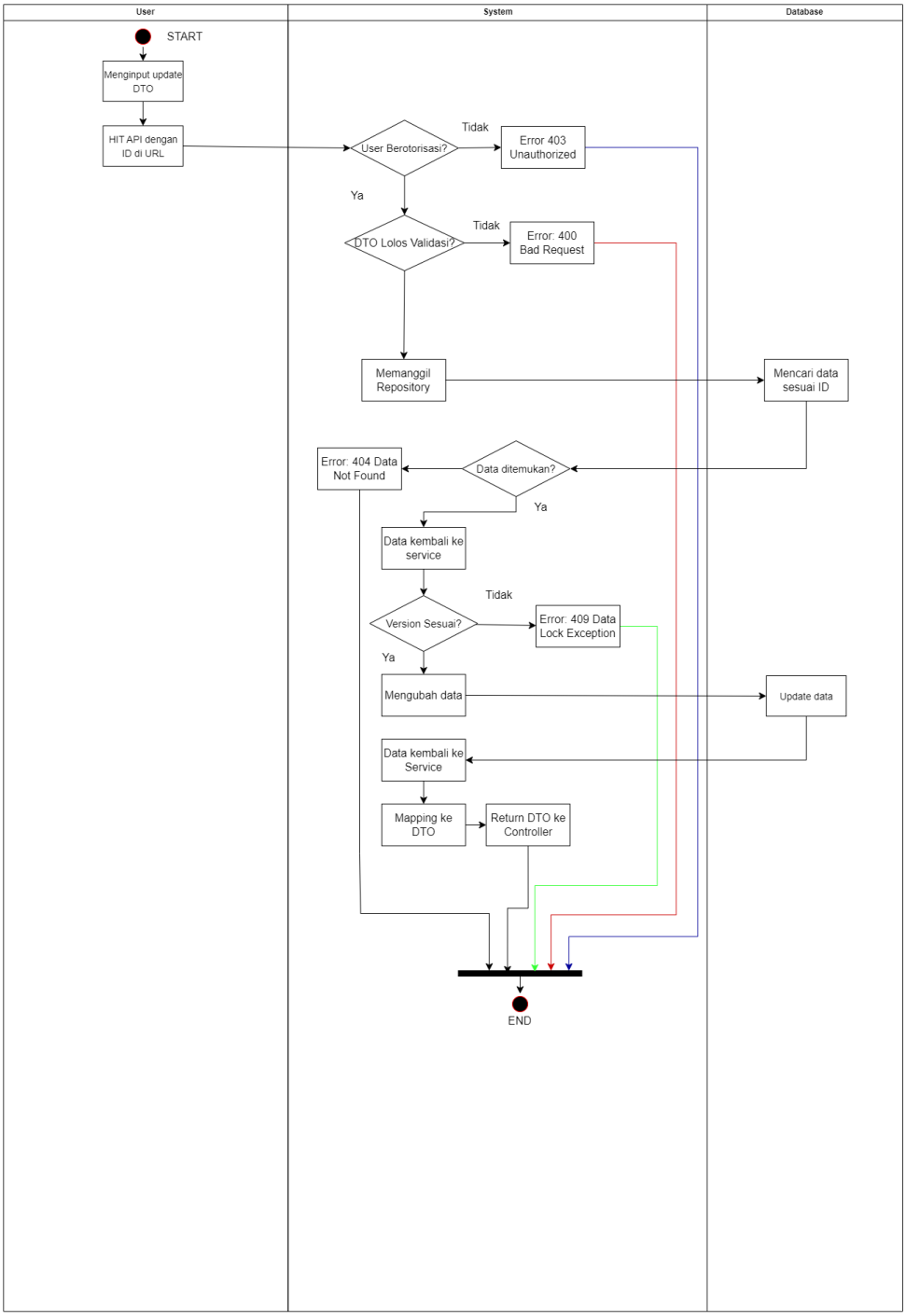
Gambar 3.15. *Activity diagram method read*

Berbeda dengan *method create*, *method read* tidak memerlukan *input* data dari *user*. Namun, karena *method read* membaca sebuah data berdasarkan id, maka *method* ini membutuhkan id yang mana dituliskan dalam URL yang digunakan *user*. Dari URL tersebut, controller akan dipanggil dan controller akan memanggil

service. Di dalam service, repository akan dipanggil untuk mencari data dalam database sesuai dengan id yang diberikan. Jika tidak ditemukan data dengan id tersebut, repository akan mengembalikan nilai *null* ke service, lalu service akan mengembalikan error 404 dengan Data Not Found Exception ke controller dan menjadi response ke *user* dengan *message* atau pesan bahwa data dengan id tersebut tidak ditemukan dalam database. Namun, jika ditemukan, repository akan mengembalikan data berupa *object entity* ke service, lalu service akan melakukan *mapping object entity* tersebut menjadi bentuk DTO. Setelah itu, DTO tersebut akan dikembalikan ke controller dan menjadi respons untuk *user*.

C. Update

Method update yang menggunakan HTTP *method* tipe PATCH digunakan untuk mengubah data yang ada di dalam database. *Method* ini menggunakan *update* DTO yang berisikan data yang telah diubah oleh *user*. Jika digambarkan dalam *activity diagram*, *method update* dapat terlihat seperti gambar 3.16.



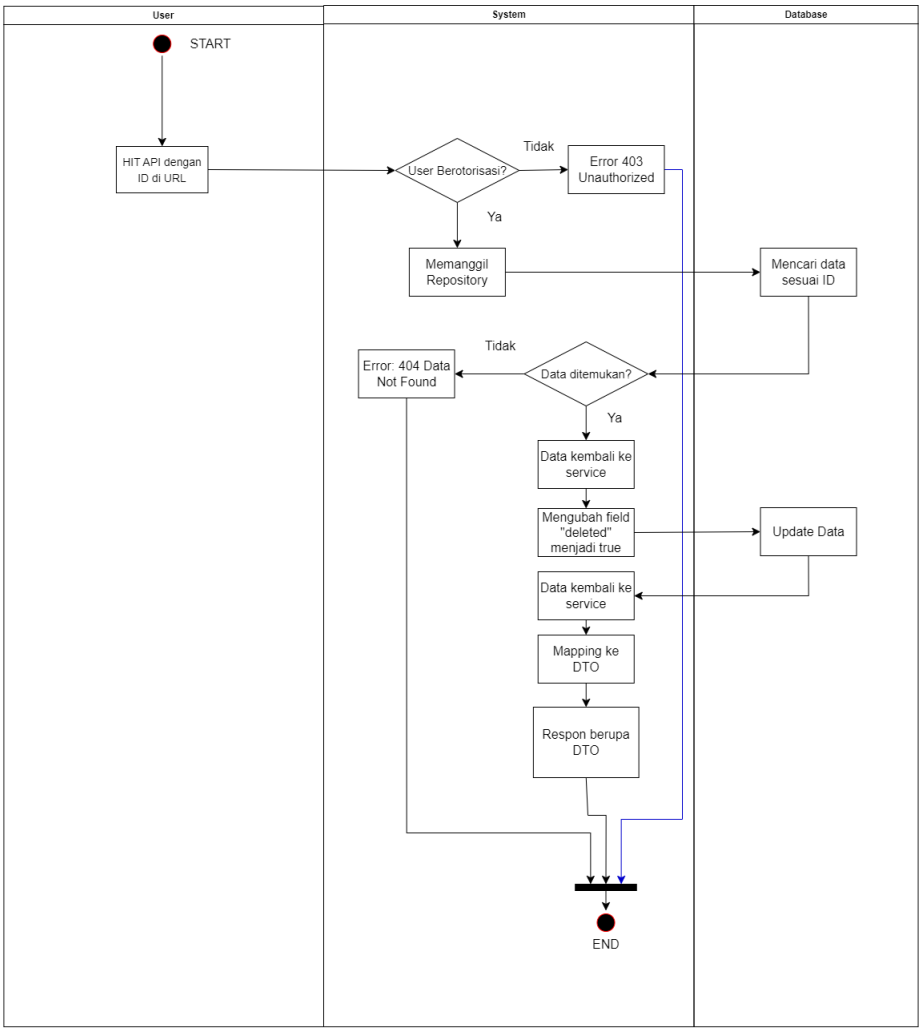
Gambar 3.16. *Activity diagram method update*

Sama seperti *method* lainnya, method update memanggil controller dengan memberikan *update* DTO dan juga id data yang hendak diubah, lalu controller akan memanggil service. Service juga akan menerima *update* DTO dan id dari *user*. Service akan mencari terlebih dahulu data dengan id yang diberikan *user*. Jika

tidak ada, maka repository akan mengembalikan nilai *null* ke service, Jika service menerima *null*, maka service akan mengembalikan error 404 dengan error 404 Data Not Found Exception ke controller dan menjadi respons ke *user* dengan *message* bahwa data yang hendak diubah tidak ditemukan. Sedangkan jika data dengan id tersebut ditemukan, repository akan mengembalikan data ke service dengan bentuk *object entity*. Setelah itu, dilakukan pengecekan terhadap *field* ”version” yang ada di *update* DTO dan yang ada dari data yang dikembalikan oleh repository. Jika berbeda, service akan mengembalikan error 409 dengan Data Lock Exception ke controller dan menjadi respons kepada *user* dengan message bahwa version di update DTO dan di database memiliki nilai yang berbeda. Lalu, jika nilai ”version” dari update DTO dengan nilai ”version” dari data yang ada di database sama, maka akan dilakukan perubahan berdasarkan *input* yang diberikan *user* dan repository kembali dipanggil untuk menyimpan data yang telah diubah tersebut. Setelah menyimpan data ke database, repository akan mengembalikan kembali data ke service dalam bentuk *object entity*. Lalu, service pun akan melakukan mapping dari entity ke DTO. DTO tersebut akan dikembalikan ke controller dan akan diterima oleh *user*.

D. Delete

Method delete menggunakan HTTP method tipe DELETE digunakan untuk menghapus data yang ada dalam database berdasarkan id yang diberikan *user* saat melakukan request. Namun, penghapusan pada sistem ERP ini bersifat *soft delete*, yang artinya data tidak sesungguhnya dihapus, namun hanya diubah *field* bernama ”deleted” menjadi true sehingga tidak ada method lain yang dapat mengakses data tersebut. Jika direpresentasikan oleh *activity diagram*, *method delete* dapat terlihat seperti gambar 3.17.



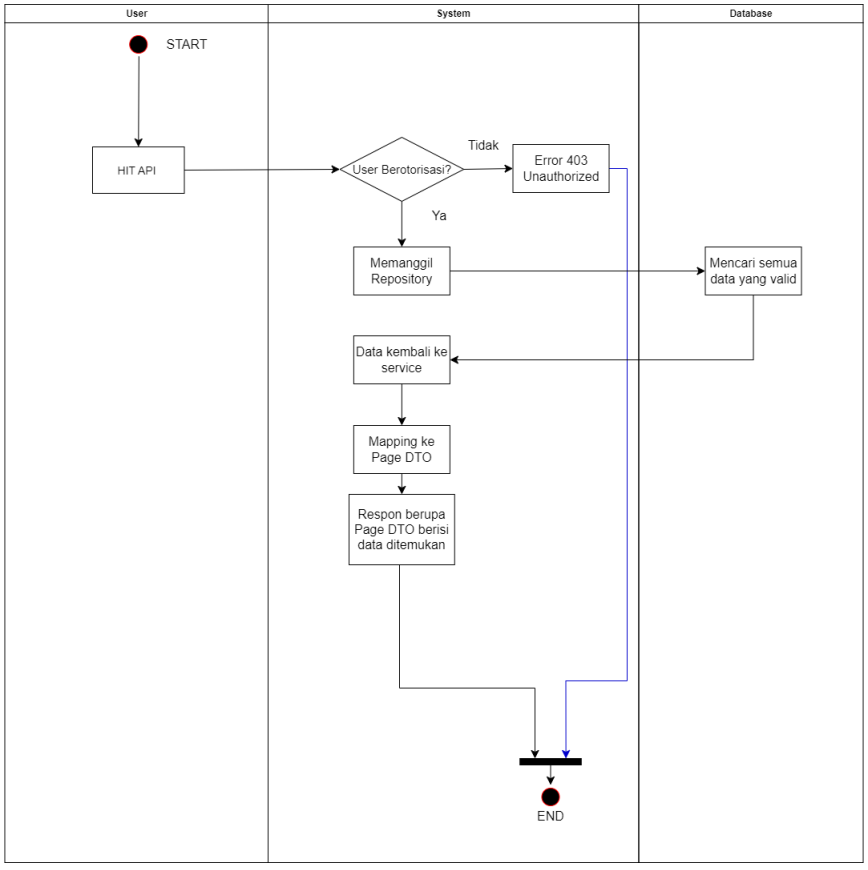
Gambar 3.17. *Activity diagram method delete*

Sama seperti *method read, method delete* juga hanya memerlukan id dari data yang hendak dihapus dan tidak memerlukan *input* lain dari *user*, id dari data yang hendak dihapus tersebut akan diletakkan pada URL saat hendak melakukan *request*. Dari URL tersebut, controller akan dipanggil dan controller akan memanggil service. Service akan menerima id yang diberikan oleh *user*. Lalu, sama dengan *method update*, service akan memanggil repository untuk mencari data dengan id tersebut. Jika data dengan id tersebut tidak ditemukan, maka repository akan mengembalikan *null* ke service. Lalu, service akan mengembalikan Data Not Found Exception atau error 404 kepada controller dan controller akan memberikan *message* bahwa data dengan id tersebut tidak ditemukan. Sebaliknya, jika data dengan id tersebut ditemukan, maka repositroy akan mengembalikkan nilai dari data tersebut kepada service. Selanjutnya, service akan mengubah *field* ”deleted”

dengan tipe data *boolean* menjadi *true* dan service akan kembali memanggil repository untuk menyimpan data yang telah diubah tersebut ke dalam database dan mengembalikannya kembali ke service. Service pun akan melakukan *mapping* dari *object entity* ke DTO dan DTO tersebut akan dikembalikan ke controller sehingga controller mengembalikan DTO tersebut sebagai respons ke*user*.

E. Find All

Method find all digunakan untuk membaca semua data yang belum terhapus dalam database submodul. Sama dengan *method read*, *method find all* ini juga menggunakan HTTP *method* tipe GET dengan mengembalikan tipe data Page. Jika *method* ini digambarkan dalam *activity diagram*, maka dapat terlihat seperti gambar 3.18 di bawah ini.



Gambar 3.18. *Activity diagram method find all*

Dalam *method find all* ini, id dari data tidak diperlukan dalam URL saat *user* melakukan *request*. URL yang melakukan *request* akan melakukan *hit* ke API

dengan memanggil controlelr dan controller akan memanggil service. Selanjutnya, service akan memanggil repository untuk mencari semua data yang masih belum terhapus dalam database submodul. Repository akan mengembalikan data dalam bentuk Page berisi *object entity* kepada service. Lalu, service akan melakukan *mapping* dari Page entity menjadi Page DTO dan DTO tersebut akan dikembalikan ke controller. Controller pun mengembalikkan DTO ini sebagai respons kepada *user*.

3.3.8 Pengetesan API Method CRUD

Pengetesan method CRUD ini dilakukan dengan mencoba api di Postman. Percobaan dilakukan untuk setiap method dan untuk setiap submodul yang dikerjakan, yaitu purchase request, purchase order return, item withdraw, item receipt, dan production order item receipt.

Pengetesan API ini dilakukan dengan menggunakan Postman sebagai API platform. Masing-masing method akan dicoba API nya dengan Postman secara lokal. Adapun Tabel 3.2 yang menunjukkan HTTP *method* dan URL dari setiap *method*.

Tabel 3.2. Detail pengetesan API

Method	**HTTP Method**	**URL**	**Request Body**
Create	POST	/{modul}-service/{submodul}	Create DTO
Read by Id	GET	/{modul}-service/{submodul}/{id}	-
Find All	GET	/{modul}-service/{submodul}s	-
Update	PATCH	/{modul}-service/{submodul}/{id}	Update DTO
Delete	DELETE	/{modul}-service/{submodul}/{id}	-

Dapat dilihat pada Tabel 3.2, untuk melakukan *hit* ke API sebuah *method* diperlukan URL dan juga HTTP *method*. Jika ingin melakukan *create*, HTTP *method* yang digunakan adalah POST. Untuk *read by id* maupun *find all*, HTTP *method* yang digunakan adalah GET. Lalu, untuk melakukan *update* atau mengedit data, HTTP *method* yang digunakan adalah PATCH. Sedangkan untuk *method delete*, HTTP *method* jenis DELETE digunakan. Untuk *method read, update dan*

delete diperlukan id di dalam URL saat melakukan *request*. Sedangkan huruf ”s” ditambahkan setelah nama submodul di dalam URL saat melakukan *find all* untuk membedakan dari method *read by id*. *Request body* sendiri merupakan *input* dari *user*. Saat membuat data atau *create*, *input* dari *user* akan berupa *create* DTO. Sedangkan saat melakukan *method update*, *request body* akan berupa *update* DTO.

3.3.9 Pembuatan Unit Test

Selain pengetesan menggunakan Postman, unit test dari *method* CRUD ini juga dibuat. Unit test di sistem ERP ini dibagi menjadi dua, yaitu service test dan juga contract test.

A. Service Test

Service test digunakan untuk melakukan test terhadap service yang telah dibuat. Service test ini menggunakan H2 database yang disimpan secara sementara di dalam memori. H2 database itu sendiri masih berdasarkan dengan database yang digunakan di *server*, namun data yang di-*insert* tidak masuk ke database yang digunakan di server. Setiap method mulai dari create, read, update, delete dan juga find all diuji coba menggunakan database tersebut.

A.1 Unit Test Create

Unit test create dilakukan dengan mempersiapkan *object* create DTO dengan nilainya yang valid. Lalu, service dipanggil untuk menjalankan method create yang telah dibuat sebelumnya. Service akan mengembalikan data berupa response DTO. Setelah itu, nilai yang ada dalam response DTO tersebut akan dibandingkan dengan setiap nilai yang ada di dalam create DTO. Jika semua nilai sama, maka test dapat dikatakan berhasil. Namun, jika ada satu atau lebih nilai yang tidak sama, maka test akan dinyatakan gagal dan menimbulkan error saat *compile*.

A.2 Unit Test Read

Unit test read dilakukan dengan membaca data dengan id tertentu dalam H2 database. Dalam test ini, dipersiapkan beberapa variabel yang merepresentasikan nilai-nilai yang ada dalam data. Setelah itu, service dipanggil untuk melakukan *method read* dengan id tertentu. Service akan mengembalikan data berupa response

DTO kembali ke unit test ini. Lalu, dilakukan perbandingan antara variabel-variabel yang telah dipersiapkan dengan nilai-nilai yang ada dalam response DTO. Semua perbandingan harus sama dan berhasil untuk unit test dapat dinyatakan berhasil.

Selain skenario tersebut, dilakukan juga test dengan skenario lain, yaitu saat data dengan id tertentu tidak ditemukan dalam database. Dalam skenario ini, dipersiapkan sebuah variabel yang berisi pesan yang diharapkan dari skenario ini. Selain itu, service juga dipanggil untuk melakukan *method read*. Karena diharapkannya sebuah Data Not Found Exception, maka pada saat service dipanggil, terdapat sebuah model Exception yang menangkap exception yang dilemparkan oleh service. Lalu, pesan dalam exception tersebut ditampung dalam sebuah variabel. Setelah itu, variabel yang berisi pesan yang diharapkan akan dibandingkan dengan variabel tersebut.

A.3 Unit Test Update

Dalam unit test ini, dipersiapkan sebuah *object update* DTO dengan nilainya yang valid. Skenario pertama yang diuji coba adalah skenario *update* berhasil dilakukan. Service dipanggil dengan parameter *update* DTO dan id tertentu. Service akan mengembalikan sebuah *response* DTO. Nilai-nilai dari *response* DTO tersebut dibandingkan dengan nilai-nilai yang ada dalam *update* DTO.

Setelah itu dilakukan juga skenario di saat update gagal dilakukan karena data dengan id tertentu tidak ditemukan. Proses ini mirip dengan skenario unit test *read* saat data tidak ditemukan. Yang berarti service yang dipanggil akan mengembalikan sebuah Data Not Found Exception. Lalu, pesan yang ada dalam exception tersebut ditampung di sebuah variabel. Setelah itu variabel tersebut dibandingkan dengan variabel yang berisi pesan yang diharapkan dari skenario ini.

Selain kedua skenario tersebut, dilakukan sebuah test dari satu skenario lain, yaitu saat *field* ”version” di dalam update DTO tidak sama dengan ”version” yang ada dalam database. Di skenario ini, ”version” dalam update DTO sengaja dibuat beda dengan yang berada pada data dalam database. Di skenario ini service akan mengembalikan Data Lock Exception dan pesan dari exception tersebut akan ditampung di sebuah variabel. Lalu, variabel tersebut dibandingkan dengan variabel yang berisi pesan bahwa ”version” dalam database tidak sama dengan ”version” dalam update DTO. Dari ketiga skenario ini, semua perbandingan yang dilakukan harus berhasil untuk membuat unit test yang berhasil.

A.4 Unit Test Find All

Unit test find all ini harus dilakukan pertama kali sebelum unit test yang lain agar data yang berubah tidak berpengaruh terhadap test ini. Dalam test ini, service dipanggil untuk melakukan *method find all* dan mengembalikan Page berisi *response* DTO yang menjadi representasi data-data yang ada di dalam H2 database. Perbandingan yang dilakukan dalam tes ini adalah jumlah elemen yang ada dalam Page tersebut dengan sebuah angka. Perbandingan tersebut juga harus berhasil untuk dapat menyatakan unit tes ini sukses.

B. Contract Test

Contract test dibuat dengan menggunakan metode *mock* atau meniru. Jadi setiap method yang ada di service akan ditiru *request* dan *response*-nya. *Request* juga termasuk URL untuk melakukan *hit* ke API. Karena menggunakan metode *mock*, data tidak harus sesuai dengan database. Terdapat dua *file* dalam tes ini, yaitu base dan juga contractnya itu sendiri. Base berisikan *request* yang telah diisi dan juga *response* DTO yang sudah ditentukan isinya.

B.1 Test Create

Dalam test ini, *request* juga berisikan *create* DTO dan *response* merupakan *response* DTO dari submodul tertentu.

B.2 Test Read

Dalam tes ini, *request* hanya berisi URL yang terdapat id di dalamnya. *Response* pun berisi *response* DTO dari submodul tertentu. *Test read* ini dilakukan untuk dua skenario, yaitu skenario berhasil dan skenario gagal karena data tidak ditemukan.

B.3 Test Update

Di test ini, *request* berisikan URL dengan id di dalamnya serta *request body* berisi *update* DTO. *Response* dari tes ini berupa *response* DTO dari sebuah data yang telah diubah berdasarkan *update* DTO. Dalam tes ini dilakukan dua skenario,

yaitu skenario *update* berhasil dan skenario *update* gagal karena ”version” dari *update* DTO tidak sama dengan yang ada di dalam database.

B.4 Test Delete

Sama seperti *test read*, *test delete* ini hanya berisikan URL dengan id di dalamnya sebagai *request*. Sedangkan response dari test ini berupa *response* DTO yang mana *field* ”deleted” dalam DTO tersebut sudah menjadi true. Test delete ini juga dilakukan dengan dua skenario, yaitu *delete* berhasil dan skenario *delete* gagal karena data tidak ditemukan.

B.5 Test Find All

Dalam test ini, *request* berisikan url untuk *hit* API, lalu *query* parameter yang berisikan *page number*dan *size* dari Page yang diharapkan di *response*. Untuk *response* dari tes ini berupa Page dari *response* DTO yang konfigurasinya sesuai dengan parameter pada *request*.

3.4 Kendala dan Solusi yang Ditemukan

Kendala yang dialami saat pelaksanaan magang adalah sebagai berikut.

- Penggunaan *tech stack* dan juga arsitektur yang belum pernah digunakan sebelumnya.
- Terjadi banyak perubahan saat pembuatan proyek.

Adapun solusi yang ditemukan saat pelaksanaan magang yaitu sebagai berikut.

- Mencari referensi atau pun sumber belajar terkait *tech stack* dan juga arsitektur yang digunakan
- Menyesuaikan diri terhadap perubahan yang terjadi dalam pengerjaan proyek.