BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Selama kegiatan magang berlangsung, pekerjaan yang dilakukan adalah sebagai *full-stack developer*. Terdapat sembilan orang *intern* yang dibagi menjadi empat kelompok dan seorang *supervisor*. Pekerjaan magang dilakukan dalam sebuah tim dengan bimbingan dari Bapak Sugito selaku *supervisor* dan juga *team leader* dalam mengembangkan proyek ERP.

3.2 Tugas yang Dilakukan

Tugas yang dilakukan selama pelaksanaan magang di PT Cranium yaitu merancang dan membangun *backend* sistem ERP. Rancang bangun dimulai dengan pembuatan *entity relationship diagram* atau ERD dan skema *database* atau ER *description* untuk modul ERP. Setelah itu dilanjutkan dengan melakukan *cloning* modul sebagai *template* awal dalam pembangunan *backend*. Setelah melakukan *cloning*, dilanjutkan dengan pembuatan *method create, read, update, delete* atau CRUD dan *unit test* pada setiap fitur yang digunakan dalam sistem ERP.

Berikut merupakan detail dari tugas yang dikerjakan selama pelaksanaan magang berlangsung:

- Membuat ERD dan ER *description* bersama tim untuk modul *purchasing, inventory control*, dan *production planning*.
- Melakukan *cloning* untuk modul *purchasing, inventory control*, dan *production planning*. Proses *cloning* dilakukan dengan membuat modul Java baru pada proyek untuk setiap modul ERP. *Cloning* dilakukan karena proyek ERP dibangun menggunakan arsitektur modular monolitik dimana satu proyek memiliki banyak modul yang dapat berdiri secara independen.
- Melakukan pembuatan CRUD untuk fitur *purchasing order, purchasing bill, inventory item withdraw reservation, inventory item transfer, production order, master plant, master area, master driver, master unit of measurement*, dan *master activity standard*.

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 -	Pekerjaan yang dilakukan
1	Mempelajari konsep dasar Springboot dan ERP
2	Melanjutkan pembelajaran Springboot dengan melakukan CRUD sederhana dan mempelajari materi keamanan menggunakan JWT
3	Melanjutkan pembelajaran materi keamanan, *exception handling*, dan validasi
4	Melakukan instalasi proyek ERP yang diberikan *supervisor*, *cloning* modul *purchasing*, membuat *method* CRUD sederhana untuk fitur *purchasing request*
5	Menambahkan modul *sales* dari kelompok lain dan melakukan *cloning template* terbaru
6	Melanjutkan *cloning template* proyek terbaru, menambahkan *custom validation* untuk fitur *purchasing request*, membuat ERD dan ER *description* untuk modul *purchasing* bersama tim
7	Melanjutkan pembuatan ERD dan ER *description* untuk modul *purchasing* dan *inventory control*, melakukan *cloning* modul *inventory control*
8	Membandingkan *table* ERP lama dengan ER *description*, menambahkan *field* yang belum ada ke ER *description*, dan membuat *database migration* untuk modul *purchasing*
9	Melanjutkan pembuatan *database migration* dan membuat *method* CRUD untuk fitur *purchasing order*
10	Melakukan *cloning* modul *production planning* dan membuat *unit test* untuk fitur *purchasing order*
11	Membuat *method* CRUD untuk fitur *purchasing bill*
12	Melanjutkan pembuatan *method* CRUD untuk fitur *purchasing bill* dan membuat *unit test* untuk fitur *purchasing bill*
13	Membuat *method* CRUD untuk fitur *inventory item withdraw reservation*
Lanjut pada halaman berikutnya	

Minggu Ke -	Pekerjaan yang dilakukan
14	Membuat *unit test* untuk fitur *inventory item withdraw reservation*
15	Membuat *method* CRUD untuk fitur *inventory item transfer* dan *unit test* untuk fitur *inventory item transfer*
16	Melanjutkan pembuatan *unit test* untuk fitur *inventory item transfer*, membuat *method* CRUD untuk fitur *production order*
17	Membuat *unit test* untuk fitur *production order* dan melakukan revisi nama *field* yang panjangnya lebih dari 25 karakter
18	Membuat *method* CRUD untuk fitur *master plant, master area* dan *unit test* untuk fitur *master plant*
19	Membuat *method* CRUD untuk fitur *master unit of measurement* dan *unit test* untuk fitur *master area, master unit of measurement*
20	Membuat *method* CRUD untuk fitur *master activity standard*
21	Membuat *unit test* untuk fitur *master activity standard*

Berdasarkan tabel 3.1, pekerjaan yang dilakukan selama melakukan kegiatan magang dapat dikelompokkan sebagai berikut:

3.3.1 Pembelajaran

A. Pengenalan ERP

Pengenalan sistem ERP dilakukan pada 2 minggu pertama kegiatan magang berlangsung pada jadwal WFO (kamis). Sistem ERP milik Cranium terdiri dari enam modul utama. Keenam modul tersebut meliputi:

- *Sales & distribution*: modul yang digunakan untuk menangani penjualan.
- *Purchasing*: modul yang digunakan untuk menangani pembelian.
- *Finance*: modul yang menangani keuangan.
- *Production planning*: modul yang digunakan dalam menangani kegiatan produksi suatu perusahaan.
- *Inventory control*: modul yang digunakan untuk menangani ketersediaan stok barang.
- *Accounting*: modul yang digunakan untuk mengelola pembukuan dan catatan keuangan perusahaan.

B. Pengenalan Tech Stack dan Arsitektur

Pengenalan *tech stack* dan arsitektur yang digunakan dalam perancangan dan pembangunan sistem ERP dilakukan pada pertemuan pertama kegiatan magang. Berikut *tech stack* dan arsitektur yang digunakan dalam pengembangan sistem ERP meliputi:

- *Framework* Springboot: *framework* Springboot digunakan dalam mengembangkan sistem *backend* ERP. Alasan digunakan *framework* ini yaitu Springboot merupakan *framework* berbasis Java sehingga bersifat fleksibel dan dapat dijalankan di semua sistem operasi. Selain itu, Springboot juga merupakan *framework* yang sudah cukup terkenal dan memiliki dokumentasi yang lengkap.
- *Database* PostgreSQL: PostgreSQL merupakan *database* relasional yang bersifat *open-source*. *Database* relasional digunakan karena terdapat banyak relasi antar *table* pada sistem ERP.
- Arsitektur modular monolitik: modular monolitik merupakan arsitektur monolitik yang bersifat modular. Arsitektur ini membagi *codebase* menjadi beberapa modul dimana setiap modul bersifat *loosely-coupled* atau dapat berdiri secara independen dan perubahan di satu modul tidak mempengaruhi modul lain [8]. Arsitektur modular monolitik dipilih karena memiliki *scalability* dan *maintanability* yang lebih baik dibandingkan arsitektur monolitik biasa. Selain itu, jika di masa depan arsitektur *microservices* ingin diterapkan pada sistem ERP, proses pergantian arsitektur akan lebih mudah dilakukan karena *codebase* yang telah dibagi menjadi beberapa modul yang sifatnya independen.

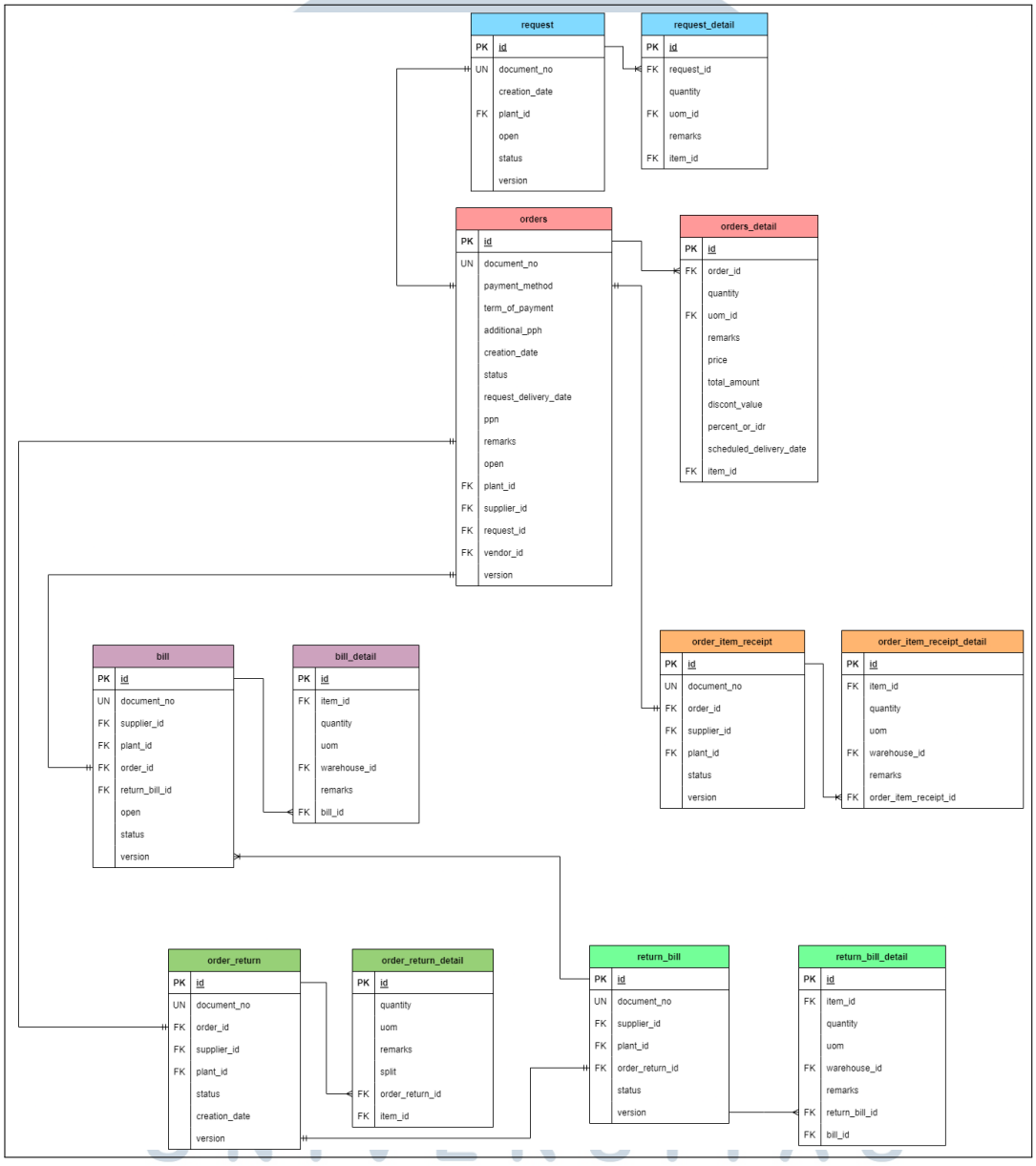
3.3.2 Perancangan

A. ERD

Dalam melaksanakan kegiatan magang, terdapat 3 modul yang dikerjakan yaitu modul *purchasing, inventory control*, dan *production planning*. Berikut merupakan ERD dari ketiga modul tersebut:

A.1 Purchasing

Berikut merupakan ERD dari modul *purchasing*



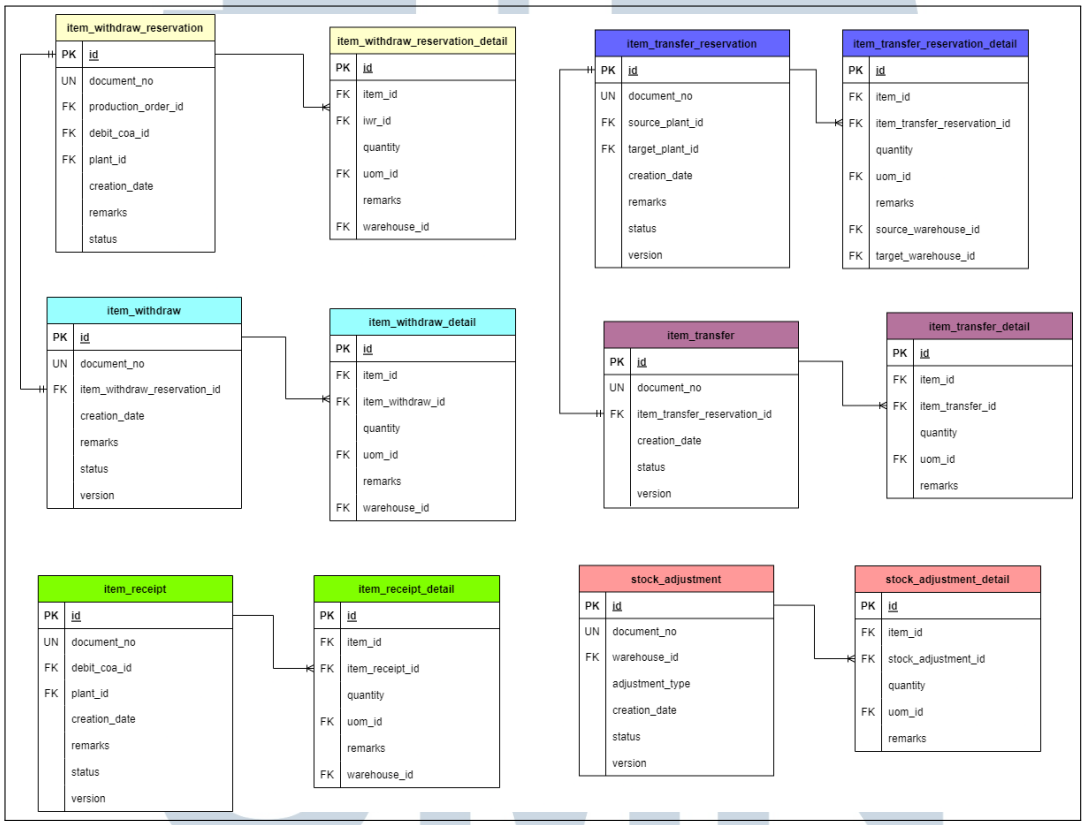
Gambar 3.1. ERD Modul Purchasing

Gambar 3.1 menunjukkan ERD dari modul *purchasing*. Terdapat total 12 tabel yang terdiri atas 6 tabel utama/*header* dan 6 tabel *detail*. 6 tabel utama tersebut meliputi *request, order, order item receipt, bill, order return*, dan *return bill*. Keenam tabel tersebut saling berelasi dimana tabel *request* berelasi *one to one* dengan tabel *order*, tabel *order* berelasi *one to one* dengan tabel *order item*

receipt, tabel *order item receipt* berelasi *one to one* dengan tabel *bill*, tabel *oder return* berelasi *one to one* dengan tabel *order*, dan tabel *return bill* berelasi *one to one* dengan tabel *order return* . Setiap tabel utama memiliki 1 tabel detail dengan relasi *one to many*.

A.2 Inventory Control

Berikut merupakan ERD dari modul *inventory control*

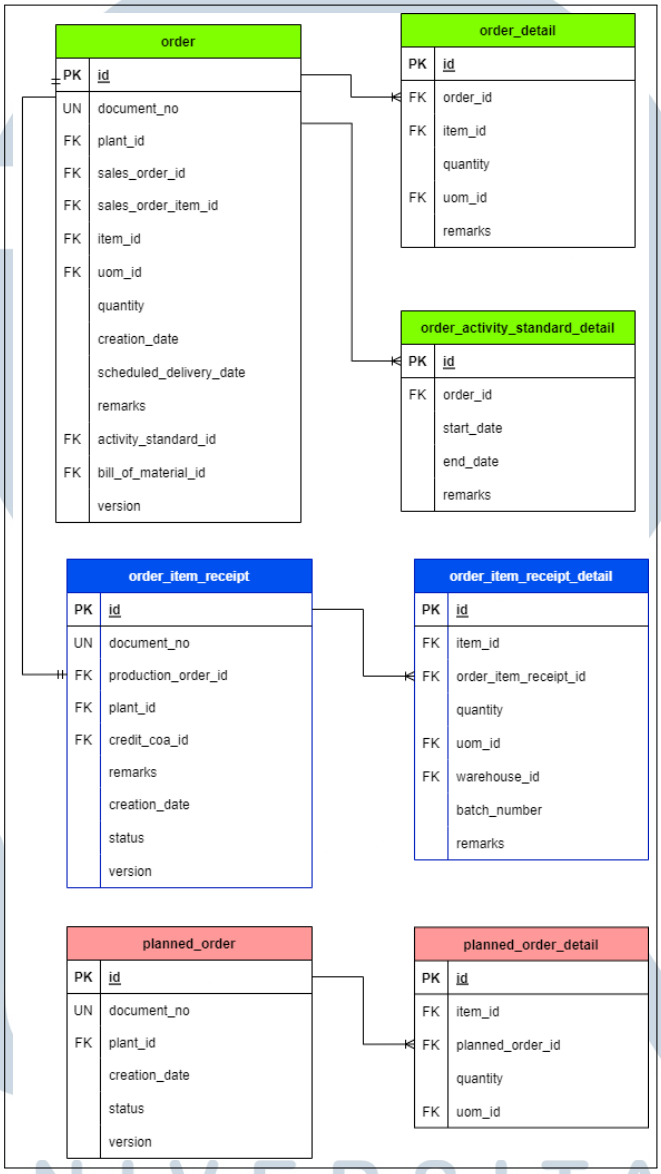


Gambar 3.2. ERD Modul Inventory Control

Gambar 3.2 menunjukkan ERD dari modul *inventory control*. Terdapat total 12 tabel yang terdiri atas 6 tabel utama/*header* dan 6 tabel *detail*. 6 tabel utama tersebut meliputi *item withdraw reservation, item withdraw, item transfer reservation, item transfer, item receipt*, dan *stock adjustment*. Dari keenam tabel tersebut, terdapat 4 tabel utama yang berelasi dengan tabel utama lainnya yaitu tabel *item withdraw reservation* berelasi *one to one* dengan tabel *item withdraw* dan tabel *item transfer reservation* berelasi *one to one* dengan tabel *item transfer*. Setiap tabel utama memiliki 1 tabel detail dengan relasi *one to many*.

A.3 Production Planning

Berikut merupakan ERD dari modul *production planning*



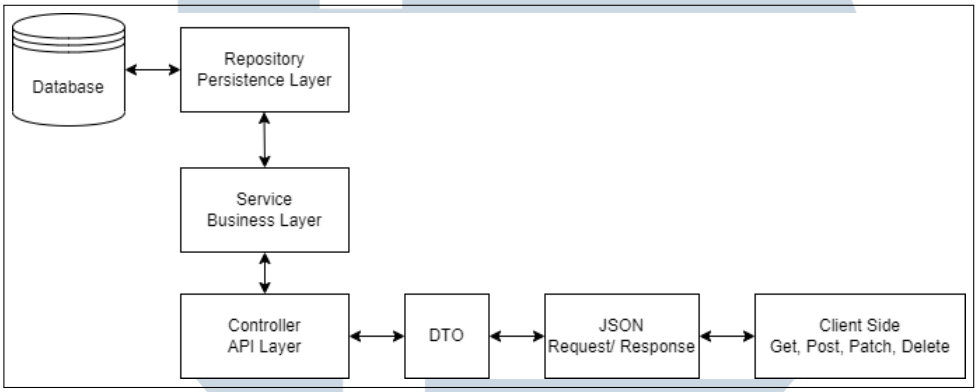
Gambar 3.3. ERD Modul Production Planning

Gambar 3.3 menunjukkan ERD dari modul *production planning*. Terdapat total 7 tabel yang terdiri atas 3 tabel utama/*header* dan 4 tabel *detail*. 3 tabel utama tersebut meliputi *order, order item receipt*, dan *planned order* dimana tabel *order* berelasi *one to one* dengan tabel *order item receipt*. Setiap tabel utama memiliki 1 tabel detail kecuali tabel *order* dimana tabel tersebut memiliki 2 *detail*. Untuk setiap tabel *header* memiliki relasi *one to many* dengan tabel *detail*.

3.3.3 Pembangunan

A. Arsitektur Springboot

Arsitektur Springboot terdiri dari 4 lapisan berbentuk hierarki yang saling berkomunikasi satu sama lain [9]. Keempat lapisan tersebut meliputi *database layer, persistence layer, business layer,* dan API *layer*.



Gambar 3.4. Spingboot Application Architecture

Sumber: [9]

Gambar 3.4 menunjukkan skema arsitektur yang digunakan oleh Springboot. Berikut merupakan fungsi dari keempat lapisan:

- *Database layer*: lapisan yang melakukan operasi CRUD.
- *Persistence layer*: lapisan yang berisi *storage logic* dan mengubah objek menjadi *row database* atau sebaliknya.
- *Business layer*: lapisan yang menangani *business logic*, validasi, dan otorisasi.
- API *layer*: lapisan yang menangani HTTP *request* dan mengubah JSON menjadi objek atau sebaliknya.

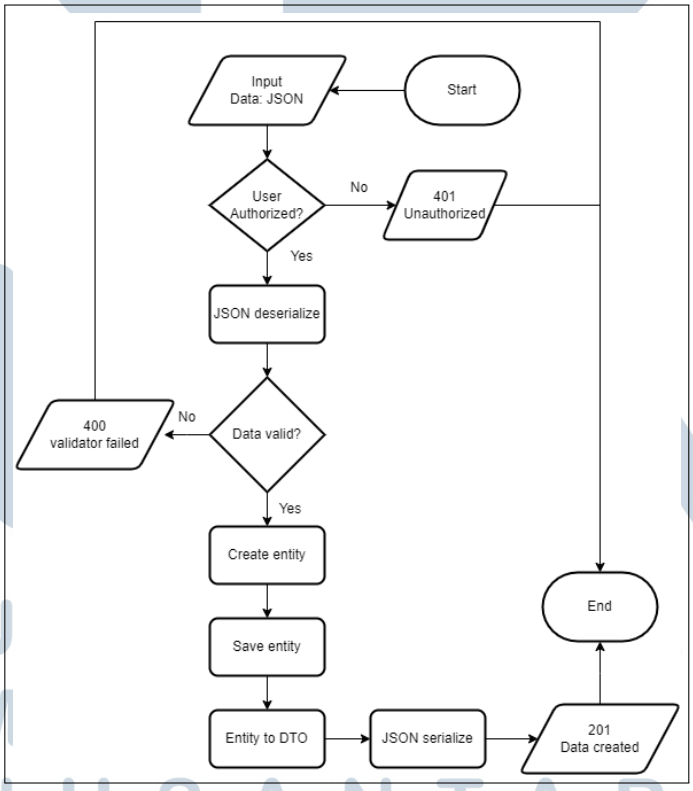
Selain menunjukkan skema arsitektur pada Springboot, gambar 3.4 juga menjelaskan *flow* Springboot saat menangani *request* dan *response*. *Client* akan melakukan *request* menggunakan REST API *methods* (*get, post, patch, delete*). Jika melakukan *request method post* atau *patch*, JSON akan di-*map* atau diubah menjadi *data transfer object* atau DTO. DTO merupakan objek yang membungkus data *request* maupun *response* [10]. Setelah itu, API *layer* atau *controller* akan

menerima objek tersebut dan diteruskan ke *business layer* atau *service*. *Service* kemudian memanggil *persistance layer* atau *repository*. *Repository* memiliki hubungan langsung terhadap *database* dalam melakukan operasi CRUD. Setiap operasi CRUD dilakukan, *repository* akan mengembalikan sebuah *entity* atau objek yang merepresentasikan satu *record database* ke *service*. *Service* akan mengubah *entity* menjadi DTO dan juga akan menangani *business logic* sebelum mengembalikan DTO tersebut ke *controller*. *Controller* akan mengembalikan DTO tersebut dan merubahnya menjadi bentuk JSON sebagai *response* yang akan diterima *client*.

B. Flowchart CRUD

B.1 Create

Gambar dibawah ini merupakan alur dari *method create*



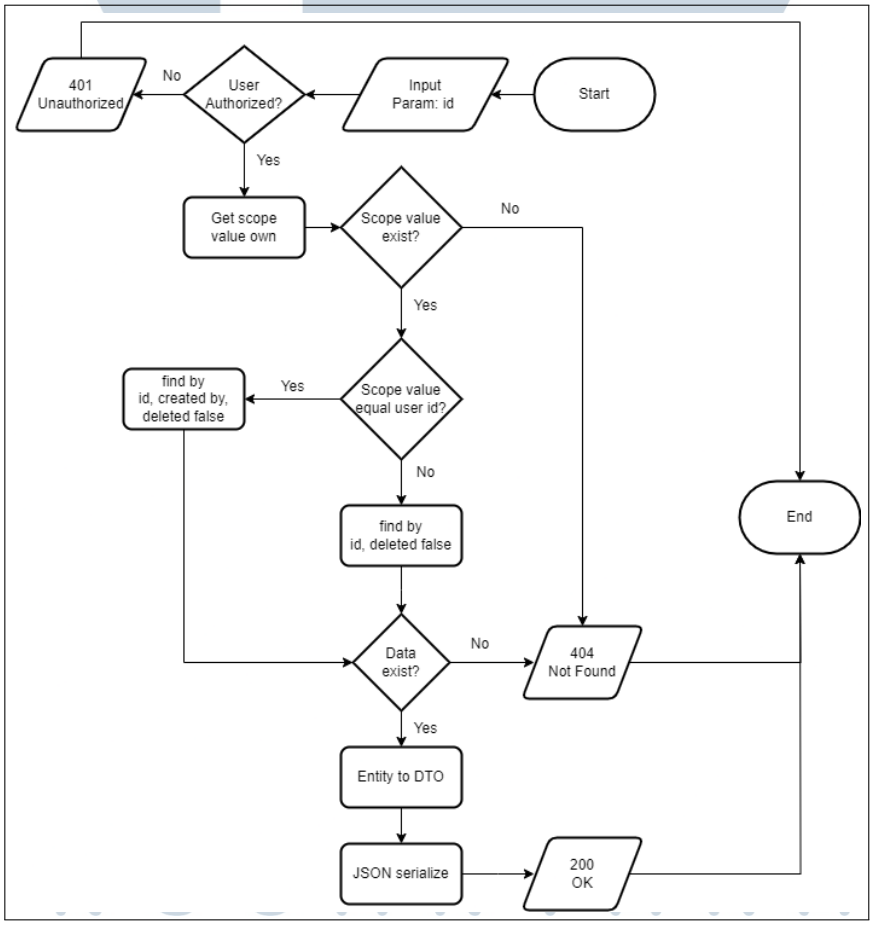
Gambar 3.5. Flowchart Create

Gambar 3.5 menunjukkan *flowchart method create* pada sistem ERP. *Flowchart* diawali dengan menginput data berformat JSON. Kemudian akan dicek

apakah *user* terotorisasi. Jika *user* tidak terotorisasi, akan memunculkan status eror 401 *unauthorized*. Jika *user* terotorisasi, dilanjutkan dengan Springboot melakukan proses *deserialize* atau mengubah JSON menjadi DTO. Kemudian dilakukan validasi terhadap DTO dan jika gagal akan memunculkan status eror 400 *validator failed*. Jika data valid, maka akan dibentuk sebuah *entity* dan *entity* tersebut akan di simpan dalam *database*. Setelah itu, *entity* akan diubah menjadi DTO, dilanjutkan dengan proses *serialize* atau mengubah DTO menjadi JSON. Data kemudian dikirim ke *client* dengan kode *response* 201 *created* atau data berhasil dibuat.

B.2 Read Satu Data

Gambar dibawah ini merupakan alur dari *method read* 1 data



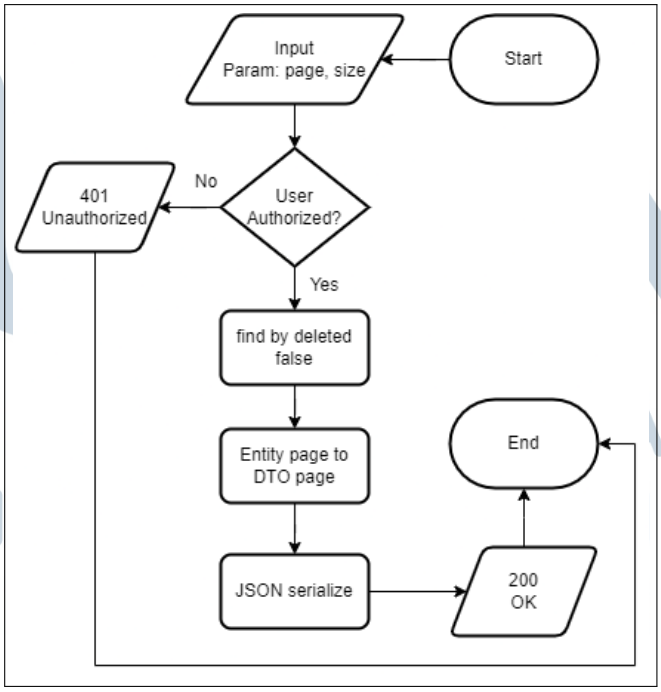
Gambar 3.6. Flowchart Read Satu Data

Gambar 3.6 menunjukkan *flowchart method read* satu data pada sistem ERP.

Flowchart diawali dengan menginput parameter id. Kemudian akan dicek apakah *user* terotorisasi. Jika *user* tidak terotorisasi, akan memunculkan status eror 401 *unauthorized*. Jika *user* terotorisasi, dilanjutkan dengan mencari *scope value*. *Scope* merupakan lingkup *user* dalam mengakses suatu data. *Scope* bisa memiliki *value* yang bermacam-macam, namun *value* dari *scope* yang digunakan saat laporan ini dibuat yaitu *user* id. Jika *scope value* tidak ditemukan, maka akan memunculkan status eror 404 *not found*. Jika ditemukan, maka akan dicek apakah *user* id sama dengan *scope value*. Jika sama, maka mencari data berdasarkan id, *created by*, dan *deleted false*. Jika tidak, maka mencari data hanya berdasarkan id dan *deleted false*. Setelah itu, akan dicek apakah data ada atau tidak. Jika tidak ada, maka akan memunculkan status 404 *not found*. Jika ada, maka akan dilanjutkan dengan mengubah data tersebut yang berupa *entity* menjadi DTO. Setelah itu, Springboot akan melakukan proses *serialize* dan data dikirim ke *client* dengan kode *response* 200 OK atau *request* sukses.

B.3 Read Paging

Gambar dibawah ini merupakan alur dari *method read paging* atau banyak data

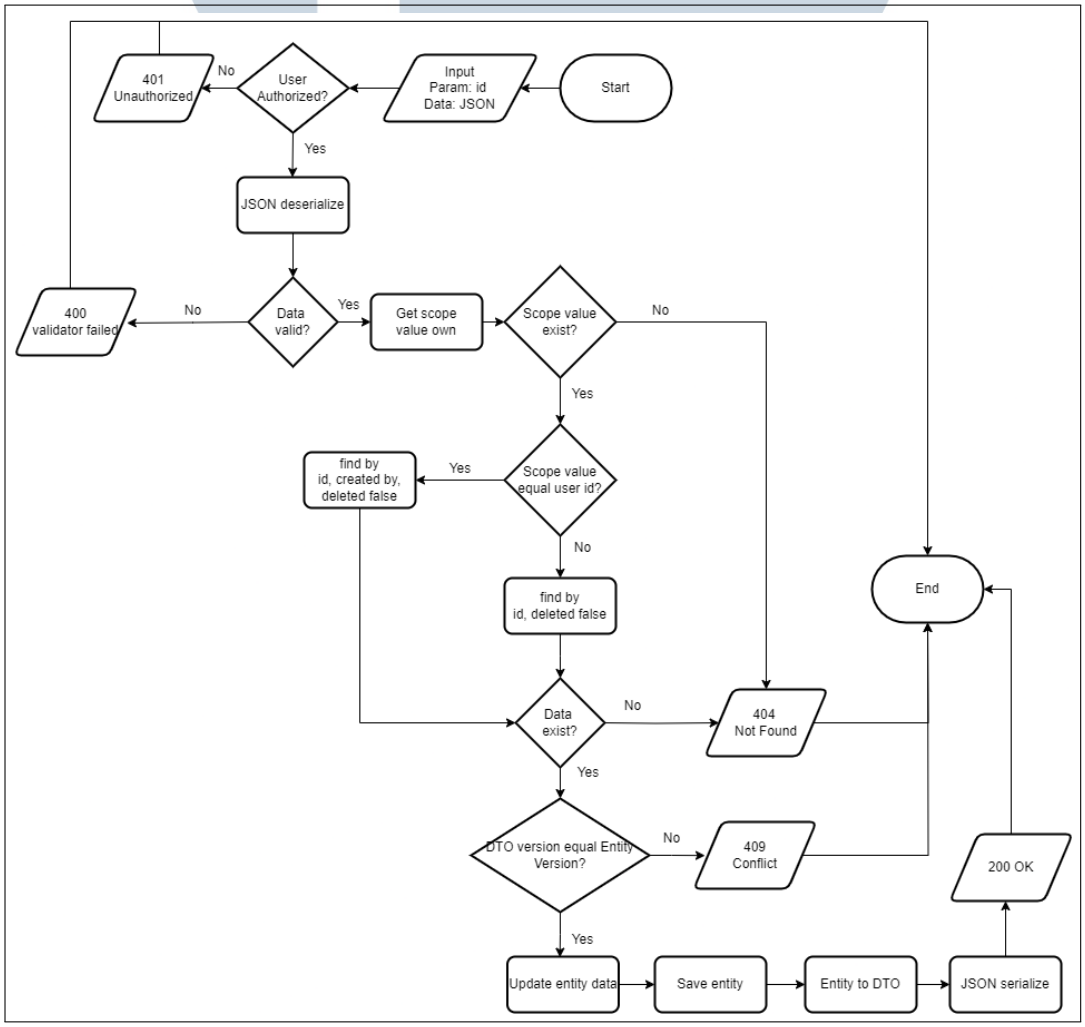


Gambar 3.7. Flowchart Read Paging

Gambar 3.7 menunjukkan *flowchart method read paging* pada sistem ERP. *Flowchart* dimulai dengan menginput parameter *page* atau halaman ke berapa dan *size* atau banyaknya data dalam satu halaman. Kemudian akan dicek apakah *user* terotorisasi. Jika *user* tidak terotorisasi, akan memunculkan status eror 401 *unauthorized*. Jika *user* terotorisasi, dilanjutkan dengan mencari data berdasarkan *deleted false* dan parameter yang telah diinput. Setelah itu, *page* yang masih berisi *entity* akan diubah menjadi *page* yang berisi DTO. Setelah itu, Springboot akan melakukan proses *serialize* dan data dikirim ke *client* dengan kode *response* 200 OK.

B.4 Update

Gambar dibawah ini merupakan alur dari *method update*

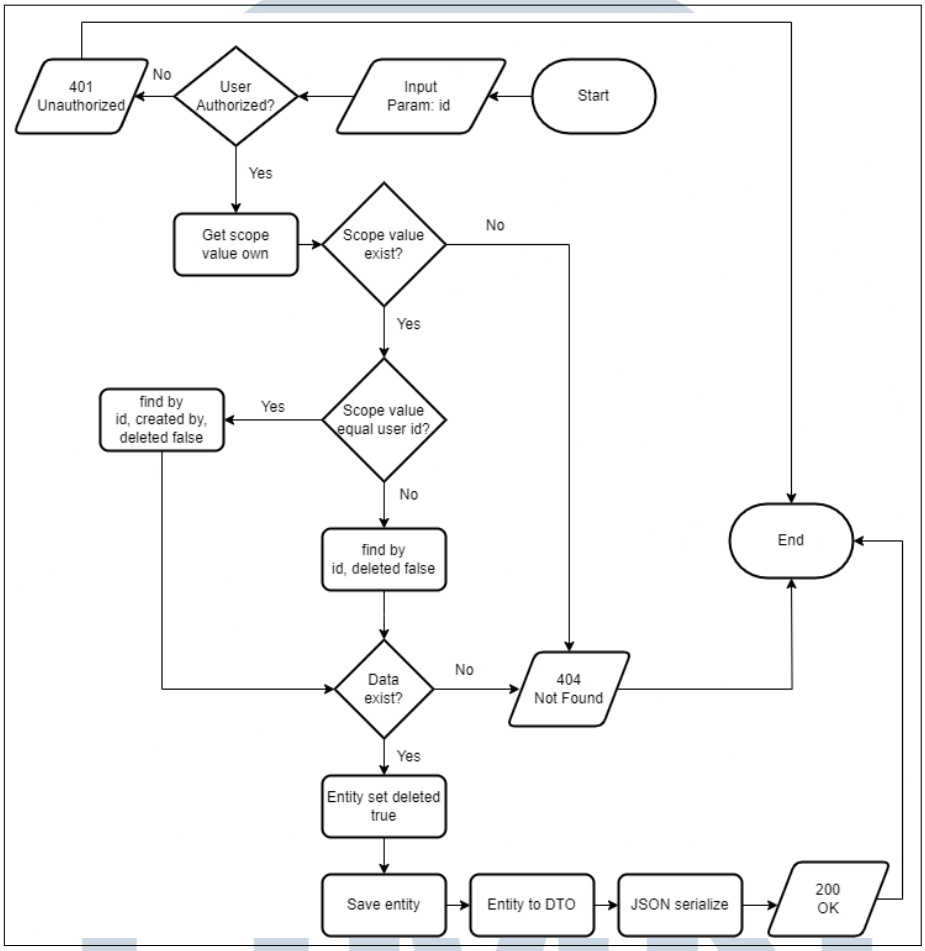


Gambar 3.8. Flowchart Update

Gambar 3.8 menunjukkan *flowchart method update* pada sistem ERP. *Flowchart* diawali dengan menginput parameter berupa id dan data berformat JSON. Kemudian akan dicek apakah *user* terotorisasi. Jika *user* tidak terotorisasi, akan memunculkan status eror 401 *unauthorized*. Jika *user* terotorisasi, dilanjutkan dengan Springboot melakukan proses *deserialize*. Kemudian dilakukan validasi terhadap DTO dan jika gagal memunculkan status eror 400 *validator failed*. Jika validasi berhasil, dilanjutkan dengan mencari *scope value*. Jika *scope value* tidak ditemukan, maka akan memunculkan status eror 404 *not found*. Jika ditemukan, maka akan dicek apakah *user* id sama dengan *scope value*. Jika sama, maka mencari data berdasarkan id, *created by*, dan *deleted false*. Jika tidak, maka mencari data hanya berdasarkan id dan *deleted false*. Setelah itu, akan dicek apakah data ada atau tidak. Jika tidak ada, maka akan memunculkan status 404 *not found*. Jika ada, akan dicek apakah *version* pada DTO bernilai sama dengan *version* pada *entity*. Jika berbeda, maka akan memunculkan status error 409 *conflict*. Jika sama, maka data yang berupa *entity* tersebut di-*update* nilainya dan disimpan ke dalam *database*. Setelah itu, *entity* akan diubah menjadi DTO. Springboot kemudian melakukan proses *serialize* dan data dikirim ke *client* dengan kode *response* 200 OK.

B.5 Delete

Gambar dibawah ini merupakan alur dari *method delete*



Gambar 3.9. Flowchart Delete

Gambar 3.9 menunjukkan *flowchart method delete* pada sistem ERP. *Method delete* dilakukan dengan mengubah *field deleted* pada data menjadi *true* atau *soft delete*. *Flowchart* dimulai dengan menginput parameter id. Kemudian akan dicek apakah *user* terotorisasi. Jika *user* tidak terotorisasi, akan memunculkan status eror 401 *unauthorized*. Jika *user* terotorisasi, dilanjutkan dengan mencari *scope value*. Jika *scope value* tidak ditemukan, maka akan memunculkan status eror 404 *not found*. Jika ditemukan, maka akan dicek apakah *user* id sama dengan *scope value*. Jika sama, maka mencari data berdasarkan id, *created by*, dan *deleted false*. Jika tidak, maka mencari data hanya berdasarkan id dan *deleted false*. Setelah itu, akan dicek apakah data ada atau tidak. Jika tidak ada, maka akan memunculkan status 404 *not found*. Jika ada, maka data berupa *entity* akan di-*update field deleted*-nya

menjadi *true* dan disimpan ke *database*. Setelah itu, *entity* akan diubah menjadi DTO. Springboot kemudian melakukan proses *serialize* dan data dikirim ke *client* dengan kode *response* 200 OK.

C. REST API Endpoint

Pembangunan REST API *endopoint* dilakukan pada 5 fitur yang dikerjakan yaitu *purchasing order, purchasing bill, inventory item withdraw reservation, inventory item transfer,* dan *production order*.

C.1 Purchasing Order

Berikut merupakan REST API *endpoint* pada fitur *purchasing order*

Tabel 3.2. Tabel Endpoint Purchasing Order

Method	Endpoint	Action
POST	/purchasing-service/order	Save data
GET	/purchasing-service/order/{id}	Find 1 data
GET	/purchasing-service/order	Find paging data
PATCH	/purchasing-service/order/{id}	Update data
DELETE	/purchasing-service/order/{id}	Delete data

Tabel 3.2 menunjukkan REST API *endpoint* pada fitur *purchasing order*. Terdapat lima *endpoint* yang memanggil lima *method* CRUD.

C.2 Purchasing Bill

Berikut merupakan REST API *endpoint* pada fitur *purchasing bill*

Tabel 3.3. Tabel Endpoint Purchasing Bill

Method	Endpoint	Action
POST	/purchasing-service/bill	Save data
GET	/purchasing-service/bill/{id}	Find 1 data
GET	/purchasing-service/bill	Find paging data
PATCH	/purchasing-service/bill/{id}	Update data
DELETE	/purchasing-service/bill/{id}	Delete data

Tabel 3.3 menunjukkan REST API *endpoint* pada fitur *purchasing bill*. Terdapat lima *endpoint* yang memanggil lima *method* CRUD.

C.3 Inventory Item Withdraw Reservation

Berikut merupakan REST API *endpoint* pada fitur *inventory item withdraw reservation*

Tabel 3.4. Tabel Endpoint Inventory Item Withdraw Reservation

Method	Endpoint	Action
POST	/inventory-service/item-withdraw-reservation	Save data
GET	/inventory-service/item-withdraw-reservation/{id}	Find 1 data
GET	/inventory-service/item-withdraw-reservation	Find paging data
PATCH	/inventory-service/item-withdraw-reservation/{id}	Update data
DELETE	/inventory-service/item-withdraw-reservation/{id}	Delete data

Tabel 3.4 menunjukkan REST API *endpoint* pada fitur *inventory item withdraw reservation*. Terdapat lima *endpoint* yang memanggil lima *method* CRUD.

C.4 Inventory Item Transfer

Berikut merupakan REST API *endpoint* pada fitur *inventory item transfer*

Tabel 3.5. Tabel Endpoint Inventory Item Transfer

Method	Endpoint	Action
POST	/inventory-service/item-transfer	Save data
GET	/inventory-service/item-transfer/{id}	Find 1 data
GET	/inventory-service/item-transfer	Find paging data
PATCH	/inventory-service/item-transfer/{id}	Update data
DELETE	/inventory-service/item-transfer/{id}	Delete data

Tabel 3.5 menunjukkan REST API *endpoint* pada fitur *inventory item transfer*. Terdapat lima *endpoint* yang memanggil lima *method* CRUD.

C.5 Production Order

Berikut merupakan REST API *endpoint* pada fitur *production order*

Tabel 3.6. Tabel Endpoint Production Order

Method	Endpoint	Action
POST	/production-service/order	Save data
GET	/production-service/order/{id}	Find 1 data
GET	/production-service/order	Find paging data
PATCH	/production-service/order/{id}	Update data
DELETE	/production-service/order/{id}	Delete data

Tabel 3.6 menunjukkan REST API *endpoint* pada fitur *production order*. Terdapat lima *endpoint* yang memanggil lima *method* CRUD.

3.3.4 Testing

A. Unit Testing

Unit testing merupakan salah satu jenis *software testing* yang mengetes komponen individual dari suatu sistem [11]. *Unit testing* dilakukan dengan mengetes setiap *method* pada *service layer* secara independen. *Unit testing* dilakukan pada 2 kondisi, yaitu kondisi sukses dan gagal pada *method* yang di tes. Pada Springboot, *unit testing* dilakukan dengan *class assertion* atau *class* yang digunakan untuk menguji program berdasarkan asumsi [12]. Jika *assertion* berhasil, instalasi projek dapat berjalan dengan aman dan jika tidak, instalasi projek akan gagal.

```
@Test
void testFindOrderById() throws DataNotFoundException {
    Long orderId = 1L;
    String documentNoExpected = "POD0000001";

    PurchasingOrderDto purchasingOrderDto = purchasingOrderService
    .findOrderById(orderId);

    assertEquals(purchasingOrderDto.getDocumentNo(),
    documentNoExpected);
}
```

Kode 3.1: Contoh Unit Test Sukses

Potongan kode 3.1 menunjukkan contoh *unit testing* pada kondisi sukses untuk *method read* satu data. *Unit testing* tersebut melakukan *assertEquals* dengan membandingkan objek documentNo (*actual value*) dengan objek documentNoExpected (*expected value*). Selain pada kondisi berhasil, *unit testing* juga dilakukan pada kondisi gagal.

```
1  @Test
2  void testFindOrderByIdThrowException() throws
       DataNotFoundException {
3      Long orderId = 10L;
4
5      Exception exception = assertThrows(DataNotFoundException.class
       , () -> {
6          PurchasingOrderDto purchasingOrderDto =
       purchasingOrderService.findOrderById(orderId);
7      });
8
9      String expectedMessage = "Order dengan id 10 tidak ada";
10     String actualMessage = exception.getMessage();
11
12     assertTrue(actualMessage.contains(expectedMessage));
13 }
```

Kode 3.2: Contoh Unit Test Gagal

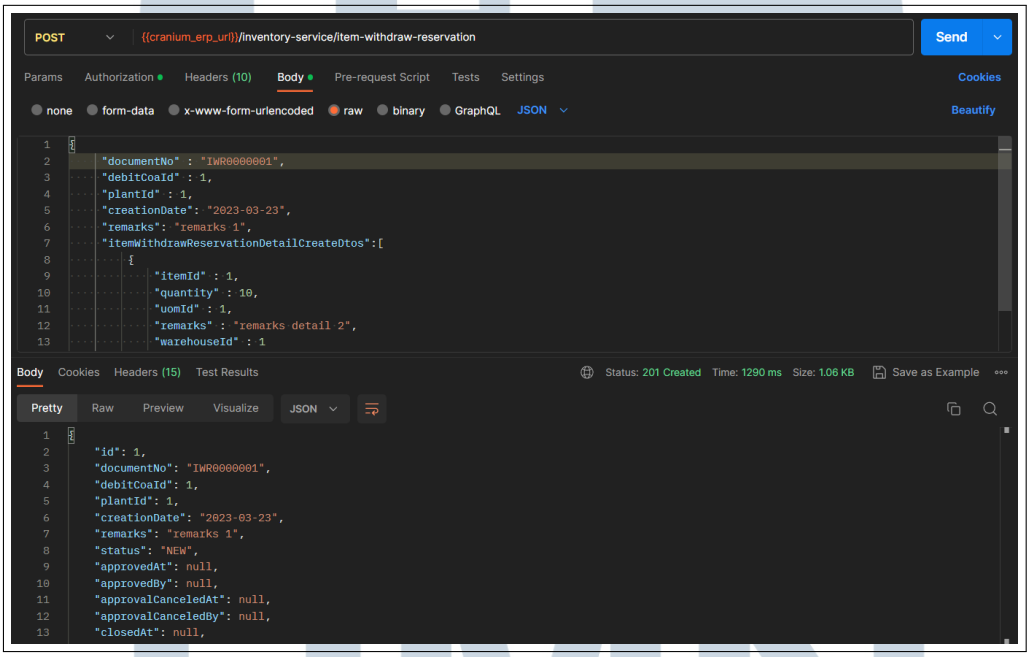
Potongan kode 3.2 menunjukkan contoh *unit testing* pada kondisi gagal untuk *method read* satu data. *Unit testing* tersebut melakukan 2 *assertion* yaitu *assertThrows* dan *assertTrue*. Pada *assertThrows*, dilakukan pemanggilan *method* dengan kondisi yang dapat menghasilkan *exception data not found*. Pada *assertTrue*, diambil pesan *exception* yang sudah didapatkan (*actualMessage*) dan dites apakah pesan yang sudah didapatkan tersebut mengandung pesan ekspetasi (*expectedMessage*).

B. REST API Testing

REST API *testing* dilakukan untuk mengetes 5 REST API *endpoint* yang telah dibangun.

B.1 Create

Berikut merupakan contoh pengetesan REST API *method create* pada fitur *inventory item withdraw reservation*

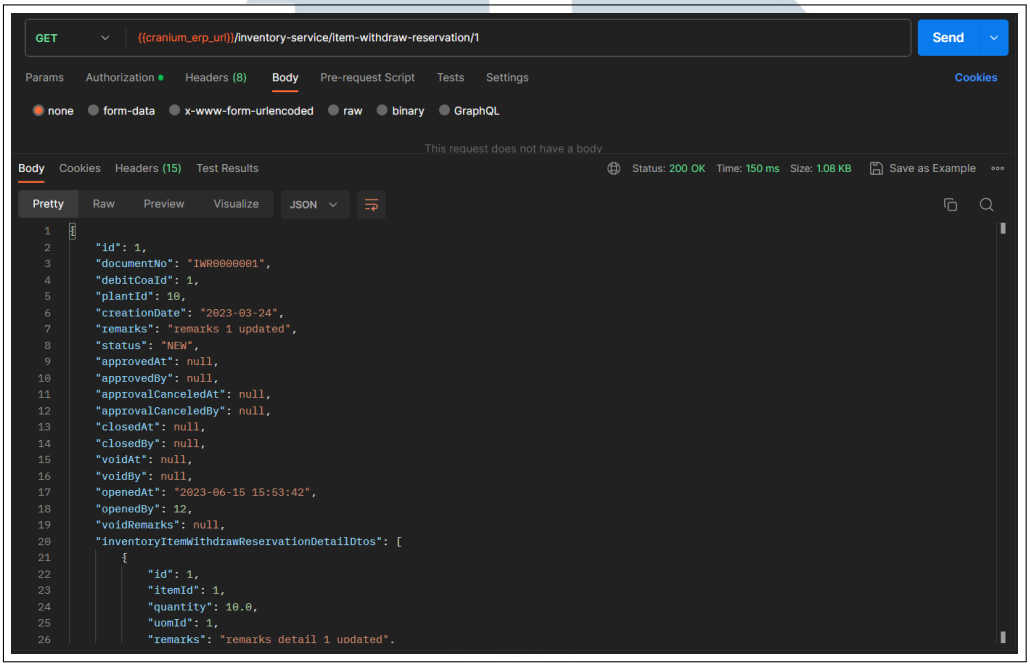


Gambar 3.10. Hasil Tes Endpoint REST API Method Create

Gambar 3.10 merupakan hasil dari pengetesan REST API *method create*. Pada *method create*, dikirimkan sebuah data berbentuk JSON sebagai *request body* yang terdiri atas *header* dan *detail*. *Detail* dapat dikirim lebih dari satu dalam *array*. Jika *request* sukses, *response* status akan menunjukkan 201 *created* dan mengembalikan sebuah data JSON sebagai *response body*.

B.2 Read Satu Data

Berikut merupakan contoh pengetesan REST API *method read* satu data pada fitur *inventory item withdraw reservation*

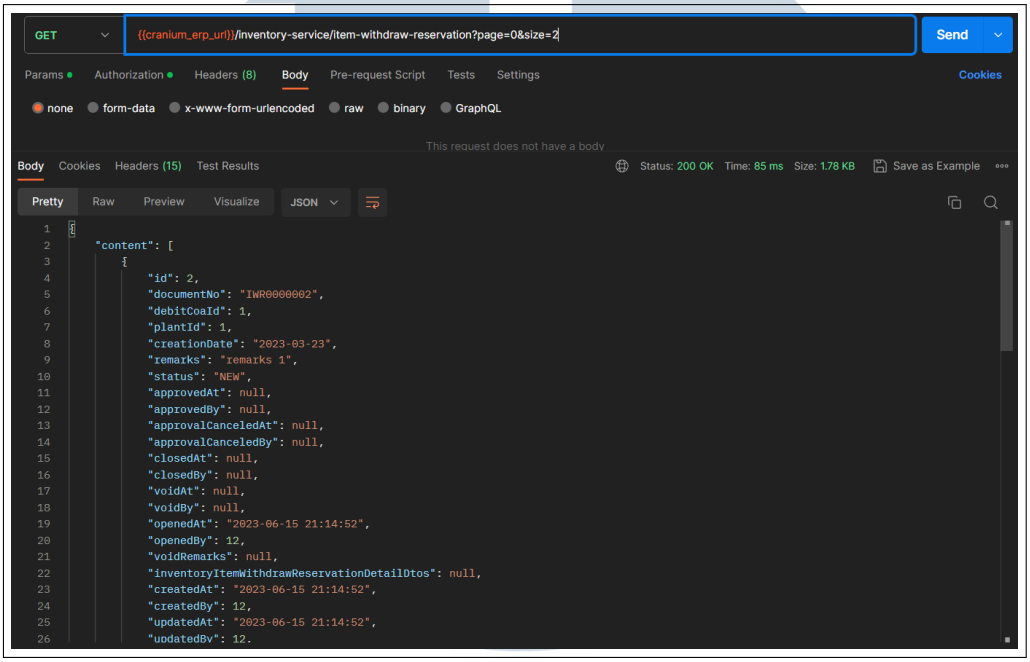


Gambar 3.11. Hasil Tes Endpoint REST API Method Read Satu Data

Gambar 3.11 merupakan hasil dari pengetesan REST API *method read* satu data. Pada *method read* satu data, dikirimkan sebuah sebuah parameter id untuk data yang ingin didapatkan. Jika *request* sukses, *response* status akan menunjukkan 200 OK dan mengembalikan sebuah data JSON sebagai *response body*.

B.3 Read Paging

Berikut merupakan contoh pengetesan REST API *method read paging* pada fitur *inventory item withdraw reservation*

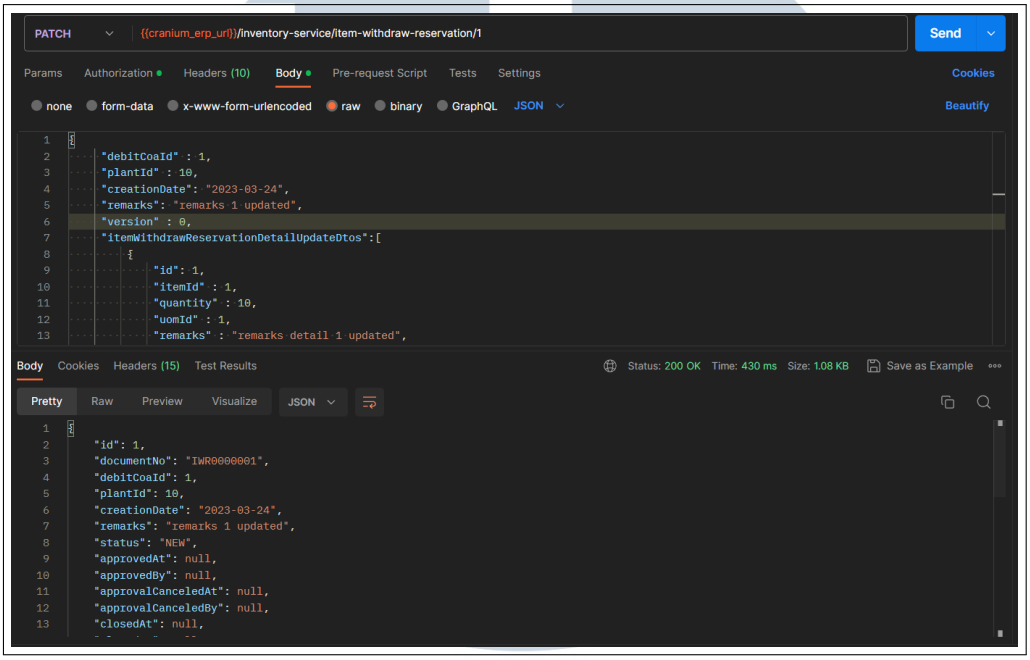


Gambar 3.12. Hasil Tes Endpoint REST API Method Read Paging

Gambar 3.12 merupakan hasil dari pengetesan REST API *method read paging*. Pada *method read paging*, dikirimkan dua parameter yaitu *page* dan *size*. Jika *request* sukses, *response* status akan menunjukkan 200 OK dan mengembalikan sebuah data JSON sebagai *response body*.

B.4 Update

Berikut merupakan contoh pengetesan REST API *method update* pada fitur *inventory item withdraw reservation*

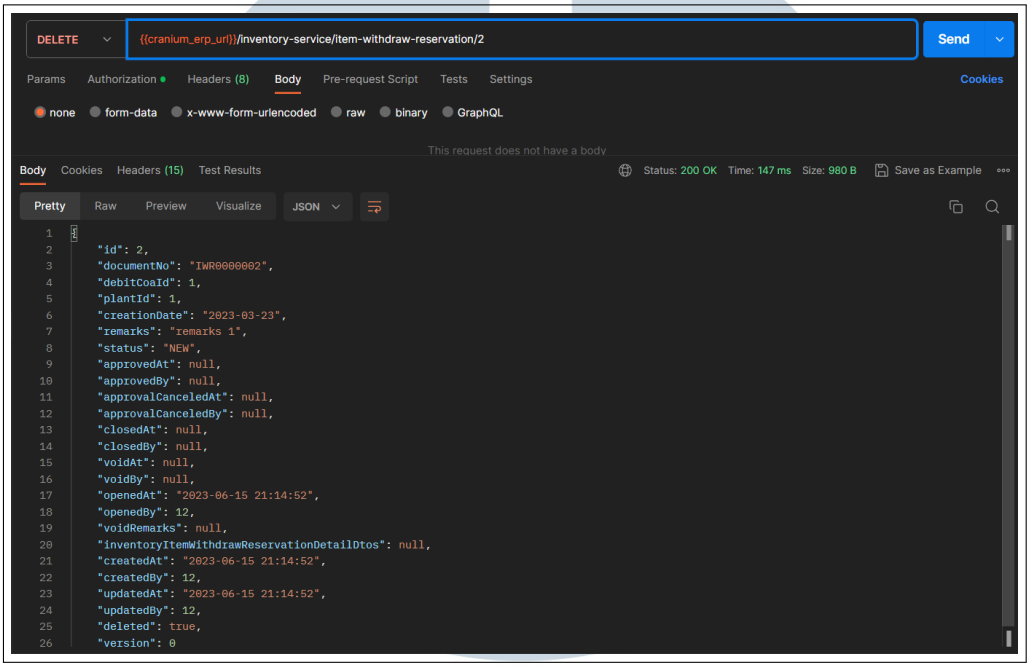


Gambar 3.13. Hasil Tes Endpoint REST API Method Update

Gambar 3.13 merupakan hasil dari pengetesan REST API *method update* atau *patch*. Pada *method update*, dikirimkan satu parameter id dan data berbentuk JSON sebagai *request body* yang terdiri atas *header* dan *detail*. *Detail* dapat dikirim lebih dari satu dalam *array*. Jika *request* sukses, *response* status akan menunjukkan 200 OK dan mengembalikan sebuah data JSON sebagai *response body*.

B.5 Delete

Berikut merupakan contoh pengetesan REST API *method delete* pada fitur *inventory item withdraw reservation*



Gambar 3.14. Hasil Tes Endpoint REST API Method Delete

Gambar 3.14 merupakan hasil dari pengetesan REST API *method delete*. Pada *method delete*, dikirimkan satu parameter id untuk data yang ingin dihapus. Jika *request* sukses, *response* status akan menunjukkan 200 OK dan mengembalikan sebuah data JSON sebagai *response body*.

3.4 Kendala dan Solusi yang Ditemukan

A. Kendala

Dalam melaksanakan kerja magang, terdapat beberapa kendala yang dihadapi yaitu:

1. Belum pernah menggunakan sistem ERP sebelumnya sehingga mengalami kesulitan dalam mengoperasikannya.
2. Arsitektur modular monolitik yang sedikit membingungkan karena pengembangan aplikasi saat masa perkuliahan menggunakan arsitektur monolitik biasa.

3. Waktu WFO hanya sekali dalam seminggu sehingga mengalami kesulitan pada beberapa minggu pertama kegiatan magang berlangsung.

B. Solusi

Berdasarkan kendala yang ada, terdapat beberapa solusi yang dilakukan yaitu:

1. Menonton rekaman ulang video pembelajaran ERP secara mandiri.
2. Mempelajari struktur projek secara mandiri untuk mendalami projek dengan arsitektur modular monolitik yang memiliki banyak modul. Pembelajaran dilakukan dengan mencoba melakukan *cloning* modul dan mencoba memodifikasi beberapa bagian untuk melihat efek yang ditimbulkan.
3. Penggunaan platform Discord untuk berdiskusi dengan kelompok sendiri maupun kelompok lain untuk mengatasi kendala waktu WFO yang hanya sekali dalam seminggu.