BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Posisi yang diberikan selama masa magang adalah sebagai *fullstack developer* dalam sebuah tim di PT Cranium Royal Aditama di bawah bimbingan dan arahan Pak Sugito, *supervisor* dan kepala tim dalam proyek ERP. Pelaksanaan magang dilaksanakan bersama 6 anggota magang dengan membagi menjadi 2 tim pengembangan. Masing-masing tim diberikan pembinaan mentor dari kelompok magang *batch* sebelumnya. Komunikasi dengan tim berlangsung melalui *platform online* seperti *Discord*, *Google Meet*, dan *Google Spreadsheet* untuk membagikan perkembangan pekerjaan.

3.2 Tugas yang Dilakukan

Tugas yang diberikan selama melaksanakan pekerjaan magang yaitu mempelajari secara mendalam pembentukan modul ERP yang digunakan di dalam perusahaan. Masa belajar modul ERP berlangsung selama 3 bulan awal bekerja. Pembelajaran dimulai dari memahami aturan dasar *clean code*, database *PostgreSQL*, *Java Springboot framework*, dan *React Typescript framework*. Setiap pembelajaran ditujukan untuk menguasai modul ERP perusahaan yang telah teruji berjalan baik untuk proses bisnis.

Pembelajaran didukung oleh modul latihan ERP yang diberikan oleh perusahaan. Modul latihan terbagi menjadi dua, yaitu modul *backend* menggunakan *Java Springboot* dan *frontend* menggunakan *ReactTS*. Dari modul tersebut, dilakukan *cloning* pada objek baru sebagai proyek yang ditugaskan untuk kelompok magang. Konsep yang dipelajari selama mempelajari backend, yaitu pembuatan *REST API* menggunakan *java springboot* dan pembuatan komponen dengan *API* yang telah dibuat menggunakan *React Typescript*. Pembelajaran yang dilakukan bertujuan untuk memahami alur kerja membuat program dalam sebuah tim yang dapat mengintegrasikan proses *backend* dan *frontend*.

Setelahnya 3 bulan berikutnya diberikan proyek pada modul *purchasing* dan *production* dengan melakukan *cloning*. Proses *cloning* ini dilakukan dengan *compile* modul proyek ERP cranium untuk setiap modul ERP yang ada. Tujuannya adalah untuk menyalin fungsionalitas dari modul yang sudah ada, sehingga kami

dapat mempelajari struktur dan cara kerjanya secara mendalam.

Tugas yang diberikan yaitu melakukan pengembangan pembuatan unit test untuk fitur-fitur pada submodul ERP. Fitur-fitur submodul ini meliputi *purchasing request*, *purchasing order item receipt*, *purchasing bill*, *purchasing order return*, *purchasing bill return*, *production order item receipt*, dan *production planned order*. Pembuatan *Unit Test* ini bertujuan untuk memverifikasi bahwa setiap bagian dari service berfungsi sesuai dengan yang diharapkan dan tidak ada kesalahan yang tidak terdeteksi selama pengembangan.

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	Instalasi alat yang digunakan dan pengenalan modul proyek latihan ERP
2	Pengenalan PostgreSQL dan inisialiasi database
3	Melakukan CRUD pada sistem latihan ERP menggunakan Springboot
4	Melakukan unit test pada sistem latihan ERP menggunakan Springboot
5	Melakukan cloning entity baru pada sistem latihan ERP
6	Melakukan cloning unit test pada sistem latihan ERP
7	Pengenalan modul latihan frontend menggunakan ReactTS
8	Mengintegrasikan backend dan frontend
9	Melakukan cloning component pada modul latihan frontend
10	Mendalami pemahaman struktur modul latihan frontend
11	Mendalami pemahaman penggunaan providers pada modul latihan frontend
12	Mempelajari unit test pada frontend
13	Mendalami pemahaman alur latihan frontend
14	Pembagian tim pengembangan dan briefing proyek
15	Pengembangan unit testing modul purchasing proyek cranium ERP
16	Pengembangan unit testing modul product proyek cranium ERP

Tabel 3.1 (lanjutan): Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
17	Evaluasi pengerjaan proyek unit testing cranium ERP
18	Memperbaiki permasalahan yang muncul akibat git merge dari branch development
19	Pengarahan dan briefing proyek cranium frontend

3.3.1 Pembelajaran

A. *Enterprise Resource Planning*

Gambar 3.1. Tampilan Cranium ERP
Sumber: cranium.id

Enterprise Resource Planning (ERP) adalah perangkat lunak yang mengintegrasikan dan mengelola berbagai proses bisnis suatu perusahaan secara keseluruhan [2]. ERP memberikan visibilitas operasional yang lebih baik, mengurangi duplikasi data, dan meningkatkan efisiensi proses. Dengan data yang akurat dan *real-time*, perusahaan dapat mengambil keputusan yang lebih baik sehingga membantu mencapai efisiensi operasional dan pertumbuhan bisnis yang berkelanjutan.

Sistem ERP yang digunakan oleh Cranium terdiri dari enam modul utama yang masing-masing dirancang untuk mengelola berbagai aspek bisnis perusahaan. Modul pertama adalah *Sales & Distribution*, yang bertanggung jawab untuk mengelola dan memantau proses penjualan. Modul *Purchasing*, yang digunakan untuk mengatur dan melacak proses pembelian. Modul *Finance*, untuk mengelola semua aspek keuangan perusahaan. Modul *Production Planning*, digunakan untuk merencanakan dan mengatur kegiatan produksi. Modul *Inventory Control*

bertugas untuk mengelola stok barang dan memastikan ketersediaan yang optimal. Terakhir, modul *Accounting* digunakan untuk menyusun dan memelihara catatan keuangan perusahaan. Dengan kombinasi modul-modul ini, sistem ERP Cranium memungkinkan perusahaan untuk mengintegrasikan dan mengoptimalkan berbagai proses bisnis mereka dalam satu platform yang terpadu, meningkatkan efisiensi operasional serta memperkuat pengambilan keputusan berbasis data yang akurat.

B. PosgresSQL

PostgreSQL adalah sistem manajemen basis data (DBMS) relasional *open-source* [3]. Sebagai DBMS relasional, PostgreSQL membantu dalam mengatur dan mengelola relasi antar tabel dalam basis data sistem ERP. Keunggulan PostgreSQL terletak pada fleksibilitasnya dalam diintegrasikan dengan lingkungan pengembangan *Java Springboot*. Selain itu, Database relasional digunakan karena terdapat banyak relasi antar table pada sistem ERP.

C. Java Springboot

Java Springboot adalah sebuah *framework* yang didesain untuk memudahkan pengembangan aplikasi *Java* dengan cara yang efisien namun tetap konvensional. *Framework* ini dibangun di atas platform *Spring* yang sudah mapan, yang awalnya dikembangkan sebagai respons terhadap kompleksitas pengembangan *Java* pada tahun 2003 [4]. *Spring* menyediakan pendekatan sederhana dan modular untuk membuat aplikasi dengan bahasa pemrograman *Java* yang populer.

Dalam konteks pengembangan sistem ERP, PT Cranium Royal Aditama memilih untuk menggunakan *Java Springboot* karena fleksibelitas dan kemudahan pengembangannya. Sebagai *framework* berbasis *Java*, *Springboot* memberikan fleksibilitas yang tinggi dan dapat dijalankan di berbagai sistem operasi, memudahkan integrasi dengan infrastruktur perusahaan yang sudah ada. Selain itu, penerapan model arsitektur modular monolitik pada *Java Springboot* akan membantu dalam *scalability* dan *maintanability*. Modular monolitik juga mempermudah pengembangan arsitektur apabila ingin meningkatkan arsitektur menjadi *microservice*.

D. React Typescript

Framework ReactTS adalah sebuah library *JavaScript* yang digunakan untuk membangun antarmuka pengguna yang interaktif dan dinamis dalam aplikasi web [5]. Dikembangkan oleh Facebook, *ReactTS* menyediakan pendekatan yang efisien dalam pengembangan dengan komponen-komponen UI yang modular. ReactTS memungkinkan pengembang untuk memisahkan tampilan aplikasi menjadi komponen-komponen yang dapat dikelola dan digunakan kembali dengan mudah.

Perusahaan memilih menggunakan *React TypeScript* sebagai library untuk membangun *User Interface* dari sistem ERP. Alasan pemilihan React adalah karena kemampuan *live render* dan banyaknya kontributor yang membuatnya memiliki referensi yang lengkap dan beragam dibandingkan dengan *library JavaScript* lainnya. *TypeScript*, sebagai bahasa pemrograman yang didukung oleh *React*, memiliki konsep yang mirip dengan *JavaScript*, tetapi setiap variabel harus memiliki tipe data yang didefinisikan. Hal ini membuat *TypeScript* terasa lebih ketat, namun sekaligus memperbaiki keterbacaan dan kualitas kode.

3.3.2 Pengembangan ERP Cranium

Java Springboot sebagai *tech stack* yang digunakan pada sistem Cranium ERP memiliki alur kerja yang menghubungkan client dengan database melalu beberapa tahapan. Tahapan tersebut dimulai dari tahap HTTP *request* melalui REST API, service sebagai *business logic* hingga dapat melakukan proses CRUD dengan database [6]. Alur kerja ini merupakan gambaran umum yang perlu dipahami sebelum mengerjakan proyek yang diberikan perusahaan. Flowchart dari arsitektur spring dapat dilihat melalui gambar 3.2.

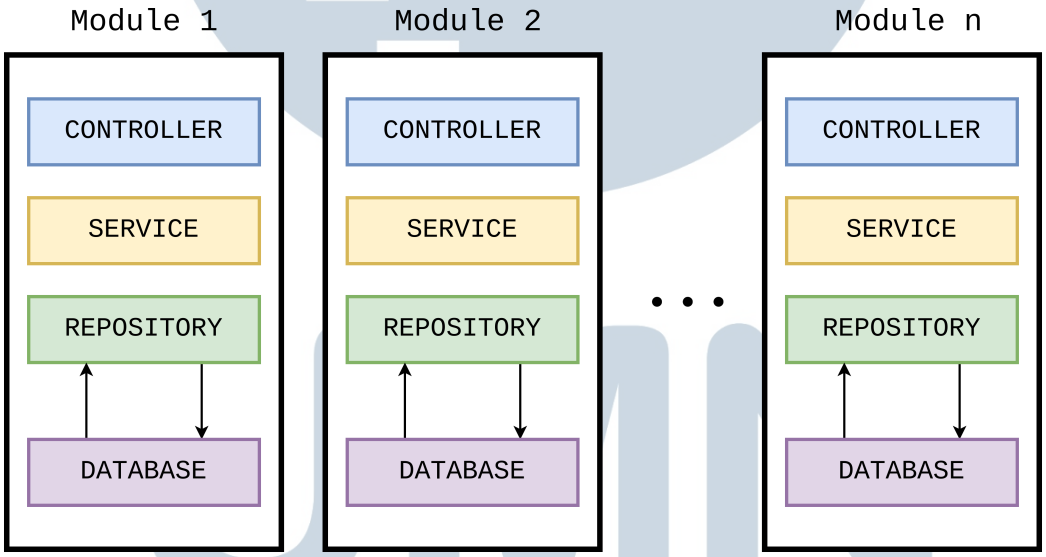
Gambar 3.2. Arsitektur Spring Boot

Setiap modul pada perusahaan memiliki proses yang sama sehingga alur tersebut perlu dipahami sebelum mendapatkan proyek. Sehingga memahami arsitektur tersebut merupakan langkah awal pada pengerjaan proyek Cranium ERP.

Dengan memahami cara kerja tersebut, pengerjaan Unit Test dari Cranium ERP mulai dapat dikerjakan dengan pemahaman cara kerja arsitektur sebenarnya.

A. Springboot Modular Monolitik

Pembangunan sistem ERP menggunakan *Java Springboot* dengan arsitektur modular monolitik. Arsitektur ini dapat mengembangkan sistem dengan *codebase* yang modular. Penerapannya *codebase* dapat dibagi menjadi beberapa modul yang dapat berdiri secara independen, sehingga perubahan dari modul tidak memengaruhi modul lainnya [7]. Dalam arsitektur monolitik, komunikasi antar modul dilakukan dengan menggunakan API publik dan menggunakan logika antar modul menggunakan metode publik. Hal tersebut telah sesuai dengan menggunakan *tech stack Java Springboot*.



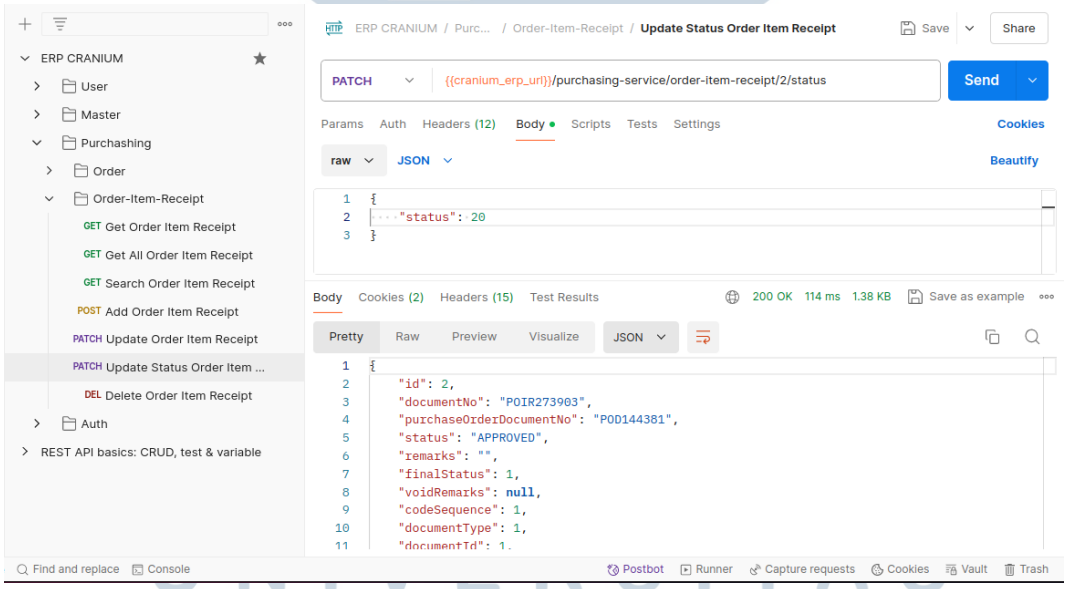
Gambar 3.3. Arsitektur Modular Monolitik

Gambar 3.3 tidak hanya menunjukkan skema arsitektur pada *Springboot*, tetapi juga menjelaskan alur kerja *Springboot* saat menangani permintaan dan respons. *Client* mengirim permintaan menggunakan metode *REST API* (*get, post, patch, delete*). Jika menggunakan metode *post* atau *patch*, JSON akan diubah menjadi *Data Transfer Object* (DTO), yang merupakan objek untuk membungkus data permintaan atau respons. Setelah itu, *API layer* atau *controller* menerima objek tersebut dan meneruskannya ke *business layer* atau *service*. *Service* kemudian memanggil *persistence layer* atau *repository*, yang berinteraksi langsung dengan

database untuk operasi CRUD. Setiap kali operasi CRUD dilakukan, *repository* mengembalikan sebuah *entity*, yaitu objek yang mewakili satu *record database*, ke *service*. *Service* kemudian mengubah *entity* menjadi DTO dan menangani logika bisnis sebelum mengembalikan DTO tersebut ke *controller*. *Controller* mengubah DTO menjadi JSON sebagai respons yang diterima oleh client.

B. Pengujian API untuk *Method Update Status*

Pengujian dan pengelompokan API akan dilakukan menggunakan postman sebagai alat. Postman merupakan alat yang digunakan untuk melakukan pengujian, pengelolaan, dan dokumentasi pada pembuatan API [8]. API yang diuji merupakan REST API yang berasal dari modul *purchasing* dan *production*. Pemformatan API untuk *method update status* menggunakan HTTP *method PATCH* dengan URL /{modul}-service/{submodul}/{id}/status dengan body ”status”. Berikut contoh dari pengujian API untuk entity *order item receipt* pada modul *purchasing* pada gambar 3.4.



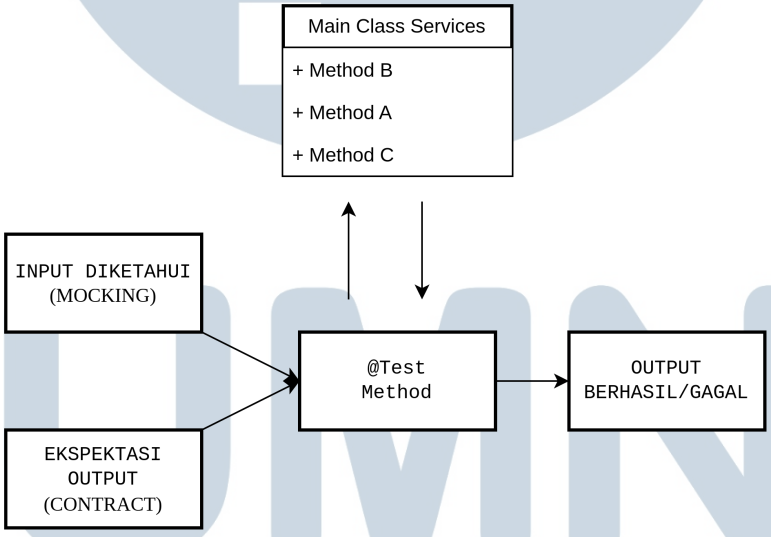
Gambar 3.4. Contoh pengujian API *order Item receipt* menggunakan Postman

Hasil dari pengujian tersebut menampilkan HTTP *status code* 200 yang menandakan permintaan berhasil diterima dan mengembalikan nilai oleh server. Perubahan status diperlihatkan pada JSON Body dari *response* yang diberikan oleh server menjadi ”APPROVED”. Untuk mengubah nilai menjadi status lainnya, dapat dilakukan dengan mengubah body dari HTTP Request tersebut dengan nilai-nilai

yang telah ditetapkan. Nilai terebut diinisialisasikan dengan enumerasi: *1* sebagai ”*NEW*”, *10* sebagai ”*APPROVED_CANCELED*”, *20* sebagai ”*APPROVED*”, *25* sebagai ”*APPROVE_CLOSED*”, dan *30* sebagai ”*VOID*”. Proses perubahan dari status tersebut mengacu kepada kondisi dari proses bisnis yang sedang terjadi.

C. Pembuatan Unit Test

Unit test merupakan program yang diciptakan untuk menguji kelayakan dari program utama yang telah sesuai dengan alur yag diharapkan. Pada pembuatan program unit test pada backend, akan diuji kesesuaian antara *client request API* dengan response yang dihasilkan. Pengujian *unit test* saat ini terbatas kepada pembuatan *unit test* untuk *layer service* dengan metode *update status*. Framework yang digunakan pada saat pengujian antara lain JUnit, Mockito, dan Spring Cloud Contract yang hubungannya dapat dilihat pada gambar 3.5.



Gambar 3.5. Diagram unit test

Setiap framework pengujian memiliki tugasnya masing-masing yang saling mendukung dan berkaitan. Alur pengerjaannya pada tahap awal akan dibuat terlebih dahulu mocking data dan contract. Mocking data akan menggunakan framework mockito untuk membat mock objek dan mengisolasi unit kode yang diuji [9]. Kontrak menggunakan Spring Cloud Contract untuk menulis kontrak antara layanan dan memastikan interaksi sesuai dengan kontrak yang telah dituliskan [10]. Dan pengujian akan menggunakan JUnit sebagai framework pengujian yang menyediakan metode verifikasi dengan notasi-notasi yang disediakan [11].

Pengujian yang dilakukan akan menjalankan Class Service pada *codeclass* yang telah dibuat.

C.1 *Service Test*

Pengujian *unit test* menggunakan *mocking* pada data *request* dan *response* yang seolah-olah memanggil API. Pada pengujian *update status*, akan dibuat *mocking* untuk status yang diubah menjadi value *"APPROVED"*. *Mocking* tersebut akan dibandingkan dengan *contract* yang merupakan simulasi dari *response* yang diharapkan. Pembandingan tersebut menggunakan *library* dari *Java Springboot* yaitu *assertionEquals* dari JUnit.

Proses pembandingan akan dijalankan pada program service dari unit test. Berikut lampiran cara kerja bagian service dari unit test untuk membandingkan Id dan Status pada gambar 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, 3.12.

```
@Test  Alfarizky Oscandar *
void testUpdateItemReceiptStatus() throws DataNotFoundException {
    OrderItemReceiptUpdateStatusDto orderItemReceiptUpdateStatusDto = OrderItemReceiptUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    OrderItemReceiptDto orderItemReceiptDto = orderItemReceiptService
                                            .updateStatusOrderItemReceipt( id: 1L, orderItemReceiptUpdateStatusDto);

    assertEquals(orderItemReceiptDto.getStatus(), PurchasingStatus.APPROVED.name());
    assertEquals(orderItemReceiptDto.getId(),  actual: 1);
}
```

Gambar 3.6. *Script unit test* modul *purchasing* submodul *order return*

```
@Test  Alfarizky Oscandar
void testUpdateOrderReturnStatus() throws DataNotFoundException{
    OrderReturnUpdateStatusDto orderReturnUpdateStatusDto = OrderReturnUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    OrderReturnDto orderReturnDto = orderReturnService.updateOrderReturnStatus( id: 1L, orderReturnUpdateStatusDto);

    assertEquals(orderReturnDto.getStatus(), PurchasingStatus.APPROVED.name());
    assertEquals(orderReturnDto.getId(),  actual: 1);
}
```

Gambar 3.7. *Script unit test* modul *purchasing* submodul *order return*

```java
@Test
void testUpdateBillStatus() throws DataNotFoundException {
    BillUpdateStatusDto billUpdateStatusDto = BillUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    BillDto billDto = billService.updateBillStatus( id: 1L, billUpdateStatusDto);

    assertEquals(billDto.getStatus(), PurchasingStatus.APPROVED.name());
    assertEquals(billDto.getId(),  actual: 1);

}
```

Gambar 3.8. *Script unit test* modul *purchasing* submodul *Bill*

```java
@Test
void testUpdateReturnBillStatus() throws DataNotFoundException {
    ReturnBillUpdateStatusDto returnBillUpdateStatusDto = ReturnBillUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    ReturnBillDto returnBillDto = returnBillService.updateStatusReturnBill( id: 1L, returnBillUpdateStatusDto);

    assertEquals(returnBillDto.getStatus(), PurchasingStatus.APPROVED.name());
    assertEquals(returnBillDto.getId(),  actual: 1);
}
```

Gambar 3.9. *Script unit test* modul *purchasing* submodul *return bill*

```java
@Test
void testUpdateRequestStatus() throws DataNotFoundException {
    RequestUpdateStatusDto requestUpdateStatusDto = RequestUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    RequestDto requestDto = requestService.updateStatusRequest( id: 1L, requestUpdateStatusDto);

    assertEquals(requestDto.getStatus(), PurchasingStatus.APPROVED.name());
    assertEquals(requestDto.getId(),  actual: 1);

}
```

Gambar 3.10. *Script unit test* modul *purchasing* submodul *request*

```
@Test
void testUpdatePlannedOrderStatus() throws DataNotFoundException {
    PlannedOrderUpdateStatusDto plannedOrderUpdateStatusDto = PlannedOrderUpdateStatusDto
            .builder()
            .status((short) ProductionStatus.APPROVED.getValue())
            .build();

    PlannedOrderDto plannedOrderDto = plannedOrderService
                                        .updateProductionPlannedOrderStatus( id: 1L, plannedOrderUpdateStatusDto);

    assertEquals(plannedOrderDto.getStatus(), ProductionStatus.APPROVED.name());
    assertEquals(plannedOrderDto.getId(), actual: 1);
}
```

Gambar 3.11. *Script unit test* modul *production* submodul *planned order*

```
@Test
void testUpdateProductionOrderItemReceiptStatus() throws DataNotFoundException {
    OrderItemReceiptUpdateStatusDto orderItemReceiptUpdateStatusDto = OrderItemReceiptUpdateStatusDto
            .builder()
            .status((short) ProductionStatus.APPROVED.getValue())
            .build();

    OrderItemReceiptDto orderItemReceiptDto = orderItemReceiptService
                                        .updateStatusOrderItemReceipt( id: 1L, orderItemReceiptUpdateStatusDto);

    assertEquals(orderItemReceiptDto.getStatus(), ProductionStatus.APPROVED.name());
    assertEquals(orderItemReceiptDto.getId(), actual: 1);
}
```

Gambar 3.12. *Script unit test* modul *production* submodul *order item receipt*

C.2 *Contract Test*

Pembuatan *mocking* atau tiruan untuk data akan dilakukan pada *contract test*. *Mocking* adalah teknik dalam pengujian unit untuk membuat tiruan atau *"mock"* dari kelas atau objek lain. *Mocking* tersebut bertujuan untuk menghasilkan simulasi untuk kondisi yang akan diuji. Pada saat ini, dilakukan pembuatan untuk menyimulasikan proses dari metode *update status* yang sukses. Berikut script program dari contract base yang telah dibuat pada gambar 3.13, 3.14, 3.15, 3.16, 3.17, 3.18, 3.19.

```
private void updateOrderItemReceiptStatus_should_be_success() {  1 usage
    OrderItemReceiptUpdateStatusDto orderItemReceiptUpdateStatusDto = OrderItemReceiptUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    OrderItemReceiptDto orderItemReceiptDto = new OrderItemReceiptDto();
    orderItemReceiptDto.setId(1L);
    orderItemReceiptDto.setDocumentNo("POIR000011");
    orderItemReceiptDto.setPurchaseOrderDocumentNo("PO000011");
    orderItemReceiptDto.setStatus(PurchasingStatus.APPROVED.name());
    orderItemReceiptDto.setRemarks("test remark");
    orderItemReceiptDto.setFinalStatus(1);
    orderItemReceiptDto.setPoirPbIsFullfilled(true);
    orderItemReceiptDto.setPoirPbIsFullfilled(true);
    orderItemReceiptDto.setSupplierDocumentReferenceNo("SUP000011");
    orderItemReceiptDto.setPurchaseOrderId(1L);
    orderItemReceiptDto.setPlantId(1L);
    orderItemReceiptDto.setSupplierId(1L);
    orderItemReceiptDto.setSupplierAddress("Supplier Address 1");
    orderItemReceiptDto.setCreationDate(LocalDate.of( year: 2023, Month.MARCH,  dayOfMonth: 28));
    orderItemReceiptDto.setCreatedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    orderItemReceiptDto.setCreatedBy(1L);
    orderItemReceiptDto.setOpenedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    orderItemReceiptDto.setOpenedBy(11L);
    orderItemReceiptDto.setUpdatedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
```

Gambar 3.13. *Script contract test* modul *purchasing* submodul *order item receipt*

```
private void updateOrderReturnStatus_should_be_success(){  1 usage
    OrderReturnUpdateStatusDto orderReturnUpdateStatusDto = OrderReturnUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    OrderReturnDto orderReturnDto = new OrderReturnDto();
    orderReturnDto.setId(1L);
    orderReturnDto.setPlantId(1L);
    orderReturnDto.setDocumentNo("POR000001");
    orderReturnDto.setStatus(PurchasingStatus.APPROVED.name());
    orderReturnDto.setSupplierId(1L);
    orderReturnDto.setSupplierAddress("Supplier Address 1");
    orderReturnDto.setRemarks("test remark");
    orderReturnDto.setCreationDate(LocalDate.of( year: 2023, Month.MARCH,  dayOfMonth: 23));
    orderReturnDto.setCreatedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    orderReturnDto.setCreatedBy(1L);
    orderReturnDto.setUpdatedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    orderReturnDto.setUpdatedBy(1L);
    orderReturnDto.setOpenedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    orderReturnDto.setOpenedBy(1L);

    Mockito.when(orderReturnService.updateOrderReturnStatus( id: 1L, orderReturnUpdateStatusDto)).thenReturn(orderReturnDto);
}
```

Gambar 3.14. *Script contract test* modul *purchasing* submodul *order return*

```
private void updateRequestStatus_should_be_success() {  1 usage
    RequestUpdateStatusDto requestUpdateStatusDto = RequestUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    RequestDto requestDto = new RequestDto();
    requestDto.setId(7L);
    requestDto.setDocumentNo("PRO0000007");
    requestDto.setCreationDate(LocalDate.of( year: 2023, Month.JANUARY,  dayOfMonth: 30));
    requestDto.setPlantId(1L);
    requestDto.setStatus(PurchasingStatus.APPROVED.name());
    requestDto.setOpened(true);

    Mockito.when(requestService.updateStatusRequest( id: 7L, requestUpdateStatusDto)).thenReturn(requestDto);

}
```

Gambar 3.15. *Script contract test* modul *purchasing* submodul *request*

```
private void updateReturnBillStatus_should_be_success() {  1 usage
    ReturnBillUpdateStatusDto returnBillUpdateStatusDto = ReturnBillUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    ReturnBillDto returnBillDto = new ReturnBillDto();
    returnBillDto.setId(7L);
    returnBillDto.setDocumentNo("PRB0003");
    returnBillDto.setOrderReturnDocumentNo("PRO0000001");

    Mockito.when(returnBillService.updateStatusReturnBill( id: 7L, returnBillUpdateStatusDto)).thenReturn(returnBillDto);
}
```

Gambar 3.16. *Script contract test* modul *purchasing* submodul *return bill*

```
private void updateBillStatus_should_be_success() {  1 usage
    BillUpdateStatusDto billUpdateStatusDto = BillUpdateStatusDto
            .builder()
            .status((short) PurchasingStatus.APPROVED.getValue())
            .build();

    BillDto billDto = new BillDto();
    billDto.setId(7L);
    billDto.setDocumentNo("PBL0000002");
    billDto.setReferenceDocumentNo("POR0000002");
    billDto.setPlantId(1L);
    billDto.setSupplierId(1L);
    billDto.setSupplierAddress("Supplier address");
    billDto.setPaymentMethod(PurchasingPaymentMethod.CASH.name());
    billDto.setTermOfPayment(30);
    billDto.setAdditionalPphPercentage(10);
    billDto.setAdditionalDiscountPercentage(10);
    billDto.setAdditionalDiscountAmount(10);
    billDto.setRemarks("testBill");
    billDto.setCreationDate(LocalDate.of( year: 2023, Month.MARCH,  dayOfMonth: 23));
    billDto.setCreatedBy(1L);
    billDto.setUpdatedBy(1L);
    billDto.setDeleted(false);
    billDto.setVersion(1L);

    Mockito.when(billService.updateBillStatus( id: 7L, billUpdateStatusDto)).thenReturn(billDto);
```

Gambar 3.17. *Script contract test* modul *purchasing* submodul *bill*

```
private void updatePlannedOrderStatus_should_be_success() {  1 usage
    PlannedOrderUpdateStatusDto plannedOrderUpdateStatusDto = PlannedOrderUpdateStatusDto
            .builder()
            .status((short) ProductionStatus.APPROVED.getValue())
            .build();

    PlannedOrderDto plannedOrderDto = new PlannedOrderDto();
    plannedOrderDto.setId(4L);
    plannedOrderDto.setDocumentNo("PRO0000009");
    plannedOrderDto.setCreationDate(LocalDate.of( year: 2023, Month.JANUARY,  dayOfMonth: 30));
    plannedOrderDto.setPlantId(1L);
    plannedOrderDto.setStatus(ProductionStatus.APPROVED.name());
    plannedOrderDto.setCreatedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    plannedOrderDto.setCreatedBy(1L);
    plannedOrderDto.setUpdatedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    plannedOrderDto.setUpdatedBy(1L);
    plannedOrderDto.setDeleted(false);
    plannedOrderDto.setVersion(1L);
    plannedOrderDto.setVoidRemarks("test remarks");
    plannedOrderDto.setProductionPlannedOrderDetailsDtos(null);

    Mockito.when(plannedOrderService.updateProductionPlannedOrderStatus( id: 4L, plannedOrderUpdateStatusDto)).thenReturn(plannedOrderDto);
}
```

Gambar 3.18. *Script contract test* modul *production* submodul *planned order*

```
private void updateProductionOrderItemReceiptStatus_should_be_success(){  1 usage
    OrderItemReceiptUpdateStatusDto orderItemReceiptUpdateStatusDto = OrderItemReceiptUpdateStatusDto
            .builder()
            .status((short) ProductionStatus.APPROVED.getValue())
            .build();
    OrderItemReceiptDto orderItemReceiptDto = new OrderItemReceiptDto();
    orderItemReceiptDto.setId(1L);
    orderItemReceiptDto.setDocumentNo("PIR1");
    orderItemReceiptDto.setOrderItemReceiptDetailDtos(null);
    orderItemReceiptDto.setProductionOrderDocumentNo("PRO0000001");
    orderItemReceiptDto.setStatus("APPROVED");
    orderItemReceiptDto.setFinalStatus(1);
    orderItemReceiptDto.setRemarks("remarks header");
    orderItemReceiptDto.setVoidRemarks("");
    orderItemReceiptDto.setCreationDate(LocalDate.of( year: 2023, Month.MARCH,  dayOfMonth: 23));
    orderItemReceiptDto.setPlantId(1L);
    orderItemReceiptDto.setCreditCoaId(1L);
    orderItemReceiptDto.setOpened(true);
    orderItemReceiptDto.setUpdatedBy(11L);
    orderItemReceiptDto.setUpdatedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    orderItemReceiptDto.setOpenedBy(11L);
    orderItemReceiptDto.setOpenedAt(LocalDateTime.of( year: 2023, Month.JANUARY,  dayOfMonth: 30,  hour: 15,  minute: 30,  second: 59));
    orderItemReceiptDto.setDeleted(false);
    orderItemReceiptDto.setVersion(1L);
    Mockito.when(orderItemReceiptService.updateStatusOrderItemReceipt( id: 1L, orderItemReceiptUpdateStatusDto)).thenReturn(orderItemReceiptDto);
}
```

Gambar 3.19. *Script contract test* modul *production* submodul *order item receipt*

C.3 *Groovy*

Groovy adalah file yang berisi definisi kontrak antara *client* dan *server* layanan dalam format Groovy. Kontrak ini mendefinisikan bagaimana interaksi antara konsumen dan produsen harus berlangsung, termasuk permintaan yang dikirim oleh *client* dan *response* yang diharapkan dari produsen. Groovy akan melakukan validasi dari cara kerja dari *request* dan *response* yang dihasilkan dari *unit test*. Contoh script dari program groovy dapat dilihat pada gambar 3.20.

```groovy
package contracts.purchasing.orderItemReceipt

import org.springframework.cloud.contract.spec.Contract

Contract.make {
    description description:"update order item receipt id = 1"

    request {
        url url:"/purchasing-service/order-item-receipt/1/status"
        method PATCH()
        headers {
            contentType(applicationJson())
            header "X-Api-Version": "1"
            header "Accept-Language": "id-ID"
        }
        body(
                status: 20,
        )
    }

    response {
        status OK()
        headers {
            contentType(applicationJson())
        }
        body(
```

Gambar 3.20. Contoh *script* groovy

C.4 Output

Ketika melakukan *compile maven* dari modul pada proyek cranium erp, *unit test* akan otomatis dijalankan. Apabila terdapat kegagalan dalam *unit test*, program akan dihentikan dan *compile* tidak berhasil dijalankan. Luaran dari *unit test* tersebut akan ditampilkan sebagai *log* hasil *compile program*. Hasil tersebut akan memperlihatkan pesan *error*, *failure*, ataupun unit test yang terlewat dilakukan (*skipped*). Hasil log output dapat dilihat melalui folder /target/surfire-reports pada gambar 3.21.

```
alfarosca@alfaroArch /home/alfarosca/code-workspaces/Cranium ERP/cranium-erp-backend/erp/purchasing-service/target/surefire-reports
 logs                 classes                                    13   id.cranium.erp.purchasing.contract.purchasing.BillDetailTest.txt
 src                  generated-sources                           1   id.cranium.erp.purchasing.contract.purchasing.BillTest.txt
 target               generated-test-sources                      2   id.cranium.erp.purchasing.contract.purchasing.OrderDetailTest.txt
 mvnw                 maven-archiver                              1   id.cranium.erp.purchasing.contract.purchasing.OrderItemReceiptTest.txt
 mvnw.cmd             maven-status                                2   id.cranium.erp.purchasing.contract.purchasing.OrderReturnDetailTest.txt
 pom.xml              site                                        1   id.cranium.erp.purchasing.contract.purchasing.OrderReturnTest.txt
                      stubs                                       1   id.cranium.erp.purchasing.contract.purchasing.OrderTest.txt
                      surefire-reports                           34   id.cranium.erp.purchasing.contract.purchasing.RequestDetailTest.txt
                      test-classes                                4   id.cranium.erp.purchasing.contract.purchasing.RequestTest.txt
                      jacoco.exec                             1.51 M  id.cranium.erp.purchasing.contract.purchasing.ReturnBillTest.txt
                      purchasing-service-0.0.1-SNAPSHOT-stubs.jar 133 K  id.cranium.erp.purchasing.purchasingservice.PurchasingServiceApplica~.txt
                      purchasing-service-0.0.1-SNAPSHOT.jar       314 K  id.cranium.erp.purchasing.service.BillServiceTest.txt
                                                                         id.cranium.erp.purchasing.service.OrderItemReceiptServiceTest.txt
                                                                         id.cranium.erp.purchasing.service.OrderReturnServiceTest.txt
                                                                         id.cranium.erp.purchasing.service.OrderServiceTest.txt
                                                                         id.cranium.erp.purchasing.service.RequestServiceTest.txt
                                                                         id.cranium.erp.purchasing.service.ReturnBillServiceTest.txt
```

Gambar 3.21. *Output unit test*

Selain menghasilkan pesan sebagai output, unit test juga menghasilkan luaran berupa hasil pembandingan yang terjadi. Pembandingan ini berguna sebagai solusi untuk melakukan *debugging* jika terdapat kendala selama pembuatan *unit test*. Berikut lampiran dari contoh output yang dihasilkan pada gambar 3.22.

```
@Test
public void validate_updateProductionOrderItemReceiptStatusSuccess() throws Exception {
    // given:
        MockMvcRequestSpecification request = given()
                .header("Content-Type", "application/json")
                .header("X-Api-Version", "1")
                .header("Accept-Language", "id-ID")
                .body("{\"status\":20}");

    // when:
        ResponseOptions response = given().spec(request)
                .patch("/production-service/order-item-receipt/1/status");

    // then:
        assertThat(response.statusCode()).isEqualTo(200);
        assertThat(response.header("Content-Type")).matches("application/json.*");

    // and:
        DocumentContext parsedJson = JsonPath.parse(response.getBody().asString());
        assertThatJson(parsedJson).field("['id']").isEqualTo(1);
        assertThatJson(parsedJson).field("['documentNo']").isEqualTo("PIR1");
        assertThatJson(parsedJson).field("['productionOrderDocumentNo']").isEqualTo("PRO0000001");
```

Gambar 3.22. *Output contract test*

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala

Berikut beberapa kendala yang dihadapi selama pelaksanaan proses magang:

1. Waktu WFO yang hanya dilaksanakan 1 hari dalam seminggu sehingga perlu penyesuaian komunikasi saat WFH.
2. Pada inisialisasi awal terdapat kendala untuk penginstalan pgadmin4 sebagai tools yag digunakan pada perangkat linux.
3. Kurangnya pemahaman dari proses bisnis yang ada pada sistem ERP Cranium.

3.4.2 Solusi

Berikut beberapa solusi dari kendala yang dihadapi selama pelaksanaan proses magang:

1. Melakukan diskusi *online* bersama tim magang disaat WFH.
2. Menggunakan Dbeaver sebagai alternatif dari pgadmin4.
3. Melakukan penelitian terhadap laporan magang pada PT Craniuam Royal Aditama pada batch sebelumnya dan aktif bertanya kendala yang dihadapi dengan mentor.