BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan kerja praktik di PT Cranium Royal Aditama, penulis tergabung dalam divisi *Product Development* dan menjalankan peran sebagai *Full Stack Developer* pada proyek pengembangan sistem *Enterprise Resource Planning* (ERP) milik perusahaan. Dalam periode tersebut, aktivitas utama penulis difokuskan pada keterlibatan dalam pengembangan modul *searching* yang berfungsi sebagai salah satu fitur pendukung utama dalam sistem ERP yang ditujukan bagi klien korporat Cranium.

Dalam menjalankan kegiatan kerja praktik sehari-hari, penulis memperoleh arahan dan pendampingan dari *mentor* serta *supervisor* yang bertugas memberikan bimbingan teknis, memantau progres pekerjaan, dan melakukan evaluasi terhadap hasil tugas yang diselesaikan. Selain itu, penulis juga bekerja sama dengan anggota tim pengembang ERP lainnya, di mana setiap anggota tim memiliki peran dan tanggung jawab masing-masing dalam mendukung proses pengembangan sistem secara keseluruhan.

Struktur kedudukan dalam proyek ERP Cranium dapat dijabarkan sebagai berikut:

1. *Supervisor*

   Dalam pelaksanaan kerja praktik ini, terdapat dua pihak yang berperan sebagai pengawas proyek. Pihak pertama adalah Bapak Sugito yang menjabat sebagai *VP of Engineering* sekaligus *Tech Lead*, serta pihak kedua yaitu Bapak Angga Tama selaku *Project Manager* ERP. Keduanya berperan dalam memberikan arahan teknis dan non-teknis kepada para *developer*, memantau progres pekerjaan, serta memastikan hasil yang dicapai telah memenuhi standar dan kebutuhan perusahaan. Selain itu, *supervisor* juga memberikan pemahaman awal mengenai budaya kerja, alur prosedur teknis, serta ekspektasi yang harus dipenuhi oleh mahasiswa magang selama menjalani kerja praktik di Cranium.

2. Mentor

   Dalam menjalankan aktivitas magang sebagai *full stack developer*,

penulis memperoleh pendampingan dari dua mentor yang masing-masing memiliki fokus pada pengembangan *back-end* dan *front-end* sistem ERP. Bimbingan yang diberikan mencakup pengenalan lingkungan pengembangan, pemahaman struktur dan alur kode program, serta penerapan kerja kolaboratif menggunakan sistem pengelolaan versi seperti *GitHub*. Para mentor juga melakukan peninjauan terhadap hasil pekerjaan untuk memastikan kesesuaian dengan standar perusahaan sebelum dilakukan evaluasi lebih lanjut oleh *supervisor*.

3. Mahasiswa Magang
   Sebagai mahasiswa magang, penulis memiliki tanggung jawab untuk melaksanakan setiap tugas yang diberikan sesuai dengan arahan *supervisor* serta tenggat waktu yang telah ditentukan. Selama periode kerja praktik, fokus utama penulis adalah terlibat dalam proses pengembangan modul *searching* pada sistem ERP.

3.2 Alur dan Prosedur Kerja Proyek

Selama masa kerja magang, PT Cranium Royal Aditama menerapkan berbagai peraturan dan prosedur untuk memastikan bahwa proses pengembangan proyek ERP dapat berlangsung secara optimal dan terstruktur. Seluruh prosedur tersebut disusun guna menjaga koordinasi yang efektif antar anggota tim serta memastikan setiap fitur dikembangkan sesuai standar yang telah ditetapkan perusahaan. Berikut merupakan alur kerja yang diterapkan dalam pelaksanaan proyek ERP Cranium:

1. Distribusi Tugas
   (a) Tugas baru diberikan setelah *developer* menyelesaikan seluruh pekerjaan sebelumnya, termasuk proses penggabungan *(merge)* ke dalam *development branch* pada *repository*.
   (b) Pemberian tugas dilakukan secara langsung oleh *supervisor* kepada *developer* dan dijelaskan melalui roadmap pengerjaan ERP. Penjelasan teknis terkait suatu tugas akan di-*brief* oleh *supervisor* baik secara pribadi maupun melalui *meeting* bersama seluruh tim ERP Cranium.
   (c) Jenis tugas dapat mencakup pengembangan tampilan antarmuka *(front-end)* maupun pengembangan logika sistem pada sisi *(back-end)*.

(d) Setiap tugas dikerjakan pada *branch* yang berbeda di dalam *repository*. Pengaturan ini bertujuan untuk mempermudah proses integrasi antar hasil pekerjaan *developer* serta meminimalkan kemungkinan terjadinya *conflict*.

2. Pengerjaan Tugas

   (a) Tugas yang telah dijelaskan melalui *briefing* oleh *supervisor* akan segera dikerjakan oleh para *developer*. Proses pengerjaan dilakukan dengan mengikuti arahan yang diberikan *supervisor* serta mematuhi standar kerja perusahaan dalam jangka waktu yang telah ditetapkan.

   (b) Mahasiswa magang diharapkan dapat menyelesaikan tugas secara mandiri dan optimal. Namun, apabila menemui hambatan selama proses pengerjaan, kedua *mentor* akan membantu memberikan arahan serta mencari solusi atas kendala yang muncul.

3. Pengujian Hasil Kerja

   Setiap hasil kerja mahasiswa magang harus melalui proses pengujian dan validasi sebelum diajukan dalam bentuk *pull request* serta digabungkan (*merge*) ke dalam *branch development* proyek ERP Cranium. Tahap pengujian ini dilakukan melalui dua metode utama:

   (a) *Unit Test* dan *Contract Test*

   Pengujian otomatis dilakukan menggunakan *Spring Boot Testing Framework* pada sisi *back-end* serta *Next.js* pada sisi *front-end*. Setiap *test case* disusun berdasarkan fungsi dan kebutuhan fitur yang dikembangkan pada ERP Cranium.

   (b) Pengujian Manual

   Pengujian manual dilakukan melalui antarmuka pengguna untuk memastikan fitur berjalan sesuai kebutuhan, serta tidak menimbulkan *bug* maupun *conflict* pada sistem.

4. Konsultasi dan Revisi

   Setelah tahap pengujian diselesaikan, hasil pekerjaan mahasiswa magang diserahkan kepada *mentor* untuk dilakukan peninjauan melalui mekanisme *pull request*. Proses peninjauan ini mencakup evaluasi terhadap struktur dan kualitas kode, kesesuaian dengan standar pengembangan ERP Cranium,

serta pemeriksaan potensi konflik dengan kode lain pada *branch development*. Apabila ditemukan bagian yang perlu diperbaiki atau disempurnakan, *mentor* akan memberikan catatan dan umpan balik secara langsung melalui *comment* pada *pull request*. Mahasiswa magang kemudian melakukan perbaikan sesuai masukan yang diberikan dalam rentang waktu yang telah disepakati. Jika selama proses revisi muncul kendala, mahasiswa magang dapat melakukan diskusi lanjutan dengan *mentor* untuk memperoleh solusi atau penyesuaian yang diperlukan. Setelah perbaikan selesai dilakukan, hasil revisi akan kembali ditinjau hingga dinyatakan sesuai.

5. Penggabungan Hasil Kerja
   Setelah seluruh hasil pekerjaan dinyatakan memenuhi standar dan tidak terdapat revisi lanjutan, *mentor* melanjutkan ke tahap pengintegrasian kode ke dalam *branch development* ERP Cranium. Proses *merge* dilakukan setelah memastikan bahwa fitur yang dikembangkan telah berjalan dengan stabil serta tidak menimbulkan konflik dengan kode lain yang telah ada. Dengan selesainya tahap ini, fitur atau pembaruan yang dikerjakan oleh mahasiswa magang secara resmi menjadi bagian dari pengembangan sistem dan dapat digunakan serta dikembangkan lebih lanjut oleh tim *developer*.

6. Finalisasi Hasil Kerja
   Suatu pekerjaan dinyatakan selesai apabila seluruh perubahan telah berhasil diintegrasikan ke dalam *branch development* ERP Cranium. Setelah proses penggabungan dilakukan, *branch* yang digunakan selama pengembangan fitur akan dihapus dari *repository*. Selanjutnya, *developer* menyiapkan *branch* baru sebagai dasar untuk pengerjaan tugas atau pengembangan berikutnya.

Alur komunikasi serta koordinasi selama periode magang di PT Cranium Royal Aditama disusun untuk memastikan setiap anggota tim dapat bekerja secara terarah, sinkron, dan efisien. Uraian mengenai mekanisme komunikasi, koordinasi, serta kolaborasi yang diterapkan dalam proyek ERP Cranium dijelaskan sebagai berikut:

1. Komunikasi
   (a) Komunikasi rutin dalam pengerjaan proyek ERP dilakukan menggunakan aplikasi *WhatsApp*, yang menjadi saluran utama untuk menyampaikan berbagai informasi penting, membahas tugas, serta memberikan pengingat terkait batas waktu penyelesaian pekerjaan.

(b) Aplikasi *Discord* dimanfaatkan sebagai sarana komunikasi tambahan, terutama saat aktivitas kerja dilakukan secara *Work From Home* (WFH). Platform ini digunakan untuk diskusi langsung, konsultasi teknis, maupun koordinasi dalam kelompok kecil.

(c) Pada hari kerja yang dilaksanakan secara WFO (*Work From Office*), komunikasi berlangsung secara tatap muka melalui obrolan informal, diskusi personal, atau rapat bersama di lingkungan kantor.

2. Koordinasi

(a) Pemantauan perkembangan pekerjaan dilakukan melalui tabel *roadmap* di *Google Sheets*, yang memuat rincian daftar tugas, status pengerjaan, serta estimasi maupun realisasi waktu penyelesaian tiap pekerjaan.

(b) Mahasiswa magang berpartisipasi secara langsung dalam memperbarui informasi pada *Google Sheets*, sehingga progres seluruh tugas dapat tercatat secara konsisten layaknya anggota tim lainnya.

(c) Dokumen kolaboratif berupa *Google Sheets* digunakan sebagai catatan harian hasil pekerjaan, memungkinkan setiap anggota tim memonitor perkembangan satu sama lain secara transparan.

3. Kolaborasi

(a) Kolaborasi teknis dalam pengembangan proyek ERP dilakukan melalui platform *GitHub*, yang berfungsi sebagai pusat penyimpanan dan pengelolaan seluruh kode sumber proyek.

(b) Setiap *developer*, termasuk peserta magang, bekerja pada *branch* yang dibuat secara terpisah untuk setiap tugas, sehingga proses integrasi menjadi lebih mudah serta membantu menjaga stabilitas keseluruhan proyek.

3.3 Tugas yang Dilakukan

Selama masa magang di PT Cranium Royal Aditama, penulis berkontribusi dalam proyek pengembangan sistem *Enterprise Resource Planning* (ERP) yang dimiliki oleh perusahaan. Tugas utama yang diberikan meliputi pengembangan berbagai fitur dalam sistem ERP, termasuk implementasi dan penyempurnaan modul *searching* yang berperan penting dalam memproses permintaan pencarian

data pada berbagai modul perusahaan. Seluruh pekerjaan disesuaikan dengan kebutuhan internal perusahaan maupun permintaan dari klien, sehingga fitur yang dikembangkan dapat mendukung operasional bisnis secara efisien dan tepat guna.

Pengembangan sistem ERP Cranium menggunakan pendekatan arsitektur modular monolitik. Pada pendekatan ini, sistem dibagi ke dalam beberapa modul fungsional yang saling terpisah, namun tetap berada dalam satu kesatuan aplikasi. Setiap modul memiliki peran dan tanggung jawab yang jelas, sehingga proses pengembangan, pengujian, serta pemeliharaan dapat dilakukan secara lebih terstruktur. Pendekatan ini memberikan keseimbangan antara kemudahan *deployment* khas sistem monolitik dan fleksibilitas modularitas yang biasa ditemukan pada arsitektur *microservices*.

Jenis tugas yang dihadapi selama periode magang memiliki cakupan yang luas, mulai dari penanganan *bugs*, pengembangan fitur baru pada sisi *front-end* maupun *back-end*, hingga melakukan penyesuaian terhadap fungsi yang telah diterapkan sebelumnya. Setiap tanggung jawab diberikan berdasarkan pembagian pekerjaan yang disusun oleh *supervisor* dan dikerjakan mengikuti standar teknis serta kebutuhan aktual proyek. Melalui rangkaian tugas tersebut, penulis memperoleh kesempatan untuk terlibat secara langsung dalam seluruh tahapan pengembangan perangkat lunak, sekaligus memperkuat pemahaman mengenai praktik profesional dalam pembangunan sistem serta menyesuaikan diri dengan kebutuhan operasional perusahaan.

Dalam menjalankan berbagai tugas pengembangan tersebut, digunakan beragam *tools*, *software*, dan *framework* yang berfungsi untuk meningkatkan produktivitas, akurasi, serta efektivitas proses kerja. Seluruh perangkat yang mendukung pengembangan ERP Cranium ini memiliki peran spesifik dalam alur pembangunan sistem. Uraian mengenai *tools*, *framework*, dan *software* yang dimanfaatkan selama pengerjaan proyek adalah sebagai berikut:

1. *IntelliJ IDEA*
   Dimanfaatkan sebagai lingkungan pengembangan utama untuk membangun komponen *back-end* berbasis *Java Spring Boot*. Integrasinya yang optimal dengan bahasa *Java* membantu mempercepat proses penulisan kode, penelusuran kesalahan (*debugging*), serta pembuatan *unit test* dan *contract test* secara lebih terarah.

2. *Java Spring Boot*
   Berfungsi sebagai *framework* inti dalam konstruksi *back-end* ERP. Teknologi

ini menyediakan pola pengembangan aplikasi *Java* yang terorganisir, bersifat *modular*, serta mendukung pembangunan layanan secara cepat dan berstandar tinggi.

3. *Visual Studio Code* (VSCode)
   Digunakan sebagai sarana utama untuk mengembangkan bagian *front-end* ERP yang dibangun menggunakan *Next.js* dan *TypeScript*. Selain untuk penulisan kode antarmuka, *VSCode* juga dimanfaatkan dalam penyusunan dan pengujian *unit test* pada sisi *front-end*.

4. *Next.js* dengan *TypeScript*
   Dimanfaatkan sebagai *framework front-end* utama dalam merancang dan membangun tampilan serta interaksi pengguna ERP, yang dijalankan sepenuhnya melalui *web application*.

5. *PostgreSQL*
   Berperan sebagai sistem manajemen basis data relasional yang digunakan untuk menampung, mengatur, dan menjaga konsistensi seluruh informasi yang dipakai dalam pengembangan ERP Cranium.

6. *DBeaver*
   Digunakan sebagai *GUI database client* untuk mempermudah proses pengelolaan, pemantauan, serta penelusuran struktur maupun isi *database PostgreSQL* selama tahap pembangunan sistem.

7. *Postman*
   Dimanfaatkan sebagai *tool* untuk melakukan pengujian, pengecekan respons, serta validasi fungsionalitas API yang dikembangkan pada sisi *back-end* sebelum diintegrasikan dengan komponen *front-end*.

8. *Git*
   Berfungsi sebagai *version control system* guna mencatat setiap perubahan kode, memfasilitasi kolaborasi antar anggota tim, serta menjaga riwayat pengembangan proyek agar tetap terstruktur.

9. *GitHub*
   Digunakan sebagai platform repositori dan kolaborasi yang memungkinkan penyimpanan kode, pengelolaan kontribusi, serta pelaksanaan *code review* secara terdistribusi untuk seluruh komponen ERP.

10. *Apache JMeter*

    Digunakan sebagai *tool* pendukung untuk melakukan evaluasi performa implementasi modul *searching* dalam membandingkan waktu respons dan konsistensi pencarian antara metode berbasis SQL dan Elasticsearch.

3.4 Uraian Pelaksanaan Magang

3.4.1 Pelaksanaan Kerja Magang

Selama mengikuti program magang selama 20 minggu, penulis terlibat dalam berbagai tugas yang mencakup banyak aspek pengembangan ERP Cranium. Seluruh kegiatan tersebut dijalankan secara bertahap dan berkesinambungan, sekaligus menunjukkan kontribusi nyata penulis dalam proses pembangunan sistem ERP Cranium. Uraian ringkas mengenai pekerjaan yang berhasil diselesaikan sepanjang periode magang disajikan pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1	*Training* dan persiapan teknis yang difokuskan pada pemahaman modul *searching* dalam sistem ERP. Kegiatan meliputi peninjauan alur kerja fitur pencarian, arsitektur modul *searching*, serta mekanisme integrasi dengan modul lain sebagai bekal awal sebelum memasuki tahap pengembangan.
2	Inisialisasi modul *searching* sebagai tahap awal pengembangan serta menyiapkan struktur dasar agar dapat terintegrasi dengan modul ERP lainnya. Serta mengembangkan komponen pendukung seperti DTO, *event*, *mapper*, dan *controller*.
3	Melanjutkan pengembangan dan penyempurnaan *controller* pada modul *searching* serta membuat *entity* yang disesuaikan dengan struktur data yang dibutuhkan. Mengembangkan, menyesuaikan, serta melakukan pengujian *event listener*.
	Lanjut di halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu Ke -	Pekerjaan yang dilakukan
4	Pengembangan *repository* pada modul *searching* sebagai lapisan akses data serta penyesuaian *query* dan struktur data agar proses pengambilan dan penyimpanan data berjalan optimal. Selain itu, dibuat *annotation* untuk mendukung konfigurasi sistem.
5	Lanjutan pengembangan *service* pada modul *searching* serta memastikan integrasinya dengan komponen lain berjalan dengan baik. Selain itu, dilakukan inisialisasi dan pengembangan *contract testing*, termasuk penyesuaian berdasarkan hasil pengujian, hingga tahap finalisasi.
6	Finalisasi *back-end* modul *searching* disertai pengecekan ulang untuk memastikan stabilitas, performa, dan kesiapan seluruh fitur yang telah dikembangkan. Selain itu, dilakukan integrasi modul *searching* dengan *front-end* hingga tahap finalisasi.
7	Pembuatan dan pengembangan *testing* pada *front-end* untuk memastikan fitur pencarian berjalan sesuai dengan kebutuhan dan berbagai skenario penggunaan. Selain itu, dilakukan penyempurnaan pengujian, perbaikan *bug*, serta finalisasi dan pengecekan ulang seluruh fitur.
8	Preparasi dan pengajuan *pull request* untuk *back-end* dan *front-end*, termasuk pengecekan kode, fungsionalitas, tampilan, serta penyesuaian dengan standar pengembangan yang berlaku.
9	Tindak lanjut terhadap hasil *review* dari *senior developer* dengan melakukan revisi pada *front-end* sesuai *feedback* yang diberikan. Revisi dilakukan secara bertahap untuk memperbaiki *bug* dan menyesuaikan kebutuhan sistem hingga seluruh perubahan dinyatakan sesuai dengan hasil *review*.
	Lanjut di halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

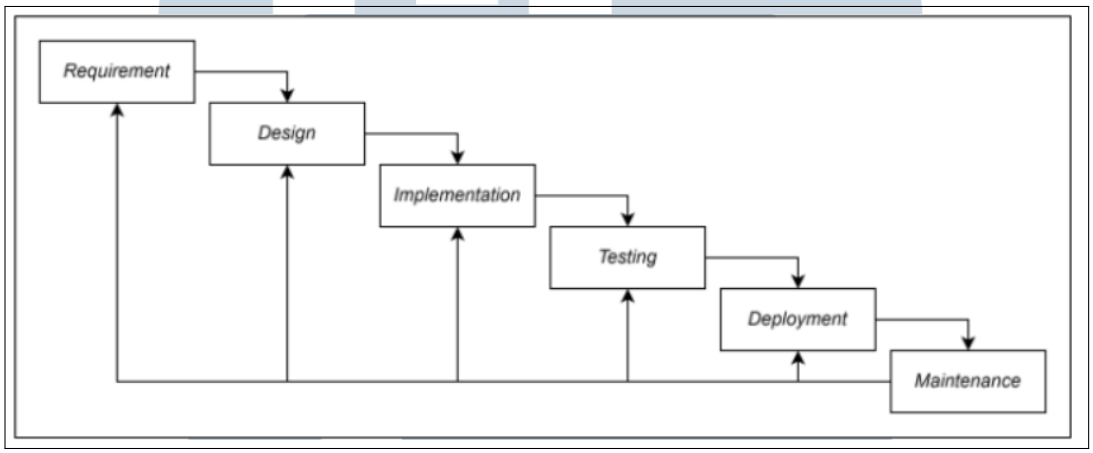
Minggu Ke -	Pekerjaan yang dilakukan
10	Revisi pada *back-end* untuk memperbaiki *bug* yang terdeteksi serta menyesuaikan implementasi berdasarkan masukan hasil review sebelumnya. Revisi dilanjutkan dengan penyempurnaan logika, alur data, dan performa sistem hingga seluruh modul terintegrasi dengan baik dan back end siap untuk tahap deployment selanjutnya.
11	Push revisi *back-end* ke *repository* serta menunggu proses *review* dari *senior developer*. Selain itu, dimulai pengerjaan *task* baru berupa pengembangan fitur *document number* pada sistem ERP, diawali dengan inisialisasi struktur dan konfigurasi dasar hingga tahap awal implementasi logika dan integrasi modul.
12	Pengerjaan fitur *document number* dilanjutkan dengan fokus pada fungsionalitas dan validasi data, kemudian dilakukan finalisasi dan proses build. Selanjutnya, diajukan pull request untuk mendapatkan *review* dari *senior developer* sambil mempersiapkan kemungkinan revisi berdasarkan masukan yang diberikan.
13	Revisi fitur document number berdasarkan masukan *senior developer* hingga fungsionalitas berjalan stabil, kemudian hasil revisi di-*push* ke *repository* dan menunggu persetujuan akhir. Selain itu, mulai dikerjakan *task* baru berupa pengembangan *master notification* untuk modul ERP.
14	Pengerjaan awal *master notification* dengan inisialisasi struktur dan konfigurasi *service*, kemudian dilanjutkan dengan integrasi fitur *notification* ke modul selling disertai pengujian dan penyempurnaan. Pada tahap akhir, integrasi *notification* diperluas ke modul *purchasing* untuk memastikan notifikasi lintas modul berjalan sesuai kebutuhan bisnis.
	Lanjut di halaman berikutnya

Tabel 3.1 – lanjutan dari halaman sebelumnya

Minggu Ke -	Pekerjaan yang dilakukan
15	Lanjutan integrasi fitur *notification* ke modul *purchasing* hingga stabil melalui pengujian fungsional. Selain itu, integrasi *notification* diperluas ke modul *finance* dengan fokus pada akurasi, kelengkapan informasi, serta pengujian menyeluruh untuk memastikan notifikasi berjalan dengan baik secara *end-to-end*.
16	Persiapan dan pengajuan *pull request* untuk task *notification*, termasuk pengecekan perubahan kode. Selanjutnya, dilakukan proses *review* dan revisi berdasarkan masukan *senior developer* hingga seluruh perbaikan diterapkan dan sistem siap untuk tahap finalisasi.
17	Inisialisasi task baru berupa penambahan *field image* pada modul *user* dengan menyiapkan struktur dan rencana implementasi. Pengerjaan dilakukan pada sisi *back-end* untuk memastikan proses penyimpanan, pengolahan, dan validasi data gambar berjalan dengan baik,
18	Pengerjaan pada sisi *front-end* untuk task penambahan *field image* pada modul *user*, termasuk implementasi form input serta tampilan *preview image*. Pengembangan difokuskan untuk memastikan kemudahan penggunaan dan kesesuaian tampilan dengan kebutuhan sistem.
19	Finalisasi task penambahan *field image* pada modul *user* serta pembuatan *pull request* untuk seluruh perubahan yang telah diimplementasikan.
20	Pengerjaan revisi untuk task penambahan *field image* pada modul *user* berdasarkan masukan dari *senior developer*. Revisi difokuskan pada penyempurnaan fungsionalitas dan penyesuaian implementasi agar sesuai dengan standar sistem, serta memastikan seluruh perubahan berjalan stabil sebelum dinyatakan selesai.

3.4.2 Konsep *Sofware Development Life Cycle* ERP Cranium

Software Development Life Cycle (SDLC) digunakan sebagai acuan dalam pelaksanaan pengembangan sistem ERP Cranium untuk menggambarkan tahapan kerja yang dilakukan secara berurutan, mulai dari perencanaan hingga tahap pemeliharaan. Penerapan SDLC membantu memastikan bahwa setiap proses pengembangan dilakukan secara terarah dan sesuai dengan kebutuhan sistem serta tujuan bisnis perusahaan. Ilustrasi alur SDLC pada pengembangan ERP Cranium ditunjukkan pada Gambar 3.1.



Gambar 3.1. Alur SDLC ERP Cranium

Dalam pengembangan perangkat lunak ERP Cranium, pendekatan SDLC yang diterapkan adalah *Waterfall Model*. Model ini menerapkan alur kerja yang sistematis dengan tahapan yang disusun secara berurutan. Setiap tahap pengembangan harus diselesaikan dan dievaluasi terlebih dahulu sebelum proses dilanjutkan ke tahap berikutnya. Dengan alur yang bersifat satu arah, model ini menekankan pentingnya dokumentasi dan persetujuan pada setiap fase guna menjaga konsistensi dan kualitas pengembangan sistem.

Tahapan-tahapan dalam *Waterfall Model* yang diterapkan pada proyek ERP Cranium dapat dijelaskan sebagai berikut:

- *Requirement:* Tahap ini merupakan fase awal yang berfokus pada identifikasi dan analisis kebutuhan sistem, baik dari sisi teknis maupun kebutuhan bisnis. Aktivitas yang dilakukan meliputi pengumpulan informasi terkait fungsionalitas sistem, penyusunan spesifikasi kebutuhan perangkat lunak, serta pemahaman terhadap alur proses bisnis yang akan diimplementasikan dalam sistem ERP.

- *Design:* Setelah kebutuhan sistem ditetapkan, tahap perancangan dilakukan untuk merumuskan struktur dan arsitektur sistem secara menyeluruh. Proses ini mencakup penyusunan desain modul, relasi antar komponen, perancangan basis data, serta rancangan antarmuka pengguna (*user interface*). Hasil dari tahap ini berupa dokumen desain yang menjadi acuan utama dalam proses pengembangan sistem.

- *Implementation:* Tahap implementasi berfokus pada penerjemahan desain sistem ke dalam bentuk kode program. Proses pengembangan dilakukan secara bertahap dan modular, dimulai dari implementasi fungsi-fungsi dasar seperti operasi CRUD (*Create, Read, Update, Delete*) pada entitas sistem. Selain itu, pada tahap ini juga diterapkan logika bisnis serta dilakukan integrasi awal antar modul yang dikembangkan.

Testing:

Tahap pengujian dilaksanakan setelah seluruh proses pengkodean sistem selesai dilakukan. Pada fase ini, sistem ERP Cranium dievaluasi untuk memastikan setiap fitur berfungsi dengan benar sesuai dengan kebutuhan yang telah ditetapkan. Proses pengujian mencakup pemeriksaan alur kerja aplikasi, pendeteksian kesalahan, serta validasi hasil keluaran, baik melalui pengujian manual maupun penerapan *unit test*, guna menjamin stabilitas dan kualitas perangkat lunak.

Deployment:

Setelah sistem dinyatakan memenuhi kriteria pada tahap pengujian, proses selanjutnya adalah melakukan *deployment* ke lingkungan produksi. Pada tahap ini, aplikasi ERP Cranium dipasang dan dikonfigurasi pada *server* yang telah disiapkan, termasuk penyesuaian lingkungan operasional, agar sistem dapat berjalan secara optimal dan siap digunakan oleh pengguna.

Maintenance:

Setelah aplikasi mulai dioperasikan, dilakukan tahap pemeliharaan sebagai upaya menjaga keberlanjutan sistem. Kegiatan pada fase ini meliputi penanganan *bug* yang muncul setelah *deployment*, peningkatan performa aplikasi, serta pengembangan fitur tambahan guna menyesuaikan sistem dengan kebutuhan pengguna yang terus berkembang.

3.4.3 Ruang Lingkup dan Konsep Pengembangan

Ruang lingkup pengembangan yang dilaksanakan selama program magang pada proyek ERP Cranium difokuskan pada pengembangan modul *searching* sebagai bagian dari peningkatan fungsionalitas sistem. Pengembangan modul ini bertujuan untuk mempermudah pengguna dalam melakukan pencarian data secara cepat, akurat, dan sesuai dengan kebutuhan operasional perusahaan. Modul *searching* dirancang agar mampu menangani berbagai parameter pencarian serta menyesuaikan hasil pencarian dengan konteks data yang digunakan pada masing-masing modul ERP.

Dalam proses pengembangannya, dilakukan penyesuaian logika pencarian pada sisi *back-end* untuk memastikan data yang ditampilkan telah melalui proses validasi dan pemfilteran yang sesuai dengan aturan bisnis. Selain itu, pengembangan juga mencakup integrasi modul *searching* dengan antarmuka pengguna, sehingga pengguna dapat melakukan pencarian melalui tampilan yang mudah digunakan. Penyesuaian struktur data antara *front-end* dan *back-end* turut dilakukan untuk menjamin konsistensi format data serta kelancaran proses pertukaran informasi.

Dalam pengembangan sistem ERP Cranium, digunakan beberapa konsep dasar yang menjadi landasan utama dalam perancangan struktur sistem serta penyusunan logika aplikasi. Konsep-konsep ini diterapkan secara konsisten untuk memastikan pengembangan sistem berjalan terarah, terstruktur, dan sesuai dengan kebutuhan operasional yang telah ditetapkan. Berikut adalah uraian konsep yang umum digunakan dalam pengembangan ERP Cranium :

1. *Data Transfer Object, Entity, & Mapper*

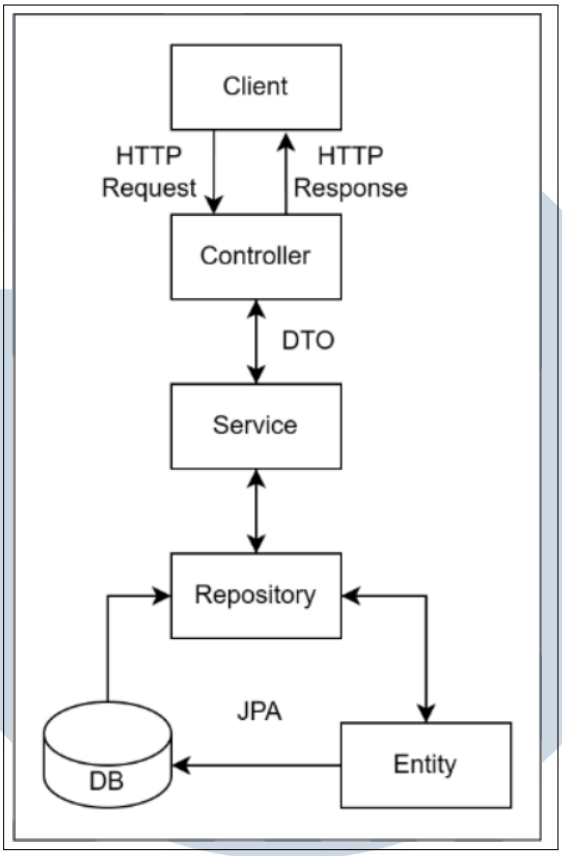
   Dalam pengembangan sistem ERP Cranium, diterapkan konsep pemisahan data melalui penggunaan *Data Transfer Object* (DTO), *entity*, dan *mapper* sebagai bagian dari perancangan arsitektur sistem. DTO berperan sebagai objek sederhana yang digunakan untuk mengirimkan data antar lapisan aplikasi tanpa memuat logika bisnis, sehingga proses komunikasi antara *client* dan *server* dapat berlangsung secara lebih efisien. Dengan hanya membawa data yang diperlukan dan menyembunyikan detail internal seperti struktur tabel pada basis data, penggunaan DTO membantu menjaga keamanan informasi sekaligus menyederhanakan alur pertukaran data antar komponen sistem.

Entity merupakan objek berbasis Java yang merepresentasikan struktur tabel pada basis data relasional dan berfungsi sebagai komponen persistensi dalam sistem. Objek ini dilengkapi dengan anotasi dari *Java Persistence API* (JPA) atau *Jakarta Persistence* yang digunakan untuk mendefinisikan pemetaan atribut, nama kolom, serta hubungan antar entitas di dalam database. Dalam konteks ORM (*Object-Relational Mapping*), *entity* memiliki peran penting karena memungkinkan proses pengelolaan data dilakukan melalui objek Java, sehingga pengembang tidak perlu berinteraksi langsung dengan *query SQL* secara manual.

Sebagai penghubung antara DTO dan *entity*, digunakan komponen *mapper* yang berperan dalam melakukan transformasi data secara dua arah. *Mapper* mengonversi DTO menjadi *entity* ketika data akan disimpan ke dalam basis data, serta mengubah *entity* kembali menjadi DTO untuk kebutuhan pengiriman data ke sisi *front-end*. Mekanisme ini membantu menjaga konsistensi struktur data, meningkatkan keamanan informasi, serta memastikan alur pemrosesan data dalam aplikasi ERP Cranium berjalan dengan baik.

2. *Controller, Service, & Repository*

   Pengembangan sisi *back-end* pada sistem ERP Cranium menerapkan pola *layered architecture* yang membagi sistem ke dalam beberapa lapisan utama, yaitu *controller*, *service*, dan *repository*. Setiap lapisan memiliki peran dan tanggung jawab yang berbeda, di mana *controller* berfungsi sebagai pengelola permintaan dari *client*, *service* menangani logika bisnis termasuk pengolahan dan pemfilteran data pada modul *searching*, serta *repository* bertanggung jawab terhadap interaksi langsung dengan basis data. Penerapan arsitektur berlapis ini bertujuan untuk menjaga keterpisahan fungsi antar komponen, meningkatkan kemudahan pemeliharaan sistem, serta mendukung pengembangan fitur pencarian yang terstruktur dan mudah dikembangkan lebih lanjut.



Gambar 3.2. Hubungan *Controller-Service-Repository* dalam arsitektur ERP Cranium

Berdasarkan ilustrasi pada Gambar 3.2, lapisan *controller* berfungsi sebagai komponen awal yang menerima permintaan *HTTP* dari sisi *client*. Pada lapisan ini, data permintaan tidak diproses secara langsung, melainkan diteruskan ke lapisan *service* untuk menjalankan logika aplikasi. Walaupun tidak menangani logika bisnis, *controller* tetap dapat melakukan validasi dasar terhadap data masukan guna memastikan kesesuaian format dan struktur permintaan sebelum diproses lebih lanjut. Seluruh pertukaran data pada lapisan ini disampaikan menggunakan objek DTO untuk menjaga keteraturan dan keamanan alur data.

Lapisan *service* berperan sebagai inti pengolahan logika bisnis dalam pengembangan *back-end* ERP Cranium. Seluruh proses utama, seperti validasi lanjutan terhadap data, pengelolaan alur bisnis, serta konversi antara DTO dan *entity*, dijalankan pada lapisan ini sebelum data diteruskan ke *repository*. Selain itu, *service* menjadi pusat koordinasi pemanggilan *mapper* dan komponen pendukung lainnya untuk memastikan alur pemrosesan data

berjalan sesuai dengan aturan sistem. Oleh karena itu, perancangan lapisan *service* harus dilakukan secara terstruktur karena memuat sebagian besar kebijakan dan aturan bisnis yang mendukung operasional ERP Cranium.

Lapisan *repository* berfungsi sebagai komponen yang menangani interaksi langsung dengan basis data dalam pengembangan ERP Cranium. Pada lapisan ini digunakan *JPA Repository* yang menyediakan berbagai operasi dasar, seperti penyimpanan data, pengambilan data berdasarkan identitas tertentu, penghapusan data, serta penelusuran seluruh data yang tersimpan. Selain fungsi standar tersebut, *repository* juga dapat dikembangkan dengan *custom query* atau pemanfaatan anotasi seperti @*Query* untuk memenuhi kebutuhan pengambilan data yang lebih kompleks dan spesifik. Dengan adanya lapisan ini, *service* dapat mengakses dan mengelola data tanpa harus menuliskan perintah *SQL* secara langsung, sehingga struktur kode menjadi lebih rapi, terjaga konsistensinya, dan mudah dipelihara.

Secara keseluruhan, alur pemrosesan data dalam sistem ERP Cranium dimulai dari permintaan yang dikirimkan oleh *client* ke lapisan *controller*. Selanjutnya, data tersebut diteruskan ke lapisan *service* dalam bentuk DTO untuk diproses sesuai dengan logika bisnis yang berlaku. Pada tahap ini, data dapat dikonversi menjadi *entity* sebelum dilakukan pengolahan lebih lanjut atau disimpan melalui lapisan *repository*. Data hasil pemrosesan yang akan dikirim kembali ke *client* mengikuti alur sebaliknya, sehingga tercipta struktur sistem yang terorganisir, modular, dan mudah dikembangkan.

3. *Unit Test & Contract Test*

   Untuk menjaga stabilitas serta kualitas perangkat lunak, pengembangan ERP Cranium menerapkan proses pengujian otomatis yang mencakup *unit test* dan *contract test*. *Unit test* difokuskan pada pengujian komponen terkecil dalam sistem, seperti fungsi atau metode pada suatu kelas, yang dijalankan secara terpisah dari komponen lainnya. Pengujian ini bertujuan memastikan bahwa setiap bagian internal sistem berfungsi sesuai dengan kebutuhan dan spesifikasi yang telah ditetapkan tanpa dipengaruhi oleh dependensi eksternal. Pada sisi *back-end*, pelaksanaan *unit test* dilakukan dengan memanfaatkan beberapa *framework* pendukung, antara lain *JUnit*, *Mockito*, serta *Spring Boot Test*, yang membantu dalam proses simulasi dependensi dan validasi logika aplikasi.

Selain *unit test*, pengembangan ERP Cranium juga menerapkan *contract test* untuk memastikan kesesuaian komunikasi antara komponen *front-end* dan *back-end*, khususnya pada lapisan *controller*. *Contract test* berfungsi untuk memverifikasi bahwa setiap *endpoint API* menghasilkan respons dengan struktur data, format, serta status yang sesuai dengan kontrak yang telah disepakati bersama oleh pengembang *front-end* dan *back-end*. Melalui pendekatan ini, potensi ketidaksesuaian integrasi dapat dideteksi lebih awal sebelum sistem digunakan secara menyeluruh. Pada sisi *back-end*, pengujian ini dilakukan dengan memanfaatkan *library* seperti *Spring Cloud Contract*, *MockMVC*, dan *JUnit* untuk memastikan bahwa *controller* mampu menangani permintaan dengan benar serta memenuhi spesifikasi kontrak yang telah ditetapkan.

Pada sisi *front-end*, proses pengujian difokuskan untuk memastikan bahwa setiap komponen *user interface* dapat di-*render* dengan baik, mampu merespons interaksi pengguna, serta menampilkan data sesuai dengan kondisi yang diharapkan. Pengujian ini mencakup verifikasi perilaku komponen terhadap berbagai skenario input maupun perubahan *state* aplikasi. Untuk mendukung proses tersebut, digunakan *library* seperti *Jest* dan *React Testing Library* dalam pelaksanaan *unit test* pada komponen berbasis *React* yang dikembangkan menggunakan *Next.js*. Dengan penerapan pengujian yang komprehensif pada sisi *back-end* dan *front-end*, pengembangan ERP Cranium dapat berjalan lebih andal, terukur, serta mendukung aspek keamanan dan kualitas sistem secara keseluruhan.

3.4.4 Pengembangan Modul *searching*

Modul *searching* dalam sistem ERP Cranium berfungsi sebagai komponen utama yang memungkinkan pengguna untuk menelusuri dan mengambil data secara cepat dan akurat dari berbagai sumber dalam sistem. Modul ini bertanggung jawab untuk menyediakan mekanisme pencarian yang efisien, fleksibel, dan dapat menyesuaikan dengan kebutuhan bisnis yang dinamis.

Modul *searching* diperlukan dalam pengembangan ERP Cranium untuk menangani kebutuhan pencarian data yang semakin kompleks seiring dengan bertambahnya volume, variasi, dan keterkaitan data antar modul. Dalam sistem ERP, proses pencarian tidak hanya terbatas pada pencarian berdasarkan satu kolom, tetapi sering melibatkan kombinasi berbagai parameter, seperti kata kunci,

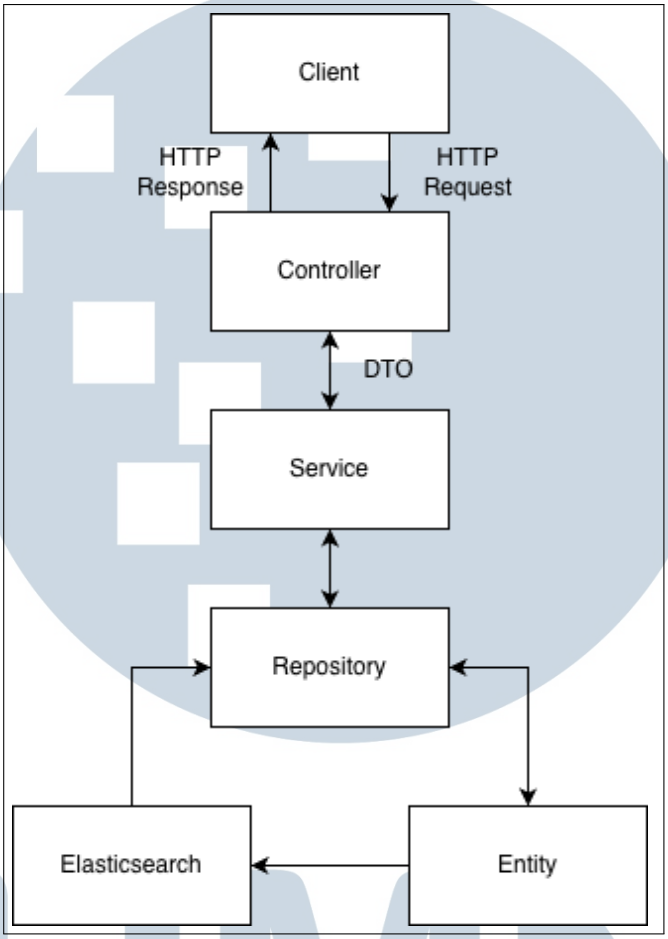
status data, rentang waktu, serta relasi dengan entitas lain. Apabila seluruh proses pencarian ini dilakukan langsung pada basis data relasional, maka beban pemrosesan akan meningkat dan berpotensi menurunkan performa sistem, terutama pada kondisi penggunaan secara bersamaan oleh banyak pengguna. Oleh karena itu, diperlukan sebuah layanan khusus yang berfokus pada proses pencarian agar sistem utama tetap responsif dan stabil.

Pemisahan fungsi pencarian ke dalam modul *searching* juga memberikan keuntungan dari sisi arsitektur sistem. Dengan adanya layanan terpisah, proses pencarian tidak lagi bercampur dengan logika transaksi utama, sehingga pengembangan dan pemeliharaan sistem menjadi lebih terstruktur. Modul *searching* dapat dikembangkan secara independen untuk menyesuaikan kebutuhan pencarian yang terus berkembang, tanpa harus mengubah struktur basis data atau logika bisnis inti pada modul lain. Pendekatan ini sejalan dengan prinsip modularitas dalam pengembangan perangkat lunak, di mana setiap layanan memiliki tanggung jawab yang jelas dan terdefinisi dengan baik.

Elasticsearch dipilih sebagai teknologi pendukung modul *searching* karena dirancang secara khusus untuk menangani kebutuhan pencarian data secara cepat dan efisien. Berbeda dengan basis data relasional yang dioptimalkan untuk transaksi, *Elasticsearch* menggunakan mekanisme *indexing* berbasis dokumen yang memungkinkan pencarian *full-text*, pemfilteran, dan pengurutan data dilakukan dalam waktu singkat meskipun pada kumpulan data yang besar. Selain itu, *Elasticsearch* mendukung skema data yang fleksibel, sehingga mudah menyesuaikan perubahan struktur data tanpa mengganggu proses pencarian yang sudah berjalan. Dengan memanfaatkan *Elasticsearch*, modul *searching* pada ERP Cranium mampu menyediakan hasil pencarian yang lebih relevan, konsisten, dan responsif, sekaligus menjaga performa sistem secara keseluruhan.

Pengembangan fitur pada modul *searching* melibatkan perancangan dan implementasi berbagai komponen pendukung yang saling terintegrasi, mulai dari pembuatan *Data Transfer Object* (DTO), *event*, *mapper*, *controller*, *entity*, *event listener*, *repository*, hingga *service*. Selain itu, proses pengembangan juga dilengkapi dengan tahap *testing* guna memastikan setiap fungsi pencarian bekerja sesuai dengan kebutuhan sistem dan dapat diintegrasikan dengan baik ke dalam modul *searching*.

A Alur Kerja Modul *Searching*



Gambar 3.3. Alur kerja Modul *Searching*

Alur kerja *modul searching* dimulai ketika *client* mengirimkan permintaan pencarian dalam bentuk *HTTP request* melalui antarmuka aplikasi. Permintaan ini berisi kata kunci atau parameter pencarian yang merepresentasikan kebutuhan informasi pengguna. Request tersebut kemudian diteruskan ke lapisan *controller* sebagai titik masuk utama layanan pencarian.

Pada lapisan *controller*, permintaan HTTP diterima dan dipetakan ke *endpoint* yang sesuai. *Controller* berperan untuk menangani komunikasi *API*, melakukan validasi awal terhadap parameter pencarian, serta membungkus data permintaan ke dalam bentuk *DTO*. *DTO* ini digunakan untuk memastikan bahwa data yang dikirim ke lapisan berikutnya memiliki struktur yang konsisten dan terkontrol. Selanjutnya, *controller* meneruskan *DTO* tersebut ke lapisan *service* untuk diproses lebih lanjut.

Lapisan *service* bertanggung jawab mengelola logika bisnis pada proses pencarian. Pada tahap ini, parameter pencarian yang diterima dalam bentuk DTO diolah sesuai kebutuhan, seperti penyesuaian format pencarian, penerapan aturan bisnis, maupun pengaturan mekanisme *pagination* dan *sorting*. *Service* kemudian berinteraksi dengan lapisan *repository* untuk mengeksekusi proses pencarian data.

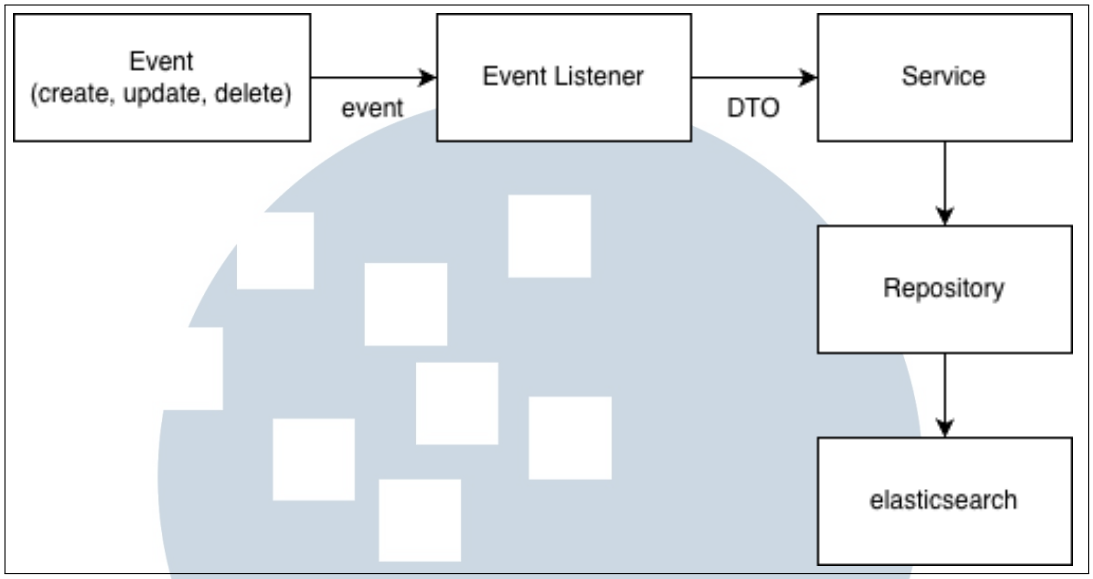
Lapisan *repository* berfungsi sebagai penghubung antara *service* dengan sumber data pencarian. *Repository* memanfaatkan *entity* sebagai representasi struktur dokumen yang tersimpan di dalam *Elasticsearch*. Pada tahap ini, *repository* menyusun dan mengeksekusi *query* pencarian ke *Elasticsearch* berdasarkan parameter yang telah diproses oleh *service*. *Elasticsearch* kemudian melakukan pencarian terhadap data yang telah diindeks dan mengembalikan hasil pencarian dalam bentuk kumpulan dokumen yang sesuai.

Hasil pencarian dari *Elasticsearch* dikonversi kembali ke dalam bentuk *entity*, kemudian dikembalikan oleh *repository* ke lapisan *service*. *Service* selanjutnya meneruskan hasil tersebut ke *controller* tanpa menambahkan logika bisnis tambahan, karena seluruh proses pencarian telah ditangani pada lapisan *repository*. *Controller* kemudian mengemas hasil pencarian ke dalam bentuk *HTTP response* yang terstruktur dan mengirimkannya kembali ke *client* untuk ditampilkan pada antarmuka aplikasi.

Secara keseluruhan, alur kerja ini menunjukkan bahwa modul *Searching* dirancang dengan pemisahan tanggung jawab yang jelas antara lapisan *controller*, *service*, *repository*, *entity*, dan *Elasticsearch*. Pendekatan ini memungkinkan proses pencarian berjalan secara efisien, terstruktur, dan mudah dikembangkan, sekaligus memastikan bahwa beban pencarian tidak langsung dibebankan pada basis data utama. Dengan memanfaatkan *Elasticsearch* sebagai mesin pencari, sistem mampu memberikan hasil pencarian yang cepat, relevan, dan skalabel sesuai kebutuhan ERP Cranium.

B Alur *Listener* dan Proses *Indexing* pada Modul *Searching*

Untuk memastikan data yang tersimpan pada *Elasticsearch* selalu selaras dengan data pada basis data utama, modul *searching* dilengkapi dengan mekanisme pembaruan indeks secara otomatis. Mekanisme ini berperan penting dalam menjaga konsistensi data pencarian ketika terjadi perubahan data pada sistem, sehingga hasil pencarian yang disajikan kepada pengguna tetap mencerminkan kondisi data terkini.



Gambar 3.4. Alur kerja Listener pada Modul *Searching*

Mekanisme diimplementasikan melalui *listener* yang bertugas mendeteksi setiap peristiwa perubahan data di modul lain, seperti proses penambahan (*create*), pembaruan (*update*), maupun penghapusan (*delete*). Setiap perubahan tersebut menghasilkan sebuah event yang merepresentasikan kondisi terbaru dari data yang mengalami perubahan. *Event* ini berfungsi sebagai pemicu awal agar modul *searching* dapat mengetahui bahwa terdapat data yang perlu diselaraskan dengan indeks pencarian.

Event yang dihasilkan kemudian diterima oleh *event listener* yang telah terdaftar pada modul *searching*. *Event listener* berperan sebagai penghubung antara modul sumber data dan modul *searching*, dengan tugas utama menangkap *event* yang masuk serta menyiapkan data yang diperlukan untuk proses lanjutan. Pada tahap ini, data dari event dikemas ke dalam bentuk *Data Transfer Object* (DTO) agar dapat diproses secara terstruktur dan konsisten oleh lapisan *service*, tanpa bergantung langsung pada struktur data modul asal.

Selanjutnya, DTO yang telah disiapkan diteruskan ke lapisan *service* untuk diproses sesuai dengan kebutuhan pembaruan indeks pencarian. Lapisan *service* mengelola logika bisnis yang berkaitan dengan proses *indexing*, termasuk penyesuaian struktur data agar sesuai dengan format dokumen *Elasticsearch*. Setelah proses tersebut selesai, *service* meneruskan data ke lapisan *repository* untuk disimpan atau diperbarui pada *Elasticsearch*. Dengan alur ini, setiap perubahan data pada sistem operasional dapat secara otomatis tercermin pada indeks pencarian, sehingga modul *searching* selalu menyajikan data yang relevan, konsisten, dan

selaras dengan kondisi terkini sistem ERP Cranium.

3.4.5 Potongan Kode

Berdasarkan alur kerja tersebut, dapat dipahami bahwa Modul *Searching* tidak berdiri sebagai satu proses tunggal, melainkan dibangun dari sejumlah komponen yang saling terintegrasi untuk mendukung pemisahan proses pencarian dari basis data utama. Setiap komponen memiliki peran spesifik dalam memastikan proses pencarian berjalan secara efisien, konsisten, dan mudah dikembangkan. Berikut adalah rincian komponen utama yang menyusun modul *Searching* beserta peran masing-masing dalam mendukung proses pencarian data pada sistem ERP Cranium.

A Metode *Get*

A.1 Lapisan *Controller*

```
1    @GetMapping(value = "/purchasing/orders", params = "input",
     headers = "X-Api-Version=1")
2    @IsSearchingPurchasingOrderRead
3    @ResponseStatus(HttpStatus.OK)
4    @LogExecutionTime
5    public Page<PurchasingOrder> getOrderDocumentNoAndSupplierName
     (
6            Pageable pageable,
7            PurchasingOrderRequestDto purchasingOrderRequestDto) {
8        return purchasingOrderService.findAllOrderAndSupplier(
     pageable, purchasingOrderRequestDto);
9    }
```

Kode 3.1: Implementasi Metode Controller untuk Menangani Permintaan Pencarian Data Purchasing Order

Pada lapisan *controller*, metode *getOrderDocumentNoAndSupplierName* berfungsi sebagai titik masuk utama dalam menangani permintaan pencarian data dari pengguna. Metode ini dipetakan menggunakan anotasi *@GetMapping* dengan parameter tertentu, sehingga hanya akan dieksekusi ketika permintaan *HTTP GET* memenuhi kriteria yang telah ditetapkan oleh sistem. Permintaan yang diterima kemudian dipetakan ke dalam objek *Pageable* untuk pengaturan *pagination* serta *PurchasingOrderRequestDto* yang berfungsi sebagai wadah parameter pencarian.

Penggunaan *request DTO* memungkinkan berbagai kriteria pencarian, seperti kata kunci atau filter tertentu, dikelompokkan dalam satu struktur data yang terorganisir dan mudah dikelola.

Setelah proses penerimaan dan validasi awal selesai, lapisan *controller* tidak menjalankan logika bisnis pencarian secara langsung. Seluruh data permintaan dalam bentuk *request DTO* diteruskan ke lapisan *service* melalui pemanggilan metode *findAllOrderAndSupplier*. Pendekatan ini menegaskan pemisahan tanggung jawab antar lapisan, di mana *controller* berfokus pada pengelolaan alur permintaan dan pengamanan akses, sementara lapisan *service* bertanggung jawab penuh atas pengolahan logika pencarian dan pengambilan data. Dengan demikian, struktur sistem tetap modular, terstruktur, dan mudah dikembangkan.

A.2 Lapisan *Service*

```
1    public Page<PurchasingOrder> findAllOrderAndSupplier(Pageable
   pageable, PurchasingOrderRequestDto dto) {
2        return purchasingOrderRepository.
   findAllOrderDocumentNoAndSupplierName(pageable, dto);
3    }
```

Kode 3.2: Implementasi *Service Layer* sebagai Penghubung *Controller* dan *Repository* pada Proses Pencarian

Pada lapisan *service*, metode *findAllOrderAndSupplier* berperan sebagai penghubung antara *controller* dan *repository* dalam proses pengambilan data *purchasing order* berdasarkan kriteria pencarian. Service ini menerima dua parameter utama, yaitu *Pageable* untuk mengatur mekanisme *pagination* serta *PurchasingOrderRequestDto* yang berisi parameter pencarian dari sisi klien. Dengan pemisahan ini, logika bisnis tetap terjaga pada lapisan *service* tanpa tercampur dengan detail teknis akses data.

Request DTO yang diterima oleh *service* diteruskan secara langsung ke lapisan *repository* tanpa perubahan, karena pada proses ini *service* berfungsi sebagai penghubung alur data antar lapisan. DTO tersebut memuat nilai pencarian seperti *order document number* dan *supplier name* yang telah divalidasi dan dipetakan sejak lapisan *controller*. Pendekatan ini memastikan bahwa *service* hanya memproses data yang telah terstruktur dengan baik dan sesuai dengan kebutuhan pencarian.

Metode *findAllOrderAndSupplier* kemudian memanggil fungsi

findAllOrderDocumentNoAndSupplierName pada *repository*. Pemanggilan ini menandakan bahwa seluruh logika *query* dan optimasi pencarian, termasuk integrasi dengan modul *searching* berbasis *Elasticsearch*, ditempatkan sepenuhnya pada lapisan *repository*. Dengan demikian, *service* tidak bergantung pada detail implementasi penyimpanan data, baik itu *database* relasional maupun mesin pencari.

A.3 Lapisan *Repository*

Pada lapisan *repository*, implementasi pencarian data *purchasing order* pada *modul searching* dibagi menjadi tiga komponen utama, yaitu *PurchasingOrderRepository*, *PurchasingOrderCustomRepository*, dan *PurchasingOrderCustomRepositoryImpl*. Pemisahan ini bertujuan untuk menjaga prinsip *separation of concerns*, di mana operasi bawaan *Spring Data Elasticsearch* dipisahkan dari logika pencarian kustom yang lebih kompleks. *PurchasingOrderRepository* berperan sebagai *repository* utama yang mewarisi *ElasticsearchRepository*, sehingga menyediakan fungsi dasar seperti pencarian berdasarkan *id* dan relasi sederhana dengan entitas lain. Potongan kode berikut menunjukkan implementasi *PurchasingOrderRepository*.

```
@Repository("elasticPurchasingOrderRepository")
public interface PurchasingOrderRepository
extends ElasticsearchRepository<PurchasingOrder, Long>,
PurchasingOrderCustomRepository {

```
Optional<PurchasingOrder> findById(Long id);
List<PurchasingOrder> findBySupplierId(Long supplierId);

@Query(value = "{\"match_all\": {}}")
Page<PurchasingOrder> getAllOrder(Pageable pageable);
```

}
```

Kode 3.3: Implementasi Repository Utama pada Elasticsearch

Selain menyediakan metode bawaan, *PurchasingOrderRepository* juga meng-*extend* *PurchasingOrderCustomRepository*. Pendekatan ini memungkinkan lapisan *service* untuk tetap berinteraksi dengan satu *repository* tanpa perlu mengetahui detail implementasi *query* yang digunakan.

PurchasingOrderCustomRepository berfungsi sebagai kontrak yang mendefinisikan kebutuhan pencarian khusus yang tidak dapat ditangani oleh metode turunan *Spring Data* secara langsung, seperti pencarian berdasarkan *order document number* atau kombinasi antara *document number* dan *supplier name*. Kontrak ini ditunjukkan pada potongan kode berikut.

```
public interface PurchasingOrderCustomRepository {
Page<PurchasingOrder> findAllOrderDocumentNo(
Pageable pageable, String documentNo);

```
Page<PurchasingOrder> findAllOrderDocumentNoAndSupplierName(
        Pageable pageable, PurchasingOrderRequestDto dto);
```

}
```

Kode 3.4: Implementasi CustomRepository pada Elasticsearch

Implementasi konkret dari kontrak *PurchasingOrderCustomRepository* direalisasikan pada kelas *PurchasingOrderCustomRepositoryImpl*. Kelas ini bertanggung jawab untuk menangani kebutuhan pencarian lanjutan yang tidak dapat dicapai melalui mekanisme *query* turunan bawaan *Spring Data*. Oleh karena itu, digunakan *ElasticsearchOperations* sebagai abstraksi tingkat lanjut yang memungkinkan penyusunan *query* secara manual dan lebih fleksibel terhadap struktur indeks *Elasticsearch*. Potongan kode berikut menunjukkan implementasi metode pencarian kustom tersebut.

```
@Repository("purchasingOrderCustomRepositoryImpl")
public class PurchasingOrderCustomRepositoryImpl
implements PurchasingOrderCustomRepository {

```
@Autowired
private ElasticsearchOperations searchingElasticsearchOperations;

@Override
public Page<PurchasingOrder> findAllOrderDocumentNoAndSupplierName
    (
        Pageable pageable, PurchasingOrderRequestDto dto) {

    String wildcardInput =
        "*" + EscapeQueryUtil.escapedQuery(dto.getInput()) + "*";
```
```

```
15
16      NativeQuery query = NativeQuery.builder()
17          .withQuery(q -> q.bool(b -> b
18              .should(s -> s.wildcard(w -> w.field("documentNo")
19                      .value(wildcardInput).caseInsensitive(true)))
20              .should(s -> s.wildcard(w -> w.field("supplierName")
21                      .value(wildcardInput).caseInsensitive(true)))
22              .minimumShouldMatch("1")
23          ))
24          .withPageable(pageable)
25          .build();
26
27      SearchHits<PurchasingOrder> response =
28          searchingElasticsearchOperations.search(query,
    PurchasingOrder.class);
29
30      return (Page<PurchasingOrder>)
31          SearchHitSupport.unwrapSearchHits(
32              searchPageFor(response, pageable));
33  }
34  ```
35
36  }
```

Kode 3.5: Implementasi Query Elasticsearch pada CustomRepositoryImpl

Pada tahap awal proses, metode ini menerima parameter pencarian dalam bentuk *PurchasingOrderRequestDto*. Nilai input terlebih dahulu divalidasi untuk memastikan bahwa pencarian hanya dilakukan ketika kata kunci tersedia. Selanjutnya, input tersebut diproses menggunakan *EscapeQueryUtil* dan dikonversi ke dalam pola *wildcard search*. Pendekatan ini memungkinkan sistem menemukan data yang mengandung kata kunci tertentu secara parsial, tanpa harus mencocokkan nilai secara eksak.

Query pencarian kemudian dibangun menggunakan *NativeQuery* dengan pendekatan *boolean query*. Pada struktur ini, beberapa kondisi *should* diterapkan untuk memungkinkan pencarian dilakukan pada lebih dari satu *field*, yaitu *documentNo* dan *supplierName*. Dengan pengaturan *minimumShouldMatch*, sistem memastikan bahwa minimal satu kriteria pencarian harus terpenuhi agar suatu dokumen dapat dikembalikan sebagai hasil. Selain itu, pencarian bersifat *case-insensitive*, sehingga perbedaan huruf besar dan kecil tidak memengaruhi hasil pencarian.

Hasil pencarian yang diperoleh dari Elasticsearch dikembalikan dalam bentuk *SearchHits*, yang selanjutnya dikonversi menjadi objek *Page* menggunakan *SearchHitSupport*. Proses ini menjaga konsistensi mekanisme *pagination* dengan lapisan *service* dan *controller*. Dengan demikian, seluruh kompleksitas logika pencarian ditempatkan secara terpusat pada lapisan repository kustom, sementara lapisan di atasnya tetap sederhana dan fokus pada pengelolaan alur bisnis.

Secara keseluruhan, alur kerja pencarian dimulai dari pemanggilan metode pada *PurchasingOrderRepository* oleh lapisan *service*. Operasi bawaan Spring Data akan langsung dieksekusi, sedangkan metode pencarian kustom secara otomatis diteruskan ke *PurchasingOrderCustomRepositoryImpl*. Hasil pencarian yang diperoleh dari *Elasticsearch* dikembalikan dalam bentuk objek *Page* ke lapisan *service*, kemudian diteruskan ke *controller* untuk dikemas sebagai *respons API* dan dikirimkan kembali ke *client*. Dengan desain ini, logika pencarian kompleks terpusat pada lapisan *repository* kustom, sementara lapisan di atasnya tetap sederhana. Pendekatan tersebut tidak hanya meningkatkan modularitas dan keterbacaan kode, tetapi juga memudahkan penambahan kebutuhan pencarian baru tanpa mengubah struktur *repository* utama maupun lapisan service, sehingga mendukung skalabilitas modul *searching* di masa mendatang.

B Metode *Listener (Create, Update, Delete)*

B.1 *Event Publisher*

```
Optional<HttpHeaderDto> httpHeaderDto = httpHeaderService.
    getHttpHeader();
if (!httpHeaderDto.isEmpty()) {
     MasterSupplierCreateEvent supplierCreateEvent =
    MasterSupplierCreateEvent.builder()
          .xUserId(UserAuthInfo.getUserId())
          .xRequestId(httpHeaderDto.get().getXRequestId())
          .xAuthorization(httpHeaderDto.get().getXAuthorization())
          .xAcceptLanguage(httpHeaderDto.get().getXAcceptLanguage())
          .xApiVersion(httpHeaderDto.get().getXApiVersion())
          .id(supplier.getId())
          .supplierName(supplier.getSupplierName())
          .supplierCode(supplier.getSupplierCode())
          .build();
     supplierPublisher.publishEvent(supplierCreateEvent);
```

```
14 }
```

Kode 3.6: Contoh Implementasi Pengiriman Event Perubahan Data Antar Modul

Potongan kode tersebut menggambarkan mekanisme pengiriman event yang digunakan untuk meneruskan informasi perubahan data dari suatu modul ke modul *searching*. Proses diawali dengan pengambilan informasi *header HTTP* melalui *httpHeaderService.getHttpHeader()*, yang disimpan dalam bentuk *HttpHeaderDto*. Apabila data *header* tersedia, sistem kemudian membentuk sebuah objek event bertipe MasterSupplierCreateEvent menggunakan pola builder. *Event* ini memuat dua jenis informasi utama, yaitu metadata permintaan dan data inti entitas yang mengalami perubahan. *Metadata* seperti *xUserId*, *xRequestId*, *xAuthorization*, *xAcceptLanguage*, dan *xApiVersion* disertakan untuk menjaga konteks komunikasi antar layanan serta mendukung proses pelacakan dan audit.

Selain *metadata*, event juga memuat data inti yang mengalami perubahan, yaitu id, *supplierName*, dan *supplierCode*. Informasi ini merepresentasikan entitas supplier yang baru dibuat dan menjadi dasar bagi modul *searching* untuk melakukan proses lanjutan, seperti pembentukan atau pembaruan indeks pencarian di *Elasticsearch*. Dengan hanya mengirimkan data yang relevan, *event* tetap bersifat ringan dan efisien untuk diproses.

Setelah *event* berhasil dibangun, *event* tersebut dipublikasikan melalui *supplierPublisher.publishEvent(supplierCreateEvent)*. Tahap ini menandai bahwa event perubahan data telah dipublikasikan ke sistem *event listener*. Dengan mekanisme ini, sinkronisasi data antar modul dapat dilakukan secara otomatis dan reaktif tanpa ketergantungan langsung antar layanan, sehingga mendukung arsitektur sistem yang modular dan terintegrasi.

B.2 Lapisan *Event Listener*

```
@Component  nasbi24 *
@Slf4j
public class SearchingMasterCreateSupplierEventListener extends BaseEventListener {

    @Autowired
    private MasterSupplierService masterSupplierService;

    @Autowired
    private UserJwtUtil userJwtUtil;

    @EventListener({MasterSupplierCreateEvent.class})  nasbi24 *
    public void handler(BaseEvent baseEvent) {
        MasterSupplierCreateEvent event = (MasterSupplierCreateEvent) baseEvent;
        setRequestId(event);
        UserAuthInfo.setEventUserAuthInfo(event, userJwtUtil);
        log.info("SearchingMasterDeleteSupplierEventListener.handler: {}", JsonUtil.dtoToJson(event));

        MasterSupplierCreateDto masterSupplierCreateDto = MasterSupplierCreateDto.builder()
            .id(event.getId())
            .supplierName(event.getSupplierName())
            .supplierCode(event.getSupplierCode()).createdBy(event.getCreatedBy())
            .build();

        HttpHeaderDto httpHeaderDto = HttpHeaderDto.builder()
                .XUserId(UserAuthInfo.getUserId().toString())
                .XApiVersion(event.getXApiVersion())
                .XAcceptLanguage(event.getXAcceptLanguage())
                .XAuthorization(event.getXAuthorization())
                .XRequestId(event.getXRequestId())
                .build();
        try {
            MasterSupplierDto masterSupplier = ScopedValue.callWhere(CustomContextHolder.HOLDER,
                    httpHeaderDto, () -> masterSupplierService.createMasterSupplier(masterSupplierCreateDto));
        } catch (Exception ex) {
            log.error("SearchingMasterCreateSupplierEventListener.handler: " + ex.getMessage());
        }
    }
}
```

Gambar 3.5. Implementasi *SearchingMasterCreateSupplierEventListener*

Setelah event perubahan data dipublikasikan oleh modul sumber, *event* tersebut akan ditangkap oleh *event listener* yang terdaftar di modul *searching*. Pada implementasi ini, class *SearchingMasterCreateSupplierEventListener* berperan sebagai listener yang secara khusus menangani *event* bertipe

MasterSupplierCreateEvent. Penggunaan anotasi @*EventListener* memungkinkan listener ini merespons *event* secara otomatis tanpa pemanggilan langsung antar modul, sehingga mendukung arsitektur sistem yang terpisah dan modular.

Ketika event diterima, *listener* terlebih dahulu melakukan inisialisasi konteks eksekusi, seperti penetapan *request identifier* dan informasi pengguna yang terkait dengan *event*. Setelah itu, data yang dibawa oleh event dipetakan ke dalam DTO khusus modul *searching*, yaitu *MasterSupplierCreateDto*, yang merepresentasikan data *supplier* dalam format yang siap diproses oleh lapisan layanan pencarian.

Tahap berikutnya adalah pemanggilan *service layer* pada modul *searching* untuk menjalankan proses bisnis utama, yaitu pembuatan atau pembaruan data supplier pada indeks pencarian. Pemanggilan ini dilakukan melalui *masterSupplierService.createMasterSupplier*, yang secara internal akan mengelola proses *indexing* ke *Elasticsearch*. Dengan alur ini, setiap perubahan data pada sistem utama akan langsung diteruskan dan direfleksikan ke dalam *indeks* pencarian secara otomatis, sehingga konsistensi data antara basis data utama dan *Elasticsearch* tetap terjaga tanpa memerlukan proses sinkronisasi manual.

B.3 Lapisan *Service*

```
public MasterSupplierDto createMasterSupplier(
    MasterSupplierCreateDto masterSupplierCreateDto) {
     MasterSupplier masterSupplier = MasterSupplier
          .builder()
          .id(masterSupplierCreateDto.getId())
          .supplierName(masterSupplierCreateDto.getSupplierName())
          .supplierCode(masterSupplierCreateDto.getSupplierCode())
          .createdBy(masterSupplierCreateDto.getCreatedBy())
          .build();
     masterSupplier = masterSupplierRepository.save(masterSupplier)
    ;
     return masterSupplierMapper.map(masterSupplier,
    MasterSupplierDto.class);
}
```

Kode 3.7: Implementasi Service dalam Menangani Event Perubahan Data

Pada lapisan *service*, proses pembaruan indeks pencarian dilakukan melalui metode *createMasterSupplier*. Metode ini menerima objek *MasterSupplierCreateDto* yang berasal dari *event listener* sebagai representasi

data hasil perubahan pada sistem utama. Data tersebut kemudian digunakan untuk membangun objek domain *MasterSupplier*, yang disesuaikan dengan struktur dokumen yang akan disimpan pada *Elasticsearch*. Tahap ini memastikan bahwa hanya atribut yang relevan untuk kebutuhan pencarian yang diproses lebih lanjut.

B.4 Lapisan *Repository*

Setelah objek domain terbentuk, data disimpan ke dalam *repository* menggunakan mekanisme *save*. Lapisan *repository* pada modul *searching* berperan sebagai komponen yang menangani interaksi langsung dengan *Elasticsearch* sebagai media penyimpanan indeks pencarian. *Repository* digunakan untuk mengelola proses penyimpanan, pembaruan, serta pengambilan data dokumen pencarian tanpa mengharuskan lapisan *service* berinteraksi secara langsung dengan detail teknis penyimpanan data. Dengan adanya lapisan ini, akses terhadap *Elasticsearch* dapat dilakukan secara terstruktur dan terabstraksi, sehingga kompleksitas operasi indeks tidak tersebar ke bagian sistem lainnya.

Repository dimanfaatkan untuk mengeksekusi operasi *indexing* ketika terjadi perubahan data dari sistem utama, termasuk penyimpanan dokumen baru, pembaruan dokumen yang sudah ada, maupun penghapusan dokumen dari indeks. Pendekatan ini memastikan setiap perubahan data yang diterima melalui *event listener* dapat segera direfleksikan ke dalam indeks pencarian secara konsisten dan terkontrol.

3.4.6 Implementasi dan Hasil

Implementasi modul *searching* pada sistem ERP Cranium dilakukan dengan mengintegrasikan layanan pencarian ke dalam sisi *front-end* melalui pemanggilan *endpoint* API yang telah disediakan. Pada tahap ini, *front-end* tidak berinteraksi langsung dengan basis data relasional, melainkan berkomunikasi dengan modul *searching* sebagai layanan khusus yang menangani proses pencarian data.

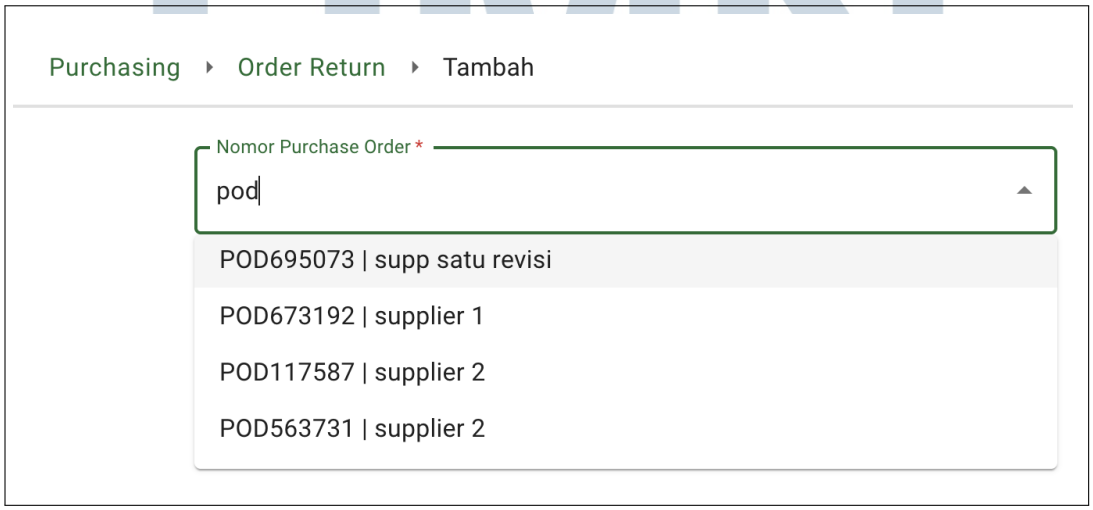
```
export const getOrderListInfinite = async (
  input: string,
  size: number,
  pageParam = 0
): Promise<Page<OrderList>> => {
  return getApiClient().get(
    '/searching-service/purchasing/orders',
```

```
    {
      params: {
        page: pageParam,
        size: size,
        input: input,
      },
    }
  );
};
```

Kode 3.8: Implementasi Pemanggilan *Endpoint* Modul *Searching* pada Sisi *Front-End*

Pada sisi *front-end*, pemanggilan modul *searching* diimplementasikan menggunakan *HTTP client* dengan mengirimkan parameter pencarian seperti kata kunci (*input*), ukuran halaman (*size*), serta nomor halaman (*page*). Parameter tersebut dikirimkan sebagai *query parameter* untuk mendukung proses pencarian dan mekanisme paginasi data secara dinamis. Pendekatan ini memungkinkan antarmuka aplikasi menampilkan hasil pencarian secara bertahap sesuai kebutuhan pengguna.

Dengan adanya modul *searching*, proses pencarian tidak lagi dilakukan secara langsung terhadap basis data relasional. Seluruh logika pencarian, pengolahan parameter, serta pengambilan data dilakukan pada layanan pencarian yang terpisah. Hal ini memberikan pemisahan tanggung jawab yang lebih jelas antara pengelolaan data transaksi dan proses pencarian, sehingga sistem menjadi lebih terstruktur dan mudah dikembangkan.



Gambar 3.6. Hasil Implementasi *Searching* pada field pencarian *Purchasing Order Return Master Supplier*

Hasil implementasi menunjukkan bahwa fitur pencarian pada sisi *front-end* dapat berjalan dengan baik dan konsisten, di mana data *purchasing order* dan *master supplier* dapat ditampilkan sesuai dengan parameter pencarian yang diberikan oleh pengguna. Selain itu, pemusatan proses pencarian pada modul *searching* membantu menjaga performa basis data utama, karena beban pencarian tidak lagi dibebankan langsung pada sistem basis data relasional.

Sebagai perbandingan performa implementasi modul *searching*, dilakukan evaluasi antara pencarian berbasis SQL pada basis data relasional dan pencarian berbasis Elasticsearch. Perbandingan ini dilakukan menggunakan *Apache JMeter* dengan skenario beban dan jumlah permintaan yang setara. Hasil perbandingan performa dirangkum pada Tabel 3.2.

Tabel 3.2. Hasil Evaluasi Performa Pencarian Data Menggunakan Basis Data Relasional dan Elasticsearch

Metode Pencarian	**#Request**	**Avg (ms)**	**Std. Dev**	**Throughput (req/s)**	**Error (%)**
SQL	70.019	508	178,69	178,3	0,00
Elasticsearch	70.010	40	122,42	178,4	0,00

Berdasarkan hasil evaluasi performa pada Tabel 3.2, metode pencarian berbasis Elasticsearch menunjukkan waktu respons rata-rata yang jauh lebih rendah dibandingkan pencarian berbasis SQL. Rata-rata waktu respons Elasticsearch berada pada kisaran 40 ms, sedangkan pencarian menggunakan SQL memiliki rata-rata waktu respons sekitar 508 ms. Perbedaan ini menunjukkan bahwa Elasticsearch mampu menangani proses pencarian data dengan lebih cepat, terutama pada kondisi jumlah permintaan yang tinggi.

Selain itu, nilai *standard deviation* pada pencarian berbasis Elasticsearch lebih rendah dibandingkan SQL, yang mengindikasikan bahwa waktu respons Elasticsearch lebih stabil dan konsisten. Nilai *throughput* pada kedua metode relatif setara, menandakan bahwa pengujian dilakukan pada beban yang seimbang. Tidak ditemukannya kesalahan (*error rate* sebesar 0%) pada kedua metode menunjukkan bahwa seluruh permintaan berhasil diproses dengan baik selama pengujian berlangsung.

Secara keseluruhan, hasil implementasi dan evaluasi performa menunjukkan bahwa penggunaan modul *searching* dengan dukungan Elasticsearch memberikan peningkatan signifikan terhadap kecepatan dan konsistensi pencarian data pada sistem ERP Cranium. Pendekatan ini tidak hanya meningkatkan pengalaman

pengguna pada sisi *front-end*, tetapi juga membantu menjaga performa dan skalabilitas sistem secara keseluruhan.

3.4.7 Testing

Pada tahap pengujian, sistem modul *searching* diuji melalui tiga pendekatan utama, yaitu pengujian *front-end*, *back-end*, serta pengujian manual oleh tim *Quality Assurance* (QA). Pengujian *back-end* dilakukan menggunakan *JUnit*, *Mockito*, *Spring Boot Test*, dan *Spring Cloud Contract* untuk memvalidasi logika bisnis, integrasi layanan, serta konsistensi kontrak API. Sementara itu, pengujian *front-end* memanfaatkan *Mock Service Worker* (MSW) untuk mensimulasikan respons API, sedangkan pengujian manual QA digunakan untuk memastikan kesesuaian fungsional sistem dengan kebutuhan pengguna. Berikut adalah penjelasan mengenai masing-masing pendekatan dalam *testing*.

A Pengujian Back-End

Pengujian pada sisi *back-end* dilakukan untuk memastikan bahwa seluruh fungsi layanan pada modul *Searching* berjalan sesuai dengan spesifikasi, baik dari sisi logika bisnis maupun kontrak API antar layanan. Pendekatan pengujian yang digunakan terdiri dari dua jenis, yaitu *unit testing* untuk memverifikasi logika internal aplikasi dan *contract testing* untuk menjamin konsistensi komunikasi antarlayanan.

Pengujian berbasis kontrak dilakukan menggunakan *Spring Cloud Contract* dengan mendefinisikan spesifikasi API dalam bentuk berkas *Groovy*. Salah satu kontrak yang diuji adalah proses pembuatan data *Master Supplier*, yang didefinisikan dalam berkas *createMasterSupplierSuccess.groovy*. Kontrak ini menetapkan struktur permintaan HTTP, *header*, serta respons yang harus dikembalikan oleh layanan *Searching* ketika proses pembuatan supplier berhasil.

```
Contract.make {
description "create master supplier"

request {
    url "/searching-service/internal/master/supplier"
    method POST()
    headers {
        contentType(applicationJson())
        header "X-Api-Version": "1"
```

```
    }
    body (
        id: 3,
        supplierName: "Supplier 06",
        supplierCode: "SUPPLIER-06",
        createdBy: 12
    )
}

response {
    status CREATED()
    body (
        id: 3,
        supplierName: "Supplier 06",
        supplierCode: "SUPPLIER-06"
    )
}

}
```

Kode 3.9: Kontrak API *createMasterSupplierSuccess.groovy*

Kontrak tersebut berfungsi sebagai kesepakatan formal antara layanan penyedia (*provider*) dan layanan pemanggil (*consumer*) agar struktur data dan perilaku API tetap konsisten meskipun terjadi pengembangan atau perubahan kode.

Agar kontrak *Groovy* dapat dieksekusi dan divalidasi terhadap implementasi aplikasi, digunakan kelas *SearchingMasterSupplierBase* sebagai *base class* untuk *contract testing*. Kelas ini bertanggung jawab menyediakan konteks aplikasi, konfigurasi *Elasticsearch*, serta dependensi yang dibutuhkan selama proses pengujian kontrak berlangsung.

```
@ContextConfiguration(classes = DataElasticSearchingConfiguration.
    class)
public abstract class SearchingMasterSupplierBase {

@Autowired
protected MasterSupplierService masterSupplierService;

}
```

Kode 3.10: Struktur *SearchingMasterSupplierBase*

Melalui kelas ini, kontrak yang didefinisikan dalam berkas *Groovy* akan dieksekusi seolah-olah permintaan HTTP sungguhan dikirimkan ke aplikasi.

Dengan demikian, hasil respons yang dihasilkan oleh *controller* dan *service* dapat divalidasi secara otomatis terhadap spesifikasi kontrak yang telah ditetapkan.

Selain pengujian kontrak, dilakukan pula *unit testing* untuk memverifikasi logika bisnis pada lapisan *service*. Pengujian ini diimplementasikan dalam kelas *MasterSupplierServiceTest* dengan menggunakan *JUnit* dan *Mockito*. Fokus utama pengujian ini adalah memastikan setiap metode pada *MasterSupplierService* menghasilkan keluaran yang sesuai dan menangani kondisi kesalahan dengan benar.

```
@Test
void testCreateMasterSupplier() throws DataNotFoundException {
MasterSupplierCreateDto dto = MasterSupplierCreateDto.builder()
.id(2L)
.supplierName("Supplier 06")
.supplierCode("SUPPLIER-06")
.createdBy(12L)
.build();

MasterSupplier entity = MasterSupplier.builder()
    .id(dto.getId())
    .supplierName(dto.getSupplierName())
    .supplierCode(dto.getSupplierCode())
    .createdBy(dto.getCreatedBy())
    .build();

Mockito.when(masterSupplierRepository.save(entity))
       .thenReturn(entity);

MasterSupplierDto result =
    masterSupplierService.createMasterSupplier(dto);

assertEquals(dto.getId(), result.getId());

}
```

Kode 3.11: Unit Test *createMasterSupplier* pada *MasterSupplierServiceTest*

Pengujian ini memastikan bahwa proses pembuatan, pembaruan, pengambilan, dan penghapusan data *Master Supplier* berjalan sesuai dengan aturan bisnis yang telah ditetapkan tanpa melibatkan lapisan HTTP atau kontrak antarlayanan.

Secara keseluruhan, *MasterSupplierServiceTest* berfungsi untuk memvalidasi logika internal aplikasi, sedangkan

createMasterSupplierSuccess.groovy yang dijalankan melalui *SearchingMasterSupplierBase* berfungsi untuk memastikan kesesuaian antarmuka API. Kombinasi kedua pendekatan ini memberikan jaminan bahwa layanan tidak hanya benar secara fungsional, tetapi juga konsisten dalam konteks integrasi antar sistem.

B Pengujian Front-End

Pengujian pada sisi *front-end* dilakukan untuk memastikan bahwa fitur pencarian *purchasing order* dapat menampilkan data secara benar sesuai dengan parameter pencarian yang diberikan oleh pengguna. Untuk menghindari ketergantungan langsung terhadap layanan *back-end*, digunakan pendekatan *mocking API* dengan *Mock Service Worker* (MSW).

MSW digunakan untuk mensimulasikan respons dari *modul searching* ketika aplikasi *front-end* melakukan permintaan pencarian. Potongan kode berikut menunjukkan *handler* yang menangani permintaan pencarian *purchasing order*.

```
const getOrderListInfiniteHandler = rest.get(
  `${API_SEARCHING_URL}/searching-service/purchasing/orders`,
  async (req, res, ctx) => {
    const page = Number(req.url.searchParams.get('page'));
    const size = Number(req.url.searchParams.get('size'));
    const input = req.url.searchParams.get('input');

    const orders = db.purchasing_order.findMany({
      where: {
        documentNo: { contains: input },
      },
      take: size,
      skip: page * size,
    });

    return res(
      ctx.status(200),
      ctx.json({
        content: orders,
        totalPages: 5,
        totalElements: 10,
      })
    );
  }
```

```
);
```

Kode 3.12: Mock API *Searching Purchasing Order*

Pada implementasi tersebut, *handler* membaca parameter pencarian *page*, *size*, dan *input* dari URL, lalu mengembalikan data tiruan yang telah difilter berdasarkan *document number*. Respons disusun menyerupai format *pagination* agar dapat diproses oleh komponen *front-end* tanpa perbedaan dengan layanan nyata.

Handler pencarian ini kemudian didaftarkan ke dalam konfigurasi MSW agar dapat digunakan selama proses pengujian, seperti ditunjukkan pada potongan kode berikut.

```
export const searchingPurchasingOrdersHandlers = [
  getOrderListInfiniteHandler,
];
```

Kode 3.13: Registrasi *Handler MSW*

Melalui pengujian ini, dapat dipastikan bahwa komponen pencarian pada *front-end* telah mampu menangani hasil pencarian, melakukan *pagination*, serta menampilkan data *purchasing order* secara konsisten sesuai dengan respons dari *modul searching*.

3.5 Kendala dan Solusi

Kendala yang ditemukan selama kerja praktik magang adalah sebagai berikut:

- Keterbatasan pemahaman awal mengenai *business process* menyebabkan proses pengerjaan modul membutuhkan waktu lebih lama karena harus mempelajari alur *business process* terlebih dahulu.
- Kurang matangnya *brief* pada beberapa penugasan, sehingga seringkali ada tambahan tugas ataupun penyesuaian di saat pengerjaan task sudah dimulai.
- Dokumentasi teknis pada beberapa modul belum lengkap, sehingga pemahaman terhadap fungsi dan alur sistem harus dilakukan melalui eksplorasi kode secara langsung.

Solusi atas kendala-kendala yang dihadapi selama kerja praktik magang adalah sebagai berikut:

- Melakukan pembelajaran secara mandiri melalui dokumen internal serta secara aktif berdiskusi dan bertanya kepada mentor guna memahami alur *business process* yang diterapkan.
- Klarifikasi kebutuhan dan ruang lingkup *task* sejak awal pengerjaan serta berkomunikasi secara berkala dengan mentor terkait perkembangan dan perubahan yang terjadi..
- Melakukan eksplorasi kode sumber secara langsung, mempelajari struktur dan alur program, serta mencatat temuan penting sebagai referensi pribadi selama proses pengembangan.