BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan kegiatan magang di PT Tri Daya Langgeng, kegiatan dilakukan di bawah Divisi Keuangan. Meskipun secara struktural berada dalam divisi tersebut, tanggung jawab utama difokuskan pada peran sebagai *Mobile App Developer* yang berorientasi pada pembuatan aplikasi berbasis *Android*. Kegiatan pembuatan aplikasi dilakukan oleh tim kecil yang terdiri atas tiga orang, yaitu dua orang *Mobile App Developer* dan satu orang *UI/UX Designer*. Setiap anggota tim memiliki tanggung jawab masing-masing, di mana *Mobile App Developer* berfokus pada proses pengkodean, implementasi fitur, serta pengujian aplikasi, sementara *UI/UX Designer* bertanggung jawab terhadap perancangan antarmuka pengguna dan pengalaman pengguna agar sesuai dengan kebutuhan sistem.

Koordinasi antara anggota tim dilakukan secara langsung melalui pertemuan tatap muka di lingkungan kerja untuk membahas progres pekerjaan, pembagian tugas, serta penyelesaian kendala teknis yang muncul selama proses pembuatan aplikasi. Selain koordinasi secara langsung, komunikasi juga dilakukan secara daring ketika bekerja dari rumah *(work from home*) dengan memanfaatkan platform *WhatsApp* dan *Discord* sebagai media utama dalam bertukar informasi, berdiskusi, serta melakukan pembaruan terkait perkembangan proyek. Kegiatan magang ini juga mendapatkan bimbingan langsung dari Bapak Nirwana selaku pembimbing lapangan sekaligus sebagai *supervisor*.

Untuk mendukung kolaborasi dalam pembuatan aplikasi, tim memanfaatkan *repository GitHub* sebagai media penyimpanan dan pengelolaan kode program secara terpusat. Melalui *platform* ini, setiap anggota tim dapat melakukan pembaruan, sinkronisasi, serta integrasi terhadap bagian kode yang dikerjakan masing-masing, sehingga proses pembuatan dapat berlangsung secara terkoordinasi, terdokumentasi, dan terstruktur dengan baik.

3.2 Tugas yang Dilakukan

Selama melaksanakan kegiatan magang di PT Tri Daya Langgeng, tanggung jawab utama adalah sebagai *Full Stack Developer*. Peran ini mencakup pengembangan antarmuka pengguna dan pengelolaan logika aplikasi. Dalam

pelaksanaannya, terdapat tiga fitur utama yang dikembangkan, yaitu *Dashboard*, Daftar Material, dan *Progress* Pengerjaan Pesanan.

Fitur *Dashboard* berfungsi sebagai tampilan ringkasan utama yang menampilkan berbagai informasi penting terkait aktivitas perusahaan. Melalui halaman ini, pengguna dapat melihat jumlah total *material* yang tersedia, total pesanan yang sedang dalam proses pengerjaan, serta total pesanan yang telah selesai dikerjakan. Fitur ini dirancang agar direktur dapat memperoleh gambaran umum mengenai kondisi dan progres operasional perusahaan secara cepat dan efisien.

Fitur berikutnya adalah Daftar Material, yang merupakan halaman dengan fungsi *CRUD*. Melalui halaman ini, pengguna dapat menambahkan data *material* baru, memperbarui informasi *material* yang telah ada, maupun menghapus data *material* yang sudah tidak digunakan. Fitur ini dikembangkan untuk mempermudah pengelolaan data *material* agar tetap terorganisasi dan dapat diakses dengan mudah oleh pihak yang membutuhkan.

Fitur terakhir adalah Progres Pengerjaan Pesanan, yang berfungsi untuk memantau perkembangan pengerjaan pesanan dari masing-masing perusahaan. Melalui fitur ini, pengguna dapat melihat sejauh mana progres pengerjaan suatu pesanan, melakukan *input* harian terkait aktivitas produksi, serta mengatur proses pengiriman barang ketika pesanan telah mencapai tahap penyelesaian. Dengan adanya fitur ini, proses pelacakan *status* pesanan menjadi lebih terukur.

3.3 Uraian Pelaksanaan Magang

Rincian pekerjaan mingguan selama kegiatan magang di PT. Tri Daya Langgeng disajikan pada Tabel 3.1 berikut.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama magang

Minggu Ke -	**Pekerjaan yang dilakukan**
1	Perkenalan lingkungan kerja magang di PT Tri Daya Langgeng
2	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, diskusi dengan direktur mengenai tugas tambahan, serta pengerjaan tugas yang diberikan oleh direktur perusahaan.
	Lanjut pada halaman berikutnya

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

Minggu Ke -	Pekerjaan yang dilakukan
3	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, diskusi dengan tim IT mengenai permasalahan perusahaan serta solusi yang ditemukan, serta pengerjaan tugas-tugas yang diberikan oleh direktur perusahaan.
4	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, pembelajaran mengenai bahasa pemrograman *Kotlin*, *responsive design view*, serta pemahaman penggunaan komponen *Android Studio* seperti *bottom sheet, card, dan navigation drawer*. Selain itu, dilakukan juga pembelajaran tentang komponen *navigation bar* dan pengerjaan tugas yang diberikan oleh direktur perusahaan.
5	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, diskusi dengan tim IT serta membantu pembuatan desain *dashboard* pada *Figma*, pengerjaan tugas yang diberikan oleh direktur perusahaan, serta kolaborasi dengan *UI/UX designer* dalam pembuatan desain *Figma* untuk aplikasi yang sedang dibuat.
6	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, diskusi dengan tim IT mengenai cara kerja aplikasi yang akan dibuat, pemberian saran kepada *UI/UX designer* terkait desain halaman histori pemesanan, serta diskusi dengan *full stack developer* mengenai mekanisme *halaman dashboard*, *order progress*, dan akumulasi pendapatan.
7	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor dan presentasi desain kepada direktur perusahaan, pengerjaan serta revisi halaman *dashboard*, perbaikan bug, koneksi *database Firestore*, dan penyelesaian tugas tambahan dari direktur.
	Lanjut pada halaman berikutnya

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

Minggu Ke -	Pekerjaan yang dilakukan
8	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, mengerjakan tugas yang diberikan oleh direktur perusahaan, perbaikan tampilan *bottom sheet*, pembuatan *navigation bar* dan *navigation drawer*, revisi tampilan *halaman dashboard*, serta pembuatan *BaseActivity.kt* untuk mempermudah penggunaan *bottom navigation* dan *drawer navigation* di seluruh halaman.
9	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, pengerjaan tugas yang diberikan oleh direktur perusahaan, revisi tampilan halaman *dashboard* sesuai desain baru, pembuatan desain *popup* untuk penambahan *material*, serta pengerjaan lanjutan halaman daftar *material* sesuai desain Figma terbaru.
10	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor dan klien untuk membahas pemesanan, membantu direktur menyiapkan data pesanan baru, penerapan *view binding* pada halaman *dashboard* dan halaman daftar *material*, pembuatan desain *popup* untuk *edit material*, serta perbaikan *bug* pada *MaterialListActivity.kt* yang menghambat pengambilan data dari *Firestore*.
11	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, membantu direktur dan staf menyiapkan kunjungan klien serta mengikuti rapat terkait, membantu *developer* memperbaiki *bug* pada halaman tambah pesanan baru akibat kesalahan input data ke *Firestore*, serta mulai mengerjakan halaman Progres Pengerjaan Pesanan dengan membuat *adapter* dan *activity* dengan nama *OrderProgressActivity.kt* dan *OrderProgressAdapter.kt*.
Lanjut pada halaman berikutnya	

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

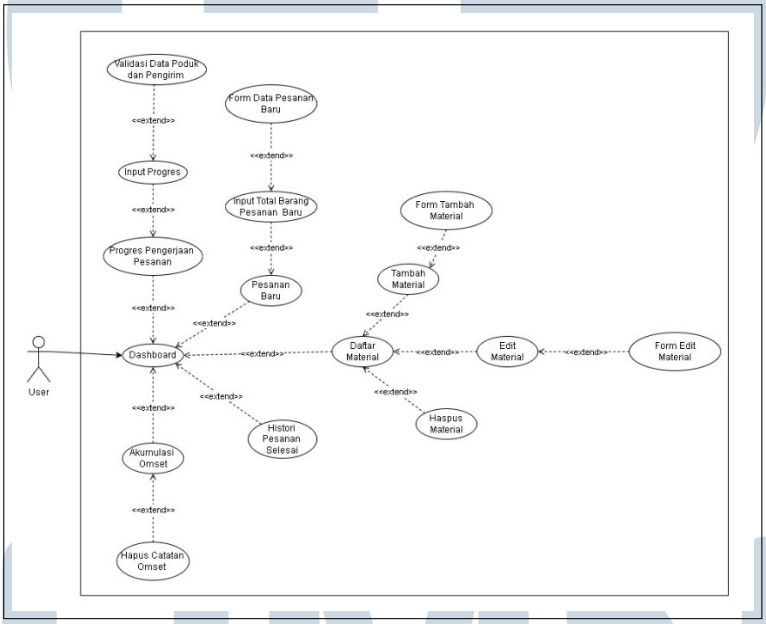
Minggu Ke -	Pekerjaan yang dilakukan
12	Kegiatan minggu ini meliputi meeting bersama karyawan kantor, pengembangan halaman *progress* pengerjaan pesanan dengan pembuatan *model PTOrderProgress.kt* dan *OrderProgressModel.kt*, pembuatan *popup* input harian, peninjauan ulang struktur *database*, serta perbaikan *bug* terkait pembaruan *data progress* yang belum *terupdate* pada tampilan *card*.
13	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, perbaikan tampilan dan logika pada *popup* input harian agar jumlah input tidak melebihi pesanan, penyesuaian tampilan *popup* sesuai desain *Figma*, *refaktor kode* dari *OrderProgressAdapter.kt* ke *OrderProgressActivity.kt*, serta penyesuaian logika *dashboard* agar penghitung total pesanan selesai hanya menampilkan data pada bulan berjalan.
14	Kegiatan minggu ini meliputi *meeting* bersama karyawan kantor, pembuatan fungsi *checkDuplicateMaterial* pada halaman daftar material, penyesuaian logika *number format* Indonesia, serta pengujian dan pengecekan *fitur* aplikasi untuk memastikan tidak terdapat *bug*.
15	Kegiatan minggu ini meliputi beberapa *meeting* bersama karyawan kantor dan presentasi hasil pembuatan aplikasi pencatatan pesanan. Selain itu, dilakukan pengerjaan berbagai tugas yang diberikan oleh direktur perusahaan, khususnya penanganan dan pengelolaan pesanan baru yang masuk.
16	Kegiatan minggu ini meliputi *meeting* dengan karyawan kantor, dan mengerjakan tugas yang diberikan oleh direktur.
17	Kegiatan minggu ini meliputi *meeting* dengan karyawan kantor, dan mengerjakan tugas yang diberikan oleh direktur.

3.4 Perancangan dan Implementasi

Pada bagian ini akan dijelaskan proses perancangan sistem serta tahap-tahap implementasinya yang disesuaikan dengan kebutuhan dalam pembuatan aplikasi pencatatan dan pelacakan pesanan di PT Tri Daya Langgeng.

3.4.1 Use Case Diagram

Gambar 3.1 menampilkan *use case diagram* yang memperlihatkan bagaimana pengguna berinteraksi dengan sistem untuk menjalankan berbagai fungsi aplikasi. Diagram tersebut menunjukkan bahwa aplikasi berawal dari halaman *Dashboard*, yang berfungsi sebagai *landing page* sekaligus pusat navigasi. Melalui *Dashboard*, pengguna dapat mengakses lima fitur utama yang saling terhubung, yaitu Progres Pengerjaan Pesanan, Pesanan Baru, Daftar Material, *History* Pesanan Selesai, dan Akumulasi Omset.



Gambar 3.1. Use case diagram aplikasi pencatatan pesanan

Pada fitur Progres Pengerjaan Pesanan, pengguna dapat memantau perkembangan produksi setiap pesanan, melakukan *input* progres harian, serta melakukan validasi data produk dan pengirim sebelum pesanan dinyatakan selesai. Sementara itu, fitur Pesanan Baru memungkinkan pengguna untuk mencatat pesanan pelanggan dengan *menginput* berbagai informasi seperti nama pemesan, jumlah barang, tanggal pemesanan, hingga perhitungan total barang dan biaya yang dihitung secara otomatis oleh sistem.

Fitur Daftar Material berfungsi sebagai pusat pengelolaan stok material yang tersedia. Di dalamnya terdapat proses Tambah Material, *Edit* Material, dan Hapus Material yang membantu memastikan ketersediaan stok tetap akurat dan terorganisasi. Selanjutnya, fitur *History* Pesanan Selesai menampilkan seluruh

pesanan yang telah selesai dikerjakan sebagai arsip digital, sehingga memudahkan pengguna dalam meninjau riwayat pesanan, termasuk tanggal pesanan, tanggal pengiriman, dan total tagihan.

Terakhir, fitur Akumulasi Omset menyediakan informasi total pendapatan yang diperoleh dalam periode tertentu. Pengguna juga dapat menghapus catatan omset yang sudah tidak diperlukan agar data keuangan tetap rapi dan terkontrol.

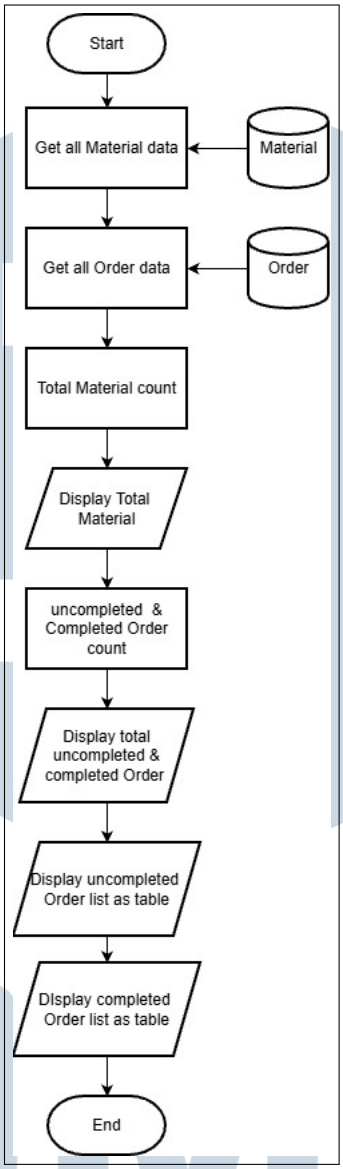
3.4.2 Flowchart Aplikasi

Dibawah ini akan di jabarkan dan dijelaskan *flowchart* yang berfungsi sebagai alur kerja aplikasi yang dibuat.

A Flowchart Halaman Dashboard

Gambar 3.2 merupakan *Flowchart* halaman *Dashboard* yang menggambarkan alur kerja sistem dalam menampilkan ringkasan informasi. Proses dimulai dengan pengambilan seluruh data material dari basis data untuk memperoleh jumlah total material yang tersedia. Setelah itu, sistem mengambil seluruh data pesanan yang tersimpan pada basis data. Data tersebut kemudian diolah untuk menghitung jumlah pesanan yang masih dalam proses pengerjaan, serta jumlah pesanan yang telah selesai.

Selanjutnya, sistem menampilkan hasil perhitungan tersebut pada halaman *Dashboard* dalam bentuk informasi ringkas meliputi total *material*, total pesanan yang sedang dikerjakan, dan total pesanan yang sudah selesai. Setelah proses perhitungan selesai, sistem menampilkan daftar pesanan yang belum selesai dalam bentuk tabel pada sebuah *card* khusus, sehingga pengguna dapat memantau secara jelas pesanan apa saja yang masih dikerjakan. Di bawahnya, sistem juga menampilkan daftar pesanan yang telah selesai dalam bentuk tabel serupa.



Gambar 3.2. Flowchart halaman dashboard

B Flowchart Halaman Daftar Material

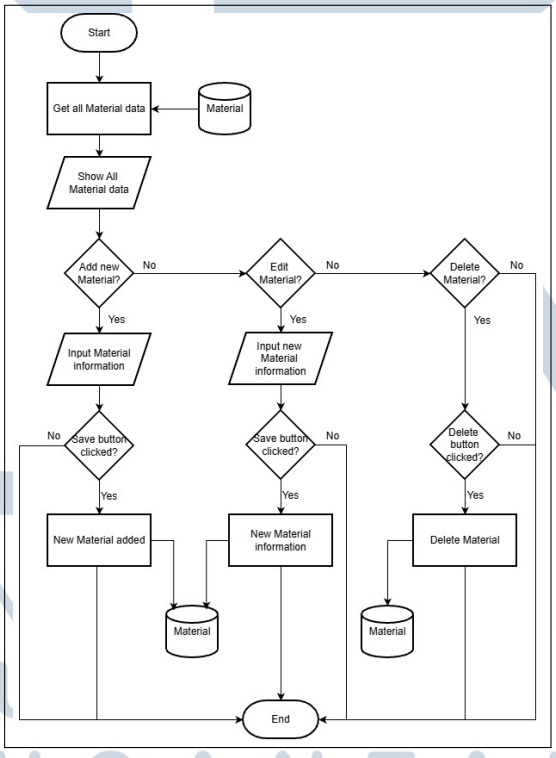
Gambar 3.3 merupakan *Flowchart* halaman Daftar Material yang berfungsi sebagai proses kerja sistem dalam mengelola data material yang terdiri atas tiga operasi utama, yaitu penambahan material baru, edit informasi material, dan penghapusan material. Proses dimulai dengan pengambilan seluruh data material dari basis data untuk ditampilkan kepada pengguna pada halaman Daftar Material.

Setelah data berhasil dimuat dan di tampilkan, sistem menyediakan tiga opsi pengelolaan data material. Pertama, apabila pengguna memilih untuk

menambahkan material baru, sistem akan menampilkan formulir untuk menginput informasi material. Pengguna kemudian memasukkan data material yang diperlukan, dan sistem akan menunggu hingga tombol "*Save*" ditekan. Setelah tombol tersebut diklik, data material baru disimpan ke dalam basis data dan daftar material diperbarui.

Kedua, apabila pengguna memilih untuk *edit* material yang sudah ada, sistem menampilkan formulir berisi informasi material yang ingin diubah. Pengguna dapat mengubah data sesuai kebutuhan, dan proses penyimpanan perubahan dilakukan setelah tombol "*Save*" ditekan. Informasi material yang sudah diperbarui kemudian disimpan kembali ke dalam basis data dan perubahan akan tercermin pada daftar material.

Ketiga, apabila pengguna memilih untuk menghapus material, sistem akan menunggu konfirmasi melalui tombol "*Delete*". Setelah tombol tersebut diklik, sistem akan menghapus data material yang dipilih dari basis data, kemudian memperbarui daftar material agar sesuai dengan kondisi terbaru.



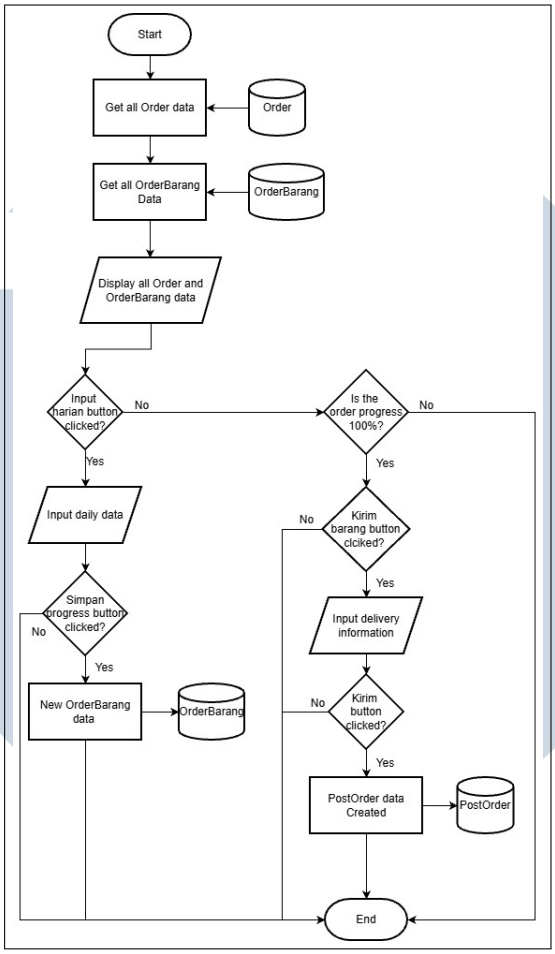
Gambar 3.3. Flowchart halaman daftar material

C Flowchart Halaman Progres Pengerjaan Pesanan

Gambar 3.4 merupakan *Flowchart* halaman Progres Pengerjaan Pesanan yang berfungsi untuk menampilkan data pesanan serta menyediakan dua fungsi utama, yaitu input data harian pengerjaan dan proses pengiriman barang ketika pesanan telah selesai dikerjakan. Alur dimulai dengan pengambilan seluruh data pesanan (*Order*) dari basis data. Setelah itu, sistem juga memuat seluruh data *OrderBarang* yang berfungsi untuk menampilkan daftar jumlah pesanan yang dipesan oleh setiap perusahaan. Kedua data tersebut kemudian ditampilkan pada halaman untuk memberikan informasi lengkap mengenai detil barang yang di pesan dan progres pengerjaan setiap pesanan.

Setelah data ditampilkan, sistem menyediakan opsi pertama berupa fitur input data harian. Ketika tombol input harian ditekan, sistem menampilkan formulir untuk memasukkan jumlah progres pengerjaan yang telah dicapai pada hari tersebut. Setelah data dimasukkan, sistem menunggu hingga tombol "Simpan Progres" diklik. Ketika tombol tersebut ditekan, sistem menyimpan data progres harian yang baru ke dalam basis data *OrderProgress*.

Selain input harian, *flowchart* juga menunjukkan proses ketika suatu pesanan telah mencapai 100% penyelesaian. Sistem memeriksa status progres pesanan dan apabila semua *OrderBarang* telah selesai, maka tombol untuk melakukan proses kirim barang akan tersedia. Ketika tombol tersebut ditekan, sistem menampilkan formulir untuk mengisi informasi pengiriman, seperti tanggal pengiriman, nama kurir, dan total barang cacat selama pengerjaan. Setelah pengguna mengisi data tersebut dan menekan tombol "Kirim", sistem menyimpan informasi pengiriman tersebut ke dalam basis data dan memperbarui status pesanan menjadi terkirim.



Gambar 3.4. Flowchart halaman progres pengerjaan pesanan

3.4.3 Document Data Model

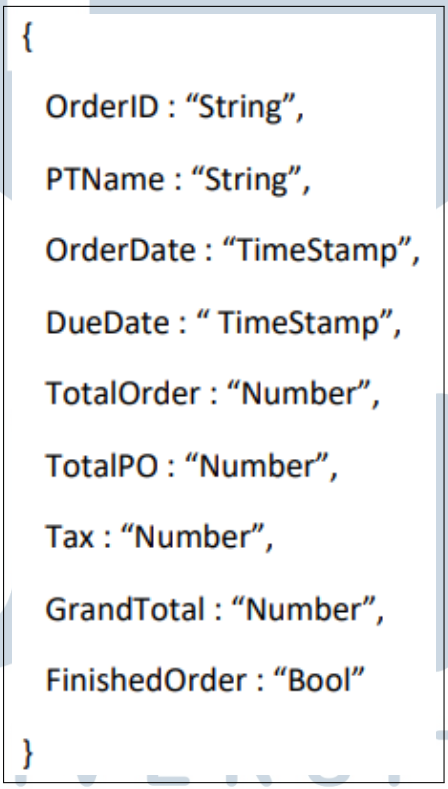
Document data model merupakan metode pemodelan basis data pada sistem *NoSQL* yang menyimpan informasi dalam bentuk dokumen. Setiap dokumen berisi sekumpulan pasangan *field* dan *value* yang digunakan untuk menggambarkan suatu data, serta tidak bergantung pada konsep relasi antar tabel sebagaimana yang diterapkan pada basis data relasional [6]. Dokumen-dokumen tersebut dikelompokkan ke dalam *collection* berdasarkan jenis data yang disimpan, sehingga struktur data menjadi lebih fleksibel dan mudah dikembangkan.

Basis data yang digunakan pada aplikasi ini adalah *Firestore*. *Firestore* merupakan layanan basis data *NoSQL* berbasis *cloud* yang dikembangkan oleh *Google* untuk menyimpan dan mengelola data aplikasi secara *real-time*. *Firestore* menerapkan *document data model* sebagai dasar penyimpanan data, di mana setiap data disimpan dalam bentuk dokumen yang terorganisasi ke dalam *collection*.Pada

aplikasi ini, struktur basis data terdiri atas beberapa *collection* utama, yaitu Order, OrderBarang, OrderProgress, PostOrder, dan Material.

A Struktur Dokumen Order

Gambar 3.5 menunjukkan struktur data dokumen pada *collection* Order yang berfungsi sebagai penyimpanan data utama terkait pemesanan dari klien. *Collection* ini menyimpan kumpulan dokumen Order, di mana setiap dokumen merepresentasikan satu pesanan yang dibuat oleh perusahaan pemesan. Informasi yang disimpan dalam setiap dokumen meliputi identitas pesanan, nama perusahaan, tanggal pemesanan, tanggal jatuh tempo, jumlah total pesanan, nilai total sebelum pajak, pajak, total keseluruhan pembayaran, serta status penyelesaian pesanan.

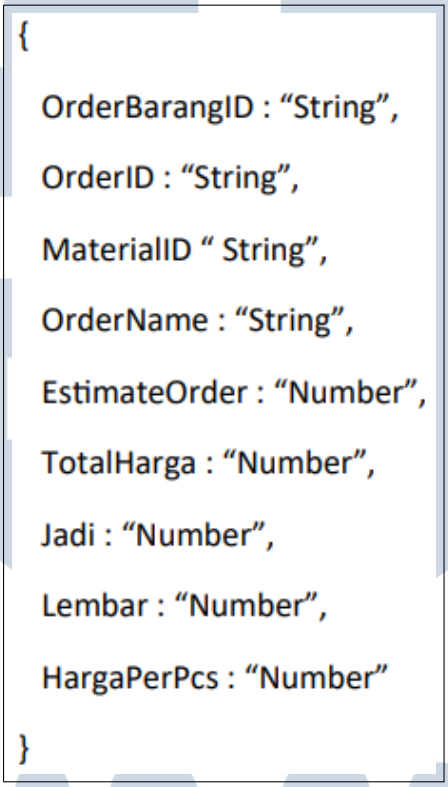


Gambar 3.5. Struktur dokumen order

B Struktur Dokumen OrderBarang

Gambar 3.6 menunjukkan struktur dokumen pada *collection* OrderBarang yang digunakan untuk menyimpan data detail barang yang dipesan dalam suatu pesanan. *Collection* ini berisi kumpulan dokumen OrderBarang, di mana setiap

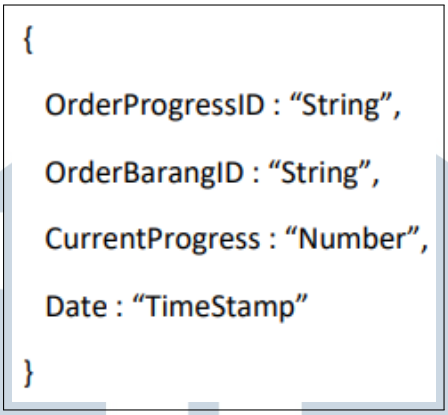
dokumen merepresentasikan satu jenis barang pesanan. Informasi yang disimpan dalam setiap dokumen meliputi nama barang, estimasi jumlah produksi, jumlah barang jadi, jumlah lembar material yang digunakan, harga per satuan, serta total harga.



Gambar 3.6. Struktur dokumen OrderBarang

C Struktur Dokumen OrderProgress

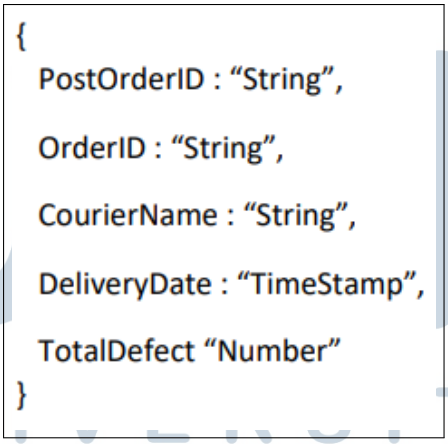
Gambar 3.7 menunjukkan struktur dokumen pada *collection* OrderProgress yang digunakan untuk mencatat perkembangan produksi harian. *Collection* ini menyimpan kumpulan dokumen OrderProgress, di mana setiap dokumen merepresentasikan satu catatan progres produksi yang terjadi pada tanggal tertentu. Setiap kali terdapat pembaruan progres produksi, data tersebut disimpan sebagai dokumen baru sehingga riwayat perkembangan produksi dapat terdokumentasi dengan baik.



Gambar 3.7. Struktur dokumen OrderProgress

D Struktur Dokumen PostOrder

Gambar 3.8 menunjukkan struktur dokumen pada *collection* PostOrder yang digunakan untuk menyimpan data terkait proses pengiriman barang setelah seluruh proses produksi selesai. *Collection* ini berisi kumpulan dokumen PostOrder, di mana setiap dokumen merepresentasikan satu catatan pengiriman. Informasi yang disimpan dalam dokumen tersebut meliputi nama kurir, tanggal pengiriman, serta jumlah barang cacat yang ditemukan selama proses produksi.

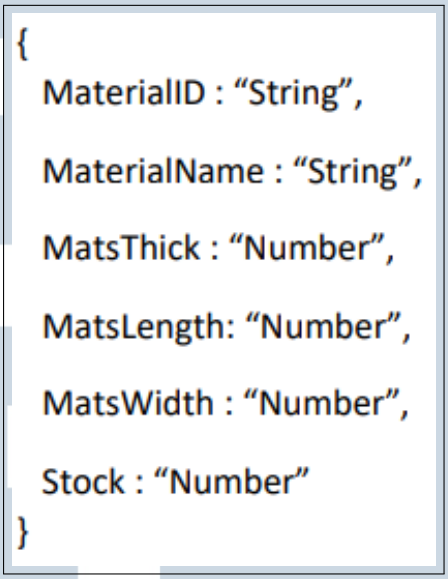


Gambar 3.8. Struktur dokumen postOrder

E Struktur Dokumen Material

Gambar 3.9 menunjukkan struktur dokumen pada *collection* Material yang digunakan untuk menyimpan seluruh informasi mengenai material yang dimiliki perusahaan. *Collection* ini berisi kumpulan dokumen Material, di mana setiap

dokumen merepresentasikan satu jenis material. Informasi yang disimpan dalam setiap dokumen meliputi identitas material, nama material, ketebalan, panjang, lebar, serta jumlah stok material yang tersedia.



Gambar 3.9. Struktur dokumen Material

3.4.4 Hasil Implementasi Aplikasi

Pada bagian ini akan ditampilkan hasil implementasi. Setiap tampilan yang disajikan akan diberikan penjelasan singkat mengenai kegunaannya.

A Halaman Dashboard

Gambar 3.10 menunjukkan tampilan halaman *Dashboard*. Halaman ini merupakan halaman utama yang pertama kali ditampilkan ketika aplikasi dijalankan. *Dashboard* berfungsi sebagai pusat informasi ringkas yang membantu pengguna melihat kondisi operasional secara cepat.

Gambar 3.10. Tampilan halaman dashboard

Pada bagian atas halaman terdapat tiga buah *card* informasi yang menampilkan data penting, yaitu total bahan material yang tersedia di gudang, jumlah pesanan yang sedang dikerjakan, serta jumlah pesanan yang telah selesai pada bulan berjalan. Informasi ini diambil dari basis data, lalu data yang di ambil di kelola, sehingga pengguna dapat memantau perkembangan produksi tanpa harus membuka menu lain.

Di bagian bawah *card* informasi tersebut terdapat dua *list* pesanan yang dipisahkan menjadi dua kategori, yaitu Pesanan yang Dikerjakan dan Pesanan Selesai. Setiap daftar menampilkan nama perusahaan pemesan beserta tanggal pesanan diterima. Untuk daftar pesanan selesai, data juga difilter berdasarkan pesanan yang telah benar-benar selesai pada bulan yang sama, sehingga informasi yang ditampilkan tetap akurat dan relevan.

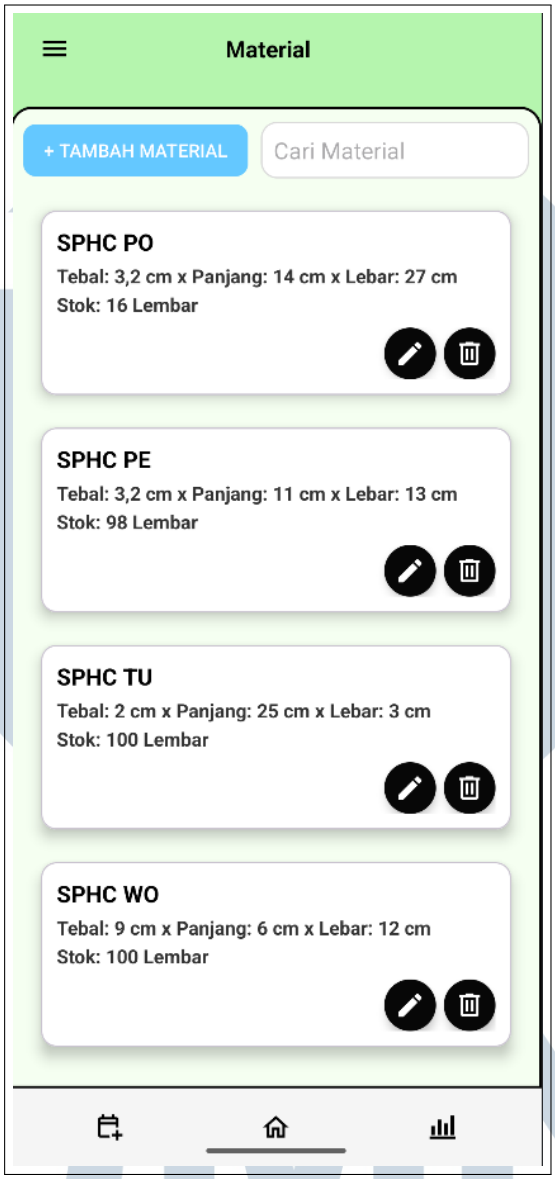
Setiap daftar pesanan juga dilengkapi tombol pada bagian kanan atas yang berfungsi membuka *bottom sheet* berisi tampilan daftar secara penuh. Fitur ini

memudahkan pengguna melihat seluruh data pesanan dengan tampilan yang lebih luas, tanpa mengganggu tampilan utama *dashboard*. Untuk tampilan *bottom sheet* bisa di lihat pada Gambar 3.11.

Gambar 3.11. Tampilan bottom sheet

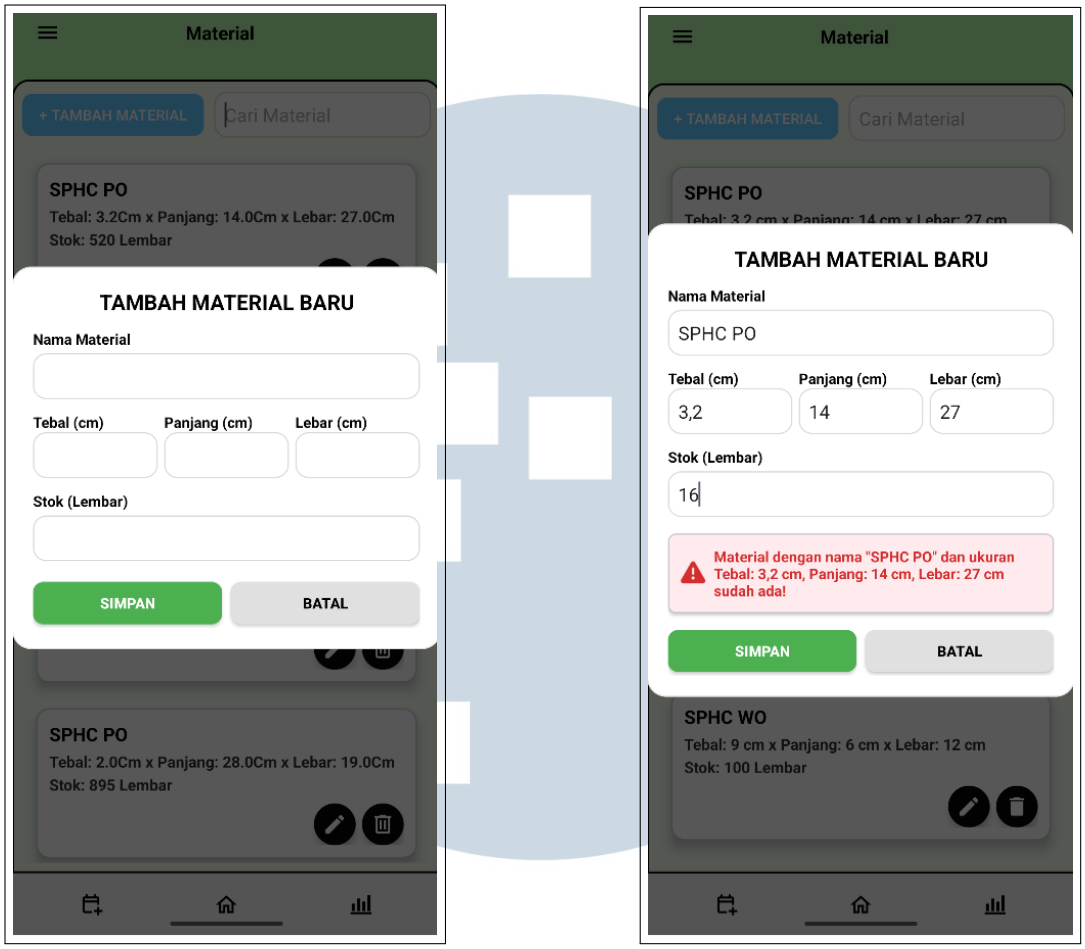
B Halaman Daftar Material

Halaman Daftar Material merupakan halaman yang digunakan untuk mengelola seluruh informasi material yang diperlukan dalam proses produksi. Tampilan utama halaman daftar material ditunjukkan pada Gambar 3.12, yang menampilkan susunan antarmuka mulai dari tombol aksi, kolom pencarian, hingga daftar material yang tersimpan di *database*.



Gambar 3.12. Tampilan halaman Daftar Material

Pada Gambar 3.12 bagian kiri atas halaman terdapat tombol "TAMBAH MATERIAL", yang berfungsi untuk menambahkan data material baru ke dalam *database*. Ketika tombol ini ditekan, sistem akan menampilkan *popup form* seperti yang terlihat pada Gambar 3.13a. Pada *form* tersebut, pengguna diminta mengisi beberapa data penting, yaitu Nama Material, Tebal, Panjang, Lebar, serta jumlah Stok material dalam satuan lembar. Jika pengguna memasukan Nama material dan dimensi material yang sudah ada pada *database* maka popup akan menampilkan *cardAlret* seperti pada Gambar 3.13b.

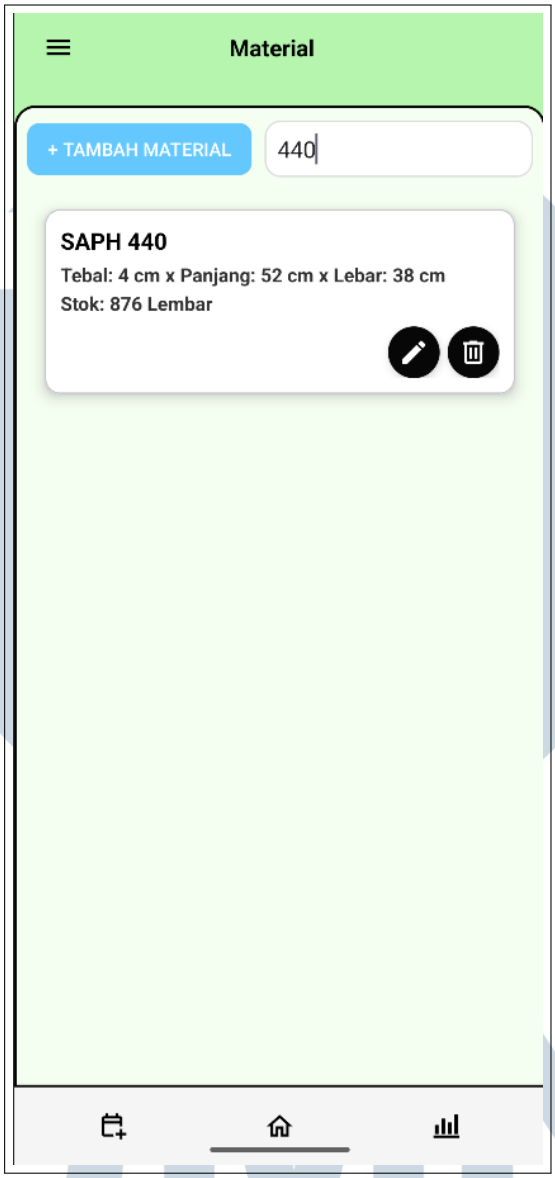


(a) Tampilan popup tambah material baru

(b) Tampilan popup tambah material dengan alert duplikasi

Gambar 3.13. Tampilan popup tambah material dan alert duplikasi

Pada Gambar 3.12 terdapat juga *search bar* yang berada pada sebelah kanan tombol tambah material, *search bar* ini memungkinkan pengguna menemukan material secara cepat berdasarkan nama material. Fitur pencarian ini sangat membantu ketika jumlah material yang tersimpan di sistem cukup banyak, sehingga pengguna tidak perlu menggulir daftar secara manual. Contoh tampilan *search bar* pada halaman Daftar Material dapat di lihat pada Gambar 3.14.

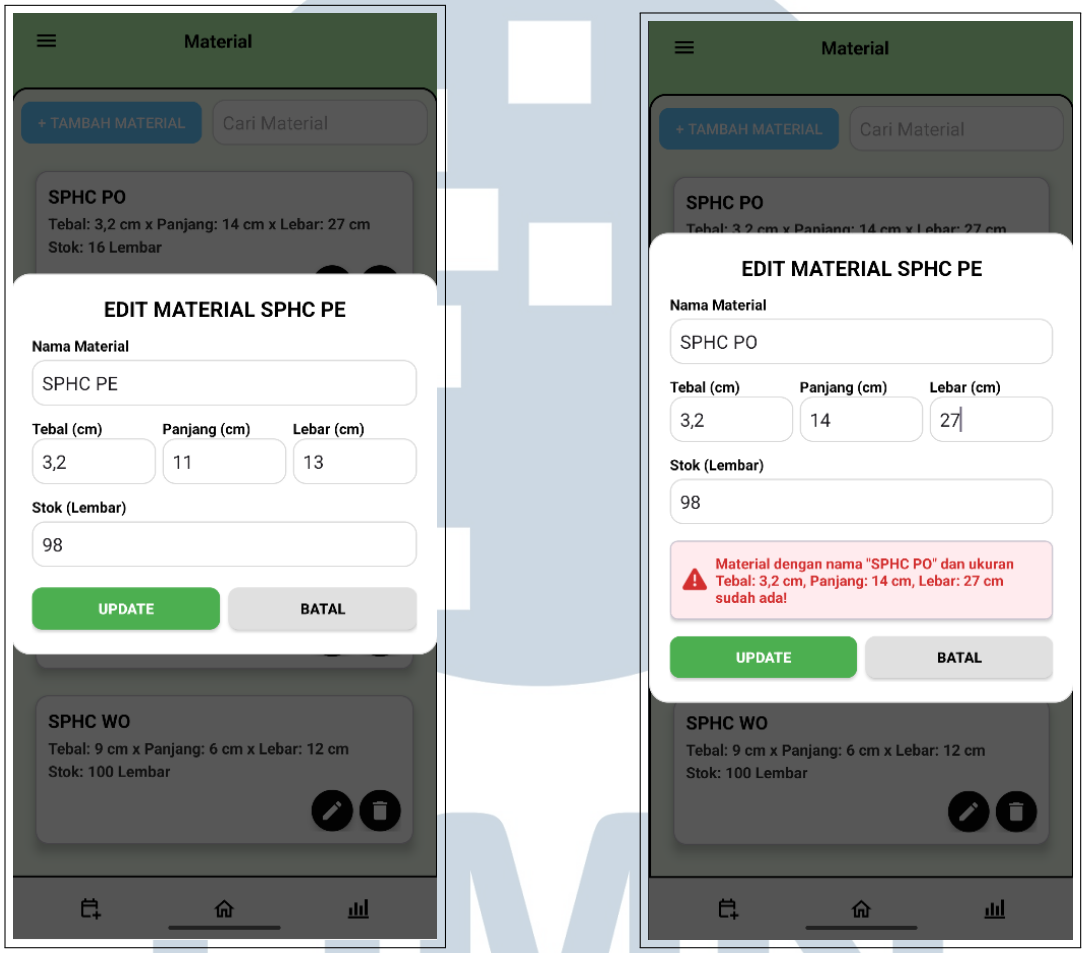


Gambar 3.14. Tampilan search bar

Di bawah kolom pencarian dan tombol tambah material, sebagaimana ditampilkan dalam Gambar 3.12, terdapat daftar material yang berbentuk *card*. Setiap *card* menampilkan informasi detail dari satu jenis material, seperti nama material, dimensi material (tebal, panjang, lebar), serta jumlah stok yang tersedia.

Pada bagian kanan bawah setiap *card* material, terdapat dua tombol, yaitu tombol *edit* dan *delete*. Ketika pengguna menekan tombol *edit*, sistem akan menampilkan *popup form edit* material seperti yang ditunjukkan pada Gambar 3.15a. Form ini menampilkan data material yang sudah ada dan memungkinkan pengguna untuk memperbarui informasi seperti dimensi material atau jumlah stok.

Sama seperti *popup* tambah material, jika pengguna melakukan *edit* material dan memasukan nama material dan dimensi material yang sudah ada di *database* maka akan muncul *cardAlert* seperti pada Gambar 3.15b.

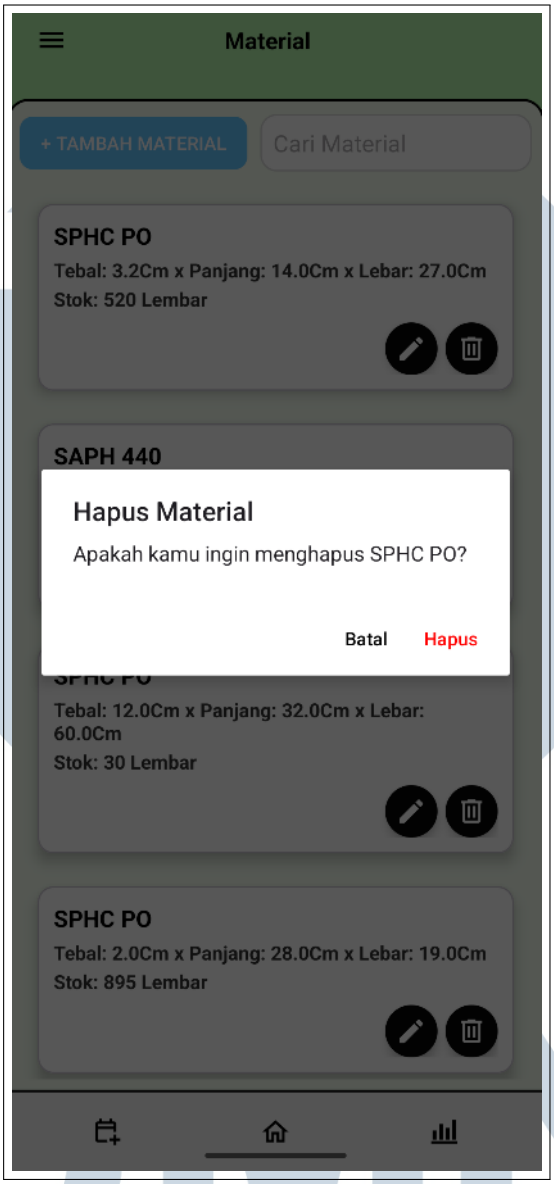


(a) Tampilan popup edit material tanpa alert duplikasi

(b) Tampilan popup edit material dengan alert duplikasi

Gambar 3.15. Tampilan popup edit material dan alert duplikasi

Sementara itu, jika pengguna menekan tombol *delete*, maka sistem akan menampilkan *popup* konfirmasi penghapusan seperti pada Gambar 3.16. *Popup* ini berfungsi memastikan bahwa pengguna benar-benar ingin menghapus material tersebut. Terdapat dua pilihan pada *popup* ini, yaitu “Batal” dan “Hapus”, sehingga pengguna tidak akan menghapus data secara tidak sengaja.

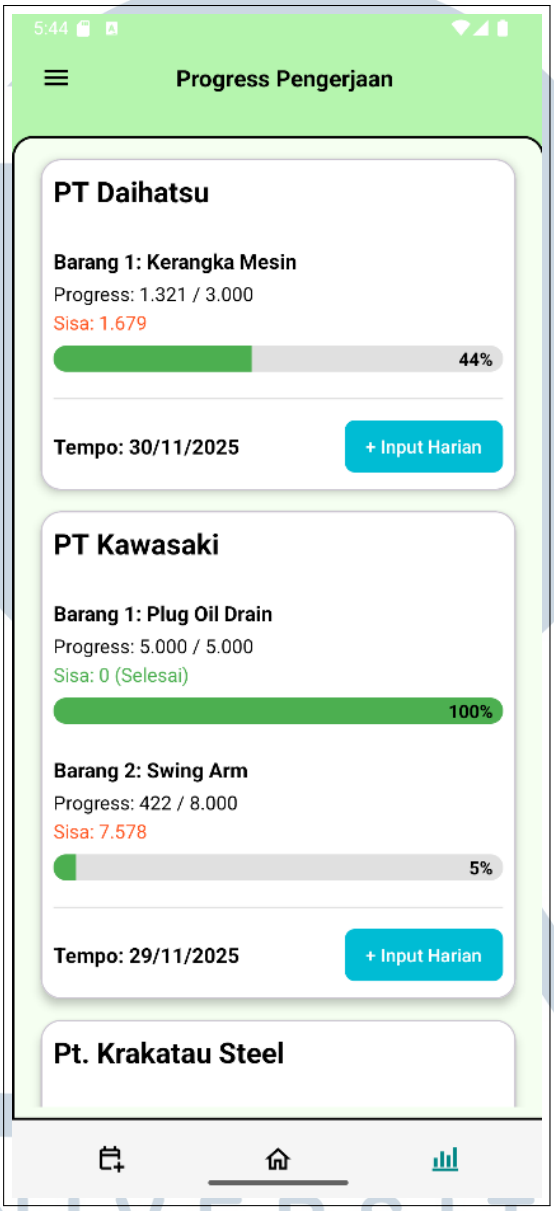


Gambar 3.16. Tampilan popup hapus material

C Halaman Progres Pengerjaan Pesanan

Pada halaman Progres Pengerjaan Pesanan, pengguna dapat melihat status produksi dari setiap pesanan yang berasal dari berbagai perusahaan . Tampilan utama halaman dapat dilihat pada Gambar 3.17, setiap pesanan ditampilkan dalam bentuk *card* yang memuat informasi nama perusahaan pemesan, daftar barang yang dipesan, jumlah produksi, sisa pekerjaan, serta persentase progres yang direpresentasikan melalui progress bar. Jika suatu perusahaan memesan lebih dari satu jenis barang, maka seluruh progres masing-masing barang akan ditampilkan

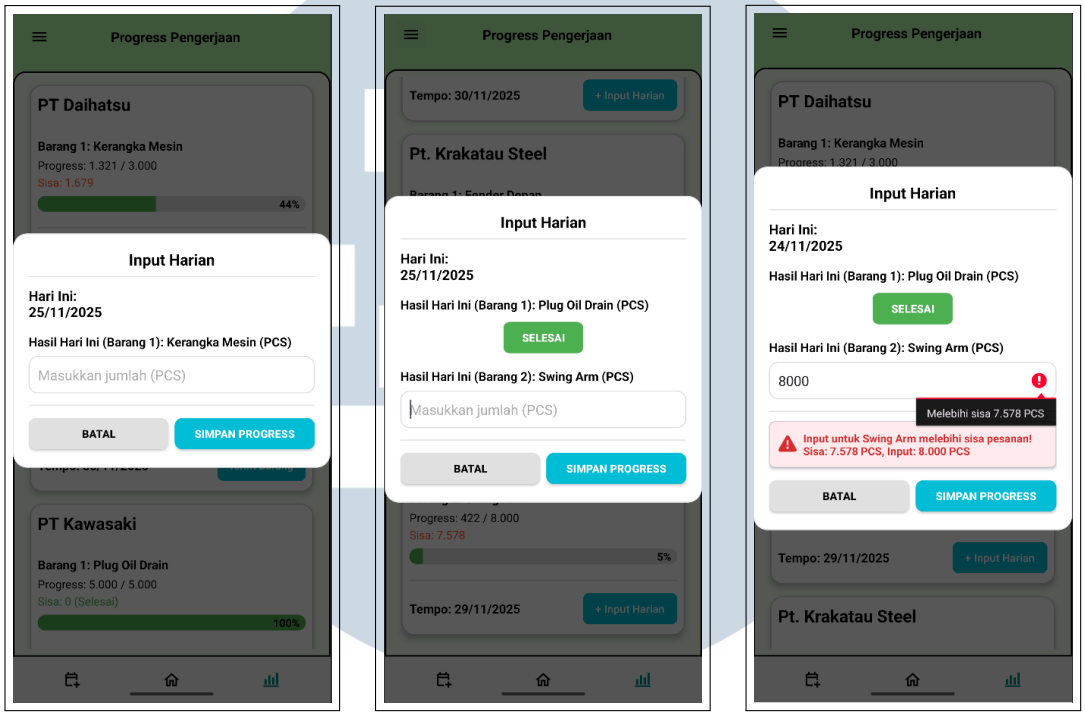
secara berurutan di dalam satu *card*, sehingga memudahkan pengguna dalam memantau perkembangan produksi secara menyeluruh.



Gambar 3.17. Tampilan halaman progres pengerjaan pesanan

Selanjutnya, ketika pengguna menekan tombol ”+ Input Harian” pada *card*, aplikasi akan menampilkan *popup form input* harian seperti pada Gambar 3.18a. Pada *popup* ini, pengguna dapat memasukkan jumlah hasil produksi harian untuk setiap barang yang dipesan, jika suatu PT memiliki pesanan lebih dari satu barang, tampilannya akan seperti pada Gambar 3.18b. Jika suatu barang telah selesai diproduksi, sistem akan menampilkan label “Selesai” sebagai penanda bahwa

barang tersebut tidak memerlukan input harian tambahan. Selain itu, aplikasi juga dilengkapi mekanisme validasi jika input harian melebihi total pesanan, maka akan muncul sebuah *alert* seperti pada Gambar 3.18c.



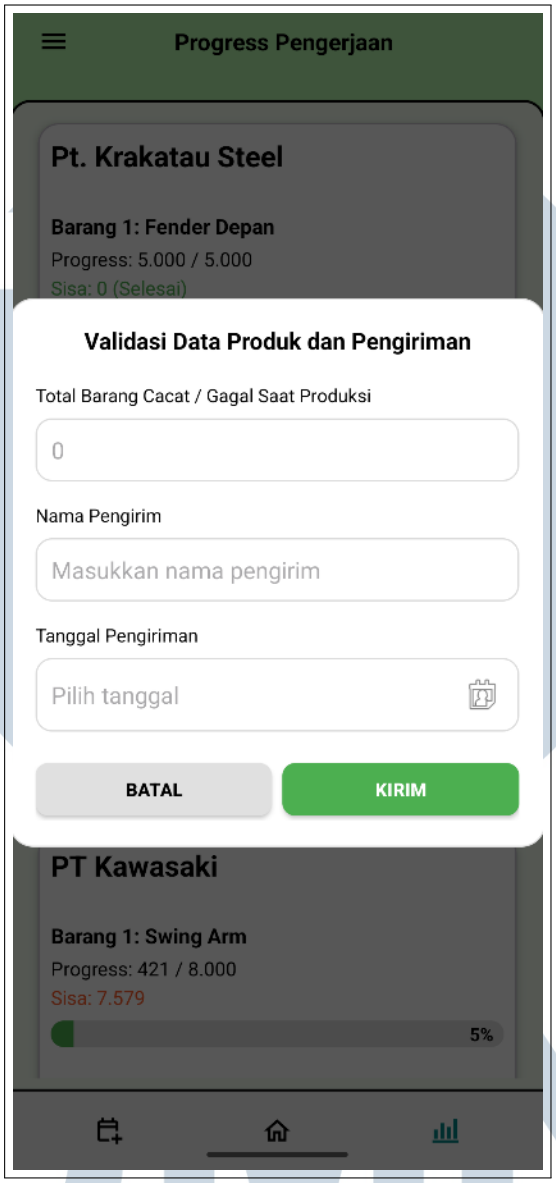
(a) Tampilan popup Input harian

(b) Tampilan popup input harian dengan dua barang

(c) Tampilan input harian dengan alret

Gambar 3.18. Tampilan popup input harian

Ketika seluruh pesanan suatu perusahaan telah selesai, tombol ”+ Input Harian” pada *card* secara otomatis berubah menjadi tombol ”Kirim Barang”. Saat tombol ini ditekan, sistem menampilkan *popup* validasi data pengiriman seperti ditunjukkan pada Gambar 3.19. Pada *popup* ini pengguna diminta untuk mengisi data seperti jumlah barang cacat, nama pengirim, serta tanggal pengiriman sebelum proses pengiriman dapat diproses.



Gambar 3.19. Tampilan popup validasi pengiriman

3.4.5 Potongan Kode Aplikasi

Pada bagian ini akan ditampilkan beberapa cuplikan kode yang digunakan pada aplikasi untuk menjalankan fungsinya.

A Potongan Kode Halaman Dashboard

Potongan kode yang digunakan untuk mengambil data pesanan pada halaman *Dashboard* dari *firestore* dapat di lihat pada Kode 3.1.

```
private fun fetchOrdersFromFirestore() {
        // Format bulan-tahun
        val currentMonthYear = SimpleDateFormat("MM-yyyy", Locale.
    getDefault()).format(Date())

        // Ambil data "Order" dari Firestore sort berdasarkan
    OrderDate terbaru
        firestore.collection("Order")
            .orderBy("OrderDate", Query.Direction.DESCENDING)
            .get()
            .addOnSuccessListener { result ->
                val ongoingOrders = mutableListOf<Order>()
                finishedOrders.clear()

                var finishedThisMonth = 0
                var processedOrders = 0
                val totalOrders = result.size()

                // Jika order kosong tampilan 0
                if (totalOrders == 0) {
                    binding.txtTotalPesanan.text = "0"
                    binding.txtTotalSelesai.text = "0"
                    return@addOnSuccessListener
                }

                // Loop tiap dokumen Order
                for (document in result) {

                    // Convert Firestore document menjadi object
    Order
                    val order = document.toObject(Order::class.
    java).apply {
                        OrderID = document.id
                    }

                    // ORDER YANG SUDAH SELESAI
                    if (order.FinishedOrder == true) {

                        // Ambil subkoleksi "PostOrder" untuk
    mencari DeliveryDate
                        firestore.collection("Order")
                            .document(order.OrderID!!)
                            .collection("PostOrder")
                            .limit(1)
                            .get()
                            .addOnSuccessListener { postOrderDocs
    ->
                                // PostOrder ada
                                if (!postOrderDocs.isEmpty) {
                                    val deliveryDate =
    postOrderDocs.documents[0].getTimestamp("DeliveryDate")
                                    val deliveryMonthYear =
    deliveryDate?.toDate()?.let {
                                        SimpleDateFormat("MM-yyyy"
    , Locale.getDefault()).format(it)
                                    }

                                    // Jika bulan pengiriman ==
    bulan saat ini  hitung sebagai selesai bulan ini
```

```
                                    if (deliveryMonthYear ==
currentMonthYear) {
                                        finishedThisMonth++
                                        finishedOrders.add(order)
                                    }
                                }

                                // Tambahkan jumlah order yang
telah diproses
                                processedOrders++

                                // Jika semua order selesai
diproses maka update UI
                                if (processedOrders == totalOrders
) {
                                    binding.txtTotalPesanan.text =
 ongoingOrders.size.toString()
                                    binding.txtTotalSelesai.text =
 finishedThisMonth.toString()
                                    finishedAdapter.submitList(
finishedOrders)
                                }
                            }
                            .addOnFailureListener {
                                // Jika gagal get PostOrder
                                processedOrders++
                                if (processedOrders == totalOrders
) {
                                    binding.txtTotalPesanan.text =
 ongoingOrders.size.toString()
                                    binding.txtTotalSelesai.text =
 finishedThisMonth.toString()
                                    finishedAdapter.submitList(
finishedOrders)
                                }
                            }
                    } else {
                        // ORDER YANG BELUM SELESAI
                        ongoingOrders.add(order)
                        processedOrders++

                        // Jika semua order selesai diproses maka
update UI
                        if (processedOrders == totalOrders) {
                            binding.txtTotalPesanan.text =
ongoingOrders.size.toString()
                            binding.txtTotalSelesai.text =
finishedThisMonth.toString()
                            finishedAdapter.submitList(
finishedOrders)
                        }
                    }
                }

                // Update adapter untuk list pesanan ongoing
                ongoingAdapter.submitList(ongoingOrders)
            }
    }
```

Kode 3.1: Fungsi fetchOrdersFromFirestore

Kode 3.1 merupakan fungsi *fetchOrdersFromFirestore()* yang digunakan untuk mengambil data pesanan dari *Firestore* dan menampilkannya pada halaman *dashboard*. Fungsi ini menampilkan dua jenis informasi utama, yaitu jumlah pesanan yang masih dikerjakan dan jumlah pesanan yang telah selesai pada bulan berjalan.

Pertama, aplikasi mengambil waktu saat ini menggunakan *SimpleDateFormat*("MM-yyyy"), kemudian mencocokkannya dengan tanggal pengiriman pesanan untuk menentukan apakah pesanan selesai pada bulan yang sama. Selanjutnya, aplikasi mengambil seluruh data dari koleksi *Order* di *Firestore* dan mengurutkannya berdasarkan *OrderDate* secara *descending*.

Setiap dokumen *Order* yang diambil dikonversi menjadi objek *Order* dan diperiksa statusnya. Jika *FinishedOrder == true*, maka sistem melakukan *query* tambahan ke sub-koleksi *PostOrder* untuk mengambil data tanggal pengiriman terakhir. Tanggal pengiriman tersebut diubah ke format MM-yyyy dan dibandingkan dengan tanggal bulan berjalan. Jika sesuai, maka data dimasukkan ke daftar *finishedOrders* dan penghitung *finishedThisMonth* ditambah satu. Sebaliknya, jika pesanan belum selesai, maka data dimasukkan ke dalam *ongoingOrders*.

Sistem juga menggunakan variabel *processedOrders* untuk memastikan bahwa proses pembacaan seluruh dokumen *Firestore* selesai, terutama karena setiap dokumen dapat memicu proses *asynchronous* terpisah. Setelah seluruh data selesai diproses, dashboard diperbarui dengan menampilkan jumlah pesanan berjalan, jumlah pesanan selesai bulan ini, dan daftar pesanan melalui *adapter RecyclerView*.

B Potongan Kode Halaman Daftar Material

Pada halaman Daftar Material, aplikasi harus menampilkan seluruh data material yang tersimpan di sistem secara akurat dan selalu dalam kondisi terbaru. Untuk memastikan data yang ditampilkan selalu sinkron dengan *database*, aplikasi menggunakan fungsi *fetchMaterials()*. Kode untuk *fetchMaterials* dapat dilihat pada Kode 3.2.

```
private fun fetchMaterials() {
        // Ambil seluruh data dari koleksi "Material" di Firestore
        firestore.collection("Material")
            .get()
            .addOnSuccessListener { result ->
                // Kosongkan list
                materialList.clear()
                for (doc in result) {
```

```
                    // Convert Firestore document menjadi object
    Material
                    val material = doc.toObject(Material::class.
    java)
                    material.MaterialID = doc.id
                    materialList.add(material)
                }
                // Update adapter
                adapter.updateList(materialList)
            }
            .addOnFailureListener {
                Toast.makeText(this, "Gagal memuat data", Toast.
    LENGTH_SHORT).show()
            }
     }
```

Kode 3.2: Fungsi fetchMaterials

Kode 3.2 merupakan fungsi *fetchMaterials()* yang digunakan untuk mengambil dan memuat seluruh data material dari koleksi Material yang tersimpan di *Firestore*. Fungsi ini dipanggil ketika aplikasi perlu menampilkan daftar material terbaru, misalnya setelah pengguna menambah material baru atau memperbarui data yang sudah ada.

Proses dimulai dengan memanggil koleksi ”Material” dari *Firestore* menggunakan metode *.get()*. Apabila proses pengambilan data berhasil, *callback addOnSuccessListener* akan dijalankan. Pada tahap ini, *materialList* dibersihkan terlebih dahulu untuk memastikan tidak ada data ganda. Kemudian, setiap dokumen pada hasil query dikonversi menjadi objek Material menggunakan *toObject(Material::class.java)*. ID dokumen *Firestore (doc.id)* juga dimasukkan ke dalam atribut *MaterialID* agar dapat digunakan untuk proses perubahan data selanjutnya, seperti *edit* atau *delete*. Setiap objek material yang telah diproses kemudian ditambahkan ke dalam *materialList*.

Setelah seluruh data berhasil dimuat, fungsi memanggil *adapter.updateList(materialList)* untuk memperbarui tampilan daftar di *RecyclerView* sehingga pengguna dapat melihat data terbaru di tampilan aplikasi. Apabila proses pengambilan data gagal, *callback addOnFailureListener* akan menampilkan pesan kesalahan melalui *Toast*.

Potongan Kode 3.3 merupakan cara kerja fungsi dalam menambah material baru serta bagaimana sistem melakukan pemeriksaan kelengkapan input, konversi data, pengecekan duplikasi, hingga mengirimkan data ke *Firestore*.

```
private fun showAddDialog() {
        // Format angka menggunakan Indonesia
        val numberFormat = NumberFormat.getNumberInstance(Locale("
    id", "ID"))

```

```
        val dialog = Dialog(this)
        val dialogBinding = PopupAddMaterialBinding.inflate(
    layoutInflater)
        dialog.setContentView(dialogBinding.root)
        dialog.window?.setBackgroundDrawable(ColorDrawable(Color.
    TRANSPARENT))
        dialog.window?.setLayout(
            ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT
        )

        // Mengatur judul popup
        dialogBinding.tvTitle.text = "TAMBAH MATERIAL BARU"
        // Hide alert card di awal
        dialogBinding.cardAlert.visibility = View.GONE

        dialogBinding.btnSaveMaterial.setOnClickListener {
            dialogBinding.cardAlert.visibility = View.GONE

            val name = dialogBinding.etMaterialName.text.toString
    ().trim()

            // Parse nilai angka menggunakan sesuai format
    Indonesia
            val thick = numberFormat.parse(dialogBinding.
    etMatsThick.text.toString())?.toDouble()
            val length = numberFormat.parse(dialogBinding.
    etMatsLength.text.toString())?.toDouble()
            val width = numberFormat.parse(dialogBinding.
    etMatsWidth.text.toString())?.toDouble()
            val stock = numberFormat.parse(dialogBinding.etStock.
    text.toString())?.toInt() ?: 0

            if (name.isEmpty() || thick == null || length == null
    || width == null) {
                Toast.makeText(this, "Semua field harus diisi
    dengan angka yang valid", Toast.LENGTH_SHORT).show()
                return@setOnClickListener
            }

            // Cek apakah material duplikasi material
            if (checkDuplicateMaterial(name, thick, length, width)
    ) {
                dialogBinding.cardAlert.visibility = View.VISIBLE
                dialogBinding.tvAlertMessage.text =
                    "Material dengan nama \"$name\" dan ukuran " +
                            "Tebal: ${numberFormat.format(thick)}
    cm, " +
                            "Panjang: ${numberFormat.format(length
    )} cm, " +
                            "Lebar: ${numberFormat.format(width)}
    cm sudah ada!"
                return@setOnClickListener
            }

            // Data material yang akan disimpan ke Firestore
            val data = hashMapOf(
                "MaterialName" to name,
                "MatsThick" to thick,
```

```
                "MatsLength" to length,
                "MatsWidth" to width,
                "Stock" to stock
            )

            // Simpan data ke Firestore
            firestore.collection("Material")
                .add(data)
                .addOnSuccessListener {
                    Toast.makeText(this, "Material berhasil
ditambahkan", Toast.LENGTH_SHORT).show()
                    dialog.dismiss()
                    fetchMaterials()
                }
                .addOnFailureListener {
                    Toast.makeText(this, "Gagal menambah material"
, Toast.LENGTH_SHORT).show()
                }
        }

        dialogBinding.btnCancelMaterial.setOnClickListener {
            dialog.dismiss()
        }

        dialog.show()
    }
```

Kode 3.3: Fungsi showAddDialog

Kode 3.3 merupakan Fungsi *showAddDialog()* digunakan untuk menampilkan dialog yang memungkinkan pengguna menambahkan data material baru ke dalam *database Firestore*. Pada awal fungsi, dibuat format angka Indonesia agar nilai numerik seperti ketebalan, panjang, lebar, dan stok material ditampilkan dan diolah sesuai format indonesia. Selanjutnya, dialog dibuat menggunakan *Dialog(this)* dan layoutnya *di-inflate* melalui *PopupAddMaterialBinding*. Properti dialog diatur agar latar belakang transparan dan ukuran dialog menyesuaikan konten. Judul dialog diatur menjadi ”TAMBAH MATERIAL BARU”, dan *cardAlert* disembunyikan di awal.

Ketika tombol simpan ditekan, fungsi melakukan validasi input. Semua field numerik diubah menjadi tipe data *Double* atau *Int* menggunakan *numberFormat*, sementara nama material diambil sebagai *string*. Fungsi memastikan semua field terisi dan bernilai valid jika tidak, sistem menampilkan *Toast* sebagai peringatan. Fungsi juga memeriksa duplikasi material dengan memanggil *checkDuplicateMaterial()*. Jika material dengan nama dan ukuran yang sama sudah ada, *cardAlert* ditampilkan dengan pesan kesalahan.

Apabila semua validasi lolos, data material dikemas ke dalam *HashMap* dan disimpan ke koleksi Material di *Firestore*. Jika penyimpanan berhasil, sistem menampilkan *Toast* berhasil, menutup dialog, lalu memanggil *fetchMaterials()*

untuk memperbarui daftar material. Jika gagal, Toast ditampilkan.

Selain Menambahkan material baru, ada juga fungsi untuk melakukan edit informasi material, fungsi yang digunakan adalah *showEditDialog()*, Kode fungsi showEditDialog bisa dilihat pada Kode 3.4.

```
private fun showEditDialog(material: Material) {
        // Format angka menggunakan Indonesia
        val numberFormat = NumberFormat.getNumberInstance(Locale("id", "ID"))

        val dialog = Dialog(this)
        val dialogBinding = PopupAddMaterialBinding.inflate(layoutInflater)
        dialog.setContentView(dialogBinding.root)
        dialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))
        dialog.window?.setLayout(
            ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.WRAP_CONTENT
        )

        // Mengatur judul popup
        dialogBinding.tvTitle.text = "EDIT MATERIAL ${material.MaterialName}"
        dialogBinding.btnSaveMaterial.text = "UPDATE"
        // Hide alert card di awal
        dialogBinding.cardAlert.visibility = View.GONE

        // Mengisi input dengan data material yang sudah ada
        dialogBinding.etMaterialName.setText(material.MaterialName ?: "")
        dialogBinding.etMatsThick.setText(material.MatsThick?.let { numberFormat.format(it) } ?: "")
        dialogBinding.etMatsLength.setText(material.MatsLength?.let { numberFormat.format(it) } ?: "")
        dialogBinding.etMatsWidth.setText(material.MatsWidth?.let { numberFormat.format(it) } ?: "")
        dialogBinding.etStock.setText(material.Stock?.let { numberFormat.format(it) } ?: "")

        dialogBinding.btnSaveMaterial.setOnClickListener {
            dialogBinding.cardAlert.visibility = View.GONE

            val name = dialogBinding.etMaterialName.text.toString().trim()

            // Parse nilai angka menggunakan sesuai format Indonesia
            val thick = numberFormat.parse(dialogBinding.etMatsThick.text.toString())?.toDouble()
            val length = numberFormat.parse(dialogBinding.etMatsLength.text.toString())?.toDouble()
            val width = numberFormat.parse(dialogBinding.etMatsWidth.text.toString())?.toDouble()
            val stock = numberFormat.parse(dialogBinding.etStock.text.toString())?.toInt() ?: 0

            if (name.isEmpty() || thick == null || length == null
```

```
    || width == null) {
                Toast.makeText(this, "Semua field harus diisi
dengan angka yang valid", Toast.LENGTH_SHORT).show()
                return@setOnClickListener
            }

            // Cek apakah material duplikasi material
            if (checkDuplicateMaterial(name, thick, length, width,
 material.MaterialID)) {
                dialogBinding.cardAlert.visibility = View.VISIBLE
                dialogBinding.tvAlertMessage.text =
                    "Material dengan nama \"$name\" dan ukuran " +
                            "Tebal: ${numberFormat.format(thick)}
cm, " +
                            "Panjang: ${numberFormat.format(length
)} cm, " +
                            "Lebar: ${numberFormat.format(width)}
cm sudah ada!"
                return@setOnClickListener
            }

            // Data update yang akan disimpan ke Firestore
            val updates = mapOf(
                "MaterialName" to name,
                "MatsThick" to thick,
                "MatsLength" to length,
                "MatsWidth" to width,
                "Stock" to stock
            )

            // Update data material berdasarkan ID dokumen
            firestore.collection("Material")
                .document(material.MaterialID!!)
                .update(updates)
                .addOnSuccessListener {
                    Toast.makeText(this, "Material berhasil
diupdate", Toast.LENGTH_SHORT).show()
                    dialog.dismiss()
                    fetchMaterials()
                }
                .addOnFailureListener {
                    Toast.makeText(this, "Gagal update material",
Toast.LENGTH_SHORT).show()
                }
        }

        dialogBinding.btnCancelMaterial.setOnClickListener {
            dialog.dismiss()
        }

        dialog.show()
    }
```


Kode 3.4: Fungsi ShowEditDialog

Kode 3.4 merupakan fungsi *showEditDialog()* yang berguna untuk menampilkan dialog khusus supaya pengguna melakukan pengeditan data material yang sudah ada dalam sistem. Dialog ini memiliki tampilan dan struktur yang sama

seperti dialog penambahan material, namun isinya sudah otomatis terisi dengan data material yang dipilih. Hal ini memudahkan pengguna dalam melakukan perubahan tanpa harus mengisi ulang seluruh data dari awal.

Pada bagian awal fungsi, sistem menyiapkan *number format* Indonesia untuk menangani input berupa angka seperti tebal, panjang, lebar, dan stok material. Kemudian, dialog diinisialisasi menggunakan *layout PopupAddMaterialBinding*, dan judul dialog diubah menjadi "*EDIT MATERIAL*" untuk menandakan bahwa dialog ini merupakan *mode edit*, bukan penambahan. Tombol simpan juga diubah teksnya menjadi "*UPDATE*".

Selanjutnya, seluruh kolom input dalam dialog diisi dengan nilai lama yang berasal dari objek Material. Proses ini dilakukan agar pengguna dapat melihat nilai sebelumnya dan cukup memperbarui bagian yang perlu diubah saja. Setiap nilai numerik diformat menggunakan *numberFormat* sehingga tampil sesuai standar penulisan angka di Indonesia.

Ketika pengguna menekan tombol *UPDATE*, sistem pertama-tama mengambil nilai input dari setiap kolom dan melakukan validasi dasar, seperti memastikan bahwa nama material dan ukuran memiliki nilai yang valid. Jika terdapat input yang tidak valid, sistem akan menampilkan pesan peringatan. Selain itu, terdapat logika untuk mencegah duplikasi material melalui fungsi *checkDuplicateMaterial()*. Apabila material dengan kombinasi nama dan ukuran yang sama sudah ada, dialog menampilkan peringatan melalui *cardAlert*.

Jika semua data valid, fungsi membuat *map* bernama *updates* yang berisi *field-field* yang akan diperbarui. Data ini kemudian dikirim ke *Firestore* menggunakan perintah *.update()* pada dokumen material sesuai *MaterialID*. Setelah pembaruan berhasil, dialog ditutup dan fungsi *fetchMaterials()* dipanggil untuk memuat ulang daftar material pada tampilan sehingga perubahan langsung terlihat oleh pengguna.

Supaya tidak terjadi duplikasi pada saat menambahkan material baru atau melakukan *edit* data material, digunakan fungsi *checkDuplicateMaterial*, fungsi ini dapat dilihat pada Kode 3.5.

```
private fun checkDuplicateMaterial(
    name: String,
    thick: Double,
    length: Double,
    width: Double,
    excludeId: String? = null
): Boolean {
    return materialList.any { material ->
    // excludeId supaya yang sedang diedit tidak di flag duplikat
        material.MaterialID != excludeId &&
```

```
        material.MaterialName.equals(name, ignoreCase = true) &&
        material.MatsThick == thick &&
        material.MatsLength == length &&
        material.MatsWidth == width
    }
}
```

Kode 3.5: Fungsi checkDuplicateMaterial

Kode 3.5 merupakan fungsi *checkDuplicateMaterial* yang digunakan untuk memeriksa apakah data material yang akan ditambahkan atau *edit* sudah ada dalam daftar material. Fungsi ini menerima beberapa *parameter* yaitu *name, thick, length,* dan *width* sebagai identitas unik sebuah material, serta *parameter excludeId* yang digunakan agar saat di periksa tidak membandingkan data dengan dirinya sendiri. Di dalam fungsi, dilakukan pengecekan menggunakan *any* terhadap *materialList*, yang akan menghasilkan *true* jika ditemukan material lain dengan nama yang sama serta ukuran ketebalan, panjang, dan lebar yang sama.

C Potongan Kode Halaman Progres Pengerjaan Pesanan

Sebelum pengguna melakukan input progres harian, sistem akan mengambil dan mengelompokkan data progres produksi dari *Firestore* berdasarkan nama perusahaan. Fungsi utama yang berperan dalam proses tersebut adalah *loadProgress()*, fungsi ini dapat di lihat pada Kode 3.6.

```
private fun loadProgress() {
        // Filter order yang belum selesai
        firestore.collection("Order")
            .whereEqualTo("FinishedOrder", false)
            .get()
            .addOnSuccessListener { orderDocs ->

                // Map untuk mengelompokkan berdasarkan PT
                val ptOrderMap = mutableMapOf<String, MutableList<
    OrderProgressModel>>()
                val ptTempoMap = mutableMapOf<String, String>()
                val ptOrderIDMap = mutableMapOf<String, String>()

                var processedOrders = 0
                val totalOrders = orderDocs.size()

                if (totalOrders == 0) {
                    convertMapToList(ptOrderMap, ptTempoMap,
    ptOrderIDMap)
                    return@addOnSuccessListener
                }

                for (order in orderDocs) {
                    val orderID = order.id
                    val ptName = order.getString("PTName") ?: "PT
    Tidak Diketahui"
```

```
24
25                      // Ambil tanggal tempo
26                      val dueDate = order.getTimestamp("DueDate")
27                      val dateFormat = SimpleDateFormat("dd/MM/yyyy"
    , Locale("id", "ID"))
28
29                      // Format tanggal
30                      val tempo = if (dueDate != null) {
31                          dateFormat.format(dueDate.toDate())
32                      } else {
33                          // jika null
34                          "-"
35                      }
36
37                      // Simpan tempo dan ID ke map sesuai PT
38                      ptTempoMap[ptName] = tempo
39                      ptOrderIDMap[ptName] = orderID
40
41                      // Ambil data "OrderBarang" untuk menghitung
    total item per order
42                      firestore.collection("Order")
43                          .document(orderID)
44                          .collection("OrderBarang")
45                          .get()
46                          .addOnSuccessListener { barangDocs ->
47
48                              var processedBarang = 0
49                              val totalBarang = barangDocs.size()
50
51                              if (totalBarang == 0) {
52                                  processedOrders++
53                                  if (processedOrders == totalOrders
    ) {
54                                      convertMapToList(ptOrderMap,
    ptTempoMap, ptOrderIDMap)
55                                  }
56                                  return@addOnSuccessListener
57                              }
58
59                              for (barang in barangDocs) {
60                                  val orderName = barang.getString("
    OrderName") ?: "-"
61                                  val total = barang.getLong("
    EstimateOrder")?.toInt() ?: 0
62
63                                  // Ambil progress pengerjaan dari
    subkoleksi "OrderProgress"
64                                  firestore.collection("Order")
65                                      .document(orderID)
66                                      .collection("OrderBarang")
67                                      .document(barang.id)
68                                      .collection("OrderProgress")
69                                      .get()
70                                      .addOnSuccessListener {
    progDocs ->
71
72                                          // Hitung total progress
    yang sudah dikerjakan
```

```
                                        val done = progDocs.sumOf {
                                            it.getLong("CurrentProgress")?.toInt() ?: 0
                                        }

                                        if (!ptOrderMap.containsKey(ptName)) {
                                            ptOrderMap[ptName] = mutableListOf()
                                        }
                                        // Tambahkan progress barang ke dalam map berdasarkan PT
                                        ptOrderMap[ptName]?.add(OrderProgressModel(orderName, total, done))

                                        processedBarang++
                                        if (processedBarang == totalBarang) {
                                            processedOrders++
                                            if (processedOrders == totalOrders) {
                                                convertMapToList(ptOrderMap, ptTempoMap, ptOrderIDMap)
                                            }
                                        }
                                    }
                                    .addOnFailureListener {
                                        processedBarang++
                                        if (processedBarang == totalBarang) {
                                            processedOrders++
                                            if (processedOrders == totalOrders) {
                                                convertMapToList(ptOrderMap, ptTempoMap, ptOrderIDMap)
                                            }
                                        }
                                    }
                            }
                        }
                        .addOnFailureListener {
                            // Jika gagal mengambil OrderBarang tetap lanjutkan proses order berikutnya
                            processedOrders++
                            if (processedOrders == totalOrders) {
                                convertMapToList(ptOrderMap, ptTempoMap, ptOrderIDMap)
                            }
                        }
                }
            }
            .addOnFailureListener {
                Toast.makeText(this, "Gagal memuat data order", Toast.LENGTH_SHORT).show()
            }
    }
```

Kode 3.6: Fungsi LoadProgress

Kode 3.6 merupakan Fungsi *loadProgress()* digunakan untuk mengambil, mengelompokkan, dan memproses data progres pesanan dari *Firestore*. Fungsi ini bekerja dengan beberapa tahap, mulai dari memuat data *order* yang masih aktif, mengambil daftar barang yang dipesan, hingga menghitung total progres yang telah dicapai pada setiap barang. Hasil akhir dari fungsi ini akan dikirimkan ke fungsi *convertMapToList()* untuk diubah menjadi bentuk *list* yang siap ditampilkan pada antarmuka aplikasi.

Pada tahap pertama, fungsi memfilter data *order* berdasarkan *field FinishedOrder = false*, yang berarti hanya pesanan yang belum selesai diproses akan ditampilkan. Setelah data berhasil diambil, fungsi menyiapkan tiga struktur data:

1. *ptOrderMap* untuk menyimpan progres setiap barang yang dikelompokkan berdasarkan nama perusahaan.
2. *ptTempoMap* untuk menyimpan tanggal jatuh tempo untuk setiap Perusahaan
3. *ptOrderIDMap* untuk menyimpan *ID order* untuk PT tersebut.

Untuk setiap dokumen *order*, fungsi mengambil informasi seperti nama PT, *ID order*, serta tanggal tempo. Setelah itu, fungsi membaca sub-koleksi *OrderBarang* yang berisi daftar barang yang termasuk dalam pesanan tersebut. Pada masing-masing barang, fungsi mendapatkan nama barang dan jumlah pesanan, lalu mengambil sub-koleksi *OrderProgress* untuk menghitung total progres yang telah dicapai dengan menjumlahkan seluruh progres harian.

Setiap hasil perhitungan dimasukkan ke dalam *ptOrderMap* dalam bentuk objek *OrderProgressModel*, yang menyimpan nama barang, jumlah total pesanan, dan jumlah progres yang telah selesai. Karena setiap pengambilan data *Firestore* bersifat asinkron, fungsi menggunakan penghitung *processedOrders* dan *processedBarang* untuk memastikan seluruh data telah sepenuhnya diproses sebelum memanggil *convertMapToList()*.

Apabila semua data *order* dan progres telah berhasil dimuat, fungsi memanggil *convertMapToList()* untuk mengonversi map menjadi struktur *list* yang akan digunakan oleh *adapter* tampilan. Jika terjadi kegagalan pada salah satu proses pengambilan data, aplikasi akan menampilkan pesan kesalahan melalui *Toast*.

Setelah proses pemuatan dan pengelompokan data progres pesanan dijelaskan melalui fungsi *loadProgress()*, langkah berikutnya adalah menyiapkan data barang yang akan digunakan pada proses input progres harian. Kode 3.7

merupakan fungsi *loadBarangDataForInput* yang digunakan untuk memuat data barang dari *Firestore* sebelum ditampilkan pada *form input* harian.

```
private fun loadBarangDataForInput(
        orderID: String,
        dialogBinding: DialogInputHarianBinding,
        dialog: Dialog
    ) {
        // Mengambil semua data barang
        firestore.collection("Order")
            .document(orderID)
            .collection("OrderBarang")
            .get()
            .addOnSuccessListener { barangDocs ->

                // List untuk data input harian setiap barang
                val inputItems = mutableListOf<InputHarianItem>()
                var processedCount = 0

                // Jika tidak ada barang
                if (barangDocs.isEmpty) {
                    Toast.makeText(this, "Tidak ada barang untuk diinput", Toast.LENGTH_SHORT).show()
                    return@addOnSuccessListener
                }

                // Loop setiap dokumen barang
                for (barang in barangDocs) {
                    val barangID = barang.id
                    val barangName = barang.getString("OrderName") ?: "-"
                    val estimateOrder = barang.getLong("EstimateOrder")?.toInt() ?: 0

                    // Ambil progres harian dari subkoleksi "OrderProgress"
                    firestore.collection("Order")
                        .document(orderID)
                        .collection("OrderBarang")
                        .document(barangID)
                        .collection("OrderProgress")
                        .get()
                        .addOnSuccessListener { progDocs ->

                            // Hitung seluruh progress sebelumnya
                            val currentProgress = progDocs.sumOf {
                                it.getLong("CurrentProgress")?.toInt() ?: 0
                            }

                            // Tambahkan item ke list InputHarian
                            inputItems.add(
                                InputHarianItem(
                                    barangID = barangID,
                                    barangName = barangName,
                                    currentProgress = currentProgress,
                                    estimateOrder = estimateOrder
                                )
```

```
                                )

                                processedCount++
                                if (processedCount == barangDocs.size
    ()) {
                                    // Semua data diload tampilkan
    form
                                    displayInputForm(dialogBinding,
    inputItems, dialog, orderID)
                                }
                            }
                            .addOnFailureListener {
                                processedCount++
                                // Tampilkan form jika semua dokumen
    sudah diproses
                                if (processedCount == barangDocs.size
    ()) {
                                    displayInputForm(dialogBinding,
    inputItems, dialog, orderID)
                                }
                            }
                    }
                }
                .addOnFailureListener {
                    Toast.makeText(this, "Gagal memuat data barang",
    Toast.LENGTH_SHORT).show()
                }
    }
```

Kode 3.7: Fungsi loadBarangDataForInput

Kode 3.7 merupakan fungsi yang untuk menyiapkan data barang yang dibutuhkan untuk proses input progres harian sebelum ditampilkan kepada pengguna. Fungsi *loadBarangDataForInput()* pertama-tama melakukan pengambilan data dari *Firestore* pada koleksi *OrderBarang* berdasarkan *orderID* dari pesanan yang dipilih. Jika pada pesanan tersebut tidak ditemukan barang, sistem menampilkan notifikasi "Tidak ada barang untuk diinput" dan proses dihentikan. Namun jika data barang tersedia, fungsi memulai proses iterasi untuk setiap dokumen barang yang ditemukan dan mengambil data dasar seperti nama barang serta jumlah estimasi pesanan yang telah ditentukan sebelumnya.

Setelah informasi dasar barang diperoleh, sistem kembali mengambil data progres pekerjaan dari sub-koleksi *OrderProgress* milik barang tersebut. Semua nilai progres yang pernah dicatat dijumlahkan menggunakan fungsi *sumOf*, sehingga menghasilkan nilai total progres terkini. Data ini kemudian dibuat menjadi objek *InputHarianItem*, yang menyimpan ID barang, nama barang, nilai progres sebelumnya, dan jumlah estimasi pekerjaan. Objek-objek data ini dimasukkan ke dalam sebuah list bernama *inputItems* yang akan digunakan untuk menampilkan *form input*.

Fungsi ini juga memastikan bahwa seluruh data barang telah selesai diproses sebelum *form* ditampilkan. Variabel *processedCount* digunakan untuk menghitung jumlah barang yang berhasil diproses, baik berhasil maupun gagal memuat data progres. Jika jumlah yang telah diproses sama dengan total data barang, maka fungsi *displayInputForm()* dipanggil untuk menampilkan formulir input harian kepada pengguna.

Setelah proses pemuatan data barang berhasil dilakukan, langkah berikutnya adalah mencatat hasil progres harian dari setiap item. Kode 3.8 merupakan fungsi *saveInputHarian* yang digunakan untuk memvalidasi dan menyimpan input progres harian ke dalam *database*.

```
private fun saveInputHarian(
        inputItems: List<InputHarianItem>,
        itemBindings: List<ItemInputBarangBinding>,
        orderID: String,
        dialog: Dialog,
        dialogBinding: DialogInputHarianBinding
    ) {
        // Hide card alert
        dialogBinding.cardAlert.visibility = View.GONE

        var hasError = false
        val updates = mutableListOf<Pair<InputHarianItem, Int>>()
        // Format angka menggunakan Indonesia
        val numberFormat = NumberFormat.getNumberInstance(Locale("id", "ID"))

        // Validasi input
        inputItems.forEachIndexed { index, item ->
            val input = itemBindings[index].etHasilHarian.text.toString()

            // Hanya validasi barang yang belum selesai
            if (item.currentProgress < item.estimateOrder) {

                //Cek apakah field input kosong
                if (input.isEmpty()) {
                    itemBindings[index].etHasilHarian.error = "Harus diisi"
                    hasError = true
                } else {
                    val hasilHarian = input.toIntOrNull() ?: 0

                    //Cek apakah input kurang dari atau sama dengan 0
                    if (hasilHarian <= 0) {
                        itemBindings[index].etHasilHarian.error = "Harus lebih dari 0"
                        hasError = true
                    } else {
                        // cek apakah total progress melebihi PO
                        val totalProgress = item.currentProgress + hasilHarian
                        val sisaPesanan = item.estimateOrder -
```

```
    item.currentProgress

                    if (totalProgress > item.estimateOrder) {
                        // Tampilkan alert
                        dialogBinding.cardAlert.visibility = View.VISIBLE
                        dialogBinding.tvAlertMessage.text = "Input untuk ${item.barangName} melebihi sisa pesanan!\n" +
                                "Sisa: ${numberFormat.format(sisaPesanan)} PCS, " +
                                "Input: ${numberFormat.format(hasilHarian)} PCS"

                        // Set error
                        itemBindings[index].etHasilHarian.error = "Melebihi sisa ${numberFormat.format(sisaPesanan)} PCS"
                        hasError = true

                        dialogBinding.scrollView.post {
                            dialogBinding.scrollView.smoothScrollTo(0, 0)
                        }
                    } else {
                        updates.add(Pair(item, hasilHarian))
                    }
                }
            }
        }
    }

    // Jika ada error hentikan proses simpan
    if (hasError) return

    // Simpan Firestore
    var savedCount = 0
    val totalUpdates = updates.size

    // Jika tidak ada data valid yang bisa disimpan
    if (totalUpdates == 0) {
        Toast.makeText(this, "Tidak ada data untuk disimpan", Toast.LENGTH_SHORT).show()
        return
    }

    // Loop setiap input harian yang valid dan simpan ke Firestore
    updates.forEach { (item, hasilHarian) ->

        // Data yang akan disimpan ke Firestore
        val progressData = hashMapOf(
            "CurrentProgress" to hasilHarian,
            "Date" to Timestamp.now()
        )

        firestore.collection("Order")
            .document(orderID)
            .collection("OrderBarang")
            .document(item.barangID)
            .collection("OrderProgress")
```

```
                .add(progressData)
                .addOnSuccessListener {
                    savedCount++
                    if (savedCount == totalUpdates) {
                        Toast.makeText(this, "Progress berhasil
disimpan!", Toast.LENGTH_SHORT).show()
                        dialog.dismiss()
                        // Refresh data setelah simpan
                        loadProgress()
                    }
                }
                .addOnFailureListener {
                    Toast.makeText(this, "Gagal menyimpan progress
", Toast.LENGTH_SHORT).show()
                }
        }
    }
```

Kode 3.8: Fungsi saveInputHarian

Kode 3.8 merupakan fungsi *saveInputHarian()* yang digunakan untuk menyimpan hasil input progres harian produksi setiap barang ke dalam *Firestore*. Fungsi ini menerima beberapa parameter, yaitu *inputItems*, *binding itemBindings*, *orderID*, dialog tampilan input, serta *dialogBinding*. Pada bagian awal, fungsi menyembunyikan *alert card* dan mendeklarasikan beberapa variabel seperti *hasError* untuk menandai kesalahan input, *updates* untuk menampung data yang valid, serta *numberFormat* untuk memformat tampilan angka sesuai *format* Indonesia.

Selanjutnya, sistem melakukan validasi pada setiap input yang dimasukkan pengguna. Setiap *item* dicek apakah hasil *input* sudah terisi serta bernilai lebih dari 0. Selain itu, fungsi juga menghitung apakah total progres setelah input tidak melebihi jumlah pesanan. Jika terjadi, sistem menampilkan pesan kesalahan pada *EditText*, memunculkan *cardAlert*, serta melakukan *scroll* otomatis ke bagian atas dialog agar pesan dapat terlihat oleh pengguna. Jika tidak ada kesalahan, input tersebut disimpan ke dalam daftar *updates* untuk kemudian diproses.

Apabila semua input telah valid, fungsi mulai melakukan penyimpanan ke *Firestore*. Setiap data progres baru dimasukkan sebagai dokumen baru pada sub-koleksi *OrderProgress* berdasarkan ID barang dan ID pesanan. Ketika semua data berhasil tersimpan, sistem menampilkan notifikasi melalui *Toast*, menutup dialog input, dan memanggil fungsi *loadProgress()* untuk memperbarui tampilan data pada halaman. Sementara itu, jika proses penyimpanan gagal, sistem menampilkan pesan kesalahan kepada pengguna.

Setelah seluruh progres produksi pada suatu pesanan perusahaan dinyatakan selesai, sistem memberikan opsi kepada pengguna untuk menekan tombol “Kirim

Barang" sebagai langkah akhir dalam proses penyelesaian order. Untuk menangani proses tersebut, aplikasi menggunakan fungsi *saveValidasiPengiriman()*, yang bertugas memvalidasi input pengguna, menyimpan data pengiriman ke *Firestore*, serta memperbarui status pesanan. Untuk kode *SaveValidasiPengiriman* dapat dilihat pada Kode 3.9.

```
private fun saveValidasiPengiriman(
        dialog: Dialog,
        orderID: String,
        dialogBinding: DialogValidasiPengirimanBinding
    ) {
        val totalDefectStr = dialogBinding.etTotalDefect.text.
    toString()
        val courierName = dialogBinding.etCourierName.text.
    toString().trim()
        val deliveryDateStr = dialogBinding.etDeliveryDate.text.
    toString()

        var hasError = false

        //Cek apakah field total barang cacat kosong
        if (totalDefectStr.isEmpty()) {
            dialogBinding.etTotalDefect.error = "Harus diisi (isi
    0 jika tidak ada)"
            hasError = true
        }

        //Cek apakah field nama pengirim kosong
        if (courierName.isEmpty()) {
            dialogBinding.etCourierName.error = "Nama pengirim
    harus diisi"
            hasError = true
        }

        //Cek apakah tanggal pengiriman sudah dipilih
        if (deliveryDateStr.isEmpty()) {
            dialogBinding.etDeliveryDate.error = "Tanggal
    pengiriman harus dipilih"
            hasError = true
        }

        // Jika ada error hentikan proses simpan
        if (hasError) return

        val totalDefect = totalDefectStr.toIntOrNull() ?: 0
        val deliveryDate = dialogBinding.etDeliveryDate.tag as?
    Date ?: Date()

        // Simpan ke Firestore subcollection PostOrder
        val postOrderData = hashMapOf(
            "CourierName" to courierName,
            "DeliveryDate" to Timestamp(deliveryDate),
            "TotalDefect" to totalDefect,
        )

        // Simpan data pengiriman ke Firestore
        firestore.collection("Order")
```

```
            .document(orderID)
            .collection("PostOrder")
            .add(postOrderData)
            .addOnSuccessListener {
                // Update status FinishedOrder di firestore
                firestore.collection("Order")
                    .document(orderID)
                    .update("FinishedOrder", true)
                    .addOnSuccessListener {
                        dialog.dismiss()
                        showSuccessDialog()
                    }
                    .addOnFailureListener {
                        // Jika gagal update status FinishedOrder
                        Toast.makeText(this, "Gagal update status
order", Toast.LENGTH_SHORT).show()
                    }
            }
            .addOnFailureListener {
                Toast.makeText(this, "Gagal menyimpan data
pengiriman", Toast.LENGTH_SHORT).show()
            }
    }
```

Kode 3.9: Fungsi SaveValidasiPengiriman

Kode 3.9 merupakan fungsi *saveValidasiPengiriman()* digunakan untuk menangani proses penyimpanan data validasi pengiriman pada akhir proses produksi suatu pesanan. Fungsi ini dijalankan setelah pengguna mengisi form validasi pengiriman pada dialog, yang mencakup informasi seperti jumlah produk cacat, nama kurir pengirim, serta tanggal pengiriman.

Pada bagian awal, fungsi mengambil nilai yang dimasukkan pengguna dari tiga input utama, yaitu *totalDefect*, *courierName*, dan *deliveryDate*. Selanjutnya, dilakukan proses validasi untuk memastikan seluruh field wajib telah terisi. Jika terdapat input yang kosong, fungsi memberikan pesan kesalahan pada masing-masing *EditText* dan menghentikan proses penyimpanan dengan menandai *hasError = true*.

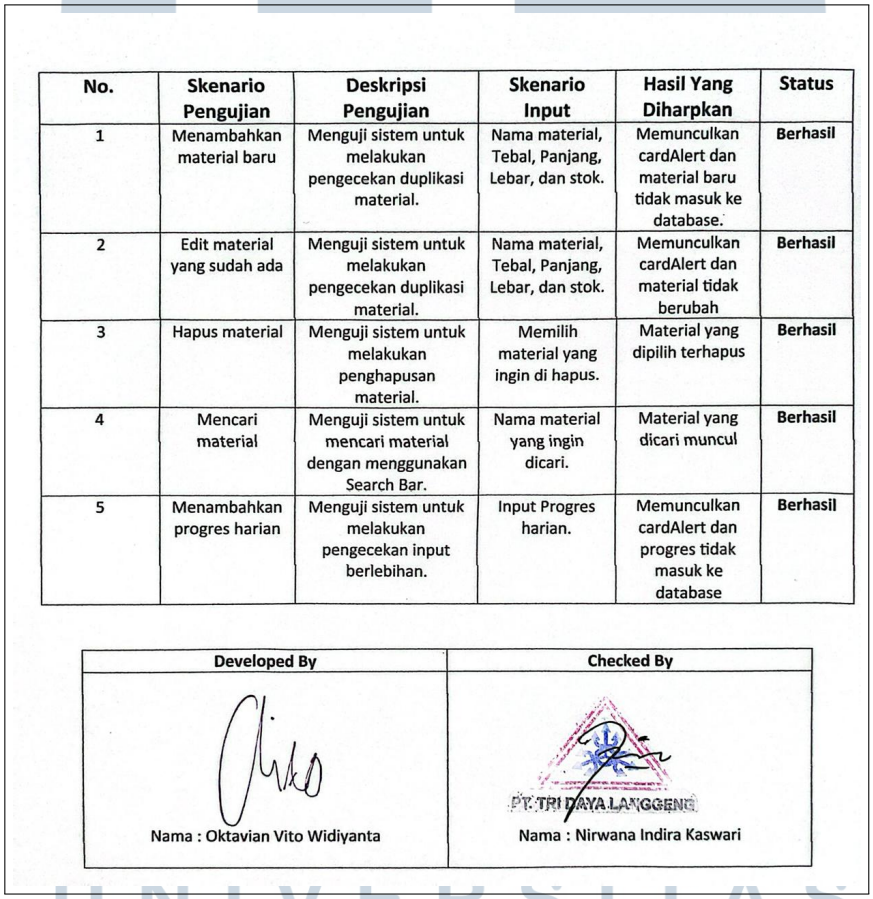
Jika semua input valid, fungsi mengonversi nilai *defect* menjadi tipe *Int* dan membaca nilai tanggal dari *tag* pada komponen tanggal. Setelah semua data siap, fungsi membuat struktur *HashMap* bernama *postOrderData* untuk disimpan ke *Firestore* pada subkoleksi *PostOrder* di dalam dokumen pesanan terkait.

Apabila penyimpanan ke *PostOrder* berhasil, fungsi melanjutkan proses dengan memperbarui status pesanan pada *Firestore*, yaitu mengubah *field FinishedOrder* menjadi *true*. Hal ini menandakan bahwa pesanan telah dinyatakan selesai dan telah melalui proses validasi pengiriman. Jika kedua proses (penyimpanan data dan pembaruan status) berhasil, dialog akan ditutup dan aplikasi

akan menampilkan *popup* berhasil melalui fungsi *showSuccessDialog()*.

3.5 Testing Aplikasi

Untuk memastikan aplikasi berfungsi sebagaimana mestinya, dilakukan *black box testing*. Gambar 3.20 memperlihatkan hasil pengujian *black box* terhadap sistem disertai tanda tangan Direktur yang memiliki peran sebagai penguji aplikasi. Dokumentasi ini menjadi bukti bahwa seluruh fitur yang diuji telah diverifikasi langsung oleh manajemen.



No.	Skenario Pengujian	Deskripsi Pengujian	Skenario Input	Hasil Yang Diharpkan	Status
1	Menambahkan material baru	Menguji sistem untuk melakukan pengecekan duplikasi material.	Nama material, Tebal, Panjang, Lebar, dan stok.	Memunculkan cardAlert dan material baru tidak masuk ke database.	Berhasil
2	Edit material yang sudah ada	Menguji sistem untuk melakukan pengecekan duplikasi material.	Nama material, Tebal, Panjang, Lebar, dan stok.	Memunculkan cardAlert dan material tidak berubah	Berhasil
3	Hapus material	Menguji sistem untuk melakukan penghapusan material.	Memilih material yang ingin di hapus.	Material yang dipilih terhapus	Berhasil
4	Mencari material	Menguji sistem untuk mencari material dengan menggunakan Search Bar.	Nama material yang ingin dicari.	Material yang dicari muncul	Berhasil
5	Menambahkan progres harian	Menguji sistem untuk melakukan pengecekan input berlebihan.	Input Progres harian.	Memunculkan cardAlert dan progres tidak masuk ke database	Berhasil

Developed By	Checked By
Nama : Oktavian Vito Widiyanta	PT TRI DAYA LANGGENG Nama : Nirwana Indira Kaswari

Gambar 3.20. Hasil testing dengan tanda tangan direktur

3.6 Kendala dan Solusi yang Ditemukan

Selama pelaksanaan kegiatan magang di PT Tri Daya Langgeng, ada beberapa kendala yang menghambat pengerjaan aplikasi.

3.6.1 Kendala

Kendala yang terjadi selama kegiatan magang yaitu:

1. Ada *error* pada saat pengunaan *firestore FirebaseFirestoreException: PERMISSION_DENIED*.
2. Permasalahan *default Format Number* yang digunakan pada *android*.

3.6.2 Solusi

Solusi yang ditemukan untuk kendala yang di alami yaitu:

1. Mengupdate *Firestore Security Rules* untuk mengizinkan *read/write access* ke *collection* yang diperlukan.
2. Menggunakan *class NumberFormat* supaya menggunakan format Indonesia.