# BAB 3
# PELAKSANAAN KERJA MAGANG

## 3.1 Kedudukan dan Koordinasi

Selama menjalani program magang di PT Tri Daya Langgeng, penempatan dilakukan di bawah Divisi Keuangan dengan peran sebagai *Mobile App Developer*. Dalam pelaksanaan kegiatan magang, keterlibatan dilakukan bersama tim kecil yang terdiri dari tiga orang, yaitu dua orang dengan posisi *Mobile App Developer* dan satu orang dengan posisi *UI/UX Designer*.

Aktivitas sehari-hari dilaksanakan di lingkungan kerja PT Tri Daya Langgeng dengan koordinasi dan komunikasi yang dilakukan secara langsung bersama Bapak Nirwana selaku supervisor dan direktur dari PT Tri Daya Langgeng. Interaksi kerja dilakukan melalui diskusi tatap muka di tempat kerja untuk membahas perkembangan proyek, penyelesaian masalah, serta penyesuaian teknis yang diperlukan selama proses pengembangan sistem.

Untuk mendukung kolaborasi dalam proses pengembangan proyek, tim pengembang menggunakan platform *GitHub* sebagai media utama dalam pengelolaan kode sumber. Seluruh anggota tim melakukan proses unggah (*push*), peninjauan (*review*), dan penggabungan (*merge*) terhadap setiap perubahan yang dilakukan pada repositori proyek. Penggunaan *GitHub* ini memungkinkan adanya transparansi, efisiensi, serta struktur kerja yang terorganisir dalam setiap tahapan pengembangan aplikasi.

## 3.2 Tugas yang Dilakukan

Selama melaksanakan kegiatan magang di PT Tri Daya Langgeng, berbagai tugas terkait pengembangan sistem informasi berbasis *Android* telah dilaksanakan dalam mendukung proses operasional perusahaan yaitu :

1. **Laman Pesanan Baru**
   Mengembangkan Laman Pesanan Baru yang berfungsi sebagai sarana bagi pengguna untuk melakukan pemesanan material. Pada Laman ini dikembangkan fitur pemilihan material yang secara otomatis menampilkan informasi terkait ketebalan, panjang, dan lebar material yang dipilih. Selain itu, sistem juga dirancang untuk menghitung total harga pemesanan secara

otomatis, termasuk perhitungan pajak, sehingga seluruh proses perhitungan dapat dilakukan dengan lebih akurat dan efisien. Implementasi laman ini bertujuan untuk mempermudah proses pemesanan yang sebelumnya banyak dilakukan secara manual.

2. **Laman Histori Pesanan**

   Mengembangkan Laman Histori Pesanan yang Sudah Selesai. Laman ini dirancang untuk menampilkan informasi mengenai daftar pesanan yang telah selesai diproduksi. Fitur utama pada laman ini mencakup penyediaan kotak pencarian untuk menampilkan histori pesanan berdasarkan nama perusahaan, serta penyajian informasi progres penyelesaian barang per hari. Informasi yang ditampilkan meliputi jumlah barang yang selesai diproduksi, jumlah barang yang mengalami kerusakan selama proses produksi, dan informasi terkait pihak pengirim barang. Laman ini berfungsi sebagai sarana monitoring yang lebih terstruktur sehingga memudahkan proses evaluasi pada kegiatan produksi.

3. **Laman Akumulasi Omset**

   Mengembangkan Laman Akumulasi Omset yang digunakan untuk menampilkan total omset perusahaan beserta rincian omset per bulan. Pada laman ini ditampilkan rekapitulasi pendapatan, dan ketika salah satu bulan dipilih, sistem akan menampilkan perusahaan-perusahaan yang melakukan pembelian pada bulan tersebut beserta nilainya. Selain itu, disediakan pula fitur untuk menghapus data omset pada bulan tertentu apabila diperlukan penyesuaian administrasi. Pengembangan laman ini bertujuan untuk mempermudah proses analisis pendapatan serta mendukung manajemen dalam pengambilan keputusan berdasarkan data keuangan yang lebih terstruktur.

## 3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan setiap minggu selama magang

| **Minggu Ke -** | **Pekerjaan yang dilakukan** |
|---|---|
| 1 | Orientasi awal, perkenalan lingkungan kerja, dan perkenalan dengan karyawan perusahaan. |
| 2 | Mengikuti meeting bersama karyawan kantor, melakukan diskusi dengan direktur terkait tugas tambahan di luar proyek utama, serta mulai mengerjakan dan melanjutkan penyelesaian tugas yang diberikan oleh direktur. |
| 3 | Mengikuti meeting bersama karyawan, mengerjakan tugas yang diberikan direktur, serta melakukan diskusi dengan tim *IT* terkait permasalahan perusahaan dan solusi yang telah disiapkan, kemudian melanjutkan pengerjaan tugas yang diberikan. |
| 4 | Mengikuti meeting bersama karyawan, mempelajari ulang dasar-dasar *Kotlin*, memahami konsep *error handling*, mendalami komponen utama Android untuk desain antarmuka seperti *Composable Function, ViewModel, dan Recomposition*, serta mempelajari *state management* menggunakan *MutableStateOf, derivedStateOf, dan StateFlow.* |
| 5 | Mengikuti beberapa meeting bersama karyawan, berdiskusi dengan tim *IT* serta membantu pembuatan desain *dashboard* di *Figma*, mengerjakan tugas yang diberikan direktur, dan turut membantu serta berdiskusi dengan *UI/UX designer* dalam menyusun desain aplikasi. |
| 6 | Mengikuti *meeting* bersama karyawan, berdiskusi dengan tim *IT* mengenai alur kerja aplikasi yang akan dibuat, memberikan masukan kepada *UI/UX designer* terkait desain halaman histori pemesanan, serta melakukan beberapa sesi diskusi dengan *full stack developer* mengenai cara kerja halaman *dashboard*, *order progress*, dan akumulasi pendapatan. |
| | Lanjut pada halaman berikutnya |

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

| **Minggu Ke -** | **Pekerjaan yang dilakukan** |
|---|---|
| 7 | Mengikuti *meeting* bersama karyawan dan mempresentasikan desain aplikasi kepada direktur, kemudian memulai pembuatan halaman pemesanan barang baru serta melanjutkan proses pengembangan dan penyusunan *layout* tampilan *Android* untuk halaman tersebut. |
| 8 | Mengikuti meeting bersama karyawan serta membantu direktur, melanjutkan pembuatan halaman pemesanan baru, menambahkan elemen navigasi seperti *navigation bar* dan n*avigation drawer*, serta melakukan beberapa revisi pada halaman pemesanan baru termasuk pada bagian *bottom navigation* dan *drawer navigation*. |
| 9 | Mengikuti *meeting* bersama karyawan, mengerjakan tugas dari direktur, melakukan perbaikan pada *AddOrderActivity* menggunakan *intent* untuk pengiriman data, menambahkan *InputOrderItemsActivity* untuk menentukan nama dan jumlah barang yang ingin di *pre-order*, serta membuat tampilan untuk *activity_add_order* dan *activity_input_order_items.* |
| 10 | Mengikuti *meeting* bersama karyawan dan klien terkait pemesanan serta membantu direktur, membuat *MaterialSelectionActivity* untuk menampung data pesanan seperti material, jumlah lembar, perkiraan hasil, dan *grand total*, membuat tampilan *activity_material_selection*, *item_input_order_item*, serta *item_material_selection*, dan membuat *OrderSummaryActivity* beserta tampilannya untuk menampilkan total harga, pajak, dan *grand total* pemesanan. |
| | Lanjut pada halaman berikutnya |

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

| **Minggu Ke -** | **Pekerjaan yang dilakukan** |
|---|---|
| 11 | Mengikuti *meeting* bersama karyawan serta membantu direktur dan staf dalam persiapan kunjungan klien, kemudian memperbaiki *bug* pada proses pengiriman data antar-*activity* di *AddOrderActivity*, *InputOrderItemsActivity*, *MaterialSelectionActivity*, dan *OrderSummaryActivity*, serta memperbaiki kesalahan penyimpanan data yang menyebabkan seluruh data masuk ke koleksi Order, sehingga alurnya kembali sesuai yaitu *Order* ke koleksi *Order* dan *OrderSummary* ke koleksi *OrderItem*. |
| 12 | Melanjutkan perbaikan *bug* pada *AddOrderActivity.kt* dan *OrderSummaryActivity.kt*, memperbaiki struktur basis data pada *Firestore* termasuk perubahan nama sub-*collection* dari *OrderItem* menjadi *OrderBarang*, serta membuat halaman histori pesanan selesai untuk menampilkan pesanan yang telah diselesaikan. |
| 13 | Mengikuti meeting bersama karyawan kantor, kemudian menambahkan fitur pencarian (*search bar*) pada halaman histori untuk memudahkan pencarian data pesanan secara spesifik, serta membuat halaman Akumulasi Omset yang menampilkan total omset penjualan dan omset per bulan |
| 14 | Mengikuti meeting bersama karyawan kantor, menambahkan fitur pada halaman Akumulasi Omset agar dapat menampilkan data penjualan berdasarkan bulan yang dipilih oleh pengguna, serta menambahkan fitur penghapusan data omset pada bulan tertentu sesuai pilihan pengguna. |
| 15 | Mengikuti meeting bersama karyawan kantor, mempresentasikan hasil pengembangan aplikasi pencatatan pesanan, serta mengerjakan tugas yang diberikan oleh direktur perusahaan terkait penanganan pesanan baru yang masuk. |
| 16 | Mengikuti meeting bersama karyawan kantor serta mengerjakan tugas yang diberikan oleh direktur perusahaan terkait penanganan pesanan baru yang masuk. |

Lanjut pada halaman berikutnya

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

| Minggu Ke - | Pekerjaan yang dilakukan |
|---|---|
| 17 | Mengikuti meeting bersama karyawan kantor serta mengerjakan tugas yang diberikan oleh direktur perusahaan terkait penanganan pesanan baru yang masuk. |

## 3.4 Proses Perancangan Sistem Aplikasi

### 3.4.1 Perancangan Flowchart

Pada bagian ini akan dijelaskan *flowchart* yang disusun untuk merepresentasikan alur kerja sistem secara terstruktur. Penyajian *flowchart* bertujuan untuk memberikan pemahaman konseptual mengenai urutan proses, hubungan antar aktivitas, serta logika operasional yang mendasari sistem. Dengan adanya *flowchart*, analisis dan perancangan sistem dapat dilakukan secara lebih sistematis, konsisten, dan mudah dipahami.

#### A Flowchart Pesanan Baru

*Flowchart* pada Gambar 3.1 tersebut menggambarkan rangkaian proses yang harus dilalui pengguna dalam melakukan pembuatan pesanan baru pada sistem. Proses dimulai dari tahap *Start*, kemudian pengguna diarahkan menuju langkah awal yaitu melakukan *Input Total Item*. Pada tahap ini, pengguna diminta menentukan jumlah keseluruhan barang yang akan dipesan sebelum melanjutkan ke proses berikutnya.

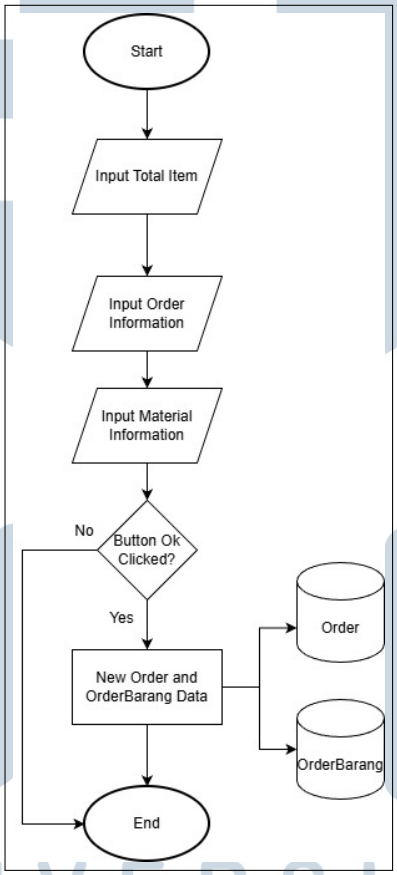
Setelah jumlah *item* ditentukan, pengguna memasuki tahap *Input Order Information*. Pada bagian ini pengguna diminta mengisi informasi pemesanan yang meliputi nama perusahaan (PT), tanggal pemesanan, tanggal jatuh tempo, nama barang untuk setiap item, serta Total PO (*Pre Order*) dari masing-masing barang yang akan dipesan.

Tahap berikutnya adalah *Input Material Information*, di mana pengguna diminta memasukkan rincian terkait material yang akan digunakan. Informasi ini mencakup pemilihan jenis material, jumlah lembar bahan yang diperlukan, jumlah pita, serta harga per satuan (*Price Per Pieces*).

Setelah seluruh informasi terisi, sistem menampilkan laman konfirmasi yang diwakili pada *flowchart* dengan proses keputusan *Button OK Clicked*?. Pengguna

diberikan kesempatan untuk meninjau ulang total pesanan, termasuk Total *PO*, pajak (*tax*), dan *grand* total yang merupakan hasil penjumlahan Total *PO* dan pajak. Jika pengguna menekan tombol OK atau *Confirm*, maka proses dilanjutkan menuju tahap *New Order and Order*Barang Data, yaitu pengiriman data ke basis data.

Pada tahap ini, sistem menyimpan data utama pesanan ke dalam koleksi *Order*, serta menyimpan detail setiap barang yang dipesan ke dalam subkoleksi *Order*Barang. Setelah data berhasil dikirim, proses berakhir pada tahap *End*. Apabila pengguna tidak menekan tombol konfirmasi pengguna dapat kembali ke laman sebelumnya atau menuju *dashboard* tanpa menyimpan data apa pun.




Gambar 3.1. Flowchart Pesanan Baru

## B Flowchart History Pesanan

*Flowchart* pada Gambar 3.2 menggambarkan alur proses sistem dalam menampilkan riwayat pesanan (*Order History*) beserta detail informasinya. Proses dimulai dari tahap *Start*, kemudian sistem melakukan langkah awal yaitu *Get Order Data*. Pada tahap ini aplikasi mengambil seluruh data pesanan dari koleksi *Order*

yang tersimpan di *database*.

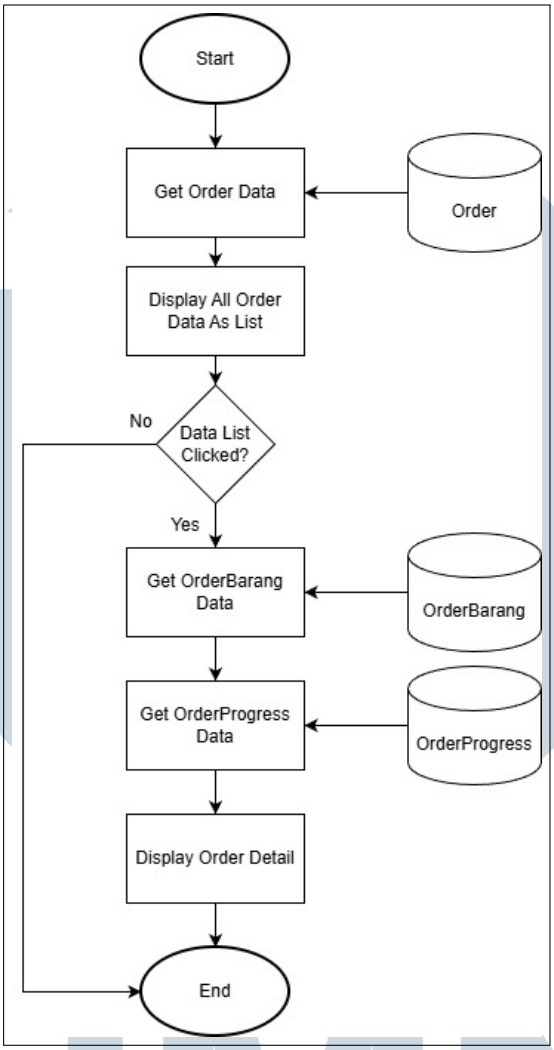
Setelah data berhasil diperoleh, sistem melanjutkan ke tahap *Display All Order Data As List*, yaitu menampilkan seluruh data pesanan dalam bentuk daftar (*list*) pada tampilan aplikasi. Pengguna dapat melihat semua pesanan yang telah dibuat sebelumnya.

Selanjutnya, *flowchart* menunjukkan sebuah proses pengambilan keputusan pada tahap *Data List Clicked?*. Pada tahap ini, sistem menunggu interaksi pengguna terhadap salah satu *item* pesanan pada daftar yang ditampilkan. Jika pengguna tidak memilih (klik) salah satu *item*, sistem akan kembali menampilkan daftar tanpa melanjutkan proses dan jika pengguna mengklik salah satu data pesanan, proses akan berlanjut mengikuti alur *Yes*.

Ketika pengguna memilih salah satu pesanan, sistem akan melakukan pengambilan data rinci melalui tahap *Get Order*Barang Data. Pada tahap ini aplikasi mengambil seluruh data barang yang terkait dengan pesanan tersebut dari subkoleksi *Order*Barang.

Setelah itu, sistem memasuki tahap *Get OrderProgress* Data, yaitu mengambil data perkembangan pesanan (*progress*) dari subkoleksi *OrderProgress*, yang berfungsi untuk menampilkan status penyelesaian atau aktivitas terkait pesanan tersebut.

Setelah seluruh data berhasil diperoleh, sistem melanjutkan ke tahap *Display Order Detail*. Pada tahap ini aplikasi menampilkan informasi detail pesanan yang dipilih, termasuk daftar barang, *progress* pekerjaan, dan informasi terkait lainnya. Proses kemudian berakhir pada tahap *End* setelah data detail pesanan selesai ditampilkan dan pengguna kembali ke laman *dashboard*.




Gambar 3.2. Flowchart History Pesanan

## C Flowchart Akumulasi Omset

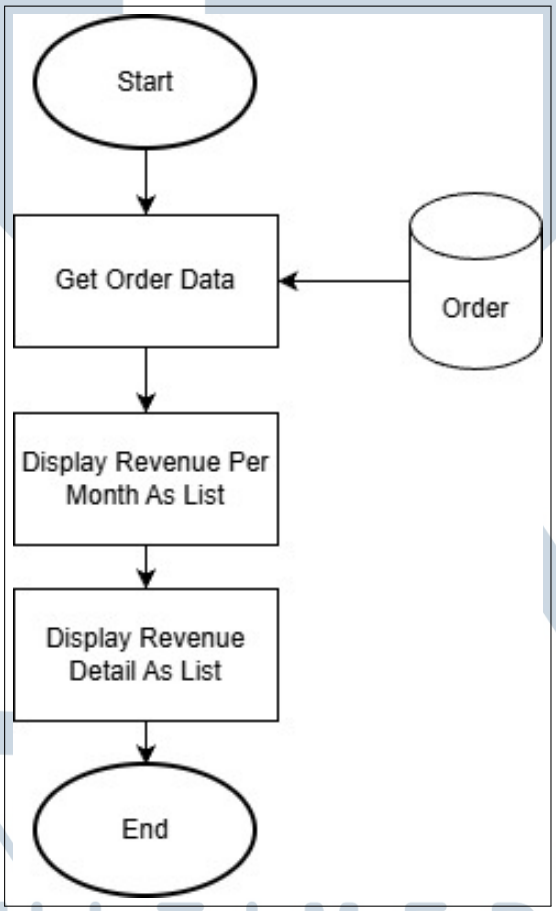
*Flowchart* pada Gambar 3.3 menggambarkan alur proses sistem dalam menampilkan informasi akumulasi omset (*revenue*) berdasarkan data pesanan yang telah tersimpan. Proses dimulai pada tahap *Start*, kemudian sistem melaksanakan langkah awal yaitu *Get Order Data*, yaitu mengambil seluruh data pesanan dari koleksi *Order* pada *database*. Data ini menjadi dasar perhitungan omset bulanan.

Selanjutnya, sistem memasuki tahap *Display Revenue Per Month As List*, di mana aplikasi mengolah data pesanan yang telah diambil dan menampilkan hasil perhitungan omset berdasarkan bulan dalam bentuk daftar (*list*). Pada tahap ini pengguna dapat melihat ringkasan total pendapatan untuk setiap bulan.

*Flowchart* kemudian menampilkan proses keputusan *Data List Clicked?*, yang

menandai interaksi pengguna terhadap salah satu item daftar omset bulanan. Jika pengguna tidak memilih salah satu *item* pada daftar, sistem akan tetap berada pada tampilan daftar omset bulanan tanpa melanjutkan proses. Jika pengguna mengklik salah satu data bulan pada daftar, proses berlanjut mengikuti alur *Yes*.

Ketika pengguna memilih salah satu bulan, sistem memasuki tahap *Display Revenue Detail As List*. Pada tahap ini aplikasi menampilkan rincian omset untuk bulan tersebut, meliputi daftar pesanan yang berkontribusi terhadap total pendapatan, Nama perusahaan (PT), Tanggal pemesanan dan *grand total* (total harga dari barang yang dipesan). Setelah informasi pendapatan ditampilkan secara lengkap, proses berakhir pada tahap *End*.

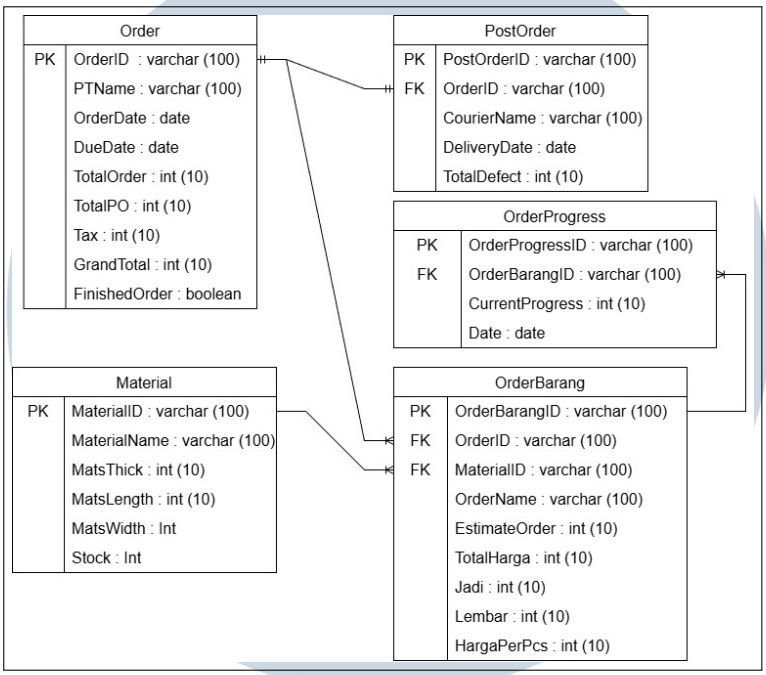



Gambar 3.3. Flowchart Akumulasi Omset

### 3.4.2 Perancangan Database dan Struktur Tabel

Pada bagian ini akan dijelaskan struktur database dan struktur tabel yang digunakan dalam perancangan sistem (Gambar 3.4). *Database* berperan

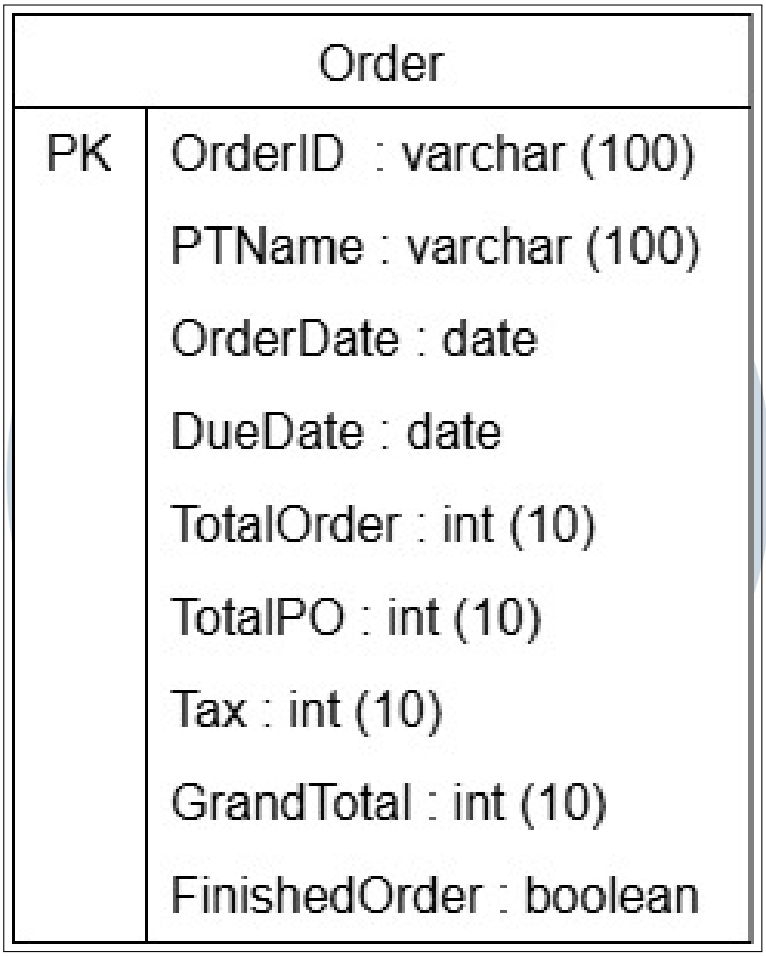
sebagai media penyimpanan terpusat untuk seluruh data operasional, sehingga memungkinkan proses pengelolaan, pengaksesan, dan pemeliharaan data dilakukan secara lebih efektif dan terstruktur.




Gambar 3.4. Skema Database

## A Tabel Order

Tabel *Order* merupakan entitas utama yang merepresentasikan data pemesanan yang dilakukan oleh pengguna (Gambar 3.5). Setiap entri pada tabel ini memuat informasi pokok terkait pesanan, seperti identitas pesanan, nama perusahaan pemesan, tanggal pemesanan dan tanggal jatuh tempo, jumlah *item* yang dipesan, total biaya sebelum pajak, nilai pajak, serta total biaya keseluruhan. Selain itu, tabel ini juga menyimpan status penyelesaian pesanan. Tabel *Order* menjadi pusat relasi terhadap tabel *OrderBarang* dan *PostOrder*, sehingga berperan penting dalam pengelolaan alur pemesanan secara keseluruhan.




Gambar 3.5. Tabel Order

**B Tabel OrderBarang**

Tabel *OrderBarang* menyimpan rincian setiap barang yang termasuk dalam suatu pesanan (Gambar 3.6). Informasi yang dicatat meliputi identitas barang pesanan, identitas material yang digunakan, nama barang, jumlah estimasi pesanan, jumlah lembar material yang digunakan, harga per satuan, serta total harga barang tersebut. Tabel ini berelasi langsung dengan tabel *Order* dan menjadi dasar bagi pencatatan perkembangan produksi pada tabel *OrderProgress*.

| OrderBarang | |
|---|---|
| PK | OrderBarangID : varchar (100) |
| FK | OrderID : varchar (100) |
| FK | MaterialID : varchar (100) |
| | OrderName : varchar (100) |
| | EstimateOrder : int (10) |
| | TotalHarga : int (10) |
| | Jadi : int (10) |
| | Lembar : int (10) |
| | HargaPerPcs : int (10) |

Gambar 3.6. Tabel OrderBarang

### C Tabel OrderProgress

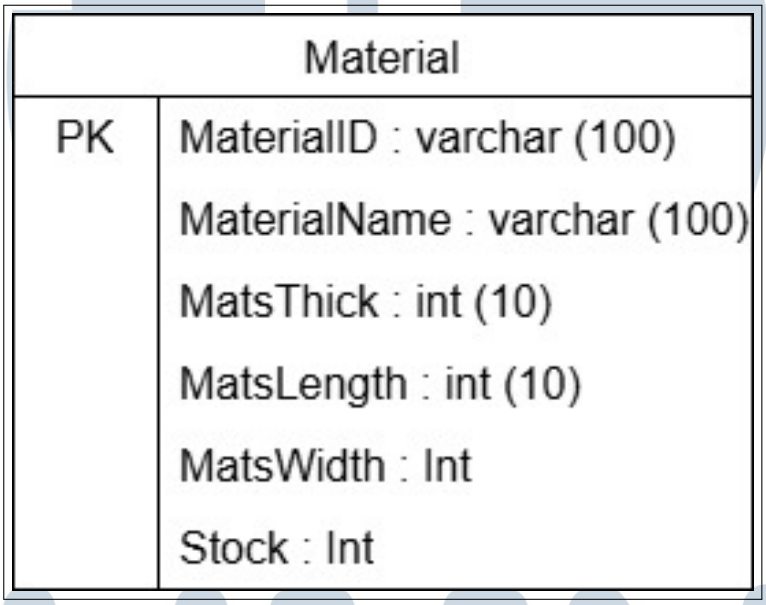
Tabel *OrderProgress* digunakan untuk mendokumentasikan perkembangan proses produksi terhadap tiap barang pesanan (Gambar 3.7). Setiap catatan pada tabel ini memuat identitas progress, identitas barang pesanan terkait, tahap pengerjaan saat ini, serta waktu pencatatan. Melalui tabel ini, sistem dapat menampilkan riwayat perkembangan pengerjaan secara runtut, terstruktur, dan mudah dipantau.

| OrderProgress | |
|---|---|
| PK | OrderProgressID : varchar (100) |
| FK | OrderBarangID : varchar (100) |
| | CurrentProgress : int (10) |
| | Date : date |

Gambar 3.7. Tabel OrderProgress

### D Tabel Material

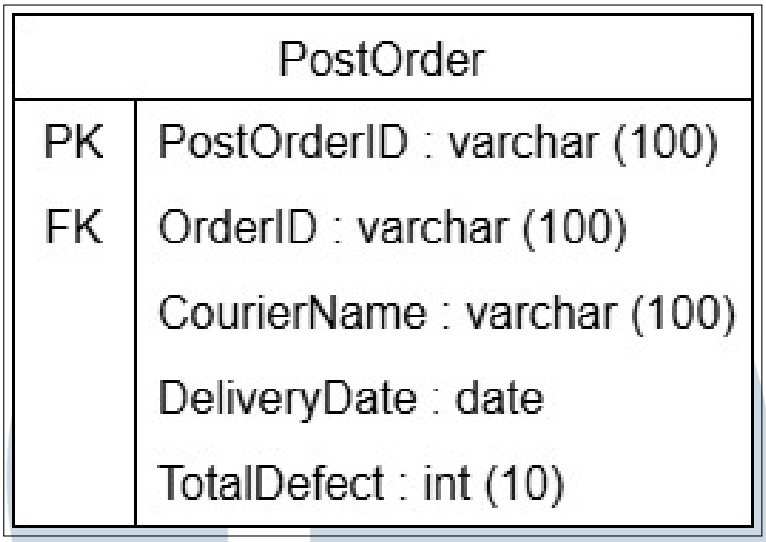
Tabel *Material* berfungsi sebagai penyimpanan data terkait material yang tersedia dan digunakan dalam proses produksi (Gambar 3.8). Informasi yang disimpan mencakup identitas *material*, nama *material*, ketebalan, panjang, lebar, dan jumlah stok *material*. Tabel ini menyediakan referensi *material* yang diperlukan dalam pembuatan entri pada tabel *OrderBarang*, sehingga memastikan konsistensi dan keterhubungan data antar entitas.




Gambar 3.8. Tabel Material

### E Tabel PostOrder

Tabel *PostOrder* menyimpan informasi terkait aktivitas setelah pesanan selesai diproduksi, terutama proses pengiriman barang kepada pelanggan (Gambar 3.9). Data yang dicatat meliputi identitas *post-order*, identitas pesanan terkait, nama kurir pengiriman, tanggal pengiriman, serta jumlah barang cacat yang ditemukan. Tabel ini mendukung pemantauan kualitas akhir produksi dan proses distribusi barang kepada pelanggan.




Gambar 3.9. Tabel PostOrder

### 3.4.3 Tahapan Perancangan Aplikasi PT Tri Daya Langgeng

Pendekatan perancangan aplikasi dilakukan dengan mengacu pada kebutuhan sistem dan alur kerja yang berjalan di PT Tri Daya Langgeng. Perancangan ini disusun sebagai pedoman dalam menentukan struktur tampilan aplikasi, alur navigasi antar laman, serta penempatan komponen antarmuka agar sesuai dengan fungsi yang dibutuhkan dalam proses pencatatan pesanan.

Perancangan dilakukan secara bertahap dengan mempertimbangkan keterkaitan antara setiap fitur yang dikembangkan, sehingga aplikasi yang dihasilkan memiliki alur penggunaan yang jelas dan terstruktur. Dengan pendekatan ini, setiap komponen aplikasi dirancang untuk mendukung proses kerja pengguna serta mempermudah pengelolaan data pesanan secara sistematis.

### 3.4.4 Analisis Kebutuhan Pengguna

Tahapan analisis kebutuhan pengguna dilakukan melalui kegiatan diskusi bersama dengan direktur PT Tri Daya Langgeng yang berperan sebagai pengguna utama aplikasi. Diskusi ini bertujuan untuk mengidentifikasi kebutuhan fungsional serta komponen sistem yang diperlukan agar aplikasi dapat mendukung kegiatan operasional perusahaan secara efektif. Hasil dari tahap analisis ini digunakan sebagai landasan dalam penyusunan fitur serta perancangan tampilan aplikasi, sehingga sistem yang dikembangkan mampu membantu proses pengelolaan data pesanan secara lebih efisien dan terkomputerisasi.

Berdasarkan hasil diskusi yang telah dilakukan, diperoleh beberapa fitur inti yang dinilai paling dibutuhkan oleh pengguna, yaitu :

1. Fitur pertama adalah pencatatan pesanan baru, yang memungkinkan pengguna untuk menambahkan data pesanan dengan dukungan perhitungan otomatis, termasuk perhitungan kebutuhan bahan dan total nilai tagihan dari setiap pesanan.
2. Fitur kedua adalah pemantauan stok barang, di mana aplikasi menyediakan informasi ketersediaan bahan baik secara keseluruhan maupun berdasarkan jenis barang, sehingga kondisi stok dapat diketahui dengan lebih jelas.
3. Fitur ketiga adalah pemantauan progres produksi yang memungkinkan pengguna untuk melihat perkembangan pengerjaan pesanan, seperti status proses produksi dan jumlah pesanan yang telah diselesaikan dalam periode tertentu.

Selain fitur utama tersebut, hasil analisis juga menghasilkan beberapa fitur pendukung yang dirancang untuk melengkapi fungsi aplikasi secara menyeluruh yaitu :

1. Fitur pendukung pertama adalah fitur riwayat pesanan, yang berfungsi untuk menyimpan data pesanan yang telah selesai secara terperinci, mencakup informasi tanggal pemesanan, tanggal pengiriman, serta total biaya.
2. Fitur pendukung kedua adalah akumulasi omset bulanan, yang menampilkan ringkasan total pendapatan perusahaan setiap bulan guna membantu proses pencocokan dengan pencatatan keuangan serta memantau perkembangan kinerja perusahaan.

Berdasarkan hasil analisis kebutuhan yang dilakukan bersama direktur PT Tri Daya Langgeng, pengembangan sistem pencatatan pesanan ini diarahkan untuk diwujudkan dalam bentuk aplikasi *Android*. Aplikasi tersebut dirancang atas kebutuhan pihak manajemen agar direktur dapat melakukan pemantauan terhadap seluruh proses pesanan secara terpusat, akurat, dan efisien, sekaligus mengurangi ketergantungan terhadap laporan manual yang selama ini disusun oleh staf administrasi maupun staf produksi.

### 3.4.5 Perancangan Desain Tampilan (High-Fidelity)

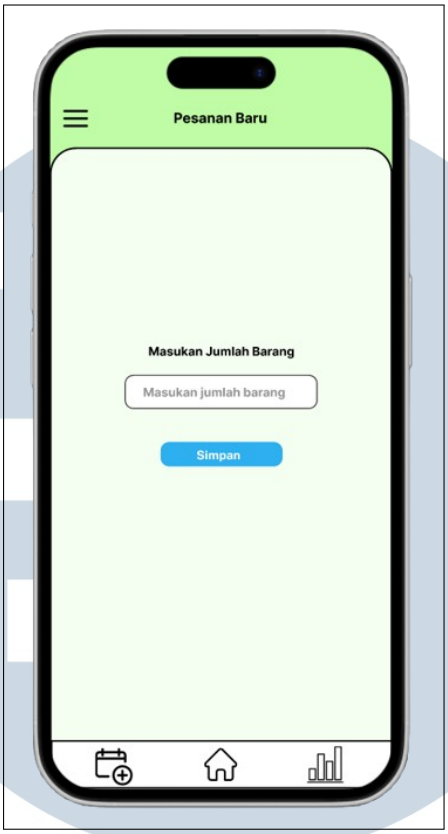
Pada tahap ini akan dijelaskan perancangan desain tampilan aplikasi dalam bentuk desain *high-fidelity* yang dibuat menggunakan *Figma*. Perancangan ini bertujuan untuk memberikan gambaran awal mengenai struktur tampilan, tata letak komponen, serta alur navigasi pada setiap laman aplikasi sebelum diimplementasikan ke dalam sistem. Desain yang ditampilkan mencakup beberapa laman utama, yaitu laman pesanan baru, laman histori pesanan selesai, serta laman akumulasi omset.

#### A Perancangan Desain Laman Pesanan Baru

Laman Pesanan Baru pada Gambar 3.10 merupakan laman yang digunakan untuk memulai proses pencatatan pesanan dalam aplikasi. Laman ini akan ditampilkan ketika pengguna menekan tombol tambah ("+") yang terdapat pada *footer* di bagian bawah kiri layar. Tujuan dari laman ini adalah untuk menerima input awal berupa jumlah barang yang akan dipesan sebelum pengguna melanjutkan ke tahap pengisian detail pesanan.

Pada tampilan laman ini, pengguna disajikan dengan judul "Pesanan Baru" yang terletak di bagian atas layar sebagai penanda konteks laman yang sedang diakses. Di bagian tengah layar terdapat komponen input berupa kolom isian jumlah barang yang disertai dengan keterangan "Masukan Jumlah Barang". Kolom input ini berfungsi sebagai tempat pengguna memasukkan jumlah item yang akan dicatat dalam pesanan baru.

Selain itu, terdapat tombol "Simpan" yang digunakan untuk menyimpan jumlah barang yang telah diinput. Setelah tombol ini ditekan, sistem akan memproses data jumlah barang tersebut dan melanjutkan pengguna ke tahap berikutnya dalam proses pencatatan pesanan. Tata letak komponen pada laman ini dirancang secara sederhana dan terfokus agar pengguna dapat langsung memahami langkah yang harus dilakukan tanpa kebingungan.




Gambar 3.10. Tampilan Figma Laman Pesanan Baru

Setelah pengguna memasukkan jumlah barang pada laman sebelumnya dan menekan tombol Simpan, sistem akan menampilkan tampilan lanjutan dari laman Pesanan Baru yang berfungsi untuk menampilkan dan mengonfirmasi detail awal pesanan. Pada tahap ini, pengguna dapat melihat informasi umum terkait pesanan serta rincian barang yang akan dicatat seperti pada Gambar 3.11.

Pada bagian atas laman ditampilkan beberapa informasi utama, yaitu nama perusahaan pemesan, tanggal pemesanan, serta tanggal jatuh tempo pesanan. Informasi ini berfungsi sebagai identitas awal pesanan dan membantu pengguna memastikan kesesuaian data sebelum pesanan diproses lebih lanjut. Seluruh data ditampilkan dalam bentuk kolom isian agar mudah dibaca dan ditinjau kembali oleh pengguna.

Selanjutnya, pada bagian tengah laman ditampilkan daftar barang yang dipesan sesuai dengan jumlah barang yang telah ditentukan pada tahap sebelumnya. Setiap barang ditampilkan secara terpisah dengan keterangan nama barang yang dipesan serta total jumlah pesanan dalam satuan *pcs*. Penyajian data ini bertujuan untuk memberikan gambaran yang jelas mengenai rincian pesanan yang sedang dibuat.

Di bagian bawah laman terdapat tombol “OKE” yang berfungsi sebagai tombol konfirmasi. Tombol ini digunakan oleh pengguna untuk menyetujui dan melanjutkan proses pencatatan pesanan setelah seluruh informasi yang ditampilkan dianggap sudah sesuai. Dengan adanya tampilan lanjutan ini, pengguna dapat melakukan pengecekan ulang terhadap data pesanan secara menyeluruh sebelum pesanan disimpan atau diproses ke tahap berikutnya.


Gambar 3.11. Tampilan Figma Laman Detail Perusahaan

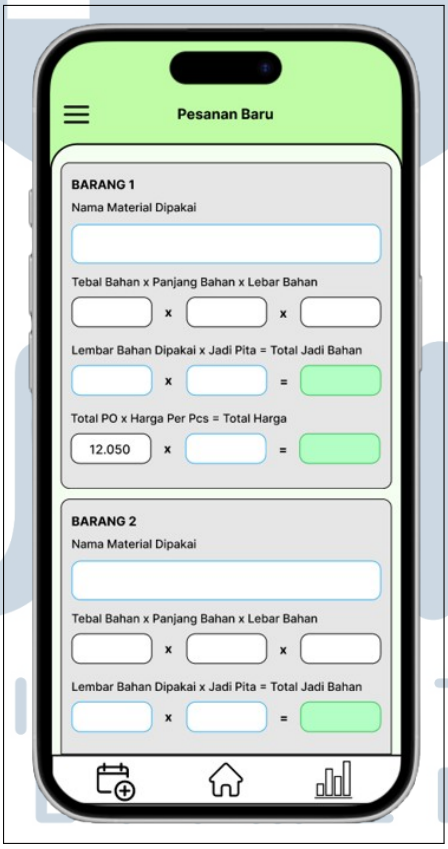
Tampilan laman Pada Gambar 3.12 merupakan lanjutan dari proses pencatatan pesanan baru yang berfungsi untuk menginput data teknis produksi secara lebih rinci. Pada tahap ini, pengguna diminta untuk memasukkan informasi yang berkaitan dengan bahan yang digunakan, perhitungan kebutuhan bahan, serta estimasi total harga pesanan berdasarkan parameter yang ditentukan.

Pada bagian atas laman ditampilkan komponen perhitungan bahan yang terdiri dari isian tebal bahan, panjang bahan, dan lebar bahan. Data tersebut digunakan sebagai dasar dalam menghitung kebutuhan bahan produksi. Selanjutnya, sistem menyediakan perhitungan lanjutan berupa jumlah lembar bahan yang digunakan dikalikan dengan jumlah hasil jadi (pita), sehingga menghasilkan total kebutuhan

bahan yang diperlukan. Hasil perhitungan ini ditampilkan secara otomatis untuk memberikan gambaran yang jelas kepada pengguna.

Selain perhitungan bahan, laman ini juga menampilkan perhitungan total harga pesanan, yang diperoleh dari perkalian antara total jumlah pesanan (*PO*) dengan harga per satuan. Informasi hasil perhitungan ditampilkan secara langsung untuk membantu pengguna mengetahui estimasi biaya dari pesanan yang sedang dicatat.

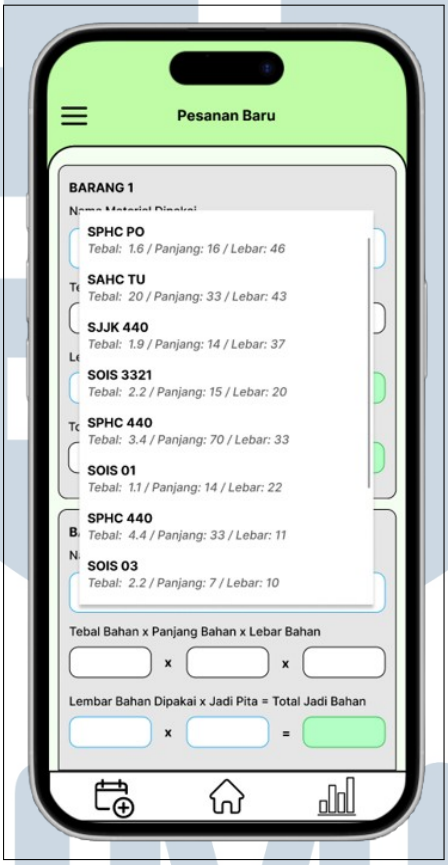
Pada bagian tengah laman, ditampilkan form input untuk setiap barang yang dipesan. Setiap bagian barang diawali dengan penanda seperti "Barang 1" untuk membedakan data antarbarang. Pengguna diminta untuk mengisi beberapa data utama, seperti nama material yang digunakan, ukuran bahan, jumlah lembar bahan yang dipakai, jumlah hasil jadi, serta harga per satuan. Kolom input yang wajib diisi oleh pengguna ditandai dengan garis tepi berwarna biru sebagai indikator bahwa kolom tersebut memerlukan perhatian dan pengisian data.




Gambar 3.12. Tampilan Figma Laman Detail Pesanan

Untuk kolom Nama Material Dipakai, sistem menyediakan mekanisme pemilihan material dalam bentuk daftar pilihan (*dropdown*) seperti pada Gambar 3.13. Ketika kolom tersebut ditekan, akan muncul daftar material yang tersedia

beserta informasi detail dari masing-masing material, seperti ketebalan, panjang, dan lebar bahan. Fitur ini bertujuan untuk membantu pengguna dalam memilih material yang sesuai dengan kebutuhan produksi tanpa harus mengingat spesifikasi bahan secara manual, sebagaimana ditampilkan pada gambar lanjutan.



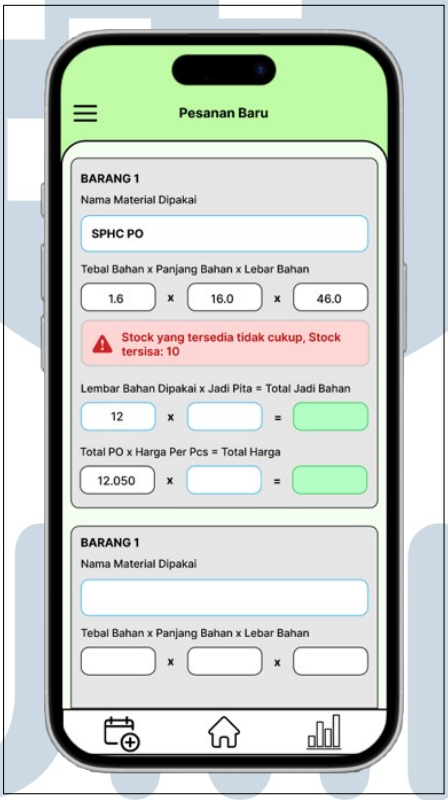

Gambar 3.13. Tampilan Figma Laman Detail Pesanan (Dropdown)

Pada laman Detail Pesanan, sistem juga dilengkapi dengan mekanisme validasi untuk memastikan kesesuaian antara jumlah bahan yang digunakan dengan stok material yang tersedia seperti pada Gambar 3.14. Validasi ini diterapkan pada kolom input jumlah lembar bahan yang dipakai. Apabila pengguna memasukkan jumlah lembar bahan yang melebihi stok material yang tersimpan di sistem, maka sistem akan menampilkan peringatan secara langsung pada tampilan antarmuka.

Peringatan tersebut ditampilkan dalam bentuk notifikasi berwarna merah yang muncul tepat di bawah komponen input ukuran bahan. Notifikasi ini disertai dengan ikon peringatan untuk menarik perhatian pengguna serta pesan teks yang menyatakan bahwa stok bahan tidak mencukupi, lengkap dengan informasi jumlah stok yang tersisa. Penyajian informasi ini bertujuan agar pengguna dapat segera

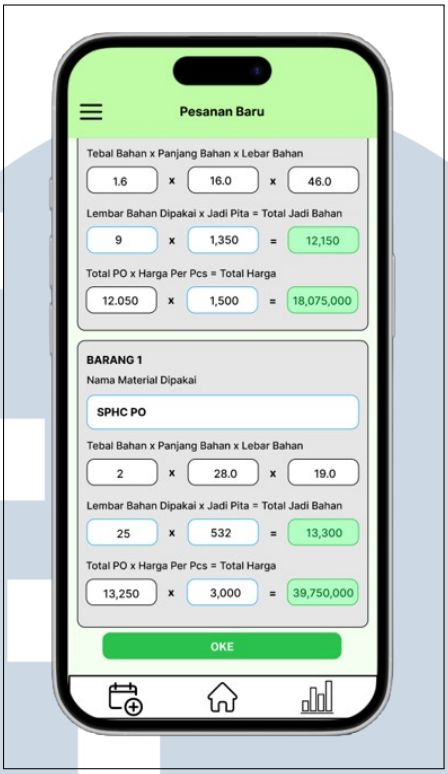
mengetahui penyebab kesalahan input dan menyesuaikan kembali jumlah bahan yang dimasukkan sesuai dengan ketersediaan stok.

Dengan adanya tampilan peringatan ini, pengguna dapat melakukan koreksi data secara langsung sebelum melanjutkan proses pencatatan pesanan. Mekanisme ini membantu mencegah terjadinya kesalahan perhitungan kebutuhan bahan, kesalahan estimasi produksi, serta potensi ketidaksesuaian antara data pesanan dan kondisi persediaan bahan di lapangan. Selain itu, fitur peringatan ini juga meningkatkan keandalan sistem dalam menjaga konsistensi data dan mendukung pengambilan keputusan yang lebih akurat selama proses input pesanan.




Gambar 3.14. Tampilan Figma Laman Detail Pesanan Peringatan

Di bagian bawah laman terdapat tombol “OKE” yang digunakan untuk mengonfirmasi seluruh data yang telah dimasukkan seperti pada Gambar 3.15. Setelah tombol ini ditekan, sistem akan memproses data yang telah diinput dan melanjutkan ke tahap berikutnya dalam alur pencatatan pesanan. Dengan adanya laman ini, proses penginputan data produksi dapat dilakukan secara terstruktur, terukur, dan terintegrasi dalam satu tampilan.




Gambar 3.15. Tampilan Figma Laman Detail Pesanan Selesai Mengisi

Setelah pengguna menyelesaikan seluruh proses penginputan dan melakukan konfirmasi pada halaman akhir, sistem akan menampilkan laman hasil pembuatan pesanan seperti yang ditunjukkan pada Gambar 3.16. Laman ini berfungsi sebagai halaman umpan balik (*feedback*) yang menandakan bahwa proses pencatatan pesanan telah berhasil dilakukan oleh sistem.

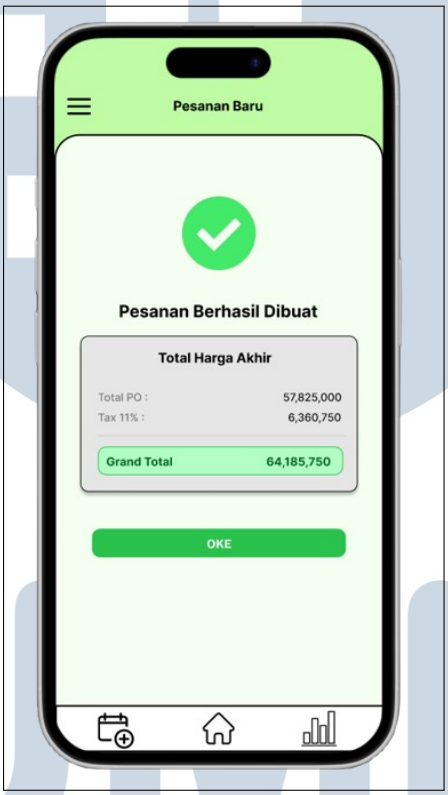
Pada bagian tengah laman ditampilkan indikator keberhasilan berupa ikon tanda centang berwarna hijau yang disertai dengan teks “Pesanan Berhasil Dibuat”. Elemen visual ini bertujuan untuk memberikan konfirmasi secara jelas kepada pengguna bahwa seluruh data pesanan telah tersimpan dan diproses tanpa kendala.

Di bawah indikator keberhasilan, ditampilkan ringkasan informasi total harga pesanan dalam sebuah komponen kartu dengan judul “Total Harga Akhir”. Informasi yang disajikan mencakup rincian Total *PO*, nilai pajak sebesar 11%, serta *Grand Total* yang merupakan hasil penjumlahan antara total pesanan dan pajak. Nilai Grand Total ditampilkan dengan penekanan visual berupa latar berwarna hijau untuk menegaskan jumlah akhir yang harus diperhatikan oleh pengguna.

Penyajian rincian harga dan pajak pada laman ini bertujuan untuk memberikan transparansi perhitungan biaya kepada pengguna, sehingga pengguna dapat memastikan kesesuaian antara data pesanan, nilai pajak yang dikenakan, dan total

biaya akhir sebelum pesanan dilanjutkan ke proses berikutnya di luar sistem.

Pada bagian bawah laman terdapat tombol “OKE” yang berfungsi sebagai aksi akhir untuk menutup tampilan ringkasan pesanan. Setelah tombol ini ditekan, pengguna akan diarahkan kembali ke halaman utama atau halaman lain sesuai dengan alur navigasi aplikasi. Dengan adanya laman ini, sistem tidak hanya memastikan keberhasilan proses pencatatan pesanan, tetapi juga meningkatkan kejelasan informasi dan pengalaman pengguna melalui penyajian ringkasan data yang terstruktur dan mudah dipahami.




Gambar 3.16. Tampilan Figma Laman Pesanan Berhasil

## B Perancangan Desain Laman Histori Pesanan Selesai

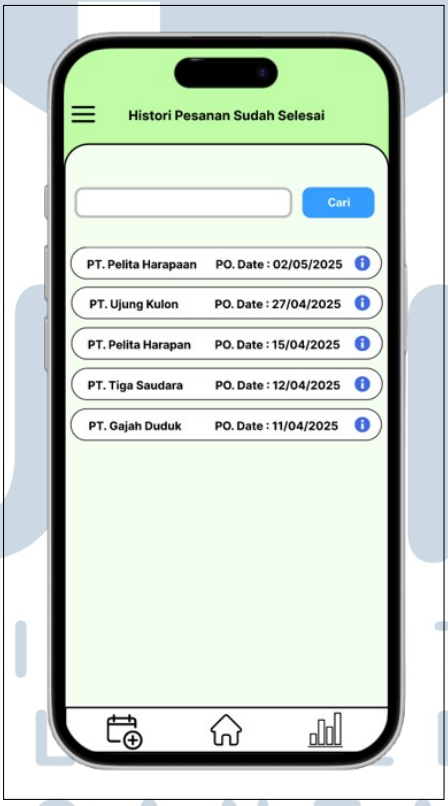
Laman Histori Pesanan Sudah Selesai seperti yang ditunjukkan pada Gambar 3.17 merupakan laman yang digunakan untuk menampilkan daftar pesanan yang telah selesai diproses. Laman ini berfungsi sebagai media bagi pengguna untuk melihat kembali riwayat pesanan yang telah diselesaikan oleh sistem dalam periode tertentu.

Pada bagian atas laman ditampilkan judul “Histori Pesanan Sudah Selesai” yang berperan sebagai penanda konteks halaman. Tepat di bawah judul tersebut,

terdapat komponen pencarian yang terdiri dari kolom input teks dan tombol “Cari”. Fitur pencarian ini memungkinkan pengguna untuk mencari data pesanan tertentu berdasarkan kata kunci, seperti nama perusahaan atau informasi pesanan lainnya, sehingga memudahkan proses penelusuran data histori.

Di bagian tengah laman ditampilkan daftar pesanan yang telah selesai dalam bentuk list. Setiap item pada daftar menampilkan informasi utama berupa nama perusahaan pemesan serta tanggal pembuatan pesanan (PO Date). Informasi ini disajikan secara ringkas agar pengguna dapat dengan cepat mengidentifikasi pesanan yang diinginkan.

Pada sisi kanan setiap item daftar terdapat ikon informasi yang dapat ditekan oleh pengguna. Ikon ini berfungsi untuk menampilkan detail lebih lanjut dari pesanan yang dipilih, seperti rincian barang, total harga, dan informasi pendukung lainnya. Dengan adanya ikon ini, pengguna dapat mengakses detail pesanan tanpa harus berpindah konteks dari daftar histori.




Gambar 3.17. Tampilan Figma Laman Histori Pesanan Selesai

Pada Gambar 3.18 ditunjukkan tampilan laman Histori Pesanan Sudah Selesai setelah pengguna menggunakan fitur pencarian untuk mencari data pesanan

berdasarkan nama perusahaan secara spesifik. Tampilan ini merepresentasikan kondisi sistem setelah pengguna memasukkan nama perusahaan yang diinginkan dan menekan tombol Cari.

Setelah proses pencarian dilakukan, sistem akan menampilkan hasil pencarian dalam bentuk daftar atau card yang berisi histori pesanan sesuai dengan nama perusahaan yang dicari. Pada contoh tampilan, data histori pesanan yang ditampilkan merupakan pesanan dari perusahaan yang sesuai dengan kata kunci pencarian yang dimasukkan oleh pengguna. Penyajian hasil pencarian ini bertujuan untuk memfokuskan tampilan hanya pada data yang relevan, sehingga memudahkan pengguna dalam melakukan peninjauan histori pesanan.


Gambar 3.18. Tampilan Figma Laman Histori Pesanan Selesai Cari Nama PT

Pada Gambar 3.19 ditunjukkan tampilan halaman detail histori pesanan yang muncul setelah pengguna menekan ikon informasi pada salah satu data histori pesanan selesai. Halaman ini berfungsi untuk menampilkan informasi pesanan secara lebih rinci dan lengkap berdasarkan pesanan yang dipilih sebelumnya.

Pada bagian atas halaman ditampilkan nama perusahaan pemesan sebagai identitas utama pesanan. Di bawahnya terdapat indikator progres penyelesaian

pesanan yang menunjukkan status pesanan telah selesai sepenuhnya. Informasi ini memberikan gambaran visual kepada pengguna mengenai tingkat penyelesaian proses pesanan.

Selanjutnya, halaman ini menyajikan informasi detail pesanan yang meliputi tanggal pembuatan pesanan (Tanggal *PO*), tanggal jatuh tempo (Tanggal Tempo), jumlah barang gagal produksi, nama pengirim barang, tanggal pengiriman, serta total biaya yang telah dibayarkan. Informasi tersebut disusun secara terstruktur untuk memastikan kejelasan data dan memudahkan pengguna dalam melakukan peninjauan terhadap riwayat pesanan yang telah diselesaikan.

Pada bagian berikutnya terdapat subbagian Detail Barang yang menampilkan jenis barang yang dipesan beserta jumlah total pesanan dalam satuan unit (*PCS*). Penyajian data ini bertujuan untuk memberikan gambaran ringkas mengenai objek pesanan tanpa harus menelusuri informasi produksi secara terpisah.

Selain itu, halaman ini juga menampilkan subbagian Histori Pengerjaan yang berisi catatan tahapan pengerjaan pesanan berdasarkan tanggal tertentu. Setiap entri pada bagian ini menunjukkan jenis barang, tanggal pengerjaan, serta jumlah barang yang diproses pada waktu tersebut. Informasi histori pengerjaan ini berfungsi sebagai dokumentasi proses produksi dan menjadi referensi bagi pengguna dalam memantau distribusi pengerjaan pesanan.


Gambar 3.19. Tampilan Figma Laman Detail Histori Pesanan Selesai

## C Perancangan Desain Laman Akumulasi Omset

Laman Akumulasi Omset seperti yang ditunjukkan pada Gambar 3.20 merupakan laman yang digunakan untuk menampilkan rekapitulasi total omset perusahaan dalam periode tertentu. Laman ini dirancang untuk memberikan gambaran ringkas mengenai pencapaian omset secara keseluruhan serta distribusi omset berdasarkan bulan.

Pada bagian atas laman ditampilkan judul "Akumulasi Omset" yang berfungsi sebagai penanda konteks halaman. Tepat di bawah judul tersebut terdapat informasi total akumulasi omset yang ditampilkan dalam bentuk nominal mata uang. Nilai ini merupakan hasil penjumlahan dari seluruh omset bulanan yang tercatat dalam sistem, sehingga pengguna dapat dengan cepat mengetahui total omset secara keseluruhan tanpa harus melakukan perhitungan manual.

Di bagian tengah laman ditampilkan daftar omset per bulan dalam bentuk card. Setiap card memuat informasi periode bulan dan tahun, serta nominal omset yang diperoleh pada periode tersebut. Penyajian data per bulan ini bertujuan untuk

memudahkan pengguna dalam melakukan pemantauan dan perbandingan omset antar periode secara lebih terstruktur.

Pada setiap card omset bulanan juga disediakan ikon informasi yang berfungsi sebagai akses menuju detail omset pada periode terkait seperti pada Gambar 3.21. Melalui ikon tersebut, pengguna dapat melihat informasi lanjutan mengenai rincian omset yang membentuk nilai pada bulan tersebut. Selain itu, terdapat pula ikon hapus yang berfungsi untuk menghapus data omset pada periode tertentu sesuai dengan kebutuhan pengelolaan data oleh pengguna.


Gambar 3.20. Tampilan Figma Laman Akumulasi Omset

Pada Gambar 3.21, ditunjukkan tampilan halaman detail akumulasi omset yang muncul setelah pengguna menekan ikon informasi pada salah satu data omset bulanan. Halaman ini berfungsi untuk menampilkan rincian omset berdasarkan periode bulan yang dipilih oleh pengguna.

Pada bagian atas halaman ditampilkan informasi periode bulan dan tahun beserta total omset pada periode tersebut. Informasi ini disajikan secara ringkas untuk memberikan gambaran umum mengenai nilai omset yang dihasilkan dalam satu periode tertentu sebelum pengguna meninjau rincian data lebih lanjut.

Di bagian tengah halaman ditampilkan daftar pesanan yang berkontribusi terhadap total omset pada periode tersebut. Setiap entri pada daftar menampilkan nama perusahaan pemesan, tanggal pembuatan pesanan , serta nominal omset yang dihasilkan dari pesanan tersebut. Penyajian data ini bertujuan untuk memberikan transparansi mengenai sumber pembentuk nilai omset bulanan.

Daftar rincian omset disusun dalam bentuk card agar informasi mudah dibaca dan dibedakan antar entri. Dengan penyajian tersebut, pengguna dapat dengan cepat melakukan peninjauan terhadap kontribusi masing-masing perusahaan terhadap total omset pada bulan yang bersangkutan.


Gambar 3.21. Tampilan Figma Laman Detail Akumulasi Omset

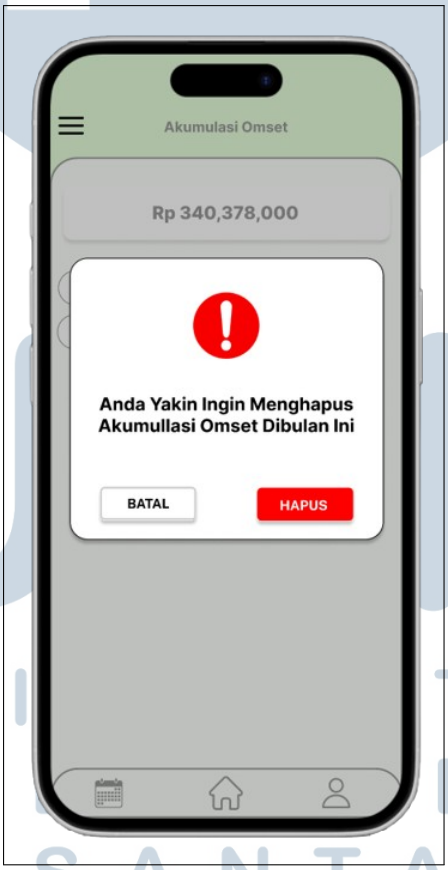
Pada Gambar 3.22, ditunjukkan tampilan dialog konfirmasi yang muncul ketika pengguna menekan ikon hapus pada salah satu data akumulasi omset bulanan. Dialog ini berfungsi sebagai mekanisme pengamanan sistem untuk mencegah terjadinya penghapusan data secara tidak disengaja.

Pada dialog konfirmasi tersebut ditampilkan pesan peringatan yang menyatakan bahwa pengguna akan menghapus data akumulasi omset pada periode

bulan tertentu. Pesan ini dirancang untuk memastikan bahwa pengguna memahami konsekuensi dari tindakan yang akan dilakukan sebelum proses penghapusan data dieksekusi oleh sistem.

Dialog konfirmasi menyediakan dua pilihan tindakan, yaitu tombol Batal dan tombol Hapus. Tombol Batal berfungsi untuk membatalkan proses penghapusan dan mengembalikan pengguna ke tampilan laman Akumulasi Omset tanpa adanya perubahan data. Sementara itu, tombol Hapus berfungsi untuk melanjutkan proses penghapusan data akumulasi omset sesuai dengan periode yang dipilih oleh pengguna.

Penggunaan dialog konfirmasi ini bertujuan untuk meningkatkan keamanan dan integritas data dalam sistem. Dengan adanya tahapan konfirmasi sebelum penghapusan data dilakukan, sistem dapat meminimalkan risiko kehilangan data yang bersifat penting atau bernilai strategis.




Gambar 3.22. Tampilan Figma Laman Hapus Akumulasi Omset

## 3.5 Hasil Implementasi Aplikasi

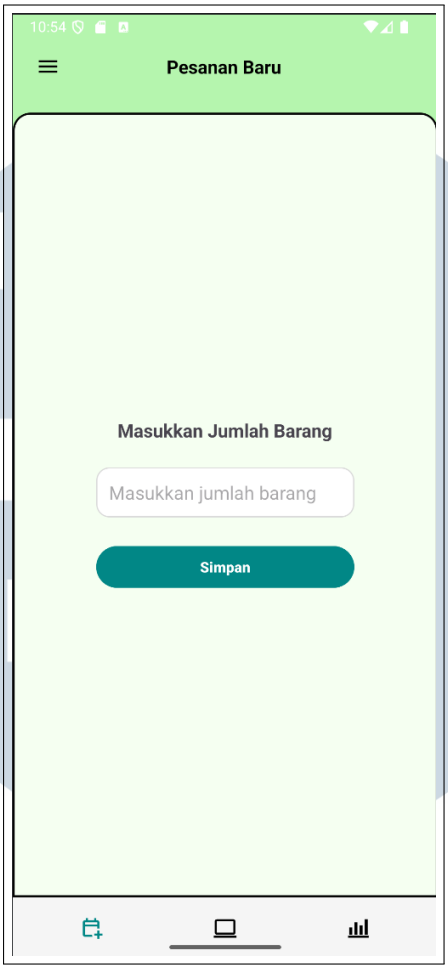
Pada bagian ini, dilakukan pembahasan mengenai hasil implementasi aplikasi pencatatan pesanan yang telah dikembangkan untuk PT Tri Daya Langgeng. Implementasi ini mencakup realisasi dari perancangan yang telah disusun sebelumnya ke dalam bentuk aplikasi Android yang dapat digunakan secara langsung.

Pada subbab ini akan dijelaskan tampilan antarmuka dari setiap fitur utama yang telah diimplementasikan, meliputi Laman Pesanan Baru, Laman Histori Pesanan Selesai, dan Laman Akumulasi Omset. Selain itu, pembahasan juga mencakup penjelasan kode program yang digunakan pada masing-masing laman, guna menunjukkan bagaimana fungsi dan alur aplikasi diimplementasikan secara teknis.

### 3.5.1 Tampilan Laman Pesanan Baru

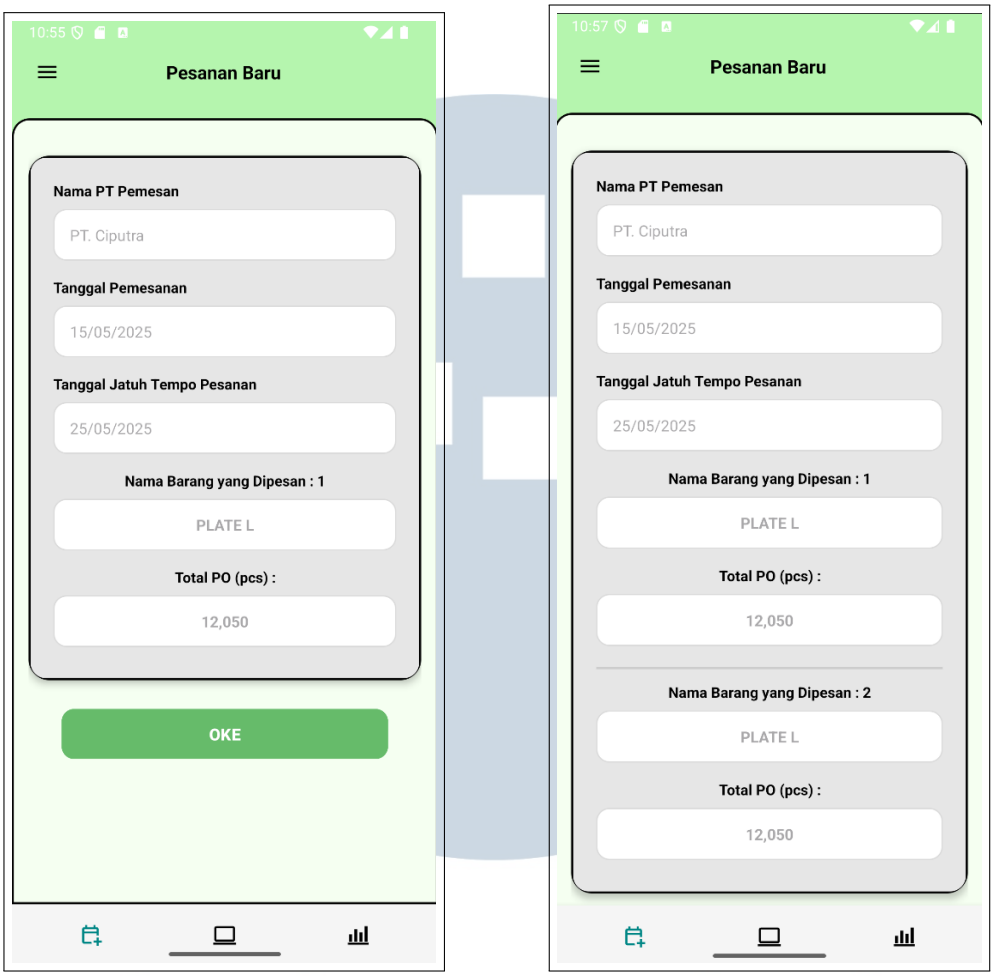
Implementasi sistem pemesanan barang telah dilakukan melalui beberapa tahap perancangan dan pengembangan antarmuka yang saling terintegrasi. Sistem ini dirancang untuk mendukung proses pemesanan mulai dari *input* data awal, pengisian detail pesanan, pemilihan material, hingga tahap konfirmasi akhir yang menampilkan hasil perhitungan otomatis. Setiap laman dalam aplikasi memiliki fungsi yang spesifik dan telah disusun berdasarkan alur kerja pengguna (*user flow*) yang logis dan sistematis agar proses pemesanan dapat dilakukan dengan lebih efisien dan minim kesalahan. Pada Gambar 3.23 adalah tampilan aplikasi saat tombol *Add Order* pada bagian bawah kiri aplikasi ditekan.

Pada tahap awal penggunaan, sistem menampilkan laman untuk memasukkan jumlah jenis barang yang akan dipesan. Data ini berfungsi sebagai dasar untuk menentukan jumlah *form* yang akan dihasilkan pada tahap selanjutnya, sehingga pengguna dapat memasukkan detail setiap barang yang dipesan secara terstruktur. Tampilan laman ini dibuat sederhana untuk memastikan proses *input* berlangsung cepat dan mudah dipahami oleh pengguna.




Gambar 3.23. Laman Awal Pesanan Baru

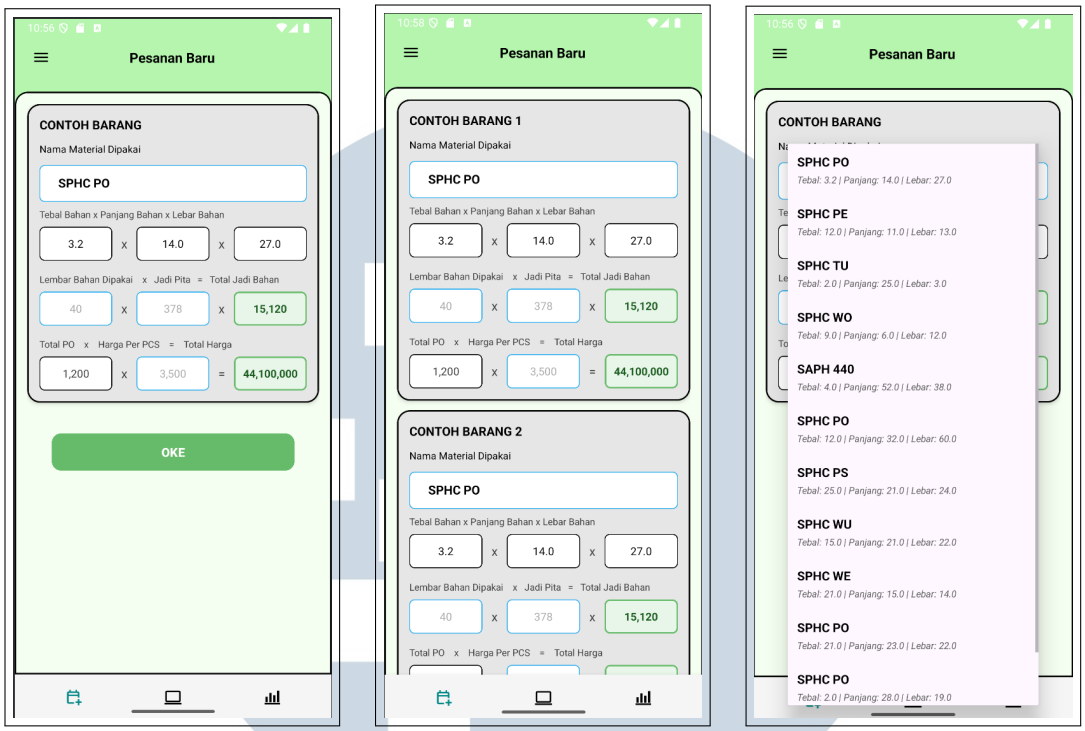
Setelah menentukan jumlah jenis barang, pengguna diarahkan menuju laman pengisian informasi terkait perusahaan pemesan seperti pada Gambar 3.24. *Form* yang disediakan mencakup nama perusahaan, tanggal pemesanan, tanggal jatuh tempo pemesanan, nama barang yang dipesan, serta total *PO* (*pieces*) yang dibutuhkan seperti pada Gambar 3.24a dan saat pengguna melakukan pemesanan untuk lebih dari 1 barang maka tampilan laman pesanan baru akan berubah seperti pada Gambar 3.24b. Antarmuka laman ini disusun dalam bentuk komponen-komponen yang terkelompok sehingga setiap informasi dapat diisi secara berurutan dan sistematis. Selain itu, aplikasi juga memberikan penanda urutan barang, misalnya "Nama Barang yang Dipesan: 1", untuk membantu pengguna mengetahui item mana yang sedang diproses ketika jumlah barang lebih dari satu.




(a) Laman Detail Perusahaan 1 Pesanan (b) Laman Detail Perusahaan 2 Pesanan

Gambar 3.24. Laman Pesanan Baru Detail Perusahaan

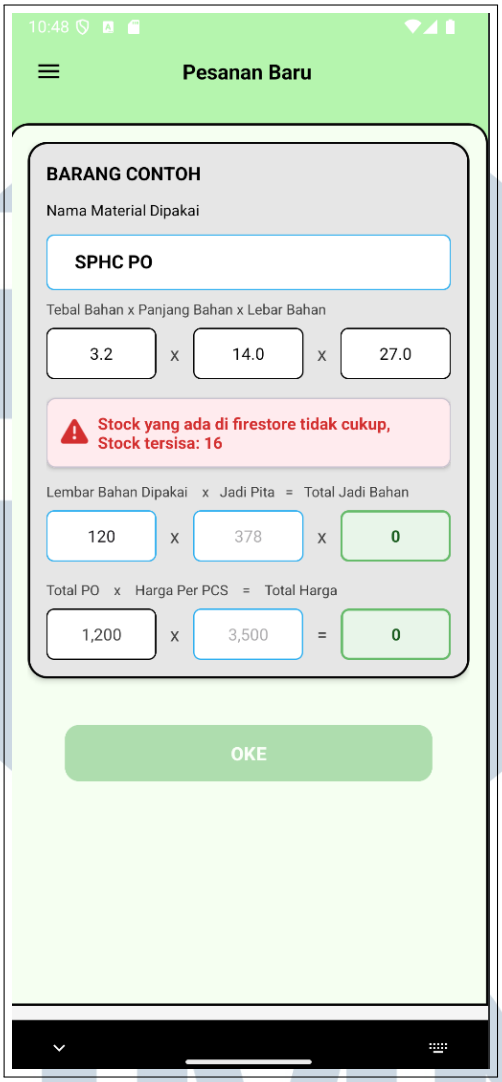
Tahap selanjutnya adalah pemilihan material yang akan digunakan untuk masing–masing barang (Gambar 3.25). Pada laman ini, pengguna dapat memilih material melalui komponen *spinner* yang menampilkan daftar material beserta spesifikasinya seperti pada Gambar 3.25c. Setelah material dipilih, aplikasi melakukan pembaruan otomatis terhadap nilai tebal bahan, panjang bahan, dan lebar bahan berdasarkan data material tersebut, seperti pada Gambar 3.25a dan jika di laman sebelumnya pengguna memilih untuk memesan lebih dari 1 barang maka aplikasi akan menyesuaikan *card*nya sesuai dengan jumlah pesanan yang *input* Gambar 3.25b. Pembaruan otomatis ini bertujuan untuk menghilangkan potensi kesalahan input manual sekaligus mempercepat proses pengisian data. Pengguna kemudian diminta untuk memasukkan jumlah lembar bahan yang digunakan, jumlah jadi pita, serta harga per *pieces*.




(a) Laman Pemilihan Material Satu Pesanan (b) Laman Pemilihan Material Dua Pesanan (c) Spinner Pemilihan Material

Gambar 3.25. Laman Pemilihan Material

Saat pengguna melakukan pengisian jumlah lembar yang dipakai, sistem akan melakukan pengecekan dengan *database* untuk melakukan verifikasi jika jumlah lembar yang di input pengguna melebihi *stock* yang tersedia atau tidak, jika melebihi maka sistem akan menampilkan *alert* yang memberitahu pada pengguna bahwa input pada jumlah lembar melebihi dari *stock* yang ada pada *database*. *Alert* ini dapat dilihat pada Gambar 3.26




Gambar 3.26. Alert Laman Pemilihan Material

Setelah seluruh data terisi, sistem melakukan kalkulasi otomatis untuk menghitung total jadi bahan, total harga berdasarkan total *PO* dan harga per *pieces*, serta memberikan tampilan nilai perhitungan secara langsung pada laman yang sama. Pemberian nilai secara *real-time* ini memungkinkan pengguna memverifikasi kesesuaian perhitungan sebelum melanjutkan ke tahap berikutnya.

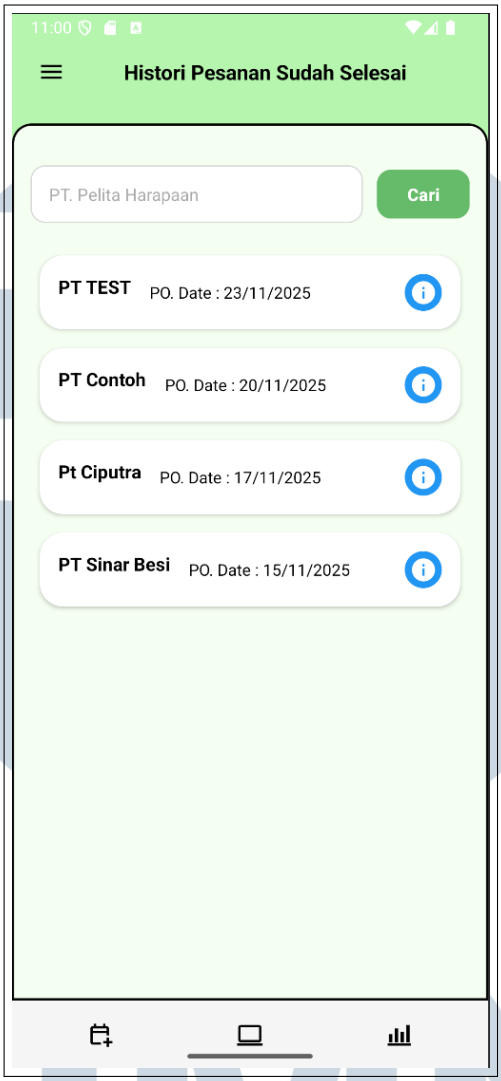
Pada laman akhir, sistem menampilkan ringkasan keseluruhan pesanan yang telah di*input* oleh pengguna. Ringkasan tersebut mencakup total *PO*, nilai pajak sebesar sebelas persen dari total *PO*, serta *grand total* yang merupakan penjumlahan antara total harga dan total pajak. Dengan adanya proses kalkulasi otomatis ini, pengguna tidak perlu melakukan perhitungan manual sehingga mengurangi risiko kesalahan dan meningkatkan keakuratan data pemesanan. Setelah seluruh

informasi ditampilkan, pengguna dapat mengonfirmasi pemesanan melalui tombol yang telah disediakan, sehingga seluruh proses pemesanan dapat diselesaikan secara terstruktur dan efisien melalui aplikasi seperti pada Gambar 3.27.


Gambar 3.27. Laman Akhir Pesanan Baru

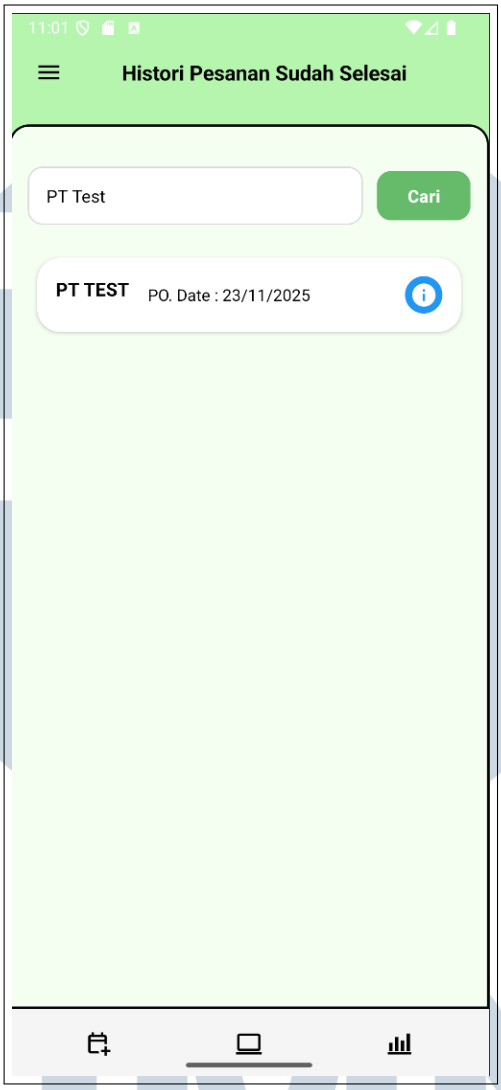
### 3.5.2 Tampilan Laman Histori Pesanan Selesai

Implementasi fitur Histori Pesanan Selesai pada aplikasi telah mampu menyediakan informasi mengenai daftar perusahaan yang memiliki pesanan yang telah selesai diproduksi. Informasi tersebut ditampilkan dalam bentuk daftar sehingga mempermudah pengguna dalam melakukan pemantauan terhadap pesanan yang telah terselesaikan (Gambar 3.28).




Gambar 3.28. Laman Histori Pesanan Selesai

Pada bagian atas laman disediakan *search bar* yang berfungsi untuk melakukan pencarian perusahaan tertentu berdasarkan nama. Setelah pengguna memasukkan kata kunci dan menekan tombol pencarian, aplikasi akan menyaring data dan hanya menampilkan perusahaan yang sesuai dengan kriteria pencarian. Fitur ini membantu pengguna dalam menemukan data pesanan secara lebih cepat dan efisien (Gambar 3.29).

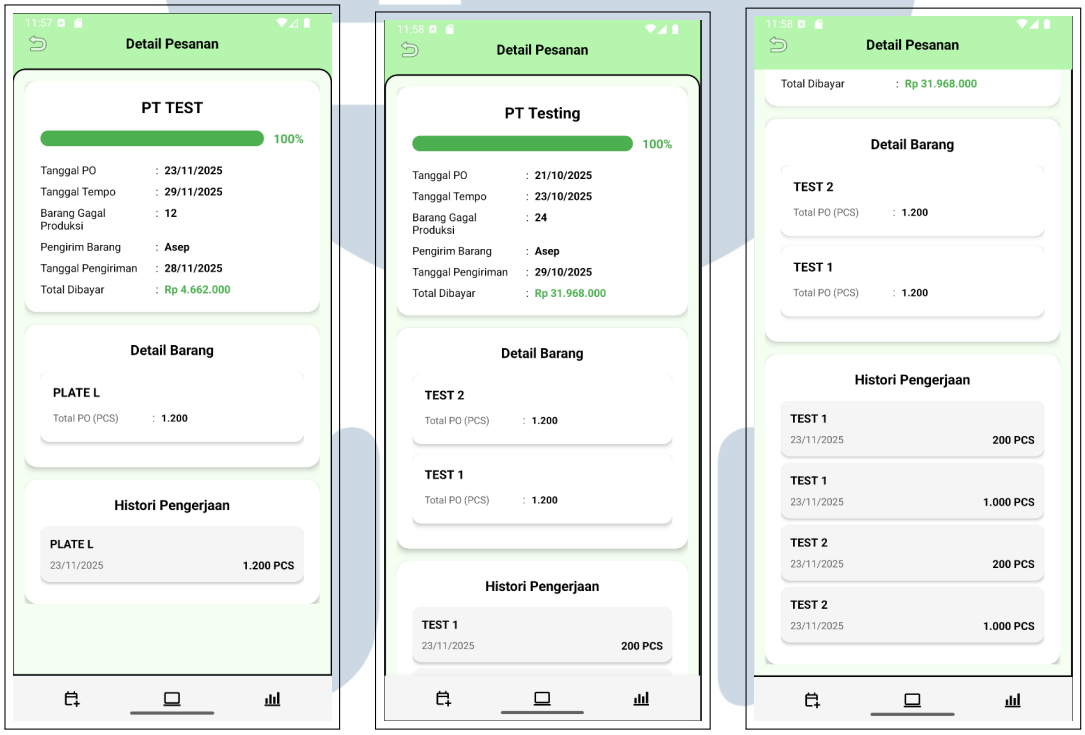



Gambar 3.29. Search Bar Histori Pesanan Selesai

Setiap entri pada daftar menampilkan nama perusahaan pemesan serta tanggal *Purchase Order (PO)*. Ketika pengguna memilih salah satu perusahaan, aplikasi akan menampilkan laman Detail Pesanan yang memuat informasi lengkap mengenai pesanan tersebut seperti pada Gambar 3.30. Informasi yang disajikan mencakup identitas perusahaan, tingkat penyelesaian pesanan yang ditampilkan dalam bentuk *progress bar*, tanggal *PO*, tanggal jatuh tempo, jumlah barang yang tidak lolos proses produksi, nama pihak pengirim barang, tanggal pengiriman, serta total pembayaran yang telah diselesaikan oleh pihak pemesan seperti pada Gambar 3.30a dan jika perusahaan tersebut melakukan pemesanan untuk lebih dari satu barang maka tampilan di aplikasi akan menyesuaikan dengan jumalah barang yang dipesan, tampilan ini dapat dilihat pada Gambar 3.30b.

Selain itu, laman ini juga menampilkan rincian barang yang dipesan oleh perusahaan. Setiap jenis barang disajikan dalam kartu informasi tersendiri yang memuat nama barang serta jumlah total pesanan dalam satuan *pieces*. Penyajian ini bertujuan memberikan gambaran yang lebih jelas mengenai jenis dan jumlah barang yang terkait dengan pesanan tersebut.

Pada bagian bawah laman, aplikasi menampilkan Histori Pengerjaan, yaitu catatan progres produksi dari barang-barang yang dipesan . Informasi yang ditampilkan meliputi tanggal proses pengerjaan serta jumlah barang yang berhasil diproduksi pada tanggal tersebut (Gambar 3.30c). Histori ini ditampilkan secara kronologis sehingga memberikan pemahaman yang komprehensif mengenai alur produksi hingga mencapai penyelesaian penuh.




(a) Tampilan Laman Histori Selesai Satu Pesanan

(b) Tampilan Laman Histori Selesai Dua Pesanan

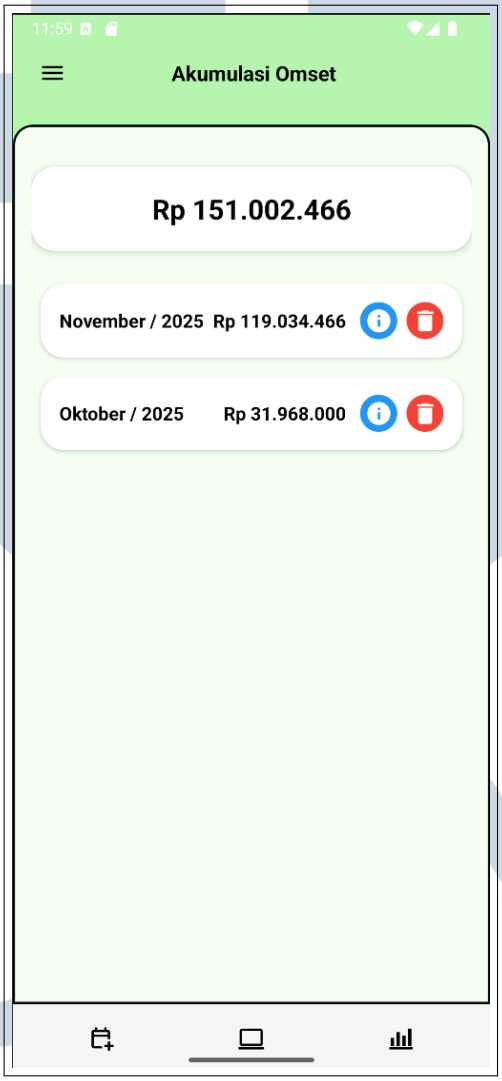
(c) Tampilan Laman Histori Histori Pengerjaan

Gambar 3.30. Detail Pesanan Selesai Satu dan Dua Pesanan

### 3.5.3 Laman Akumulasi Omset

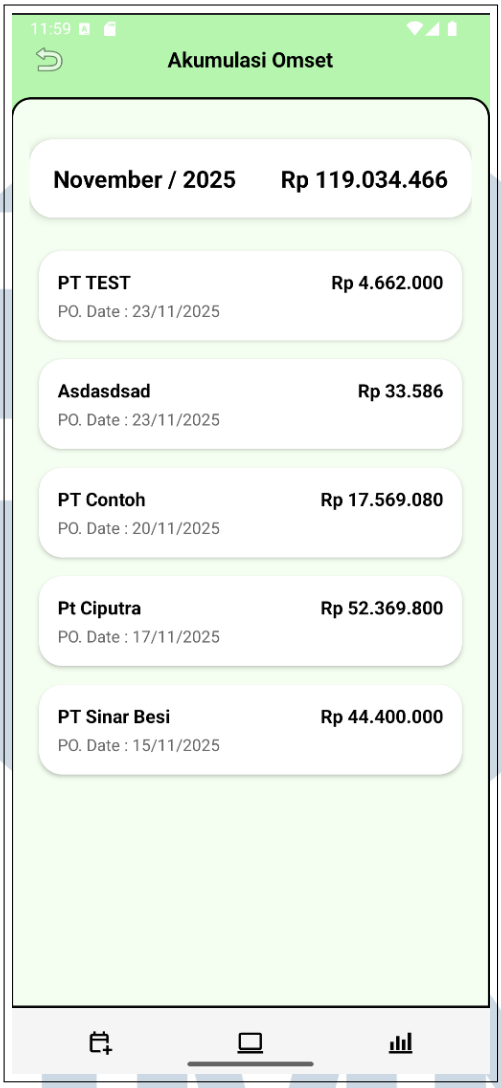
Implementasi fitur Akumulasi Omset pada aplikasi bertujuan untuk membantu pengguna dalam melakukan pemantauan omset bulanan secara terstruktur dan komprehensif. Ketika pengguna melakukan navigasi menuju laman ini, aplikasi

menampilkan daftar omset yang sudah dirangkum berdasarkan bulan. Setiap data bulan berisi total omset yang berhasil dicapai pada periode tersebut, dan seluruh total bulanan tersebut dihitung kembali sehingga menghasilkan nilai akumulasi keseluruhan yang ditampilkan pada bagian atas laman seperti pada Gambar 3.31.



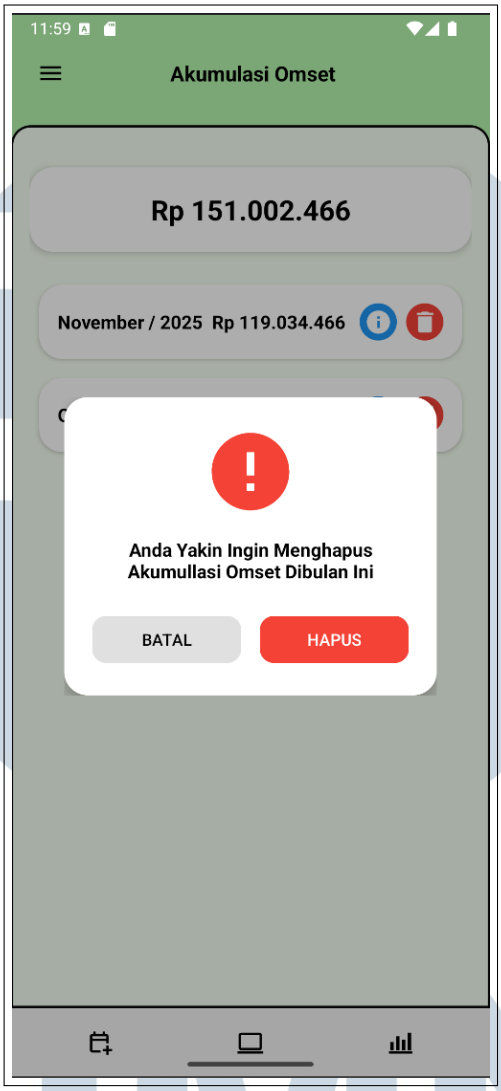

Gambar 3.31. Laman Akumulasi Omset

Setiap data bulan disajikan dalam bentuk kartu yang dapat dipilih oleh pengguna. Ketika salah satu periode bulan ditekan, aplikasi akan menampilkan laman yang memuat detail pesanan-pesanan yang telah diselesaikan pada bulan tersebut. Informasi yang ditampilkan mencakup nama perusahaan pemesan (PT), tanggal pemesanan, serta total harga dari pesanan yang telah diselesaikan. Tampilan ini disusun agar memudahkan pengguna dalam menelusuri sumber omset yang berkontribusi pada pendapatan bulan tersebut (Gambar 3.32).




Gambar 3.32. Detail Akumulasi Omset

Untuk mendukung pengelolaan data, aplikasi juga menyediakan opsi penghapusan omset pada salah satu bulan. Ketika pengguna memilih fungsi penghapusan, sistem akan menampilkan *pop-up warning* sebagai bentuk konfirmasi sebelum tindakan dilakukan. Mekanisme ini diterapkan untuk mencegah kesalahan penghapusan data yang bersifat penting dan menjaga integritas informasi di dalam aplikasi. Pengguna hanya dapat melanjutkan proses penghapusan apabila telah menyetujui peringatan tersebut (Gambar 3.33).




Gambar 3.33. Delete Akumulasi Omset Pop Up

## 3.6 Potongan Kode Aplikasi

### 3.6.1 Potongan Kode Laman Pesanan Baru

Pada bagian ini akan dijelaskan potongan kode dari awal pemesanan barang baru sampai konfirmasi pemesanan barang baru. Potongan kode untuk laman awal pesanan baru dapat di lihat pada 3.1

```
1 btnSimpan.setOnClickListener {
2             val total = edtTotalBarang.text.toString().toIntOrNull
    () ?: 0
3             if (total <= 0) {
4                 Toast.makeText(this, "Masukkan jumlah barang
    dengan benar", Toast.LENGTH_SHORT).show()
5             } else {
```

```
                val intent = Intent(this, AddOrderActivity::class.
    java)
                intent.putExtra("ITEM_COUNT", total)
                startActivity(intent)
            }
        }
```

Kode 3.1: Function btnSimpan

Fungsi *btnSimpan.setOnClickListener()* berperan sebagai mekanisme utama untuk memproses *input* awal yang diberikan oleh pengguna pada laman penentuan jumlah barang. Ketika tombol Simpan ditekan, sistem terlebih dahulu mengambil nilai yang diinput pada komponen *EditText*, kemudian melakukan konversi ke dalam tipe bilangan bulat melalui metode *toIntOrNull()*. Proses ini disertai validasi untuk memastikan bahwa nilai yang dimasukkan bukan nilai kosong maupun nilai yang tidak dapat dikonversi menjadi angka. Apabila hasil validasi menunjukkan nilai kurang dari atau sama dengan nol, aplikasi menampilkan pesan peringatan melalui *Toast* untuk memberitahukan bahwa *input* tidak valid. Sebaliknya, apabila *input* dinyatakan valid, sistem membentuk objek *Intent* untuk melakukan navigasi menuju *AddOrderActivity*. Pada *intent* tersebut, sistem menyertakan data tambahan berupa jumlah barang yang ingin dipesan melalui metode *putExtra("ITEM_COUNT", total)*. Dengan demikian, fungsi ini tidak hanya bertugas menangani interaksi pengguna terhadap tombol, tetapi juga memastikan integritas data yang akan digunakan pada tahapan proses berikutnya dalam alur pemesanan barang.

Setelah melakukan *input* jumlah barang yang ingin dipesan pengguna akan melanjutkan pesanan di laman selanjutnya dengan melakukan *input* terkait informasi pemesanan barang baru, potongan kode ini dapat dilihat pada Kode 3.2

```
private fun generateBarangForms() {
        containerBarang.removeAllViews()
        barangDataList.clear()

        val inflater = LayoutInflater.from(this)

        for (i in 0 until totalBarang) {
            val barangView = inflater.inflate(R.layout.
    item_input_barang_form, containerBarang, false)

            val tvBarangTitle = barangView.findViewById<TextView>(
    R.id.tvBarangTitle)
            val etItemName = barangView.findViewById<EditText>(R.
    id.etItemName)
            val etTotalOrder = barangView.findViewById<EditText>(R
    .id.etTotalOrder)
            val divider = barangView.findViewById<View>(R.id.
    dividerBarang)

            // Set title
```

```
            tvBarangTitle.text = "Nama Barang yang Dipesan : ${i + 1}"

            // Tampilkan divider hanya untuk item ke-2 dan seterusnya
            divider.visibility = if (i > 0) View.VISIBLE else View.GONE

            // Inisialisasi data
            val barangData = BarangInputData()
            barangDataList.add(barangData)

            // Set listener untuk auto-save data
            etItemName.setOnFocusChangeListener { _, hasFocus ->
                if (!hasFocus) {
                    barangData.itemName = etItemName.text.toString().trim()
                }
            }

            etTotalOrder.setOnFocusChangeListener { _, hasFocus ->
                if (!hasFocus) {
                    barangData.totalOrder = etTotalOrder.text.toString().toIntOrNull() ?: 0
                }
            }

            containerBarang.addView(barangView)
        }
    }
```

Kode 3.2: Function generateBarangForms

Fungsi *generateBarangForms()* bertanggung jawab untuk membangun dan menampilkan *form input* dinamis sesuai dengan jumlah barang yang telah ditentukan pada tahap sebelumnya. Pada awal pemanggilan fungsi, sistem menghapus seluruh komponen yang mungkin tersisa pada *containerBarang* serta mengosongkan daftar *barangDataList* agar tidak terjadi duplikasi data ketika *form* dipanggil ulang. Selanjutnya, sistem menggunakan objek *LayoutInflater* untuk memuat template tampilan setiap form barang dari berkas layout *item_input_barang_form*. Proses ini dilakukan secara iteratif berdasarkan nilai *totalBarang*, sehingga setiap barang memperoleh form mandiri yang terdiri dari judul, kolom input nama barang, dan kolom input total pesanan.

Judul *form* disesuaikan secara dinamis dengan menampilkan urutan barang, sedangkan komponen pemisah (*divider*) hanya ditampilkan untuk *form* kedua dan seterusnya guna menjaga keteraturan visual antar *form*. Pada setiap iterasi, sistem juga menginisialisasi objek *BarangInputData* yang digunakan untuk menyimpan sementara data *input* pengguna. Untuk meningkatkan konsistensi dan akurasi data, sistem menerapkan mekanisme penyimpanan otomatis (*auto-save*) melalui *listener*

*onFocusChangeListener* pada kedua kolom *input*; ketika pengguna meninggalkan kolom *input*, nilai tersebut langsung disimpan ke dalam objek *barangData*. Setelah seluruh elemen form dipersiapkan, sistem menambahkan tampilan *form* ke dalam kontainer utama. Dengan demikian, fungsi ini berperan penting dalam menghasilkan struktur input dinamis yang adaptif terhadap jumlah barang, sekaligus menjaga validitas data yang dimasukkan oleh pengguna pada tahap awal proses pemesanan.

Setelah pengguna selesai mengisi data informasi perusahaan dan barang, maka pengguna akan diarahkan ke laman selanjutnya untuk melakukan pemilihan material yang akan digunakan untuk setiap barang yang di pesan, potongan kode ini dapat dilihat pada Kode 3.3

```
private fun showMaterialSelection() {
        val inflater = layoutInflater
        container.removeAllViews()
        stockValidationFlags.clear()

        for ((index, barang) in barangList!!.withIndex()) {
            // Inisialisasi flag validasi untuk setiap barang
            stockValidationFlags[index] = true
            val view = inflater.inflate(R.layout.
    item_material_selection, container, false)
            val tvBarang = view.findViewById<TextView>(R.id.
    tvNamaBarang)
            val spinner = view.findViewById<Spinner>(R.id.
    spinnerMaterial)
            val etTebal = view.findViewById<EditText>(R.id.etTebal
    )
            val etPanjang = view.findViewById<EditText>(R.id.
    etPanjang)
            val etLebar = view.findViewById<EditText>(R.id.etLebar
    )
            val etLembar = view.findViewById<EditText>(R.id.
    etLembar)
            val etJadi = view.findViewById<EditText>(R.id.etJadi)
            val tvTotalPO = view.findViewById<TextView>(R.id.
    etTotalPO)
            val etHarga = view.findViewById<EditText>(R.id.
    etHargaPerPcs)
            val tvTotal = view.findViewById<TextView>(R.id.
    tvTotalHarga)
            val tvTotalJadi = view.findViewById<TextView>(R.id.
    tvTotalJadi)
            val cardAlert = view.findViewById<MaterialCardView>(R.
    id.cardAlert)
            val tvAlertMessage = view.findViewById<TextView>(R.id.
    tvAlertMessage)

            tvBarang.text = barang.namaBarang ?: "Barang ${index +
     1}"

            // Tampilkan Total PO dari data barang (otomatis
    terisi)
```

```
            val totalPOValue = barang.estimateOrder.toDouble()
            tvTotalPO.text = "%,.0f".format(totalPOValue)

            // Setup spinner dengan custom adapter
            val adapter = MaterialSpinnerAdapter(this, materials)
            spinner.adapter = adapter

            spinner.onItemSelectedListener = object : AdapterView.
OnItemSelectedListener {
                override fun onItemSelected(parent: AdapterView
<*>?, viewItem: android.view.View?, position: Int, id: Long) {
                    val mat = materials[position]

                    // Update EditText dimensi
                    etTebal.setText(mat.MatsThick?.toString() ?: "
")
                    etPanjang.setText(mat.MatsLength?.toString()
?: "")
                    etLebar.setText(mat.MatsWidth?.toString() ?: "
")

                    barang.materialID = mat.MaterialID

                    // Reset alert saat ganti material
                    cardAlert.visibility = View.GONE
                    stockValidationFlags[index] = true
                    updateButtonState()
                }

                override fun onNothingSelected(parent: AdapterView
<*>?) {}
            }

            // Hitung total otomatis
            val watcher = object : TextWatcher {
                override fun beforeTextChanged(s: CharSequence?,
start: Int, count: Int, after: Int) {}
                override fun onTextChanged(s: CharSequence?, start
: Int, before: Int, count: Int) {}
                override fun afterTextChanged(s: Editable?) {
                    val lembar = etLembar.text.toString().
toDoubleOrNull() ?: 0.0
                    val jadi = etJadi.text.toString().
toDoubleOrNull() ?: 0.0
                    val hargaPerPcs = etHarga.text.toString().
toDoubleOrNull() ?: 0.0

                    // Validasi stock material
                    if (lembar > 0 && !barang.materialID.
isNullOrEmpty()) {
                        val selectedMaterial = materials.find { it
.MaterialID == barang.materialID }
                        val stock = selectedMaterial?.Stock?.
toDouble() ?: 0.0

                        if (lembar > stock) {
                            // Tampilkan alert
                            cardAlert.visibility = View.VISIBLE
                            tvAlertMessage.text = "Stock yang
```

```
    tersedia tidak cukup, Stock tersisa: %,.0f".format(stock)
                            stockValidationFlags[index] = false
                        } else {
                            // Sembunyikan alert
                            cardAlert.visibility = View.GONE
                            stockValidationFlags[index] = true
                        }
                    } else {
                        cardAlert.visibility = View.GONE
                        stockValidationFlags[index] = true
                    }

                    // Update status tombol berdasarkan semua
    validasi
                    updateButtonState()

                    val totalJadi = lembar * jadi
                    tvTotalJadi.text = "%,.0f".format(totalJadi)

                    // Ambil Total PO dari barang.estimateOrder
                    val totalPO = barang.estimateOrder.toDouble()
                    val totalHarga = totalPO * hargaPerPcs
                    tvTotal.text = "%,.0f".format(totalHarga)

                    barang.lembar = lembar
                    barang.jadi = jadi
                    barang.hargaPerPcs = hargaPerPcs
                    barang.totalHarga = totalHarga
                }
            }

            etLembar.addTextChangedListener(watcher)
            etJadi.addTextChangedListener(watcher)
            etHarga.addTextChangedListener(watcher)

            container.addView(view)
        }
    }
```

Kode 3.3: Function showMateriailSelection

Fungsi *showMaterialSelection()* merupakan komponen kunci pada tahap pemilihan material dalam proses pemesanan. Fungsi ini bertugas memanggil *form* pemilihan material secara dinamis untuk setiap barang yang telah di*input* pada tahap sebelumnya. Pada awal proses, sistem menghapus seluruh tampilan yang ada pada kontainer utama dan mereset struktur penyimpanan *flag* validasi *stock* untuk memastikan bahwa setiap barang memiliki status validasi yang independen.

Selanjutnya, fungsi melakukan iterasi terhadap daftar barang dan menghasilkan tampilan pemilihan material untuk masing-masing item menggunakan *layout inflater*. Setiap *form* yang dibangkitkan memuat informasi nama barang, *spinner* pemilihan material, kolom dimensi material, jumlah lembar yang akan digunakan, jumlah jadi pita, harga per satuan, serta ringkasan

perhitungan seperti total jumlah jadi dan total harga. *Spinner* material dikonfigurasi menggunakan *Adapter* khusus sehingga mampu menampilkan daftar material secara terstruktur. Ketika pengguna memilih suatu material, sistem secara otomatis mengisi kolom tebal, panjang, dan lebar bahan berdasarkan data material yang dipilih, sekaligus menyimpan informasi material tersebut pada objek barang terkait.

Fungsi ini juga dilengkapi mekanisme *real-time validation* terhadap *stock* material. Proses validasi dilakukan melalui *TextWatcher* yang memantau perubahan *input* pada kolom jumlah lembar, jumlah jadi, dan harga per satuan. Apabila jumlah lembar yang dimasukkan melebihi *stock* material yang tersedia, sistem menampilkan peringatan berupa *alert card* dan menandai bahwa barang tersebut tidak memenuhi syarat validasi. Sebaliknya, apabila *input* sesuai dengan batas stok, peringatan disembunyikan dan status validasi diperbarui. Validasi stok ini sangat penting untuk mencegah pemesanan material melebihi kemampuan inventaris yang tersedia. Selain validasi, *TextWatcher* juga menangani perhitungan otomatis total jumlah jadi serta total harga berdasarkan data pesanan yang telah dimasukkan pengguna. Setiap hasil perhitungan ditampilkan secara langsung pada antarmuka sehingga pengguna dapat segera mengetahui konsekuensi dari setiap perubahan input. Pada akhir proses, seluruh *form* yang telah dipersiapkan ditambahkan ke dalam kontainer utama. Melalui mekanisme ini, fungsi *showMaterialSelection()* tidak hanya menyediakan interface yang fleksibel dan dinamis, tetapi juga memastikan integritas data melalui validasi stok dan perhitungan otomatis yang berlangsung secara simultan sepanjang interaksi pengguna.

Selanjutnya pengguna akan diarahkan ke laman akhir yaitu *Order Summary* untuk melakukan konfirmasi ulang terhadap pesanan yang telah dilakukan, potongan kode ini dapat dilihat pada Kode 3.4

```
1  private fun saveOrderBarangBatch(firestore: FirebaseFirestore,
       orderId: String) {
2          val batch = firestore.batch()
3          val orderRef = firestore.collection("Order").document(
       orderId)
4
5          // Simpan setiap barang ke subcollection OrderBarang dan
       update stock
6          barangList?.forEach { b ->
7              val barangRef = orderRef.collection("OrderBarang").
       document()
8              val data = hashMapOf(
9                  "OrderBarangID" to barangRef.id,
10                 "OrderName" to b.namaBarang,
11                 "EstimateOrder" to b.estimateOrder,
12                 "MaterialID" to b.materialID,
13                 "Lembar" to b.lembar,
14                 "Jadi" to b.jadi,
```

```
                "HargaPerPcs" to b.hargaPerPcs,
                "TotalHarga" to b.totalHarga
            )
            batch.set(barangRef, data)

            // Update stok material (mengurangi sesuai jumlah lembar)
            if (!b.materialID.isNullOrEmpty() && b.lembar != null) {
                val materialRef = firestore.collection("Material").document(b.materialID!!)
                batch.update(materialRef, "Stock", FieldValue.increment(-b.lembar!!))
            }
        }

        // Commit batch
        batch.commit()
            .addOnSuccessListener {
                Snackbar.make(btnConfirmOrder, "Pesanan berhasil dikonfirmasi!", Snackbar.LENGTH_SHORT).show()
                btnConfirmOrder.postDelayed({
                    val intent = Intent(this, DashboardActivity::class.java)
                    intent.flags = Intent.FLAG_ACTIVITY_CLEAR_TOP or Intent.FLAG_ACTIVITY_NEW_TASK
                    startActivity(intent)
                    finish()
                }, 1000)
            }
            .addOnFailureListener { e ->
                btnConfirmOrder.isEnabled = true
                Toast.makeText(this, "Gagal menyimpan detail barang: ${e.message}", Toast.LENGTH_SHORT).show()
            }
    }
```

Kode 3.4: Function saveOrderBarangBatch

Fungsi *saveOrderBarangBatch()* berperan sebagai mekanisme utama dalam tahap akhir proses pemesanan, yaitu melakukan penyimpanan data pesanan secara terstruktur ke dalam database Firestore setelah pengguna melakukan konfirmasi. Fungsi ini memanfaatkan fitur *batch write* dari *Firestore* untuk memastikan bahwa seluruh operasi penyimpanan berlangsung secara atomik, sehingga mencegah terjadinya inkonsistensi data apabila sebagian proses gagal. Pada awal pemrosesan, sistem membentuk objek *batch* dan menentukan referensi dokumen utama pesanan berdasarkan *orderId*. Selanjutnya, fungsi melakukan iterasi terhadap setiap barang yang telah diinput pengguna pada tahap sebelumnya. Untuk setiap barang, sistem membuat dokumen baru dalam subkoleksi *OrderBarang* dan menyimpan atribut penting seperti nama barang, estimasi pesanan, jenis material, jumlah lembar bahan yang digunakan, jumlah jadi pita, harga per satuan, serta total harga yang telah dihitung pada tahap pemilihan material.

Selain penyimpanan data, fungsi ini juga menangani pembaruan *stock* material secara otomatis. Jika sebuah barang memiliki material yang valid beserta nilai lembar yang telah ditentukan, sistem akan mengurangi stok material tersebut menggunakan operasi *increment* negatif pada dokumen material yang bersangkutan. Langkah ini memastikan bahwa inventaris material selalu sesuai dengan penggunaan aktual berdasarkan pesanan yang telah dikonfirmasi. Setelah seluruh operasi ditambahkan ke dalam *batch*, sistem mengeksekusi perintah *commit*. Jika proses berjalan sukses, aplikasi menampilkan notifikasi keberhasilan melalui *Snackbar* dan mengalihkan pengguna kembali ke laman *Dashboard* untuk menyelesaikan alur pemesanan. Namun, jika proses penyimpanan mengalami kegagalan, tombol konfirmasi diaktifkan kembali dan aplikasi menampilkan pesan kesalahan. Melalui mekanisme ini, fungsi *saveOrderBarangBatch()* memastikan integritas data pesanan, sinkronisasi stok material, serta memberikan pengalaman pengguna yang lebih reliabel pada tahap finalisasi.

### 3.6.2 Potongan Kode Laman Histori Pesanan

Potongan kode berikut digunakan pada laman histori pesanan untuk menampilkan daftar pesanan yang telah selesai diproses. Prosedur ini memastikan bahwa data pesanan selesai diambil dari basis data secara akurat, diurutkan berdasarkan tanggal pesanan terbaru, dan kemudian disajikan kepada pengguna melalui komponen antarmuka yang telah disediakan, potongan kode ini dapat dilihat pada Kode 3.5.

```
1  private fun loadFinishedOrders() {
2          firestore.collection("Order")
3              .whereEqualTo("FinishedOrder", true)
4              .get()
5              .addOnSuccessListener { result ->
6                  allOrders.clear()
7                  for (document in result) {
8                      val order = document.toObject(Order::class.
       java)
9                      order.OrderID = document.id
10                     allOrders.add(order)
11                 }
12
13                 // Sort manual di Kotlin berdasarkan OrderDate (
       descending)
14                 allOrders.sortByDescending { it.OrderDate?.toDate
       ()?.time }
15
16                 adapter.submitList(allOrders)
17
18                 if (allOrders.isEmpty()) {
```

```
19                      Toast.makeText(this, "Belum ada pesanan
    selesai", Toast.LENGTH_SHORT).show()
20                  }
21              }
22              .addOnFailureListener { e ->
23                  Toast.makeText(this, "Gagal memuat data: ${e.
    message}", Toast.LENGTH_LONG).show()
24              }
25      }
```

Kode 3.5: Function loadFinishedOrders

Fungsi *loadFinishedOrders()* berfungsi untuk memuat data seluruh pesanan yang berstatus selesai dari *Database Firestore* dan menampilkannya pada antarmuka histori pesanan. Proses dimulai dengan melakukan permintaan data kepada koleksi *Order* dengan menerapkan kondisi penyaringan yang membatasi hasil hanya pada dokumen yang memiliki nilai *FinishedOrder* bernilai *true*. Setelah data berhasil diperoleh, daftar *allOrders* dibersihkan terlebih dahulu untuk menghindari duplikasi, kemudian setiap dokumen hasil *query* dikonversi ke dalam objek *Order*. Pada tahap ini, fungsi juga menetapkan kembali nilai *OrderID* berdasarkan ID dokumen di *Firestore* agar informasi identitas pesanan tersimpan secara konsisten.

Selanjutnya, seluruh data pesanan yang telah dikumpulkan diurutkan secara manual menggunakan metode *sortByDescending*, dengan acuan waktu pada atribut *OrderDate*. Pengurutan ini memastikan bahwa pesanan yang paling baru ditampilkan pada urutan teratas, sehingga meningkatkan kemudahan akses bagi pengguna. Setelah proses pengurutan selesai, daftar tersebut diteruskan ke *adapter* melalui *submitList* untuk diperbarui pada tampilan antarmuka. Sistem kemudian melakukan pemeriksaan tambahan untuk menentukan apakah tidak ada pesanan yang ditemukan; jika daftar kosong, pengguna diberikan notifikasi melalui pesan *Toast*. Di sisi lain, apabila terjadi kegagalan dalam pengambilan data, fungsi menampilkan pesan kesalahan yang informatif sehingga pengguna mengetahui adanya kendala dalam proses pemuatan data.

Selain memuat dan menampilkan daftar pesanan yang telah selesai, laman histori pesanan juga dilengkapi dengan fungsi *filterOrders* yang berperan sebagai mekanisme pencarian data secara dinamis. Fungsi ini memungkinkan pengguna untuk melakukan penyaringan daftar pesanan berdasarkan nama perusahaan (*PTName*) yang terkait dengan masing-masing pesanan. Proses penyaringan dilakukan dengan mencocokkan kata kunci pencarian yang dimasukkan pengguna terhadap atribut *PTName*, dengan menerapkan pencocokan tidak sensitif terhadap huruf kapital (*case-insensitive*) guna meningkatkan fleksibilitas dan akurasi

pencarian (Kode 3.6).

```
private fun filterOrders(query: String) {
        val filtered = allOrders.filter { order ->
            order.PTName?.contains(query, ignoreCase = true) ==
    true
        }
        adapter.submitList(filtered)

        if (filtered.isEmpty()) {
            Toast.makeText(this, "Tidak ditemukan: $query", Toast.
    LENGTH_SHORT).show()
        }
     }
```

Kode 3.6: Function filterOrders

Hasil penyaringan kemudian diserahkan kembali kepada *adapter* melalui *submitList*, sehingga tampilan daftar pesanan pada antarmuka diperbarui secara *real-time* sesuai dengan hasil pencarian. Apabila tidak ditemukan data yang sesuai dengan kata kunci pencarian, sistem memberikan umpan balik kepada pengguna dalam bentuk notifikasi *Toast* yang menyampaikan bahwa tidak ada hasil yang cocok. Dengan demikian, fungsi ini berperan penting dalam meningkatkan pengalaman pengguna melalui penyediaan fitur navigasi dan pencarian data yang lebih efisien, khususnya ketika jumlah pesanan yang tersimpan cukup besar.

Setelah pengguna menekan salah satu perusahaan yang muncul di histori pesanan selesai maka sistem akan menampilkan informasi lengkap mengenai pesanan yang dilakukan oleh perusahaan tersebut, potongan kode ini dapat dilihat pada Kode 3.7.

```
private fun loadOrderDetail() {
        val orderRef = firestore.collection("Order").document(
    orderId!!)

        // Load Order data
        orderRef.get()
            .addOnSuccessListener { orderDoc ->
                if (!orderDoc.exists()) {
                    Toast.makeText(this, "Order tidak ditemukan",
    Toast.LENGTH_SHORT).show()
                    finish()
                    return@addOnSuccessListener
                }

                // Display Order info
                val ptName = orderDoc.getString("PTName") ?: "-"
                val orderDate = orderDoc.getTimestamp("OrderDate")
                val dueDate = orderDoc.getTimestamp("DueDate")
                val grandTotal = orderDoc.getDouble("GrandTotal")
    ?: 0.0

                tvPTName.text = ptName
                tvTanggalPO.text = formatDate(orderDate)
```

```
                tvTanggalTempo.text = formatDate(dueDate)
                tvDibayar.text = formatCurrency(grandTotal)

                // Progress bar 100% karena sudah selesai
                progressBar.progress = 100
                tvProgressPercent.text = "100%"

                // Load SEMUA OrderBarang (tidak dibatasi hanya 1)
                loadAllOrderBarang(orderRef)

                // Load PostOrder
                loadPostOrder(orderRef)

                // Load semua progress dari semua barang
                loadAllOrderProgress(orderRef)
            }
            .addOnFailureListener { e ->
                Toast.makeText(this, "Error: ${e.message}", Toast.
    LENGTH_SHORT).show()
            }
     }
```

Kode 3.7: Function loadOrderDetail

Fungsi *loadOrderDetail()* bertanggung jawab untuk memuat keseluruhan informasi utama terkait suatu pesanan yang dipilih dari laman histori. Proses dimulai dengan mengambil dokumen pesanan berdasarkan *orderId* dari koleksi *Order* pada *Firestore*. Setelah data berhasil diperoleh, sistem melakukan validasi untuk memastikan bahwa dokumen tersebut benar-benar ada. Apabila valid, fungsi ini kemudian mengekstraksi informasi penting seperti nama perusahaan, tanggal pemesanan, tanggal jatuh tempo, serta total biaya pesanan. Seluruh informasi tersebut ditampilkan pada elemen antarmuka untuk memberikan gambaran umum mengenai pesanan tersebut. Selain itu, karena status pesanan sudah selesai, indikator progres secara otomatis diatur ke 100%. Setelah memuat data utama, fungsi ini memanggil tiga fungsi lanjutan, yaitu *loadAllOrderBarang()* untuk mengambil daftar barang yang dipesan, *loadPostOrder()* untuk memuat informasi pengiriman, dan *loadAllOrderProgress()* untuk menampilkan riwayat progres setiap barang selama proses produksi. Kemudian sistem akan memuat seluruh barangnya pada pesanan tersebut, kode ini dapat dilihat pada Kode 3.8.

```
private fun loadAllOrderBarang(orderRef: com.google.firebase.
    firestore.DocumentReference) {
        orderRef.collection("OrderBarang")
            .get()
            .addOnSuccessListener { barangSnapshot ->
                val barangList = mutableListOf<OrderBarang>()

                for (barangDoc in barangSnapshot.documents) {
                    val barang = OrderBarang(
                        namaBarang = barangDoc.getString("
    OrderName"),
```

```
                        estimateOrder = barangDoc.getLong("
    EstimateOrder")?.toInt() ?: 0,
                        hargaPerPcs = barangDoc.getDouble("
    PricePerPcs"),
                        totalHarga = barangDoc.getDouble("
    TotalPrice")
                    )
                    barangList.add(barang)
                }

                // Tampilkan semua barang di RecyclerView
                barangAdapter.submitList(barangList)

                if (barangList.isEmpty()) {
                    Toast.makeText(this, "Tidak ada data barang",
    Toast.LENGTH_SHORT).show()
                }
            }
            .addOnFailureListener { e ->
                Toast.makeText(this, "Error loading barang: ${e.
    message}", Toast.LENGTH_SHORT).show()
            }
     }
```

Kode 3.8: Function loadAllOrderBarang

Fungsi *loadAllOrderBarang()* digunakan untuk memuat seluruh barang yang terdapat dalam suatu pesanan. Fungsi ini mengakses subkoleksi *OrderBarang* pada dokumen pesanan dan mengambil seluruh dokumen di dalamnya tanpa batasan jumlah. Setiap dokumen barang kemudian dikonversi menjadi objek *OrderBarang*, yang mencakup informasi seperti nama barang, estimasi total pesanan, harga per satuan, serta total harga masing-masing barang. Seluruh data yang berhasil dikumpulkan kemudian dikirim ke *adapter RecyclerView* untuk ditampilkan pada antarmuka pengguna. Dengan demikian, fungsi ini menyediakan gambaran komprehensif mengenai komponen penyusun pesanan. Apabila tidak ditemukan barang apa pun, sistem memberikan notifikasi kepada pengguna bahwa data tidak tersedia. Setelah itu sistem akan memuat informasi untuk proses pengiriman pesanan tersebut sesudah pesanan selesai produksi, kode ini dapat dilihat pada Kode 3.9.

```
private fun loadPostOrder(orderRef: com.google.firebase.firestore.
    DocumentReference) {
        orderRef.collection("PostOrder")
            .limit(1)
            .get()
            .addOnSuccessListener { postSnapshot ->
                if (!postSnapshot.isEmpty) {
                    val postDoc = postSnapshot.documents[0]
                    val courierName = postDoc.getString("
    CourierName") ?: "-"
                    val deliveryDate = postDoc.getTimestamp("
    DeliveryDate")
```

```
                    val totalDefect = postDoc.getLong("TotalDefect
    ")?.toInt() ?: 0

                    tvPengirim.text = courierName
                    tvTanggalPengiriman.text = formatDate(
    deliveryDate)
                    tvBarangGagal.text = totalDefect.toString()
                } else {
                    tvPengirim.text = "-"
                    tvTanggalPengiriman.text = "-"
                    tvBarangGagal.text = "0"
                }
            }
            .addOnFailureListener { e ->
                Toast.makeText(this, "Error loading post order: ${
    e.message}", Toast.LENGTH_SHORT).show()
            }
     }
```

Kode 3.9: Function loadPostOrder

Fungsi *loadPostOrder()* berperan untuk memuat informasi terkait proses pengiriman setelah pesanan selesai diproduksi. Data tersebut diambil dari subkoleksi *PostOrder* pada dokumen pesanan, namun hanya satu dokumen pertama yang diambil karena setiap pesanan diasumsikan hanya memiliki satu catatan pengiriman. Informasi yang diekstraksi meliputi nama kurir, tanggal pengiriman, serta jumlah barang cacat (*defect*) yang dihasilkan selama proses produksi. Jika data *PostOrder* tersedia, informasi ini ditampilkan pada antarmuka pengguna. Namun, apabila dokumen tidak ditemukan, maka sistem menampilkan nilai *default* berupa tanda strip sebagai penanda bahwa informasi belum tersedia. Fungsi ini memastikan bahwa pengguna dapat mengetahui detail pengiriman secara jelas setelah pesanan diselesaikan. setelah itu sistem akan memuat seluruh riwayat progress pengerjaan barang yang ada pada setiap pesanan, potongan kode ini dapat dilihat pada Kode 3.10.

```
private fun loadAllOrderProgress(orderRef: com.google.firebase.
    firestore.DocumentReference) {
        orderRef.collection("OrderBarang")
            .get()
            .addOnSuccessListener { barangSnapshot ->
                val allProgressList = mutableListOf<Pair<String,
    OrderProgress>>()
                var processedCount = 0
                val totalBarang = barangSnapshot.documents.size

                if (totalBarang == 0) {
                    progressAdapter.submitList(emptyList())
                    return@addOnSuccessListener
                }

                // Loop semua barang untuk mengambil progress
    masing-masing
```

```
                for (barangDoc in barangSnapshot.documents) {
                    // Ambil nama barang dari field OrderName
                    val namaBarang = barangDoc.getString("OrderName") ?: "-"

                    barangDoc.reference.collection("OrderProgress")
                        .orderBy("Date", Query.Direction.ASCENDING)
                        .get()
                        .addOnSuccessListener { progressSnapshot ->
                            for (doc in progressSnapshot.documents) {
                                val progress = OrderProgress(
                                    CurrentProgress = doc.getLong("CurrentProgress")?.toInt() ?: 0,
                                    Date = doc.getTimestamp("Date")
                                )
                                // Simpan sebagai Pair (namaBarang, progress)
                                allProgressList.add(Pair(namaBarang, progress))
                            }

                            processedCount++

                            // Jika semua barang sudah diproses, tampilkan progress
                            if (processedCount == totalBarang) {
                                // Sort berdasarkan nama barang dulu, lalu tanggal
                                allProgressList.sortWith(compareBy({ it.first }, { it.second.Date?.toDate()?.time }))
                                progressAdapter.submitList(allProgressList)
                            }
                        }
                        .addOnFailureListener { e ->
                            processedCount++
                            if (processedCount == totalBarang) {
                                allProgressList.sortWith(compareBy({ it.first }, { it.second.Date?.toDate()?.time }))
                                progressAdapter.submitList(allProgressList)
                            }
                        }
                }
            }
            .addOnFailureListener { e ->
                Toast.makeText(this, "Error loading progress: ${e.message}", Toast.LENGTH_SHORT).show()
            }
    }
```

Kode 3.10: Function loadAllOrderProgress

Fungsi *loadAllOrderProgress()* digunakan untuk memuat seluruh riwayat

progres produksi dari setiap barang yang termasuk dalam pesanan. Proses dimulai dengan mengambil semua dokumen dari subkoleksi *OrderBarang*. Untuk setiap barang, fungsi ini kemudian mengakses subkoleksi *OrderProgress*, yang berisi perkembangan produksi yang dicatat berdasarkan tanggal. Setiap catatan progres dikonversi menjadi objek *OrderProgress* dan dipasangkan dengan nama barang yang bersangkutan. Semua data progres kemudian dikumpulkan dalam sebuah daftar berisi pasangan nama barang dan progresnya. Setelah seluruh barang selesai diproses, daftar progres tersebut diurutkan berdasarkan nama barang serta urutan waktu, sehingga menghasilkan tampilan kronologis yang runtut dan mudah dipahami. Daftar yang sudah terurut ini kemudian dikirim ke *adapter* untuk ditampilkan melalui *RecyclerView*. Dengan demikian, fungsi ini menyediakan dokumentasi lengkap mengenai perkembangan setiap barang selama proses produksi, mulai dari awal hingga pesanan diselesaikan.

### 3.6.3 Potongan Kode Laman Akumulasi Omset

Kode berikut merupakan implementasi dari fitur akumulasi omset yang bertugas mengelompokkan total pendapatan berdasarkan bulan, menampilkan total keseluruhan omset (Kode 3.11), serta menyediakan mekanisme penghapusan data omset pada bulan tertentu beserta seluruh entitas pesanan yang terkait. Setiap fungsi memiliki peran spesifik dalam proses pengambilan data, pengelompokan, agregasi, dan penghapusan, sehingga keseluruhan alur dapat berjalan secara konsisten dan terkontrol.

```
1  private fun loadRevenueData() {
2          firestore.collection("Order")
3              .whereEqualTo("FinishedOrder", true)
4              .get()
5              .addOnSuccessListener { result ->
6                  val monthMap = mutableMapOf<String, MutableList<
       OrderRevenue>>()
7                  var totalRevenue = 0.0
8
9                  for (document in result) {
10                     val grandTotal = document.getDouble("
       GrandTotal") ?: 0.0
11                     val orderDate = document.getTimestamp("
       OrderDate")
12                     val ptName = document.getString("PTName") ?: "
       -"
13                     val orderId = document.id
14
15                     if (orderDate != null) {
16                         // Format bulan: "April / 2025"
17                         val monthKey = formatMonth(orderDate)
18                         val dateString = formatDate(orderDate)
```

```
                        val orderRevenue = OrderRevenue(
                            ptName = ptName,
                            orderDate = dateString,
                            grandTotal = grandTotal,
                            orderId = orderId
                        )

                        if (monthMap.containsKey(monthKey)) {
                            monthMap[monthKey]?.add(orderRevenue)
                        } else {
                            monthMap[monthKey] = mutableListOf(
orderRevenue)
                        }

                        totalRevenue += grandTotal
                    }
                }

                // Convert map to list
                monthlyRevenueList.clear()
                for ((month, orders) in monthMap) {
                    val monthTotal = orders.sumOf { it.grandTotal
}
                    monthlyRevenueList.add(
                        MonthlyRevenue(
                            month = month,
                            totalRevenue = monthTotal,
                            orders = orders
                        )
                    )
                }

                // Sort by date (newest first)
                monthlyRevenueList.sortByDescending { parseMonth(
it.month) }

                adapter.submitList(monthlyRevenueList)
                tvTotalRevenue.text = formatCurrency(totalRevenue)

                if (monthlyRevenueList.isEmpty()) {
                    Toast.makeText(this, "Belum ada data omset",
Toast.LENGTH_SHORT).show()
                }
            }
            .addOnFailureListener { e ->
                Toast.makeText(this, "Gagal memuat data: ${e.
message}", Toast.LENGTH_LONG).show()
            }
    }
```

Kode 3.11: Function loadRevenueData

Fungsi *loadRevenueData()* bertanggung jawab untuk mengambil seluruh data pesanan yang telah berstatus selesai dari koleksi *Order*. Proses dimulai dengan melakukan *query Firestore* menggunakan kondisi *FinishedOrder = true*. Setiap dokumen yang berhasil diambil kemudiaan diproses menjadi struktur data yang

berorientasi pada bulan. Pengelompokan dilakukan dengan cara mengekstraksi bulan dan tahun dari atribut *OrderDate*, sehingga setiap pesanan ditempatkan pada kelompok bulan yang sesuai. Pada tahap ini, objek *OrderRevenue* disusun untuk menyimpan informasi perusahaan, tanggal pemesanan, total nilai transaksi, dan ID pesanan. Selanjutnya, fungsi menghitung akumulasi omset keseluruhan dengan menjumlahkan seluruh nilai *GrandTotal*. Setelah data dikelompokkan, fungsi membentuk daftar *MonthlyRevenue* yang memuat nama bulan, total omset pada bulan tersebut, dan daftar pesanan terkait. Daftar ini kemudian diurutkan dari bulan terbaru ke terlama sebelum ditampilkan ke antarmuka melalui *adapter*. Pada bagian akhir, fungsi menampilkan total omset keseluruhan pada sisi atas laman. Apabila tidak terdapat data, sistem memberikan notifikasi bahwa omset belum tersedia.

Setelah itu pengguna dapat menghapus seluruh data pemesanan yang ada pada bulan tertentu, potongan kode ini dapat dilihat pada Kode 3.12.

```
private fun deleteMonthRevenue(monthlyRevenue: MonthlyRevenue) {
        val orderIds = monthlyRevenue.orders.map { it.orderId }

        if (orderIds.isEmpty()) {
            Toast.makeText(this, "Tidak ada order untuk dihapus",
    Toast.LENGTH_SHORT).show()
            return
        }

        // Show loading
        val progressDialog = Dialog(this)
        progressDialog.setContentView(R.layout.dialog_loading)
        progressDialog.window?.setBackgroundDrawable(ColorDrawable
    (Color.TRANSPARENT))
        progressDialog.setCancelable(false)
        progressDialog.show()

        // Hapus semua order menggunakan batch
        val batch = firestore.batch()
        var deletedCount = 0
        var failedCount = 0

        for (orderId in orderIds) {
            val orderRef = firestore.collection("Order").document(
    orderId)

            // Delete subcollections first (OrderBarang dan
    PostOrder)
            deleteOrderSubcollections(orderId) { success ->
                if (success) {
                    // Kemudian hapus document Order
                    batch.delete(orderRef)
                    deletedCount++
                } else {
                    failedCount++
                }

                // Jika semua sudah diproses
```

```
                if (deletedCount + failedCount == orderIds.size) {
                    // Commit batch delete
                    batch.commit()
                        .addOnSuccessListener {
                            progressDialog.dismiss()
                            Toast.makeText(
                                this,
                                "Berhasil menghapus $deletedCount order dari ${monthlyRevenue.month}",
                                Toast.LENGTH_LONG
                            ).show()

                            // Reload data
                            loadRevenueData()
                        }
                        .addOnFailureListener { e ->
                            progressDialog.dismiss()
                            Toast.makeText(
                                this,
                                "Gagal menghapus: ${e.message}",
                                Toast.LENGTH_LONG
                            ).show()
                        }
                }
            }
        }
    }
```

Kode 3.12: Function deleteMonthRevenue

Fungsi *deleteMonthRevenue()* mengelola proses penghapusan seluruh data pesanan pada bulan tertentu. Langkah awal dilakukan dengan mengekstraksi seluruh *orderId* yang terdapat dalam objek *MonthlyRevenue*. Jika tidak ada ID yang ditemukan, sistem menampilkan pesan bahwa tidak ada data untuk dihapus. Selanjutnya, fungsi menampilkan dialog *loading* sebagai indikator bahwa proses penghapusan sedang berlangsung. *Firestore batch operation* kemudian diinisialisasi untuk menghapus dokumen utama dari koleksi *Order*. Namun, sebelum dokumen order dapat dihapus, seluruh subkoleksi yang berada di bawahnya, yaitu *OrderBarang* dan *PostOrder*, harus dihapus terlebih dahulu. Proses ini dilakukan secara asinkron melalui pemanggilan fungsi *deleteOrderSubcollections()* yang dapat dilihat pada Kode 3.13. Setelah seluruh subkoleksi berhasil dihapus, *batch* akan menghapus dokumen *Order* untuk setiap ID. Jika seluruh operasi berhasil, sistem menampilkan pesan keberhasilan dan memuat ulang data omset untuk memperbarui tampilan. Sebaliknya, jika terjadi kesalahan pada salah satu proses, sistem menampilkan pesan kegagalan.

```
private fun deleteOrderSubcollections(orderId: String, onComplete: (Boolean) -> Unit) {
        val orderRef = firestore.collection("Order").document(orderId)
        var tasksCompleted = 0
```

```
        val totalTasks = 2 // OrderBarang dan PostOrder

        // Delete OrderBarang subcollection
        orderRef.collection("OrderBarang")
            .get()
            .addOnSuccessListener { barangSnapshot ->
                val barangBatch = firestore.batch()

                for (barangDoc in barangSnapshot.documents) {
                    // Delete OrderProgress for each OrderBarang
                    barangDoc.reference.collection("OrderProgress")
                        .get()
                        .addOnSuccessListener { progressSnapshot ->
                            for (progressDoc in progressSnapshot.documents) {
                                barangBatch.delete(progressDoc.reference)
                            }
                        }

                    // Delete OrderBarang document
                    barangBatch.delete(barangDoc.reference)
                }

                barangBatch.commit()
                    .addOnSuccessListener {
                        tasksCompleted++
                        if (tasksCompleted == totalTasks) {
                            onComplete(true)
                        }
                    }
                    .addOnFailureListener {
                        onComplete(false)
                    }
            }
            .addOnFailureListener {
                tasksCompleted++
                if (tasksCompleted == totalTasks) {
                    onComplete(false)
                }
            }

        // Delete PostOrder subcollection
        orderRef.collection("PostOrder")
            .get()
            .addOnSuccessListener { postSnapshot ->
                val postBatch = firestore.batch()

                for (postDoc in postSnapshot.documents) {
                    postBatch.delete(postDoc.reference)
                }

                postBatch.commit()
                    .addOnSuccessListener {
                        tasksCompleted++
                        if (tasksCompleted == totalTasks) {
                            onComplete(true)
```

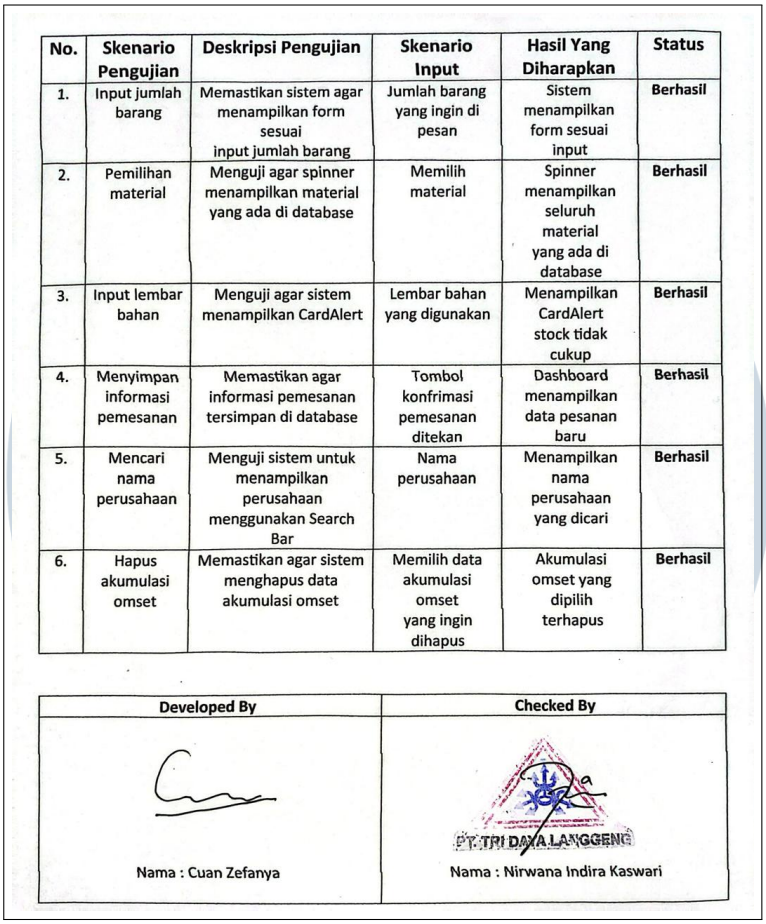
```
                        }
                    }
                    .addOnFailureListener {
                        onComplete(false)
                    }
            }
            .addOnFailureListener {
                tasksCompleted++
                if (tasksCompleted == totalTasks) {
                    onComplete(false)
                }
            }
    }
```

Kode 3.13: Function deleteOrderSubCollections

Fungsi *deleteOrderSubcollections()* bertugas menghapus seluruh subkoleksi yang berada di dalam suatu dokumen *Order*, yaitu *OrderBarang* dan *PostOrder*. Proses dimulai dengan mengambil seluruh dokumen dari subkoleksi *OrderBarang*. Setiap dokumen barang diperiksa untuk mengetahui apakah memiliki subkoleksi *OrderProgress*, dan jika ada, seluruh dokumen *progress* tersebut turut dihapus dengan *batch*. Setelah itu, dokumen *OrderBarang* juga dihapus dalam *batch* yang sama. Setelah seluruh dokumen pada subkoleksi ini berhasil dihapus dan *batch* diselesaikan, fungsi mencatat bahwa satu tugas penghapusan telah selesai. Langkah berikutnya dilakukan pada subkoleksi *PostOrder*, di mana seluruh dokumen di dalamnya juga dihapus menggunakan *batch* terpisah. Ketika kedua subkoleksi telah diproses dan *batch commit* selesai, fungsi menjalankan *callback onComplete(true)* sebagai indikator bahwa penghapusan subkoleksi berhasil. Apabila salah satu operasi mengalami kegagalan, *callback* dipanggil dengan nilai *false*. Secara keseluruhan, fungsi ini memastikan bahwa tidak ada data anak yang tertinggal sebelum dokumen induk dihapus, sehingga struktur penyimpanan tetap bersih dan konsisten.

## 3.7 Pengujian Sistem

Tabel pada Gambar 3.34 digunakan untuk merangkum hasil pengujian black box untuk fitur login yang dikerjakan.



| No. | Skenario Pengujian | Deskripsi Pengujian | Skenario Input | Hasil Yang Diharapkan | Status |
|---|---|---|---|---|---|
| 1. | Input jumlah barang | Memastikan sistem agar menampilkan form sesuai input jumlah barang | Jumlah barang yang ingin di pesan | Sistem menampilkan form sesuai input | Berhasil |
| 2. | Pemilihan material | Menguji agar spinner menampilkan material yang ada di database | Memilih material | Spinner menampilkan seluruh material yang ada di database | Berhasil |
| 3. | Input lembar bahan | Menguji agar sistem menampilkan CardAlert | Lembar bahan yang digunakan | Menampilkan CardAlert stock tidak cukup | Berhasil |
| 4. | Menyimpan informasi pemesanan | Memastikan agar informasi pemesanan tersimpan di database | Tombol konfrimasi pemesanan ditekan | Dashboard menampilkan data pesanan baru | Berhasil |
| 5. | Mencari nama perusahaan | Menguji sistem untuk menampilkan perusahaan menggunakan Search Bar | Nama perusahaan | Menampilkan nama perusahaan yang dicari | Berhasil |
| 6. | Hapus akumulasi omset | Memastikan agar sistem menghapus data akumulasi omset | Memilih data akumulasi omset yang ingin dihapus | Akumulasi omset yang dipilih terhapus | Berhasil |

| Developed By | Checked By |
|---|---|
| Nama : Cuan Zefanya | PT. TRI DAYA LANGGENG<br>Nama : Nirwana Indira Kaswari |

Gambar 3.34. Hasil testing dengan tanda tangan direktur

## 3.8 Kendala dan Solusi

Pada tahap pengembangan sistem, terdapat sejumlah hambatan teknis yang muncul selama proses implementasi. Berbagai kendala tersebut perlu dianalisis secara sistematis agar dapat ditemukan pendekatan penyelesaian yang tepat. Oleh karena itu, bagian ini menguraikan beberapa kendala yang dihadapi beserta solusi yang diterapkan untuk memastikan proses pengembangan tetap berjalan efektif dan hasil akhir sesuai dengan kebutuhan sistem.

### 3.8.1 Kendala

Dalam proses pengembangan sistem, terdapat sejumlah hambatan teknis yang muncul dan berpengaruh terhadap efektivitas serta kelancaran pelaksanaan implementasi, yaitu:

1. Kesulitan dalam melakukan kustomisasi tampilan Spinner pada Android Kotlin. Proses penyesuaian gaya visual, dropdown, dan layout komponen ini membutuhkan pengaturan tambahan yang cukup kompleks, sehingga sering menimbulkan inkonsistensi tampilan antar perangkat.

2. Pengelolaan data pesanan baru memiliki alur proses yang cukup kompleks. Data harus diproses secara bertahap mulai dari input jumlah barang, input detail barang, pemilihan material, hingga perhitungan total harga. Setiap tahap saling bergantung, sehingga kesalahan kecil pada satu tahap dapat memengaruhi keseluruhan proses.

### 3.8.2 Solusi

Berdasarkan kendala yang muncul selama proses implementasi, diperoleh pula beberapa langkah penyelesaian yang dapat digunakan untuk mengatasi permasalahan tersebut, yaitu:

1. Menggunakan Spinner bawaan Android sebagai pendekatan yang lebih stabil. Dengan tidak memaksakan kustomisasi berlebihan, proses pengembangan menjadi lebih efektif dan komponen Spinner dapat bekerja dengan konsisten di berbagai perangkat tanpa memerlukan penyesuaian tambahan.

2. Menerapkan pemrosesan data secara bertahap pada setiap activity disertai validasi otomatis. Proses dimulai dari laman penentuan jumlah barang, dilanjutkan ke laman input detail barang, kemudian pemilihan material yang memanfaatkan TextWatcher untuk memantau stok dan menghitung total produksi serta harga secara real-time, dan diakhiri dengan laman ringkasan pesanan yang menghitung total PO, pajak, dan grand total. Pembagian proses ini memastikan bahwa setiap tahap melakukan pengecekan data secara mandiri, sehingga alur pemesanan menjadi lebih terstruktur dan meminimalkan risiko kesalahan.