BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Selama periode magang di Bestada (Bestari Aditama Persada), mahasiswa magang ditempatkan sebagai *Fullstack* Developer di divisi Developer, saya berada di bawah bimbingan Muhammad Yuedhitia yang menjabat sebagai *Tech Lead* sekaligus supervisor saya. Dalam peran ini, saya bertanggung jawab untuk berkolaborasi dengan tim UI/UX untuk mengembangkan *website* baik dari sisi *frontend* maupun *backend*. Selama pemagangan dilangsungkan penugasan dipertanggungjawabkan kepada supervisi yang selanjutnya supervisi akan bertanggung jawab kepada direktur Bestada yaitu Pak Safei.

Proses penugasan dan alokasi proyek di Bestada dimulai dengan penyampaian semua proyek yang akan dilakukan kepada direktur Bestada dan tim HR (Human Resource) kepada Tech Lead dari *team* developer. Kemudian Tech Lead menyampaikan proyek tersebut kepada *team developer* dan menugaskan satu atau lebih developer untuk mengerjakan proyek tersebut tergantung kepada kompleksitas dari proyek yang diberikan.

3.2 Pekerjaan yang Dilakukan

Selama melakukan kerja magang di Bestada sebagai Fullstack Developer, pekerjaan yang dilakukan adalah melanjutkan pengembangan website e-commerce untuk salah satu klien bestada, pengembangan website mencakup pengembangan dari sisi Backend maupun Frontend terutama pada fitur pembayaran dan kustomisasi pakaian.

3.3 Uraian Pelaksanaan Magang

Tabel 3.1. Uraian pekerjaan yang dilakukan per minggu

Minggu ke -	Pekerjaan yang dilakukan
1 - 2	Mempelajari alur kerja dari tiap *team* sekaligus mencoba berbagai pekerjaan dari berbagai *team* untuk menentukan penempatan posisi magang.
3	Setup awal untuk tiga *source code* dari projek e-commerce milik perusahaan XYZ.
4	5 Berdiskusi dengan developer sebelumnya terkait fitur-fitur yang perlu dikembangkan lebih lanjut.
5	Mengembangkan fitur keranjang untuk menyimpan pesanan klien ke dalam sebuah keranjang belanja.
6 - 8	Mengembangkan fitur pemesanan dan pembayaran melalui *payment gateway* Midtrans.
9	Mengembangkan fitur *customize style* pakaian pada halaman admin
10	Mengembangkan fitur bulk upload pada halaman admin dengan detail: • *User* dapat melakukan *upload* folder zip yang berisi gambar-gambar *style* pakaian. • Folder zip akan di unzip di *frontend* selanjutnya isi folder zip akan di sortir berdasar nama dan di *upload* secara *bulk* ke *database* MongoDb.
11 -12	Mengembangkan fitur *customize style* pada halaman *user*.
13	Melakukan testing bersama supervisi untuk mencari *bug* dan *error*.
14	Melakukan testing dan revisi berdasarkan arahan dari klien Bestada.

Berdasarkan tabel 3.1 pekerjaan-pekerjaan yang telah disebutkan dapat dikelompokkan sebagai berikut:

3.3.1 Pembelajaran

Selama masa pembelajaran yang dimulai dari minggu pertama hingga minggu kelima dengan detail sebagai berikut:

A. Penempatan Posisi

Pada minggu pertama hingga minggu kedua mahasiswa magang ditempatkan di berbagai *team* mulai dari UI/UX hingga developer hal ini dilakukan untuk mengevaluasi kemampuan mahasiswa magang sekaligus untuk mempelajari *jobdesk* dari masing-masing *team*.

B. Alur Kerja di dalam Bestada

Setelah mempelajari *jobdesk* dari masing-masing tim, mahasiswa magang dikenalkan dengan alur kerja di dalam Bestada yang mengaplikasikan metodologi Kanban dalam pengembangan *project*. Kanban diterapkan di Bestada untuk memastikan fleksibilitas dalam pengelolaan proyek serta memungkinkan klien untuk mengajukan revisi atau perubahan secara langsung. Dengan menggunakan metode Kanban, seluruh tugas diatur secara visual melalui papan Kanban yang terdiri dari kolom-kolom seperti *To Do*, *In Progress*, dan *Done* [2][3].
Metode Kanban terdiri dari beberapa tahap utama yang dijalankan secara iteratif, yaitu:

- **Persiapan (Preparation)**: Pada tahap ini, seluruh tugas yang perlu diselesaikan dalam proyek diidentifikasi dan dibagi ke dalam item-item yang lebih kecil. Setiap tugas tersebut kemudian dimasukkan ke dalam kolom *To Do* pada papan Kanban. Persiapan ini melibatkan pemahaman mengenai prioritas tugas dan alokasi sumber daya yang dibutuhkan.
- **Pengerjaan (Execution)**: Setelah tugas dipersiapkan, tim mulai bekerja untuk menyelesaikan tugas yang ada di kolom *To Do*. Ketika seseorang mulai mengerjakan tugas tersebut, tugas dipindahkan ke kolom *In Progress*. Pada tahap ini, tim fokus pada penyelesaian tugas dengan mengikuti aturan WIP (Work In Progress) yang membatasi jumlah pekerjaan yang dapat berlangsung pada satu waktu untuk meningkatkan efisiensi dan menghindari *multitasking* yang berlebihan.

- **Penyelesaian (Completion)**: Setelah tugas selesai, pekerjaan dipindahkan ke kolom *Done*. Pada tahap ini, pekerjaan dianggap selesai dan siap untuk dievaluasi atau diserahkan kepada klien atau *stakeholder* lainnya. Kolom *Done* memberikan gambaran visual mengenai progres yang telah dicapai oleh tim.

- **Evaluasi (Review)**: Evaluasi dilakukan secara berkala untuk menilai kualitas pekerjaan yang telah diselesaikan dan memastikan apakah tugas tersebut memenuhi kriteria yang ditetapkan. Jika perlu, tim dapat melakukan revisi dan memperbaiki pekerjaan berdasarkan masukan yang diterima. Evaluasi ini juga berfungsi untuk menentukan apakah alur kerja perlu disesuaikan untuk peningkatan di masa mendatang.

Dengan alur kerja yang terstruktur dalam Metodologi Kanban, tim dapat memantau status setiap tugas secara transparan dan membuat penyesuaian secara cepat berdasarkan kebutuhan proyek dan masukan klien.

Proses ini mempermudah tim untuk memantau perkembangan proyek secara real-time dan memastikan prioritas tugas dapat disesuaikan berdasarkan permintaan klien. Pembatasan pekerjaan yang sedang berjalan (Work In Progress) menjaga fokus tim pada penyelesaian tugas dengan kualitas optimal sebelum melanjutkan ke tugas berikutnya.

Penerapan Metodologi Kanban memungkinkan perubahan prioritas dilakukan tanpa mengganggu alur kerja, sehingga revisi dari klien dapat segera diintegrasikan ke dalam proyek, meningkatkan efisiensi dan kepuasan klien.

C. Tools yang Digunakan

Berikut adalah beberapa *tools* yang digunakan dalam proyek Bestada:

1. **Figma**: Figma adalah platform desain grafis berbasis web yang digunakan untuk membuat antarmuka pengguna (UI) dan pengalaman pengguna (UX). Salah satu fitur unggulannya adalah kemampuan kolaborasi *real-time*, di mana beberapa anggota tim dapat mengedit dan memberi masukan pada desain yang sama secara bersamaan. Selain itu, Figma juga mendukung pembuatan prototipe interaktif dan desain responsif, serta menyimpan riwayat revisi, yang memudahkan pengelolaan perubahan dalam tim besar. Sebagai alat berbasis cloud, Figma sangat fleksibel dan mudah diakses dari berbagai

perangkat, menjadikannya sangat cocok untuk tim yang bekerja secara *remote*[4].

2. **Postman**. Postman adalah alat yang digunakan untuk menguji dan mengelola API (Application Programming Interface). Dengan Postman, pengembang dapat mengirim permintaan HTTP (GET, POST, PUT, DELETE, dll) ke server, menerima respons, dan menganalisis hasilnya dalam bentuk yang lebih mudah dibaca. Postman sangat berguna untuk pengujian API selama tahap pengembangan aplikasi. Selain itu, Postman menyediakan fitur untuk mengorganisir koleksi permintaan API, menyimpan variabel, serta otomatisasi pengujian API dengan skrip. Postman juga memungkinkan kolaborasi tim, di mana pengguna dapat berbagi koleksi dan dokumentasi API dalam tim mereka. Dengan antarmuka pengguna yang ramah, Postman menjadi alat yang sangat populer di kalangan pengembang dan tester API[5].

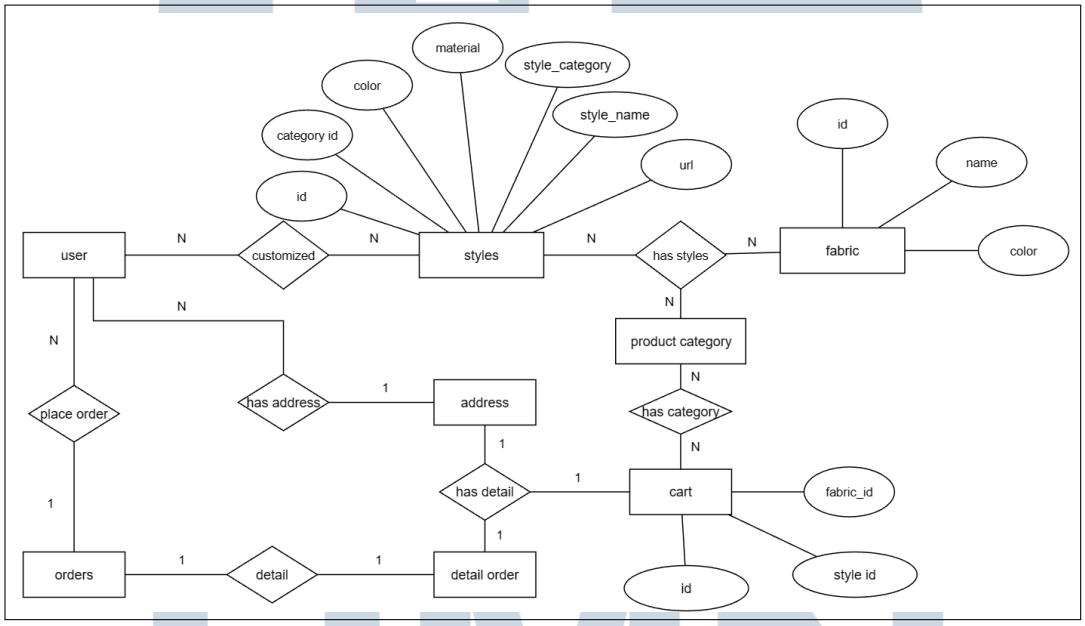
3. **PuTTY**. PuTTY adalah aplikasi terminal dan klien SSH (Secure Shell) yang digunakan untuk mengakses dan mengelola server jarak jauh, terutama di lingkungan sistem operasi berbasis Unix atau Linux. Dengan PuTTY, pengguna dapat membuat koneksi yang aman ke server untuk menjalankan perintah, mentransfer file, dan mengelola konfigurasi server melalui antarmuka baris perintah (CLI). PuTTY mendukung berbagai protokol seperti SSH, Telnet, Rlogin, dan Raw Socket, yang memudahkan administrasi dan pengelolaan server jarak jauh. Selain itu, PuTTY juga memungkinkan pengguna untuk menyimpan sesi koneksi untuk akses lebih cepat di masa depan. Keamanannya yang kuat, terutama dalam hal enkripsi data, menjadikannya pilihan utama bagi banyak profesional TI dan administrator sistem untuk mengelola server secara aman dan efisien[6].

3.3.2 Pengembangan Website

Website e-commerce perusahaan XYZ yang dikerjakan oleh mahasiswa magang dibagai menjadi tiga yaitu Web User yang bisa diakses oleh user untuk membeli pakaian, Web Admin yang hanya bisa diakses oleh administrator dari pihak perusahaan XYZ, dan *Backend* Api yang menangani semua *request* API. Adapun pengembangan terhadap ketiga projek tersebut dapat dibagi dengan detail sebagai berikut:

A. Pengembangan Database

Dalam pengembangan web-app e-commerce untuk Perusahaan XYZ dimulai dengan memahami *database* yang telah ada kemudian melakukan pengembangan terhadap *database* yang sudah ada, sebelumnya *database* memiliki 13 *table* namun setelah dilakukan pembersihan data yang kurang berguna maka tersisa 8 *table* dengan ERD yang digambarkan seperti pada gambar 3.1. Gambar tersebut hanya berupa *relational* antar *table* sedangkan *feature* dari *table* tersebut *confidential* sehingga tidak ditampilkan.



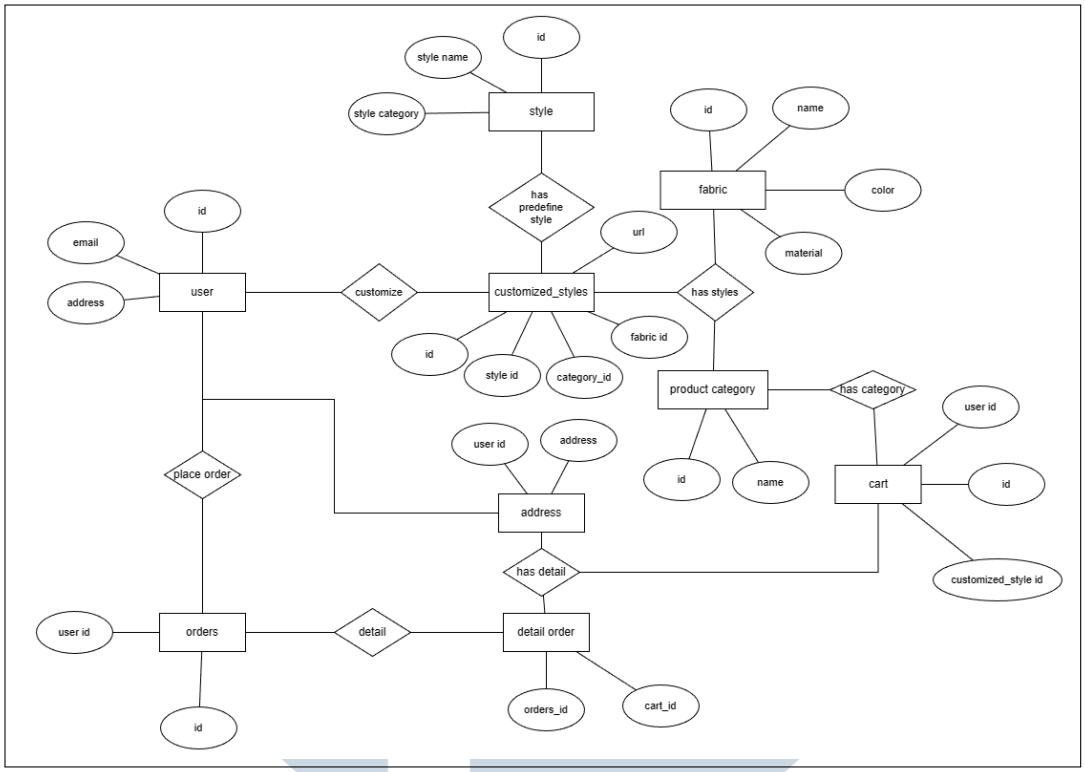
Gambar 3.1. Skema ERD web-app e-commerce

- **User dan Customized**: Terdapat hubungan banyak-ke-banyak (N:N) antara entitas *User* dan *Customized*. Artinya, seorang *user* dapat menyesuaikan banyak *styles*, dan setiap *style* dapat disesuaikan oleh banyak *user*.
- **Customized dan Styles**: Hubungan antara *Customized* dan *Styles* juga bersifat banyak-ke-banyak (N:N). Seorang *user* dapat menyesuaikan banyak *styles*, dan setiap *style* dapat disesuaikan oleh banyak *user*.
- **User dan Address**: Terdapat hubungan satu-ke-satu (1:1) antara entitas *User* dan *Address*, yang berarti setiap *user* hanya memiliki satu alamat (address).
- **User dan Orders**: Hubungan antara *User* dan *Orders* bersifat satu-ke-banyak (1:N). Seorang *user* dapat melakukan banyak *orders*, namun setiap *order* hanya dapat dilakukan oleh satu *user*.

- **Orders dan Detail**: Hubungan antara *Orders* dan *Detail* bersifat satu-ke-banyak (1:N). Setiap *order* dapat memiliki banyak *detail* (item yang dibeli), tetapi setiap *detail* hanya terkait dengan satu *order*.
- **Detail dan Detail Order**: Terdapat hubungan satu-ke-satu (1:1) antara entitas *Detail* dan *Detail Order*, yang berarti setiap *detail* dalam *order* terkait langsung dengan satu *detail order*.
- **Cart dan Product Category**: Hubungan antara *Cart* dan *Product Category* bersifat banyak-ke-satu (N:1). Setiap *cart* dapat berisi produk dari berbagai *product category*, namun satu *cart* hanya dapat berhubungan dengan satu kategori produk.
- **Cart dan Orders**: Hubungan antara *Cart* dan *Orders* bersifat satu-ke-banyak (1:N). Setiap *cart* dapat berisi banyak *orders*, namun setiap *order* hanya terkait dengan satu *cart*.

Skema ERD pada gambar 3.1 tersebut memiliki hubungan yang belum di normalisasi akibatnya data cenderung diulang terutama pada *table* styles yang seringkali memiliki data *fabric color*, *style category* dan *style name* yang sama, menyebabkan pemborosan ruang penyimpanan dan meningkatkan risiko inkonsistensi data. Redundansi yang tinggi meningkatkan kemungkinan terjadinya inkonsistensi, misalnya, jika data yang sama disalin ke beberapa tabel, pembaruan pada satu tempat tanpa pembaruan di tempat lain dapat menyebabkan data yang tidak sinkron. Selain itu, operasi pembaruan, penyisipan, dan penghapusan data dapat menimbulkan anomali, seperti menghapus satu entri yang dapat menghapus data penting lainnya yang tidak terkait, atau menambahkan data baru yang menyebabkan duplikasi yang tidak diinginkan.

Perbaikan dari permasalahan struktur *table* pada gambar 3.1 adalah dengan melakukan normalisasi hingga 3NF, normalisasi 3NF memiliki tiga langkah dimulai dari First Normal Form (1NF) memastikan bahwa setiap kolom memiliki nilai atomik dan tidak ada duplikasi baris. Kemudian Second Normal Form (2NF) mengatasi ketergantungan parsial, memastikan setiap atribut non-primer bergantung sepenuhnya pada kunci utama. Terakhir Third Normal Form (3NF) menghapus ketergantungan transitif antara atribut non-primer, sehingga setiap atribut non-primer hanya bergantung langsung pada kunci utama



Gambar 3.2. Normalized ERD

1NF (First Normal Form)

Pada tahap pertama normalisasi, 1NF, tujuan utama adalah memastikan bahwa setiap kolom dalam tabel hanya berisi nilai atomik, yaitu satu nilai per kolom dan tidak ada duplikasi, skema pada gambar 3.1 telah memenuhi standar 1NF karena tidak ada duplikasi nilai pada *table*.

2NF (Second Normal Form)

Pada tahap kedua, 2NF, tabel harus memenuhi 1NF dan memastikan bahwa tidak ada ketergantungan parsial. Artinya, setiap atribut non-primer harus sepenuhnya bergantung pada kunci utama, bukan hanya sebagian dari kunci utama.

Pada gambar 3.1 *database* memiliki masalah pada *table style* yaitu *style_name* dan *style_category* tidak bergantung pada kunci apapun selain id, hal ini juga menimbulkan redundansi karena data *style_name* dan *style_category* bisa berisi nilai yang sama berulang kali. Bentuk normalisasi dari *table* ini adalah memisahkan *table* menjadi dua yaitu *customize style* yang berisi id, *style name* dan *style category*, dan *table customize style* yang berisi data url image yang mana data url image ini

bergantung sepenuhnya kepada seluruh *foreign key* yang mereferensi kepada *table style*, *product category*, dan *fabric*.

Dalam gambar 3.1, masih ada ketergantungan parsial pada *table styles* yaitu terdapat material dan *color* yang hanya bergantung pada kunci *fabric id*, sehingga untuk mengatasi masalah ini, kustomisasi warna baju dimasukkan ke *table fabric* saja, sedangkan *color* di *style* dihapus.

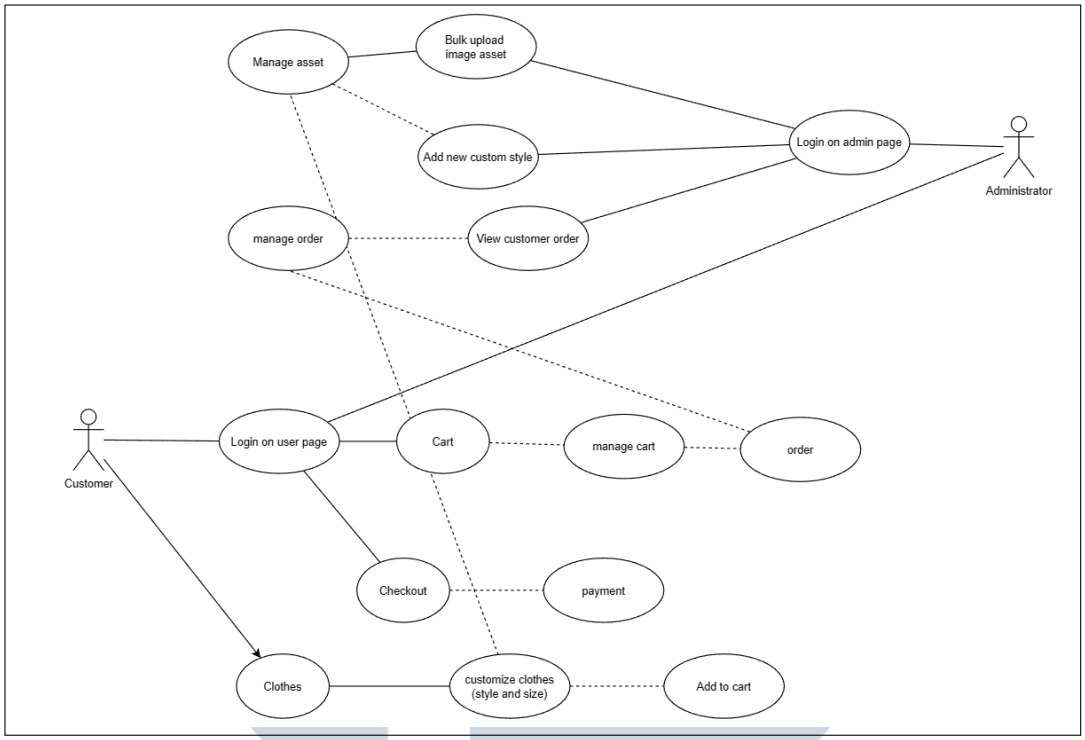
3NF (Third Normal Form)

Pada tahap ketiga, 3NF, tabel harus memenuhi 2NF dan memastikan bahwa tidak ada ketergantungan transitif, yaitu setiap atribut non-primer harus bergantung langsung pada kunci utama, dan tidak ada atribut non-primer yang bergantung pada atribut non-primer lainnya. Setelah *table* memenuhi kriteria 2NF ternyata tidak ditemukan ketergantungan transitif pada *table* sehingga bentuk dari skema 3.2 bisa dibilang telah memenuhi aturan 3NF.

Dengan mengikuti proses normalisasi ini, database menjadi lebih terstruktur, efisien, dan konsisten, mengurangi redundansi serta potensi inkonsistensi data.

3.3.3 User Relational

Pembuatan platform e-commerce ini mengacu pada *relational* antara dua tipe *user* yaitu *customer* dan administrator, *customer* hanya bisa mengakses halaman web *user* sedangkan administrator dapat mengakses halaman web *user* dan halaman web admin, pada halaman web user *customer* dapat berbelanja dan mengkustomisasi pakaian sesuai keinginan, pada halaman admin administrator bertugas untuk melakukan *upload* asset-asset produk agar *customer* dapat memilih pakaian dan mengkustomisasi pakaian. diagaram yang menggambarkan hubungan antara pengguna dari platform ini digambarkan seperti diagram pada gambar 3.3.



Gambar 3.3. User Relational Diagram

Pada diagram 3.3 terdapat dua aktor atau dua jenis pengguna yang mengakses platform e-commerce yaitu customer dan administrator, *customer* merupakan *user* biasa yang mengakses platform sedangkan administrator adalah pengguna dengan akses tertinggi dari pihak perusahaan yang bertugas untuk mengunggah aset-aset produk. Untuk lebih jelas berikut merupakan penjelasan dari gambar 3.3.

- **Customer:**
  - **Login on user page:** Customer harus *login* ke halaman pengguna untuk mengakses fitur-fitur lainnya.
  - **Clothes:** Setelah *login*, *Customer* dapat melihat berbagai pakaian yang tersedia.
  - **Customize clothes (style and size):** *Customer* dapat menyesuaikan pakaian sesuai dengan gaya dan ukuran yang diinginkan.
  - **Add to cart:** Setelah memilih dan menyesuaikan pakaian, *Customer* dapat menambahkannya ke keranjang belanja.
  - **Cart:** *Customer* dapat melihat dan mengelola keranjang belanja mereka.

  - **Manage cart:** *Customer* dapat mengubah jumlah atau menghapus produk dalam keranjang.
  - **Checkout:** Setelah selesai memilih produk, Customer dapat melanjutkan ke proses *checkout* untuk pembayaran.
  - **Payment:** Pada tahap akhir, *customer* melakukan pembayaran untuk memproses pesanan mereka.

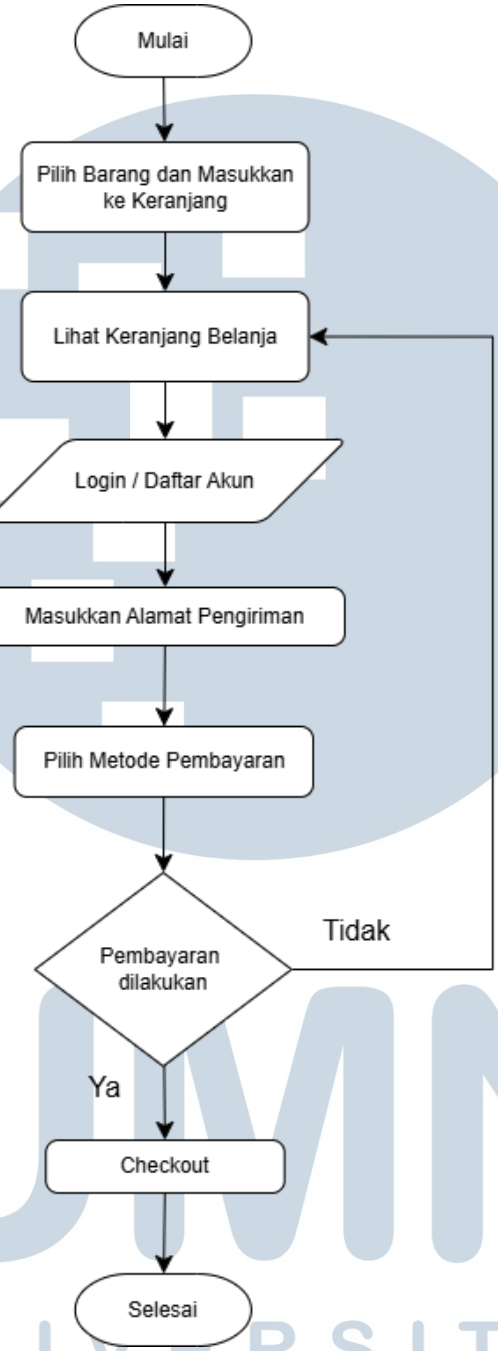
- **Administrator:**
  - **Login on admin page:** Administrator harus *login* ke halaman admin untuk mengakses fitur-fitur administrasi.
  - **Manage asset:** Administrator dapat mengelola aset-aset sistem, seperti gambar produk atau data lainnya.
  - **Bulk upload image asset:** Administrator dapat mengunggah gambar produk secara massal untuk efisiensi.
  - **Add new custom style:** Administrator dapat menambahkan gaya pakaian baru ke sistem.
  - **View customer order:** Administrator dapat melihat pesanan yang dibuat oleh *customer* dan mengelola statusnya.
  - **Manage order:** Administrator dapat mengelola pesanan yang ada, seperti mengonfirmasi atau membatalkan pesanan.

Hubungan Antara Use Case

- **Garis Lurus:** Menunjukkan hubungan *langsung dan wajib* antara aktor dan use case. Misalnya, *Customer* wajib *login* sebelum bisa mengakses fitur lainnya seperti melihat produk atau melakukan pembayaran.
- **Garis Putus-Putus:** Menunjukkan hubungan *opsional atau bergantung pada kondisi tertentu*. Misalnya, Administrator hanya perlu mengakses fitur seperti **View customer order** atau **Manage asset** jika diperlukan, tetapi tidak selalu digunakan dalam setiap sesi *login*.

A. Pengembangan fitur checkout

Fitur *checkout* memungkinkan pengguna untuk menyelesaikan proses pembelian dengan mudah dan cepat. Dalam fitur ini, pengguna dapat meninjau ulang barang yang akan dibeli, memilih metode pembayaran yang diinginkan, dan mengonfirmasi pesanan. Proses ini dirancang agar sederhana dan intuitif, memastikan pengalaman berbelanja yang nyaman bagi pengguna, proses dari fitur *checkout* digambarkan dalam *flowchart* 3.4.



Gambar 3.4. Flowchart Fitur Checkout

Untuk memahami *flowchart* ini lebih mendalam, Berikut adalah deskripsi lengkap dari setiap tahap yang ditampilkan dalam *flowchart* tersebut:

1. **Pilih Barang dan Masukkan ke Keranjang**: Langkah pertama adalah memilih barang yang ingin dibeli dan menambahkannya ke dalam keranjang belanja. Pengguna dapat menambahkan lebih dari satu produk sebelum melanjutkan ke tahap berikutnya.

2. **Lihat Keranjang Belanja**: Setelah barang-barang dimasukkan ke keranjang, pengguna akan memeriksa daftar barang yang dipilih. Pada tahap ini, mereka bisa memastikan jumlah, varian, dan harga produk yang akan dibeli. Jika ada kesalahan, pengguna bisa mengedit atau menghapus produk dari keranjang.

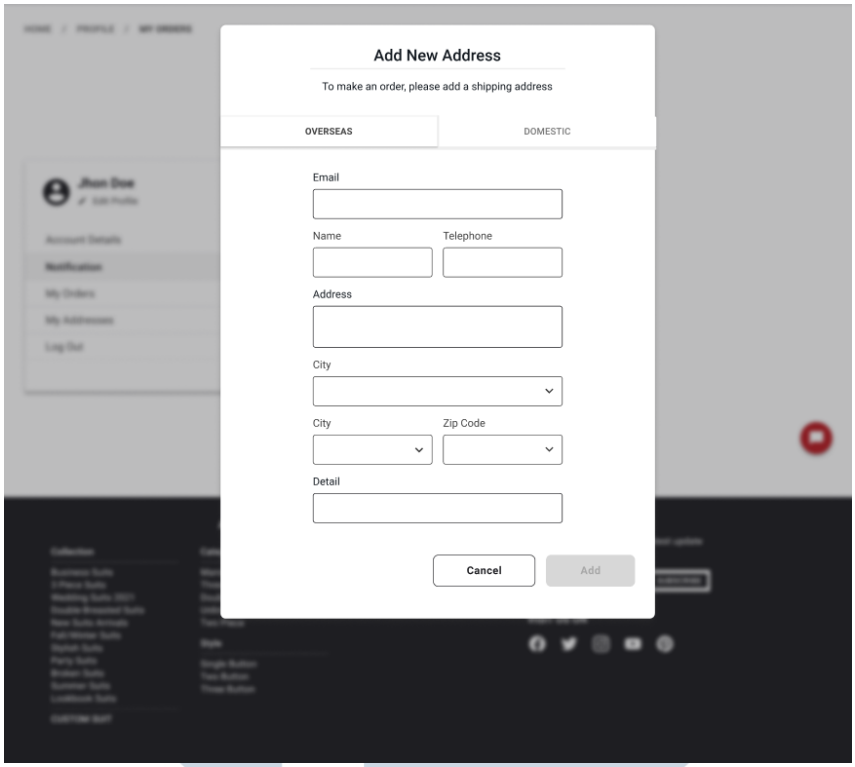
3. **Login / Daftar Akun**: Untuk melanjutkan proses *checkout*, pengguna perlu *login* ke akun mereka. Jika belum memiliki akun, mereka akan diminta untuk mendaftar terlebih dahulu. Ini penting untuk memudahkan pengiriman barang dan untuk melacak riwayat pembelian.

4. **Masukkan Alamat Pengiriman**: Setelah *login* atau mendaftar, pengguna akan diminta untuk mengisi alamat pengiriman, tempat di mana barang yang dibeli akan dikirimkan. Pengguna bisa memilih alamat yang sudah ada sebelumnya atau memasukkan alamat baru, tampilan input alamat dapat dilihat pada gambar 3.5.

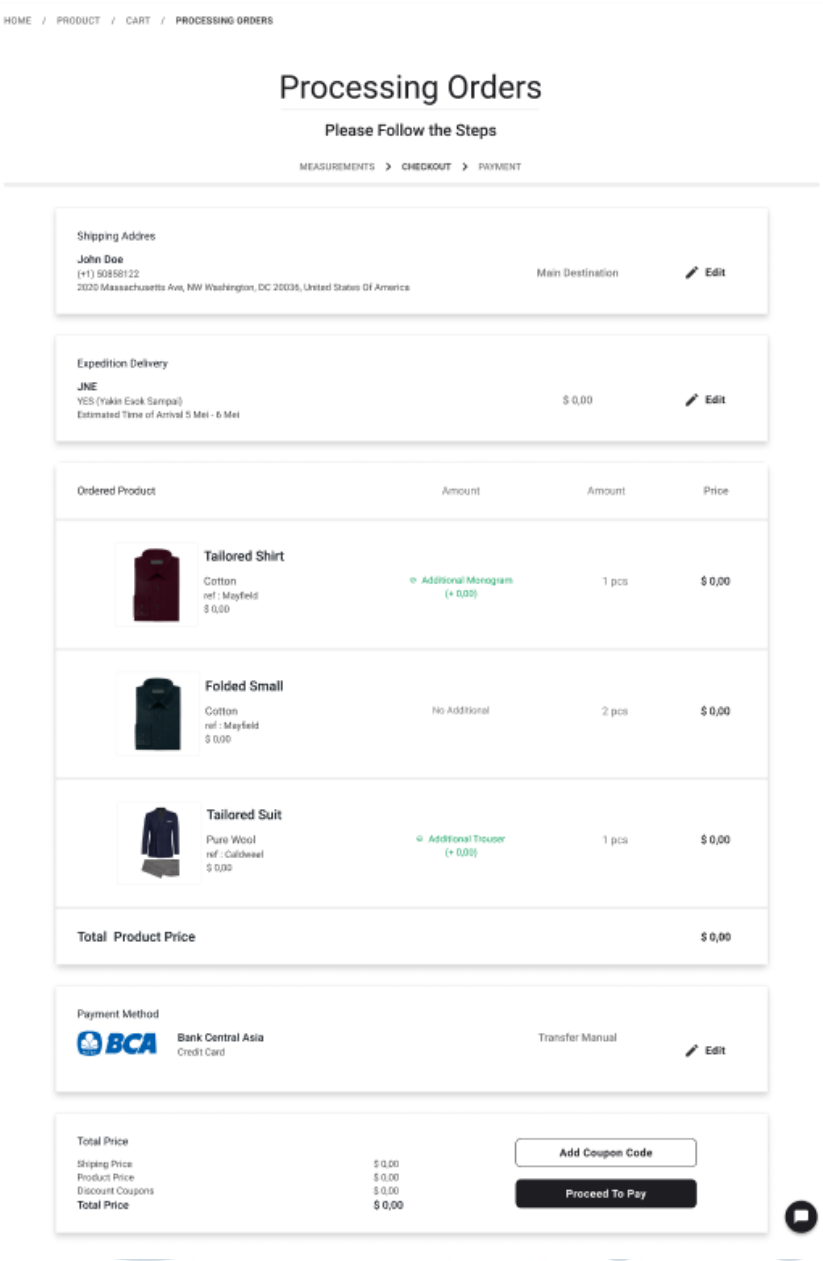
5. **Pilih Metode Pembayaran**: Setelah memasukkan alamat, pengguna akan memilih metode pembayaran yang tersedia, seperti kartu kredit, transfer bank, atau pembayaran menggunakan dompet digital. Pilihan ini biasanya bergantung pada opsi yang disediakan oleh situs *e-commerce*, tampilan halaman *checkout* dan pembayaran dapat dilihat pada gambar 3.6.

6. **Konfirmasi Pembayaran**: Pada tahap ini, sistem akan meminta pengguna untuk mengonfirmasi semua detail pembayaran dan pengiriman. Pengguna memiliki dua pilihan: jika mereka yakin dengan pilihan mereka, mereka dapat melanjutkan dengan memilih ”Ya” untuk memproses pembayaran; jika ada perubahan atau kesalahan, mereka bisa memilih ”Tidak” dan kembali ke langkah sebelumnya untuk memperbaiki detail.

7. **Checkout**: Setelah pembayaran berhasil diproses, pengguna akan menerima konfirmasi akhir bahwa *checkout* selesai dan barang akan segera diproses untuk pengiriman ke alamat yang telah diberikan.



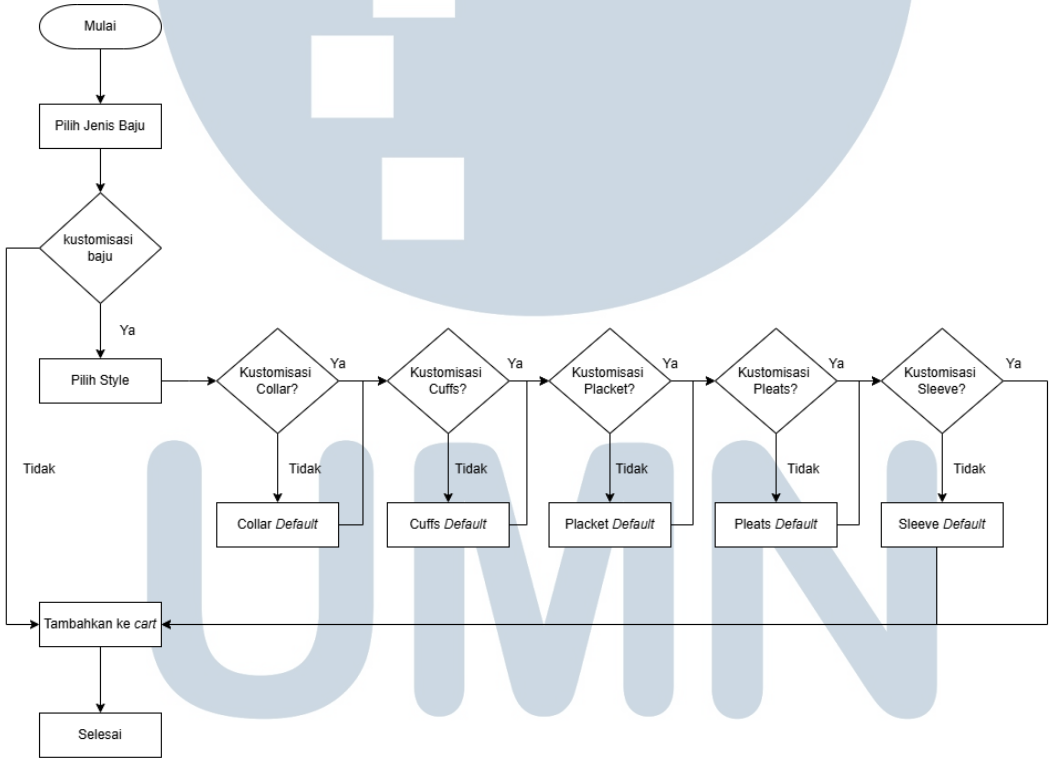
Gambar 3.5. Tampilan Input Alamat



Gambar 3.6. Halaman *Checkout*

B. Pengembangan Fitur Customize Style Baju

Fitur *customize style* pada *website e-commerc*e milik perusahaan xyz dimaksudkan memberikan kesan unik dibanding *website* e-commerce lain, fitur ini memungkinkan pengguna untuk melakukan kustomisasi desain baju sesuai preferensi mereka. Dalam fitur ini, pengguna dapat memilih berbagai elemen yang dapat disesuaikan, seperti jenis kerah *(collar)*, manset *(cuffs)*, bagian kancing *(pleats)*, lipatan *(pleats)*, hingga model lengan *(sleeve)*. Setiap pilihan kustomisasi memberikan opsi *default* maupun pilihan lain yang tersedia. Fitur ini dirancang untuk memberikan fleksibilitas dan kebebasan bagi pengguna dalam menciptakan baju dengan gaya unik yang sesuai dengan selera dan kebutuhan mereka.



Gambar 3.7. *Flowchart* Kustomisasi Pakaian

Untuk memahami *flowchart* ini lebih mendalam, Berikut adalah deskripsi lengkap dari setiap tahap yang ditampilkan dalam *flowchart* tersebut:

1. **Mulai**
   Proses dimulai dari langkah awal di mana pengguna masuk ke dalam sistem.

2. **Pilih Jenis Baju**
   Pengguna diminta untuk memilih jenis baju yang ingin dikustomisasi, seperti

kemeja, blus, atau jenis lainnya, tampilan halaman ini dapat dilihat pada gambar 3.8.

3. **Kustomisasi Baju**
   Setelah memilih jenis baju, pengguna diberi pilihan apakah ingin melakukan kustomisasi pada baju tersebut. Jika memilih *tidak* maka baju yang dipilih pengguna akan ditambahkan ke *cart* dengan *style default*. Jika memilih *ya*, maka proses kustomisasi akan dilanjutkan ke langkah berikutnya.

4. **Pilih Style**
   Pengguna diminta untuk memilih gaya atau model dasar baju sesuai dengan preferensi mereka, tampilan halaman ini dapat dilihat pada gambar 3.9.

5. **Kustomisasi Collar**
   Pengguna dapat memilih untuk mengkustomisasi kerah (*collar*). Jika tidak, sistem akan menggunakan pengaturan *default collar*.

6. **Kustomisasi Cuffs**
   Pengguna dapat memilih untuk mengkustomisasi manset (*cuffs*). Jika tidak, sistem akan menggunakan pengaturan *default cuffs*.

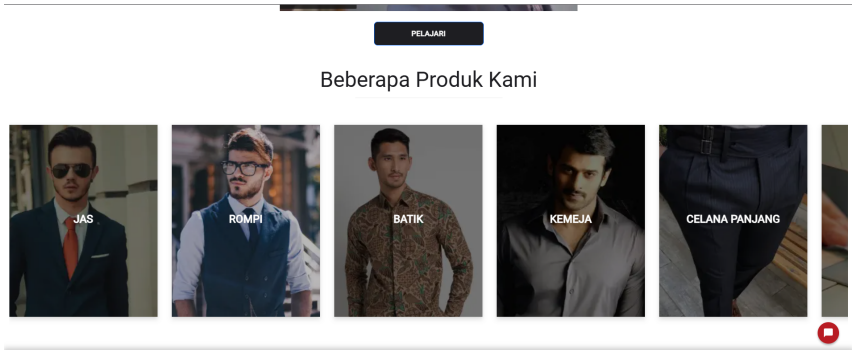
7. **Kustomisasi Placket**
   Pengguna dapat memilih untuk mengkustomisasi bagian placket (daerah kancing baju). Jika tidak, sistem akan menggunakan pengaturan *default placket*.

8. **Kustomisasi Pleats**
   Pengguna dapat memilih untuk mengkustomisasi lipatan (*pleats*). Jika tidak, sistem akan menggunakan pengaturan *default pleats*.

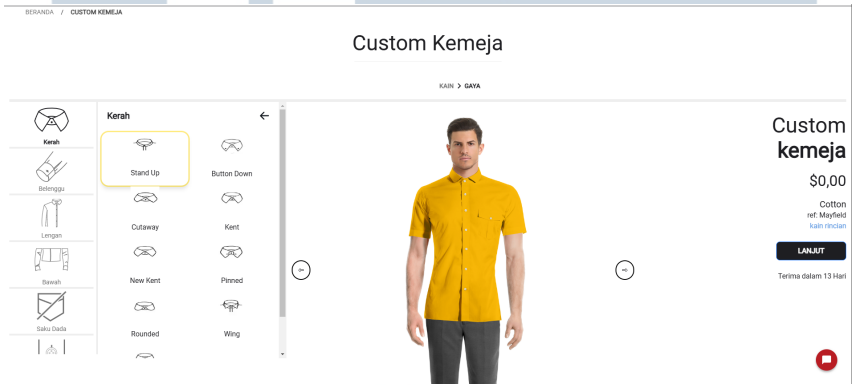
9. **Kustomisasi Sleeve**
   Pengguna dapat memilih untuk mengkustomisasi lengan baju (*sleeve*). Jika tidak, sistem akan menggunakan pengaturan *default sleeve*.

10. **Tambahkan ke Keranjang**
    Setelah semua kustomisasi selesai, pengguna dapat menambahkan baju yang telah dikustomisasi ke dalam *cart* atau keranjang belanja.

11. **Selesai**
    Proses berakhir, dan pengguna dapat melanjutkan ke langkah pembayaran atau kembali untuk membuat kustomisasi lainnya.



Gambar 3.8. Tampilan Halaman Pemilihan Baju



Gambar 3.9. Tampilan Halaman Kustomisasi Pakaian

Ukuran pakaian juga dapat di-kustomisasi namun *user* harus melakukan pengukuran sendiri, setelah *user* melakukan pengukuran mandiri maka *user* dapat melakukan input data ukuran yang pas bagi *user*, hal-hal yang dapat di-kustomisasi adalah:

- **Ukuran Pakaian**:
  - **Kerah *(Neck Size)***: kerah pakaian diukur mengelilingi bagian bawah leher atau kerah kemeja.
    - Ukuran umum: 35–45 cm (dengan interval 2,5 cm).
  - **Lingkar Tangan (*Sleeve Length*)**: Diukur dari bagian belakang leher (persilangan dengan garis bahu) menuju pergelangan tangan.
    - Ukuran umum: 81–94 cm (dengan interval 2,5 cm).
  - **Lingkar Pinggang (*Waist*)**: Diukur di sekitar bagian pinggang (di sekitar area pusar atau sedikit lebih tinggi).
    - Ukuran umum: 71–102 cm (dengan interval 5 cm).

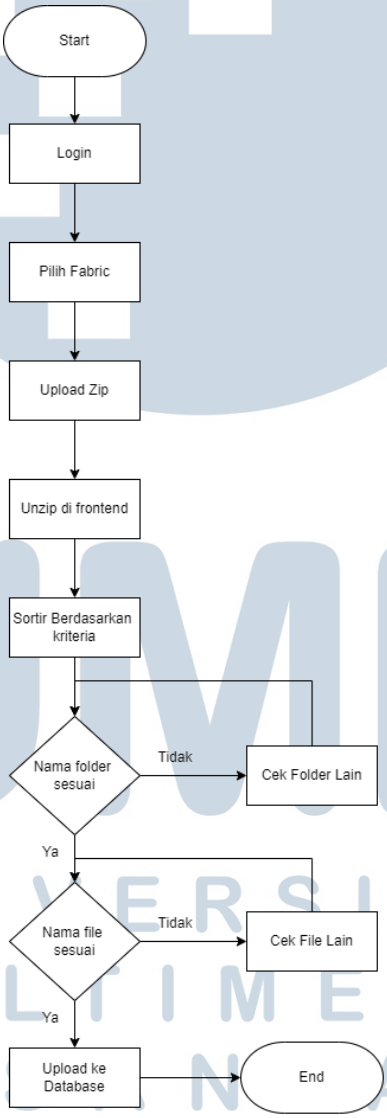
– **Tinggi Baju (*Shirt Length*)**: Diukur dari garis leher (kerah) hingga ke bagian bawah baju, memperhitungkan panjang yang diinginkan.
    * Ukuran umum: 71–86 cm (tergantung panjang tubuh dan desain baju).

- **Tabel Ukuran Standar dalam ukuran centimeter(cm)**:

Ukuran	Kerah	Lingkar Pinggang	Lingkar Tangan	Tinggi Baju
S (Small)	35–38	71–76	81–84	71–74
M (Medium)	38–41	76–81	84–86	74–76
L (Large)	41–43	81–86	86–89	76–79
XL (Extra Large)	43–46	86–91	89–92	79–81

C. Pengembangan Fitur Bulk Upload

Fitur *bulk upload* pada halaman admin mengharuskan administrator mengunggah file dalam jumlah besar secara sekaligus dengan format ZIP. Setelah file ZIP diunggah, sistem akan mengekstrak file yang ada di dalamnya, memvalidasi format dan integritas setiap file, lalu mengunggah file-file tersebut ke database beserta metadata yang relevan. Proses ini dirancang untuk memberikan kemudahan dan efisiensi bagi perusahaan XYZ untuk mengunggah banyak file sekaligus, alur dari fitur *bulk upload* ini dapat digambarkan pada *flowchart* 3.10

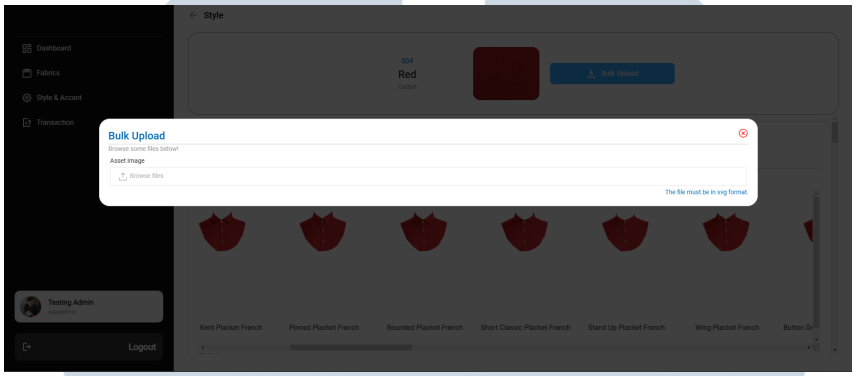


Gambar 3.10. Flowchart Bulk Upload

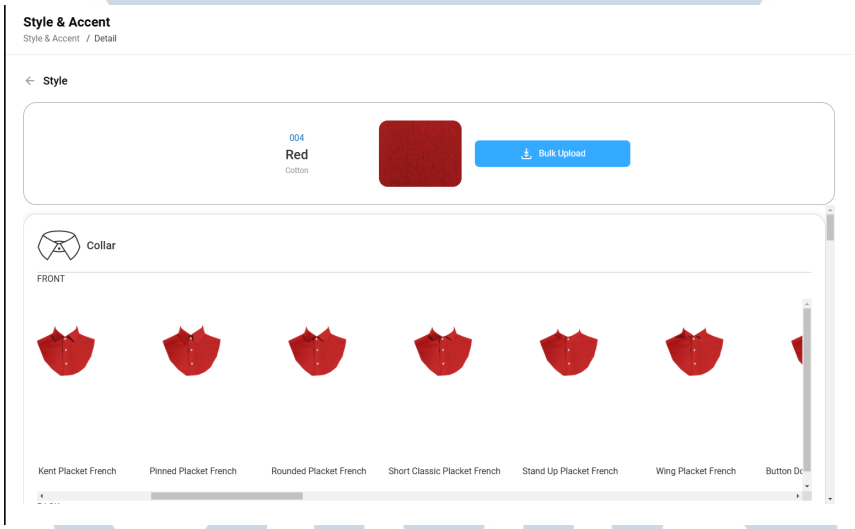
Untuk memahami *flowchart* ini lebih mendalam, Berikut adalah deskripsi lengkap dari setiap tahap yang ditampilkan dalam *flowchart* tersebut:

1. **Login**: Pengguna memulai dengan *login* ke aplikasi sistem untuk dapat mengakses fitur *upload* file.
2. **Pilih Fabric**: Setelah berhasil *login*, pengguna memilih jenis *fabric* yang akan diproses, yang akan menentukan pengolahan file lebih lanjut.
3. **Upload ZIP**: Pengguna mengunggah file ZIP yang berisi data fabric dalam format zip untuk diproses, tampilan input file zip pada web dapat dilihat pada gambar 3.11.
4. **Unzip di Frontend**: Sistem mengekstrak file ZIP di *frontend* (di sisi pengguna) untuk memisahkan file yang ada di dalamnya, sehingga bisa diproses lebih lanjut.
5. **Sortir Berdasarkan Kriteria**:
    - Sistem melakukan sortir terhadap file yang telah diekstrak berdasarkan kriteria yang telah ditentukan.
    - Kriteria sortir:
        - Jika folder di dalam ZIP memiliki nama ”front”, ”back”, atau ”folded” pada level 2, maka folder tersebut akan diproses.
        - Jika nama file adalah salah satu dari ”cuffs”, ”collar”, ”bottom”, ”placket”, ”pleats”, atau ”sleeve”, maka file tersebut akan diproses.
6. **Nama folder sesuai**: Sistem memeriksa apakah folder yang ada di dalam ZIP memiliki nama yang sesuai dengan kriteria, yaitu ”front”, ”back”, atau ”folded”. Jika iya, folder tersebut akan diproses.
7. **Nama file sesuai**: Setelah memeriksa folder, sistem melanjutkan untuk memeriksa nama file. Jika nama file cocok dengan kriteria seperti ”cuffs”, ”collar”, ”bottom”, dll., file tersebut akan diproses.
8. **File/Folder Dibaca**: Jika folder atau file memenuhi kriteria, file tersebut dibaca dan diproses lebih lanjut untuk tahap *upload* ke database.
9. **File/Folder Diabaikan**: Jika folder atau file tidak memenuhi kriteria, maka file atau folder tersebut diabaikan dan tidak diproses lebih lanjut.
10. **Upload Bulk ke Database**: Setelah proses sortir selesai dan file yang valid telah ditemukan, file-file yang memenuhi kriteria akan diunggah ke database secara massal.

11. **Selesai**: Proses selesai, dan file-file yang telah diunggah berhasil disimpan di database, pengguna dapat melihat file-file yang telah diunggah di halaman *style & accents* seperti yang terlihat pada gambar 3.12.



Gambar 3.11. Tampilan Input Bulk Upload



Gambar 3.12. Halaman Bulk Upload

*B*erikut adalah potongan kode yang melakukan *handle features bulk upload*: 3.1.

```
const handleBulkUploadZip = async (file) => {
        console.log(file);
        try {
            const zip = await JSZip.loadAsync(file);
            const matchingEntries = [];
            zip.forEach((relativePath, zipEntry) => {
                // Memisahkan path menjadi bagian-bagian
                const pathParts = zipEntry.name.split('/');
                if (pathParts.length < 2) return; // Pastikan ada setidaknya satu folder dan file
                const folderName = pathParts[0];
                const fileName = pathParts[pathParts.length - 1];

                // Mencari 'Front', 'Back', atau 'Folded' dalam nama folder (tidak sensitif huruf)
                const povMatch = folderName.match(/Front|Back|Folded/i);

                if (povMatch && zipEntry.name.endsWith('.svg')) {
                    matchingEntries.push({
                        zipEntry,
                        pov: povMatch[0].toUpperCase(), // Mengonversi ke huruf kapital jika diperlukan
                    });
                }
            });

            const filePromises = matchingEntries.map(async ({ zipEntry, pov }) => {
                try {
                    console.log(zipEntry);
                    const blob = await zipEntry.async("blob");
                    const url = window.URL.createObjectURL(blob);
                    console.log(url);

                    const code = removeExtension(removePath(zipEntry.name));
                    const fileObj = new File([blob], zipEntry.name, { type: blob.type });

                    let style = '';
                    // Menentukan style berdasarkan kode
```

```
                    if (/collar/i.test(code)) style = 'Collar';
                    else if (/cuffs/i.test(code)) style = 'Cuffs';
                    else if (/sleeve/i.test(code)) style = 'Sleeve
';
                    else if (/bottom/i.test(code)) style = 'Bottom
';
                    else if (/chestpocket/i.test(code)) style = '
Chestpocket';
                    else if (/pleats/i.test(code)) style = 'Pleats
';
                    else if (/placket/i.test(code)) style = '
Placket';
                    else {
                        throw new Error(`Style tidak dikenali
untuk file: ${zipEntry.name}`);
                    }

                    const name = code.replace(/_/g, ' ')
                        .replace(new RegExp(style, 'i'), '')
                        .replace(new RegExp(pov, 'i'), '')
                        .trim();
                        console.log(name);
                    if(name === "") {
                        throw new Error(`Name tidak dikenali untuk
 file: ${zipEntry.name}`);
                    }
                    return {
                        code,
                        files: fileObj,
                        name,
                        style,
                        pov, // Menambahkan pov di sini
                    };
                } catch (error) {
                    console.error('Error processing zip entry:',
error);
                    return { error: error.message, zipEntryName:
zipEntry.name };
                }
            });

            const processedFiles = await Promise.all(filePromises)
;
```

```
69
70          const validFiles = processedFiles.filter(file => !file
    .error);
71          const erroredFiles = processedFiles.filter(file =>
    file.error);
72
73          if (validFiles.length > 0) {
74              console.log('Valid Files:', validFiles);
75              setFileZip(validFiles);
76          }
77          if (erroredFiles.length > 0) {
78              console.warn('Errored Files:', erroredFiles);
79          }
80
81      } catch (error) {
82          console.error('Error handling bulk upload:', error);
83      }
84  };
```

Kode 3.1: Konfigurasi *tax number*

3.4 Kendala dan Solusi yang Ditemukan

Selama periode magang di Bestada dilakukan ditemui beberapa permasalahan antara lain:

A. Kendala Yang Dihadapi

1. Ketika melakukan *hosting* ke server maka projek harus terlebih dilakukan *build* dengan next build, namun build dengan cara ini menemui masalah yaitu memori pada server Bestada mengelami *overload* karena *build* projek yang besar.
2. Kendala komunikasi dengan klien akibat perbedaan pemahaman antara apa yang dilakukan oleh *developer* dan apa yang diinginkan klien.

Untuk mengatasi permasalahan tersebut solusi yang diterapkan adalah sebagai berikut:

B. Solusi yang Ditemukan

1. Untuk mengatasi permasalahan *build* projek yang terlalu berat, maka yang perlu dilakukan adalah melakukan *build* tanpa *cache*, hal ini secara signifikan mengurangi jumlah memori yang perlu dialokasikan pada server namun hal ini mengurangi kecepatan loading ketika pengguna memasuki web, walaupun terdapat efek samping solusi ini dinyatakan cukup tepat untuk mengurangi *overload* akibat file build project yang terlalu besar.
2. Untuk mengatasi permasalahan mengenai kesalahan komunikasi dengan klien maka tim developer perlu menjelaskan secara detail kepada klien apa yang bisa dilakukan dan apa yang tidak bisa dilakukan oleh tim developer, tidak jarang juga tim developer mengingatkan kepada klien bahwa fitur yang diminta sudah melewati *scope* projek yang telah direncanakan sebelumnya.