Jurnal Informatika dan Sistem Informasi (JIFoSI)

Volume 7 Nomor 2 Agustus 2026
ISSN (Online): 2722-130X
Halaman: 73-85

IMPLEMENTASI REUSABLE COMPONENTS APLIKASI MULTI-BISNIS BERBASIS REACT

* [1] **Tahta Karisma Tiwi Shine Prameswari,** [1]**Dinda Aisa Selvira,** [1]**Amriza Saskirana**
[1]Fakultas Ilmu Komputer, Universitas Pembangunan Nasional "Veteran" Jawa Timur, Indonesia

DOI: https://doi.org/10.33005/jifosi.v7i2.613

*Corresponding Author: 23082010130@student.upnjatim.ac.id

Submit: Juni 2026 | Accepted: Juli 2026 | Published: Agustus 2026

ABSTRAK

Pengembangan platform multi-bisnis "Bisnisyu" menghadapi kendala duplikasi kode dan inkonsistensi antarmuka pada enam modul operasionalnya. Penelitian ini mengusulkan implementasi arsitektur reusable components terpusat berbasis React dan TypeScript menggunakan kontrak data statis (props) serta mekanisme tema dinamis via custom hook useTheme(). Karena dilakukan secara retroaktif, evaluasi kuantitatif diukur menggunakan pendekatan Hypothetical Baseline. Hasil pengujian runtime menunjukkan komponen tunggal mampu mengadaptasi identitas visual tiap jenis usaha secara presisi. Secara struktural, arsitektur ini memberikan dampak efisiensi signifikan dengan estimasi reduksi redundansi kode mencapai 84,9% pada komponen Button dan 79,2% pada Header. Penelitian ini membuktikan bahwa sentralisasi komponen berbasis modular efektif meningkatkan maintainability basis kode dan konsistensi antarmuka sistem multi-bisnis.

Kata Kunci: Komponen Reusable, React, Typescript, Multi-bisnis, Redundansi kode, Tema Dinamis

IMPLEMENTATION OF REUSABLE COMPONENTS IN A REACT-BASED MULTI-BUSINESS APPLICATION

ABSTRACT

The development of the "Bisnisyu" multi-business platform faces challenges related to code duplication and interface inconsistencies across its six operational modules. This study proposes the implementation of a centralized reusable components architecture based on React and TypeScript, utilizing static data contracts (props) and a dynamic theme mechanism via the custom hook `useTheme()`. Since this was implemented retroactively, quantitative evaluation was measured using the Hypothetical Baseline approach. Runtime test results show that a single component is capable of precisely adapting the visual identity of each business type. Structurally, this architecture yields significant efficiency gains, with an estimated reduction in code redundancy of 84.9% for the Button component and 79.2% for the Header component. This study demonstrates that modular, component-based centralization effectively improves the maintainability of the codebase and the consistency of the user interface in a multi-business system.

Keywords: Reusable Components, React, TypeScript, Multi-business, Code Redundancy, Dynamic Themes

PENDAHULUAN

Perkembangan aplikasi web modern menuntut antarmuka yang modular, konsisten, dan mudah dipelihara. Antarmuka pengguna berperan sebagai media interaksi antara pengguna dan sistem, sehingga rancangan antarmuka perlu mudah dipahami dan sesuai kebutuhan pengguna [2]. Pendekatan berbasis komponen menjadi relevan karena elemen antarmuka dapat dipisahkan menjadi unit mandiri dan digunakan kembali. React mendukung pola ini melalui komponen modular, composition through props, custom hooks, dan pengelolaan state yang terstruktur [1]. Komponen React juga dapat dikombinasikan untuk membentuk halaman kompleks tanpa menuliskan ulang kode yang sama [1]. Selain itu, kualitas antarmuka berkaitan dengan usability dan kepuasan pengguna, sehingga konsistensi tampilan perlu dijaga dalam aplikasi berbasis web [3].

Kebutuhan tersebut muncul dalam pengembangan platform Bisnisyu. Bisnisyu merupakan aplikasi manajemen bisnis berbasis web yang dikembangkan dalam satu codebase menggunakan React dan TypeScript. Platform ini mendukung beberapa jenis usaha, yaitu Laundry, Restoran/Cafe, Klinik Kecantikan, Barbershop, Rental, dan Online Shop. Setiap jenis usaha memiliki alur operasional dan kebutuhan visual berbeda, tetapi tetap menggunakan pola antarmuka dasar yang serupa, seperti navigasi, tombol aksi, formulir input, dan dialog konfirmasi. Karena antarmuka menjadi media interaksi pengguna dengan sistem, rancangan antarmuka perlu konsisten dan mudah dipahami [2]. Kondisi ini membuat Bisnisyu perlu mendukung fleksibilitas tampilan sesuai konteks bisnis sekaligus menjaga konsistensi pengalaman pengguna antar modul, sejalan dengan konsep pengembangan berbasis komponen [1].

Seiring bertambahnya modul bisnis dan halaman operasional, pengembangan antarmuka Bisnisyu menghadapi masalah duplikasi kode. Elemen seperti tombol, header, form input, dan dialog konfirmasi digunakan berulang kali, tetapi masih ditulis terpisah sesuai kebutuhan tiap modul. Kondisi ini membuat struktur kode kurang efisien, sulit ditelusuri saat terjadi perubahan, dan rawan menimbulkan inkonsistensi antarmuka. Pada pengembangan frontend, duplikasi logika UI dan markup berulang dapat meningkatkan kompleksitas kode, sedangkan reusable component dapat mendorong code sharing dan menyederhanakan maintainability [1]. Refactoring juga digunakan untuk meningkatkan kualitas perangkat lunak dan reusability melalui analisis metrik struktural [4]. Selain itu, kualitas kode yang rendah berkaitan dengan meningkatnya defect dan waktu penyelesaian issue yang lebih lama [5].

Permasalahan tersebut perlu ditangani karena duplikasi komponen antarmuka berdampak langsung pada beban pemeliharaan. Penelitian mengenai component-based development menunjukkan bahwa reusable component dapat meningkatkan produktivitas, kualitas, dan maintainability perangkat lunak [6]. Pendekatan ini mendukung reusability yang tinggi ketika komponen dikembangkan secara terstruktur, terdokumentasi, dan digunakan secara konsisten dalam arsitektur sistem [7]. Studi lain menunjukkan bahwa reusability komponen dipengaruhi oleh complexity, maintainability, quality, adaptability, dan reuse frequency [8]. Thakral et al. [9] juga menegaskan bahwa CBSE dapat mengurangi waktu dan biaya pengembangan serta meningkatkan kualitas perangkat lunak. Dengan demikian, reusability, maintainability, dan kualitas komponen saling berkaitan dalam pengembangan sistem yang efisien dan mudah dipelihara.

Meskipun software reuse dan pengembangan berbasis komponen telah banyak dibahas, sebagian besar kajian masih berfokus pada komponen perangkat lunak, proses pengembangan, atau strategi reuse secara umum [6], [8], [10]. Kajian tentang strategi penggunaan kembali juga menyoroti manfaat berupa efisiensi biaya, percepatan pengembangan, peningkatan kualitas, serta kendala seperti sulit menemukan komponen yang tepat, integrasi yang rumit, dan dokumentasi yang belum memadai [10]. Studi terkait reusability metrics menegaskan bahwa keterpakaian kembali komponen bergantung pada konteks penggunaan, karakteristik komponen, dan kemudahan adaptasinya [8]. Namun, pembahasan reusable component pada sisi antarmuka pengguna dalam platform multi-bisnis dengan beragam identitas visual dalam satu codebase masih terbatas.

Berdasarkan celah tersebut, penelitian ini mengusulkan arsitektur reusable component pada sisi antarmuka pengguna untuk mengurangi duplikasi kode dan inkonsistensi tampilan pada Bisnisyu. Solusi ini diwujudkan melalui empat komponen dasar, yaitu Button, Header, Input, dan ConfirmDialog, yang ditempatkan dalam direktori terpusat dan digunakan kembali melalui konfigurasi props. Pendekatan ini sejalan

dengan prinsip component-based development yang menekankan struktur, dokumentasi, konsistensi, reusability, dan maintainability [7]. Dalam React, komponen dapat dirancang sebagai unit modular yang menerima konfigurasi melalui props, memanfaatkan custom hooks, dan mendukung code sharing [1]. Penerapan Tailwind CSS memungkinkan variasi gaya dan perilaku komponen diatur melalui konfigurasi seperti variant, size, dan className [11]. Selain itu, mekanisme tema dinamis melalui custom hook useTheme() digunakan agar setiap modul bisnis dapat memiliki identitas visual berbeda tanpa mengubah struktur komponen inti. Rancangan ini diharapkan dapat meningkatkan konsistensi tampilan, mengurangi kompleksitas pemeliharaan, dan mendukung skalabilitas antarmuka Bisnisyu [8], [10].

Penelitian ini bertujuan untuk merancang dan mengimplementasikan reusable component pada platform Bisnisyu berbasis React dan TypeScript, dengan fokus pada Button, Header, Input, dan ConfirmDialog. Penelitian ini juga menguji kemampuan adaptasi komponen terhadap enam modul bisnis yang berbeda serta mengevaluasi tingkat reusability berdasarkan frekuensi penggunaan pada berbagai halaman dan fleksibilitas konfigurasi melalui props serta tema dinamis. Hasil penelitian diharapkan dapat memberikan kontribusi terhadap perancangan arsitektur antarmuka platform multi-bisnis agar pengembangan lebih efisien, konsisten, dan mudah dipelihara.

METODOLOGI

Penelitian ini menggunakan pendekatan rekayasa perangkat lunak (software engineering) dengan menerapkan pengembangan reusable component pada antarmuka Platform Bisnisyu berbasis React dan TypeScript. Pendekatan ini dipilih karena penelitian berfokus pada perancangan, implementasi, serta evaluasi komponen antarmuka yang dapat digunakan kembali pada berbagai modul bisnis dalam satu codebase. Tahapan penelitian disusun secara sistematis agar proses pengembangan dapat direplikasi oleh peneliti lain, mulai dari identifikasi kebutuhan komponen, perancangan kontrak data, implementasi komponen, pengujian pada seluruh modul bisnis, hingga evaluasi tingkat reusability. Alur penelitian yang digunakan ditunjukkan pada Gambar 1.

Gambar 1. Flowchart Tahapan Implementasi dan Pengujian Komponen

Tahap pertama adalah *identification*, yaitu mengidentifikasi komponen antarmuka yang memiliki tingkat penggunaan paling tinggi pada seluruh halaman aplikasi. Proses identifikasi dilakukan menggunakan fitur *Find References* pada Visual Studio Code untuk menelusuri seluruh pemanggilan komponen di dalam codebase. Dari proses tersebut diperoleh empat komponen utama yang paling sering digunakan, yaitu *Button, Header*, *Input*, dan *ConfirmDialog*. Keempat komponen tersebut dipilih sebagai objek penelitian karena memiliki tingkat duplikasi yang tinggi sehingga berpotensi memberikan dampak terbesar terhadap efisiensi pengembangan apabila diubah menjadi *reusable component*.

Tahap berikutnya adalah *design*, yaitu merancang struktur *reusable component* menggunakan pendekatan kontrak data (*contract-based design*). Pada tahap ini setiap komponen didefinisikan menggunakan interface TypeScript sehingga seluruh parameter masukan (*props*) memiliki tipe data yang jelas dan konsisten. Perancangan ini bertujuan untuk meningkatkan keamanan tipe data (*type safety*), mempermudah penggunaan kembali komponen, serta mengurangi potensi kesalahan implementasi yang dapat terjadi sebelum aplikasi dijalankan.

Tahap *implementation* dilakukan dengan membangun setiap reusable component sebagai berkas terpisah di dalam direktori komponen bersama (*shared components*). Pengembangan dilakukan menggunakan React Functional Component dengan dukungan TypeScript dan Tailwind CSS sebagai kerangka utama antarmuka. Selain itu, mekanisme tema dinamis diimplementasikan menggunakan *custom hook useTheme()*

sehingga satu komponen yang sama dapat menyesuaikan identitas visual masing-masing modul bisnis tanpa memerlukan perubahan pada struktur logika komponen.

Setelah implementasi selesai, dilakukan tahap *testing* untuk memastikan seluruh komponen dapat digunakan pada berbagai modul bisnis yang terdapat di Platform Bisnisyu, yaitu Laundry, Restoran, Klinik Kecantikan, Barbershop, Rental, dan Online Shop. Pengujian dilakukan dengan mengintegrasikan komponen ke dalam setiap modul kemudian mengamati kesesuaian fungsi, tampilan antarmuka, serta kemampuan adaptasi tema ketika aplikasi dijalankan (*runtime*).

Tahap terakhir adalah evaluation, yaitu mengevaluasi tingkat keterpakaian kembali (reusability) dan efisiensi ruang kode dari setiap komponen yang telah dikembangkan. Karena penelitian dilakukan secara retroaktif pada proyek yang telah selesai dikembangkan, evaluasi kuantitatif dilakukan menggunakan pendekatan Hypothetical Baseline. Efisiensi arsitektur komponen diukur secara matematis berdasarkan rasio perbandingan antara estimasi akumulasi baris kode konvensional (menggunakan asumsi kebutuhan minimal 20 baris per elemen UI mandiri dikalikan frekuensi sebaran file hasil temuan) terhadap total baris kode aktual pasca-sentralisasi komponen. Hasil evaluasi ini digunakan sebagai pembuktian empiris mengenai efektivitas penerapan reusable component dalam mereduksi redundansi kode sekaligus menjaga konsistensi antarmuka pada Platform Bisnisyu.

Alat dan Stack Teknologi Pendukung

Penelitian ini menggunakan beberapa perangkat lunak dan teknologi yang saling mendukung dalam proses pengembangan *reusable component*. React 18 digunakan sebagai pustaka JavaScript berbasis komponen karena menyediakan mekanisme *component composition* yang memungkinkan antarmuka dibangun dari komponen-komponen modular yang dapat digunakan kembali. Sementara itu, TypeScript digunakan sebagai bahasa pemrograman utama untuk menyediakan mekanisme *static type checking* sehingga kesalahan implementasi dapat dideteksi sejak proses kompilasi sebelum aplikasi dijalankan.

Selain itu, Tailwind CSS digunakan sebagai *utility-first* CSS framework untuk membangun tampilan antarmuka secara modular melalui kombinasi kelas utilitas tanpa harus menuliskan berkas CSS secara terpisah. Pendekatan ini memudahkan pengelolaan variasi tampilan komponen pada berbagai kondisi penggunaan. Penelitian ini juga memanfaatkan *custom hook useTheme()* sebagai mekanisme pengelolaan tema dinamis yang bertugas mendistribusikan identitas visual, seperti warna utama dan gradasi, kepada seluruh komponen sehingga tampilan antarmuka dapat menyesuaikan karakteristik masing-masing modul bisnis secara otomatis.

```
"dependencies": {
    "@mui/icons-material": "^7.3.8",
    "axios": "^1.13.6",
    "react": "^19.2.0",
    "react-dom": "^19.2.0",
    "react-router-dom": "^7.13.0"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^5.1.1",
    "tailwindcss": "^3.4.4",
    "typescript": "~5.9.3",
    "vite": "^7.3.1"
  }
```

Gambar 2. Berkas package.json Proyek Bisnisyu Frontend

Arsitektur dan Spesifikasi Berkas Komponen

Untuk meningkatkan keteraturan struktur proyek, seluruh komponen yang mengalami proses refaktorisasi ditempatkan ke dalam direktori khusus *components/ui* sehingga terpisah dari logika bisnis aplikasi. Pemisahan struktur ini bertujuan untuk meningkatkan keterbacaan kode, mempermudah proses pemeliharaan, serta memungkinkan setiap komponen digunakan kembali pada berbagai halaman tanpa perlu

melakukan penyalinan kode. Dengan pendekatan tersebut, perubahan pada satu komponen hanya dilakukan pada satu lokasi sehingga seluruh halaman yang menggunakan komponen tersebut akan memperoleh pembaruan secara otomatis.

Table 1. Spesifikasi Berkas Komponen Reusable

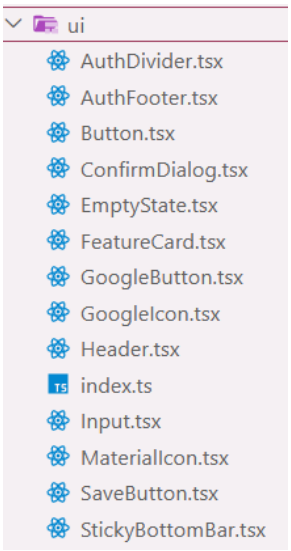
Nama Komponen	Target File Fisik	Detail Jenis Komponen
Button	Button.tsx	Komponen tombol aksi
Header	Header.tsx	Komponen navigasi dan judul halaman
Input	Input.tsx	Komponen formulir masukan pengguna
ConfirmDialog	ConfirmDialog.tsx	Komponen dialog konfirmasi

Sumber: Data Diolah

Keempat komponen tersebut dikembangkan sebagai komponen dasar (*base components*) yang dapat dikonfigurasi melalui *props* sesuai kebutuhan masing-masing modul bisnis. Pendekatan ini memungkinkan satu implementasi komponen digunakan pada berbagai halaman dengan variasi tampilan maupun perilaku yang berbeda tanpa perlu membuat komponen baru. Dengan demikian, struktur proyek menjadi lebih modular, konsisten, dan mudah dikembangkan seiring bertambahnya kebutuhan fitur pada Platform Bisnisyu seperti pada gambar 3.

HASIL DAN PEMBAHASAN

Keempat komponen tersebut dikembangkan sebagai komponen dasar (*base components*) yang dapat dikonfigurasi melalui *props* sesuai kebutuhan masing-masing modul bisnis. Pendekatan ini memungkinkan satu implementasi komponen digunakan pada berbagai halaman dengan variasi tampilan maupun perilaku yang berbeda tanpa perlu membuat komponen baru. Dengan demikian, struktur proyek menjadi lebih modular, konsisten, dan mudah dikembangkan seiring bertambahnya kebutuhan fitur pada Platform Bisnisyu.



Gambar 3. Representasi Berkas Komponen dalam Direktori Proyek

Identifikasi Kebutuhan Komponen

Proses identifikasi kebutuhan komponen dilakukan dengan memanfaatkan fitur pencarian global dan find references pada editor kode Visual Studio Code untuk melacak setiap pemanggilan elemen antarmuka di seluruh basis kode proyek. Hasil penelusuran menunjukkan bahwa elemen tombol memiliki tingkat duplikasi tertinggi, diikuti oleh elemen header, formulir input, dan kotak dialog.

Table 2. Hasil Identifikasi Penggunaan Elemen Antarmuka

Elemen	Jumlah Hasil	Jumlah File	Komponen Target
Button	784	149	Button.tsx
Header	414	127	Header.tsx
Input	160	37	Input.tsx
Dialog	112	13	ConfirmDialog.tsx

Sumber: Data Diolah

Tingginya angka sebaran ini mengindikasikan bahwa keempat komponen tersebut merupakan kandidat utama untuk direfaktor menjadi komponen *reusable* yang terpusat, sehingga upaya sentralisasi akan sangat bermanfaat untuk memastikan konsistensi tampilan dan perilaku komponen di seluruh aplikasi.

Desain Interface Props (Arsitektur Kontrak Data)

Perancangan kontraktual objek dilakukan dengan menggunakan tipe data statis pada props untuk membatasi nilai input agar editor mampu mendeteksi kesalahan penulisan kode sebelum program dijalankan (*compile-time handling*). Pendekatan ini memanfaatkan sistem tipe TypeScript untuk memastikan setiap komponen menerima data sesuai spesifikasi yang telah ditentukan, sehingga mengurangi potensi kesalahan logika di tahap *runtime*. Berikut adalah rincian desain antarmuka untuk masing-masing komponen.

Table 3. Spesifikasi Interface Props Komponen Button

Props	Tipe Data	Deskripsi
variant	'*primary*' | '*secondary*' | '*outline*' | '*text*' | '*google*' | '*dashed*' | '*save*'	Mengunci 7 pilihan gaya visual tombol
size	'sm' | 'md' | 'lg'	Mengatur skala tampilan tombol
fullWidth	*boolean*	Mengatur skala tampilan tombol
loading	*boolean*	Mengontrol status *loading* dengan animasi *spinner*
loadingText	*string*	Teks alternatif saat *loading*
icon	*string*	Nama ikon *material symbols* di sisi kiri atau kanan
iconPosition	'*left*' | '*right*'	Posisi ikon terhadap teks

Sumber: Data Diolah

Interface ButtonProps dirancang dengan *prop variant* untuk menentukan gaya *visual*, *size* untuk skala tampilan, serta loading dan *loadingText* untuk memberikan umpan balik saat proses asinkron berlangsung. *Prop fullWidth* memungkinkan tombol memanjang mengisi lebar kontainer, sementara *prop icon* dan *iconPosition* memberikan fleksibilitas penambahan ikon dekoratif. Desain ini memungkinkan komponen tombol diadaptasi ke berbagai konteks penggunaan tanpa modifikasi internal.

```
type ButtonVariant = 'primary' | 'secondary' | 'outline' | 'text' |
                     'google' | 'dashed' | 'save'
type ButtonSize = 'sm' | 'md' | 'lg'

interface ButtonProps extends ButtonHTMLAttributes<HTMLButtonElement> {
  variant?: ButtonVariant
  size?: ButtonSize
  fullWidth?: boolean
  icon?: string
  iconPosition?: 'left' | 'right'
  children: ReactNode
  loading?: boolean
  loadingText?: string
}

const sizeClasses = {
  sm: 'h-12 px-4 text-sm',
  md: 'h-14 px-5 text-base',
  lg: 'h-16 px-5 text-lg',
}
```

Gambar 4. Code Interface Props Komponen Button

Table 4. Spesifikasi Interface Props Komponen Header

Props	Tipe Data	Deskripsi
type	*'dashboard' | 'back' | 'search' | 'search-no-back' | 'profile' | 'profile-search' | 'profile-back' | 'title-only'*	Menentukan 8 tipe *layout header*
title	*string*	Judul *header*
onBack	*() => void*	Fungsi *callback* tombol kembali
rightElement	*ReactNode*	Elemen kustom di sisi kanan *header*
searchQuery	*string*	Nilai pencarian (untuk tipe *search*)
onSearchChange	*(query: string) => void*	*Callback* perubahan pencarian

Sumber: Data Diolah

```
type HeaderType = 'dashboard' | 'back' | 'search' | 'search-no-back' |
                  'profile' | 'profile-search' | 'profile-back' | 'title-only';

interface HeaderProps {
  type: HeaderType;
  title?: string;
  onBack?: () => void;
  rightElement?: React.ReactNode;
  onDelete?: () => void;
  isDeleting?: boolean;
  searchQuery?: string;
  onSearchChange?: (query: string) => void;
  searchPlaceholder?: string;
  businessName?: string;
  onSwitchBusiness?: () => void;
  onNotifications?: () => void;
  hasNotification?: boolean;
}
```

Gambar 5. Code Interface Props Komponen Header

Interface HeaderProps menggunakan *prop type* untuk menentukan delapan variasi *layout header*, sehingga satu komponen dapat melayani seluruh kebutuhan navigasi. *Prop rightElement* memanfaatkan ReactNode untuk menyisipkan elemen kustom seperti tombol unduh PDF atau hapus. Komponen ini juga terintegrasi dengan hook tema untuk menyesuaikan gradasi warna latar belakang sesuai identitas bisnis yang aktif.

Table 5. Spesifikasi Interface Props Komponen Input

Props	Tipe Data	Deskripsi
label	*string*	Teks penunjuk di atas *input*
icon	*string*	Ikon dekoratif di sisi kiri
iconRight	*string*	Ikon di sisi kanan
error	*string*	Pesan *error* dengan *styling* merah
helperText	*string*	Teks bantuan di bawah input
type	*string*	Tipe *input*

Sumber: Data Diolah

```
interface InputProps extends InputHTMLAttributes<HTMLInputElement> {
  label?: string
  labelRight?: ReactNode
  icon?: string
  iconRight?: string
  error?: string
  helperText?: string
  onIconRightClick?: () => void
}
```

Gambar 6. Code Interface Props Komponen Header

Interface InputProps menyediakan *prop* label untuk konteks formulir, *icon* dan *iconRight* untuk dekorasi, serta *error* dan *helperText* untuk umpan balik pengguna. *Prop error* mengubah gaya visual menjadi status *error* dengan *border* merah, sementara *helperText* menampilkan petunjuk atau pesan kesalahan di bawah input. Desain ini memungkinkan komponen input beradaptasi dengan berbagai kebutuhan formulir di seluruh modul bisnis.

Implementasi Berkas Komponen

Transisi dari tahap desain kontrak data menuju penulisan fungsi manipulasi DOM dilakukan ke dalam berkas berekstensi .tsx dengan total kode mencapai lebih dari 700 baris gabungan untuk keempat komponen.

```
const variantStyles: Record<ButtonVariant, React.CSSProperties> = {
  primary: {
    background: `linear-gradient(to right, ${primaryColor}, ${accentColor})`,
    color: 'white',
    boxShadow: `0px 4px 6px -4px ${primaryBgValue}, 0px 10px 15px -3px ${prim
    transition: 'all 0.2s ease-in-out',
  },
  secondary: {
    background: `linear-gradient(to right, ${primaryColor}, ${accentColor})`,
    color: 'white',
    boxShadow: '0 1px 2px rgba(0,0,0,0.1)',
  },
  outline: {
    backgroundColor: 'white',
    border: '1px solid #E5E7EB',
    color: '#374151',
    boxShadow: '0 1px 2px rgba(0,0,0,0.05)',
  },
  text: {
    backgroundColor: 'transparent',
    color: '#6B7280',
  },
  google: {
    backgroundColor: 'white',
    border: '1px solid #E5E7EB',
    color: '#374151',
    boxShadow: '0 1px 2px rgba(0,0,0,0.05)',
  },
```

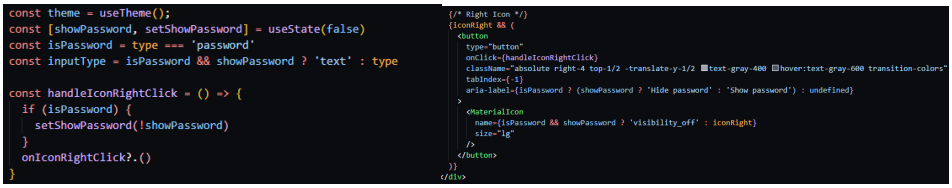
```
{/* Loading state */}
{loading && (
  <>
    <div className="w-4 h-4 border-2 border-current
    border-t-transparent rounded-full animate-spin" />
    <span>{loadingText}</span>
  </>
)}

{/* Normal state */}
{!loading && icon && iconPosition === 'left' && (
  <MaterialIcon name={icon} size="xl" />
)}
{!loading && (
  <span className="flex items-center justify-center">
    {children}
  </span>
)}
{!loading && icon && iconPosition === 'right' && (
  <MaterialIcon name={icon} size="xl" />
)}
</button>
```

Gambar 7. Implementasi Komponen Button

Komponen *Button* pada file Button.tsx memiliki panjang kode lebih dari 300 baris yang menggunakan teknik pemetaan objek literal *variantStyles* untuk mengatur tampilan visual setiap varian tombol. Objek *variantStyles* mendefinisikan properti *CSS* seperti latar belakang gradien menggunakan *linear-gradient*, warna teks, bayangan (*boxShadow*), dan transisi untuk *variant primary*, *secondary*, *outline*, *text*, dan *google*. Pada *variant primary*, latar belakang tombol menggunakan *gradien* dua warna dari *primaryColor* ke *accentColor* yang diambil dari hook tema, dilengkapi dengan bayangan dan efek transisi. Komponen *Button* juga dilengkapi logika kondisional yang merender animasi *spinner* saat *prop loading* bernilai *true*, sehingga memberikan umpan balik visual kepada pengguna bahwa proses sedang berlangsung.

Gambar 8. Implementasi Komponen Header

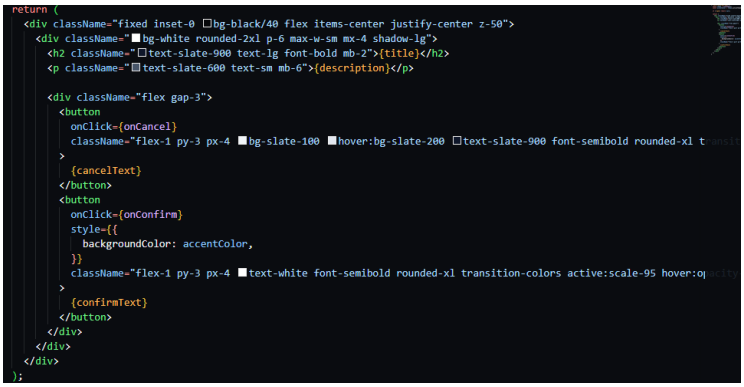
Komponen *Header* pada file Header.tsx memiliki panjang lebih dari 200 baris kode yang menggunakan pendekatan *conditional rendering* dengan struktur *if* (*type* === '...') untuk menangani delapan tipe layout yang berbeda. Kedelapan tipe tersebut adalah *dashboard*, *back*, *search*, *search-no-back*, *profile*, *profile-search*, *profile-back*, dan *title-only*. Setiap tipe menghasilkan struktur DOM yang berbeda, mulai dari *header* dengan tombol kembali dan judul (*tipe back*), *header* dengan bilah pencarian (*tipe search*), hingga header berukuran besar dengan profil dan fitur hapus (t*ipe profile-back*). Komponen ini juga memanfaatkan hook *useTheme*() untuk mengambil nilai *headerGradientValue* yang diterapkan sebagai latar belakang *gradien*, sehingga tampilan *header* otomatis menyesuaikan dengan identitas visual bisnis yang sedang aktif.



Gambar 9. Implementasi Komponen Input

Komponen *Input* pada file Input.tsx mengimplementasikan logika *state management internal* menggunakan hook *useState* untuk mengatur visibilitas kata sandi melalui variabel *boolean showPassword*. Ketika pengguna mengklik ikon mata di sisi kanan *input*, fungsi *handleIconRightClick* akan mengubah nilai *showPassword* sehingga tipe *input* berubah dari *password* menjadi *text* dan sebaliknya, dengan ikon yang ditampilkan juga berganti antara *visibility_off* dan *visibility*. Komponen ini juga menangani fokus dinamis dengan mengubah warna *border* dan menambahkan efek bayangan saat *input* menerima fokus, serta menampilkan pesan *error* dengan gaya visual konsisten menggunakan warna merah pada *border* dan teks. Pendekatan ini meningkatkan *usability* formulir, terutama pada halaman *login* dan *registrasi*, karena pengguna dapat memverifikasi kata sandi yang diketikkan sebelum mengirimkan formulir.

Komponen *ConfirmDialog* pada file ConfirmDialog.tsx mengimplementasikan arsitektur lapisan portal kotak *dialog* dengan membungkus kerangka layout dasar (*backdrop layout*) secara terpusat. Komponen ini menggunakan teknik *fixed positioning* dengan lapisan *backdrop* semi-transparan *(bg-black*/40) untuk menutupi konten di belakangnya, sehingga pengguna fokus pada *dialog* konfirmasi yang muncul. Properti utama yang diterima adalah *open* untuk mengontrol *visibilitas modal*, *title* dan *description* untuk mengatur konten pesan, serta *onConfirm* dan *onCancel* sebagai fungsi *callback*. Warna tombol konfirmasi diambil dari *accentColor* pada hook tema, dengan *fallback* ke #EF4444 (merah) jika tema tidak tersedia. Pendekatan ini memungkinkan satu komponen modal digunakan untuk berbagai keperluan konfirmasi seperti konfirmasi penghapusan data, pembatalan pesanan, atau perubahan status, dengan konsistensi tampilan di seluruh aplikasi.



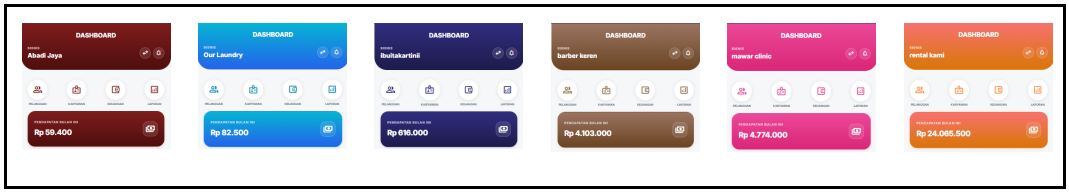
Gambar 10. Implementasi Komponen Dialog

Hasil Pengujian Fungsional dan Adaptasi Tema Multi-Bisnis

Pada fase uji coba jalan (*runtime testing*), satu berkas komponen tunggal terbukti mampu mengenali modul vertikal bisnis yang sedang aktif dan mengeksekusi perubahan warna visual identitas usaha secara presisi berkat integrasi data dinamis dari *useTheme*(). Proses pengujian dilakukan dengan mengaktifkan setiap modul bisnis secara bergantian dan mengamati perubahan tampilan komponen secara *real-time*. Hasil pengujian menunjukkan bahwa seluruh komponen berhasil menyesuaikan warna dan gaya visualnya sesuai dengan tema yang sedang aktif tanpa memerlukan modifikasi kode atau konfigurasi tambahan. Hal ini membuktikan bahwa arsitektur komponen *reusable* yang dirancang mampu beradaptasi dengan perubahan konteks bisnis secara otomatis, sehingga mengurangi kebutuhan untuk memelihara beberapa versi komponen yang berbeda untuk setiap modul bisnis.

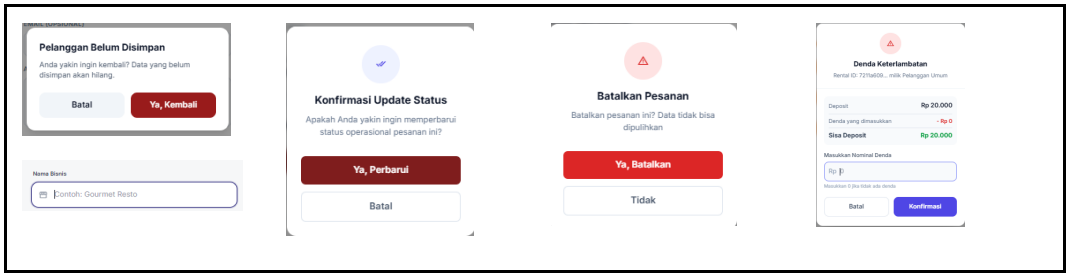
Gambar 11. Hasil Pengujian Komponen Button

Komponen *Button* berhasil menampilkan paduan warna dasar yang keluar otomatis di tiap modul operasional yaitu *Laundry* (biru #0891B2), *Restaurant* (merah #7F1D1D), *Barbershop* (cokelat #8B6F47), *Rental* (oranye #FB923C), dan *Clinic* (pink #EC4899). Seluruh *variant* tombol mempertahankan konsistensi interaksi *hover* dan *active* di setiap tema, di mana tombol memberikan efek perubahan skala dan opasitas saat kursor berada di atasnya atau saat ditekan. Hal ini memastikan bahwa pengguna tetap mendapatkan pengalaman interaksi yang seragam meskipun tampilan visual berubah sesuai dengan identitas bisnis.



Gambar 12. Hasil Pengujian Komponen Header

Komponen *Header* berhasil merender efek gradasi warna yang menyesuaikan identitas visual khas masing-masing jenis usaha, seperti gradasi dari *cyan* menuju biru pada modul *laundry*, merah tua pada restoran, dan gradasi khas lainnya. Efek gradasi ini dicapai dengan memanfaatkan properti *CSS linear-gradient* yang nilainya diambil dari *headerGradientValue* pada tema aktif. Pengguna dapat langsung mengenali identitas bisnis yang sedang diakses hanya dari tampilan *header*.



Gambar 13. Hasil Pengujian Komponen Confirm dan Input

Seluruh status visual komponen *Input* berfungsi dengan baik saat menerima aksi fokus kursor maupun pemicu pesan kesalahan, serta fitur *password visibility toggle* berjalan sesuai harapan. Empat ragam variasi dialog *popup* (*ConfirmDialog*, *ConfirmationModal*, *CancelConfirmationModal*, *LateFeeModal*) menunjukkan stabilitas eksekusi yang konsisten di seluruh modul bisnis. Seluruh modal berhasil menampilkan pesan yang sesuai dengan konteks dan mengeksekusi fungsi *callback* dengan benar.

Analisis Kuantitatif Tingkat Reusability dan Efisiensi Kode

Analisis kuantitatif dan karakteristik struktural komponen dilakukan dengan mengacu pada kerangka klasifikasi tingkat reusability serta pendekatan *Hypothetical Baseline*. Evaluasi ini memetakan efisiensi komponen berdasarkan dua variabel utama, yaitu frekuensi penggunaan ulang (*reuse frequency*) dan tingkat adaptabilitas melalui konfigurasi *props*. Untuk membuktikan efisiensi pemeliharaan kode secara objektif pada proyek yang telah selesai dikembangkan, potensi reduksi baris kode (*Lines of Code*) dihitung dengan mengasumsikan standar minimal kebutuhan 20 baris kode konvensional untuk menyusun elemen antarmuka mandiri (yang mencakup *status loading*, variasi visual, dan interaksi *hover*) jika ditulis secara manual dan berulang di setiap berkas operasional.

Berdasarkan data identifikasi awal, elemen tombol (*Button*) digunakan sebanyak 784 kali di 149 berkas fisik. Tanpa implementasi komponen *reusable*, penulisan elemen ini secara konvensional diestimasikan akan memakan ruang hingga 2.980 baris kode ($149 \text{ berkas} \times 20 \text{ baris}$). Melalui sentralisasi pada berkas Button.tsx (300 baris) dan 149 baris deklarasi pemanggilan komponen, total kode aktual yang perlu dipelihara hanya 449 baris, yang berarti berhasil mereduksi redundansi kode sebesar 84,9%.

Table 6. Klasifikasi Karakteristik dan Tingkat Keterpakaian Komponen

Komponen	***Reuse Frequency***	***Adaptability***	**Tingkat *Reusability***
Button	784 hasil di 149 file	7 varian × 3 ukuran	Sangat *Reusable*
Heade	414 hasil di 127 file	8 tipe layout	Sangat *Reusable*
Input	160 hasil di 37 file	3 varian × 2 mode	*Reusable*
Dialog	112 hasil di 13 file	4 varian	*Reusable*

Sumber: Data Diolah

Hasil analisis pada Tabel 6 menunjukkan bahwa komponen *Button* dan *Header* memiliki frekuensi penggunaan yang sangat tinggi serta kemampuan adaptasi yang luas di 6 modul bisnis yang aktif, sehingga meraih predikat sangat *reusable*. Sementara itu, komponen Input dan *ConfirmDialog* menempati predikat *reusable* karena meskipun frekuensi pemanggilannya lebih rendah, keduanya tetap menunjukkan tingkat adaptabilitas yang baik dan memberikan kontribusi signifikan terhadap pemangkasan duplikasi markup pada *platform* Bisnisyu.

Meskipun memberikan efisiensi ruang kode yang tinggi, batasan arsitektural dari implementasi aktual saat ini adalah ukuran berkas Button.tsx yang membengkak hingga melebihi 300 baris kode (*Fat Component*). Hal ini terjadi akibat penumpukan seluruh definisi varian gaya visual, objek literal, interaksi *hover*, *status loading*, dan animasi di satu tempat tunggal. Pendekatan penggabungan seluruh logika ini mempermudah pelacakan awal, namun berisiko menurunkan keterbacaan kode (*readability*) untuk pemeliharaan jangka

panjang, memperbesar potensi merge conflict pada repositori tim, serta mempersulit proses *code review* karena perubahan minor pada satu varian visual mewajibkan modifikasi pada keseluruhan berkas inti komponen.

KESIMPULAN

Penelitian ini berhasil menjawab tantangan duplikasi kode dan inkonsistensi antarmuka pada *platform* multi-bisnis "Bisnisyu" melalui perancangan dan implementasi arsitektur *reusable components* berbasis React dan TypeScript. Dengan mengisolasi empat komponen dasar terpusat yaitu *Button*, *Header*, *Input*, dan *ConfirmDialog* serta memanfaatkan integrasi *custom* hook *useTheme*(), aplikasi mampu mengadaptasi identitas visual dari enam modul bisnis berbeda secara dinamis pada tingkat *runtime*.

Evaluasi struktural menggunakan pendekatan *Hypothetical Baseline* membuktikan secara empiris bahwa sentralisasi ini memberikan dampak efisiensi yang sangat signifikan, dengan estimasi reduksi redundansi baris kode mencapai 84,9% pada komponen *Button* dan 79,2% pada komponen *Header*. Hasil ini menegaskan bahwa pendekatan *component-based development* yang terstruktur mampu meningkatkan aspek *maintainability* dan konsistensi pengalaman pengguna secara masif pada satu basis kode (*codebase*) yang kompleks.

Meskipun arsitektur yang diusulkan terbukti efektif memangkas duplikasi *markup*, penelitian ini memiliki batasan pada ukuran berkas Button.tsx yang membengkak melebihi 300 baris akibat penumpukan objek literal gaya visual (*Fat Component*). Sebagai saran untuk pengembangan dan penelitian selanjutnya, disarankan untuk melakukan refaktorisasi arsitektur komponen menggunakan pustaka *Design System Utility* seperti *Class Variance Authority* (CVA) guna memisahkan logika fungsional dari deklarasi varian gaya. Selain itu, manajemen tema dinamis multi-bisnis sebaiknya didelegasikan langsung ke dalam berkas konfigurasi Tailwind CSS (*tailwind.config.js*) memanfaatkan CSS *Variables*, sehingga kode komponen inti menjadi lebih ramping, memiliki keterbacaan yang tinggi, serta lebih mudah dirawat dalam kolaborasi tim jangka panjang.

DAFTAR RUJUKAN

[1] M. Ayyarrappan, "Creating Modular and Reusable Web Components with React," *International Journal Research of Leading Publication (IJLRP)*, vol. 2, no. 5, pp. 1–7, 2021. doi: 10.70528/IJLRP

[2] R. Firdaus, M. F. Duskarnaen, and B. P. Adhi, "Perancangan dan Implementasi Antarmuka Pengguna Sistem Informasi Berbasis Website SMK Negeri 6 Jakarta dengan Metode Prototype," *PINTER: Jurnal Pendidikan Teknik Informatika dan Komputer*, vol. 8, no. 2, pp. 66–74, 2024. doi: https://doi.org/10.21009/pinter.8.2.9

[3] D. L. Kaligis and R. R. Fatri, "Pengembangan Tampilan Antarmuka Aplikasi Survei Berbasis Web dengan Metode User Centered Design," *JUST IT: Jurnal Sistem Informasi, Teknologi Informatika dan Komputer*, vol. 10, no. 2, pp. 106–114, 2020. doi: https://doi.org/10.24853/justit.10.2.106-114

[4] E. A. AlOmar, T. Wang, V. Raut, M. W. Mkaouer, C. Newman, and A. Ouni, "Refactoring for Reuse: An Empirical Study," *arXiv preprint arXiv:2111.07002*, 2021. doi: https://doi.org/10.48550/arXiv.2111.07002

[5] A. Tornhill and M. Borg, "Code Red: The Business Impact of Code Quality: A Quantitative Study of 39 Proprietary Production Codebases," *arXiv preprint arXiv:2203.04374*, 2022. doi: https://doi.org/10.48550/arXiv.2203.04374

[6] N. S. Gill, "Reusability Issues in Component-Based Development," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 6, pp. 4–7, 2003. https://doi.org/10.1145/882240.882255

[7] S. Korra, S. V. Raju, and A. V. Babu, "Strategies for Designing and Building Reusable Software Components," *International Journal of Computer Science and Information Technologies*, vol. 4, no. 5, pp. 655–659, 2013.

[8] D. Hristov, O. Hummel, M. Huq, and W. Janjic, "Structuring Software Reusability Metrics for Component-Based Software Development," in *Proceedings of the Seventh International Conference on Software Engineering Advances (ICSEA 2012)*, 2012, pp. 421–428. doi: http://dx.doi.org/10.1109/ijcnn.1993.714224

[9] S. Thakral, S. Sagar, and Vinay, "Reusability in Component Based Software Development: A Review," *World Applied Sciences Journal*, vol. 31, no. 12, pp. 2068–2072, 2014, doi: 10.5829/idosi.wasj.2014.31.12.671.

[10] M. Radiyudin, A. F. Abidin, and M. A. Yaqin, "Keuntungan dan Kendala Pada Strategi Penggunaan Kembali dalam Proses Pengembangan Perangkat Lunak," *JUISI: Jurnal Informatika dan Sistem Informasi*, vol. 10, no. 2, pp. 101–108, 2024, doi: 10.37715/juisi.v10i2.4895.

[11] Frontend Web KM#6, "Reusable Component Menggunakan React dan Tailwind," *Digital Amoeba Documentation*, Mar. 8, 2024. [Online]. Available: halaman dokumentasi Digital Amoeba. [Accessed: 22-Jun-2026].