BAB III

PELAKSANAAN KERJA

3.1 Kedudukan dan Koordinasi

Bagian ini berisi keterangan/informasi mengenai posisi penulis dan alur koordinasi penulis dengan pembimbing lapangan pada saat pengerjaan suatu proyek/pengerjaan.

3.1.1 Kedudukan

Kedudukan di sini merupakan kedudukan penulis di perusahaan tempat kerja. Tergantung dari besar kecilnya perusahaan, di bagian ini dapat juga berisi bagan struktur organisasi divisi tempat penulis ditempatkan. Jika perusahaan tempat magang merupakan perusahaan kecil/perusahaan dengan struktur organisasi sederhana, maka bagan cukup diletakkan di poin 2.2.

3.1.2 Koordinasi

Bagian ini menjelaskan alur koordinasi pelaksanaan pekerjaan yang dilakukan penulis selama kegiatan magang di instansi terkait. Uraian tersebut memuat informasi mengenai hubungan kerja, proses komunikasi, serta mekanisme koordinasi yang terjalin dalam mendukung penyelesaian tugas dan tanggung jawab yang diberikan. Untuk memperjelas pemahaman terhadap alur koordinasi tersebut, penjelasan ini dilengkapi dengan bagan alur kerja yang disajikan pada Gambar 3.1.2.

Gambar 3.1.2 Bagan Alur Koordinasi

3.2 Tugas yang Dilakukan

Berisi tabel hal-hal yang penulis lakukan selama menjalankan program.

Tabel 3.2 Detail Pekerjaan yang Dilakukan

No.	Minggu	Proyek	Keterangan
1.	Agustus (Minggu 1-4)	Analisis & Perancangan UI/UX	Melakukan observasi kebutuhan sistem dan menyusun kerangka desain (wireframe). Fokus utamanya adalah memastikan tampilan website nantinya responsif dan mudah digunakan oleh warga.
2.	September (Minggu 1-2)	Implementasi Frontend Dasar	Mulai membangun struktur halaman menggunakan React.js berdasarkan desain yang sudah dibuat. Saya memastikan setiap komponen antarmuka sudah tampil dengan rapi di berbagai ukuran layar.
3.	September (Minggu 3-4)	Pemodelan & Basis Data	Merancang skema database dan menentukan relasi antar tabel. Tahap ini bertujuan agar penyimpanan data fitur terorganisir dan mudah diakses oleh sistem.
4.	Oktober (Minggu 1-2)	Pengembangan Backend & API	Membangun logika di sisi server dan menyusun *endpoints* API. Fokusnya adalah agar aliran data antara database dan tampilan depan website bisa berjalan dengan lancar.
5.	Oktober (Minggu 3-4)	Integrasi & Pengujian Sistem	Menghubungkan sisi frontend dan backend secara menyeluruh. Setelah itu, saya melakukan serangkaian tes untuk menemukan dan memperbaiki *bug* atau eror sebelum sistem dirilis.
6.	Minggu 6 (18–22 Ags)	Deployment & Publikasi	Menyiapkan lingkungan server di Diskominfo dan melakukan proses unggah sistem. Di sini saya memastikan konfigurasi server sudah tepat agar website dapat diakses publik dengan stabil.
7.	November (Minggu 3-4)	Finalisasi & Dokumentasi	Menyusun dokumen teknis yang berisi panduan penggunaan sistem dan struktur kode. Selain itu, saya merapikan laporan hasil kerja magang untuk diserahkan kepada pembimbing.

3.3 Uraian Pelaksanaan Kerja

Kegiatan magang di Diskominfo Kabupaten Tangerang saya laksanakan secara bertahap dengan mengikuti alur pengembangan sistem yang sistematis. Seluruh rangkaian pekerjaan difokuskan pada proyek pengembangan website layanan publik Pemerintah Kabupaten Tangerang. Di sini, saya bertanggung jawab untuk memastikan sistem dapat berfungsi

optimal, baik dari segi tampilan antarmuka maupun pengolahan data di sisi server.

Pada bulan **Agustus**, saya mengawali pekerjaan dengan tahap analisis kebutuhan dan perancangan UI/UX. Proses ini melibatkan identifikasi fitur-fitur krusial yang dibutuhkan oleh masyarakat serta administrator instansi. Saya menyusun kerangka desain (*wireframe*) dengan mengedepankan prinsip responsif agar tampilan website tetap konsisten dan nyaman saat diakses melalui berbagai perangkat, mulai dari komputer hingga ponsel pintar.

Memasuki bulan **September**, fokus pekerjaan bergeser pada implementasi teknis. Di paruh awal bulan, saya melakukan *slicing* desain ke dalam kode program menggunakan framework React.js untuk membangun komponen frontend yang dinamis. Setelah struktur antarmuka terbentuk, pada paruh akhir September, saya mulai merancang arsitektur basis data. Tahap ini meliputi pembuatan skema tabel dan pengaturan relasi data agar seluruh informasi dalam sistem informasi daerah ini dapat terorganisir dengan efisien.

Pada bulan **Oktober**, kegiatan difokuskan pada pembangunan sisi backend dan integrasi sistem. Saya menyusun logika server serta membangun *endpoints* API sebagai jembatan komunikasi data. Setelah API siap, saya melakukan integrasi menyeluruh antara bagian frontend dan backend. Di akhir Oktober, saya menjalankan serangkaian pengujian (*debugging*) secara intensif untuk mendeteksi celah eror, memperbaiki bug tampilan, serta memastikan validasi keamanan data bekerja sesuai spesifikasi yang diharapkan.

Sebagai tahap akhir pada bulan **November**, saya melakukan proses *deployment* sistem ke lingkungan server resmi milik Diskominfo Kabupaten Tangerang. Saya memastikan konfigurasi server telah stabil agar aplikasi dapat diakses publik tanpa kendala performa. Rangkaian magang ini ditutup dengan penyusunan dokumentasi teknis yang komprehensif, mencakup panduan penggunaan sistem dan laporan hasil pekerjaan sebagai bentuk

pertanggungjawaban profesional atas proyek pengembangan website tersebut.

3.3.1 Proses Pelaksanaan

Pelaksanaan kegiatan magang difokuskan pada keterlibatan penulis dalam pengembangan website Kabupaten Tangerang. Pada tahap awal, penulis langsung melaksanakan pengembangan sisi frontend dengan menggunakan React.js, yang mencakup pembuatan struktur halaman, komponen antarmuka, serta pengaturan navigasi website sesuai dengan kebutuhan informasi dan layanan publik.[2]

Setelah pengembangan frontend berjalan, penulis melanjutkan pekerjaan pada tahap integrasi sistem, yang meliputi penghubungan frontend dengan backend dan database melalui API. Pada tahap ini, penulis turut mengimplementasikan fitur pengelolaan konten, autentikasi pengguna, serta dashboard admin untuk mendukung pengelolaan data secara terpusat.

Tahap akhir pelaksanaan magang meliputi kegiatan pengujian, perbaikan sistem, dokumentasi, serta pendampingan proses deployment. Seluruh rangkaian pekerjaan tersebut dilakukan secara bertahap dan terkoordinasi dengan pembimbing lapangan guna memastikan sistem yang dikembangkan berjalan optimal, stabil, dan siap digunakan sebagai media layanan digital Kabupaten Tangerang.

3.3.1.1 Metode Pengembangan Sistem

Metode pengembangan sistem yang diterapkan dalam pelaksanaan kegiatan magang ini adalah **metode Waterfall**. Metode ini dipilih karena memiliki tahapan kerja yang jelas dan terstruktur, sehingga sesuai dengan karakteristik pengembangan sistem informasi di lingkungan pemerintahan yang menuntut perencanaan dan dokumentasi yang rapi. Setiap tahapan dalam metode Waterfall dilaksanakan secara berurutan dan saling berkaitan.

Tahap awal dimulai dengan **analisis kebutuhan**, yaitu mengidentifikasi permasalahan dan kebutuhan sistem melalui observasi serta koordinasi dengan pembimbing lapangan. Selanjutnya dilakukan **perancangan sistem** yang mencakup perancangan alur kerja, struktur sistem, dan tampilan antarmuka sebagai dasar pengembangan. Setelah perancangan selesai, sistem dikembangkan pada tahap **implementasi** sesuai dengan rancangan yang telah ditetapkan.

Tahap berikutnya adalah **pengujian**, yang bertujuan untuk memastikan seluruh fungsi sistem berjalan sesuai dengan kebutuhan. Apabila ditemukan kendala, dilakukan perbaikan sebelum sistem digunakan. Tahap akhir adalah **pemeliharaan**, yang mencakup perbaikan dan penyesuaian sistem guna mendukung keberlanjutan penggunaan sistem di lingkungan instansi.

3.3.1.2 Membuat Front-End Website

Pada tahap awal pelaksanaan proyek pengembangan website Kabupaten Tangerang, fokus pekerjaan diarahkan terlebih dahulu pada pengembangan sisi front-end. Penentuan prioritas ini didasarkan pada pertimbangan bahwa front-end merupakan komponen utama yang berperan sebagai penghubung antara sistem dengan pengguna, khususnya masyarakat sebagai penerima layanan informasi dan layanan publik berbasis digital. Melalui antarmuka yang baik, informasi dapat disampaikan secara jelas, terstruktur, dan mudah diakses, sehingga keberadaan website dapat menjalankan fungsinya secara optimal sebagai media pelayanan dan transparansi informasi pemerintah daerah[3].

Pengembangan front-end pada tahap awal juga dilakukan sebagai langkah strategis dalam membangun kerangka dasar sistem secara keseluruhan. Dengan merancang tampilan antarmuka, struktur

halaman, serta alur navigasi sejak awal, penulis dapat memperoleh gambaran yang lebih jelas mengenai kebutuhan fungsional sistem yang akan dikembangkan pada tahap selanjutnya. Hal ini membantu dalam mengidentifikasi jenis data yang diperlukan, hubungan antar fitur, serta pola interaksi pengguna yang harus didukung oleh sistem backend dan database. Dengan demikian, proses perancangan sistem menjadi lebih terarah dan meminimalkan terjadinya perubahan signifikan di tahap implementasi lanjutan.

Selain itu, fokus awal pada pengembangan front-end memungkinkan dilakukan evaluasi dini terhadap aspek pengalaman pengguna (user experience) dan kemudahan penggunaan (usability). Melalui pembuatan komponen antarmuka dan simulasi alur penggunaan website, penulis bersama pembimbing lapangan dapat menilai apakah tampilan dan navigasi yang dirancang telah sesuai dengan kebutuhan pengguna serta karakteristik layanan. Masukan yang diperoleh pada tahap ini kemudian dijadikan dasar untuk penyempurnaan desain sebelum sistem dikembangkan lebih jauh, sehingga risiko ketidaksesuaian antara kebutuhan pengguna dan implementasi teknis dapat diminimalkan.

Dari sisi teknis pengembangan, pembuatan front-end di awal proyek juga mendukung efisiensi proses kerja tim. Dengan tersedianya kerangka antarmuka yang jelas, proses integrasi antara front-end dan backend dapat dilakukan secara lebih sistematis melalui penyesuaian endpoint API dan struktur data yang telah terdefinisi. Pendekatan ini membantu memastikan bahwa pengembangan sistem berjalan secara bertahap, terstruktur, dan selaras dengan tujuan proyek secara keseluruhan. Oleh karena itu, pemilihan pengembangan front-end sebagai tahap awal proyek merupakan langkah yang tepat dan strategis dalam mendukung keberhasilan pengembangan website Kabupaten Tangerang.

1. Main: Fondasi Teknis Aplikasi

File *main.jsx* ini memegang peranan krusial sebagai titik inisiasi utama bagi seluruh aplikasi frontend Kabupaten Tangerang, bertindak lebih dari sekadar *entry point* standar, melainkan sebagai gerbang kualitas yang menyiapkan fondasi teknis yang solid sebelum konten disajikan kepada pengguna. Proses inisiasi yang dikendalikan oleh fungsi *init()* dimulai dengan memastikan ketersediaan DOM, kemudian menjalankan serangkaian fungsi persiapan teknis[4]. Langkah awalnya mencakup pemanggilan *setupSEO()* dan *setupPWA()* sebagaimana ditunjukkan pada Gambar 3.2.

Gambar 3.2 Code main.jsx

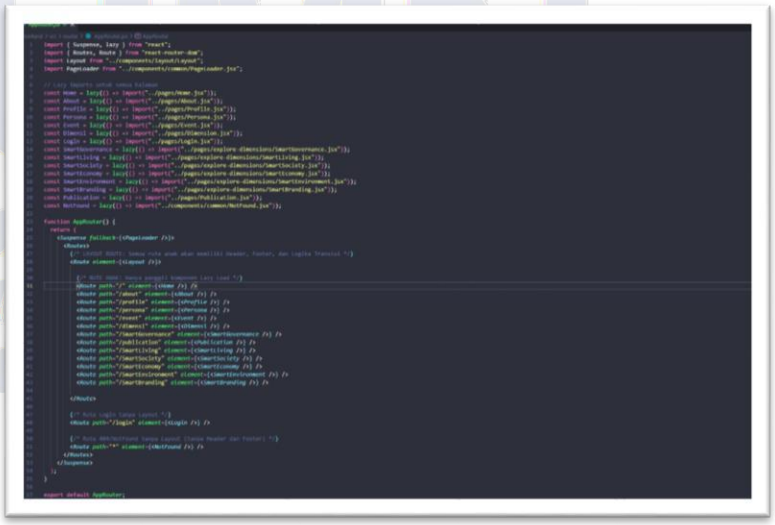
Fungsi SEO secara dinamis menyuntikkan *meta tags* penting untuk optimasi mesin pencari, termasuk deskripsi, *keywords*, Open Graph untuk media sosial, dan data terstruktur JSON-LD, memastikan website terindeks dengan baik dan terlihat menarik saat dibagikan. Sementara itu, setupPWA()

meletakkan dasar untuk fungsionalitas Progressive Web App, memungkinkan website diinstal di perangkat seluler dengan pengalaman yang menyerupai aplikasi *native*. Setelah persiapan fondasi selesai, fungsi inti **initializeApp()** dijalankan, yang bertugas menciptakan React Root menggunakan API modern createRoot(), menanamkan komponen akar **<App />** di dalam *wrapper* <StrictMode> untuk memastikan praktik terbaik selama pengembangan. Keunggulan signifikan dari kode ini adalah fokusnya pada performa dan stabilitas, dibuktikan dengan implementasi **Global Error Handling** untuk menangkap kesalahan *unhandled promise rejection* dan menyediakan *fallback* pesan kesalahan yang ramah pengguna jika proses *rendering* React gagal secara kritis. Selain itu, terdapat fitur **Performance Monitoring** menggunakan PerformanceObserver bawaan browser untuk secara aktif melacak metrik Core Web Vitals (LCP, FID, CLS), yang menunjukkan komitmen pada kecepatan loading dan pengalaman pengguna terbaik. Terakhir, untuk menjamin transisi visual yang mulus, kode ini memiliki mekanisme untuk menyembunyikan elemen *preloader* secara bertahap (*fade out*) hanya setelah komponen <App /> benar-benar selesai di-render, menghilangkan *flicker* dan memastikan pengguna melihat konten yang lengkap.

2. AppRouter: Fondasi Perutean Berkinerja Tinggi dan Struktur Aplikasi

File *AppRouter.jsx* merepresentasikan inti fungsional dari arsitektur frontend, beralih dari fase inisiasi lingkungan yang ditangani oleh *main.jsx* ke fase penentuan alur navigasi dan penyajian konten. Komponen ini merupakan manifestasi dari penerapan strategi pemisahan kode (*code splitting*) yang agresif, sebagai salah satu *best practice* dalam pengembangan

aplikasi skala besar untuk mengoptimalkan waktu muat dan kinerja aplikasi. Melalui pemanfaatan fungsi *lazy* dan *Suspense* dari React, seluruh komponen halaman yang didefinisikan dalam folder *src/pages* diimpor secara asinkron dan hanya dimuat ketika rute tersebut benar-benar diakses oleh pengguna. Pendekatan ini secara signifikan mengurangi ukuran *initial bundle* JavaScript yang harus diunduh oleh browser saat pertama kali aplikasi dijalankan, yang menjadi faktor penting dalam pencapaian skor *Core Web Vitals* yang optimal, khususnya pada metrik *First Contentful Paint* dan *Time to Interactive[5]*. Selama proses pengunduhan dan pemrosesan komponen yang di-*lazy load* tersebut berlangsung, pembungkus <Suspense> secara otomatis menampilkan komponen *fallback* berupa <PageLoader />, sehingga memberikan pengalaman transisi yang profesional serta mencegah terjadinya tampilan layar kosong (*blank screen*) kepada pengguna, sebagaimana ditunjukkan pada Gambar 3.3.



Gambar 3.3 Code AppRouter.jsx

Struktur navigasi itu sendiri didasarkan pada hierarki rute yang cerdas menggunakan React Router DOM. Inti dari

hierarki ini adalah rute utama yang menggunakan element {<Layout />}. Komponen **Layout** ini bertindak sebagai **kerangka antarmuka universal (universal interface framework)** atau *shell* yang membungkus elemen struktural yang persisten, seperti *header* navigasi utama dan *footer* situs. Dengan menetapkan semua rute konten inti, termasuk beranda (/), halaman informasi (/about, /dimensi), halaman khusus (/publication, /profile), dan yang terpenting, keenam pilar dimensi kota pintar (/SmartGovernance, /SmartEconomy, dst.), sebagai rute anak, aplikasi menjamin bahwa semua konten utama secara otomatis mewarisi tata letak visual yang konsisten tanpa perlu *re-render* elemen global. Desain *nested routing* ini meningkatkan efisiensi pemeliharaan dan memastikan koherensi visual yang merupakan karakteristik penting dari portal pemerintahan digital yang kredibel.

Pemisahan tanggung jawab juga tercermin dalam pengecualian rute dari hierarki utama; jalur **/login** didefinisikan secara independen, tanpa *wrapper Layout*. Pilihan arsitektural ini disengaja, dirancang untuk meminimalkan gangguan visual pada halaman autentikasi dan memfokuskan pengguna sepenuhnya pada proses masuk. Akhirnya, untuk mengatasi potensi kesalahan pengguna atau perubahan URL yang tidak valid, AppRouter dilengkapi dengan mekanisme penanganan kesalahan navigasi yang elegan. Penggunaan rute *wildcard* **<Route path="*" />** pada urutan paling akhir memastikan bahwa setiap permintaan URL yang gagal dicocokkan dengan rute yang valid akan ditangkap dan dialihkan untuk menampilkan komponen **<NotFound />**. Ini adalah langkah defensif penting yang meningkatkan keandalan aplikasi dengan menggantikan *error* peramban yang tidak informatif dengan pesan kesalahan

aplikasi yang kontekstual. Dengan mengombinasikan strategi *lazy loading* yang agresif, *nested routing* yang konsisten, dan penanganan kesalahan yang komprehensif, AppRouter.jsx berfungsi sebagai pusat orkestrasi yang mendefinisikan *flow* pengguna yang cepat, stabil, dan terstruktur di portal.

3. Layout: Orkestrasi Struktur dan Pengalaman Transisi Halaman

Komponen *Layout.jsx* bertindak sebagai kerangka utama (*master shell*) bagi sebagian besar aplikasi, mewujudkan konsistensi antarmuka yang telah didefinisikan melalui mekanisme *nested routing* pada *AppRouter.jsx*. Peran utamanya adalah mengorkestrasi elemen statis dan dinamis dalam aplikasi, sekaligus mengelola pengalaman pengguna, khususnya terkait transisi halaman dan penerapan konteks CSS[6]. File ini mengimpor dan menyajikan komponen struktural yang bersifat persisten pada setiap rute turunan, yaitu <Header /> dan <Footer />, yang dibingkai oleh elemen <main>. Pada elemen <main> tersebut, konten spesifik dari setiap halaman dimuat melalui komponen <Outlet />, yaitu *placeholder* fungsional dari React Router DOM yang secara otomatis menyuntikkan komponen halaman yang sedang aktif (misalnya *Home.jsx* atau *SmartGovernance.jsx*), sebagaimana ditunjukkan pada Gambar 3.4.

Gambar 3.4 Code Layout.jsx

Arsitektur *Layout* ini menunjukkan perhatian yang mendalam terhadap kualitas pengalaman pengguna (UX) melalui implementasi **transisi halaman kustom**. Dengan memanfaatkan *hook* useLocation dari React Router DOM, *Layout* mampu mendeteksi perubahan URL. Setiap kali location berubah, serangkaian *state* (pageTransition dan isLoading) diaktifkan sebentar selama 150 milidetik. Periode singkat ini digunakan untuk menerapkan kelas CSS (page-transitioning dan loading) pada elemen *wrapper* dan konten utama, yang memungkinkan CSS melakukan efek seperti *fade-out* singkat atau meredupkan konten, memberikan *feedback* visual yang halus kepada pengguna bahwa konten sedang diperbarui.

Selain manajemen transisi, komponen ini juga berperan sebagai **kontroler konteks CSS** dan **manajer aksesibilitas**. Dengan menggunakan useEffect yang terpicu oleh perubahan location, komponen ini secara dinamis menambahkan kelas CSS spesifik pada elemen <body> HTML. Skema penamaan

kelas ini (page-home, page-about, page-dimensi) memungkinkan *developer* untuk menargetkan *styling* CSS yang unik untuk setiap halaman spesifik tanpa perlu mengubah komponen halaman itu sendiri, sebuah praktik yang meningkatkan modularitas dan pemisahan kekhawatiran (*separation of concerns*). Fungsi yang sama juga secara konsisten menjamin **aksesibilitas** dan *user flow* yang logis dengan memanggil window.scrollTo(0, 0) setiap kali navigasi terjadi, memastikan pengguna selalu mulai membaca dari puncak halaman baru. Selain itu, *Layout* ini memperkuat ketahanan aplikasi dengan membungkus seluruh kontennya dalam **<ErrorBoundary />**, memastikan bahwa *error* yang terjadi di komponen anak tidak menyebabkan seluruh aplikasi *crash*, melainkan hanya menampilkan *fallback* UI yang aman. Komponen pembantu lainnya, seperti **<BackToTopButton />**, juga secara otomatis disajikan di dalam *Layout*, menyediakan fungsionalitas navigasi yang konsisten di seluruh aplikasi. Secara keseluruhan, Layout.jsx adalah pusat orkestrasi yang tidak hanya memberikan struktur visual, tetapi juga mengelola *flow* transisi, kontekstualisasi *styling*, dan lapisan *error handling* global.

4. Home: Komposisi Konten, Interaktivitas, dan Aksesibilitas

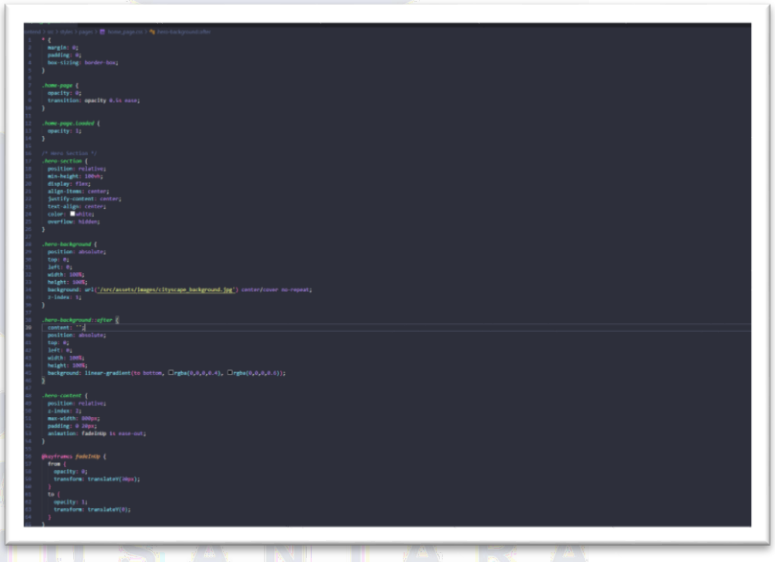
Komponen *Home.jsx* berfungsi sebagai halaman pendaratan (*landing page*) utama dan pusat penyajian informasi ringkas bagi portal, yang merepresentasikan integrasi antara logika fungsional, manajemen *state*, serta presentasi visual yang telah dirancang pada tingkat arsitektur aplikasi yang lebih tinggi. Secara struktural, komponen ini merupakan perakitan tingkat atas (*top-level assembly*) dari berbagai bagian antarmuka pengguna (*user interface*), namun

keunggulannya terletak pada penerapan mekanisme interaktif yang canggih serta perhatian terhadap aspek pengalaman pengguna (*user experience*), sebagaimana ditunjukkan pada Gambar 3.5.

Gambar 3.5 Code Home.jsx

Inti dari komponen ini adalah objek statis **dimensiData**, yang bertindak sebagai *single source of truth* untuk informasi keenam pilar. Penggunaan data statis yang terstruktur ini secara signifikan meningkatkan *maintainability* dan *readability*, memisahkan data deskriptif dari logika presentasi. Logika interaktif dikelola melalui *React hooks*, di mana *state* seperti selectedDimensi dan isModalOpen memfasilitasi alur *drill-down* informasi. Mekanisme modal, yang diaktifkan melalui fungsi showModal, tidak hanya menampilkan data dimensi secara rinci tetapi juga secara cermat memanipulasi DOM; dengan mengatur **document.body.style.overflow = 'hidden'** saat modal terbuka, aplikasi secara efektif mencegah *scrolling* latar belakang yang dapat mengganggu, menjamin fokus pengguna sepenuhnya pada pop-up[7].

Aspek desain yang menonjol adalah perhatiannya pada **aksesibilitas (WAI-ARIA)** dan *user flow* yang logis. Setiap kartu di bagian *Features Section* dikonfigurasi dengan atribut tabIndex="0" dan role="button", serta penanganan *event* onKeyDown, memungkinkan pengguna untuk menavigasi dan mengaktifkan kartu menggunakan keyboard, sebuah praktik esensial untuk memenuhi standar aksesibilitas web. Selain itu, terdapat *listener* useEffect yang secara cerdas menangani penutupan modal; setiap kali tombol **Escape** ditekan, fungsi closeModal akan dipanggil, memberikan jalur keluar yang cepat dan intuitif bagi pengguna. Implementasi ini dilengkapi dengan *hook* useNavigate, yang memfasilitasi navigasi programatik dari modal ke halaman dimensi spesifik, menjaga alur transisi halaman yang mulus sesuai janji yang dibuat oleh **AppRouter.jsx**.



Gambar 3.6 Code home_page.css

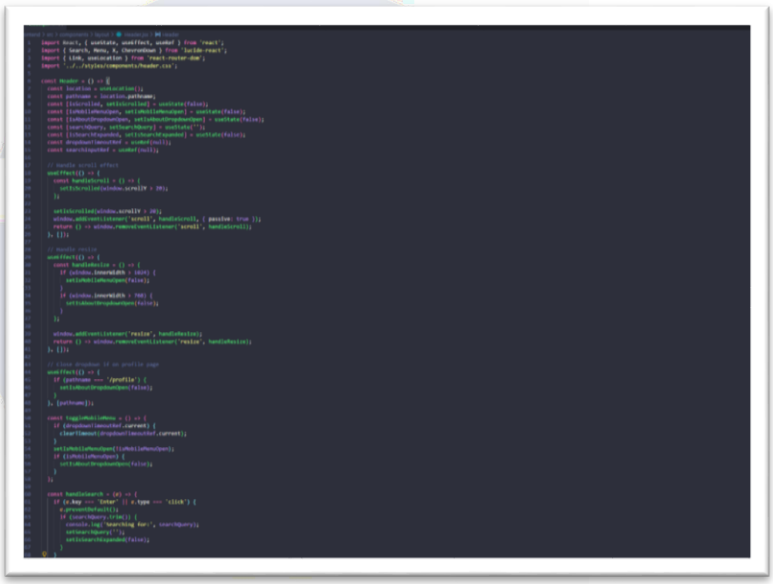
Secara visual, komponen ini dibagi menjadi beberapa bagian modular—termasuk **Hero Section** dengan animasi CSS dinamis, **Features Section** yang menyajikan dimensi dalam tata letak Grid yang responsif, dan **Stats Section** yang

menyajikan data kuantitatif kota. Kohesi antara logika Home.jsx dan berkas *styling* **home_page.css** memastikan konsistensi dan adaptabilitas. *Styling* tersebut diperkaya dengan *media queries* ekstensif yang menjamin bahwa tata letak halaman beranda beradaptasi secara optimal pada perangkat seluler, dari tampilan Grid multi-kolom hingga tata letak tumpukan satu kolom, menunjukkan komitmen proyek terhadap desain responsif sejati. Dengan demikian, Home.jsx adalah puncak dari arsitektur modular, menyajikan konten interaktif dengan mengedepankan efisiensi, aksesibilitas, dan kualitas visual yang tinggi.

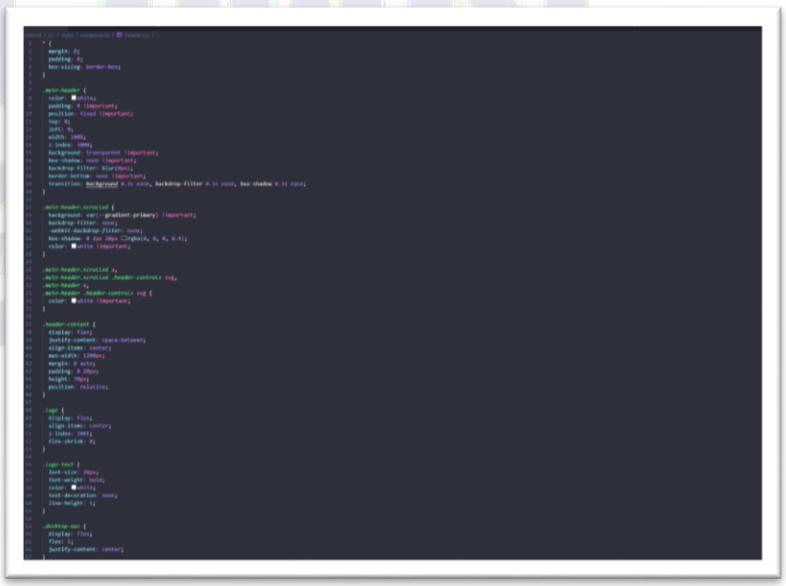
5. Header: Menjaga Integritas Navigasi dan Adaptasi Multidimensi

Komponen *Header.jsx* memainkan peran kritis sebagai elemen *scaffolding* yang bersifat persisten, memastikan aksesibilitas navigasi serta identitas merek terjaga secara berkelanjutan sepanjang *user journey*. Keunggulan desain komponen ini terletak pada perpaduan manajemen *state* yang relatif kompleks dengan penerapan teknik frontend modern untuk mengatasi tantangan implementasi *fixed header* [8]. Integritas visual dan fungsional header dipertahankan melalui pemanfaatan *useEffect* yang berlapis guna mengelola tiga kondisi adaptif utama. Pertama, efek *scroll* diimplementasikan melalui *event listener* non-blokir dengan opsi *passive*, yang secara efisien memperbarui *state* isScrolled. Perubahan *state* ini memicu transisi visual berbasis CSS, di mana header beralih dari latar belakang transparan ke gradien solid, sebagai strategi penting untuk menjaga kontras dan keterbacaan teks navigasi ketika header melintasi konten dengan variasi warna. Kedua, adaptasi terhadap perubahan ukuran *viewport* diatur melalui

kombinasi *event listener resize* dan *useEffect*, sehingga menu versi mobile secara otomatis dinonaktifkan ketika lebar layar kembali ke dimensi desktop. Pendekatan ini mencegah terjadinya inkonsistensi *state* menu yang kerap muncul pada desain responsif, sebagaimana ditunjukkan pada Gambar 3.7.



Gambar 3.7 Code Header.jsx



Gambar 3.8 Code header.css

Ketiga, komponen ini mendemonstrasikan penanganan interaktivitas yang disesuaikan berdasarkan platform. Untuk navigasi desktop, *dropdown* "Tentang" menggunakan *event* **onMouseEnter** dan **onMouseLeave**, diimbangi oleh mekanisme *debounce* (dropdownTimeoutRef) yang cerdas untuk mencegah penutupan *dropdown* prematur akibat pergerakan kursor yang cepat. Sementara itu, pada mode seluler, *dropdown* dikonfigurasi secara eksplisit untuk menggunakan **pemicu klik** (toggleAboutDropdown), sebuah pendekatan yang lebih andal dan sesuai dengan interaksi berbasis sentuhan. Lebih jauh, Header.jsx mengintegrasikan fitur pencarian yang halus; *input* pencarian awalnya diminimalkan dan baru diekspos melalui transisi CSS yang elegan ketika ikon *Search* diklik. Logika yang terkait dengan **useRef** dan setTimeout secara programatis segera memindahkan fokus *input*, menghilangkan langkah interaksi tambahan bagi pengguna, dan *event handler* onBlur yang dikombinasikan dengan validasi searchQuery.trim() menjamin bahwa elemen pencarian akan kembali ke kondisi minimalnya jika pengguna tidak memasukkan *query*, sehingga menjaga estetika minimalis *header*. Secara keseluruhan, **Header.jsx** adalah implementasi yang matang dan berorientasi pada ketahanan, yang menyatukan manajemen *state* yang kompleks, pengoptimalan performa *scroll*, dan pertimbangan aksesibilitas (melalui penanganan *event* keyboard) untuk menghasilkan komponen navigasi yang konsisten, adaptif, dan memenuhi tuntutan aplikasi web modern dengan integritas fungsional yang tinggi.

6. Footer: Integritas Informatif dan Desain Imersif

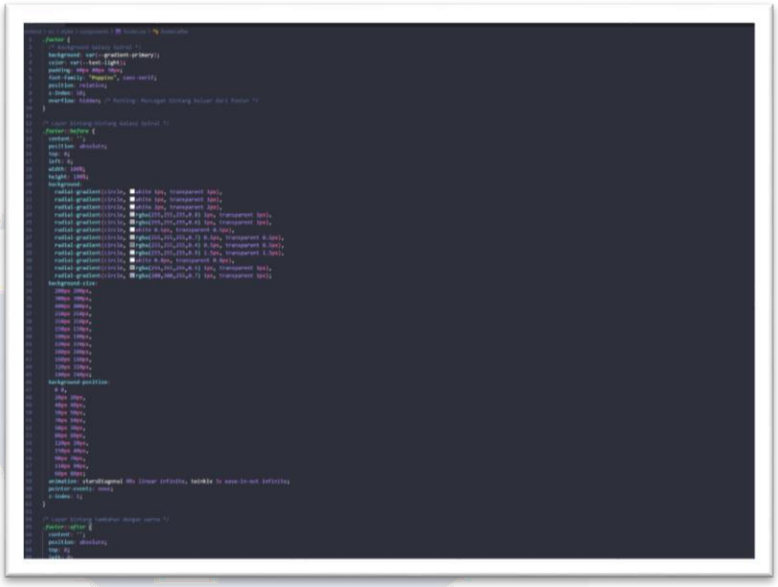
Komponen *Footer.jsx* berfungsi sebagai penjamin integritas informatif dan kredibilitas digital dari portal,

sekaligus menyempurnakan struktur kerangka antarmuka yang telah dibangun oleh *Layout.jsx*. Secara fungsional, komponen ini dikonstruksi secara modular untuk mengonsolidasikan empat kategori informasi utama, yaitu identitas korporat, navigasi sekunder, informasi kontak, dan konektivitas media sosial. Bagian *About* memperkuat citra merek melalui penyajian logo serta deskripsi naratif singkat mengenai misi program, sedangkan bagian *Links* secara strategis menyediakan jalur navigasi *deep linking* menuju segmen konten utama seperti *Dimensi* dan *Publikasi*. Navigasi ini memanfaatkan komponen *Link* dari React Router guna mempertahankan transisi halaman yang cepat tanpa memicu pemuatan ulang dari sisi server. Selanjutnya, bagian *Contact* meningkatkan transparansi operasional dengan menampilkan informasi alamat fisik, surat elektronik, dan nomor telepon, yang diperkaya dengan penggunaan ikon dari pustaka *Lucide-React* untuk meningkatkan keterbacaan visual dan aksesibilitas informasi. Dengan demikian, meskipun tidak bergantung pada manajemen *state* atau *hook* yang kompleks, komponen *Footer.jsx* tetap berperan sebagai pilar penting dalam pemenuhan kebutuhan informasi dan navigasi non-utama (*off-page navigation*), sebagaimana ditunjukkan pada Gambar 3.9.

Gambar 3.9 Code Footer.jsx

Aspek teknik yang paling menonjol pada *Footer* ini terletak pada *styling* CSS yang imersif dan berorientasi pada estetika tematik "Galaxy Spiral." Desain ini dicapai melalui penggunaan pseudo-elemen **::before** dan **::after** yang menumpuk, di mana masing-masing lapisan dikonfigurasi dengan serangkaian fungsi **radial-gradient()** dengan parameter ukuran dan opasitas yang bervariasi. Teknik ini secara efektif menghasilkan latar belakang bintang-bintang yang kaya tekstur dan bernuansa kedalaman. Untuk melampaui elemen statis, *keyframe* CSS **@keyframes starsDiagonal** dan **@keyframes twinkle** diaplikasikan untuk menyuntikkan dimensi dinamis; yang pertama memberikan ilusi pergerakan latar belakang yang sinematik, sedangkan yang kedua memodulasi opasitas bintang-bintang, menciptakan efek berkelap-kelip yang halus, sehingga memperkuat konsistensi visual dengan tema teknologi dan masa depan. Lebih jauh lagi, elemen interaktif seperti judul kolom ditingkatkan dengan efek garis bawah gradien animasi, sementara tautan navigasi sekunder memiliki efek *hover* yang

menggabungkan perubahan warna, *text shadow* untuk efek *glow*, dan garis bawah yang diperluas, secara kolektif meningkatkan keterlibatan pengguna melalui umpan balik visual yang canggih [9].



Gambar 3.10 Code footer.css

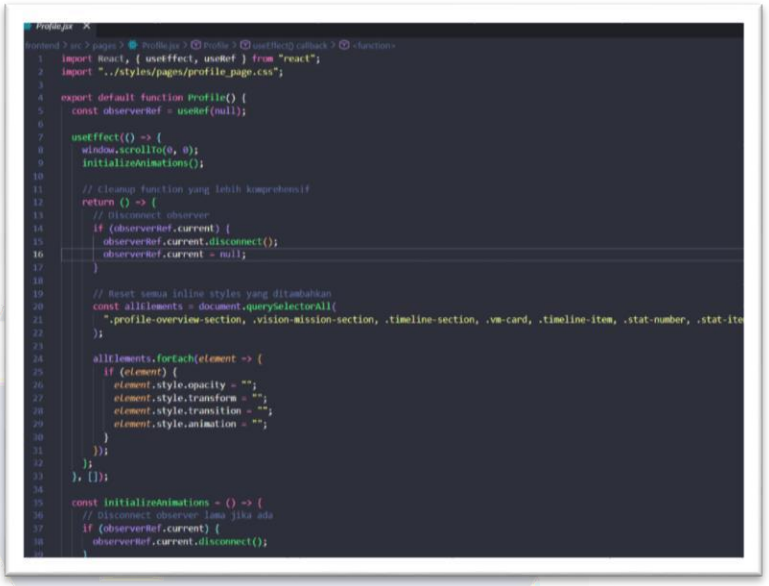
Struktur *Footer* ini mengadopsi tata letak **CSS Grid** yang kokoh untuk mengatur empat bagian fungsional menjadi format multi-kolom yang optimal pada tampilan desktop. Desain ini menunjukkan komitmen yang kuat terhadap **responsivitas**, yang dimanifestasikan melalui *media queries* ekstensif yang memandu adaptasi tata letak secara mulus. Pada titik henti (*breakpoint*) tertentu, tata letak empat kolom bertransisi menjadi dua kolom, dan akhirnya menjadi format tumpukan satu kolom pada perangkat seluler. Transisi adaptif ini memastikan bahwa konten informasi tetap terstruktur dan mudah diakses, terlepas dari dimensi *viewport* pengguna. Selain aspek tata letak, komponen ini secara ketat mematuhi praktik **kepatuhan digital** dengan menyematkan atribut **target="_blank", rel="noopener noreferrer", dan *aria-label*** pada semua tautan eksternal media sosial. Praktik ini

tidak hanya memitigasi risiko keamanan (*tabnapping*) tetapi juga meningkatkan aksesibilitas bagi pengguna alat bantu pembaca layar, menegaskan bahwa Footer.jsx tidak hanya berfungsi sebagai penutup visual, tetapi juga sebagai lapisan penting dalam menjaga fungsionalitas dan keamanan arsitektur *frontend* secara menyeluruh.

7. Profile: Presentasi Data Dinamis

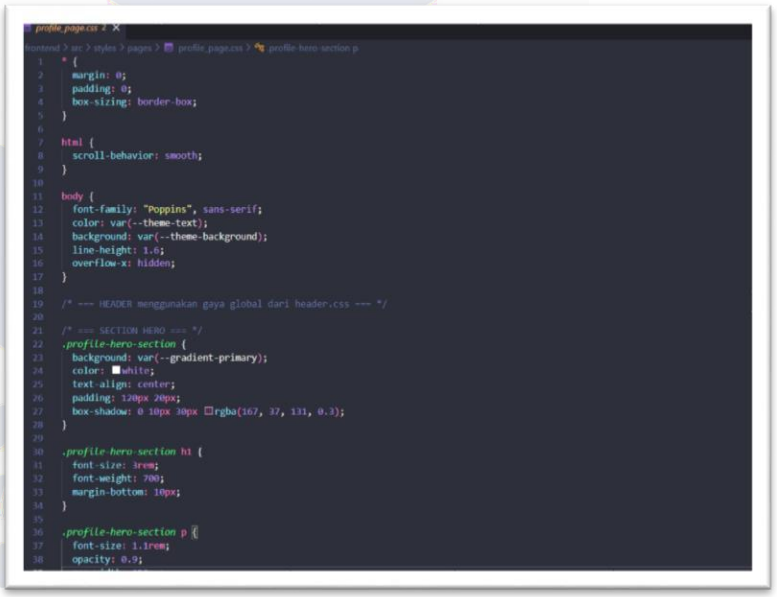
Komponen *Profile.jsx* berfungsi sebagai halaman konten utama yang didedikasikan untuk memvisualisasikan Visi, Misi, serta linimasa strategis (*roadmap*), dengan nilai teknis yang menonjol pada penerapan manajemen *state* yang berorientasi pada pengalaman pengguna (*user experience*) dan kinerja aplikasi. Inti dari fungsionalitas visual komponen ini terletak pada implementasi *IntersectionObserver* yang dikelola melalui *useRef* dan diinisialisasi di dalam *useEffect*, sebagai paradigma desain yang lebih efisien untuk animasi berbasis *scroll-triggered*. *Observer* ini secara selektif memantau elemen-elemen kunci pada DOM, seperti .profile-overview-section dan .vm-card. Ketika elemen-elemen tersebut memasuki area pandang pengguna (*viewport*), *inline styles* yang sebelumnya diterapkan akan dimodifikasi secara programatis sehingga memicu transisi CSS yang halus berupa efek *fade-in* dan *slide-up [10]*. Pendekatan ini secara signifikan mengurangi beban pada *main thread* jika dibandingkan dengan penggunaan *event listener scroll* konvensional. Selain itu, *hook useEffect* juga menyertakan fungsi *cleanup* yang komprehensif, tidak hanya untuk memutus koneksi *observer* tetapi juga menghapus seluruh *inline styles* yang ditambahkan, yang merupakan praktik penting dalam arsitektur React guna mencegah terjadinya

memory leak dan konflik gaya saat komponen di-*unmount*, sebagaimana ditunjukkan pada Gambar 3.11.



```jsx
import React, { useEffect, useRef } from "react";
import "../styles/pages/profile_page.css";

export default function Profile() {
  const observerRef = useRef(null);

  useEffect(() => {
    window.scrollTo(0, 0);
    initializeAnimations();

    // Cleanup function yang lebih komprehensif
    return () => {
      // Disconnect observer
      if (observerRef.current) {
        observerRef.current.disconnect();
        observerRef.current = null;
      }

      // Reset semua inline styles yang ditambahkan
      const allElements = document.querySelectorAll(
        ".profile-overview-section, .vision-mission-section, .timeline-section, .vm-card, .timeline-item, .stat-number, .stat-ite
      );

      allElements.forEach(element => {
        if (element) {
          element.style.opacity = "";
          element.style.transform = "";
          element.style.transition = "";
          element.style.animation = "";
        }
      });
    };
  }, []);

  const initializeAnimations = () => {
    // Disconnect observer lama jika ada
    if (observerRef.current) {
      observerRef.current.disconnect();
    }
```

Gambar 3.11 Code Profile.jsx



```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

html {
  scroll-behavior: smooth;
}

body {
  font-family: "Poppins", sans-serif;
  color: var(--theme-text);
  background: var(--theme-background);
  line-height: 1.6;
  overflow-x: hidden;
}

/* === HEADER menggunakan gaya global dari header.css === */

/* === SECTION HERO === */
.profile-hero-section {
  background: var(--gradient-primary);
  color: white;
  text-align: center;
  padding: 120px 20px;
  box-shadow: 0 10px 30px rgba(167, 37, 131, 0.3);
}

.profile-hero-section h1 {
  font-size: 3rem;
  font-weight: 700;
  margin-bottom: 10px;
}

.profile-hero-section p {
  font-size: 1.1rem;
  opacity: 0.9;
```

Gambar 3.12 Code profile_page.css

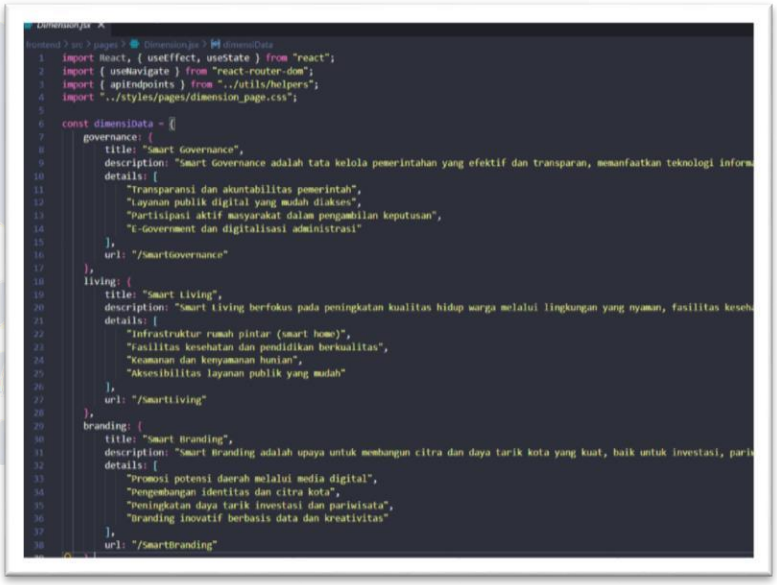
Fungsionalitas dinamis diperkuat oleh mekanisme animasi penghitung statistik yang canggih. Fungsi **animateStatNumber** dan **animateNumber** bekerja secara sinergis, menggunakan **requestAnimationFrame** untuk

menjalankan animasi *count-up* yang mulus pada elemen .stat-number selama periode waktu tertentu. Logika ini mampu mengurai dan menormalisasi nilai statistik yang mengandung sufiks (seperti "M" atau "K"), dan menggunakan fungsi *easing* non-linear untuk menghasilkan pergerakan angka yang terasa lebih alami. Aspek interaksi dan aksesibilitas juga dipertimbangkan secara matang; *card* Visi dan Misi (.vm-card) dikonfigurasi dengan atribut **tabIndex="0"** dan **role="button"**, memungkinkannya diakses oleh pengguna yang mengandalkan navigasi keyboard, dengan *event handler* onKeyDown yang menanggapi tombol *Enter* dan *Space* layaknya klik, sehingga meningkatkan kepatuhan terhadap standar WAI-ARIA. Secara estetika, *styling* CSS menggunakan teknik gradasi teks (-webkit-text-fill-color: transparent) pada judul dan angka statistik, sementara tata letak konten diatur secara hierarkis dan responsif menggunakan **CSS Grid**, menjamin transisi tata letak yang optimal dan konsisten di berbagai perangkat, mulai dari tampilan multi-kolom hingga tumpukan vertikal pada perangkat seluler.

8. Dimensi: Hibridisasi Sumber Data

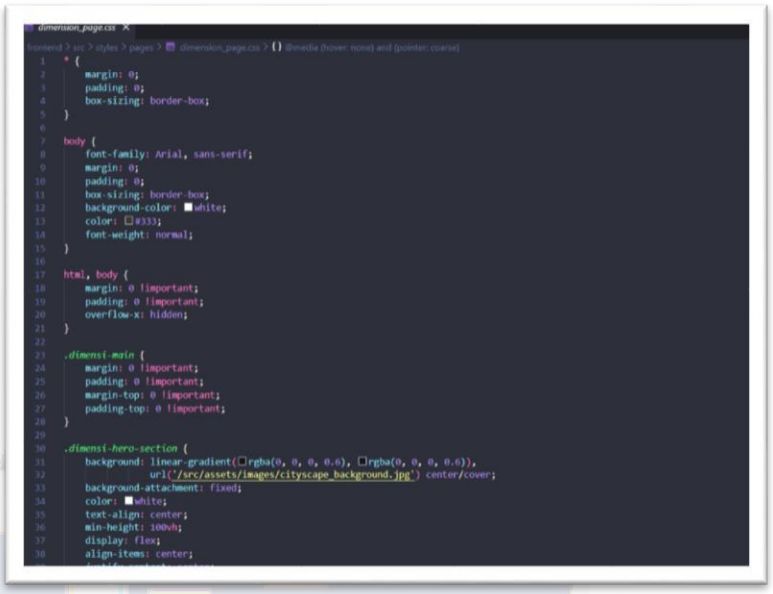
Komponen *Dimensi.jsx* berfungsi sebagai direktori sentral yang memvisualisasikan enam pilar utama, yaitu *Smart Governance, Smart Living, Smart Branding, Smart Society, Smart Economy,* dan *Smart Environment*, dari proyek yang dikembangkan. Fungsi ini dicapai melalui penerapan strategi hibridisasi data yang adaptif. Komponen mengombinasikan data statis yang didefinisikan secara lokal dalam objek dimensiData, yang berperan sebagai *placeholder* dan penjamin keandalan tampilan sisi frontend, dengan data dinamis yang diambil secara asinkron dari *endpoint* API

(apiEndpoints.dimensi.getAll()) melalui *hook useEffect*. Pendekatan hibrida ini memastikan bahwa meskipun terjadi kegagalan pengambilan data (*fetching error*) atau keterlambatan proses pemuatan, sebagian dimensi utama tetap dapat dirender sehingga menjaga ketahanan parsial (*partial resilience*) pada antarmuka pengguna. *State* lokal (apiDimensi, loading, dan error) digunakan untuk memantau status operasi jaringan secara ketat, serta memberikan umpan balik yang tepat kepada pengguna melalui indikator *loading* dan pesan kesalahan, yang merupakan praktik baku dalam pengembangan aplikasi React berbasis data. Selain itu, *hook useNavigate* dari React Router diintegrasikan ke dalam komponen untuk mendukung navigasi terprogram menuju halaman detail dimensi tertentu setelah interaksi pengguna dengan *modal*, sehingga menyempurnakan alur pengalaman pengguna dari ringkasan informasi ke eksplorasi konten yang lebih mendalam, sebagaimana ditunjukkan pada Gambar 3.13.



```
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { apiEndpoints } from "../utils/helpers";
import "../styles/pages/dimension_page.css";

const dimensiData = {
    governance: {
        title: "Smart Governance",
        description: "Smart Governance adalah tata kelola pemerintahan yang efektif dan transparan, memanfaatkan teknologi informa
        details: [
            "Transparansi dan akuntabilitas pemerintah",
            "Layanan publik digital yang mudah diakses",
            "Partisipasi aktif masyarakat dalam pengambilan keputusan",
            "E-Government dan digitalisasi administrasi"
        ],
        url: "/SmartGovernance"
    },
    living: {
        title: "Smart Living",
        description: "Smart Living berfokus pada peningkatan kualitas hidup warga melalui lingkungan yang nyaman, fasilitas kesehat
        details: [
            "Infrastruktur rumah pintar (smart home)",
            "Fasilitas kesehatan dan pendidikan berkualitas",
            "Keamanan dan kenyamanan hunian",
            "Aksesibilitas layanan publik yang mudah"
        ],
        url: "/SmartLiving"
    },
    branding: {
        title: "Smart Branding",
        description: "Smart Branding adalah upaya untuk membangun citra dan daya tarik kota yang kuat, baik untuk investasi, pari
        details: [
            "Promosi potensi daerah melalui media digital",
            "Pengembangan identitas dan citra kota",
            "Peningkatan daya tarik investasi dan pariwisata",
            "Branding inovatif berbasis data dan kreativitas"
        ],
        url: "/SmartBranding"
```

Gambar 3.13 Code Dimension.jsx



```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    background-color: white;
    color: #333;
    font-weight: normal;
}

html, body {
    margin: 0 !important;
    padding: 0 !important;
    overflow-x: hidden;
}

.dimensi-main {
    margin: 0 !important;
    padding: 0 !important;
    margin-top: 0 !important;
    padding-top: 0 !important;
}

.dimensi-hero-section {
    background: linear-gradient(rgba(0, 0, 0, 0.6), rgba(0, 0, 0, 0.6)),
                url('/src/assets/images/cityscape_background.jpg') center/cover;
    background-attachment: fixed;
    color: white;
    text-align: center;
    min-height: 100vh;
    display: flex;
    align-items: center;
```

Gambar 3.14 Code dimension_page.css

Fungsionalitas inti dari Dimensi.jsx didasarkan pada implementasi **pola desain Modal Interaktif** yang mengutamakan aksesibilitas. *State* **isModalOpen** dan **selectedDimensi** secara eksklusif mengontrol visibilitas dan konten modal, dengan fungsi **showModal** yang bertanggung jawab untuk menampilkan detail dimensi yang dipilih dan mengunci *scroll* pada *body* dokumen (document.body.style.overflow = 'hidden'), sebuah praktik yang penting untuk mencegah *double scrolling* saat modal aktif. Mekanisme penutupan modal telah dirancang agar fleksibel dan inklusif: tidak hanya dapat ditutup melalui tombol *close* dan klik di luar area konten (**handleModalClick**), tetapi juga secara elegan menanggapi tombol **Escape** pada *keyboard* melalui *event listener* global yang dikelola dalam *hook* **useEffect** terpisah. Komponen dimensi-card sendiri juga mematuhi prinsip aksesibilitas WAI-ARIA dengan menyertakan atribut **tabIndex="0"** dan **role="button"**, didukung oleh *handler* **handleCardKeyDown** yang memungkinkan aktivasi modal

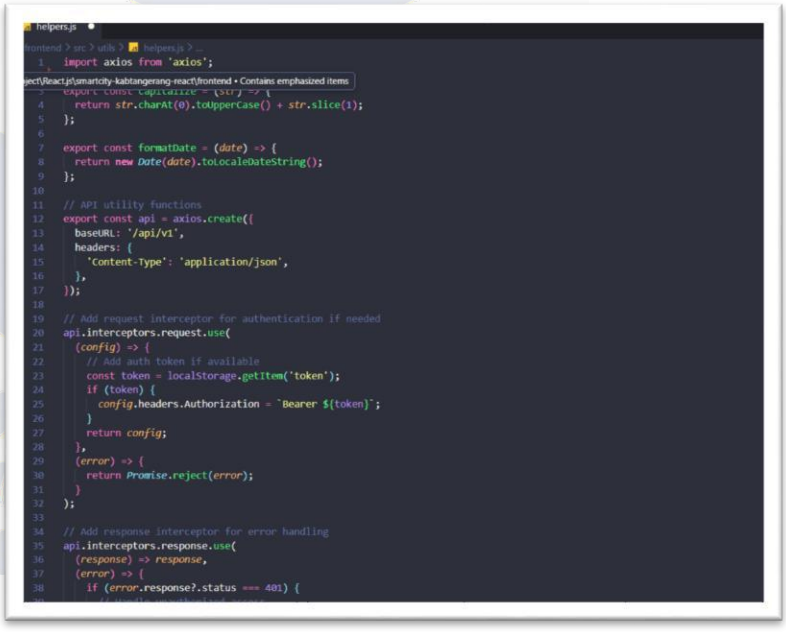
menggunakan tombol *Enter* atau *Space*, memastikan navigasi yang lancar bagi pengguna alat bantu pembaca layer [11].

Dari perspektif desain visual dan interaksi, komponen CSS mendukung tema dengan estetika modern dan responsif. *Hero Section* menggunakan citra latar belakang perkotaan yang *fixed* dan efek *parallax* (melalui background-attachment: fixed), memberikan kedalaman visual pada bagian pembuka. Bagian **dimensi-grid** menggunakan tata letak *CSS Grid* yang beradaptasi secara mulus, mengkonfigurasi ulang *card* dari multi-kolom menjadi tumpukan tunggal pada perangkat seluler. Setiap **.dimensi-card** ditingkatkan dengan efek *hover* yang canggih, termasuk transformasi **translateY(-10px)** untuk menonjolkan elevasi, dan efek *shine* yang halus melalui *pseudo-element* **::before** dengan linear-gradient yang beranimasi, memperkuat kesan modern dan teknologi. Detail desain interaktif juga terintegrasi pada *modal*, dengan tombol modal yang menggunakan *styling* berbeda (*primary* dengan *gradient* dan *secondary* bergaris), dan animasi *keyframe* **modalSlideIn** yang memberikan transisi visual yang berkesan saat modal muncul, secara keseluruhan menegaskan dedikasi proyek pada antarmuka yang tidak hanya fungsional tetapi juga menarik secara visual.

9. Helper.js: Analisis Arsitektur Layanan dan Implementasi Klien HTTP Global

File *helpers.js* merepresentasikan lapisan *infrastructure* yang esensial dalam arsitektur frontend, yang mengonsolidasikan berbagai fungsionalitas umum aplikasi sekaligus mendefinisikan lapisan abstraksi data melalui integrasi pustaka Axios. Secara fungsional, file ini menyediakan utilitas yang bersifat independen dari React, seperti fungsi capitalize dan formatDate, guna menjamin

konsistensi representasi data di seluruh bagian frontend. Peran utamanya adalah mendefinisikan instans klien HTTP global, yaitu variabel api, yang diinisialisasi menggunakan axios.create dengan *base URL* terpusat (/api/v1) serta pengaturan *header* Content-Type: application/json sebagai nilai default. Pendekatan ini secara efektif menerapkan prinsip *Single Source of Truth* untuk konfigurasi komunikasi API. Lebih lanjut, pemanfaatan *request* dan *response interceptors* pada instans api mencerminkan praktik *software engineering* yang matang. *Request interceptor* secara otomatis menyuntikkan token otentikasi (Authorization: Bearer <token>) yang diambil dari *localStorage* sebelum setiap permintaan dikirim, sehingga proses otentikasi dapat diterapkan secara konsisten dan terotomatisasi di seluruh aplikasi, sebagaimana ditunjukkan pada Gambar 3.15.



Gambar 3.15 Code Helper.js

Desain *Service Layer* ini diperkuat secara signifikan oleh *response interceptor* yang berorientasi pada ketahanan dan pengalaman pengguna. *Interceptor* ini secara khusus bertugas

menangani kegagalan otentikasi: ketika server mengembalikan kode status **401 (Unauthorized)**, *interceptor* segera menjalankan kebijakan *cleanup* dengan menghapus *token* otentikasi yang kedaluwarsa dari `localStorage`. Tindakan ini tidak hanya mencegah upaya *request* yang gagal di masa mendatang tetapi juga menciptakan titik *hook* logis yang ideal untuk mengimplementasikan pengalihan (*redirect*) pengguna secara paksa ke halaman *login*, menjamin bahwa sesi pengguna ditangani secara aman dan konsisten. Mekanisme penanganan kesalahan terpusat ini membebaskan setiap komponen React dari kewajiban untuk menulis ulang logika otentikasi dan penanganan status 401 secara berulang, sehingga meningkatkan modularitas dan mengurangi redundansi kode. Instans **`api`** yang terkapsulasi ini menjadi dasar bagi definisi *object* **`apiEndpoints`** yang kaya akan fungsionalitas [12].

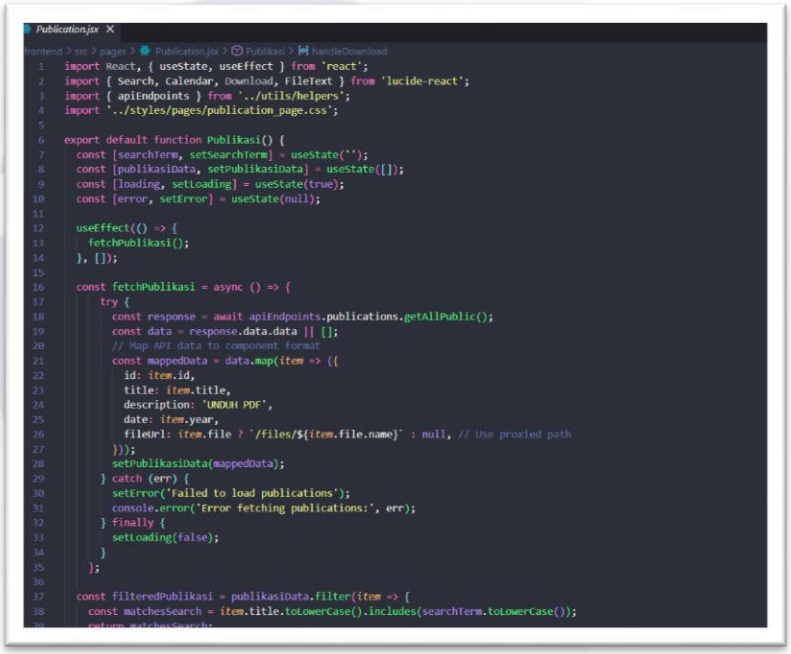
Struktur *object* **`apiEndpoints`** menunjukkan tata kelola *endpoint* yang sangat terorganisir, mengelompokkan operasi berdasarkan sumber daya entitas (seperti `dimensi`, `events`, `publications`, dan `auth`). Pola desain ini, yang dikenal sebagai *Repository Pattern* atau *Service Module Pattern* di *frontend*, menyediakan antarmuka yang bersih dan intuitif untuk semua operasi CRUD dan operasi kustom lainnya (*e.g.,* `getAllPublic`, `upload`, `login`). Setiap *method* di dalam kelompok ini secara langsung memanggil instans **`api`** yang sudah terkonfigurasi (*instance* Axios global), misalnya, `apiEndpoints.dimensi.create(data)` secara internal menggunakan `api.post('/dimensi', data)`. Kasus penggunaan khusus, seperti *upload* file (*e.g.,* `images.upload` dan `files.upload`), ditangani dengan tepat dengan menetapkan *header* **`Content-Type: multipart/form-data`**

yang diperlukan secara lokal, memastikan fungsionalitas API yang kompleks terbungkus dalam antarmuka yang disederhanakan dan mudah digunakan. Keseluruhan struktur *file* ini menegaskan bahwa fondasi aplikasi dibangun di atas prinsip *software architecture* modern yang berfokus pada skalabilitas, pemeliharaan, dan praktik keamanan dasar.

10. Publikasi: Analisis Manajemen Data dan Desain Akses Informasi

Komponen *Publikasi.jsx* dirancang sebagai repositori publik untuk dokumen resmi, yang menampilkan implementasi pola *fetch-and-render* secara efisien dengan penekanan pada keterbacaan data dan fungsionalitas pencarian. Komponen ini memanfaatkan *hook useEffect* untuk melakukan pengambilan data asinkron satu kali melalui *endpoint* apiEndpoints.publications.getAllPublic(), sehingga data publikasi dapat tersedia segera setelah komponen di-*mount*. Mekanisme pengelolaan *state* internal, termasuk loading dan error, diimplementasikan secara sistematis untuk memberikan umpan balik visual yang jelas kepada pengguna selama proses pengambilan data berlangsung maupun ketika terjadi kegagalan jaringan, yang merupakan praktik terbaik dalam perancangan *user experience* pada aplikasi berbasis data. Data yang diterima dari API selanjutnya dipetakan (*mappedData*) agar selaras dengan struktur presentasi yang dibutuhkan oleh komponen tabel, sebagai langkah penting dalam memisahkan logika backend dari lapisan tampilan frontend. Fungsionalitas pencarian (*filtering*) diintegrasikan secara reaktif melalui *state* searchTerm dan *event handler* handleSearch, yang menyaring array publikasiData berdasarkan kecocokan judul secara *case-insensitive*, sehingga daftar dokumen yang ditampilkan selalu

relevan dengan kueri pengguna, sebagaimana ditunjukkan pada Gambar 3.16.

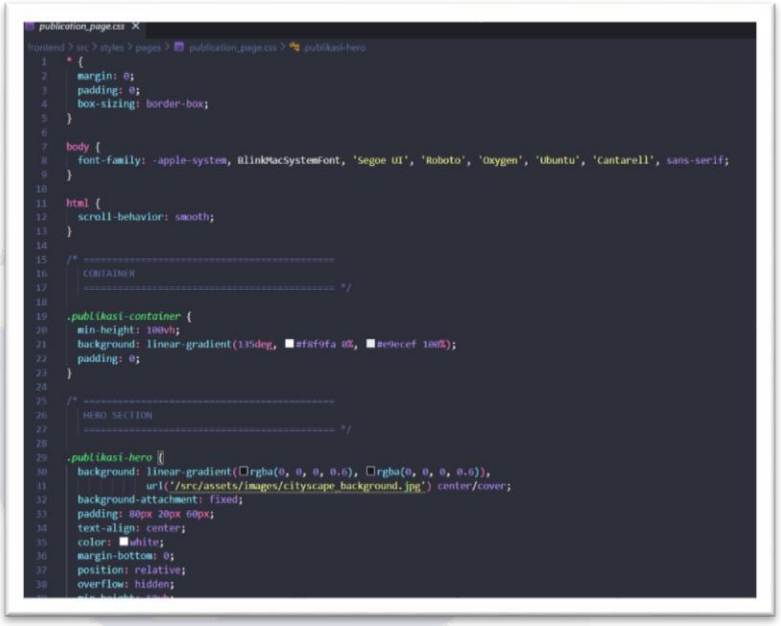


```jsx
import React, { useState, useEffect } from 'react';
import { Search, Calendar, Download, FileText } from 'lucide-react';
import { apiEndpoints } from '../utils/helpers';
import '../styles/pages/publication_page.css';

export default function Publikasi() {
  const [searchTerm, setSearchTerm] = useState('');
  const [publikasiData, setPublikasiData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetchPublikasi();
  }, []);

  const fetchPublikasi = async () => {
    try {
      const response = await apiEndpoints.publications.getAllPublic();
      const data = response.data.data || [];
      // Map API data to component format
      const mappedData = data.map(item => ({
        id: item.id,
        title: item.title,
        description: 'UNDUH PDF',
        date: item.year,
        fileUrl: item.file ? `/files/${item.file.name}` : null, // Use proxied path
      }));
      setPublikasiData(mappedData);
    } catch (err) {
      setError('Failed to load publications');
      console.error('Error fetching publications:', err);
    } finally {
      setLoading(false);
    }
  };

  const filteredPublikasi = publikasiData.filter(item => {
    const matchesSearch = item.title.toLowerCase().includes(searchTerm.toLowerCase());
```

Gambar 3.16 Code Publication.jsx

Dari perspektif desain antarmuka, *markup* HTML difokuskan pada penyajian data tabular yang terstruktur dan mudah diakses. Elemen kunci adalah *card* dokumen yang membungkus tabel, yang diposisikan secara strategis di bawah *Hero Section* dengan *negative margin* (margin-top: -60px), sebuah teknik desain yang memberikan kedalaman visual dan kesan konten "muncul" dari latar belakang. Tabel itu sendiri dioptimalkan untuk aksesibilitas dan kemudahan penggunaan. Kolom judul menampilkan tautan yang dapat diklik (dengan *styling underline* dan warna biru) untuk memicu fungsi pratinjau dokumen (**handleReview**), sementara kolom deskripsi dialihfungsikan menjadi tombol **Download** yang memanggil **handleDownload**, keduanya memanfaatkan **window.open(fileUrl, '_blank')** untuk membuka atau mengunduh dokumen di *tab* baru. Penggunaan ikon **Lucide-**

React (*Search, Calendar, FileText*) secara konsisten memperkaya estetika dan semantik kolom data, meningkatkan pemahaman visual terhadap fungsi setiap elemen [13].



Gambar 3.17 Code publication_page.css

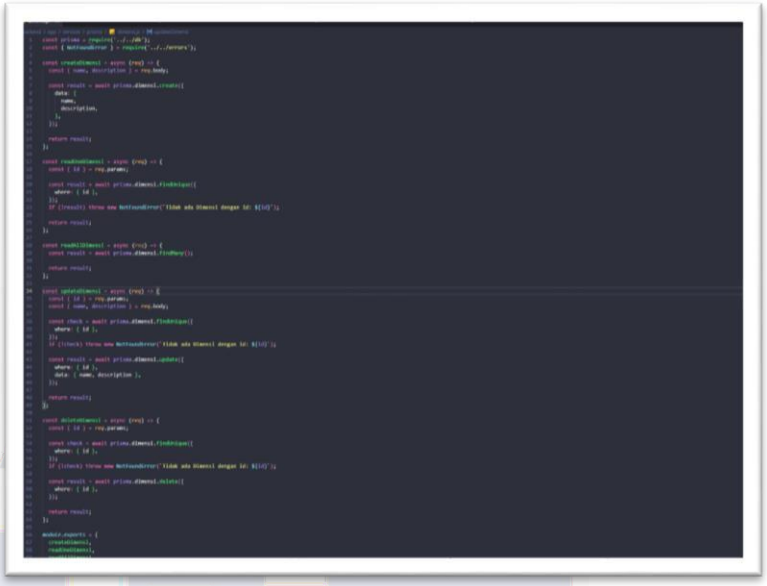
Pada lapisan CSS, *styling* mencapai adaptabilitas dan profesionalisme yang tinggi. *Hero Section* memanfaatkan citra latar belakang *fixed* dengan *overlay* gelap, menciptakan kontras yang kuat untuk judul halaman. Aspek paling canggih terletak pada *responsive design* untuk tabel. Pada ukuran layar yang lebih kecil (@media (max-width: 640px)), tata letak tabel secara programatis diubah dari format baris-kolom tradisional menjadi tampilan tumpukan vertikal (*stacked*). Transformasi ini dicapai melalui properti display: block pada baris dan sel tabel, dikombinasikan dengan penggunaan *pseudo-element* **::before** untuk menampilkan label *header* kolom (**content: attr(data-label)**), yang secara efektif mempertahankan konteks data tanpa mengorbankan keterbacaan pada perangkat seluler. Selain itu, transisi halus dan animasi **fadeIn** diterapkan pada *card* dan baris tabel,

secara kolektif meningkatkan persepsi kecepatan dan kualitas aplikasi.

3.3.1.3 Membuat Back-End Website

1. **Analisis Arsitektur Service Layer dan Operasi Persistensi Data Berbasis Prisma ORM**

   Modul **dimensi.js** dalam lapisan layanan (*service layer*) ini merupakan implementasi konkret dari pola deskriptor objek yang memfasilitasi interaksi antara logika bisnis aplikasi dan basis data relasional melalui **Prisma ORM**. Secara arsitektural, kode ini berfungsi sebagai fasilitator operasi CRUD (*Create, Read, Update, Delete*) yang terenkapsulasi, memastikan bahwa setiap manipulasi data pada entitas "Dimensi" dilakukan melalui prosedur yang terstandarisasi. Penggunaan **prisma.dimensi** menunjukkan pemanfaatan abstraksi kueri yang kuat, di mana fungsi-fungsi asinkron seperti **createDimensi** dan **updateDimensi** secara efisien mengelola persistensi data berdasarkan objek request yang diterima. Strategi ini memisahkan kerumitan sintaksis SQL dari logika aplikasi, sehingga meningkatkan keterbacaan dan mempermudah pemeliharaan jangka panjang terhadap skema data yang ada. sebagaimana ditunjukkan pada Gambar 3.18.



Gambar 3.18 Code dimensi.js

Aspek krusial lainnya dari implementasi ini adalah integrasi mekanisme penanganan kesalahan yang proaktif dan presisi. Dalam fungsi-fungsi seperti **readOneDimensi**, **updateDimensi**, dan **deleteDimensi**, terdapat logika validasi keberadaan data menggunakan metode **findUnique** sebelum operasi utama dieksekusi. Jika identitas data yang diminta tidak ditemukan dalam repositori, sistem secara otomatis melemparkan **NotFoundError** dengan pesan spesifik yang mencantumkan identitas unik terkait. Pendekatan ini merupakan manifestasi dari prinsip desain yang defensif, menjamin integritas referensial dan mencegah eksekusi operasi pada data yang tidak eksis. Dengan mengonsolidasikan seluruh logika manipulasi entitas Dimensi ke dalam modul tunggal ini, arsitektur *backend* berhasil mencapai kohesi fungsional yang tinggi sekaligus meminimalkan redundansi kode pada lapisan pengontrol (*controller*) [14].

2. Analisis Pemodelan Data Relasional dan Spesifikasi Skema Basis Data Menggunakan Prisma ORM

Berkas schema.prisma ini mendefinisikan arsitektur data fundamental aplikasi dengan menggunakan PostgreSQL sebagai sistem manajemen basis data relasional. Secara struktural, skema ini mengadopsi pendekatan normalisasi data untuk menjamin integritas dan efisiensi penyimpanan melalui pendefinisian model-model entitas yang saling terhubung. Entitas **User** berfungsi sebagai pusat otentikasi, yang memiliki relasi *one-to-many* terhadap **UserRefreshToken** untuk mendukung manajemen sesi yang aman melalui mekanisme *refresh token*. Penggunaan tipe data String dengan *default value* uuid() pada setiap atribut identitas (id) menunjukkan implementasi standar industri untuk menjamin keunikan kunci primer secara universal di seluruh tabel, sementara atribut *timestamp* seperti createdAt dan updatedAt secara otomatis merekam jejak audit temporal untuk setiap perubahan data. sebagaimana ditunjukkan pada Gambar 3.19.

Gambar 3.19 Code schema.prima

Kekuatan utama dari skema ini terletak pada desain relasi antar entitas yang mencerminkan logika bisnis secara mendalam. Model **Dimensi** berperan sebagai entitas induk yang memiliki keterkaitan *one-to-many* dengan model **Inovasi**, di mana setiap inovasi secara wajib merujuk pada satu dimensi tertentu melalui kunci tamu (*foreign key*) dimensiId. Selain itu, terdapat pola abstraksi media yang menarik melalui model **Images** dan **Files**. Model **Images** digunakan secara polimorfik namun tetap terikat kuat dengan entitas **Events** dan **Inovasi** melalui relasi *one-to-one*, memastikan setiap aset visual memiliki asosiasi data yang presisi. Hal serupa diterapkan pada model **Publikasi** yang terintegrasi dengan entitas **Files**, memfasilitasi manajemen dokumen digital secara terstruktur. Secara keseluruhan, skema ini mencerminkan integritas referensial yang solid, di mana relasi antar tabel didefinisikan secara eksplisit untuk mencegah anomali data dan mendukung skalabilitas sistem di masa mendatang .

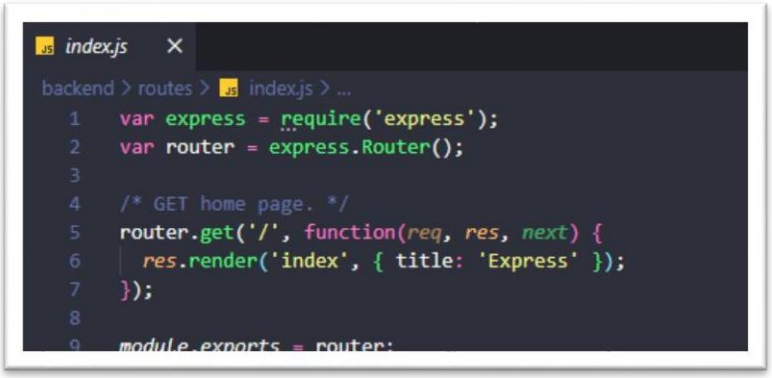
3. Analisis Manajemen Rute Primer dan Penentuan Titik Masuk Utama Aplikasi

Berkas index.js yang berlokasi di dalam direktori routes ini berfungsi sebagai pengatur navigasi utama pada tingkat fundamental aplikasi, yang menggunakan objek express.Router() untuk mengisolasi logika rute secara modular. Penggunaan metode router.get pada jalur akar atau *root path* (/) menunjukkan pendefinisian titik kontak pertama bagi pengguna ketika mengakses alamat dasar server. Melalui fungsi *callback* yang menerima parameter req, res, dan next, modul ini memastikan bahwa setiap permintaan HTTP yang masuk dapat ditangani atau diteruskan secara tepat ke lapisan

berikutnya sesuai dengan kebutuhan operasional sistem. sebagaimana ditunjukkan pada Gambar 3.20.

Gambar 3.20 Code app.js

Fungsi utama dari instruksi res.render('index', { title: 'Express' }) dalam rute ini mencerminkan penggunaan mesin *templating* di sisi server untuk menghasilkan tampilan antarmuka secara dinamis. Perintah tersebut menginstruksikan server untuk memproses berkas tampilan bernama 'index' dan menyuntikkan objek data berupa judul aplikasi sebelum dikirimkan kembali ke klien dalam format HTML. Meskipun aplikasi Anda secara dominan berfungsi sebagai penyedia layanan API, keberadaan rute dasar ini sangat krusial sebagai halaman landas atau mekanisme pemeriksaan kesehatan (*health check*) sederhana guna memastikan bahwa server telah beroperasi dan siap melayani permintaan data lebih lanjut.



Gambar 3.21 Code index.js

Integrasi berkas ini ke dalam struktur sistem secara keseluruhan ditutup dengan pernyataan module.exports = router, yang memungkinkan konfigurasi rute ini untuk diimpor dan didaftarkan pada berkas utama app.js. Dengan memisahkan rute-rute dasar ke dalam modul mandiri seperti ini, arsitektur aplikasi menjadi lebih terorganisir dan mengikuti prinsip pemisahan tanggung jawab (*separation of concerns*). Strategi ini sangat menguntungkan dalam pengembangan skala besar, karena pengembang dapat mengelola rute publik dan rute API secara terpisah tanpa menciptakan konflik kode pada titik masuk utama aplikasi.

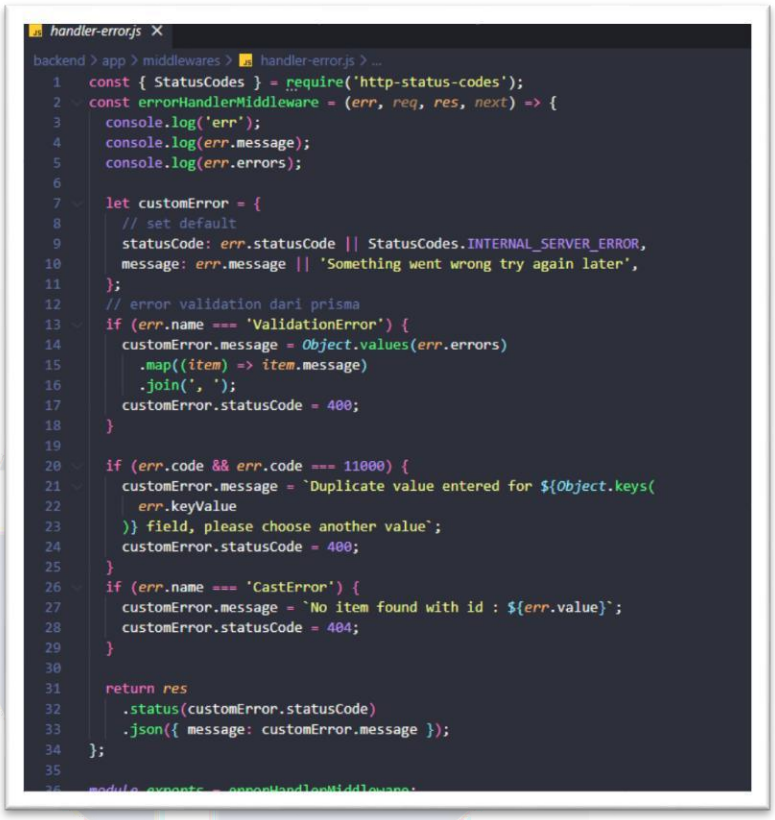
4. Analisis Implementasi Manajemen Kesalahan Terpusat dan Penanganan Terminasi Permintaan HTTP

Implementasi mekanisme penanganan kesalahan dalam aplikasi ini dirancang secara sistematis melalui modul **handler-error.js** dan **not-found.js**, yang berfungsi sebagai jaring pengaman terakhir dalam siklus permintaan-respons. Modul errorHandlerMiddleware mengadopsi pola desain penanganan kesalahan terpusat dengan mengonsolidasikan berbagai jenis anomali sistem ke dalam satu struktur respons JSON yang konsisten. Dengan memanfaatkan pustaka http-status-codes,

sistem secara dinamis menetapkan derajat kegagalan operasional, mulai dari kesalahan internal server hingga validasi input yang tidak memenuhi kriteria. Proses ini memastikan bahwa setiap interupsi pada lapisan logika bisnis tidak menyebabkan penghentian aplikasi secara paksa, melainkan menghasilkan umpan balik yang informatif bagi sisi klien. sebagaimana ditunjukkan pada Gambar 3.22.

Gambar 3.22 Code not-found.js

Lebih lanjut, arsitektur manajemen kesalahan ini menunjukkan kapabilitas yang canggih dalam mendeteksi dan menginterpretasikan kesalahan spesifik yang dihasilkan oleh lapisan persistensi data. Sistem mampu mengidentifikasi anomali seperti ValidationError, duplikasi data pada entitas unik (kode 11000), hingga kegagalan konversi tipe data atau CastError yang sering terjadi pada identitas unik (ID) yang tidak valid. Strategi pemetaan ulang pesan kesalahan asli menjadi pesan kustom melalui objek customError bertujuan untuk menjaga keamanan informasi dengan tidak mengekspos rincian teknis internal atau *stack trace* database kepada pengguna luar. Pendekatan ini secara substansial meningkatkan profesionalisme antarmuka pemrograman aplikasi (API) dengan menyediakan petunjuk yang jelas mengenai langkah korektif yang harus diambil oleh pengguna atau sistem integrasi lainnya.

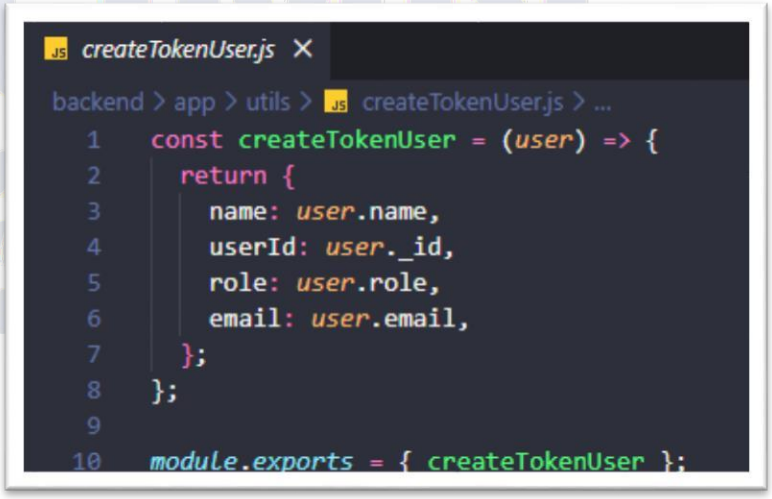


Gambar 3.23 Code handler-error.js

Sebagai pelengkap, modul notFound memainkan peran krusial dalam mengelola terminasi permintaan pada jalur akses yang tidak terdefinisi dalam skema perutean aplikasi. Ketika sebuah permintaan HTTP melewati seluruh rangkaian rute tanpa menemukan kecocokan, *middleware* ini akan secara otomatis mengintersep proses dan mengembalikan status kode 404 beserta pesan indikasi yang eksplisit. Penempatan kedua modul ini di bagian akhir rantai eksekusi pada app.js menjamin bahwa setiap interaksi pengguna dengan server, baik yang valid, salah sasaran, maupun yang mengalami kegagalan teknis, selalu berakhir pada sebuah respons yang terkendali. Secara kolektif, integrasi ini memperkuat aspek ketahanan sistem dan memastikan integritas alur kerja aplikasi secara menyeluruh.

5. Analisis Mekanisme Enkapsulasi Keamanan dan Transformasi Data Identitas Pengguna

Modul utilitas yang mencakup jwt.js dan createTokenUser.js ini merepresentasikan lapisan abstraksi kritis yang mengelola integritas otentikasi serta standardisasi data subjek dalam sistem. Pada berkas jwt.js, implementasi protokol *JSON Web Token* dilakukan dengan mengadopsi skema *dual-token*, yakni penggunaan *access token* dan *refresh token* yang memiliki kunci rahasia serta masa kedaluwarsa yang berbeda. Melalui fungsi createJWT dan createRefreshJWT, sistem melakukan proses penandatanganan kriptografis terhadap *payload* data pengguna, sementara fungsi isTokenValid memastikan bahwa setiap instruksi yang masuk telah melewati verifikasi integritas guna mencegah manipulasi data oleh pihak yang tidak berwenang. Pemisahan tanggung jawab antara akses jangka pendek dan penyegaran sesi ini merupakan manifestasi dari prinsip pertahanan berlapis untuk memitigasi risiko pembajakan sesi. sebagaimana ditunjukkan pada Gambar 3.24.



Gambar 3.24 Code createTokenUser.js

Selanjutnya, integrasi fungsi createTokenUser berperan sebagai mekanisme transformasi objek atau *Data Transfer Object* (DTO) yang menyaring informasi sensitif sebelum proses enkripsi dilakukan. Dengan melakukan ekstraksi atribut spesifik seperti name, userId, role, dan email dari objek pengguna mentah, sistem memastikan bahwa *payload* yang disematkan ke dalam JWT hanya mengandung data minimal yang diperlukan untuk keperluan otorisasi. Praktik ini secara signifikan meningkatkan efisiensi transmisi data serta memperkuat aspek privasi, karena informasi internal seperti *hash* kata sandi atau data meta-data lainnya tidak ikut terekspos dalam struktur token yang dikirimkan melalui jaringan. Standardisasi format data ini juga menjamin konsistensi akses informasi identitas di seluruh lapisan *middleware* dan layanan lainnya dalam aplikasi.

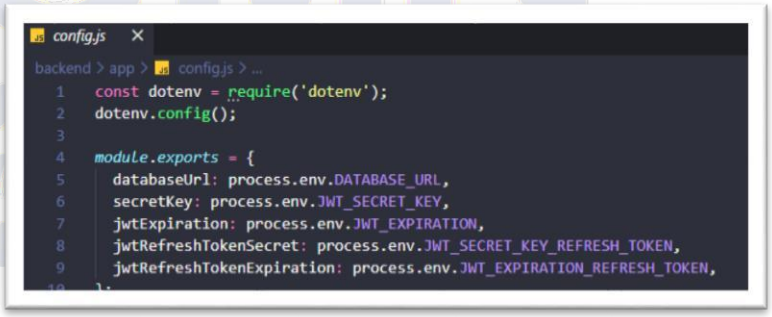
Gambar 1 3.25 Code jwt.js

Secara arsitektural, sinergi antara kedua modul pembantu ini menciptakan fondasi yang stabil bagi manajemen identitas digital dalam ekosistem. Penggunaan variabel konfigurasi terpusat untuk mengelola parameter keamanan menunjukkan desain perangkat lunak yang mengutamakan fleksibilitas dan kemudahan pemeliharaan, di mana kebijakan kedaluwarsa token dapat dimodifikasi tanpa harus mengubah logika inti kriptografi. Melalui enkapsulasi logika teknis yang kompleks ke dalam fungsi-fungsi pembantu yang bersifat *reusable*, sistem berhasil mengurangi redundansi kode sekaligus memperkecil kemungkinan terjadinya kesalahan manusia dalam implementasi protokol keamanan. Secara keseluruhan, integrasi utilitas ini melengkapi siklus hidup data pengguna,

mulai dari identifikasi entitas hingga proses validasi akses yang aman dan terstandarisasi.

6. Analisis Manajemen Dependensi dan Konfigurasi Lingkungan Sistem

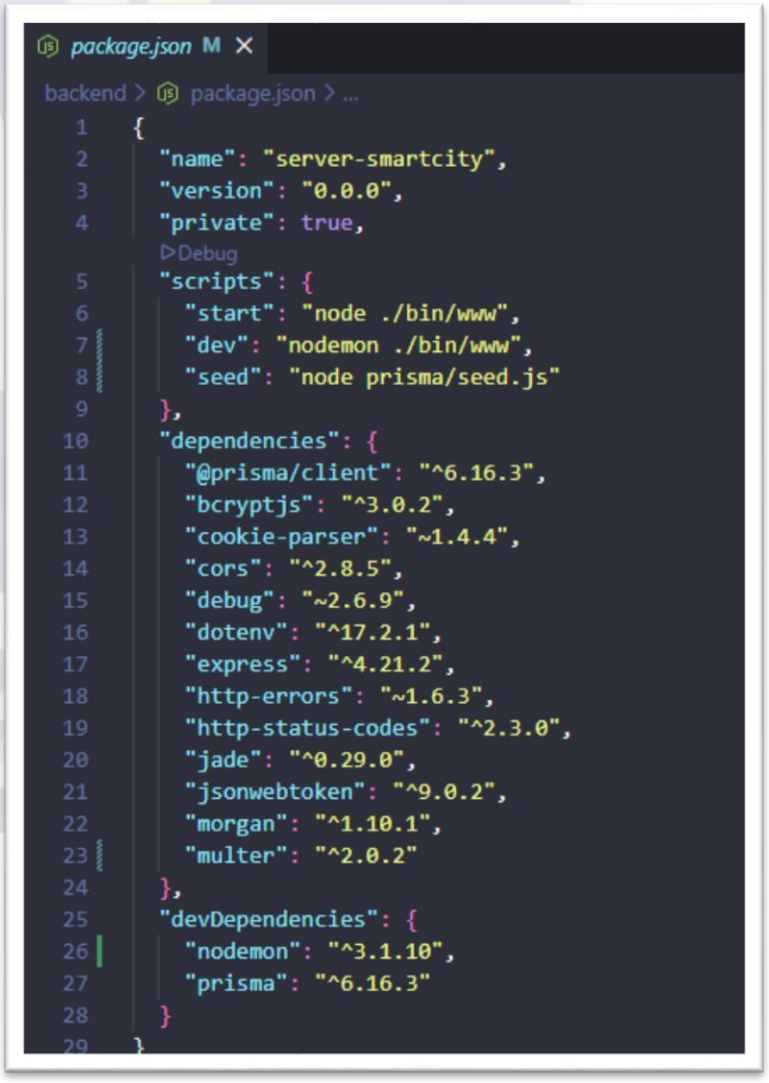
Berkas **package.json** dalam arsitektur proyek ini berfungsi sebagai manifestasi teknis yang mendokumentasikan seluruh ekosistem pustaka serta skrip operasional yang mendukung keberfungsian server. Penggunaan **Prisma Client** sebagai dependensi utama menegaskan adopsi teknologi ORM modern untuk manajemen basis data, sementara keberadaan pustaka seperti bcryptjs dan jsonwebtoken mengindikasikan implementasi standar keamanan kriptografis untuk proteksi kredensial dan sesi pengguna. Selain manajemen pustaka, berkas ini mengatur alur kerja pengembangan melalui skrip otomasi seperti nodemon untuk lingkungan pengembangan dan perintah seed untuk inisialisasi data awal, yang secara kolektif meningkatkan efisiensi siklus hidup pengembangan perangkat lunak serta memastikan konsistensi lingkungan kerja antar pengembang. sebagaimana ditunjukkan pada Gambar 3.26.



Gambar 3.26 Code config.js

Secara paralel, berkas **config.js** berperan sebagai lapisan abstraksi yang mengelola parameter sensitif melalui integrasi dengan modul dotenv. Strategi ini menerapkan prinsip

pemisahan antara kode program dan data konfigurasi (*separation of config from code*), di mana informasi kritikal seperti URL basis data dan kunci rahasia enkripsi JWT dimuat secara dinamis dari variabel lingkungan sistem. Pendekatan ini merupakan praktik terbaik dalam keamanan aplikasi untuk mencegah paparan informasi rahasia pada repositori kode, sekaligus memberikan fleksibilitas tinggi bagi pengembang untuk mengubah parameter operasional seperti durasi kedaluwarsa token tanpa harus melakukan modifikasi pada logika inti aplikasi.



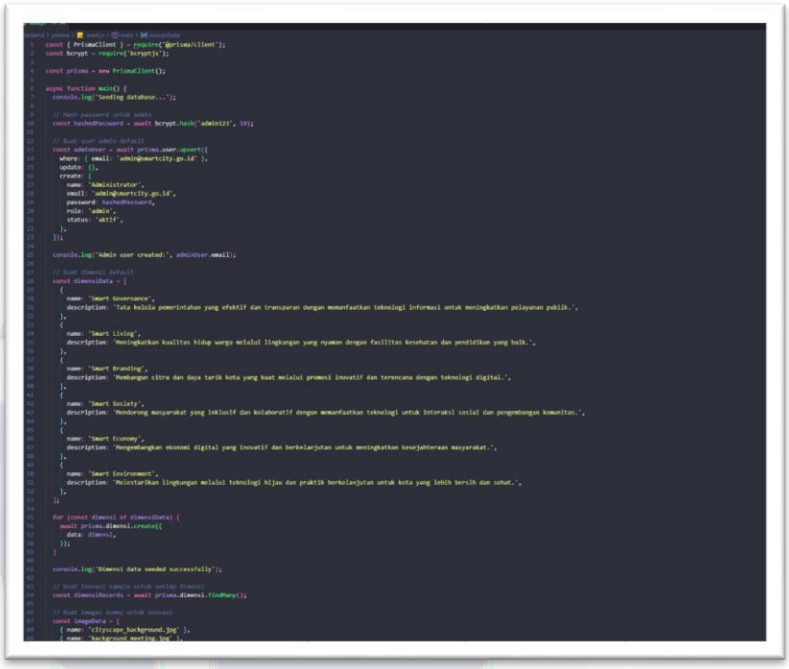
Gambar 3.27 Code package.json

Sinergi antara manajemen dependensi dan konfigurasi ini menciptakan fondasi yang kokoh bagi skalabilitas dan portabilitas sistem. Dengan memusatkan seluruh parameter teknis pada berkas konfigurasi, sistem menjadi lebih adaptif terhadap perubahan infrastruktur, baik saat berada dalam tahap pengujian lokal maupun ketika dideploy ke lingkungan produksi. Pemilihan dependensi yang spesifik, mulai dari penanganan unggahan berkas dengan multer hingga manajemen status HTTP melalui http-status-codes, menunjukkan bahwa aplikasi ini dibangun dengan pertimbangan modularitas yang matang. Secara keseluruhan, kedua berkas ini memastikan bahwa infrastruktur *backend* tidak hanya berfungsi secara teknis, tetapi juga memenuhi standar industri dalam hal tata kelola kode dan keamanan siber.

7. Strategi Inisialisasi Data Otomatis dan Konfigurasi Data Master Sistem

Berkas **seed.js** merupakan komponen krusial yang berfungsi untuk melakukan inisialisasi data awal secara otomatis ke dalam basis data, memastikan sistem memiliki landasan informasi yang siap digunakan segera setelah instalasi. Implementasi ini diawali dengan penggunaan pustaka PrismaClient untuk berinteraksi dengan basis data dan bcryptjs untuk mengamankan kredensial administratif melalui teknik enkripsi satu arah. Proses *seeding* ini mencakup pembuatan akun pengguna administratif utama dengan status aktif, yang bertujuan untuk menjamin ketersediaan akses kendali bagi pengelola sistem tanpa perlu melakukan registrasi manual. Melalui penggunaan fungsi upsert, kode ini menunjukkan ketahanan operasional dengan mencegah terjadinya duplikasi data saat proses inisialisasi dijalankan

berulang kali pada lingkungan pengembangan maupun produksi. sebagaimana ditunjukkan pada Gambar 3.28.



Gambar 3.28 Code seed.js

Lebih lanjut, skrip ini mendefinisikan struktur data master untuk entitas **Dimensi** yang menjadi pilar utama dalam konsep, mencakup aspek *Governance*, *Living*, *Branding*, *Society*, *Economy*, dan *Environment*. Pengisian data dimensi dilakukan secara sistematis melalui iterasi objek yang memuat deskripsi komprehensif untuk masing-masing kategori, yang kemudian dijadikan referensi utama bagi entitas lainnya. Hal ini mencerminkan penerapan integritas referensial di mana setiap data master yang dibuat akan menyediakan identitas unik yang akan digunakan oleh entitas **Inovasi** dan **Images**. Pendekatan terprogram ini meminimalisir kesalahan manusia dalam input data awal dan memastikan bahwa hubungan relasional antar tabel tetap konsisten sesuai dengan skema yang telah didefinisikan sebelumnya.

Pada bagian akhir siklus eksekusi, skrip ini mengimplementasikan pembuatan data sampel untuk entitas **Inovasi** yang secara langsung terhubung dengan dimensi yang telah dibuat sebelumnya melalui mekanisme pencarian ID secara dinamis. Integrasi antara aset gambar dummy dan deskripsi inovasi fungsional memberikan gambaran nyata mengenai bagaimana aplikasi akan menampilkan konten kepada pengguna akhir. Seluruh proses ini ditutup dengan blok penanganan kesalahan (*error handling*) dan pemutusan koneksi basis data secara aman melalui metode $disconnect, yang merupakan praktik terbaik dalam manajemen sumber daya sistem. Secara keseluruhan, keberadaan mekanisme *seeding* ini tidak hanya mempercepat proses distribusi perangkat lunak, tetapi juga berfungsi sebagai dokumentasi teknis mengenai struktur data minimal yang diperlukan agar aplikasi dapat beroperasi secara optimal.

8. Analisis Implementasi Logika Bisnis Terpusat pada Modul Inovasi dalam Arsitektur Service Layer

Implementasi modul inovasi.js dalam repositori ini merepresentasikan penerapan pola *service layer* yang berfungsi untuk mengabstraksi logika bisnis dari lapisan kendali utama, sehingga menciptakan pemisahan tanggung jawab yang tajam dan terorganisir. Melalui pemanfaatan Prisma sebagai instrumen pemetaan objek relasional, modul ini mampu mengelola entitas inovasi secara deklaratif dengan mengintegrasikan relasi antar-tabel yang kompleks, seperti keterhubungan antara inovasi dengan kategori dimensi dan aset citra. Arsitektur ini tidak hanya memfasilitasi kemudahan dalam pemeliharaan kode jangka panjang, tetapi juga memastikan bahwa setiap interaksi dengan pangkalan data dilakukan melalui satu pintu logika yang terstandarisasi guna

menjaga koherensi sistem secara keseluruhan. sebagaimana ditunjukkan pada Gambar 3.29.

Gambar 3.29 Code inovasi.js

Secara fungsional, modul ini mengorkestrasi siklus hidup data inovasi melalui serangkaian fungsi asinkron yang mencakup operasi penciptaan, ekstraksi, modifikasi, hingga eliminasi data secara sistematis. Dalam proses pendaftaran dan pemutakhiran informasi, sistem secara proaktif melakukan verifikasi terhadap keberadaan aset gambar melalui prosedur pemeriksaan internal guna menjamin bahwa setiap entitas inovasi memiliki referensi data yang valid. Selain itu, kapabilitas pengambilan data kolektif dalam modul ini didesain dengan fleksibilitas tinggi, yang memungkinkan

adanya penyaringan informasi berdasarkan identitas dimensi tertentu serta penyertaan relasi data yang mendalam, sehingga informasi yang dihasilkan bersifat komprehensif dan siap dikonsumsi oleh lapisan antarmuka.

Ketahanan operasional sistem ini diperkuat dengan pengadopsi strategi penanganan eksepsi yang ketat melalui integrasi kelas kesalahan kustom seperti NotFoundError. Mekanisme ini memastikan bahwa setiap instruksi yang melibatkan identitas data yang tidak valid akan segera diinterupsi dan diarahkan pada jalur penanganan kesalahan yang informatif, alih-alih membiarkan kegagalan sistem terjadi secara mendadak. Dengan melakukan validasi keberadaan record sebelum mengeksekusi perintah pemutakhiran atau penghapusan, modul ini mendemonstrasikan prinsip kehati-hatian dalam manajemen persistensi data yang krusial bagi integritas informasi dalam ekosistem. Penggabungan antara logika bisnis yang solid dan manajemen kesalahan yang responsif ini pada akhirnya menjamin stabilitas fungsional sistem dalam melayani kebutuhan pengolahan data inovasi.

9. Analisis Fungsionalitas Pengendali Administrasi Inovasi pada Arsitektur Manajemen Konten

Modul pengendali atau **controller** pada entitas inovasi ini merupakan lapisan orkestrasi yang secara spesifik menjembatani protokol komunikasi HTTP dengan logika bisnis internal yang telah didefinisikan pada *service layer*. Setiap fungsi di dalam modul ini dirancang menggunakan pola asinkron untuk menangani siklus permintaan (*request*) dan tanggapan (*response*) secara efisien, dengan mengintegrasikan blok try-catch sebagai mekanisme pertahanan awal dalam menangkap kegagalan operasional.

Penggunaan pustaka http-status-codes memberikan presisi semantik pada setiap balasan yang dikirimkan kepada klien, sehingga status keberhasilan seperti **CREATED** untuk pencatatan data baru atau **OK** untuk pengambilan data dapat mencerminkan hasil eksekusi prosedur pangkalan data secara akurat dan terstandarisasi.

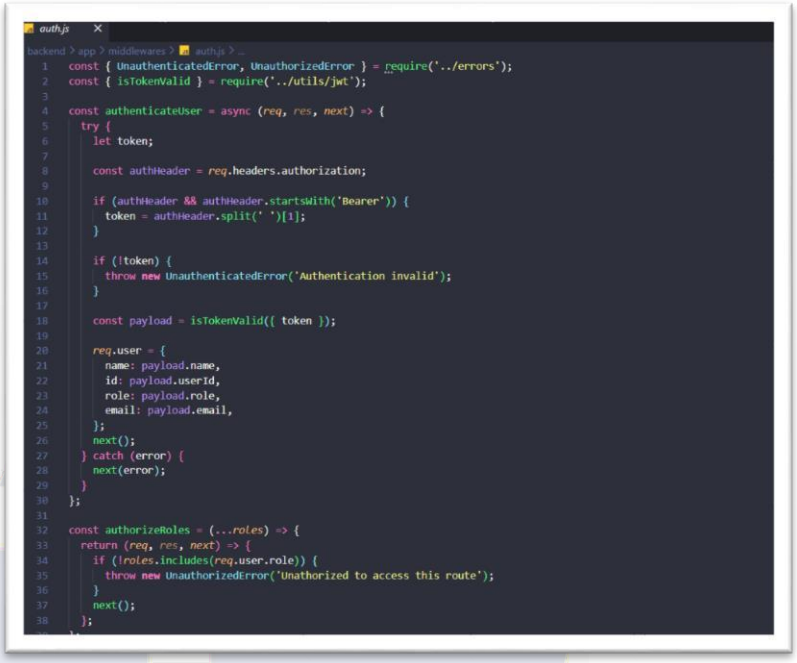
Signifikansi teknis dari pengendali ini terlihat menonjol pada kemampuannya melakukan transformasi data dinamis, khususnya dalam penyajian informasi kolektif melalui fungsi readAllCMSInovasi. Sistem secara otomatis mengonstruksi alamat URL absolut untuk setiap aset citra dengan menyatukan parameter protokol, *host*, dan nama berkas yang tersimpan di server, sehingga meminimalkan beban pemrosesan pada sisi klien saat melakukan perenderan konten visual. Proses pemetaan data ini menunjukkan adanya pertimbangan mendalam terhadap aspek **usabilitas API**, di mana data mentah dari lapisan persistensi disempurnakan menjadi format yang siap saji untuk kebutuhan antarmuka pengguna pada sistem manajemen konten (CMS).

Secara arsitektural, implementasi ini memperkokoh prinsip **Separation of Concerns** dengan mendelegasikan manipulasi data mendalam kepada modul layanan terkait, sementara pengendali tetap berfokus pada manajemen alur navigasi dan penyusunan struktur respons. Sinergi yang terbangun antara fungsi-fungsi operasional seperti pembuatan, pembaruan, hingga penghapusan inovasi memastikan bahwa setiap transaksi informasi tetap terjaga konsistensinya di bawah pengawasan sistem manajemen yang terpusat. Dengan demikian, lapisan pengendali ini tidak hanya berfungsi sebagai gerbang masuk data, tetapi juga sebagai filter administratif yang menjamin bahwa setiap interaksi antara

pengguna administratif dan basis data berlangsung dalam parameter yang telah ditetapkan secara sistemik.

10. Analisis Implementasi Middleware Otentikasi dan Otorisasi (JWT)

Modul auth.js ini secara fundamental menyediakan dua fungsi *middleware* yang krusial untuk mengamankan *endpoint* API, yaitu **authenticateUser** dan **authorizeRoles**. Fungsi **authenticateUser** dirancang untuk memproses dan memvalidasi kredensial pengguna yang dikirimkan dalam setiap permintaan. Mekanismenya dimulai dengan ekstraksi *token* JWT dari *header* HTTP **Authorization**, secara khusus mencari skema **Bearer Token**. Setelah *token* berhasil diisolasi, fungsi ini segera memanggil utilitas **isTokenValid** yang diimpor, yang bertanggung jawab untuk memverifikasi integritas dan validitas *token* tersebut. Jika proses validasi ini berhasil, *payload* yang telah didekodekan, yang berisi data identitas kritis seperti name, id, role, dan email, akan disuntikkan ke dalam *object* **req.user**. Tindakan penyuntikan ini merupakan langkah arsitektur yang vital, memastikan bahwa data pengguna yang terotentikasi dapat diakses secara instan oleh *controller* atau *middleware* berikutnya dalam rantai pemrosesan permintaan. Kegagalan dalam menemukan atau memvalidasi *token* akan segera memicu pelemparan objek kesalahan khusus **UnauthenticatedError**, yang kemudian diteruskan ke lapisan penanganan kesalahan global oleh **next(error)**. sebagaimana ditunjukkan pada Gambar 3.30.



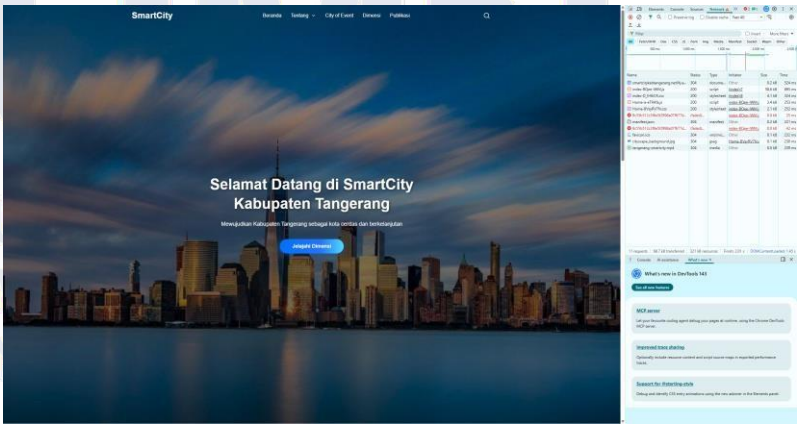
Gambar 3.30 Code auth.js

Selain otentikasi identitas, modul ini juga mengimplementasikan mekanisme kontrol akses yang terperinci melalui *middleware* **authorizeRoles**. Fungsi ini adalah sebuah *Higher-Order Function* (HOF) yang menerima parameter sisa (...roles), yaitu daftar peran yang diizinkan untuk mengakses *endpoint* tertentu, dan mengembalikan *middleware* yang akan dieksekusi. Tujuan utama dari *middleware* yang dikembalikan ini adalah untuk melakukan pemeriksaan otorisasi deklaratif: ia membandingkan peran pengguna yang sudah terekstrak dan tersimpan di **req.user.role** dengan daftar peran yang telah ditentukan. Jika peran pengguna tidak ditemukan dalam daftar yang diizinkan, ini mengindikasikan pelanggaran terhadap kebijakan akses, dan *middleware* akan segera melempar objek kesalahan **UnauthorizedError**. Desain *Higher-Order Function* ini sangat efisien, memungkinkan para pengembang untuk menerapkan kebijakan kontrol akses berbasis peran (*Role-Based Access Control* atau RBAC) dengan sintaksis yang bersih dan ekspresif langsung pada

definisi *route*, sehingga memisahkan logika keamanan dari logika bisnis utama aplikasi.

3.3.1.4 Hasil Pengujian Website Kabupaten Tangerang Berbasis React.JS

Berdasarkan pengujian yang dilakukan menggunakan Google Chrome Developer Tools, website Kabupaten Tangerang menunjukkan penerapan teknologi React.js yang cukup baik. Halaman utama website menampilkan antarmuka yang menarik dengan header bertuliskan "Selamat Datang di Kabupaten Tangerang" yang dilengkapi gambar latar belakang pemandangan kota dan tombol "Jelajahi Dimensi" sebagai navigasi utama. Menu-menu seperti Beranda, Tentang, City of Event, Dimensi, dan Publikasi tertata rapi di bagian atas halaman. Dari sisi teknis, struktur file dan komponen yang terdeteksi pada Developer Tools membuktikan bahwa website ini memang dibangun menggunakan framework React.js, yang memungkinkan tampilan lebih dinamis dan responsif terhadap interaksi pengguna.



Gambar 2 Website Kabupaten Tangerang

Dari hasil pemeriksaan pada panel Network, website berhasil memuat berbagai jenis file yang dibutuhkan termasuk file HTML, CSS, JavaScript, dan beberapa file gambar dengan

format JPEG dan PNG. Meskipun semua resource dapat dimuat dengan baik, terlihat bahwa beberapa file memiliki ukuran yang cukup besar terutama untuk gambar background. Hal ini berpotensi memperlambat waktu loading website khususnya bagi pengguna dengan koneksi internet terbatas. Secara keseluruhan, website dapat berfungsi dengan baik dan struktur kodenya sudah mengikuti standar pengembangan web modern, namun masih ada ruang untuk optimasi terutama dalam hal kompresi gambar agar performa loading menjadi lebih cepat.

3.3.2 Kendala yang Ditemukan

1. **Kesenjangan antara Konsep Teoretis dan Implementasi Sistem**
   Kendala utama yang dihadapi berkaitan dengan perbedaan antara pemahaman konseptual arsitektur perangkat lunak dan penerapannya secara langsung pada lingkungan produksi yang terus berkembang. Penerapan arsitektur berlapis (multilayer architecture) yang mengombinasikan Service Layer dengan Object-Relational Mapping (ORM) Prisma menuntut kemampuan analisis yang matang terhadap alur logika bisnis. Penulis mengalami tantangan dalam menjaga keterpaduan antar lapisan sistem, terutama ketika terjadi perubahan pada struktur basis data. Setiap modifikasi skema mengharuskan peninjauan ulang terhadap alur distribusi data guna menghindari kesalahan fungsional serta memastikan konsistensi dan integritas relasi data.
2. **Kompleksitas Implementasi Keamanan Berbasis JSON Web Token (JWT)**
   Pengembangan sistem autentikasi menggunakan JSON Web Token menghadirkan tantangan tersendiri, khususnya dalam merancang mekanisme rotasi token yang aman namun tetap ramah bagi pengguna. Penulis mengalami kesulitan dalam menyusun penanganan kesalahan (error handling) yang seragam dan terkontrol. Sistem diharapkan mampu menyampaikan informasi kesalahan yang cukup bagi kebutuhan debugging pengembang, tanpa menampilkan detail teknis sensitif yang

berpotensi menimbulkan celah keamanan. Proses ini menuntut pengujian berbagai skenario kegagalan untuk menilai ketahanan arsitektur layanan terhadap gangguan operasional.

3. **Manajemen Aset Digital dan Konfigurasi Lingkungan Server** Tantangan lainnya muncul dalam pengelolaan aset digital serta pengaturan lingkungan pada server statis, khususnya terkait portabilitas sistem. Penulis menemui kendala saat mengimplementasikan pembentukan alamat URL secara dinamis pada lapisan pengendali (controller). Permasalahan ini menuntut pemahaman yang lebih mendalam mengenai struktur direktori, mekanisme asinkron, serta protokol jaringan yang digunakan. Kendala tersebut memerlukan penyesuaian konfigurasi agar sistem dapat berjalan secara konsisten di berbagai lingkungan pengembangan dan produksi.

3.3.3 Solusi atas Kendala yang Ditemukan

1. **Pendekatan Implementasi Arsitektur Service Layer** Untuk mengatasi kesulitan dalam menerjemahkan konsep arsitektur Service Layer ke dalam implementasi nyata, penulis melakukan pendalaman materi melalui dokumentasi resmi serta referensi praktik terbaik yang umum digunakan dalam pengembangan aplikasi berbasis Node.js. Solusi yang diterapkan menitikberatkan pada penerapan pola desain yang konsisten dengan memisahkan logika bisnis secara tegas dari lapisan lain. Setiap komponen dalam Service Layer dirancang memiliki fungsi yang spesifik dan tidak saling tumpang tindih, sehingga alur pemrosesan data menjadi lebih terstruktur dan mudah dipelihara. Selain itu, pemanfaatan Prisma ORM dilakukan secara optimal dengan menerapkan pengujian skema secara berkala serta penggunaan fitur *upsert* dan *connect* untuk menjaga konsistensi relasi data. Pendekatan ini terbukti efektif dalam meminimalkan kesalahan sinkronisasi serta memastikan integritas data tetap terjaga meskipun terjadi penyesuaian pada struktur basis data.

2. **Standarisasi Keamanan dan Penanganan Kesalahan Sistem**

Dalam menghadapi kompleksitas sistem keamanan dan ketidakseragaman penanganan kesalahan, penulis menerapkan strategi sentralisasi logika autentikasi melalui middleware. Dengan pendekatan ini, seluruh proses verifikasi akses pengguna dapat dikelola secara terstruktur dan konsisten. Sistem rotasi token dikembangkan dengan memanfaatkan mekanisme *refresh token* untuk memperpanjang sesi pengguna secara aman tanpa mengorbankan aspek perlindungan data. Untuk mendukung proses debugging dan pemeliharaan sistem, penulis juga mengembangkan kelas kesalahan khusus yang berfungsi menyeragamkan format respons ketika terjadi kegagalan sistem. Solusi ini memungkinkan pengembang memperoleh informasi yang relevan untuk analisis teknis, sementara detail internal yang sensitif tetap terlindungi dari akses pengguna umum. Dengan demikian, sistem mampu menjaga keseimbangan antara keamanan, keterbacaan kesalahan, dan kenyamanan penggunaan.

3. **Optimalisasi Manajemen Aset Digital dan Portabilitas Sistem**
   Kendala terkait pengelolaan aset digital serta perbedaan lingkungan server diatasi dengan menerapkan mekanisme pembentukan URL secara dinamis menggunakan variabel lingkungan. Konfigurasi host dan protokol ditentukan secara terprogram pada lapisan pengendali, sehingga aplikasi dapat menyesuaikan alamat akses aset secara otomatis sesuai dengan lingkungan yang digunakan, baik pada tahap pengembangan maupun produksi.
   Selain itu, penataan struktur direktori berkas statis dilakukan secara sistematis agar dapat diakses secara konsisten oleh aplikasi klien. Pendekatan ini tidak hanya meningkatkan fleksibilitas sistem dalam berbagai lingkungan server, tetapi juga berkontribusi pada peningkatan skalabilitas dan keandalan aplikasi secara keseluruhan.