BAB III
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pada masa awal pelaksanaan magang di Sinarmas Land, posisi yang ditempat adalah *Application Development Intern* di bawah supervisi seorang *IT Web Developer* senior yang juga menjadi pembimbing lapangan. Peserta magang bergabung dalam divisi teknologi informasi perusahaan yang memiliki tanggung jawab utama dalam pengembangan dan pemeliharaan sistem digital, termasuk berbagai *website* proyek properti yang dimiliki oleh Sinarmas Land.

Dalam pelaksanaannya, peran utama difokuskan pada bagian *frontend development*. Aktivitas harian dimulai dengan menerima tugas melalui *task management platform* internal, kemudian melaporkan progres melalui diskusi langsung atau pertemuan rutin seperti *daily stand-up*. Pengerjaan setiap fitur dilakukan secara kolaboratif dan melewati proses *code review*, *merge request*, serta *deployment* ke *staging environment* sebelum akhirnya dirilis ke *production*.

Salah satu proyek utama yang dikerjakan adalah proses *revamp website* Kota Wisata Cibubur. Dalam proyek tersebut, tugas yang dilaksanakan adalah membangun ulang antarmuka halaman (*UI layout*), mengatur ulang struktur *routing*, serta mengintegrasikan data dari *backend* melalui *API* yang disediakan dalam sistem *Content Management System* (CMS) berbasis *Strapi*. Proses ini meliputi perencanaan, pengembangan, pengujian, hingga penyusunan dokumentasi teknis.

Alur kerja pengembangan dimulai dari penerimaan *task*, implementasi fitur, hingga tahap validasi hasil kerja melalui *cross-browser testing* dan pengujian pada berbagai perangkat. Proses kolaborasi dilakukan menggunakan *GitHub* untuk pengelolaan versi kode. Setiap perubahan dikerjakan di *branch* terpisah, kemudian diajukan melalui *pull request* sebelum digabungkan ke main *branch*.

Koordinasi lintas divisi juga dilakukan jika diperlukan, seperti dengan tim desain, tim konten, atau tim *backend*, untuk memastikan bahwa setiap fitur yang

dikembangkan terintegrasi secara menyeluruh dan tidak mengganggu fungsi lain.

Melalui sistem kerja yang kolaboratif dan berbasis *agile*, pelaksanaan magang ini memeberikan pengalaman langsung dalam pengembangan perangkat lunak di lingkungan profesional. Selain itu, kegiatan ini memberikan pemahaman mengenai pentingnya koordinasi, dokumentasi, dan kedisiplinan dalam mengikuti setiap tahapan dalam *software development lifecycle*.

3.2 Tugas yang Dilakukan

Pelaksanaan kerja magang berlangsung selama 24 minggu atau 6 bulan, terhitung sejak tanggal 17 Februari 2025 hingga 31 Juli 2025. Selama periode tersebut, kegiatan magang dilaksanakan melalui beberapa tahapan yang meliputi pembelajaran teknologi, analisis *website* lama, pengembangan ulang halaman, hingga integrasi *API* dan optimasi sistem. Rincian kegiatan mingguan ditampilkan pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu

Minggu Ke	Pekerjaan yang Dilakukan
1-2	• Orientasi kerja dan pengenalan tim *Application Development*. • Mempelajari teknologi utama: *Next.js*, *Tailwind CSS*, *Strapi CMS*, *GitHub*. • Mengikuti *daily stand-up meeting* untuk memahami alur kerja tim.
3-4	• Analisis struktur *website* Kota Wisata Cibubur berbasis *WordPress*. • *Breakdown* halaman utama (*Home*, *About*, *Contact*, *News*, *Facilities*, dll). • Pembagian tugas pengembangan halaman di tim.
5-6	• *Setup* awal proyek *Next.js* (struktur folder, *globals.css*, *layout.tsx*).

	• Pembuatan komponen global: *Header* dan *Footer*. • Implementasi awal halaman *HomePage* dengan tampilan statis.
7-8	• Penyempurnaan tampilan *HomePage* dengan *Tailwind CSS*. • Pengembangan halaman *About* dan *Contact*. • *Review* kode melalui *GitHub Pull Request*.
9-10	• Pengembangan halaman *Facilities* dan *Developments*. • Integrasi awal data *dummy* untuk simulasi isi konten. • Validasi tampilan halaman dengan *Chrome DevTools* (*desktop* & *mobile*).
11-12	• Implementasi halaman *News* dan *Access*. • Penyesuaian *layout* navigasi global. • *Debugging bug* tampilan di beberapa resolusi layar.
13-14	• Mulai pengembangan fitur *Auth* (*Login*, *Register*, *Subscribe*). • Pembuatan *form login* & *register* menggunakan komponen *Next.js*. • Validasi input sederhana (*required field*, *format email*).
15-16	• Penyempurnaan tampilan halaman *Auth* (*Login*, *Register*, *Subscribe*). • Integrasi halaman *Auth* dengan *backend Strapi*. • Penyesuaian proteksi *routing* untuk halaman tertentu.
17-18	• Pengembangan halaman *Search*. • Implementasi fitur pencarian artikel/berita.

	• Uji coba *filter* pencarian berdasarkan kata kunci.
19-20	• Integrasi *API Strapi* untuk seluruh halaman konten (*About*, *Contact*, *News*, *Facilities*, dll). • Penyesuaian *format response API* ke *frontend*. • *Debugging error* terkait data *JSON*.
21-22	• Finalisasi halaman *Promo Event* (jika ada *update* dari *backend*). • *Refactoring* struktur folder untuk *dynamic routing* [lang] (multibahasa). • Pengujian responsivitas penuh di berbagai perangkat.
23-24	• Finalisasi seluruh halaman *website* Kota Wisata Cibubur (*Home*, *About*, *Access*, *Contact*, *Developments*, *Facilities*, *News*, *Search*, *Auth*). • Penyerahan *source code project* kepada tim *DevOps* untuk proses *deployment* ke *server*/*hosting*.

Perangkat Lunak yang Digunakan:

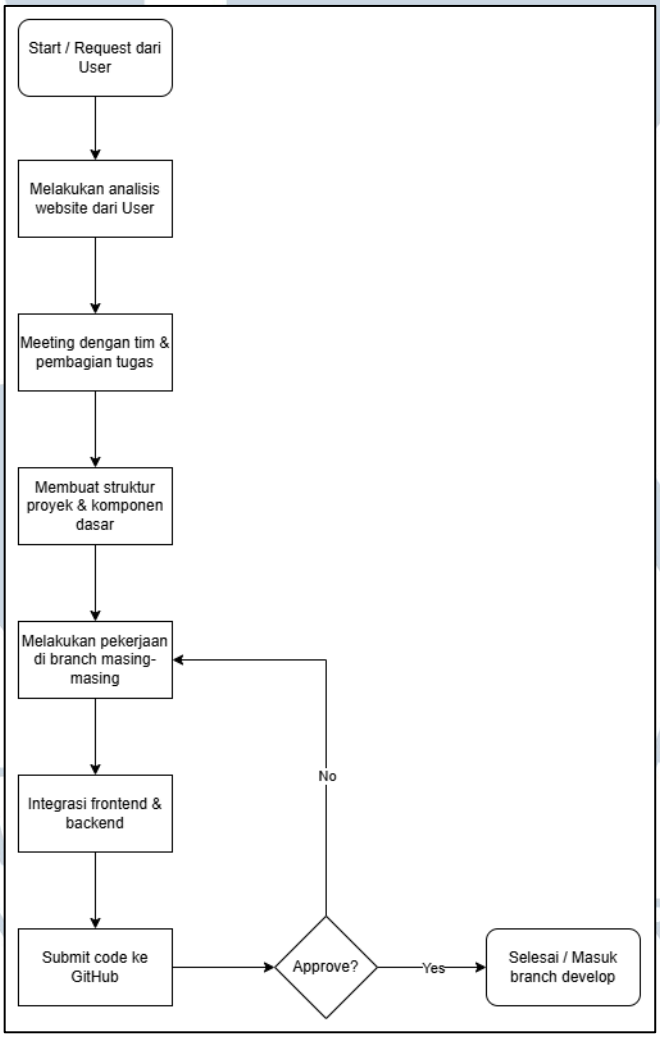
1. *Framework Next.js* versi 14.2.3
2. *Tailwind CSS* versi 3.4.4
3. *Strapi CMS* versi 4.24.2
4. *GitHub Desktop* versi 3.4.20 (x64)
5. *Visual Studio Code* versi 1.90.0
6. *Chrome DevTools* (*Google Chrome* 140)
7. Sistem Operasi *Windows 11 Pro* versi 24H2

Perangkat Keras yang Digunakan:

1. Perangkat PC Kantor *HP ProDesk 400 G5 MT*
2. *Processor Intel Core i5-8500 CPU @ 3.00 GHz* (6 Core, 6 Logical Processor)
3. *RAM 16 GB*
4. *Storage HDD 512 GB*
5. *GPU integrated* (*onboard dari Intel UHD Graphics*)

3.3 Uraian Pelaksanaan Magang

Gambar 3.1 menunjukkan alur kerja magang yang diterapkan selama pelaksanaan proyek pengembangan *website* di Sinarmas Land. Proses dimulai dari penerimaan *request* atau permintaan dari *user*, kemudian dilanjutkan dengan tahap analisis kebutuhan dan struktur *website* yang ada. Setelah itu, dilakukan *meeting* bersama tim untuk menentukan pembagian tugas dan peran masing-masing anggota. Tahap berikutnya adalah pembuatan struktur proyek dan komponen dasar yang menjadi fondasi pengembangan. Setiap anggota tim kemudian mengerjakan tugas pada *branch* masing-masing di *GitHub* untuk menjaga keteraturan kode. Setelah seluruh bagian selesai, dilakukan proses integrasi antara *frontend* dan *backend*.



Gambar 3.1. Flowchart alur kerja magang

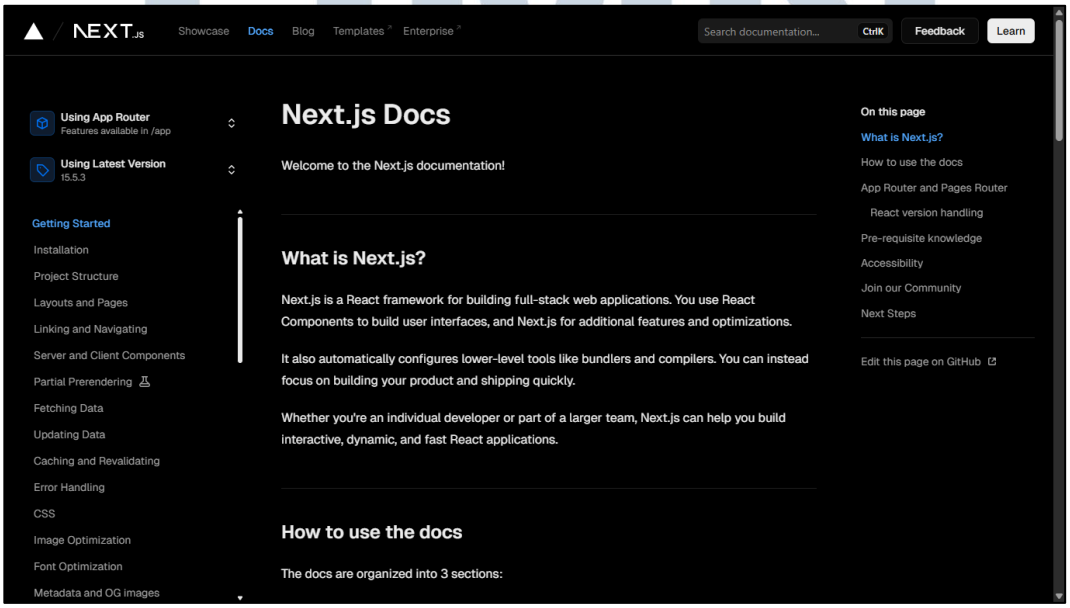
Hasil pekerjaan kemudian dikirimkan (*submit*) ke *GitHub* untuk dilakukan

proses *review* dan *approval*. Jika sudah disetujui, kode akan digabungkan ke dalam *branch develop* sebagai versi final yang siap untuk tahap *deployment*. Dengan alur kerja ini, proses pengembangan dapat berjalan secara kolaboratif, terstruktur, dan efisien.

3.3.1 Pembelajaran Teknologi Dasar

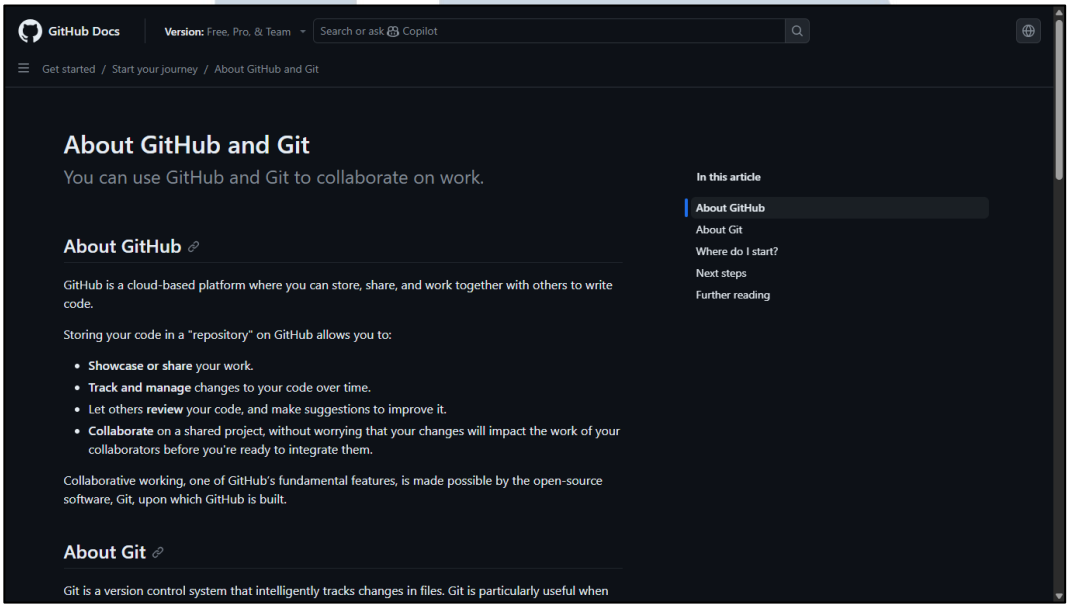
Tahap awal kegiatan magang difokuskan pada pembelajaran mandiri yang bertujuan memperkuat pemahaman terhadap teknologi utama yang akan digunakan dalam proyek pengembangan *website*. Tahap ini menjadi sangat penting karena sebagian besar teknologi yang dipakai, seperti *Next.js*, *Tailwind CSS*, *Strapi CMS*, serta sistem *version control Git* dan *GitHub*. Proses pembelajaran dilakukan secara intensif dengan memanfaatkan berbagai sumber, antara lain dokumentasi resmi, artikel teknis, serta video *tutorial* melalui *platform YouTube*.

Gambar 3.2 merupakan tampilan Dokumentasi resmi *Next.js* menjadi salah satu sumber utama pembelajaran teknologi. Melalui dokumentasi ini, dijelaskan berbagai konsep-konsep dasar seperti *routing*, *server-side rendering* (SSR), *static site generation* (SSG), serta pengelolaan komponen. Dokumentasi ini juga digunakan sebagai acuan praktik langsung sebelum diterapkan pada proyek pengembangan *website.*



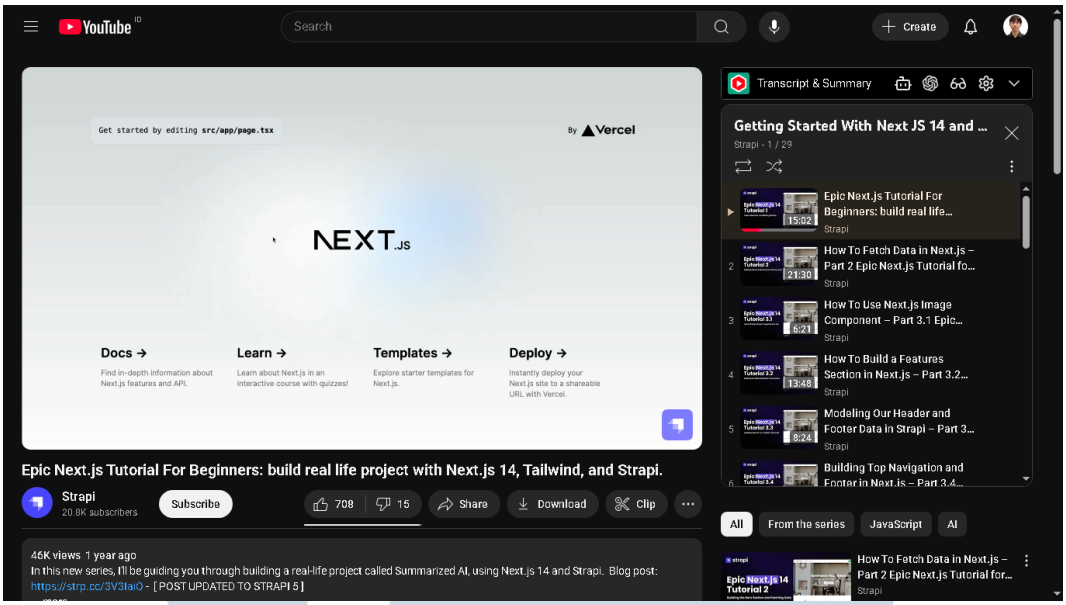
Gambar 3.2. Halaman Dokumentasi Resmi Next.js

Gambar 3.3 adalah halaman dokumentasi resmi *GitHub* digunakan sebagai referensi dalam memahami konsep dasar *version control*. Melalui dokumentasi ini, dijelaskan perbedaan antara *Git* dan *GitHub*, alur kerja kolaborasi tim, serta praktik dasar seperti membuat *repository*, melakukan *commit*, *push*, dan *pull request*. Pemahaman ini menjadi landasan penting untuk berkontribusi dalam pengembangan proyek secara kolaboratif bersama tim.



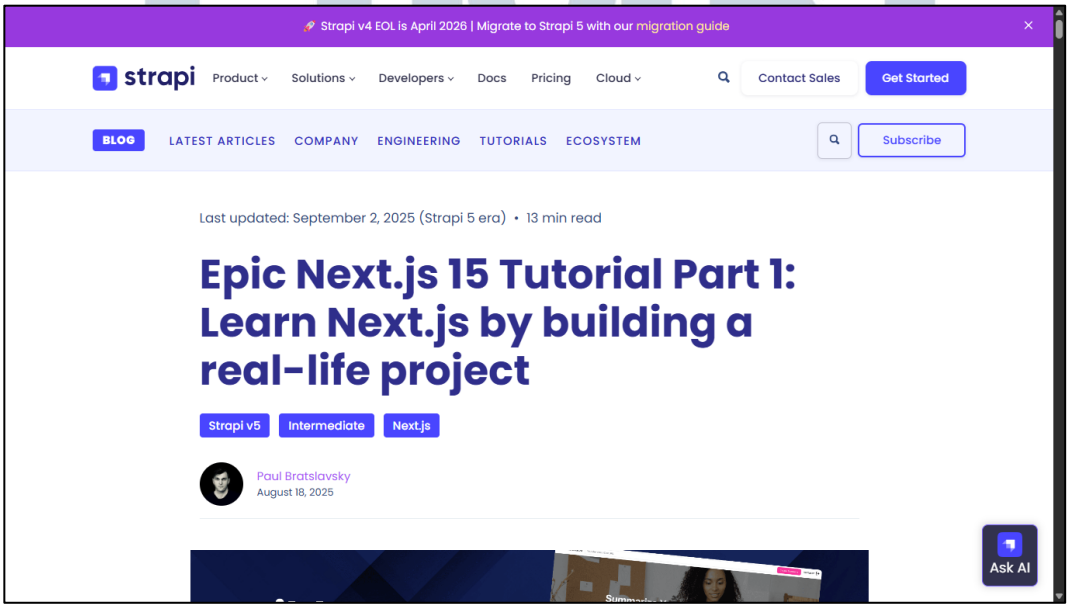
Gambar 3.3. Pengenalan Git dan GitHub dari Dokuementasi Resmi

Gambar 3.4 menunjukkan salah satu materi pembelajaran yang digunakan pada tahap awal magang. Video *tutorial* tersebut berisi penjelasan dasar-dasar *Next.js*, mulai dari struktur proyek, *routing*, hingga implementasi *server-side rendering*. Materi ini berfungsi untuk memperkuat pemahaman terhadap konsep fundamental *framework Next.js* secara lebih praktis melalui studi kasus nyata yang dipandu langsung dalam video.



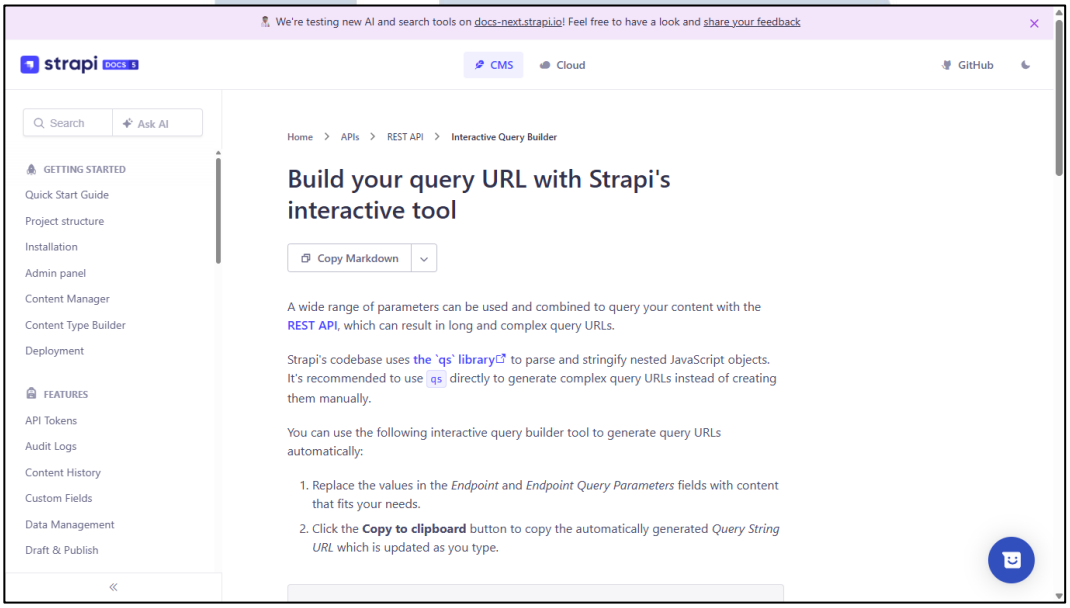
Gambar 3.4. Tutorial Next.js melalui YouTube

Gambar 3.5 menunjukkan artikel *tutorial* pada situs resmi *Strapi* yang dimanfaatkan sebagai referensi praktis untuk mempelajari integrasi *Next.js* dengan *Strapi CMS*. Melalui *tutorial* ini, dijelaskan cara membangun proyek nyata, mulai dari pembuatan *content type* pada *Strapi* hingga proses *fetching* data di *Next.js*. Materi ini berfungsi untuk menghubungkan konsep teoretis dari dokumentasi dengan penerapan langsung dalam pengembangan *website*.



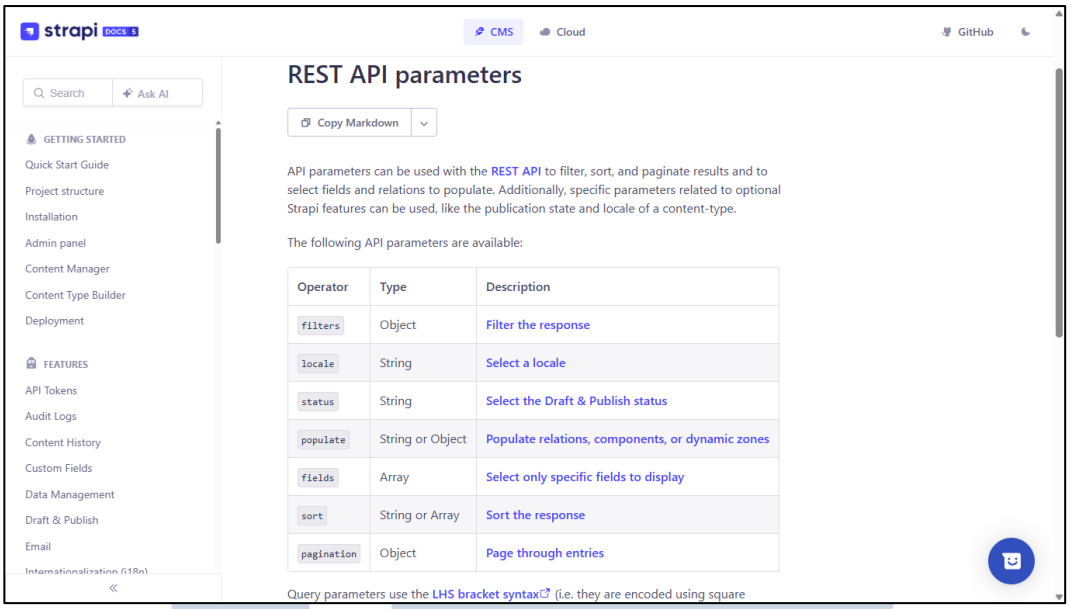
Gambar 3.5. Tutorial Strapi Melalui Proyek Nyata

Gambar 3.6 menunjukkan dokumentasi resmi *Strapi* mengenai REST *API* dan fitur *Interactive Query Builder* yang dimanfaatkan sebagai referensi praktis untuk memahami cara melakukan pengambilan data dari *backend*. Dokumentasi ini menjelaskan *format endpoint*, *parameter query*, serta contoh *response JSON* yang dihasilkan. Pemahaman ini penting karena menjadi dasar dalam proses integrasi *API* antara *Strapi* sebagai *backend* dengan *Next.js* pada sisi *frontend website*.



Gambar 3.6. Dokumentasi Strapi REST API Interactive Query Builder

Gambar 3.7 menunjukkan dokumentasi *Strapi* mengenai *REST API Parameters* yang digunakan untuk mempelajari cara mengatur parameter dalam proses permintaan data, seperti *filters*, *pagination*, *sort*, dan *populate*. Pengetahuan ini berperan penting dalam menampilkan data secara lebih spesifik di halaman *website*, misalnya hanya menampilkan berita terbaru atau memfilter konten tertentu sesuai kebutuhan. Dengan memahami parameter ini, integrasi antara *frontend Next.js* dan *backend Strapi* dapat dilakukan secara lebih optimal.



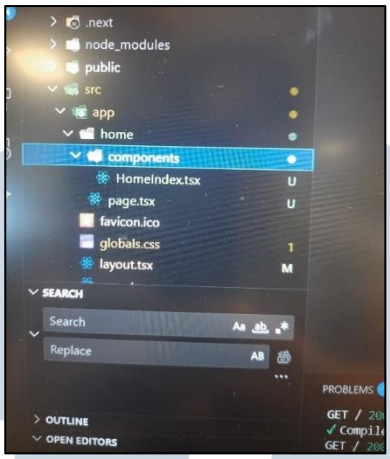
Gambar 3.7. Dokumentasi Strapi REST API Parameters

3.3.2 Analisis dan Pengerjaan Proyek

Setelah tahap pembelajaran teknologi dasar selesai, proses magang dilanjutkan dengan keterlibatan langsung dalam proyek nyata, yaitu *revamp website* Kota Wisata Cibubur. Tahap awal pekerjaan ini dimulai dengan melakukan analisis menyeluruh terhadap *website* lama yang masih menggunakan *WordPress*. Karena tim tidak diberikan akses ke kode sumber *WordPress* sebelumnya, proses *revamp* dilakukan dengan cara meniru langsung tampilan dan struktur dari *website* yang ada, kemudian membangunnya kembali menggunakan teknologi modern yaitu *Next.js* agar hasil akhirnya memiliki tampilan yang sama persis namun dengan performa dan fleksibilitas yang lebih baik.

A. Pembuatan Folder Awal

Gambar 3.8 menunjukkan struktur awal proyek yang dikembangkan menggunakan *framework Next.js*. Pada tahap awal pengerjaan, dilakukan penyusunan struktur folder secara terorganisir agar proses pengembangan *website* lebih mudah dilakukan, terutama karena proyek ini dikerjakan oleh beberapa anggota tim secara bersamaan.



Gambar 3.8. Struktur folder awal pada Visual Studio Code

Struktur folder terdiri dari beberapa bagian utama. Folder *app/* digunakan sebagai pusat dari seluruh halaman (*pages*) dan komponen inti *website*. Di dalam folder ini, setiap *file* atau sub-folder akan merepresentasikan sebuah halaman sesuai dengan arsitektur *App Router* yang diperkenalkan di *Next.js* versi terbaru. Dengan pendekatan ini, pengaturan *routing* menjadi lebih efisien dan konsisten antarhalaman.

Selanjutnya, terdapat folder *components/* yang berfungsi menyimpan berbagai komponen yang dapat digunakan kembali, seperti *header*, *footer*, *navigation bar*, dan *layout* utama. Pemisahan komponen ke dalam folder khusus ini memudahkan proses modularisasi, sehingga kode lebih terstruktur dan mudah dipelihara. Selain itu, praktik ini juga mendukung kolaborasi tim, karena setiap anggota dapat fokus mengerjakan komponen tertentu tanpa mengganggu bagian lain.

Folder *public/* disiapkan sebagai tempat penyimpanan aset statis, misalnya gambar banner, logo, ikon, maupun *file* lain yang mendukung tampilan visual *website*. Semua *file* di dalam folder ini dapat diakses langsung oleh *browser*, sehingga menjadi lokasi ideal untuk konten statis yang tidak perlu diproses ulang di sisi *server*.

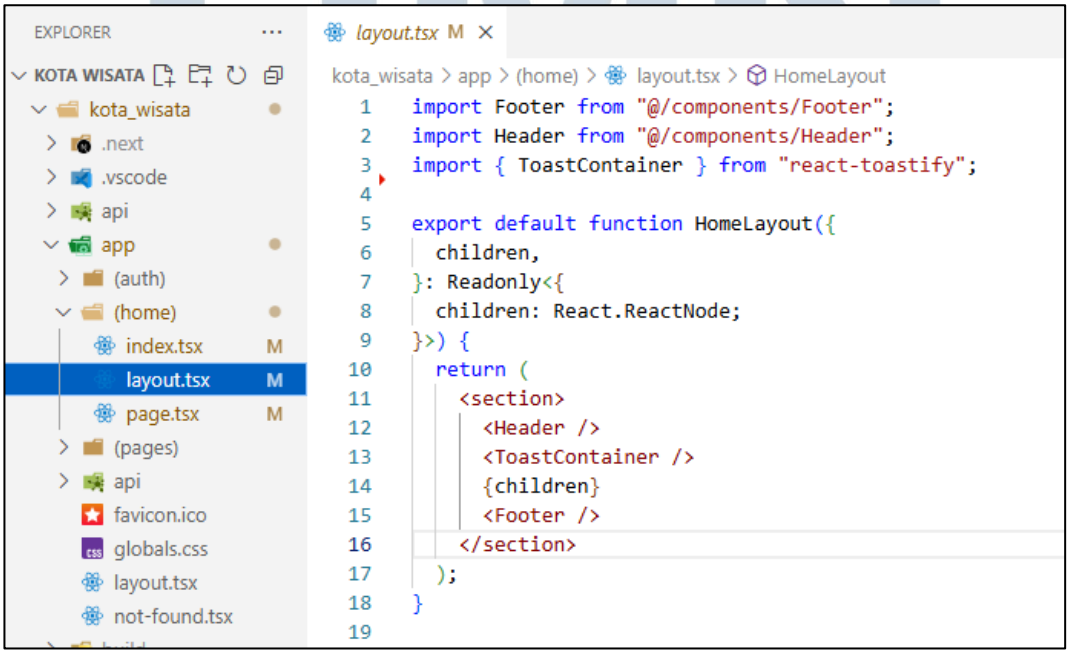
Kemudian terdapat *file globals.css* yang berfungsi sebagai pusat pengaturan *styling* global. Dalam proyek ini, *Tailwind CSS* digunakan untuk mendukung konsistensi desain sekaligus mempercepat proses *styling*. *File* ini memastikan bahwa gaya tampilan seperti warna, tipografi, serta pengaturan *layout* dapat diterapkan secara seragam di seluruh halaman.

Terakhir, *file layout.tsx* berperan sebagai kerangka utama (*main layout*) untuk setiap halaman *website*. Dengan adanya *file* ini, setiap halaman yang dikembangkan akan memiliki struktur tampilan yang konsisten, misalnya selalu menampilkan *header* di bagian atas, konten utama di tengah, dan *footer* di bagian bawah. Konsep *layout* global ini membantu mengurangi duplikasi kode sekaligus menjaga pengalaman pengguna tetap seragam.

B. Pengerjaan HomePage

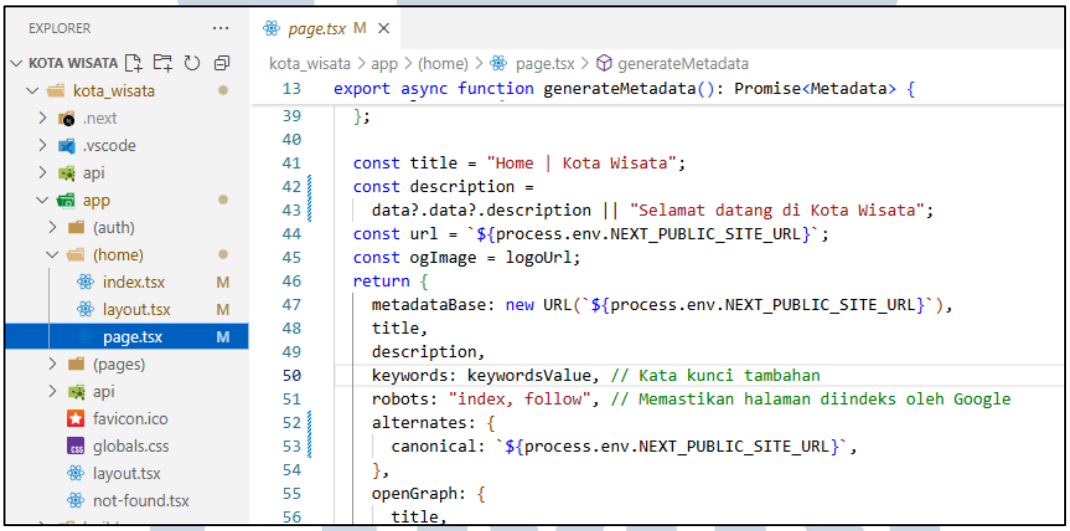
Setelah struktur folder utama selesai dibuat, tahap berikutnya adalah membangun kerangka dasar halaman *website*. Tujuan dari tahap ini adalah memastikan setiap halaman yang dikembangkan nantinya memiliki tampilan konsisten, khususnya pada bagian *header* dan *footer*, serta menyediakan ruang bagi konten utama di setiap halaman.

Kode 3.1 adalah kode *layout.tsx* yang digunakan sebagai kerangka utama yang mengatur struktur global *website*. Pada *file* ini, komponen *Header* dan *Footer* ditempatkan agar muncul secara konsisten di semua halaman. Bagian {*children*} disini berfungsi sebagai wadah untuk menampilkan konten spesifik dari masing-masing halaman. Dengan pendekatan ini, pengembangan halaman menjadi lebih terstruktur dan efisien.



Kode 3.1. Kode Layout Utama pada Folder layout.tsx

Kode 3.2 merupakan kode *page.tsx* yang dibuat sebagai halaman awal (*Home*). Pada tahap ini, halaman *Home* masih bersifat statis, namun sudah dilengkapi dengan konfigurasi *metadata* untuk keperluan *Search Engine Optimization* (SEO). Di dalam *file* ini, diterapkan fitur *generateMetadata* dari *Next.js* untuk menghasilkan judul halaman, deskripsi, kata kunci, hingga *Open Graph tags* dan *Twitter cards* secara otomatis. Bahkan, disertakan juga struktur data *JSON-LD* agar mesin pencari seperti *Google* dapat mengenali identitas *website* Kota Wisata dengan lebih baik. Penerapan *metadata* yang lengkap ini bertujuan untuk memastikan halaman Kota Wisata lebih mudah ditemukan pengguna melalui mesin pencari, sekaligus menampilkan *preview* yang menarik ketika *website* dibagikan melalui *media sosial*.



Kode 3.2. Kode pada Folder page.tsx untuk Halaman Utama

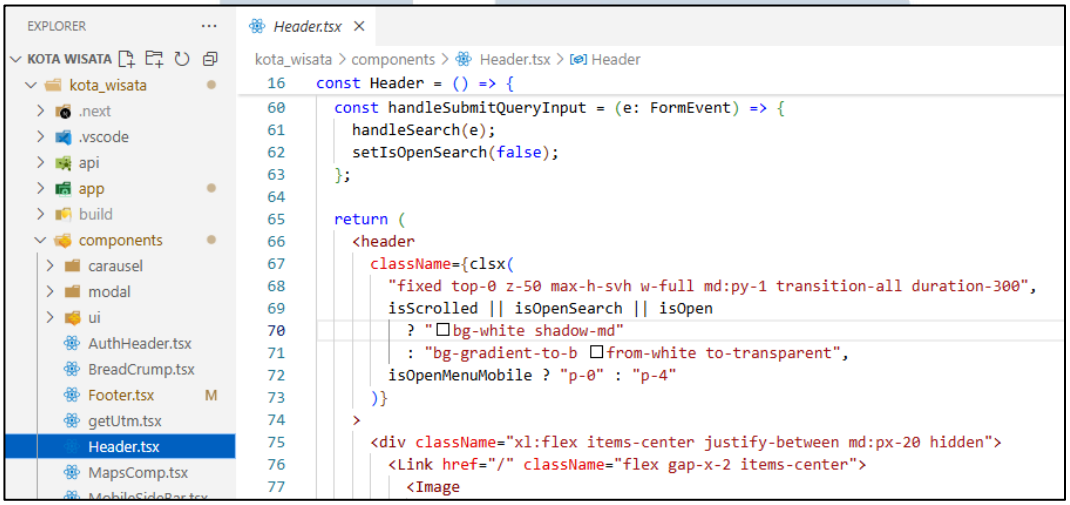
C. Pengerjaan Komponen Global

Setelah struktur dasar halaman utama selesai dibuat, tahap berikutnya adalah mengembangkan komponen global yang digunakan pada seluruh halaman *website*, yaitu *Header* dan *Footer*. Kedua komponen ini bersifat reusable sehingga dapat dipanggil di berbagai halaman tanpa perlu menuliskan ulang kode.

Komponen *Header* berfungsi sebagai navigasi utama *website*. Di dalamnya terdapat elemen logo, menu navigasi, serta *dropdown* bahasa untuk mendukung fitur multibahasa. Implementasi *Header* dirancang responsif dengan memanfaatkan

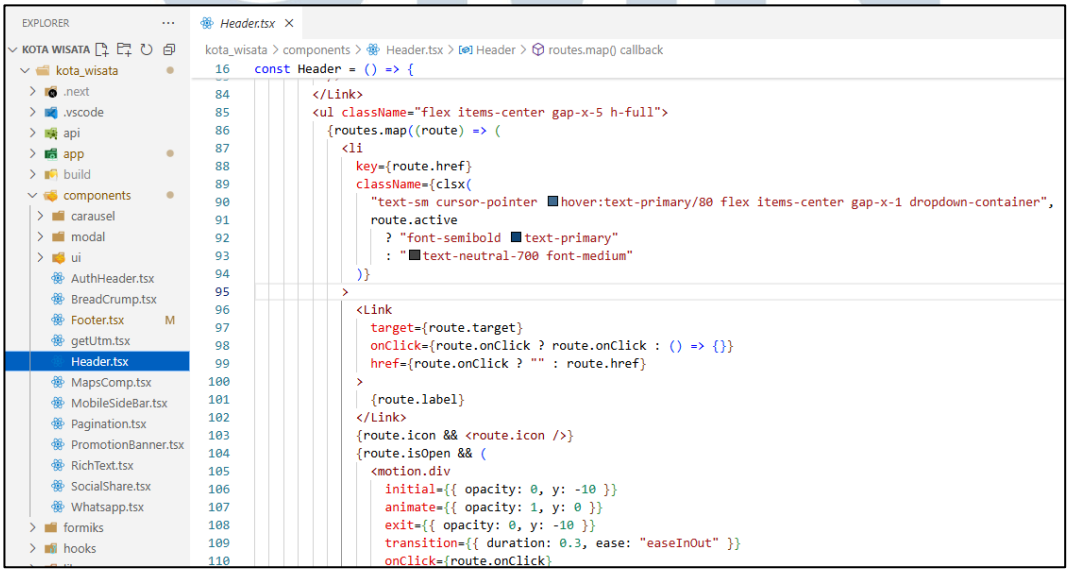
Tailwind CSS, sehingga tampilan navigasi tetap rapi pada layar *desktop* maupun perangkat *mobile*. Selain itu, struktur navigasi dihubungkan dengan sistem *routing Next.js* agar setiap tautan langsung mengarah ke halaman yang sesuai.

Kode 3.3 menunjukkan struktur dasar komponen *Header*. Pada potongan kode ini, *Tailwind CSS* digunakan untuk mengatur posisi tetap (*fixed*), memberikan efek bayangan saat pengguna melakukan *scroll*, serta mengubah gaya *header* secara dinamis sesuai kondisi halaman.



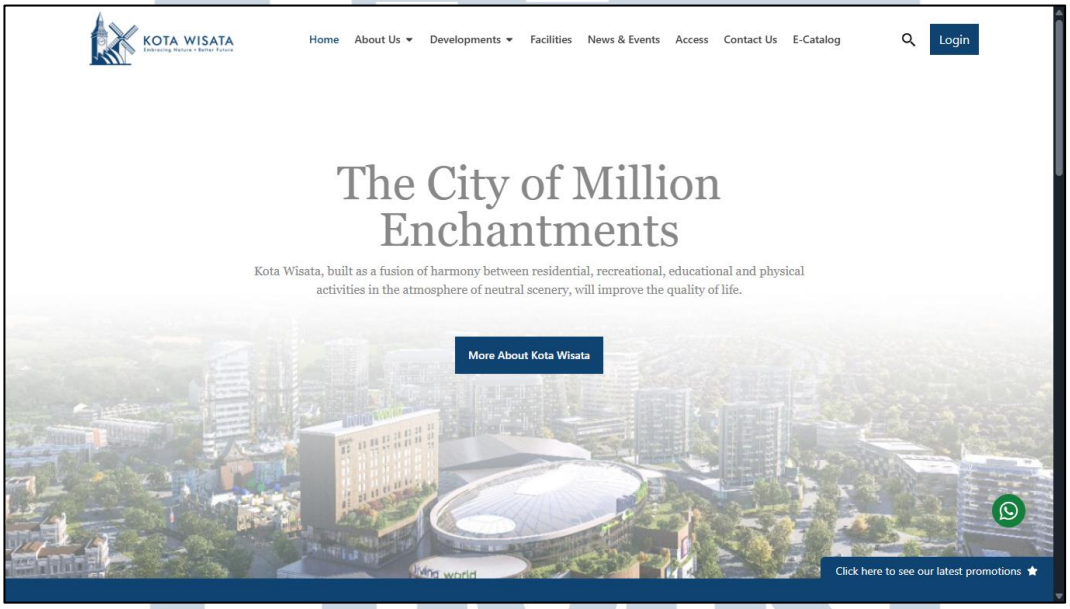
Kode 3.3. Kode Struktur Dasar Komponen Header

Kode 3.4 memperlihatkan bagian navigasi utama pada *Header*. Daftar menu ditampilkan menggunakan perulangan *routes.map()*, sehingga struktur navigasi bersifat dinamis dan mudah diatur. Setiap menu dilengkapi interaksi *hover* dan penanda aktif untuk meningkatkan *user experience*.



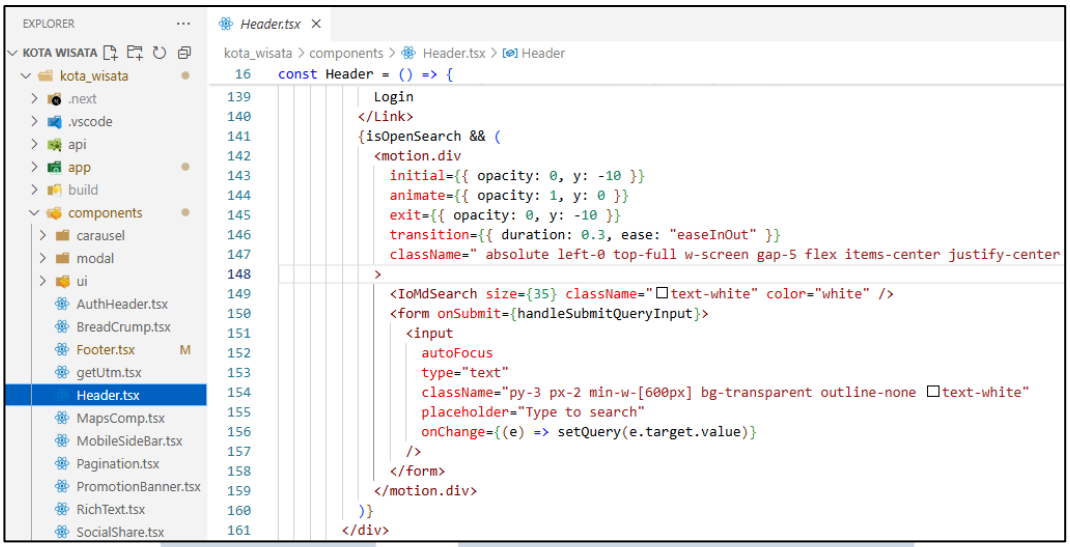
Kode 3.4. Kode Navigasi Menu pada Komponen Header

Gambar 3.9 menunjukkan implementasi efek *hover* pada salah satu menu navigasi, yaitu *Facilities*, yang terdapat di bagian *Header website*. Saat kursor diarahkan ke menu tersebut, warna teks berubah menjadi biru sesuai dengan kelas *Tailwind CSS hover:text-primary/80* yang diterapkan pada elemen. Efek ini memberikan umpan balik visual yang jelas kepada pengguna bahwa menu dapat diklik, sekaligus meningkatkan interaktivitas serta pengalaman pengguna dalam menavigasi halaman *website*.



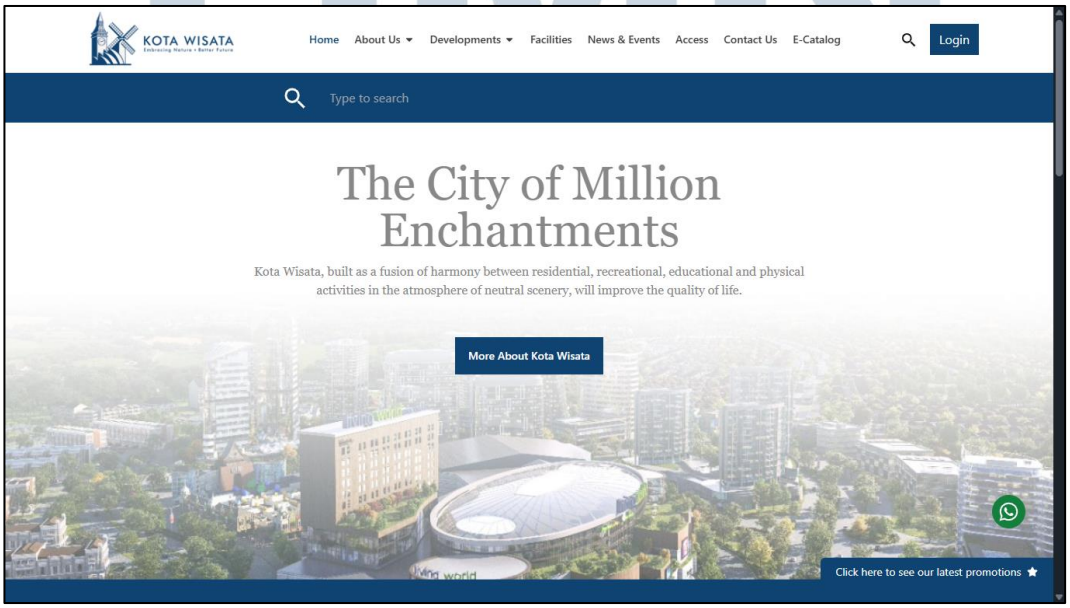
Gambar 3.9. Tampilan efek Hover pada Menu Facilities di Header

Kode 3.5 adalah potongan kode fitur pencarian (*search bar*) yang muncul secara animatif menggunakan *Framer Motion*. Input pencarian ditempatkan secara responsif di bagian atas, sehingga pengguna dapat langsung melakukan pencarian konten di dalam *website*.



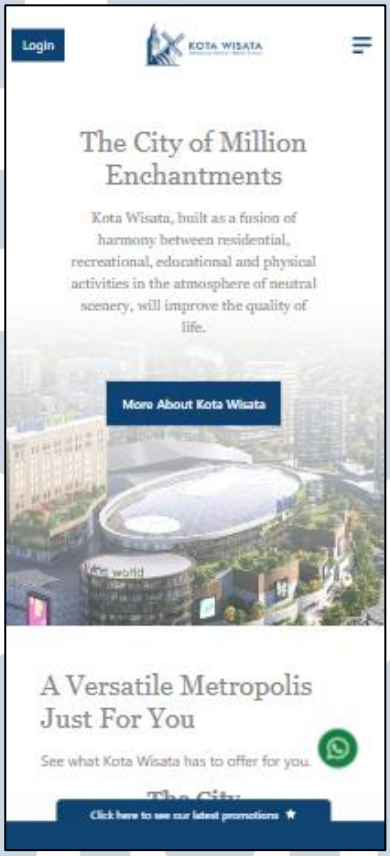
Kode 3.5. Kode Implementasi Fitur Search Bar pada Komponen Header

Gambar 3.10 memperlihatkan fitur *search bar* yang muncul ketika ikon pencarian ditekan. Elemen ini diimplementasikan menggunakan *motion.div* dari *Framer Motion*, sehingga tampilan *search bar* muncul dengan transisi halus dari atas ke bawah. Kolom input memiliki properti *autoFocus* agar pengguna dapat langsung mengetik tanpa harus mengklik kotak teks terlebih dahulu. *Styling* komponen menggunakan *Tailwind CSS* dengan latar belakang berwarna biru tua (*bg-primary*) dan teks berwarna putih untuk menjaga konsistensi dengan identitas visual *website* Kota Wisata. Fitur ini memudahkan pengguna dalam melakukan pencarian konten secara cepat dan intuitif, serta meningkatkan aksesibilitas *website*.



Gambar 3.10. Tampilan Search Bar pada Header

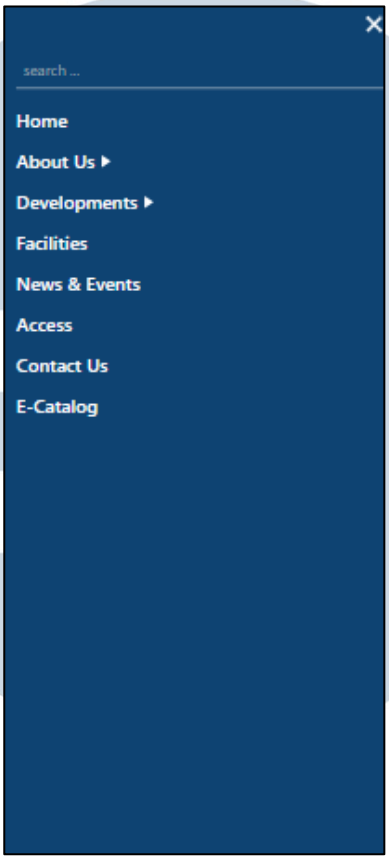
Gambar 3.11 memperlihatkan tampilan *header website* dalam mode *mobile* sesuai hasil implementasi kode pada *file Header.tsx*. Pada mode ini, elemen-elemen *header* disusun lebih ringkas agar sesuai dengan ukuran layar yang lebih kecil. Tombol *Login* ditampilkan di sisi kiri dengan ukuran yang lebih kecil dibandingkan versi *desktop*, sedangkan logo Kota Wisata ditempatkan di bagian tengah agar tetap menjadi fokus utama pengguna.



Gambar 3.11. Tampilan Header dalam mode Mobile

Gambar 3.12 memperlihatkan tampilan *Mobile Sidebar* yang muncul setelah pengguna menekan ikon menu (☰) pada *header* versi *mobile*. *Sidebar* ini ditampilkan secara *overlay* dari sisi kanan layar, berisi daftar menu navigasi utama yang sama dengan versi *desktop*, seperti *Home*, *About Us*, *Developments*, dan menu lainnya. Implementasi *sidebar* ini menggunakan komponen *MobileSideBar* yang dipanggil di dalam *Header.tsx*. Saat *state isOpenMenuMobile* bernilai *true*, *sidebar* akan ditampilkan dengan animasi transisi yang halus, sehingga pengalaman pengguna terasa lebih interaktif. Kelebihan dari penggunaan *sidebar* ini adalah navigasi tetap lengkap namun tidak mengganggu tata letak utama, karena hanya

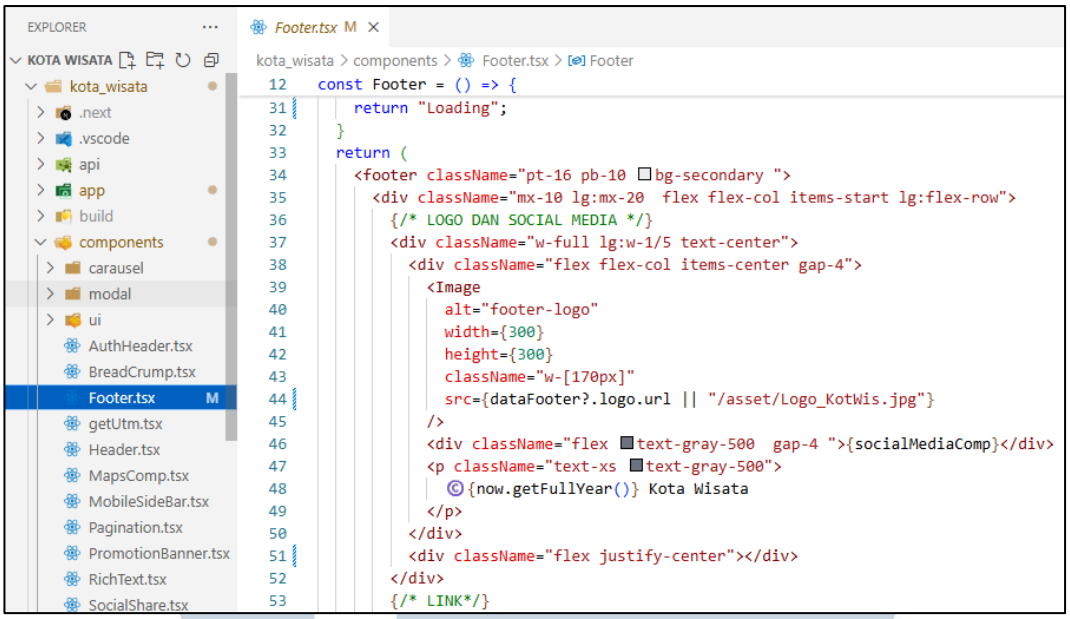
muncul ketika dibutuhkan. Dengan pendekatan ini, antarmuka *website* tetap responsif, efisien, dan ramah pengguna *mobile*.



Gambar 3.12. Tampilan Mobile Sidebar

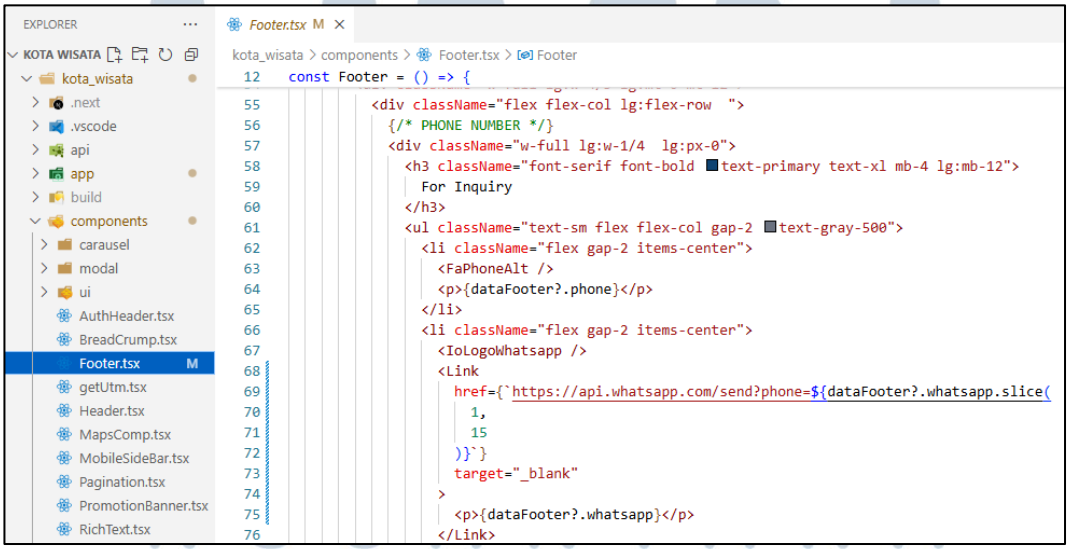
Sementara itu, komponen *Footer* ditempatkan di bagian bawah halaman *website* dan berfungsi sebagai penutup setiap halaman. *Footer* berisi informasi penting mengenai perusahaan, tautan menuju *media sosial* resmi, serta navigasi tambahan. *Footer* juga dikembangkan menggunakan pendekatan responsive design agar konten tetap proporsional di berbagai ukuran layar.

Kode 3.6 memperlihatkan potongan kode yang digunakan untuk menampilkan logo perusahaan beserta ikon *media sosial*. Logo diambil secara dinamis dari *API dataFooter*, sementara ikon sosial media dibangkitkan melalui fungsi *toIconSosmed*. Ikon tersebut ditampilkan dalam bentuk *link* yang dapat langsung mengarahkan pengguna ke *platform media sosial* resmi Kota Wisata. Pada bagian bawah logo juga ditampilkan teks hak cipta (© 2025 Kota Wisata), yang di-*generate* secara otomatis menggunakan fungsi *Date.*



Kode 3.6. Kode Implementasi Logo dan Media Sosial pada Footer

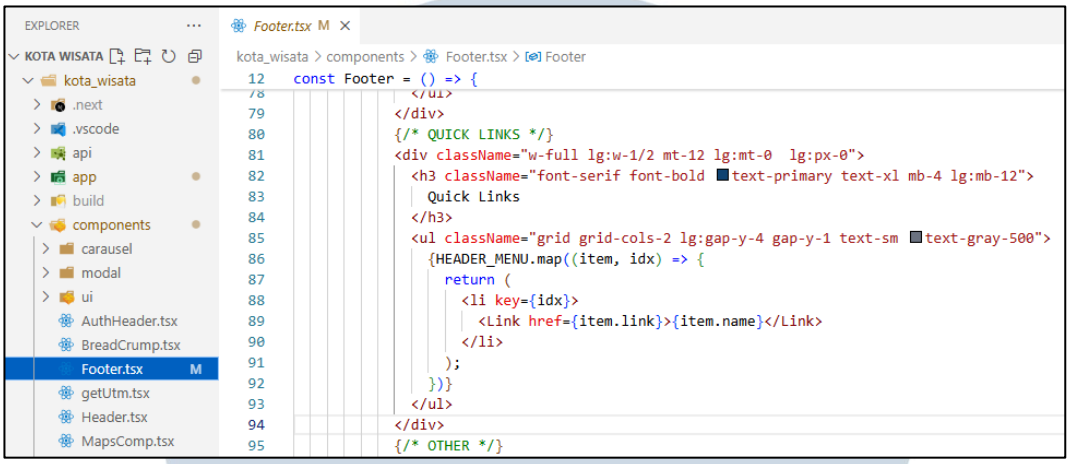
Kode 3.7 memperlihatkan potongan kode yang digunakan untuk menampilkan nomor telepon serta *WhatsApp*. Ikon telepon (*FaPhoneAlt*) dan *WhatsApp* (*IoLogoWhatsapp*) digunakan untuk memperjelas informasi kontak. Nomor *WhatsApp* tidak hanya ditampilkan sebagai teks, tetapi juga dibuat menjadi tautan aktif menggunakan *API WhatsApp*. Dengan begitu, pengunjung *website* dapat langsung memulai percakapan hanya dengan satu kali klik.



Kode 3.7. Kode Implementasi Kontak Telepon dan WhatsApp pada Footer

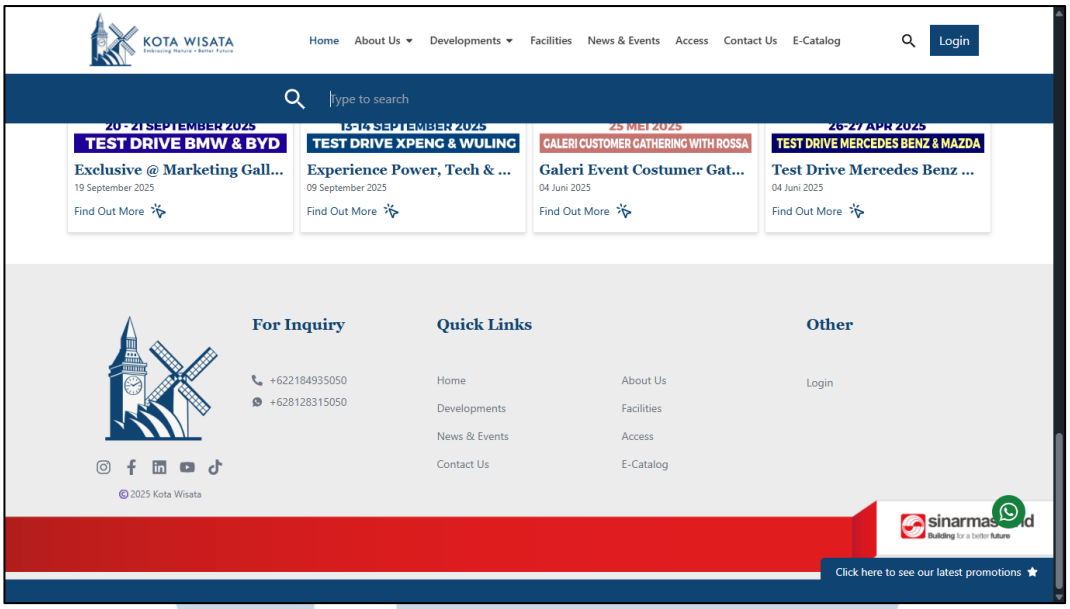
Kode 3.8 adalah potongan kode *Quick Links* yang berfungsi untuk menampilkan tautan cepat menuju halaman-halaman penting di dalam *website*. Dengan pendekatan ini, daftar tautan pada *footer* akan selalu sinkron dengan menu

utama di *header*, sehingga memudahkan pemeliharaan kode. Tautan ditampilkan dalam bentuk grid dengan dua kolom, sehingga rapi dan mudah dibaca.



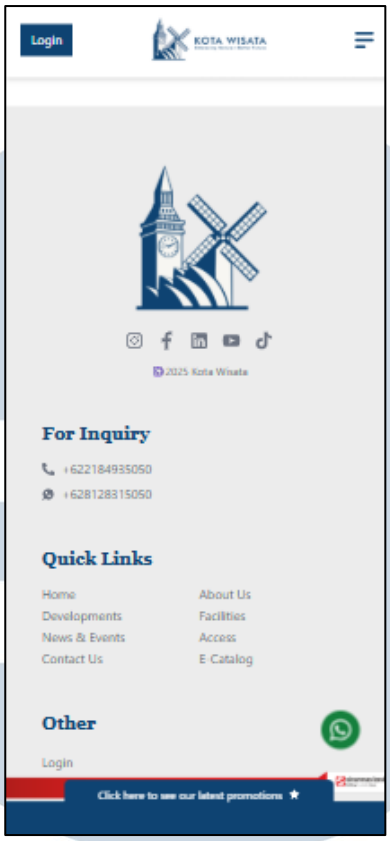
Kode 3.8. Kode Implementasi Quick Links pada Footer

Gambar 3.13 menampilkan *footer website* Kota Wisata yang dihasilkan dari implementasi kode pada *Footer.tsx*. Pada tampilan *desktop*, *footer* tersusun atas beberapa elemen penting. Bagian kiri menampilkan logo Kota Wisata yang ditempatkan bersama ikon *media sosial*, serta teks *copyright* yang menyesuaikan tahun berjalan secara otomatis. Bagian tengah menampilkan daftar tautan cepat atau Quick Links yang mempermudah pengguna mengakses halaman-halaman utama *website*. Sementara itu, di sisi kanan ditampilkan informasi kontak seperti nomor telepon, *WhatsApp* yang terhubung langsung ke aplikasi, serta menu tambahan lain. Di bagian paling bawah terdapat elemen dekoratif berupa gambar *ribbon* yang mempertegas batas antara *footer* dengan konten utama *website*.



Gambar 3.13. Tampilan Footer versi Desktop

Gambar 3.14 menunjukkan tampilan *footer* versi *mobile*, susunan elemen *footer* ditata ulang secara vertikal sehingga lebih mudah dibaca di layar yang lebih kecil. Logo, ikon *media sosial*, daftar tautan, hingga informasi kontak ditampilkan secara berurutan dengan jarak yang cukup lega, sehingga pengguna tetap dapat mengakses semua informasi dengan nyaman. Penataan ini sesuai dengan prinsip *mobile-first design* yang diterapkan dalam proyek, sehingga *footer* dapat berfungsi dengan baik di berbagai perangkat, baik *desktop* maupun *mobile*.



Gambar 3.14. Tampilan Footer versi Mobile

Dengan adanya *Header* dan *Footer* sebagai komponen global, struktur halaman *website* menjadi lebih konsisten dan mudah untuk dikelola. Perubahan pada navigasi atau informasi perusahaan cukup dilakukan pada satu *file* komponen, sehingga akan langsung tercermin di seluruh halaman *website*.

D. Pengerjaan Halaman About Us

Implementasi halaman *About Us* berfungsi untuk menampilkan profil singkat, visi, misi, serta informasi umum mengenai proyek Kota Wisata Cibubur. Halaman ini merupakan salah satu bagian utama dalam struktur *website* karena menjadi sarana untuk memperkenalkan identitas dan nilai perusahaan kepada pengunjung.

Pengerjaan halaman *About Us* dilakukan dengan pendekatan komponen modular, di mana setiap bagian halaman seperti banner, visi, dan video profil dikembangkan dalam *file* terpisah agar mudah dikelola dan dipelihara. Data pada

halaman ini bersifat dinamis, diambil langsung dari *backend Strapi CMS* melalui pemanggilan *API*, sehingga perubahan konten dapat dilakukan dengan mudah tanpa perlu melakukan modifikasi kode di *frontend*.

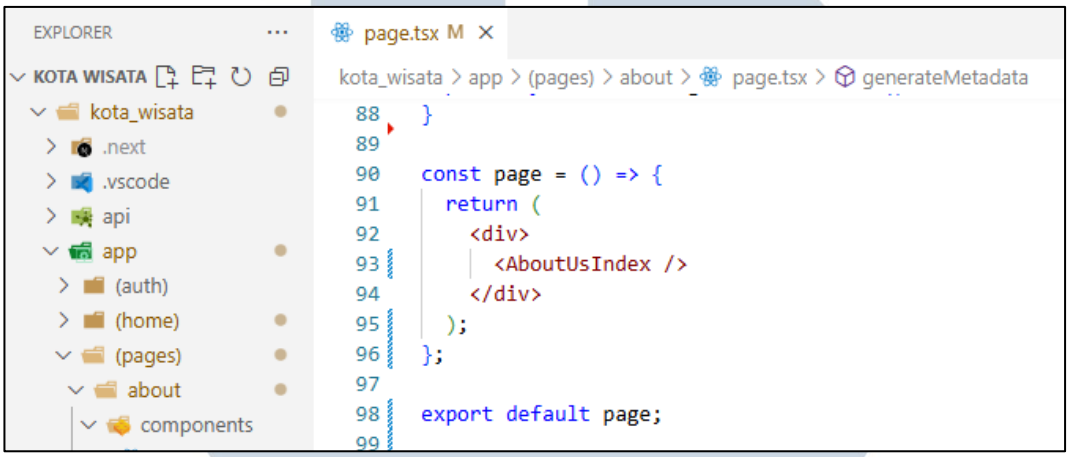
Selain itu, halaman ini juga dilengkapi dengan pengaturan *metadata* otomatis menggunakan fitur *generateMetadata* dari *Next.js*. Pengaturan *metadata* ini bertujuan untuk mendukung optimasi mesin pencari (*SEO*) dengan menampilkan judul, deskripsi, kata kunci, serta gambar pratinjau (*Open Graph* dan *Twitter Card*) secara dinamis sesuai data yang diterima dari *backend*.

Kode 3.9 merupakan implementasi fungsi *generateMetadata* pada halaman *About Us*. Fungsi ini digunakan untuk menghasilkan *metadata* halaman secara dinamis berdasarkan data yang diambil dari *API getAboutUs*(). Data yang diperoleh mencakup judul halaman, deskripsi, kata kunci (*keywords*), serta gambar banner. Informasi ini kemudian digunakan untuk mengatur elemen *metadata* seperti *Open Graph*, *Twitter Card*, dan *structured data* (*JSON-LD*) guna meningkatkan optimasi mesin pencari (*SEO*). Selain itu, *metadata* juga mengatur elemen seperti robots: "*index*, *follow*" agar halaman *About Us* dapat diindeks oleh mesin pencari. Pendekatan dinamis ini memastikan setiap perubahan konten pada *backend Strapi* langsung tercermin di *metadata* tanpa perlu melakukan pembaruan manual di *frontend*.

Kode 3.9. Kode generateMetadata pada Halaman About Us

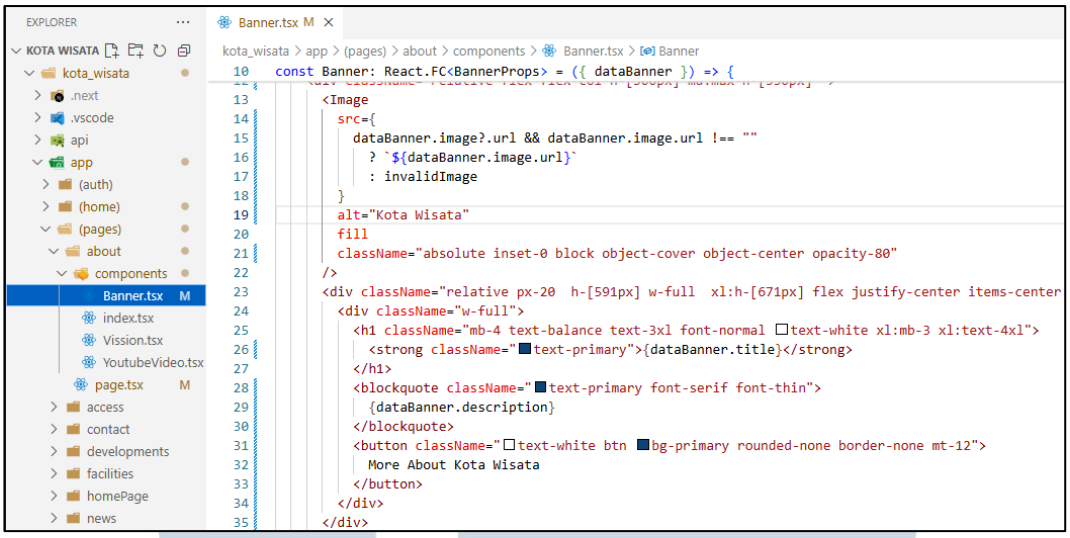
Kode 3.10 menunjukkan struktur dasar halaman *About Us* yang memanggil komponen utama *AboutUsIndex*. Komponen ini berfungsi sebagai wadah (*wrapper*) yang menggabungkan seluruh elemen pembentuk halaman, seperti Banner, *Vission*,

dan *Youtube* Video. Pendekatan ini mengikuti prinsip modularitas pada *React* dan *Next.js*, di mana setiap bagian halaman dikembangkan sebagai komponen terpisah agar lebih mudah dikelola, diuji, serta digunakan kembali di halaman lain jika diperlukan.



Kode 3.10. Kode Struktur Dasar Halaman About Us

Kode 3.11 menampilkan implementasi komponen Banner yang digunakan sebagai elemen utama pada bagian atas halaman *About Us*. Komponen ini memuat elemen gambar latar belakang, judul, deskripsi, serta tombol "*More About Kota Wisata*". Gambar latar diambil dari properti *dataBanner.image.url* yang berasal dari data *API*, dengan pengaturan tampilan menggunakan kelas *Tailwind CSS* seperti *object-cover* dan *opacity-80* untuk menjaga proporsionalitas serta estetika tampilan. Judul dan deskripsi ditampilkan secara dinamis berdasarkan data yang dikirim melalui *props dataBanner*, sedangkan tombol interaktif pada bagian bawah berfungsi sebagai ajakan untuk melihat informasi lebih lanjut mengenai Kota Wisata. Secara keseluruhan, komponen ini memberikan kesan visual yang kuat sekaligus memperkuat identitas halaman *About Us* melalui penggunaan gambar dan tipografi yang konsisten dengan tema *website*.



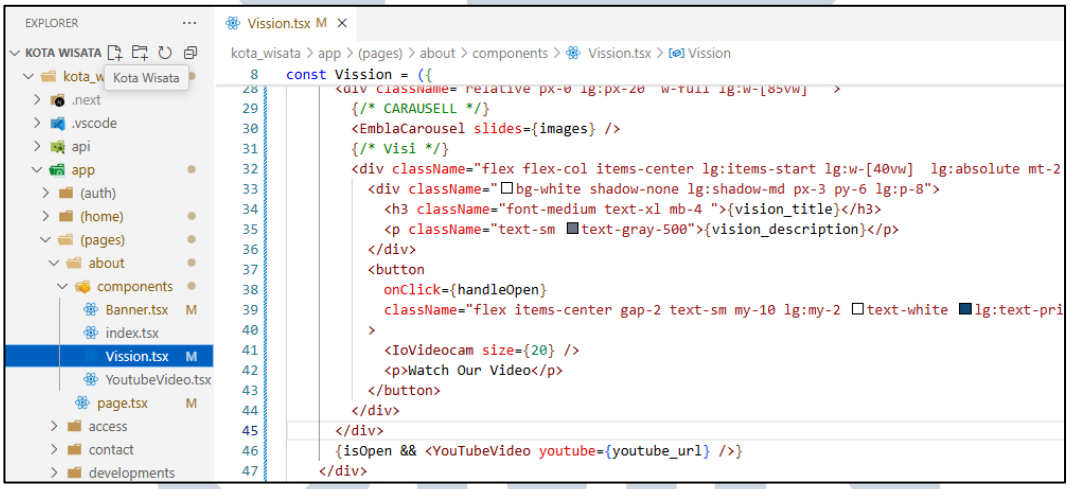
Kode 3.11. Kode Implementasi Komponen Banner pada Halaman About Us

Gambar 3.15 memperlihatkan tampilan banner halaman *About Us* yang menampilkan latar belakang visual dengan teks judul dan deskripsi. Elemen ini menjadi fokus utama halaman karena memuat informasi pengantar mengenai profil Kota Wisata.

Gambar 3.15. Tampilan Banner pada Halaman About Us

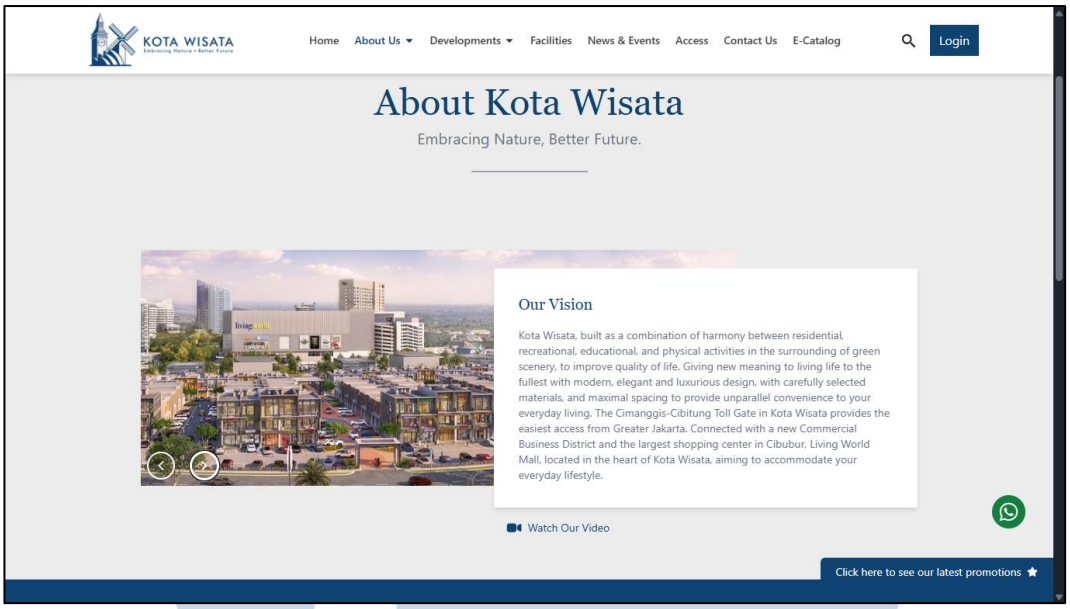
Kode 3.12 menampilkan implementasi komponen *Vission*, yang berfungsi untuk menampilkan visi perusahaan melalui kombinasi elemen visual dan multimedia. Bagian utama komponen ini terdiri atas *carousel* gambar dan deskripsi visi, yang diambil secara dinamis dari data *API* menggunakan properti

CarouselImage, *vision_title*, dan *vision_description*. Carousel diimplementasikan menggunakan komponen *EmblaCarousel*, yang memungkinkan pengguna melihat beberapa gambar secara bergulir (*sliding*). Selain itu, komponen ini memiliki tombol interaktif bertuliskan "*Watch Our Video*" yang dihubungkan dengan fungsi *handleOpen*. Saat tombol ditekan, *state isOpen* akan berubah menjadi *true*, dan elemen *YouTube* Video akan muncul untuk menampilkan video profil Kota Wisata langsung dari *URL* yang dikirimkan melalui properti *youtube_url*. Tata letak dan warna pada bagian ini diatur dengan *Tailwind CSS*, menggunakan kombinasi latar belakang berwarna putih dan primer (biru), serta bayangan lembut (*shadow*) untuk memberikan kesan modern dan profesional. Secara keseluruhan, komponen *Vission* berfungsi tidak hanya sebagai penyaji informasi visi perusahaan, tetapi juga sebagai elemen interaktif yang memperkuat daya tarik visual serta memberikan pengalaman pengguna yang lebih hidup dan informatif.



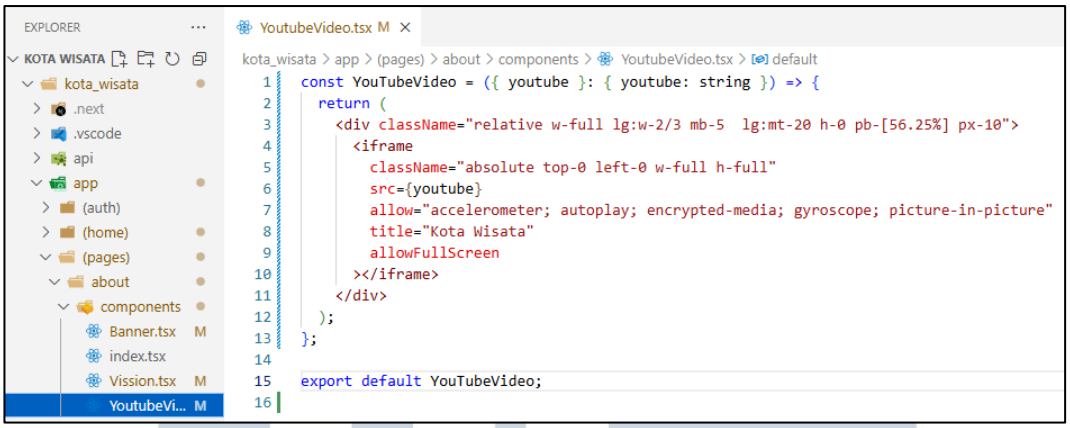
Kode 3.12. Kode Implementasi Komponen Vission pada Halaman About Us

Gambar 3.16 memperlihatkan tampilan komponen *Vission* yang terdiri dari *carousel* gambar dan informasi visi Kota Wisata. Bagian kanan menampilkan teks visi beserta tombol "*Watch Our Video*" yang dapat diklik untuk membuka video profil perusahaan. Komponen ini memberikan nuansa interaktif dan informatif yang menonjol dalam halaman *About Us*.



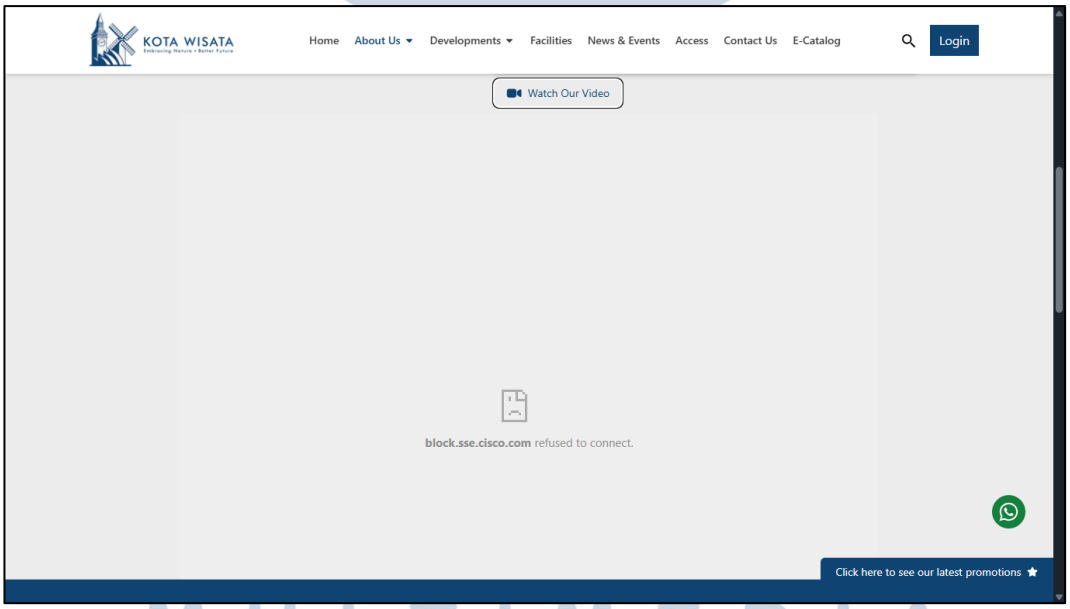
Gambar 3.16. Tampilan Komponen Vission pada Halaman About Us

Kode 3.13 menampilkan implementasi komponen *YouTube* Video, yang berfungsi untuk menampilkan video profil perusahaan secara langsung dari platform *YouTube*. Komponen ini menerima sebuah properti bernama *youtube* berupa tautan (*URL*) video, yang kemudian dimasukkan ke dalam elemen <*iframe*>. Elemen <*iframe*> digunakan untuk menampilkan video secara *embed*, sehingga pengguna dapat menonton video tanpa harus meninggalkan halaman website. Atribut seperti *allowFullScreen* mengizinkan tampilan video dalam mode layar penuh, sementara *allow* memberikan izin bagi berbagai fitur seperti *autoplay* dan *gyroscope*. Dari sisi tata letak, *Tailwind CSS* digunakan untuk menjaga proporsi aspek rasio video agar tetap 16:9 dengan pendekatan pb-[56.25%] (*padding-bottom*), serta memosisikan elemen video secara *absolute* di dalam wadah responsif. Pendekatan ini memastikan bahwa video akan tampil secara optimal baik pada layar *desktop* maupun perangkat *mobile*. Komponen ini biasanya ditampilkan setelah pengguna menekan tombol "*Watch Our Video*" pada bagian *Vission*, yang mengubah *state* menjadi aktif sehingga video muncul secara dinamis di halaman.



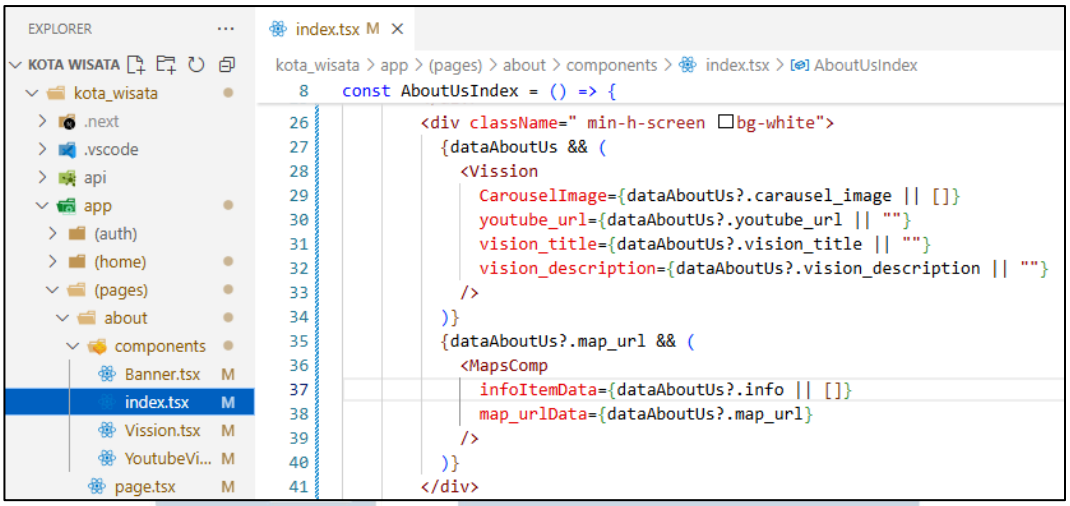
Kode 3.13. Kode Implementasi YouTube Video pada Halaman About Us

Gambar 3.17 memperlihatkan tampilan komponen *YouTube* Video yang menampilkan video profil Kota Wisata secara langsung dari *YouTube*. Video ini muncul setelah tombol "*Watch Our Video*" ditekan pada bagian *Vission*. Komponen ini meningkatkan daya tarik visual halaman dan memberikan informasi perusahaan dalam bentuk multimedia yang menarik.



Gambar 3.17. Tampilan YouTube Video pada Halaman About Us

Kode 3.14 merupakan bagian utama dari komponen *index.tsx* yang mengatur susunan tampilan halaman *About Us*. Komponen *Vission* dan *MapsComp* dipanggil secara terintegrasi untuk menampilkan konten visi, video, dan lokasi proyek Kota Wisata. Data diambil dari *Strapi* melalui *hook useGetAboutUs()* sehingga seluruh konten bersifat dinamis.

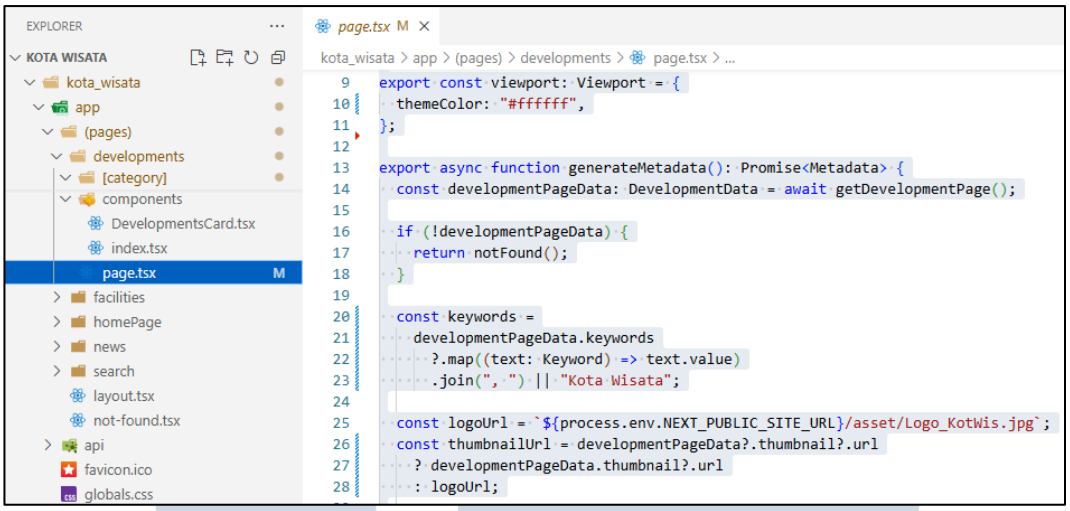


Kode 3.14. Kode Integrasi Keseluruhan Halaman About

E. Pengerjaan Halaman Developments

Halaman *Developments* berfungsi untuk menampilkan seluruh proyek perumahan dan komersial yang ada di Kota Wisata. Halaman ini menampilkan daftar kategori seperti *Residential*, *Commercial*, serta detail tiap *cluster* di dalamnya. Implementasi halaman ini menggunakan pendekatan *dynamic routing* pada *Next.js* agar setiap kategori dan proyek dapat diakses melalui *URL* yang berbeda tanpa harus membuat banyak *file* secara manual.

Kode 3.15 berfungsi untuk menghasilkan *metadata SEO* secara otomatis menggunakan fungsi *generateMetadata()* bawaan *Next.js*. *Metadata* ini mencakup *title*, *description*, *keywords*, serta konfigurasi *Open Graph* dan *Twitter Card*. Semua informasi tersebut diambil dari *endpoint API getDevelopmentPage()*, sehingga konten yang ditampilkan di mesin pencari akan selalu sesuai dengan data terbaru dari *backend Strapi*. Pendekatan ini membantu meningkatkan *search engine visibility* halaman *Developments* tanpa perlu melakukan pembaruan manual di sisi *frontend*.



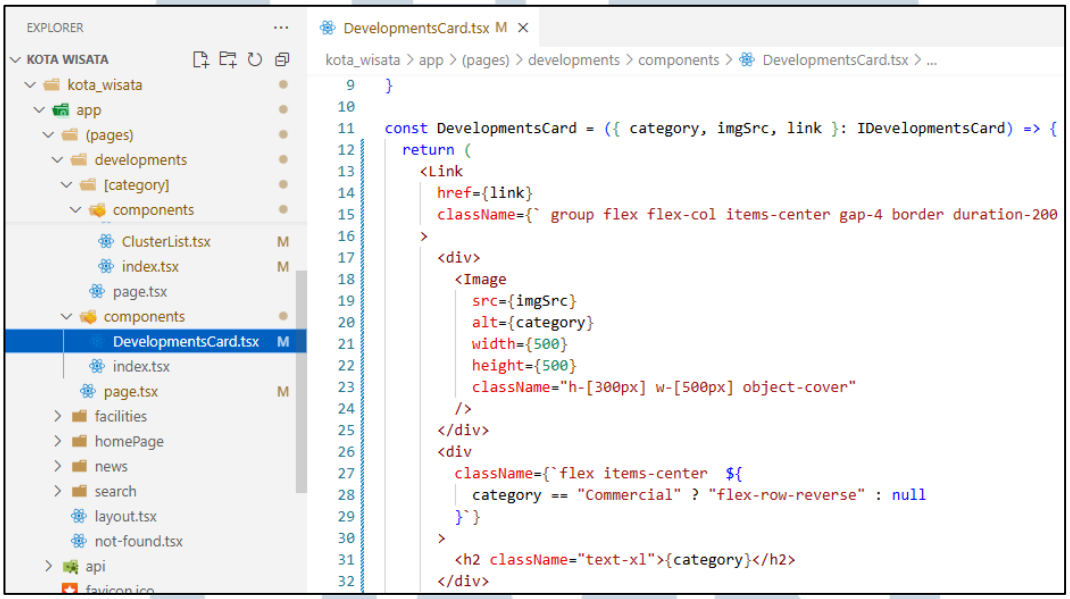
Kode 3.15. Kode Halaman Utama Developments

Kode 3.16 merupakan struktur utama tampilan halaman *Developments*. Data judul dan deskripsi halaman diambil secara langsung dari hasil pemanggilan *API getDevelopmentPage()*. Komponen *DevelopmentsIndex* kemudian dipanggil untuk menampilkan daftar kategori proyek yang tersedia. *Styling* halaman diatur menggunakan *Tailwind CSS*, dengan tata letak berbasis *flexbox* agar tampilan tetap responsif baik di layar *desktop* maupun perangkat *mobile*. Pendekatan ini memastikan halaman memiliki struktur yang konsisten dan mudah dikembangkan lebih lanjut.

Kode 3.16. Kode Struktur Dasar Halaman Developments

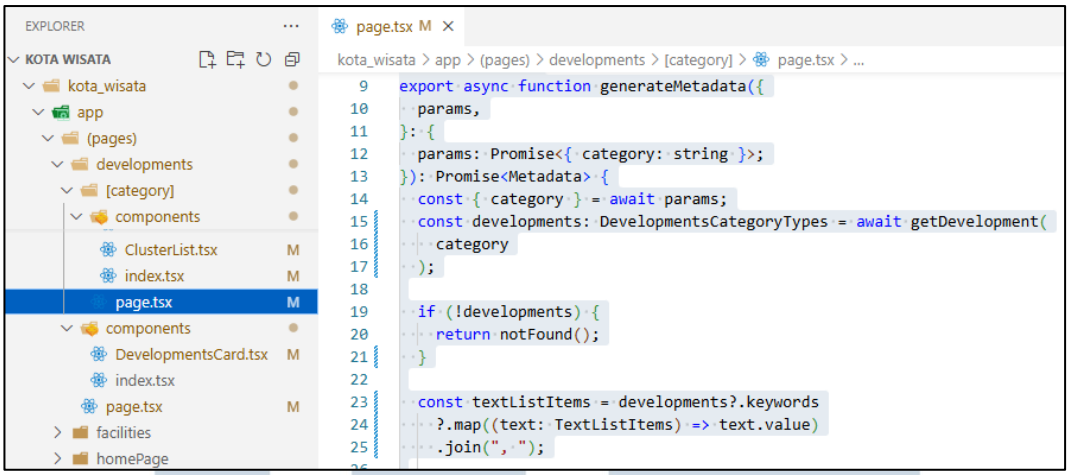
Kode 3.17 menampilkan kartu (*card*) untuk setiap kategori pengembangan yang tersedia di Kota Wisata, seperti *Residential* dan *Commercial*. Komponen ini menggunakan elemen *Link* dari *Next.js* agar setiap kartu dapat langsung mengarahkan pengguna ke halaman kategori yang sesuai. Setiap kartu terdiri dari

gambar dan teks nama kategori yang diatur secara responsif menggunakan *Tailwind CSS*. Efek *hover shadow* (*hover:shadow-md*) diterapkan untuk memberikan kesan interaktif saat pengguna mengarahkan kursor ke kartu. Selain itu, terdapat logika kondisional *category == "Commercial" ? "flex-row-reverse" : null* yang berfungsi untuk membalikkan posisi elemen teks dan gambar jika kategori yang ditampilkan adalah *Commercial*, sehingga tata letak antar kategori terlihat lebih dinamis. Dengan adanya komponen ini, tampilan daftar pengembangan (*Developments*) menjadi lebih terstruktur, interaktif, dan menarik secara visual bagi pengguna.



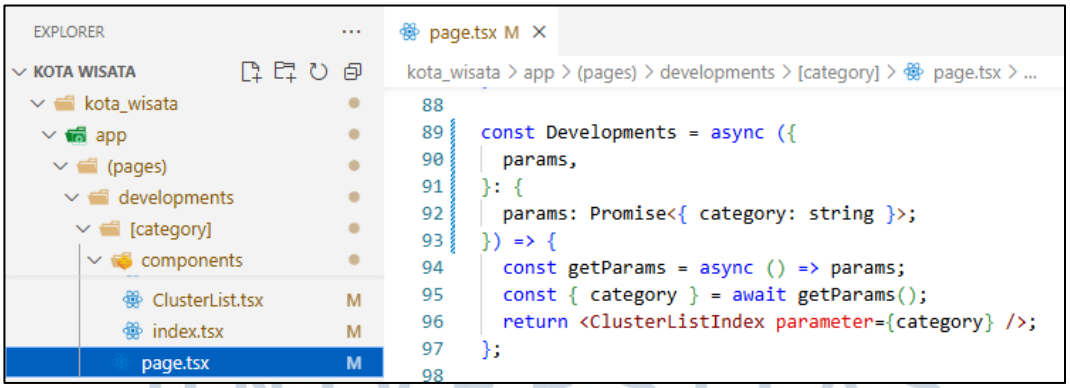
Kode 3.17. Kode Komponen DevelopmentsCard

Kode 3.18 menunjukkan proses pembuatan *metadata* dinamis untuk setiap halaman kategori *Developments*. Fungsi *generateMetadata()* merupakan fitur bawaan dari *Next.js* yang memungkinkan pengaturan elemen *SEO* seperti *title*, *description*, dan *keywords* secara otomatis berdasarkan data yang diambil dari *API Strapi*. Kode ini memanggil fungsi *getDevelopment(category)* untuk mengambil data spesifik sesuai kategori — seperti *residential*, *commercial*, atau lainnya — dan kemudian mengolah nilai *keywords* menjadi *string* yang akan digunakan pada *metadata* halaman. Jika data tidak ditemukan, halaman akan menampilkan respon *notFound()* agar tidak menimbulkan *error* saat proses *rendering*.



Kode 3.18. Kode Metadata Dinamis pada Halaman Developments

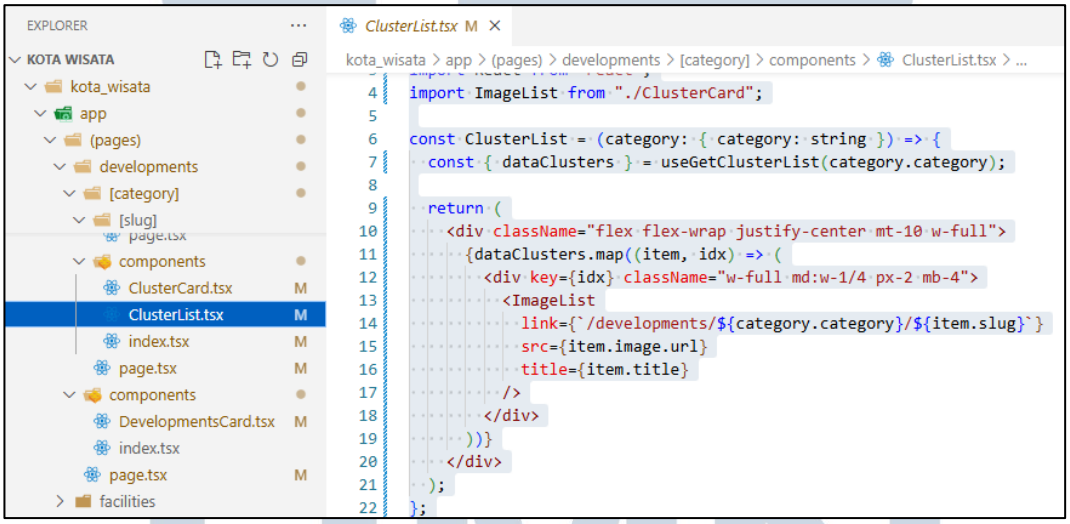
Kode 3.19 merupakan bagian utama dari struktur halaman kategori *Developments*. Bagian ini memanggil komponen *ClusterListIndex* dengan parameter kategori yang diterima dari *URL* dinamis, misalnya */developments/residential* atau */developments/commercial*. Dengan pendekatan ini, setiap halaman kategori dapat menampilkan data *cluster* sesuai dengan kategori yang dipilih, tanpa perlu membuat *file* halaman terpisah untuk setiap jenis pengembangan.



Kode 3.19. Kode Struktur Halaman Kategori Developments

Kode 3.20 merupakan bagian utama dari komponen *ClusterList*, yang bertanggung jawab untuk menampilkan daftar *cluster* perumahan atau komersial berdasarkan kategori yang dipilih pengguna di halaman *Developments*. Komponen ini menggunakan *custom hook useGetClusterList(category.category)* untuk mengambil data *cluster* dari *backend Strapi CMS* sesuai dengan kategori yang sedang aktif — misalnya *residential* atau *commercial*. Setelah data berhasil diambil,

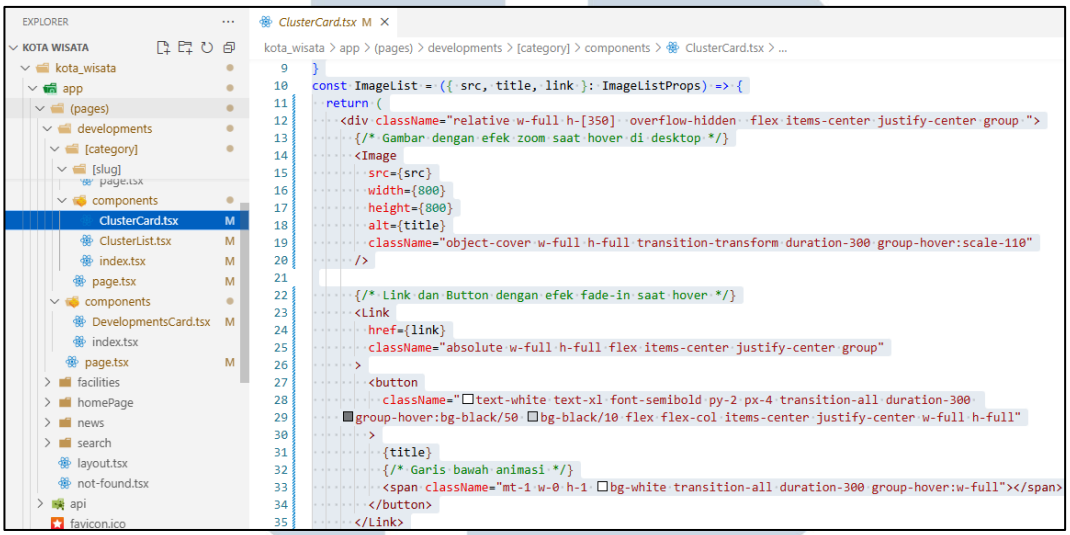
fungsi *map()* digunakan untuk menampilkan setiap *item cluster* secara dinamis. Setiap *cluster* ditampilkan dalam bentuk komponen *ImageList*, yang berisi gambar, judul, serta tautan menuju halaman *detail cluster* (*/developments/{category}/{slug}*). Tata letak komponen menggunakan *utility classes* dari *Tailwind CSS*, seperti *flex*, *flex-wrap*, dan *justify-center*, agar tampilan daftar *cluster* tersusun responsif dalam beberapa kolom, baik pada tampilan *desktop* maupun *mobile*. Pendekatan modular ini membuat struktur kode menjadi lebih efisien dan mudah untuk dikembangkan kembali. Jika suatu saat ada kategori baru di *Strapi*, sistem akan secara otomatis menampilkan datanya tanpa perlu menambahkan halaman baru secara manual.



Kode 3.20. Kode Pengambilan dan Penampilan Data Cluster

Kode 3.21 merupakan inti dari komponen *ImageList*, yang berfungsi untuk menampilkan setiap kartu cluster pada halaman *Developments*. Komponen ini menggabungkan elemen visual dan interaktif menggunakan *Next.js Image Optimization* (*<Image />*) serta *Next.js Link Routing* (*<Link />*) agar gambar dapat dimuat secara efisien dan navigasi menuju halaman detail *cluster* berjalan mulus tanpa *reload*. Tata letak komponen dikendalikan dengan *utility classes* dari *Tailwind CSS*, seperti *relative*, *overflow-hidden*, dan *group-hover*, untuk menghasilkan efek animasi yang menarik. Saat pengguna mengarahkan kursor ke gambar (*hover*), terjadi efek *zoom in* halus pada gambar (*group-hover:scale-110*) serta *fade-in* pada tombol *overlay* berisi judul *cluster*. Efek visual ini memperkuat *user experience* dengan cara memberikan kesan interaktif dan modern, sesuai

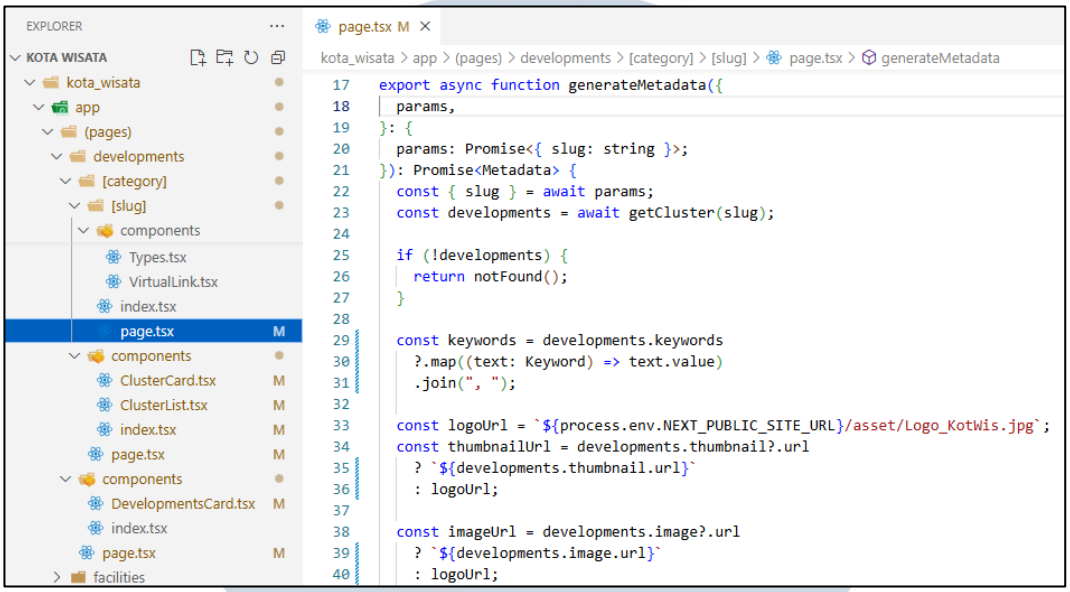
dengan karakteristik *website* korporat milik Sinarmas Land. Dengan pendekatan modular ini, satu komponen *ImageList* dapat digunakan berulang kali untuk seluruh daftar *cluster* di berbagai kategori, sehingga pengembangan menjadi lebih efisien, konsisten, dan mudah dikelola.



Kode 3.21. Kode Komponen Kartu Cluster pada Halaman Developments

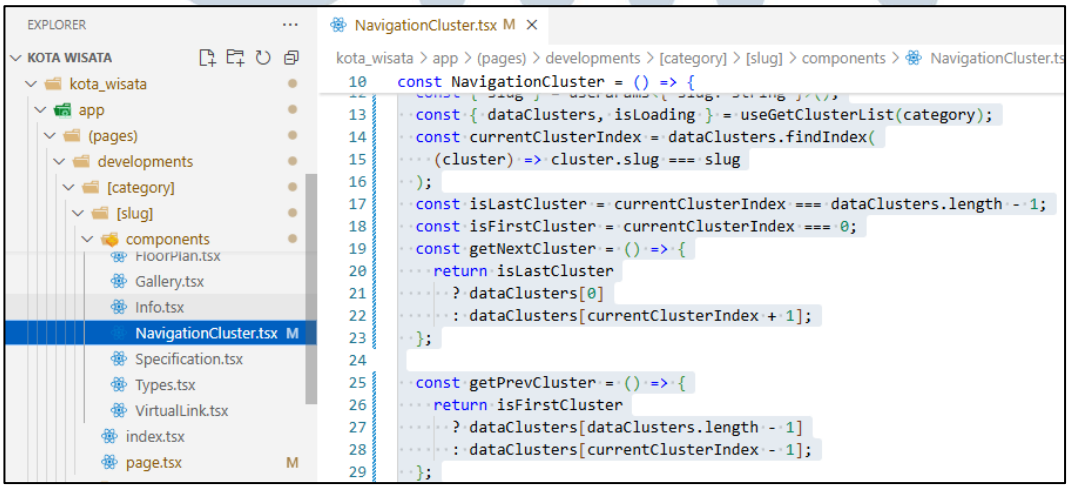
Kode 3.22 merupakan implementasi fungsi *generateMetadata()* pada *file /developments/[slug]/page.tsx*, yang berfungsi untuk mengatur *metadata* halaman detail *cluster* secara dinamis berdasarkan data yang diperoleh dari *API backend Strapi* melalui fungsi *getCluster(slug)*. Pendekatan ini memanfaatkan fitur bawaan *Next.js 14 App Router*, yang memungkinkan setiap halaman memiliki *metadata* (seperti *title*, *description*, dan *keywords*) berbeda sesuai isi konten tanpa harus didefinisikan manual. Data seperti judul proyek (*developments.title*), deskripsi (*developments.description*), dan kata kunci (*keywords*) diambil langsung dari *response API Strapi*. Nilai-nilai ini kemudian dimasukkan ke dalam konfigurasi *Open Graph* dan *Twitter Card*, agar ketika halaman dibagikan ke media sosial (misalnya *WhatsApp*, *Facebook*, atau *Twitter*), akan muncul *preview card* berisi gambar dan deskripsi proyek secara otomatis. Selain itu, kode juga membangkitkan skema *JSON-LD* (melalui properti *other: { "script:ld+json": ... }*) yang mempermudah mesin pencari seperti *Google* untuk memahami struktur halaman. Dengan demikian, halaman detail *cluster* tidak hanya informatif bagi pengguna, tetapi juga *SEO-friendly* dan siap diindeks oleh mesin pencari secara optimal. Pendekatan dinamis ini menjadi kunci penting dalam proyek *revamp website* Kota

Wisata, karena setiap *cluster* memiliki konten unik yang perlu ditampilkan dan dioptimalkan secara individual.



Kode 3.22. Kode Metadata Dinamis pada Halaman Detail Cluster

Kode 3.23 digunakan untuk menentukan posisi *cluster* yang sedang dibuka oleh pengguna. Fungsi *findIndex()* mencari posisi cluster aktif berdasarkan *parameter slug*. Selanjutnya, dua variabel logika — *isLastCluster* dan *isFirstCluster* — digunakan untuk mendeteksi apakah *cluster* tersebut berada di urutan pertama atau terakhir. Jika pengguna berada di cluster terakhir, maka tombol *next* akan mengarahkan kembali ke cluster pertama, dan sebaliknya untuk tombol *previous*. Pendekatan ini memastikan navigasi bersifat *looping* dan tidak berhenti di ujung daftar, sehingga pengalaman pengguna lebih lancar.

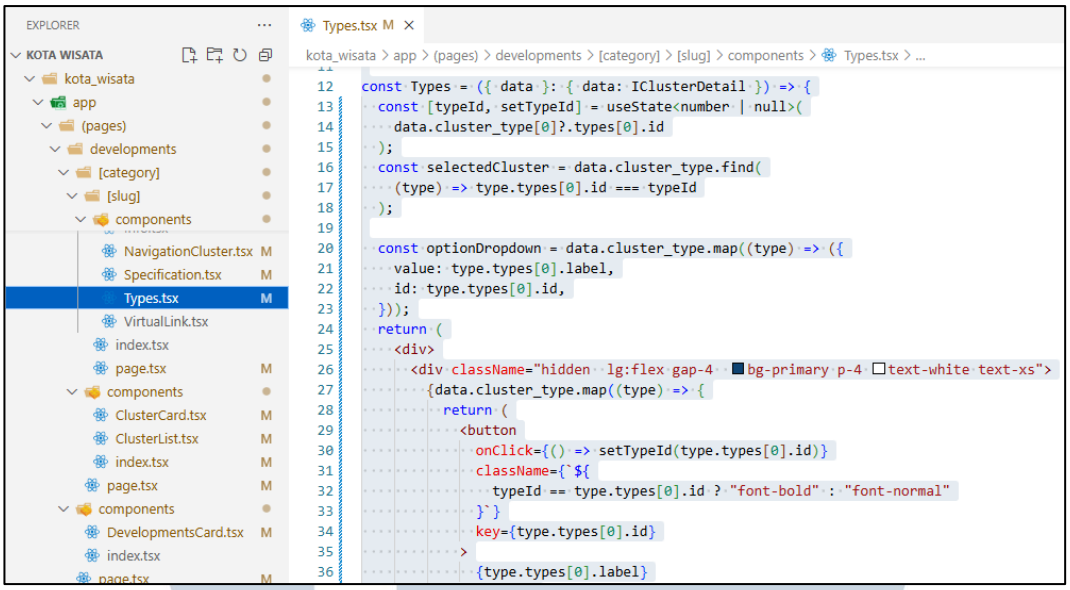


Kode 3.23. Kode Implementasi Komponen Navigasi Antar Cluster

Kode 3.24 berfungsi untuk menampilkan daftar spesifikasi dari masing-masing *cluster*. Data spesifikasi diambil secara dinamis dari properti *spec* yang bertipe *array* dan kemudian dirender menggunakan metode *map()*. Setiap elemen berisi pasangan label dan nilai, misalnya "Luas Bangunan: 120 m²" atau "Kamar Tidur: 3". Selain itu, pada bagian awal daftar terdapat judul cluster (*title*) dan tipenya (*type*), misalnya "*Cluster Amethyst – Residential*". Struktur tampilan diatur menggunakan *Tailwind CSS*, dengan kombinasi grid *layout* pada layar besar dan *flexbox* pada layar kecil agar tetap responsif. Penggunaan kelas seperti *font-serif* dan *text-primary* menjaga konsistensi gaya visual dengan elemen lain di *website*.

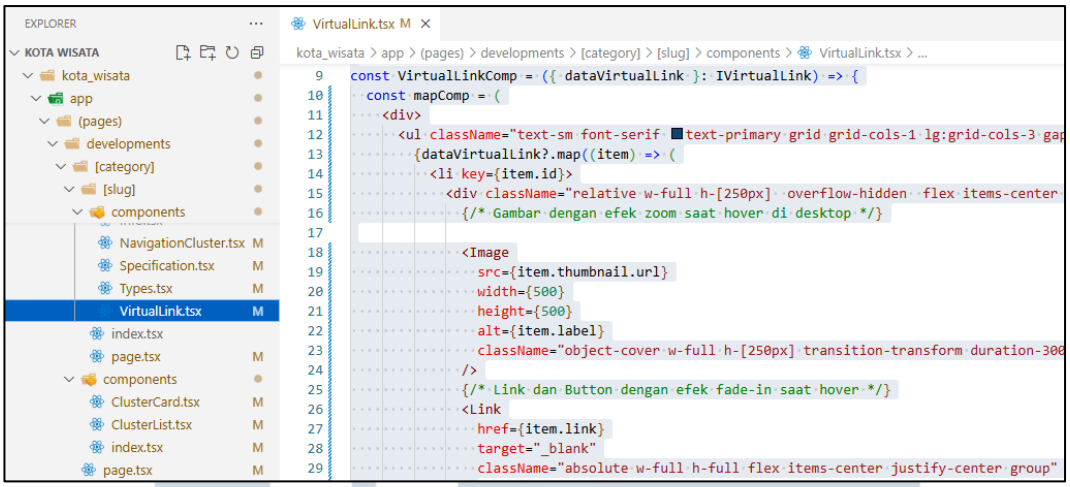
Kode 3.24. Kode Implementasi Komponen Specification.tsx

Kode 3.25 merupakan inti dari komponen *Types.tsx* yang berfungsi menampilkan informasi detail setiap tipe unit rumah dalam halaman *cluster* pada menu *Developments*. Kode ini menggunakan *state typeId* untuk menentukan tipe rumah mana yang sedang dipilih, sehingga konten halaman dapat berubah secara dinamis sesuai interaksi pengguna. Bagian navigasi tipe rumah ditampilkan dalam bentuk deretan tombol pada tampilan *desktop*, di mana setiap tombol akan memperbarui nilai *typeId* saat diklik. Dengan cara ini, data tipe yang aktif — seperti gambar galeri, video promosi, tur virtual, dan spesifikasi bangunan — akan diperbarui tanpa perlu memuat ulang halaman. Komponen *Gallery*, *ClusterVideo*, *VirtualLinkComp*, dan *SpecificationPage* masing-masing menampilkan data visual dan informasi yang diperoleh dari *backend Strapi*, sedangkan tampilan dan tata letaknya dikendalikan dengan bantuan *Tailwind CSS* agar tetap responsif dan konsisten di seluruh perangkat. Pendekatan ini membuat tampilan halaman detail *cluster* menjadi interaktif, informatif, dan efisien, karena setiap elemen hanya akan dirender bila data yang relevan tersedia dari *API*.



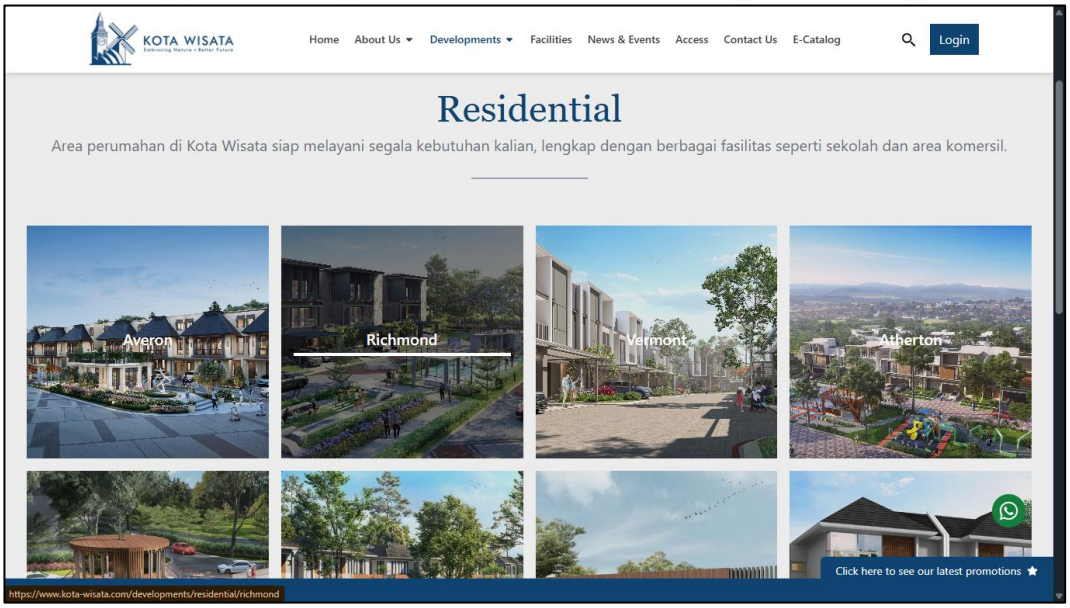
Kode 3.25 Potongan kode komponen Types.tsx

Kode 3.26 menampilkan implementasi utama dari komponen *Virtual Link* yang berfungsi untuk menampilkan daftar tautan virtual tour setiap *cluster* perumahan pada halaman *Developments*. Setiap item dalam daftar ditampilkan sebagai gambar interaktif menggunakan elemen <*Image*> dari *Next.js* yang diambil secara dinamis dari properti *thumbnail.url*. Efek visual dibuat menggunakan *class Tailwind CSS* seperti *group-hover:scale-110* untuk memberikan animasi *zoom* saat pengguna mengarahkan kursor ke gambar. Selain itu, setiap gambar dibungkus oleh elemen <*Link*> yang mengarahkan pengguna ke halaman tur virtual (tautan eksternal) ketika tombol diklik. Di dalam *link*, terdapat elemen <*button*> transparan yang menampilkan label dari virtual tour dengan efek *fade-in* dan animasi garis bawah putih yang muncul saat *hover*, menambah kesan interaktif pada tampilan. Pendekatan ini tidak hanya membuat tampilan menjadi menarik secara visual, tetapi juga mempertahankan struktur yang responsif dan mudah digunakan di berbagai ukuran layar.



Kode 3.26. Kode struktur Komponen Virtual Link

Gambar 3.18 menampilkan kategori pengembangan kawasan seperti *Residential* dan *Commercial* dalam bentuk kartu interaktif dengan efek *hover zoom*. Tampilan halaman ini dibangun menggunakan *Next.js* dan *Tailwind CSS*, serta terintegrasi dengan *Strapi CMS* agar seluruh konten dapat dikelola secara dinamis. Desainnya responsif dan modern, menampilkan informasi setiap kategori secara jelas serta memudahkan pengguna menavigasi ke halaman detail pengembangan yang diinginkan.

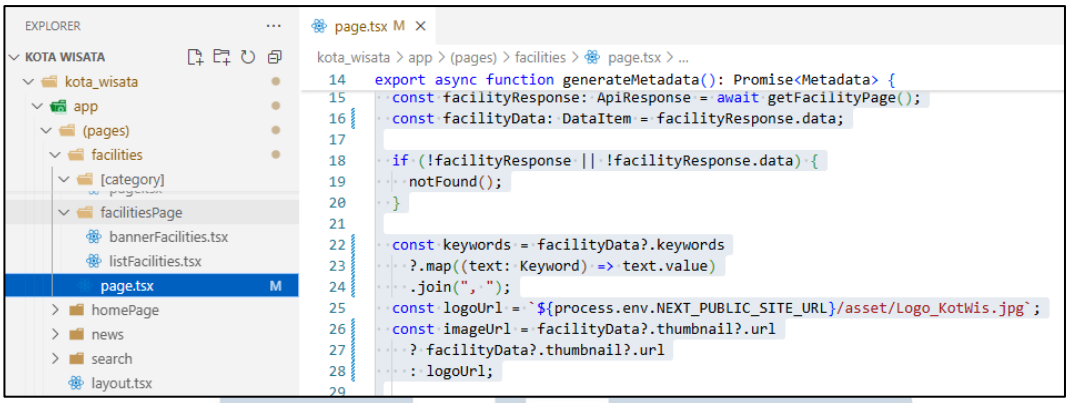


Gambar 3.18. Tampilan Halaman Developments Website Kota Wisata

F. Pengerjaan Halaman Facilities

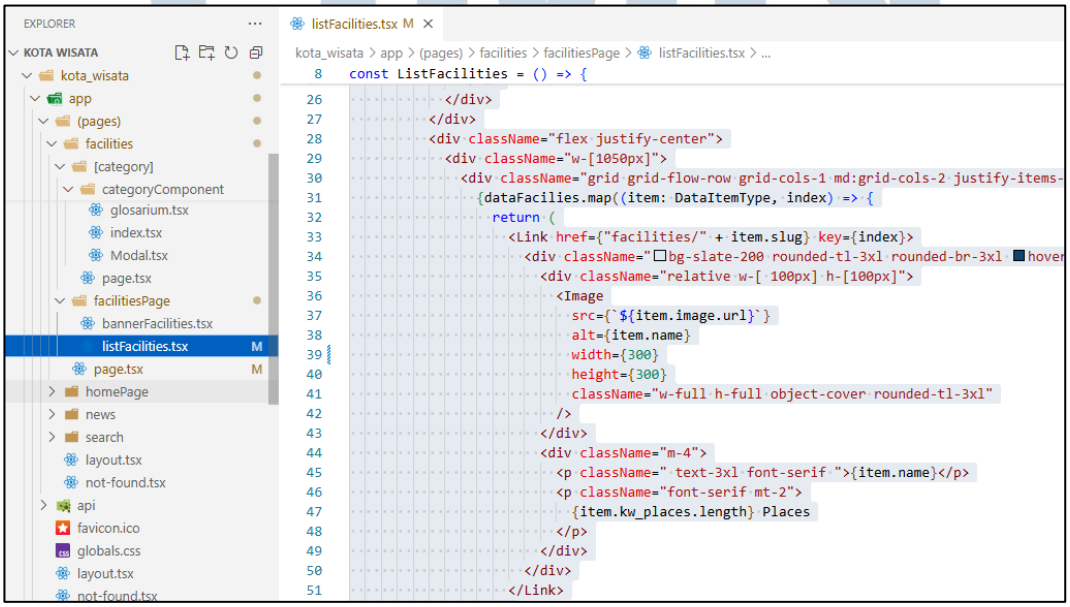
Halaman *Facilities* berfungsi untuk menampilkan berbagai fasilitas unggulan yang terdapat di kawasan Kota Wisata Cibubur, seperti area rekreasi, pusat perbelanjaan, fasilitas pendidikan, serta sarana olahraga. Tujuan utama pengembangan halaman ini adalah untuk memberikan informasi yang menarik dan mudah diakses oleh pengunjung *website* melalui tampilan visual yang informatif dan responsif. Dari sisi teknis, halaman *Facilities* dikembangkan menggunakan *Next.js* sebagai *framework* utama, dengan dukungan *Tailwind CSS* untuk penataan tampilan yang fleksibel dan efisien. Data pada halaman ini diambil secara dinamis dari *Strapi CMS*, sehingga seluruh informasi dapat diperbarui melalui sistem *backend* tanpa perlu melakukan perubahan langsung pada kode *frontend*. Struktur halaman ini terdiri dari beberapa komponen, seperti Banner, *List of Facilities*, dan *Facility Detail Page*, yang masing-masing diatur agar mudah digunakan dan memiliki performa optimal di berbagai perangkat.

Kode 3.27 menunjukkan struktur dasar dari halaman *Facilities* yang dibangun menggunakan *framework Next.js*. Pada kode tersebut, digunakan fungsi *generateMetadata()* untuk menghasilkan *metadata* halaman secara dinamis, seperti judul, deskripsi, *keywords*, dan konfigurasi *Open Graph* yang penting untuk keperluan *SEO* serta tampilan ketika halaman dibagikan di media sosial. Selain itu, terdapat pemanggilan komponen *<ListFacilities />* yang berfungsi untuk menampilkan daftar fasilitas secara dinamis berdasarkan data yang diambil dari *backend* melalui fungsi *getFacilityPage()*. Data yang dikembalikan mencakup elemen seperti judul, deskripsi, gambar, serta *slug* dari setiap kategori fasilitas yang telah disusun di sistem *CMS Strapi*. Pendekatan ini memastikan bahwa konten halaman *Facilities* dapat dikelola secara terpusat melalui *CMS* tanpa perlu melakukan perubahan langsung di sisi *frontend*. Hal tersebut meningkatkan fleksibilitas dan efisiensi dalam pengelolaan konten *website*.



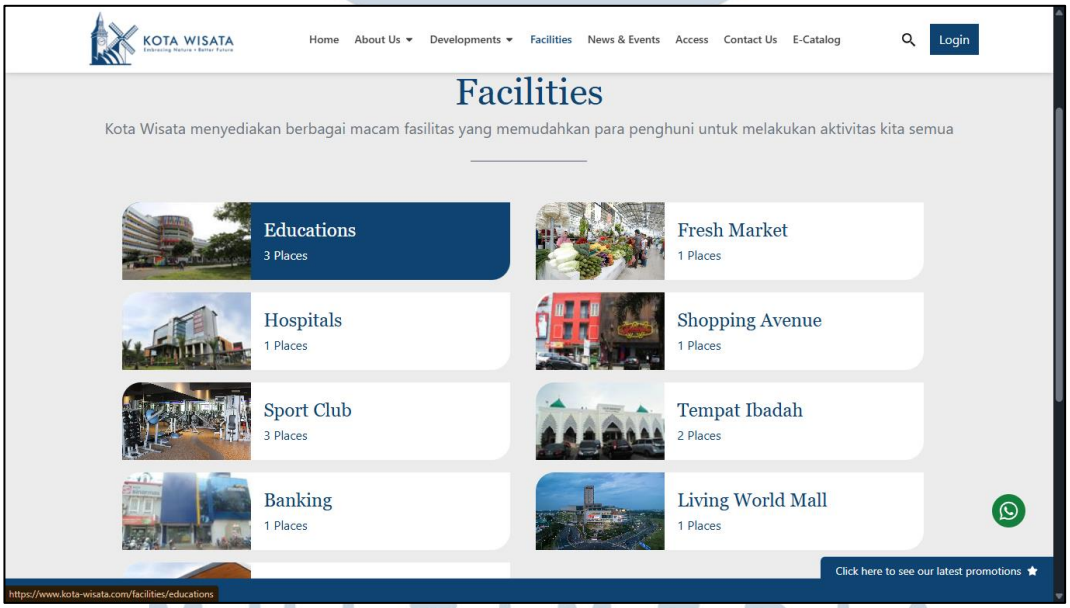
Kode 3.27. Kode Struktur Dasar Halaman Facilities

Kode 3.28 berfungsi untuk menampilkan daftar fasilitas yang ada pada *website* Kota Wisata. Data fasilitas diambil melalui *hook useGeFacilities()* yang terhubung dengan *backend Strapi*, lalu ditampilkan secara dinamis menggunakan metode *.map()*. Setiap fasilitas ditampilkan dalam bentuk kartu (*card*) yang berisi gambar, nama fasilitas, serta jumlah tempat (*places*) yang tersedia di dalamnya. Elemen <*Image*> digunakan untuk memuat gambar dari *API* secara efisien melalui fitur bawaan *Next.js*, sedangkan elemen <*Link*> memungkinkan pengguna menavigasi ke halaman detail fasilitas tertentu. Tampilan setiap *card* juga dilengkapi dengan efek *hover* animatif menggunakan *Tailwind CSS*, seperti perubahan warna latar menjadi biru (*bg-primary*) dan warna teks menjadi putih. Pendekatan ini memberikan pengalaman interaktif bagi pengguna serta menjaga konsistensi desain antarhalaman di seluruh *website*.



Kode 3.28. Kode List Halaman Facilities

Gambar 3.19 menunjukkan tampilan halaman *Facilities* pada *website* Kota Wisata yang telah dikembangkan menggunakan *Next.js* dan *Tailwind CSS*. Pada halaman ini, sistem menampilkan daftar berbagai fasilitas yang tersedia di kawasan Kota Wisata, seperti sekolah, pusat olahraga, tempat ibadah, hingga area komersial. Setiap fasilitas ditampilkan dalam bentuk kartu (*card*) yang berisi gambar representatif, nama fasilitas, serta jumlah lokasi (*places*) yang termasuk di dalam kategori tersebut. Desain kartu dibuat dengan gaya *rounded corner* (sudut melengkung) dan animasi *hover* yang membuat latar berubah warna menjadi biru serta teks menjadi putih ketika pengguna mengarahkan kursor, sehingga tampilan terlihat lebih interaktif dan modern. Selain itu, seluruh data pada halaman ini ditarik secara dinamis melalui *API Strapi*, memastikan konten selalu terbarui sesuai data yang ada di sistem *backend*. Pengguna juga dapat mengklik salah satu kartu untuk diarahkan ke halaman detail fasilitas terkait.



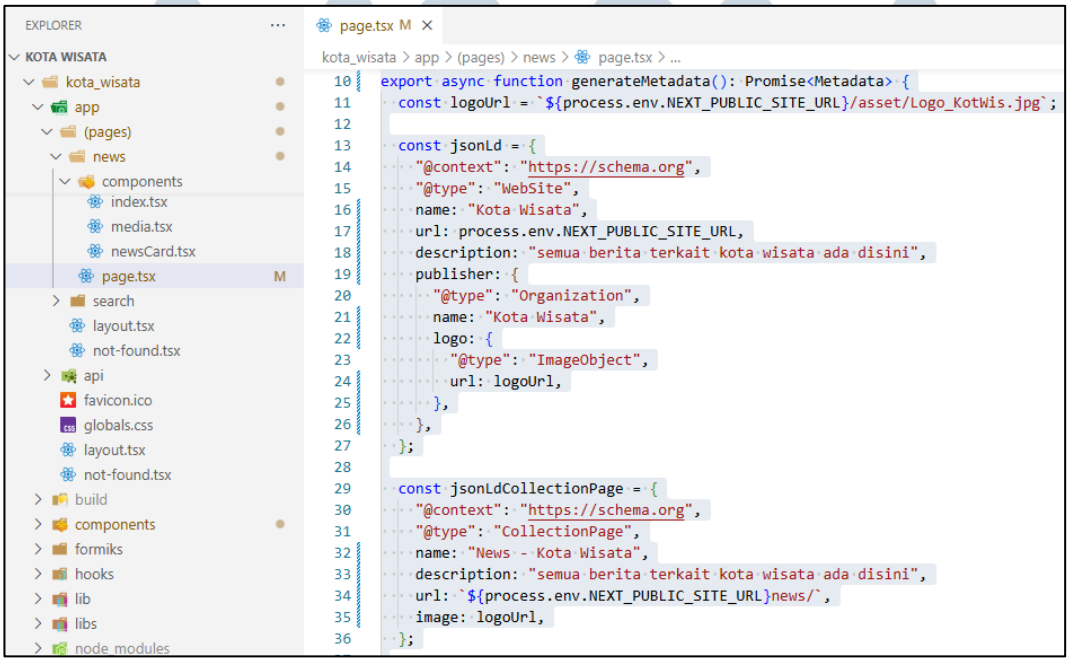
Gambar 3.19 Tampilan Halaman Facilities Website Kota Wisata

G. Pengerjaan Halaman News

Halaman *News* dan *Event* berfungsi untuk menampilkan seluruh berita, artikel, serta kegiatan terkini yang berkaitan dengan kawasan Kota Wisata. Tujuan dari pengembangan halaman ini adalah untuk memberikan informasi terbaru

kepada pengunjung *website* secara dinamis, terintegrasi langsung dengan *backend Strapi* sebagai sumber data. Dalam proses pengembangannya, halaman ini dirancang menggunakan pendekatan modular dengan memisahkan logika pengambilan data, tampilan daftar berita, dan detail artikel. Struktur folder *app/pages/news* mencakup beberapa komponen utama seperti *page.tsx*, *components/index.tsx*, dan komponen pendukung lainnya yang berfungsi menampilkan daftar serta isi berita. Bagian ini akan menjelaskan proses pembuatan halaman *News* mulai dari konfigurasi metadata, pengambilan data berita melalui *API*, hingga penampilan data dalam bentuk daftar berita di antarmuka pengguna.

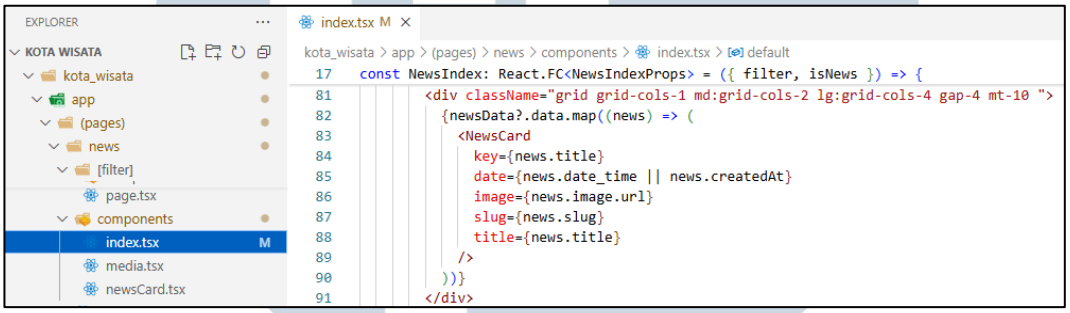
Kode 3.29 menampilkan potongan kode *generateMetadata()* pada halaman *News*. Fungsi ini digunakan untuk menghasilkan metadata dinamis yang membantu optimasi *SEO* dan meningkatkan tampilan tautan ketika dibagikan ke media sosial. Di dalam fungsi tersebut, struktur data *JSON-LD* digunakan untuk menambahkan informasi terstruktur (*structured data markup*) agar halaman berita dapat diidentifikasi dengan benar oleh mesin pencari seperti *Google*. Selain itu, bagian *Open Graph* dan *Twitter Card* memastikan bahwa ketika tautan halaman berita dibagikan di *platform* media sosial, akan muncul pratinjau (*preview*) yang berisi judul, deskripsi, dan gambar dari situs Kota Wisata.



Kode 3.29. Kode fungsi generateMetadata pada halaman News

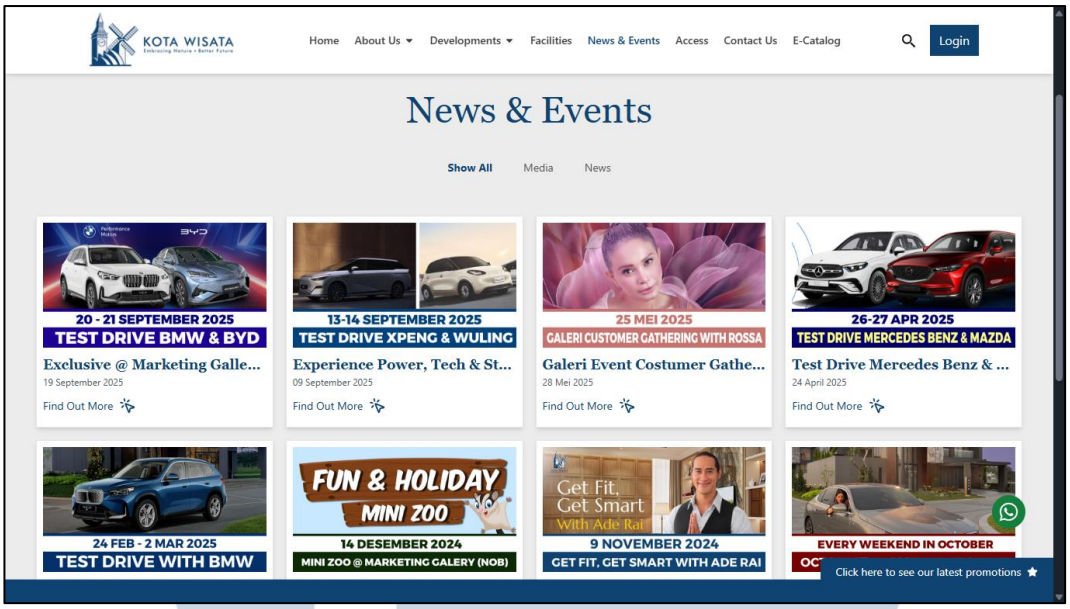
Kode 3.30 berfungsi untuk menampilkan seluruh berita yang diambil dari

backend ke dalam grid *layout* menggunakan komponen *NewsCard*. Masing-masing *item* berita menampilkan gambar, judul, serta tanggal publikasinya. Setelah daftar berita, terdapat komponen *Pagination* yang digunakan untuk membagi tampilan berita menjadi beberapa halaman agar pengguna dapat menavigasi dengan lebih mudah. Bagian ini menjadi inti dari halaman *News & Events*, karena mengatur tampilan utama dari seluruh konten berita yang telah di-*fetch* melalui *hook useGetAllNews()* dan mengatur perpindahan antar halaman secara dinamis menggunakan *state page*.



Kode 3.30. Kode pagination pada halaman News

Gambar 3.20 menampilkan tampilan halaman *News & Events* pada *website* Kota Wisata. Halaman ini menyajikan daftar berita dan acara terkini dalam bentuk kartu (*card*) yang memuat gambar, judul acara/berita, tanggal, dan tombol "*Find Out More*". Pengguna dapat menavigasi berita melalui komponen *Pagination*, yang membagi konten menjadi beberapa halaman agar lebih mudah diakses. Halaman ini juga menggunakan data yang di-*fetch* melalui *hook useGetAllNews()* untuk menampilkan konten secara dinamis, serta mengatur perpindahan antar halaman menggunakan *state page*, sehingga memudahkan pengalaman pengguna saat melihat seluruh konten berita.



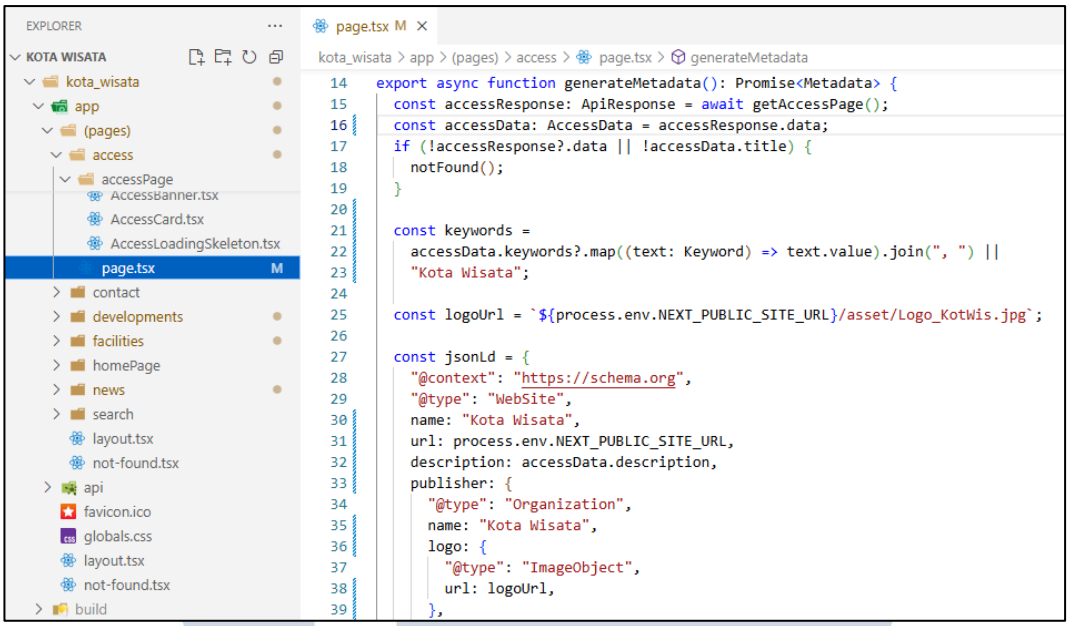
Gambar 3.20. Tampilan Halaman News & Events Kota Wisata

H. Pengerjaan Halaman Access

Halaman *Access* pada *website* Kota Wisata berfungsi untuk memberikan informasi mengenai berbagai akses transportasi menuju kawasan Kota Wisata, baik dari jalan tol, transportasi umum, maupun rute strategis lainnya. Halaman ini dikembangkan dengan pendekatan *dynamic content*, di mana seluruh data seperti judul, deskripsi, dan gambar peta diambil secara otomatis dari *backend Strapi CMS*.

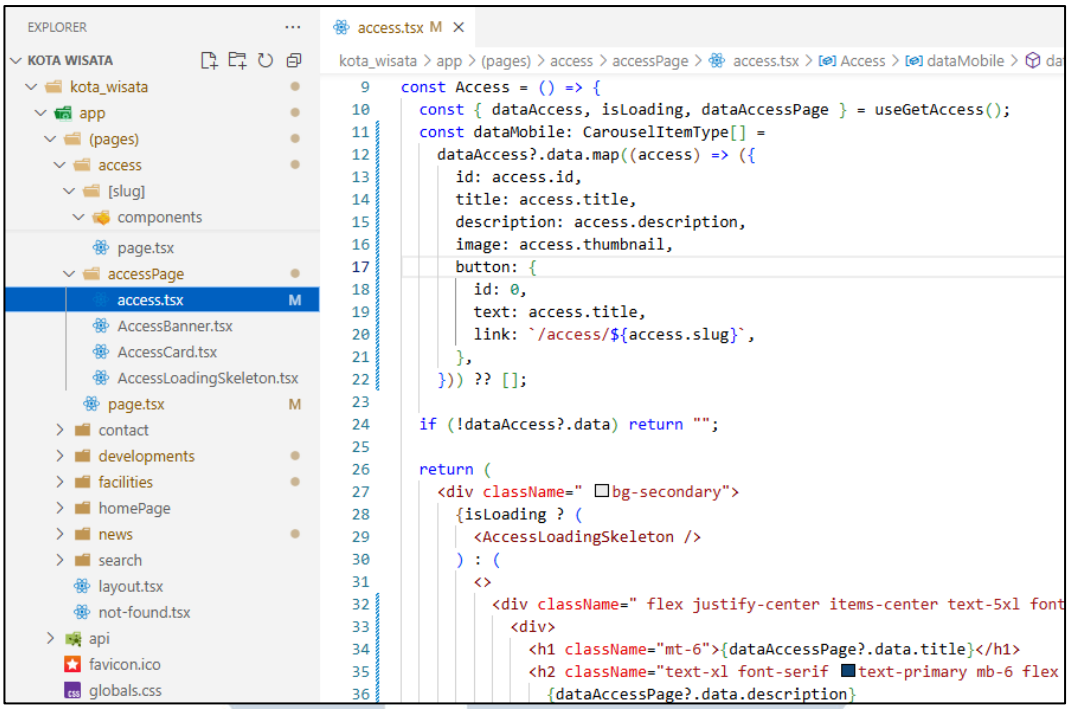
Pada sisi *frontend*, struktur halaman *Access* dibangun menggunakan *Next.js* dengan pendekatan modular *components*, sehingga setiap bagian dapat digunakan kembali (*reusable*). Selain itu, implementasi *metadata* juga diterapkan melalui fungsi *generateMetadata()* untuk mendukung *Search Engine Optimization* (*SEO*) dan memastikan halaman dapat ditampilkan dengan informasi yang lengkap saat dibagikan di media sosial.

Kode 3.31 merupakan fungsi *generateMetadata()* yang digunakan untuk membuat *metadata* halaman *Access* secara dinamis berdasarkan data dari *Strapi API*. *Metadata* ini mencakup judul, deskripsi, kata kunci, serta elemen *Open Graph* dan *Twitter Card* yang berfungsi untuk meningkatkan optimasi *SEO* dan tampilan pratinjau halaman di media sosial.



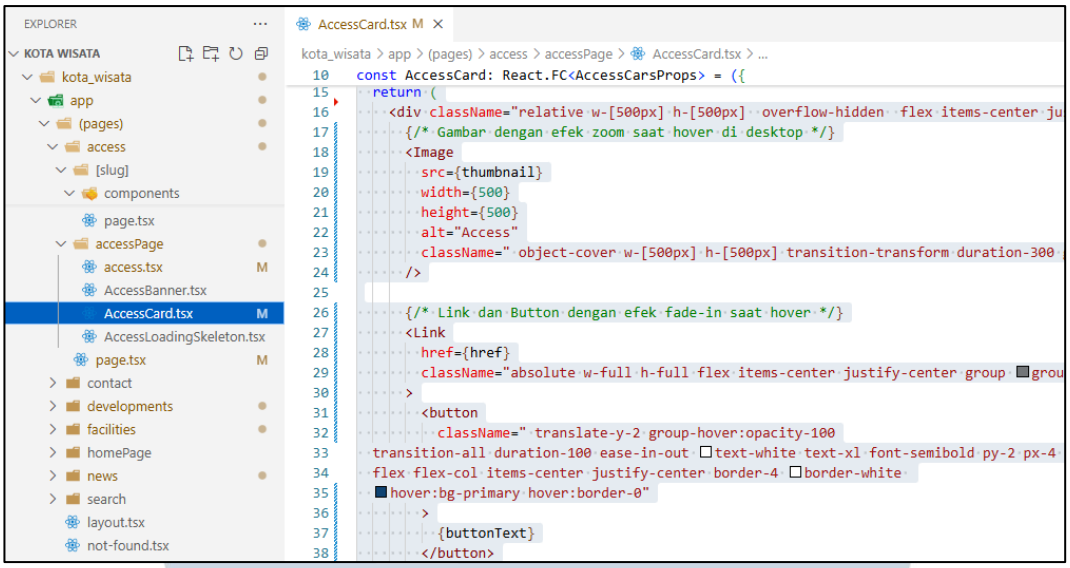
Kode 3.31. Kode generateMetadata pada Halaman Access

Kode 3.32 memperlihatkan struktur utama komponen *Access*, yang berfungsi untuk menampilkan seluruh daftar akses menuju kawasan Kota Wisata. Data halaman diambil secara dinamis dari *backend* menggunakan *custom hook useGetAccess()*, yang mengembalikan data halaman (*dataAccessPage*), daftar akses (*dataAccess*), dan status pemuatan (*isLoading*). Saat proses pengambilan data masih berlangsung, sistem akan menampilkan komponen *loading skeleton* bernama *AccessLoadingSkeleton* sebagai *placeholder* agar tampilan tetap interaktif. Setelah data berhasil dimuat, daftar akses ditampilkan dalam bentuk grid tiga kolom menggunakan *component AccessCard* untuk tampilan *desktop*, sedangkan versi mobile ditampilkan dalam bentuk *carousel* menggunakan *CarauselMobile*. Pendekatan ini menjadikan tampilan halaman responsif dan dinamis, karena *layout* dapat menyesuaikan perangkat pengguna tanpa kehilangan informasi penting. Selain itu, implementasi struktur komponen seperti ini memudahkan pengembangan di masa depan karena setiap bagian *UI* sudah dipisahkan ke dalam *component* modular yang terkelola dengan baik.



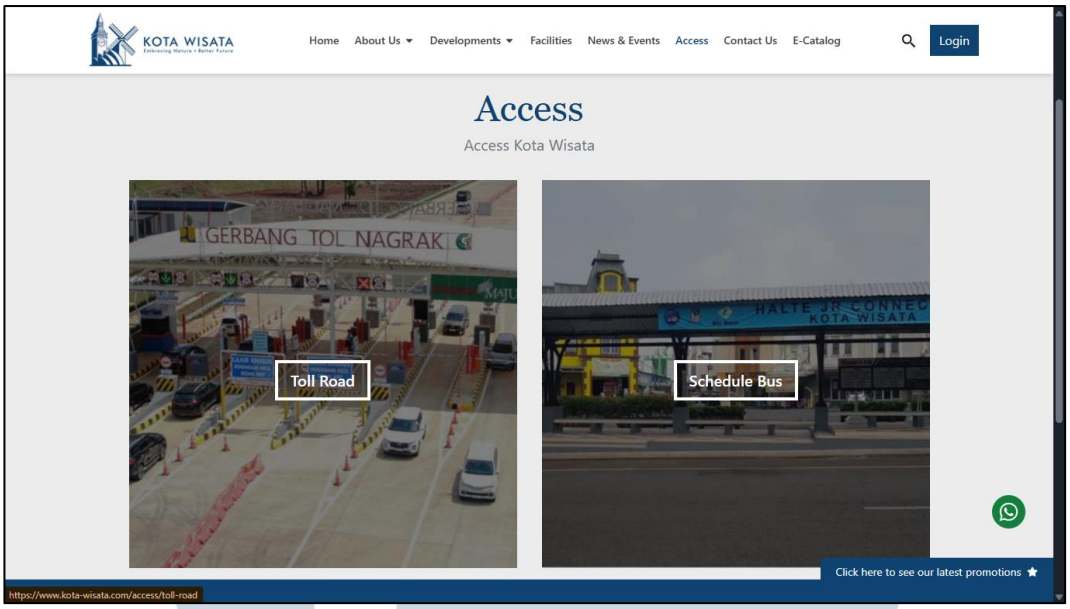
Kode 3.32. Kode Komponen Access pada Halaman Access

Kode 3.33 memperlihatkan potongan kode dari komponen *AccessCard*, yang digunakan untuk menampilkan setiap kartu akses pada halaman *Access*. Setiap kartu berisi gambar area akses dan tombol yang mengarah ke halaman detail terkait. Kode ini memanfaatkan komponen *Image* dan *Link* dari *Next.js* untuk menampilkan gambar yang dioptimalkan serta navigasi yang cepat antarhalaman. Efek *hover* interaktif diterapkan menggunakan kelas *Tailwind CSS* seperti *group-hover:scale-110* untuk memperbesar gambar secara halus, serta *group-hover:bg-black/50* untuk memberikan efek *overlay* gelap ketika kursor diarahkan ke elemen. Selain itu, tombol di tengah gambar muncul dengan animasi lembut melalui properti *translate-y-2* dan *group-hover:opacity-100*, yang membuat transisi terlihat elegan. Desain ini tidak hanya mempercantik tampilan antarmuka, tetapi juga memberikan pengalaman pengguna yang intuitif dan responsif di seluruh ukuran layar.



Kode 3.33. Kode Komponen AccessCard

Gambar 3.21 memperlihatkan tampilan halaman *Access* pada versi *desktop* setelah seluruh komponen seperti *Access.tsx* dan *AccessCard.tsx* diimplementasikan. Pada tampilan ini, setiap area akses (seperti tol, *LRT*, dan rute menuju kawasan Kota Wisata) ditampilkan dalam bentuk kartu bergambar besar yang tersusun dalam grid berkolom tiga. Setiap kartu menampilkan foto representatif beserta tombol berlabel sesuai nama aksesnya, misalnya "Tol Cimanggis – Cibubur" atau "*LRT Station*". Saat pengguna mengarahkan kursor ke gambar, animasi *zoom-in* halus dan overlay berwarna gelap akan muncul, sementara tombol di tengah gambar menonjol dengan efek *hover* berwarna biru (kelas *hover:bg-primary*). Desain halaman ini dibuat responsif — pada perangkat *mobile*, susunan kartu akan berubah menjadi *carousel* horizontal menggunakan komponen *CarauselMobile*, sehingga pengguna tetap dapat menelusuri seluruh informasi akses dengan mudah. Pendekatan ini menjaga konsistensi tampilan dan memastikan pengalaman pengguna tetap optimal di berbagai perangkat.



Gambar 3.21. Tampilan Halaman Access Versi Desktop

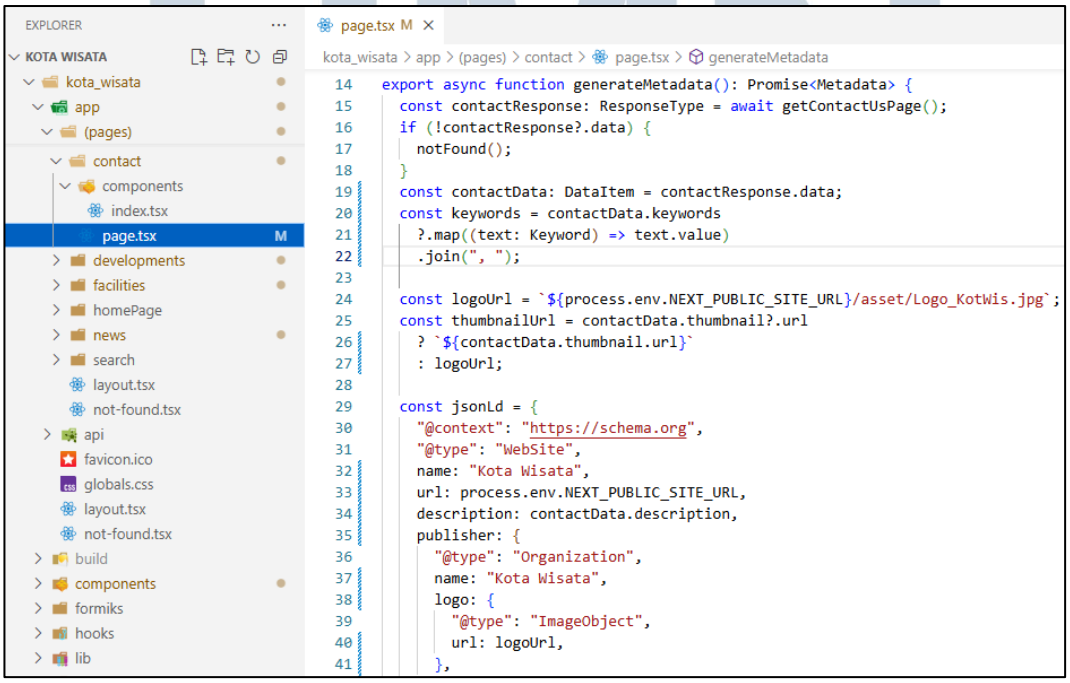
Secara keseluruhan, halaman *Access* dikembangkan untuk menampilkan informasi aksesibilitas menuju kawasan Kota Wisata secara informatif dan interaktif. Implementasi halaman ini memanfaatkan *Next.js* dan *Tailwind CSS* dengan pendekatan *responsive design*, sehingga tampilan dapat beradaptasi baik pada perangkat *desktop* maupun *mobile*. Data yang ditampilkan pada halaman ini diambil secara dinamis melalui *API Strapi*, kemudian dipetakan dalam bentuk komponen *reusable* seperti *AccessCard* dan *CarauselMobile*. Setiap kartu menampilkan gambar representatif serta tautan menuju halaman detail masing-masing akses. Dengan adanya halaman ini, pengunjung *website* dapat dengan mudah memperoleh informasi mengenai jalur transportasi dan sarana pendukung yang menghubungkan berbagai area ke kawasan Kota Wisata. Tampilan yang menarik dan navigasi yang intuitif juga meningkatkan kenyamanan pengguna dalam menelusuri konten *website* secara keseluruhan.

I. Pengerjaan Halaman Contact Us

Halaman *Contact Us* atau dikembangkan sebagai media komunikasi antara pengunjung *website* dengan pihak pengelola Kota Wisata. Tujuan utama dari halaman ini adalah untuk memudahkan pengguna dalam menyampaikan pertanyaan, permintaan informasi, maupun keperluan bisnis melalui formulir

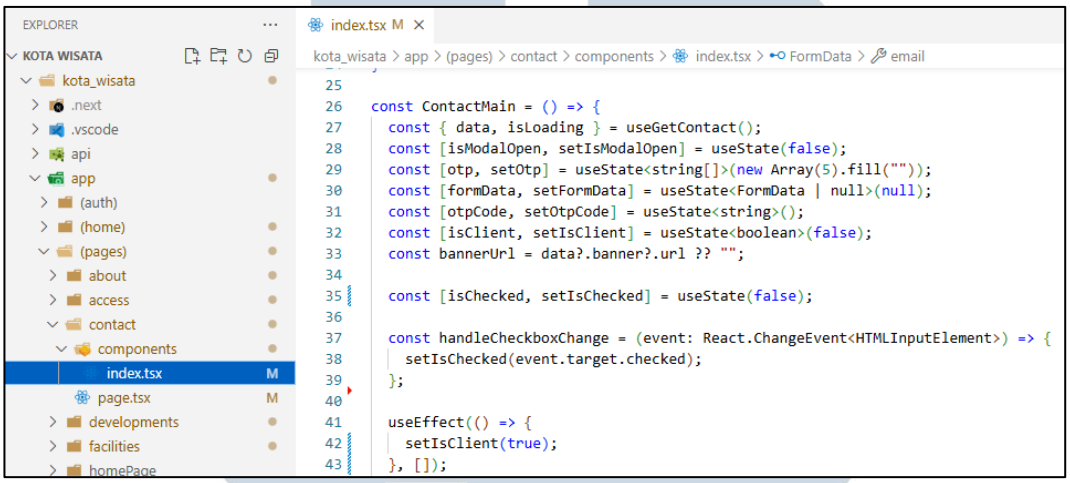
kontak yang terhubung langsung ke sistem *backend*. Dalam proses pengembangannya, halaman ini dibuat dengan menggunakan *Next.js* dan *Tailwind CSS*, serta mengintegrasikan data dari *Strapi CMS* agar setiap elemen seperti alamat kantor, nomor telepon, tautan media sosial, dan peta lokasi dapat ditampilkan secara dinamis. Selain itu, halaman ini juga dirancang dengan pendekatan *responsive design* sehingga tampilan tetap optimal baik di perangkat *desktop* maupun *mobile*. Struktur komponen pada halaman ini terdiri dari banner, formulir kontak, informasi perusahaan, serta peta interaktif yang menunjukkan lokasi kantor pemasaran. Setiap bagian diimplementasikan secara modular menggunakan komponen *React* agar mudah dipelihara dan diintegrasikan dengan halaman lain dalam proyek.

Kode 3.34 merupakan bagian dari fungsi *generateMetadata* pada *file page.tsx*. Fungsi ini digunakan untuk menghasilkan metadata dinamis seperti judul halaman, deskripsi, kata kunci, dan gambar pratinjau (*open graph* & *twitter card*) secara otomatis berdasarkan data yang diambil dari *API Strapi*. Selain itu, terdapat konfigurasi *structured* data (*JSON-LD*) yang membantu meningkatkan *SEO* dengan memberikan konteks tambahan kepada mesin pencari bahwa halaman ini merupakan halaman *Contact Page* milik *website* Kota Wisata. Dengan pendekatan ini, setiap halaman dapat memiliki *metadata* yang relevan tanpa harus diatur secara manual.



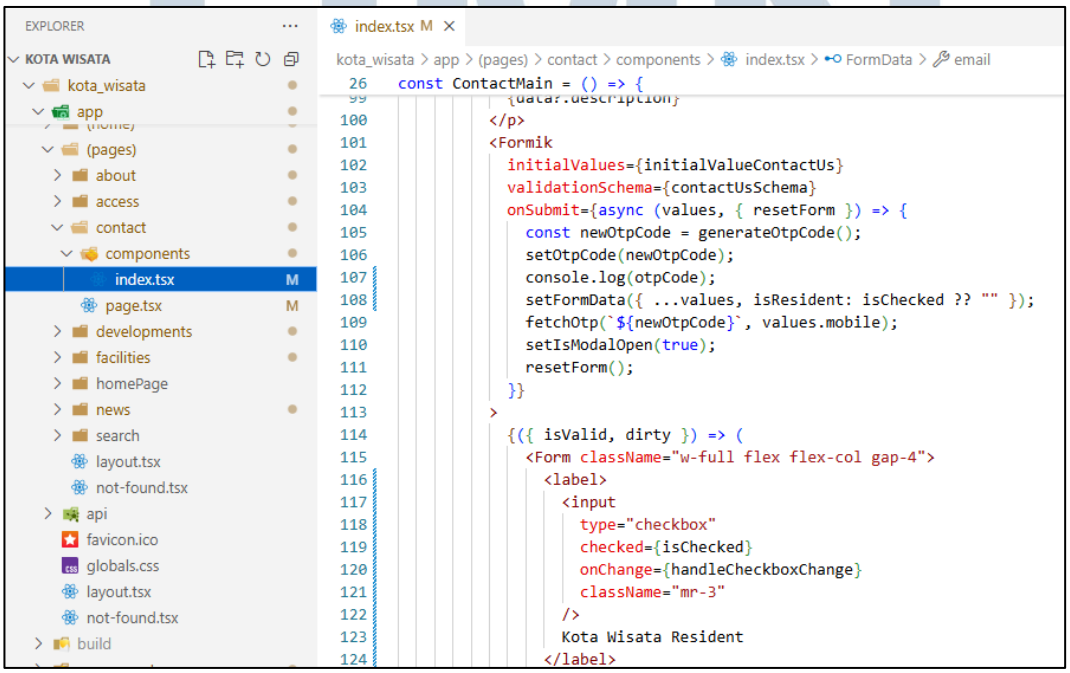
Kode 3.34. Kode generateMetadata pada file page.tsx halaman Contact Us

Kode 3.35 berfungsi untuk mengatur *state* utama pada formulir *Contact Us* seperti *modal OTP*, data input pengguna, serta kode *OTP* acak. Fungsi *useEffect* digunakan untuk memastikan halaman hanya dijalankan di sisi klien (*client-side rendering*).



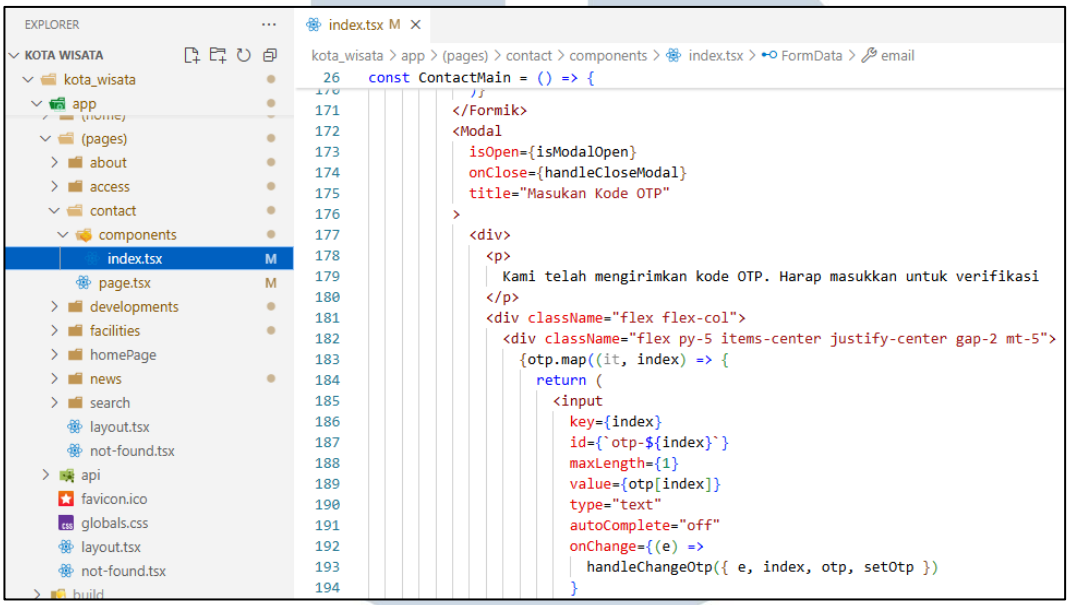
Kode 3.35. Kode inisialisasi state dan OTP pada Contact Us

Kode 3.36 membentuk formulir utama pengguna dengan validasi menggunakan *Formik* dan *Yup Schema*. Terdapat input untuk nama, *email*, nomor telepon, pesan, dan opsi "Kota Wisata Resident". Ketika *form* disubmit, sistem otomatis mengirim kode *OTP* ke nomor pengguna untuk verifikasi sebelum data benar-benar dikirim ke *backend*.



Kode 3.36. Kode struktur Formik dan Form pada halaman Contact Us

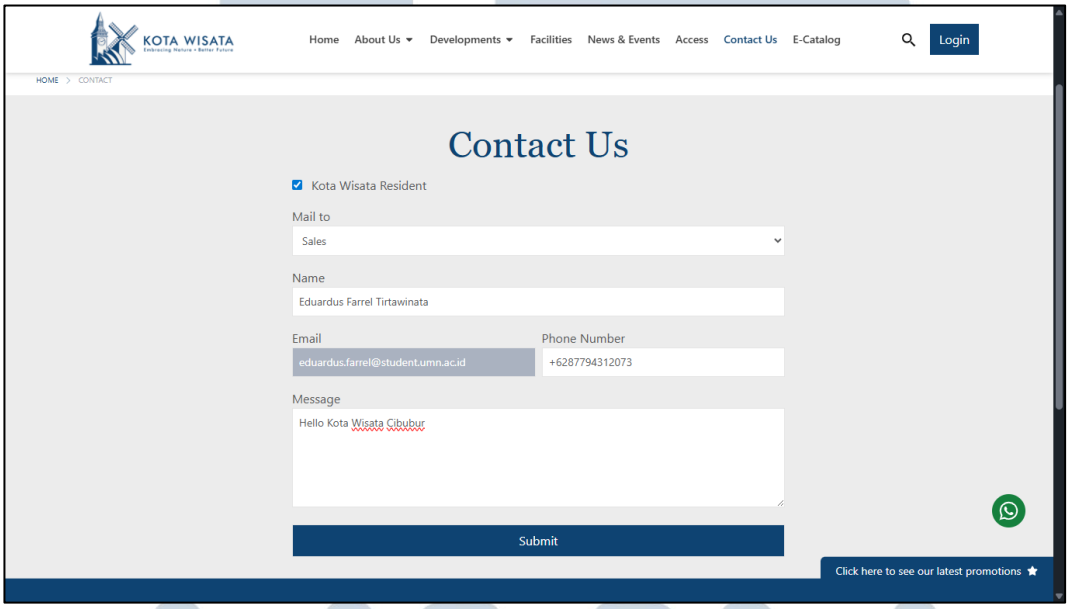
Kode 3.37 menampilkan *popup modal OTP* yang muncul setelah pengguna menekan tombol "*Submit*". Pengguna harus memasukkan kode *OTP* berjumlah 5 digit, dan sistem akan memverifikasi input tersebut dengan kode yang dikirim ke nomor telepon.



Kode 3.37. Kode modal OTP verifikasi pada halaman Contact Us

Gambar 3.22 menampilkan halaman *Contact Us* pada *website* Kota Wisata Cibubur dirancang dengan antarmuka yang bersih, profesional, dan responsif menggunakan *framework Tailwind CSS*. Pada bagian kiri halaman, ditampilkan gambar banner yang diambil secara dinamis dari data *backend* melalui *API useGetContact*. Sementara itu, di sisi kanan halaman terdapat formulir utama yang berfungsi sebagai sarana pengunjung untuk mengirimkan pesan, pertanyaan, maupun permintaan informasi kepada pihak pengelola Kota Wisata Cibubur. Formulir ini terdiri atas beberapa elemen input seperti *checkbox* bertuliskan Kota Wisata Resident yang digunakan untuk menandai apakah pengirim pesan merupakan penghuni kawasan, *dropdown Mail To* untuk menentukan tujuan pengiriman pesan, serta kolom input *Name*, *Email*, *Phone Number*, dan *Message* yang wajib diisi. Ketika pengguna menekan tombol *Submit*, sistem akan menampilkan *modal* verifikasi berupa kode OTP (*One Time Password*) yang berfungsi untuk memastikan keaslian data dan keamanan proses pengiriman pesan. Pengguna kemudian diminta untuk memasukkan lima digit kode *OTP* yang dikirimkan ke nomor telepon yang telah dimasukkan sebelumnya. Jika kode *OTP*

yang dimasukkan benar, maka data dari formulir akan dikirim ke *backend* melalui fungsi *postContactUs()* dan tersimpan dengan aman di sistem. Validasi setiap input dilakukan secara otomatis menggunakan *Formik* dan *Yup Schema*, sehingga memastikan seluruh kolom diisi dengan format yang benar sebelum formulir dapat dikirim. Apabila terdapat kesalahan input atau kode *OTP* yang dimasukkan tidak valid, sistem akan menampilkan notifikasi kesalahan agar pengguna dapat segera memperbaikinya.



Gambar 3.22. Tampilan antarmuka halaman Contact Us

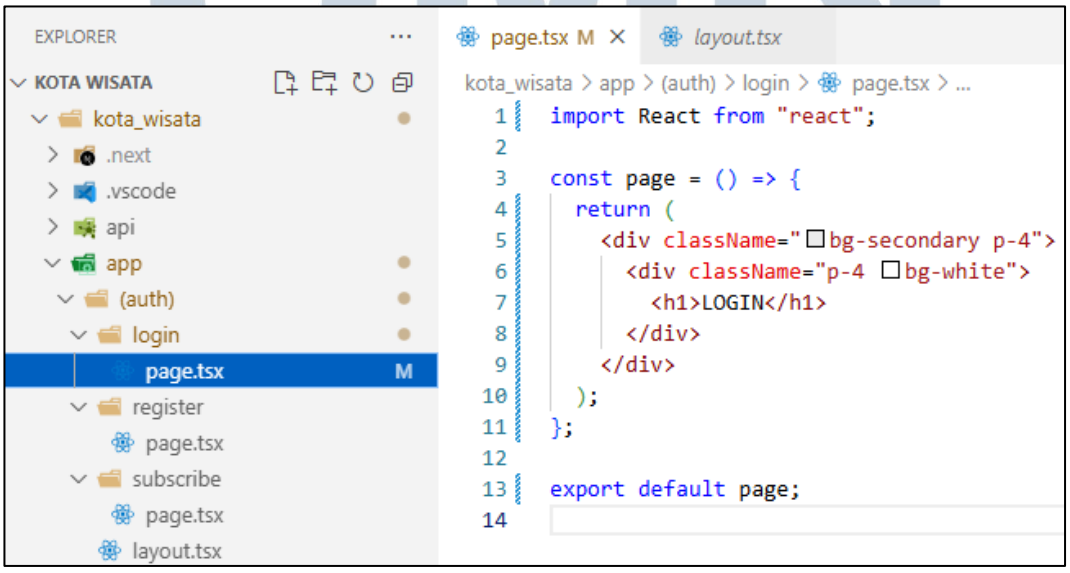
Secara keseluruhan, halaman *Contact Us* pada *website* Kota Wisata Cibubur ini berhasil menggabungkan fungsi dan estetika dengan baik. Desainnya sederhana namun informatif, dan fitur verifikasi *OTP* menambah lapisan keamanan yang penting dalam proses komunikasi antara pengguna dan pihak pengelola. Hal ini menjadikan halaman *Contact Us* tidak hanya menarik secara visual, tetapi juga interaktif, aman, dan sesuai dengan tampilan serta alur versi aslinya di *website* resmi Kota Wisata Cibubur.

J. Pengerjaan Fitur Autentikasi Pengguna

Fitur autentikasi merupakan bagian penting dalam sistem *website* Kota Wisata Cibubur, karena berfungsi untuk mengelola akses pengguna yang ingin melakukan *login*, mendaftar akun baru, maupun berlangganan informasi terbaru

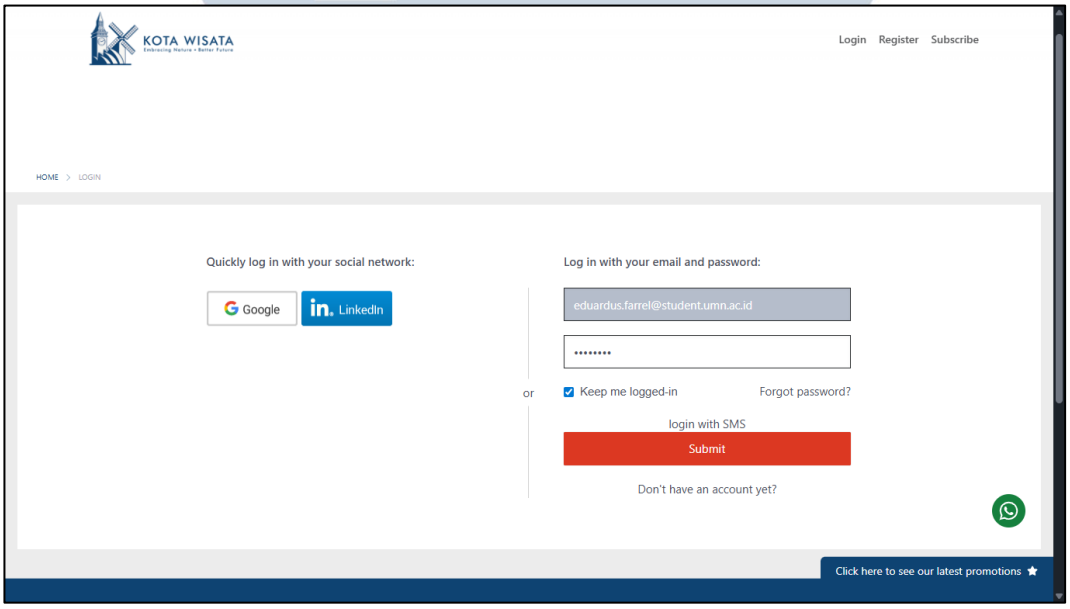
melalui halaman *subscribe*. Seluruh proses autentikasi ini dibangun menggunakan *framework Next.js* dengan pendekatan *client-side rendering* agar interaksi pengguna lebih cepat dan responsif. Halaman *login* memungkinkan pengguna yang telah terdaftar untuk mengakses akun mereka menggunakan *email* dan kata sandi yang valid. Sementara itu, halaman registrasi digunakan bagi pengguna baru untuk membuat akun dengan mengisi informasi pribadi yang dibutuhkan. Selain dua halaman utama tersebut, dikembangkan pula halaman *subscribe* yang berfungsi untuk mengumpulkan data pengunjung yang ingin menerima berita atau promo terbaru dari *website* Kota Wisata Cibubur. Setiap halaman autentikasi dirancang dengan antarmuka yang sederhana dan konsisten dengan identitas visual *website*. Implementasi logika autentikasi dilakukan melalui integrasi *API* yang berkomunikasi dengan *backend*, sehingga proses validasi akun dan penyimpanan data pengguna dapat berjalan dengan aman dan efisien.

Kode 3.38 masih bersifat dasar dan berfungsi sebagai struktur awal dari halaman *login*. Komponen ini menggunakan pendekatan *functional component* di *React* dan disusun dengan *Tailwind CSS* untuk pengaturan tampilan. Di dalam elemen utama terdapat dua bagian: *div* pembungkus dengan *background* berwarna sekunder (*bg-secondary*) dan elemen di dalamnya berwarna putih (*bg-white*) yang berfungsi sebagai wadah konten utama. Pada tahap ini, halaman login hanya menampilkan teks "*LOGIN*" sebagai *placeholder* sebelum ditambahkan *form* autentikasi dan logika validasi pengguna.



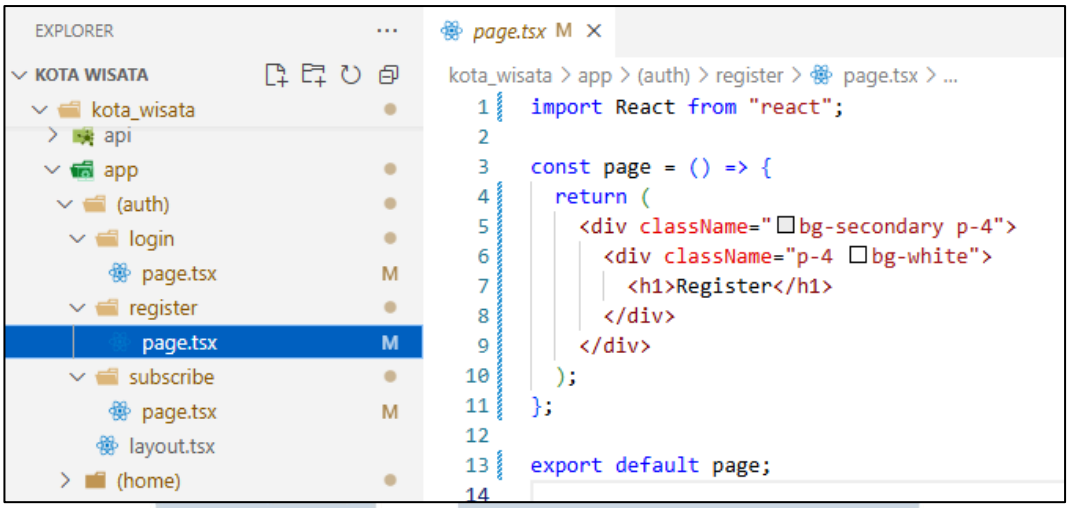
Kode 3.38. Kode Login Page Kota Wisata

Gambar 3.23 memperlihatkan tampilan antarmuka halaman *Login* pada *website* Kota Wisata Cibubur. Pengguna diberikan dua opsi untuk melakukan autentikasi, yaitu melalui akun sosial seperti *Google* dan *LinkedIn*, atau dengan memasukkan email dan password secara manual. Tersedia pula opsi tambahan seperti "*Keep me logged-in*" untuk mempertahankan sesi *login*, serta tautan "*Forgot password?*" bagi pengguna yang lupa kredensialnya. Tombol *Submit* berwarna merah menjadi elemen utama yang menonjol untuk mengirimkan data *login*. Selain itu, terdapat tautan "*Don't have an account yet?*" yang mengarahkan pengguna menuju halaman pendaftaran (*Register*). Desain halaman ini menggunakan kombinasi warna putih dan abu-abu lembut yang memberikan kesan bersih dan profesional, dengan *layout* yang responsif serta terintegrasi dengan navigasi utama situs di bagian atas.



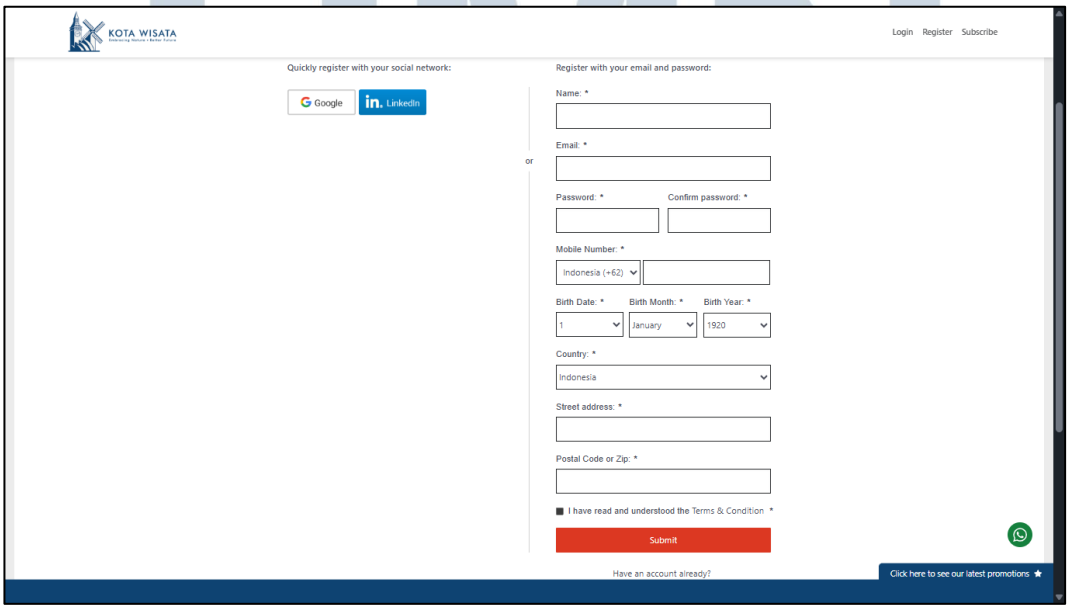
Gambar 3.23. Tampilan Halaman Login Kota Wisata Cibubur

Kode 3.39 terdiri dari dua lapisan utama, yakni pembungkus dengan latar belakang abu muda (*bg-secondary*) dan wadah konten berwarna putih (*bg-white*) yang menampilkan teks "*Register*". Struktur ini disiapkan sebagai kerangka dasar sebelum ditambahkan elemen *form* pendaftaran pengguna, seperti input nama, *email*, dan kata sandi. Dengan adanya *file* ini, halaman *Register* melengkapi proses autentikasi pengguna di *website* Kota Wisata Cibubur, sehingga nantinya pengguna dapat melakukan registrasi akun secara mandiri sebelum masuk melalui halaman *Login*.



Kode 3.39 Kode Register Page Kota Wisata

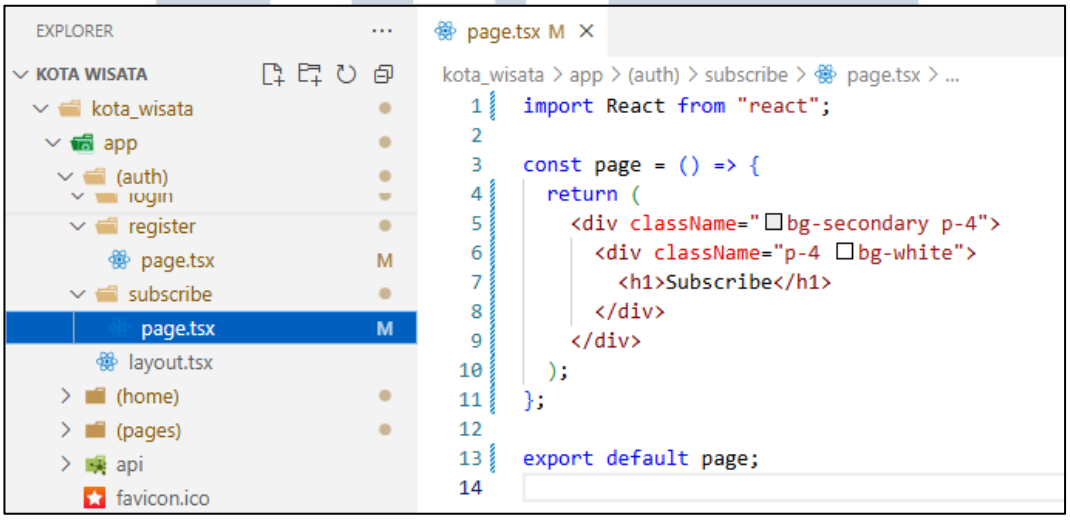
Gambar 2.24 menampilkan halaman *Register* pada *website* Kota Wisata Cibubur. Pada halaman ini, pengguna dapat melakukan pendaftaran akun baru dengan mengisi data pribadi seperti nama, alamat email, kata sandi, nomor telepon, tanggal lahir, negara, serta alamat tempat tinggal. Desain *form* pendaftaran dibuat sederhana dan responsif, dilengkapi opsi pendaftaran cepat menggunakan akun *Google* atau *LinkedIn* agar pengguna dapat masuk dengan lebih mudah. Setelah semua kolom diisi, pengguna dapat menekan tombol *Submit* untuk menyelesaikan proses registrasi.



Gambar 3.24. Tampilan Halaman Register Kota Wisata Cibubur

Kode 3.40 menampilkan struktur dasar dari halaman *Subscribe* pada *website*

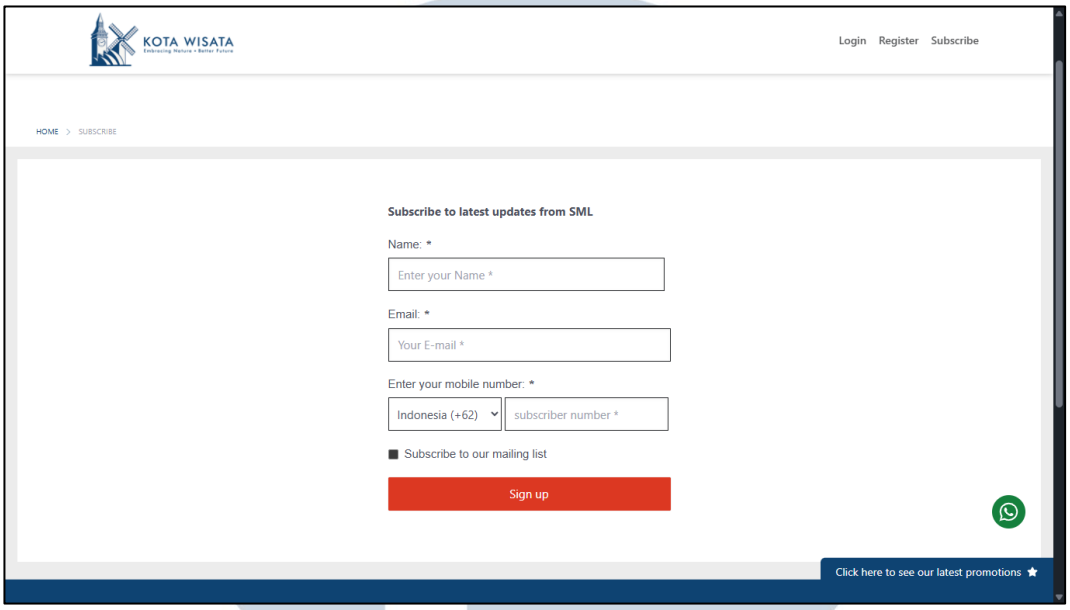
Kota Wisata Cibubur. Halaman ini dibuat sederhana dengan latar belakang berwarna abu-abu muda (*bg-secondary*) dan sebuah *container* berwarna putih di tengah halaman yang menampilkan teks "*Subscribe*". Tujuan dari halaman ini adalah untuk menjadi template dasar yang nantinya dapat dikembangkan menjadi halaman berfungsi penuh, misalnya untuk pendaftaran *newsletter* atau notifikasi terbaru dari *website*. Struktur dan gaya komponen ini dibuat konsisten dengan halaman *Login* dan *Register*, agar ketiga halaman dalam folder *auth* memiliki tampilan yang seragam dan mudah dipahami pengguna.



Kode 3.40. Kode Subscribe Page Kota Wisata

Gambar 3.25 memperlihatkan antarmuka sederhana yang dirancang agar pengguna dapat berlangganan pembaruan informasi terbaru dari Sinarmas Land, khususnya terkait kawasan Kota Wisata Cibubur. Pada bagian atas halaman, terdapat navigasi yang konsisten dengan struktur utama *website*, menampilkan menu *Login*, *Register*, dan *Subscribe* di sisi kanan. Bagian utama halaman berisi formulir berlangganan dengan tiga kolom input utama, yaitu *Name*, *Email*, dan *Mobile Number*, yang wajib diisi oleh pengguna. Selain itu, terdapat opsi untuk mencentang kotak *Subscribe to our mailing list* yang menandakan persetujuan untuk menerima pembaruan rutin melalui *email*. Di bagian bawah, terdapat tombol berwarna merah bertuliskan *Sign Up* yang berfungsi untuk mengirimkan data berlangganan. Desain halaman didominasi oleh warna putih dan abu muda untuk menciptakan kesan bersih dan profesional, dengan sedikit aksen merah yang memberikan daya tarik visual pada elemen utama. Tata letak yang sederhana ini

memastikan pengalaman pengguna tetap fokus dan mudah digunakan tanpa elemen visual yang berlebihan.

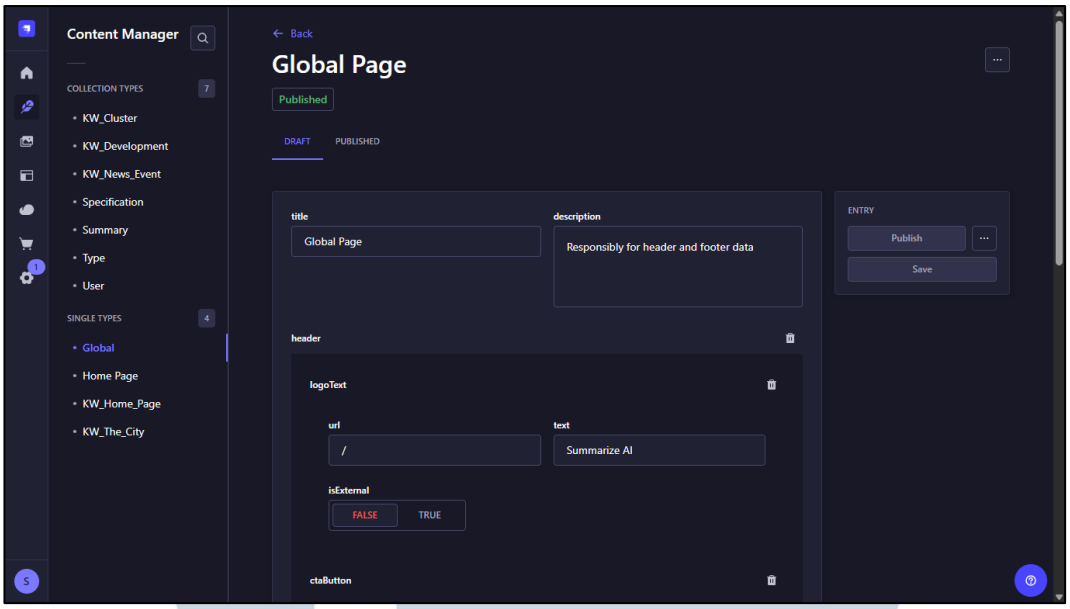


Gambar 3.25. Tampilan Halaman Subscribe Kota Wisata Cibubur

K. Pengerjaan Backend Menggunakan Strapi

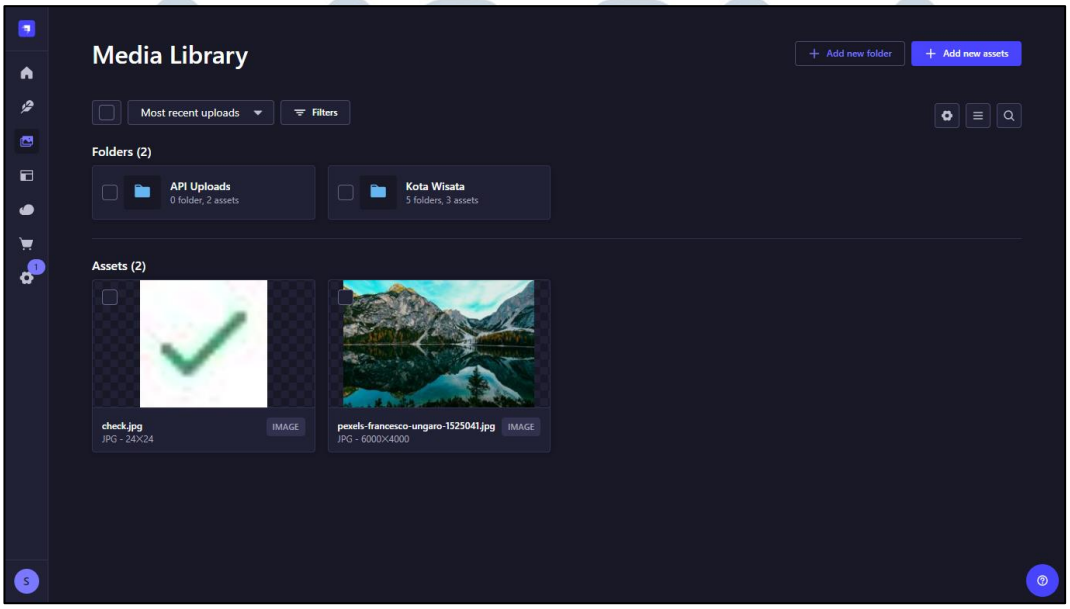
Strapi digunakan sebagai *headless CMS* untuk mengelola seluruh data yang ditampilkan di *website* Kota Wisata Cibubur, seperti data halaman *Home*, *Developments*, *Facilities*, *News*, dan lainnya. *Platform* ini memisahkan logika *backend* dari *frontend* sehingga memungkinkan *website* yang dibuat dengan *Next.js* menampilkan data secara dinamis melalui *API* yang disediakan *Strapi*.

Gambar 3.26 memperlihatkan halaman *Content Manager* di *Strapi* yang digunakan untuk menambahkan, mengedit, dan menghapus data dari berbagai *collection types* maupun *single types*. Misalnya, pada contoh yang terlihat adalah halaman *Global Page*, yang berfungsi untuk mengatur konten global seperti *header* dan *footer* agar dapat digunakan di seluruh halaman *website*. Melalui tampilan ini, setiap perubahan yang dilakukan akan langsung tersinkronisasi dengan *API* yang dikonsumsi oleh *frontend Next.js*.



Gambar 3.26. Tampilan Content Manager Strapi

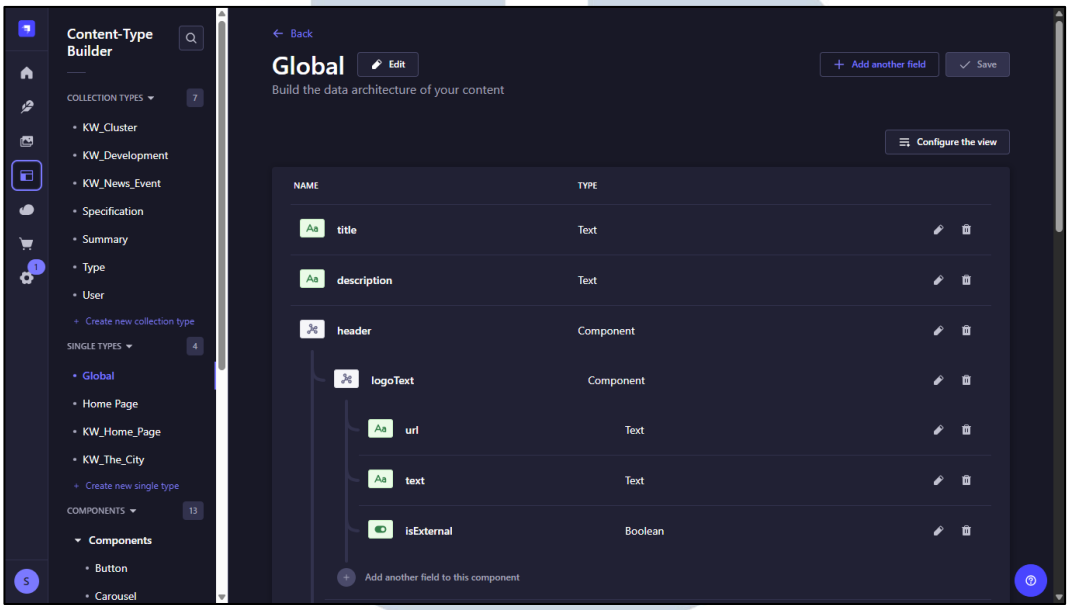
Gambar 3.27 adalah *Media Library* yang berfungsi sebagai tempat penyimpanan seluruh aset media seperti gambar banner, ikon, dan file lainnya yang digunakan di setiap halaman *website*. Semua *file* yang diunggah di sini dapat diakses melalui *URL* publik yang disediakan oleh *Strapi* dan digunakan di komponen *frontend* seperti <*Image*> dari *Next.js*.



Gambar 3.27. Tampilan Media Library Strapi

Gambar 3.28 adalah *Content-Type Builder* yaitu fitur yang memungkinkan pembuatan struktur data sesuai kebutuhan proyek. Melalui tampilan ini, dapat

dibuat collection type seperti *KW_Cluster*, *KW_Development*, dan *KW_News_Event*, maupun *single type* seperti *Global* atau *Home Page*. Setiap *content type* dapat berisi berbagai *field* seperti *title*, *description*, *image*, *slug*, dan sebagainya, yang nantinya menjadi dasar *API endpoint* untuk diakses oleh *frontend*.



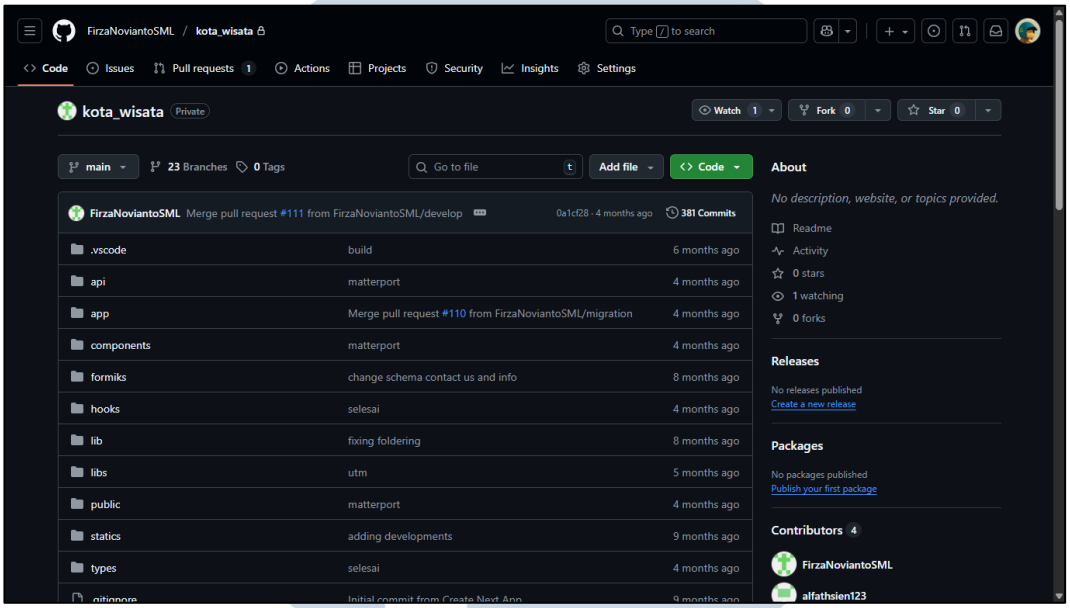
Gambar 3.28. Tampilan Content-Type Builder Strapi

L. Penggunaan Git & GitHub dalam Pengembangan Proyek

Dalam proses pengembangan *website* Kota Wisata Cibubur, sistem kontrol versi yang digunakan adalah *Git* dengan layanan repositori *GitHub*. Penggunaan *Git* dan *GitHub* memiliki peranan penting dalam menjaga konsistensi versi kode, memudahkan kolaborasi antar anggota tim, serta memastikan setiap perubahan pada proyek tercatat dengan baik. Dengan adanya sistem ini, seluruh proses pengembangan dapat dilakukan secara terstruktur dan terpantau melalui *commit*, *branch*, serta riwayat perubahan yang terdokumentasi secara otomatis di *repository GitHub*. Pada awal proses pengembangan, *repository* proyek dibuat di *GitHub* untuk menyimpan seluruh *source code* aplikasi, termasuk *frontend* berbasis *Next.js* dan integrasi *backend* melalui *Strapi*. *Repository* ini juga berfungsi sebagai tempat penyimpanan utama agar setiap pembaruan dari lingkungan pengembangan lokal (*VSCode*) dapat diunggah secara langsung dan tersinkronisasi dengan versi daring.

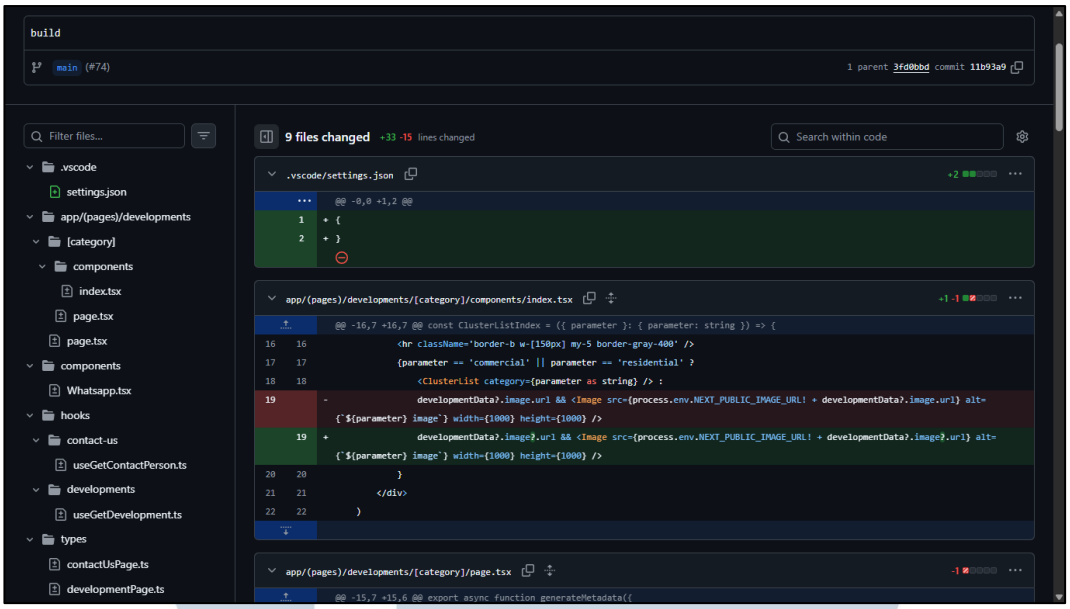
Gambar 3.29 memperlihatkan struktur direktori dan *file* utama proyek yang telah diunggah ke *GitHub*. *Repository* ini menjadi pusat kolaborasi antara tim

pengembang, di mana setiap anggota dapat mengakses, memperbarui, dan memeriksa kode secara *real-time*. Melalui *repository* ini, proses pengembangan proyek dilakukan dengan transparan dan terdokumentasi dengan baik.



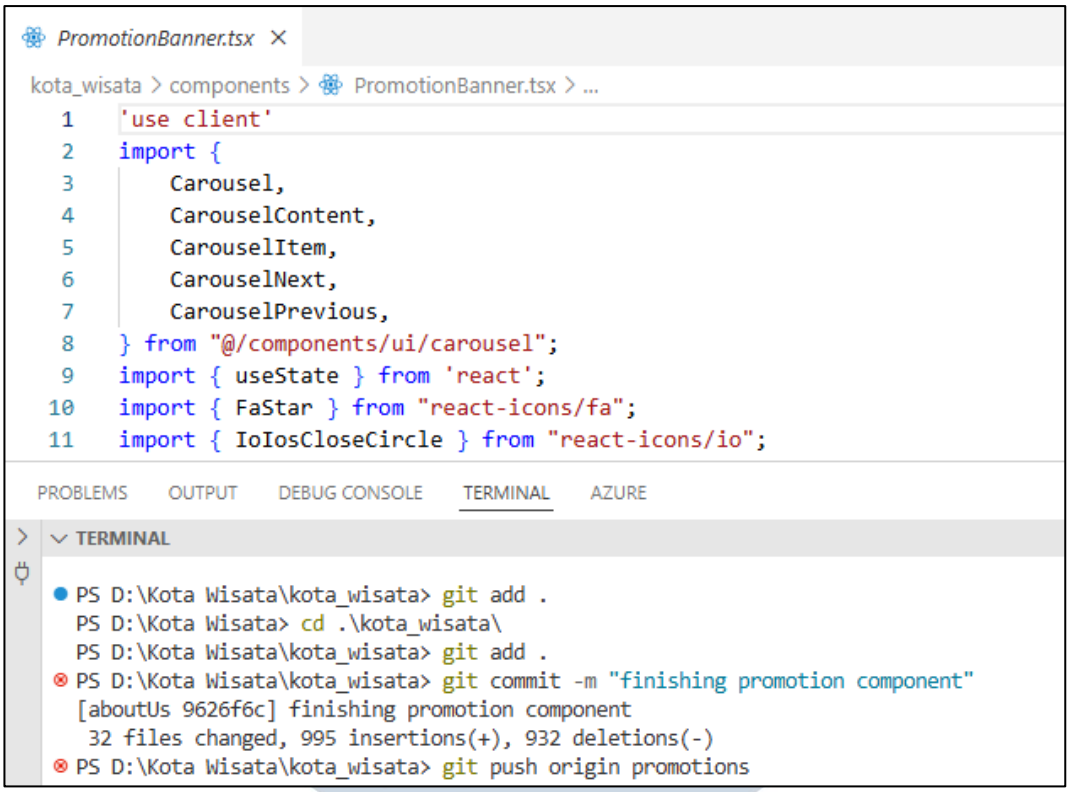
Gambar 3.29. Tampilan Repository Proyek di GitHub

Gambar 3.30 menampilkan contoh riwayat *commit* pada *repository* yang berfungsi sebagai catatan setiap perubahan yang dilakukan terhadap kode. Setiap *commit* memiliki pesan deskriptif untuk menjelaskan tujuan pembaruan, seperti penambahan fitur baru, perbaikan *bug*, atau penyempurnaan tampilan halaman. Dengan riwayat *commit* ini, seluruh proses pengembangan dapat ditelusuri dengan mudah, sehingga mempermudah pengelolaan versi serta proses *debugging* ketika terjadi kesalahan.



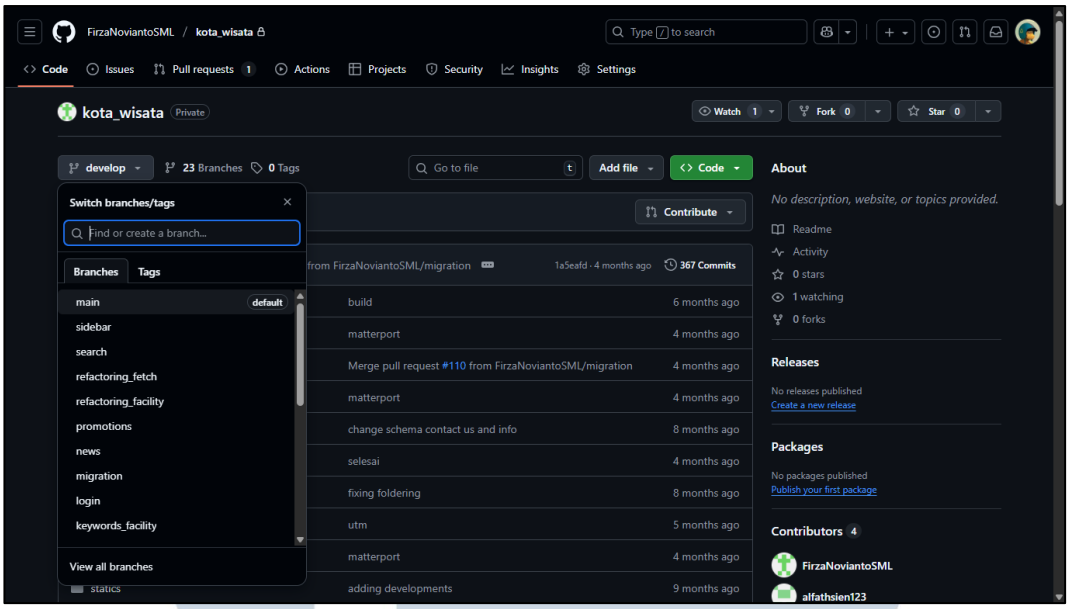
Gambar 3.30. Riwayat Commit pada Repository

Gambar 3.31 menampilkan proses unggah (*push*) kode dilakukan melalui terminal di *Visual Studio Code* (*VSCode*) menggunakan perintah *Git*. Setelah dilakukan pengujian lokal dan memastikan tidak ada *error*, pengembang menambahkan perubahan menggunakan *git add*, menyimpannya dengan *git commit*, dan mengunggahnya ke *repository GitHub* dengan *git push*. Integrasi ini membuat sinkronisasi antara kode lokal dan *repository* daring berjalan secara efisien dan *real-time*.



Gambar 3.31. Proses Push Kode dari VSCode ke GitHub

Gambar 3.32 memperlihatkan contoh tampilan *branch* pada *GitHub*. Dalam pengembangan proyek, fitur *branch* digunakan untuk memisahkan pekerjaan berdasarkan fitur atau modul tertentu agar tidak mengganggu *branch* utama (*main*). Misalnya, ketika menambahkan halaman baru atau memperbaiki *bug*, pengembang dapat membuat *branch* baru, melakukan perubahan, lalu menggabungkannya kembali ke branch utama melalui proses *pull request*. Penerapan sistem *branching* ini membantu menjaga stabilitas proyek sekaligus mendukung pengembangan paralel.



Gambar 3.32. Tampilan Branch pada Repository GitHub

Secara keseluruhan, penerapan *Git* dan *GitHub* dalam proyek *revamp website* Kota Wisata Cibubur memberikan manfaat besar dalam hal pengelolaan kode sumber, kolaborasi tim, dan dokumentasi perubahan. Setiap pembaruan dapat dipantau dengan jelas, versi kode dapat dikendalikan dengan baik, dan proses pengembangan berjalan lebih efisien, terstruktur, serta terdokumentasi secara profesional.

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala yang Ditemukan

1. Kendala dalam proses pengambilan data dari backend ke frontend
   - Struktur data API tidak konsisten.
   - Respons API dari *Strapi CMS* sering terlambat.
   - Dokumentasi sistem lama kurang lengkap sehingga proses *data fetching* sering gagal.
2. Error pada proses data fetching
   - Data dinamis tidak muncul pada halaman tertentu.
   - Terjadi error ketika struktur JSON berubah.
   - Membutuhkan debugging berulang dan bantuan tim backend.

3. Masalah responsivitas tampilan (UI/UX)
    - Perbedaan tampilan pada beberapa resolusi perangkat.
    - Elemen antarmuka tidak konsisten pada versi mobile.
    - Breakpoint Tailwind CSS harus disesuaikan ulang.
4. Keterbatasan referensi dari sistem lama berbasis WordPress
    - Tidak semua aset dan struktur konten dapat digunakan kembali.
    - Beberapa modul harus dibangun ulang dari awal.
    - Membutuhkan waktu untuk mempelajari user flow lama sebelum direkonstruksi di Next.js.
5. Kendala kolaborasi tim
    - Ketidaksesuaian antara dokumentasi teknis dan implementasi aktual.
    - Muncul miskomunikasi kecil yang menyebabkan revisi fitur.
    - Terjadi *merge conflict* ketika penggabungan *branch*.
6. Kendala saat proses code review
    - Waktu tambahan dibutuhkan jika ada perubahan besar.
    - Penyesuaian ulang pada struktur kode agar selaras dengan *main branch*.

3.4.2 Solusi atas Kendala yang Ditemukan

1. Memperbaiki proses integrasi data frontend–backend
    - Mempelajari struktur API Strapi secara menyeluruh.
    - Membaca dokumentasi resmi Strapi dan referensi proyek sebelumnya.
    - Berdiskusi aktif dengan tim backend untuk memastikan alur data.
2. Mengatasi error pada proses data fetching
    - Melakukan debugging menggunakan *console log* dan pengecekan JSON.
    - Mengatur parameter *populate* sesuai kebutuhan data.
    - Menambahkan *error handling* pada setiap permintaan API.
3. Memperbaiki tampilan responsif UI/UX
    - Menggunakan pendekatan *mobile-first design*.
    - Menyesuaikan breakpoint dan layout menggunakan Tailwind CSS.

- Menguji tampilan melalui berbagai perangkat dan *developer tools*.

4. Mengatasi kompatibilitas dengan sistem lama
   - Melakukan analisis komparatif antara WordPress lama dan Next.js baru.
   - Mengidentifikasi bagian yang bisa diadaptasi dan yang harus dibangun ulang.
   - Menyusun ulang konten agar lebih terstruktur dalam sistem baru.
5. Peningkatan kolaborasi dalam pengembangan
   - Mengikuti proses *pull request* secara rutin.
   - Melakukan *code review* untuk memastikan standar tim terpenuhi.
   - Menyelesaikan *merge conflict* dan menata ulang struktur kode.
6. Meningkatkan kualitas stabilitas fitur
   - Melakukan penyelarasan branch secara berkala.
   - Menggunakan pola commit yang konsisten.
   - Melakukan tes manual setiap kali fitur selesai dibuat.