BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama melakukan pelaksanaan kegiatan praktik kerja magang di PT Winnicode Garuda Teknologi, lebih tepatnya berada di divisi *Development* dengan posisi sebagai *Fullstack Developer*. Pelaksanaan magang tersebut berada di bawah pengawasan Muhamad Widyantoro selaku Supervisor divisi Development PT Winnicode Garuda Teknologi yang bertugas untuk mengarahkan dan membimbing agar proyek dan tugas yang diberikan dapat dijalankan sesuai dengan apa yang diharapkan. Selain itu, perusahaan memberikan kebebasan untuk mengeksplorasi solusi dan teknologi terbaru yang berhubungan dengan pengembangan *website Fullstack*. Kebebasan ini membantu untuk mendapatkan pengalaman yang baru dalam mengembangkan suatu *website*.

3.2 Tugas yang Dilakukan

Dalam pelaksanaan kerja magang di PT Winnicode Garuda Teknologi sebagai *Fullstack Developer*, pengerjaan proyek dapat dikerjakan setiap harinya. Kegiatan asistensi proyek dilakukan setiap 1 minggu sekali dengan mengumpulkan *logbook* yang akan dinilai langsung oleh supervisor. Adapun tanggung jawab yang diberikan dalam pengembangan proyek website portal berita ini, antara lain :

1. Membuat design *prototype* berupa tampilan UI/UX dari *website* portal berita dengan menggunakan Figma.
2. Bertugas membuat fitur-fitur yang diperlukan dalam website portal berita.

3.3 Uraian Pelaksanaan Magang

Selama menjalani kegiatan magang di PT Winnicode Garuda Teknologi, berbagai pekerjaan telah dilaksanakan sebagai *Fullstack Developer* pada Departemen *Development*. Rincian pelaksanaan tugas selama masa magang dapat dilihat pada Tabel3.1.

Tabel 3.1. *Timeline* pekerjaan yang dilakukan

Minggu Ke -	Pekerjaan yang dilakukan
1	Melakukan riset untuk menyiapkan ide dan refrensi untuk mengembangkan proyek website portal berita
2	Mengerjakan proposal untuk proyek yang akan dikerjakan
3-4	Membuat desain *wireframe* dan *prototype* untuk website portal berita menggunakan Figma.
5	Penyiapan *repository* pada *GitHub*, pengaturan *backend* menggunakan *Laravel* dan *MySQL*, pengaturan *frontend* dengan *ReactJS*.
6	Pengerjaan proyek website Portal Berita berupa halaman *login* dan *Sign Up*.
7	Membuat Home Page untuk proyek website Portal Berita
8	Pengerjaan proyek website Portal Berita berupa halaman *login.* dan *Sign Up*
9	Membuat fitur-fitur *Data User* dan *Data Profile* yang ditampilkan di *Profile Page*
10	Membuat *page Contact Us* dan menarik *NewsAPI* agar berita yang ditampilkan berupa berita asli.
11	Membuat *page About Us* dan membuat page *Admin* agar pesan-pesan yang diberikan melalui *page Contact Us* dapat dilihat oleh *Admin*.
12	Melakukan penarikan data dari *API* Winnicode dan mengintegrasikan *API* tersebut agar data yang tersedia dapat ditampilkan pada halaman utama (*landing page*) web
13	Membuat halaman *NewsPage* yang berfungsi untuk menampilkan seluruh data berita yang diperoleh dari API Winnicode.
14	Merancang fungsi *filter* kategori yang bertujuan untuk menyaring berita berdasarkan kategori, serta membuat halaman kategori agar pengguna dapat melihat berita sesuai dengan kategori yang dipilih.
15	Merancang halaman *NewsDetail* untuk menampilkan informasi lengkap dari berita yang diambil melalui API Winnicode.
16	Menambahkan fitur komentar pada halaman NewsDetail, yang hanya dapat digunakan oleh *user* yang sudah melakukan login.

3.4 Proses Pelaksanaan

Pelaksanaan kegiatan magang ini dilakukan melalui beberapa tahapan yang dimulai dari penentuan ide, perancangan sistem dan *database*, perancangan desain aplikasi, dan implementasi pengembangan *website*. Setiap tahapan dilakukan secara berurutan dan saling berkaitan untuk memastikan tercapainya tujuan pembangunan *website* portal berita yang berkualitas dan sesuai dengan kebutuhan pengguna.

3.4.1 Menentukan Ide

Langkah awal yang dilakukan dalam pelaksanaan proyek pembangunan website portal berita adalah melakukan riset dan menentukan ide dasar serta mengumpulkan beberapa refrensi yang relevan. Riset ini dilakukan untuk memahami kebutuhan pengguna, tren desain antarmuka yang *modern*, serta fitur fitur yang umum diterapkan pada portal berita *digital*.

Dalam proses riset ini, dilakukan analisis terhadap berbagai situs portal berita yang sudah populer seperti Kompas, Detik, CNN Indonesia sebagai acuan. Selain itu, refrensi juga diperoleh melalui artikel, video *Youtube*, dan dokumentasi pengembangan web untuk memperdalam pemahaman mengenai teknologi yang digunakan, seperti *ReactJS* untuk sisi *Frontend* dan *Laravel* untuk sisi *Backend*. Hasil dari riset tersebut menjadi dasar dalam perencanaan fitur, alur navigasi, tampilan *UI/UX* pada website yang akan dikembangkan

3.4.2 Menyusun Proposal Proyek

Setelah ide dan referensi proyek berhasil dikumpulkan, tahap berikutnya adalah menyusun proposal proyek yang akan dikerjakan. Proposal ini berperan sebagai dokumen awal yang memberikan gambaran umum kepada perusahaan mengenai sistem website portal berita yang akan dibangun, sekaligus menjelaskan tujuan serta rencana pelaksanaan proyek kepada seluruh pihak perusahaan.

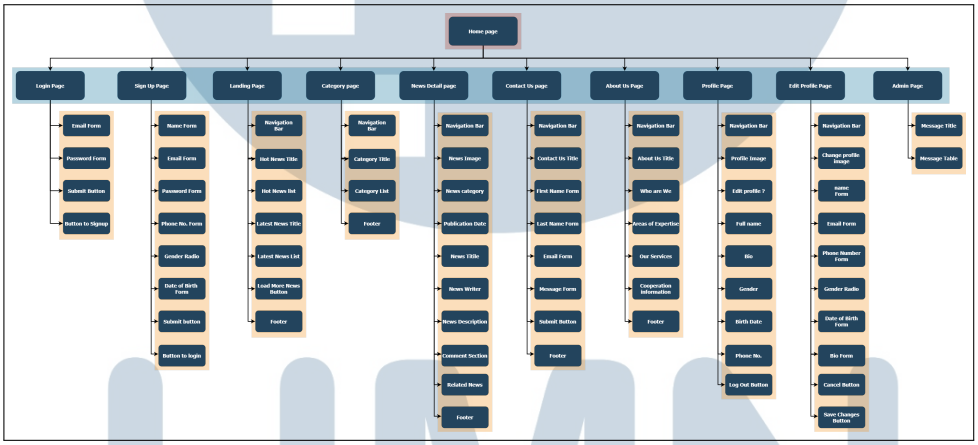
3.4.3 Membuat perancangan desain Aplikasi

Perancangan desain merupakan tahap penting dalam pengembangan sebuah aplikasi. Pembuatan desain aplikasi dibuat untuk memudahkan perancangan dan pembangunan website portal berita ini. Proses ini diawali dengan pembuatan *Sitemap* yang bertujuan untuk menggambarkan alur navigasi dan isi dari tiap

halaman yang akan dibuat. Selanjutnya, dilakukan pembuatan *wireframe* sebagai dasar awal dalam tahap melakukan peletakan elemen-elemen utama dalam *website*. Terakhir, yaitu melakukan tahapan hasil akhir desain yang dimana sudah menampilkan tampilan *visual* lengkap dengan warna, tipografi, dan elemen grafis lainnya. Seluruh tahapan ini dilakukan guna menghasilkan antarmuka yang efisien, informatif, dan nyaman digunakan.

A Sitemap

Sitemap disusun untuk memetakan struktur halaman aplikasi secara menyeluruh, sehingga alur navigasi dapat tergambar dengan jelas. Tahap ini membantu dalam mengidentifikasi hierarki informasi dan hubungan antarfitur. Gambar hasil *sitemap* dapat dilihat pada Gambar 3.1.

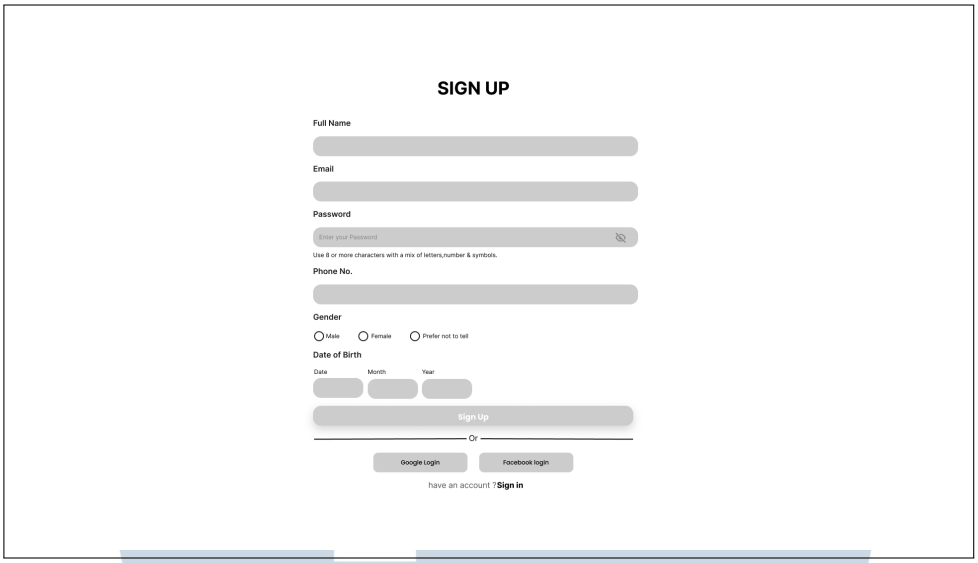


Gambar 3.1. Sitemap Aplikasi Web Portal Berita

B Wireframe

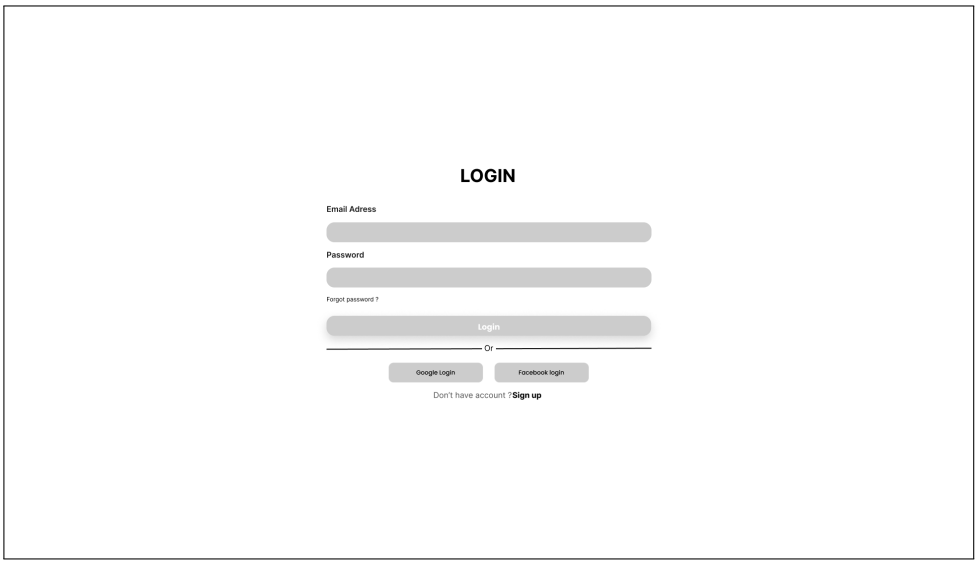
Wireframe dibuat sebagai kerangka awal tampilan antarmuka tanpa elemen *visual* seperti warna atau gambar. Fokus utama pada tahap ini adalah menempatkan komponen-komponen penting agar fungsionalitas dan kenyamanan pengguna dapat diuji sejak awal. Hasil rancangan wireframe dapat dilihat pada Gambar 3.2 sampai 3.10.

Gambar 3.2 merupakan desain untuk halaman *Sign Up* yang dirancang untuk mempermudah pengguna untuk mendaftarkan akun ke dalam website portal berita. Halaman *Sign up* terdapat form pengisian nama, *email*, *password*, nomor telepon, jenis kelamin, tanggal lahir.

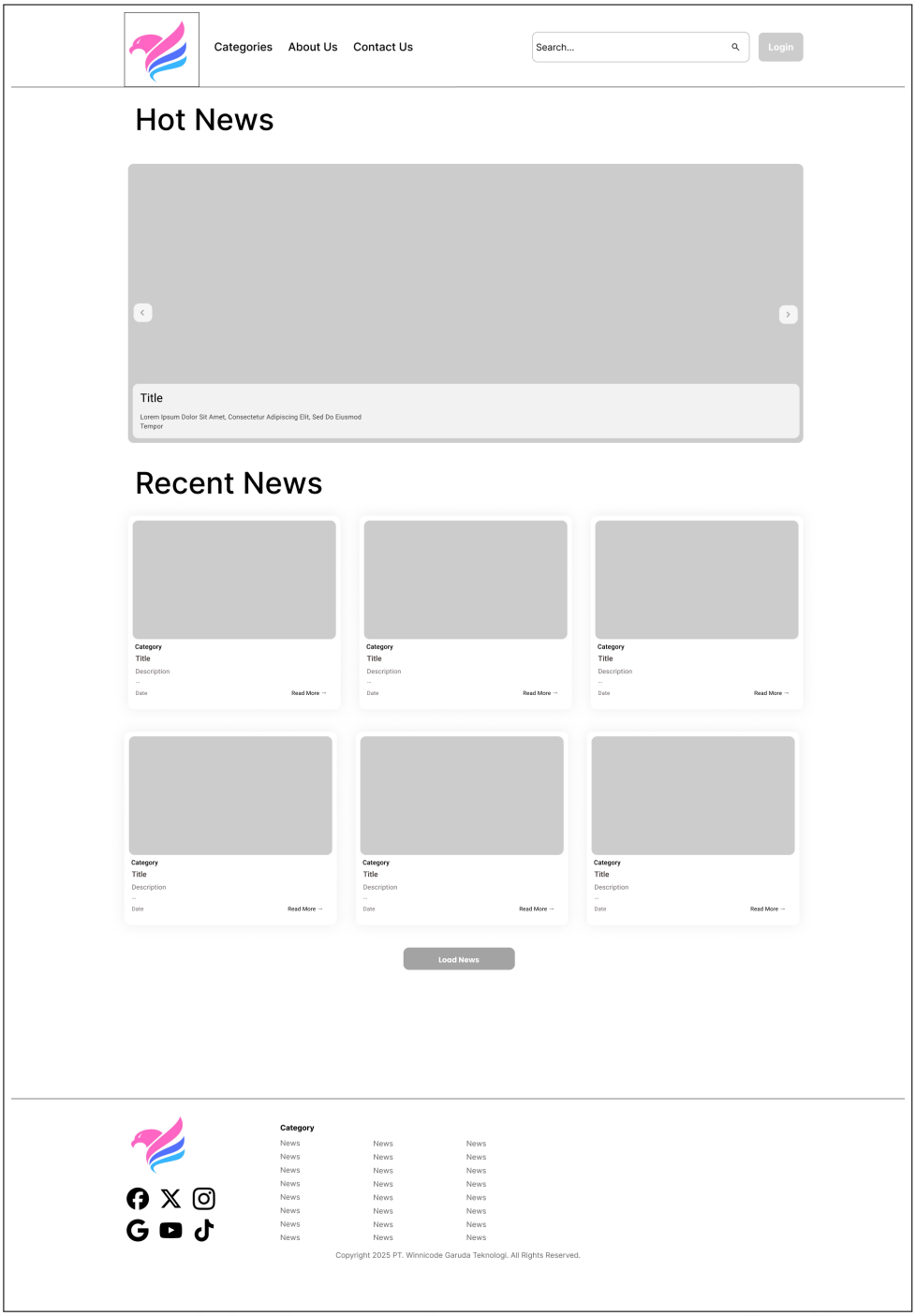


Gambar 3.2. *Wireframe* Desain Halaman *Signup*

Gambar 3.3 merupakan hasil rancangan desain untuk halaman Login yang dapat digunakan oleh pengguna yang sudah terdaftar untuk dapat masuk ke akun masing-masing. Pengguna harus memasukkan *email* dan *password* yang benar di *form* yang tersedia untuk menjaga keamanan akses.



Gambar 3.3. *Wireframe* Desain Halaman *Login*

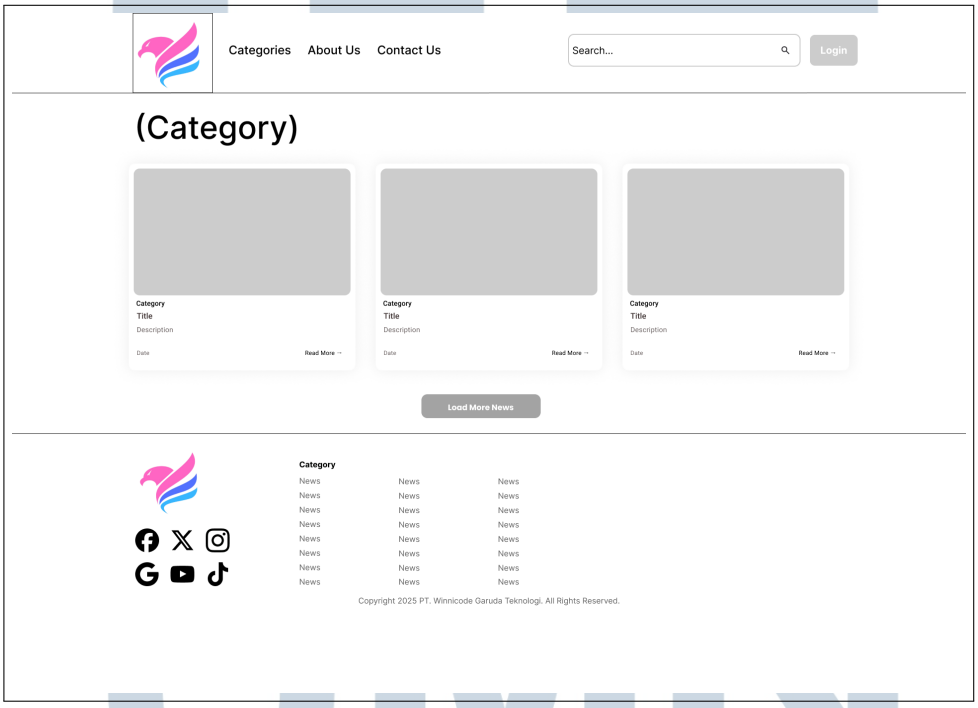


Gambar 3.4. *Wireframe* Desain Halaman Utama

Gambar 3.4 menggambarkan Halaman utama dirancang untuk menyajikan rangkuman berita utama dan terkini secara efisien. Bagian atas menampilkan segmen ”*Hot News*” dengan desain *carousel* yang menonjolkan satu berita besar lengkap dengan judul dan deskripsi singkat, dilengkapi dengan navigasi panah

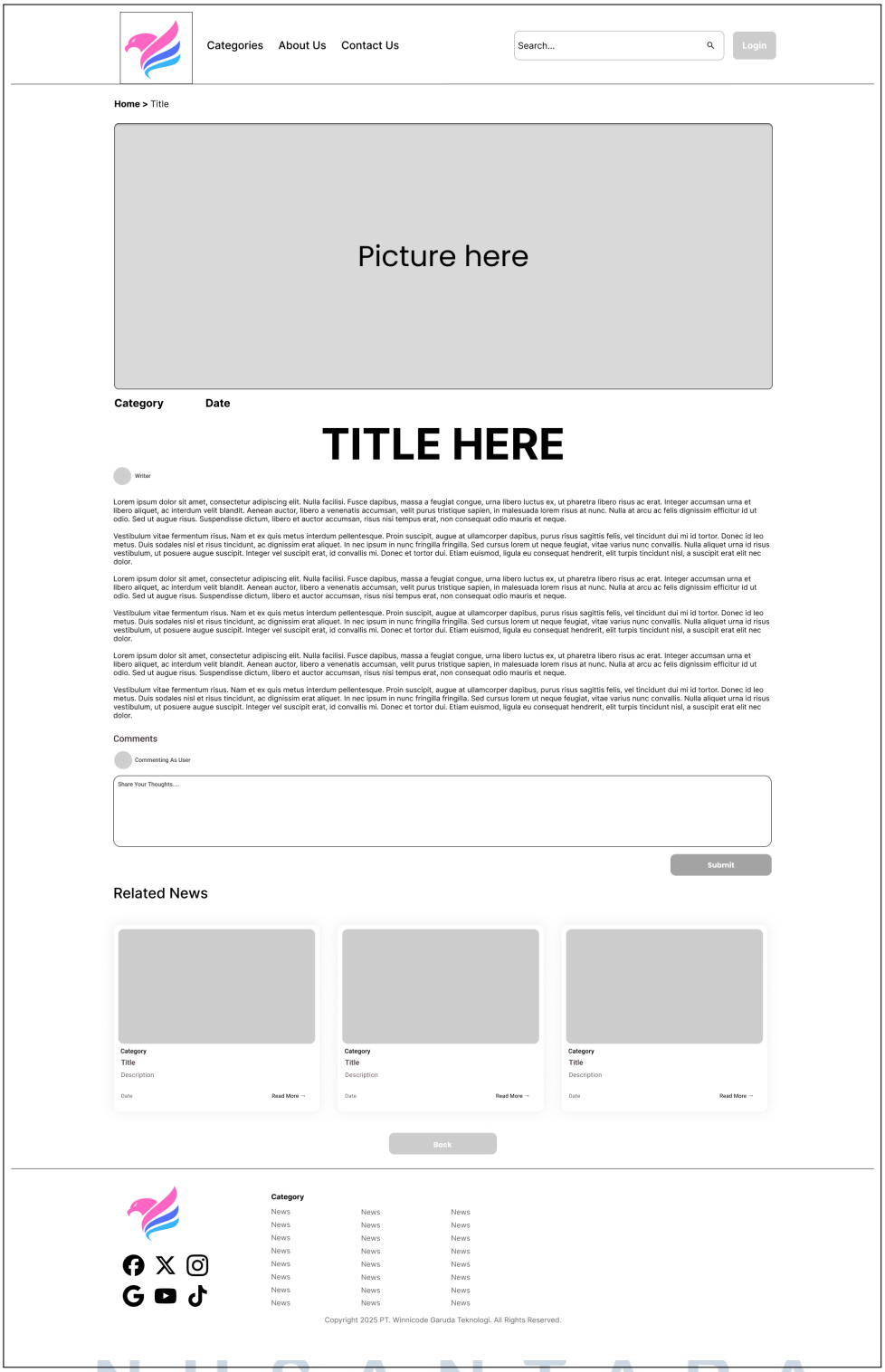
untuk melihat berita lainnya. Di bawahnya, terdapat bagian ”*Recent News*” yang menyajikan koleksi berita-berita terbaru dalam format kartu *grid*, menampilkan kategori, judul, deskripsi, dan tanggal rilis. Tombol ”*Load News*” disediakan untuk memuat lebih banyak berita, memastikan pengguna dapat dengan mudah menemukan konten yang relevan.

Gambar 3.5 merupakan rancangan desain untuk halaman kategori yang menampilkan daftar berita berdasarkan kategori yang dipilih oleh pengguna. Setiap kartu pada halaman tersebut berisi judul, deskripsi, dan tautan “*Read More*” untuk membaca berita selengkapnya.



Gambar 3.5. *Wireframe* Desain Halaman isi berita per kategori

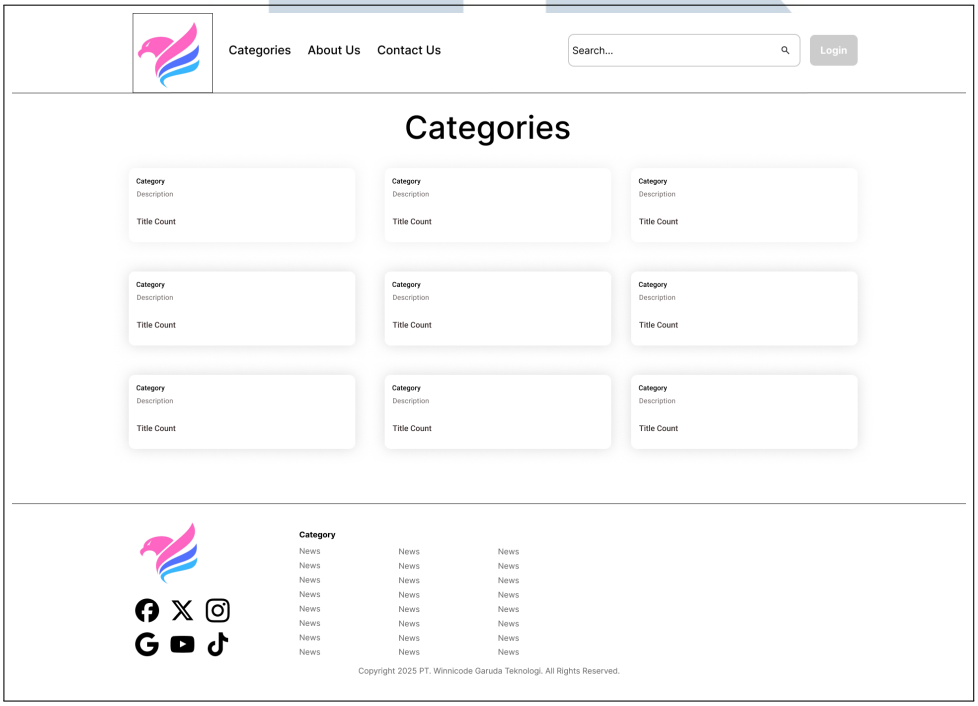
Gambar 3.6 atau halaman detail dirancang dengan tata letak yang bersih dan simpel, menonjolkan bagian utama untuk ”*Picture here*” sebagai poin utama yang menonjol. Di bawahnya, terdapat informasi kategori dan tanggal, diikuti oleh bagian ”*TITLE HERE*” yang menjadi fokus utama judul artikel. Desain ini bertujuan untuk menyajikan informasi terkini secara jelas dan menarik, dengan fitur komentar serta bagian ”Related News” di bagian bawah untuk mendorong eksplorasi konten lebih lanjut.



Gambar 3.6. *Wireframe* Desain Halaman isi detail berita

Gambar 3.7 menggambarkan desain dari halaman Kategori yang dirancang

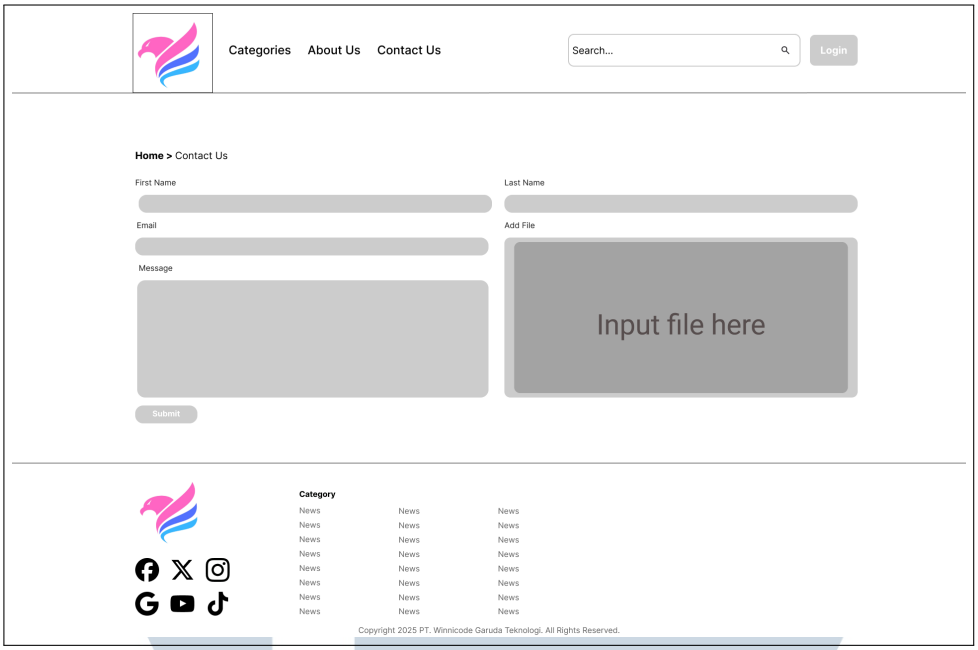
untuk menampilkan daftar kategori konten yang tersedia, masing-masing disajikan dalam kartu terpisah. Setiap kartu kategori menyertakan deskripsi singkat dan jumlah judul (*Title Count*) yang terkait, memudahkan pengguna untuk menjelajahi dan menemukan topik yang diminati.



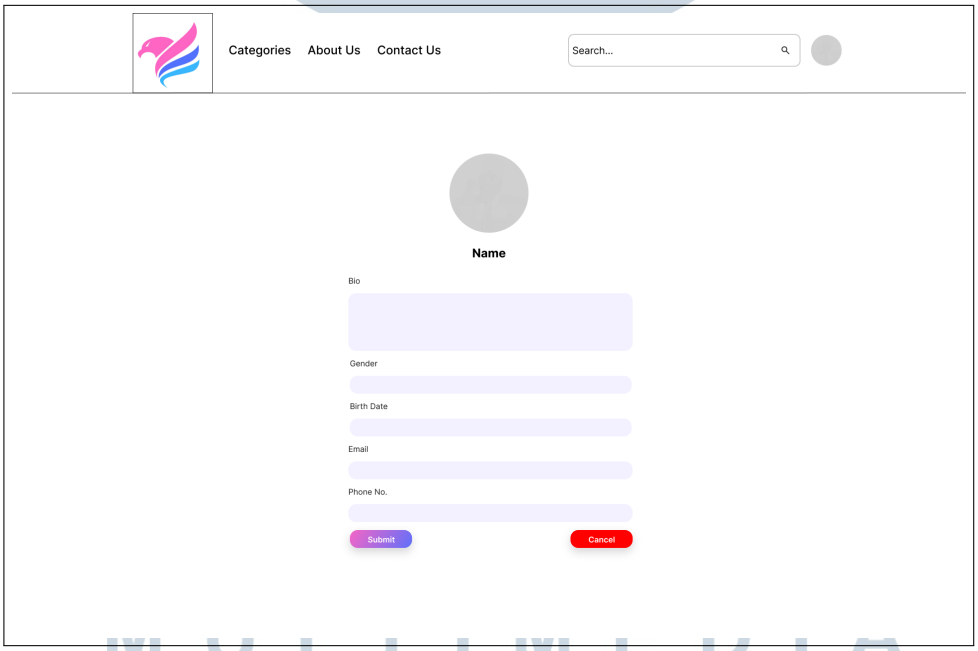
Gambar 3.7. *Wireframe* Desain Halaman Kategori

Gambar 3.8 merupakan desain untuk halaman *Contact Us* dirancang untuk memfasilitasi komunikasi pengguna dengan menyediakan formulir kontak yang intuitif. Pengguna dapat mengisi nama, *email*, pesan, serta melampirkan berkas, kemudian mengirimkannya melalui tombol *Submit*.

Gambar 3.9 merupakan desain dari halaman *Edit Profile* yang dibuat agar pengguna dapat dengan mudah memperbarui informasi pribadi pengguna, seperti *bio*, gender, tanggal lahir, *email*, dan nomor telepon. Setelah perubahan dilakukan, pengguna dapat menyimpannya dengan tombol *Submit* atau membatalkan dengan tombol *Cancel*.

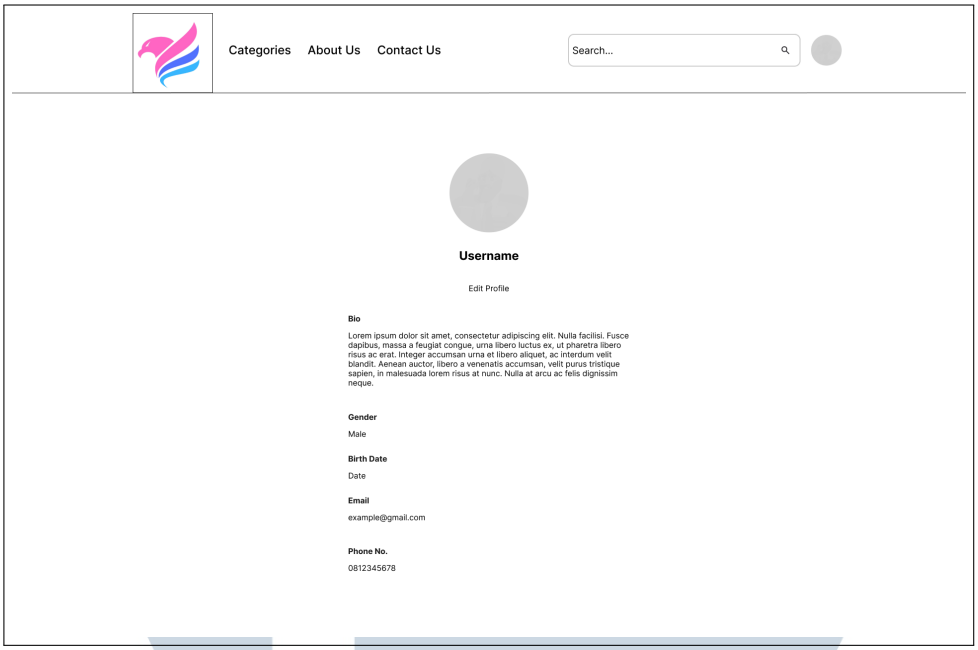


Gambar 3.8. *Wireframe* Desain Halaman *Contact Us*



Gambar 3.9. *Wireframe* Desain Halaman *Edit Profile*

Gambar 3.10 memberikan gambaran Halaman Profile yang berguna untuk menampilkan dan mengelola informasi pengguna yang sudah melakukan *sign up* dan *login*. Pengguna dapat melihat dan menyunting data diri sesuai kebutuhan.



Gambar 3.10. *Wireframe* Desain Halaman *Profile*

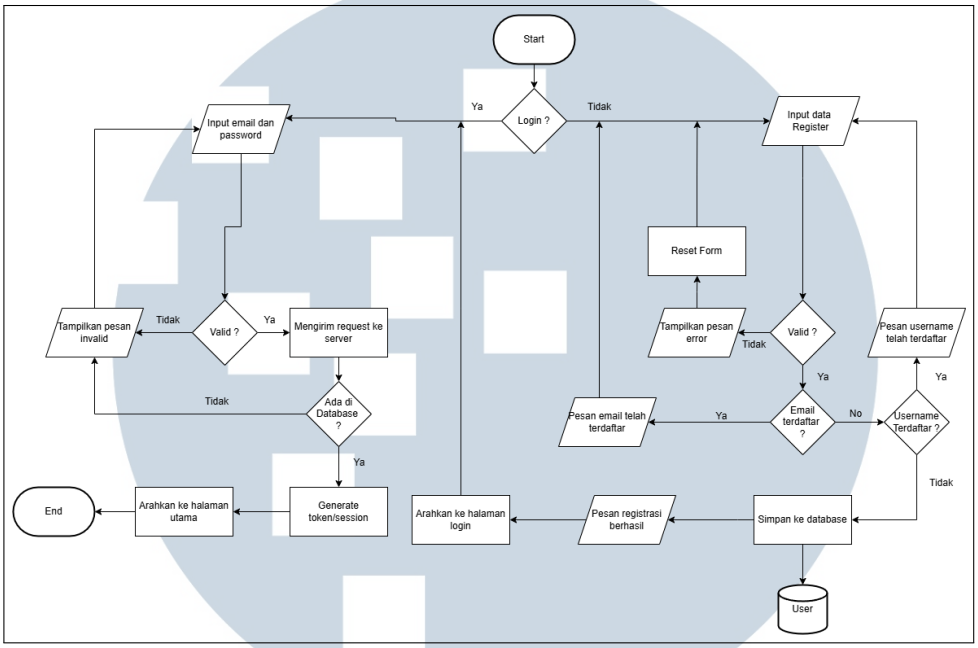
3.4.4 Flowchart Sistem Aplikasi

Untuk memberikan gambaran yang jelas mengenai alur kerja sistem dan proses bisnis yang terjadi dalam *website* portal berita, diperlukan visualisasi berupa *flowchart* yang menggambarkan langkah-langkah sistematis dari setiap fungsi utama aplikasi. *Flowchart* ini berfungsi sebagai panduan *visual* yang memudahkan pemahaman terhadap alur logika sistem, mulai dari proses autentikasi pengguna, proses komentar, proses sistem *filter* dan kategori, dan proses pengiriman pesan melalui halaman *Contact Us*. Pembuatan *flowchart* ini juga membantu dalam dokumentasi sistem dan mempermudah proses *maintenance* atau pengembangan fitur tambahan di masa mendatang. Adapun *flowchart* yang dibuat mencakup seluruh proses kritis yang terjadi dalam sistem portal berita ini.

A Flowchart Sistem Login dan Sign up

Flowchart sistem *login* dan *register* menggambarkan alur autentikasi pengguna yang mencakup dua jalur utama, yaitu proses masuk untuk pengguna yang telah terdaftar dan proses pendaftaran untuk pengguna baru. Sistem melakukan validasi data dan kredensial pada kedua proses tersebut, kemudian memberikan respons berupa akses ke halaman utama jika berhasil atau pesan *error*

jika terjadi kesalahan dalam proses autentikasi. Rincian *Flowchart* sistem *login* dan *register* dapat dilihat pada Gambar 3.11.

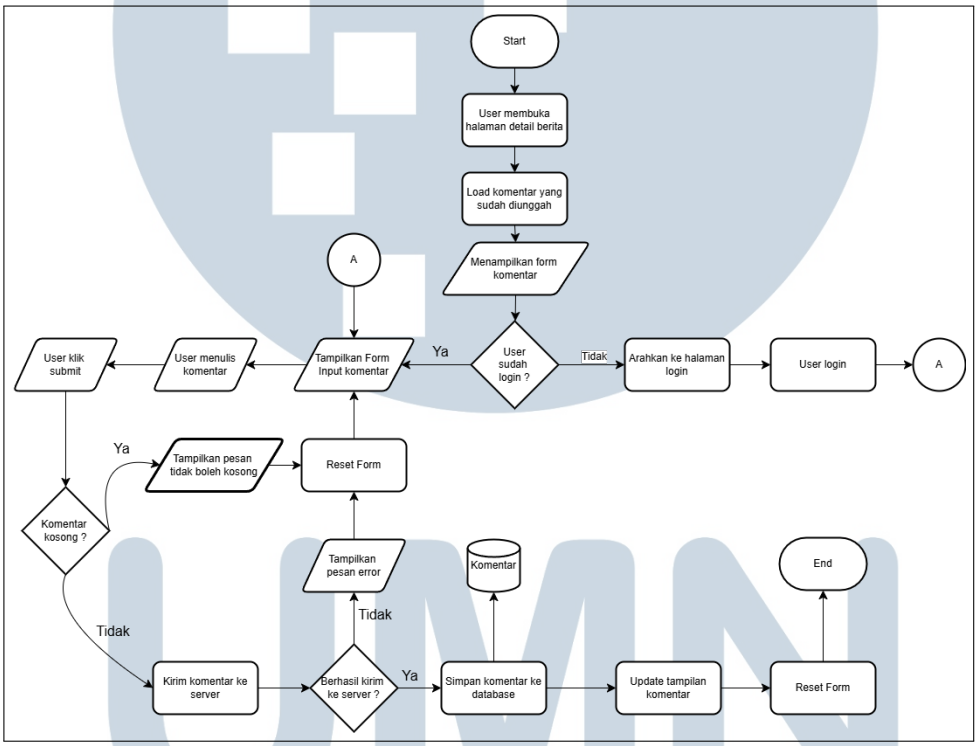


Gambar 3.11. *Flowchart Login* dan *Sign Up*

Proses autentikasi dalam sistem portal berita dimulai ketika pengguna mengakses halaman login. Pada tahap awal, pengguna diberikan dua pilihan utama yaitu melakukan *login* untuk pengguna yang sudah memiliki akun atau melakukan registrasi untuk pengguna baru. Jika pengguna memilih opsi *login*, sistem akan meminta *input* berupa *email* dan *password*, kemudian melakukan validasi terhadap data yang dimasukkan. Setelah validasi *input* berhasil, sistem akan mengirimkan *request* ke server untuk memverifikasi kredensial dengan data yang tersimpan dalam *database*. Apabila kredensial yang dimasukkan sesuai dengan data di *database*, sistem akan membuat *token* atau *session* sebagai penanda bahwa pengguna telah berhasil masuk dan mengarahkan pengguna ke halaman utama. Sebaliknya, jika terjadi kesalahan dalam proses *login*, sistem akan menampilkan pesan *error* dan memberikan kesempatan kepada pengguna untuk mencoba kembali. Untuk proses registrasi, sistem melakukan validasi *input* untuk melakukan cek apakah *email* yang didaftarkan sudah terdaftar atau belum. Jika seluruh data *valid* dan *email* belum terdaftar, sistem akan menyimpan data pengguna baru ke *database* dan mengarahkan ke halaman *login* untuk melakukan autentikasi.

B Flowchart Sistem Komentar

Flowchart sistem komentar menggambarkan alur interaksi pengguna dalam memberikan komentar pada halaman detail berita, dimana sistem melakukan pengecekan status *login* pengguna sebelum mengizinkan akses ke fitur komentar. Proses ini mencakup validasi *session* pengguna, validasi input komentar, penyimpanan data ke *database*, dan pembaruan tampilan komentar secara *real-time* setelah komentar berhasil ditambahkan. Rincian *Flowchart* sistem *login* dan *register* dapat dilihat pada Gambar 3.12.



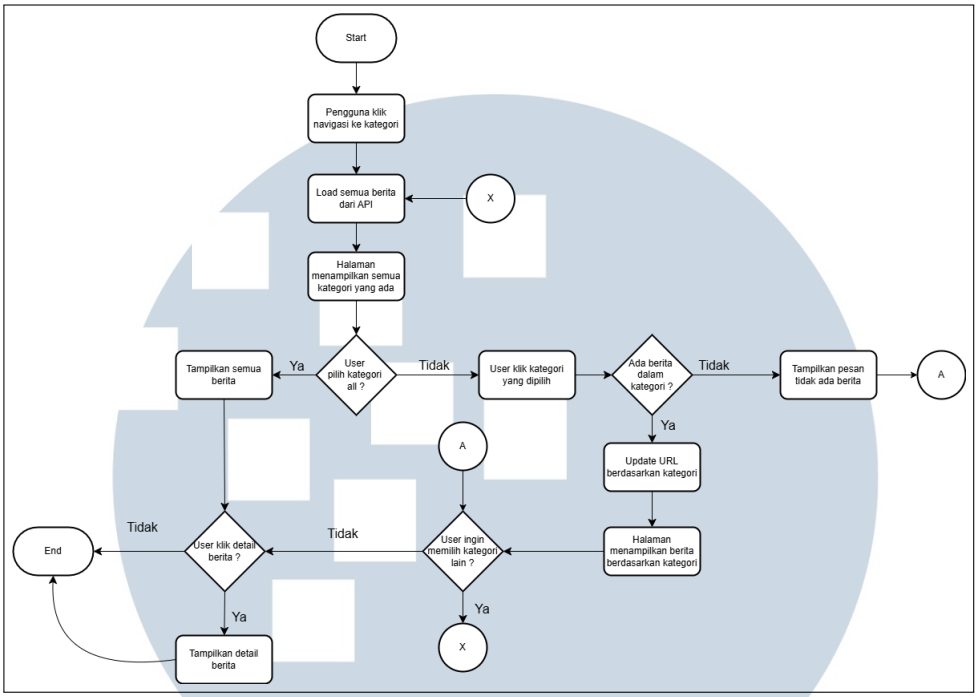
Gambar 3.12. *Flowchart* komentar

Fitur komentar pada halaman detail berita dirancang dengan sistem keamanan yang memastikan hanya pengguna yang telah terautentikasi yang dapat memberikan komentar. Ketika pengguna mengakses halaman detail berita, sistem akan memuat komentar yang sudah ada dan menampilkan formulir komentar. Sebelum mengizinkan pengguna untuk menulis komentar, sistem melakukan pengecekan status *login* pengguna. Jika pengguna belum *login*, sistem akan menampilkan pesan yang mengarahkan pengguna untuk masuk terlebih dahulu dan secara otomatis mengalihkan ke halaman *login*. Setelah pengguna berhasil

login dan kembali ke halaman detail berita, formulir komentar akan aktif dan dapat digunakan. Proses penambahan komentar dimulai ketika pengguna menulis komentar dan menekan tombol kirim. Sistem kemudian melakukan validasi untuk memastikan komentar tidak kosong. Jika validasi berhasil, data komentar akan disimpan ke *database* beserta informasi pengguna dan waktu pembuatan. Setelah penyimpanan berhasil, sistem akan memperbarui tampilan halaman untuk menampilkan komentar baru tanpa perlu me-*refresh* halaman, sehingga memberikan pengalaman pengguna yang lebih responsif.

C Flowchart Sistem Filter dan Kategori

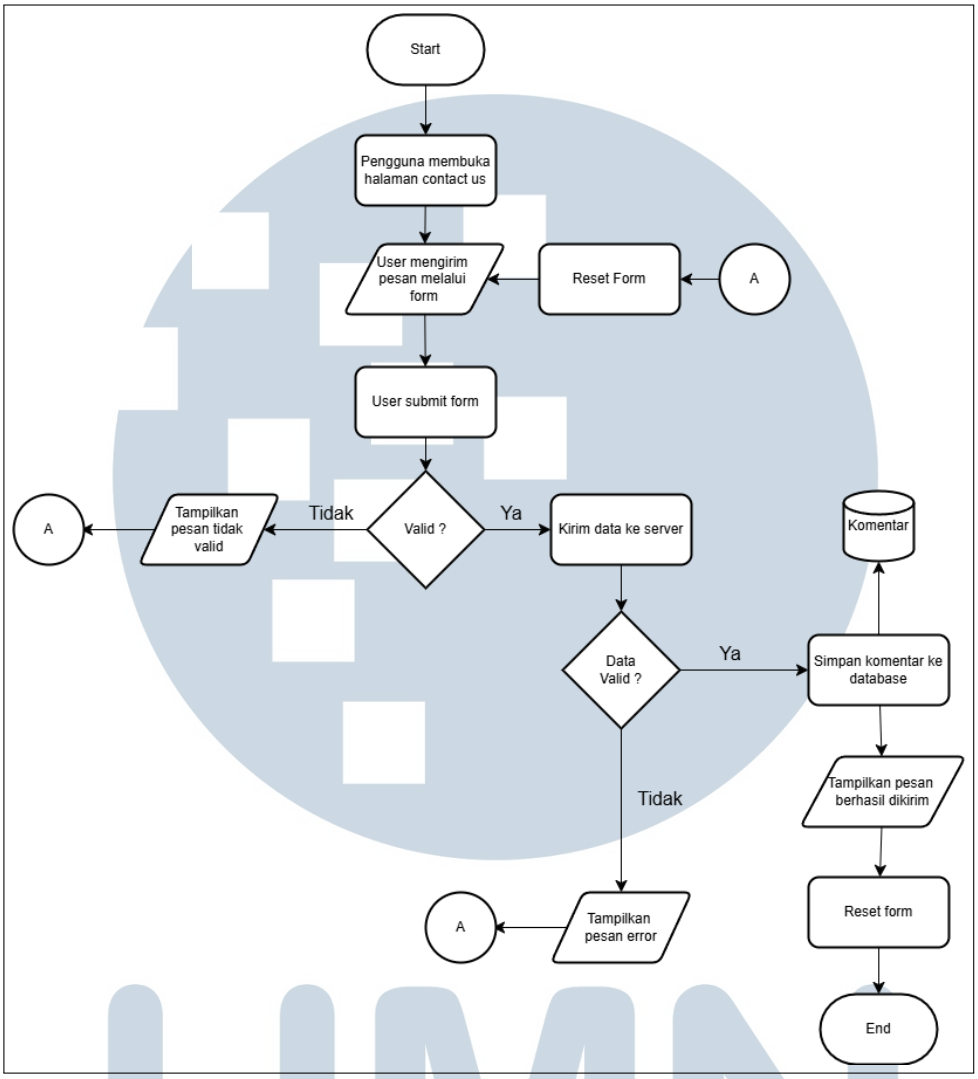
Flowchart Sistem *filter* kategori berita memungkinkan pengguna untuk menyaring konten berita sesuai dengan minat atau topik tertentu. Proses dimulai ketika pengguna mengakses halaman utama dan sistem memuat seluruh data berita dari *API* eksternal. Pada tahap ini, *website* juga menampilkan daftar kategori yang tersedia sebagai opsi *filter*. Pengguna dapat memilih untuk melihat semua berita atau memilih kategori spesifik yang diminati. Ketika pengguna memilih kategori tertentu, *website* akan mengambil parameter kategori tersebut dan melakukan validasi untuk memastikan kategori yang dipilih *valid* dan tersedia. Setelah validasi berhasil, *website* melakukan proses *filtering* dengan menyaring data berita berdasarkan kategori yang dipilih. Jika terdapat berita dalam kategori tersebut, *website* akan menampilkan hasil *filter* kepada pengguna dan memperbarui *URL* dengan parameter kategori. Namun jika tidak ada berita dalam kategori yang dipilih, *website* akan menampilkan pesan informatif dan memberikan opsi untuk memilih kategori lain. Rincian *flowchart* sistem *login* dan *register* dapat dilihat pada Gambar 3.13.



Gambar 3.13. *Flowchart* kategori

D Flowchart Conctact Us

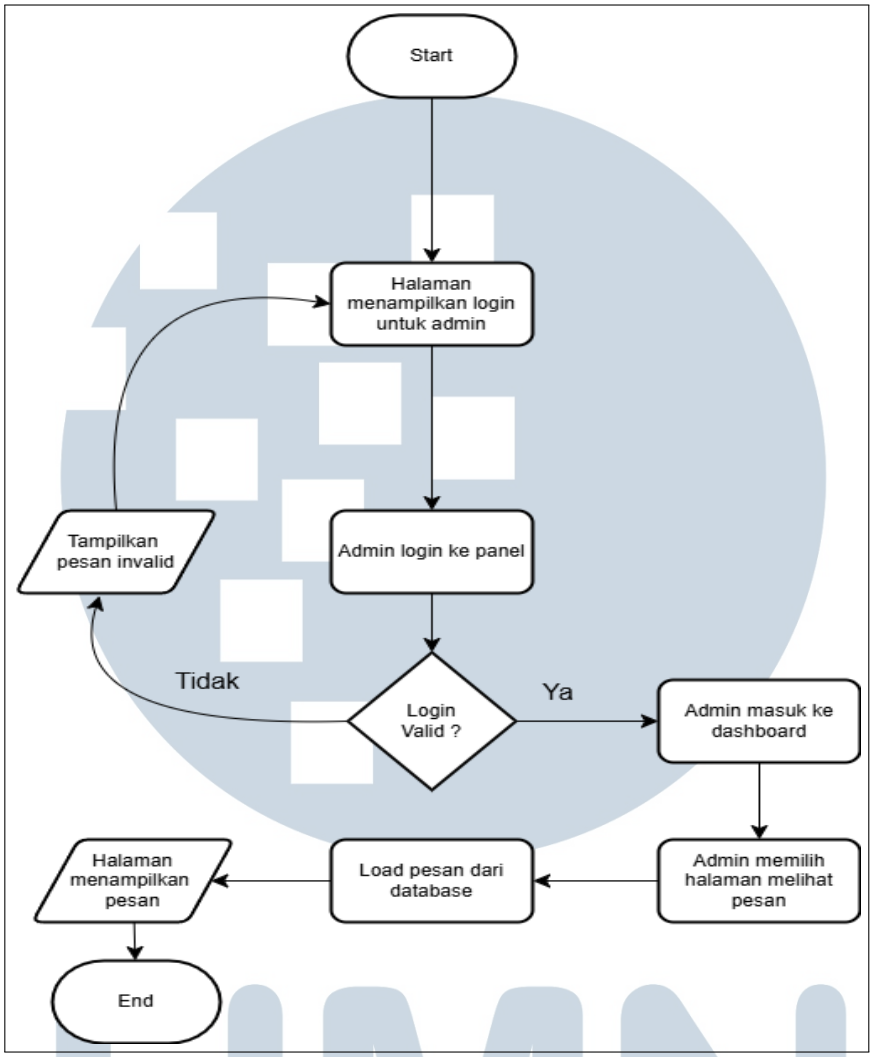
Flowchart Sistem *Contact Us* menyediakan saluran komunikasi antara pengguna dan administrator melalui formulir kontak yang terintegrasi dengan panel administrasi. Proses dimulai ketika pengguna mengakses halaman *Contact Us* dan mengisi formulir yang berisi informasi seperti nama awal, nama akhir, *email*, pesan, dan file yang ingin dikirim. Setelah pengguna mengirimkan formulir, sistem melakukan validasi *input* untuk memastikan semua *field* yang wajib telah diisi dengan format yang benar. Data yang telah tervalidasi kemudian dikirim ke server dan disimpan ke dalam *database*. Rincian *Flowchart* pengiriman pesan melalui halaman *Contact Us* dapat dilihat pada Gambar 3.14.



Gambar 3.14. *Flowchart* pengunjung mengirim pesan melalui *Contact Us*

E Flowchart Admin

Flowchart admin menggambarkan proses bagaimana *admin* dapat melihat pesan pesan yang dikirimkan oleh pengunjung web melalui halaman *Contact Us*. Gambaran detail dari *flowchart admin* dapat dilihat pada Gambar 3.15



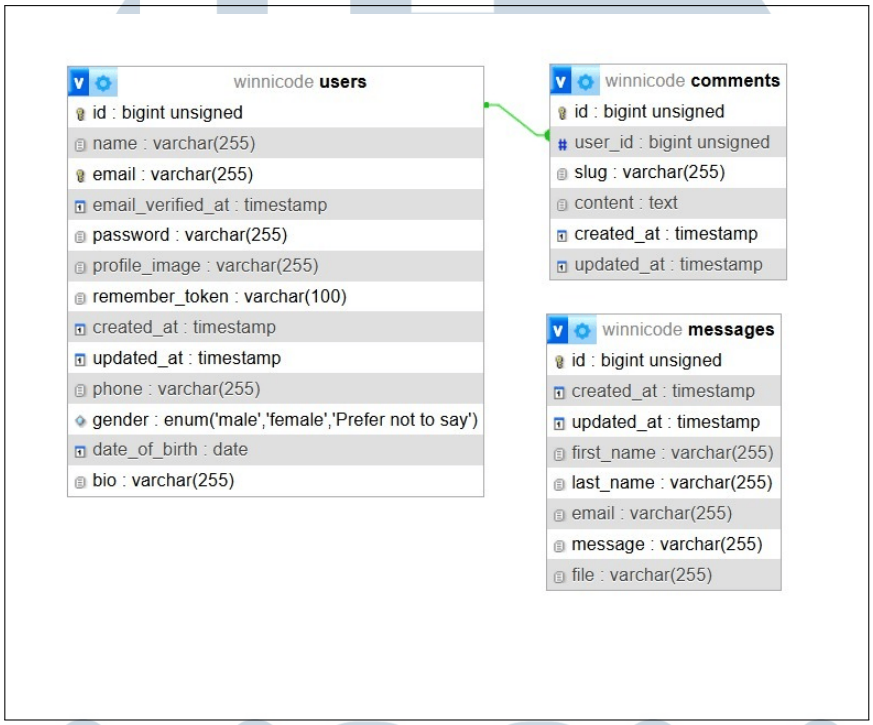
Gambar 3.15. *Flowchart admin* melihat pesan yang masuk

Sistem menyediakan panel khusus yang memungkinkan *admin* untuk mengelola pesan yang masuk. Administrator harus melakukan *login* dengan kredensial khusus untuk mengakses panel administrasi. Setelah berhasil masuk, *admin* dapat melihat daftar semua pesan yang diterima, termasuk detail pengirim dan isi pesan. Sistem ini memungkinkan administrator untuk mengelola komunikasi dengan pengguna secara efisien dan terorganisir.

3.4.5 Skema Database

Untuk mendukung fungsionalitas *website* Portal Berita, dibutuhkan skema *database* yang tersusun secara sistematis dan efisien. Database dirancang

menggunakan *MySQL* dengan mempertimbangkan kebutuhan utama sistem, seperti pengelolaan data pengguna, komentar, dan pesan dari pengguna. Setiap tabel disusun dengan struktur dan tipe data yang sesuai agar mendukung integrasi dengan *backend* dan *frontend*, serta memastikan data dapat diakses, disimpan, dan dikelola dengan optimal.



Gambar 3.16. *ERD database web* portal berita

Gambar 3.16 menggambarkan relasi antara tabel *users*, *comments*, dan *messages*, di mana satu pengguna (*users*) dapat memiliki banyak komentar (*comments*), sementara tabel *messages* berdiri sendiri sebagai wadah pesan dari pengunjung *website* tanpa keterkaitan langsung ke pengguna terdaftar. Struktur ini mendukung sistem komentar pengguna dan *form* pada halaman *Contact Us*.

A Skema Database Pengguna

Database pengguna merupakan salah satu komponen penting dalam sistem portal berita yang berfungsi untuk menyimpan informasi terkait pengguna yang terdaftar, baik sebagai pembaca maupun *admin*. Tabel ini dirancang untuk mendukung kebutuhan autentikasi dan pengelolaan data identitas pengguna secara efisien dan aman. Setiap atribut dalam tabel memiliki fungsi spesifik, seperti

id sebagai *primary key*, *name* untuk menyimpan nama lengkap, serta *email* dan *password* yang diperlukan dalam proses *login*. Selain itu, tabel ini juga mencakup atribut peran pengguna yang menentukan hak akses di dalam sistem. Dengan struktur data yang baik dan relasional, sistem dapat memberikan layanan yang optimal serta menjaga keamanan dan integritas data.

Struktur tabel ini juga mencakup informasi tambahan seperti *email_verified_at* yang mencatat waktu verifikasi *email*, *remember_token* untuk mendukung fitur *remember me*, serta *profile_image*, *phone*, *gender*, *date_of_birth*, dan *bio* yang memperkaya data profil pengguna. Atribut *created_at* dan *updated_at* secara otomatis mencatat waktu pembuatan dan pembaruan data, sejalan dengan standar pengembangan menggunakan *framework* Laravel. Dengan desain yang terstruktur dan efisien, skema tabel ini memungkinkan pengelolaan data pengguna yang optimal serta mendukung integrasi yang lancar dengan antarmuka *frontend* berbasis *ReactJS*. Adapun rincian dari tabel pengguna dapat dilihat pada Tabel 3.2.

Tabel 3.2. *Timeline* tabel skema *database* dari pengguna

Nama	**Tipe Data**	**Deskripsi**
id	Bigint(20)	*Primary key* unik pengguna
name	varchar(255)	Nama lengkap dari pengguna
email	varchar(255)	Alamat *email* pengguna
email_verified_at	timestamp	Waktu saat *email* pengguna telah diverifikasi
password	varchar(255)	Kata sandi pengguna yang telah dienkripsi
profile_image	varchar(255)	*Path* atau *URL* dari foto profil pengguna
remember_token	varchar(100)	*Token* yang digunakan saat login
created_at	timestamp	Tanggal dan waktu data dibuat
updated_at	timestamp	Tanggal dan waktu data diperbarui
phone	varchar(255)	Nomor telepon pengguna
gender	enum(*'male','female','prefer not to say'*)	Jenis kelamin pengguna yang dipilih
date_of_birth	date	Tanggal lahir dari pengguna
bio	varchar(255)	Deskripsi singkat pengguna

B Skema Database Pesan/Messages

Skema database pesan/*messages* dirancang khusus untuk menyimpan seluruh data komunikasi yang dikirimkan oleh pengguna melalui formulir di halaman *Contact Us* pada *website*. *Database* ini memiliki struktur yang komprehensif dengan delapan kolom utama yang mencakup identitas unik pesan melalui kolom *id* sebagai *primary key*, informasi lengkap pengirim yang terdiri dari *first name*, *last name*, dan *email*, serta konten komunikasi berupa *message* dan *file attachment* yang dapat diunggah oleh pengirim. Selain itu, tabel ini juga dilengkapi dengan *timestamp* berupa *created_at* dan *updated_at* yang berfungsi untuk mencatat waktu pembuatan dan pembaruan data secara otomatis, sehingga memungkinkan *admin* untuk melakukan *tracking* dan manajemen pesan dengan lebih efektif serta memastikan setiap komunikasi dari pengguna dapat tersimpan dan dikelola dengan baik.

Tabel 3.3. *Timeline* Tabel skema *database* dari pesan

Nama	**Tipe Data**	**Deskripsi**
id	Bigint(20)	*primary key* sebagai identitas unik pesan
first_name	varchar(255)	Nama awal pengirim pesan
last_name	varchar(255)	Nama akhir pengirim pesan
email	varchar(255)	Email pengirim pesan
message	varchar(255)	Pesan yang ingin disampaikan
file	varchar(255)	*Path* atau *URL* dari *file*
created_at	timestamp	Tanggal dan waktu saat data pesan dibuat
updated_at	timestamp	Tanggal dan waktu saat data pesan terakhir diperbarui

Tabel 3.3 digunakan untuk menyimpan data yang dikirimkan oleh pengguna melalui formulir *Contact Us* di *website*. Setiap data dalam tabel ini merepresentasikan satu pesan yang masuk dari pengunjung kepada *admin*. Kolom *id* berfungsi sebagai identitas unik untuk setiap pesan yang dikirim. Informasi pengirim disimpan melalui kolom *first_name*, *last_name*, dan *email*, yang masing-masing menyimpan nama depan, nama belakang, serta alamat *email* dari pengirim pesan.

Kolom *message* berisi isi atau konten pesan yang disampaikan oleh pengguna, sementara kolom *file* digunakan untuk menyimpan *path* atau *URL* dari

file yang dilampirkan. Selain itu, kolom *created_at* dan *updated_at* mencatat waktu saat pesan dibuat dan terakhir diperbarui. Tabel ini memudahkan *admin* dalam memantau, mengelola, serta menindaklanjuti setiap pertanyaan, kritik, atau masukan dari pengunjung situs.

C Skema Database Comments

Tabel *comments* dirancang untuk mengelola dan menyimpan seluruh aktivitas komentar yang dilakukan oleh pengguna pada artikel-artikel yang dipublikasikan dalam *website* portal berita. Struktur tabel ini terdiri dari enam kolom utama yang mencakup identitas unik komentar melalui kolom *id* sebagai *primary key*, referensi pengguna melalui *user_id* yang terhubung dengan tabel pengguna, serta *slug* sebagai *identifier* spesifik yang menunjukkan lokasi artikel tempat komentar tersebut ditempatkan. Selain itu, tabel ini juga memuat kolom *content* untuk menyimpan isi komentar yang ditulis pengguna, dilengkapi dengan *created_at* dan *updated_at* sebagai *timestamp* yang mencatat waktu pembuatan dan pembaruan komentar secara otomatis, sehingga memungkinkan sistem untuk menampilkan interaksi pengguna secara *real-time* dan memfasilitasi moderasi komentar dengan lebih efektif.

Tabel 3.4. *Timeline* Tabel skema *database* dari komentar

Nama	**Tipe Data**	**Deskripsi**
id	bigint	*primary key*, identitas unik untuk setiap komentar
user_id	bigint	ID pengguna yang mengirim komentar
slug	varchar(255)	Identifier yang menunjukkan lokasi/artikel dimana komentar diunggah
content	text	Isi atau teks komentar yang dikirim oleh pengguna
created_at	timestamp	Tanggal dan waktu saat komentar dibuat
updated_at	timestamp	Tanggal dan waktu saat komentar diperbarui

Tabel 3.4 digunakan untuk menyimpan setiap komentar yang dikirimkan oleh pengguna pada suatu artikel atau berita di dalam *website* portal berita. Setiap baris dalam tabel ini merepresentasikan satu komentar yang ditulis dan

diunggah oleh pengguna. Kolom *id* berfungsi sebagai *primary key* yang menjadi identitas unik dari setiap komentar. Kolom *user_id* berisi identitas pengguna yang mengirimkan komentar dan terhubung dengan tabel pengguna sebagai referensi relasional.

Kolom *slug* digunakan sebagai penanda atau *identifier* artikel tempat komentar tersebut ditampilkan, sedangkan kolom *content* menyimpan isi dari komentar. Kolom *created_at* dan *updated_at* masing-masing mencatat waktu saat komentar dibuat dan saat diperbarui. Dengan adanya tabel ini, sistem dapat menampilkan serta mengelola komentar pengguna pada setiap artikel secara dinamis dan efisien.

3.4.6 Cuplikan Kode

Untuk memberikan gambaran yang lebih jelas mengenai proses pengembangan dan implementasi sistem, cuplikan kode disajikan dengan penjelasan yang mendalam. Setiap bagian menunjukkan langkah-langkah teknis yang diterapkan dalam menyelesaikan tugas selama kegiatan magang. Penjabaran ini bertujuan agar pembaca memahami alur logika, penggunaan teknologi, serta pertimbangan yang diambil dalam proses pengembangan aplikasi.

A Sign up

Fitur *Sign Up* merupakan salah satu komponen penting dalam sistem yang berfungsi untuk memungkinkan pengguna baru melakukan pendaftaran akun. Proses registrasi dilakukan dengan mengisi data pribadi seperti nama, *email*, kata sandi, nomor telepon, jenis kelamin, tanggal lahir, dan secara opsional mengunggah foto profil. Setiap data yang dimasukkan pengguna akan divalidasi terlebih dahulu sebelum disimpan ke dalam basis data. Cuplikan kode proses *Sign Up* dapat dilihat pada Kode 3.1 sampai Kode 3.6

```
public function register(Request $request)
{
    // Validasi request
    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:6',
        'phone' => 'required|string',
        'gender' => 'required|string',
        'date_of_birth' => 'required|date',
        'profile_image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
    ]);
}
```

Kode 3.1. Kode fungsi *register* pada proses *Sign Up*

Kode 3.1 menjelaskan fungsi register yang digunakan untuk memproses pendaftaran pengguna baru. Pertama-tama, sistem melakukan validasi *input* menggunakan *Validator*. Validasi ini memastikan bahwa semua data yang dikirimkan sesuai dengan kriteria, seperti nama harus berupa *string* dan tidak lebih dari 255 karakter, *email* harus unik dan *valid*, serta kata sandi harus memiliki minimal 8 karakter. Validasi juga mencakup data opsional seperti gambar profil.

Kode 3.2 menjelaskan kalau terjadinya proses validasi gagal. Jika proses validasi gagal, sistem akan mengembalikan respons error berupa *JSON* dengan kode status 422. Ini bertujuan untuk memberi tahu pengguna bahwa ada data yang tidak *valid* dan perlu diperbaiki.

```
if ($validator->fails()) {
            return response()->json(['errors' => $validator->errors()], 422);
        }
```

Kode 3.2. Kode fungsi jika terjadi *error* saat validasi

Kode 3.3 memperlihatkan proses pengelolaan gambar profil pengguna pada saat registrasi. Apabila pengguna mengunggah gambar profil, sistem akan

menyimpannya ke dalam direktori *profile-images* pada penyimpanan publik. Url atau *path file* dari gambar yang telah disimpan akan disimpan dalam variabel *profileImagePath*, yang kemudian digunakan untuk disimpan di *database* sebagai bagian dari informasi pengguna.

```
$profileImagePath = null;
    if ($request->hasFile('profile_image')) {
        $image = $request->file('profile_image');
        $profileImagePath = $image->store('profile-images',
    'public');
    }
```

Kode 3.3. Kode fungsi proses gambar profil pengguna

Kode 3.4 memperlihatkan proses akhir dari registrasi pengguna. Setelah seluruh data berhasil divalidasi, sistem akan memasukkan data baru ke dalam tabel *users* menggunakan data yang telah diperoleh dari *input* pengguna. Untuk menjaga keamanan, *password* pengguna akan di-*hash* terlebih dahulu sebelum disimpan ke dalam basis data. Jika pengguna tidak mengunggah gambar profil atau tidak mengisi *bio*, maka kedua atribut tersebut akan diubah secara otomatis menjadi *null*. Apabila proses registrasi berjalan dengan sukses, sistem akan memberikan respons berupa data *JSON* yang memuat informasi pengguna baru beserta *URL* dari foto profilnya. Kode *status* 201 pada *respons* tersebut menunjukkan bahwa data telah berhasil dibuat oleh sistem.

```
    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
        'phone' => $request->phone,
        'gender' => $request->gender,
        'date_of_birth' => $request->date_of_birth,
        'bio' => $request->bio ?? null,
        'profile_image' => $profileImagePath,
    ]);
```

Kode 3.4. Kode fungsi memasukkan data pengguna ke *database*

Potongan kode pada Kode 3.5 mendeklarasikan *state* menggunakan *hook useState*. Objek *formData* digunakan untuk menyimpan nilai dari setiap isian formulir seperti *name*, *email*, *password*, *phone*, *gender*, dan *date_of_birth*. Selain itu, terdapat variabel *message* yang berfungsi untuk menyimpan pesan kesalahan apabila terdapat *input* yang belum diisi atau terjadi kesalahan selama proses pendaftaran. Fungsi *handleChange* bertanggung jawab untuk memperbarui nilai dari *formData* setiap kali pengguna mengetik atau memilih *input* pada formulir. Fungsi ini dirancang secara umum untuk menangani semua *input* berdasarkan atribut *name*, sehingga dapat digunakan untuk berbagai jenis isian seperti *text field* dan *radio button*.

```
function SignUP() {
  const [showPassword, setShowPassword] = useState(false);
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    password: "",
    phone: "",
    gender: "",
    day : "",
    month: "",
    year: "",
    date_of_birth: "",
  });
  const [message, setMessage] = useState("");

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.
     value });
  };
```

Kode 3.5. Kode fungsi *register* pada proses *Sign Up* di *frontend*

```
const handleSubmit = async (e) => {
  e.preventDefault();
  if (!formData.name.trim() || !formData.email.trim() || !formData
    .password.trim() ||
      !formData.phone.trim() || !formData.day.trim() || !formData.
    month.trim() || !formData.year.trim()) {
    setMessage("All fields are required!");
    return;
  }

  const formattedDate = `${formData.year}-${formData.month.
    padStart(2, "0")}-${formData.day.padStart(2, "0")}`;
  const finalData = {
    name: formData.name.trim(),
    email: formData.email.trim(),
    password: formData.password,
    phone: formData.phone.trim(),
    gender: formData.gender,
    date_of_birth: formattedDate,
  };

  try {
    const response = await axios.post("http://127.0.0.1:8000/api/
    register", finalData, {
      headers: { "Content-Type": "application/json" },
    });

    Swal.fire({
      title: "Success!",
      text: "Registration successful!",
      icon: "success"
    }).then(() => {
      window.location.href = '/login';
    });

  } catch (error) {
    console.error("Validation errors:", error.response?.data);
  }
};
```

Kode 3.6. Kode fungsi *handleSubmit* pada proses Sign Up di sisi klien

Potongan kode pada Kode 3.6, fungsi *handleSubmit* menangani proses

saat formulir *sign up* dikirimkan oleh pengguna. Baris pertama dari fungsi ini memanggil *e.preventDefault()* yang bertujuan untuk mencegah perilaku bawaan dari *form*, yaitu memuat ulang halaman saat tombol *submit* dipilih. Selanjutnya dilakukan proses validasi untuk memastikan bahwa semua kolom wajib seperti *name*, *email*, *password*, *phone*, serta tanggal lahir (hari, bulan, dan tahun) telah diisi oleh pengguna. Nilai hari, bulan, dan tahun kemudian digabungkan ke dalam satu *string* berformat *YYYY-MM-DD* dan disimpan dalam variabel *formattedDate*. Data lengkap pendaftaran dikemas dalam objek *finalData*, yang berisi semua informasi pengguna dan siap dikirim ke *backend*. Pengiriman data dilakukan menggunakan *axios.post* ke */api/register*, yaitu *endpoint* pada server *Backend*. Jika proses registrasi berhasil, maka akan ditampilkan notifikasi sukses menggunakan *SweetAlert*, dan pengguna akan diarahkan ke halaman *login*. Namun, jika terjadi kesalahan, maka akan ditampilkan notifikasi kegagalan beserta pesan kesalahannya.

B Login

Fitur *login* digunakan untuk mengautentikasi pengguna yang telah terdaftar sebelumnya. Proses ini dilakukan dengan mencocokkan *email* dan kata sandi yang dimasukkan pengguna dengan data yang tersimpan di sistem. Jika berhasil, pengguna akan diberikan *token* autentikasi yang digunakan untuk mengakses fitur lain yang memerlukan izin. Cuplikan kode proses *login* dapat dilihat pada Kode 3.7 sampai Kode 3.11.

```
1     public function login(Request $request)
2     {
3         // Validate request
4         $validator = Validator::make($request->all(), [
5             'email' => 'required|string|email',
6             'password' => 'required|string',
7         ]);
8
9         if ($validator->fails()) {
10            return response()->json(['errors' => $validator->
    errors()], 422);
11    }
```

Kode 3.7. Kode fungsi validasi *input email* dan *password*

Kode 3.7 menjelaskan mengenai fungsi *login* yang digunakan untuk

memproses pengguna yang ingin melakukan *login* ke dalam sistem. Sama seperti saat registrasi, sistem memvalidasi *input* dari pengguna, dalam hal ini *email* dan *password*. Jika data yang dimasukkan tidak sesuai (misalnya *email* tidak *valid* atau *password* tidak *valid*), maka sistem akan menolak permintaan *login* dan mengembalikan pesan kesalahan.

Kode 3.8 menggambarkan sistem yang akan mencari pengguna berdasarkan *email*. Jika tidak ditemukan, atau jika *password* yang dimasukkan tidak cocok dengan *password* yang tersimpan di database (yang sudah di-*hash*), maka akan dikembalikan pesan kesalahan bahwa kredensial tidak *valid*.

```
1  $user = User::where('email', $request->email)->first();
2
3          // Check if user exists and password matches
4          if (!$user || !Hash::check($request->password, $user->
       password)) {
5              return response()->json([
6                  'message' => 'Invalid credentials'
7              ], 401);
8      }
```

Kode 3.8. Kode fungsi autentikasi *email* dan *password*

Kode 3.9 menjelaskan proses autentikasi yang terjadi setelah pengguna berhasil melakukan *login*. Jika data yang dimasukkan pengguna *valid* dan sesuai dengan yang terdapat pada *database*, maka sistem akan membuat token autentikasi sebagai tanda bahwa pengguna telah terverifikasi. *Token* ini akan digunakan untuk mengakses berbagai *endpoint* yang memerlukan otorisasi (*authenticated routes*). Selanjutnya, sistem akan mengembalikan respons berupa data *JSON* yang berisi token autentikasi, informasi pengguna yang berhasil *login*, serta *URL* gambar profil apabila tersedia. Kode *status* 200 yang ditampilkan menunjukkan bahwa permintaan telah berhasil diproses oleh sistem.

```
// Generate token for the user
        Auth::login($user);
        $token = $user->createToken('auth_token')->plainTextToken;

        return response()->json([
            'message' => 'Login successful',
            'token' => $token,
            'user' => $user,
            'profile_image_url' => $profileImageUrl
        ], 200);
```

Kode 3.9. Kode fungsi berhasil *Log In*

Bagian awal kode 3.10 memuat sejumlah *import* dari *library* dan aset yang digunakan dalam proses *login*. *React, useState, dan useEffect* merupakan komponen inti dari *ReactJS* yang digunakan untuk mengelola *state* dan efek samping. Selain itu, *axios* digunakan sebagai alat untuk melakukan permintaan *HTTP* ke *server backend*, sedangkan *Swal* dari *library SweetAlert2* digunakan untuk menampilkan notifikasi *pop-up* secara interaktif. Dua gambar yang dimuat, yaitu *googlepng* dan *facebookpng*, digunakan sebagai ikon *visual* untuk opsi *login* dengan akun *Google* dan *Facebook*. Inisialisasi variabel *state* seperti *showPassword, formData, loading, dan googleLoginUrl* dilakukan dengan *useState*, yang masing-masing bertugas mengelola tampilan sandi, data yang dimasukkan oleh pengguna dan status pemrosesan *login*.

```
import React, { useState, useEffect } from "react";
import axios from "axios";
import googlepng from "../../assets/img/google-logo.png";
import facebookpng from "../../assets/img/fb-logo.png";
import Swal from "sweetalert2";

function Login() {
    const [showPassword, setShowPassword] = useState(false);
    const [formData, setFormData] = useState({ email: "", password
    : "" });
    const [loading, setLoading] = useState(false);
    const [googleLoginUrl, setGoogleLoginUrl] = useState("");
```

Kode 3.10. Kode *import library Log In* di sisi klien

```
const handleSubmit = async (e) => {
        e.preventDefault();

        if (!formData.email.trim() || !formData.password.trim()) {
            Swal.fire({ title: "Error", text: "Email and password are required!", icon: "error" });
            return;
        }

        setLoading(true);
        try {
            const response = await axios.post("http://localhost:8000/api/login", {
                email: formData.email.trim(),
                password: formData.password
            },{
                headers: {
                    "Content-Type": "application/json",
                    "Accept": "application/json",
                    "X-Requested-With": "XMLHttpRequest",
                }
            });

            localStorage.setItem("user", JSON.stringify(response.data.user));
            localStorage.setItem("token", response.data.token);

            window.location.href = response.data.user.role === "admin" ? "/admin" : "/";
        } catch (error) {
            Swal.fire({
                title: "Login Failed",
                text: error.response?.data.message || "Invalid credentials",
                icon: "error"
            });
        } finally {
            setLoading(false);
        }
    };
```

Kode 3.11. Kode fungsi *handleSubmit* pada proses *Log In* di sisi klien

Fungsi *handleSubmit* seperti ditampilkan pada Kode 3.11 bertanggung jawab untuk menangani proses pengiriman *formulir login*. Fungsi ini pertama-tama mencegah perilaku bawaan dari *form* agar tidak memuat ulang halaman. Validasi sederhana dilakukan untuk memastikan bahwa *email* dan kata sandi telah diisi. Jika *valid*, aplikasi akan mengirim permintaan *POST* ke *endpoint /api/login* menggunakan *axios*, beserta email dan *password* sebagai data. Bila *login* berhasil, data pengguna dan *token autentikasi* disimpan ke dalam *localStorage* yang kemudian dikirimkan dan disimpan ke database oleh *server*, dan pengguna akan diarahkan ke halaman yang sesuai dengan perannya (*admin* atau pengguna biasa). Jika terjadi kesalahan, notifikasi kesalahan akan ditampilkan melalui *SweetAlert*, dan status *loading* akan dihentikan setelah proses selesai.

C Halaman Detail

Halaman detail dirancang untuk menampilkan isi detail data berita yang meliputi gambar pendukung berita, judul berita, penulis, tanggal publikasi, isi artikel berita, dan fitur komentar. Rincian potongan kode untuk halaman detail dapat dilihat pada Gambar 3.12 dan 3.13

Kode 3.12 menggambarkan proses pengambilan data detail suatu artikel berdasarkan parameter *slug* yang diberikan. Fungsi *fetchArticleDetails* akan dipicu ketika nilai *slug* tersedia, kemudian mengirim permintaan ke *API* eksternal dengan metode *GET* dan autentikasi *token* yang sesuai. Setelah data berhasil diambil dan diubah menjadi format *JSON*, data tersebut disimpan ke dalam *state* yang digunakan untuk menampilkan detail berita kepada pengguna. Setelah data berita utama berhasil didapatkan, sistem melanjutkan proses dengan memanggil fungsi *fetchRelatedArticles* menggunakan kategori dari berita tersebut. Fungsi ini mengambil seluruh data berita dari *backend lokal*, kemudian menyaring berita yang masih memiliki kategori sama namun berbeda *slug*. Tiga berita pertama dari hasil penyaringan tersebut ditampilkan sebagai artikel terkait. Seluruh proses ini dilakukan secara *asynchronous*, dengan penanganan kesalahan dan pengaturan status *loading* untuk menjaga pengalaman pengguna tetap nyaman dan informatif.

```
const fetchArticleDetails = async () => {
  setLoading(true);
  try {
    const res = await fetch(`https://winnicode.com/api/publikasi-berita/${slug}`, {
      headers: {
        "Authorization": `Bearer ${apiKey}`,
        "Accept": "application/json",
        "Content-Type": "application/json"
      }
    });
    if (!res.ok) throw new Error(`HTTP error! ${res.status}`);
    const data = await res.json();
    setArticle(data);
    fetchRelatedArticles(data.kategori);
  } catch (err) {
    console.error("Error:", err);
    setError("Failed to load article.");
  } finally {
    setLoading(false);
  }
};

const fetchRelatedArticles = async (category) => {
  try {
    setLoading(true);
    const res = await fetch("http://localhost:8000/api/news");
    if (!res.ok) throw new Error(`HTTP error! ${res.status}`);
    const data = await res.json();
    const articles = Array.isArray(data) ? data :
      Array.isArray(data?.data) ? data.data :
      typeof data === 'object' ? [data] : [];

    const filtered = articles.filter(a => a.slug !== slug).slice(0, 3);
    setRelatedArticles(filtered);
  } catch (err) {
    console.error("Related article error:", err);
  }
};

if (slug) fetchArticleDetails();
```

Kode 3.12. Kode menyaring berita berdasarkan kategori

Kode 3.13 menggambarkan fungsi *getImageUrl* digunakan untuk menghasilkan URL gambar yang akan ditampilkan pada antarmuka. Jika parameter *gambarId* tersedia, maka fungsi ini akan membentuk URL dengan format khusus dari *Google Drive thumbnail*, yakni "*https://drive.google.com/thumbnail?id=...*", yang memungkinkan gambar ditampilkan secara langsung tanpa terkena kendala *CORS*. CORS (*Cross-Origin Resource Sharing*) adalah mekanisme yang mengatur akses sumber daya sebuah *website* dari *domain* lain. Jika *gambarId* tidak tersedia, maka fungsi akan mengembalikan *URL* gambar *placeholder* sebagai alternatif tampilan *default* agar antarmuka tetap konsisten dan tidak menampilkan gambar rusak.

```
1 const getImageUrl = (gambarId) => {
2         return gambarId
3         ? `https://drive.google.com/thumbnail?id=${gambarId}&sz=
    w1000`
4         : "https://via.placeholder.com/150";
5     };
```

Kode 3.13. Kode Fungsi Pengambilan Gambar

D Contact Us

Halaman *Contact Us* merupakan halaman yang dibuat agar pengelola dapat menerima pesan atau saran dan kritik dari pengguna. Dalam *form* di halaman *Contact Us* terdapat *form* yang berisi *First Name, Last Name, email, messages*, dan *File*. Untuk rincian kode fitur dan halaman *Contact Us* dapat dilihat pada Kode 3.14 sampai Kode 3.16

Kode 3.14 menunjukkan implementasi fungsi *store* pada halaman *Contact Us* yang bertanggung jawab untuk memproses data formulir yang dikirimkan oleh pengguna. Fungsi ini melakukan validasi terhadap *input* yang diterima, termasuk nama depan, nama belakang, *email*, pesan, dan lampiran (jika ada). Jika pengguna mengunggah berkas, maka berkas tersebut disimpan dalam direktori *contact_files* menggunakan sistem penyimpanan *public*. Data yang telah lolos validasi kemudian disimpan ke dalam *database* melalui model *Messages*. Sebagai umpan balik, pengguna akan menerima respons dalam bentuk *JSON* yang menyatakan bahwa pesan telah berhasil dikirim.

```
const getImageUrl = (gambarId) => {
        return gambarId
        ? `https://drive.google.com/thumbnail?id=${gambarId}&sz=w1000`
        : "https://via.placeholder.com/150";
    };
```

Kode 3.14. Kode Fungsi Menyimpan data pesan ke *database*

```
function ContactUs() {
  const [formData, setFormData] = useState({
    first_name: "",
    last_name: "",
    email: "",
    message: "",
    file: null,
  });

  const [status, setStatus] = useState("");

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prev) => ({
      ...prev,
      [name]: value,
    }));
  };

  const handleFileChange = (e) => {
    setFormData((prev) => ({
      ...prev,
      file: e.target.files[0],
    }));
  };
```

Kode 3.15. Kode Fungsi *useState*, *handleChange*, *handleFileChange*

Potongan kode 3.15 merupakan bagian dari komponen *Contact Us* di sisi *frontend* yang menggunakan *React*. Dalam bagian ini, state *formData* didefinisikan menggunakan *useState* untuk menyimpan data input pengguna, seperti nama depan, nama belakang, *email*, pesan, dan berkas yang diunggah. Selain itu, terdapat state

tambahan bernama *status* untuk menyimpan informasi status pengiriman formulir. Fungsi *handleChange* digunakan untuk memperbarui data masukan teks secara dinamis setiap kali pengguna mengetik pada kolom formulir, sementara fungsi *handleFileChange* secara khusus menangani perubahan pada *input* berkas dan menyimpan berkas yang dipilih ke dalam *state*. Pendekatan ini memungkinkan pengelolaan *form* yang efisien dan responsif dalam antarmuka pengguna.

```
1  const handleSubmit = async (e) => {
2      e.preventDefault();
3      const payload = new FormData();
4      payload.append("first_name", formData.first_name);
5      payload.append("last_name", formData.last_name);
6      payload.append("email", formData.email);
7      payload.append("message", formData.message);
8      if (formData.file) {
9        payload.append("file", formData.file);
10     }
11     try {
12       await axios.post("http://localhost:8000/api/messages",
       payload, {
13         headers: {
14           "Content-Type": "multipart/form-data",
15         },
16       });
17       setFormData({
18         first_name: "",
19         last_name: "",
20         email: "",
21         message: "",
22         file: null,
23       });
24     } catch (err) {
25       console.error(err);
26     }
27   };
```

Kode 3.16. Kode fungsi *handleSubmit Contact Us*

Fungsi *handleSubmit* yang ditunjukkan pada Kode 3.16 berfungsi untuk menangani proses pengiriman formulir *Contact Us*. Fungsi ini pertama-tama mencegah perilaku bawaan *form* dengan *e.preventDefault()*, kemudian membuat

objek *FormData* yang berisi data pengguna seperti nama depan, nama belakang, *email*, pesan, dan berkas (jika ada). Selanjutnya, data dikirimkan ke *endpoint API Laravel* menggunakan *axios* melalui metode *POST* dengan tipe konten *multipart/form-data*, agar berkas dapat dikirim dengan benar. Jika permintaan berhasil, maka pengguna akan menerima umpan balik berupa pesan keberhasilan menggunakan *SweetAlert*, dan *state* formulir akan di-reset. Namun, jika terjadi kesalahan saat proses pengiriman, maka akan ditampilkan peringatan kesalahan agar pengguna mengetahui bahwa pesan tidak berhasil dikirim. Pendekatan ini memberikan pengalaman pengguna yang informatif dan ramah.

E Fitur Komentar

Fitur komentar merupakan bagian penting dalam sebuah aplikasi berita atau blog karena memungkinkan interaksi langsung antara pembaca dan konten. Di dalam aplikasi ini, sistem komentar dibuat agar pengguna yang sudah *login* dapat memberikan komentar pada setiap artikel atau konten yang diidentifikasi berdasarkan *slug* (penanda unik setiap artikel). Rincian potongan kode untuk fitur komentar dapat dilihat pada Kode 3.17 sampai Kode 3.20.

Kode 3.17 menunjukkan proses pengiriman komentar baru oleh pengguna yang telah melakukan *login*. Fungsi *store()* pada potongan kode berikut bertugas untuk menerima dan menyimpan komentar dari pengguna. Data yang divalidasi terdiri dari *slug* sebagai penanda halaman dan *content* sebagai isi komentar. Sementara itu, *user_id* diambil langsung dari pengguna yang sedang *login* melalui *Auth::user()*. Setelah data divalidasi, komentar disimpan ke dalam basis data menggunakan metode *create()*, lalu relasi dengan pengguna dimuat menggunakan *load('user')* agar respons *JSON* mencakup data pengguna yang mengirimkan komentar. Respons yang dikembalikan memiliki kode *status* 201, yang menandakan bahwa data berhasil dibuat.

```
public function store(Request $request)
    {
        // Validate only slug and content; user_id from auth
        $request->validate([
            'slug'    => 'required|string|max:255',
            'content' => 'required|string|max:1000',
        ]);

        $user = Auth::user();


        $comment = Comment::create([
            'user_id' => $user->id,
            'slug'    => $request->slug,
            'content' => $request->content,
        ]);

       $comment->load('user');
       return response()->json($comment, 201);
    }
  };
```

Kode 3.17. Kode menyimpan komentar ke *Database*

```
public function getComments($slug)
    {
        $comments = Comment::with('user')
            ->where('slug', $slug)
            ->latest()
            ->get();

        return response()->json($comments);
    }
```

Kode 3.18. Kode menampilkan data komentar berdasarkan *slug*

Kode 3.18 juga memperlihatkan bagaimana komentar ditampilkan berdasarkan halaman yang sesuai. Fungsi *getComments()* digunakan untuk mengambil semua komentar yang terkait dengan suatu halaman berdasarkan *slug*-nya. Pada fungsi ini, sistem akan melakukan pencarian ke basis data menggunakan metode *where()* untuk memfilter komentar yang memiliki *slug* tertentu. Relasi

antara komentar dan pengguna juga dimuat menggunakan *with('user')* sehingga data pengguna yang mengirimkan komentar dapat langsung ditampilkan. Komentar kemudian diurutkan dari yang terbaru menggunakan *latest()* dan dikembalikan dalam bentuk respons *JSON*.

```
1     useEffect(() => {
2         const fetchComments = async () => {
3             try {
4                 const res = await fetch(`http://localhost:8000/api
    /comments/${slug}`);
5                 if (res.ok) {
6                     const data = await res.json();
7                     setComments(data);
8                 }
9             } catch (error) {
10                console.error("Error fetching comments:", error);
11            }
12        };
13
14        if (slug) {
15            fetchComments();
16        }
17    }, [slug]);
```

Kode 3.19. Kode fungsi *useEffect* untuk mengambil dan menampilkan *komentar* berdasarkan *slug*

Kode 3.19 memperlihatkan bagaimana komentar ditampilkan secara dinamis pada antarmuka pengguna. Potongan kode berikut merupakan bagian dari *frontend* yang bertugas mengambil data komentar dari server. Dengan memanfaatkan *useEffect*, fungsi ini akan dijalankan setiap kali nilai *slug* berubah. Di dalamnya, terdapat fungsi *asinkron fetchComments* yang menggunakan *fetch API* untuk melakukan permintaan ke *endpoint http://localhost:8000/api/comments/slug*. Jika permintaan berhasil, data hasil respon diubah menjadi format *JSON* dan disimpan dalam *state* lokal melalui *setComments*. Dengan demikian, setiap kali pengguna membuka halaman dengan *slug* tertentu, sistem secara otomatis menampilkan komentar-komentar yang sesuai tanpa perlu memuat ulang seluruh halaman.

```
const submitComment = async (e) => {
  e.preventDefault();
  const token = localStorage.getItem("token");
  if (!token) return setCommentError("Please login to comment.");
  if (!commentText.trim()) return setCommentError("Comment cannot be empty.");

  setCommentLoading(true);
  setCommentError("");

  try {
    const res = await fetch("http://localhost:8000/api/comments", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${token}`,
        "Accept": "application/json"
      },
      body: JSON.stringify({ slug, content: commentText.trim() })
    });

    if (res.ok) {
      const newComment = await res.json();
      setComments([newComment, ...comments]);
      setCommentText("");
      if (!isLoggedIn) {
        setIsLoggedIn(true);
        if (newComment.user) setUser(newComment.user);
      }
    } else {
      const err = await res.json();
      if (res.status === 401) {
        setCommentError("Session expired. Please login again.");
        localStorage.removeItem("token");
        setIsLoggedIn(false);
        setUser(null);
      } else {
        setCommentError(err.message || "Failed to submit comment.");
      }
    }
  } catch (err) {
    setCommentError("Network error. Please try again.");
    console.error("Comment error:", err);
  } finally {
    setCommentLoading(false);
  }
};
```

Kode 3.20. Kode fungsi *submitComment*

Kode 3.20 menampilkan antarmuka pengguna tempat pengguna dapat memberikan komentar pada suatu konten. Untuk mendukung fitur tersebut, fungsi *submitComment* pada sisi *frontend* digunakan untuk mengirim data komentar ke

backend. Ketika pengguna menekan tombol kirim, fungsi ini akan menghentikan aksi bawaan *form*, kemudian memeriksa apakah pengguna telah masuk dan apakah komentar tidak kosong. Jika *valid*, maka permintaan *POST* dikirim ke *endpoint /api/comments* menggunakan *fetch API*, dilengkapi dengan *token* autentikasi pada bagian *header*. Jika respon berhasil, komentar yang baru ditambahkan akan dimunculkan paling atas di daftar komentar tanpa perlu memuat ulang halaman. Selain itu, fungsi ini juga menangani kondisi ketika *token* sudah tidak *valid* atau koneksi internet bermasalah, dengan memberikan *feedback* kesalahan kepada pengguna secara jelas dan mudah dipahami.

F Kategori

Untuk memberikan pengalaman pengguna yang lebih interaktif dan informatif, website portal berita menyediakan fitur penyaringan berita berdasarkan kategori. Setiap kategori juga dilengkapi dengan informasi jumlah artikel yang tersedia agar pengguna dapat mengetahui cakupan konten yang ada sebelum memilihnya. Rincian cuplikan kode untuk fitur dan halaman kategori dapat dilihat pada Kode 3.21 dan Kode 3.22

Kode 3.21 menunjukkan proses yang terjadi saat pengguna memilih kategori tertentu pada halaman berita. Fungsi *handleCategoryChange* berperan penting dalam menangani interaksi ini. Ketika pengguna memilih suatu kategori, fungsi ini akan mengatur kategori aktif melalui *setActiveCategory*, mengatur ulang halaman ke halaman pertama menggunakan *setPage*, lalu menyaring daftar berita berdasarkan kategori tersebut. Jika pengguna memilih kategori ”*all*”, maka seluruh berita akan ditampilkan, sedangkan untuk kategori tertentu, hanya berita yang berasal dari sumber dengan nama yang sesuai yang akan ditampilkan. Hasil penyaringan kemudian dibatasi jumlahnya sesuai jumlah berita per halaman menggunakan *slice*, dan dilakukan pengecekan apakah masih ada berita lainnya melalui *setHasMore*. Terakhir, dilakukan navigasi ke *URL* yang sesuai dengan kategori yang dipilih agar status tampilan tetap konsisten meskipun halaman dimuat ulang.

```
    const handleCategoryChange = (category) => {
        setActiveCategory(category);
        setPage(1);

        const filteredNews = category === "all"
            ? allNews
            : allNews.filter(news => news.source.name === category);

        setDisplayedNews(filteredNews.slice(0, newsPerPage));
        setHasMore(filteredNews.length > newsPerPage);

        navigate(`/news?category=${category === "all" ? "" : encodeURIComponent(
    category)}`);
    };
```

Kode 3.21. Kode alur perubahan kategori berita

Gambar 3.22 menggambarkan cara sistem menghitung jumlah artikel yang termasuk dalam suatu kategori tertentu. Fungsi *getCategoryCount* digunakan untuk memperoleh jumlah artikel yang bersumber dari kategori tertentu dengan mencocokkan nama sumber berita. Fungsi ini bekerja dengan menyaring seluruh data berita dalam variabel *allNews* menggunakan metode *filter*, kemudian menghitung jumlah elemen yang cocok dengan kategori yang dimaksud menggunakan *length*. Informasi ini dapat digunakan untuk menampilkan jumlah artikel di samping nama kategori, sehingga pengguna dapat mengetahui seberapa banyak konten yang tersedia dalam masing-masing kategori.

```
const getCategoryCount = (categoryName) => {
        return allNews.filter(news => news.source.name === categoryName).length;
    };
```

Kode 3.22. Kode fungsi menghitung jumlah artikel

G Admin

Halaman Admin dibuat untuk melihat pesan pesan yang dikirim oleh pengguna melalui *form* di halaman *Contact Us*. Hal ini bertujuan agar pengelola dapat menerima masukan masukan yang masuk dari pengguna. Cuplikan kode untuk fungsi dan halaman admin dapat dilihat pada Kode 3.23 sampai Kode 3.24.

```
    public function index()
    {
        // Mengambil semua pesan dan mengembalikannya dalam format JSON
        $messages = Messages::all();
        return response()->json($messages);
    }
```

Kode 3.23. Kode fungsi *index* di backend untuk menampilkan pesan

Kode 3.23 menunjukkan fungsi *index()* yang dibuat untuk memungkinkan *admin* melihat pesan-pesan yang dikirim oleh *user* melalui halaman *Contact Us*. Fungsi ini berperan sebagai *endpoint* dalam API yang mengambil seluruh data dari model *Messages* menggunakan metode bawaan *Eloquent* yaitu *all()*. Setelah data diambil, informasi tersebut kemudian dikembalikan dalam format *JSON* agar dapat dengan mudah digunakan oleh bagian *frontend*. Pendekatan ini mempermudah proses integrasi antara antarmuka pengguna dan basis data secara efisien dan terstruktur.

Kode 3.24 menunjukkan proses pengambilan data pesan dari *API* menggunakan *hook* bawaan React yaitu *useEffect()*. Pada potongan kode ini, fungsi *fetchMessages* dijalankan sekali saat komponen dimuat untuk melakukan permintaan *HTTP GET* ke endpoint */api/messages*. Data yang berhasil diambil kemudian disimpan dalam *state* menggunakan fungsi *setMessages*, dan status *loading* diatur menjadi *false* setelah proses pengambilan data selesai, baik berhasil maupun gagal. Dengan pendekatan ini, data pesan dari server dapat ditampilkan secara dinamis pada tampilan admin yang relevan.

```
useEffect(() => {
    const fetchMessages = async () => {
      try {
        const response = await axios.get("http://localhost:8000/api/messages");
        setMessages(response.data);
        setLoading(false);
      } catch (error) {
        console.error("Error fetching messages:", error);
        setLoading(false);
      }
    };

    fetchMessages();
  }, []);
```

Kode 3.24. Kode menarik data pesan dari *backend*

H Edit Profile

Fitur *edit profile* dibuat agar pengguna dapat melakukan kustomisasi nama, email, gender, tanggal lahir, dan *bio*. Rincian potongan kode *edit profile* dapat dilihat pada Kode 3.25 sampai Kode 3.29.

```
public function updateUser(Request $request, $id)
{
    if (!Auth::check()) return response()->json(['message' => 'Unauthenticated'], 401);

    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users,email,' . $id,
        'phone' => 'nullable|string',
        'gender' => 'nullable|string',
        'date_of_birth' => 'nullable|date',
        'bio' => 'nullable|string',
        'profile_image' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
    ]);

    if ($validator->fails())
        return response()->json(['errors' => $validator->errors()], 422);

    $user = User::find($id);
    if (!$user)
        return response()->json(['message' => 'User not found'], 404);

    if ($request->hasFile('profile_image')) {
        if ($user->profile_image)
            Storage::disk('public')->delete($user->profile_image);
        $user->profile_image = $request->file('profile_image')->store('profile-images', 'public');
    }

    $user->fill($request->only(['name', 'email', 'phone', 'gender', 'date_of_birth', 'bio']))->save();

    return response()->json([
        'message' => 'User updated successfully',
        'user' => $user,
        'profile_image_url' => $user->profile_image ? url('storage/' . $user->profile_image) : null
    ]);
}
```


Kode 3.25. Kode *backend* untuk edit profil

Kode 3.25 menunjukkan salah satu potongan kode penting yang penulis implementasikan pada bagian pengelolaan data profil pengguna. Fungsi

updateUser di atas berfungsi untuk memperbarui informasi akun pengguna. Proses ini mencakup validasi data seperti nama, *email*, nomor telefon, jenis kelamin, tanggal lahir, *bio*, dan gambar profil, serta memastikan bahwa alamat *email* yang diperbarui tetap unik di basis data. Selain itu, jika terdapat gambar profil baru, sistem akan menghapus gambar lama dari penyimpanan dan menggantinya dengan gambar yang baru diunggah. Setelah seluruh data berhasil diperbarui, sistem akan menyimpan perubahan dan mengembalikan respons dalam format *JSON*, termasuk *URL* gambar profil yang terbaru. Fitur ini bertujuan untuk memberikan fleksibilitas bagi pengguna dalam mengelola informasi pribadinya secara langsung melalui antarmuka aplikasi.

Kode 3.26 menunjukkan potongan kode fungsi *uploadProfileImage* yang digunakan untuk menangani proses unggah gambar profil pengguna secara terpisah. Fungsi ini diawali dengan melakukan validasi terhadap *file* yang dikirim, memastikan bahwa *file* tersebut merupakan gambar dengan format yang diizinkan dan ukuran maksimal 2MB. Selanjutnya, sistem akan mencari data pengguna berdasarkan *ID* yang diberikan. Jika pengguna ditemukan, dan sebelumnya telah memiliki gambar profil, maka gambar lama akan dihapus dari penyimpanan untuk menghindari *file* yang menumpuk. Kemudian, gambar baru akan disimpan ke dalam direktori *public*, dan jalur gambar tersebut diperbarui ke dalam data pengguna. Setelah seluruh proses selesai, sistem akan menyimpan perubahan dan mengembalikan respons dalam format *JSON* yang berisi informasi pengguna serta *URL* gambar profil yang baru diunggah. Fitur ini memberikan kemudahan bagi pengguna dalam memperbarui foto profilnya secara terpisah dari data lainnya.

```php
public function uploadProfileImage(Request $request, $id)
    {
        // Validate request
        $validator = Validator::make($request->all(), [
            'profile_image' => 'required|image|mimes:jpeg,png,jpg,gif|max:2048',
        ]);

        if ($validator->fails()) {
            return response()->json(['errors' => $validator->errors()], 422);
        }

        // Find user
        $user = User::find($id);

        if (!$user) {
            return response()->json(['message' => 'User not found'], 404);
        }

        // Delete old image if exists
        if ($user->profile_image) {
            Storage::disk('public')->delete($user->profile_image);
        }

        // Store new image
        $image = $request->file('profile_image');
        $profileImagePath = $image->store('profile-images', 'public');
        $user->profile_image = $profileImagePath;

        // Save changes
        $user->save();

        return response()->json([
            'message' => 'Profile image uploaded successfully',
            'user' => $user,
            'profile_image_url' => url('storage/' . $profileImagePath)
        ], 200);
    }
```

Kode 3.26. Kode *backend* untuk mengunggah foto profil

```
const storedUser = localStorage.getItem("user");
if (!storedUser) return navigate("/login");

try {
  const user = JSON.parse(storedUser);
  const formattedDate = user.date_of_birth
    ? new Date(user.date_of_birth).toISOString().split("T")[0]
    : "";

  setFormData({
    name: user.name || "",
    email: user.email || "",
    phone: user.phone || "",
    gender: user.gender || "",
    date_of_birth: formattedDate,
    bio: user.bio || ""
  });

  const imgUrl = localStorage.getItem("profile_image_url") ||
    (user.profile_image && `${process.env.REACT_APP_API_URL || "http
    ://127.0.0.1:8000"}/storage/${user.profile_image}`);
  if (imgUrl) setPreviewUrl(imgUrl);

  setLoading(false);
} catch (err) {
  console.error("Error loading user data:", err);
  setLoading(false);
}
```

Kode 3.27. Kode *frontend* untuk menampilkan data pengguna

Kode 3.27 menunjukkan cuplikan kode *useEffect* yang bertugas untuk memuat data pengguna saat halaman *edit profil* dibuka. Fungsi ini akan mengambil data pengguna yang telah disimpan sebelumnya melalui *localStorage*. Jika tidak ditemukan, pengguna akan langsung diarahkan ke halaman *login*. Apabila data tersedia, maka informasi seperti nama, *email*, nomor telepon, jenis kelamin, tanggal lahir, dan biodata akan dimasukkan ke dalam formulir secara otomatis. Tanggal lahir diformat ulang agar sesuai dengan format masukan bertipe *date*. Selain itu, gambar profil juga diatur agar dapat ditampilkan sebagai pratinjau jika tersedia. Pendekatan ini sangat membantu dalam meningkatkan pengalaman pengguna karena memuat data secara otomatis dan efisien saat proses pengeditan profil.

```
const handleImageChange = (e) => {
  const file = e.target.files[0];
  if (!file) return;

  const validTypes = ['image/jpeg', 'image/png', 'image/gif', 'image/jpg'];
  if (!validTypes.includes(file.type) || file.size > 2 * 1024 * 1024) {
    Swal.fire({
      title: "Upload Error",
      text: !validTypes.includes(file.type)
        ? "Please upload an image file (JPEG, PNG, GIF)"
        : "Image must be smaller than 2MB",
      icon: "error"
    });
    return;
  }

  setProfileImage(file);
  const reader = new FileReader();
  reader.onload = () => setPreviewUrl(reader.result);
  reader.readAsDataURL(file);
};

const triggerFileInput = () => fileInputRef.current.click();
```

Kode 3.28. Kode *frontend* untuk menampilkan foto profil pengguna

Kode 3.28 menampilkan potongan kode yang digunakan untuk menangani perubahan gambar profil oleh pengguna. Fungsi *handleImageChange* akan dipicu ketika pengguna memilih gambar dari perangkatnya. Pertama-tama, sistem akan memeriksa apakah *file* yang diunggah memiliki tipe yang sesuai seperti *JPEG, PNG*, atau *GIF*, dan memastikan ukuran *file* tidak melebihi *2MB*. Jika validasi berhasil, gambar tersebut akan disimpan dalam *state profileImage*. Untuk mempermudah interaksi pengguna, fungsi *triggerFileInput* dibuat agar *file* yang dimasukkan oleh user dapat memasukkan foto profilnya ke dalam *database*. Fitur ini memberikan kenyamanan dan kontrol bagi pengguna dalam memperbarui gambar profil yang digunakan.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  if (!formData.name.trim() || !formData.email.trim()) {
    return Swal.fire({ title: "Error", text: "Name and email are required!", icon: "error" });
  }

  setSubmitting(true);
  try {
    const userData = JSON.parse(localStorage.getItem("user"));
    const token = localStorage.getItem("token");

    const formDataToSend = new FormData();
    ["name", "email", "phone", "gender", "date_of_birth", "bio"].forEach(key =>
      formDataToSend.append(key, formData[key] || "")
    );
    if (profileImage) formDataToSend.append("profile_image", profileImage);

    const res = await axios.post(
      `http://127.0.0.1:8000/api/users/${userData.id}?_method=PUT`,
      formDataToSend,
      { headers: { "Content-Type": "multipart/form-data", "Authorization": `Bearer ${token}` } }
    );

    const updatedUser = { ...userData, ...formData, profile_image: res.data.user.profile_image };
    localStorage.setItem("user", JSON.stringify(updatedUser));
    if (res.data.profile_image_url)
      localStorage.setItem("profile_image_url", res.data.profile_image_url);

    Swal.fire({ title: "Success!", text: "Profile updated successfully!", icon: "success" })
      .then(() => navigate("/profile"));

  } catch (err) {
    console.error("Profile update error:", err.response?.data);
    Swal.fire({ title: "Update Failed", text: err.response?.data?.message || "Something went wrong!", icon: "error" });
  } finally {
    setSubmitting(false);
  }
};
```

Kode 3.29. Kode *frontend* untuk mengganti data pengguna

Kode 3.29 menunjukkan cuplikan kode yang menangani proses pengiriman data pembaruan profil pengguna. Fungsi *triggerFileInput* digunakan untuk secara otomatis membuka jendela pemilihan *file* saat pengguna mengklik elemen "*edit profile*". Selanjutnya, fungsi *handleSubmit* bertugas mengelola proses pembaruan data saat formulir dikirimkan. Fungsi ini memeriksa kelengkapan data wajib seperti

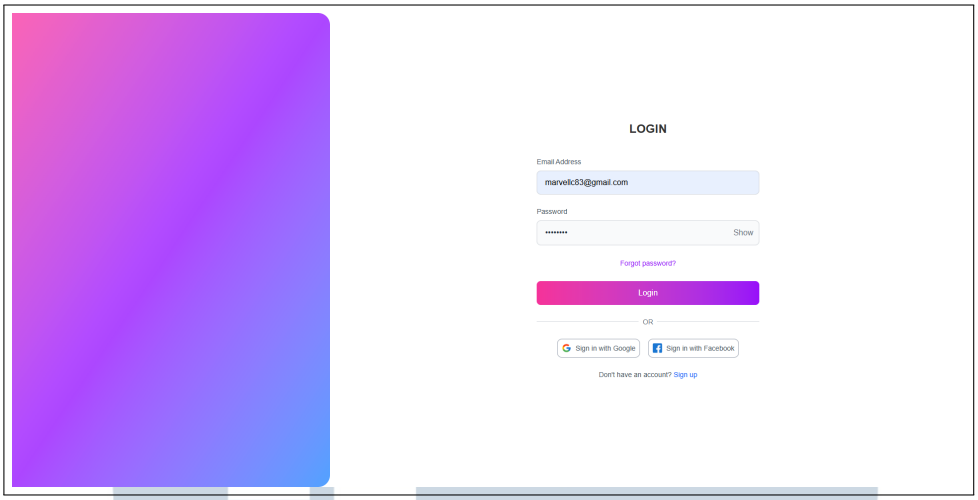
nama dan *email*, lalu mengirimkan data menggunakan format *multipart/form-data* agar gambar profil dapat disertakan. Apabila pembaruan berhasil, data terbaru akan disimpan kembali ke dalam *localStorage*, termasuk *URL* gambar terbaru jika ada dan dikirimkan ke *backend* agar data dapat tersimpan di dalam *database*. Umpan balik interaktif diberikan kepada pengguna melalui komponen *SweetAlert* untuk menunjukkan keberhasilan atau kegagalan proses.

3.4.7 Implementasi

Bagian ini menjelaskan hasil implementasi dari sistem *website* portal berita yang telah dirancang dan dibangun. Proses implementasi ini dilakukan berdasarkan rancangan desain aplikasi *flowchart*, dan cuplikan kode yang telah disusun sebelumnya, sehingga menghasilkan gambaran nyata dari fungsionalitas sistem yang telah dikembangkan. Adapun halaman-halaman yang diimplementasikan meliputi halaman *Login* untuk autentikasi pengguna, halaman *Signup* untuk proses pendaftaran akun, halaman Utama sebagai beranda portal berita, halaman Kategori untuk menampilkan berita berdasarkan klasifikasi tertentu, halaman Detail Berita untuk melihat isi berita secara lengkap, halaman *Contact Us* sebagai sarana komunikasi antara pengguna dan pengelola, halaman *About Us* yang memuat informasi mengenai perusahaan, halaman *Profile* untuk menampilkan data pengguna, serta halaman *Edit Profile* yang memungkinkan pengguna memperbarui informasi pribadinya.

A Halaman Login

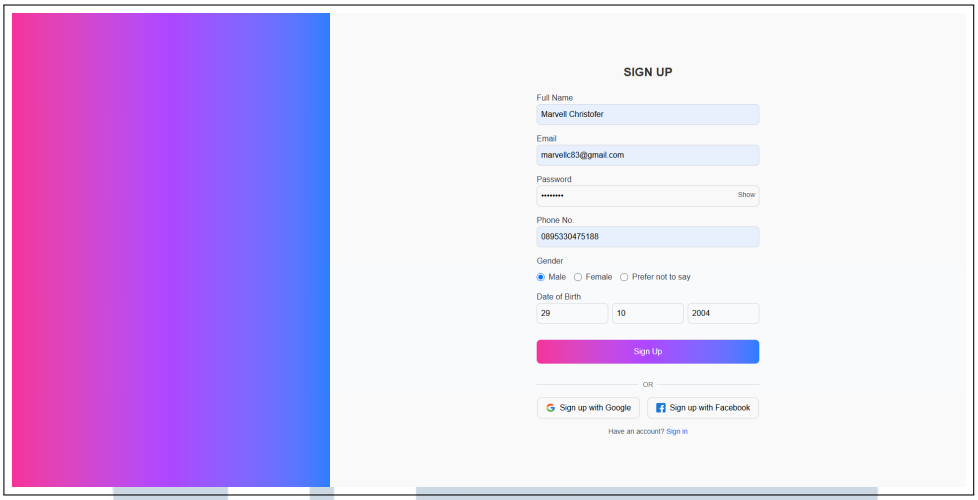
Halaman *login* merupakan gerbang awal yang digunakan oleh pengguna untuk mengakses sistem portal berita. Pada tampilan ini, pengguna diminta untuk memasukkan alamat email dan kata sandi yang telah terdaftar sebelumnya. Tersedia juga fitur "*Show*" untuk menampilkan atau menyembunyikan kata sandi guna memastikan keakuratan input pengguna. Di bawah kolom input, terdapat tautan "*Forgot password?*" yang dapat digunakan untuk memulihkan akses apabila pengguna lupa kata sandinya. Tampilan halaman didesain dengan latar gradasi warna menarik di sisi kiri dan antarmuka bersih serta minimalis di sisi kanan, menciptakan kesan profesional dan ramah pengguna. Di bagian bawah, tersedia juga tautan menuju halaman pendaftaran bagi pengguna yang belum memiliki akun.. Rincian tampilan halaman *login* dapat dilihat di Gambar 3.17.



Gambar 3.17. Tampilan halaman *login*

B Halaman Sign up

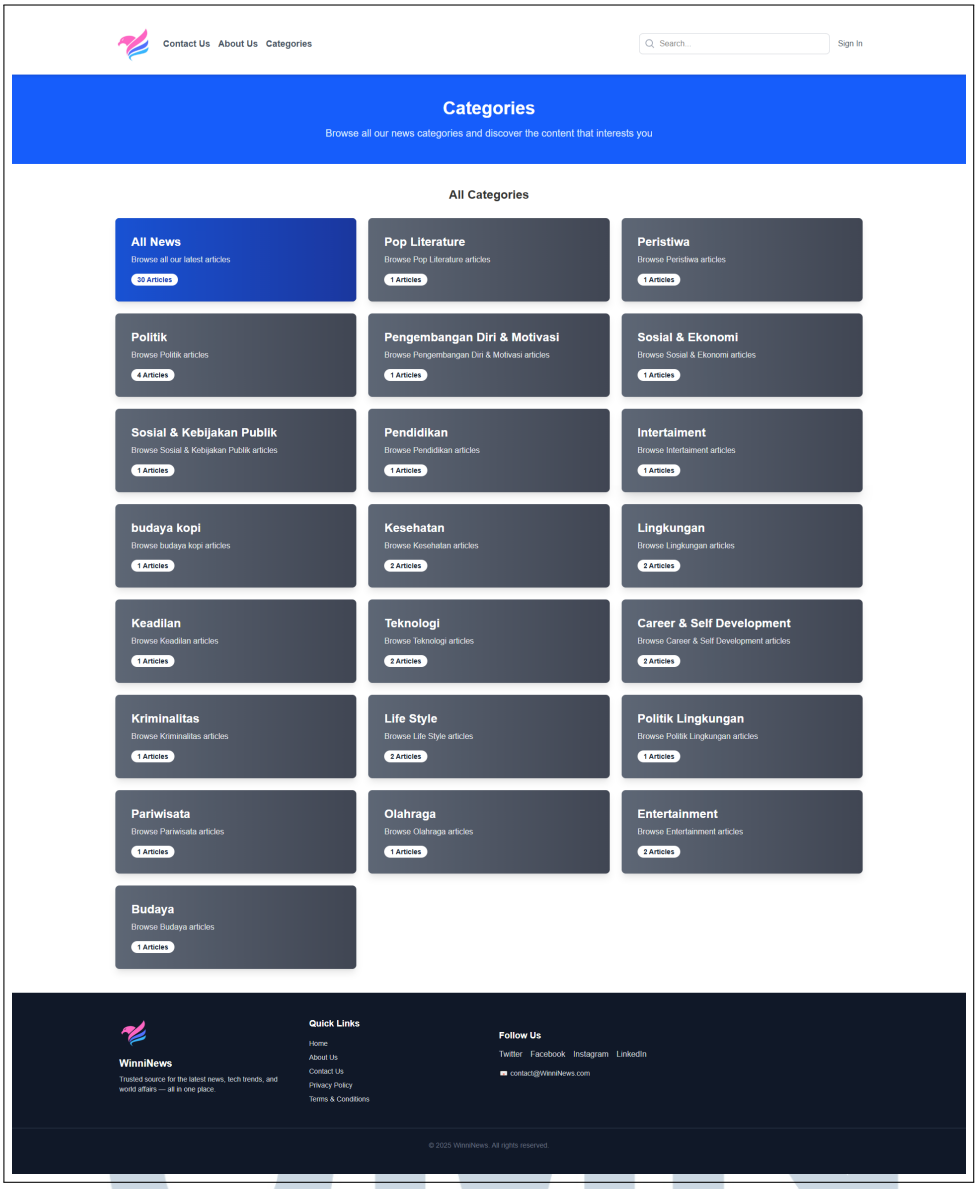
Halaman *Sign Up* merupakan halaman yang dirancang untuk memfasilitasi proses pendaftaran akun baru oleh pengguna. Pengguna diminta untuk mengisi sejumlah informasi penting seperti nama lengkap, alamat *email*, kata sandi, nomor telepon, jenis kelamin, dan tanggal lahir. Seluruh elemen *input* ditata secara vertikal dan terstruktur, sehingga memudahkan pengguna dalam mengisi data dengan cepat dan jelas. Desain *visual* halaman ini memanfaatkan latar belakang gradasi warna cerah di sisi kiri yang memberikan kesan segar dan *modern*, sementara sisi kanan difokuskan pada formulir pendaftaran dengan tampilan minimalis yang bersih dan profesional. Halaman ini bagian dari alur autentikasi yang hanya diakses ketika pengguna memilih untuk membuat akun baru, sebelum diarahkan ke halaman *Login* setelah proses pendaftaran berhasil diselesaikan. Rincian tampilan halaman *Sign up* dapat dilihat di Gambar 3.18



Gambar 3.18. Tampilan halaman *Sign Up*

C Halaman Kategori

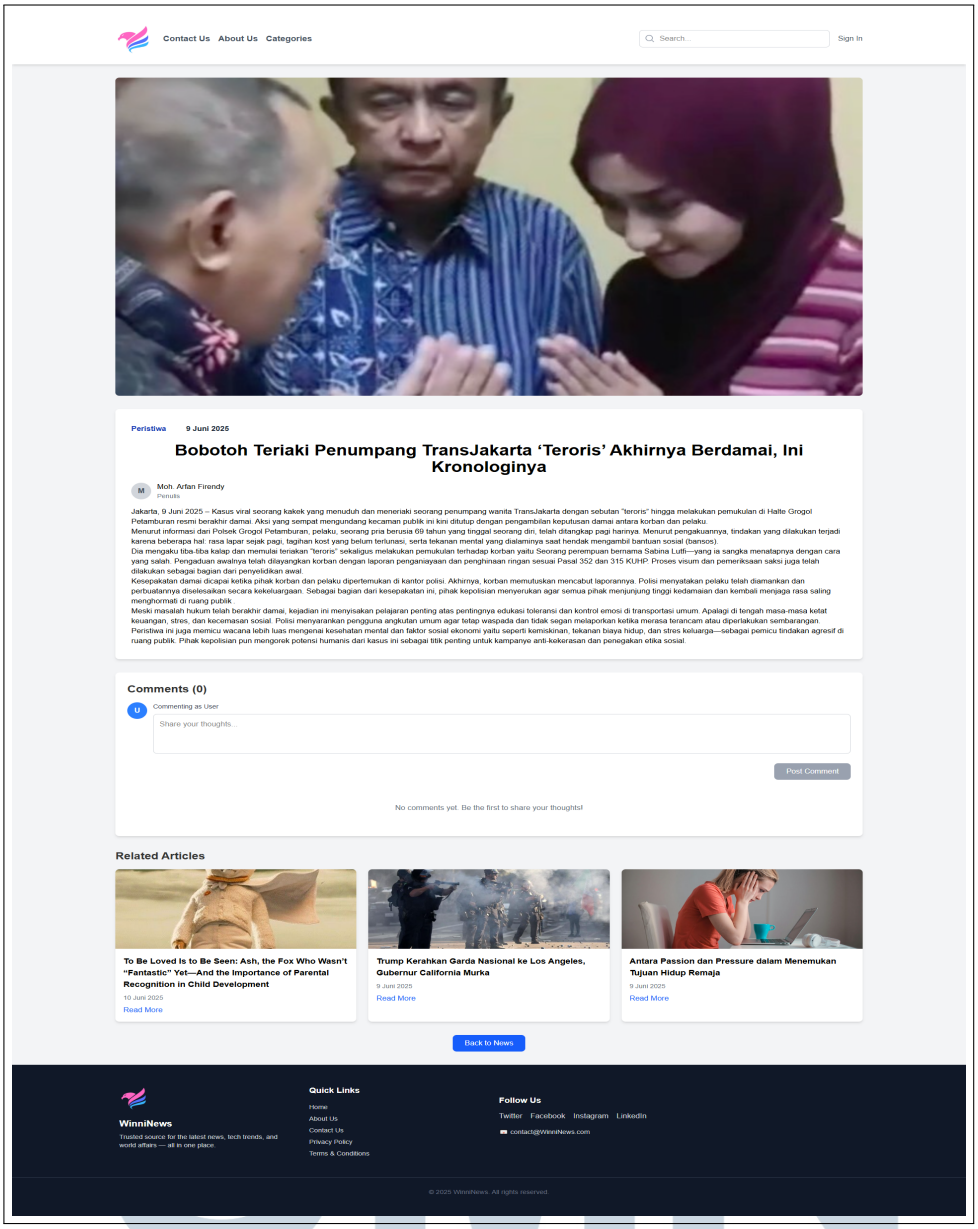
Halaman kategori dari *website* portal berita ini menyajikan daftar lengkap seluruh kategori berita yang tersedia. Hal ini memudahkan pengguna untuk menelusuri konten berdasarkan minat masing-masing pengguna. Terdapat judul *"Categories"* yang memperjelas bahwa halaman yang sedang diakses merupakan halaman kategori. Antarmukanya disusun dengan rapi dan mudah dipahami, sehingga pengguna dapat dengan cepat menemukan informasi yang sesuai dengan ketertarikan yang diminati. Setiap kartu kategori memuat nama kategori, deskripsi singkat, serta jumlah artikel yang tersedia di dalamnya, seperti Politik (*4 Articles*), Teknologi (*2 Articles*), hingga kategori yang lebih unik seperti Budaya Kopi dan *Career & Self Development*. Tata letak berbentuk *grid* yang responsif memungkinkan pengguna melihat semua kategori dalam satu tampilan tanpa perlu banyak menggulir. Fitur ini mempermudah pengguna dalam menemukan berita sesuai minat atau kebutuhan informasi. Rincian tampilan halaman kategori dapat dilihat di Gambar 3.19



Gambar 3.19. Tampilan halaman kategori

D Halaman Detail Berita

Halaman detail berita pada *website* ini menyajikan informasi lengkap mengenai suatu berita, lengkap dengan judul berita, nama penulis, tanggal terbit, serta isi berita secara menyeluruh.Rincian tampilan halaman detail berita dapat dilihat di Gambar 3.20.



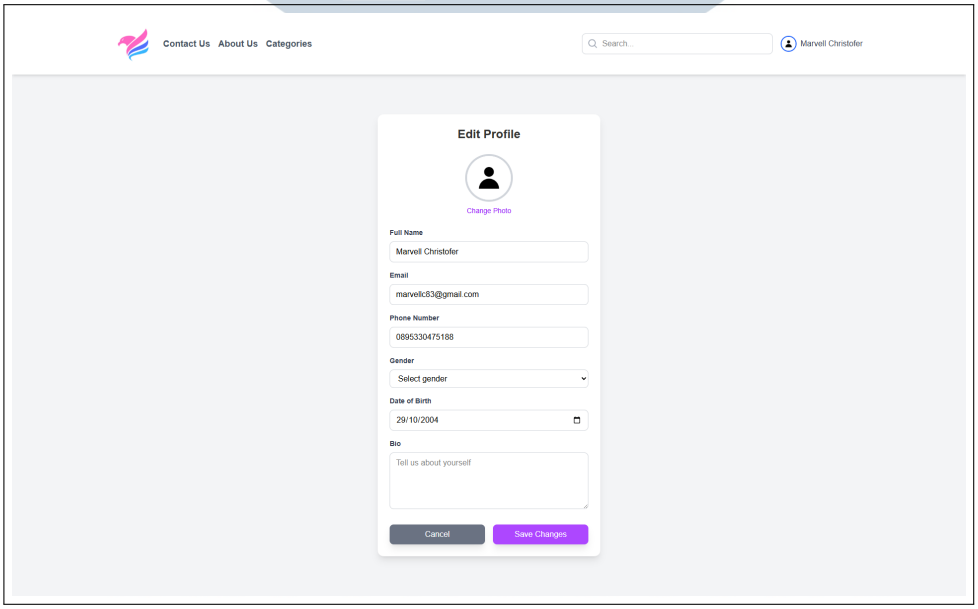
Gambar 3.20. Tampilan halaman detail berita

Tampilan halaman dirancang agar fokus utama tertuju pada konten berita dengan dukungan visual berupa gambar utama di bagian atas. Selain itu, terdapat kolom komentar di bagian bawah artikel yang memungkinkan pengguna memberikan tanggapan, meskipun fitur ini belum diisi oleh pengguna. Di bagian akhir halaman, disediakan juga bagian *Related Articles* yang menampilkan artikel-artikel terkait sebagai referensi tambahan bagi pembaca. Navigasi kembali ke halaman utama berita juga disediakan secara jelas melalui tombol *Back to News*.

Desain antarmukanya sederhana dan informatif, sehingga memudahkan pengguna untuk memahami isi berita dan menjelajahi konten lain yang relevan.

E Halaman Edit Profile

Halaman *Edit Profile* memungkinkan pengguna untuk memperbarui informasi pribadi pengguna dengan cara yang sederhana dan mudah dipahami. Di halaman ini, pengguna dapat mengunggah atau mengubah foto profil, nama lengkap, email, nomor telepon, jenis kelamin, tanggal lahir, dan menuliskan *bio* atau deskripsi singkat tentang diri pengguna. Tersedia dua tombol di bagian bawah: tombol *Save Changes* untuk menyimpan perubahan dan tombol *Cancel* untuk membatalkan dan kembali ke halaman sebelumnya. Tujuan dari halaman ini adalah untuk memberikan kenyamanan dan kebebasan bagi pengguna dalam mengelola informasi pribadinya. Perubahan data yang dilakukan akan langsung disimpan ke dalam *database* setelah pengguna mengkonfirmasi pembaruan. Rincian tampilan halaman utama dapat dilihat di Gambar 3.21

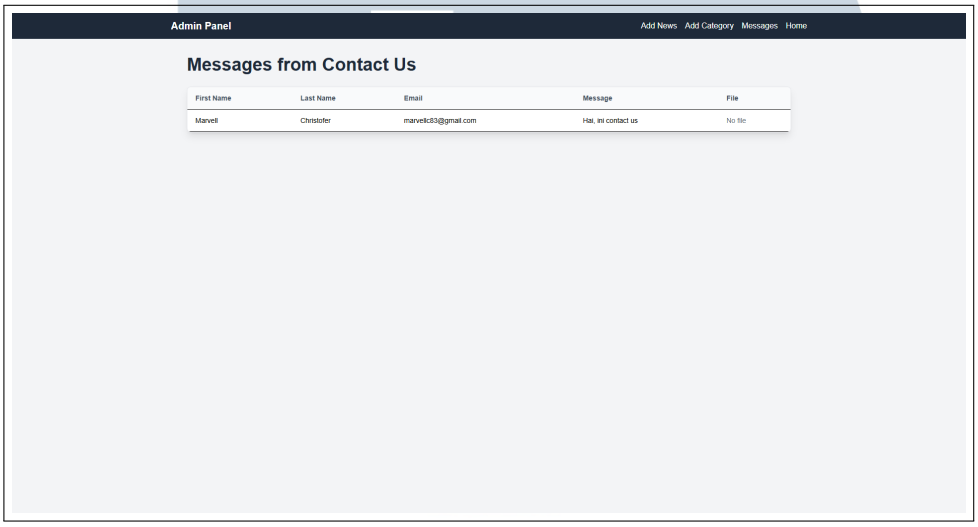


Gambar 3.21. Tampilan halaman *Edit Profile*

F Halaman admin

Halaman *Admin* ini digunakan untuk menampilkan pesan-pesan yang dikirim oleh pengguna melalui formulir di halaman *Contact Us*. Data ditampilkan

dalam bentuk tabel yang terdiri dari kolom *First Name*, *Last Name*, *Email*, *Message*, dan *File*. Admin dapat melihat detail setiap pesan yang masuk, termasuk jika ada file yang diunggah oleh pengguna (jika ada, *file* tersebut akan ditampilkan. Namun jika tidak ada *file*, tertulis "*No file*"). Fitur ini memungkinkan admin untuk memantau, membaca, dan menindaklanjuti setiap masukan atau pertanyaan yang masuk secara efisien. Rincian tampilan halaman utama dapat dilihat di Gambar 3.22



Gambar 3.22. Halaman Admin

3.5 Kendala dan Solusi yang Ditemukan

Selama pelaksanaan kegiatan magang di PT Winnicode Garuda Teknologi, penulis menghadapi beberapa kendala yang memengaruhi kelancaran dalam menyelesaikan tugas dan tanggung jawab yang diberikan. Meskipun demikian, berbagai upaya dilakukan untuk mengatasi hambatan tersebut agar proses magang tetap berjalan secara optimal. Adapun kendala yang dihadapi serta solusi yang diterapkan dijabarkan pada subbab berikut.

3.5.1 Kendala

Kendala yang ditemukan selama pelaksanaan kerja magang di PT Winnicode Garuda Teknologi, yaitu:

1. Bekerja dari rumah berpotensi menimbulkan gangguan dan distraksi,serta menurunkan fokus terhadap tugas yang diberikan.

2. Kurangnya *Feedback* atau arahan langsung dari *supervisor* teknis terkait hasil kerja yang telah diselesaikan. Hal ini menyulitkan proses evaluasi dan pengembangan diri dalam jangka pendek.

3.5.2 Solusi

Solusi yang dilakukan untuk mengatasi kendala yang telah dijabarkan adalah sebagai berikut:

1. Menerapkan jadwal kerja yang disiplin, menjauhi hal-hal yang menyebabkan distraksi untuk menjaga produktivitas, serta menciptakan lingkungan kerja yang kondusif di rumah.
2. Mengambil inisiatif untuk aktif bertanya dan meminta tinjauan terhadap hasil pekerjaan melalui pesan pribadi dan laporan mingguan.