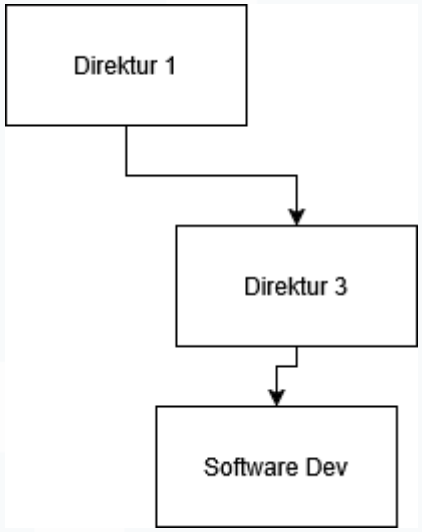
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi



Gambar 3.1 Posisi & Kedudukan di PT Digital Sandi Informasi

Gambar 3.1 menggambarkan struktur kerja magang di PT. Digital Sandi Informasi, di mana mahasiswa ditempatkan di divisi *Software Development* di bawah pengawasan langsung dari Direktur 3, yang mengawasi semua aspek pengembangan teknologi di dalam perusahaan. Divisi *Software Development* memainkan peran penting dalam memastikan bahwa infrastruktur digital perusahaan selaras dengan kebutuhan bisnis, memberikan solusi inovatif untuk efisiensi sistem dan optimalisasi kinerja secara keseluruhan. Meskipun divisi *Software Development* berkolaborasi dengan tim IT untuk dukungan teknologi yang lebih luas, peran mahasiswa bersifat independen, melapor langsung kepada Senior *Software Engineer* dan Supervisor tanpa koordinasi langsung dengan anggota tim lainnya.

Selama magang, mahasiswa bertanggung jawab untuk mengembangkan, mengoptimalkan, dan mendesain ulang alur situs web perusahaan, memastikan bahwa sistem tersebut efisien dan ramah pengguna. Di bawah bimbingan Senior *Software Engineer* dan *Supervisor*, mahasiswa memperoleh pengalaman langsung

dalam pengembangan sistem berbasis Laravel, pemecahan masalah, dan alur kerja rekayasa perangkat lunak di dunia nyata. Pendekatan bimbingan langsung memberikan lingkungan belajar yang terfokus, memungkinkan mahasiswa untuk menerima umpan balik yang konstruktif dan menyempurnakan keterampilan teknis mereka secara progresif.

Sebagai *software engineer*, mahasiswa diberi beberapa tanggung jawab utama, termasuk tetapi tidak terbatas pada:

1) Mengembangkan dan Mengoptimalkan Sistem Berbasis Laravel
    a. Meningkatkan fungsionalitas sistem yang ada untuk meningkatkan kinerja dan pengalaman pengguna.
    b. Mendesain ulang alur situs web, memastikan struktur navigasi yang lebih lancar dan efisien.
    c. Melakukan debug dan menyelesaikan kesalahan sistem, memastikan stabilitas dan konsistensi kinerja.
    d. Menerapkan fitur-fitur baru berdasarkan persyaratan khusus yang ditetapkan oleh supervisor.
2) Analisis dan Peningkatan Kinerja Sistem
    a. Mengoptimalkan arsitektur sistem dan alur kerja untuk efisiensi yang lebih baik.
    b. Menulis dan mengeksekusi kode yang dioptimalkan untuk meningkatkan kecepatan dan daya tanggap sistem.
    c. Melakukan pengujian sistem untuk memastikan stabilitas, keamanan, dan fungsionalitas secara keseluruhan.
3) Pengembangan dan Implementasi Otomasi
    a. Menerapkan otomatisasi pengikisan data untuk merampingkan manajemen informasi.
    b. Memastikan bahwa data yang dikumpulkan diproses, terstruktur, dan terintegrasi secara efektif di dalam sistem.
4) Pelaporan dan Koordinasi

a. Semua kemajuan proyek dilaporkan langsung kepada Senior *Software Engineer* dan *Supervisor*.
b. Tidak ada koordinasi langsung yang diperlukan dengan anggota tim lainnya, dengan semua komunikasi yang terjadi secara eksklusif antara mahasiswa dan insinyur senior.

Selama empat bulan magang, mahasiswa bekerja sepenuhnya di lokasi, dengan jadwal harian yang terstruktur dari pukul 09:00 hingga 18:00 WIB, dengan sesi lembur tambahan selama satu jam hingga pukul 19:00 WIB. Lingkungan berbasis kantor yang penuh waktu ini memungkinkan komunikasi yang lancar dan bimbingan langsung, memastikan bahwa mahasiswa dapat secara aktif terlibat dengan tugas-tugas mereka dan menerima umpan balik secara *real-time*. Meskipun WhatsApp digunakan untuk diskusi sesekali, sebagian besar komunikasi dilakukan secara tatap muka di kantor, memperkuat alur kerja yang terstruktur dan pendelegasian tugas yang efisien.

Pengalaman magang ini memberikan paparan yang berharga terhadap tantangan rekayasa perangkat lunak di dunia nyata, yang memungkinkan mahasiswa untuk mengembangkan pemahaman yang lebih dalam tentang arsitektur sistem berbasis Laravel, metodologi debugging, dan strategi otomatisasi. Melalui pelatihan teknis yang ketat dan bimbingan langsung, mahasiswa dapat meningkatkan kemampuan pemecahan masalah, meningkatkan kemahiran teknis, dan mendapatkan wawasan tentang standar pengembangan perangkat lunak profesional. Pengalaman langsung dan pendekatan pembelajaran terstruktur secara signifikan berkontribusi pada pertumbuhan profesional mahasiswa, mempersiapkan mereka untuk upaya masa depan dalam industri rekayasa perangkat lunak.

3.2 Tugas dan Uraian Kerja Magang

Program magang di PT. Digital Sandi Informasi berlangsung selama empat bulan, dari 3 Februari 2024, hingga 3 Juni 2024, di mana mahasiswa secara aktif terlibat dalam berbagai tugas yang berkaitan dengan pengembangan web, optimalisasi sistem, dan otomatisasi. Selama magang, mahasiswa bekerja di lokasi

secara penuh waktu, dengan semua komunikasi dilakukan secara tatap muka dengan Senior *Software Engineer* dan Supervisor, tanpa koordinasi dengan anggota tim lainnya. Program magang ini memberikan kesempatan untuk menerapkan pengetahuan teknis dalam lingkungan dunia nyata, meningkatkan keterampilan pemecahan masalah, dan mendapatkan pengalaman langsung dalam mengembangkan dan mengoptimalkan sistem berbasis web.

Salah satu tugas utama yang diberikan selama magang adalah mendesain ulang alur situs web untuk meningkatkan efisiensi dan pengalaman pengguna. Hal ini melibatkan analisis struktur yang ada, mengidentifikasi hambatan, dan menerapkan alur kerja yang lebih efisien untuk memastikan bahwa sistem beroperasi lebih efektif. Desain ulang ini bertujuan untuk meningkatkan navigasi, mengurangi langkah-langkah yang tidak perlu dalam proses, dan menciptakan antarmuka pengguna yang lebih intuitif. Melalui proses ini, mahasiswa mengembangkan pemahaman yang lebih dalam tentang cara mengoptimalkan arsitektur sistem untuk kegunaan dan kinerja yang lebih baik.

Selain itu, mahasiswa memainkan peran penting dalam menerapkan otomatisasi untuk merampingkan proses pengumpulan data. Fokus utamanya adalah menciptakan sistem penggalian data otomatis yang dapat mengumpulkan, memproses, dan menyimpan informasi secara efisien dengan intervensi manual yang minimal. Dalam memanfaatkan alat otomatisasi, mahasiswa membantu mengurangi tugas-tugas yang berulang, sehingga meningkatkan produktivitas dan efisiensi dalam sistem. Implementasi otomatisasi *scraping* juga membutuhkan pengujian yang ketat untuk memastikan keakuratan dan keandalan data yang dikumpulkan.

Pengalaman langsung yang diperoleh selama magang memberikan wawasan yang berharga ke dalam praktik rekayasa perangkat lunak dunia nyata, termasuk pemecahan masalah yang sistematis, strategi pengoptimalan, dan alur kerja pengembangan perangkat lunak profesional. Kemampuan untuk bekerja secara mandiri sambil menerima bimbingan langsung dari mentor yang berpengalaman secara signifikan meningkatkan keterampilan teknis dan analitis mahasiswa.

Pengetahuan yang diperoleh selama magang diharapkan dapat menjadi fondasi yang kuat untuk usaha masa depan di bidang pengembangan perangkat lunak.

Tabel 3.1 Tugas & Urain Kerja Magang

No	Aktivitas	Waktu Mulai	Waktu Selesai
1	Pengenalan Kantor & Pembahasan *Jobdesc*	3 Februari 2025	4 Februari 2025
2	Pembuatan Konsep Otomatisasi *Scraping* Data	5 Februari 2025	18 Februari 2025
3	Integrasi *Scraping* Data Dengan *Website*	19 Februari 2025	27 Maret 2025
4	Mendesain *Flow Website*	5 Februari 2025	27 Maret 2025
5	Perbaikan *Website*	8 April 2025	2 Mei 2025
6	Mendesain *Flow Database*	8 April 2025	2 Mei 2025
7	Integrasi Ulang *Website* & *Database*	5 Mei 2025	3 Juni 2025

Selama proses kerja magang di PT. Digital Sandi Informasi, berbagai perangkat lunak dan perangkat keras diperlukan untuk mendukung tugas pengembangan dan optimasi sistem. Di bawah ini adalah alat-alat yang digunakan oleh mahasiswa:

1) *Software*
   a. Notebook Jupyter: Digunakan untuk membuat skrip dan mengotomatiskan tugas pemrosesan data.
   b. Visual Studio Code: Editor kode utama untuk mengembangkan dan men-debug aplikasi web.
   c. Google Chrome: Browser web yang digunakan untuk menguji dan men-debug aplikasi web.

d. Gmail: Platform komunikasi untuk menerima pembaruan tugas dan melaporkan kemajuan.
e. WhatsApp: Alat komunikasi cepat untuk mendiskusikan masalah teknis dengan supervisor.

2) Hardware

a. Prosesor: AMD Ryzen 3 5300U
b. RAM: 12GB

3.2.1 Pengenalan Kantor & Pembahasan *Jobdesc*

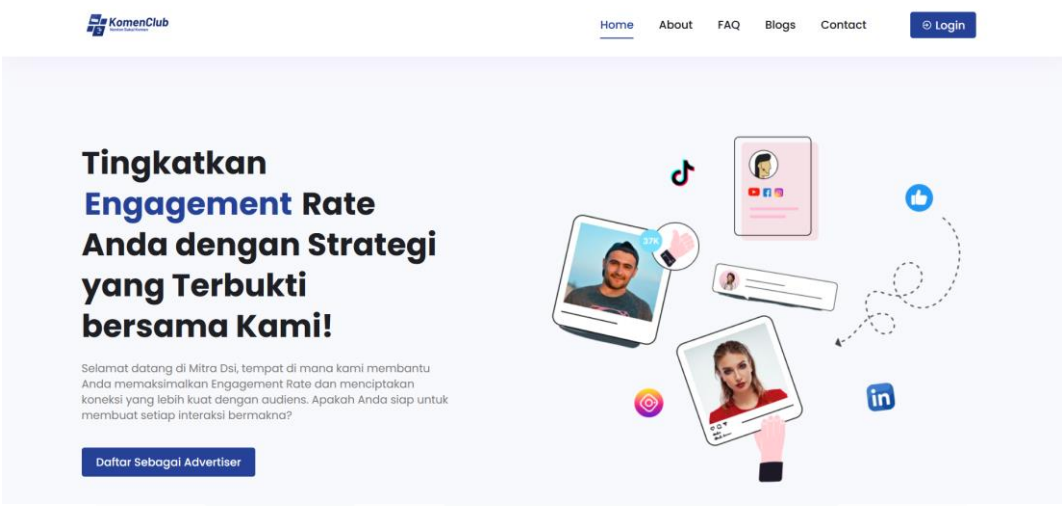
Selama minggu pertama magang di PT. Digital Sandi Informasi, yang berlangsung dari tanggal 3 Februari 2025 hingga 3 Juni 2025, mahasiswa diperkenalkan dengan lingkungan kerja dan struktur tim perusahaan. Fase pengenalan berlangsung dari tanggal 3 Februari hingga 4 Februari 2025, memungkinkan mahasiswa untuk membiasakan diri dengan divisi *Software Development* dan memahami alur kerja secara keseluruhan dalam organisasi. Fase ini mencakup pengenalan umum tentang operasi perusahaan, berbagai divisi dalam organisasi, dan tanggung jawab utama tim *software developer*.

Senior *Software Engineer* dan *Supervisor* memberikan pengarahan terperinci mengenai ruang lingkup pekerjaan magang dan tugas-tugas spesifik yang diberikan untuk periode empat bulan. Tanggung jawab utama termasuk mengembangkan otomatisasi untuk penggalian data, mengintegrasikan otomatisasi data dengan situs web perusahaan, mendesain ulang dan mengoptimalkan alur situs web, dan bekerja dengan Laravel dan MySQL untuk meningkatkan efisiensi sistem. Pengarahan ini juga mencakup jadwal proyek, hasil yang diharapkan, dan tolok ukur kinerja untuk memastikan mahasiswa memiliki pemahaman yang jelas tentang peran mereka.

Fase perkenalan ini membantu mahasiswa memahami alur kerja dalam divisi *software developer* dan memahami bagaimana berbagai proyek disusun. Selain itu, pengarahan ini memberikan wawasan tentang praktik dalam *software development*, metodologi manajemen proyek, dan ekspektasi untuk memberikan solusi berkualitas tinggi. Setelah menyelesaikan orientasi,

mahasiswa secara resmi mulai mengerjakan proyek yang ditugaskan, dimulai dengan konseptualisasi otomatisasi untuk pengikisan data sebelum beralih ke integrasi situs web dan peningkatan sistem.

3.2.2 Desain Ulang *Flow* dan Memperbaiki *Website*



Gambar 3.2 *Home Screen Website*

Gambar 3.2 merupakan gambar halaman utama saat baru masuk ke *website*. Pada proses desain ulang situs web PT. Digital Sandi Informasi (DSI), peningkatan yang signifikan dan strategis dilakukan untuk meningkatkan efisiensi dan efektivitas pengelolaan kampanye promosi digital secara keseluruhan. Salah satu perubahan yang paling menonjol dalam alur baru ini berkaitan dengan proses verifikasi tugas keterlibatan pengguna.

Pada sistem sebelumnya, pemilik merek atau kampanye bertanggung jawab secara langsung untuk memverifikasi apakah pengguna telah menyelesaikan tindakan yang ditugaskan, seperti mengomentari konten media sosial tertentu atau platform lain yang ditunjuk. Namun, dalam alur yang baru didesain ulang, tanggung jawab ini telah sepenuhnya dialihkan ke tim admin internal.

Transisi ini dilakukan sebagai bagian dari upaya untuk memusatkan dan menstandarisasi proses verifikasi, memastikan prosesnya lebih konsisten, dan akurat. Penugasan validasi ini diberikan kepada tim admin, platform ini

memastikan akurasi lebih tinggi dan lebih sedikit kesalahan, karena proses ini dikelola oleh mereka yang memiliki kendali penuh atas parameter evaluasi. Selain itu, perubahan ini memberikan keuntungan signifikan bagi *brand*, membebaskan mereka dari beban melakukan verifikasi manual yang memakan waktu.

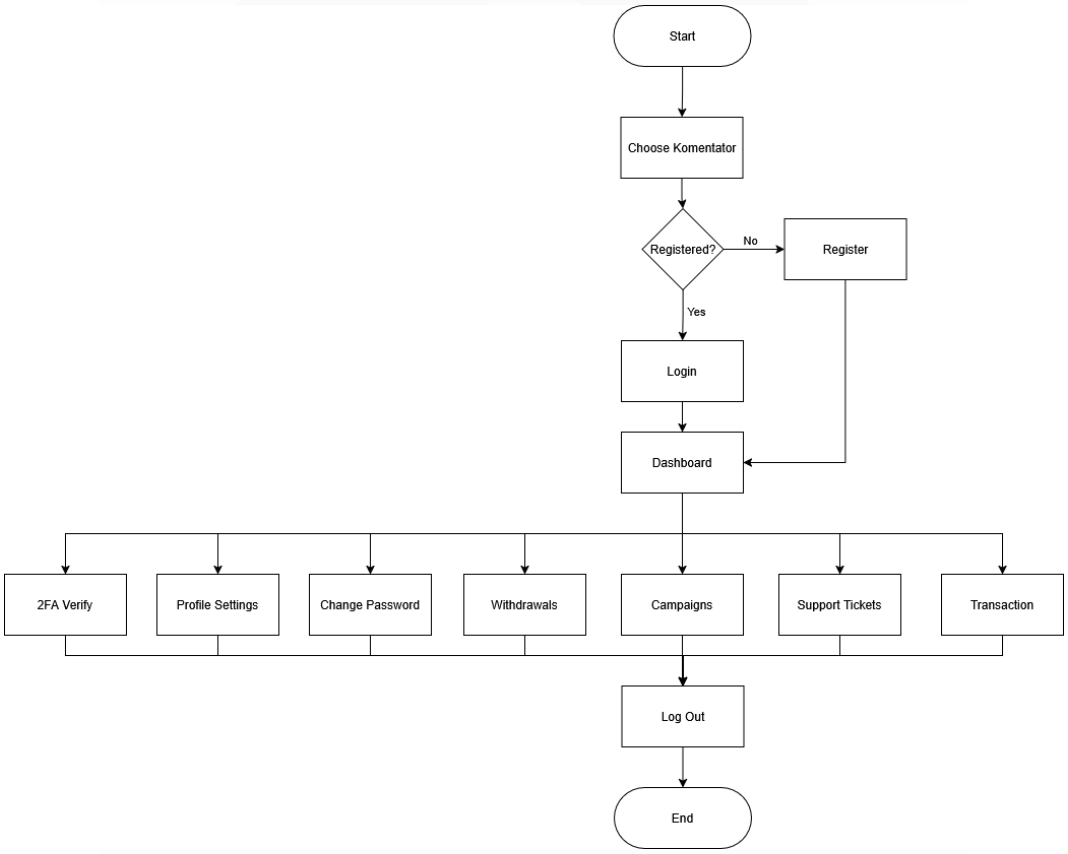
Desain ulang ini tidak hanya mewakili penyempurnaan teknis tetapi juga berfungsi sebagai peningkatan strategis terhadap pengalaman pengguna secara keseluruhan, menguntungkan baik bagi merek maupun pengguna platform. Adanya sistem verifikasi terpusat, alur kerja menjadi lebih cepat, transparan, dan akurat. Hal ini secara langsung berkontribusi dalam membangun kepercayaan pengguna terhadap platform sekaligus meminimalkan potensi perselisihan atau salah tafsir terkait penyelesaian tugas.

Selain itu, situs secara khusus dirancang sebagai platform promosi berbasis tugas yang bertujuan untuk meningkatkan tingkat keterlibatan merek melalui tindakan terstruktur, berskala besar, dan otentik. *Website* memungkinkan *brand* untuk membuat kampanye yang memberikan tugas keterlibatan nyata kepada pengguna (biasanya disebut sebagai komentator atau *influencer*), seperti meninggalkan komentar, mengikuti akun, berbagi postingan, dan aktivitas interaktif lainnya. Tugas-tugas ini secara alami meningkatkan visibilitas *brand* dan membantu mereka menjangkau audiens yang lebih luas.

Adanya sistem ini, *brand* tidak hanya mendapatkan eksposur lebih tinggi, tetapi juga memiliki akses *realtime* tentang kampanye mereka melalui fitur-fitur seperti status pekerjaan, dan aktivitas terbaru. Di sisi lain, pengguna yang menyelesaikan tugas diberikan akan mendapatkan kompensasi sesuai dengan ketentuan kampanye. Platform ini bertindak sebagai jembatan penghubung kebutuhan promosi merek dengan kontribusi aktif dari komunitas pengguna, menumbuhkan ekosistem digital yang saling

menguntungkan. Berikut merupakan hasil *design* ulang *flow* dan penjelasan *website*:

3.2.2.1 Komentator



Gambar 3.3 *Flow Website* Komentator

Gambar 3.3 merupakan *flow website* berbasis tugas yang digunakan oleh PT. Digital Sandi Informasi (DSI), komentator memainkan peran penting dalam keberhasilan kampanye *brand*. Komentator ini adalah pengguna yang secara sukarela melakukan tugas-tugas dengan meninggalkan komentar bermakna pada konten yang ditentukan sebagai bagian dari upaya *brand* untuk meningkatkan visibilitas.

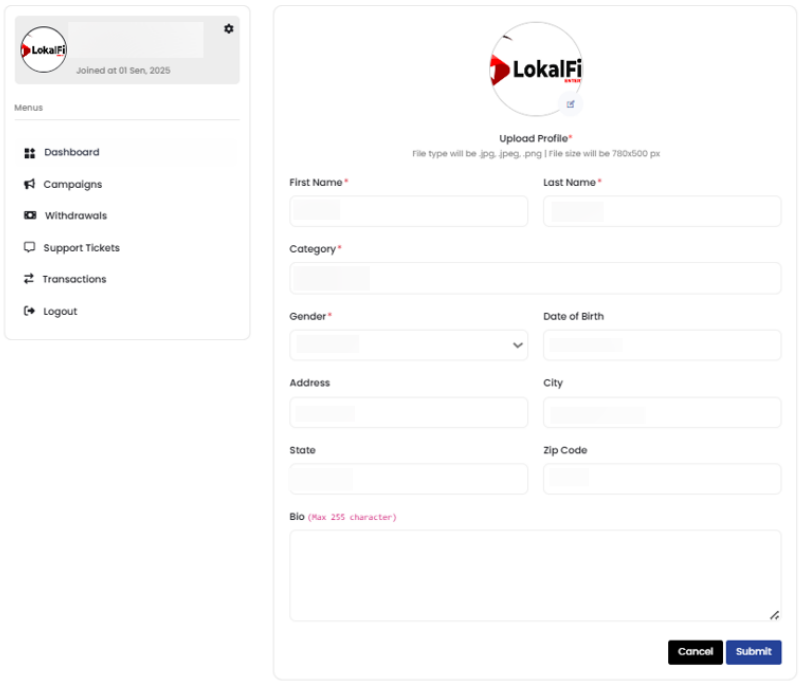
Untuk menjadi komentator aktif, pengguna harus terlebih dahulu berlangganan ke platform, masuk ke dalam model *membership*, berfungsi sebagai bentuk komitmen dan metode monetisasi untuk

platform. Sistem berlangganan ini memastikan bahwa hanya individu yang memiliki dedikasi yang diberikan akses ke platform ini, sehingga memungkinkan pelaksanaan tugas yang lebih profesional.

Adanya pendekatan ini, *brand* dapat menggunakan cara yang sangat interaktif untuk meningkatkan metrik *engagement*, sekaligus memastikan akuntabilitas dan keaslian melalui verifikasi admin yang terpusat. Untuk komentator, model ini menciptakan jalur terstruktur untuk mendapatkan imbalan sambil berkontribusi pada kampanye.

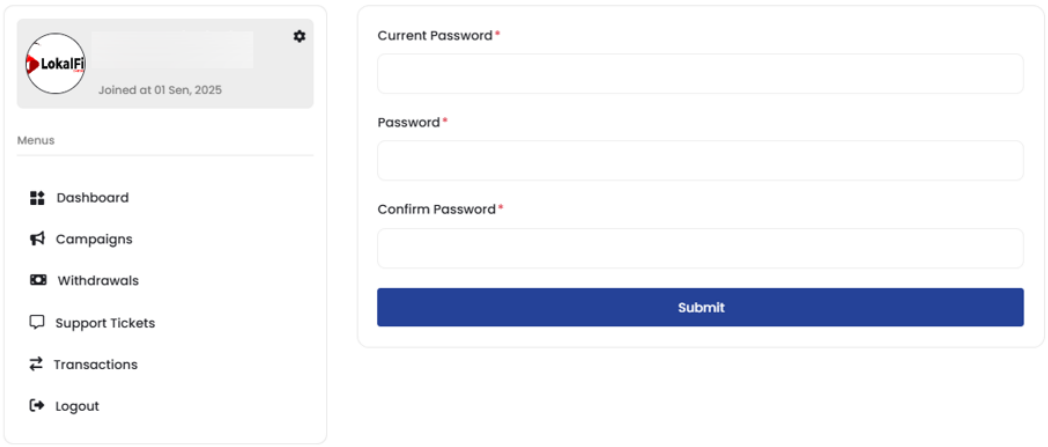
Secara keseluruhan, *website* ini menjadikan sebagai platform promosi berbasis komunitas di mana pengguna terlibat dengan konten untuk meningkatkan eksposur merek, memperdalam interaksi pengguna, dan memastikan nilai timbal balik antara brand dan pemberi komentar.

Setelah pengguna memilih peran sebagai komentator, mereka akan diarahkan ke tampilan *login* utama platform. Setelah berhasil masuk, pengguna mendapatkan akses ke Dasbor, berfungsi sebagai pusat untuk semua aktivitas yang terkait dengan tanggung jawab promosi berbasis tugas mereka. Dasbor memberikan gambaran umum yang komprehensif tentang kinerja dan riwayat interaksi komentator. Termasuk akses ke tugas kampanye yang tersedia, saldo saat ini, riwayat penarikan, pengajuan tiket dukungan, dan catatan transaksi secara keseluruhan. Hal ini memungkinkan komentator untuk memantau kemajuan dan penghasilan mereka secara efisien, sambil tetap memiliki akses langsung ke dukungan platform saat dibutuhkan.



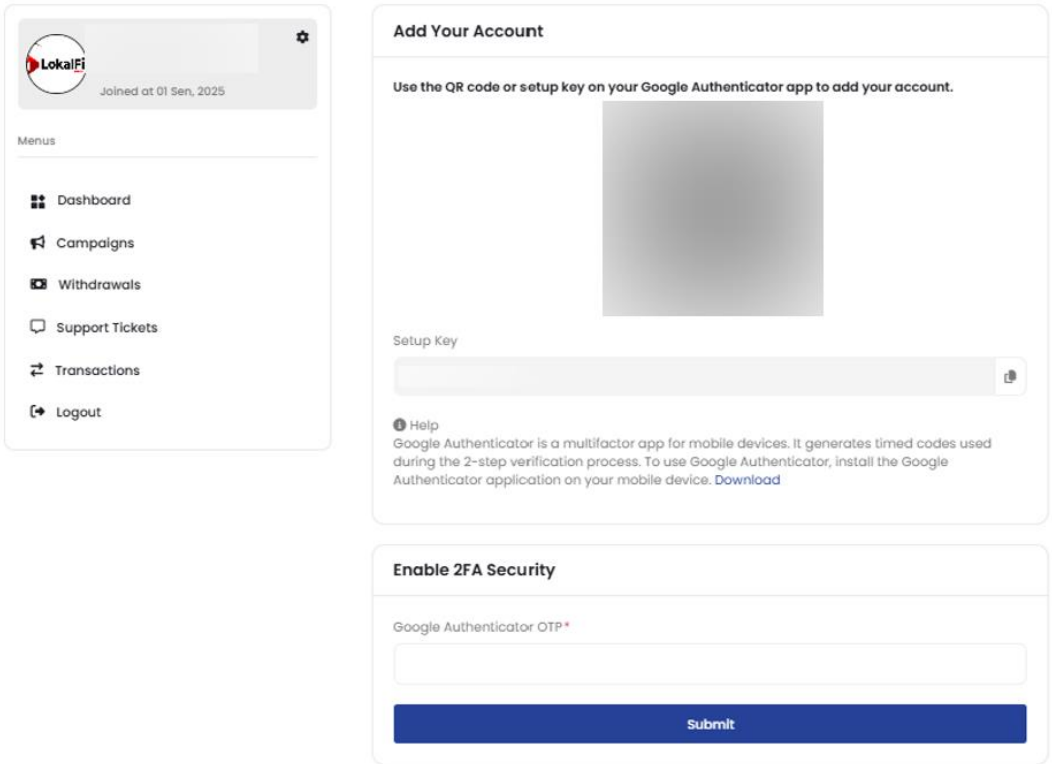
Gambar 3.4 *Page Profile Settings* Komentator

Gambar 3.4 merupakan gambar *profile settings*, pengguna dapat memperbarui informasi pribadi mereka. Bagian ini mencakup kemampuan untuk mengunggah foto profil dan mengedit detail pribadi utama seperti nama depan, nama belakang, dan kategori. Pengguna juga dapat menentukan jenis kelamin, tanggal lahir, alamat, kota, negara bagian, dan kode pos. Selain itu, terdapat kolom bagi pengguna untuk menulis biografi singkat, yang memungkinkan mereka untuk memberikan deskripsi singkat tentang diri mereka sendiri. Fitur-fitur ini membantu memastikan bahwa profil setiap komentator lengkap dan dipersonalisasi, sehingga mendukung kehadiran yang lebih dapat dipercaya dan profesional di platform.



Gambar 3.5 *Page Change Password* Komentator

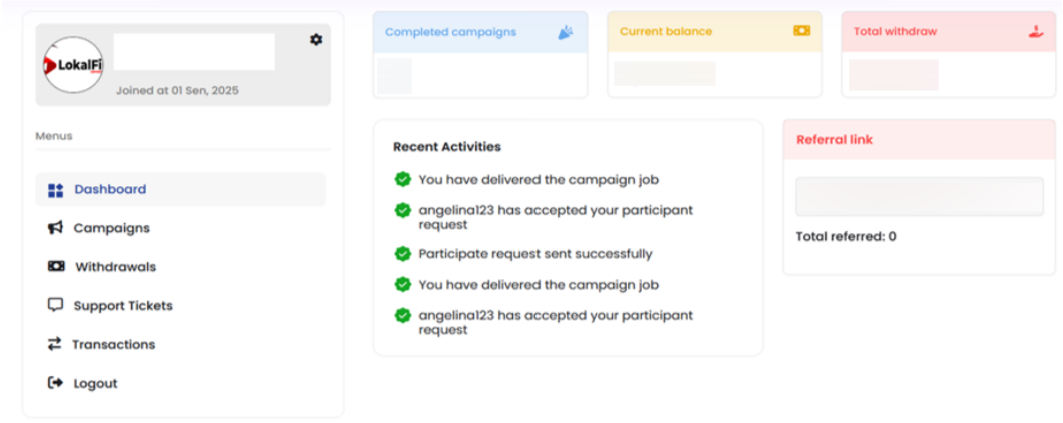
Gambar 3.5 merupakan bagian *Change Password*, pengguna diberikan kemampuan untuk memperbarui keamanan akun mereka dengan mengatur kata sandi baru.



Gambar 3.6 *Page* 2FA Komentator

Halaman Keamanan 2FA menyediakan lapisan tambahan perlindungan akun dengan mengaktifkan Autentikasi Dua Faktor

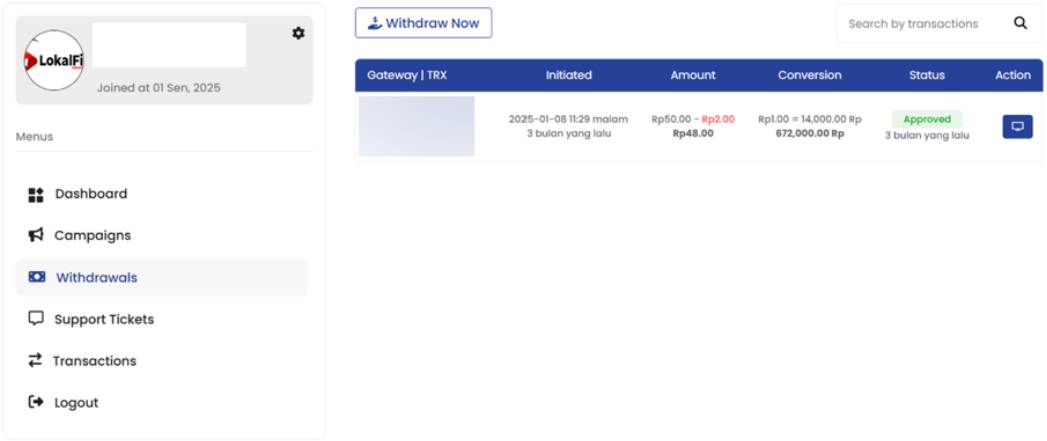
menggunakan Google *Authenticator*. Pada halaman ini, pengguna diminta untuk memindai kode QR atau secara manual memasukkan kunci pengaturan ke dalam aplikasi Google *Authenticator* di perangkat seluler mereka. Setelah dikonfigurasi, aplikasi akan menghasilkan kata sandi sekali pakai (OTP) berbasis waktu yang harus dimasukkan selama proses *login*, menambahkan langkah verifikasi tambahan. Hal ini memastikan bahwa meskipun seseorang mengetahui kredensial akun, mereka tidak dapat mengakses akun tanpa kode sementara dari perangkat pengguna. Bagian bantuan juga disediakan untuk memandu pengguna melalui instalasi dan pengaturan aplikasi Google *Authenticator*, sehingga lebih mudah untuk mengamankan akun mereka secara efektif.



Gambar 3.7 Page Dasbor Komentator

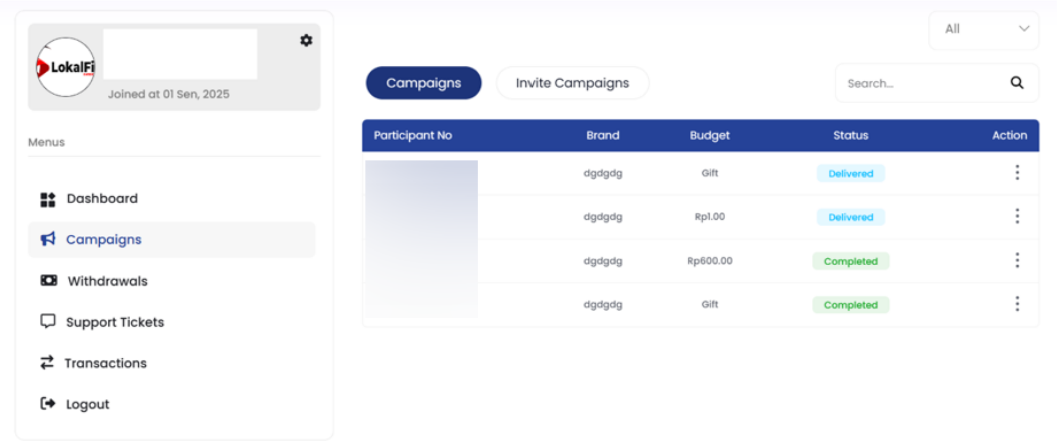
Gambar 3.7 merupakan dasbor komentator untuk menavigasi ke empat modul: *Withdraw, Campaign, Support Ticket,* dan *Transactions*. Bagian *Withdrawals* memungkinkan pengguna untuk mengajukan permintaan pembayaran setelah mereka mengumpulkan penghasilan dari tugas yang telah diselesaikan. Bagian Kampanye menampilkan daftar tugas promosi aktif yang ditugaskan oleh admin, berdasarkan persyaratan kampanye *brand*. Di bagian ini, komentator dapat melihat instruksi serta *link* ke konten mereka ikuti.

Dari Dasbor, komentator dapat menavigasi ke empat modul utama: *Withdraw, Campaign, Support Ticket,* dan *Transactions*. Bagian *Withdrawals* memungkinkan pengguna untuk mengajukan permintaan pembayaran setelah mereka mengumpulkan penghasilan dari tugas yang telah diselesaikan. Bagian Kampanye menampilkan daftar tugas promosi aktif yang ditugaskan oleh admin, berdasarkan persyaratan kampanye *brand*. Di bagian ini, komentator dapat melihat instruksi serta *link* ke konten mereka ikuti.



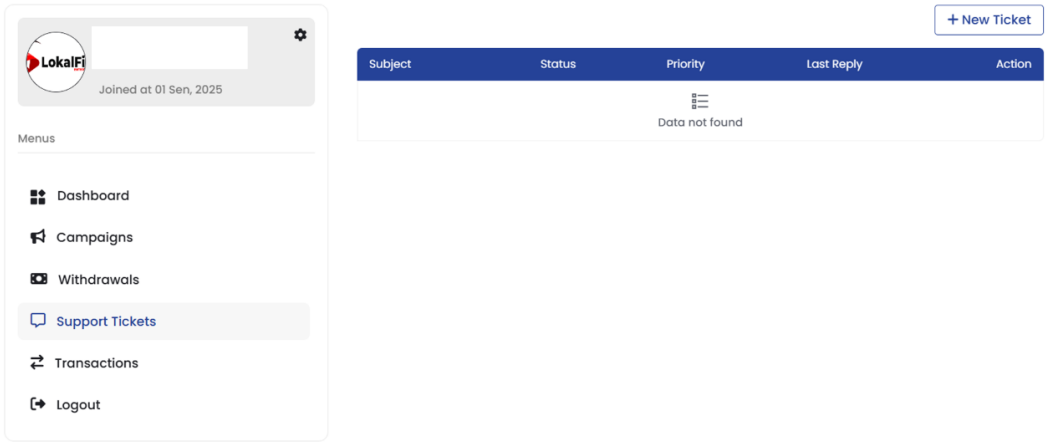
Gambar 3.8 *Page Withdrawal* Komentator

Gambar 3.8 merupakan modul *Withdrawals*. Bagian ini memungkinkan pengguna untuk meminta pembayaran setelah mengumpulkan penghasilan dari tugas yang berhasil diselesaikan. Proses ini terintegrasi ke dalam platform untuk memastikan bahwa setiap permintaan penarikan diverifikasi dan diproses secara efisien, berdasarkan saldo komentator yang tersedia.



Gambar 3.9 *Page Campaigns* Komentator

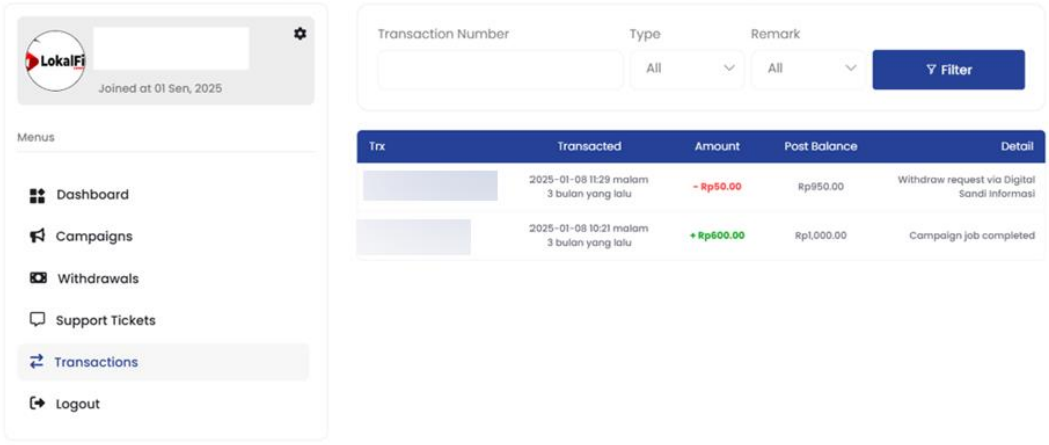
Gambar 3.9 merupakan bagian *Campaigns.* Bagian ini menampilkan daftar tugas promosi aktif yang ditugaskan oleh admin, berdasarkan persyaratan kampanye *brand*. Di sini, komentator dapat melihat info terperinci dan mengakses tautan ke konten yang harus mereka ikuti. Modul ini berfungsi sebagai ruang kerja utama bagi komentator untuk menyelesaikan tugas promosi mereka.



Gambar 3.10 *Page Support Tickets* Komentator

Gambar 3.10 merupakan bagian *Support Tickets* yang berfungsi sebagai saluran komunikasi antara komentator dan tim dukungan platform. Ini memungkinkan pengguna untuk melaporkan masalah, mengajukan pertanyaan, atau meminta bantuan teknis. Hal ini

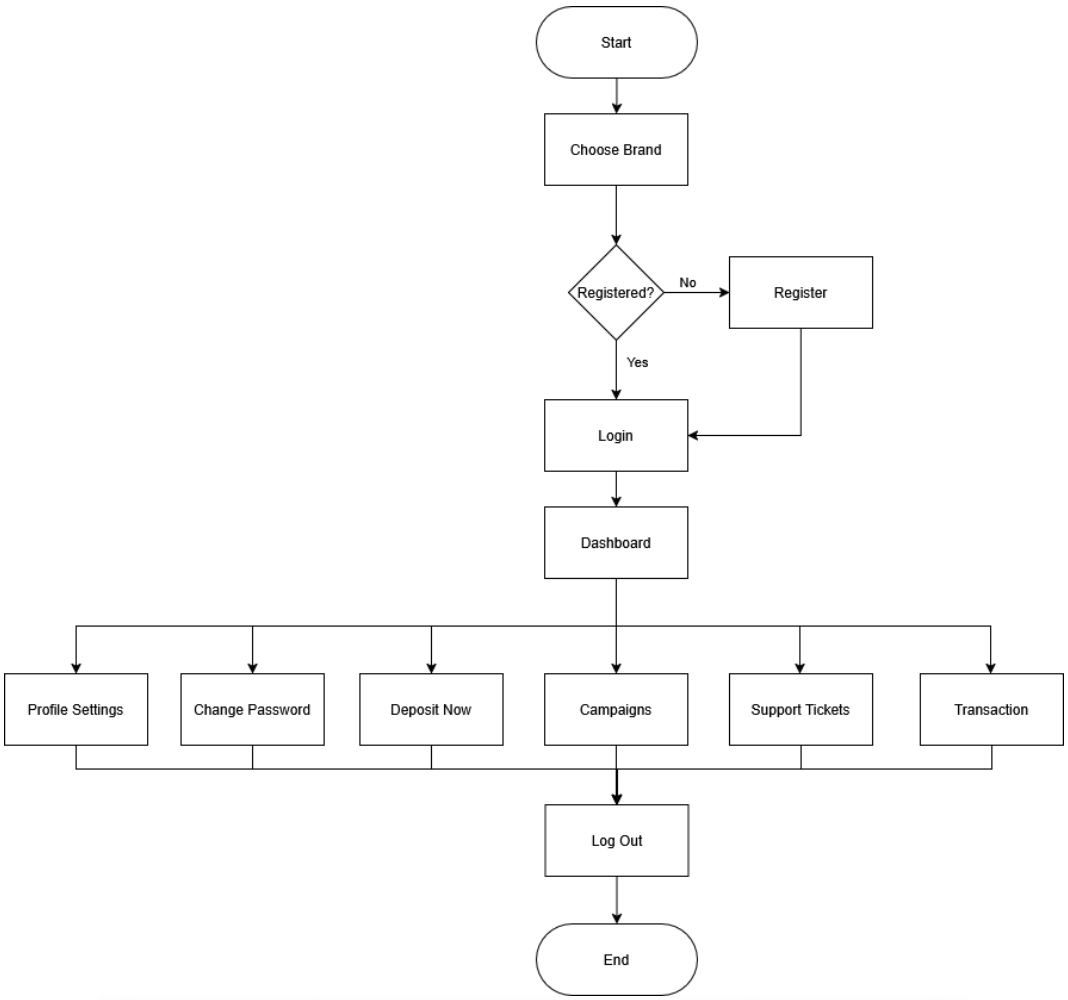
memastikan bahwa komentator memiliki akses ke bantuan setiap kali mereka menghadapi kendala selama pelaksanaan tugas.



Gambar 3.11 *Page Transactions* Komentator

Gambar 3.11 merupakan bagian *Transactions* menyediakan catatan lengkap dari semua aktivitas keuangan dan tindakan yang berhubungan dengan tugas. Ini termasuk pembayaran langganan, hadiah kampanye, dan riwayat penarikan. Modul ini membantu komentator melacak kinerja dan penghasilan mereka dengan jelas.

3.2.2.2 Brand



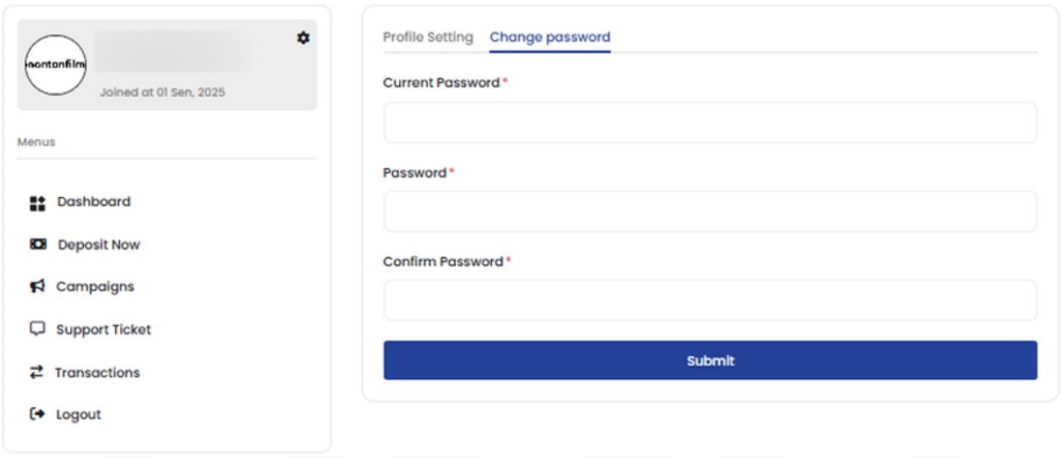
Gambar 3.12 *Flow Website Brand*

Gambar 3.12 merupakan *flow website brand,* sebagai *brand*, tanggung jawab utama adalah membuat *campaign* yang nantinya akan ditugaskan kepada komentator untuk dieksekusi. Tidak seperti komentator, *brand* tidak perlu membeli langganan untuk mengunggah *campaign*. Sebaliknya, mereka diharuskan untuk menyetor dana ke dalam saldo mereka. Pembuatan *campaign* hanya diperbolehkan selama saldo tersedia.

Setelah *campaign* dikirimkan, *campaign* akan diperiksa terlebih dahulu oleh admin untuk menentukan apakah konten video yang disediakan oleh *brand* memenuhi standar platform. Jika disetujui,

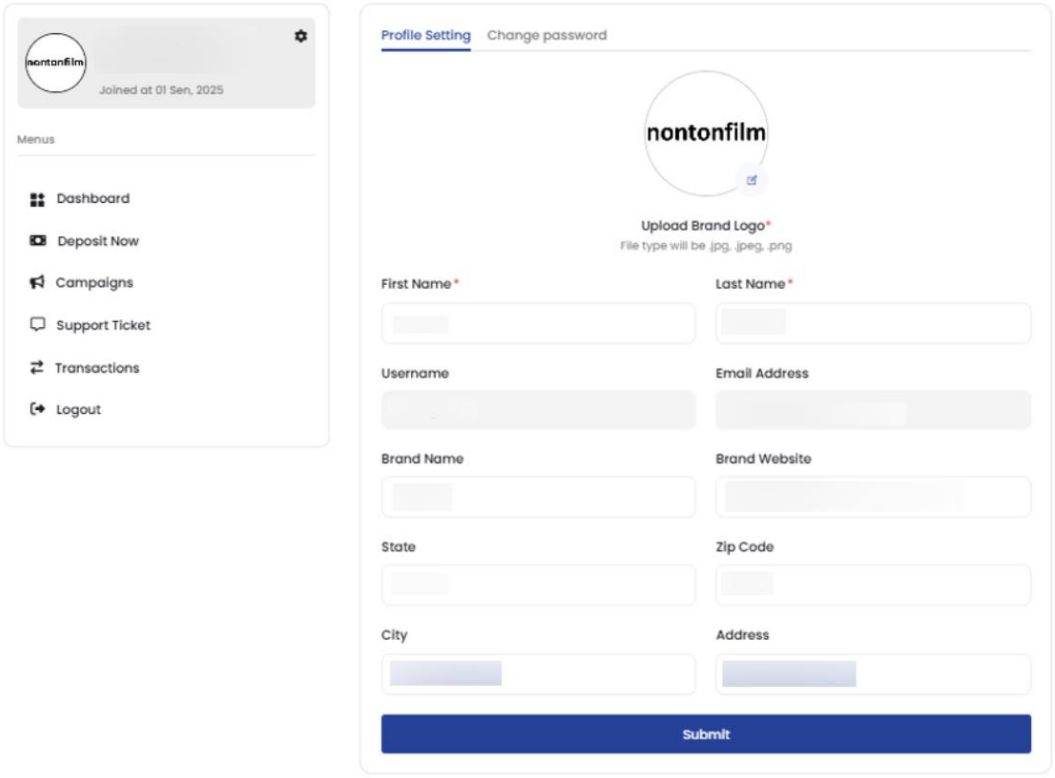
campaign akan diteruskan ke komentator, yang kemudian akan melakukan tugas memberikan komentar pada video yang disediakan. Jika konten tidak memenuhi standar, *campaign* akan ditolak.

Setelah komentator menyelesaikan tugasnya, hasilnya akan diperiksa kembali oleh admin untuk diverifikasi. Setelah disetujui, platform akan memberi tahu *brand* bahwa tugas komentar telah berhasil diselesaikan dan diverifikasi. Pada titik ini, komentator diberi imbalan, dan jumlah yang sesuai akan dipotong dari saldo *brand*. Cara ini memastikan proses yang terkendali, terverifikasi, dan dapat dipertanggung jawabkan untuk setiap eksekusi kampanye.



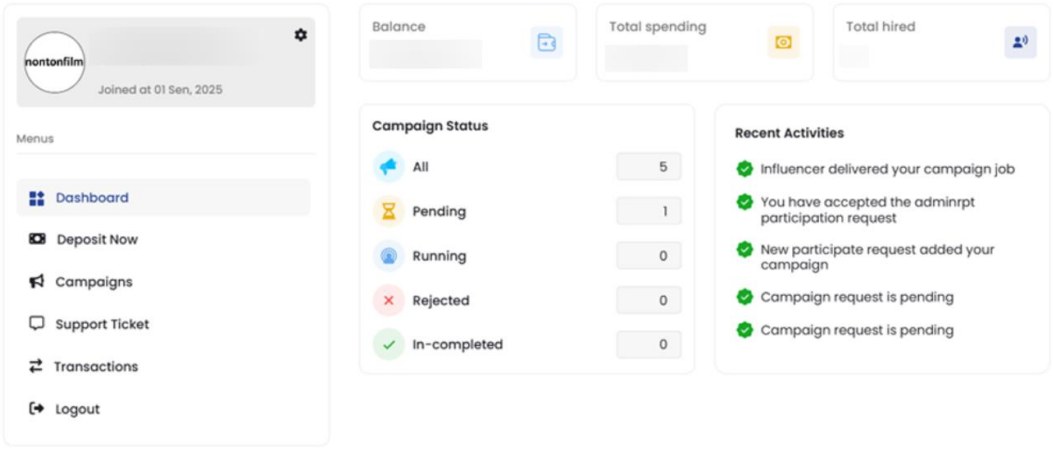
Gambar 3.13 *Page Change Password Brand*

Gambar 3.13 merupakan halaman *Change Password* yang menyediakan kemampuan bagi pengguna untuk memperbarui kredensial akun mereka. Halaman ini mencakup kolom untuk memasukkan kata sandi saat ini, kata sandi baru yang diinginkan, dan konfirmasi kata sandi baru untuk memastikan keakuratan dan keamanan. Fitur ini membantu menjaga keamanan akun dengan memungkinkan pengguna memperbarui detail *login*.



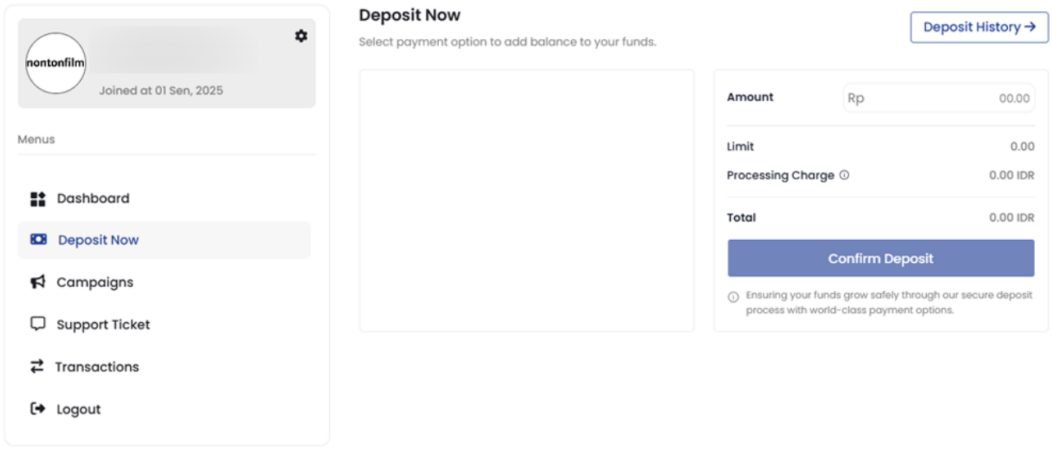
Gambar 3.14 *Page Profile Settings Brand*

Gambar 3.14 merupakan halaman Profil Brand memungkinkan pengguna *brand* untuk mengelola dan memperbarui informasi terkait perusahaan mereka di platform. Halaman ini mencakup bagian untuk mengunggah logo *brand*, membantu mempersonalisasi dan merepresentasikan identitas *brand* secara visual. Selain itu, halaman ini menyediakan kolom *input* untuk memasukkan detail penting seperti nama depan, nama belakang, dan nama pengguna perwakilan *brand*, serta informasi kontak seperti alamat email. Brand juga dapat mengisi informasi khusus bisnis termasuk nama *brand* dan situs web. Untuk memastikan identifikasi lokasi yang tepat, kolom untuk negara bagian, kode pos, kota, dan alamat lengkap juga disertakan. Pengaturan profil ini memastikan platform menyimpan data *brand* yang akurat dan lengkap untuk tujuan manajemen kampanye dan komunikasi.



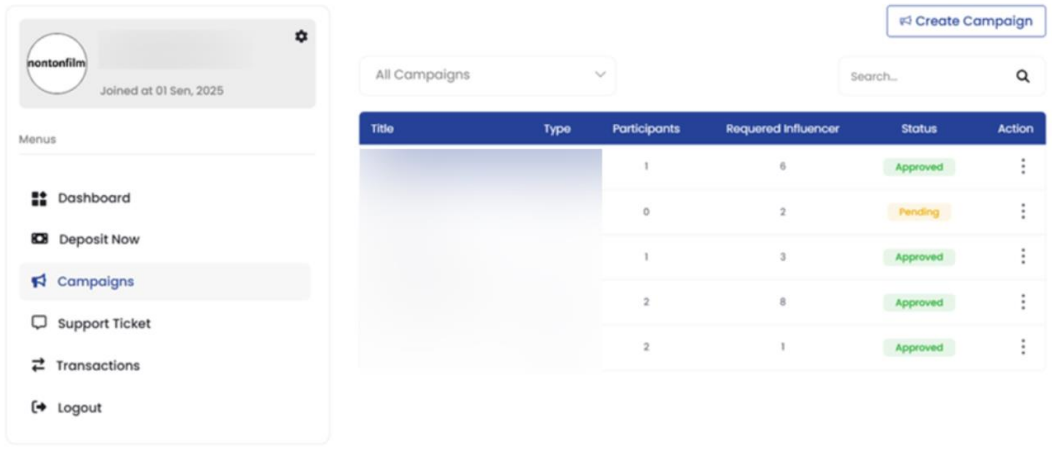
Gambar 3.15 *Page Dasbor Brand*

Gambar 3.15 merupakan dasbor *brand* memberikan gambaran umum tentang aktivitas akun merek dan kinerja kampanye. Dasbor ini menampilkan saldo akun saat ini, jumlah total yang dibelanjakan, dan jumlah total *influencer* atau komentator yang dipekerjakan. Dasbor juga menyertakan bagian Status Kampanye, yang merangkum jumlah total kampanye dan mengkategorikannya berdasarkan status seperti Semua, Menunggu, Berjalan, Ditolak, dan Dalam penyelesaian. Selain itu, *feed* Aktivitas Terbaru menyoroti interaksi dan pembaruan terbaru terkait kemajuan kampanye, termasuk tindakan *influencer*, persetujuan partisipasi, dan permintaan kampanye yang tertunda. Dasbor ini membantu merek memantau alur kerja kampanye dan metrik keuangan mereka dalam satu tampilan.



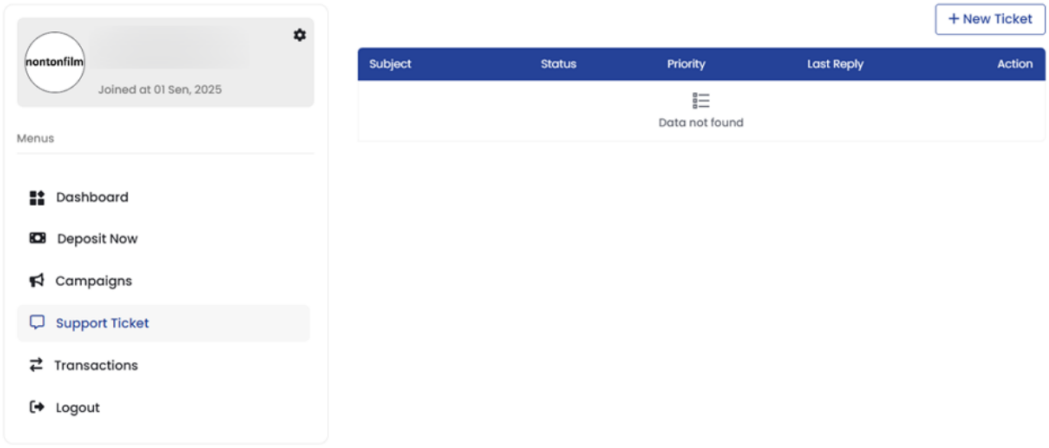
Gambar 3.16 *Page Deposit Now Brand*

Gambar 3.16 merupakan halaman *Deposit Now* menyediakan antarmuka bagi pengguna untuk menambah saldo akun mereka. Halaman ini mencakup bagian untuk memilih opsi pembayaran dan menampilkan rincian keuangan yang dihitung seperti jumlah deposit, limit, biaya pemrosesan, dan total yang harus dibayar. Halaman ini menekankan keamanan dan keandalan proses deposit, meyakinkan pengguna bahwa dana mereka ditangani dengan aman melalui metode pembayaran terpercaya.



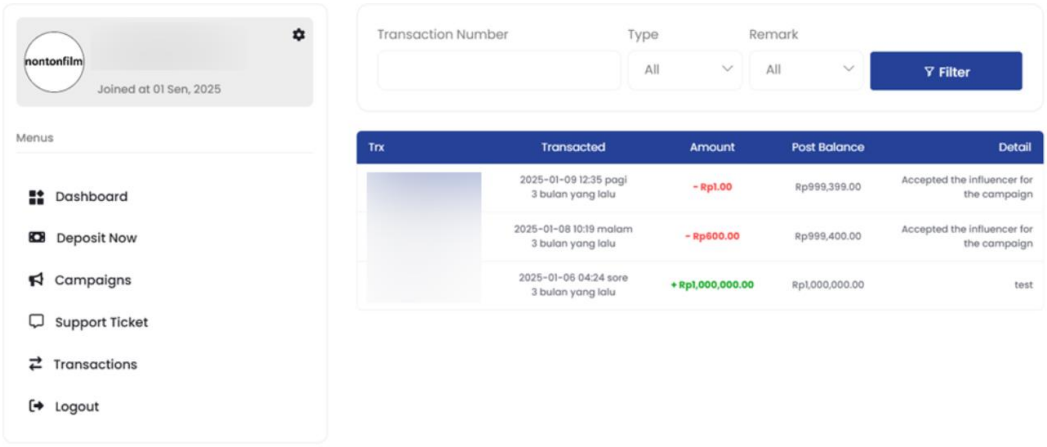
Gambar 3.17 *Page Campaigns Brand*

Gambar 3.17 merupakan halaman *Campaign* menampilkan daftar kampanye dengan detail penting seperti judul kampanye, jenis, jumlah peserta, jumlah *influencer* yang diperlukan, dan status saat ini dari setiap kampanye. Pengguna juga dapat mengambil tindakan pada setiap entri kampanye, sehingga mereka dapat mengelola atau memantau kemajuan secara efektif. Halaman ini mendukung pencarian cepat dan menyediakan tampilan terpusat untuk mengawasi aktivitas kampanye.



Gambar 3.18 *Page Support Ticket Brand*

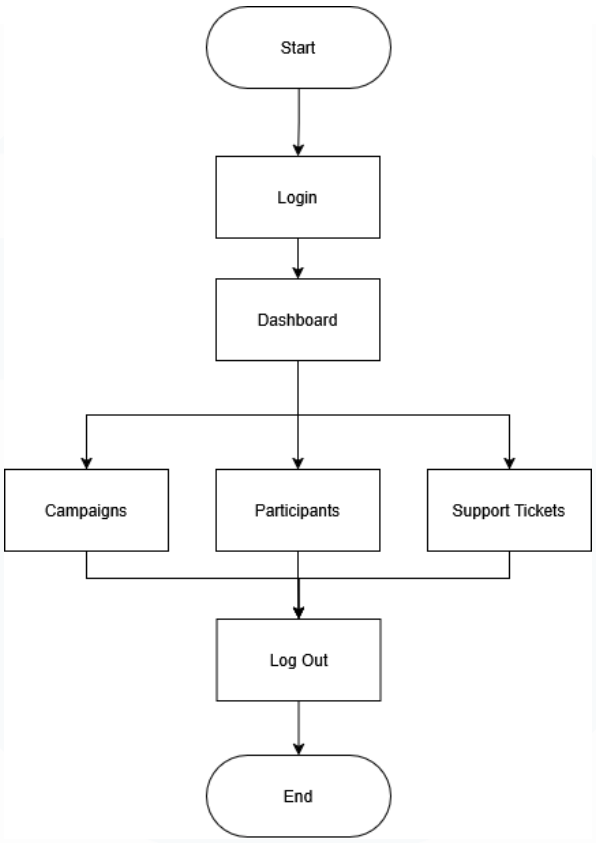
Gambar 3.18 pada halaman *Support Ticket* memberikan gambaran umum tentang semua permintaan dukungan yang diajukan. Halaman ini mencakup kolom untuk subjek tiket, status saat ini, tingkat prioritas, dan *timestamp* balasan terakhir. Terdapat juga kolom tindakan untuk mengelola setiap tiket. Selain itu, pengguna dapat membuat tiket baru menggunakan tombol "*New Ticket*" untuk meminta bantuan atau melaporkan masalah.



Gambar 3.19 *Page Transaction brand*

Gambar 3.19 merupakan halaman *Transactions* menampilkan daftar aktivitas keuangan yang terkait dengan akun. Setiap entri mencakup nomor transaksi, tanggal dan waktu transaksi, waktu relatif sejak transaksi terjadi, jumlah transaksi, saldo pasca-transaksi yang dihasilkan, dan komentar atau deskripsi singkat tentang transaksi, seperti tindakan yang berhubungan dengan kampanye. Halaman ini memberikan pengguna pandangan yang jelas tentang riwayat keuangan dan aktivitas akun mereka.

3.2.2.3 Admin



Gambar 3.20 *Flow* Admin

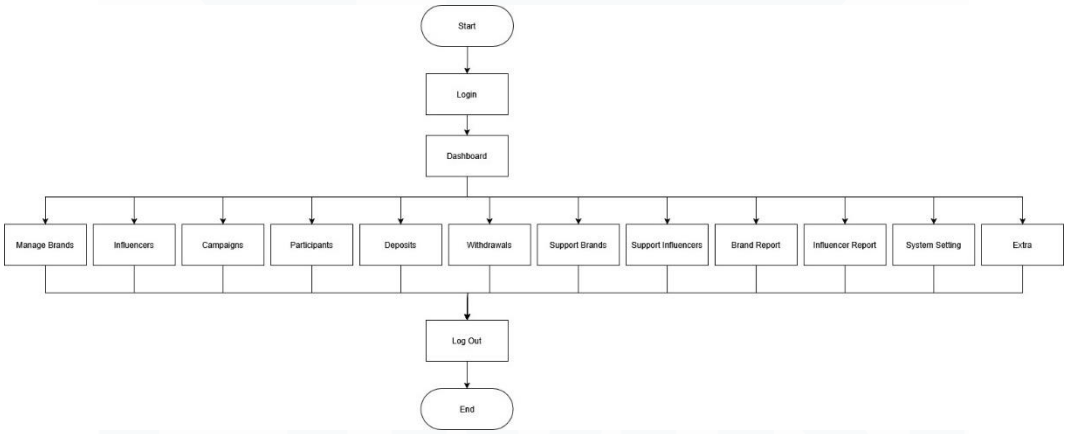
Gambar 3.20 merupakan alur admin yang dimulai dengan proses *login*, di mana administrator mengakses sistem untuk mengelola operasi. Setelah masuk, admin diarahkan ke dasbor, yang memberikan gambaran umum tentang metrik utama platform, seperti jumlah merek yang telah membuat kampanye, jumlah *influencer* yang telah mengirimkan pekerjaan mereka, dan jumlah tiket dukungan yang diajukan oleh pengguna. Dari dasbor, admin dapat menavigasi ke tiga bagian utama: *Campaigns, Participants,* dan *Support Tickets*.

Pada bagian *Campaigns*, admin dapat melihat daftar lengkap *campaign* yang dibuat oleh *brand*. Ini termasuk kemampuan untuk menyetujui atau menolak *campaign* dan mengecek hasil kerjaan *influencer*. Pengajuan *influencer* harus ditinjau dan disetujui oleh admin untuk memastikan mereka memenuhi syarat untuk menerima *reward*.

Bagian Peserta memungkinkan admin untuk melacak status partisipasi kampanye, apakah mereka diterima, selesai, dibatalkan, ditolak, atau masih tertunda. Hal ini membantu menjaga transparansi dan pelacakan kemajuan di semua kampanye yang aktif.

Bagian *Support Tickets* digunakan untuk mengelola komunikasi mengenai masalah apa pun yang dialami oleh *brand* atau komentator, seperti penundaan tinjauan atau masalah terkait tugas. Bagian ini juga berfungsi sebagai saluran bagi admin untuk menyampaikan masalah kepada super admin jika diperlukan. Setelah mengelola semua tugas yang diperlukan, admin dapat keluar, yang menandai akhir sesi alur kerja.

3.2.2.4 Super Admin



Gambar 3.21 *Flow Super* Admin

Gambar 3.21 merupakan dasbor super admin yang berfungsi sebagai pusat pusat untuk mengelola semua komponen platform *influencer marketing*. Dasbor ini diatur ke dalam beberapa bagian fungsional yang memungkinkan administrator untuk mengontrol pengguna, memantau transaksi keuangan, mengonfigurasi pengaturan sistem, dan menangani operasi dukungan dengan presisi.

Bagian manajemen *brand* bertanggung jawab untuk mengawasi semua *brand* yang terdaftar di platform. Bagian ini mengklasifikasikan

brand berdasarkan status mereka saat ini. *Brand* yang aktif adalah *brand* yang terverifikasi, sedangkan *brand* yang dilarang adalah *brand* yang telah diblokir karena pelanggaran kebijakan atau keputusan administratif. Kategori tambahan termasuk *brand* yang belum menyelesaikan verifikasi melalui email atau ponsel, yang akhirnya belum bisa membuat kampanye. Verifikasi KYC (*Know Your Customer*) sangat penting, dan *brand* ditandai sebagai belum diverifikasi atau tertunda berdasarkan status pengiriman dokumen mereka. *Brand*, apa pun statusnya, terdaftar dalam tampilan "Semua *Brand*" yang komprehensif. Selain itu, platform ini memungkinkan administrator untuk berkomunikasi langsung dengan *brand* melalui sistem notifikasi khusus.

Manajemen *influencer* merefleksikan struktur pengawasan merek. *Influencer* dikategorikan sebagai aktif atau dilarang tergantung pada platform mereka. Email atau nomor ponsel yang belum diverifikasi disorot untuk pemantauan keamanan. Status KYC dilacak dengan cara yang sama, dengan kategori untuk pengajuan yang belum diverifikasi dan tertunda. Daftar utama semua *influencer* tersedia untuk pengawasan umum, dan fungsionalitas pemberitahuan langsung memastikan bahwa administrator dapat secara efisien menyiarkan pembaruan penting ke jaringan *influencer*.

Manajemen kampanye menyediakan antarmuka yang ramping untuk memantau siklus hidup kampanye pemasaran. Kampanye dimulai dalam status tertunda, menunggu peninjauan dan persetujuan oleh administrator. Setelah disetujui, kampanye menjadi aktif dan dapat dilihat oleh *influencer*. Kampanye yang ditolak ditandai dengan alasan ketidakpatuhan. Administrator dapat mengakses pandangan menyeluruh dari semua kampanye yang diluncurkan di platform, terlepas dari statusnya, memberikan kejelasan dan wawasan historis tentang kinerja kampanye.

Pemantauan peserta memungkinkan administrator untuk melacak partisipasi *influencer* dalam kampanye. Status peserta termasuk tertunda (menunggu peninjauan atau pengiriman konten), diterima (dipilih untuk kampanye), terkirim (konten telah dikirim), dan selesai (sudah menyelesaikan kampanye). Laporan dan bendera dikelola melalui bagian "Dilaporkan", sementara entri yang dibatalkan atau ditolak mengidentifikasi peserta atau kampanye yang telah dihentikan atau ditolak sebelum waktunya. Tampilan konsolidasi dari semua aktivitas peserta tersedia untuk analisis mendalam.

Transaksi keuangan dikelola melalui modul deposit dan penarikan. *Interface* deposit melacak kontribusi keuangan yang dilakukan pengguna ke platform. Transaksi dikelompokkan ke dalam status tertunda, disetujui, berhasil, ditolak, dan dimulai. Perincian ini memastikan pemantauan yang tepat terhadap arus masuk keuangan. Ringkasan dasbor menyajikan metrik utama seperti jumlah total deposit dan biaya yang dikeluarkan. Pada modul penarikan, dana yang keluar dari sistem juga dipantau dengan cara yang sama di seluruh status tertunda, disetujui, ditolak, dan status aktivitas secara keseluruhan. Sistem berstruktur ganda ini memastikan transparansi dan tata kelola keuangan yang ketat. Angka-angka penting, seperti total dana yang ditarik dan biaya penarikan yang berlaku, ditampilkan dengan jelas untuk tinjauan administratif.

Fungsionalitas dukungan dibagi menjadi dua modul paralel yaitu, Bantuan untuk Brand dan Bantuan untuk *Influencer*. Setiap modul mencakup status tiket seperti tertunda (menunggu tanggapan), ditutup (diselesaikan), dijawab (ditangani secara aktif), dan tampilan komprehensif dari semua tiket dukungan. Sistem ini memastikan bahwa merek dan *influencer* dapat mengakses dukungan tepat waktu dan administrator dapat memantau dan menyelesaikan masalah secara efisien.

Modul pelaporan menyediakan data historis untuk *brand* dan *influencer*. Laporan merek mencakup riwayat transaksi, riwayat *login*, dan riwayat notifikasi, yang menawarkan wawasan tentang perilaku pengguna, interaksi keuangan, dan komunikasi sistem. Laporan *influencer* mencerminkan elemen-elemen ini, memungkinkan transparansi dan akuntabilitas yang sama di kedua kelompok pengguna.

Pengaturan sistem menyediakan lingkungan yang sangat mudah dikonfigurasi untuk mengelola seluruh platform. Pengaturan umum dan khusus platform mengatur operasi inti. Administrator dapat mengunggah aset *branding* seperti logo platform dan favicon, mengkonfigurasi sistem notifikasi, mengintegrasikan *gateway* pembayaran, dan menentukan metode penarikan. Pengaturan SEO seperti meta *tag*, *sitemap* XML, dan robots.txt memastikan penemuan mesin pencari. Fitur manajemen konten termasuk mengelola halaman dan tata letak *frontend*. Pengaturan KYC memungkinkan penyesuaian bidang verifikasi identitas untuk merek dan *influencer*. Platform ini mendukung integrasi dengan layanan eksternal melalui kredensial *login* sosial, manajemen program *referral*, dan dukungan multi-bahasa. Kepatuhan kebijakan ditangani melalui *banner cookie* GDPR yang dapat dikonfigurasi dan halaman kebijakan hukum. Untuk kustomisasi yang lebih dalam, tersedia pengaturan CSS khusus. Mode pemeliharaan dapat diaktifkan untuk menonaktifkan sementara platform untuk peningkatan atau perubahan penting.

Modul "Ekstra" khusus memberikan akses kepada administrator ke fungsi *back-end*. Tab aplikasi memberikan wawasan tentang operasi di seluruh platform. Konfigurasi dan kesehatan server dapat dipantau secara langsung. Alat manajemen *cache* memungkinkan administrator untuk menghapus data sementara, mengoptimalkan kinerja sistem. Pembaruan perangkat lunak dikelola di sini, memastikan bahwa platform tetap aman dan mutakhir.

Ringkasan dasbor di tingkat teratas mengkonsolidasikan statistik dan wawasan utama. Ini menampilkan statistik pengguna dan *influencer*, termasuk total, pengguna aktif, dan status verifikasi. Metrik kampanye mengungkapkan distribusi kampanye berdasarkan status. Ringkasan keuangan menunjukkan total jumlah yang disetorkan dan ditarik beserta biaya terkait. Sistem ini juga mencakup laporan grafis dan tabel yang disaring berdasarkan rentang tanggal, termasuk log transaksi dan catatan *login*. Analisis *login* memberikan wawasan tentang perangkat dan wilayah yang digunakan untuk mengakses platform, yang dikategorikan berdasarkan browser, sistem operasi, dan negara. Visualisasi ini memberikan pemahaman waktu nyata kepada administrator tentang perilaku pengguna dan tren akses sistem.

Interface administratif yang komprehensif ini memastikan bahwa setiap aspek dari platform, mulai dari manajemen pengguna dan pengawasan kampanye hingga transaksi keuangan dan kepatuhan-dikendalikan, dipantau, dan dioptimalkan untuk kesempurnaan operasional.

3.2.3 Membuat Konsep Automasi Data *Scraping* di Jupyter

Dalam mendukung efisiensi proses pengumpulan data dari berbagai platform media sosial, konsep otomasi *scraping* dirancang dengan memanfaatkan bahasa pemrograman Python yang dijalankan melalui Jupyter Notebook. Tujuan utama dari konsep ini adalah mempermudah proses pencarian data secara spesifik, terutama data dari komentar pengguna pada platform seperti Twitter, TikTok, Instagram, Facebook, dan YouTube.

Pendekatan yang digunakan dalam konsep ini adalah dengan memanfaatkan sistem "*tabbing*", yaitu teknik otomatisasi yang meniru perilaku menekan tombol Tab secara berulang untuk menavigasi elemen-elemen dalam antarmuka pengguna sebuah situs atau aplikasi. Melalui pendekatan ini, program akan secara otomatis berpindah dari satu elemen ke elemen lain,

khususnya pada kolom komentar sambil melakukan pencarian terhadap nama pengguna atau kata kunci tertentu yang telah ditentukan sebelumnya.

Setelah sistem berhasil menemukan target (nama pengguna yang dicari), maka sistem akan secara otomatis mengambil tangkapan layar (*screenshot*) pada bagian tersebut sebagai bentuk dokumentasi visual dari hasil pencarian. Proses ini akan terus berlanjut hingga seluruh target yang telah ditentukan ditemukan atau hingga seluruh komentar selesai diproses.

Konsep ini diuji dan dikembangkan untuk masing-masing platform media sosial, mengingat setiap platform memiliki struktur antarmuka yang berbeda. Oleh karena itu, pendekatan otomasi ini memerlukan penyesuaian khusus pada setiap platform agar proses *scraping* dapat berjalan dengan lancar dan tidak terganggu oleh perubahan *layout* atau kebijakan keamanan situs.

Memanfaatkan Jupyter Notebook, proses pengujian, *debugging*, dan dokumentasi berjalan lebih mudah dan terstruktur. Selain itu, Jupyter memungkinkan pemantauan hasil secara langsung sehingga mempermudah pengambilan keputusan terkait pengembangan lebih lanjut. Berikut merupakan penjelasan lebih lanjutnya:

3.2.3.1 X (Twitter)

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import undetected_chromedriver as uc
from IPython.display import display, Image

# Set your X (Twitter) login credentials
EMAIL = "EMAIL"
PASSWORD = "PASSWORD."

# Start undetectable Chrome
options = webdriver.ChromeOptions()
options.add_argument("--disable-blink-features=AutomationControlled")  # Helps avoid detection
driver = uc.Chrome(options=options, headless=False)

# Open Twitter login page
driver.get("https://x.com/login")
time.sleep(5)

# Function to log in
def login_x():
    try:
        wait = WebDriverWait(driver, 10)

        # Locate and enter email
        email_input = wait.until(EC.presence_of_element_located((By.XPATH, "//input[@autocomplete='username']")))
        email_input.send_keys(EMAIL)
        email_input.send_keys(Keys.ENTER)
        time.sleep(3)

        # Check if Twitter asks for extra verification (e.g., phone number)
        try:
            verification_input = driver.find_element(By.XPATH, "//input[@type='text']")
            print("⚠️ Twitter requested extra verification. Please check your phone/email.")
            verification_code = input("Enter the verification code: ")
            verification_input.send_keys(verification_code)
            verification_input.send_keys(Keys.ENTER)
            time.sleep(3)
        except Exception:
            pass  # No extra verification needed

        # Locate and enter password
        password_input = wait.until(EC.presence_of_element_located((By.XPATH, "//input[@autocomplete='current-password']")))
        password_input.send_keys(PASSWORD)
        password_input.send_keys(Keys.ENTER)
        time.sleep(5)

        print("✅ Login successful!")
    except Exception as e:
        print(f"❌ Login failed: {e}")

# Perform Login
login_x()
```

Gambar 3.22 *Notebook* X *Login*

Gambar 3.22 merupakan gambar skrip yang dirancang untuk mengotomatiskan proses masuk ke platform media sosial X (Twitter) menggunakan Selenium dan modul undetected_chromedriver untuk mengurangi kemungkinan ditandai sebagai bot. Otomatisasi dimulai dengan menyiapkan browser Chrome yang menyembunyikan fitur kontrol otomatis, sehingga membuatnya terlihat seperti browser yang dioperasikan oleh manusia. Cara ini menggunakan opsi Chrome khusus untuk mengurangi kemungkinan terdeteksi oleh sistem anti-bot.

Setelah browser diluncurkan, browser akan menavigasi ke halaman *login* X dan menunggu hingga halaman dimuat. Skrip kemudian mencoba mencari kolom *input* untuk alamat email pengguna dan memasukkan kredensial email yang disediakan. Setelah mengirimkan email, skrip ini akan memeriksa apakah X meminta langkah verifikasi tambahan-seperti memasukkan kode verifikasi yang dikirim ke ponsel atau email pengguna. Jika langkah seperti itu diperlukan, skrip akan berhenti sejenak dan meminta pengguna untuk memasukkan kode secara manual, yang kemudian dikirimkan secara otomatis.

Setelah verifikasi email, skrip melanjutkan untuk mencari kolom *input* kata sandi, memasukkan kata sandi pengguna, dan mengirimkannya untuk menyelesaikan proses *login*. Selama proses berlangsung, pesan ditampilkan pada konsol untuk memberi tahu pengguna apakah *login* berhasil atau ada kesalahan yang terjadi. Penyiapan ini sangat berguna untuk tugas-tugas seperti pengumpulan data otomatis, memposting konten, atau memantau aktivitas di X, sambil mempertahankan ketahanan terhadap mekanisme deteksi *login* otomatis.

```python
# Open the tweet page
tweet_url = "INPUT URL"
driver.get(tweet_url)
time.sleep(5)

# Define a list of search terms (usernames or keywords) to look for in the replies
search_terms = ["INPUT NAME"]

# Scroll to load as many replies as possible.
prev_count = 0
scroll_attempts = 0
max_scroll_attempts = 10  # Increase if necessary

while scroll_attempts < max_scroll_attempts:
    driver.find_element(By.TAG_NAME, "body").send_keys(Keys.PAGE_DOWN)
    time.sleep(2)

    # Click "Show more replies" if available
    try:
        more_replies = driver.find_element(By.XPATH, "//span[text()='Show more replies']")
        more_replies.click()
        time.sleep(2)
    except Exception:
        pass  # Button not available

    articles = driver.find_elements(By.XPATH, "//article[@data-testid='tweet']")
    if len(articles) == prev_count:
        scroll_attempts += 1
    else:
        scroll_attempts = 0
        prev_count = len(articles)

print("Total tweets loaded:", len(articles))

# Iterate over each tweet and screenshot if it contains any search term
screenshot_files = []
for idx, article in enumerate(articles):
    try:
        tweet_text = article.text.lower()
    except Exception as e:
        print(f"Skipping tweet {idx} due to error: {e}")
        continue

    for term in search_terms:
        if term.lower() in tweet_text:
            # Scroll the tweet into view to ensure proper screenshot
            driver.execute_script("arguments[0].scrollIntoView(true);", article)
            time.sleep(1)  # Allow time for the element to settle in view
            filename = f"screenshot_{term}_{idx}.png"
            try:
                article.screenshot(filename)
                screenshot_files.append(filename)
                print(f"Found '{term}' in tweet {idx}. Screenshot saved as {filename}.")
            except Exception as e:
                print(f"Error taking screenshot for tweet {idx}: {e}")
            break  # Stop checking other terms for this tweet

# Display screenshots in notebook
for file in screenshot_files:
    display(Image(filename=file))

# Close browser
driver.quit()
```

Gambar 3.23 *Notebook* X *Scraping*

Gambar 3.23 merupakan skrip yang mengotomatiskan proses pemindaian melalui *reply* pada tweet tertentu di platform X (Twitter) untuk menemukan dan menangkap tweet yang mengandung nama pengguna atau kata kunci tertentu. Skrip ini menggunakan Selenium dengan driver Chrome yang tidak terdeteksi untuk mengurangi kemungkinan ditandai sebagai bot. Otomatisasi dimulai dengan menavigasi ke URL tweet yang diberikan. Setelah berada di halaman tersebut, robot siap untuk mencari kata kunci tertentu dengan terlebih

dahulu menentukan daftar nama atau kata kunci target. Ini bisa berupa nama pengguna, frasa, atau kata kunci apa pun yang mungkin muncul dalam tweet.

Memastikan sebanyak mungkin *reply* dimuat, skrip menirukan perilaku pengguna yang melakukan *scroll* ke bawah halaman secara berulang-ulang. Saat *scroll*, skrip ini juga mencoba mengklik tombol "Tampilkan lebih banyak balasan" setiap kali muncul, yang memastikan bahwa balasan yang tersembunyi atau diciutkan akan diperluas dan ditampilkan. Perulangan *scroll* dan *load* ini terus berlanjut hingga balasan berhenti bertambah.

Setelah skrip mengumpulkan semua *reply* yang terlihat di halaman, skrip akan menganalisis konten dari masing-masing *reply*. Untuk setiap balasan, skrip ini memeriksa apakah ada kata kunci pencarian yang telah ditentukan sebelumnya yang muncul di dalam teks. Jika ditemukan kecocokan, skrip ini akan menampilkan tweet tersebut dengan men-scrollnya. Hal ini memastikan elemen tersebut ditampilkan secara penuh dan benar di browser sebelum mengambil *screenshot*. Setelah tweet terlihat, skrip mengambil *screenshot* dari *reply* tersebut dan menyimpannya dengan nama *file* yang menyertakan istilah yang cocok dan indeks tweet dalam daftar. Kemudian menyimpan nama file dari semua *screenshot* yang berhasil diambil.

Terakhir, menampilkan setiap *screenshot* yang diambil dan kemudian menutup sesi browser untuk mengakhiri otomatisasi. Pendekatan ini berguna untuk mengumpulkan bukti visual dari interaksi, atau konten tertentu pada komentar, *mention*, atau *reply* publik untuk tujuan pengarsipan, moderasi, atau penelitian.

3.2.3.2 TikTok

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import undetected_chromedriver as uc
from IPython.display import display, Image
import time

# Start undetected Chrome
options = webdriver.ChromeOptions()
options.add_argument("--disable-blink-features=AutomationControlled")
driver = uc.Chrome(options=options, headless=False)

# Open TikTok video page
video_url = "INPUT URL"
driver.get(video_url)
time.sleep(5)

search_name = "Target username"
found = False

# Function to get the currently focused element
def get_focused_element():
    return driver.execute_script("return document.activeElement;")

tab_count = 0
safety_limit = 500  # Prevent infinite loops

while not found and tab_count < safety_limit:
    driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
    time.sleep(0.1)
    tab_count += 1

    focused_element = get_focused_element()
    if focused_element:
        highlighted_text = focused_element.text.strip()
        print(f"🔍 Highlighted: {highlighted_text}")
        if search_name == highlighted_text:
            print(f"\n✅ Found '{search_name}'! Taking screenshot...\n")
            found = True
            break

if tab_count >= safety_limit:
    print("⚠️ Safety limit reached. Username may not have been found.")

# Take screenshot and quit
screenshot_path = "tiktok_screenshot.png"
driver.save_screenshot(screenshot_path)
driver.quit()

# Display screenshot in the notebook
display(Image(filename=screenshot_path))
```

Gambar 3.24 *Notebook* TikTok *Scraping*

Gambar 3.24 merupakan skrip yang mengotomatiskan tugas untuk menemukan nama pengguna atau elemen teks tertentu pada halaman video TikTok menggunakan Selenium dengan driver Chrome yang tidak terdeteksi. Tujuannya adalah untuk menavigasi halaman menggunakan simulasi *keyboard* dan menangkap tangkapan layar setelah target ditemukan. Prosesnya dimulai dengan meluncurkan peramban Chrome yang dikonfigurasikan untuk mengurangi kemungkinan terdeteksi sebagai bot otomatis. Kemudian menavigasi ke halaman video TikTok yang diberikan. Jeda singkat diberikan agar halaman dimuat sepenuhnya sebelum interaksi dimulai.

Nama pengguna tertentu ditetapkan sebagai target yang akan dicari di halaman tersebut. Untuk menemukannya, skrip meniru penekanan tombol TAB berulang kali. Tindakan ini berputar melalui semua elemen yang dapat difokuskan pada halaman seperti tautan, tombol, dan nama profil, membuat setiap elemen menjadi fokus satu per satu. Setiap kali sebuah elemen difokuskan, teks yang terlihat akan diambil dan diperiksa terhadap nama target.

Loop ini terus berlanjut sampai teks yang diinginkan ditemukan, atau batas yang sudah ditentukan sebelumnya pada jumlah penekanan TAB tercapai. Batas ini berfungsi sebagai pengaman terhadap perulangan tak terbatas jika target tidak ada pada halaman. Jika teks ditemukan, skrip akan mengonfirmasi kecocokan dan melanjutkan untuk mengambil *screenshot* dari seluruh jendela browser yang terlihat, menangkap target dalam konteksnya.

Berhasil atau tidak, sesi browser akan ditutup pada akhirnya. *Screenshot* yang disimpan kemudian ditampilkan di dalam buku catatan atau lingkungan untuk pemeriksaan atau pencatatan lebih lanjut. Metode ini berguna untuk menangkap elemen di *interface* di mana pemilih elemen tradisional mungkin tidak dapat diandalkan, terutama di halaman yang dinamis atau kompleks secara visual seperti yang ada di TikTok.

3.2.3.3 Instagram

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import undetected_chromedriver as uc
from IPython.display import display, Image

# Set Instagram credentials
IG_EMAIL = "email/username"
IG_PASSWORD = "password"

# Set the Instagram post URL and target username
POST_URL = "INPUT URL"
TARGET_USERNAME = "INPUT TARGET"

# Start undetectable Chrome
options = webdriver.ChromeOptions()
options.add_argument("--disable-blink-features=AutomationControlled")  # Avoid detection
driver = uc.Chrome(options=options, headless=False)

# Open Instagram login page
driver.get("https://www.instagram.com/accounts/login/")
time.sleep(5)

# Function to log in
def login_instagram():
    try:
        wait = WebDriverWait(driver, 10)

        # Enter email
        email_input = wait.until(EC.presence_of_element_located((By.NAME, "username")))
        email_input.send_keys(IG_EMAIL)

        # Enter password
        password_input = driver.find_element(By.NAME, "password")
        password_input.send_keys(IG_PASSWORD)
        password_input.send_keys(Keys.ENTER)
        time.sleep(5)

        # Handle "Save Login Info" popup (optional)
        try:
            not_now_button = wait.until(EC.presence_of_element_located((By.XPATH, "//button[text()='Not Now']")))
            not_now_button.click()
            time.sleep(3)
        except:
            pass

        print("✅ Login successful!")
    except Exception as e:
        print(f"❌ Login failed: {e}")

# Perform Login
login_instagram()
```

Gambar 3.25 *Notebook* Instagram *Login*

Gambar 3.25 merupakan skrip yang mengotomatiskan proses *login* ke Instagram menggunakan Selenium dengan driver Chrome yang tidak terdeteksi, yang membantu melakukan *bypass* mekanisme deteksi yang digunakan situs web untuk memblokir bot. Tujuan dari skrip ini adalah untuk membuka Instagram, memasukkan kredensial *login*, dan mengakses postingan tertentu untuk otomatisasi lebih lanjut (seperti mencari nama pengguna di komentar). Prosesnya dimulai dengan meluncurkan browser Chrome yang menghindari deteksi oleh sistem deteksi bot Instagram. Browser ini menavigasi langsung ke halaman

login Instagram dan berhenti sebentar untuk memastikan halaman tersebut dimuat sepenuhnya.

Setelah halaman siap, skrip berinteraksi dengan formulir *login* dengan secara otomatis mengisi nama pengguna (atau email) dan kata sandi yang ditentukan. Setelah memasukkan kredensial, skrip akan mengirimkan formulir untuk memulai *login*. Setelah upaya *login* berhasil, skrip menyertakan pemeriksaan bawaan untuk menangani *popup* umum yang menanyakan apakah akan menyimpan informasi *login*. Jika *popup* muncul maka akan ditutup, sehingga skrip dapat dilanjutkan dengan lancar tanpa gangguan.

Hasil dari proses *login* baik berhasil atau tidak akan dilaporkan di konsol. Hal ini menyiapkan skrip untuk melakukan tugas-tugas lebih lanjut, seperti menavigasi ke postingan Instagram tertentu dan berinteraksi dengannya, yang biasanya akan dilakukan setelah rutinitas *login* ini. Penyiapan ini sering digunakan dalam tugas otomatisasi media sosial seperti pemantauan konten, analisis keterlibatan, atau deteksi pengguna.

```python
# Open the Instagram post
driver.get(POST_URL)
time.sleep(5)

# Function to extract usernames and comments
def get_comments():
    try:
        usernames = driver.find_elements(By.XPATH, "//h3[@class='_a9zc']/span/a")
        comments = driver.find_elements(By.XPATH, "//div[@class='_a9zr']/span")

        user_comment_list = []
        for user, comment in zip(usernames, comments):
            user_comment_list.append((user.text, comment.text))
        return user_comment_list
    except:
        return []

# Function to get the currently highlighted element
def get_focused_element():
    return driver.execute_script("return document.activeElement;")

found = False
tab_count = 0

while not found:
    driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
    time.sleep(0.5)
    tab_count += 1

    # Get the currently focused element
    focused_element = get_focused_element()
    if focused_element:
        highlighted_text = focused_element.text.strip()
        print(f"🔍 Highlighted: {highlighted_text}")

        # Check if the highlighted text contains the target username
        if TARGET_USERNAME.lower() in highlighted_text.lower():
            print(f"\n✅ Found '{TARGET_USERNAME}'! Stopping tabbing...\n")
            found = True
            break

    # Click "Load more comments" button if it appears
    try:
        load_more = driver.find_element(By.XPATH, "//svg[@aria-label='Load more comments']")
        driver.execute_script("arguments[0].click();", load_more)
        print("🔄 Clicked 'Load more comments'")
        time.sleep(2)
    except:
        pass

    # Prevent infinite loop (optional safety limit)
    if tab_count > 500:
        print("⚠️ Stopping after 500 tabs to prevent infinite loop.")
        break

# Screenshot and display
screenshot_path = "instagram_screenshot.png"
driver.save_screenshot(screenshot_path)
driver.quit()

# Display screenshot in Jupyter Notebook
display(Image(filename=screenshot_path))
```

Gambar 3.26 *Notebook* Instagram *Scraping*

Gambar 3.26 merupakan skrip yang dirancang untuk menavigasi ke postingan Instagram tertentu dan mengidentifikasi apakah nama pengguna target muncul di komentar. Skrip ini menggunakan kontrol browser otomatis untuk melakukan tugas ini, mensimulasikan perilaku pengguna yang sebenarnya. Setelah masuk ke Instagram dan membuka postingan yang diinginkan, skrip mencoba mengekstrak semua nama pengguna yang terlihat dan komentar terkait dari postingan tersebut. Hal

ini dilakukan dengan menggunakan pemilihan elemen HTML, meskipun ekstraksi ini bukanlah fokus utama di sini.

Mendeteksi keberadaan nama pengguna target, skrip mensimulasikan penekanan tombol TAB berulang kali. Setiap penekanan tombol TAB akan memindahkan fokus ke seluruh elemen interaktif pada halaman. Setelah setiap penekanan tombol, skrip akan memeriksa elemen apa yang sedang difokuskan dan membaca teksnya. Jika teks ini menyertakan nama pengguna target, skrip akan menyimpulkan bahwa pengguna tersebut telah ditemukan dan menghentikan perulangan.

Instagram sering memuat komentar secara berkelompok, skrip juga mencoba mengklik tombol "*load more comments*" jika muncul, sehingga komentar tambahan dapat dipindai. Hal ini meningkatkan peluang untuk menemukan nama pengguna yang diinginkan. Dalam mencegah perulangan berjalan tanpa henti apabila nama pengguna tidak ditemukan, terdapat batas keamanan bawaan pada jumlah penekanan TAB.

Setelah proses selesai apakah nama pengguna ditemukan atau tidak, skrip mengambil *screenshot* dari kondisi halaman. Kemudian menutup browser dan menampilkan *screenshot* secara langsung di *notebook*, berguna untuk konfirmasi visual atau dokumentasi.

3.2.3.4 Facebook

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import undetected_chromedriver as uc
from IPython.display import display, Image
import time

# Facebook login credentials
FB_EMAIL = "EMAIL"
FB_PASSWORD = "PASSWORD."

# Start undetected Chrome
options = webdriver.ChromeOptions()
options.add_argument("--disable-blink-features=AutomationControlled")  # Helps avoid detection
driver = uc.Chrome(options=options, headless=False)

# Open Facebook login page
driver.get("https://www.facebook.com/")
time.sleep(5)  # Allow time for page to load

# Perform Login
try:
    wait = WebDriverWait(driver, 10)
    email_input = wait.until(EC.presence_of_element_located((By.ID, "email")))
    email_input.send_keys(FB_EMAIL)

    password_input = driver.find_element(By.ID, "pass")
    password_input.send_keys(FB_PASSWORD)
    password_input.send_keys(Keys.ENTER)

    time.sleep(5)
    print("✅ Login successful!")
except Exception as e:
    print(f"❌ Login failed: {e}")
```

Gambar 3.27 *Notebook* Facebook *Login*

Gambar 3.27 merupakan skrip untuk mengotomatiskan proses masuk ke Facebook menggunakan Selenium dan versi browser Chrome yang “tidak terdeteksi”, yang membantu mengurangi kemungkinan diblokir atau ditandai sebagai bot. Pertama, *script* ini menyiapkan konfigurasi peramban khusus untuk menghindari deteksi , kemudian, meluncurkan browser dan menavigasi ke beranda Facebook. Saat halaman sudah dimuat sepenuhnya, skrip menemukan bidang input untuk email dan kata sandi pengguna. Skrip ini memasukkan kredensial *login* dan mengirimkan formulir dengan mensimulasikan penekanan *ENTER* pada *keyboard*. Saat formulir *login* sudah dikirim, ia menunggu beberapa detik untuk mengizinkan Facebook menyelesaikan proses *login*. Jika semuanya berjalan lancar, pesan sukses akan ditampilkan. Jika ada langkah yang gagal seperti jika kolom input tidak dapat ditemukan atau diisi, pesan *error* akan ditampilkan. Skrip bergantung pada kondisi menunggu untuk memastikan elemen-elemen yang ada

sebelum berinteraksi dengannya, sehingga mengurangi kemungkinan kesalahan akibat elemen yang belum dimuat.

```
# Open Facebook post page
post_url = "URL"
driver.get(post_url)
time.sleep(5)

search_name = "TARGETED USERNAME"
found = False

# Function to get the currently focused element
def get_focused_element():
    return driver.execute_script("return document.activeElement;")

tab_count = 0
safety_limit = 500

while not found and tab_count < safety_limit:
    driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
    time.sleep(0.1)
    tab_count += 1

    focused_element = get_focused_element()
    if focused_element:
        highlighted_text = focused_element.text.strip()
        print(f"🔍 Highlighted: {highlighted_text}")

        if "View more comments" in highlighted_text:
            focused_element.send_keys(Keys.ENTER)
            time.sleep(2)
            continue

        if search_name == highlighted_text:
            print(f"\n✅ Found '{search_name}'! Pressing Tab 3 more times before screenshot...\n")
            for _ in range(3):
                driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
                time.sleep(0.1)
            found = True
            break

if tab_count >= safety_limit:
    print("⚠️ Safety limit reached. Username may not have been found.")

# Take screenshot and quit
screenshot_path = "facebook_screenshot.png"
driver.save_screenshot(screenshot_path)
driver.quit()

# Display screenshot in the notebook
display(Image(filename=screenshot_path))
```

Gambar 3.28 *Notebook* Facebook *Scraping*

Gambar 3.28 merupakan gambar dengan skrip yang dirancang untuk menavigasi ke postingan Facebook tertentu dan mencari nama pengguna yang ditargetkan dalam komentar atau konten halaman menggunakan *tabbing keyboard*. Skrip ini dimulai dengan membuka URL postingan Facebook dan memberikan waktu untuk memuat halaman tersebut. Kemudian memasuki sebuah lingkaran di mana bot berulang kali menekan tombol Tab untuk memindahkan fokus browser melalui semua elemen interaktif pada halaman, seperti tombol, nama pengguna, dan komentar.

Setiap kali membuka tab ke elemen baru, bot akan memeriksa teks dari item yang sedang difokuskan. Jika menemukan tombol "Lihat lebih banyak komentar", bot akan mengaktifkannya untuk memuat konten tambahan dan melanjutkan tab. Jika menemukan nama pengguna yang ditargetkan, bot akan mensimulasikan penekanan tombol Tab beberapa kali lagi untuk mengalihkan fokus ke elemen-elemen di sekitar pengguna tersebut. Cara mencegah perulangan berjalan tanpa batas waktu, batas keamanan ditetapkan. Jika nama pengguna ditemukan, skrip akan mengambil *screenshot* dari tampilan peramban saat ini, menyimpannya, menutup browser, dan menampilkan *screenshot* di *notebook*.

3.2.3.5 YouTube

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import undetected_chromedriver as uc
from IPython.display import display, Image

# Start undetected Chrome (no login required for YouTube)
options = webdriver.ChromeOptions()
options.add_argument("--disable-blink-features=AutomationControlled")  # Helps avoid detection
driver = uc.Chrome(options=options, headless=False)

# Open YouTube video page
video_url = "VIDEO URL"  # Replace with your desired video URL
driver.get(video_url)

# Wait for the comments section to load
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.CSS_SELECTOR, "#comments"))
)

search_name = "TARGET NAME"  # Target username or text to search for
found = False

# Function to extract comment texts using JavaScript
def get_comments_via_js(driver):
    script = """
    let commentElements = document.querySelectorAll('#content-text');
    let result = [];
    commentElements.forEach(comment => result.push(comment.innerText));
    return result;
    """
    return driver.execute_script(script)

# Function to extract usernames using JavaScript
def get_usernames_via_js(driver):
    script = """
    let usernameElements = document.querySelectorAll('#author-text span');
    let result = [];
    usernameElements.forEach(user => result.push(user.innerText));
    return result;
    """
    return driver.execute_script(script)

# Infinite scrolling loop to load comments and check for the target name
while not found:
    # Scroll down a page
    driver.find_element(By.TAG_NAME, "body").send_keys(Keys.PAGE_DOWN)
    WebDriverWait(driver, 2).until(
        EC.presence_of_element_located((By.CSS_SELECTOR, "#comments"))
    )

    # Extract the current list of usernames and comment texts
    usernames = get_usernames_via_js(driver)
    comments = get_comments_via_js(driver)

    # Debug: print out all usernames and comment texts
    for user, comment in zip(usernames, comments):
        print(f"User: {user} | Comment: {comment}")
        if search_name.lower() in user.lower() or search_name.lower() in comment.lower():
            print(f"\n✅ Found '{search_name}'! Taking screenshot...\n")
            found = True
            break

    # If found, exit loop, else keep scrolling
    if found:
        break

# Take screenshot and quit
screenshot_path = "youtube_screenshot.png"
driver.save_screenshot(screenshot_path)
driver.quit()

# Display screenshot in the notebook
display(Image(filename=screenshot_path))
```

Gambar 3.29 *Notebook* YouTube *Scraping*

Gambar 3.29 merupakan gambar Skrip yang mengotomatiskan proses menemukan nama pengguna atau teks tertentu di dalam bagian komentar video YouTube menggunakan Selenium dan driver Chrome

yang tidak terdeteksi untuk meminimalkan deteksi bot. Skrip ini dimulai dengan meluncurkan browser Chrome, menavigasi ke URL video YouTube yang ditentukan, dan menunggu hingga bagian komentar dimuat sepenuhnya pada halaman. Selain mengandalkan *scraping* statis, bot memanfaatkan eksekusi JavaScript dalam konteks browser untuk mengekstrak nama pengguna dan teks komentar yang terlihat saat ini dari bagian halaman yang dimuat secara dinamis.

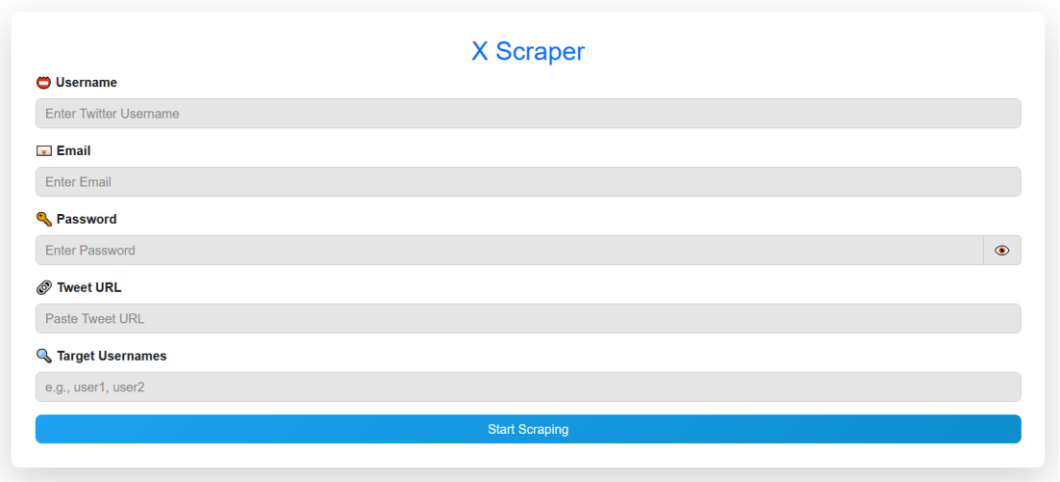
Memastikan semua komentar pada akhirnya diambil, skrip mensimulasikan perilaku alami pengguna dengan berulang kali menggulir ke bawah menggunakan tombol "*Page Down*". Setelah setiap *scroll*, skrip ini berhenti sejenak untuk memungkinkan komentar baru dimuat dan kemudian mengekstrak kembali konten yang diperbarui. Selama setiap siklus, bot memeriksa apakah target yang disediakan muncul dalam komentar apa pun. Saat kecocokan ditemukan, perulangan diakhiri, *screenshot* dari tampilan browser saat ini diambil, dan sesi browser ditutup. Tangkapan layar akhir kemudian ditampilkan di buku catatan, memberikan konfirmasi visual atas hasilnya. Pendekatan ini efisien untuk menangkap keterlibatan sosial atau memantau sebutan tertentu di YouTube tanpa bergantung pada batas API atau autentikasi pengguna.

3.2.4 Transformasi Notebook ke *Script* Python dan Integrasi dengan HTML

Setelah konsep awal dikembangkan di Jupyter Notebook, konsep tersebut diubah menjadi skrip Python mandiri agar lebih mudah diakses dan dikelola oleh admin. Proses ini dimulai dengan membangun antarmuka berbasis HTML sederhana, diikuti dengan mengubah *file* .ipynb menjadi skrip .py. Hal ini memungkinkan otomatisasi berjalan secara independen tanpa bergantung pada lingkungan Jupyter. Setelah nama pengguna atau komentar target berhasil dideteksi oleh skrip, *screenshot* dari hasil tersebut diambil secara otomatis. Gambar tersebut kemudian ditampilkan pada

halaman hasil yang terpisah, sehingga memudahkan admin untuk melihat dan memverifikasi hasilnya.

3.2.4.1 X (Twitter)



Gambar 3.30 X *Scraping* Web *Page*

Gambar 3.30 merupakan gambar *interface* X *Scraper* memungkinkan pengguna untuk mencari nama pengguna tertentu di dalam komentar atau balasan dari tweet tertentu pada platform yang sebelumnya dikenal sebagai Twitter (sekarang X). Untuk menggunakannya, pengguna perlu mengisi beberapa kolom seperti nama pengguna X mereka sendiri, email, dan kata sandi untuk mengautentikasi sesi, bersama dengan URL tweet yang berisi percakapan yang ingin mereka analisis. Ada juga opsi untuk mengalihkan visibilitas kata sandi agar lebih mudah dimasukkan.

Begitu masuk, sistem menggunakan URL tweet yang disediakan untuk mencari dan memproses bagian komentar. Pengguna harus memasukkan daftar nama pengguna target yang ingin mereka cari. Scraper kemudian akan mensimulasikan interaksi dengan tweet, memindai balasan atau *mention* menggunakan alat otomatisasi dan memeriksa kecocokan yang tepat dari nama pengguna yang disediakan. Tujuannya adalah untuk mengidentifikasi apakah pengguna yang terdaftar telah berinteraksi dengan tweet tertentu dan, jika ditemukan,

secara opsional mengambil *screenshot* sebagai bukti. Alat ini berguna untuk moderasi konten, investigasi, atau tujuan penelitian media sosial, terutama ketika memantau aktivitas pengguna di sekitar konten yang sensitif atau signifikan.

```python
from flask import Blueprint, request, render_template, redirect, url_for
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time
import os
import undetected_chromedriver as uc

# Flask Blueprint for Twitter Scraper
twitter_bp = Blueprint('twitter', __name__, template_folder="templates")

# Store results
found_users = {}
screenshot_paths = {}

@twitter_bp.route('/twitter_scrape', methods=['POST'])
def twitter_scrape():
    global found_users, screenshot_paths
    found_users.clear()
    screenshot_paths.clear()

    tweet_url = request.form.get("tweet_url", "").strip()
    username = request.form.get("username", "").strip()
    email = request.form.get("email", "").strip()
    password = request.form.get("password", "").strip()
    target_usernames = request.form.get("target_usernames", "").strip().split(",")

    if not tweet_url or not username or not email or not password or not target_usernames:
        return render_template("twitter.html", error="❌ Please fill in all fields.")

    target_usernames = {user.strip().lower() for user in target_usernames}  # Normalize usernames

    # Configure Selenium WebDriver
    options = webdriver.ChromeOptions()
    options.add_argument("--disable-blink-features=AutomationControlled")
    driver = uc.Chrome(options=options, headless=False)

    try:
        driver.get("https://x.com/i/flow/login")
        time.sleep(3)

        def enter_text(field_locator, value):
            """Finds an input field, clears it, and types a value (handles errors)."""
            for _ in range(3):  # Retry up to 3 times
                try:
                    field = WebDriverWait(driver, 10).until(
                        EC.element_to_be_clickable((By.CSS_SELECTOR, field_locator))
                    )
                    field.clear()
                    field.send_keys(value + Keys.RETURN)
                    time.sleep(2)
                    return True
                except Exception:
                    print(f"⚠ Retrying input for {field_locator}...")
                    time.sleep(2)
            return False
```

Gambar 3.31 *Script* Python X Bagian 1

```python
# Enter Email
enter_text("input", email)

# Check if username confirmation is needed
try:
    username_field = WebDriverWait(driver, 3).until(
        EC.presence_of_element_located((By.CSS_SELECTOR, "input"))
    )
    username_field.send_keys(username + Keys.RETURN)
    time.sleep(2)
except Exception:
    print("✅ No username confirmation needed.")

# Ensure Password Field Exists
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.NAME, "password"))
)

# Enter Password
enter_text("input[name='password']", password)

time.sleep(5)  # Wait for login to complete

# Navigate to Tweet
driver.get(tweet_url)
time.sleep(5)

static_folder = os.path.join(os.getcwd(), "static")
os.makedirs(static_folder, exist_ok=True)

found_targets = set()  # Track found usernames

print("🚀 Starting tab navigation...\n")

# Ensure Page Focus
body = driver.find_element(By.TAG_NAME, "body")
driver.execute_script("arguments[0].focus();", body)

tab_count = 0
```

Gambar 3.32 *Script* Python X Bagian 2

```
while True:
    try:
        driver.execute_script("arguments[0].focus();", body)

        webdriver.ActionChains(driver).send_keys(Keys.TAB).perform()
        time.sleep(0.05)

        focused_element = driver.execute_script("return document.activeElement;")
        if not focused_element:
            continue

        highlighted_text = focused_element.text.strip().lower()
        print(f"🔍 Highlighted: {highlighted_text}")

        for target in target_usernames:
            if target in highlighted_text and target not in found_targets:
                print(f"\n✅ Found '{target}'! Taking screenshot...\n")
                found_users[target] = True
                found_targets.add(target)

                screenshot_filename = f"{target}_screenshot.png"
                screenshot_path = os.path.join(static_folder, screenshot_filename)
                driver.save_screenshot(screenshot_path)
                screenshot_paths[target] = f"/static/{screenshot_filename}"

        tab_count += 1

        if found_targets == target_usernames:
            print("\n✅✅✅ All target usernames found! Stopping tabbing...\n")
            break

    except Exception as e:
        print(f"⚠️ Warning: Issue encountered while tabbing - {e}")
        time.sleep(0.2)

        print("🚀 Search finished. Tabs used:", tab_count)

    except Exception as e:
        print(f"❌ Error: {e}")
        found_users = {}
        screenshot_paths = {}

    driver.quit()
    return redirect(url_for('twitter.twitter_result'))

@twitter_bp.route('/twitter_result')
def twitter_result():
    return render_template("result.html",
                           found_usernames=found_users,
                           screenshot_paths=screenshot_paths,
                           platform="Twitter/X")
```

Gambar 3.33 *Script* Python X Bagian 3

Gambar 3.31, Gambar 3.32, Gambar 3.33 merupakan gambar yang mengimplementasi *scraping* nama pengguna Twitter/X menggunakan Selenium dan ChromeDriver. Metode ini memungkinkan pengguna untuk mengirimkan formulir yang berisi kredensial X mereka, URL tweet, dan daftar nama pengguna target. Setelah formulir dikirimkan, sistem memvalidasi *input* dan meluncurkan sesi browser otomatis. Sesi ini masuk ke X.com dengan mengisi email yang disediakan, mungkin mengonfirmasi nama pengguna jika diminta, dan memasukkan kata sandi. Setelah berhasil masuk, browser akan menavigasi ke tweet yang ditentukan.

Scraper kemudian memulai proses membuka tab melalui elemen-elemen pada halaman tweet, meniru navigasi *keyboard* manusia. Pada setiap tab, *scraper* memeriksa konten teks elemen yang sedang difokuskan. Jika salah satu nama pengguna target ditemukan di dalam teks, *scraper* menandainya sebagai ditemukan, mengambil tangkapan layar dari tampilan saat ini, dan menyimpannya. Hal ini terus berlanjut hingga semua nama pengguna yang ditentukan ditemukan atau hingga skrip selesai memindai halaman.

Selama proses tersebut, kode menyertakan penanganan kesalahan untuk mencoba kembali interaksi elemen dan menangani masalah yang tidak terduga tanpa mengalami kerusakan. Setelah operasi selesai, kode ini mengarahkan pengguna ke halaman hasil untuk melihat nama pengguna mana yang terdeteksi bersama dengan *screenshot* yang sesuai. Pendekatan ini memastikan cara yang tersembunyi dan kuat untuk *scraping* keberadaan pengguna Twitter dari konteks tweet tertentu.

3.2.4.2 TikTok

Gambar 3.34 TikTok *Scraping* Web *Page*

Gambar 3.34 menjelaskan *interface Scraper* Komentar TikTok di situs web. *Interface* ini memungkinkan pengguna untuk memasukkan URL video TikTok, daftar nama pengguna yang ingin mereka cari di dalam komentar, dan pengaturan penundaan yang menentukan berapa lama *scraper* harus menunggu di antara setiap tindakan *tabbing*. Selain

itu, ada tombol pengaman yang bisa diaktifkan atau dinonaktifkan. Ketika diaktifkan, mekanisme keamanan ini memperkenalkan penundaan yang disengaja di antara tindakan, mengurangi risiko terdeteksi atau ditandai sebagai bot oleh sistem TikTok. Alat ini bekerja dengan mensimulasikan navigasi *keyboard* melalui bagian komentar video TikTok, memeriksa kecocokan dengan nama pengguna target, dan mengambil *screenshot* jika ada yang ditemukan. *Tool* ini dirancang untuk membantu pengguna mengidentifikasi individu tertentu yang berinteraksi di dalam kolom komentar video TikTok dengan tetap mempertahankan profil yang lebih rendah untuk menghindari pembatasan.

```python
from flask import Blueprint, request, render_template, redirect, url_for
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
import os

# Flask Blueprint for TikTok Scraper
tiktok_bp = Blueprint('tiktok', __name__, template_folder="templates")

# Store results
found_users = {}
screenshot_paths = {}

@tiktok_bp.route('/tiktok_scrape', methods=['POST'])
def tiktok_scrape():
    global found_users, screenshot_paths
    found_users.clear()
    screenshot_paths.clear()

    video_url = request.form.get("video_url", "").strip()
    target_names = request.form.get("target_names", "").strip().split(",")
    tab_sleep_time = float(request.form.get("tab_sleep_time", 0.2))

    if not video_url or not target_names:
        return render_template("tiktok.html", error="❌ Please fill in all fields.")

    target_names = [name.strip() for name in target_names]  # Normalize names (keep case sensitivity)

    # Configure Selenium WebDriver
    options = webdriver.ChromeOptions()
    options.add_argument("--disable-blink-features=AutomationControlled")
    driver = webdriver.Chrome(options=options)

    try:
        driver.get(video_url)
        time.sleep(5)  # Allow page to load

        # Scroll down to load comments
        for _ in range(5):  # Adjust this number if needed
            driver.execute_script("window.scrollBy(0, 500);")  # Scroll down
            time.sleep(1)  # Allow comments to load

        tab_count = 0
        found_count = 0

        static_folder = os.path.join(os.getcwd(), "static")
        os.makedirs(static_folder, exist_ok=True)
```

Gambar 3.35 *Script* Python TikTok Bagian 1

```
        while found_count < len(target_names):
            driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
            time.sleep(tab_sleep_time)
            tab_count += 1

            focused_element = driver.execute_script("return document.activeElement;")
            if not focused_element:
                continue

            highlighted_text = focused_element.text.strip()
            print(f"🔍 Highlighted: {highlighted_text}")

            for name in target_names:
                # Ensure exact match of the username
                if highlighted_text == name and name not in found_users:
                    print(f"\n✅ Found '{name}'!\n")

                    # Tab one more time before screenshot
                    driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
                    time.sleep(0.2)  # Small delay for visibility

                    # Take screenshot
                    screenshot_filename = f"{name}_screenshot.png"
                    screenshot_path = os.path.join(static_folder, screenshot_filename)
                    driver.save_screenshot(screenshot_path)
                    screenshot_paths[name] = f"/static/{screenshot_filename}"

                    found_users[name] = True
                    found_count += 1  # Increase found counter
                    break  # No need to check further usernames for this tab

        print("📝 Search finished.")

    except Exception as e:
        print(f"❌ Error: {e}")
        found_users = {}
        screenshot_paths = {}

    driver.quit()
    return redirect(url_for('tiktok.tiktok_result'))

@tiktok_bp.route("/tiktok_result")
def tiktok_result():
    return render_template("result.html",
                           found_usernames=found_users,
                           screenshot_paths=screenshot_paths,
                           platform="TikTok")
```

Gambar 3.36 *Script* Python TikTok Bagian 2

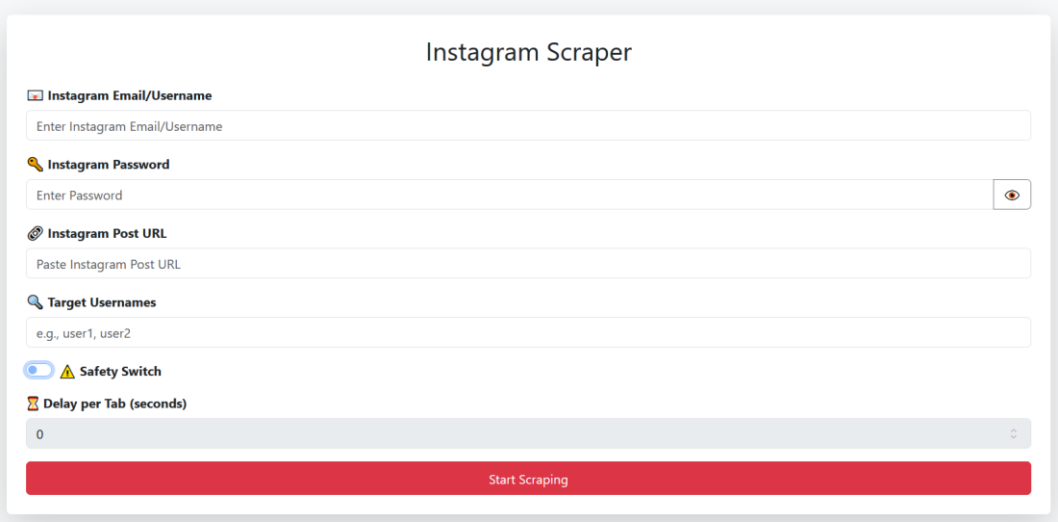
Gambar 3.35, Gambar 3.36 merupakan Skrip Python ini mendefinisikan *Blueprint* Flask secara khusus untuk *scraping* nama pengguna dari komentar video TikTok menggunakan otomatisasi Selenium. *Scraper* dipicu ketika pengguna mengirimkan formulir yang berisi URL video TikTok, daftar nama pengguna yang akan dicari, dan penundaan opsional di antara tindakan *tabbing*. Setelah dimulai, skrip meluncurkan *browser* Chrome menggunakan Selenium dan memuat URL video yang ditentukan.

Cara memuat lebih banyak komentar, skrip akan melakukan *scroll* halaman beberapa kali sebelum memulai proses *scraping*. Skrip ini mensimulasikan tab pada *keyboard* melalui elemen-elemen halaman web untuk menemukan nama pengguna. Setiap kali membuka tab, skrip ini akan memeriksa apakah teks elemen yang sedang difokuskan cocok dengan nama pengguna target yang disediakan. Jika ditemukan

kecocokan, alat ini akan menyimpan *screenshot* pada saat itu dan menyimpannya, menyimpan jalur gambar dan menandai nama pengguna tersebut sebagai nama pengguna yang ditemukan.

Alat ini terus melakukan penelusuran hingga semua nama pengguna target ditemukan atau hingga menghabiskan semua elemen yang tersedia. Setelah proses berakhir, browser ditutup, dan pengguna diarahkan ke halaman hasil yang menunjukkan nama pengguna mana yang ditemukan dan *screenshot* . Sistem ini juga menyertakan penanganan kesalahan untuk mengelola masalah apa pun selama eksekusi, seperti *input* yang hilang atau perubahan situs web. Otomatisasi ini menyediakan cara yang efisien untuk mengidentifikasi dan memverifikasi keberadaan pengguna di kolom komentar TikTok sambil menjaga interaksi tetap minimal dan efisien.

3.2.4.3 Instagram



Gambar 3.37 Instagram *Scraping* Web Page

Gambar 3.37 merupakan situs web yang berfungsi sebagai *interface* pengguna untuk alat *scraping* Instagram yang memungkinkan pengguna untuk mengotomatiskan proses pencarian nama pengguna tertentu dalam bagian komentar dari sebuah postingan Instagram. Formulir ini mencakup kolom *input* untuk email atau nama pengguna

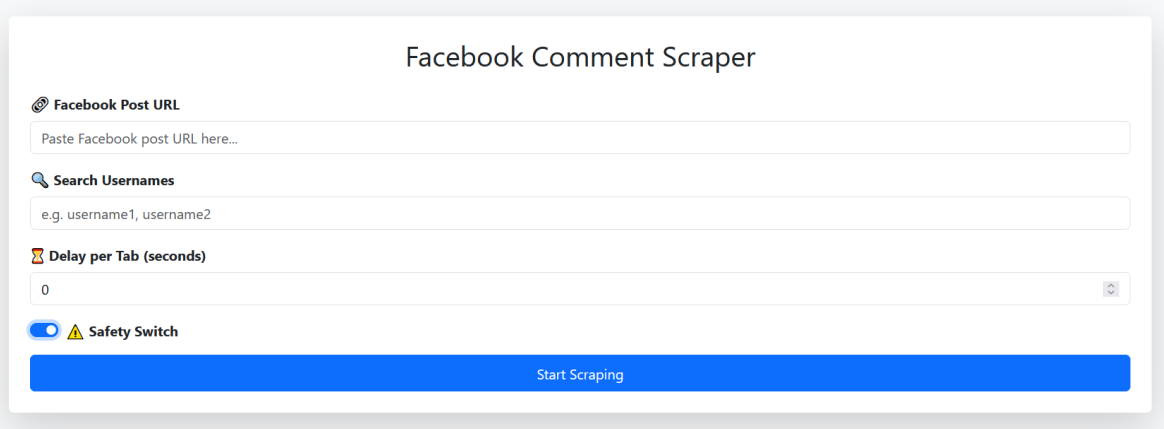
Instagram pengguna, kata sandi, URL postingan Instagram target, dan daftar nama pengguna yang akan dicari. Selain itu, ada opsi untuk mengaktifkan tombol pengaman, yang memperkenalkan penundaan di antara setiap tindakan tab otomatis. Penundaan ini membantu mengurangi risiko akun ditandai atau dilarang oleh Instagram untuk perilaku yang mencurigakan. Pengguna juga bisa mengatur waktu tunda spesifik (dalam hitungan detik) untuk setiap pergerakan tab. Secara keseluruhan, situs ini menyederhanakan proses *scraping* dengan menyediakan *interface* yang terstruktur dan ramah pengguna sambil juga menggabungkan langkah-langkah keamanan untuk menghindari deteksi.

```python
from flask import Blueprint, request, render_template
import os
import time
import undetected_chromedriver as uc
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

instagram_bp = Blueprint('instagram', __name__, template_folder="templates")

@instagram_bp.route('/instagram_scrape', methods=['POST'])
def instagram_scrape():
    ig_email = request.form.get("ig_email")
    ig_password = request.form.get("ig_password")
    post_url = request.form.get("post_url")
    target_usernames_str = request.form.get("target_usernames")
    safety_switch = request.form.get("safety_switch")
    tab_sleep_time = float(request.form.get("tab_sleep_time", 0.1))

    target_list = [username.strip() for username in target_usernames_str.split(",") if username.strip()]
    found_usernames = {username: False for username in target_list}
    screenshot_paths = {}

    options = uc.ChromeOptions()
    options.add_argument("--disable-blink-features=AutomationControlled")

    driver = uc.Chrome(options=options, headless=False)

    try:
        driver.get("https://www.instagram.com")
        time.sleep(5)
        wait = WebDriverWait(driver, 10)
        email_input = wait.until(EC.presence_of_element_located((By.NAME, "username")))

        email_input.send_keys(ig_email)
        password_input = driver.find_element(By.NAME, "password")
        password_input.send_keys(ig_password)
        password_input.send_keys(Keys.ENTER)
        time.sleep(5)

        try:
            not_now_button = wait.until(EC.presence_of_element_located((By.XPATH, "//button[text()='Not Now']")))
            not_now_button.click()
            time.sleep(3)
        except Exception:
            pass

        driver.get(post_url)
        time.sleep(5)

        static_folder = os.path.join(os.getcwd(), "static")
        os.makedirs(static_folder, exist_ok=True)

        tab_count = 0

        while not all(found_usernames.values()):
            driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
            time.sleep(tab_sleep_time)
            tab_count += 1

            highlighted_text = driver.execute_script("return document.activeElement.textContent;").strip()

            for username in target_list:
                if not found_usernames[username] and highlighted_text == username:
                    found_usernames[username] = True

                    driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
                    time.sleep(tab_sleep_time)

                    screenshot_filename = f"instagram_screenshot_{username}.png"
                    screenshot_path = os.path.join(static_folder, screenshot_filename)
                    driver.save_screenshot(screenshot_path)
                    screenshot_paths[username] = "/static/" + screenshot_filename

            if not all(found_usernames.values()):
                try:
                    load_more = driver.find_element(By.XPATH, "//svg[@aria-label='Load more comments']")
                    driver.execute_script("arguments[0].click();", load_more)
                    time.sleep(2)
                except Exception:
                    pass

    except Exception as e:
        print(f"Error: {e}")

    return render_template("result.html", screenshot_paths=screenshot_paths,
                           found_usernames=found_usernames, platform="Instagram")
```

Gambar 3.38 *Script* Python Instagram

Gambar 3.38 merupakan kode Python yang mendefinisikan *blueprint* Flask yang menangani logika *backend* untuk *scraping* komentar dari postingan Instagram menggunakan Selenium dan ChromeDriver yang tidak terdeteksi. Ketika permintaan POST dibuat ke rute tersebut, alat ini mengambil data formulir yang dikirimkan oleh pengguna, termasuk kredensial *login* Instagram, URL postingan, daftar nama pengguna target yang akan dicari di komentar, dan opsi-opsi seperti jeda keamanan di antara tindakan. Alat ini meluncurkan browser, masuk ke Instagram menggunakan kredensial yang disediakan, menavigasi ke pos yang ditentukan, dan memulai proses *tabbing* otomatis melalui elemen halaman. Alat ini memeriksa konten teks elemen yang sedang ditandai untuk melihat apakah konten tersebut cocok dengan nama pengguna target yang ditentukan. Jika ditemukan kecocokan, *screenshot* diambil dan disimpan, dan hasilnya disimpan untuk ditampilkan nanti. Jika tidak semua nama pengguna ditemukan, skrip akan mencoba mengeklik tombol "Muat lebih banyak komentar" dan melanjutkan pencarian. Setelah selesai, rute akan menampilkan halaman hasil yang menunjukkan status setiap nama pengguna target dan *screenshot* yang sesuai jika ditemukan. Skrip ini menyertakan penanganan kesalahan dan menggunakan folder statis khusus untuk menyimpan *screenshot*, yang kemudian ditampilkan melalui HTML.

3.2.4.4 Facebook



Gambar 3.39 Facebook *Scraping* Web Page

Gambar 3.39 merupakan *interface* yang menggambarkan *Scraper* Komentar Facebook yang dirancang untuk mengekstrak nama pengguna tertentu dari bagian komentar pada *postingan* Facebook dengan menggunakan otomatisasi. Pengguna diharuskan untuk memberikan URL *postingan* Facebook yang ingin mereka *scrape*, daftar nama pengguna target yang akan dicari di komentar, dan penundaan opsional per tab yang mengontrol seberapa cepat *scraper* mensimulasikan navigasi *keyboard* melalui elemen halaman. Penundaan ini membantu meniru perilaku manusia secara lebih realistis.

Fitur utama di sini adalah "*Safety Switch*", tombol yang mengaktifkan atau menonaktifkan fitur penundaan. Ketika diaktifkan, skrip menambahkan jeda singkat di antara setiap penekanan tombol tab yang disimulasikan, secara signifikan mengurangi kemungkinan Facebook mendeteksi dan memblokir aktivitas otomatis. Meskipun hal ini memperlambat proses *scraping*, namun ini meningkatkan *stealth* dan stabilitas. Ketika dinonaktifkan, scraper beroperasi pada kecepatan yang lebih cepat dan lebih efisien tetapi lebih berisiko dalam hal pendeteksian oleh mekanisme anti-bot Facebook.

Scraper bekerja dengan memuat postingan yang diberikan, melakukan *scroll* dan tab pada halaman, dan memeriksa setiap teks

elemen yang difokuskan untuk melihat apakah teks tersebut cocok dengan nama pengguna target. Jika ditemukan kecocokan, *scraper* akan mengambil *screenshot* dan menyimpannya untuk ditinjau oleh pengguna. Setelah semua nama pengguna ditemukan atau pencarian selesai, alat ini mengarahkan pengguna ke halaman hasil yang menampilkan nama pengguna mana yang terdeteksi beserta *screenshot*. Alat ini menggabungkan otomatisasi, penanganan kesalahan, dan kecepatan yang dapat disesuaikan untuk menyeimbangkan efisiensi dan keamanan platform.

```
from flask import Blueprint, request, render_template, jsonify, redirect, url_for
import os
import time
import random
import re
import undetected_chromedriver as uc
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

facebook_bp = Blueprint('facebook', __name__, template_folder="templates")

# Global Variables
found_usernames = {}  # Store search results
screenshot_paths = {}  # Store screenshot paths

@facebook_bp.route('/facebook_scrape', methods=['POST'])
def facebook_scrape():
    global found_usernames, screenshot_paths
    found_usernames.clear()
    screenshot_paths.clear()

    post_url = request.form.get("post_url", "").strip()
    target_usernames = request.form.get("target_usernames", "").strip().split(",")
    tab_sleep_time = float(request.form.get("tab_sleep_time", 0.2))
    remove_safety_limit = request.form.get("safety_limit") == "off"  # Check if safety limit is
disabled

    # Normalize and preserve exact usernames, including spaces
    target_usernames = [name.strip() for name in target_usernames]

    if not post_url or not target_usernames:
        return render_template("facebook.html", error="❌ Missing required fields. Please fill in all
fields.")

    options = uc.ChromeOptions()
    options.add_argument("--disable-blink-features=AutomationControlled")
    driver = uc.Chrome(options=options, headless=False)

    try:
        driver.get(post_url)
        time.sleep(random.uniform(5, 7))

        tab_count = 0

        static_folder = os.path.join(os.getcwd(), "static")
        os.makedirs(static_folder, exist_ok=True)
```

Gambar 3.40 *Script* Python Facebook Bagian 1

```
        # Keep searching until all usernames are found
        while len(found_usernames) < len(target_usernames) and tab_count < max_tabs:
            driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
            time.sleep(tab_sleep_time)
            tab_count += 1

            focused_element = driver.execute_script("return document.activeElement;")

            if not focused_element:
                continue  # Skip if no element is focused

            highlighted_text = focused_element.text.strip()

            print(f"Focused Text: {highlighted_text}")

            if "view more comments" in highlighted_text.lower():
                focused_element.send_keys(Keys.ENTER)
                time.sleep(random.uniform(2, 4))
                continue

            for username in target_usernames:
                # Only take a screenshot for the exact match
                if username in highlighted_text and username not in found_usernames:
                    if highlighted_text == username:
                        print(f"✅ Found exact match for username: {username}")

                        # Mark the username as found
                        found_usernames[username] = True

                        # Take a screenshot for the exact match
                        screenshot_filename = f"{username}_screenshot.png"
                        screenshot_path = os.path.join(static_folder, screenshot_filename)
                        driver.save_screenshot(screenshot_path)
                        screenshot_paths[username] = f"/static/{screenshot_filename}"
                        print(f"📸 Screenshot saved for {username}: {screenshot_path}")

                    break  # Break after finding the username to continue searching for the others

        print("🚀 Stopping search after finding all usernames.")

    except Exception as e:
        print(f"❌ Error: {e}")
        found_usernames = {}
        screenshot_paths = {}

    driver.quit()
    print("🔄 Redirecting to results page...")
    return redirect(url_for('facebook.facebook_result'))

@facebook_bp.route('/facebook_result')
def facebook_result():
    print("🔄 Rendering result.html now...")
    return render_template("result.html",
                           found_usernames=found_usernames,
                           screenshot_paths=screenshot_paths,
                           platform="facebook")
```

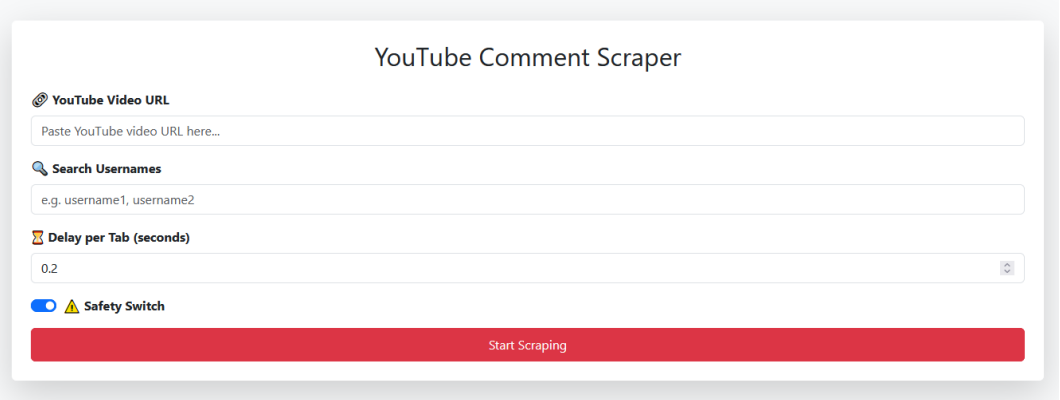
Gambar 3.41 *Script* Python Facebook Bagian 2

Gambar 3.40, Gambar 3.41 merupakan gambar kode yang mendefinisikan Blueprint Flask yang menangani penggalian komentar postingan Facebook untuk menemukan nama pengguna tertentu. Ketika pengguna mengirimkan formulir dengan URL postingan Facebook dan daftar nama pengguna target, skrip akan meluncurkan sebuah *instance browser* Chrome yang tidak terdeteksi menggunakan Selenium. Skrip ini menavigasi ke postingan Facebook yang disediakan dan mulai mensimulasikan penekanan "Tab" pada *keyboard* untuk bergerak melalui elemen interaktif pada halaman, meniru pengguna manusia.

Fungsi inti dari skrip ini adalah untuk memeriksa setiap elemen yang mendapatkan fokus melalui tab. Skrip ini memeriksa apakah teks dari elemen yang sedang difokuskan mengandung salah satu nama pengguna target. Jika ditemukan kecocokan yang tepat, skrip akan mengambil *screenshot* dan menyimpannya. Hal ini memberikan bukti visual bahwa nama pengguna tersebut terlihat pada postingan. Skrip ini melanjutkan pencarian hingga semua nama pengguna ditemukan atau hingga batasnya tercapai.

Penundaan di antara aksi tab dapat dikonfigurasi oleh pengguna untuk mengontrol kecepatan interaksi. Kecepatan yang lebih lambat lebih aman tetapi membutuhkan waktu lebih lama. Setelah pencarian selesai, sistem ini mengarahkan pengguna ke halaman hasil yang menunjukkan nama pengguna mana yang ditemukan bersama dengan tangkapan layar terkait. Sistem ini menyertakan penanganan kesalahan dan keluaran konsol untuk melacak kemajuan dan masalah selama proses penggalian.

3.2.4.5 YouTube



Gambar 3.42 YouTube *Scraping* Web *Page*

Gambar 3.42 merupakan *interface* YouTube Comment Scraper ini memungkinkan pengguna untuk mencari nama pengguna tertentu di dalam bagian komentar video YouTube. Pengguna mulai dengan memasukkan URL lengkap video YouTube di bidang “URL Video

YouTube". Kemudian, di bidang "'*Search Usernames*", mereka dapat memasukkan satu atau beberapa nama pengguna (dipisahkan dengan koma) yang mereka inginkan agar alat ini dapat menemukannya di antara komentar. Opsi "Penundaan per Tab (detik)" memungkinkan pengguna mengontrol berapa lama alat ini harus menunggu di antara tindakan *tabbing*, sehingga memungkinkannya untuk mensimulasikan perilaku yang lebih mirip manusia dengan menambahkan jeda setelah setiap interaksi. Selain itu, terdapat tombol "*Safety Switch*", yang ketika diaktifkan, akan mengaktifkan mode penelusuran yang lebih aman. Mode ini menambahkan penundaan dan membatasi jumlah tindakan untuk mengurangi kemungkinan terdeteksi sebagai bot atau diblokir oleh YouTube. Ketika tombol pengaman dimatikan, alat ini melakukan tindakan lebih cepat tanpa penundaan bawaan, membuatnya lebih cepat tetapi lebih berisiko dalam hal memicu sistem deteksi bot YouTube. Secara keseluruhan, pengaturan ini menyediakan cara yang dapat disesuaikan dan fleksibel untuk *scraping* dan memverifikasi nama pengguna dari bagian komentar YouTube sambil menyeimbangkan kecepatan dan keamanan.

```python
from flask import Blueprint, request, render_template, redirect, url_for
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time
import os
import undetected_chromedriver as uc

# Flask Blueprint for YouTube Scraper
youtube_bp = Blueprint('youtube', __name__, template_folder="templates")

# Store results
found_users = {}
screenshot_paths = {}

@youtube_bp.route('/youtube_scrape', methods=['POST'])
def youtube_scrape():
    global found_users, screenshot_paths
    found_users.clear()
    screenshot_paths.clear()

    video_url = request.form.get("video_url", "").strip()
    target_names = request.form.get("target_names", "").strip().split(",")
    tab_sleep_time = float(request.form.get("tab_sleep_time", 0.2))

    if not video_url or not target_names:
        return render_template("youtube.html", error="❌ Please fill in all fields.")

    # Normalize the target names and preserve exact names (spaces intact)
    target_names = [name.strip() for name in target_names]

    # Configure Selenium WebDriver
    options = webdriver.ChromeOptions()
    options.add_argument("--disable-blink-features=AutomationControlled")
    driver = uc.Chrome(options=options, headless=False)

    try:
        driver.get(video_url)
        time.sleep(5)  # Let page load

        # Scroll down multiple times to load comments properly
        for _ in range(5):  # Adjust the range if needed
            driver.find_element(By.TAG_NAME, "body").send_keys(Keys.PAGE_DOWN)
            time.sleep(1.5)
```

Gambar 3.43 *Script* Python YouTube Bagian 1

```
        # Extra wait to ensure comments are fully loaded
        time.sleep(3)

        tab_count = 0
        safety_limit = 500
        found_any = False  # To stop when all usernames are found

        static_folder = os.path.join(os.getcwd(), "static")
        os.makedirs(static_folder, exist_ok=True)

        # Continue searching until all usernames are found
        while len(found_users) < len(target_names) and tab_count < safety_limit:
            driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
            time.sleep(tab_sleep_time)
            tab_count += 1

            focused_element = driver.execute_script("return document.activeElement;")
            if not focused_element:
                continue  # Skip if no element is focused

            highlighted_text = focused_element.text.strip()
            print(f"🔍 Highlighted: {highlighted_text}")

            for name in target_names:
                if name == highlighted_text:  # Exact match check
                    if name not in found_users:
                        print(f"\n✅ Found exact match for '{name}'! Preparing to tab 3 more times...
\n")

                        # Tab 3 more times before taking a screenshot
                        for _ in range(3):
                            driver.find_element(By.TAG_NAME, "body").send_keys(Keys.TAB)
                            time.sleep(tab_sleep_time)

                        # Now take the screenshot after tabbing 3 more times
                        print(f"\n✅ Found '{name}'! Taking screenshot...\n")
                        found_users[name] = True

                        screenshot_filename = f"{name}_screenshot.png"
                        screenshot_path = os.path.join(static_folder, screenshot_filename)
                        driver.save_screenshot(screenshot_path)
                        screenshot_paths[name] = f"/static/{screenshot_filename}"

                    break  # Break after processing the found username

        print("🚀 Search finished. Found all usernames.")

    except Exception as e:
        print(f"❌ Error: {e}")
        found_users = {}
        screenshot_paths = {}

    driver.quit()
    return redirect(url_for('youtube.youtube_result'))

@youtube_bp.route('/youtube_result')
def youtube_result():
    return render_template("result.html",
                           found_usernames=found_users,
                           screenshot_paths=screenshot_paths,
                           platform="YouTube")
```

Gambar 3.44 *Script* Python YouTube Bagian 2

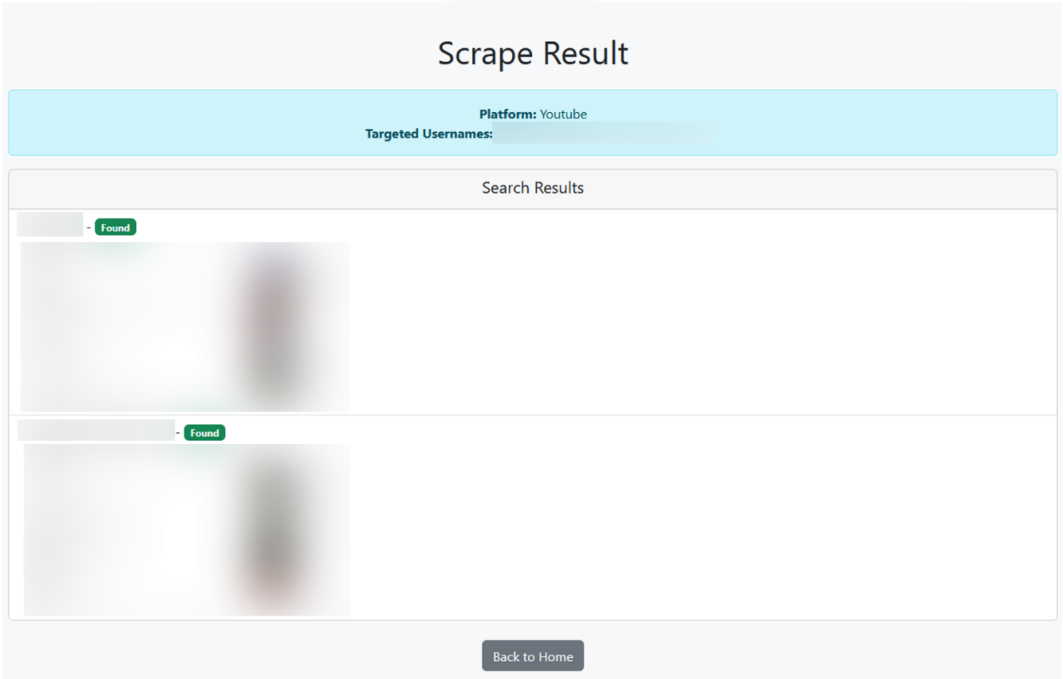
Gambar 3.43, Gambar 3.44 Kode ini mendefinisikan logika *backend* dari pengikis komentar YouTube menggunakan Flask dan Selenium. *Scraper* disusun sebagai Cetak Biru Flask untuk menjaga rute khusus YouTube tetap teratur. Hal ini memungkinkan pengguna untuk mengirimkan URL video YouTube bersama dengan daftar nama pengguna yang ingin mereka temukan di dalam komentar video tersebut. Ketika pengguna mengirimkan formulir, *backend* menerima URL video dan daftar nama pengguna target, dan juga menerima parameter penundaan opsional yang menentukan berapa lama program menunggu di antara *tabbing*. Penundaan ini penting untuk

mensimulasikan perilaku seperti manusia dan menghindari deteksi oleh sistem otomatis.

Scraper menggunakan driver browser Chrome yang tidak terdeteksi untuk membantu menghindari deteksi bot oleh YouTube. Pertama-tama, *scraper* membuka URL video yang disediakan, melakukan *scroll* ke bawah beberapa kali untuk memastikan bahwa komentar telah dimuat secara penuh, dan kemudian mulai memindai halaman dengan menggunakan tombol Tab. Setiap kali fokus berubah ke elemen baru, skrip akan memeriksa apakah teks yang difokuskan sama persis dengan nama pengguna target.

Jika ditemukan kecocokan yang tepat, *scraper* menunggu beberapa tab lagi, kemudian mengambil *screenshot*, dan menyimpan *screenshot* tersebut di bawah nama pengguna yang cocok. Proses ini terus berlanjut hingga semua nama pengguna ditemukan.

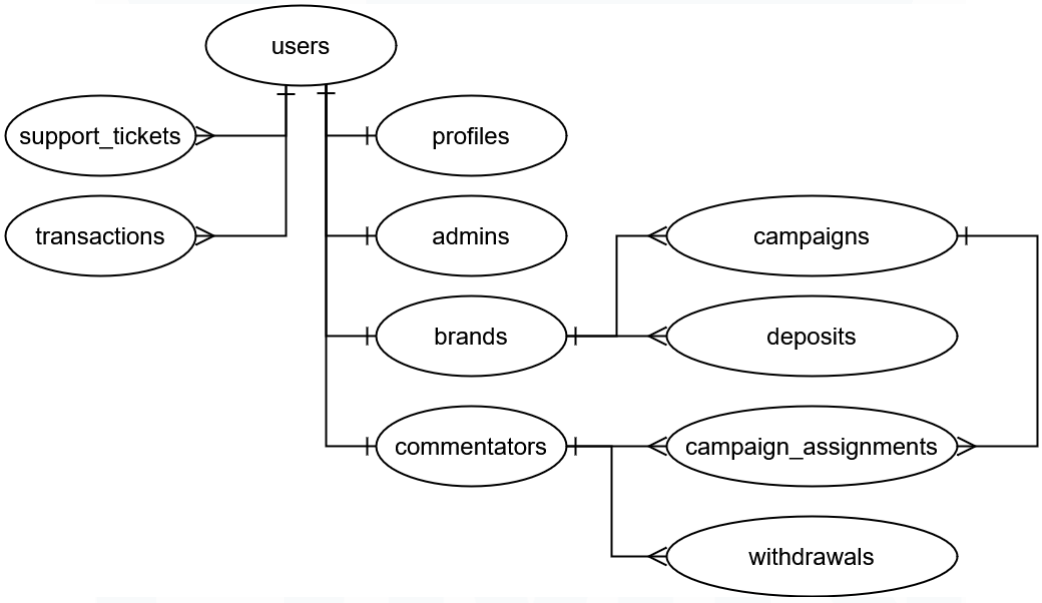
3.2.4.6 Result



Gambar 3.45 *Result Web Page*

Gambar 3.45 merupakan tampilan halaman hasil (*result page*) dari proses *scraping*. Halaman ini menyajikan informasi mengenai platform yang digunakan untuk *scraping* (misalnya: YouTube, Facebook, atau TikTok) serta daftar *username* yang menjadi target pencarian. Pada bagian bawah halaman, terdapat bagian "*Search Result*" yang menampilkan hasil pencarian dalam bentuk *screenshot*. *Screenshot-screenshot* ini menunjukkan bukti visual bahwa *username* yang dicari berhasil ditemukan dalam komentar pada platform yang ditentukan.

3.2.5 Perbaikan *Flow Database* dan Integrasi Ulang *Database* Sesuai Dengan *Website*



Gambar 3.46 ERD *Database*

Gambar 3.46 merupakan gambar ERD dari *database* yang sudah didesain ulang. Mulai dari tengah-tengah sistem adalah tabel pengguna, yang berperan sebagai entitas dasar yang mewakili setiap individu dalam sistem tanpa memandang peran mereka-apakah sebagai admin, merek, atau komentator. Setiap pengguna diasosiasikan dengan satu profil, menciptakan hubungan *one-to-one* (1:1) antara pengguna dan profil. Desain ini merupakan desain yang umum di mana data yang

berhubungan dengan autentikasi (seperti email dan kata sandi) disimpan terpisah dari informasi pengguna yang diperluas (seperti nama, biodata, atau avatar).

Dalam hal aktivitas dan dukungan sistem, setiap pengguna dapat mengirimkan beberapa *support_tickets*, membangun *hubungan one-to-many* (1:N) dari pengguna ke *support_tickets*. Begitu juga, satu pengguna dapat melakukan beberapa transaksi (seperti operasi keuangan), yang mengarah ke hubungan *one-to-many* dari pengguna ke transaksi. Struktur ini biasa terjadi pada sistem di mana pengguna terlibat dalam interaksi berkelanjutan yang perlu dicatat secara independen.

Peran dalam platform admin, merek, dan komentator-adalah perpanjangan dari tabel pengguna dasar dan memiliki hubungan *one-to-one* (1:1) dengan pengguna. Maksudnya, setiap pengguna hanya bisa menjadi bagian dari salah satu peran ini dalam satu waktu, dan setiap entitas peran menunjuk kembali ke satu pengguna. Pengaturan ini mendukung kontrol akses berbasis peran, yang memungkinkan berbagai jenis pengguna untuk melakukan tindakan yang berbeda tanpa menduplikasi tabel pengguna.

Pada admin, interaksi utama mereka adalah dengan kampanye. Seorang admin dapat membuat dan mengelola beberapa kampanye, membangun hubungan *one-to-many* (1:N) dari admin ke kampanye. Demikian juga, merek dikaitkan dengan kampanye yang mereka sponsori dan dengan setoran yang mereka buat untuk mendanai kampanye tersebut. Keduanya merupakan hubungan *one-to-many*, karena setiap merek dapat menjalankan beberapa kampanye dan melakukan beberapa deposit, tetapi setiap kampanye dan deposit hanya milik satu merek.

Komentator berinteraksi dengan kampanye melalui tabel *campaign_assignments*. Satu komentator dapat ditugaskan ke banyak kampanye, sehingga menghasilkan hubungan *one-to-many* dari komentator ke *campaign_assignments*. Setiap penugasan menghubungkan satu komentator ke satu kampanye. Selain itu, komentator dapat meminta beberapa penarikan, membentuk hubungan *one-to-many* lainnya, karena setiap penarikan dikaitkan dengan satu komentator.

Terakhir, kampanye itu sendiri terkait dengan *campaign_assignments* melalui hubungan *one-to-many* (1:N). Setiap kampanye dapat melibatkan beberapa komentator, masing-masing diwakili oleh catatan penugasan yang terpisah.

3.3 Kendala yang Ditemukan

Selama periode magang, Mahasiswa menghadapi beberapa tantangan penting yang berdampak pada produktivitas dan efisiensi. Tantangan-tantangan ini muncul baik dari aspek teknis maupun non-teknis dari kegiatan magang:

1) Salah satu masalah yang cukup signifikan adalah jarak yang jauh antara tempat tinggal mahasiswa di Lippo Karawaci dengan lokasi magang di Mega Kuningan, Jakarta Selatan. Perjalanan setiap hari tidak hanya melelahkan secara fisik tetapi juga membebani secara finansial karena mahalnya biaya transportasi pribadi atau layanan transportasi *online*. Situasi ini menjadi kekhawatiran yang berulang, terutama selama hari kerja penuh di lokasi.
2) Tantangan utama lainnya adalah keterbatasan teknis dalam mengumpulkan data keterlibatan pengguna, memverifikasi komentar pengguna di platform media sosial tanpa akses ke API resmi. Banyak platform media sosial yang membatasi penggalian otomatis dan memberlakukan batas nilai atau *captcha*, menjadi sulit untuk mengotomatiskan proses pemeriksaan komentar secara efisien. Hal ini menghambat tujuan otomatisasi sistem dan

membutuhkan solusi kreatif untuk tetap memberikan data yang akurat dan dapat diverifikasi.

3.4 Solusi atas Kendala yang Ditemukan

Terlepas dari masalah yang dihadapi, Mahasiswa menerapkan solusi yang efektif dan praktis untuk mengatasi hambatan ini dan memastikan kelangsungan dan keberhasilan kerja magang:

1) Untuk mengatasi biaya transportasi yang tinggi, Mahasiswa memilih untuk beralih ke transportasi umum yang lebih terjangkau dan efisien. Dalam rutin penggunaan *Commuter Line* (KRL), Mahasiswa dapat mengurangi pengeluaran secara signifikan dan juga mempersingkat waktu perjalanan. Perubahan ini tidak hanya meminimalisir beban keuangan, tetapi juga meningkatkan ketepatan waktu dan mengurangi stres akibat perjalanan.
2) Menanggapi kesulitan dalam melakukan *scraping* pada platform media sosial tanpa API resmi, Mahasiswa mengembangkan solusi otomatisasi tab. Sistem ini meniru perilaku pengecekan manual dengan membuka tab browser secara terprogram dan mencari nama pengguna di dalam kolom komentar. Ketika nama pengguna yang cocok ditemukan, halaman tersebut secara otomatis melakukan *screenshot* dan disimpan dalam *database*. Ini berfungsi sebagai bukti visual bahwa tugas yang diberikan telah diselesaikan oleh pengguna. Meskipun tidak semulus integrasi API, metode ini terbukti menjadi solusi yang praktis untuk mengotomatiskan verifikasi dengan tingkat akurasi yang tinggi.