BAB 3
METODOLOGI PENELITIAN

3.1 Metodologi Penelitian

3.1.1 Studi Literatur

Studi literatur merupakan langkah pertama dalam tahapan penelitian, karena pada tahapan ini penulis akan mempelajari dasar-dasar teori yang relevan terhadap topik penelitian. Literatur yang akan dipelajari adalah teori-teori yang berkaitan dengan pembangunan *smart contract* pada NEAR Protocol dan sosial media terdesentralisasi.

3.1.2 Analisis Kebutuhan

Analisis kebutuhan merupakan salah satu *building block* dalam merancang sebuah perangkat lunak. Analisis kebutuhan akan mencakup daftar-daftar permintaan yang harus dipenuhi dalam perancangan perangkat lunak agar dapat membuat suatu aplikasi yang utuh. Analisis kebutuhan akan berisikan fitur-fitur dasar sesuai yang dijelaskan pada bagian Batasan Masalah. Analisis kebutuhan disampaikan dalam bentuk *user stories* (poin-poin dari perspektif orang ketiga).

3.1.3 Perancangan *Smart Contract* dan Antarmuka

Tahap perancangan di sini dimaksud dengan cetak biru (*blueprint*) dari sistem yang akan dirancang. Cetak biru akan mencakup struktur data, diagram alir, dan purwarupa antarmuka. Dari cetak biru yang telah dibuat, maka proses implementasi dapat segera dilakukan.

3.1.4 Implementasi Sistem

Berdasarkan perancangan yang sudah dilakukan, maka pembuatan *smart contract* akan dilakukan dengan menggunakan NEAR Protocol dengan Rust SDK. Setelah *smart contract* selesai dirancang, maka akan dilakukan *deployment* sehingga dapat dipakai saat dihubungkan dengan kode *client*.

3.1.5 Pengujian Sistem dan Evaluasi

Pengujian sistem dilakukan setelah sistem telah selesai dirancang sesuai dengan cetak biru beserta dengan penyesuaian terhadap keterbatasan-keterbatasan yang ada (seperti waktu dan biaya). Pengujian dilakukan untuk memastikan bahwa tidak ada masalah yang dapat mengancam siklus hidup sistem dan dilakukan dengan cara memanggil fungsi-fungsi yang relevan. Pengujian dilakukan untuk meredam masalah-masalah yang berpotensi merusak alur aplikasi. Evaluasi akan dilakukan dengan menganalisis biaya transaksi yang dilakukan dalam simulasi penggunaan sehari-hari kemudian dilakukan perhitungan rata-ratanya, dan juga mengukur tingkat penerimaan pengguna terhadap hasil analisis terhadap biaya transaksi yang telah dilakukan.

3.1.6 Penulisan Hasil Penelitian

Penulisan hasil penelitian akan dilakukan sebagai hasil rekapan dan dokumentasi keseluruhan siklus hidup penelitian, sehingga dapat dijadikan sumber ilmu pengetahuan dan juga untuk sebagai salah satu referensi yang dapat dijadikan acuan oleh peneliti lain di masa depan.

3.2 Perancangan Aplikasi

3.2.1 *Sitemap*

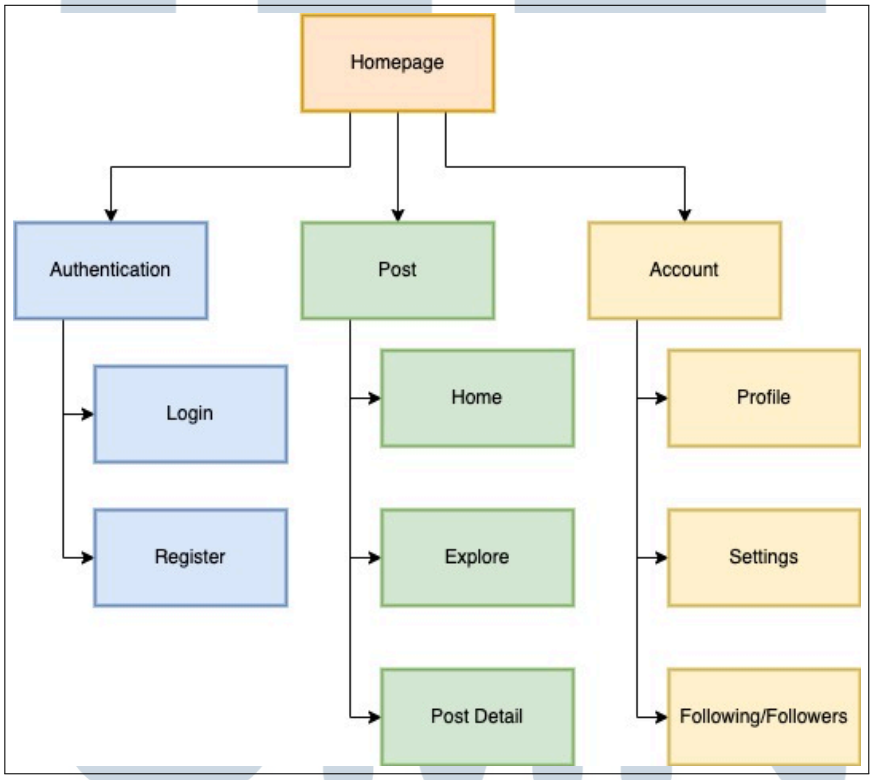
Sitemap digunakan untuk menggambarkan relasi halaman-halaman antarmuka yang dibangun pada sistem, relasi antarhalaman untuk aplikasi media sosial terdesentralisasi dapat digambarkan sebagai berikut. Pada Gambar 3.1 dapat dilihat bahwa terdapat tiga komponen utama dalam aplikasi media sosial terdesentralisasi, yaitu *authentication*, *post*, dan *account*.

Komponen *authentication* yang terdiri atas dua halaman digunakan sebagai gerbang masuk ke dalam aplikasi sehingga aplikasi dapat mengenali penggunanya. Halaman *login* digunakan untuk masuk ke aplikasi menggunakan dompet NEAR, dan halaman *register* akan diarahkan oleh sistem kepada pengguna jika saat melakukan proses *login* pengguna belum memiliki akun yang terdaftar pada *smart contract*.

Komponen *post* terdiri atas tiga halaman. Halaman *home* digunakan untuk membuat *post* baru dan melihat keseluruhan *post* oleh pengguna yang telah diikuti

oleh *user* yang telah masuk ke dalam aplikasi, halaman *explore* digunakan untuk melihat *post* terhadap *post* yang dibuat oleh pengguna lain baik telah diikuti atau belum, dan halaman *post detail* digunakan untuk melihat detail dari sebuah *post* secara keseluruhan berdasarkan ID *post* tersebut.

Komponen *account* terdiri atas tiga halaman. Halaman *profile* digunakan untuk melihat detail profil dari sebuah pengguna yang telah masuk ke dalam aplikasi, halaman *settings* digunakan untuk segala hal yang berkaitan dengan detail akun sehingga pengguna dapat mengubah data yang ada, dan halaman *following/followers* digunakan untuk melihat daftar pengikut dan akun yang telah diikuti oleh pengguna.



Gambar 3.1. *Sitemap* media sosial terdesentralisasi

3.2.2 Kebutuhan Aplikasi

Fitur-fitur pada media sosial terdesentralisasi terinspirasi dari media sosial *Twitter*, maka berikut adalah fitur-fitur dasar dari *Twitter* [16] yang akan diimplementasikan ke dalam penelitian ini.

- Pengguna dapat melakukan pendaftaran akun

- Pengguna dapat masuk ke dalam sosial media menggunakan akun yang sudah terdaftar
- Pengguna dapat membuat *post* baru
- Pengguna dapat melihat *post* yang dibuat oleh akun lain
- Pengguna dapat memberikan komentar kepada *post* yang tersedia
- Pengguna dapat menyukai *post* yang tersedia
- Pengguna dapat batal menyukai *post* yang telah disukai
- Pengguna dapat mengikuti (*follow*) akun milik pengguna lain
- Pengguna dapat batal mengikuti akun milik pengguna lain
- Pengguna dapat menyunting data profil

3.2.3 Skema *Struct*

Proses penyimpanan data di *smart contract* pada NEAR akan dilakukan dengan pendekatan yang berbeda dibandingkan dengan sistem penyimpanan menggunakan basis data konvensional seperti *RDBMS (relational database management system)* atau NoSQL. Alih-alih menggunakan pendekatan basis data konvensional untuk menyimpan data, keseluruhan proses mutasi data harus dilakukan menggunakan *struct* untuk mendefinisikan skema—*struct* pada konteks ini digunakan untuk menyimpan data yang saling berkorelasi dan memiliki kemampuan untuk memiliki tipe data yang berbeda untuk setiap *key*, serupa dengan konsep *object* pada pemrograman berorientasi objek. Nantinya, *struct* akan dipadukan dengan *collections* sehingga dapat menyimpan data secara majemuk sehingga dapat direpresentasikan sebagai basis data—*collections* pada konteks ini mengacu terhadap sekumpulan struktur data yang disediakan oleh NEAR dan Rust (bahasa pemrograman yang digunakan dalam perancangan *smart contract*).

Pada *smart contract* yang dirancang, terdapat 8 definisi skema basis data yang dibuat dalam bentuk *struct*. Berikut adalah kesebelas definisi skema tersebut.

Tabel 3.1. Skema *Struct UserAccountDetail*

Nama Fields	**Tipe Data**	**Deskripsi**
address	*String*	Menyimpan alamat *address*
name	*String*	Menyimpan nama pengguna (opsional)
profile_image_url	*String*	Menyimpan URL gambar profil (opsional)
location	*String*	Menyimpan lokasi pengguna (opsional)
url	*String*	Menyimpan URL pribadi pengguna (opsional)
description	*String*	Menyimpan bio pengguna (opsional)
created_at	*u64* (unsigned int 64-bit)	Menyimpan tanggal pembuatan akun, dilakukan secara otomatis saat akun dibuat dengan *block_timestamp*

Tabel 3.2. Skema *Struct UserFollowers*

Nama *Fields*	Tipe Data	Deskripsi
user_account_id	*AccountId*	Menyimpan alamat *address* yang diikuti
follower_account_id	*AccountId*	Menyimpan alamat *address* yang mengikuti *user_account_id*

Tabel 3.3. Skema *Struct PostDetail*

Nama *Fields*	Tipe Data	Deskripsi
post_id	*u64*	ID *post*
user_address	*AccountId*	*Address* pembuat *post*
content	*String*	Isi *post*
created_at	*u64*	Tanggal pembuatan *post*

Tabel 3.4. Skema *Struct PostOutputFormat*

Nama *Fields*	Tipe Data	Deskripsi
name	*String*	Nama pembuat *post*
profile_image_url	*String*	URL gambar profil pembuat *post*
post	*PostDetail*	Detail *post*
like_count	*u64*	Jumlah *post likes*
comment_count	*u64*	Jumlah *post comments*
like_details	*Option<Vec <PostLikes>>*	Detail *likes*
comment_details	*Option<Vec <PostCom-ment>>*	Detail komentar
is_liked	*Option<bool>*	Apakah post disukai oleh seorang user?

Tabel 3.5. Skema *Struct PostLikes*

Nama *Fields*	Tipe Data	Deskripsi
post_id	*u64*	ID *post*
user_address	*AccountId*	*Address* yang memberikan *like*
created_at	*u64*	Tanggal like diberikan

Tabel 3.6. Skema *Struct PostComment*

Nama *Fields*	Tipe Data	Deskripsi
comment_id	*u64*	ID komentar
post_id	*u64*	ID *post*
user_address	*AccountId*	*Address* yang memberikan ko-mentar
comment	*String*	Isi komentar yang diberikan
created_at	*u64*	Tanggal komentar diberikan

Tabel 3.7. Skema *Struct PostLikesDetailsOutput*

Nama *Fields*	Tipe Data	Deskripsi
user_address	*AccountId*	*Address* yang memberikan *like*
profile_image_url	*String*	URL gambar profil

Tabel 3.8. Skema *Struct PostCommentDetailsOutput*

Nama *Fields*	Tipe Data	Deskripsi
comment_id	*u64*	ID komentar
user_address	*AccountId*	*Address* yang memberikan komentar
profile_image_url	*String*	URL gambar profil
comment	*String*	Isi komentar yang diberikan
created_at	*u64*	Tanggal komentar diberikan

Setelah mendefinisikan skema data, berikut adalah inisialisasi *contract states* yang dilakukan.

Tabel 3.9. Inisialisasi *Contract States*

Nama *Fields*	Tipe Data	Deskripsi
user_list	*LookupMap<AccountId, UserAccountDetail>*	Menyimpan daftar pengguna
user_followers	*Vector<User Followers>*	Menyimpan daftar pengguna yang mengikuti pengguna lain
all_posts	*Vector<Post Detail>*	Menyimpan semua *posts* yang dibuat oleh pengguna
post_likes	*Vector<Post Likes>*	Menyimpan daftar pengguna yang menyukai *post* tertentu
post_comments	*Vector<Post Comment>*	Menyimpan daftar pengguna yang meninggalkan komentar di *post* tertentu

3.2.4 Perancangan *Smart Contract*

Smart contract berfungsi untuk mengatur keseluruhan logika yang terjadi pada aplikasi dan juga digunakan untuk menyimpan *state* ke dalam *blockchain* sehingga mutasi data yang terjadi bersifat tetap, dan juga digunakan untuk membaca data yang telah tersimpan. NEAR sendiri membedakan fungsi pada *smart contract* menjadi 2 jenis yaitu *viewMethods* dan *changeMethods*. *viewMethods* berguna sebagai fungsi yang dapat digunakan untuk membaca data namun tidak memiliki akses untuk melakukan mutasi data, sedangkan *changeMethods* digunakan untuk kebutuhan aplikasi di mana suatu mutasi data pada *contract* dibutuhkan.

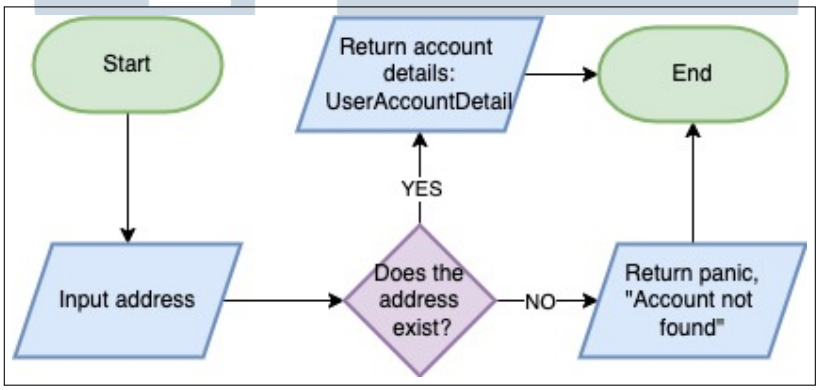
Pada perancangan *contract* media sosial terdesentralisasi, terdapat 10 *viewMethods* dan 7 *changeMethods*, yang perinciannya dapat dilihat pada Tabel 3.10.

Tabel 3.10. Daftar fungsi beserta jenisnya pada *smart contract*

Nama Fungsi (*method*)	Jenis
is_user_exists	*viewMethods*
get_account_details	*viewMethods*
is_user_followed	*viewMethods*
get_user_following_list	*viewMethods*
get_user_followers_list	*viewMethods*
get_all_posts	*viewMethods*
get_all_posts_personalized	*viewMethods*
get_single_post	*viewMethods*
get_user_posts	*viewMethods*
get_post_likes_details	*viewMethods*
get_post_comments_details	*viewMethods*
create_account	*changeMethods*
follow_user	*changeMethods*
create_post	*changeMethods*
like_post	*changeMethods*
comment_on_post	*changeMethods*
edit_account_details	*changeMethods*
edit_profile_image	*changeMethods*

A Diagram Alir *get_account_details*

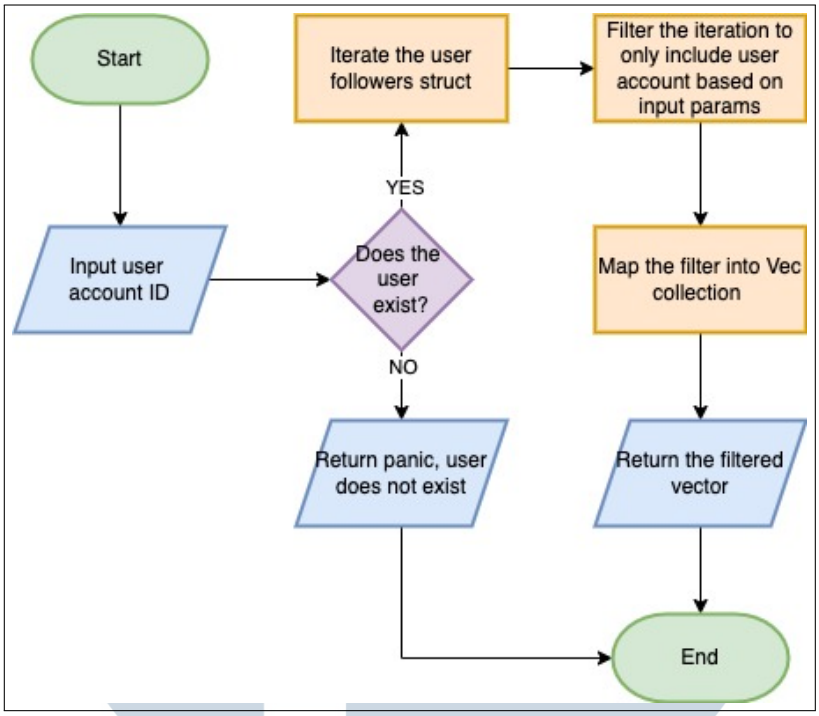
Fungsi *get_account_details* digunakan untuk mendapatkan keseluruhan detail dari sebuah akun yang terdaftar, meliputi *address*, nama, URL gambar profil, lokasi, dan bio profil. Fungsi ini bersifat *immutable* dan menerima satu parameter, yaitu *address* yang memiliki tipe *AccountId* yang nantinya akan digunakan untuk mencari keberadaan detail akun berdasarkan argumen yang diberikan. Sebelum melakukan pencarian pada *LookupMap*, akan dipastikan terlebih dahulu bahwa *key* yang diberikan ada. Hasil yang dikembalikan dari fungsi ini memiliki tipe *UserAccountDetail*. Diagram alir dari fungsi ini dapat dirujuk pada Gambar 3.2.



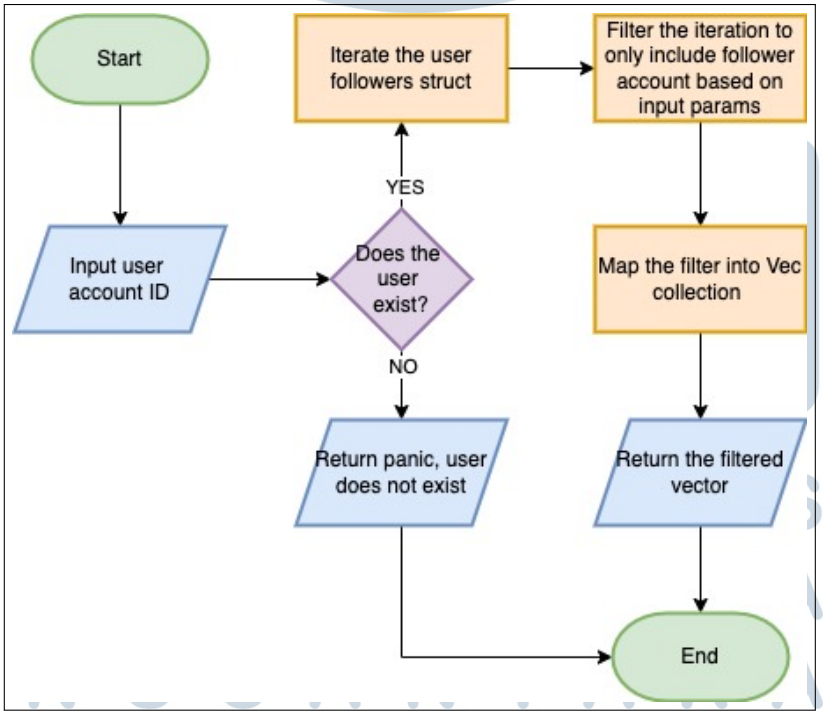
Gambar 3.2. Diagram Alir *get_account_details*

B Diagram Alir *get_user_following_list* dan *get_user_followers_list*

Fungsi *get_user_following_list* digunakan untuk mendapatkan daftar *following* yang dimiliki oleh suatu akun, dan fungsi *get_user_followers_list* digunakan untuk mendapatkan daftar *followers* yang dimiliki. Kedua fungsi ini bersifat *immutable* dan menerima satu parameter, yaitu *user_account_id* yang memiliki tipe *AccountId*. Parameter ini dimaksudkan untuk mencari daftar *following/followers* yang dimiliki user berdasarkan argumen yang diberikan. Sebelum melakukan pencarian, akan dilakukan pengecekan terlebih dahulu apakah nilai parameter yang diberikan merupakan sebuah user yang terdaftar. Fungsi ini akan mengembalikan nilai berupa *Vec¡String¿*. Diagram alir dari fungsi ini dapat dirujuk pada Gambar 3.3 dan 3.4.



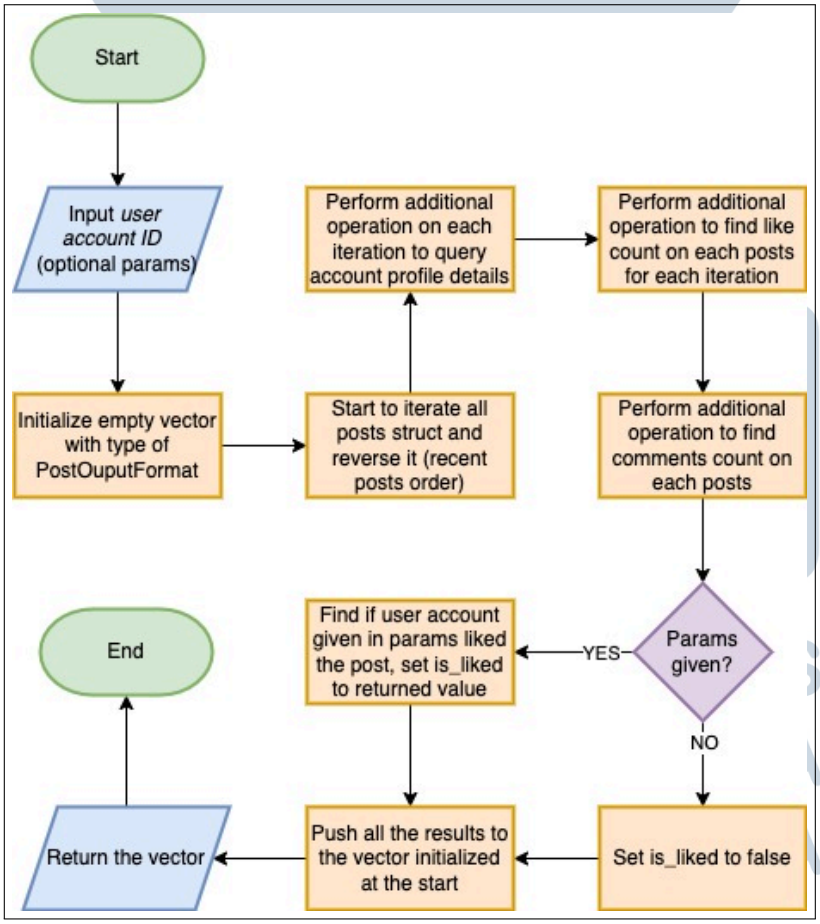
Gambar 3.3. Diagram Alir *get_user_following_list*



Gambar 3.4. Diagram Alir *get_user_followers_list*

C Diagram Alir *get_all_posts*

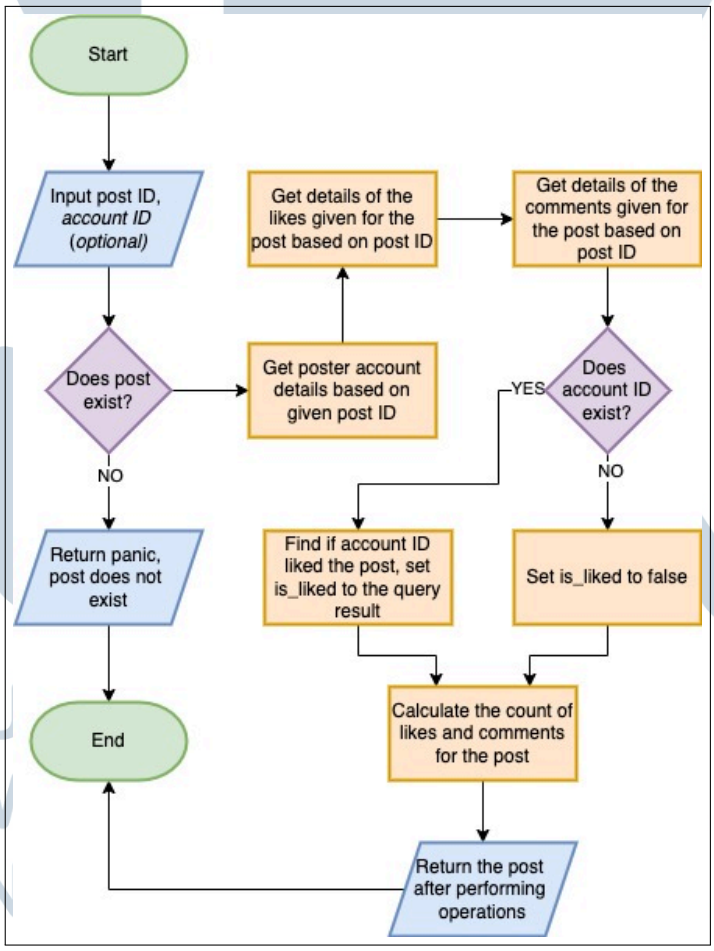
Fungsi *get_all_posts* digunakan untuk mendapatkan keseluruhan *post* yang disimpan pada *contract*. Fungsi ini bersifat *immutable* dan menerima satu parameter **opsional**, yaitu *account_id* yang memiliki tipe *AccountId*. Parameter opsional ini jika diberikan nilai akan mengubah nilai dari *is_liked* sehingga dapat memberikan persektif dari sebuah user telah memberikan *like* terhadap suatu *post*. Fungsi ini bekerja dengan cara melakukan iterasi pada *all_posts*, kemudian melakukan populasi data profil, jumlah *like*, dan jumlah *comment* untuk setiap iterasi yang terjadi. Jika parameter diberikan, maka akan dilakukan operasi tambahan untuk mengetahui apakah user yang nilainya diberikan dari parameter telah menyukai *post* tersebut. Setelah selesai melakukan iterasi, fungsi ini akan mengembalikan *Vector* dengan tipe *PostOutputFormat*. Diagram alir dari fungsi ini dapat dirujuk pada Gambar 3.5.



Gambar 3.5. Diagram Alir *get_all_posts*

D Diagram Alir *get_single_post*

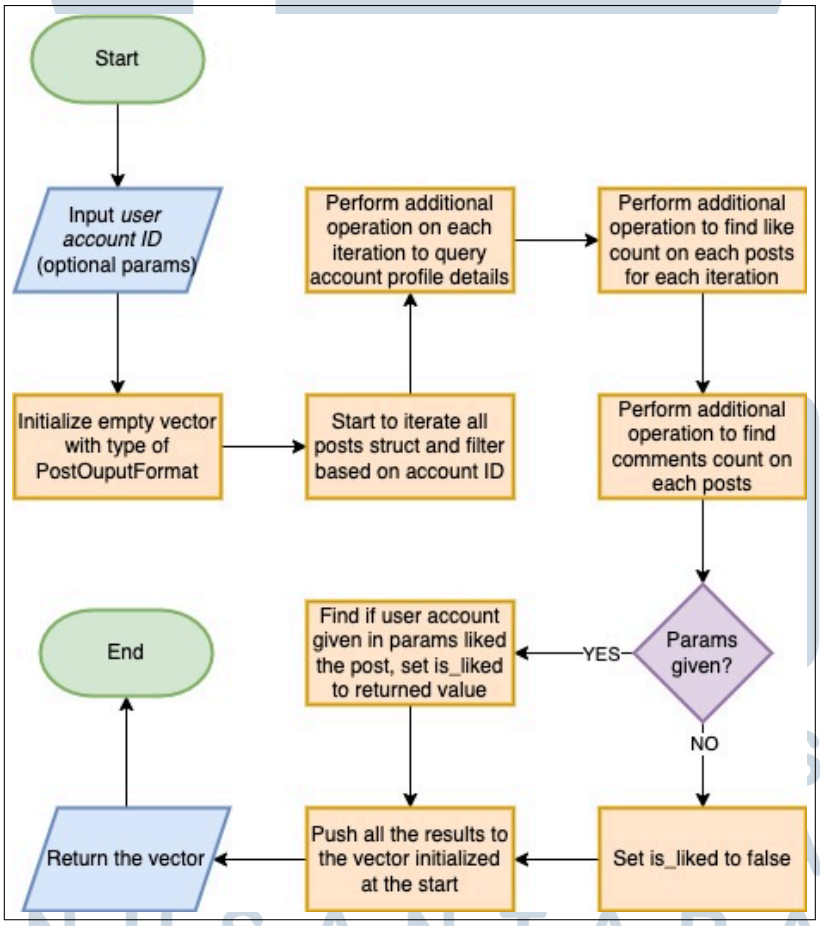
Fungsi *get_single_post* digunakan untuk mendapatkan detail dari sebuah *post* yang dapat diidentifikasi melalui ID *post* yang diberikan melalui parameter yang tersedia pada definisi fungsi ini. Juga terdapat satu parameter opsional lain yaitu *account_id* untuk melihat apakah user yang nilainya diberikan melalui parameter telah menyukai *post* tersebut. Sebelum melakukan operasi pencarian detail *post*, akan dipastikan terlebih dahulu bahwa *post* berdasarkan *post ID* tersebut tersedia di dalam *contract*. Operasi pencarian profil *poster* akan dilakukan dan juga untuk pencarian detail *likes* dan *comments*. Hampir serupa dengan fungsi sebelumnya, fungsi ini akan memgembalikan nilai dengan tipe *PostOutputFormat* tetapi tidak dengan *Vector* karena tujuan keluaran dari fungsi ini adalah *post* tunggal. Diagram alir dari fungsi ini dapat dirujuk pada Gambar 3.6.



Gambar 3.6. Diagram Alir *get_single_post*

E Diagram Alir *get_user_posts*

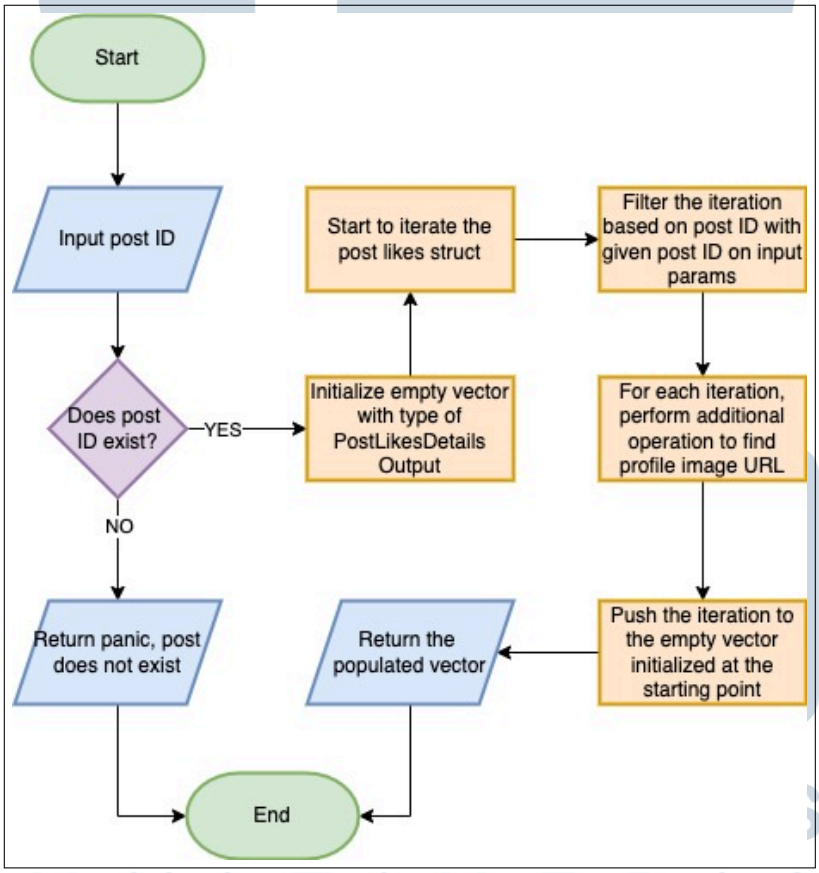
Fungsi *get_user_posts* digunakan untuk mendapatkan keseluruhan *post* yang dimiliki oleh sebuah user berdasarkan parameter *account_id* yang diberikan. Serupa dengan fungsi sebelumnya, juga terdapat parameter opsional yaitu *perspective* dengan tipe *AccountId* untuk melihat apakah suatu akun telah memberikan *likes* pada *post* yang dibuat oleh user tersebut. Cara kerja dari fungsi ini serupa dengan *get_all_posts*, dengan sedikit modifikasi pada iterasi: sebelum melakukan iterasi terhadap keseluruhan *post* yang tersedia pada *contract*, akan dilakukan *filter* terlebih dahulu sehingga *post* yang diiterasi merupakan hanya *post* yang dibuat oleh user pada *account_id*. Fungsi ini akan mengembalikan nilai *Vector* dengan tipe *PostOutputFormat*. Diagram alir dari fungsi ini dapat dirujuk pada Gambar 3.7.



Gambar 3.7. Diagram Alir *get_user_posts*

F Diagram Alir *get_post_likes_details*

Fungsi *get_post_likes_details* digunakan untuk mendapatkan detail *likes* pada suatu *post* berdasarkan ID post yang diberikan melalui parameter. Fungsi ini bekerja dengan cara melakukan iterasi pada koleksi *post_likes* kemudian melalukan filtrasi terhadap ID *post* yang diberikan. Untuk meningkatkan *user experience* (UX) pada sisi *client*, dilakukan operasi untuk mencari *profile_image_url* pada setiap iterasi yang terjadi. Keluaran dari fungsi ini akan berisikan alamat *address* user yang menyukai *post* tersebut beserta dengan foto profilnya, yang skemanya didefinisikan pada *struct PostLikesDetailsOutput*. Diagram alir dari fungsi ini dapat dirujuk pada Gambar 3.8.

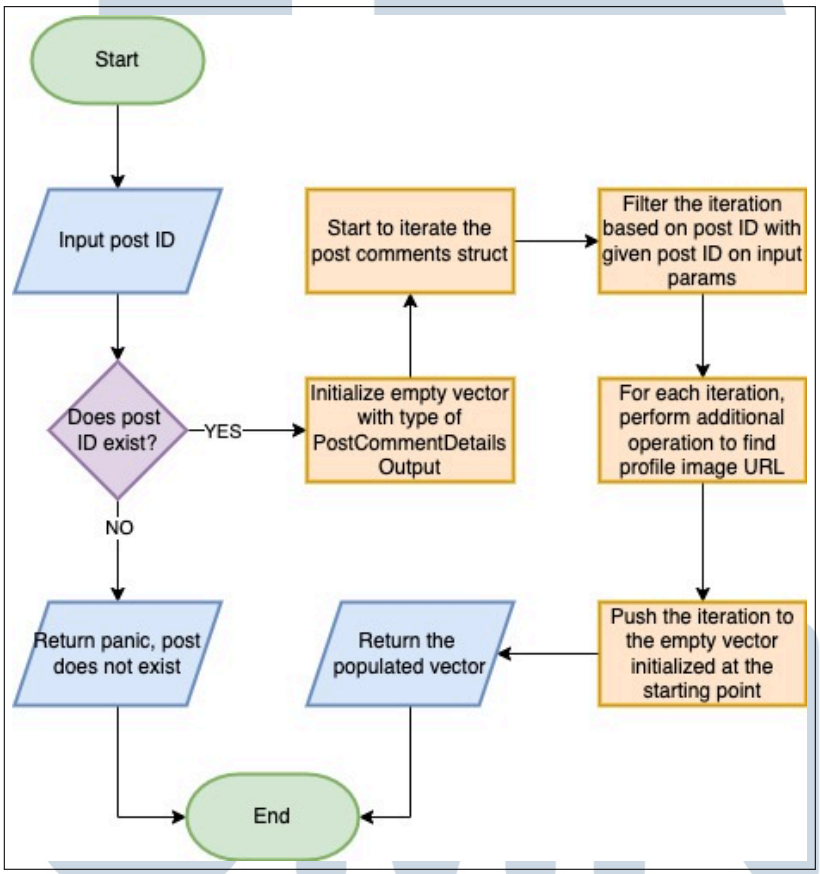


Gambar 3.8. Diagram Alir *get_post_likes_details*

G Diagram Alir *get_post_comment_details*

Fungsi *get_post_comment_details* digunakan untuk mendapatkan detail komentar pada suatu *post* berdasarkan ID post yang diberikan melalui parameter. Cara

kerja fungsi ini serupa dengan fungsi *get_post_likes_details*, hanya saja iterasi dan filtrasi dilakukan pada koleksi *post_comments*. Juga dilakukan pencarian gambar profil untuk setiap user yang memberikan komentar untuk meningkatkan *user experience* pada sisi *client*. Fungsi ini memberikan keluaran berupa *user address*, gambar profil, komentar yang diberikan, dan tanggal pembuatan komentar, sesuai dengan definisi skema *PostCommentDetailsOutput*. Diagram alir dari fungsi ini dapat dirujuk pada Gambar 3.9.



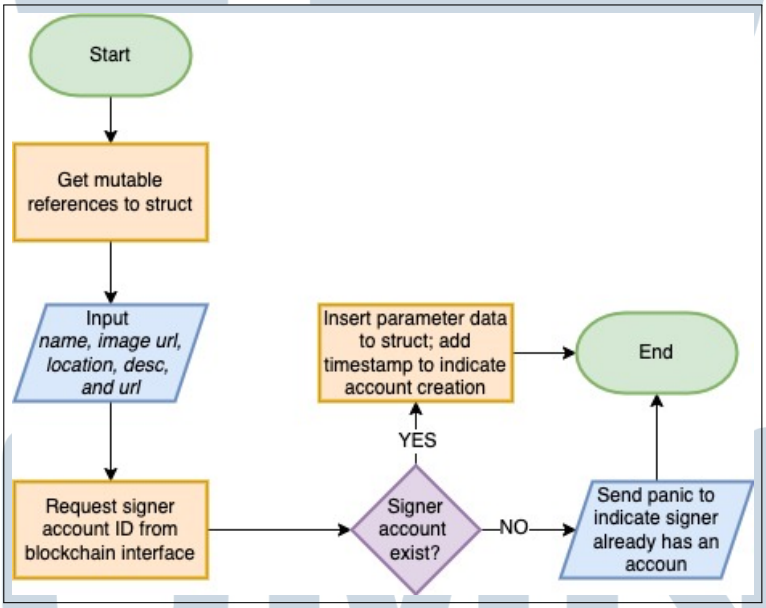
Gambar 3.9. Diagram Alir *get_post_comment_details*

H Diagram Alir *create_account*

Fungsi *create_account* digunakan untuk membuat suatu akun baru pada *contract* menggunakan *AccountId* yang dimiliki oleh pengguna melalui NEAR *Wallet*. Fungsi ini merupakan *mutable function* karena akan mengubah *state* pada *contract*, dan menerima 5 parameter opsional: *name, profile_image_url, location, url,* dan *description* yang semuanya memiliki tipe *String*. Alamat *address* akan didapatkan secara otomatis saat pengguna melakukan *sign transaction* saat pembuatan akun,

untuk memastikan bahwa hanya yang berkehendak dapat membuat akun sesuai dengan nama *address* yang dimiliki pada *wallet* pengguna.

Saat fungsi ini dipanggil, akan dilakukan pengecekan terlebih dahulu apakah suatu pengguna sudah terdaftar dengan melihat apakah *signer address* sudah terdapat pada koleksi *user_list*. *Panic* akan terjadi jika pengguna sudah terdaftar dengan *wallet* yang digunakan. Semua *user* akan disimpan pada struktur data *near_sdk::collections::LookupMap* yang ekuivalen dengan *HashMap* di struktur data pada umumnya, yang memiliki *signer* sebagai *key* dan value berupa nilai yang diambil dari parameter pemanggilan fungsi dan ditambahkan data pembuatan akun melalui *block_timestamp*. Fungsi ini tidak akan mengembalikan nilai apapun. Diagram alir dari fungsi ini dapat dirujuk melalui Gambar 3.10.

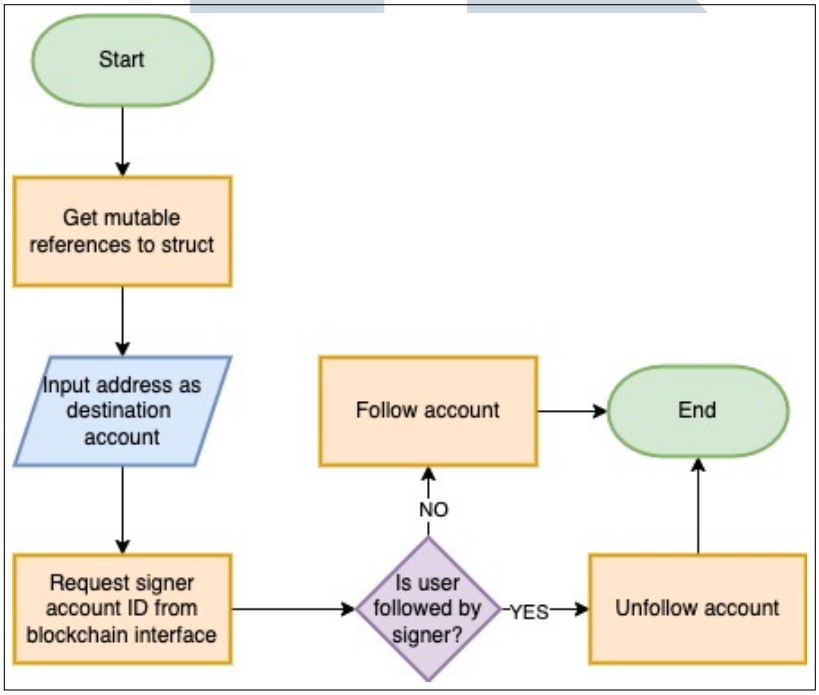


Gambar 3.10. Diagram Alir *create_account*

I Diagram Alir *follow_user*

Fungsi *follow_user* digunakan untuk mengikuti sebuah user dari sebuah user. Fungsi ini menerima satu parameter yaitu *address*, yang merupakan akun yang akan diikuti (*followed*). Akun yang akan mengikuti akun tujuan akan ditentukan dari *signer*, yang melakukan transaksi saat pemanggilan fungsi ini. Cara kerja dari fungsi ini akan mengecek terlebih dahulu apakah pengguna yang melakukan *sign* pada transaksi ini sudah mengikuti user yang nilainya diambil dari parameter. Jika belum mengikuti, maka *signer* akan mengikuti pengguna yang ditujukan dan akan

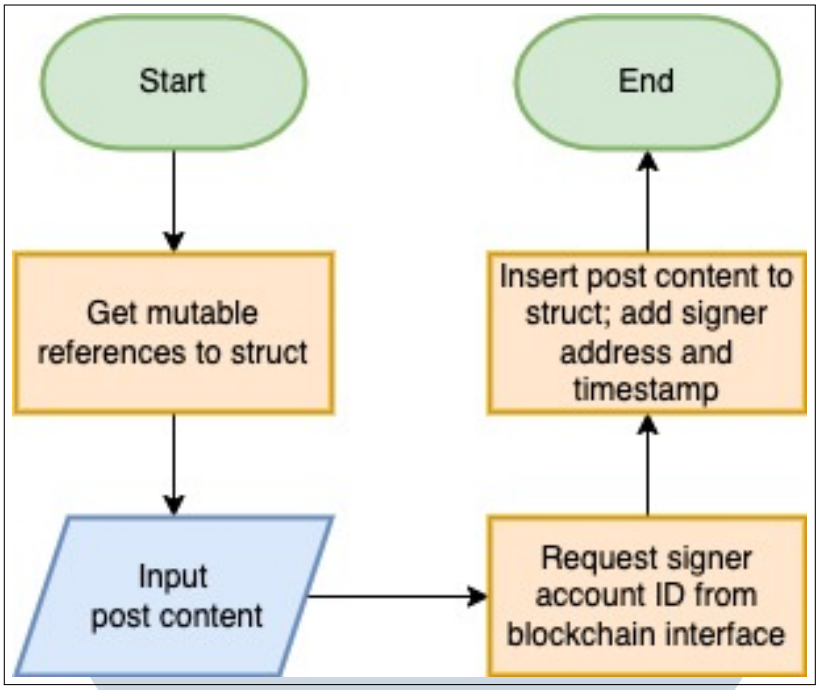
terjadi mutasi data pada koleksi *user_followers*. Jika *signer* telah mengikuti user yang ditujukan saat pemanggilan fungsi ini, maka akan terjadi operasi *unfollow*, yaitu batal mengikuti user tujuan. Diagram alir dari fungsi ini dapat dirujuk melalui Gambar 3.11.



Gambar 3.11. Diagram Alir *follow_user*

J Diagram Alir *create_post*

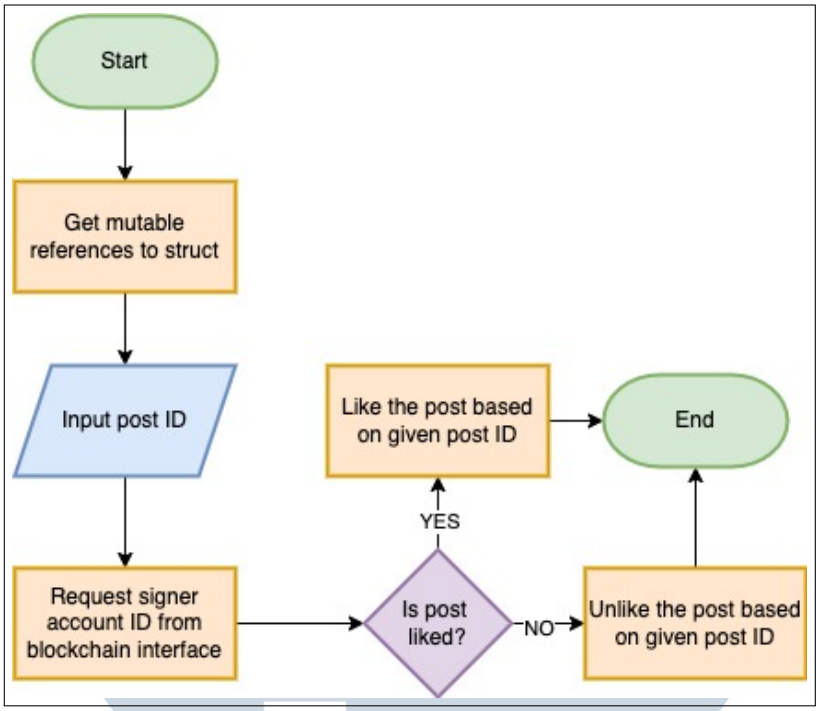
Fungsi *create_post* digunakan untuk membuat sebuah *post* baru pada *contract*. Fungsi ini menerima satu parameter yaitu *content*, yang merupakan isi *post* yang ingin dikirimkan oleh user. Serupa seperti fungsi-fungsi sebelumnya, *poster* akan ditentukan dari *transaction signer* untuk menghindari manipulasi pada pembuatan *post*, dan pemanggil fungsi ini harus memastikan bahwa ia memiliki akun pada *contract* media sosial terdesentralisasi. Operasi ini akan menambah *state post_counter* menjadi lebih besar 1 nilai. Diagram alir dari fungsi ini dapat dirujuk melalui Gambar 3.12.



Gambar 3.12. Diagram Alir *create_post*

K Diagram Alir *like_post*

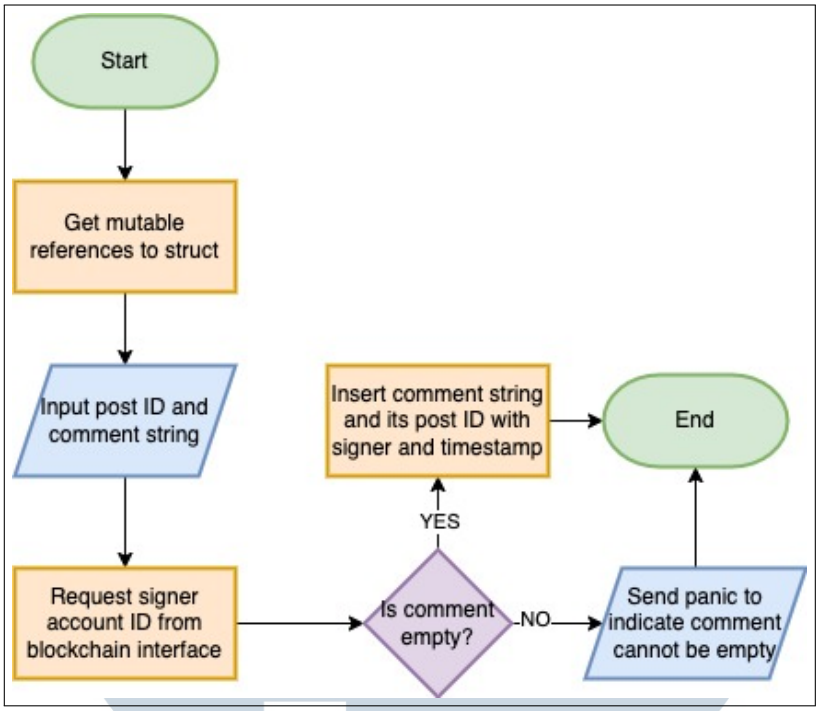
Fungsi *like_post* digunakan untuk menyukai sebuah *post* yang sudah ada, yang ditentukan dari ID post yang diberikan melalui parameter saat pemanggilan fungsi. User yang berhak menyukai *post* saat fungsi ini dipanggil ditentukan melalui *transaction signer* untuk menghindari manipulasi oleh pihak yang tidak memiliki otoritas terhadap akun. Fungsi ini juga memastikan bahwa *post* yang akan disukai harus tersedia pada koleksi yang akan diverifikasi melalui ID post. Cara kerja dari fungsi ini adalah dengan terlebih dahulu melakukan operasi apakah *signer* sudah menyukai *post* tujuan. Jika belum, maka fungsi ini akan menambahkan *signer* ke dalam daftar *likes post* tersebut melalui mutasi data pada koleksi *post_likes*. Apabila *signer* sudah menyukai *post* dan memanggil fungsi ini, maka akan terjadi operasi *unlike post*. Diagram alir dari fungsi ini dapat dirujuk melalui Gambar 3.13.



Gambar 3.13. Diagram Alir *like_post*

L Diagram Alir *comment_on_post*

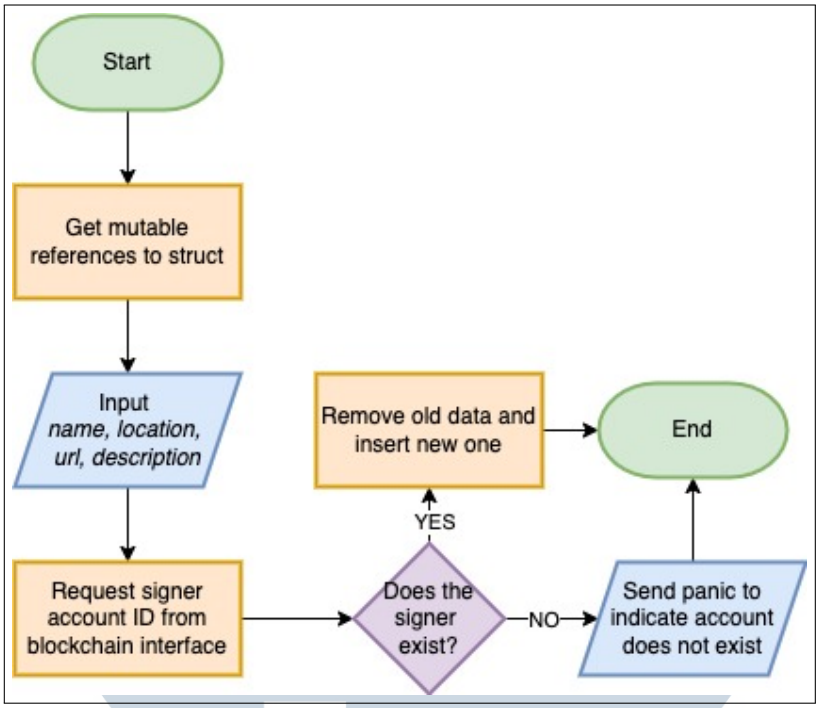
Fungsi *comment_on_post* digunakan untuk memberikan komentar pada *post* yang terkait melalui ID *post* yang diberikan melalui parameter. Selain menerima parameter ID *post*, fungsi ini menerima parameter lain yaitu *comment*, yang merupakan isi komentar yang ingin dikirimkan oleh pengguna. Serupa dengan fungsi *changeMethods* lainnya, user yang memberikan komentar akan ditentukan melalui *transaction signer* pada pemanggilan fungsi ini untuk menghindari manipulasi. Pemanggilan sukses terhadap fungsi ini akan menambahkan *state* pada *post_counter* menjadi lebih besar satu nilai. Diagram alir dari fungsi ini dapat dirujuk melalui Gambar 3.14.



Gambar 3.14. Diagram Alir *comment_on_post*

M Diagram Alir *edit_account_details*

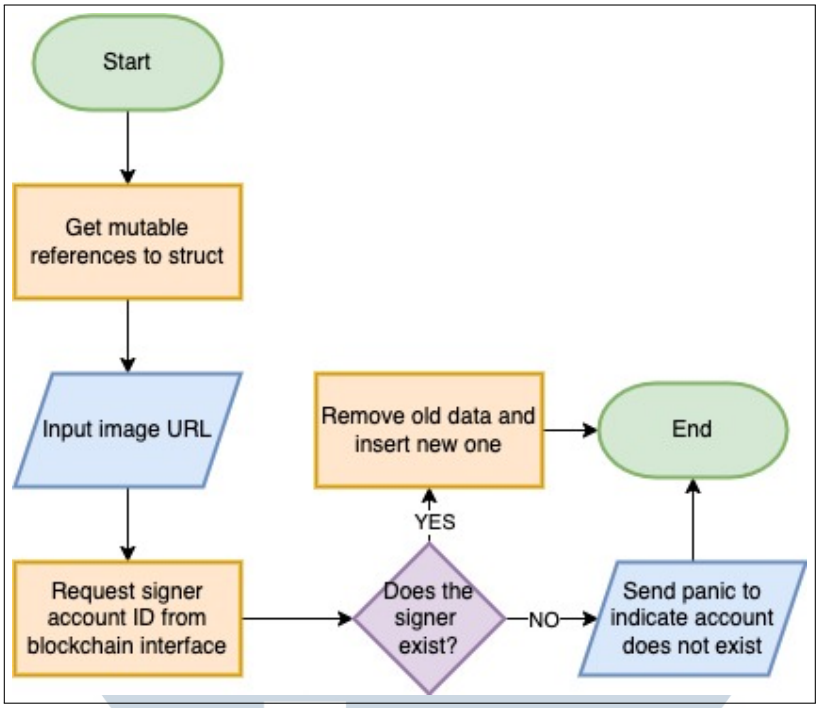
Fungsi *edit_account_details* digunakan untuk menyunting detail akun berdasarkan *signer* yang memanggil fungsi ini. Fungsi akan menerima 4 parameter yaitu *name, location, url,* dan *description*, di mana semua parameter menerima tipe *String*. Karena *LookupMap* tidak secara alami mendukung mutasi data, operasi yang dilakukan untuk menyunting akun profil adalah dengan cara menghapus *key* tujuan (di mana metode bawaan yang diberikan akan mengembalikan *value* dari *key LookupMap* yang dihapus) kemudian melakukan operasi *insert* ulang kepada koleksi. Diagram alir dari fungsi ini dapat dirujuk melalui Gambar 3.15.



Gambar 3.15. Diagram Alir *edit_account_details*

N Diagram Alir *edit_profile_image*

Fungsi *edit_profile_image* digunakan untuk menyunting gambar profil yang dimiliki oleh suatu akun berdasarkan *signer* yang memanggil fungsi. Fungsi ini hanya menerima satu parameter yaitu URL dari gambar yang sudah selesai terunggah pada IPFS. Sama seperti *edit_account_details*, operasi penyuntingan akan dilakukan dengan cara *remove key* dari *LookupMap* kemudian memasukkan data yang baru ke dalam koleksi. Diagram alir dari fungsi ini dapat dirujuk melalui Gambar 3.16.

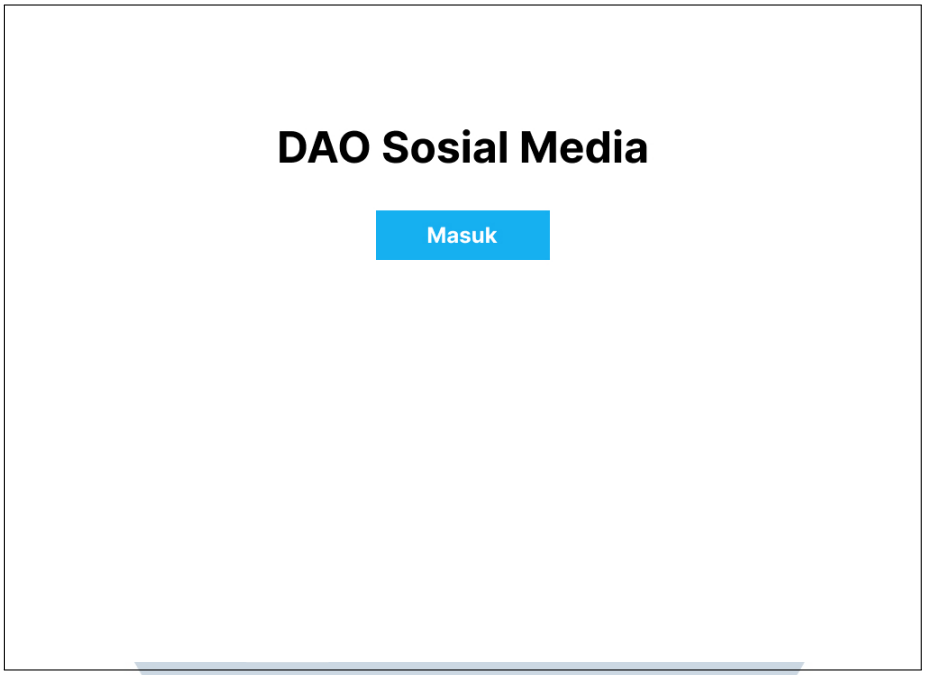


Gambar 3.16. Diagram Alir *edit_profile_image*

3.2.5 Perancangan Antarmuka

A Sketsa Antarmuka *Login*

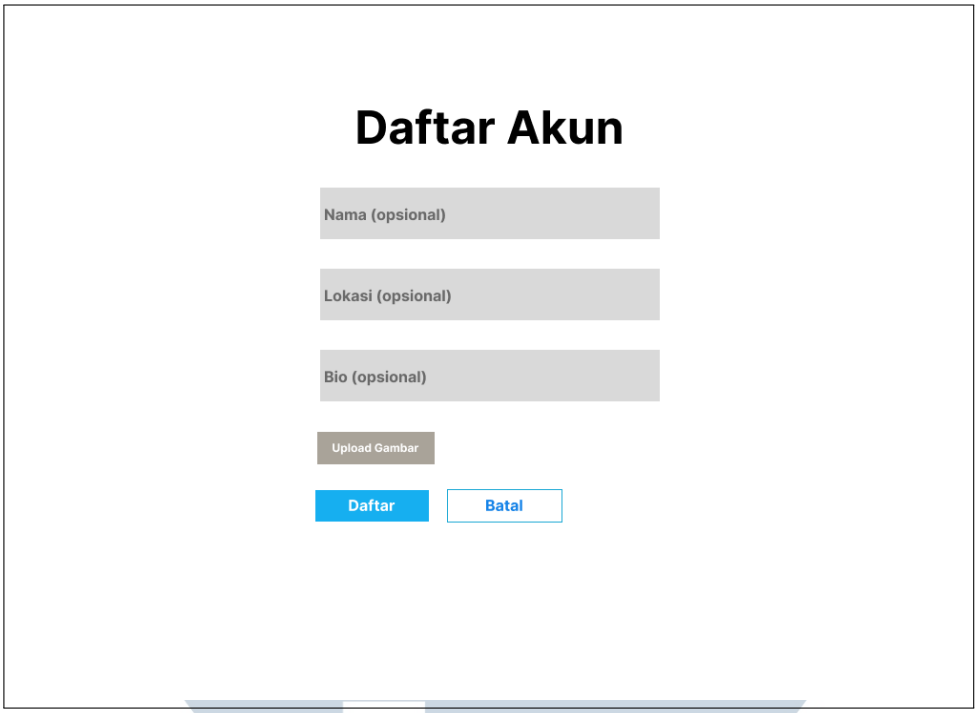
Halaman yang pertama kali muncul saat mengunjungi situs pertama kali adalah halaman *Login*. Pada halaman ini, pengguna akan diminta untuk melakukan proses masuk dengan menghubungkan NEAR *Wallet* yang dimiliki. Jika pengguna belum memiliki akun yang terdaftar pada *smart contract* saat menghubungkan dompet, maka akan dialihkan ke halaman *Register*, dan jika pengguna sudah memiliki akun terdaftar maka akan diarahkan ke halaman *Home*.



Gambar 3.17. Sketsa Antarmuka *Login*

B Sketsa Antarmuka *Register*

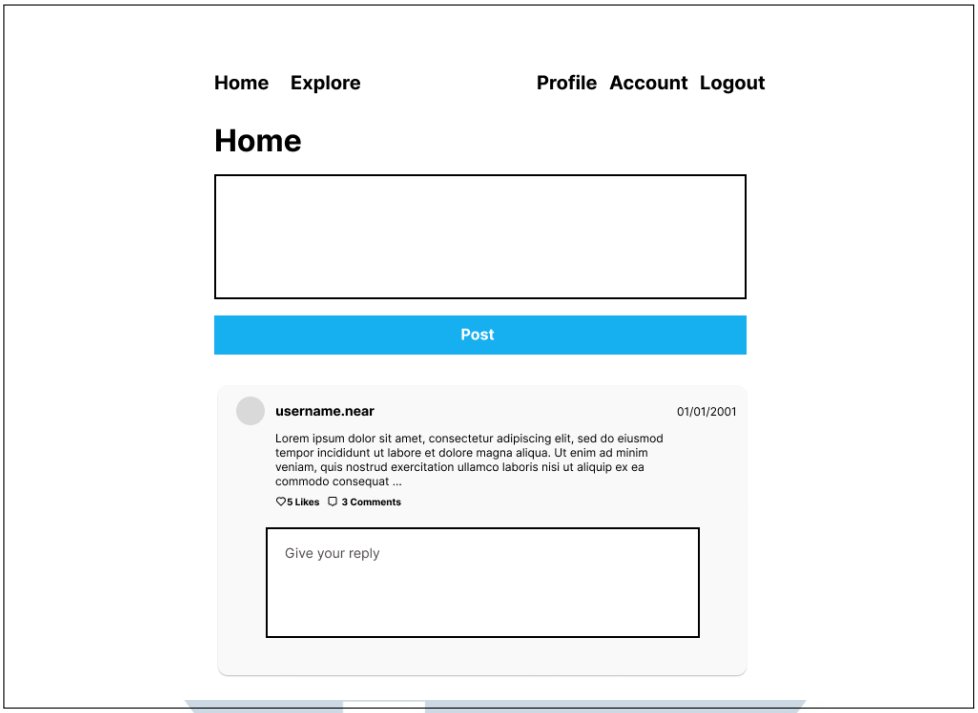
Halaman ini akan dimunculkan ke pengguna jika mereka tidak memiliki akun saat menghubungkan dompet yang dimiliki pada proses *login*. Saat proses registrasi berlangsung, pengguna dapat memberikan 3 data yang bersifat opsional, yaitu nama, lokasi, dan bio mereka. Selain itu, pengguna dapat mengunggah gambar sebagai foto profil mereka jika diinginkan.



Gambar 3.18. Sketsa Antarmuka *Register*

C Sketsa Antarmuka *Home*

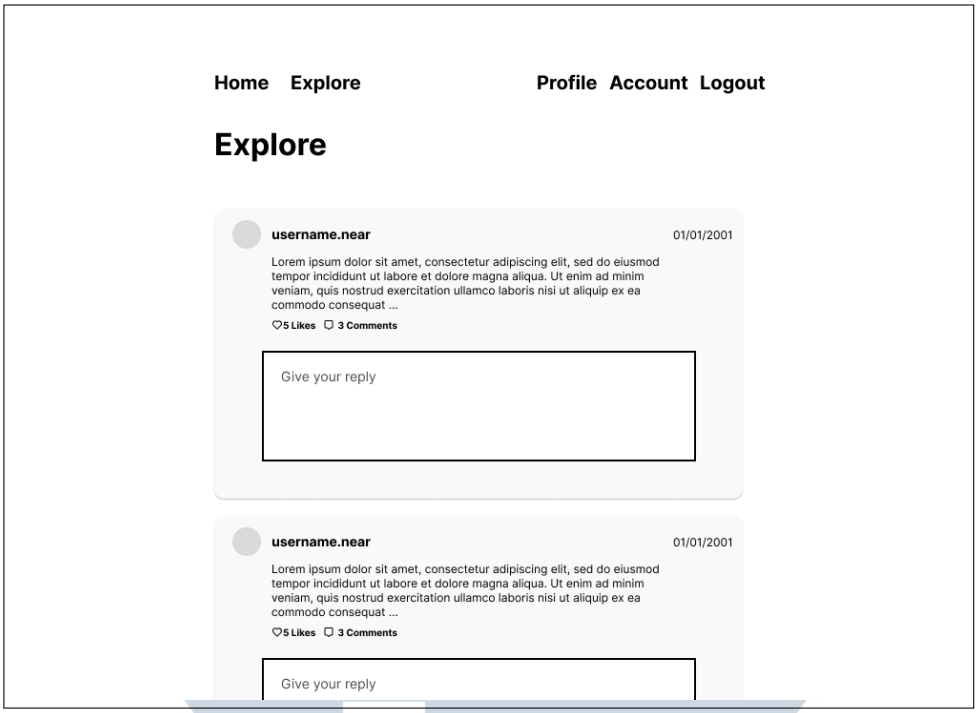
Saat pengguna sudah memiliki akun di dalam *smart contract*, maka pengguna memiliki akses untuk memasuki halaman *Home* setelah pengguna menghubungkan dompetnya di halaman *Login*. Halaman ini memiliki fitur untuk membuat *post* baru sesuai dengan pengguna yang masuk dan melihat *post-post* yang dibuat oleh pengguna lain yang diikuti oleh pengguna tersebut. Selain itu, pengguna juga dapat menyukai dan memberikan komentar pada *post* yang tersedia, termasuk melihat daftar *likes* dan komentar-komentar yang diberikan pengguna lain.



Gambar 3.19. Sketsa Antarmuka *Home*

D Sketsa Antarmuka *Explore*

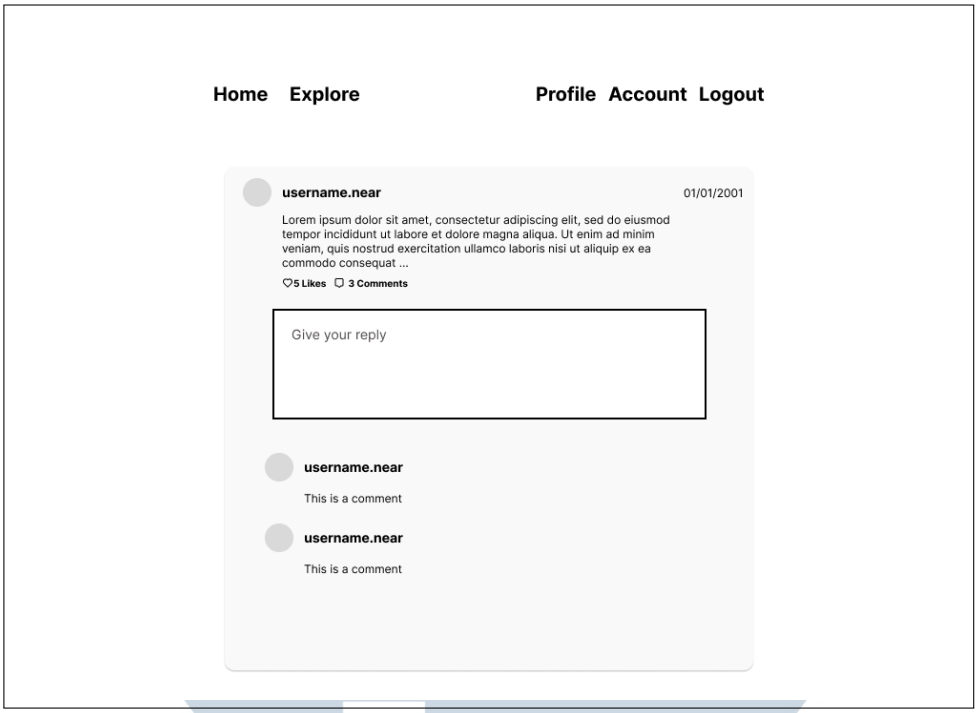
Jika halaman *Home* memberikan *post-post* yang dibuat oleh akun yang diikuti pengguna, maka halaman *Explore* memiliki fungsi untuk melakukan ”eksplorasi” — melihat *post* dari akun yang tidak diikuti pengguna, dan dapat digunakan untuk melihat aktivitas akun lain. Pengguna juga dapat menyukai dan memberikan komentar pada laman ini.



Gambar 3.20. Sketsa Antarmuka *Explore*

E Sketsa Antarmuka *Post Detail*

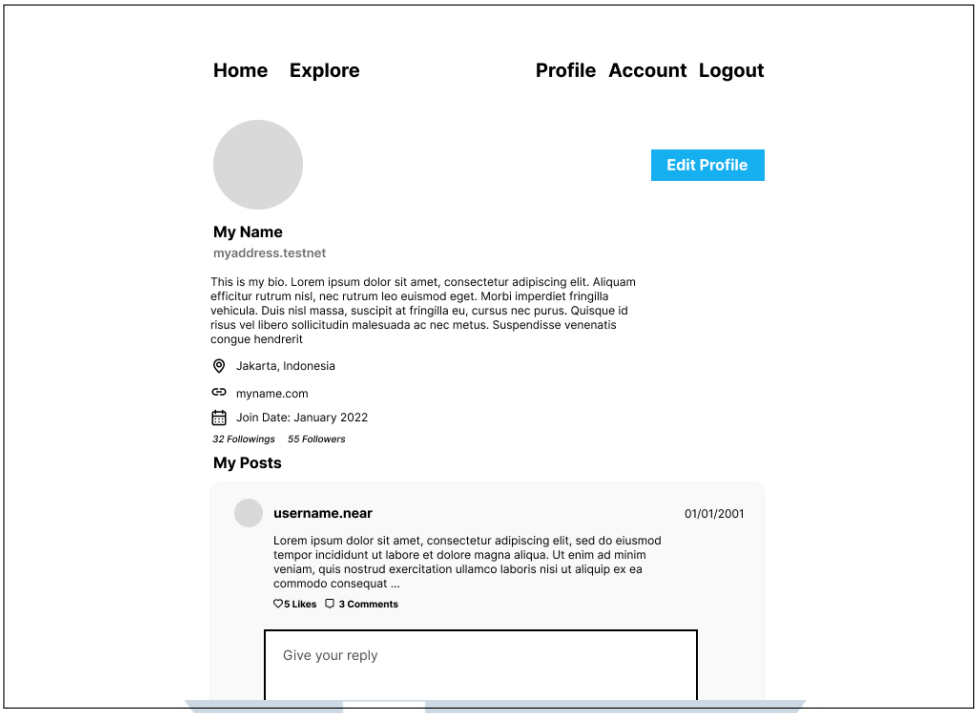
Setiap *post* yang muncul di halaman *Home* atau *Explore* memiliki kemampuan untuk diklik pada *card*, yang akan masuk ke halaman *Post Detail* berdasarkan ID *post* tersebut. Halaman ini digunakan sebagai halaman tunggal untuk detail *post*, dan juga dapat digunakan sehingga tautan dari laman ini dapat dibagikan ke orang lain untuk menunjukkan ke post tertentu. Pengguna dapat menyukai, memberikan komentar, serta melihat detail dari *likes* dan komentar yang diberikan dari pengguna lain. Jika pengguna melakukan klik pada *address* pembuat *post*, maka akan diarahkan ke profil pembuat *post* tersebut.



Gambar 3.21. Sketsa Antarmuka *Post Detail*

F Sketsa Antarmuka *Profile*

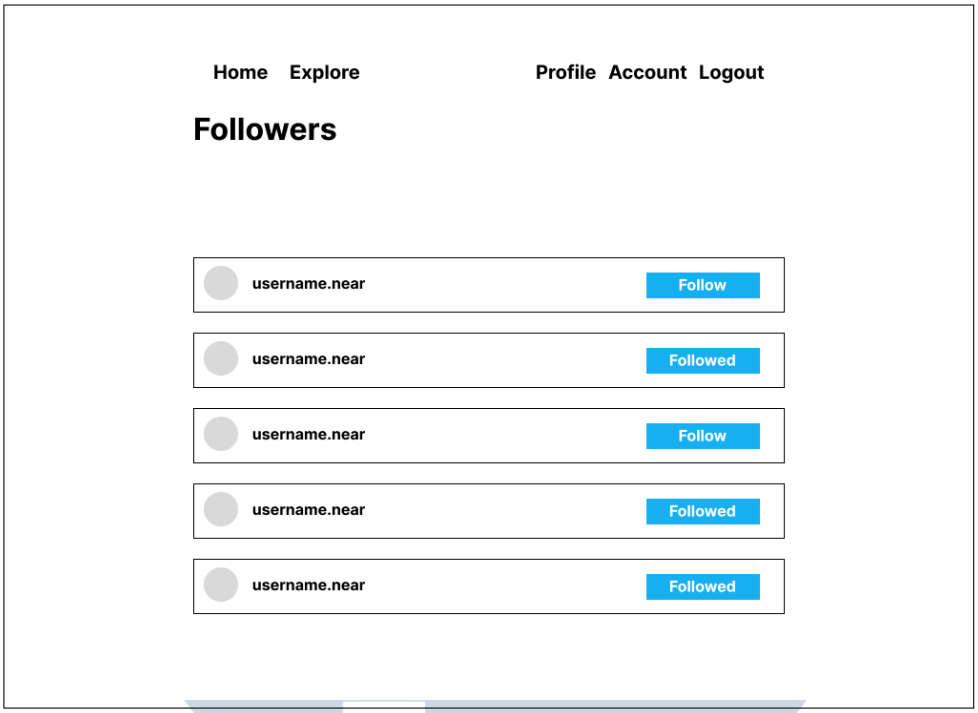
Halaman *profile* ditujukan untuk memberikan detail dari profil sebuah pengguna, dari nama yang dimiliki, bio, lokasi, dan web URL. Pada halaman ini, pengguna juga dapat memilih untuk mengikuti atau batal mengikuti sebuah pengguna, dan melihat daftar *followers/following* yang dimiliki oleh suatu akun. Selain itu, juga terdapat kompilasi terhadap *post* yang dibuat oleh akun profil tujuan.



Gambar 3.22. Sketsa Antarmuka *Profile*

G Sketsa Antarmuka *Followers/Following*

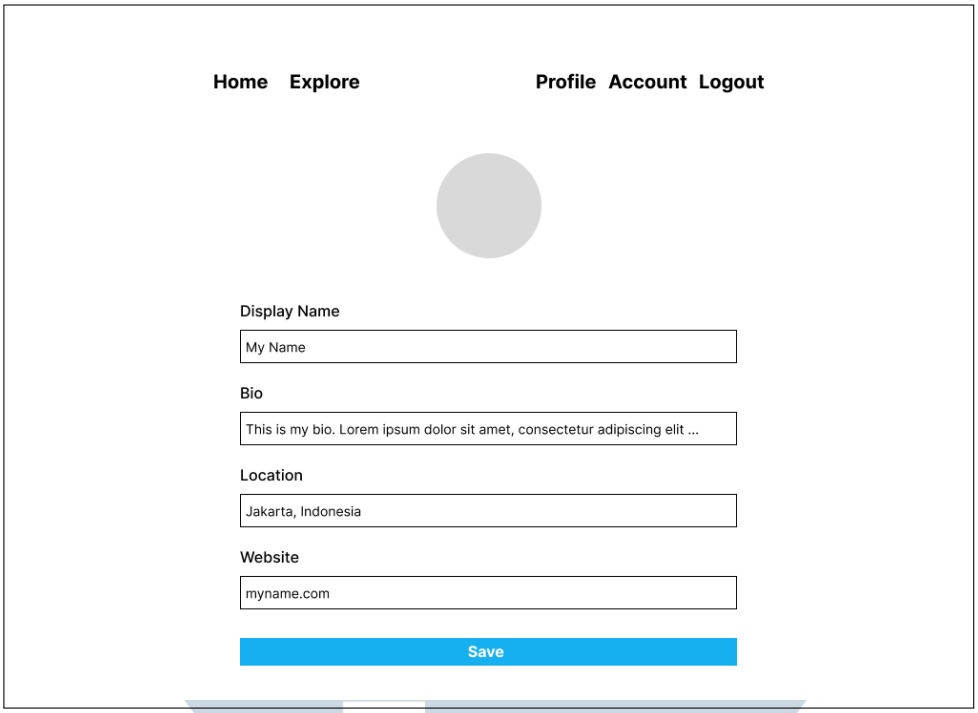
Halaman ini digunakan untuk melihat daftar pengikuti atau akun yang diikuti oleh pengguna, dan juga bisa melakukan aksi mengikuti/batal mengikuti dari daftar akun yang tersedia.



Gambar 3.23. Sketsa Antarmuka *Followers/Following*

H Sketsa Antarmuka *Account*

Halaman ini digunakan untuk menyunting data akun pengguna yang sudah terdaftar, dan hanya bisa diakses oleh pengguna yang masuk menggunakan dompet NEAR. Keseluruhan data yang dimiliki dapat diubah kecuali *wallet address*, termasuk foto profil.



Gambar 3.24. Sketsa Antarmuka *Account*