BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1 Metode Penelitian

Dalam penelitian ini, metode penelitian yang dilakukan peneliti adalah dengan studi pustaka, perancangan, implementasi, dan pengujian. Studi pustaka merupakan tahapan dimana peneliti mencari referensi mengenai masalah yang akan diteliti penulis. Tahap perancangan adalah tahap dimana peneliti merancang sistem untuk menyelesaikan masalah penelitian. Tahapan implementasi adalah tahapan dimana peneliti melakukan implementasi terhadap sistem yang sudah dirancang dan tahapan pengujian adalah tahapan dimana sistem yang telah diimplementasi kemudian diuji.

Studi pustaka yang dilakukan penulis adalah dengan membaca dan mempelajari *research paper* dan penelitian yang telah dilakukan mengenai sistem akses kontrol elektronik, *face recognition*, *blockchain* dan penggunaan *QR Code*. Penulis juga melakukan studi Pustaka melalui website, buku dan video pada website Youtube. Penulis juga melakukan studi Pustaka dengan mempelajari bahasa Python, Javascript dan Solidity yang diperlukan untuk implementasi dari sistem. Perancangan yang dilakukan penulis akan dibahas pada sub-bab pada bab III. Implementasi dari sistem serta hasil pengujian dari sistem yang dirancang akan dibahas pada bab IV.

3.2 Perancangan Modul

Pada penelitian ini, modul dari sistem dibagi menjadi 3 bagian antara lain adalah: Frontend, Blockchain dan Node.

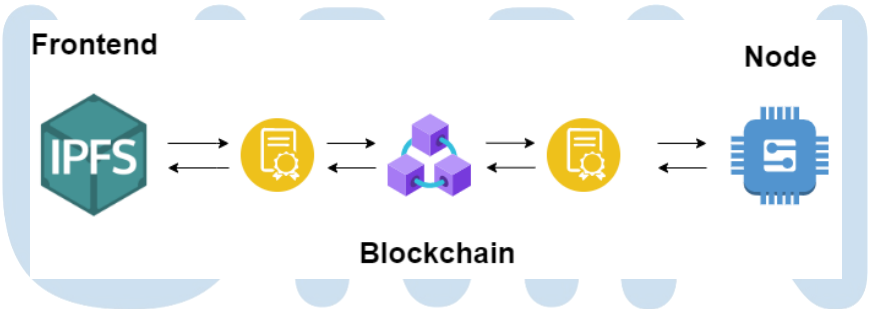
Frontend merupakan website yang menjadi titik interaksi antara pengguna dengan blockchain. Rancangan sistem Frontend akan dibahas dalam perancangan aplikasi. Frontend akan berada pada IPFS, sehingga website juga menjadi bagian

dari Blockchain. Frontend akan dibuat menggunakan bahasa pemrograman Javascript dengan framework Vue.

Blockchain merupakan jaringan yang akan menjadi *database* dari sistem. Blockchain yang digunakan adalah *blockchain* Ethereum, dan akan menggunakan konsensus *Proof of Works*. Rancangan dari sistem *blockchain* akan dibahas pada sub-bab rancangan *Blockchain*. Jaringan *blockchain* yang akan digunakan pada penelitian adalah *Private Blockchain*, dimana node dari *Blockchain* akan disimulasikan dengan menggunakan *virtual machine*, yang menjalankan program Geth dan IPFS.

Node merupakan sistem yang akan dikontrol dari *blockchain*. Sistem Node merupakan perangkat IoT yang. Node terdiri dari Raspberry Pi, Pi Camera, LCD *Screen* dan *Magnetic Lock*. Node akan menjalankan sistem operasi Raspberry Pi OS dan menjalankan aplikasi berbasis Python. Rancangan sistem Node akan dibahas pada sub-bab rancangan Node.

Untuk berinteraksi dengan *blockchain*, Frontend dan Node akan menggunakan *smart contract*. *Smart contract* akan dibuat menggunakan bahasa pemrograman Solidity. Rancangan dari *smart contract* akan dibahas pada sub-bab rancangan *Blockchain*.

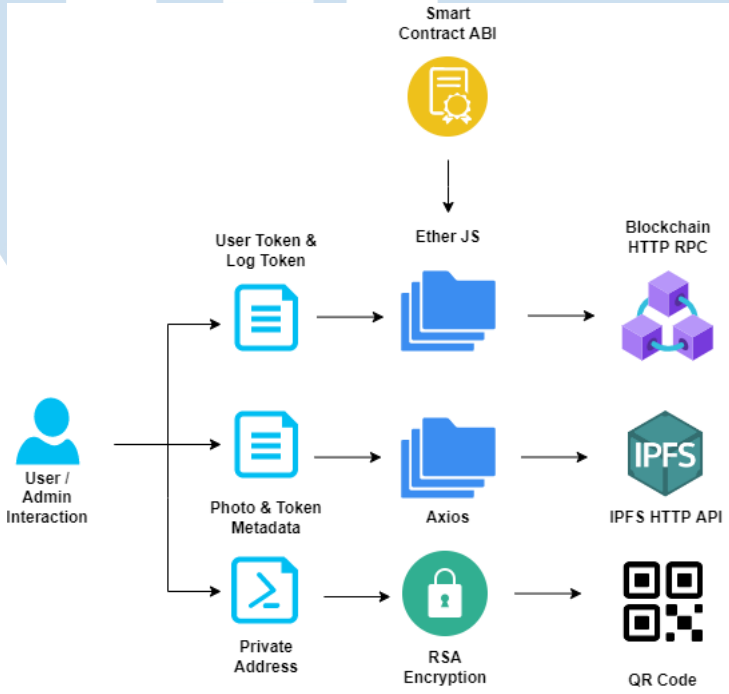


Gambar 3.1 Rancangan Modul

3.3 Rancangan Aplikasi

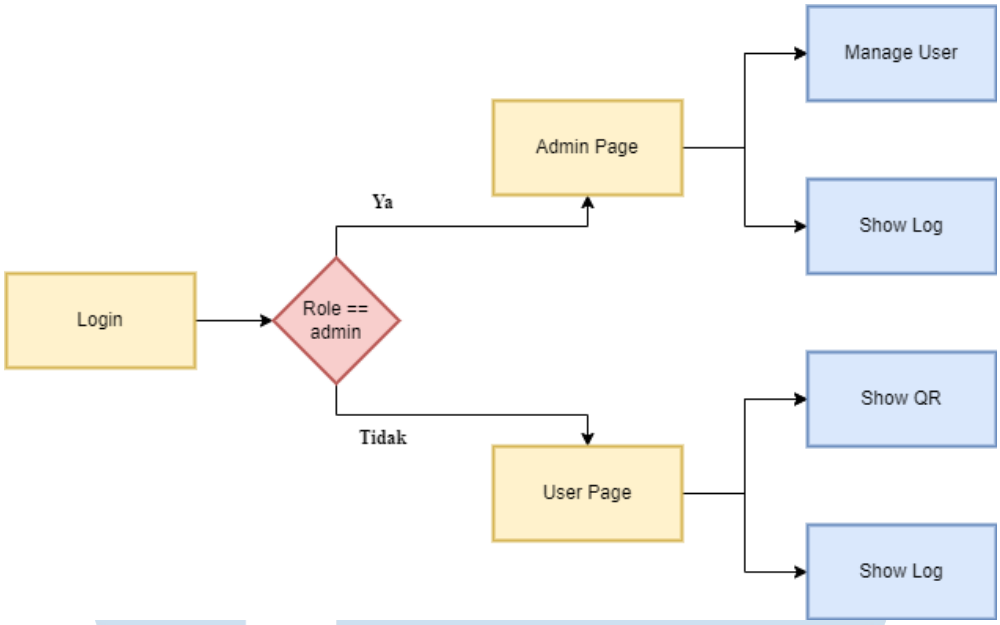
Rancangan Aplikasi dari sistem terdapat pada modul Frontend. Aplikasi akan dibuat menggunakan bahasa pemrograman Javascript dengan *framework* Vue.

Untuk membuat aplikasi menjadi *Single Page Application*, maka digunakan *library* Vue-Router. Untuk berkomunikasi dengan *blockchain*, akan digunakan library ethers, dan untuk berkomunikasi dengan IPFS akan menggunakan *library* Axios yang digunakan untuk memanggil API REST pada IPFS. *Library* Axios juga digunakan untuk memanggil *endpoint* HTTP Fluentd yang digunakan untuk *logging*. *Library* node-forge akan digunakan untuk melakukan enkripsi RSA yang akan digunakan pada pembentukan *QR Code*. Arsitektur dari sistem Frontend dapat dilihat pada gambar 3.2



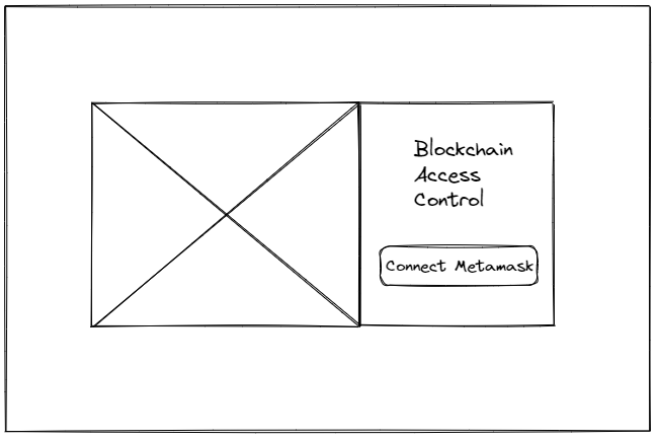
Gambar 3.2 Arsitektur sistem Frontend

Aplikasi dapat dibagi menjadi 2 bagian, yaitu admin dan pengguna. Pada bagian admin, admin dapat mengelola pengguna serta membaca log hasil akses. Sedangkan pengguna dapat mendapatkan *QR Code* yang akan digunakan sebagai kredensial untuk memasuki ruangan yang akan dipindai oleh Node. *Flow* dari Frontend dapat dilihat pada gambar 3.3



Gambar 3.3 *Flow* Aplikasi Frontend

Ketika membuka website Frontend, pengguna akan ditampilkan laman *login*, untuk melakukan *login*, maka pengguna tidak menggunakan *password* dan *username*, melainkan menggunakan Metamask yang merupakan aplikasi *extension* yang dapat ditambahkan pada *browser*. Metamask digunakan karena, untuk melakukan *login* dan untuk melakukan semua fungsi pada *blockchain*, dibutuhkan *address* dan *private key* dari akun, aplikasi Metamask berisi akun dan *private key* yang terdapat pada *blockchain*. Laman *login* akan memiliki desain seperti pada gambar 3.4

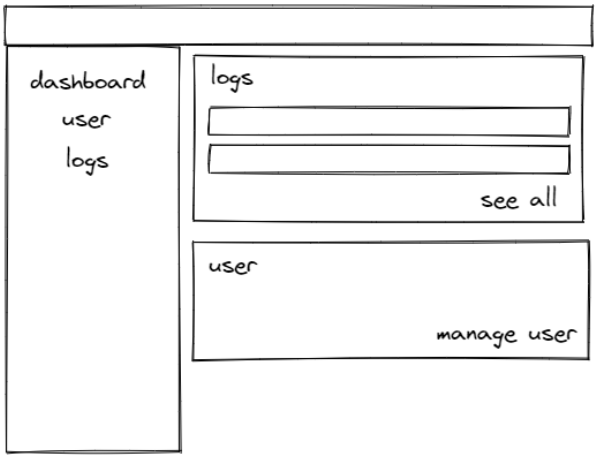


Gambar 3.4 Desain laman login

Setelah pengguna masuk, maka sistem akan melakukan pengecekan, apakah pengguna merupakan admin atau bukan, hal ini dapat dilakukan dengan mengecek fungsi yang terdapat pada *smart contract*, jika pengguna adalah *admin* maka pengguna akan ditujukan ke laman *admin*, sedangkan jika pengguna adalah *user* maka akan ditujukan ke laman *user*. Hasil dari pengecekan dari *smart contract*, kemudian akan disimpan kedalam *cookie* dengan nama *role* dari *browser*, namun dimana sebelum disimpan, akan di *encrypt* dengan menggunakan algoritma AES, dengan *key* berupa *address* dari pengguna yang melakukan *login.*

Pada laman *admin*, laman terdiri dari *navigation bar*, bagian *log* dan *bagian user*. Bagian log akan berisi *log-log* yang paling baru masuk kedalam *blockchain*, bagian *user* akan berisi mengenai jumlah *user*. Pada setiap bagian, akan terdapat tombol yang dapat membawa ke laman lainnya. Pada bagian *admin*, tombol "*see all"* akan membawa admin ke laman *log*, sedangkan pada bagian *user* tombol "*manage user"* akan membawa ke laman *manage user*. Pada *navigation bar*, akan berisi *menu* laman, yaitu *dashboard*, *manage user*, *logs*. *Navigation bar* akan berubah berdasarkan *cookies role* yang terdapat pada browser, jika hasil *decrypt cookies* menggunakan key *address* pengguna, menunjukan hasil berupa *admin* maka *navigation bar* akan menampilkan menu *admin*, namun jika hasil menunjukan hasil berupa *user* maka *navigation bar* akan menampilkan menu *user*, yang berisi *dashboard user* dan *log*. *Menu dashboard* akan membawa pengguna menuju laman awal *admin*, menu *manage user* akan membawa pengguna menuju laman *manage user*, dan menu *logs* akan membawa pengguna menuju laman *logs*. Desain dari laman *admin* dapat dilihat pada gambar 3.5

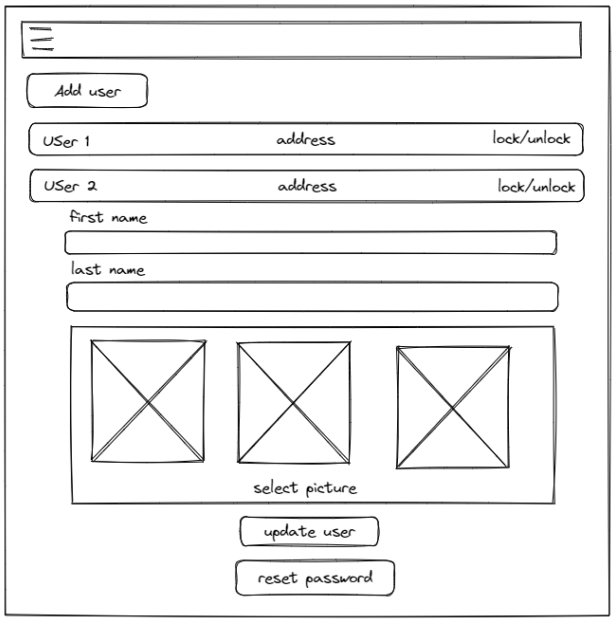
UNIVERSITAS
MULTIMEDIA
NUSANTARA



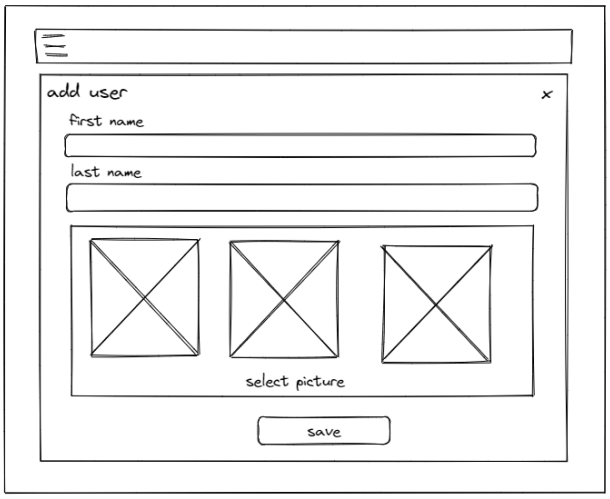
Gambar 3.5 Desain laman *admin*

Laman *manage user* berisi *list* yang menampilkan semua user yang terdaftar. Pada setiap *item*, berisi nama *user*, *address user* pada *blockchain* dan tombol untuk menonaktifkan atau mengaktifkan *user*. Setiap *item* pada *list*, didapatkan dengan memanggil *smart contract*. Jika salah satu *item* di-klik, maka akan menampilkan *dropdown* dari *user*. *Dropdown* ini dapat digunakan untuk mengubah nama *user*, dan menambahkan foto *user*, selain itu juga ada tombol untuk melakukan *reset* terhadap *password user*.Untuk melakukan *reset password user* maka akan memanggil *smart contract* untuk mendapatkan *hash* dari *default keystore* yang akan diambil menggunakan IPFS. Pada laman *manage user* juga terdapat tombol *add user*, yang akan menampilkan *modal add user*. *Modal add user* berfungsi untuk menambahkan *user* kedalam *blockchain*. *Modal add user* berisi *input field* untuk nama depan dan nama belakang *user*, *sebuah dropzone* untuk menambahkan foto *user* serta tombol untuk menyimpan user dalam *blockchain*. *Dropzone* dari *foto* dapat di-klik untuk mencari foto pada perangkat atau dapat dilakukan *drag* foto ke dalam *dropzone* untuk menambahkan foto. Setelah *user* tersimpan, maka *modal* akan tertutup dan kembali ke laman *manage user*. Untuk mendapatkan *address* dan *private Key* akun baru, maka akan digunakan fungsi *library* Etherjs, yang akan menghasilkan *address* dan *private key* random. *Private key* kemudian juga dibuat menjadi sebuah *keystore*, dimana *keystore* akan diberikan *password* berupa hasil hash dari nama depan dan nama belakang pengguna baru. *Keystore* ini kemudian akan dikirimkan ke IPFS dengan menggunakan *library* Axios. Untuk menyimpan

user, akan digunakan fungsi yang memanggil library IPFS yang digunakan untuk menambahkan file data *user*. Struktur file data *user* akan dibahas pada perancangan Blokchain. Setiap foto dari pengguna baru, akan dikirimkan ke IPFS, dimana setiap foto akan mendapatkan *hash* yang kemudian akan dimasukan kedalam struktur data *user*. Selain itu, admin juga akan memberikan Ethereum sebanyak 10 ETH kepada *user*. Untuk menyimpan pengguna baru, maka sistem akan memanggil *smart contract.* Desain dari laman *manage user* dan *add user* terdapat pada gambar 3.6 dan gambar 3.7



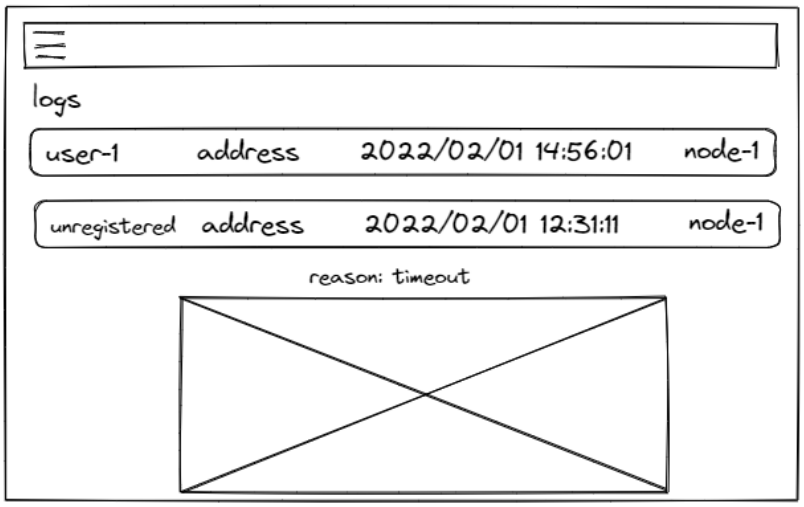
Gambar 3.6 Laman *manage user*



Gambar 3.7 Laman *add user*

Laman *log* akan berisi *list* dari *log blockchain.* Sama seperti pada laman *user*, maka untuk mendapatkan *item* untuk *list*, maka sistem akan memanggil *smart contract.* Setiap *item* dari *log* berisi nama *user* yang melakukan akses, *address user* yang melakukan akses pada *blockchain*, waktu *user* yang melakukan akses dan lokasi dari *node.* Setiap *item*, akan memiliki warna yang berbeda-beda sesuai dari status yang terdapat pada *log*, jika status *log* merupakan *user* yang tidak terdaftar, *timeout*, ataupun wajah atau *address* yang berbeda dengan apa yang didapati oleh *node* dari pemindaian *QR Code* ataupun pemindaian wajah, maka warna *item* akan menjadi warna merah, sedangkan jika *user* merupakan *user* yang telah dinonaktifkan, maka *item* akan menjadi warna kuning, terakhir, jika *user* merupakan *user* yang memenuhi semua kebutuhan, maka *item* akan menjadi warna hijau. Setiap *item* dapat di-klik, dan akan menunjukan foto yang diambil pada saat *user* melakukan proses akses. Foto tersebut akan diambil dengan menggunakan fungsi dari *library* Axios. Alasan penolakan akses juga akan terlihat di atas foto pada *item* yang berwarna kuning atau merah. Desain dari laman *log* terdapat pada gambar 3.8

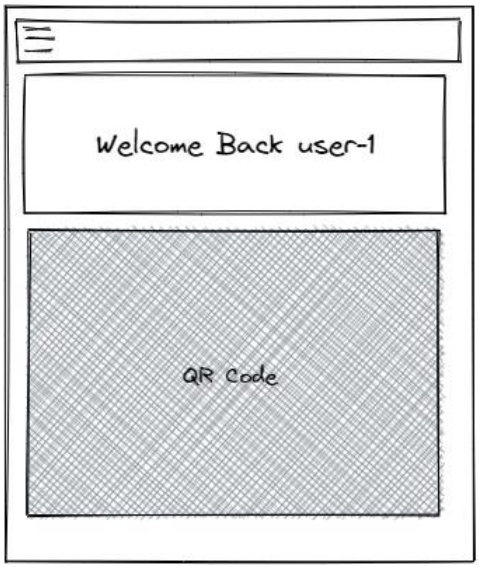
UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.8 Laman *log*

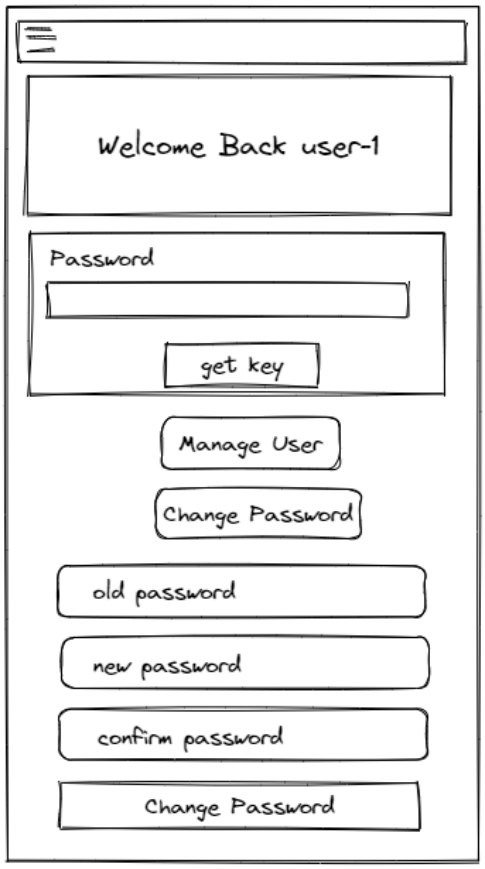
Pada laman *user*, dimana ketika *user* melakukan login akan dituju. Akan menampilkan bagian password, yang dapat digunakan pengguna untuk mendapatkan *QR Code*, yang kemudian digunakan untuk mengidentifikasi diri pada node. Selain itu juga terdapat tombol yang digunakan untuk mengganti *password* yang digunakan untuk mendapatkan *QR Code*. Sebuah QR Code, berisi *private key* dari pengguna, yang di encrypt dengan menggunakan enkripsi RSA 1024 Bit. Untuk melakukan enkripsi, akan digunakan *public key* dari RSA tersebut. Untuk mendapatkan *QR Code*, sistem akan memanggil *smart contract* yang digunakan untuk mengambil *keystore* yang terdapat pada struktur data *user*, kemudian sistem akan mengambil *private key* dari *user* tersebut, serta melakukan enkripsi dengan menggunakan RSA. Untuk mengganti *password* dibutuhkan *password* yang digunakan sebelumnya, *password* baru dan konfirmasi *password* yang baru. Tombol *manage user*, akan menampilkan tombol *change password*, ketika diklik, yang akan menampilkan bagian yang digunakan untuk mengganti *password* sebelumnya. Untuk mengganti *password* sistem akan memanggil *smart contract*, kemudian akan mengubah *keystore* pada struktur data *user*. *Keystore* baru didapatkan dengan cara yang sama dengan pembuatan *keystore* pada laman *add user*. Laman *user* akan didesain *mobile-first* karena laman *user* akan lebih banyak dipakai pada *smartphone*. Desain dari laman *user* terdapat pada gambar 3.9, 3.10 dan 3.11.

Gambar 3.9 Laman *user*



Gambar 3.10 Laman *user* saat menunjukan *QR Code*

UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.11 Laman *user* untuk mengganti *password*

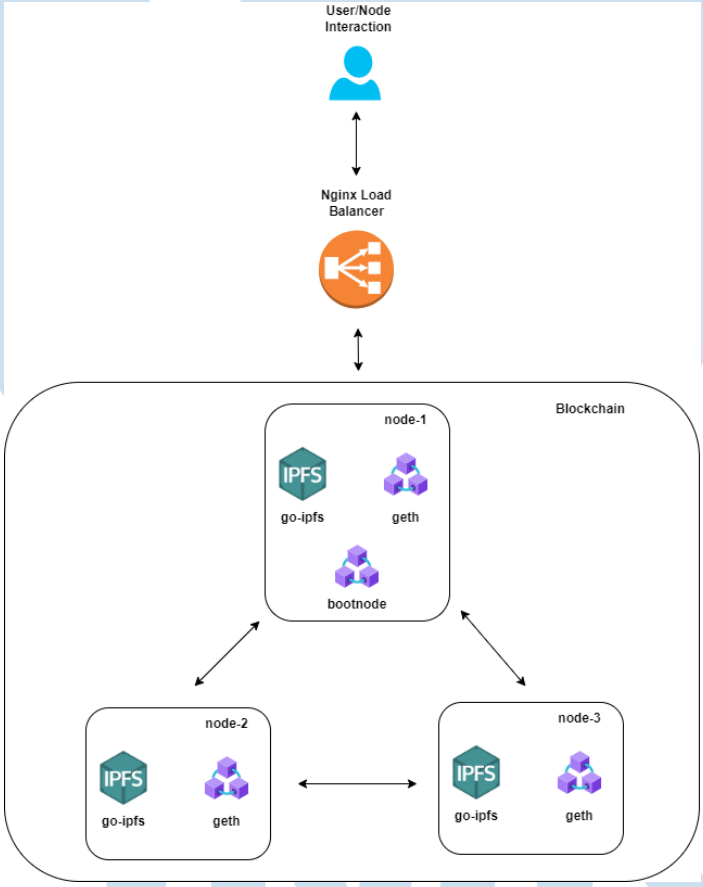
Laman *log* pada *user* akan memiliki desain yang sama dengan laman *admin*, namun, log yang akan ditampilkan hanya *log* yang memiliki *address* dari pengguna yang masuk kedalam aplikasi.

Bagian Frontend akan di-*hosting* pada IPFS, sehingga sistem menjadi terdistribusi, sehingga jika salah satu *blockchain node down*, maka sistem frontend akan tetap berjalan.

3.4 Rancangan Blockchain

Sistem Blockchain akan dibuat menggunakan arsitektur *blokchain* Ethereum. Ethereum dipilih karena merupakan salah satu arsitektur *blockchain* yang paling banyak digunakan pada penelitian, mendukung *private* dan *public blockchain*. Untuk menjalankan *blockchain*, akan digunakan program geth yang merupakan

program berbasis bahasa Go, yang digunakan untuk menjalankan *node* Ethereum. Ethereum yang digunakan pada penelitian ini adalah *private network*, dimana satu *node* dengan *node* lainnya hanya dapat mengetahui satu dengan lainnya dengan memasukan secara manual *enode* masing-masing. Jaringan Ethereum bekerja secara *peer-to-peer* sehingga membentuk sebuah network. Arsitektur dari sistem Blockchain dapat dilihat pada gambar 3.12



Gambar 3.12 Arsitektur sistem Blockchain

Pada penelitian ini, jaringan dibentuk dengan menggunakan 3 *node*. *Node* ini akan dijalankan pada *virtual machine* (VM). Setiap VM akan dialokasikan berupa 2 prosesor, RAM sebesar 2GB, dan 20GB Storage. Sistem operasi yang akan digunakan adalah Linux Lubuntu 20.04, yang merupakan versi ringan dari Ubuntu 20.04. Lubuntu digunakan karena merupakan salah satu sistem operasi yang banyak digunakan pada *server*, serta lebih mendukung banyak program daripada sistem operasi *windows*. Untuk melakukan supervisi terhadap VM, maka digunakan

aplikasi berupa VMware Workstation Pro 15. VM akan dijalankan pada sistem dengan spesifikasi berupa Intel Core i7-8750H dengan 6 Core, RAM 16GB dan Storage berupa SSD 512GB.

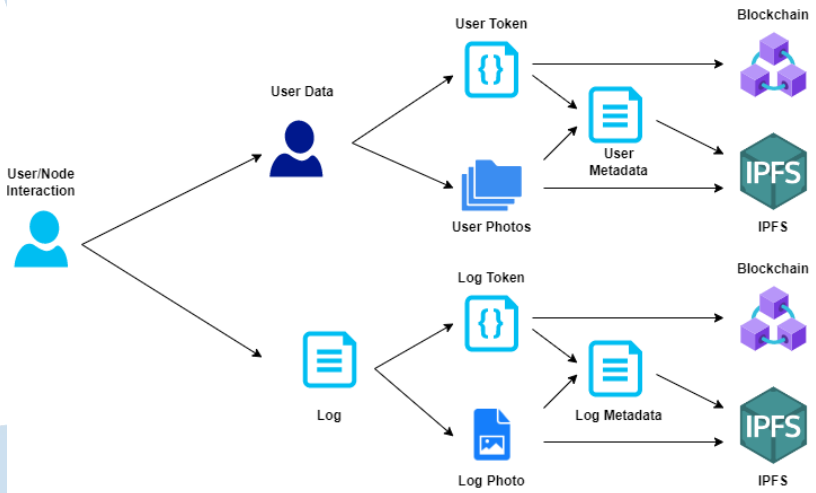
Algoritma Konsensus yang akan digunakan adalah *Proof-of-Work* (PoW) dan *Proof-of-Authority* (PoA). Kedua konsensus akan digunakan, dan akan diuji. Pada program geth, konsensus PoW akan menggunakan algoritma Ethash dan konsensus PoA akan menggunakan algoritma Clique.

Untuk sistem IPFS, program yang akan digunakan adalah go-ipfs, yang merupakan program IPFS dengan basis bahasa Go. program ini akan dijalankan bersamaan program geth yang terdapat pada VM. Sehingga satu *node blockchain*, berisi program geth dan go-ipfs. Jaringan *private* IPFS akan juga digunakan. Untuk membuat IPFS menjadi *private* maka akan dibuat sebuah key, untuk mengetahui satu *node* dengan *node* lainnya, dan akan dimasukan sebuah *environment variable* yang menandakan bahwa IPFS akan menggunakan jaringan *private.*

Setiap VM akan meng-*expose* port berupa 22 untuk ssh, 8080 sebagai gateway IPFS, 30303 sebagai port geth untuk berkomunikasi satu dengan yang lainnya, port 8545 untuk api HTTP geth. Sebuah docker *image* yang dibuat menggunakan *image* nginx, akan dibuat untuk menjadi sebuah *load balancer*, sehingga *node* hanya memanggil API pada satu IP *address* saja.

Di dalam *blockchain*, akan diluncurkan 2 *smart contract*, yaitu *UserToken*, dan *LogToken.* Kedua *smart contract* ini, akan dibuat berdasarkan ERC-721, yang dibuat oleh *OpenZepellin*. Sebuah *Token* akan memiliki fungsi yang digunakan pada sistem antara lain *balanceOf()*, untuk menghitung *balance token* dari sebuah *address*, *OwnerOf()* untuk mengetahui pemilik dari *Token* tersebut, *TokenURI()* yang digunakan untuk mendapatkan URI dari sebuah *token*, *TotalSupply()* yang digunakan untuk mendapatkan jumlah *token* yang beredar, *tokenOfOwnerByIndex()* yang digunakan untuk mendapatkan jumlah *token* yang dimiliki sebuah *address*, *hashRole()* yang digunakan untuk mengecek apakah *addres* memiliki suatu *role*, *grantRole()* untuk memberikan suatu *role* kepada sebuah *address UserToken* berisi mengenai manajemen *user* pada *blockchain*, sedangkan *LogToken* berisi mengenai manajemen sistem.

Sebuah *UserToken* akan berisi *address, URI metadata* dan *URI keystore. URI metadata* berisi *hash* dari *metadata user* yang terdapat pada IPFS, yang berisi nama dan foto, Foto berisi kumpulan *hash* dari foto *user* yang terdapat pada IPFS. sedangkan *URI Keystore* berisi *hash* dari file *keystore* yang ada pada IPFS. Sedangkan sebuah *LogToken* berisi *address,* dan *hash* dari *URI metadata.* Sebuah *metadata log* berisi *name, location, timestamp, type* dan *photo. Name* berisi *address* dari *user* yang melakukan akses, *location* berisi nama *node* yang menangani akses, *timestamp* berisi waktu pada saat akses dilakukan, *type* berisi tipe dari *log*, seperti *face_mismatch, allowed, timeout* dan *photo* berisi *hash* dari foto yang diambil saat akses dilakukan yang terdapat di IPFS. Gambar 3.13 menampilkan arsitektur penyimpanan dari data *log* dan data *user.*

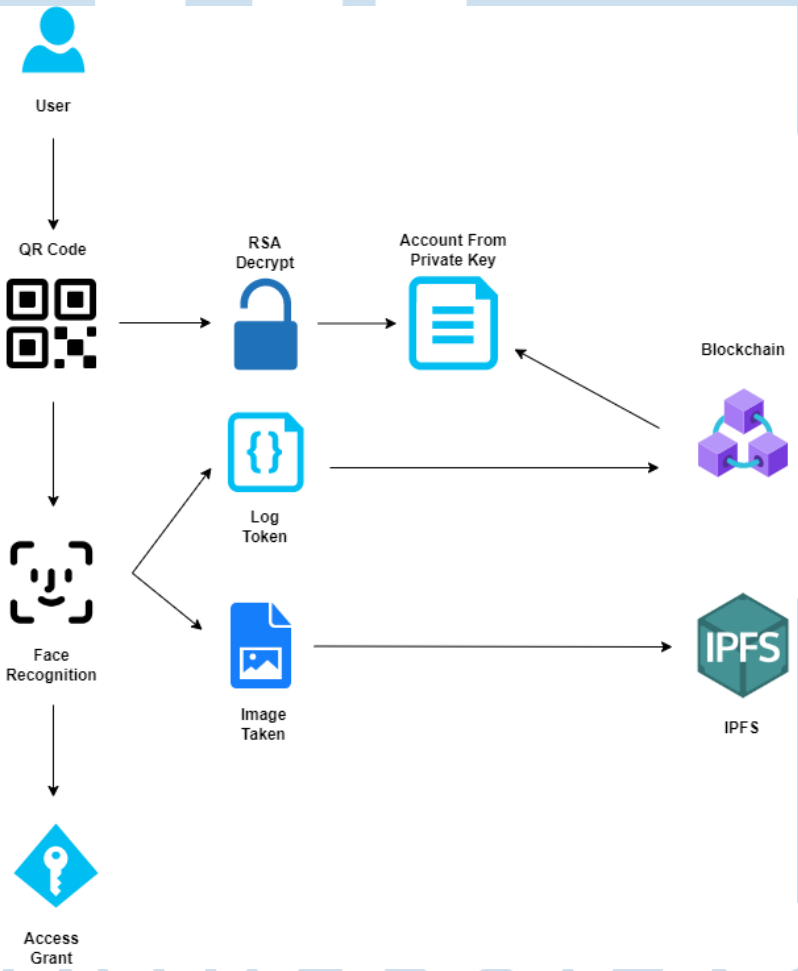


Gambar 3.13 Arsitektur penyimpanan data *log* dan data *user*

3.5 Rancangan Node

Node merupakan sistem yang berinteraksi dengan *blockchain* dan bertujuan untuk memberikan akses fisik kepada pengguna. Node sendiri terdiri atas gabungan dari Hardware dan Software. Hardware yang digunakan pada Node adalah Raspberry Pi sebagai otak dari Node, Pi Camera sebagai masukan berupa gambar kedalam Node, Relay yang digunakan untuk memberikan kontrol akses fisik pada Node serta layer LCD yang digunakan sebagai titik interaksi antara pengguna

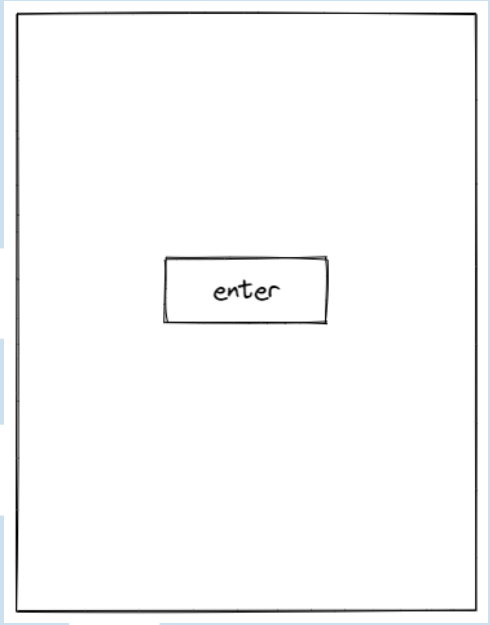
dengan sistem Node. Untuk Software, Node terutama Raspberry PI menggunakan sistem operasi Raspberry Pi OS, yang merupakan salah satu distribusi Linux yang mendukung perangkat Raspberry Pi. Untuk menjalankan program, Node akan menggunakan bahasa pemrograman Python. Bahasa ini dipilih karena mendukung *library* yang dapat berinteraksi dengan *blockchain*, serta memiliki *library* yang dapat mengontrol GPIO yang terdapat di Raspberry Pi. Untuk menampilkan UI, digunakan *library* Tkinter yang merupakan *library* yang telah *built-in* di dalam paket instalasi Python. Flow dari sistem Node dapat dilihat pada gambar 3.14



Gambar 3.14 Flow sistem Node

Rancangan UI dari sistem Node adalah sebagai berikut: Ketika dinyalakan atau sistem telah memberikan akses, maka laman yang akan ditampilkan adalah laman *home*. Laman ini berisi sebuah tombol dengan teks

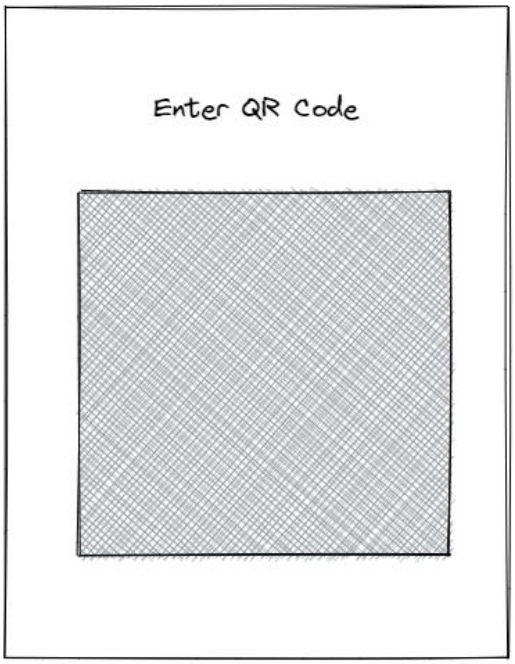
"*enter*" yang ketika di tekan akan memulai sistem akses kontrol Node. Desain dari laman *home* dapat dilihat pada gambar 3.15.



Gambar 3.15 Laman *home*

Ketika pengguna menekan tombol enter, maka pengguna akan dibawa menuju laman *QR Code*, dimana pada laman ini akan ditampilkan sebuah *video feed*, yang akan menampilkan hasil dari Pi *Camera*. Untuk membaca *QR Code*, akan digunakan *library* OpenCV yang mendukung pendeteksian *QR Code*, dan *library* Pyzbar yang digunakan untuk membaca *QR Code* yang telah terdeteksi. Sistem akan menunggu sampai user menampilkan *QR Code* yang berisi *address* dan *private key* yang telah dilakukan *hashing* dengan *public key* dari rsa. Jika *QR Code* yang diberikan tidak sesuai, maka sebuah *log* akan dikirim ke *blockchain* yang menandakan bahwa pengguna mencoba untuk memasuki dengan *QR Code* yang berbeda. Sedangkan jika pengguna tidak menampilkan *QR Code* selama 30 detik, maka sistem akan memanggil *timeout* yang akan membawa pengguna kembali ke laman *home.* Ketika *timeout* terjadi sebanyak 3 kali, maka sistem akan menambahkan *log* ke *blockchain* yang menandakan ada pengguna yang ingin mencoba masuk, tetapi tidak memiliki kredensial yang diperlukan. Jika sistem mendeteksi bahwa *user* dikunci oleh *admin*, maka *user* tidak akan diberikan akses,

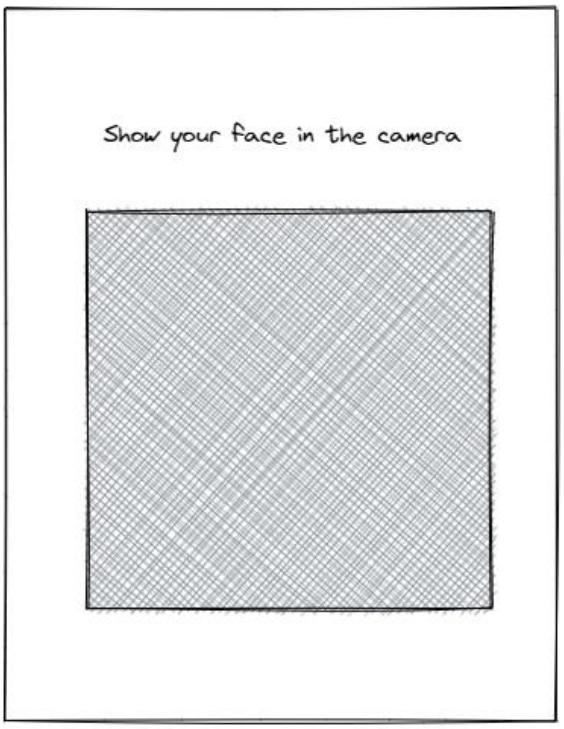
dan sistem akan membuat *log* token yang berisi bahwa *user* yang terkunci ingin melakukan akses. Desain dari laman *QR Code* ada pada gambar 3.16.



Gambar 3.16 Laman *QR Code*

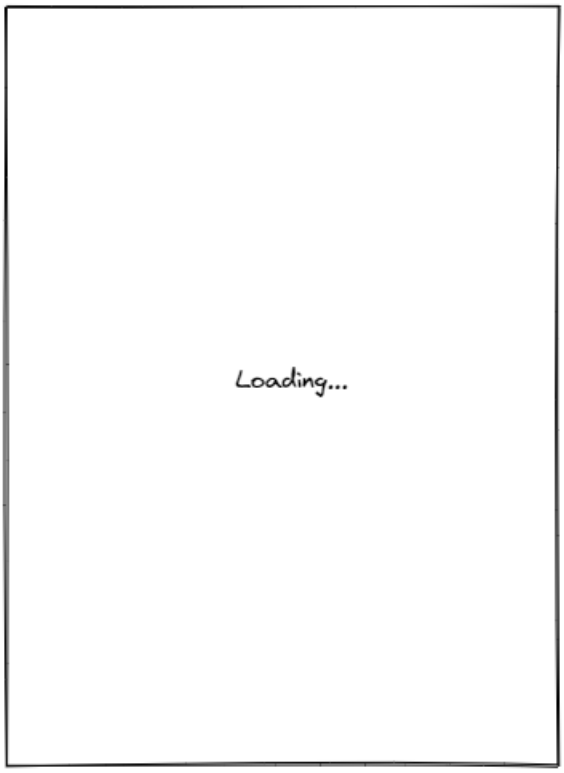
Setelah menampilkan *QR Cod*e, dan *QR Code* berhasil diidentifikasi dan dibaca sebagai *address* dan *private key* yang sesuai, sistem kemudian akan menampilkan laman *face identification.* Sama seperti pada lama *QR Code*, sistem akan menampilkan *video feed* yang menampilkan hasil tangkapan Pi Camera. Sistem akan mendeteksi apakah terdapat wajah pada hasil tangkapan kamera dengan menggunakan *library* OpenCV yang menggunakan *haarcascade classifier* untuk mendeteksi wajah. Kemudian untuk mengenali wajah, akan digunakan *library* deepface. Deepface merupakan *library* pengenal wajah yang dibuat dengan menggunakan arsitektur *Siamese neural network*, dimana pada arsitektur ini, hasil dari 2 *CNN* akan dibandingkan untuk mengambil sebuah kesimpulan. Arsitektur CNN yang akan digunakan pada penelitian ini menggunakan VCGFace sebagai arsitektur utama. Untuk mendapatkan *dataset*, maka pertama-tama *node* akan mengecek apakah terdapat folder *'db'* pada perangkat, jika tidak, maka sistem akan memulai dengan menghitung banyak *user token* dengan memanggil *smart contract.* Kemudian, sistem akan melakukan *looping* sebanyak jumlah *user token* untuk

mengambil identitas *user* dan mendapatkan *foto user*. Setiap user akan memiliki foldernya sendiri yang ditandai dengan *address user* tersebut. Ketika ada *user* baru terbentuk, maka sistem akan mendengarkan *event* pada *smart contract*, dan akan mengambil details dari *user token* yang disebarkan melalui *event.* Jika penambahan *user* terjadi ketika *node offline* maka pada saat *startup* aplikasi, sistem akan mengecek apakah jumlah folder pada *node* sama dengan jumlah *user token* yang beredar. Ketika melakukan pengenalan wajah, deepface kemudian akan menampilkan *folder* dengan data wajah yang paling mendekati, kemudian sistem akan mengambil *address* dari *folder* tersebut, jika *address* dari *folder* tersebut sama, maka sistem akan melanjutkan untuk membuat sebuah *log token* yang menandakan bahwa *user* berhasil teridentifikasi dan akses kontrol akan diberikan, namun jika *address* yang terdapat berbeda atau tidak ada wajah yang terdeteksi, maka sistem akan menambahkan *log token* yang berisi bahwa *user* tidak berhasil untuk diindentifikasi. Setelah *user* berhasil diindentifikasi ataupun tidak berhasil diidentifikasi, sistem akan menampilkan kembali halaman *home* dan *address* dan *private key* akan yang terdapat pada sistem akan dihapus.



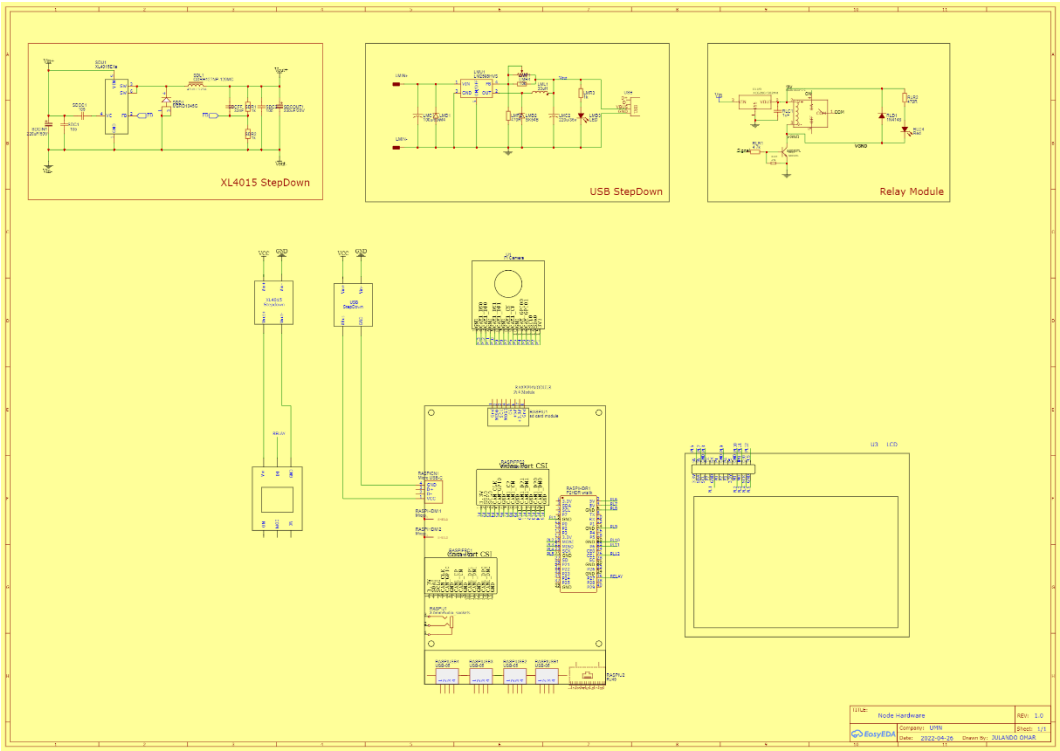
Gambar 3.17 Laman *face recognition*

Setiap perpindahan laman, maka sistem akan menampilkan laman *loading*. Laman *loading* dibuat untuk memberitahu *user* bahwa sistem masih bekerja, dan memproses hasil dari setiap *laman*. Desain dari laman *loading* ada pada gambar 3.18



Gambar 3.18 Laman *loading*

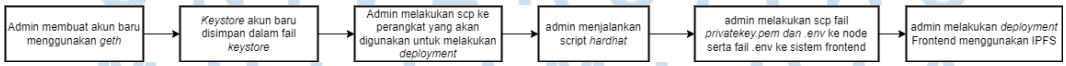
Hardware diagram dari *node* terdapat pada gambar 3.19. Pi Camera berkomunikasi dengan Raspberry Pi dengan menggunakan *port* CSI, LCD berkomunikasi dengan menggunakan *port* GPIO, *port* GPIO yang digunakan adalah *port* 2, *port* 4, *port* 6, *port* 9, *port* 19 – 23, dan *port* 26, dimana berkomunikasi dengan protokol SPI yang terdapat pada *port* 19 sebagai MOSI, *port 21* sebagai MISO, *port* 23 sebagai SCK. Untuk memproses *relay* digunakan *relay module* dengan Vin dan Vout berasal dari *module stepdown* X4015. Signal kontrol *relay* dikontrol menggunakan GPIO 16 Raspberry Pi. Untuk menyalakan Raspberry Pi digunakan USB-C yang diambil dari modul *USB stepdown*.



Gambar 3.19 *hardware diagram node*

3.6 Rancangan Deployment

Untuk melakukan *deployment smart contract* sistem akan menggunakan *library* Hardhat yang dibuat menggunakan bahasa *Javascript*. Hardhat merupakan *library* yang digunakan untuk *deployment smart contrat*, Hardhat juga sudah terintegrasi dengan *library* etherjs. Untuk melakukan *deployment* juga diperlukan akun admin yang sudah diberikan *ether* terlebih dahulu, akun tersebut sebelumnya harus dibuat dengan menggunakan geth. Alur dari rancangan *deployment smart contract* ada pada gambar 3.20

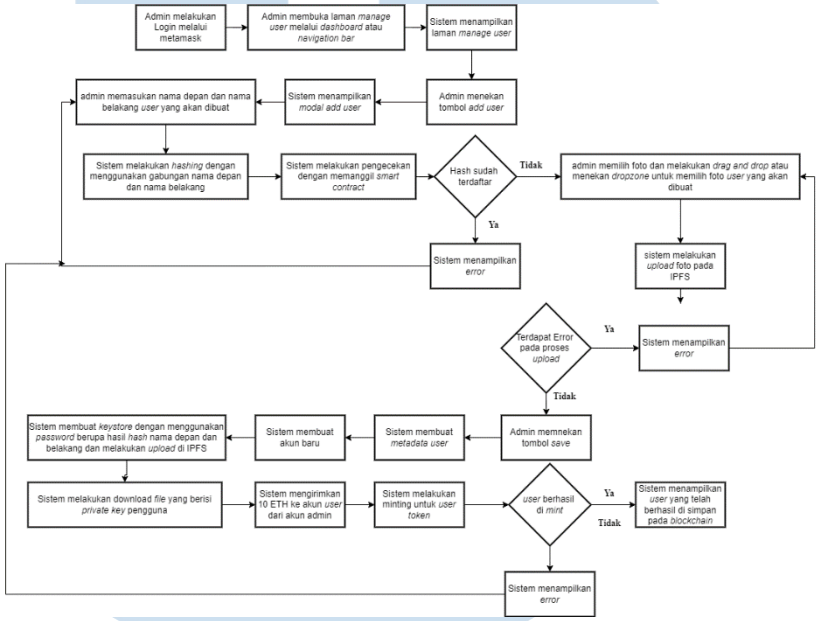


Gambar 3.20 Alur *deployment smart contract*

3.7 Use Case Sistem

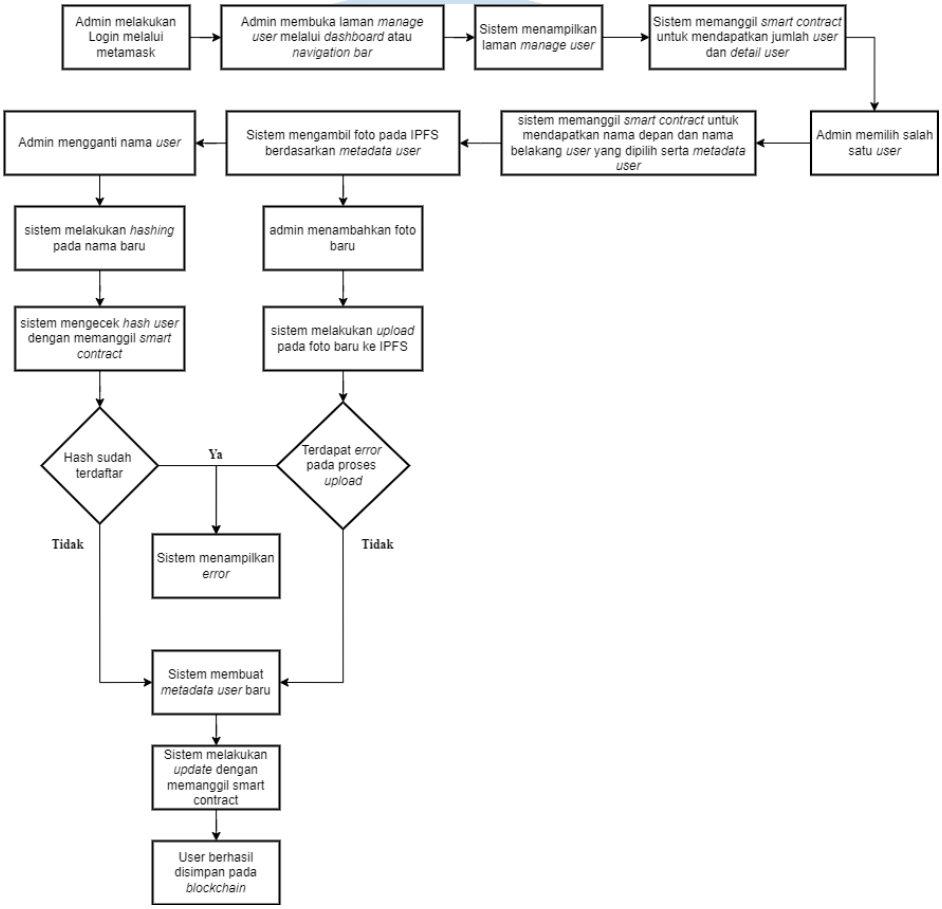
3.7.1 Admin Membuat Akun Baru

Untuk membuat akun baru, alur yang akan dilakukan oleh admin adalah sebagai berikut



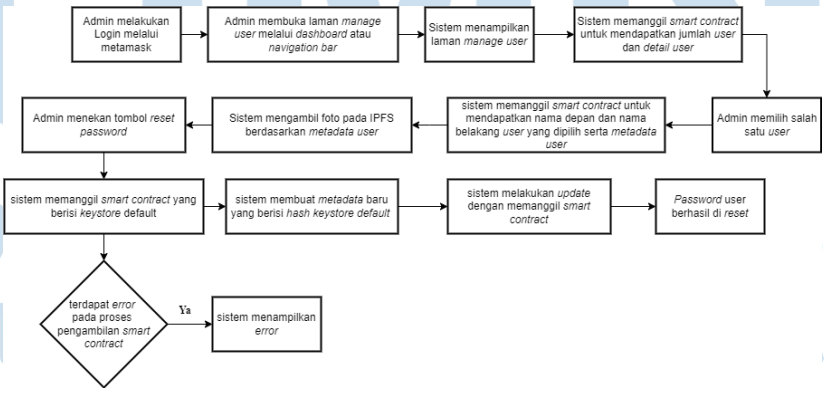
Gambar 3.21 *Use Case* ketika *admin* membuat *user* baru

3.7.2 Admin Mengganti Detail User



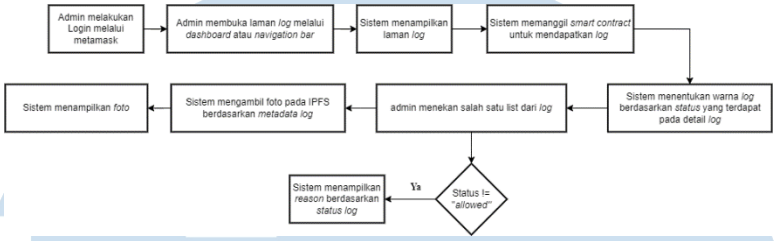
Gambar 3.22 *Use Case* ketika *admin* melakukan *update user*

3.7.3 Admin Melakukan Reset Password User



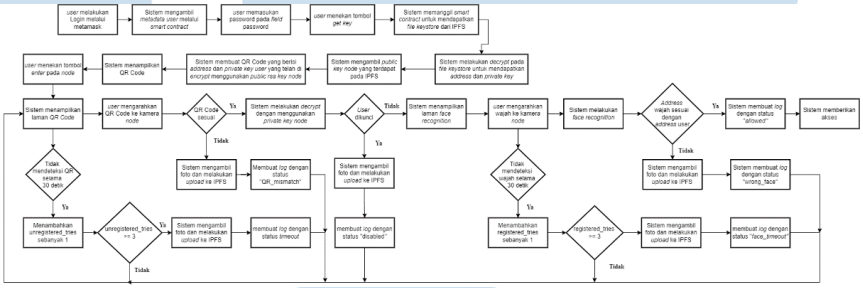
Gambar 3.23 *Use case* ketika *admin* melakukan *update password user*

3.7.4 Admin Melihat Log



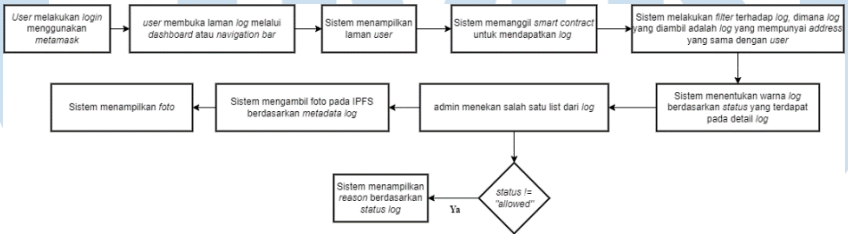
Gambar 3.24 *Use Case admin* melihat *log*

3.7.5 User Melakukan Akses



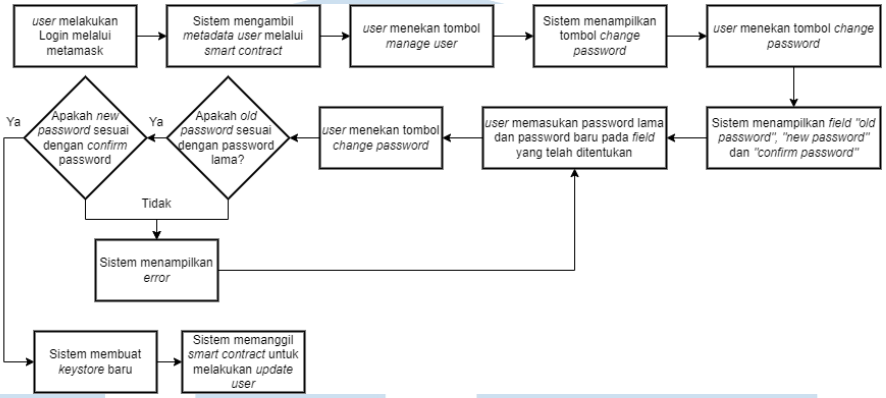
Gambar 3.25 *Use case user* melakukan akses

3.7.6 User Melihat Log Dari User



Gambar 3.26 *Use case user* melihat *log* dari *user*

3.7.7 User Mengganti Password



Gambar 3.27 *Use case user* mengganti *password*