BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan kerja magang di PT Mitra Integrasi Digital, mahasiswa ditempatkan pada posisi *Junior Programmer* dengan fokus utama pada pengembangan *website* pengelolaan aset digital. Dalam posisi tersebut, mahasiswa memiliki tanggung jawab untuk merancang dan mengimplementasikan komponen antarmuka pengguna, serta mengembangkan fitur-fitur *frontend* sesuai dengan desain dan spesifikasi yang telah ditentukan. Selain itu, mahasiswa juga terlibat dalam proses pengembangan *Application Programming Interface* (API) dengan memanfaatkan teknologi Node.js.

Pelaksanaan pekerjaan dilakukan dengan mengikuti pola kerja yang bersifat iteratif dan berbasis penugasan harian. Setiap hari, mahasiswa menerima daftar tugas dari *mentor* atau *supervisor* yang kemudian dikerjakan secara mandiri. Apabila terdapat tugas yang belum dapat diselesaikan dalam satu hari kerja, maka pengerjaan dilanjutkan pada hari berikutnya. Setelah tugas diselesaikan, hasil pekerjaan disampaikan kepada *supervisor* untuk mendapatkan evaluasi, masukan, serta persetujuan melalui sesi *review* yang umumnya dilaksanakan pada sore atau malam hari.

Selain memberikan arahan tugas, *supervisor* juga melakukan pemantauan secara berkala terhadap perkembangan pekerjaan mahasiswa. Pemantauan tersebut meliputi proses pengujian fitur dan pelaksanaan *code review* apabila diperlukan. Melalui proses ini, mahasiswa memperoleh umpan balik yang membantu dalam memahami standar kualitas kode serta penerapan praktik pengembangan perangkat lunak yang baik. Pola kerja tersebut mendukung proses pembelajaran yang berkelanjutan dan membiasakan mahasiswa dengan siklus pengembangan perangkat lunak yang iteratif.

3.2 Tugas yang Dilakukan

Bagian ini menjelaskan ruang lingkup tugas yang dilaksanakan oleh mahasiswa selama menjalani kerja magang. Pembahasan mencakup uraian tugas kerja magang, penjelasan detail setiap tugas, tahapan pelaksanaan, *project requirement* dan rancangan awal sistem, proses *development*, hingga tahap

implementasi dan pengujian sistem.

3.2.1 Tugas Kerja Magang

Selama periode kerja magang yang berlangsung dari bulan Desember 2024 hingga April 2025, mahasiswa melaksanakan berbagai tugas yang berkaitan dengan pengembangan *website* pengelolaan aset digital. Seluruh tugas tersebut dikerjakan berdasarkan arahan serta supervisi langsung dari Bapak Boby Hartanto dan Bapak Boby Harmoko. Rincian tugas yang dikerjakan disusun secara bertahap setiap minggu sesuai dengan kebutuhan proyek dan progres pengembangan sistem.

Tabel 3.1. Tugas yang dilakukan selama magang

Minggu	**Tugas yang dilakukan**	**Tanggal Mulai**	**Tanggal Selesai**
1	Membahas *project* yang ingin dikerjakan dan mempelajari *node js*	18 Agustus 2025	22 Agustus 2025
2	Mempelajari teknologi yang digunakan (*node js*, *vue*, *nuxt*)	25 Agustus 2025	29 Agustus 2025
3	Mempelajari konsep *blockchain*, sistem *custodian*, dan *Hierarchical Deterministic Wallet*	1 September 2025	4 September 2025
4-8	Membuat *Microservice* untuk *website* menggunakan *node js* dan *express js*, seperti fitur *login*, *register*, *wallet creation* dan lain lain	8 September 2025	9 Oktober 2025
8-16	Membuat *page* dan integrasi ke *api*	10 Oktober 2025	9 Desember 2025

3.2.2 Penjelasan Tugas

A Project Multisig Bitcoin Wallet System

Pada *project multisig bitcoin wallet system* ini terdapat 12 halaman *frontend* dan sistem *backend API* yang komprehensif. Setiap halaman mempunyai fungsi dan fiturnya masing-masing. Mahasiswa magang akan menjelaskan secara singkat melalui tabel di bawah ini.

Tabel 3.2. Halaman pada *frontend*

Halaman	**Fungsi**	**Fitur**
Login	Halaman untuk autentikasi *user* dengan sistem *role-based access*	*User* dapat melakukan *login* menggunakan *username/email* dan *password*, sistem mendukung 3 *role*: *Customer*, *Admin*, dan *Super Admin*
Register	*User* dapat melakukan registrasi akun baru	Registrasi dengan informasi *username*, *email*, *password*, dan *assignment* ke *Admin* yang bertanggung jawab
Dashboard	*Page* utama setelah *login* yang menampilkan ringkasan aktivitas	Menampilkan total *vault*, total saldo *Bitcoin*, *recent transactions*, dan *quick actions* sesuai dengan *role user*
Personal Wallet	*User* dapat mengelola *personal Bitcoin wallet* untuk transaksi cepat	Menampilkan saldo, alamat *wallet*, *send Bitcoin* dengan *single signature*, dan riwayat transaksi *personal*
Vault List	*User* dapat melihat semua *multisig vault* yang dimiliki atau menjadi *signer*	Menampilkan daftar *vault* dengan informasi saldo, status, dan *role user* pada masing-masing *vault*
Vault Details	*Page* untuk melihat detail *multisig vault* dan melakukan *transfer*	Menampilkan informasi *vault*, daftar *signers*, saldo, alamat *vault*, *form transfer* yang memerlukan *3-of-5 signatures*

Berlanjut ke halaman berikutnya

Tabel 3.2 Halaman pada *frontend*(Lanjutan)

Halaman	**Fungsi**	**Fitur**
Create Vault	*User* dapat membuat *multisig vault* baru dengan sistem *3-of-5*	*Form* pembuatan *vault* dengan pemilihan 2 *friend signers*, sistem otomatis *assign Super Admin* dan *Admin* sebagai *signers*
Transaction History	*User* dapat melihat riwayat semua transaksi *vault*	Menampilkan *pending transactions*, *completed transactions*, status *signature*, dan *explorer links*
Pending Signatures	*Page* untuk *signers* melihat transaksi yang perlu ditandatangani	Daftar transaksi *pending*, informasi detail transaksi, tombol *approve/reject*, dan *tracking progress signatures*
Admin Panel	Halaman khusus *Admin* untuk mengelola *customers* dan *vault* mereka	Melihat daftar *customers*, *vault* yang dikelola, *transaction monitoring*, dan *customer management*
Super Admin Panel	Halaman khusus *Super Admin* untuk mengelola seluruh sistem	*User management*, sistem *monitoring*, *vault oversight*, dan *broadcast manual transactions*
Change Password	*User* dapat melakukan perubahan *password*	*Form* ubah *password* dengan validasi *password* lama dan konfirmasi *password* baru

Selain mengembangkan halaman antarmuka pengguna (*frontend*), mahasiswa magang juga bertanggung jawab dalam merancang dan membangun sistem *backend API* yang *robust*. Sistem *backend* ini menggunakan *Express.js* dan berfungsi sebagai *bridge* antara *frontend* dengan *Bitcoin testnet*, mengelola *cryptographic operations*, dan menyediakan *security layer* untuk *multisig operations*. *Backend* dilengkapi dengan *comprehensive API endpoints* untuk *wallet management*, *vault operations*, *transaction processing*, dan *Bitcoin network integration*. Sistem ini juga menangani *sensitive operations* seperti *WIF key generation*, *encryption/decryption*, dan *PSBT* (*Partially Signed Bitcoin Transaction*) *creation* untuk memastikan keamanan maksimal dalam pengelolaan *Bitcoin assets*.

Tabel 3.3. API *Endpoints* pada *backend*

Endpoint	**Fungsi**	**Fitur**
POST /login	*Endpoint* untuk autentikasi *user* dan *generate JWT token*	Validasi *credentials*, *role-based authentication*, *JWT token generation* dengan *expiration time*
POST /register	*Endpoint* untuk registrasi *user* baru dengan *role assignment*	*User registration* dengan *auto-assignment* ke *Admin*, *email validation*, *password hashing* dengan *bcrypt*
GET /wallet/my-wallet	Mengambil atau membuat *personal wallet* untuk *user*	*Auto-generate deterministic wallet* dengan *index system*, *WIF key generation*, *Bitcoin testnet address creation*
POST /transaction/send	*Endpoint* untuk *instant Bitcoin transfer* dari *personal wallet*	*UTXO management*, *PSBT creation*, *single signature*, *instant broadcast* ke *Bitcoin testnet* via *Blockstream API*
GET /vaults/my-vaults	Mengambil daftar *multisig vault* yang dimiliki atau menjadi *signer*	*Role-based vault access*, *balance formatting*, informasi *signer* dengan *encrypted key storage*
GET /vaults/:id	Mengambil detail spesifik *vault* dengan informasi lengkap	*Vault details*, *signer list*, *balance calculation*, *role-based access control* dengan *security validation*
POST /vaults/create	*Endpoint* untuk membuat *multisig vault* baru	*Generate* 5 *WIF keys*, *create 3-of-5 P2WSH multisig*, *encrypt private keys*, *assign signers* (*Super Admin*, *Admin*, *Owner*, 2 *Friends*)
POST /vaults/:id/transfer	Membuat *pending transaction* untuk *multisig vault*	*Create pending transaction*, *generate signature requests* untuk 5 *signers*, *transaction validation*, *amount verification*

Berlanjut ke halaman berikutnya

Tabel 3.3 API *Endpoints* pada *backend* (Lanjutan)

Endpoint	**Fungsi**	**Fitur**
GET /vaults/pending-transactions	Mengambil daftar transaksi yang memerlukan *signature*	*List pending transactions* berdasarkan *user role*, *signature status tracking*, *progress calculation*
GET /vaults/completed-transactions	Mengambil riwayat transaksi yang sudah selesai	*Transaction history* dengan *role-based filtering*, *explorer integration*, *confirmation status tracking*

Untuk hasil dari perancangan, beserta skema *database* berupa *ERD* dan *Bitcoin network architecture*, akan dilampirkan pada halaman selanjutnya pada bagian *project development* dan halaman implementasi.

B Arsitektur Teknis dan Security

Project multisig bitcoin wallet system ini menggunakan arsitektur *dual-system* yang inovatif, menggabungkan kemudahan *personal wallet* dengan keamanan *enterprise-grade multisig vault*. Sistem *personal wallet* menggunakan *single WIF key* untuk *instant transfer* melalui *Blockstream API*, sementara sistem *multisig vault* mengimplementasikan *3-of-5 signature scheme* dengan *P2WSH* (*Pay to Witness Script Hash*) *format* untuk *maximum security*.

Teknologi yang digunakan meliputi *Node.js* sebagai *runtime environment*, *Express.js* untuk *RESTful API*, *Nuxt.js* untuk *modern frontend* dengan *SSR capabilities*, *BitcoinJS-lib* untuk *Bitcoin protocol implementation*, dan *real Bitcoin testnet integration* untuk *production-ready testing*. *Security layer* menggunakan *AES-256-GCM encryption* untuk *private key storage*, *deterministic key generation* untuk *reproducibility*, dan *comprehensive role-based access control system*.

Database schema dirancang dengan *normalization* yang optimal, menggunakan *Prisma ORM* untuk *type-safe database operations*, dengan *tables* untuk *Users*, *Vaults*, *VaultSigners*, *PendingTransactions*, dan *TransactionSignatures*. Setiap transaksi dalam *multisig vault* memerlukan *consensus* dari 3 *out of* 5 *signers* (*Super Admin*, *Admin*, *Owner*, dan 2 *Friends*), memastikan *distributed security* dan *eliminating single points of failure*.

3.3 Detail Pelaksanaan Magang

Pada minggu pertama hingga minggu terakhir, mahasiswa magang diposisikan sebagai *full-stack blockchain developer*, dimana bertanggung jawab atas pengembangan sistem *multisig Bitcoin wallet* secara keseluruhan, mulai dari *frontend Vue.js*, *backend Express.js*, integrasi *Bitcoin network*, dan juga *cryptographic security implementation*. Total jam kerja yang diselesaikan selama magang mencapai lebih dari 640 jam, melebihi batas minimal yang ditetapkan oleh kampus.

3.3.1 Project Requirement dan Rancangan Awal

Multisig Bitcoin wallet system ini dikembangkan untuk memfasilitasi pengelolaan *Bitcoin* dengan tingkat keamanan *enterprise-grade* melalui implementasi *multisignature technology*. Sistem ini dirancang untuk memberikan solusi *custodial* yang aman bagi organisasi atau individu yang memerlukan *distributed control* atas *Bitcoin assets*, dimana tidak ada *single point of failure* dalam pengelolaan *private keys*.

Dengan arsitektur *dual-system* yang inovatif, platform ini menyediakan dua mode operasi: *personal wallet* untuk transaksi cepat dengan *single signature*, dan *multisig vault* dengan skema *3-of-5 signatures* untuk keamanan maksimal. Sistem ini mengotomasi proses pembuatan *multisig vault*, *signature collection*, dan *transaction broadcasting* ke *Bitcoin testnet*, yang sebelumnya memerlukan *technical expertise* yang tinggi dan proses manual yang *complex*.

System ini dibangun dengan arsitektur modern yang terdiri dari empat *layer* utama, yaitu *Frontend* menggunakan *Nuxt.js* untuk *user interface*, *Backend API* menggunakan *Express.js* untuk *business logic*, *Bitcoin Integration Layer* untuk *real blockchain operations*, serta *Security Layer* untuk *cryptographic operations* dan *key management*. Fokus pengembangan mencakup seluruh *stack technology* untuk memastikan *end-to-end functionality* yang *seamless*.

Fitur-fitur utama yang disediakan meliputi *role-based authentication system* dengan *three-tier hierarchy* (*Customer*, *Admin*, *Super Admin*), *deterministic wallet generation* dengan *WIF format*, *real Bitcoin testnet integration* untuk *production-ready testing*, *3-of-5 multisig vault creation* dengan *P2WSH implementation*, *pending transaction management* dengan *signature workflow*, serta *real-time balance tracking* dan *transaction broadcasting* ke *Bitcoin network*.

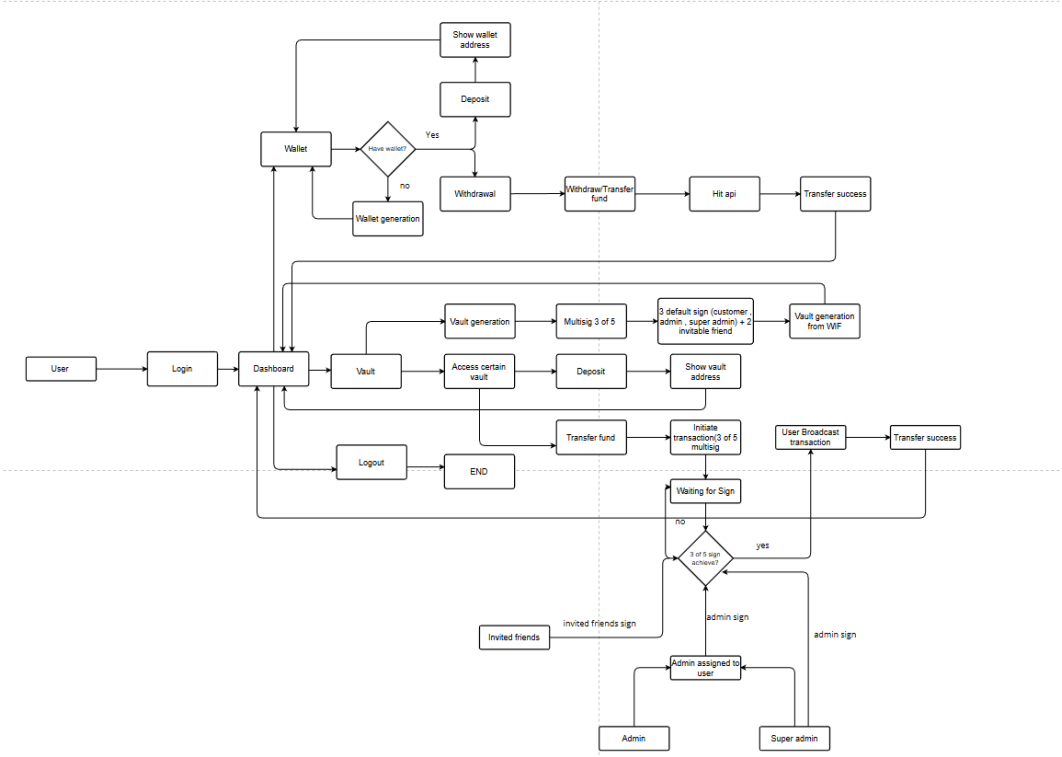
Pengguna dapat mengakses *personal wallet* untuk *instant Bitcoin transfers*, namun untuk operasi yang melibatkan *multisig vault*, sistem memerlukan *consensus* dari minimal 3 *out of* 5 *designated signers*. Setiap *vault* secara otomatis *assign Super Admin* dan *Admin* sebagai *built-in signers*, ditambah *Owner* dan 2 *Friends* yang dipilih oleh *user*, memastikan *distributed governance* dan *eliminating single points of failure*.

System ini juga mengimplementasikan *advanced security measures* seperti *AES-256-GCM encryption* untuk *private key storage*, *deterministic key generation* untuk *reproducible wallet creation*, *PSBT* (*Partially Signed Bitcoin Transaction*) untuk *secure transaction signing*, dan *real UTXO management* untuk *optimal transaction fee calculation*. Dengan integrasi ke *Mempool.space* dan *Blockstream API*, sistem dapat melakukan *real-time monitoring* terhadap *Bitcoin testnet* dan *broadcasting transactions* dengan *reliable confirmation tracking*.

Di sisi *technical implementation*, sistem menggunakan *BitcoinJS-lib* untuk *core Bitcoin operations*, *Prisma ORM* dengan *MySQL* untuk *data persistence*, *JWT authentication* untuk *session management*, dan *comprehensive API endpoints* untuk *frontend-backend communication*. Seluruh *cryptographic operations* menggunakan *industry-standard libraries* seperti *tiny-secp256k1* untuk *elliptic curve operations* dan *ECPair Factory* untuk *key pair generation*, memastikan *compatibility* dengan *Bitcoin protocol standards*.

3.3.2 Flowchart Diagram

Flowchart atau diagram alur merupakan representasi visual yang mengilustrasikan tahapan-tahapan dan *decision points* yang diperlukan untuk mengeksekusi proses dalam sistem *multisig Bitcoin wallet* secara sistematis dan terorganisir. Diagram ini memanfaatkan simbol-simbol standar yang telah ditetapkan untuk merepresentasikan berbagai jenis operasi *cryptographic*, keputusan *security validation*, dan alur data *blockchain* dalam *ecosystem Bitcoin*.



Gambar 3.1. *Flowchart Front end*

Dalam *multisig Bitcoin wallet system* ini, terdapat dua alur utama berdasarkan sistem operasi, yaitu ***Personal Wallet*** dan ***Multisig Vault***. Untuk ***Personal Wallet***, alur dimulai dari halaman *dashboard*, di mana pengguna dapat mengakses *personal wallet*, melihat saldo *Bitcoin*, dan melakukan *transfer instant*. Sistem akan otomatis men-*generate deterministic wallet* menggunakan *walletIndex* jika pengguna belum memiliki *wallet*. Pengguna dapat langsung melakukan *withdrawal/transfer* dengan *single signature* yang akan di-*broadcast* secara *instant* ke *Bitcoin testnet* melalui *Blockstream API*.

Untuk ***Multisig Vault***, alur dimulai dengan pembuatan *vault* baru di mana pengguna harus memilih 2 teman sebagai *signers*. Sistem otomatis akan meng-*assign Super Admin* dan *Admin* sebagai *additional signers*, sehingga terbentuk skema *3-of-5 multisig*. Setelah *vault* berhasil dibuat dengan 5 *WIF keys* yang ter-*encrypt*, pengguna dapat melakukan *deposit* dengan menampilkan alamat *vault* atau melakukan *transfer fund* yang memerlukan proses *signature collection*.

Ketika pengguna inisiasi *transfer* dari *multisig vault*, sistem akan membuat *pending transaction* dan mengirim *notification* kepada semua 5 *signers*. Proses ”*Waiting for Sign*” akan berlangsung hingga minimal 3 dari 5 *signers* memberikan *approval*. Jika 3 *signatures* terkumpul, sistem otomatis akan mem-*broadcast*

transaction ke *Bitcoin testnet* melalui *Mempool.space API* dan menampilkan "*Transfer success*". Sebaliknya, jika tidak mencapai *threshold* atau ada *rejection*, transaksi akan gagal.

Seluruh alur ini dilengkapi dengan *role-based access control* dimana *Customer* dapat mengakses *personal wallet* dan *vault* miliknya, *Admin* dapat mengelola *customers* dan *vault* yang di-*assign* kepadanya, sedangkan *Super Admin* memiliki akses penuh terhadap seluruh sistem termasuk *manual transaction broadcasting* untuk *recovery purposes*.

3.3.3 Proses Development

Tahap awal pengembangan proyek *Bitcoin Custodian* dilaksanakan dalam rentang waktu kurang lebih tiga minggu, yang dimulai sejak 18 November 2025 dan masih berlangsung hingga saat penulisan laporan ini. Pada fase awal ini, fokus utama pengembangan diarahkan pada pembangunan antarmuka pengguna (*frontend*) menggunakan *Nuxt.js* dengan *framework Vue.js*, serta pengembangan layanan *backend* berbasis *Node.js* dan *Express* untuk mendukung sistem *multisignature wallet Bitcoin*. Selama proses tersebut, pengembangan dilakukan secara lokal dengan memanfaatkan lingkungan *localhost* sebagai media pratinjau dan pengujian fungsi dasar sistem.

Aktivitas pengembangan pada tahap ini mencakup pembuatan sejumlah halaman utama, seperti halaman *login*, *register*, *dashboard*, pengelolaan *wallet*, pembuatan *vault*, serta proses *transaction signing*. Selain itu, dikembangkan pula berbagai *endpoint API* yang mendukung mekanisme autentikasi (*login*, *register*, dan *change password*), pengelolaan *vault multisig*, transaksi *Bitcoin* pada jaringan *testnet*, serta penerapan sistem *role-based access control*. Integrasi dengan jaringan *Bitcoin testnet* berhasil diimplementasikan untuk mendukung pembuatan *multisignature wallet*, proses penandatanganan transaksi menggunakan *PSBT* (*Partially Signed Bitcoin Transaction*), hingga proses *broadcast* transaksi ke *blockchain*.

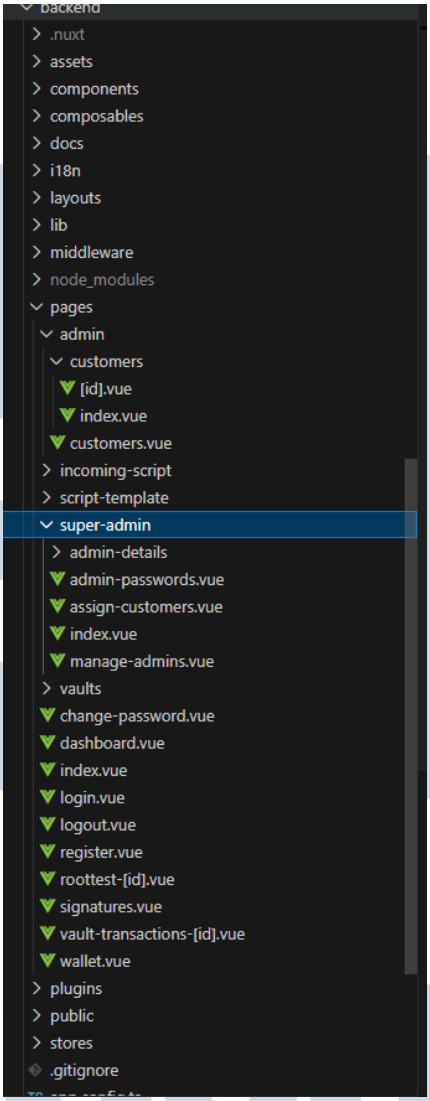
Salah satu capaian penting dalam pengembangan sistem ini adalah penerapan konsep *Zero-Database Security*. Pada mekanisme ini, data finansial transaksi tidak disimpan secara langsung di dalam basis data, melainkan direpresentasikan dan diverifikasi melalui mekanisme kriptografi pada *blockchain*. Pendekatan ini bertujuan untuk meminimalkan risiko manipulasi data pada tingkat basis data. Untuk menjaga keutuhan data transaksi, sistem menerapkan proses

verifikasi menggunakan algoritma *hash SHA-256*.

Setelah fitur-fitur utama dapat berjalan secara stabil, proses pengembangan dilanjutkan dengan tahap *website development* dan pengujian internal. Setiap tugas yang telah diselesaikan diwajibkan untuk diunggah ke *repository Git* internal perusahaan (https://git.integrasi-digital.com/mid/btc-custodian/api) guna dilakukan pemeriksaan oleh *supervisor*. Selama periode ini, *supervisor* berperan sebagai penguji sekaligus *quality control* yang melakukan evaluasi secara berkala. Mekanisme ini memungkinkan pemberian umpan balik secara cepat dan mendukung proses perbaikan sistem secara iteratif.

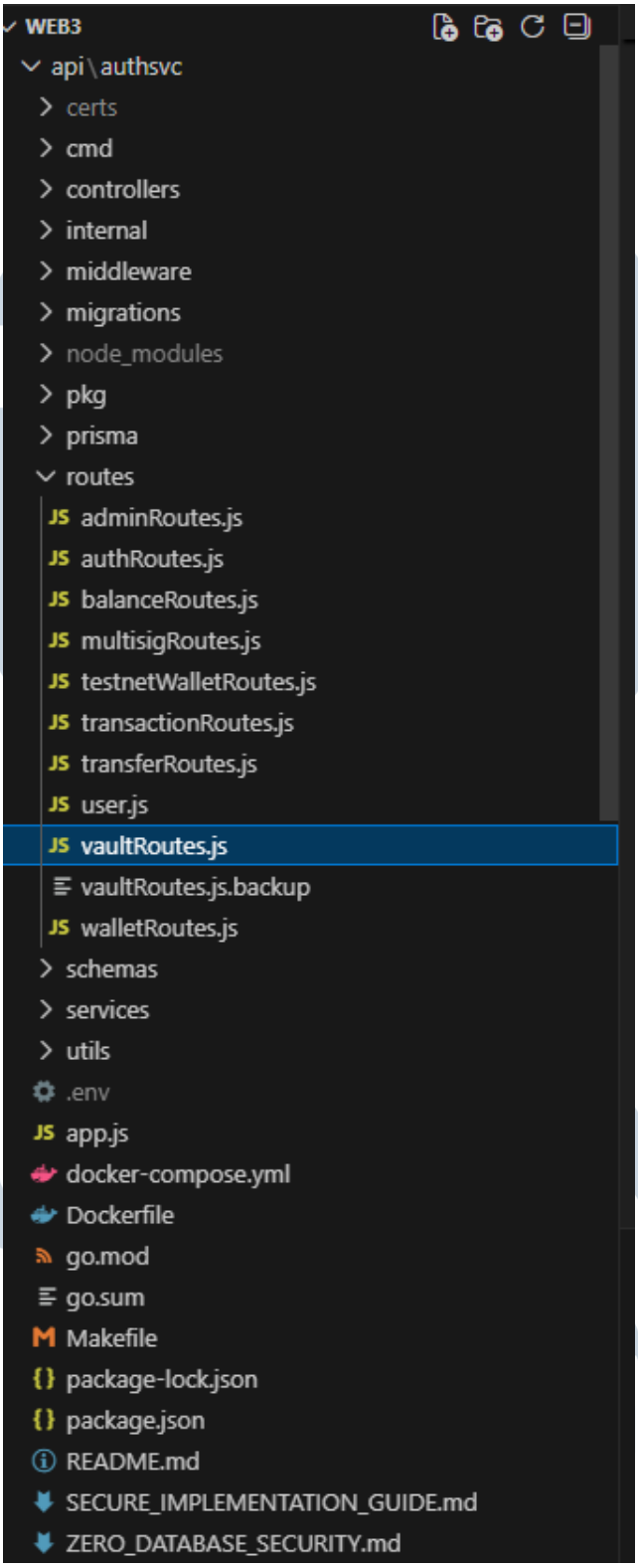
Walaupun hingga saat ini sistem masih berada pada tahap pengembangan dan belum memasuki fase produksi sepenuhnya, proyek ini memiliki potensi untuk dikembangkan lebih lanjut dan digunakan sebagai solusi nyata *custodian multisignature wallet Bitcoin* di lingkungan perusahaan. Tahapan pengujian lanjutan seperti *security testing*, *penetration testing*, maupun *sanity testing* belum dilaksanakan, namun direncanakan akan menjadi bagian dari proses validasi sebelum sistem dirilis ke publik (*go-public*).

Pada saat laporan kerja magang ini disusun, proses *development* proyek *Bitcoin Custodian API* masih terus berjalan. Sistem yang telah dikembangkan mencakup sejumlah halaman *frontend* yang dapat diakses, antara lain halaman autentikasi (*login*, *register*, *logout*), *dashboard*, pengelolaan *wallet*, pengelolaan *vault* (*create*, *detail*, dan *transactions*), proses *transaction signing*, serta halaman administrasi untuk manajemen pengguna. Pada sisi *backend*, sistem menyediakan *API* dengan struktur yang terbagi ke dalam sepuluh berkas *route*, meliputi *authentication routes*, *vault routes*, *transaction routes*, *admin routes*, *balance routes*, *multisig routes*, *transfer routes*, dan *wallet routes*. Setiap halaman dan *endpoint* diatur aksesnya berdasarkan peran pengguna, yang diklasifikasikan ke dalam tiga *role* utama, yaitu *Customer*, *Admin*, dan *Super Admin*. Berikut merupakan *asset-asset* yang digunakan dalam pengembangan proyek *Bitcoin Custodian*.



Gambar 3.2. *Asset* untuk *Front End*

Gambar 3.2 menampilkan berbagai aset yang digunakan dalam pembangunan *Bitcoin Custodian API*, khususnya pada bagian *front-end*. Aset-aset ini mencakup elemen visual seperti komponen *UI* dari *Shadcn/UI*, ikon dari *Lucide Icons*, serta komponen-komponen antarmuka pengguna (*UI*) yang mendukung tampilan dan interaksi dalam *website*. Penggunaan *framework Nuxt.js* dengan *Vue.js* memungkinkan pembuatan komponen yang *reusable* dan *reactive*, seperti tabel transaksi, *form* pembuatan *vault*, *modal* untuk *signing* transaksi, dan *dashboard* yang menampilkan informasi *wallet* secara *real-time*. Penggunaan aset yang konsisten dan terstruktur berperan penting dalam menciptakan pengalaman pengguna yang intuitif, menarik, dan responsif.



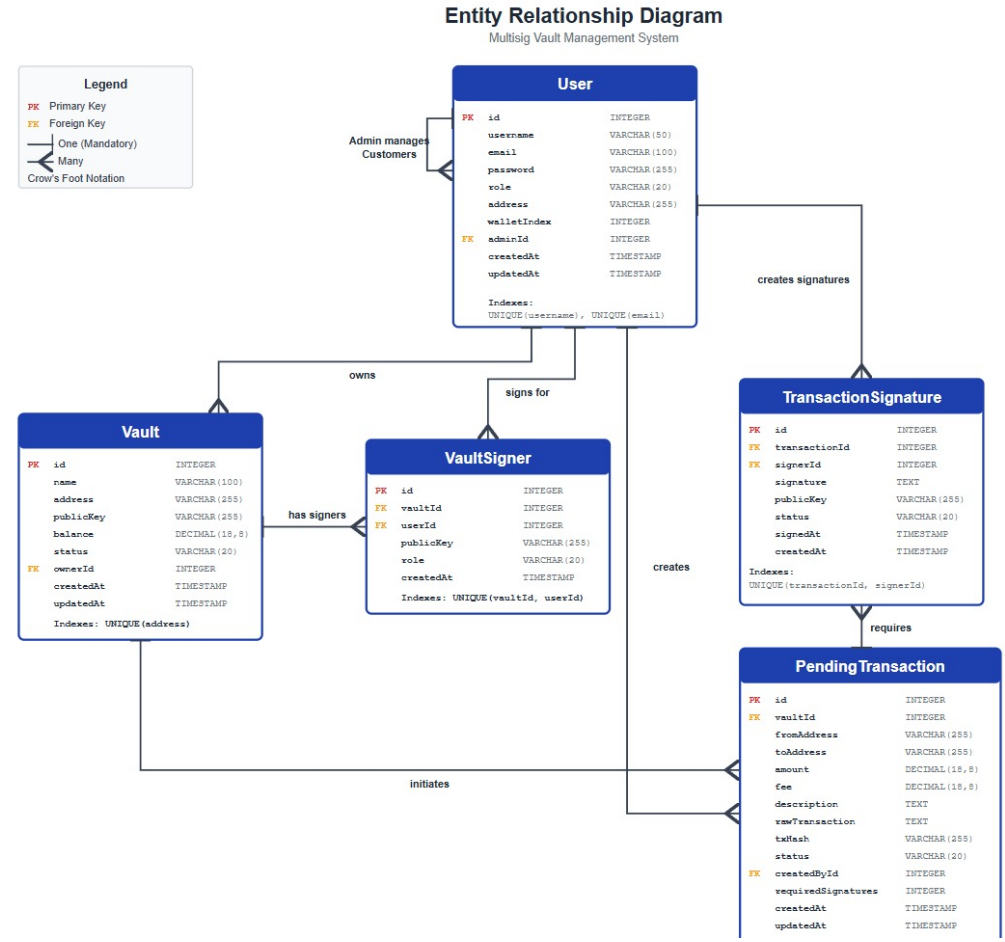
Gambar 3.3. *Asset untuk API*

Gambar 3.3 menampilkan berbagai aset yang digunakan dalam

pembangunan *API Bitcoin Custodian*, khususnya pada sisi *backend*. Aset-aset ini mencakup struktur *endpoint* yang terbagi ke dalam 10 berkas *routes* (*authRoutes*, *adminRoutes*, *vaultRoutes*, *transactionRoutes*, *transferRoutes*, *walletRoutes*, *balanceRoutes*, *multisigRoutes*, *testnetWalletRoutes*, *user.js*), dokumentasi internal *API*, konfigurasi *middleware* (*JWT*, *role-based access control* untuk *Customer*, *Admin*, dan *Super Admin*), serta dependensi dan *library* kunci untuk operasi *Bitcoin* dan keamanan. Penggunaan aset yang konsisten dan terstruktur ini memastikan komunikasi yang aman, cepat, dan efisien antara *front-end* dan server, terutama untuk fungsi-fungsi *multisignature* seperti pembuatan *vault*, penandatanganan *PSBT*, dan *broadcasting* transaksi ke jaringan *Bitcoin testnet*.

3.3.4 Skema Basis Data

Database schema merupakan gambaran visual dan logis dari arsitektur sistem basis data yang digunakan dalam pengembangan *website ticketing* ini. Skema tersebut memuat struktur tabel, hubungan antar entitas, serta atribut-atribut utama yang berperan dalam penyimpanan dan pengelolaan data secara efisien. Dengan adanya skema ini, proses pengelolaan informasi—meliputi data pengguna, tiket, transaksi lainnya—dapat dilakukan secara lebih terstruktur, konsisten, dan terintegrasi.



Gambar 3.4. Skema Basis data

Skema basis data ini dirancang untuk mendukung sistem *digital asset custodian* yang mengelola *vault*, transaksi kripto, serta mekanisme *multi-signature* guna memastikan keamanan proses penandatanganan. Struktur basis data terdiri atas lima entitas utama, yaitu *User*, *Vault*, *VaultSigner*, *PendingTransaction*, dan *TransactionSignature*, yang saling terintegrasi dan merepresentasikan alur kerja sistem secara keseluruhan.

A User

Entitas *User* digunakan untuk menyimpan informasi mengenai pengguna yang terlibat dalam sistem, baik sebagai pemilik *vault*, penanda tangan, maupun sebagai administrator. Atribut yang disimpan meliputi *username*, *password* (terenkripsi dengan *bcryptjs*), *wallet index*, alamat, serta informasi kontak seperti *email*. Atribut *role* menentukan level akses pengguna (*CUSTOMER*, *ADMIN*, atau

SUPER_ADMIN), sedangkan *adminId* menunjukkan hubungan *self-relation* untuk mekanisme pengelolaan *customer* oleh *admin* tertentu. Setiap pengguna juga dapat menjadi pemilik *vault* atau penanda tangan pada *vault* milik pengguna lain.

B Vault

Entitas *Vault* merepresentasikan dompet atau wadah penyimpanan aset digital milik pengguna. Setiap *vault* menyimpan informasi kriptografi seperti *publicKey* dan *privateKeyEncrypted* (terenkripsi menggunakan *password* pengguna), serta atribut lain seperti alamat (*address*), saldo dalam format *decimal* dengan presisi 8 desimal untuk representasi *Bitcoin*, skrip transaksi (*script*), dan status (*VaultStatus*: *PENDING*, *ACTIVE*, *INACTIVE*, *SUSPENDED*). Atribut *ownerId* menghubungkan *vault* dengan pengguna pemiliknya, dan relasi ini menggunakan *CASCADE delete* untuk menjaga *referential integrity*.

C VaultSigner

Entitas *VaultSigner* mendefinisikan daftar penanda tangan yang berwenang memberikan persetujuan terhadap transaksi pada sebuah *vault*. Setiap penanda tangan memiliki atribut *publicKey*, *role* (*OWNER*, *ADMIN*, *SUPER_ADMIN*, *FRIEND*), serta *privateKeyEncrypted* yang digunakan dalam proses penandatanganan *PSBT*. Hubungan antara *VaultSigner*, *Vault*, dan *User* memungkinkan implementasi mekanisme *multi-signature* pada sistem, dengan *constraint UNIQUE* pada kombinasi *vaultId* dan *userId* untuk mencegah duplikasi *signer* yang sama pada *vault* yang sama.

D PendingTransaction

Entitas *PendingTransaction* menyimpan data transaksi yang sedang menunggu persetujuan dari satu atau lebih penanda tangan. Informasi yang dicatat mencakup *fromAddress*, *toAddress*, jumlah (*amount*), biaya transaksi (*fee*), dan *rawTransaction* (yang menyimpan *PSBT* dalam format *text*). Perlu dicatat bahwa *field amount* dan *fee* bersifat *nullable* untuk mendukung fitur *Zero-Database Security*, di mana data finansial tidak disimpan di *database* melainkan di-*encode* dalam *blockchain data*. Atribut *requiredSignatures* menentukan jumlah minimum tanda tangan yang diperlukan (*default* 3 untuk *multisig*), *status* melacak tahap transaksi (*PENDING*, *PARTIAL*, *SIGNED*, *BROADCAST*, *CONFIRMED*, *FAILED*,

CANCELLED), *txHash* menyimpan *hash* transaksi ketika sudah di-*broadcast*, dan *createdById* merekam siapa yang membuat transaksi. Relasi dengan *Vault* juga menggunakan *CASCADE delete*.

E TransactionSignature

Entitas *TransactionSignature* berfungsi mencatat setiap tanda tangan yang diberikan oleh penanda tangan terhadap sebuah transaksi. Atribut yang disimpan meliputi *signature* (*DER encoded signature*), *publicKey*, *signedAt* (*timestamp* kapan transaksi ditandatangani), dan *status* (*PENDING*, *SIGNED*, *REJECTED*). Entitas ini berelasi langsung dengan *PendingTransaction* dan *User*, dengan *constraint UNIQUE* pada kombinasi *transactionId* dan *signerId* untuk mencegah duplikasi *signature* dari *signer* yang sama. *Cascade delete* digunakan untuk memastikan integritas data ketika transaksi dihapus.

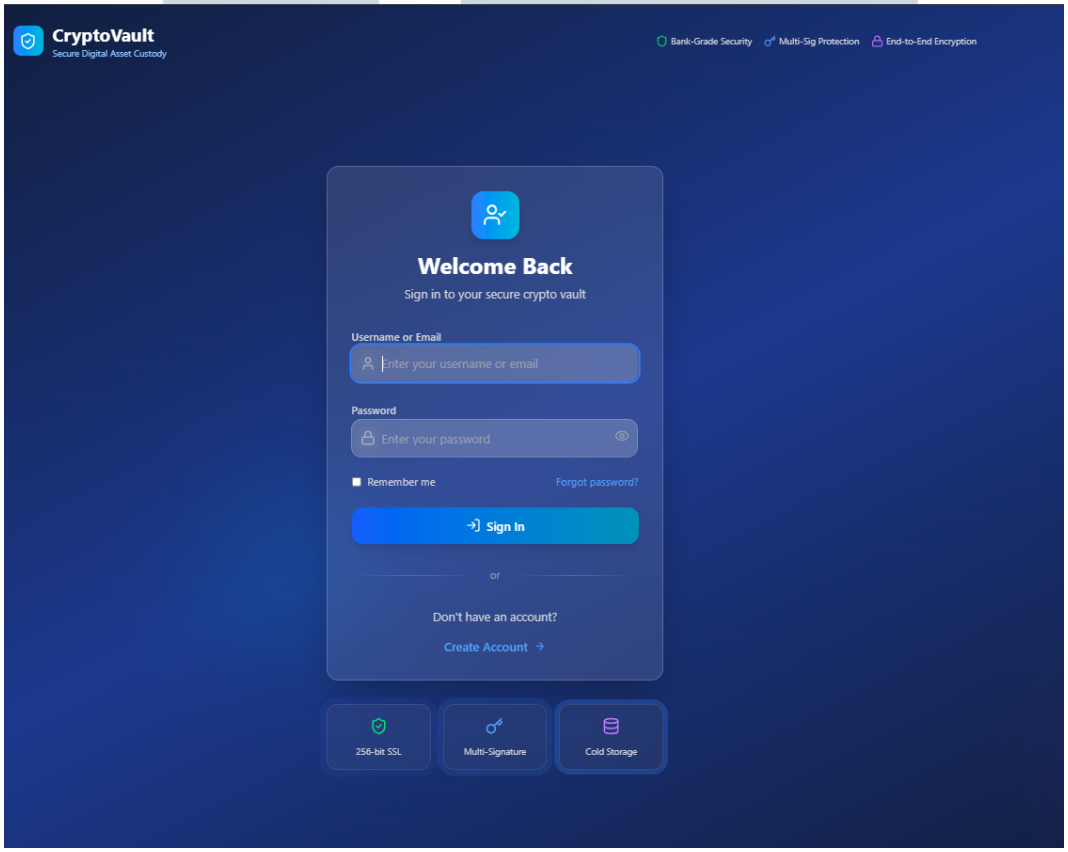
F Ringkasan Relasi

Secara keseluruhan, hubungan antar entitas dalam skema basis data ini dapat diringkas sebagai berikut:

- *User* dapat menjadi pemilik *vault*, penanda tangan pada *vault* tertentu, maupun *admin* yang mengelola *user* lain (*self-relation*).
- Setiap *Vault* dapat memiliki beberapa *VaultSigner* dan beberapa *PendingTransaction*.
- *PendingTransaction* dibuat oleh seorang pengguna dan terhubung dengan satu *Vault* tertentu.
- Setiap *PendingTransaction* dapat memiliki banyak entri *TransactionSignature* dari berbagai penanda tangan.
- Transaksi dianggap valid dan siap untuk di-*broadcast* apabila jumlah tanda tangan yang terkumpul telah memenuhi nilai *requiredSignatures*.
- Implementasi *Zero-Database Security* memastikan bahwa data finansial (*amount*, *fee*) pada transaksi dapat disimpan sebagai *NULL* dan diverifikasi dari *blockchain data* saja.

3.3.5 Halaman implementasi

Gambar-gambar berikut menampilkan beberapa halaman antarmuka yang terdapat pada *Bitcoin Custodian* yang dirancang selama pelaksanaan magang. Setiap halaman dirancang dengan mempertimbangkan aspek *user experience* (pengalaman pengguna) dan *user interface* (antarmuka pengguna), agar mudah digunakan dan dipahami oleh pengguna dari berbagai latar belakang, termasuk *customer*, *admin*, dan *super admin*. Desain antarmuka difokuskan pada kemudahan navigasi antar *vault*, kejelasan informasi transaksi *Bitcoin*, serta konsistensi elemen visual untuk menciptakan pengalaman yang intuitif dan menyenangkan dalam mengelola *multisignature wallet*.

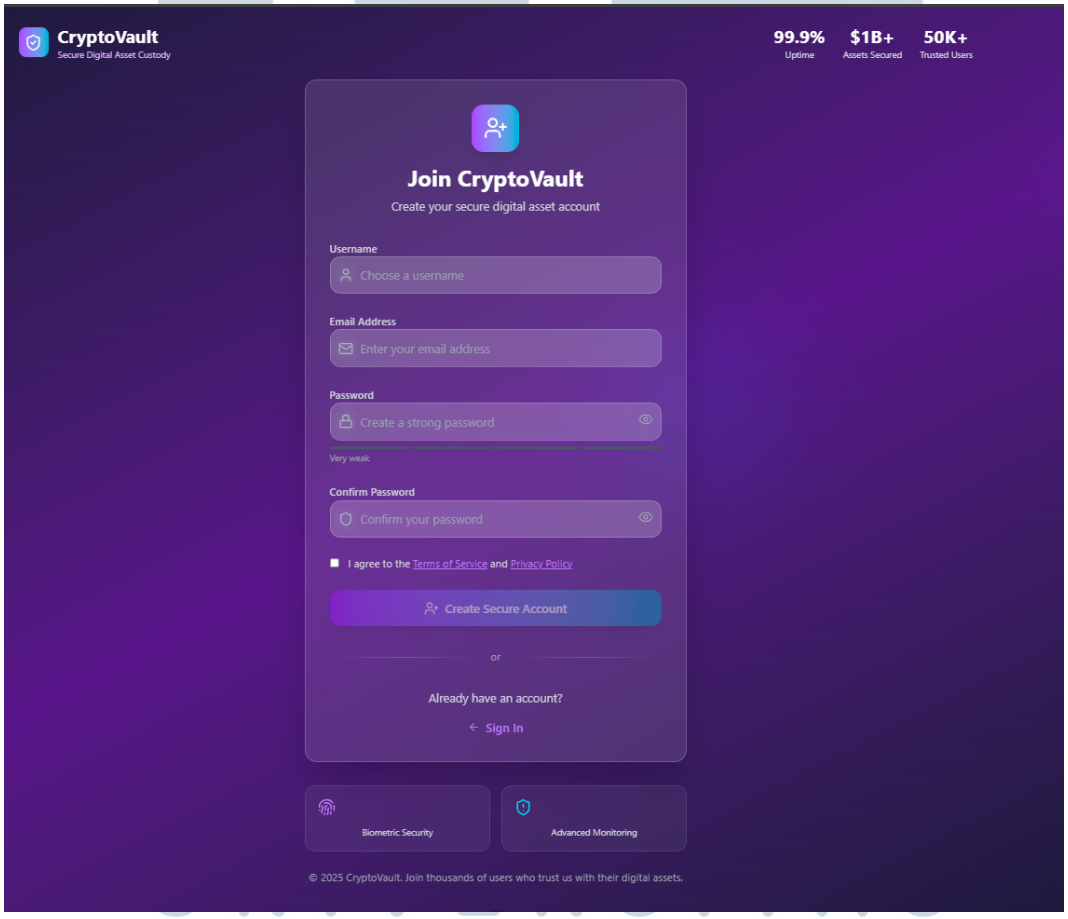


Gambar 3.5. Halaman *Login*

Halaman *login* merupakan pintu utama bagi pengguna untuk mengakses sistem *Bitcoin Custodian*. Seperti yang ditunjukkan pada gambar di atas, halaman *login* dirancang dengan antarmuka yang elegan, modern, dan aman menggunakan *gradient background* berwarna biru dan ungu yang mencerminkan tema keamanan dan kepercayaan. Desain visual menggunakan elemen-elemen animatif seperti

floating orbs dan *grid pattern* untuk menciptakan kesan profesional dan futuristik.

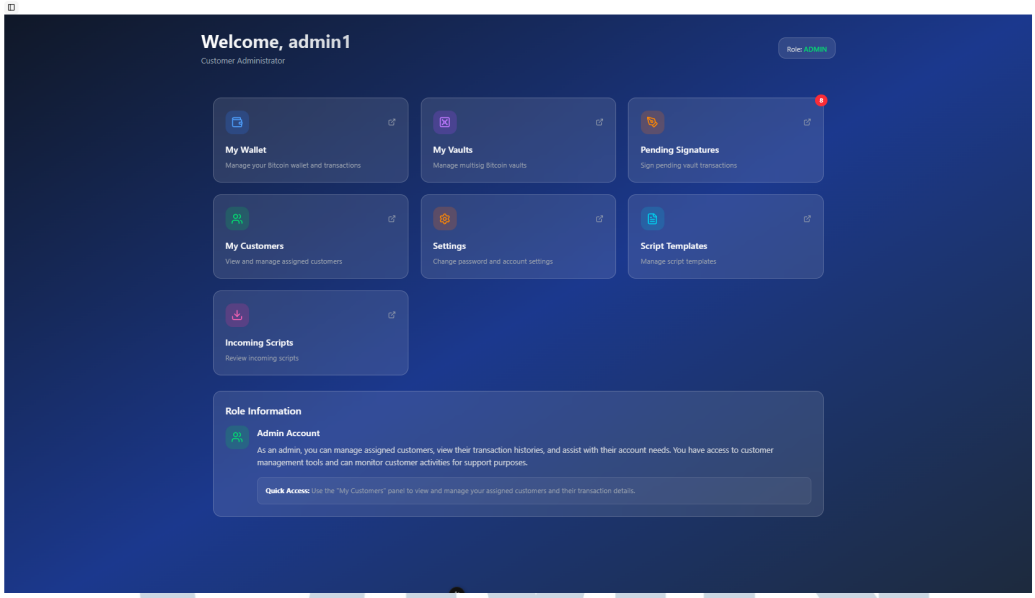
Implementasi teknis pada halaman ini menggunakan *Vue.js* dengan *Composition API*, memanfaatkan *Pinia Store* untuk manajemen *state* autentikasi, serta komponen *UI* dari *Shadcn/Nuxt* untuk konsistensi desain. Sistem validasi *form* dilakukan pada sisi klien sebelum pengiriman ke *backend*, dan respons autentikasi ditangani dengan *error handling* yang informatif. Setelah *login* berhasil, pengguna akan diarahkan ke halaman *dashboard* sesuai dengan *role* mereka (*Customer*, *Admin*, atau *Super Admin*). Keamanan *login* diperkuat dengan penggunaan *JWT* (*JSON Web Token*) di sisi *backend* dan *HTTPS* untuk enkripsi data transmisi.



Gambar 3.6. Halaman *Register*

Halaman *register* merupakan pintu masuk bagi pengguna baru untuk membuat akun di sistem *Bitcoin Custodian API*. Seperti yang ditunjukkan pada gambar di atas, halaman *register* dirancang dengan antarmuka yang intuitif dan menarik menggunakan *gradient background* berwarna ungu dan biru yang mencerminkan identitas keamanan platform.

Fitur-fitur utama yang terdapat pada halaman *register* mencakup: (1) Formulir pendaftaran dengan empat *field input* utama yaitu *Username*, *Email Address*, *Password*, dan *Confirm Password* yang dilengkapi dengan ikon dan validasi *real-time*; (2) Indikator kekuatan *password* (*password strength indicator*) yang menampilkan tingkat keamanan *password* secara visual dengan empat *level* (*Weak*, *Fair*, *Good*, *Strong*); (3) Tombol *Show/Hide Password* untuk memudahkan pengguna melihat input mereka; (4) *Checkbox* untuk penerimaan *Terms of Service* dan *Privacy Policy*; (5) Pesan *error* dan *success* yang responsif untuk memberikan *feedback* kepada pengguna; (6) Tombol *Create Secure Account* dengan status *loading* untuk umpan balik visual; (7) Tautan navigasi ke halaman *login* untuk pengguna yang sudah memiliki akun.



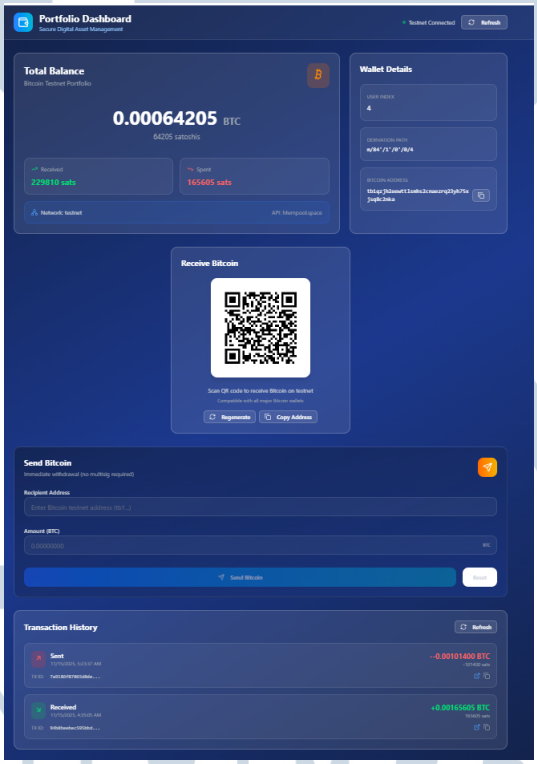
Gambar 3.7. Halaman *dashboard*

Halaman *Dashboard* merupakan halaman utama yang menjadi pusat kontrol bagi setiap pengguna setelah melakukan autentikasi ke sistem *Bitcoin Custodian API*. Seperti yang ditunjukkan pada gambar di atas yang menampilkan *dashboard* dari pengguna dengan *role Admin*, halaman ini dirancang dengan antarmuka yang intuitif menggunakan *card-based layout* yang responsif dan navigasi yang mudah. Desain visual menggunakan *gradient background* berwarna biru gelap dengan kartu-kartu fitur yang tersusun dalam *grid* 3 kolom.

Fitur-fitur yang ditampilkan pada halaman *Dashboard* berbeda-beda sesuai dengan *role* pengguna, yaitu: (1) **Fitur Umum untuk Semua *Role*** mencakup

My Wallet untuk mengelola *wallet Bitcoin* dan riwayat transaksi, *My Vaults* untuk mengelola *multisig Bitcoin vaults*, *Pending Signatures* untuk menandatangani transaksi yang menunggu dengan *badge* notifikasi merah, dan *Settings* untuk mengubah *password* dan pengaturan akun; (2) **Fitur Khusus *Admin*** mencakup *My Customers* untuk melihat dan mengelola *customer* yang ditugaskan.

Bagian bawah halaman menampilkan panel *Role Information* yang memberikan penjelasan kontekstual berdasarkan *role* pengguna. Untuk pengguna *Customer*, panel ini menjelaskan bahwa mereka dapat mengelola *wallet* dan transaksi dengan dukungan dari *admin* yang ditugaskan. Untuk pengguna *Admin*, panel menunjukkan bahwa mereka memiliki akses ke alat manajemen *customer* dan dapat memantau aktivitas *customer*. Untuk *Super Admin*, panel memberikan informasi tentang akses penuh sistem termasuk manajemen pengguna dan penugasan *customer*.

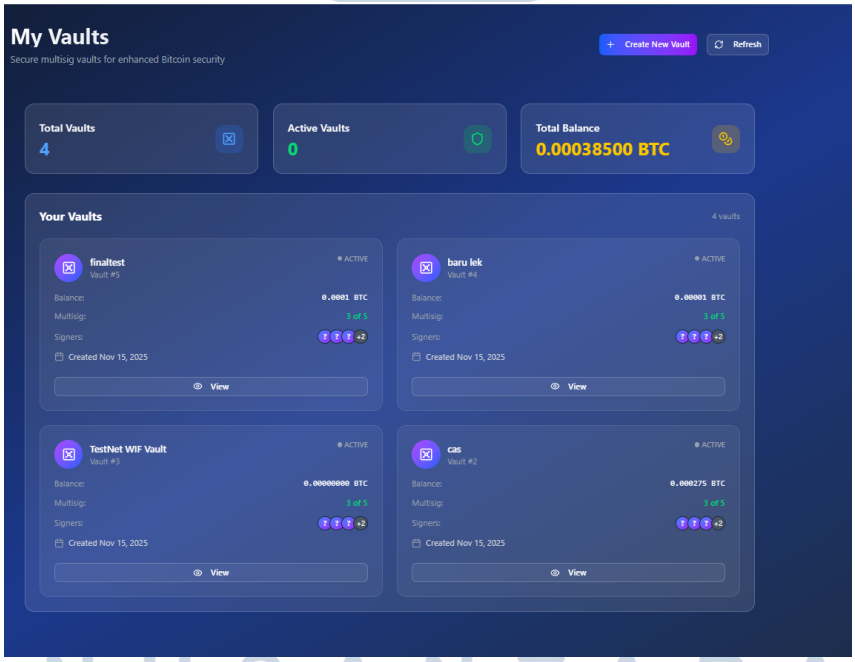


Gambar 3.8. Halaman *My wallet*

Halaman *My Wallet* merupakan halaman yang menampilkan detail lengkap *portfolio Bitcoin* pengguna dengan fokus pada manajemen aset digital. Seperti yang ditunjukkan pada gambar di atas, halaman ini dirancang untuk memberikan visibilitas penuh terhadap saldo dan transaksi *Bitcoin* pengguna dengan antarmuka

yang terstruktur dan informatif. Desain visual menggunakan *gradient background* berwarna biru gelap dengan kartu-kartu informasi yang tersusun dalam *layout grid* responsif.

Fitur-fitur utama yang terdapat pada halaman *My Wallet* mencakup: (1) Bagian *Total Balance* yang menampilkan saldo *Bitcoin* total dalam satuan *BTC* (0.00064205 *BTC*) dan *satoshi* (64205 *satoshi*), dengan pemecahan detail untuk saldo yang diterima (*received*: 223810 *sats*) dan dikirim (*spent*: 165605 *sats*); (2) Panel *Wallet Details* di sebelah kanan yang menampilkan informasi teknis seperti *user index*, *derivation path* (*m'/44'/0'/0'*), dan alamat *Bitcoin testnet* yang lengkap dengan tombol salin; (3) Seksi *Receive Bitcoin* yang menampilkan kode *QR* untuk menerima *Bitcoin*, dilengkapi dengan tombol "*Regenerate*" untuk menghasilkan alamat penerima baru dan "*Copy Address*" untuk menyalin alamat; (4) Formulir *Send Bitcoin* dengan input untuk alamat penerima dan jumlah *BTC* yang akan dikirim, serta tombol *Send Bitcoin* yang dapat disesuaikan; (5) Bagian *Transaction History* yang menampilkan riwayat transaksi terbaru dengan status (*sent/received*), waktu, jumlah, dan identitas transaksi yang dapat disalin atau dibuka di *blockchain explorer*.

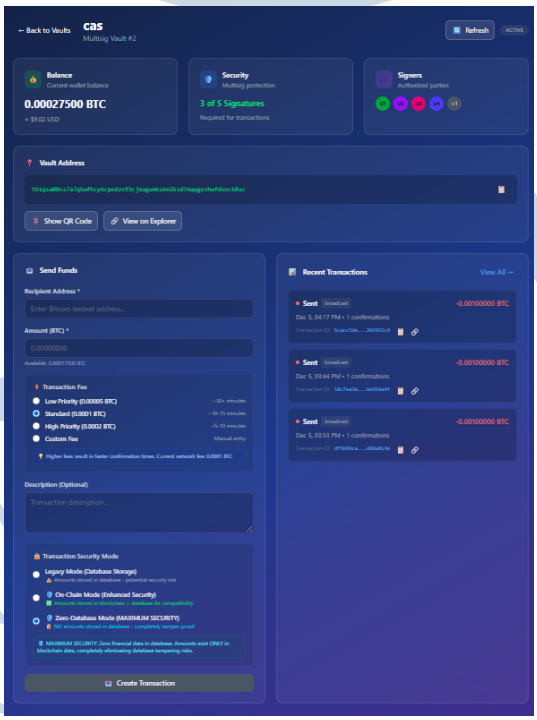


Gambar 3.9. Halaman *My Vaults*

Halaman *My Vaults* merupakan halaman yang menampilkan daftar lengkap semua *multisig Bitcoin vaults* yang dimiliki oleh pengguna. Seperti yang

ditunjukkan pada gambar di atas, halaman ini dirancang untuk memberikan gambaran umum tentang *vault-vault* pengguna dengan tampilan statistik ringkas dan kartu *vault* yang informatif. Desain visual menggunakan *gradient background* berwarna biru gelap dengan *layout* responsif yang dapat menampilkan *vault* dalam *grid* 2 kolom di perangkat *desktop*.

Fitur-fitur utama yang terdapat pada halaman *My Vaults* mencakup: (1) ***Header Section*** dengan judul halaman, deskripsi, serta tombol *Create New Vault* untuk membuat *vault* baru dan tombol *Refresh* untuk memperbarui data *vault*; (2) ***Statistics Cards*** yang menampilkan tiga metrik penting yaitu *Total Vaults* (jumlah total *vault*), *Active Vaults* (jumlah *vault* yang sedang aktif), dan *Total Balance* (total saldo *Bitcoin* di semua *vault*); (3) ***Vaults Grid*** yang menampilkan kartu-kartu *vault* dengan detail masing-masing *vault* termasuk nama *vault*, *ID*, status (*ACTIVE*), saldo dalam *BTC*, konfigurasi *multisig* (3 *of* 5), *avatar* pengguna penandatangan dengan inisial, tanggal pembuatan, dan tombol *View* serta *Manage*; (4) ***Empty State*** yang ditampilkan jika pengguna belum memiliki *vault*, dengan tombol untuk membuat *vault* pertama; (5) ***Error Handling*** dengan pesan *error* yang informatif dan tombol percobaan ulang jika terjadi kesalahan.

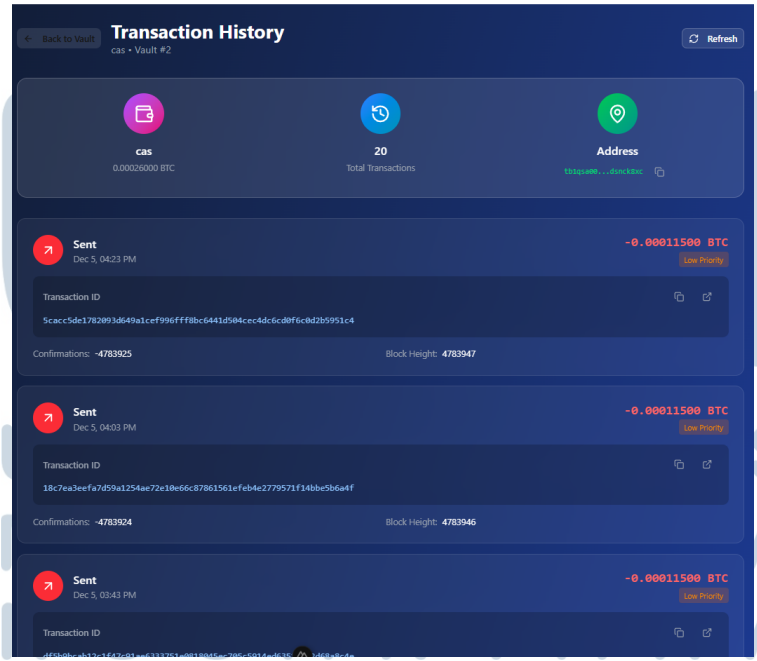


Gambar 3.10. Halaman *Vault*

Halaman detail *Vault* merupakan halaman yang menampilkan informasi

lengkap tentang sebuah *vault multisignature* tertentu serta memungkinkan pengguna untuk mengelola transaksi dan mengirim dana. Seperti yang ditunjukkan pada gambar di atas untuk *vault* bernama "*CAS*" (*Multisig Vault* 2), halaman ini dirancang dengan *layout* yang komprehensif dan informatif menggunakan *card-based design* yang tersusun dalam kolom-kolom yang responsif.

Fitur-fitur utama yang terdapat pada halaman detail *Vault* mencakup: (1) ***Header Section*** dengan nama *vault*, nomor *vault*, tombol *Back to Vaults*, tombol *Refresh*, dan *badge* status *vault*; (2) ***Statistics Cards*** yang menampilkan tiga metrik penting yaitu *Balance* (saldo *vault* dalam *BTC* dan *USD*), *Security* (konfigurasi *multisig* 3 *of* N *signature* yang diperlukan), dan *Signers* (daftar pengguna yang berwenang menandatangani transaksi dengan *avatar* warna); (3) ***Vault Address Section*** yang menampilkan alamat *Bitcoin vault* lengkap dengan tombol salin, tombol untuk menampilkan kode *QR*, dan tombol untuk melihat *vault* di *blockchain explorer*; (4) ***Send Funds Section*** yang berisi formulir untuk mengirim *Bitcoin* dengan input alamat penerima, jumlah *BTC* yang akan dikirim, pilihan *fee* transaksi (*Low Priority*, *Standard*, *High Priority*, atau *Custom*), deskripsi opsional, dan tombol *create transaction*; (5) ***Recent Transactions Section*** yang menampilkan riwayat transaksi terbaru dengan status, waktu, jumlah, dan *ID* transaksi.



Gambar 3.11. Halaman *Transaction History*

Halaman *Transaction History* merupakan halaman yang menampilkan

riwayat lengkap semua transaksi yang telah dilakukan melalui *vault* tertentu. Seperti yang ditunjukkan pada gambar di atas untuk *vault* "*CAS*" (*Vault* 2), halaman ini dirancang untuk memberikan visibilitas penuh terhadap aktivitas transaksi dengan detail yang komprehensif dan terstruktur. Desain visual menggunakan *gradient background* berwarna biru gelap dengan kartu-kartu transaksi yang tersusun secara vertikal.

Fitur-fitur utama yang terdapat pada halaman *Transaction History* mencakup: (1) ***Header Section*** dengan tombol *Back to Vault* untuk kembali ke halaman detail *vault*, judul halaman, nama dan nomor *vault*, serta tombol *Refresh* untuk memperbarui data transaksi; (2) ***Vault Summary Card*** yang menampilkan informasi ringkas *vault* dalam *grid* 3 kolom meliputi nama *vault* dengan saldo, jumlah total transaksi, dan alamat *vault* dengan tombol salin; (3) ***Transaction List*** yang menampilkan kartu-kartu transaksi individual dengan detail lengkap termasuk:

- Ikon dan tipe transaksi (*Sent Bitcoin* atau *Received Bitcoin*)
- Tanggal dan waktu transaksi serta jumlah konfirmasi
- Jumlah *Bitcoin* dalam format berwarna (merah untuk *sent*, hijau untuk *received*)
- *Transaction ID* (*TXID*) dengan tombol salin dan tombol untuk membuka di *blockchain explorer*
- Detail teknis termasuk *block height*, *fee* dalam *BTC*, dan ukuran transaksi dalam *bytes*

(4) ***State Management*** termasuk *loading state* dengan *spinner* animasi, *error state* dengan pesan *error* informatif, dan *empty state* jika *vault* belum memiliki transaksi.

Implementasi teknis pada halaman *Transaction History* menggunakan *Nuxt.js* dengan *Vue 3 Composition API*, memanfaatkan *API endpoint* untuk mengambil data transaksi dari `/vaults/:id/transactions`, menampilkan informasi *blockchain* termasuk *confirmations* dan *block height*. Setiap transaksi dapat disalin *ID*-nya untuk keperluan verifikasi, dan dapat dibuka di *Mempool Explorer* untuk melihat detail transaksi di *blockchain explorer* publik. Halaman ini juga menyediakan mekanisme *refresh* manual untuk memperbarui status konfirmasi dan data transaksi terbaru, serta *formatting* yang jelas untuk jumlah dan data teknis transaksi.

Gambar 3.12. Halaman *Pending Signature*

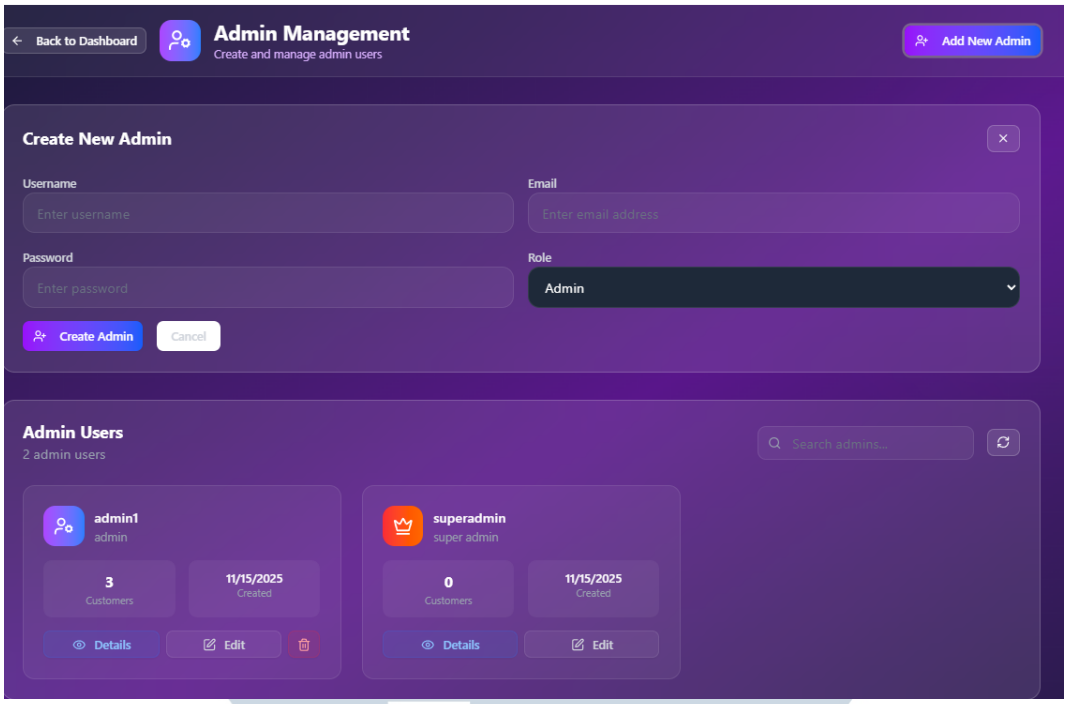
Halaman *Pending Signatures* merupakan halaman yang menampilkan daftar transaksi *vault* yang menunggu penandatanganan dari pengguna. Seperti yang ditunjukkan pada gambar di atas, halaman ini dirancang untuk memberikan visibilitas lengkap terhadap status penandatanganan transaksi *multisig* dengan informasi detail dan *progress tracking* yang jelas. Desain visual menggunakan *gradient background* berwarna biru gelap dengan kartu-kartu transaksi yang tersusun vertikal.

Fitur-fitur utama yang terdapat pada halaman *Pending Signatures* mencakup: (1) ***Header Section*** dengan judul halaman, deskripsi, tombol *Refresh* untuk

memperbarui data, dan *badge* yang menampilkan jumlah transaksi yang menunggu; (2) ***Transaction Cards*** yang menampilkan setiap transaksi dengan detail lengkap termasuk:

- Nama *vault* yang melakukan *transfer* dan informasi pembuatan transaksi
- Status transaksi (*SIGNED*, *PENDING*, dll) dalam *badge* berwarna
- Jumlah *Bitcoin* yang akan dikirim dengan indikator keamanan *Zero-DB* jika berlaku
- Alamat penerima dengan tombol salin
- Deskripsi opsional transaksi
- *Progress bar* yang menampilkan persentase penandatanganan (misalnya 3/3 untuk transaksi yang siap *broadcast*)
- Daftar lengkap penandatangan dengan status masing-masing (*Signed* dalam warna hijau atau *Pending* dalam warna kuning)
- Nama penandatangan dengan indikator ”(*You*)” untuk pengguna saat ini

(3) ***Action Buttons*** yang ditampilkan berdasarkan status penandatangan pengguna, termasuk tombol *Sign Transaction* untuk menandatangani transaksi jika belum ditandatangani oleh pengguna saat ini, dan tombol *Broadcast to Bitcoin Network* untuk mengirim transaksi ke jaringan *Bitcoin* jika semua tanda tangan telah terkumpul; (4) ***State Management*** termasuk *loading state* dengan *spinner*, *error state* dengan opsi coba lagi, dan *empty state* jika tidak ada transaksi yang menunggu.



Gambar 3.13. Halaman *Admin Management*

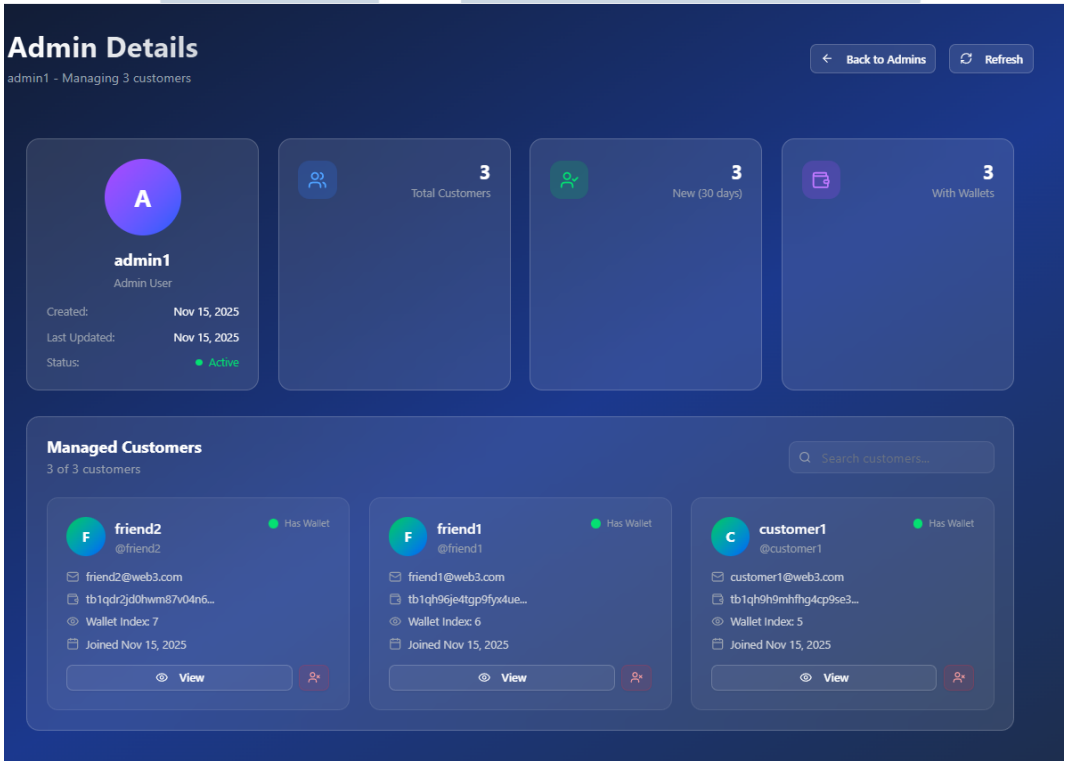
Halaman *Admin Management* merupakan halaman khusus yang hanya dapat diakses oleh pengguna dengan *role Super Admin*, berfungsi untuk mengelola dan memantau semua pengguna *admin* dalam sistem. Seperti yang ditunjukkan pada gambar di atas, halaman ini dirancang dengan antarmuka yang komprehensif untuk memberikan kontrol penuh atas administrasi pengguna *admin*. Desain visual menggunakan *gradient background* berwarna ungu gelap dengan *layout* yang terstruktur dan responsif.

Fitur-fitur utama yang terdapat pada halaman *Admin Management* mencakup: (1) ***Header Section*** dengan tombol kembali ke *dashboard*, ikon dan judul halaman, serta tombol *Add New Admin* untuk membuat *admin* baru; (2) ***Create Admin Form*** yang muncul ketika tombol *Add New Admin* diklik, berisi *field* input untuk *username*, *email*, *password*, dan *role selection*, dengan tombol *Create Admin* dan *Cancel*; (3) ***Admin List Section*** yang menampilkan daftar semua *admin* dalam bentuk kartu *grid* dengan fitur:

- Ikon dan nama *admin* dengan *role badge* (*Admin* atau *Super Admin*)
- Indikator visual yang berbeda untuk *Super Admin* (ikon mahkota, warna merah-oranye) dan *Admin* biasa (ikon *user cog*, warna ungu-biru)
- Jumlah *customer* yang ditugaskan kepada *admin* tersebut

- Tanggal pembuatan akun *admin*
- Tombol *Details* untuk melihat informasi lengkap *admin*
- Tombol *Edit* untuk mengubah informasi *admin*
- Tombol *delete* (*trash icon*) untuk menghapus *admin*

(4) ***Search and Filter*** dengan kolom pencarian untuk mencari *admin* berdasarkan nama atau *email*, serta tombol *Refresh* untuk memperbarui daftar *admin*; (5) ***State Management*** termasuk *loading state* dengan *spinner* animasi, *error state* dengan pesan informatif, dan *handling* untuk *form submission*.



Gambar 3.14. Halaman *Admin Details*

Halaman *Admin Details* merupakan halaman yang menampilkan informasi lengkap dan detail tentang seorang *admin* tertentu, termasuk profil *admin*, statistik, dan daftar *customer* yang ditugaskan kepadanya. Halaman ini hanya dapat diakses oleh *Super Admin* untuk *monitoring* dan manajemen *admin* yang lebih mendalam. Seperti yang ditunjukkan pada gambar di atas untuk *admin* "*admin1*", halaman ini dirancang dengan *layout* yang komprehensif dan informatif menggunakan *card-based design*.

Fitur-fitur utama yang terdapat pada halaman *Admin Details* mencakup: (1) ***Header Section*** dengan judul halaman, deskripsi yang menunjukkan nama *admin* dan jumlah *customer* yang dikelola, serta tombol *Back to Admins* dan *Refresh* untuk kembali ke daftar *admin* atau memperbarui data; (2) ***Admin Profile Card*** yang menampilkan:

- *Avatar* dengan *initial* nama *admin* dalam *background gradient*
- Nama *admin* dan *role badge* ("*Admin User*")
- Tanggal pembuatan akun *admin*
- Tanggal *last updated*
- Status aktif dengan indikator warna hijau

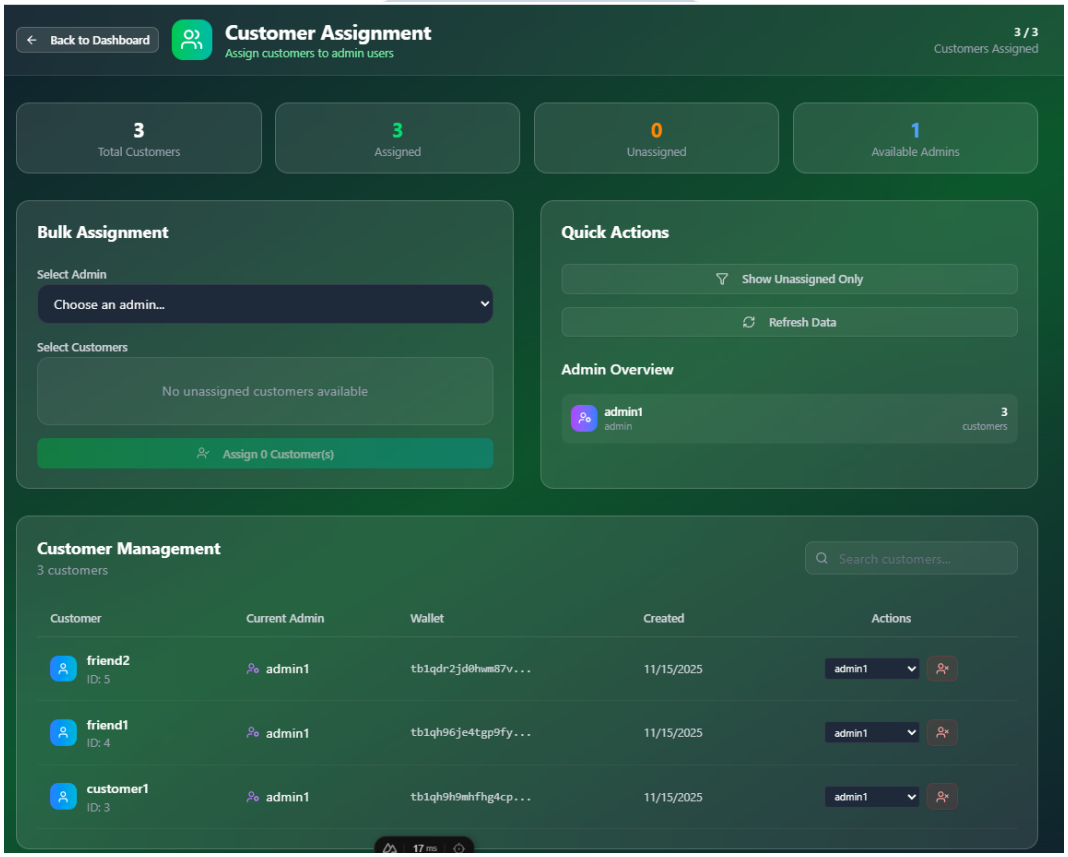
(3) ***Statistics Cards*** yang menampilkan tiga metrik penting dalam *grid* 3 kolom:

- *Total Customers*: Jumlah total *customer* yang ditugaskan (contoh: 3)
- *New* (30 *days*): Jumlah *customer* baru dalam 30 hari terakhir (contoh: 3)
- *With Wallets*: Jumlah *customer* yang telah membuat *wallet* (contoh: 3)

(4) ***Managed Customers Section*** yang menampilkan daftar lengkap *customer* yang ditugaskan kepada *admin* dengan fitur:

- Kolom pencarian untuk mencari *customer* berdasarkan nama atau *email*
- Kartu *customer* dengan *avatar*, nama, *username*, dan status *wallet* (*Has Wallet* atau *No Wallet*)
- *Email customer* yang terdiskoneksi
- *Wallet address Bitcoin customer* (*partial/truncated* untuk keamanan)
- *Wallet index* untuk *customer*
- Tanggal *customer* bergabung dengan platform
- Tombol *View* untuk melihat detail *customer* lebih lanjut
- Tombol *delete* (*trash icon*) untuk menghapus *customer* dari *admin*

(5) ***Empty State*** yang ditampilkan jika *admin* belum memiliki *customer* yang ditugaskan, dengan tombol untuk menuju halaman *Assign Customers*.



Gambar 3.15. Halaman *Assign Customer*

Halaman *Customer Assignment* merupakan halaman khusus *Super Admin* yang dirancang untuk mengelola penugasan *customer* kepada *admin users* secara efisien. Halaman ini memungkinkan *Super Admin* untuk melakukan penugasan massal *customer*, melihat status penugasan, dan mengelola hubungan antara *customer* dan *admin*. Seperti yang ditunjukkan pada gambar di atas, halaman ini menggunakan *gradient background* warna hijau-*teal* dengan *layout* yang terstruktur.

Fitur-fitur utama yang terdapat pada halaman *Customer Assignment* mencakup: (1) ***Header Section*** dengan tombol kembali ke *dashboard*, ikon dan judul halaman, serta tampilan statistik penugasan *customer* (3/3 *Customers Assigned*); (2) ***Quick Statistics Cards*** yang menampilkan empat metrik penting dalam *grid* 4 kolom:

- *Total Customers*: Jumlah total *customer* di sistem

- *Assigned*: Jumlah *customer* yang sudah ditugaskan ke *admin*
- *Unassigned*: Jumlah *customer* yang belum ditugaskan
- *Available Admins*: Jumlah *admin* yang tersedia untuk penugasan

(3) ***Bulk Assignment Section*** yang memungkinkan *Super Admin* untuk menugaskan *multiple customer* sekaligus kepada satu *admin* dengan fitur:

- *Dropdown* untuk memilih *admin target*
- *Checkbox list* untuk memilih *customer* yang ingin ditugaskan
- Tombol *Assign* yang menampilkan jumlah *customer* yang akan ditugaskan

(4) ***Quick Actions Panel*** yang menyediakan:

- Tombol untuk menampilkan/menyembunyikan hanya *customer* yang belum ditugaskan
- Tombol *Refresh Data* untuk memperbarui data *assignment*
- *Admin Overview* yang menampilkan daftar *admin* dengan jumlah *customer* yang ditugaskan

(5) ***Customer Management Table*** yang menampilkan daftar lengkap *customer* dalam format tabel dengan kolom:

- *Avatar* dan nama *customer*
- *Admin* yang mengelola *customer* (dengan *dropdown* untuk mengubah *admin*)
- *Wallet address customer* (*truncated*)
- Tanggal *customer* bergabung
- *Action buttons* (*view*, *delete*)

(6) ***Search Functionality*** untuk mencari *customer* berdasarkan nama atau *email*.

3.4 Alur Pengambilan dan Pengolahan Data pada Sistem

Untuk memahami bagaimana sistem *Bitcoin Custodian* berfungsi secara keseluruhan, penting untuk memahami alur perjalanan data dari antarmuka pengguna hingga ke basis data dan *blockchain*. Bagian ini menjelaskan arsitektur *data flow*, mekanisme komunikasi antar komponen sistem, dan proses pengolahan data pada setiap halaman utama.

3.4.1 Arsitektur Data Flow Sistem Umum

Sistem *Bitcoin Custodian* menggunakan arsitektur *client-server* yang terdiri dari empat komponen utama:

1. ***Frontend* (*Nuxt.js* + *Vue.js* 3)**: Menampilkan antarmuka pengguna, mengirim *HTTP request* ke *API Backend*, dan mengelola *state* aplikasi menggunakan *Pinia Store*.
2. ***API Backend* (*Node.js* + *Express.js*)**: Menerima *HTTP request*, memverifikasi autentikasi dengan *JWT*, mengakses *database MySQL* melalui *Prisma ORM*, dan menerapkan *role-based access control*.
3. ***Database* (*MySQL* + *Prisma ORM*)**: Menyimpan data *user*, *vault*, transaksi, dan *signature* secara persisten dengan relasi yang kompleks.
4. ***Bitcoin Testnet***: *Network blockchain* untuk *testing* dengan logika kriptografi *Bitcoin* sesungguhnya, diakses melalui *bitcoinjs-lib* untuk operasi *wallet* dan *transaction signing*.

Alur komunikasi umum: *Frontend* $\rightarrow$ *Backend* (*Verifikasi JWT*) $\rightarrow$ *Database/Blockchain* $\rightarrow$ *Backend* $\rightarrow$ *Frontend*.

3.4.2 Data Flow untuk Halaman Login

A Request Flow

Endpoint `POST /login`

Request { `login: string, password: string` }

Response { `token: string, user: { id, username, email, role }` }

Alur Proses:

1. Pengguna mengisi *form login* dengan *username/email* dan *password* di halaman `login.vue`
2. *Frontend* memanggil `auth.login()` dari *Pinia Store* dan mengirim `POST /login`
3. *Backend* menerima *request* di *endpoint* `POST /login`
4. *Backend* melakukan *query database*: `User.findUnique()` untuk mencari *user*
5. *Backend* verifikasi *password* menggunakan `bcryptjs.compare()`
6. Jika valid, *backend generate JWT token* dengan *payload* { `userId, username, role` }
7. *Backend return response* dengan *token* dan *user info*
8. *Frontend* menyimpan *token* di *Pinia Store* dan *localStorage* via *pinia-plugin-persistedstate*
9. *Frontend redirect* pengguna ke halaman `/dashboard`

3.4.3 Data Flow untuk Halaman Dashboard

A Request Flow

Endpoint `GET /vaults/pending-transactions`

Request Headers: `Authorization: Bearer <jwt_token>`

Response { `success: true, data: PendingTransaction[]` }

Alur Proses:

1. Halaman `dashboard.vue` dimuat, mengambil *user data* dari *Pinia Store* (dari *login* sebelumnya)
2. *Dashboard* menampilkan *user greeting* dan *role badge* langsung dari *store* (tidak perlu *fetch* ulang)
3. *Frontend* mengirim `GET /vaults/pending-transactions` untuk *fetch badge* notifikasi
4. *Backend middleware* verifikasi *JWT token* pada *header Authorization*
5. *Backend extract userId* dari *token* dan *query database*: `PendingTransaction.findMany()` *with status PENDING/PARTIAL/SIGNED*
6. *Backend apply RBAC filter* berdasarkan *user role*
7. *Backend return transaction list* dengan *count*
8. *Frontend* menerima *response* dan *render badge* dengan *count pending transactions*
9. *Dashboard* menampilkan *navigation cards* ke fitur utama: *My Wallet*, *My Vaults*, *Pending Signatures*, *Admin panels*

3.4.4 Data Flow untuk Halaman My Wallet

A Request Flow

Endpoint `GET /wallets/my-wallet` dan `GET /balance/wallet` (*parallel requests*)

Request Headers: `Authorization: Bearer <jwt_token>`

Response `{ wallet: ..., balance: { confirmedBTC, satoshi, transactions: [...] } }`

Alur Proses:

1. Halaman `wallet.vue` dimuat dengan `loading = true`
2. *Frontend* melakukan dua *HTTP requests* secara paralel untuk efisiensi:
   - *Request* 1: `GET /wallets/my-wallet` untuk mengambil *wallet metadata* dari *database*
   - *Request* 2: `GET /balance/wallet` untuk mengambil *balance real-time* dari *Bitcoin testnet*
3. *Backend Request* 1 *processing*: *Extract userId* dari *JWT*, *query* `User.findUnique()` dengan *wallet relation*, *remove private key* dari *response*
4. *Backend Request* 2 *processing*: *Query Bitcoin node* untuk *UTXO* di *wallet address*, hitung *confirmed balance*, *query database* untuk *transaction history*
5. *Frontend* menggunakan `Promise.all()` untuk menunggu kedua *request* selesai
6. *Frontend update component state* dengan *wallet* dan *balance data*
7. Set `loading = false` dan *template re-render* dengan data terbaru
8. Tampilkan: *Total balance* (*BTC/satoshi*), *wallet address* dengan *QR code*, *receive form*, *send form*, *transaction history*

3.4.5 Data Flow untuk Halaman My Vaults

A Request Flow

Endpoint `GET /vaults/my-vaults`

Request Headers: `Authorization: Bearer <jwt_token>`

Response { `success: true, data: Vault[] with owner & signers relation` }

Alur Proses:

1. Halaman `vaults/index.vue` dimuat dengan `loading = true`
2. *Frontend* mengirim `GET /vaults/my-vaults` dengan *JWT token*
3. *Backend middleware* verifikasi *token* dan *extract userId*, *role*
4. *Backend* menerapkan *Role-Based Access Control*:
   - *SUPER_ADMIN*: `whereClause = {}` (lihat semua *vault*)
   - *ADMIN*: *Filter vault* dimana *admin* adalah *signer* atau *customer owner* memiliki *adminId = userId*
   - *CUSTOMER*: `whereClause = { ownerId: userId }` (hanya *vault* milik sendiri)
5. *Backend query*: `Vault.findMany()` dengan *include owner* dan *signers relation*
6. *Backend return vault list* dengan relasi data
7. *Frontend* menerima *response* dan *calculate statistics*: *Total Vaults*, *Active Vaults*, *Total Balance*
8. *Frontend render vault cards* dalam *grid layout* dengan: nama, *ID*, status, *balance*, *signers avatars*, *created date*, *action buttons*
9. *Display empty state* jika tidak ada *vault*

3.4.6 Data Flow untuk Halaman Vault Detail

A Request Flow

Endpoint `GET /vaults/:id`, `GET /vaults/:id/balance`, `GET /vaults/:id/signers`, `GET /vaults/:id/transactions?limit=5`

Request *Headers*: `Authorization: Bearer <jwt_token>`

Response *Vault object* dengan *owner*, *signers*, *balance*, dan *recent transactions*

Alur Proses:

1. *Component* `vaults/[id]/index.vue` menerima *route parameter* `id` dari *URL*
2. *Frontend* melakukan empat *HTTP requests* secara paralel:
   - *GET* /vaults/:id untuk detail *vault*
   - *GET* /vaults/:id/balance untuk *balance* dari *blockchain*
   - *GET* /vaults/:id/signers untuk daftar penanda tangan
   - *GET* /vaults/:id/transactions?limit=5 untuk 5 transaksi terakhir
3. *Backend processing*: Setiap *endpoint* melakukan verifikasi *JWT*, *authorization check*, *query database/blockchain* sesuai dengan *endpoint*
4. *Backend return combined response* dengan *vault detail*, *balance*, *signers list*, dan *recent transactions*
5. *Frontend* menggunakan `Promise.all()` untuk menunggu semua 4 *requests complete*
6. *Frontend combine* semua *response data* ke *component state*
7. *Display*: *Header* dengan *vault info*, *statistics cards* (*Balance*, *Security*, *Signers*), *vault address* dengan *QR code*, *send funds form* dengan *fee options*, *recent transactions list*

3.4.7 Data Flow untuk Halaman Transaction History

A Request Flow

Endpoint `GET /vaults/:id/transactions?page=1&limit=20`

Request *Headers*: `Authorization: Bearer <jwt_token>`

Response `{ success: true, data: Transaction[], pagination: ... }`

Alur Proses:

1. *Component* `vaults/[id]/transactions.vue` dimuat dengan *route param* `id`
2. *Frontend* mengirim `GET /vaults/:id/transactions?page=1&limit=20`
3. *Backend middleware* verifikasi *JWT*, *extract vaultId* dari *route param*
4. *Backend authorization check*: Verifikasi pengguna punya akses ke *vault* ini
5. *Backend query*: `PendingTransaction.findMany()` dengan *pagination*, *filter vaultId*, *orderBy createdAt DESC*
6. *Backend enrich* setiap transaksi dengan data *blockchain*: *block height*, *confirmations*, *fee* dari *Bitcoin node*
7. *Backend return paginated transaction list* dengan *blockchain data*
8. *Frontend* menerima *response* dan *store* ke *component state*
9. *Display*: *Header* dengan *back button*, judul, *vault name*, *refresh button*; *Vault summary card*; *Transaction list cards* dengan *type icon*, *timestamp*, *amount* (*color coded*), *block height*, *confirmations*, *TXID*; *Pagination controls*

3.4.8 Data Flow untuk Halaman Pending Signatures

A Request Flow

Endpoint `GET /vaults/pending-transactions`, `POST /vaults/transactions/:id/sign`, `POST /vaults/transactions/:id/broadcast`

Request GET *Headers*: `Authorization: Bearer <jwt_token>`

Request POST *Request body* dengan *signature data* dan *broadcast payload*

Alur Proses:

1. Halaman `signatures.vue` dimuat dengan `loading = true`
2. *Frontend* mengirim `GET /vaults/pending-transactions`
3. *Backend query*: `PendingTransaction.findMany()` dengan status *PENDING/PARTIAL/SIGNED* dan *include vault*, *signatures* dengan *signer relation*
4. *Backend filter*: Hanya *return* transaksi dimana pengguna adalah *required signer*
5. *Backend calculate* untuk setiap transaksi: jumlah *signature* terkumpul, status per *signer*
6. *Backend return enhanced transaction list* dengan *progress info*
7. *Frontend group transactions* berdasarkan status dan *display* dalam *tab/filtered view*
8. *User* klik "*Sign Transaction*" → *Frontend show confirmation dialog*
9. *User confirm* → *Frontend send* `POST /vaults/transactions/:id/sign` dengan *signature data*
10. *Backend validate signature*, *store* ke *database* `TransactionSignature.create()`, *update transaction status* jika semua *signatures* terkumpul
11. *Frontend show success notification*, *refresh transaction list*
12. *User* klik "*Broadcast to Bitcoin Network*" (hanya jika semua *signatures* ada) → *Frontend send* `POST /vaults/transactions/:id/broadcast`
13. *Backend combine signatures*, *verify transaction integrity*, *broadcast* ke *Bitcoin testnet*, *store txHash*, *update transaction status* ke *BROADCAST*
14. *Frontend show success notification* dengan *blockchain explorer link*

3.4.9 Data Flow untuk Halaman Admin Management

A Request Flow

Endpoint `GET /admin/admins`, `POST /admin/admins`, `PUT /admin/admins/:id`, `DELETE /admin/admins/:id`

Request GET *Headers*: `Authorization: Bearer <jwt_token>`

Response *Admin list* atau *individual admin with customer count*

Alur Proses:

1. Halaman `super-admin/manage-admins.vue` dimuat (*middleware check isSuperAdmin*)
2. *Frontend* mengirim `GET /admin/admins`
3. *Backend query*: `User.findMany({ where: { role: 'ADMIN' } })`
4. *Backend* untuk setiap *admin*, *count customers*: `User.count({ where: { adminId: admin.id } })`
5. *Backend return admin list* dengan *customer count*
6. *Frontend display admin cards* dalam *grid* dengan: *avatar*, *name*, *role badge*, *customer count*, *created date*, *action buttons*
7. *User* klik "*Add New Admin*" → *Frontend show create form modal*
8. *User submit* → *Frontend send* `POST /admin/admins` dengan *username*, *email*, *password*, *role*
9. *Backend create user* di *database* dengan *role ADMIN*

3.4.10 Data Flow untuk Halaman Customer Assignment

A Request Flow

Endpoint `GET /admin/statistics`, `GET /admin/admins`, `GET /admin/customers`, `POST /admin/assign-customers`

Request GET *Headers*: `Authorization: Bearer <jwt_token>`

Request POST *Body*: { `adminId: number, customerIds: [number, ...]` }

Alur Proses:

1. Halaman `super-admin/assign-customers.vue` dimuat (*middleware check isSuperAdmin*)
2. *Frontend* melakukan *parallel requests*: *GET statistics*, *admins*, *customers*
3. *Backend GET statistics*: *Query* `User.count()` untuk total, *count where adminId IS NULL* untuk *unassigned*
4. *Backend GET admins*: *Query* `User.findMany({ where: { role: 'ADMIN' } })`
5. *Backend GET customers*: *Query* `User.findMany({ where: { role: 'CUSTOMER' } })` dengan *current admin assignment*
6. *Frontend combine response data* dan *render*: *Quick statistics cards* (*Total*, *Assigned*, *Unassigned*, *Available Admins*), *admin dropdown*, *customer checkbox list*, *assign button*, *customer table*
7. *User select admin target* dan *check multiple customers*
8. *User click* "*Assign*" *button* → *Frontend send* `POST /admin/assign-customers` dengan *adminId* dan *selected customerIds*
9. *Backend loop through customerIds* dan *update* setiap *customer*: `User.update({ where: { id: customerId }, data: { adminId } })`
10. *Backend return success response* dengan *count of updated records*
11. *Frontend refresh data* dan *show success notification*: "3 *customers assigned successfully*"

3.4.11 Data Flow untuk Halaman Register

A Request Flow

Endpoint `POST /register`

Request { `username: string, email: string, password: string` }

Response { `message: string, user: { id, username, email, role }` }

Alur Proses:

1. Pengguna mengakses halaman `register.vue` tanpa *authentication*
2. Pengguna mengisi *form*: *username*, *email*, *password*, *confirm password*
3. *Frontend validate form*: *email format*, *password length* minimal 6 karakter, *password match*
4. *Frontend validate terms acceptance checkbox*
5. Jika valid, *frontend send* `POST /register` dengan *username*, *email*, *password*
6. *Backend* menerima *request* dan *validate input* (*email format*, *password strength*)
7. *Backend check duplicate*: *Query* `User.findFirst()` untuk cek *username* dan *email* sudah ada
8. *Backend hash password* menggunakan *bcrypt* dengan *salt* 10-12
9. *Backend create user* di *database*: `User.create({ username, email, password: hashed, role: 'CUSTOMER' })`
10. *Backend return response* dengan *user info* (tanpa *password*)
11. *Frontend show success notification*: "*Account created successfully! Redirecting to login...*"
12. *Frontend clear form* dan *redirect* ke `/login` setelah 2 detik

3.4.12 Data Flow untuk Halaman Change Password

A Request Flow

Endpoint `POST /user/change-password`

Request { `oldPassword: string, newPassword: string` }

Response { `success: true, message: string` }

Alur Proses:

1. Halaman `change-password.vue` dimuat dengan *middleware auth check*
2. Pengguna mengisi *form*: *current password*, *new password*, *confirm new password*
3. *Frontend validate* menggunakan *Zod schema*: *password fields filled*, *passwords match*, *new password min* 6 karakter
4. Jika valid, *frontend send* `POST /user/change-password` dengan *oldPassword* dan *newPassword*
5. *Backend middleware* verifikasi *JWT token* dari *header Authorization*
6. *Backend extract userId* dari *token*
7. *Backend query database*: `User.findUnique({ where: { id: userId } })`
8. *Backend* verifikasi *oldPassword* menggunakan `bcryptjs.compare()` dengan *stored hash*
9. Jika *old password* cocok, *backend hash newPassword* dan *update*: `User.update({ data: { password: newHash } })`
10. *Backend return success response*
11. *Frontend show success toast notification*: "*Password has been successfully changed*"
12. *Frontend redirect* ke `/logout` setelah 5 detik atau *user click OK*

3.4.13 Data Flow untuk Halaman Admin Customers Management

A Request Flow

Endpoint `GET /admin/customers`

Request Headers: `Authorization: Bearer <jwt_token>`

Response { `success: true, data: User[] with admin assignment` }

Alur Proses:

1. Halaman `admin/customers.vue` dimuat (*middleware check isAdmin*)
2. *Frontend* mengirim `GET /admin/customers`
3. *Backend middleware verify JWT*, *extract userId* dan *role*
4. *Backend authorization check*: *Verify user is ADMIN* atau *SUPER_ADMIN*
5. *Backend query* berdasarkan *role*:
   - *SUPER_ADMIN*: *Query* semua *customers*: `User.findMany({ where: { role: 'CUSTOMER' } })`
   - *ADMIN*: *Query customers assigned* ke *admin* ini: `User.findMany({ where: { adminId: userId, role: 'CUSTOMER' } })`
6. *Backend include* untuk setiap *customer*: *admin info*, *vaults count*, *created date*
7. *Backend return customer list* dengan *admin assignment info*
8. *Frontend* menerima *response* dan *render customer table* dengan *columns*: *ID*, *Name*, *Email*, *Admin Assigned*, *Vaults Count*, *Created Date*, *Actions*
9. *Frontend support search/filter functionality* untuk *find customers*
10. *User* dapat klik *customer row* untuk *see details* atau *manage assignments*

3.5 Keamanan dan Best Practices

3.5.1 Mekanisme Keamanan

Sistem menerapkan keamanan berlapis pada setiap *layer*:

1. ***Authentication***: *JWT token-based authentication* dengan *expiration* 24 jam, *stored* di *secure localStorage*
2. ***Authorization***: *Role-Based Access Control* (*RBAC*) dengan 3 *roles* - *CUSTOMER*, *ADMIN*, *SUPER_ADMIN*
3. ***Transport Security***: *HTTPS* dengan *TLS* 1.2+ untuk enkripsi data *in-transit*
4. ***Password Security***: *Password* di-*hash* menggunakan *bcrypt* dengan *salt rounds* 10-12
5. ***Data Protection***: *Private keys encrypted* dengan *AES-256*, *sensitive data verified* dari *blockchain* dengan *SHA-256 hashing*
6. ***Input Validation***: *Client-side validation* untuk *UX feedback*, *server-side validation* untuk *security*
7. ***SQL Injection Prevention***: *Prisma parameterized queries prevent SQL injection*
8. ***XSS Prevention***: *Vue.js automatic template escaping*

3.5.2 Error Handling

Setiap *HTTP response* mengikuti *status code standard*:

- **200 *OK***: *Request* berhasil diproses
- **400 *Bad Request***: *Input validation error* atau *malformed request*
- **401 *Unauthorized***: *Token invalid*, *expired*, atau *missing*
- **403 *Forbidden***: *User* tidak punya *permission* ke *resource*
- **404 *Not Found***: *Resource* tidak ditemukan di *database*
- **500 *Internal Server Error***: *Unexpected server error*

Frontend menampilkan *user-friendly error messages* dan *retry options* sesuai dengan *error type*.

3.5.3 Data Integrity

- ***Foreign Keys***: *Database cascade delete* untuk *referential integrity*
- ***Unique Constraints***: *Email unique*, *vault address unique*, *prevent duplicates*
- ***Zero-Database Security***: *Sensitive transaction data* (*amount*, *fee*) *stored as NULL*, *verified* dari *blockchain PSBT data*
- ***Timestamp Tracking***: *createdAt* dan *updatedAt* untuk *audit trail*
- ***Multisig Requirement***: 3 *of* 5 *signatures required* untuk *transaction approval*

3.5.4 Tools dan Teknologi yang Digunakan

Dalam pengembangan sistem *Bitcoin Custodian* di PT Mitra Integrasi Digital, mahasiswa magang memanfaatkan berbagai *tools* dan teknologi pendukung untuk meningkatkan efisiensi kerja serta menjaga kualitas hasil pengembangan. Pemilihan *tools* dan teknologi dilakukan untuk menunjang proses *development*, pengelolaan basis data, serta kolaborasi tim selama pelaksanaan proyek. Adapun *tools* dan teknologi yang digunakan adalah sebagai berikut:

1. ***Visual Studio Code* (*VS Code*)**
   Visual Studio Code digunakan sebagai lingkungan pengembangan utama dalam proses penulisan dan pengelolaan kode sumber pada proyek *Bitcoin Custodian*. *Code editor* ini bersifat ringan dan mendukung berbagai ekstensi yang membantu proses pengembangan, seperti integrasi *Git*, fitur *auto-complete*,*debugging*, serta *REST Client* untuk pengujian layanan *API*. Dukungan tersebut mempermudah proses pengembangan dan pengujian aplikasi secara terintegrasi.
2. ***DBeaver***
   DBeaver dimanfaatkan sebagai alat bantu dalam pengelolaan dan pemantauan basis data proyek. Aplikasi ini digunakan untuk mengakses struktur *database* yang dikelola melalui *Prisma ORM*, menampilkan relasi antar tabel, serta mengeksekusi *query SQL*. Antarmuka grafis yang disediakan oleh *DBeaver* membantu proses analisis data dan *debugging* selama pengembangan sistem.

3. ***Git* dan *Git Repository***

   Git digunakan sebagai sistem kontrol versi untuk mengelola perubahan kode sumber selama proses pengembangan. Seluruh kode proyek disimpan pada *repository Git* yang diakses melalui jaringan *VPN* internal perusahaan. Melalui *Git*, mahasiswa magang dapat melakukan *commit*, *push*, dan sinkronisasi perubahan kode secara berkala. Penggunaan sistem ini juga mendukung kolaborasi tim, pencatatan riwayat pengembangan, serta memungkinkan proses *rollback* apabila terjadi kesalahan pada versi tertentu.

4. ***Nuxt.js* dan *Vue.js***

   Nuxt.js adalah *framework* modern berbasis *Vue.js* yang digunakan untuk mengembangkan *frontend* aplikasi dengan fitur *server-side rendering*, *routing* otomatis, dan *API integration* yang *powerful*. *Vue.js* sendiri adalah *JavaScript framework* yang memungkinkan pembuatan antarmuka pengguna interaktif dengan sistem *reactive data binding*, yang sangat cocok untuk aplikasi *single-page* yang responsif.

5. ***Express.js***

   Express.js adalah *web framework* untuk *Node.js* yang ringan dan fleksibel, digunakan untuk mengembangkan *API backend Bitcoin Custodian*. *Express.js* menyediakan sistem *routing*, *middleware*, dan manajemen *request-response* yang mudah digunakan, memungkinkan pengembangan *API* yang *scalable* dan *maintainable*.

6. ***Bitcoinjs-lib*, *BIP32*, *BIP39*, dan *Ecpair***

   Bitcoinjs-lib adalah *library JavaScript* untuk manipulasi *Bitcoin transactions*, pembuatan *wallet*, dan *signing*. *BIP32* digunakan untuk *hierarchical deterministic wallet generation*, *BIP39* untuk *mnemonic seed generation*, dan *Ecpair* untuk operasi kriptografi *elliptic curve* yang dibutuhkan dalam proses *signing PSBT* (*Partially Signed Bitcoin Transaction*). Kombinasi *library* ini memungkinkan implementasi *multisig wallet* yang aman dan sesuai dengan standar *Bitcoin*.

7. ***Prisma ORM***

   Prisma adalah *Object-Relational Mapping* modern yang digunakan untuk mengelola *database* dengan cara yang *type-safe* dan intuitif. *Prisma* menyediakan *schema definition*, *automatic migration*, dan *query builder* yang

powerful, mempermudah operasi *database* seperti *CRUD* tanpa perlu menulis *raw SQL queries*.

8. ***Tailwind CSS* dan *Shadcn/UI***
   Tailwind CSS adalah *utility-first CSS framework* yang memungkinkan pembuatan desain antarmuka yang responsif dan modern dengan cepat. *Shadcn/UI* adalah *component library* berbasis *Tailwind CSS* dan *Radix UI* yang menyediakan komponen-komponen siap pakai yang dapat dikustomisasi, sehingga mempercepat proses pengembangan *frontend* tanpa mengorbankan kualitas desain.

9. ***Pinia* dan *Pinia-persist***
   Pinia adalah *state management library* untuk *Vue.js* yang menggantikan *Vuex*, menyediakan sistem yang lebih sederhana dan *type-safe* untuk mengelola *state* aplikasi. *Pinia-persist plugin* digunakan untuk melakukan *persistence state* ke *localStorage*, sehingga data *state* tetap tersimpan bahkan setelah *page refresh*.

10. ***QRCode***
    QRCode adalah *library* untuk *generate QR code* di aplikasi *frontend*, digunakan untuk menampilkan alamat *Bitcoin vault* dalam format *QR code* yang dapat dipindai, memudahkan pengguna untuk berbagi alamat *wallet* tanpa perlu *copy-paste* manual.

11. ***Vue-i18n***
    Vue-i18n adalah *plugin* untuk *Vue.js* yang menangani *internationalization* (*i18n*), memungkinkan aplikasi mendukung *multiple* bahasa. Dalam proyek ini, *vue-i18n* digunakan untuk *support* bahasa Indonesia dan Inggris di antarmuka aplikasi.

12. ***Axios***
    Axios adalah *HTTP client library* yang digunakan untuk melakukan *request* ke *API backend* dari *frontend*. *Axios* menyediakan fitur seperti *interceptors*, *timeout handling*, dan *automatic JSON transformation* yang mempermudah komunikasi antara *frontend* dan *backend*.

13. ***JWT* (*JSON Web Token*)**
    JWT adalah standar untuk autentikasi *stateless* yang digunakan dalam *API*, memungkinkan pengguna untuk *login* sekali dan mendapatkan *token* yang

dapat digunakan untuk mengakses *resource* yang dilindungi tanpa perlu *login* berulang kali.

14. ***Bcryptjs***
    Bcryptjs adalah *library* untuk *hashing password* dengan algoritma *bcrypt*, memberikan keamanan tambahan pada sistem autentikasi dengan melakukan *hashing* iteratif yang membuat *password* sulit untuk di-*crack*.

3.6 Pengujian

Pengujian terhadap sistem dilakukan secara bertahap untuk memastikan bahwa setiap fitur dapat berjalan sesuai dengan fungsinya dan memenuhi standar keamanan yang tinggi. Pengujian dilakukan dalam dua tahap.

Pertama, pengujian dilakukan secara mandiri oleh penulis dengan mencoba seluruh alur sistem secara menyeluruh, termasuk validasi data, tampilan antarmuka, proses *backend*, integrasi *Bitcoin testnet*, pembuatan *multisig vault*, *signing* transaksi *PSBT*, dan *broadcasting* transaksi. Pengujian ini juga meliputi verifikasi fitur keamanan seperti *Zero-Database Security* dan sistem *role-based access control*.

Kedua, pengujian dilakukan oleh dua *supervisor* dari perusahaan tempat magang, yaitu Bapak Boby Hartanto dan Bapak Boby Harmoko, untuk mengevaluasi sistem dari sudut pandang pengguna dan memberikan masukan terkait fungsionalitas, kenyamanan penggunaan, serta *security posture*.

Masukan dari kedua *supervisor* digunakan untuk melakukan beberapa perbaikan dan optimisasi, seperti penyesuaian tampilan *interface* untuk kemudahan navigasi, penambahan validasi *form* untuk mencegah *invalid input*, perbaikan *error handling*, dan *enhancement* terhadap implementasi keamanan sistem untuk memastikan *zero-database security* berjalan dengan baik.

3.7 Kendala dan Solusi yang Ditemukan

3.7.1 Pemahaman Fundamental tentang Cryptocurrency dan Multisignature

Kendala: Pada awal magang, mahasiswa magang menghadapi kesulitan dalam memahami konsep-konsep fundamental *cryptocurrency* seperti *UTXO model*, *public-private key cryptography*, *HD wallet* (*Hierarchical Deterministic*), dan *multisignature transactions*. Pemahaman yang kurang terhadap konsep-konsep

ini menjadi hambatan utama dalam mengimplementasikan *Bitcoin Custodian API* yang memerlukan pengetahuan mendalam tentang mekanisme *Bitcoin*.

Solusi: Melakukan pembelajaran mandiri yang intensif melalui berbagai sumber, termasuk membaca artikel teknis tentang *Bitcoin*, menonton video *educational* dari *channel YouTube* seperti *Andreas M. Antonopoulos* dan *BitcoinTech*, serta mempelajari dokumentasi *bitcoinjs-lib* dan *BIP standard* (*BIP32*, *BIP39*). Selain itu, berkonsultasi langsung dengan *supervisor* untuk klarifikasi konsep-konsep yang sulit dipahami dan mendapatkan penjelasan praktis tentang implementasinya.

3.7.2 Implementasi Zero-Database Security

Kendala: Mengimplementasikan fitur *Zero-Database Security* merupakan tantangan tersendiri karena memerlukan pendekatan yang tidak konvensional. Awalnya, mahasiswa magang bingung tentang bagaimana cara menyimpan data transaksi tanpa menyimpan *amount* di *database*, dan bagaimana cara memverifikasi integritas data transaksi ketika data disimpan di *blockchain*.

Solusi: Merancang sistem yang menggunakan *cryptographic hashing* (*SHA-256*) untuk memverifikasi integritas data transaksi. Data finansial (*amount*) disimpan secara terenkripsi dalam *blockchain PSBT data*, bukan di *database*, sehingga mencegah risiko manipulasi data di *level database*. Implementasi ini melibatkan pemahaman mendalam tentang *PSBT structure*, ekstraksi *amount* dari *PSBT data*, dan verifikasi *hash* untuk memastikan integritas transaksi. Proses pembelajaran ini dilakukan melalui eksperimen langsung, membaca *PSBT specification*, dan konsultasi dengan *supervisor*.

3.7.3 Kompleksitas PSBT Signing dan Broadcasting

Kendala: Proses *Partially Signed Bitcoin Transaction* (*PSBT*) *signing* dan *broadcasting* menghadapi beberapa kendala teknis, seperti *error* pada saat *parsing PSBT data*, kesulitan dalam mengekstrak *amount* dari *PSBT structure*, dan masalah pada saat *broadcasting* transaksi yang sudah *signed*.

Solusi: Melakukan *debugging* secara sistematis dengan menambahkan *comprehensive logging* untuk melacak proses *signing* dan *broadcasting*. Mempelajari *PSBT format* secara detail dari *Bitcoin Development Kit documentation*, menggunakan *online PSBT decoder tools* untuk memahami

struktur *PSBT*, dan melakukan *testing* berulang kali dengan berbagai skenario transaksi. *Supervisor* juga memberikan *guidance* tentang *best practices* dalam *handling PSBT* dan *error prevention strategies*.

3.7.4 State Management dan Frontend Integration

Kendala: Mengelola *state* yang kompleks di *frontend*, terutama terkait data *vault*, transaksi, dan status *signing*, memerlukan sistem *state management* yang *robust* dan mudah dikelola.

Solusi: Menggunakan *Pinia Store* untuk *centralized state management* dengan memisahkan *concerns* ke dalam *modules* yang berbeda (*auth*, *vault*, *transaction*). Implementasi *action* dan *mutations* yang jelas memudahkan *tracking state changes* dan *debugging*. Penggunaan *composition API* dari *Vue* 3 juga memudahkan *code organization* dan *reusability*.