BAB II
TINJAUAN PUSTAKA

2.1 Rantai Pasok

Rantai pasok adalah seri yang terhubung kegiatan yang berkaitan dengan perencanaan, koordinasi dan pengendalian bahan dan barang jadi dari pemasok ke pelanggan. Ruang lingkup rantai pasokan dimulai dengan sumber pasokan dan berakhir pada titik konsumsi [10]. Sejumlah entitas bisnis yang berbeda yaitu pemasok, produsen, distributor dan pengecer bekerja sama dalam upaya untuk memperoleh bahan mentah, mengubah bahan mentah ini menjadi produk akhir yang ditentukan, dan mengirimkan produk akhir ini ke pengecer [11]. Pengoperasian rantai pasok yang efektif sangat penting bagi setiap perusahaan. Rantai ini secara tradisional ditandai dengan aliran material ke depan dan arus informasi yang mundur [11].

Biasanya, rantai pasokan terdiri dari dua proses bisnis utama yaitu manajemen material (logistik masuk) dan distibusi fisik (logistik keluar) [12]. Manajemen material mendukung siklus lengkap aliran bahan dari pembelian dan pengendalian internal bahan produksi untuk perencanaan dan pengendalian barang ke dalam proses, ke pergudangan, pengiriman, dan distribusi produk yang sudah jadi [13]. Sedangkan distribusi fisik mencakup penerimaan dan pemrosesan pesanan, penyebaran inventaris, penyimpanan dan penanganan, transportasi keluar, penetapan harga, dukungan promosi, penanganan produk yang dikembalikan serta dukungan siklus hidup [14]. Kegiatan manajemen material dan distribusi fisik digabungkan sehingga tercipta rantai pasok yang tidak hanya mewakili rantai linier dari hubungan bisnis satu-satu, tetapi jaringan dari beberapa hubungan bisnis [12].

2.2 Manajemen Rantai Pasok

Manajemen rantai pasok (MRP) merupakan pengelolaan terhadap aliran material dan aliran informasi serta modal yang mengikutinya dari awal sampai akhir mata rantai bisnis untuk mengoptimalkan pemenuhan kebutuhan setiap entitas di dalam rantai pasok. Kegiatan-kegiatan yang tercakup dalam rantai tidak dapat berdiri sendiri karena saling berkaitan satu dengan lainnya, seperti pengadaan material, pengubahan material menjadi barang setengah jadi atau barang jadi, distribusi serta penyimpanan apabila diperlukan [15]. Munculnya MRP dilatar belakangi oleh dua hal pokok sebagai berikut [16].

1. Adanya praktik manajemen logistik tradisional yang bersifat adversarial pada era modern ini sudah tidak relevan lagi, karena tidak dapat menciptakan keunggulan kompetitif.

   Lingkungan industri telah berkembang secara dinamis pada era global ini dan memberikan pemicu bagi banyak organisasi perusahaan untuk menggali potensi yang mereka miliki, serta mengidentifikasi faktor kunci sukses untuk unggul dalam persaingan dengan perusahaan lain. Banyak usaha yang dilakukan hanya untuk memberikan produk yang terbaik bagi konsumen. Produk yang ditawarkan perusahaan kepada konsumen dalam pengertian manajemen produksi dan operasi adalah kombinasi antara barang dan jasa.

   Menyajikan sebuah produk kepada konsumen merupakan sebuah tantangan sekaligus peluang bagi sistem produksi operasi yang harus dijalankan oleh perusahaan. Proses yang perlu dilewati adalah dimulai dari mengidentifikasi selera konsumen hingga harus mengupayakan seluruh kebutuhan untuk memproduksi dan mendistribusikan produk tersebut kepada konsumen. Pada dasarnya konsumen akan mengharapkan sebuah produk yang bermanfaat, harga mudah dijangkau dan sesuai dengan harga yang dikeluarkan. Dalam upaya mewujudkan keinginan konsumen, maka setiap perusahaan akan berusaha semaksimal mungkin untuk menggunakan aset dan kemampuan yang dimiliki untuk memberikan hasil yang sesuai harapan. Namun, upaya ini akan menimbulkan konsekuensi dalam segi biaya, sehingga perusahaan harus berusaha mereduksi seluruh biaya tanpa mengurangi kualitas dari produk yang sudah ditetapkan.

   Salah satu upaya untuk mereduksi biaya adalah melalui optimalisasi distribusi material dari pemasok, aliran material dalam proses produksi sampai dengan distribusi produk ke tangan konsumen. Distribusi yang optimal dapat dicapai melalui penerapan konsep Manajemen Rantai Pasok.

2. Perubahan lingkungan bisnis yang semakin cepat dengan persaingan yang semakin ketat.

   Lingkungan bisnis selalu berubah dan perubahan tersebut semakin lama semakin cepat. Perubahan ini disebabkan oleh beberapa faktor:

   a. Konsumen berubah menjadi terlalu menuntut. Mereka menuntut harga murah, kualitas bagus, penyerahan yang tepat waktu dan sesuai dengan selera mereka.
   b. Infrastruktur telekomunikasi, informasi, transportasi, dan perbankan yang semakin canggih memungkinkan berkembangnya model baru dalam aliran material/produk.
   c. Daur hidup produk sangat pendek seiring dengan perubahan yang terjadi dalam lingkungan pasar.

d. Kesadaran konsumen akan pentingnya aspek sosial dan lingkungan dalam kehidupan. Konsumen jadi menuntut industri manufaktur memasukkan konsep-konsep ramah lingkungan mulai dari proses perancangan produk hingga proses distribusinya.
e. Globalisasi dan perubahan peta ekonomi dunia telah membuat meningkatnya persaingan antara produk jasa di pasaran.

2.3 *Traceability* pada Manajemen Rantai Pasok

Traceability adalah kemampuan untuk melacak pergerakan tahap tertentu dari rantai pasokan, serta menelusuri sejarah atau lokasi dari rantai pasokan [17]. *Traceability* dapat dibagi menjadi kemampuan untuk melacak dan kemampuan untuk mengikuti jejak. Kemampuan untuk melacak atau sering disebut dengan *tracking* adalah sebuah prosedur mengungkap lokasi dari sebuah benda [18]. Sedangkan kemampuan untuk mengikuti jejak atau sering disebut dengan *tracing* adalah sebuah prosedur mencari asal-usul dari sebuah benda [19].

Masalah utama dalam meningkatkan *traceability* adalah keuntungan ekonomi dari *traceability* harus lebih besar daripada biaya sistem [20]. *Traceability* hanya akan menghasilkan keuntungan jika pengeluaran yang dikeluarkan lebih sedikit serta memberikan manfaat yang berguna bagi pelanggan. Selain daripada itu, ada beberapa masalah lain yang berhubungan dengan *traceability*, yaitu [18]:

1. Kurangnya standarisasi format dari entitas di dalam rantai pasok, terutama antara negara yang memiliki regulasi berbeda.
2. Kertas masih dianggap sebagai alternatif penyimpanan data yang paling murah dan paling terpercaya.
3. Kemampuan untuk melacak proses internal barang dalam jumlah banyak masih kurang.
4. Transparansi dan kerahasiaan dari informasi harus seimbang.
5. Tidak semua pihak memiliki *software* dan *hardware* yang dapat melacak secara internal.

Dapat ditarik sebuah kesimpulan bahwa *traceability* sulit ditingkatkan karena masalah teknologi dan pertukaran informasi terpercaya antara dua belah pihak dalam rantai pasok.

2.4 Agroindustri

Agroindustri berasal dari dua kata *agricultural* dan *industry* yang berarti suatu industri yang menggunakan hasil pertanian sebagai bahan baku utamanya atau suatu industri yang menghasilkan suatu produk yang digunakan sebagai sarana atau input dalam usaha pertanian.

Definisi agroindustri dapat dijabarkan sebagai sebuah kegiatan industri yang memanfaatkan hasil pertanian sebagai bahan baku, merancang dan menyediakan peralatan serta jasa untuk kegiatan tersebut. Maka agroindustri meliputi mulai dari industri pengolahan hasil pertanian, industri yang memproduksi peralatan dan mesin pertanian, industri input pertanian (pupuk, pestisida, herbisida, dan lain-lain) dan industri jasa sektor pertanian [21].

Karakteristik agroindustri yang menonjol sebenarnya adalah adanya ketergantungan antar elemen-elemen agroindustri, yaitu pengadaan bahan baku, pengolahan dan pemasaran produk. Agroindustri dapat dipandang sebagai suatu sistem yang terdiri dari empat keterkaitan sebagai berikut [22].

1. Keterkaitan mata rantai produksi yaitu keterkaitan antara tahapan-tahapan operasional mulai dari arus bahan baku pertanian sampai ke proses pembuatan lalu ke konsumen.
2. Keterkaitan kebijaksanaan makro-mikro yaitu keterkaitan berupa pengaruh kebijakan makro pemerintah terhadap kinerja agroindustri.
3. Keterkaitan kelembagaan yaitu hubungan antar berbagai jenis organisasi yang beroperasi dan berinteraksi dengan mata rantai produksi agroindustri.
4. Keterkaitan internasional yaitu saling ketergantungan antara pasar nasional dan pasar internasional dimana agroindustri berfungsi.

2.5 *Blockchain*

Blockchain pertama kali diperkenalkan pada Oktober 2008 sebagai bagian dari mata uang kripto *Bitcoin* yang diusulkan oleh Satoshi Nakamoto. Nama tersebut ternyata merupakan sebuah nama samaran dan hingga saat ini belum ada yang dapat mengungkapkan penemu aslinya [23] [24]. *Bitcoin* adalah aplikasi yang pertama kali mengimplementasikan teknologi *blockchain*, sebuah mata uang virtual yang menghindari sistem otoritas terpusat untuk pengeluaran mata uang, pemindahan kepemilikan, dan konfirmasi transaksi [23]. *Blockchain* memperoleh ketenaran sebagai teknologi yang mendasari *Bitcoin* dan sampai saat ini penggunaan *blockchain* meluas ke area aplikasi lain, salah satunya adalah rantai pasok [25].

Blockchain adalah solusi *database* terdistribusi penyimpanan data yang terus bertumbuh dan telah dikonfirmasi oleh *node* (komputer) yang berpartisipasi di dalamnya. Data-data dicatat di sebuah buku besar yang bersifat publik, termasuk informasi setiap transaksi yang telah diselesaikan. *Blockchain* adalah solusi terdesentralisasi yang mana tidak memerlukan organisasi pihak ketiga sebagai perantara. Informasi setiap transaksi yang telah

diselesaikan di *blockchain* bersifat terbuka dan tersedia untuk semua *node* yang berpartisipasi. Tujuan dari teknologi *blockchain* adalah untuk menciptakan lingkungan terdesentralisasi yang mana tidak memerlukan pihak ketiga yang mengontrol transaksi maupun data-data yang ada [25].

Adapun lima prinsip dasar teknologi *blockchain* yang diusulkan oleh Iansiti dan Lakhani [23] sebagai berikut.

1. *Distributed Database*

   Blockchain adalah *database* yang terdistribusi. Setiap *node* yang berpartisipasi di dalam sistem *blockchain* memiliki akses terhadap seluruh *database* dan riwayat lengkapnya, meskipun tidak ada satupun *node* yang mengontrol data/informasi yang disimpan di dalam *blockchain*. Setiap entri transaksi baru pada sistem *blockchain* diverifikasi oleh *node* yang berpartisipasi, tanpa memerlukan pihak ketiga sebagai penengah.

2. *Peer-to-Peer Transmission*

   Alih-alih menggunakan *platform* terpusat untuk komunikasi antar pihak, setiap *node* menyimpan dan meneruskan informasi secara langsung satu sama lain dalam jaringan *peer-to-peer*. *Blockchain* tersimpan di dalam setiap *node* yang berpartisipasi di dalam sistem *blockchain*, dan setiap *node* meneruskan informasi ke semua *node* lainnya.

3. *Transparency with Pseudonymity*

   Transparansi di dalam sistem teknologi *blockchain* tercapai dengan setiap *node* atau *user* memiliki akses terhadap setiap transaksi dalam sistem. Setiap *node* memiliki alamat unik (*unique address*) sebagai identitas mereka, dan transaksi terjadi antar alamat tersebut. *User* dapat memilih untuk menyamar sebagai anonim, dan tidak perlu mengungkapkan identitas aslinya.

4. *Irreversibility of Records*

   Sistem teknologi *blockchain* menggunakan algoritma komputasi untuk memastikan bahwa data yang disimpan di dalam *blockchain* bersifat permanen. Istilah "*chain*/rantai" berasal dari semua entri transaksi baru akan terhubung ke setiap transaksi sebelumnya dan terurut berdasarkan kejadian kronologis sehingga membentuk sebuah rantaian. Seluruh rantai blok menyimpan semua transaksi yang pernah tercatat di dalam *blockchain*.

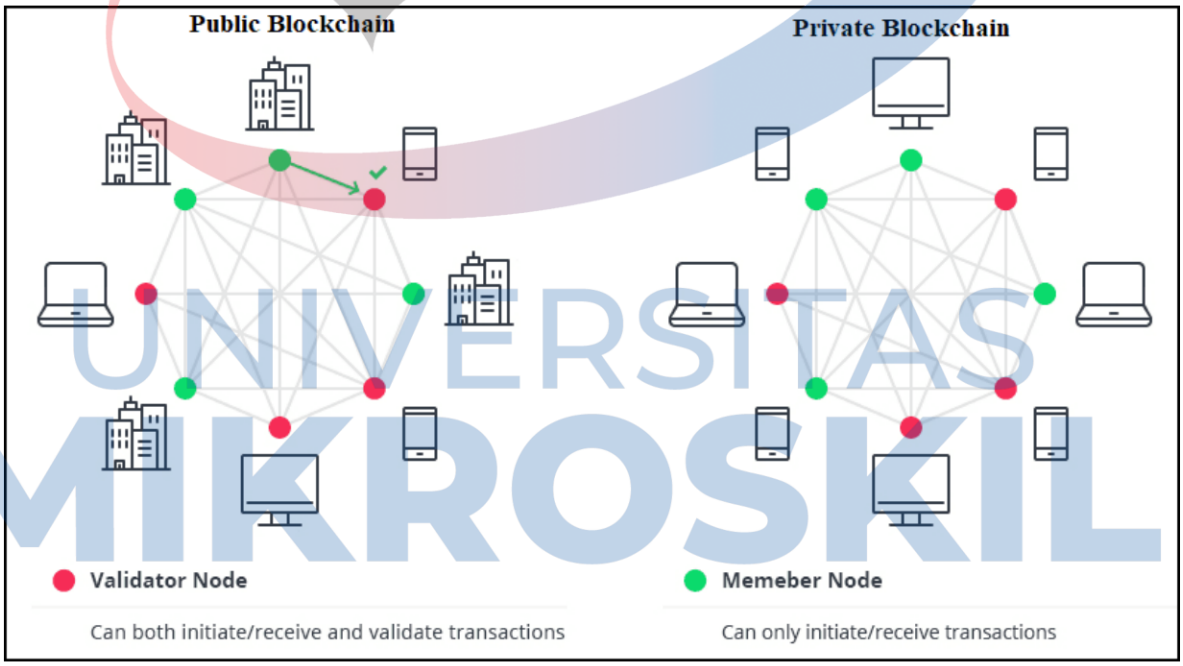
5. *Computational Logic*

   Sistem teknologi *blockchain* sebagai sifat digital, logika komputasi dapat terikat terhadap transaksi-transaksi di dalam *blockchain*. Setiap *node* dapat menggunakan algoritma atau aturan untuk memicu transaksi secara otomatis. Sistem teknologi *blockchain* dapat

diprogram untuk menangani semua jenis informasi, seperti transaksi, kontrak, atau jenis informasi apapun yang berhubungan dengan masalah bisnis tertentu.

Blockchain adalah salah satu solusi yang dapat meningkatkan performa Manajemen Rantai Pasok dalam segi transparansi. Teknologi ini disebut dapat menyelesaikan masalah transparansi karena dapat bertindak sebagai buku besar yang didistribusikan dan mampu menyediakan transparansi kepada seluruh pihak yang terlibat dalam proses transaksi, namun privasi terjamin, tidak dapat diubah dan aman. *Blockchain* menyediakan keamanan seperti perlindungan dan pencegahan terhadap duplikasi atau distorsi data dari luar [26].

Dalam *public blockchain*, setiap entitas dalam jaringan memiliki hak yang sama untuk membaca dan menulis data di *blockchain* dengan mengikuti aturan umum yang mereka setujui. Siapapun dapat memvalidasi transaksi dengan perangkat lunak dan perangkat keras yang diperlukan. Sedangkan dalam *permissioned blockchain*, data tidak bersifat publik. Setiap entitas hanya dapat menulis, membaca serta mengakses data jika sudah memiliki hak akses [27].



Gambar 2. 1 (a) *Public Blockchain* (b) *Private Blockchain* [28]

2.6 Komponen *Blockchain*

Melalui sudut pandang pada tingkat tinggi, teknologi *blockchain* memanfaatkan mekanisme ilmu komputer dan kriptografi yang umum dan terkemuka digabung dengan konsep pencatatan data di dalam buku besar. Pada subbab ini akan membahas setiap

komponen utama di dalam *blockchain*, yakni fungsi *hash* kriptografi, transaksi, *asymmetric-key cryptography*, alamat, *ledger*, blok dan rantaian blok.

2.6.1 Fungsi *Hash* Kriptografi

Hashing adalah metode pengaplikasian sebuah fungsi *hash* kriptografi kepada suatu data, yang mana menghasilkan keluaran yang relatif unik, biasa disebut sebagai pesan intisari/*message digest* atau hanya *digest* dari masukan dalam ukuran apapun misalnya *file*, teks, atau gambar. Setiap data masukan yang di-*hash* menggunakan fungsi *hash* kriptografi yang sama akan selalu menghasilkan keluaran yang sama, sehingga fungsi ini dapat digunakan untuk membuktikan bahwa data tidak pernah diubah [29].

Fungsi *hash* kriptografi memiliki beberapa properti keamanan yang penting [29], yaitu:

1. Bersifat *preimage resistant*. Artinya nilai masukan tidak mungkin dapat diperoleh dengan cara apapun dari hasil keluaran *hash* (contoh: cari x dimana $hash(x) = digest$).
2. Bersifat *second preimage resistant*. Artinya tidak mungkin mendapatkan nilai masukan lain yang jika di-*hash* menggunakan fungsi *hash* kriptografi yang sama akan menghasilkan keluaran yang sama (contoh: diberi x, cari y dimana $hash(x) = hash(y)$).
3. Bersifat *collision resistant*. Artinya tidak mungkin menemukan dua nilai masukan yang jika di-*hash* menggunakan fungsi *hash* kriptografi yang sama akan menghasilkan keluaran yang sama (contoh: cari x dan y dimana $hash(x) = hash(y)$).

Secure Hash Algorithm (SHA) merupakan fungsi *hash* kriptografi yang banyak diimplementasikan pada *blockchain*, khususnya SHA-256. SHA-256 memiliki hasil keluaran dengan ukuran 256 bit dan biasanya ditampilkan sebagai 64 karakter heksadesimal (Lihat tabel 2.1). Hasil keluaran berukuran 256 bit berarti terdapat $2^{256} \approx 10^{77}$, atau 115.792.089.237.316.195.423.570.985.008.687.907.853.269.984.665.640.564.039.457.584.007.913.129.639.936 kemungkinan yang dapat dihasilkan. Algoritma untuk SHA-256 telah terperinci pada *Federal Information Processing Standard* (FIPS) 180-4 [30]. Pada *website Secure Hashing National Institute of Standards and Technology* (NIST) mengandung rincian FIPS untuk semua algoritma *hashing* yang telah disetujui oleh NIST [31] [29].

Tabel 2. 1 Contoh Teks Masukan dan Hasil Keluaran SHA-256 [29]

Teks Masukan	**Hasil Keluaran SHA-256**
1	0x6b86b273ff34fce19d6b804eff5a3f5747ada4eaa22f1d49c01e52ddb7875b4b

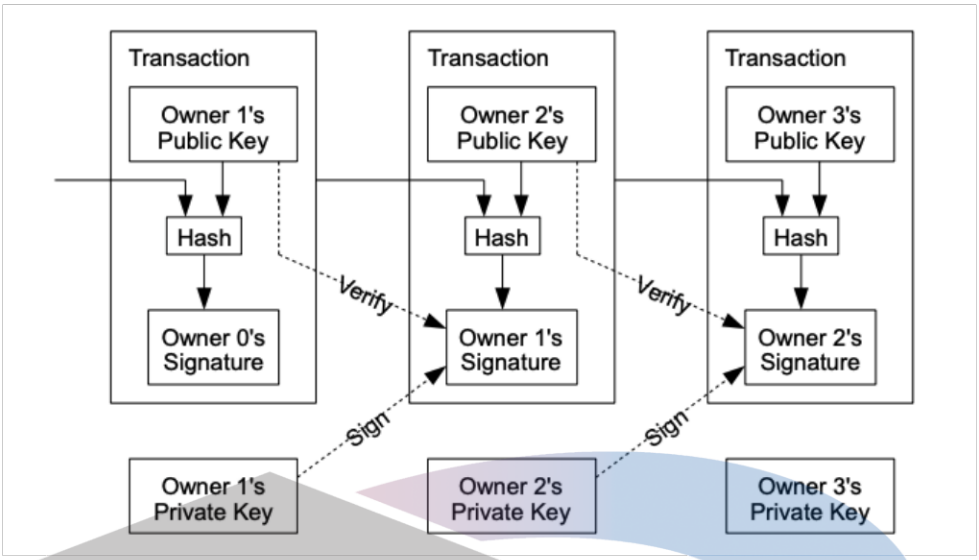
2	0xd4735e3a265e16eee03f59718b9b5d03019c07d8b6c51f90da3a666eec13ab35
Hello, World!	0xdffd6021bb2bd5b0af676290809ec3a53191dd81c7f70a4b28688a362182986f

Namun, karena terdapat jumlah yang tak terbatas dari kemungkinan nilai masukan dan hasil keluaran yang terbatas, maka mungkin untuk terjadi dua nilai masukan yang berbeda menghasilkan keluaran yang sama. Meski demikian, SHA-256 dikatakan *collision resistant* karena untuk mendapatkan dua masukan tersebut membutuhkan secara rata-rata 2^{128} kali perhitungan ($3{,}402 \times 10^{38}$, atau 340.282.366.920.938.463.463.374.607.431.768.211.456). Untuk menggambarkan berapa lama waktu yang dibutuhkan kemungkinan terjadinya *collision*, kecepatan *hash* per detik dari seluruh jaringan *Bitcoin* pada tahun 2015 adalah 300 kuadriliun *hash* per detik ($300.000.000.000.000/s$). Pada kecepatan itu, seluruh jaringan *Bitcoin* membutuhkan kira-kira $35.942.991.748.521 (3{,}6 \times 10^{13})$ tahun [29].

Adapun beberapa jenis fungsi *hash* kriptografi lainnya yang diimplementasi pada teknologi *blockchain*, seperti Keccak (terpilih oleh NIST sebagai pemenang kompetisi untuk membuat standar *hashing* SHA-3) dan RIPEMD-160 [32] [29].

2.6.2 Transaksi

Pada mata uang kripto *Bitcoin*, *Bitcoin* mendefinisikan sebuah transaksi koin elektronik sebagai rantaian tanda tangan digital (*digital signature*). *Digital signature* adalah sebuah mekanisme autentikasi yang memungkinkan pengirim dari sebuah pesan untuk melampirkan sebuah kode unik yang berperan sebagai tanda tangan yang biasanya terbentuk dari *hash* pesan tersebut dan mengenkripsi pesan tersebut dengan *private key* pengirim [33]. Setiap pemilik melakukan pengiriman koin kepada penerima dengan menandatangani secara digital *hash* transaksi sebelumnya dan *public key* penerima lalu menambahkannya pada akhir koin yang dikirim. Penerima koin dapat melakukan verifikasi tanda tangan digital tersebut untuk memeriksa rantaian kepemilikan koin tersebut [34].



Gambar 2. 2 Ilustrasi Transaksi pada *Bitcoin* [34]

Pada umumnya, transaksi menggambarkan interaksi antar pihak yang terlibat. Pada contoh mata uang kripto, transaksi merupakan pengiriman mata uang kripto antar *user* jaringan *blockchain*. Pada skenario bisnis ke bisnis (*business-to-business*), transaksi bisa saja merupakan pencatatan aktivitas yang terjadi pada aset digital maupun fisik. Sebagai contoh, sebuah transaksi dapat digunakan sebagai perubahan atribut aset digital seperti lokasi pengiriman pada sistem rantai pasok berbasis teknologi *blockchain*. Data yang ada di dalam transaksi bisa saja berbeda-beda pada setiap implementasi *blockchain*, namun mekanisme bertransaksi pada dasarnya adalah sama. Pada jaringan *blockchain, user* mengirim informasi ke jaringan *blockchain*, yang mana informasi tersebut mungkin termasuk alamat pengirim atau identitas lain yang relevan, *public key* pengirim, *digital signature*, masukan transaksi dan keluaran transaksi [29].

Terlepas dari bagaimana data dibentuk dan ditransaksikan, menentukan keabsahan dan keaslian suatu transaksi itu penting. Keabsahan dari sebuah transaksi memastikan bahwa transaksi tersebut memenuhi persyaratan protokol dan format data atau persyaratan *smart contract* yang spesifik pada tiap implementasi *blockchain*. Keaslian dari suatu transaksi juga penting, karena keaslian transaksi menentukan bahwa pengirim aset digital memiliki akses pada digital tersebut. Transaksi biasanya ditandatangani secara digital menggunakan *private key user* dan dapat diverifikasi kapan saja menggunakan *public key* terkait.

2.6.3 *Asymmetric-Key Cryptography*

Teknologi *blockchain* menggunakan *asymmetric-key cryptography* (disebut juga *public key cryptography*). *Asymmetric-key cryptography* menggunakan sepasang kunci: kunci

publik (*public key*) dan kunci *private* (*private key*) yang secara matematis berhubungan satu sama lain. *Public key* telah dirancang sedemikian rupa sehingga ketika dipublikasikan tidak akan mengurangi keamanannya, namun *private key* harus tetap dirahasiakan jika data ingin tetap dipertahankan keamanan kriptografinya. Meskipun kedua kunci memiliki hubungan, *private key* tidak dapat secara efisien didapatkan hanya dengan *public key* yang dimiliki. *Private key* digunakan untuk enkripsi data dan *public key* digunakan untuk dekripsi data. Atau alternatif lain, data dapat dienkripsi menggunakan *public key* dan didekripsi menggunakan *private key* [29].

Asymmetric-key cryptography memungkinkan *user* yang tidak mengenal atau mempercayai satu sama lain memiliki suatu hubungan yang dapat dipercaya, dengan menyediakan mekanisme untuk memverifikasi integritas dan keaslian dari sebuah transaksi sementara pada waktu yang sama memungkinkan transaksi bersifat publik. Untuk merealisasikan ini, transaksi ditandatangani secara digital yang berarti sebuah *private key* digunakan untuk mengenkripsi sebuah transaksi sehingga siapa pun yang memiliki *public key*-nya dapat mendekripsinya. Karena *public key* tersedia secara publik, pengenkripsian transaksi menggunakan *private key*-nya membuktikan bahwa penandatangan transaksi tersebut memiliki akses terhadap *private key*-nya. Atau alternatif lain, data dienkripsi oleh *public key user* dan hanya *user* yang memiliki akses terhadap *private key*-nya yang dapat mendekripsi transaksi tersebut [29].

2.6.4 Alamat dan Asal Alamat

Beberapa jaringan *blockchain* memanfaatkan alamat (*address*) sebagai identitas *user*, sebuah teks yang terdiri atas kumpulan karakter alfanumerik yang dibuat dari *public key user* menggunakan fungsi *hash* kriptografi, bersama dengan beberapa data tambahan seperti nomor versi atau *checksum*. Panjang alamat lebih pendek daripada *public key* dan tidak bersifat rahasia. Umumnya, implementasi *blockchain* memanfaatkan alamat untuk menandakan titik asal dan tujuan dalam sebuah transaksi. Salah satu metode untuk membuat sebuah alamat adalah dengan membuat *public key*, menerapkan fungsi *hash* kriptografi pada *public key* tersebut, dan konversi keluaran hasil *hash* menjadi bentuk teks [29].

Setiap implementasi *blockchain* mungkin saja menggunakan metode yang berbeda untuk memperoleh sebuah alamat. Pada jaringan *permissionless blockchain*, yang mana memperbolehkan pembuatan akun anonim, *user* dapat membuat sepasang *asymmetric-key* sebanyak-banyaknya, dan alamat sebanyak yang diinginkan, sehingga memungkinkan *user*

mencapai titik anonimitas yang diinginkan. Alamat dapat berperan sebagai identitas publik di dalam jaringan *blockchain* untuk *user*, dan seringkali alamat dikonversi ke dalam bentuk kode QR untuk mempermudah pengguna *smartphone* [29].

Gambar 2.3 Contoh Kode QR yang Telah Mengkonversi Teks "NISTIR 8202 – *Blockchain Technology Overview* QR *Code Example*" [29]

Pada beberapa jaringan *blockchain*, terutama pada jaringan *permissionless blockchain*, *user* wajib mengurus dan menyimpan *private key* mereka masing-masing. Namun alih-alih menyimpannya secara manual, *user* seringkali menggunakan *software* untuk menyimpannya secara aman. *Software* ini biasanya disebut sebagai *wallet*. *Wallet* digunakan untuk menyimpan *private key*, *public key*, dan *address* terkait. *Wallet* juga memiliki fungsi lainnya seperti kalkulasi jumlah aset digital yang dimiliki [29].

Jika *user* kehilangan *private key*-nya, maka semua aset digital yang terkait dengan *key* tersebut akan hilang juga, dan *user* tidak mungkin dapat menghasilkan *private key* yang sama. Jika sebuah *private key* dicuri, pelaku akan memiliki akses penuh pada semua aset digital yang dikontrol oleh *private key* tersebut [29].

2.6.5 Blok

User jaringan *blockchain* mengajukan transaksi ke jaringan *blockchain* melalui *software* seperti aplikasi *desktop*, aplikasi *smartphone*, *digital wallet* atau *web service*. *Software* kemudian mengirimkan transaksi tersebut ke *node* yang ada di dalam jaringan *blockchain*. *Node* yang dipilih mungkin saja *non-publishing full node*, yaitu sebuah *node* yang menyimpan seluruh data *blockchain*, meneruskan data ke *node* lain, dan memastikan blok baru yang ditambahkan valid dan autentik [29], maupun *publishing node*, yaitu sebuah *node* yang berfungsi untuk membuat dan menambahkan blok baru ke dalam *blockchain*, biasa juga disebut sebagai *mining node*, *committing node*, atau *minting node* [29]. Transaksi yang diajukan kemudian disebarluaskan ke *node-node* lain, namun pada proses ini transaksi tersebut belum ditambahkan ke dalam *blockchain*. Umumnya, pada implementasi *blockchain*,

setelah transaksi *pending* telah didistribusikan ke *node-node*, transaksi akan masuk ke dalam antrian untuk diproses dan ditambahkan ke dalam *blockchain* oleh *publishing node* [29].

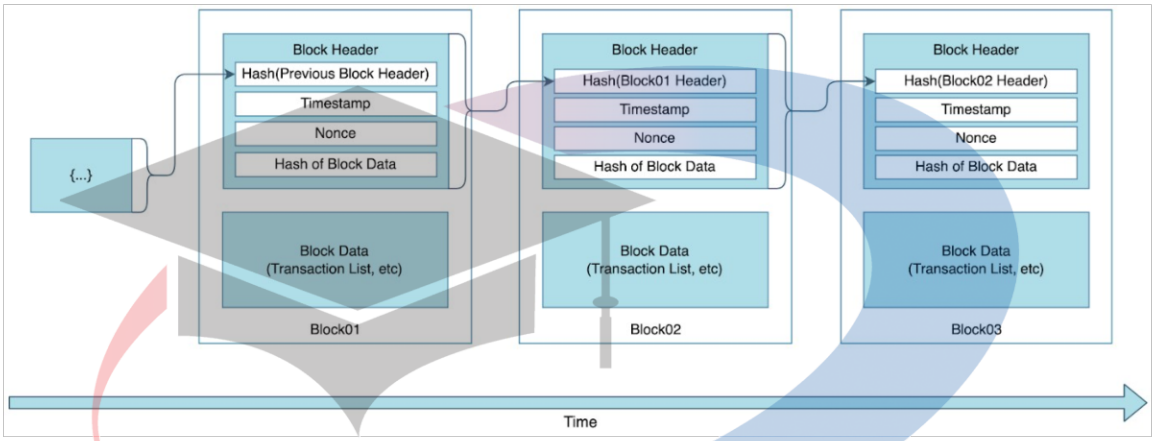
Transaksi-transaksi kemudian ditambahkan ke dalam *blockchain* ketika sebuah *publishing node* menerbitkan sebuah blok. Struktur di dalam sebuah blok berisi *block header* dan *block data*. *Block header* mengandung *metadata* yang mendeskripsikan data yang ada di dalam blok tersebut. *Block data* mengandung kumpulan transaksi yang telah divalidasi dan autentik yang telah diajukan ke jaringan *blockchain*. Keabsahan dan keaslian dari sebuah transaksi dipastikan melalui pengecekan bahwa transaksi tersebut memiliki format data yang benar dan transaksi tersebut telah ditandatangani secara digital oleh pemilik aset digital. Proses pengecekan ini memverifikasi bahwa pemiliki aset digital pada transaksi tersebut memiliki akses pada *private key* yang mampu menandatangani aset digital tersebut. *Full node* lainnya yang kemudian melakukan pengecekan keabsahan dan keaslian semua transaksi yang ada di dalam blok dan tidak akan menerima blok tersebut jika memiliki transaksi yang tidak valid [29].

Perlu dicatat bahwa pada setiap implementasi *blockchain* dapat memiliki struktur data yang berbeda-beda, namun pada kebanyakan implementasi *blockchain* menggunakan struktur data seperti sebagai berikut [29]:

1. *Block Header*
    a. *Block number* (nomor blok), juga disebut sebagai *block height* pada beberapa jaringan *blockchain*, merupakan urutan blok yang ada di dalam jaringan *blockchain*
    b. Nilai *hash block header* sebelumnya
    c. Nilai *hash block data*
    d. *Timestamp*
    e. *Block size* (ukuran blok)
    f. *Nonce*, hanya digunakan pada beberapa jaringan *blockchain* yang memanfaatkan metode *mining*, angka ini digunakan oleh *publishing node* untuk menyelesaikan *hash puzzle*
2. *Block Data*
    a. Kumpulan transaksi
    b. Data lainnya yang bersifat opsional

2.6.6 Rantaian Blok

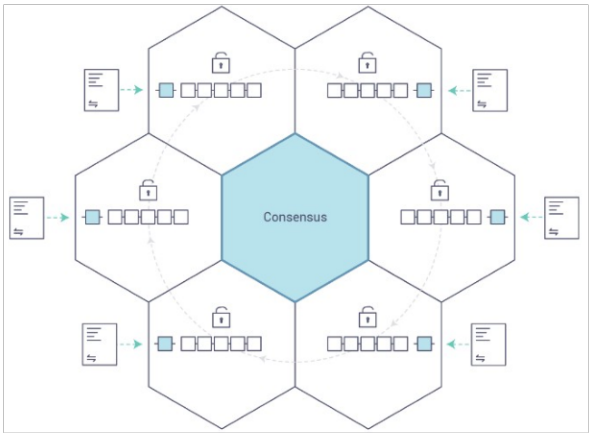
Blok terhubung satu sama lain dengan cara setiap blok menyimpan nilai *hash* dari *block header* sebelumnya, sehingga membentuk rantaian blok. Jika blok sebelumnya diubah, maka nilai *hash* blok tersebut akan berubah. Yang mana pada akhirnya akan menyebabkan nilai *hash* blok berikutnya juga berubah sebab setiap blok menyimpan nilai *hash* dari blok sebelumnya. Ini memungkinkan untuk dengan mudah mendeteksi dan menolak blok yang diubah [29].



Gambar 2.4 Struktur Rantaian Blok pada Umumnya [29]

2.7 *Consensus*

Pada jaringan *blockchain*, cara untuk mencapai persetujuan umum (*consensus*) di antara *node* yang tidak dapat dipercaya adalah transformasi dari masalah *Byzantine Generals Problem*. Pada *Byzantine Generals Problem* terdapat sekumpulan jenderal yang memimpin sebagian tentara *Byzantine* mengelilingi sebuah kota. Beberapa jenderal memilih untuk menyerang sedangkan beberapa jenderal lainnya memilih untuk mundur. Namun, serangan akan gagal jika hanya sebagian dari jenderal yang menyerang kota tersebut. Pada *blockchain*, tidak ada sebuah *node* terpusat yang dapat memastikan *ledger* pada *node* terdistribusi semunya sama [35]. Untuk membuat sistem ini bekerja, maka teknologi *blockchain* memanfaatkan model *consensus* untuk memungkinkan sekelompok *user* yang tidak mempercayai satu sama lain untuk bekerja sama.



Gambar 2. 5 Proses Consensus [36]

Setiap jaringan *blockchain* pasti memiliki sebuah *genesis block* dan setiap blok wajib ditambahkan setelah *genesis block* berdasarkan model *consensus* yang disepakati, sehingga *genesis block* menjadi blok pertama di dalam jaringan *blockchain*. *Genesis block* merupakan blok yang mencatat keadaan awal sistem yang telah dikonfigurasi sebelumnya. Ikut sertanya *user* ke dalam jaringan *blockchain* menandakan bahwa mereka setuju terhadap keadaan awal sistem. Terlepas dari model *consensus*, setiap blok wajib bersifat valid dan dapat divalidasi secara independen oleh setiap *user* jaringan *blockchain*. Dengan mengkombinasikan keadaan awal dan kemampuan untuk memverifikasi setiap blok, *user* dapat secara independen menyetujui keadaan *blockchain* saat ini. Namun jika terdapat dua rantaian yang valid pada sebuah *full node*, mekanisme bawaan jaringan *blockchain* pada umumnya akan menganggap rantaian yang lebih panjang yang benar dan akan digunakan, karena rantaian yang lebih panjang telah jumlah pekerjaan paling banyak untuk menghasilkan rantaian tersebut [29].

Properti pada setiap model *consensus* adalah sebagai berikut [29].

1. Keadaan awal sistem telah disepakati (*genesis block*)
2. *User* menggunakan model *consensus* untuk menyetujui blok mana yang akan ditambahkan ke dalam jaringan *blockchain*
3. Setiap blok terhubung ke blok sebelumnya dengan menyimpan *hash block header* sebelumnya, kecuali *genesis block* karena merupakan blok pertama di dalam jaringan *blockchain*
4. *User* dapat memverifikasi setiap blok secara independen

Berikut merupakan beberapa model *consensus* yang umum digunakan:

2.7.1 *Proof of Work*

Pada model *Proof of Work* (PoW), *node* yang dapat mempublikasikan blok selanjutnya adalah yang pertama kali menyelesaikan *puzzle* yang membutuhkan komputasi yang intensif. *Puzzle* ini didesain agar untuk menyelesaikan *puzzle* ini sulit namun untuk pengecekan solusinya bersifat valid mudah. Tujuannya adalah agar semua *full node* lainnya dapat dengan mudah melakukan pengecekan blok yang diajukan, dan setiap blok yang diusulkan tidak memenuhi syarat dari *puzzle* akan ditolak [29].

Metode *puzzle* yang umum adalah mengharuskan nilai *hash* dari *block header* kurang dari nilai yang ditentukan. Untuk menemukan nilai *hash* yang memenuhi persyaratan *puzzle*, *publishing node* akan membuat perubahan kecil pada *block header*, contohnya mengubah nilai *nonce*. Untuk setiap percobaan, *publishing node* diwajibkan menghitung untuk mendapatkan nilai *hash* keseluruhan *block header*. Percobaan melakukan *hash block header* berkali-kali menjadi proses komputasi yang intensif. Nilai persyaratan *puzzle* ini dapat dimodifikasi dari waktu ke waktu untuk menyesuaikan tingkat kesulitan untuk mengatur seberapa sering blok ditambahkan [29].

Sebagai contoh, terdapat sebuah *puzzle* dimana dengan menggunakan algoritma SHA-256, komputer harus menemukan nilai *hash* yang memenuhi kriteria target berikut:

```
SHA256("blockchain" + Nonce) = Hash Digest starting with "000000"
```

Pada contoh ini, teks "*blockchain*" ditambahkan dengan nilai *nonce* kemudian hitung nilai *hash* dari teks tersebut. Nilai *nonce* yang digunakan hanya nilai numerik saja. Beberapa contoh keluaran sebagai berikut:

```
SHA256("blockchain0") =
0xbd4824d8ee63fc82392a6441444166d22ed84eaa6dab11d4923075975acab938 (not solved)
SHA256("blockchain1") =
0xdb0b9c1cb5e9c680dfff7482f1a8efad0e786f41b6b89a758fb26d9e223e0a10 (not solved)
...
SHA256("blockchain10730895") =
0x000000ca1415e0bec568f6f605fcc83d18cac7a4e6c219a957c10c6879d67587 (solved)
```

Untuk menyelesaikan *puzzle* ini, membutuhkan 10.730.896 percobaan. Ini selesai dalam 54 detik pada perangkat keras lama, dimulai dari 0 dan mencoba satu nilai pada satu waktu.

Pada contoh ini, setiap penambahan nilai "*leading zero*" (awalan digit nol) akan meningkatkan tingkat kesulitannya. Dengan menambahkan satu digit nol menjadi "**0000000**" dan dengan menggunakan perangkat keras yang sama akan membutuhkan 934.224.175 percobaan untuk menyelesaikan *puzzle* ini. Ini selesai dalam 1 jam, 18 menit, 12 detik:

```
SHA256("blockchain934224174")                                    =
0x0000000e2ae7e4240df80692b7e586ea7a977eacbd031819d0e603257edb
3a81
```

Pada jaringan *blockchain* yang menggunakan metode *Proof of Work*, *publishing node* biasanya akan bekerja sama ke dalam kelompok ("*pools*" atau "*collectives*") untuk menyelesaikan *puzzle* dan membagikan hadiahnya. Ini bisa terjadi karena pekerjaan dapat didistribusikan di antara dua atau lebih *node* di dalam kelompok dengan berbagi beban kerja dan hadiah. Dengan contoh membagi pekerjaan menjadi empat bagian dan setiap *node* akan mengambil jumlah nilai *nonce* yang sama untuk dicoba:

```
Node 1: check nonce 0000000000 to 0536870911
Node 2: check nonce 0536870912 to 1073741823
Node 3: check nonce 1073741824 to 1610612735
Node 4: check nonce 1610612736 to 2147483647
```

Hasil berikut adalah solusi yang pertama kali ditemukan untuk memecahkan *puzzle*:

```
SHA256("blockchain1700876653")                                   =
0x000000073ba55d20c9cbd1b6fb34dd81c3553360ed918d07acf16dc9e75d7
c7f1
```

Ini adalah nilai *nonce* yang berbeda dengan yang sebelumnya, tetapi memenuhi kriteria target untuk memecahkan *puzzle*. Metode ini membutuhkan 90.263.918 percobaan. Ini selesai dalam 10 menit, 14 detik. Dengan membagi pekerjaan kepada lebih banyak mesin akan menghasilkan hasil yang jauh lebih baik serta hadiah yang lebih konsisten dalam model *Proof of Work*.

2.7.2 *Proof of Stake*

Motivasi di belakang model *Proof of Stake* (PoS) adalah semakin banyak saham (*stake*) *user* yang diinvestasikan ke dalam sistem, semakin besar mereka ingin sistem itu

berhasil. *Stake* sering kali merupakan jumlah nilai *cryptocurrency* yang diinvestasikan oleh *user* pada jaringan *blockchain*. Ini dapat dilakukan melalui berbagai cara, seperti dengan menguncinya melalui tipe transaksi khusus, atau dengan mengirimkannya ke alamat tertentu, atau menahannya di dalam *wallet* khusus. Sekali diinvestasikan, *cryptocurrency* tersebut pada umumnya tidak dapat dibelanjakan lagi. Jaringan *blockchain* yang menggunakan model *Proof of Stake* menggunakan jumlah saham *user* sebagai faktor penentu *user* mana yang akan menerbitkan blok baru. Dengan demikian, kemungkinan *user* yang terpilih menjadi penerbit blok baru selanjutnya berkaitan dengan rasio saham mereka terhadap jumlah keseluruhan jaringan *blockchain* dari *cryptocurrency* yang diinvestasikan [29].

Sehingga dengan model *consensus* ini, tidak diperlukan untuk melakukan komputasi yang memakan sumber daya secara intensif yang melibatkan waktu, listrik, dan kekuatan pemrosesan seperti *Proof of Work*. Karena model *consensus* ini tidak menggunakan banyak sumber daya, pada beberapa jaringan *blockchain* memutuskan untuk tidak memberikan hadiah pada *node* penerbit blok baru, sistem ini dirancang sehingga semua *cryptocurrency* sudah didistribusikan di antara *user* daripada menghasilkan *cryptocurrency* baru dengan kecepatan konstan. Pada sistem seperti ini, hadiah untuk menerbitkan blok baru biasanya berupa biaya transaksi *user* [29].

Metode pada jaringan *blockchain* yang memanfaatkan model *Proof of Stake* dapat beragam. Pada bagian ini akan dibahas empat pendekatan yaitu: *random selection of staked users*, *multi-round voting*, *coin aging systems* and *delegate systems*. Namun terlepas dari empat pendekatan tersebut, *user* dengan *stake* yang lebih banyak yang lebih mungkin menjadi *node* yang menerbitkan blok baru [29].

1. *Random choice*

   Pada metode ini (disebut juga sebagai *chain-based proof of stake*), jaringan *blockchain* akan mencatat semua *user* yang memiliki saham pada sistem ini dan memilih salah satu di antara mereka berdasarkan rasio saham mereka terhadap jumlah keseluruhan *cryptocurrency* yang diinvestasikan. Sehingga, jika *user* memiliki 42% dari seluruh jumlah saham jaringan *blockchain* maka kemungkinan *user* tersebut akan terpilih sebagai penerbit blok baru adalah 42% dan *user* yang memiliki 1% akan memiliki kemungkinan terpilih sebesar 1%.

2. *Multi-round voting system*

   Pemilihan *publisher* pada metode ini (disebut sebagai *Byzantine fault tolerance proof of stake*), jaringan *blockchain* akan memilih beberapa *user* yang memiliki saham untuk

membuat blok yang diajukan. Kemudian semua *user* yang memiliki saham akan memberikan suara untuk memilih blok yang diajukan. Pemilihan suara mungkin saja terjadi beberapa kali sebelum blok baru diputuskan. Metode ini membuka peluang bagi semua *user* yang memiliki saham menjadi memiliki suara dalam proses pemilihan blok pada setiap blok baru yang diajukan.

3. *Coin age system*

   Pada metode ini (disebut juga sebagai *coin age proof of stake*), *cryptocurrency* yang diinvestasikan memiliki umur (*age*). Setelah jangka waktu tertentu, seperti 30 hari *cryptocurrency* yang diinvestasikan akan terhitung menjadi calon kemungkinan terpilih untuk menerbitkan blok baru. Namun ketika koin *cryptocurrency* terpilih, umur *cryptocurrency* tersebut akan di-*reset* dan tidak dapat digunakan lagi sampai waktu yang telah ditentukan berlalu. Metode ini dirancang agar *user* yang memiliki lebih banyak saham akan menerbitkan lebih banyak blok, namun tidak mendominasi sistem karena terdapat *timer cooldown*. Kemungkinan untuk terpilih sebagai penerbit blok selanjutnya akan semakin meningkat seiring bertambahnya usia koin dan banyaknya koin yang terkumpul. Namun untuk mencegah *user* menimbun koin *cryptocurrency*, umumnya terdapat nilai maksimal probabilitas *user* tersebut terpilih sebagai penerbit blok selanjutnya.

4. *Delegate system*

   Pada metode ini, *user* dapat melakukan *voting* pemilihan *node* untuk menjadi *publishing node*. Hak suara *user* dalam jaringan *blockchain* ini terikat pada jumlah saham mereka, semakin banyak jumlah saham maka semakin besar hak suara mereka. *Node* yang mendapatkan suara terbanyak akan menjadi *publishing node* dan dapat memvalidasi serta menerbitkan blok. *User* juga dapat melakukan *voting* untuk menurunkan jabatan *publishing node*. *Voting* untuk pemilihan *publishing node* ini dilakukan secara terus menerus dan bersifat kompetitif. Karena pemilihan *publishing node* ini bersifat kompetitif dan setiap *publishing node* terdapat ancaman atas kehilangan status *publishing node*-nya, maka hadiah dan reputasi diberikan secara konsisten agar *publishing node* tidak melakukan tindak kejahatan apapun.

2.7.3 *Round Robin*

Round robin adalah model *consensus* yang biasanya dipakai pada beberapa jaringan *permissioned blockchain*. Pada model *consensus* ini, setiap *node* bergiliran dalam pembuatan

blok baru. Untuk mengatasi situasi dimana ketika sebuah *publishing node* tidak tersedia untuk menerbitkan blok pada gilirannya, sistem dapat menyertakan batas waktu untuk mengaktifkan *node* lain yang tersedia untuk menerbitkan blok tersebut sehingga *node* yang tidak tersedia tidak akan menyebabkan hambatan dalam penerbitan blok baru. Model *consensus* ini menjamin tidak ada *node* yang membuat sebagian besar blok di dalam jaringan *blockchain*. Manfaat dari model *consensus* ini adalah pendekatan yang mudah untuk diimplementasikan, tidak memiliki *puzzle* kriptografi, dan memiliki kebutuhan daya yang rendah. Namun karena model *consensus* ini membutuhkan kepercayaan di antara *node*, maka model ini tidak cocok digunakan pada jaringan *permissionless blockchain* yang banyak digunakan pada *cryptocurrency* [29].

2.7.4 *Proof of Authority/Proof of Identity*

Model *consensus proof of authority* (juga disebut sebagai *proof of identity*) mengandalkan kepercayaan parsial dari *publishing node* melalui keterkaitan mereka yang diketahui pada identitas dunia nyata. *Publishing node* harus memiliki identitas yang sah dan dapat diverifikasi pada jaringan *blockchain*, misalnya mengidentifikasi dokumen yang telah diverifikasi dan disahkan dalam jaringan *blockchain*. Ide dari model *consensus* ini adalah bahwa setiap *publishing node* menginvestasikan identitas/reputasinya untuk dapat menerbitkan blok baru. *User* pada jaringan *blockchain* ini secara langsung mempengaruhi reputasi *publishing node* berdasarkan perilaku *publishing node*. *Publishing node* dapat kehilangan reputasi mereka jika mereka bertindak dengan cara yang tidak disetujui oleh *user* jaringan *blockchain*, sama seperti mereka yang dapat memperoleh reputasi dengan bertindak sesuai dengan cara yang disetujui oleh *user* jaringan *blockchain*. Semakin rendahnya reputasi sebuah *publishing node*, semakin rendah pula kemungkinan untuk dapat menerbitkan blok. Namun metode ini hanya berlaku pada jaringan *permissioned blockchain* karena membutuhkan tingkat saling kepercayaan yang tinggi [29].

2.7.5 *Proof of Elapsed Time*

Di dalam model *consensus proof of elapsed time* (PoET), setiap *publishing node* melakukan *request* waktu tunggu dari perangkat keras yang aman dalam sistem komputer mereka. Perangkat keras yang aman kemudian akan menghasilkan waktu tunggu secara acak dan mengirimkannya ke *software publishing node*. *Publishing node* mengambil waktu tunggu acak tersebut dan menjadi tidak aktif selama durasi tersebut. Setelah sebuah *publishing node*

bangun dari keadaan tidak aktif, *publishing node* tersebut akan membuat dan menerbitkan blok baru ke jaringan *blockchain* dan memperingatkan *node* lain bahwa blok baru telah diterbitkan. Setiap *publishing node* lainnya yang masih tidak aktif akan berhenti menunggu dan seluruh proses dimulai dari awal [29].

Model ini perlu untuk memastikan bahwa waktu yang digunakan untuk menunggu benar-benar acak, karena jika waktu tunggu tidak dipilih secara acak maka sebuah *publishing node* yang berbahaya hanya perlu menunggu dalam jumlah waktu minimum untuk mendominasi sistem. Model ini juga perlu untuk memastikan bahwa *publishing node* menunggu selama waktu yang telah diberikan dan tidak memulai lebih awal. Persyaratan ini dapat diselesaikan dengan menjalankan *software* pada beberapa prosesor komputer terpercaya seperti *Software Guard Extensions* pada *Intel*, atau *Platform Security Processor* pada *AMD*, atau *TrustZone* pada *ARM* [29].

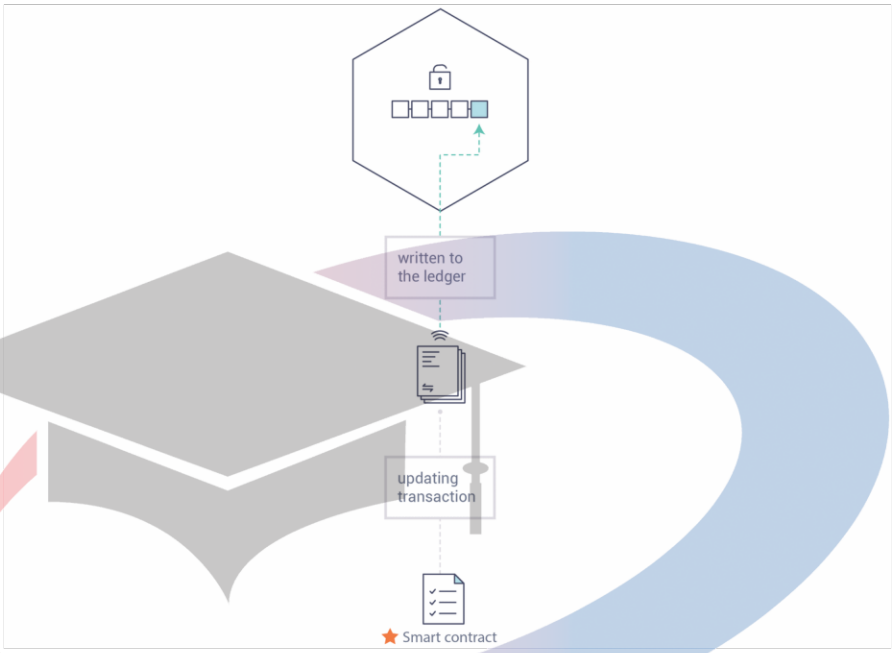
2.8 *Smart Contract*

Smart contract adalah program komputer yang memiliki properti yang dapat memverifikasi diri sendiri, mengeksekusi dengan sendirinya, dan tahan terhadap kerusakan. *Smart contract* dapat mengeksekusi kode tanpa pihak ketiga. Sebuah *smart contract* terdiri dari suatu kumpulan fungsi yang dapat dieksekusi, variabel status, dan diidentifikasi oleh alamat unik [37]. *Smart contract* menerima transaksi sebagai masukan, mengeksekusi kode yang sesuai dan menghasilkan keluaran. *Smart contract* dapat dilacak (*trackable*) dan tidak dapat diubah (*irreversible*). Semua informasi transaksi terdapat di dalam *smart contract* dan dijalankan secara otomatis. Berikut adalah karakteristik *smart contract* [38]:

1. *Smart contract* adalah kode yang dapat dibaca oleh mesin yang dijalankan pada *platform blockchain*
2. *Smart contract* adalah bagian dari satu program aplikasi
3. *Smart contract* adalah program yang dijalankan berdasarkan suatu peristiwa (*event driven program*)
4. *Smart contract* bersifat otonom dan tidak perlu dimonitor setelah dibuat
5. *Smart contract* bersifat terdistribusi

Smart contract adalah kumpulan kode dan data (disebut juga sebagai fungsi dan keadaan) yang di-*deploy* menggunakan transaksi yang ditandatangani secara kriptografis pada jaringan *blockchain,* seperti *smart contracts* pada *Ethereum* dan *chaincode* pada *Hyperledger Fabric*. *Smart contract* dieksekusi oleh *node* yang berpartisipasi di dalam jaringan *blockchain*,

semua *node* yang mengeksekusi *smart contract* harus menghasilkan hasil yang sama, dan hasil eksekusi dicatat pada *blockchain*. *Smart contract* dapat melakukan perhitungan, menyimpan informasi, dan jika sesuai, dapat secara otomatis mengirim dana ke akun lain [29].



Gambar 2.6 Alur *Smart Contract* [39]

Pada banyak implementasi *blockchain*, *publishing node* mengeksekusi kode *smart contract* secara bersamaan saat menerbitkan blok baru. Namun pada beberapa implementasi *blockchain*, *publishing node* tidak mengeksekusi kode *smart contract* melainkan memvalidasi hasil dari *node* yang melakukannya. Pada jaringan *permissionless blockchain* yang menggunakan *smart contract* seperti *Ethereum*, *user* yang melakukan transaksi harus membayar biaya eksekusi kode *smart contract*. Pada *smart contract* juga terdapat limitasi seberapa lama eksekusi dapat dijalankan berdasarkan kompleksitas yang ada pada kode. Jika batasan ini terlampaui, eksekusi diberhentikan, dan transaksi akan dihapus. Mekanisme ini tidak hanya memberikan penerbit hadiah untuk mengeksekusi kode *smart contract*, tetapi juga mencegah *user* yang jahat melakukan *deploy* dan kemudian mengakses *smart contract* yang akan melakukan *Denial-of-Service* (DoS) pada *publishing node* dengan menghabiskan semua sumber daya, misalnya *infinite loop* [29].

Smart contract pada jaringan *permissioned blockchain*, seperti *chaincode* pada *Hyperledger Fabric*, mungkin saja tidak ada persyaratan bagi *user* untuk membayar eksekusi kode *smart contract*. Karena pada jaringan ini dirancang untuk hanya memiliki *user* yang

telah dikenal, dan terdapat metode untuk mencegah perilaku buruk seperti mencabut hak akses (*revoking access*) [29].

2.9 *Hyperledger Fabric Blockchain*

Hyperledger Fabric adalah sebuah platform teknologi *distributed ledger* yang bersifat *open-source*. Platform ini sangat interaktif untuk membuat kerangka kerja dari *blockchain* karena arsitekturnya modular dan dapat dikonfigurasi [40]. *Hyperledger Fabric* tergolong dalam jenis *permissioned blockchain*. *Permissioned blockchain* adalah jenis *blockchain* dimana pihak yang melakukan transaksi dapat menentukan hak akses pihak lain terhadap transaksi yang dilakukan. Dengan demikian, pihak yang tidak berpartisipasi dalam transaksi tersebut tidak bisa melihat isi detail dari proses transaksi [41]. *Hyperledger Fabric*, juga disebut sebagai *Fabric* merupakan sistem *blockchain* pertama yang mendukung pengeksekusian aplikasi terdistribusi yang ditulis dalam bahasa pemrograman standar, yang oleh karena itu membuat *platform* ini dapat dieksekusi di *node* (komputer) pada umumnya [42].

Pada arsitektur tingkat tinggi, *Fabric* terdiri dari komponen modular sebagai berikut [42].

1. *Ordering service*, yang berfungsi mengurutkan transaksi, mengemaskannya ke dalam blok, dan kemudian menyiarkan blok ke semua *peer*
2. *Membership service provider* yang bertanggung jawab untuk menghubungkan *peer* dengan identitas kriptografi
3. *Peer-to-peer gossip service* yang bersifat opsional yang berfungsi untuk menyebarkan blok hasil keluaran oleh *ordering service* kepada semua *peer*
4. *Smart contract* (“*chaincode*”) yang dijalankan di dalam *container*, misalnya *Docker*, untuk tujuan isolasi. *Smart contract* ditulis menggunakan bahasa pemrograman standar namun tidak dapat memiliki akses secara langsung ke *ledger*
5. Setiap *peer* secara lokal memelihara *ledger*-nya sendiri yang terdiri dari *blockchain* dan *world state* yang menyimpan keadaan terakhir dari sebuah objek

2.10 Komponen *Hyperledger Fabric*

Hyperledger Fabric memiliki beberapa komponen penting, seperti *Membership Service Provider*, *Peer*, *Ledger*, *Channel*, *Ordering Service*, *Peer Gossip*, *Chaincode* dan *Organization*. Komponen-komponen tersebut akan dijelaskan dengan rinci pada subbab ini.

2.10.1 *Membership Service Provider*

Membership Service Provider (MSP) merupakan pada komponen abstrak dari sistem yang memberikan kredensial kepada *client* dan *peer* untuk berpartisipasi dalam jaringan *Hyperledger Fabric*. *Client* menggunakan kredensial ini untuk melakukan autentikasi terhadap transaksi mereka, dan *peer* menggunakan kredensial ini untuk melakukan autentikasi pada proses transaksi [36]. MSP memelihara identitas dari semua node dalam sistem *Hyperledger Fabric*. Pemeliharaan identitas ditangani berdasarkan tanda tangan digital. *Node* dapat mengambil satu hingga tiga peran yaitu *client* yang melaksanakan *smart contract*, *peer* yang menjaga salinan *blockchain* serta melakukan validasi transaksi dan *orderer* yang menetapkan urutan semua transaksi dan menggabungkannya menjadi blok.

Sebuah *channel* mengandung informasi MSP organisasi tambahan, seperti *peer* dan *orderer* mana yang bergabung atau keluar dari *channel*. Sebagai contoh, *peer* A mungkin bergabung ke dalam *channel* pada waktu tertentu, dan *peer* B mungkin meninggalkan *channel* pada waktu tersebut. Semua informasi ini akan dikonfigurasi secara dinamis di MSP *channel*. Sehingga *peer* yang baru bergabung ke dalam *channel* dapat menanyakan MSP *channel* untuk mengetahui *peer* lain yang berpartisipasi dalam *channel* tersebut.

MSP adalah abstraksi yang memungkinkan diinstantiasi dengan berbagai cara. Implementasi bawaan MSP pada *Fabric* sudah mencakup standar metode PKI (*Public Key Infrastructure*) untuk otentikasi dan penggunaan *certification authorities* (CA) [42]. Dimana CA yang merupakan bawaan dari *Fabric* adalah *Fabric-CA* yang bertugas untuk mengeluarkan kredensial kepada *peer* (organisasi) dan *user* yang ada di dalam organisasi [43].

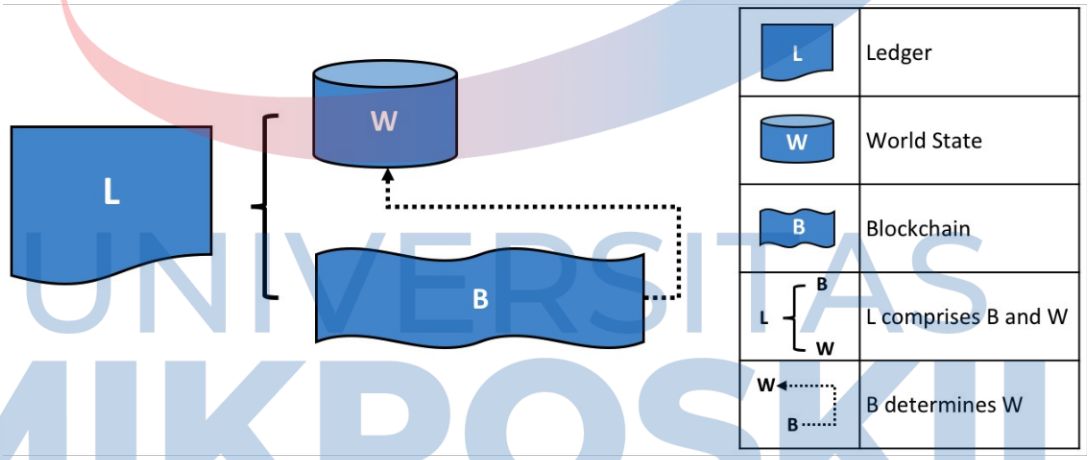
2.10.2 *Peer*

Komponen utama pada jaringan *blockchain Fabric* merupakan kumpulan *peer* karena pada *peer* adalah tempat di mana *ledger* dan *smart contract*, atau disebut sebagai *chaincode* pada *Fabric*, dipelihara. Lebih spesifiknya, *peer* memelihara instansi dari *ledger* dan *chaincode*. Pemeliharaan instansi dari *ledger* dan *chaincode* pada masing-masing *peer* disengaja untuk mencegah terjadinya *single points of failure*, sebuah sistem yang tidak menyimpan redundansinya dan akan terjadi gangguan layanan jika terjadi kegagalan pada sistem tersebut [44]. Karena *ledger* dan *chaincode* dipelihara pada *peer*, maka setiap interaksi pada *ledger* dan *chaincode* harus melalui *peer*. Sehingga membuat *peer* sebagai komponen utama pada jaringan *blockchain Fabric*.

Aplikasi (*client*) akan selalu terhubung ke *peer* ketika ingin melakukan pengaksesan pada *ledger* dan *chaincode*. *Fabric* memiliki *Software Development Kit* (SDK) yang menyediakan API (*Application Programming Interface*) untuk memungkinkan aplikasi terhubung ke *peer*, memanggil *chaincode* untuk menghasilkan transaksi, mengirimkan transaksi ke jaringan yang kemudian akan diurutkan, divalidasi dan disimpan di dalam *ledger*, dan menerima notifikasi ketika proses telah selesai dijalankan [45].

Untuk memastikan semua *ledger* yang ada di dalam jaringan *blockchain* konsisten, sebelum melakukan perubahan pada *ledger*, *peer* akan berinteraksi dengan *peer* lain untuk melakukan simulasi transaksi yang diajukan dan memastikan setiap *peer* menghasilkan keluaran yang sama. Namun tidak semua *peer* melakukan simulasi transaksi yang diajukan, melainkan hanya beberapa di antara mereka yang disebut *endorsing peer,* atau *endorser* [45]. Hasil keluaran masing-masing *endorser* (*endorsement*) kemudian divalidasi oleh *client* untuk memastikan semua *endorsement* sama dan akan membatalkan transaksi jika tidak konsisten.

2.10.3 *Ledger*



Gambar 2.7 Komponen *Ledger* pada *Fabric* [46]

Ledger adalah konsep utama dalam *Fabric*, di mana *ledger* menyimpan informasi faktual mengenai objek bisnis baik keadaan atribut objek saat ini maupun riwayat transaksi yang menghasilkan keadaan objek saat ini. Pada *Fabric*, *ledger* dibagi menjadi dua bagian yang saling berhubungan, yaitu [46]:

1. *World state*

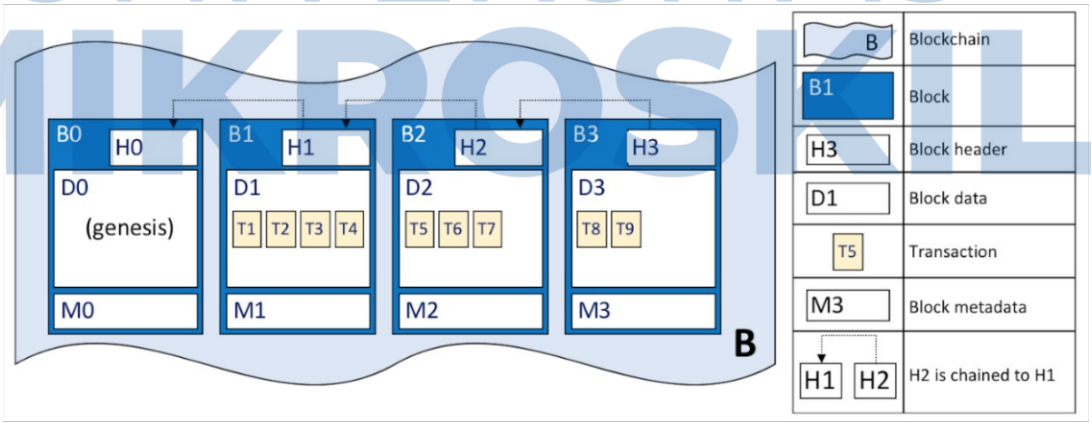
   World state merupakan sebuah *database* yang menyimpan nilai atribut saat ini sebagai sebuah objek. Pada proses bisnis, pengguna biasanya membutuhkan informasi nilai sebuah objek saat ini dan menjadi tidak praktis jika perlu untuk melakukan perhitungan keseluruhan *blockchain* setiap membutuhkan nilai objek saat ini.

World state diimplementasikan dalam *database* karena *database* menyediakan kumpulan operasi yang kaya untuk penyimpanan dan pengambilan status yang efisien. *State* dapat berupa nilai sederhana atau majemuk, dan implementasi *database world state* dapat beragam untuk mendukung nilai yang disimpan dapat dioperasikan secara efisien. Pilihan untuk *database world state* dapat berupa *LevelDB*, *database key-value store* yang memetakan kunci teks (*string key*) ke nilai teks (*string value*) [47], dan *CouchDB*, *database document store* dimana setiap dokumen memiliki sebuah kunci unik (*unique key*) yang digunakan untuk mendapatkan dokumen dalam bentuk JSON (*JavaScript Object Notation*) [48].

Setiap *state* memiliki nomor versi yang dimulai dari 0. Nomor versi digunakan untuk kepentingan *Internal Fabric* dan bertambah setiap ada perubahaan pada *state*. Nomor versi digunakan untuk pengecekan bahwa nomor versi *state* saat ini sama dengan saat perubahan objek *state*. Pengecekan nomor versi ini dilakukan untuk memastikan tidak ada perubahan yang terjadi secara bersamaan (*concurrency*) yang dapat mengakibatkan *ledger* tidak konsisten.

2. *Blockchain*

Blockchain menyimpan blok transaksi dan memiliki struktur membentuk sebuah rantaian blok berurutan yang saling terkait, dimana setiap blok berisi kumpulan transaksi yang berurutan dan setiap transaksi merupakan catatan perubahan pada *world state*. *Blockchain* diimplementasikan sebagai *file* karena operasi utama pada *blockchain* hanya penambahan blok pada akhir *blockchain* dan operasi yang kompleks relatif jarang dilakukan.

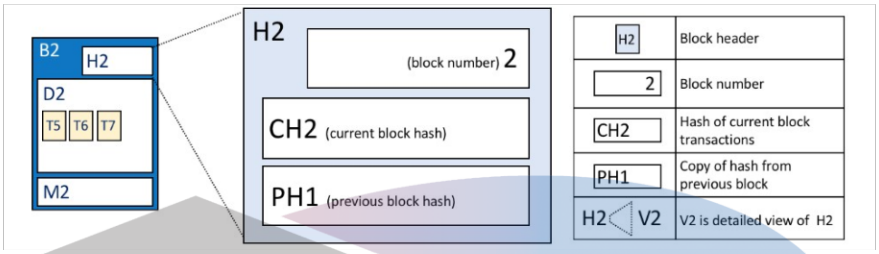


Gambar 2. 8 Struktur Blockchain pada Fabric [46]

Blok pertama pada *blockchain* disebut sebagai *genesis block* dan merupakan titik awal untuk *ledger* meskipun tidak memiliki transaksi apapun melainkan berisi konfigurasi transaksi keadaan awal suatu *channel*.

Adapun detail struktur blok dan transaksi dijelaskan sebagai berikut.

1. Blok

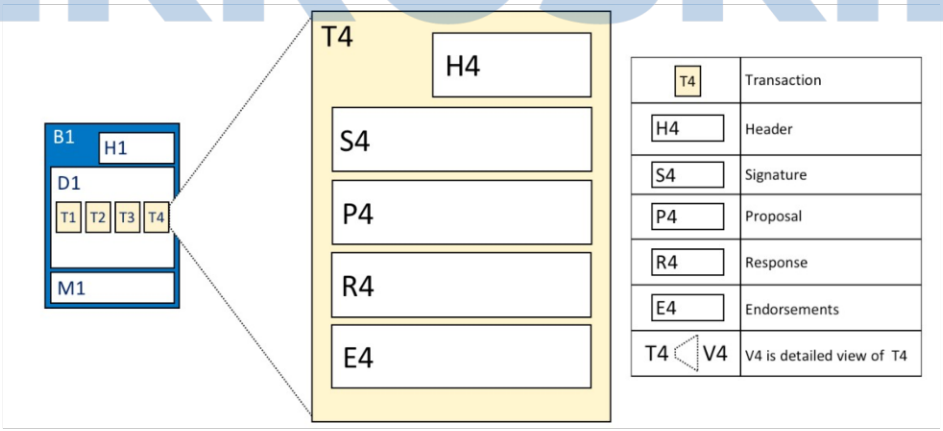


Gambar 2.9 Struktur Blok pada Fabric [46]

Di dalam blok terdiri dari tiga bagian, yaitu:

a. *Block Header*, bagian ini terdiri dari tiga data, yaitu sebuah bilangan bulat yang dimulai dari 0 dan bertambah 1 setiap blok baru ditambahkan ke dalam *blockchain* (*block number*), nilai *hash* seluruh transaksi pada blok ini (*current block hash*) dan nilai *hash block header* sebelumnya (*previous block header hash*)
b. *Block Data*, pada bagian ini terdiri dari kumpulan transaksi yang diatur secara berurutan
c. *Block Metadata*, pada ini berisi sertifikat (*certificate*) dan tanda tangan pembuat blok yang digunakan untuk memeriksa blok oleh *node* pada jaringan. Indikator bahwa transaksi valid/tidak juga ditulis pada *block metadata* dalam bentuk *bitmap* maupun nilai *hash* dari *state* kumulatif hingga dan termasuk blok ini. Bagian ini tidak termasuk sebagai masukan komputasi *block hash*

2. Transaksi



Gambar 2.10 Struktur Transaksi pada Fabric [46]

Di dalam transaksi terdapat data sebagai berikut.

a. *Header*, mencatat beberapa *metadata* penting tentang transaksi tersebut, seperti nama *chaincode* dan nomor versi.
b. *Signature*, mengandung tanda tangan kriptografi yang ditandatangani oleh *client*. Data ini digunakan pengecekan bahwa detail transaksi belum dirusak, karena tanda tangan ini membutuhkan *private key client*.
c. *Proposal*, berisi parameter masukan yang diberikan oleh *client* kepada *smart contract* yang menghasilkan pengajuan perubahan *ledger*.
d. *Response*, mencatat nilai sebelum dan sesudah perubahan *world state* dalam bentuk *Read Write set* (*RW-set*). Data ini merupakan keluaran *smart contract* dan jika transaksi bersifat valid akan diterapkan pada *ledger* untuk melakukan perubahan pada *world state*.
e. *Endorsement*, berisi kumpulan *response* transaksi yang telah ditandatangani oleh *endorser*.

2.10.4 *Channel*

Channel adalah tempat seluruh pihak yang bertransaksi dapat berinteraksi. *Channel* membuat transaksi tersembunyi dan hanya bisa dilihat oleh *user* yang berada di *channel* yang sama. *Organization* yang berada di *channel* tersebut otomatis dapat melihat seluruh data dan melakukan transaksi di *channel* tersebut. Namun, ada beberapa transaksi yang memerlukan hak akses, seperti hanya *organization* tertentu yang dapat melakukan transaksi kepada *organization* yang lain [40]. Transaksi yang berjalan di dalam suatu *channel* tidak dapat diakses dari *channel* yang lain dan tidak ada hubungan antar satu *channel* dengan *channel* lainnya [49].

Sebuah *channel* didefinisikan pada blok konfigurasi (*configuration block*), yang berisikan data konfigurasi yang menentukan anggota dan aturan pada *channel* tersebut. Pada saat *channel* baru dibuat, maka perlu ditentukan anggota yang memiliki hak akses terhadap *channel* tersebut, dan izin apa yang anggota-anggota tersebut miliki. Seperti hanya beberapa anggota saja yang dapat melihat dan mengubah isi buku besar transaksi, sementara anggota lain tidak dapat melakukannya. Setiap ada modifikasi konfigurasi pada sebuah *channel*, seperti seorang anggota meninggalkan atau bergabung akan menghasilkan blok konfigurasi baru yang ditambahkan pada rantaian (*chain*). Blok ini akan berisi konten dari *genesis block* ditambah dengan perubahannya [50].

2.10.5 *Ordering Service*

Orderer juga merupakan salah satu *node* di dalam jaringan *blockchain* [51]. *Orderer* adalah kumpulan *node* yang menerima dan mengumpulkan transaksi ke dalam blok lalu mendistribusikan blok ke *peer* yang terhubung agar divalidasi [36]. *Orderer* memiliki fungsi untuk mengelola dan memelihara konfigurasi *channel* dan pengeksekusian proses transaksi. Di dalam konfigurasi *channel*, *orderer* dapat melaksanakan kontrol akses dasar pada *channel*, membatasi organisasi yang dapat membaca dan menulis data dan yang dapat melakukan konfigurasi pada *channel* (kumpulan organisasi ini dikenal sebagai "*consortium*"), yang mana telah dikonfigurasi pada blok konfigurasi. Setiap terdapat perubahan konfigurasi yang valid, *orderer* akan membuat sebuah transaksi konfigurasi baru dan menyimpannya pada sebuah blok yang kemudian disampaikan kepada semua *peer* yang ada di dalam *channel* yang sama [52].

Orderer juga memiliki fungsi untuk melakukan pengurutan transaksi bersama dengan *orderer* lainnya sehingga membentuk *ordering service* [52]. Hasil pengurutan transaksi kemudian dikelompokkan menjadi blok. Blok dipotong setelah salah satu dari tiga kondisi terpenuhi yaitu blok mencapai jumlah maksimal transaksi (*batchSize*), blok mencapai ukuran maksimal (dalam *byte*), atau lama waktu yang telah berlaku sejak *transaction* pertama pada blok ini diterima (*batchTimeout*) [53].

Proses pengelompokan bersifat deterministik dan oleh karena itu menghasilkan blok yang sama pada semua *node*. Selain deterministik, memisahkan dukungan eksekusi *chaincode* (yang mana terjadi pada *peer*) dari *orderer* memberikan *Fabric* keuntungan dalam segi performa dan skalabilitas, menghilangkan kemacetan (*bottleneck*) yang dapat terjadi ketika pengeksekusian dan pengurutan dilakukan pada *node* yang sama [52].

Pada implementasi *ordering service*, terdapat beberapa cara untuk mencapai kesepakatan bersama (*consensus*) yaitu sebagai berikut.

1. *Solo*

   Solo bersifat *orderer* terpusat. *Solo* beroperasi tanpa algoritma *consensus* dan hanya memiliki satu *ordering node*. Implementasi *solo* ditujukan untuk penggunaan pada *development*.

2. *Kafka*

   Kafka adalah *ordering service* yang bersifat *crash fault tolerant* (CFT) dimana operasi tetap dapat dijalankan meskipun beberapa *node* mengalami kegagalan dan terdapat

$N/2+1$ *node* yang masih berjalan [54]. Implementasi *consensus* ini menggunakan *node* konfigurasi "*leader and follower*" dan memanfaatkan *ZooKeeper Ensembled* [52].

3. *Raft*

Implementasi *Raft* mirip dengan *Kafka* karena *Raft* juga bersifat CFT dan mengikuti model "*leader and follower*", dimana *leader node* secara dinamis dipilih di antara *orderer* di dalam *channel* (kumpulan *node* ini juga disebut sebagai "*consenter set*") dan *leader node* mereplikasi keputusan yang diambil kepada *follower node* [52]. Aplikasi yang akan dikembangkan pada tugas akhir ini menggunakan *Raft*.

a. Konsep *Raft*

i. *Log entry*. Kumpulan urutan entri (atau dikenal sebagai "*log*") merupakan *log entry*. *Log* dapat dikatakan konsisten jika mayoritas (*quorum*) anggota setuju pada entri dan urutannya.

ii. *Consenter set*. Kumpulan *orderer* yang secara aktif berpartisipasi dalam mekanisme *consensus* dan menerima *log* yang telah direplikasi kepada *channel*. Kumpulan *orderer* dapat berarti seluruh *node* yang tersedia ataupun sebagian dari *node* tersebut.

iii. *Finite-State Machine* (FSM). Setiap *orderer* di dalam *Raft* memiliki sebuah FSM dan digunakan untuk memastikan bahwa urutan *log* pada berbagai *orderer* bersifat deterministik.

iv. *Quorum*. Jumlah minimum *consenter* yang dibutuhkan untuk mengesahkan sebuah proposal sehingga transaksi dapat diurutkan. Pada setiap kumpulan *consenter*, *quorum* merupakan mayoritas dari kumpulan *node* tersebut. Di dalam klaster dengan lima *node*, tiga di antaranya harus tersedia agar terciptanya *quorum*. Jika mayoritas dari *node* tidak tersedia, klaster *ordering service* menjadi tidak tersedia untuk melakukan operasi *read* dan *write* pada *channel* dan tidak ada *log* baru yang dapat dicatat.

v. *Leader*. *Consenter set* dalam sebuah *channel* melakukan pemilihan sebuah *node* untuk dijadikan *leader*. *Leader* bertanggung jawab untuk menerima entri *log* baru, mereplikasikannya kepada *follower node*, dan mengelola entri untuk dicatat.

vi. *Follower*. *Follower* menerima *log* dari *leader* dan mereplikasikannya secara deterministik, memastikan bahwa *log* tetap bersifat konsisten. Pada pemilihan *leader*, *follower* juga menerima detak jantung ("*heartbeat*") dari *leader*. Jika

leader berhenti mengirimkan *heartbeat* pada waktu yang telah ditentukan, *follower* akan memulai pemilihan *leader* dan salah satu di antara mereka akan dipilih sebagai *leader* yang baru.

b. Alur transaksi *Raft*

Setiap *channel* menjalankan instansi protokol *Raft* yang berbeda, sehingga setiap *channel* dapat memiliki *leader* yang berbeda. Mekanisme ini membuat tercapainya desentralisasi servis dimana ketika klaster terdiri dari *orderer* yang dikontrol oleh organisasi yang berbeda-beda. Semua *node Raft* harus merupakan bagian dari *system channel* namun tidak diwajibkan untuk menjadi bagian dari semua *application channel*. Pembuat *channel* dan *admin channel* memiliki kemampuan untuk memilih beberapa dari *orderer* yang tersedia dan menambah atau menghapus *orderer* sesuai kebutuhan [52].

Dalam *Raft*, transaksi pada bentuk proposal atau perubahan konfigurasi secara otomatis diarahkan oleh *orderer* yang menerima transaksi tersebut kepada *leader* saat ini pada *channel* tersebut. Sehingga *peer* dan aplikasi tidak perlu mengetahui siapa *leader node* saat ini, hanya *orderer* yang perlu mengetahuinya [52].

c. Cara pemilihan *leader* bekerja pada *Raft*

Node pada *Raft* selalu berada pada salah satu dari ketiga status: *follower*, *candidate*, atau *leader*. Semua *node* mulai sebagai *follower*. Pada keadaan ini, *node* menerima entri *log* dari *leader*, atau memberikan suara untuk pemilihan *leader*. Jika tidak ada entri *log* atau *heartbeat* yang diterima dalam sejumlah waktu, misalnya lima detik, *node* mempromosikan diri sendiri menjadi status *candidate*. Dalam keadaan status *candidate*, *node* meminta suara dari *node* lain. Jika *candidate* menerima *quorum* suara, maka *node* tersebut dipromosikan menjadi *leader*. *Leader* harus menerima entri *log* baru dan mereplikasikannya kepada *follower* [52].

2.10.6 *Peer Gossip*

Salah satu keuntungan dari memisahkan fase pengeksekusian, pengurutan dan validasi adalah ketiga fase tersebut dapat dikembangkan secara independen. Namun, karena pada kebanyakan algoritma *consensus* pada model *Crash Fault Tolerant* dan *Byzantine Fault Tolerant* terikat pada *bandwidth*, jumlah keluaran dari *orderer* menjadi terbatas pada kapasitas jaringan *node*-nya. Namun, karena *fase* pengurutan dan validasi terpisah, penyebaran hasil eksekusi kepada semua *peer* untuk divalidasi setelah fase pengurutan dan

pengiriman *state* kepada *peer* yang baru bergabung dan *peer* yang telah lama terputus dapat dilakukan secara efisien dengan memanfaatkan *epidemic multicast*, setiap *node* menyampaikan setiap pesan kepada kumpulan *node* tetangga yang terpilih secara *random* [55]. Karena blok telah ditandatangani, setiap *peer* dapat menerima semua blok dan secara independen membuat *blockchain*-nya dan melakukan verifikasi integritas blok [53].

Fabric gossip menggunakan dua fase untuk penyebaran informasi, yaitu *push*, setiap *peer* meneruskan pesan kepada kumpulan *peer* di dalam *channel* secara *random*, dan *pull*, setiap *peer* secara berkala memeriksa kumpulan *peer* secara acak dan meminta pesan yang hilang. Untuk mengurangi beban jumlah pengiriman blok dari *orderer* ke jaringan, protokol ini juga melakukan pemilihan *leader peer* dalam melakukan pemintaan blok dari *ordering service* atas inisiatif sendiri dan memulai distribusi *gossip* [53].

2.10.7 *Chaincode*

Hyperledger Fabric memanfaatkan teknologi kontainer untuk meng-*host smart contract* yang disebut dengan *chaincode*. *Chaincode* berisi aturan bisnis dari sistem. *Chaincode* dirancang untuk mendukung beberapa komponen yang *pluggable* dan untuk mengakomodasi kompleksitas yang ada di bidang perekonomian [56]. Proses eksekusi dari *chaincode* terpisah dari penanganan transaksi untuk membatasi tingkat kepercayaan dan verifikasi yang diperlukan di seluruh *node* dan mengoptimalkan skalabilitas dan kinerja jaringan [57].

Definisi *chaincode* digunakan oleh organisasi untuk menyetujui parameter *chaincode* sebelum *chaincode* dapat digunakan pada *channel*. Setiap anggota *channel* yang ingin menggunakan *chaincode* untuk mendukung transaksi mereka harus menyetujui definisi dari *chaincode* untuk organisasi mereka. Setelah banyak anggota *channel* menyetujui definisi *chaincode* tersebut, *chaincode* dapat diterapkan ke *channel*.

Secara umum, *smart contract* merupakan logika transaksi yang mengontrol siklus hidup objek bisnis yang terdapat pada *world state*. *Smart contract* kemudian dikelompokkan menjadi sebuah *chaincode* yang akan di-*deploy* ke jaringan *blockchain*. Sehingga *smart contract* berfungsi sebagai yang mengatur transaksi, sedangkan *chaincode* berfungsi untuk mengatur bagaimana *smart contract* dikelompokkan untuk di-*deploy* [39].

Smart contract memiliki fungsi *put*, *get*, dan *delete state* pada *world state* dan juga dapat melakukan *query* pada *blockchain* yang tidak dapat diubah (*immutable*) [39]:

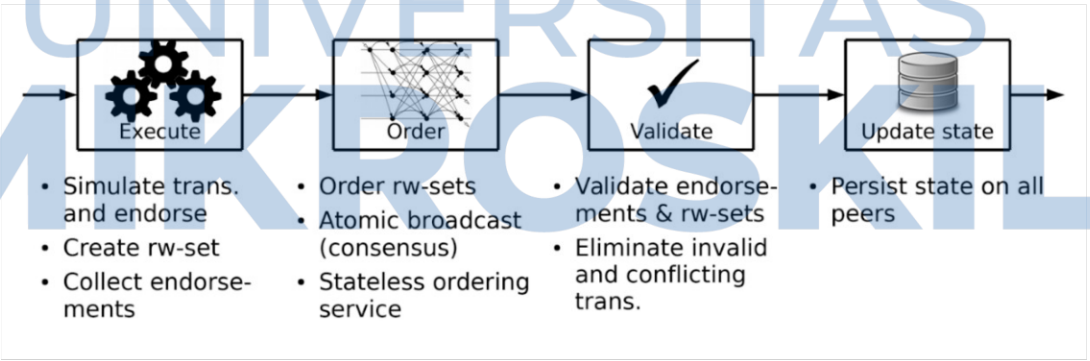
1. *Get*, berfungsi melambang *query* untuk mengambil informasi tentang *state* saat ini sebuah objek.
2. *Put*, berfungsi membuat objek yang baru atau memodifikasi yang sudah ada pada *world state*.
3. *Delete*, berfungsi untuk menghapus objek dari *state* saat ini pada *world state*, namun tidak menghapus riwayatnya yang ada pada *blockchain*.

2.10.8 *Organization*

Organization disebut juga sebagai anggota dan setiap *node* diwakilkan oleh organisasi. Sebuah organisasi bergabung ke jaringan dengan menambahkan *Membership Service Provider* (MSP) dari organisasinya ke dalam jaringan tersebut. MSP mendefinisikan bagaimana anggota tersebut memiliki identitas yang valid. Sebuah organisasi dapat berupa perusahaan multi-nasional atau bahkan bisa satu individu [36].

2.11 Arsitektur *Hyperledger Fabric*

Fabric memperkenalkan arsitektur *blockchain execute-order-validate*, tidak mengikuti arsitektur *order-execute* yang mana jaringan *blockchain* melakukan pengurutan transaksi terlebih dahulu kemudian mengeksekusinya dalam urutan yang sama pada semua *peer*, misalnya pada *Ethereum*. Pada subbab ini, akan dijelaskan alur transaksi dalam ketiga fase tersebut.



Gambar 2.11 Arsitektur *Execute-Order-Validate* pada *Fabric* [53]

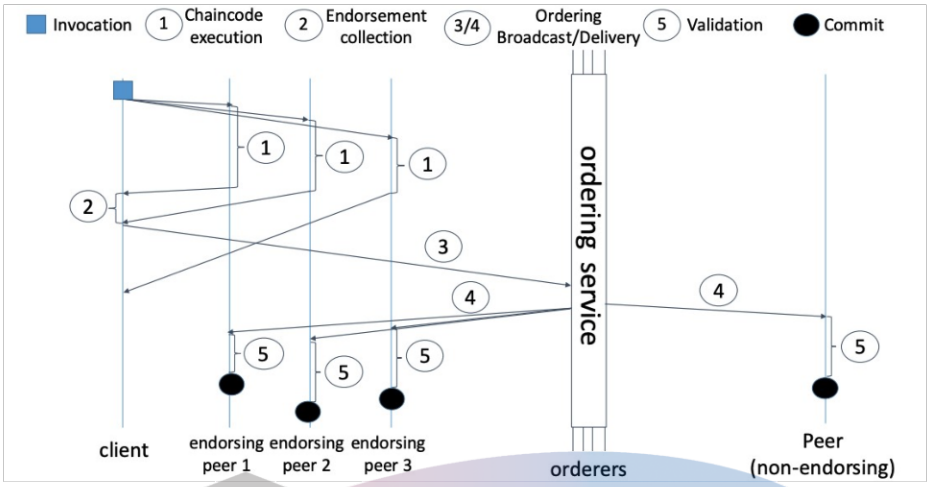
2.11.1 *Execution Phase*

Pada fase pengeksekusian (*execution phase*), *client* menandatangani dan mengirimkan *transaction proposal* ke satu atau beberapa *endorser* untuk dieksekusi. Untuk menentukan kumpulan *endorser* yang diperlukan agar proposal dapat diterima oleh jaringan *blockchain*, *client* memanfaatkan *endorsement policy*. *Endorsement policy* merupakan sebuah aturan yang

dimuat di dalam *chaincode* yang mengharuskan sebuah transaksi untuk didukung oleh jumlah minimal *endorser*, minimal persentase *endorser*, atau oleh semua *endorser* yang telah ditentukan secara spesifik [36]. Sebuah proposal terdiri dari identitas *client*, isi transaksi dalam bentuk operasi pengeksekusian, parameter, dan ID *chaincode*, *nonce* (seperti penghitung atau nilai acak), dan ID transaksi yang didapatkan dari ID *client* dan *nonce* [53].

Sebuah proposal hanya melakukan simulasi pengeksekusian transaksi pada *blockchain* lokal *endorser*. Namun, *endorser* tidak menyimpan hasil simulasi ke dalam *ledger*. *State* yang dihasilkan oleh *chaincode* hanya berada di dalam cakupan *chaincode* tersebut dan tidak dapat diakses secara langsung oleh *chaincode* lain. Sebuah *chaincode* dapat memanggil *chaincode* lain untuk mengakses *state* tersebut yang ada pada *channel* yang sama (*intercommunication*) [39]. Perlu diketahui bahwa *chaincode* tidak boleh mengakses *state* secara langsung, melainkan untuk berinteraksi dengan *state blockchain* harus melalui operasi *GetState*, *PutState*, dan *DelState* [53].

Pada hasil simulasi pengeksekusian transaksi, *endorser* menghasilkan *writeset*, yang terdiri dari perubahan *state* yang dihasilkan oleh simulasi tersebut, dan *readset*, berisi kumpulan nomor versi sebuah transaksi yang dicatat pada saat simulasi berlangsung [58]. Setelah melakukan simulasi, *endorser* akan menghasilkan *endorsement* dengan menandatangani secara digital *readset*, *writeset* serta *metadata* seperti ID transaksi, ID *endorser*, dan tanda tangan *endorser* dan kemudian mengembalikannya kepada *client* yang disebut sebagai *proposal response*. *Client* akan mengumpulkan *endorsement* hingga memenuhi persyaratan *endorsement policy* dan setiap *endorser* wajib untuk menghasilkan *endorsement* yang sama, yaitu memiliki *writeset* dan *readset* yang identik. *Client* bebas untuk memberhentikan proposal lebih awal jika terdapat proposal *response* yang tidak konsisten. *Client* kemudian membuat transaksi yang mengandung *endorsement* yang dihasilkan oleh *endorser* dan mengirimkannya kepada *ordering service* (*orderer*).



Gambar 2.12 *Flow* Transaksi Tingkat Tinggi pada *Fabric* [53]

2.11.2 *Ordering Phase*

Pada fase ini, *client* sudah mengumpulkan *endorsement* melalui fase pengeksekusian kemudian membuat sebuah transaksi dan mengirimkannya kepada *ordering service*. Transaksi mengandung isi transaksi seperti operasi *chaincode* dan parameter, metadata transaksi, dan kumpulan *endorsement*. Penetapan urutan keseluruhan transaksi yang diajukan per *channel* dilakukan pada fase ini, dengan demikian *consensus* pada transaksi tercapai [53]. *Ordering service* mengelompokkan kumpulan transaksi ke dalam blok dan menghasilkan urutan rantaian *hash* blok yang mengandung kumpulan transaksi.

Pada antarmuka tingkat tinggi, *ordering service* hanya mendukung kedua operasi berikut yang dipanggil oleh sebuah *peer* dan secara implisit telah diparameterisasi oleh ID *channel* [53]:

1. $broadcast(tx)$: *client* memanggil operasi ini untuk menyiarkan sebuah transaksi tx yang mengandung isi transaksi dan tanda tangan dari *client* untuk disebarkan
2. $B \leftarrow deliver(s)$: *client* memanggil operasi ini untuk menerima *block* B dengan nomor urut non-negatif s. Blok memuat kumpulan transaksi $[tx_1,\ldots,tx_k]$ dan nilai rantaian *hash* h mewakili blok dengan nomor urut $s-1$ (contoh $B=([tx_1,\ldots,tx_k],h)$. Karena *client* dapat memanggil operasi tersebut berkali-kali dan selalu menerima blok yang sama, maka dapat dikatakan bahwa *peer* membuat blok B dengan nomor urut s ketika *peer* menerima B untuk pertama kalinya sejak memanggil operasi $deliver(s)$

Ordering service menjamin bahwa semua blok yang telah dibuat pada suatu *channel* secara keseluruhan terurut. Secara spesifik, *orderer* menjamin properti keamanan berikut untuk setiap *channel* [53]:

1. *Agreement*: Untuk setiap dua blok B dengan nomor urut s dan B' dengan nomor urut s' dimana $s=s'$, maka $B=B'$
2. *Hash chain integrity*: Jika *peer* membuat blok B dengan nomor s dan *peer* lain membuat blok $B^{'}=\left(\left[t\,x_1,\ldots,t\,x_k\right],h'\right)$ dengan nomor $s+1$, maka $h^{'}=H\left(B\right)$ dimana $H\left(\cdot\right)$ merupakan fungsi *hash* kriptografi
3. *No skipping*: Ketika *peer* p membuat blok dengan nomor $s>0$ maka untuk setiap $i=0,\ldots,s-1$, *peer* p telah membuat blok dengan nomor i
4. *No creation*: Ketika *peer* membuat blok B dengan nomor s, maka untuk setiap $tx \in B$ beberapa *client* telah menyiarkan tx

Untuk keaktifan, *ordering service* mendukung properti terakhir yaitu [53]:

Validity: Jika sebuah *client* memanggil $broadcast\left(tx\right)$, maka setiap *peer* akan membuat sebuah blok x yang mengandung tx dengan sebuah nomor urut

2.11.3 *Validation Phase*

Blok kemudian disebarkan ke seluruh *peer* baik secara langsung oleh *ordering service* maupun melalui *gossip*. Blok baru tersebut masuk ke fase validasi yang memiliki tiga langkah berurutan yaitu [53]:

1. Evaluasi *endorsement policy* dilakukan secara paralel pada semua transaksi yang ada di dalam blok. Evaluasi ini disebut sebagai *validation system chaincode* (VSCC), sebuah konfigurasi yang merupakan bagian dari *blockchain* yang bersifat statik dan bertanggung jawab atas validasi *endorsement* menggunakan aturan yang ada pada *endorsement policy* yang telah dikonfigurasi pada *chaincode*. Jika *endorsement* tidak memenuhi persyaratan, maka transaksi ditandai sebagai tidak valid dan diabaikan.
2. Pengecekan konflik *read-write* dilakukan pada semua transaksi yang ada di dalam blok secara berurutan. Pengecekan setiap transaksi dilakukan dengan membandingkan nomor versi transaksi yang ada pada *readset* dengan *state* yang ada pada *ledger* lokal dan memastikan kedua nomor versi tersebut sama. Jika nomor versi tidak sama, maka transaksi ditandai sebagai tidak valid dan diabaikan.
3. Fase perubahan *ledger* dimana blok ditambahkan pada *ledger* lokal dan *state blockchain* diubah. Ketika menambahkan blok ke dalam *ledger*, hasil pengecekan yang ada pada kedua tahap sebelumnya juga akan disimpan dalam bentuk *bitmask* yang menandakan apakah transaksi valid atau tidak valid. Dengan mencatat sebuah transaksi valid atau tidak dapat memudahkan rekonstruksi *state blockchain* di lain waktu. Semua perubahan

dilakukan dengan menerapkan hasil perubahan *state* yang ada pada *writeset*. Transaksi yang ditandai sebagai tidak valid tidak akan diterapkan pada *state blockchain*.

2.12 *Test-Driven Development*

Test-Driven Development (TDD) adalah praktik pengembangan perangkat lunak *agile* yang mendukung inovasi dalam siklus pengembangan singkat [59]. TDD bukan aktivitas pengujian saja, melainkan adalah aktivitas desain dan pemrograman. Aspek pengujian dari TDD sebagian besar bersifat pengecekan ulang [60]. Manfaat utama yang diklaim dari TDD adalah peningkatan kualitas produk [61].

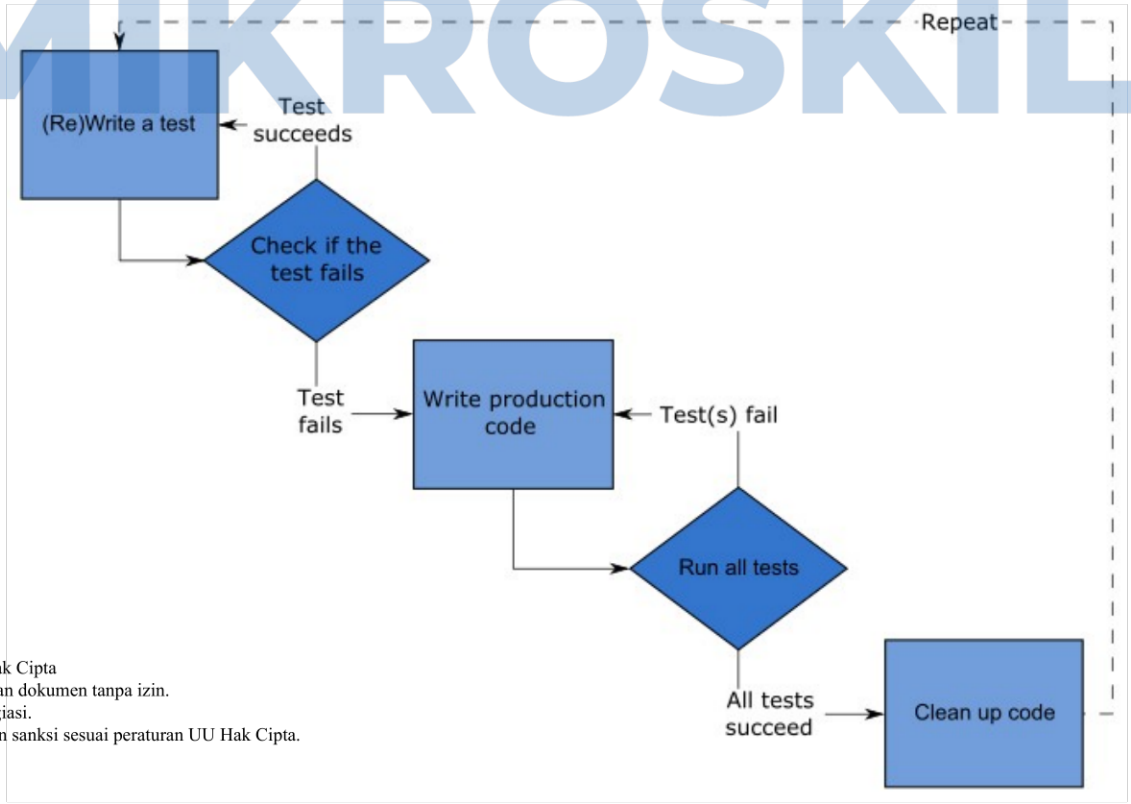
Dalam TDD, hal-hal yang dilakukan adalah menuliskan kode baru hanya jika *testing* yang dilakukan bersifat gagal dan mengeliminasi duplikasi. TDD memiliki sebuah sinonim yaitu *Red-Green-Refactor* yang memiliki definisi [62]:

1. *Red* adalah situasi pada saat menuliskan *test* yang sedikit, yang bahkan hasilnya akan gagal di-*compile*.
2. *Green* adalah situasi pada saat membuat kode agar *test* berhasil. *Red-Green* akan dilakukan berulang-ulang sehingga menghasilkan sebuah fitur yang berhasil dijalankan.
3. *Refactor* adalah situasi pada saat mengeliminasi kode duplikasi yang telah dibuat.

2.12.1. *Test-Driven Development* pada *Blockchain*

Blockchain tidak dapat diubah dalam arti bahwa, setelah disetujui, transaksi akan tetap ada dan tidak dapat dihapus. Karena penerapan *smart contract* terjadi melalui transaksi, hal ini mengakibatkan ketidakmampuan untuk memperbaiki masalah dengan cepat [63]. Pengujian harus memastikan bahwa *smart contract* berfungsi seperti yang diharapkan [63]:

1.
2.
3.
1.
2.
3.
4.
5.



Mocking adalah fenomena pengujian unit yang membantu menguji objek secara terpisah dengan mengganti objek dependen yang memiliki perilaku kompleks dengan objek uji yang memiliki perilaku yang telah ditentukan/disimulasikan. Objek uji ini disebut sebagai objek *Mock*. Pada *Hyperledger Fabric*, tersedia *package shim* yang berisi implementasi *MockStub* yang membungkus panggilan ke *chaincode* (*smart contract*), mensimulasikan perilakunya di lingkungan *peer* HLF. *MockStub* tidak perlu memulai beberapa *docker container*, *world state database*, *chaincode* dan memungkinkan untuk mendapatkan hasil pengujian dengan lebih cepat. *MockStub* pada dasarnya menggantikan SDK dan lingkungan *peer*, serta memungkinkan untuk menguji *chaincode* tanpa perlu memulai jaringan *blockchain* [63].

2.13 *Black-box Testing*

Black-box testing adalah pengujian berdasarkan spesifikasi *requirement* dan tidak perlu dilakukan pemeriksaan kode program. Pengujian ini murni dilakukan berdasarkan sudut pandang pelanggan dan dilakukan pada produk yang benar-benar sudah jadi [64]. *Black-box testing* memiliki peran penting dalam pengujian perangkat lunak, karena membantu validasi fungsionalitas secara keseluruhan dari sebuah sistem. *Black-box testing* dilakukan berdasarkan kebutuhkan klien, sehingga jika ada yang tidak lengkap atau *requirement* tidak terduga dapat diidentifikasi dan dapat diatasi [65].

Keuntungan dari *black-box testing* adalah penguji tidak perlu memiliki pengetahuan tentang bahasa pemrograman dan implementasi. Dalam *black-box testing*, *programmer* dan penguji saling independen tanpa ada hubungan apapun. Keuntungan lainnya adalah karena pengujian dilakukan dari sudut pandang pengguna sehingga lebih membantu untuk mengekspos ambiguitas atau inkonsistensi dalam *requirement* [65].

2.14 Penelitian Terdahulu

Dalam jurnal [66] ada dibahas tentang sistem *traceability* produk berdasarkan teknologi *blockchain*, di mana semua riwayat perjalanan produk secara terus-menerus dicatat dalam buku besar yang didistribusikan dengan menggunakan *smart contract* dan rantai dibentuk agar dapat melacak kembali ke sumber produk. Pada jurnal tersebut, telah dirancang sebuah mekanisme untuk memverifikasi identitas kedua pihak transaksi, sehingga validitas dari transaksi dapat dijamin. Seluruh peristiwa disimpan secara permanen di *blockchain* dalam bentuk *log*. Hasil akhir dari penelitian tersebut adalah sebuah aplikasi terdesentralisasi berdasarkan kerangka *truffle*, *smart contract* diterapkan dan diuji melalui jaringan uji *TestRpc* yang berjalan sepenuhnya di memori lokal, dan sebuah halaman web interaksi juga telah diimplementasikan.

Salah satu jurnal lainnya yang membahas tentang *blockchain* pada *supply chain* agroindustri adalah jurnal ini [67]. Dalam jurnal tersebut, diusulkan sebuah sistem yang seluruh transaksinya ditulis ke *blockchain* yang pada akhirnya diunggah ke *Interplanetary File Storage System* (IPFS). Sistem penyimpanan akan mengembalikan *hash* data yang disimpan di *blockchain* dan memastikan solusi yang efisien, aman dan handal. Sistem tersebut juga menyediakan *smart contract* beserta algoritmanya untuk menunjukkan interaksi entitas dalam sistem. Sebuah rantai pasok tentunya melibatkan beberapa proses dan sub-proses yang perlu dilakukan secara terdesentralisasi untuk mencapai *traceability* dan keamanan, Sehingga, pada jurnal tersebut telah diusulkan sebuah solusi *end-to-end* untuk rantai pasok agroindustri berbasis *blockchain*. Pada jurnal tersebut kinerja *smart contract* telah dievaluasi dan dianalisis secara cermat untuk memastikan bahwa solusi yang diusulkan telah kuat dan efisien.