BAB 2
TINJAUAN PUSTAKA

2.1 Profil Perusahaan

BBPLK Bandung adalah salah satu pusat pelatihan terlengkap untuk menyelenggarakan program-program pelatihan dan pengembangan kompetensi dan keterampilan sumber daya manusia di bidang industri jasa dan manufaktur. Kemitraan dengan berbagai institusi di dalam maupun Luar Negeri merupakan salah satu langkah strategi BBPLK Bandung guna menunjang kualitas program pelatihan serta menambah ruang lingkup pelayanannya. Dalam pengembangan program dan kelembagaan BBPLK Bandung sejak tahun 2001 telah melaksanakan kerjasama kelembagaan dengan *Indonesian German Institute* (IGI) *Alliance For Training, Consultancy and Production*. BBPLK Bandung telah melaksanakan sistem manajemen mutu ISO 9001 & 2000 sejak bulan Juli 2004 dan telah mendapatkan Sertifikat SAI *Global Sertification Service Pty. Ltd* pada bulan Februari 2005. Sejak tanggal 15 Maret 2006 Balai Besar Pengembangan Latihan Kerja (BBPLK/NVTDC) Bandung adalah UPT di bidang pengembangan pelatihan instruktur, tenaga pelatih serta tenaga kerja yang berada di bawah dan bertanggung jawab kepada Direktur Jenderal Pembinan Pelatihan dan Produktivitas Kemenakertrans R.I.

Gambar 2.1 Logo BBPLK Bandung

2.2 Landasan Teori

2.2.1 Sertifikat

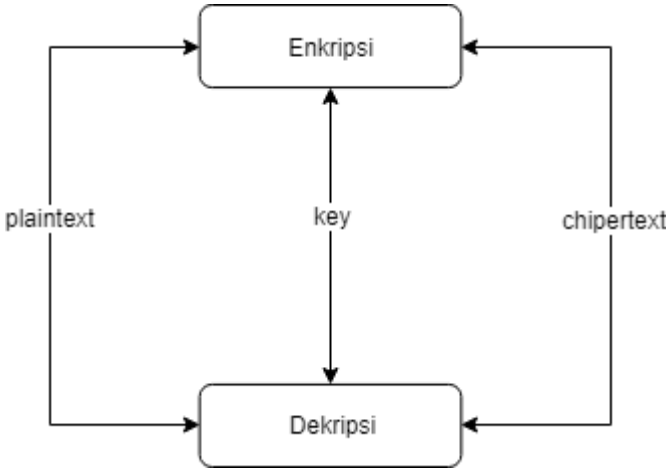
Sertifikasi Kompetensi merupakan jaminan tertulis sebuah produk, jasa, atau sebuah proses yang telah mampu memenuhi standar yang ditetapkan berdasarkan prosedur yang dilakukan untuk melakukan audit dari pihak ketiga. Selain itu, sertifikasi ini juga dapat diterapkan untuk standarisasi bagi profesional yang mempunyai kompetensi di bidang pekerjaan. Terdapat berbagai jenis sertifikasi yang bisa dimiliki oleh orang-orang yang membutuhkan. Salah satunya adalah standarisasi kompetensi yang berupa ukuran tentang keterampilan, sikap kerja, pengetahuan dari seseorang untuk melakukan pekerjaan sesuai dengan persyaratan yang ditentukan.

BBPLK Bandung adalah salah satu pusat pelatihan penyelenggaraan program-program pelatihan dan pengembangan kompetensi dan keterampilan sumber daya manusia di bidang industri jasa dan manufaktur. Bagi peserta yang telah menyelesaikan pelatihan, BBPLK Bandung memberikan sertifikat sebagai tanda telah memiliki kompetensi di bidang kejuruan yang diambil. Sertifikat tersebut merupakan bukti kompetensi yang digunakan seseorang untuk melamar kerja.

2.2.2 Kriptografi

Kriptografi adalah ilmu dan seni untuk menjaga kerahasiaan pesan dengan cara menyandikannya ke dalam bentuk yang tidak dapat dimengerti lagi maknanya. Dalam ilmu kriptografi, terdapat dua buah proses yaitu melakukan enkripsi dan dekripsi. Enkripsi adalah proses penyandian dari pesan asli menjadi pesan yang tidak dapat diartikan seperti pesan aslinya. Dekripsi sendiri berarti merubah pesan yang sudah disandikan menjadi pesan aslinya. Pesan asli biasanya disebut plaintext, sedangkan pesan yang sudah disandikan disebut *ciphertext* [10]. Adapun algoritma matematis yang digunakan pada proses enkripsi yakni disebut chiper dan sistem yang memanfaatkan Kriptografi untuk mengamankan sitem

informasi disebut kriptosistem. Gambar 2.2 berikut ini memperlihatkan aliran proses enkripsi dan deskripsi.



Gambar 2.2 Aliran Proses Enkripsi dan Dekripsi

Adapun tujuan dari sistem Kriptografi adalah sebagai berikut :

1. Kerahasiaan, adalah layanan yang digunakan untuk menjaga isi dari informasi dari siapapun kecuali yang memiliki otoritas atau kunci rahasia untuk membuka/mengupas informasi yang telah disandi.
2. Integritas data, adalah berhubungan dengan penjagaan dari perubahan data secara tidak sah. Untuk menjaga integritas data, sistem harus memiliki kemampuan untuk mendeteksi manipulasi data oleh pihak-pihak yang tidak berhak, antara lain penyisipan, penghapusan, dan pensubsitusian data lain kedalam data yang sebenarnya.
3. Autentikasi, adalah berhubungan dengan identifikasi/pengenalan, baik secara kesatuan sistem maupun informasi itu sendiri. Dua pihak yang saling berkomunikasi harus saling memperkenalkan diri. Informasi yang dikirimkan melalui kanal harus diautentikasi keaslian, isi datanya, waktu pengiriman, dan lain-lain.
4. Non-repudiasi, adalah usaha untuk mencegah terjadinya penyangkalan terhadap pengiriman atau terciptanya suatu informasi oleh yang mengirimkan atau membuat.

2.2.2.1 *Chipper*

Chiper merupakan teknologi untuk enkripsi dan dekripsi data. Teknik Kriptografi untuk enkripsi data ada 2 macam yaitu :

1. Kriptografi Simetri

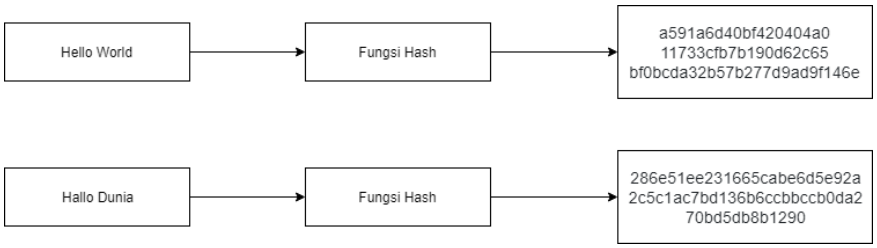
   Algoritma simetris atau disebut juga algoritma Kriptografi konvensional adalah algoritma yang menggunakan kunci yang sama untuk proses enkripsi dan proses dekripsi. Algoritma Kriptografi simetris dibagi menjadi 2 kategori yaitu algoritma aliran (*Stream Ciphers*) dan algoritma blok (*Block Ciphers*). Pada algoritma aliran, proses penyandiannya berorientasi pada satu bit atau satu *byte* data. Sedang pada algoritma blok, proses penyandiannya berorientasi pada sekumpulan bit atau *byte* data (per blok). Contoh algoritma kunci simetris adalah DES (Data *Encryption Standard*), *blowfish*, *twofish*, MARS, IDEA, 3DES (DES diaplikasikan 3 kali), AES (*Advanced Encryption Standard*) yang bernama asli Rijndael.

2. Kriptografi Asimetri

   Kriptografi asimetrik (*asymmetric cryptography*) adalah algoritma yang menggunakan kunci yang berbeda untuk proses enkripsi dan dekripsi. Kunci enkripsi dapat disebarkan kepada umum dan dinamakan sebagai kunci publik (*public key*) sedangkan kunci dekripsi disimpan untuk digunakan sendiri dan dinamakan sebagai kunci pribadi (*private key*). Oleh karena itulah, Kriptografi ini dikenal pula dengan nama Kriptografi kunci publik (*public key cryptography*). Contoh algoritma terkenal yang menggunakan kunci asimetris adalah RSA (Riverst Shamir Adleman) dan ECC (Elliptic Curve *Cryptography*). Pada kriptosistem asimetrik, setiap pelaku sistem informasi memiliki sepasang kunci, yaitu kunci publik dan kunci pribadi. Kunci publik didistribusikan kepada umum, sedangkan kunci pribadi disimpan untuk diri sendiri.

2.2.2.2 Fungsi *Hash* Satu Arah

Didalam sistem Kriptografi, selain ada algortima enkripsi/dekripsi yang mempunyai tujuan untuk kerahasiaan data, juga ada suatu fungsi yang mempunyai tujuan untuk menjaga integritas dan otentikasi data. Fungsi itu disebut dengan fungsi *hash*. Fungsi *hash* merupakan fungsi yang menerima masukan untuk diubah menjadi keluaran yang mempunyai panjang yang tetap (biasanya lebih pendek dari masukannya) [11]. Keluaran dari fungsi *hash* disebut dengan nilai *hash* atau *message digest*. Contoh pesan yang diubah menjadi *message diggest* dapat dilihat pada Gambar 2.3.

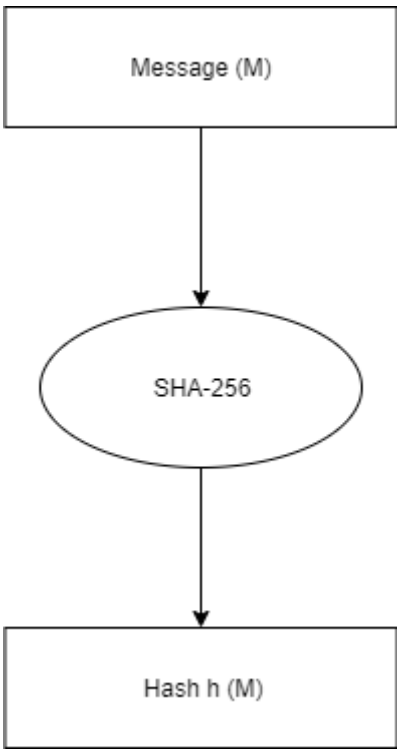


Gambar 2.3 Contoh Pesan yang Diubah Menjadi *Message Diggest*

Dari gambar 2.3 terdapat dua pesan yang berbeda. Kemudian diproses dengan fungsi *hash* yang menghasilkan *message digest* dengan panjang yang sama. Fungsi *hash* "satu arah" adalah jika pesan atau data yang sudah diubah menjadi *message digest* tidak dapat dikembalikan lagi menjadi menjadi pesan semula. Fungsi *hash* bersifat tidak dirahasiakan, dan keamanannya terletak pada sifat satu arahnya itu[11]. Contoh dari fungsi *hash* yang telah ada, antara lain : MD2, MD4, MD5, WHIRLPOOL, SHA (Secure *hash* Function), RIPMEND, dan lain-lain.

1. *Secure Hash Algorithm*-256

   Secure Hash Algorithm-256 adalah salah satu jenis *hash* yang masih umum digunakan. Fungsi ini adalah varian dari SHA-1, SHA-256 dibuat karena telah ditemukan bentrokan dari SHA-1, SHA-1 sendiri adalah pengganti dari SHA-0. Sampai saat ini belum ada yang dapat memecahkan algoritma untuk SHA256. SHA-256 umumnya digunakan sebagai fungsi antara untuk fungsi lain, termasuk fungsi *hash* MAC, HMAC, dan beberapa fungsi penghasil *digital signature* [12]. Fungsi utama SHA-256 dapat dilihat pada Gambar 2.4.



Gambar 2.4 Fungsi SHA-256

Fungsi utama SHA-256 menerima masukan berupa data atau pesan M dengan panjang sembarang, lalu akan menghasilkan nilai *hash* h(M) dengan panjang 256- bit. SHA-256 mempergunakan 6 fungsi logika, yang setiap fungsi tersebut beroperasi pada 32-bit words yang direperesentasikan sebagai x, y, and z. Hasil dari *Message* (M) SHA-256 *Hash* h(M) 20 setiap fungsi merupakan sebuah 32-bit word baru. Berikut fungsi logika dalam SHA-256:

$Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z)$

$Maj(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

= ROTR2

(x) ⊕ ROTR13(x) ⊕ ROTR22(x)

= ROTR6

(x) ⊕ ROTR11(x) ⊕ ROTR25(x)

= ROTR7

(x) ⊕ ROTR18(x) ⊕ SHR3

(x)

= ROTR17(x) ⊕ ROTR19(x) ⊕ SHR10(x)

Keterangan:

ROTR (Right-rotate)

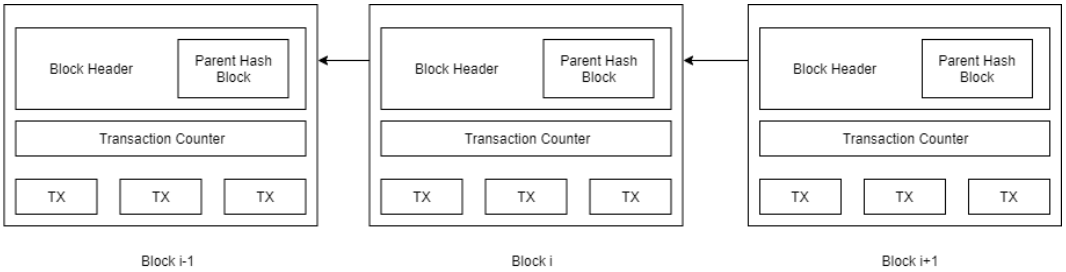
SHR (Right-shift)

2.2.3 *Blockchain*

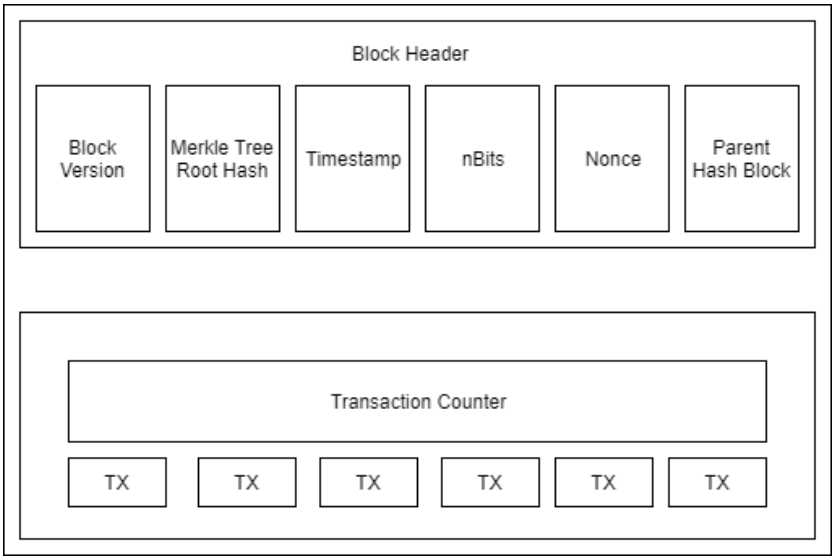
Blockchain awalnya diperkenalkan melalui Bitcoin oleh Satosi Nakamoto pada tahun 2008 dalam bidang *cryptocurrency*. Pada saat itu, *Blockchain* dirancang hanya untuk menghindari *double spending*. Namun, kini *Blockchain* telah banyak diimplementasikan dalam berbagai hal, diantaranya sertifikat digital, identitas digital, pemungutan suara digital, atau notaris terdesentralisasi [13].

2.2.3.1 Definisi *Blockchcain*

Dalam istilah sederhana, *Blockchain* dapat digambarkan sebagai basis data yang terdesentralisasi, tanpa adanya kepercayaan antar peserta. Aset digital (seperti unit kredit, obligasi, kepemilikan, atau hak fundamental) dikelola sebagai daftar blok berisi transaksi yang terurut. Setiap blok dalam *Blockchain* akan terhubung dengan blok sebelumnya melalui *hash*. Dengan demikian, riwayat transaksi dalam *Blockchain* tidak dapat diubah atau dihapus tanpa mengubah keseluruhan isi dari *Blockchain* [14]. Hal tersebut membuat *Blockchain* aman dari serangan peretas. Perbedaan mendasar dengan data*base* saat ini adalah penghilangan elemen pusat akibatnya data didistribusikan dan didesentralisasi. Ini berarti bahwa tidak ada unit kontrol pusat yang dapat memeriksa keakuratan informasi. Oleh karena itu, *Blockchain* menggunakan mekanisme konsensus. Ini memungkinkan informasi yang dikirimkan untuk diintegrasikan ke dalam *Blockchain* hanya setelah persetujuan (konsensus). Jika persyaratan yang relevan terpenuhi, transaksi yang dikonfirmasi dengan konsensus dapat dilacak dan diamankan dari manipulasi atau pemalsuan oleh pihak ketiga.



Gambar 2.5 Struktur *Blockchain*



Gambar 2.6 Struktur *Block* dalam *Blockchain*

2.2.3.2 Karakteristik *Blockchain*

Teknologi *Blockchain* memiliki beberapa karakteristik yang menjadi keunggulan dari teknologi ini. Berikut ini beberapa karakteristik *Blockchain* [15]:

a. *Immutable* adalah sifat dimana sebuah data atau variabel tidak dapat diubah lagi setelah nilainya diberikan (*assigned*) dalam eksekusi program. Dalam lingkungan eksekusi program, *immutable* dapat dicapai dengan mudah melalui serangkaian aturan logika, sedangkan dalam konteks data yang tersimpan dalam media penyimpanan kondisi tersebut lebih sulit dicapai. Salah satu cara untuk membuat sebuah *file* atau data *immutable* adalah dengan menuliskannya pada media sekali tulis seperti pada CD atau DVD, cara lainnya adalah menggunakan metode kriptografi yang menghasilkan *immutability* semu, semu karena secara fisik data tetap dapat diubah namun

secara logik dapat diperiksa dengan menggunakan perhitungan matematis. Teknologi *blockchain* menggunakan metode kriptografi untuk mengamankan data transaksi dari adanya perubahan tanpa izin.

b. *Decentralization* : tidak dibutuhkan pihak ketiga dalam sebuah transaksi. Algoritma konsensus digunakan untuk menjaga konsistensi data dalam jaringan terdistribusi.
c. *Distributed* data*base* : setiap pihak pada *Blockchain* memiliki akses ke seluruh data*base* dan riwayatnya yang lengkap. Tidak ada satu pihak pun yang mengendalikan data atau informasi tersebut. Setiap pihak dapat memverifikasi catatan mitra transaksinya secara langsung, tanpa perantara.
d. *Persistency* : proses validasi transaksi berlangsung cepat dan transaksi yang tidak valid tidak akan diakui oleh *miners*. Pada *Blockchain*, tidak mungkin menghapus transaksi yang telah terjadi.
e. *Anonymity* : setiap pengguna dalam jaringan *Blockchain* dapat berinteraksi satu sama lain menggunakan suatu alamat tertentu. Dalam hal ini, identitas sebenarnya dari setiap pengguna tidak ditampilkan pada interaksi tersebut.
f. *Auditability* : setiap transaksi dalam suatu jaringan *Blockchain* merujuk pada transaksi sebelumnya. Hal ini akan mempermudah dalam proses verifikasi dan pencarian transaksi.

Berikut ini, merupakan gambaran dan perbandingan karakteristik dari *distributed ledger* dengan sistem sentralisasi [16].

b. Verifikasi Publik memungkinkan siapa pun untuk memverifikasi kebenaran kondisi sistem. Dalam *distributed ledger*, setiap pergantian kondisi akan dikonfirmasi oleh *verifier* (misalnya penambang dalam Bitcoin), yang dapat berupa kumpulan peserta terbatas. Namun, setiap pengamat dapat memverifikasi bahwa keadaan *distributed ledger* diubah sesuai dengan protokol dan semua pengamat pada akhirnya akan memiliki pandangan yang sama tentang *distributed ledger*, setidaknya hingga panjang tertentu. Dalam sistem terpusat, pengamat yang berbeda mungkin memiliki pandangan yang sama sekali berbeda mengenai suatu kondisi. Dengan demikian, mereka mungkin tidak dapat memverifikasi bahwa semua transisi kondisi dijalankan

dengan benar. Sebagai gantinya, pengamat perlu mempercayai entitas pusat untuk memberi mereka kondisi yang benar.

c. Transparansi data dan proses pembaharuan kondisi adalah persyaratan untuk verifikasi publik. Namun, jumlah informasi yang transparan bagi pengamat dapat berbeda, dan tidak setiap peserta perlu memiliki akses ke setiap informasi.
d. Privasi adalah properti penting dari sistem apa pun. Ada ketidaksinambungan antara privasi dan transparansi. Privasi tentu lebih mudah dicapai dalam sistem terpusat karena transparansi dan verifikasi publik tidak diperlukan untuk berfungsinya sistem.
e. Integritas informasi menunjukkan bahwa informasi dilindungi dari modifikasi yang tidak sah. Integritas informasi sangat berkaitan dengan verifikasi publik. Jika suatu sistem mendukung verifikasi publik, siapa pun dapat memverifikasi integritas data.
f. Redundansi data penting dalam beberapa hal. Dalam sistem *Blockchain*, redundansi secara inheren disediakan melalui replikasi di seluruh penulis. Dalam sistem terpusat, redundansi umumnya dicapai melalui replikasi pada server fisik yang berbeda dan melalui cadangan.
g. *Trust Anchor* mendefinisikan siapa yang mewakili otoritas tertinggi dari sistem yang diberikan yang memiliki wewenang untuk memberikan dan mencabut akses baca dan tulis ke suatu sistem.

2.2.3.3 Jenis-jenis *Blockchain*

Teknologi *Blockchain* memiliki beberapa jenis untuk diimplementasikan dalam kondisi yang diharapkan. Terdapat tiga jenis *Blockchain* yaitu:

a. *Public Blockchain* : *Blockchain* publik adalah *Blockchain* yang dapat diakses dan digunakan oleh siapa pun. *Blockchain* publik tidak dikontrol oleh individu atau organisasi mana pun. Buku besar pada *Blockchain* ini terbuka dan transparan. Namun, terdapat kekurangan pada *Blockchain* publik, yaitu biaya operasi dan pemeliharaan yang tinggi, serta kecepatan transaksinya yang lambat. Contoh penggunaannya adalah pada Bitcoin, Ethereum, dan Hyperledger.

b. *Private Blockchain* atau *Permissioned Blockchain* : *Blockchain* privat dibentuk untuk memfasilitasi pertukaran data secara pribadi di antara sekelompok individu atau organisasi. Pengguna yang tidak dikenal tidak dapat mengakses jaringan *Blockchain* ini tanpa undangan khusus. Contoh penggunaannya adalah pada R3 Corda.
c. *Consortium Blockchain* : *Blockchain* konsorsium merupakan gabungan dari *Blockchain public* dan *private*, di mana tidak ada organisasi tunggal yang bertanggungjawab yang dapat mengontrol jaringan melainkan beberapa *node* yang telah ditentukan sebelumnya. *Node* ini dapat memutuskan siapa saja yang dapat menjadi bagian dari jaringan dan siapa saja yang dapat menjadi penambang. Untuk validasi blok digunakan skema *multisignature*, di mana blok dianggap valid hanya jika ditandatangani oleh beberapa *node* tersebut. Contoh penggunaannya adalah pada Fabric.

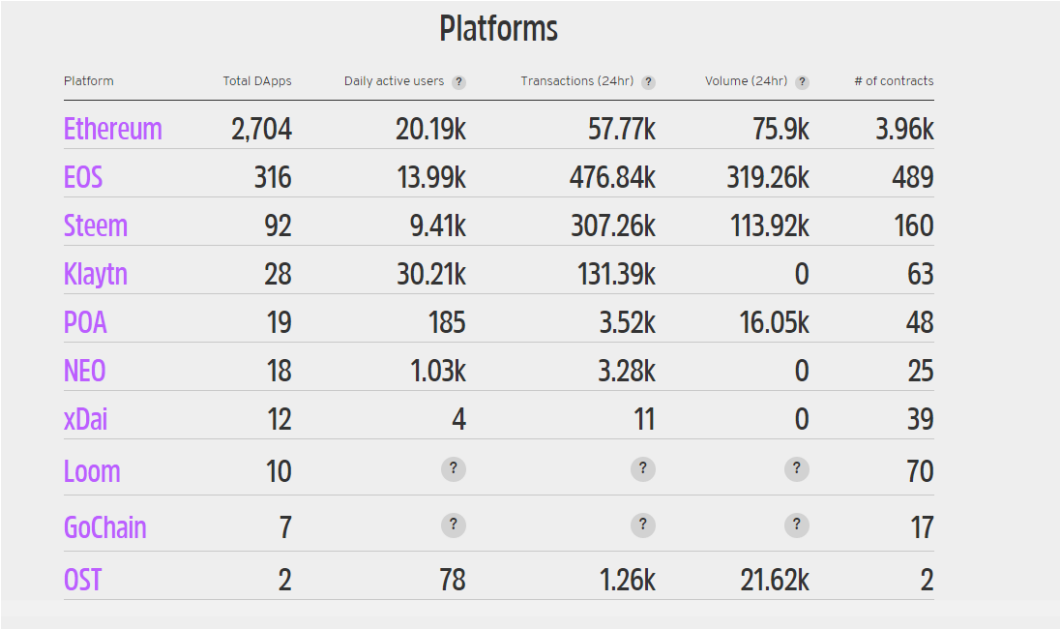
2.2.4 *Decentralized Application* (DApps)

Decentralized application (DApps) adalah aplikasi yang berkomunikasi dengan memanfaatkan teknologi *Blockchain*. Antarmuka Dapps tidak berbeda dengan aplikasi seperti tradisional seperti biasanya. Prosedur perbedaan aplikasi tradisional yang ada saat ini dengan Dapps terdapat pada server yang terpusat dimana server bertindak sebagai otoritas pengontrol seluruh aplikasi. Sedangkan pada Dapps didasarkan pada jaringan yang tersebar (Desentralisasi). Perbedaan lainnya juga terdapat pada bagian *backend* dimana *smart contract* menjadi logika dari aplikasi yang bersifat *decentralized* (DApps) [17]. Karakteristik *Decentralized Application* (DApps) adalah sebagai berikut [18].

1. *Open source* : Karena sifat kepercayaan yang ada pada *Blockchain*, Dapps perlu membuat kode *open source* sehingga memungkinkan audit dari pihak ketiga.
2. *Internal Cryptocurrency Support* : Mata uang internal merupakan ekosistem yang menjalankan Dapps. Dengan mata uang internal seperti token, semua kredit dan transaksi di antara peseta sistem termasuk penyedia konten dan konsumen dapat diukur.

3. *Decentralized Consensus* : Konsensus diantara *node* yang terdesentralisasi merupakan pondasi yang transparan.
4. *No central point of failure* : Sistem yang sepenuhnya terdesentralisasi seharusnya tidak memiliki titik kelemahan karena semua komponen aplikasi akan di *host* dan di eksekusi di *Blockchain*.

   Berikut merupakan beberapa *platform blockchain* yang banyak digunakan untuk membuat *Decentralized Application* (Dapps) berdasarkan situs https://www.stateofthedapps.com/stats.



Platforms

Platform	Total DApps	Daily active users	Transactions (24hr)	Volume (24hr)	# of contracts
Ethereum	2,704	20.19k	57.77k	75.9k	3.96k
EOS	316	13.99k	476.84k	319.26k	489
Steem	92	9.41k	307.26k	113.92k	160
Klaytn	28	30.21k	131.39k	0	63
POA	19	185	3.52k	16.05k	48
NEO	18	1.03k	3.28k	0	25
xDai	12	4	11	0	39
Loom	10	?	?	?	70
GoChain	7	?	?	?	17
OST	2	78	1.26k	21.62k	2

Gambar 2.7 Statistik *Platform Blockchain* Pada Dapps

2.2.5 Ethereum

Ethereum diluncurkan pada tahun 2015 dengan *Blockchain* yang dapat diprogram. Dengan kemampuan ini *Blockchain* dapat digunakan untuk membuat aplikasi desentralisasi sesuai dengan kebutuhan pengguna. Karakteristik *Blockchain* Ethereum adalah *statefull* dan *Turing Completeness* [6]. Sehingga dimungkinkan untuk menyimpan data selain transaksi dan membuat program yang kompleks. *Blockchain* pada Ethereum merupakan *public ledger* yang digunakan untuk mencatatkan transaksi yang pernah terjadi. Sehingga, keseluruhan data transaksi dari awal diluncurkannya Ethereum tercatat di dalam

Blockchain. Dikarenakan *ledger* dapat diakses secara *public* maka Ethereum menghubungkan setiap blok *ledger* tersebut menggunakan algoritma *hash*. Dengan mekanisme tersebut perubahan satu data transaksi mengharuskan perubahan *hash* pada blok-blok berikutnya.

Ethereum, secara keseluruhan, dapat dipandang sebagai mesin status berbasis transaksi. kita mulai dengan kondisi *genesis* dan secara bertahap melakukan transaksi untuk mengubahnya menjadi beberapa kondisi akhir. Keadaan akhir inilah yang kita terima sebagai "versi" kanonik dari dunia Ethereum. Negara dapat memasukkan informasi seperti saldo akun, reputasi, pengaturan kepercayaan, data yang berkaitan dengan informasi dunia fisik; singkatnya, apa pun yang saat ini dapat diwakili oleh komputer dapat diterima. Transaksi dengan demikian mewakili busur yang valid antara dua negara; bagian 'valid' adalah penting ada perubahan negara yang jauh lebih tidak valid daripada perubahan negara yang valid. Perubahan status yang tidak valid mungkin, misalnya menjadi hal-hal seperti mengurangi saldo akun tanpa peningkatan yang sama dan berlawanan di tempat lain. Transisi keadaan yang valid adalah transisi yang terjadi melalui transaksi. Secara formal:

Blockchain Ethereum merupakan *statefull Blockchain* dimana *asset* yang disimpan dalam blok bukan hanya transaksi. Melainkan terdapat tiga *Merkle Patricia Tree* (modifikasi dari *Merkle Tree* pada Bitcoin) untuk transaksi, *account*, dan *receipt*. Hal ini menyebabkan terdapat tiga *root* yang disimpan pada *header* sebuah blok. Selain itu, data pada sebuah blok Ethereum lebih banyak dibandingkan Bitcoin. Hal ini, tidak lain untuk mendukung proses pembuatan aplikasi terdesentralisasi menggunakan *Blockchain* Ethereum.

2.2.5.1 *Smart Contract*

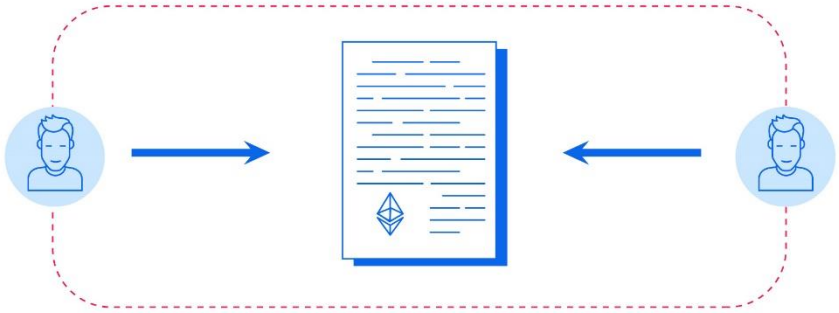
Smart contract merupakan satu set perjanjian yang diubah kedalam bentuk digital sehingga pelanggaran terhadap perjanjian tersebut sulit untuk dilakukan. *Smart contract* Ethereum merupakan protokol komputer yang berfungsi untuk memfasilitasi, memverifikasi, atau menegakan negoisasi secara digital yang ditulis melalui kode program. *Smart contract* bekerja tanpa melalui pihak ketiga

dan memiliki proses transaksi yang kredibel sehingga tidak bisa di *hack* ataupun diubah [19].

Smart contract disimpan beserta transaksinya pada *Blockchain*. *Blockchain* menggunakan teknologi jaringan *peer-ro-peer* yang terdistribusi, sehingga *Blockchain* adalah tempat penyimpanan yang paling aman untuk data digital seperti *cryptocurrency, smart contract* , properti, saham, berkas, ataupun yang berharga dengan adanya kerahasiaan, integritas dan keaslian data. *Blockchain* terdiri dari beberapa daftar blok yang terus bertambah dan dihubungkan oleh algoritma kriptografi. Hal ini didasarkan pada *Distributed ledger Technology* (DLT) yang merupakan sistem untuk mencatat transaksi digital dalam penyimpanan terdistribusi tanpa memiliki penyimpanan secara terpusat [19].

Dengan mengunakan *smart contracts,* kita dapat melakukan pertukaran uang, properti, saham atau apapun secara transparan, tanpa konflik dan tanpa perantara. *Smart contracts* dapat memberikan keamanan yang lebih unggul dari hukum kontrak tradisional. Kemudian mengurangi biaya transaksi lainnya yang terkait dengan kontrak. Berbagai mata uang kripto telah menerapkan jenis *smart contract* ini [20].

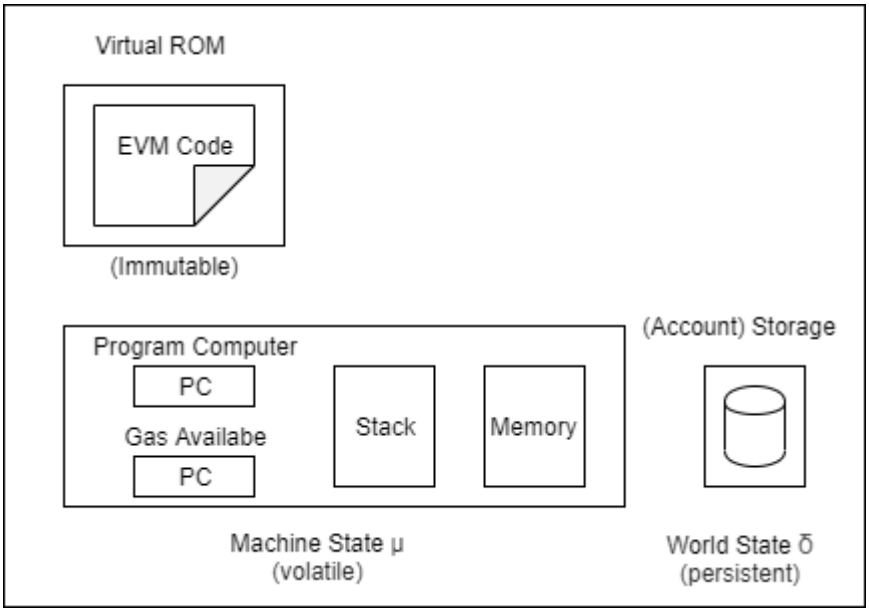
Untuk mengetahui gambaran *smart contract* adalah membandingkannya dengan sebuah *vending machine*. Jika biasanya menemui seorang pengacara atau notaris, kemudian membayarnya, dan menunggu untuk mendapatkan dokumen yang dibutuhkan. Dengan *smart contract*s, cukup memasukkan bitcoin ke dalam *vending machine* (dalam hal ini *ledger*) kemudian dokumen-dokumen dapat langsung masuk ke akun kita. Terlebih lagi, *smart contract* tidak hanya menerangkan perihal peraturan dan penalti seperti yang ada dalam kontrak tradisional. Tetapi juga secara otomatis memastikan hal-hal yang ada dalam kontrak tersebut ditegakkan [20].



Gambar 2.8 *Smart Contract* Tanpa Pihak Ketiga

2.2.5.2 *Ethereum Virtual Machine* (EVM)

Ethereum merupakan sebuah *Blockchain* yang memberikan kesempatan untuk siapapun membuat aplikasi terdesentralisasi. Hal ini terjadi karena Ethereum memiliki sebuah *account* yang disebut dengan istilah *contract account* yang memiliki kode algoritma didalamnya. Pengeksekusian kode tersebut dilakukan pada sebuah *compiler machine* yang disebut dengan *Ethereum Virtual Machine* (EVM). Kode tersebut dieksekusi di *Blockchain* dalam format *Ethereum-specific binary* (EVM *bytecode*). Namun penulisan kode tersebut dilakukan menggunakan bahasa *high level*, yang di *compile* oleh EVM *compiler*, dan diunggah kedalam *Blockchain* menggunakan Ethereum *client* sehingga kode tersebut bisa digunakan oleh *node* lain [6]. Arsitektur Ethereum *Virtual Machine* (EVM) dapat dilihat pada Gambar 2.9.



Gambar 2.9 Arsitektur *Ethereum Virtual Machine* (EVM)

2.2.5.3 *Byte Code*

Smart contract biasanya ditulis dalam bahasa pemrograman tingkat tinggi seperti Solidity. Kode ini dikompilasi ke sebuah *bytecode* EVM yang akan digunakan untuk *Blockchain* Ethereum. Kompilasi ini sangat mirip dengan bahasa pemrograman seperti Java di mana kode akan dikonversi ke *bytecode* JVM. *Runtime Environtment Ethereum* hanya memahami dan dapat menjalankan *bytecode* [21].

2.2.5.4 Ether dan Denominasi

Ether (ETH) adalah token yang mendasari *Blockchain* Ethereum, yang digunakan sebagai bahan bakar atau *gas* untuk menggerakan jaringan Etherum. Di dunia nyata, mata uang memiliki berbagai denominasi. Misalnya, Dolar AS sama dengan 100 sen dan memiliki berbagai denominasi seperti uang (1 sen), nikel (5 sen), uang receh (10 sen), seperempat (25 sen). Ether juga memiliki berbagai denominasi yaitu dalam satuan Ether dan Wei. Wei adalah denominasi terendah dan merupakan denominasi yang digunakan dalam *smart contract* [22]. Denominasi ether dapat dilihat pada Gambar 2.10.

Value (in wei)	Exponent	Common Name	SI Name
1	1	wei	wei
1,000	10^3	babbage	kilowei or femtoether
1,000,000	10^6	lovelace	megawei or picoether
1,000,000,000	10^9	shannon	gigawei or nanoether
1,000,000,000,000	10^{12}	szabo	microether or micro
1,000,000,000,000,000	10^{15}	finney	milliether or milli
1,000,000,000,000,000,000	*10^{18}*	*ether*	*ether*
1,000,000,000,000,000,000,000	10^{21}	grand	kiloether
1,000,000,000,000,000,000,000,000	10^{24}		megaether

Gambar 2.10 Denominasi *Ether*

2.2.5.5 *Address*

Address merupakan identitas unik dalam Ethereum *Blockchain*. *Address* memerlukan pasangan *private key* untuk berinteraksi dengan *Blockchain*. Ethereum *Address* bersifat *public* karena dapat dibagikan dengan siapapun. Sedangkan *private key* tidak boleh dibagikan dengan siapapun dan tidak disimpan di data*base* manapun. Ethereum *Address* merupakan turunan dari *public key* atau kontrak menggunakan Keccak-256 *hash function*. Berikut merupakan contoh untuk menghasilkan Ethereum *Address* [21].

5. *Generate Private Key*

   Private key merupakan 64 karakter dari heksadesimal. Setiap *string* 64 hex, secara hipotetis, adalah kunci *private* Ethereum yang akan mengakses akun.

   k(*private key)* = f8f8a2f43c8376ccb0871305060d7b27b0554d2cc72bccf4 b2705608452f3

6. Menurunkan *Public key* ke *Private Key*

   K(*public key*) = 6e145ccef1033dea239875dd00dfb4fee6e3348b84985c92
   f103444683bae0

7. Mengkalkulasikan *Hash* dari *Public key* menggunakan Algoritma *Keccak256*

   Keccak256(K) = 2a5bc342ed616b5ba5732269001d3f1ef827552ae111402
   7bd3ecf1f086ba0

8. 20 *byte* dari *hash* yang dihasilkan merupakan Ethereum *Address* .

   Address = 0x001d3f1ef827552ae1114027bd3ecf1f086ba0f9

2.2.5.6 *Account*

Salah satu data yang tersimpan dalam blok Ethereum adalah *state*. *State* tersebut merupakan data dari akun-akun yang ada terdapat di dalam jaringan Ethereum. Terdapat dua jenis akun di dalam jaringan Ethereum , yaitu *extenally owned account (EOA)* dan *contract account*. Adapun penjelasannya adalah sebagai berikut.

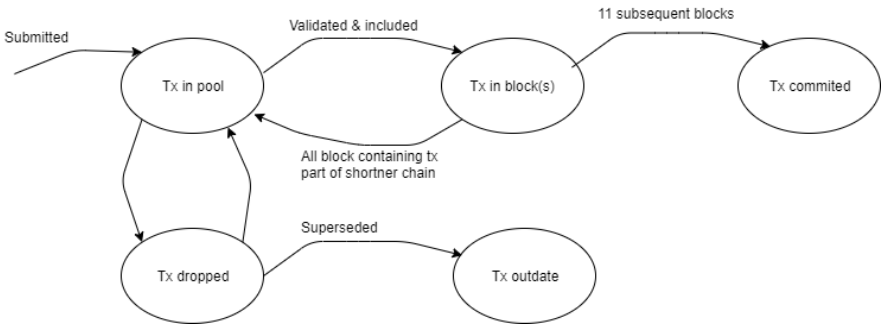
1. *Externally Owned Account* (EOA), merupakan sebuah akun yang dapat dikendalikan oleh pihak luar (manusia) menggunakan *private key*.
2. *Contract Account*, merupakan program-program yang dibuat akan disimpan di dalam *blokchcain* Ethereum dalam bentuk *contract account*.

2.2.5.7 *Transaksi*

Istilah transaksi dalam Ethereum merupakan definisi untuk sebuah data yang telah ditandatangani yang menyimpan pesan pengiriman dari satu EOA ke akun lainnya di dalam *Blockchain*. Proses transaksi dapat memicu dieksekusinya sebuah kode pada *contract account* dan juga menyebabkan perubahan saldo pada sebuah akun. Transaksi pada Ethereum berbeda dengan transaksi pada umumnya. Jika kita ingin membeli barang menggunakan uang koin, sekalipun harganya lebih rendah dari nilai koin yang dimiliki, yang harus kita lakukan adalah memberikan seluruh koin tersebut dan mendapatkan kembaliannya. Adapun struktur transaksi pada Ethereum adalah sebagai berikut [4].

1. *Nonce* merupakan nilai yang setara dengan jumlah transaksi yang dikirim dari akun pengirim. Sehingga, *nonce* pada transaksi akan disesuaikan dengan *nonce* pada *account*. Dengan cara ini *double spending* dapat dihindari.
2. *Signature* merupakan bukti atau tanda tangan kriptografi dari pengirim yang dibuat menggunakan algoritma ECDSA dengan *input private key*.
3. *Value* jumlah ether yang dikonversi ke dalam satuan terkecil, yaitu wei (1 ether = 10^{18} Wei).
4. Data merupakan data opsional yang bisa dimasukkan kedalam transaksi berguna untuk dikirimkan ke *contract account* untuk mengeksekusi sebuah kode.
5. *Gas limit* nilai maksimum dari langkah komputasi yang bisa dilakukan pada sebuah transaksi. Jumlah ini ditentukan oleh pengirim, ketika *miner* melakukan langkah komputasi lebih dari yang ditentukan maka transaksi akan gagal.
6. *Gas price*, harga yang dikenakan untuk setiap langkah komputasi yang dilakukan. Harga tersebut dibuat dalam satuan gwei (1 gwei = = 10& Wei). Sebagai contoh, langkah komputasi dari sebuah transaksi setara dengan 50 *gas*, maka akan dikalikan dengan 3 gwei. Hasil tersebut akan menjadi biaya transaksi yang harus dibayarkan pengirim kepada *miner*.
7. *To* alamat akun dari penerima transaksi.

   Alur transaksi pada Ethereum dapat dilihat pada Gambar 2.11.



Gambar 2.11 Alur Transaksi Ethereum

2.2.5.8 *Gas*

Proses transaksi dalam Ethereum memungkinkan melibatkan sebuah proses pengeksekusian kode yang ada pada *contract account*. Proses pengeksekusian kode tentunya akan menggunakan sumber daya, terkhusus dari mereka yang berperan sebagai *miner*. Sebagai ganti sumber daya yang telah digunakan Ethereum memberikan perhitungan untuk setiap langkah komputasi yang terjadi dari sebuah transaksi dalam bentuk *gas*. Sehingga nantinya total *gas* yang digunakan dikonversi kedalam satuan wei dan akan dikirimkan ke akun miner sebagai hadiah dari keberhasilan membuat suatu blok. Adapun beberapa istilah *gas* pada adalah sebagai berikut.

1. *Gas cost* merupakan konstanta bernilai statis yang menunjukkan harga *gas* untuk langkah komputasi.
2. *Gas usage* merupakan total langkah komputasi dalam bentuk gas pada sebuah transaksi.
3. *Gas price* merupakan nilai dalam satuan gwei (1 gwei = 10& Wei), yang bisa digunakan untuk mengkonversi total keseluruhan *gas* pada sebuah transaksi ke satuan ether.
4. *Gas limit* adalah jumlah maksimal gas yang bisa digunakan pada setiap blok. Hal ini akan mempengaruhi jumlah transaksi yang ada dalam sebuah blok dan juga ukuran blok itu sendiri. Total *gas limit* dapat berubah sesuai dengan konsumsi *gas* yang dilakukan *miner* setiap pembuatan blok.
5. *Gas fee* merupakan jumlah *gas* yang perlu dibayarkan oleh pengirim transaksi pada setiap transaksinya. Perhitungannya adalah *gas usage* x *gas pric*e.

2.2.5.9 *Consesus*

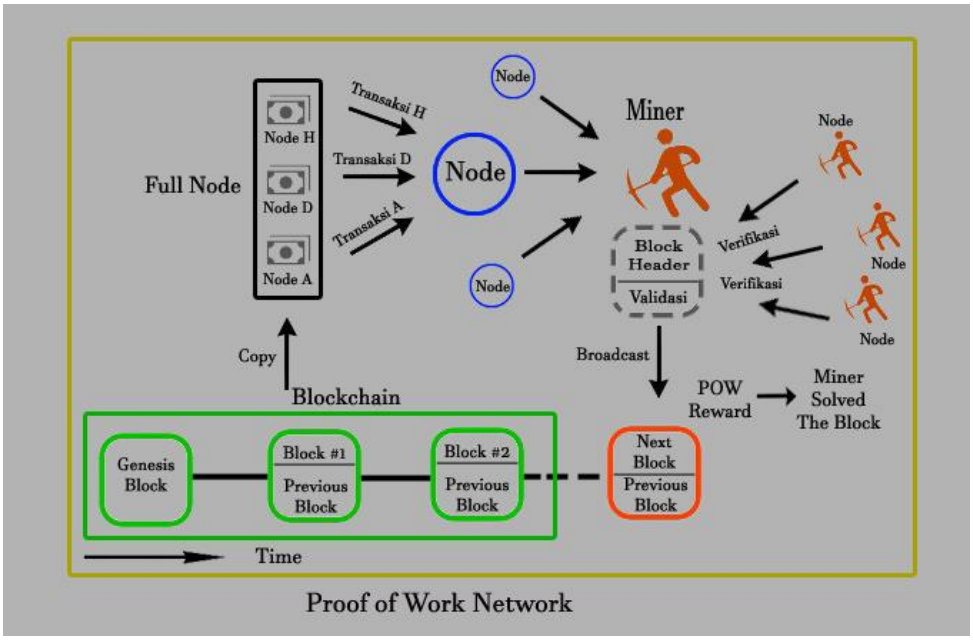
Konsesus merupakan kemampuan untuk mencapai kesepakatan bersama. Konsensus dalam *Blockchain* adalah suatu proses perhitungan rumit untuk menghasilkan kesepakatan bersama tentang validasi suatu transaksi. Dengan kata lain konsensus dimaksudkan untuk menghasilkan sistem yang ketat tanpa aturan penguasa. Tidak ada satu orang, organisasi atau kelompok yang bertanggungjawab atau lebih tepatnya kekuatan dan kontrol tersebar di seluruh

jaringan peserta. Kemampuan untuk mencapai konsensus di seluruh jaringan terdistribusi dibawah kondisi persaingan dan tanpa kontrol yang bersifat sentral merupakan prinsip inti dari *public blockchain* [21]. Berikut merupakan beberapa algoritma konsensus yang digunakan untuk memvalidasi transaksi pada *blockchain*.

1. *Proof of Work*

   Proof of Work merupakan sebuah protokol yang mempunyai fungsi untuk mencegah aktifitas serangan *cyber* (DDos), yang dapat melumpuhkan/melemahkan suatu sumber daya sistem komputer. Konsep POW pertama kali dikenalkan oleh Cynthia Dwork & Moni Naor pada tahun 1993 dan baru di implementasikan oleh Markus Jakobsson (mata uang Shell) pada tahun 2009. Dalam teknologi *Blockchain*, POW digunakan oleh Satoshi Nakamoto sebagai algoritma konsensus dan Bitcoin sendiri sebagai Mata Uang dari konsensus *Proof of Work*. Persyaratan utama dalam konsensus POW adalah proses kegiatan mining (proses komputasi dari CPU, GPU, ASIC, FPGA) yang berfungsi sebagai penemu, pencari solusi dan memvalidasi setiap masalah (*hash*) kedalam sebuah *block* dan akan didistribusikan ke dalam sebuah buku besar (*ledger*) yang disebut dengan *Blockchain*.

   Untuk mencapai sebuah konsensus, sebuah transaksi harus melewati beberapa proses yang juga melibatkan adanya proses komputasi yang dilakukan oleh beberapa miners, sehingga bisa tercipta sebuah *Block* yang valid. Sistem distribusi konsensus *proof of work* dapat dilihat pada pada Gambar 2.12.



Gambar 2.12 *Proof of Work Network*

Berikut merupakan penjelasan dari *Proof of Work Network*:

a) Suatu (beberapa) transaksi yang muncul dari sebuah *wallet* yang bertindak sebagai *Full Node* (salinan *Blockchain*) akan di *publish* pada jaringan (P2P).

b) Transaksi-transaksi ini akan tehubung ke sebuah jaringan yg juga terhubung dengan node *Miners*.

c) Miner akan melakukan proses komputasi (*hash function*) untuk menyelesaikan persoalan matematika rumit ini ke dalam sebuah *block*.

d) Jumlah maksimal transaksi dalam setiap *block* tergantung dari protokol yang berlaku.

e) Setelah masalah (*hash*) terpecahkan, maka miner yang pertama kali memecahkan masalah ini akan mem-broadcast *block* baru ke jaringan (P2P).

f) *Node* (*miner*) lainnya yang menerima *block* ini akan melakukan proses verifikasi (validasi).

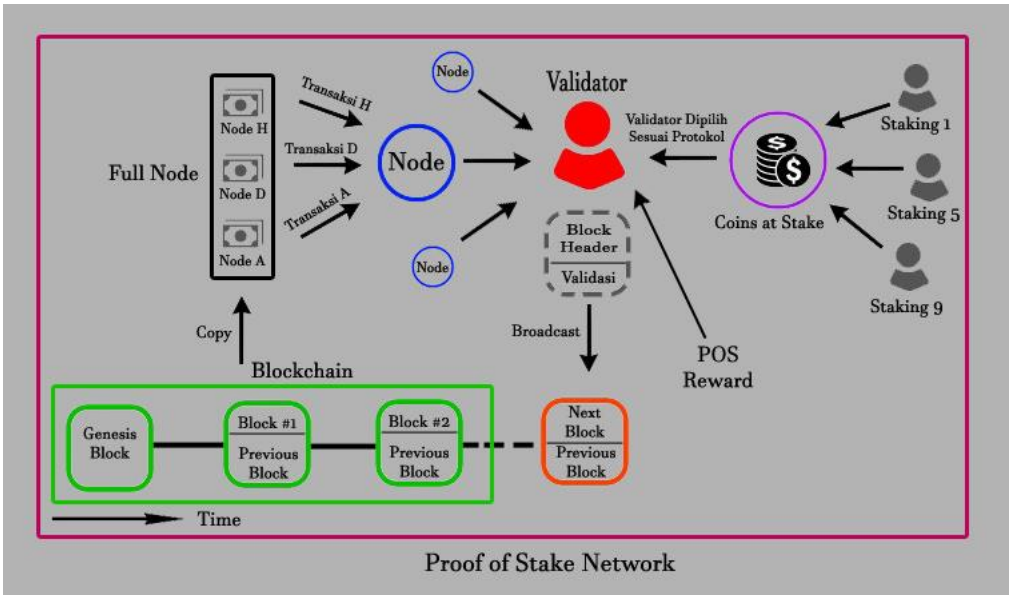
g) Setelah *block* mendapat validasi, maka *block* ini akan di distribusikan ke dalam *blockchain* sebagai *Block* baru yang valid.

h) *Miners* yang bertindak sebagai pembuat *block* valid akan menerima reward.

2. *Proof Of Stake*

Algoritma konsensus Proof of *Stake* (PoS) pertama kali dikenalkan oleh Sunny King dan Scott Nadal pada tahun 2012. Penggunaan metode ini diharapkan dapat mengatasi besarnya daya yang dibutuhkan untuk melakukan proses komputasi pada konsensus PoW. Secara garis besar, proses Staking ialah mengunci beberapa jumlah coin (*cryptocurrency*) sebagai jaminan, didalam sebuah *wallet* yang menjalankan *Full Node* dari sebuah *blockchain*. Jaminan ini sebagai bukti kepemilikan seorang pemegang koin (tokens holder) sebagai fungsi node, dalam berkontribusi didalam jaringan konsensus PoS dengan proses yang dinamakan *Forging* (menempa koin).

Berbeda dengan konsensus PoW yang membutuhkan proses komputasi menggunakan peralatan-peralatan dengan sumber daya *list*rik. Pada metode PoS, untuk mencapai sebuah konsensus hanya dibutuhkan sebuah *Full Node* (*Wallet*) dengan token/koin didalamnya. Dan setiap token/koin holder yang senantiasa terhubung dengan jaringan (*node*) *blockchain*, mempunyai peluang untuk menjadi seorang *staker/forger* dengan memenuhi syarat sesuai dengan protokol yang berlaku. Oleh karena itu didalam konsensus PoS ini, tidak dibutuhkan banyak sumber daya energi (*list*rik) dalam proses pencapaian suatu konsensus. Sistem distribusi konsensus *proof of work* dapat dilihat pada pada Gambar 2.13.



Gambar 2.13 *Proof of Stake Network*

Berikut merupakan penjelasan dari *Proof of Stake Network*:

a) Untuk bisa melakukan *forging* (*minting*), maka pemegang token/koin harus mengunci sejumlah koin didalam *wallet*nya (*Full Node*) yg terhubung dengan jaringan *blockchain*.

b) Pada saat sebuah/beberapa transaksi masuk ke dalam jaringan (P2P). Pembuatan sebuah *block* (validasi transaksi) dipilih secara *Pseudorandom* berdasarkan jumlah koin yang di *stake* dan berapa lama koin tersebut sudah di *stake*.

c) *Token/Coin Holde*r dengan jumlah *Staking Coin* yang besar dan waktu (umur) *staking* yang lama, mempunyai peluang (kesempatan) lebih tinggi untuk melakukan proses *forging* pada *block* berikutnya.

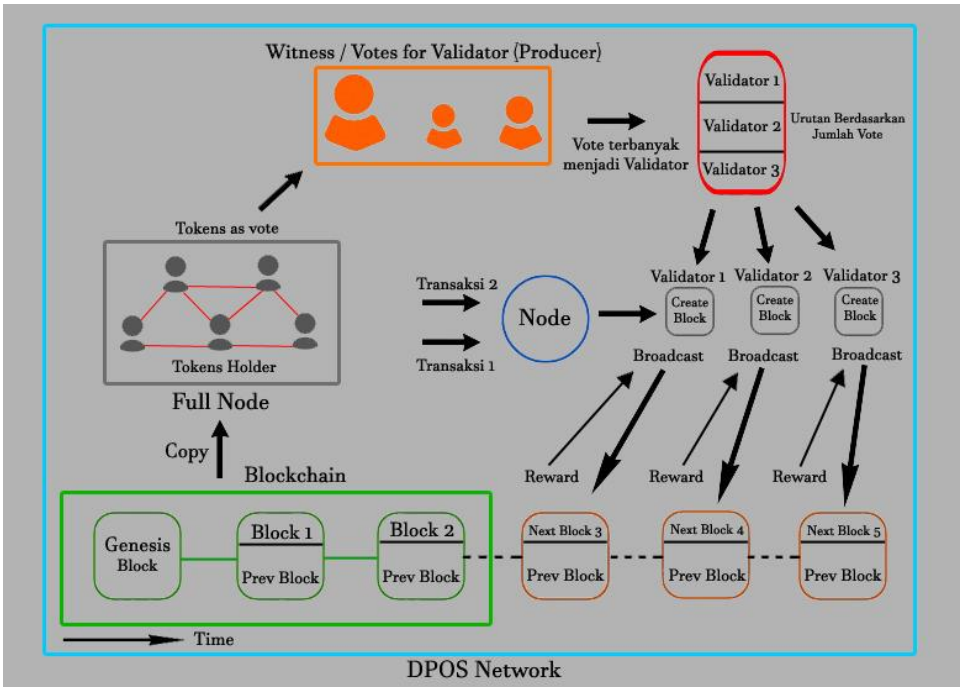
d) Setelah proses pembuatan *block* dan proses validasi selesai, maka *block* ini akan didistribusikan ke dalam jaringan *blockchain* sebagai *block* baru yang valid.

e) *Forger* (*staker*) akan menerima *reward* dari hasil kerjanya dan untuk umur *staking coinnya* akan direset ulang.

3. *Delegated Proof of Stake*

Delegated Proof of Stake (DPoS) diciptakan oleh Daniel Larimer pada saat membuat sebuah *cryptocurrency* yang disebut dengan BitShares pada tahun 2014. Algoritma konsensus ini adalah sebuah metode baru sebagai tindakan pengamanan terhadap sebuah jaringan *cryptocurrency* untuk mencapai sebuah konsensus tanpa memerlukan adanya otoritas terpusat. Prinsip kerjanya hampir mirip dengan konsensus *Proof of Stake*, dimana untuk mencapai konsensus yang valid maka dibutuhkan proses verifikasi dari pelaku-pelaku otoritas yang terdistribusi dan berjalan sesuai dengan protokol yang ada didalam metode konsensus tersebut.

Jika didalam PoS pemilihan pembuatan *block* bersifat *Pseudorandom* (berdasar jumlah dan umur *Coin/Token*), maka pada DPoS setiap delegasi/witness (pembuat *block*) akan ditentukan dengan *voting*. Setiap pemegang koin/token mempunyai suara (*vote*) yang bisa digunakan untuk memilih satu atau beberapa witness/validator dan dapat memindahkan *vote*-nya kepada witness lainnya setiap saat. *Validator/witness* terpilih (diurutkan berdasarkan jumlah *vote*) akan mempunyai otoritas untuk membuat sebuah *block* dan mempublikasikannya ke dalam jaringan *Blockchain*. Sistem distribusi konsensus *proof of work* dapat dilihat pada pada Gambar 2.11.



Gambar 2.14 *Delegated Proof of Stake Network*

Berikut merupakan penjelasan dari *Delegated Proof of Work Network*:

a) Setiap transaksi yang terjadi akan masuk ke jaringan (P2P).

b) *Token/Coin holder* yang menjalankan *Full Node* (*wallet*) akan melakukan *Vote* untuk menentukan beberapa *witness/delegasi*. Semakin besar jumlah *token* maka semakin besar juga nilai *vote*nya.

c) *Token/Coin holder* bisa melakukan *vote* terhadap dirinya sendiri setelah melakukan registrasi menjadi *witness* (*validator*) sesuai dengan protokol yang berlaku.

d) Dalam setiap putaran, masing-masing validator/witness terpilih (jumlah maksimal (N) *witness* berdasarkan protokol yang berlaku) akan memproses beberapa transaksi ke dalam sebuah *block.*

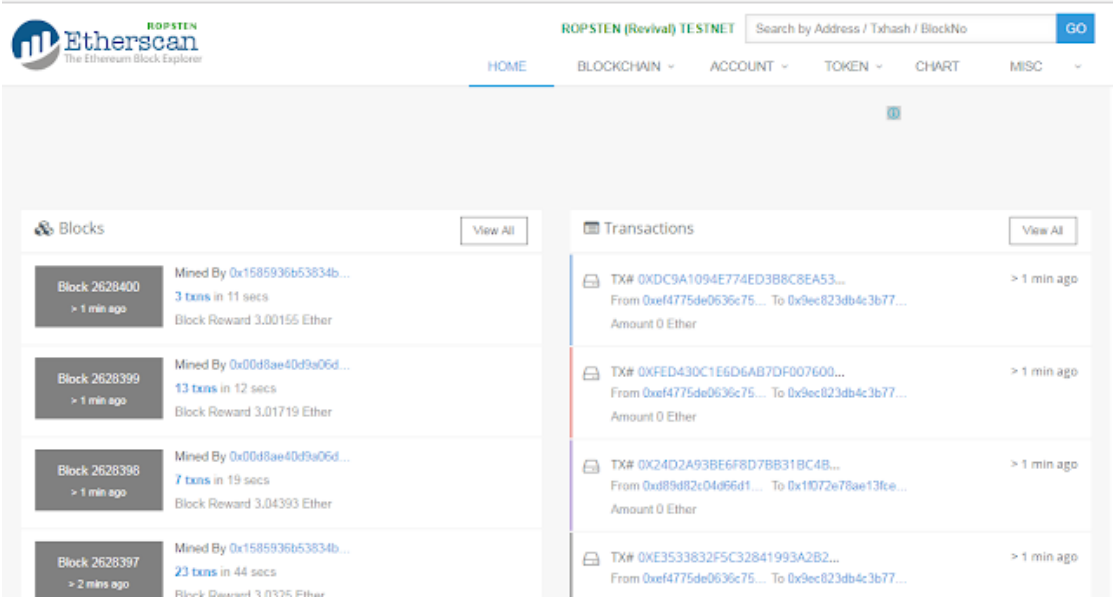
e) Sebelum *block valid* dibroadcast ke dalam *blockchain*, urutan ranking dari *validator/witness* akan berubah-ubah sesuai dengan jumlah *vote* yang didapat.

f) Setelah *block* berhasil divalidasi oleh *witness/validator*, maka masing-masing *block* akan didistribusikan ke dalam jaringan *blockchain* berdasarkan urutan *rank* dari *Validator*.

g) Masing-masing *Validator* (*witness*) akan menerima *reward* sesuai protokol yang berlaku.

2.2.5.10 *Test Network* (Testnet)

Didalam Ethereum dikenal juga sebuah jaringan *Blockchain* yang hanya difungsikan sebagai pengetesan saja. Jaringan testnet ini dibuat sangat mirip dengan jaringan *Blockchain* Ethereum yang asli karena agar memudahkan penggunaanya. Didalamnya juga berisi tentang catatan transaksi *blokchain,tx hash,token* yang pernah dibuat,dll. Contoh jaringan testnet Ethereum yang ada sekarang ini adalah Ropsten, Kovan dan Rinkeby [6]. Berikut adalah gambar dari *Ropsten* testnet.

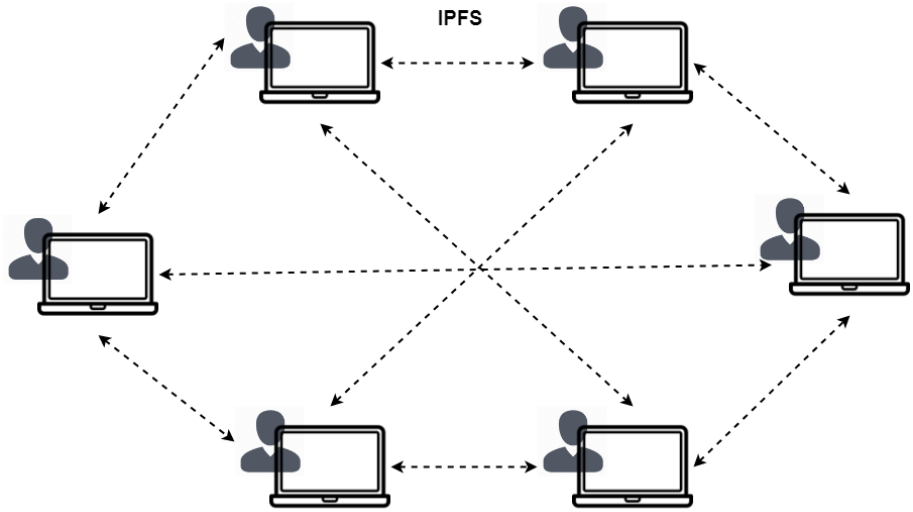


Gambar 2.15 Ropsten Tesnet

2.2.6 *InterPlanetary File System* (IPFS)

InterPlanetary File System (IPFS) adalah sistem *file* terdistribusi yang mensintesis ide-ide sukses dari sistem *peer-to-peer* sebelumnya, termasuk DHT, BitTorrent, Git, dan SFS. Kontribusi *InterPlanetary File System* (IPFS) menyederhanakan, mengembangkan, dan menghubungkan teknik yang terbukti ke dalam sistem kohesif tunggal, lebih besar dari jumlah bagian. *InterPlanetary File*

System (IPFS) menghadirkan platform baru untuk menulis dan menyebarkan aplikasi, dan sistem baru untuk mendistribusikan dan versi data besar [23].



Gambar 2.16 *Peer to Peer InterPlanetary File System* (IPFS)

2.2.6.1 *Content Based Address*

Dalam dunia sentralisasi untuk mengakses sebuah *file* dilakukan menggunakan *location-based addressing*. Dengan cara ini komunikasi dengan *hosting* komputer dan *file* yang berada didalamnya dapat dilakukan dengan mudah. Cara ini merupakan yang paling umum dilakukan saat ini. Namun, dalam kasus offline dan distributed skenario cara ini tidak dapat dilakukan. Maka dari itu *InterPlanetary File System* (IPFS) menggunakan cara yang disebut dengan *content-based address ing*. Untuk mendapatkan sebuah *file* yang diinginkan maka pengguna mencarinya menggunakan hasil *hash* dari objek *file* tersebut. Lalu, *node* yang tergabung dalam jaringan, yang memiliki *file* dengan *hash* yang dimaksud akan memberikannya kepada yang meminta.

2.2.6.2 *No Duplication*

File dengan konten yang sama tidak dapat diduplikasi dan hanya disimpan satu kali.

2.2.6.3 *Tamper Proof*

Data diverifikasi dengan checksum-nya, jadi jika *hash* berubah, maka *InterPlanetery File System* (IPFS) akan tahu data tersebut dirusak.

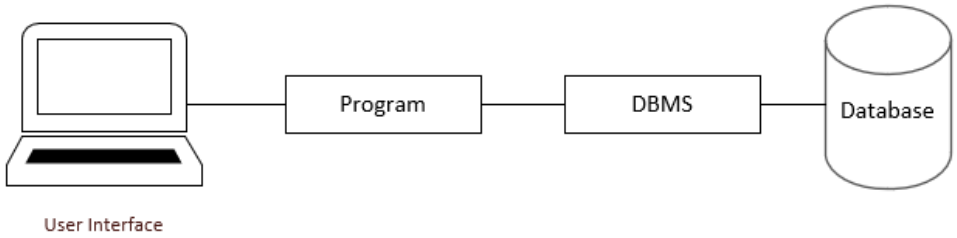
2.2.7 Basis Data

Basis Data terdiri dari dua kata, yaitu Basis dan Data. Basis dapat diartikan sebagai markas atau gudang, tempat bersarang/berkumpul. Sedangkan Data adalah representasi fakta dunia nyata yang mewakili seuatu objek seperti manusia (pegawai, siswa, pembeli pelanggan), barang, hewan, peristiwa, konsep, keadaan, dan sebagainya, yang diwujudkan dalam bentuk angka, huruf, simbol, teks, gambar, bunyi, atau kombinasinya [24].

Sebagai satu kesatuan istilah, basis data dapat diartikan didefinisikan sebagai kumpulan data yang terintegrasi dan diatur sedemikian rupa sehingga data tersebut dapat dimanipulasi, diambil, dan dicari secara cepat [25].

2.2.7.1 *Database Management Sistem* (DBMS)

Data*base* berbeda dengan Data*base Managenent Sistem (DBMS).* DBMS adalah kumpulan program yang digunakan untuk mendefinisikan, mengatur, dan memproses data*base*; sedangkan data*base* itu sendiri esensinya adalah sebuah struktur yang dibangun untuk keperluan penyimpanan data. DBMS alat yang berperan untuk membangun struktur tersebut [25]. Jadi dapat diartikan bahwa DBMS merupakan perantara antara user dengan data*base*. Peranan DBMS pada



suatu sistem dapat dilihat pada gambar 2.17.

Gambar 2.17 Peranan DBMS dalam Sistem

2.2.7.2 Bahasa Basis Data

Cara berinteraksi antara pemakai dengan basis data diatur dalam suatu bahasa khusus yang ditetapkan oleh perusahaan pembuat DBMS. Bahasa itu dapat kita sebut sebagai Bahasa Basis Data yang terdiri atas sejumlah perintah (statement) yang diformulasikan dan dapat diberikan user dan dikenali/diproses oleh DBMS untuk melakukan suatu aksi tertentu. Bahasa Basis Data dapat dibedakan kedalam dua bentuk yaitu Data *Definition Language* (DDL) dan Data *Manipulation Language* (DML).

a) Data *Definition Language* (DDL)

DDL merupakan struktur basis data yang menggambarkan skema basis data secara keseluruhan dan didesain dengan bahasa khusus. Adapun perintah-perintah yang dapat dilakukan dengan DDL yaitu :

1) Membuat, mengubah, menghapus tabel baru
2) Membuat indeks
3) Menentukan struktur penyimpanan tabel
4) Dan sebagainya

b) Data *Manipulation Language* (DML)

DML merupakan bentuk Bahasa Basis Data yang berguna untuk melakukan manipulasi dan pengambilan data pada suatu basis data. Manipulasi data dapat berupa :

1) Penambahan data baru ke suatu basis data
2) Penghapusan data dari suatu basis data
3) Pengubahan data di suatu basis data
4) Pengambilan data dari suatu basis data

2.2.8 *Object Oriented Analysis dan Design* (OOAD)

Konsep OOAD mencakup analisis dan desain sebuah sistem dengan pendekatan objek, yaitu analisis berorientasi objek (OOA) dan desain berorientasi objek (OOD). Analisis berorientasi objek (OOA) adalah tahapan menganalisis

spesifikasi atau kebutuhan akan sistem yang akan dibangun dengan konsep berorientasi objek. Sedangkan desain berorientasi objek (OOD) adalah tahapan perantara untuk memetakan spesifikasi atau kebutuhan sistem yang akan dibangun dengan konsep berorientasi objek. OOA dan OOD dalam proses yang berulang-ulang sering kali memiliki batasan yang samar, sehingga kedua tahapan ini sering juga disebut Analisis dan Desain Berorientasi Objek (OOAD) [1].

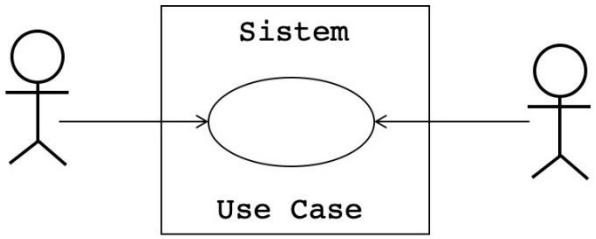
2.2.8.1 *Unified Modeling Language* (UML)

Unified Modeling Language (UML) adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan requirement, membuat analisis & desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek [26]. Dalam pengembangan suatu perangkat lunak, UML digunakan untuk memodelkan suatu sistem yang menggunakan konsep berorientasi object agar lebih bisa dipahami oleh banyak pihak yang terlibat dalam pengembangan.

Terdapat beberapa diagram yang biasanya digunakan untuk memodelkan analisis fungsional dalam rangka pengembangan perangkat lunak. Berikut diantaranya diagram yang umum digunakan :

a) Use Case Diagram

Menggambarkan sejumlah external actors dan hubungannya ke use case yang diberikan oleh sistem. Use case adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari use case symbol. Use case digambarkan hanya yang dilihat dari luar oleh *actor* dan bukan bagaimana fungsi yang ada di dalam sistem. Ilustrasi dari *actor*, *use case* dan *boundary* dapat dilihat pada Gambar 2.18.



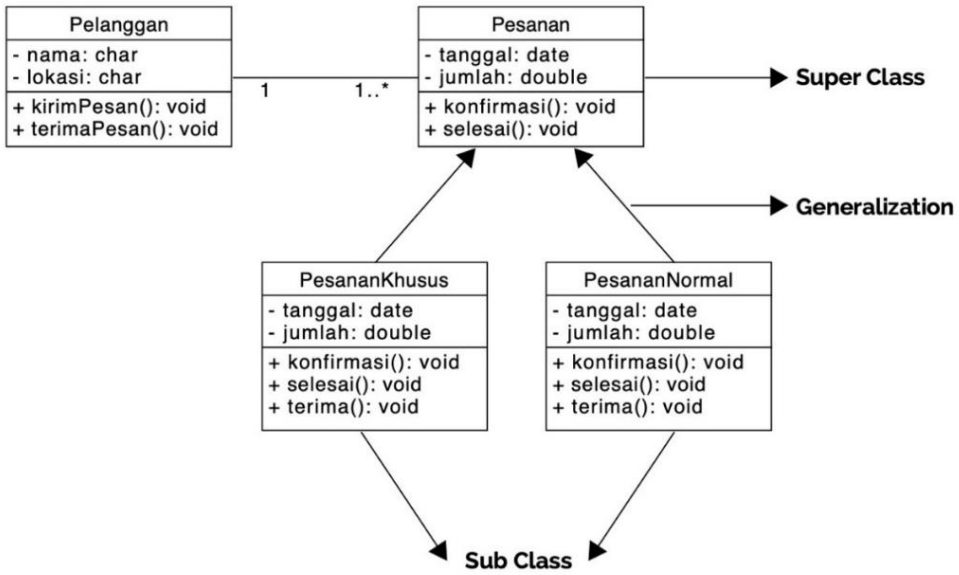
Gambar 2.18 ***Use Case Model***

b) Activity Diagram

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi. Activity diagram dibuat sebanyak aktivitas yang digambarkan pada use case diagram.

c) Class Diagram

Menggambarkan struktur statis class di dalam sistem. Class merepresentasikan sesuatu yang ditangani oleh sistem. Class dapat berhubungan dengan yang lain melalui berbagai cara: associated (terhubung satu sama lain), dependent (satu class tergantung/menggunakan class yang lain), specialed (satu class merupakan spesialisasi dari class lainnya), atau package (grup bersama sebagai satu unit). Sebuah sistem biasanya mempunyai beberapa class diagram. Contoh *class diagram* dapat dilihat pada gambar 2.19.

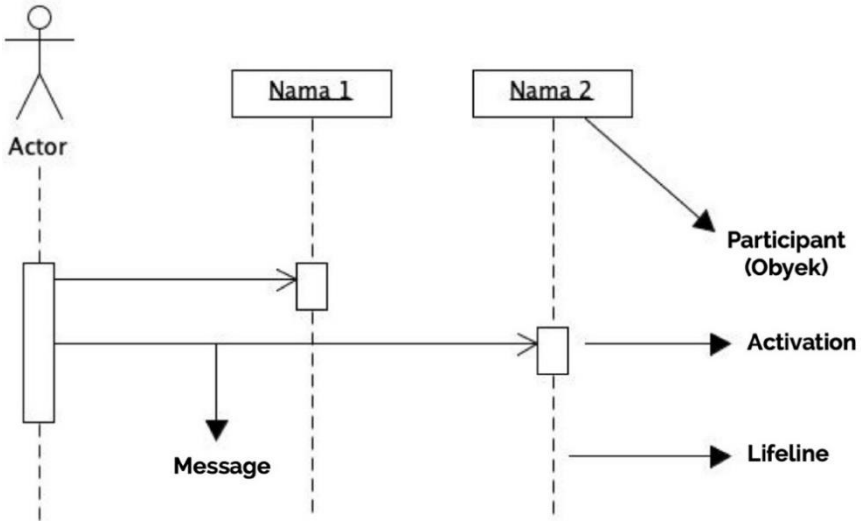


Gambar 2.19 Contoh *Class Diagram* Sistem Pemesanan

d) Sequence Diagram

Menggambarkan kolaborasi dinamis antara sejumlah object. Kegunaanya untuk menunjukkan rangkaian pesan yang dikirim antara object juga interaksi antara object, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

Simbol – symbol yang ada pada *sequence diagram* dapat dilihat pada gambar 2.20.



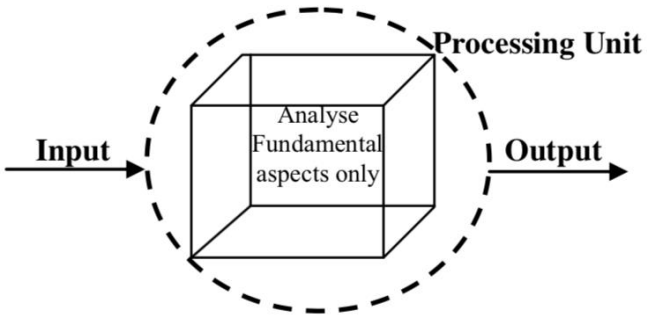
Gambar 2.20 Simbol-Simbol Yang Ada Pada *Sequence Diagram*

2.2.9 Pengujian Perangkat Lunak

Pengujian merupakan satu set aktifitas yang direncanakan dan sistematis untuk menguji atau mengevaluasi kebenaran yang diinginkan. Tujuan dari pengujian perangkat lunak adalah untuk menemukan kemungkinan terbesar kesalahan dengan jumlah yang dapat dikelola dari usaha yang diterapkan dalam kurun waktu yang rea*list*is. Betikut merupakan beberapa metode pengujian yang dilakukan pada penelitian ini.

2.2.9.1 *Black Box Testing*

Black Box Testing yaitu pengujian perangkat lunak dari segi spesifikasi fungsional tanpa menguji desain dan kode program [26]. Metode ini dimaksudkan untuk memastikan semua fungsionalitas berjalan dengan baik dan sesuai dengan apa yang telah direncanakan.



Gambar 2.21 Ilustrasi Metode *Black Box*

Metode *Black Box* memiliki keuntungan dan kekurangan diantaranya :

A. Keuntungan

1. Efisien diterapkan pada segmentasi kode yang besar.
2. Persepsi yang harus dimiliki *tester* sederhana.
3. Perspektif pengguna dipisahkan dari perspektif pengembang (*programmer* dan *tester* independen satu sama lain).
4. Pengembangan kasus uji relatif cepat.

B. Kekurangan

1. Hanya sejumlah skenario yang dilakukan dan dipilih dalam pengujian. Akibatnya, cakupan pada pengujian terbatas.
2. Tanpa spesifikasi yang jelas sehingga kasus uji sulit untuk dirancang.
3. Pengujian tidak efisien.

2.2.9.2 Pengujian Beta

Pengujian Beta dilakukan menggunakan kuesioner. Kuesioner adalah sebuah daftar pernyataan yang harus diisi oleh orang yang akan dievaluasi (responden). Metode yang digunakan dalam kuesioner pada penelitian ini adalah skala Likert. Dalam skala likert, responden diminta untuk membaca dengan seksama setiap pernyataan yang disajikan, kemudian responden diminta untuk menilai pernyataan-pernyataan tersebut [26].

Derajat penilaian responden terhadap suatu pernyataan terbagi dalam 5 kategori yang tersusun secara bertingkat, mulai dari Sangat Tidak Setuju (STS), Tidak Setuju (TS), Ragu-Ragu (R), Setuju (S), dan Sangat Setuju (SS). Atau dapat

pula sebaliknya. Pernyataan tiap kuesioner dibuat berdasarkan aspek-aspek yang diteliti. Bobot pemberian skor yang digunakan dapat dilihat pada Tabel 2.1.

Tabel 2.1 Bobot Pemberian Skor

Jenis pertanyaan	Bobot Pendapat				
	SS	S	R	TS	STS
Positif	5	4	3	2	1
Negatif	1	2	3	4	5

Skor yang telah dihitung pada setiap pernyataan kemudian dikalikan dengan masing-masing bobot tersebut sesuai dengan skenario kuesioner yang telah dibuat. Setelah itu, totalkan seluruh bobot jawaban tersebut kemudian bagi dengan total responden yang nantinya menjadi nilai rata-rata. Nilai rata-rata inilah yang diambil sebagai acuan sikap dimana jika nilai rata-rata kurang dari 3, maka dapat diartikan responden bersikap negatif dan jika nilai rata-rata lebih dari sama dengan 3, maka dapat diartikan responden bersikap positif terhadap tujuan yang ingin dicapai. Untuk lebih jelasnya dapat dilihat pada rumus dibawah ini [27].

$$x = \frac{\sum Total}{n}$$

Dimana

$x \geq 3\ bersikap\ positif$

$x \leq 3\ bersikap\ negatif$

Keterangan

x = nilai rata-rata

$\sum Total$ = jumlah seluruh nilai setelah dikalikan dengan bobot

n = total responden

2.2.10 Perangkat Lunak Pendukung

Perangkat lunak pendukung merupakan perangkat berupa bahasa pemrograman, aplikasi, *framework*, dan sebagainya. yang digunakan untuk

mendukung proses pengembangan sistem. Berikut merupakan beberapa perangkat lunak pendukung dalam penelitian ini.

2.2.10.1 Visual Studio Code

Visual Studio Code (VS Code) adalah sebuah teks editor ringan dan handal yang dibuat oleh Microsoft untuk sistem operasi *multiplatform*, artinya tersedia juga untuk versi Linux, Mac, dan Windows. Teks editor ini secara langsung mendukung bahasa pemrograman JavaScript, Typescript, dan Node.js, serta bahasa pemrograman lainnya dengan bantuan plugin yang dapat dipasang melalui *marketplace* Visual Studio Code (seperti C++, C#, Python, Go, Java, dst). Fitur-fitur yang disediakan oleh Visual Studio Code, diantaranya *Intellisense*, Git *Integration*, *Debugging*, dan fitur ekstensi yang menambah kemampuan teks editor [28].

2.2.10.2 Remix

Remix adalah alat *open source* yang membantu untuk menulis kontrak Solidity langsung dari *browser*. Ditulis dalam JavaScript, Remix mendukung penggunaan di *browser* dan secara lokal. Remix juga mendukung pengujian, debugging, penyebaran *smart contract* dan lainnya [29]. Untuk dapat mengakses Remix dapat dengan cara online melalui *browser* web seperti Chrome, dari salinan yang dipasang secara lokal, atau dari Mist (*browser* Ethereum Dapp).

2.2.10.3 Solidity

Solidity adalah bahasa tingkat tinggi yang berorientasi objek, untuk menerapkan *smart contract*. Solidity dipengaruhi oleh C ++, Python dan JavaScript dan dirancang dengan target Ethereum Virtual Machine (EVM). Solidity ditulis secara statis, mendukung *inheritance*, *library*, dan tipe kompleks yang ditentukan pengguna. Dengan Solidity Anda dapat membuat kontrak untuk penggunaan seperti *voting*, *crowdfunding*, dan *multi-signature wallets*. Saat menggunakan kontrak, Anda harus menggunakan versi solidity terbaru yang dirilis. Karena untuk memecah perubahan serta fitur baru dan perbaikan *bug* diperkenalkan secara teratur. Saat ini solidity menggunakan nomor versi 0.x untuk

menunjukkan laju perubahan yang cepat [6]. Berikut beberapa komponen yang ada di Solidity.

1. Variabel dan Tipe Data

Variabel global dalam sebuah *smart contract* akan disimpan secara permanent di dalam *storage* dari *contract account* tersebut. Terdapat beberapa tipe data yang disediakan solidity, tipe data ini dapat digunakan untuk menyusun tipe *struct* yang lebih kompleks. Berikut adalah beberapa tipe data tersebut, yaitu:

a) *Boolean*, menyimpan nilai *boolean* berupa *true* atau *false*.
b) *Integer*, pada tipe data ini terdapat sub-tipe berupa *signed* (int) dan *unsigned* (uint) *integer* dengan panjang *byte*s yang berbeda.
c) *Address*, merupakan tipe data yang menyimpan 20 *byte*s alamat akun Ethereum.
d) *Fixed-sized byte arrays*, berisi *byte array* dengan panjang dari 1 sampai 32.
e) *Dinamically-sized arrays*, berisikan tipe *bytes* dan tipe *string*.

2. Fungsi

Fungsi dalam bahasa pemrograman solidity sama seperti fungsi pada javascript. Sebuah fungsi dapat mengembalikan nilai ataupun tidak. Berikut adalah beberapa jenis fungsi, yaitu:

a) Fungsi *view*, sebuah fungsi dapat dideklarasi sebagai view apabila didalamnya tidak terjadi perubahan *state* terhadap variabel. Contohnya adalah fungsi yang hanya membaca nilai dari suatu variabel di dalam *contract*. Sehingga pemanggilan fungsi ini dapat dilakukan tanpa melakukan pengiriman transaksi.
b) Fungsi *pure*, sebuah fungsi yang tidak melalukan pembacaan ataupun perubahan terhadap *state* dari variabel yang ada di dalam *contract*. Sama halnya dengan fungsi *view* pemanggilan fungsi ini dapat dilakukan tanpa melakukan pengiriman transaksi. Fungsi *payable*, merupakan sebuah fungsi yang dapat menerima sejumlah nilai ether yang dikirimkan dari EOA ataupun *contract account* lain. Fungsi ini hanya dapat dipanggil melalui sebuah transaksi.

Fungsi yang tidak dideklarasikan sebagai *view*, *pure*, ataupun *payable* merupakan fungsi yang melakukan perubahan state terhadap variable namun tidak menerima sejumlah ether.

3. Visibility

Terdapat empat tipe yang berbeda dari visibilitas fungsi ataupun variabel di Solidity, yaitu:

a) *Public*, sebuah fungsi dengan visibilitas *public* dapat diakses oleh *internal contract*, *contract* turunan, dan panggilan dari luar *contract* tersebut. Pada variable visibilitas *public* membuat *getter* dan *setter* dibuat secara otomatis.
b) *Private*, fungsi ataupun variable *private* hanya dapat diakses di lokal *contract* saja.
c) *External*, seperti fungsi public hanya saja fungsi dengan visibilitas ini tidak dapat diakses oleh *internal contract*.
d) *Internal*, sama halnya seperti fungsi *private* fungsi dengan visibilatas ini tidak dapat diakses oleh *contract* turunannya.

4. Function Modifier

Modifier merupakan fungsi yang bisa digunakan untuk merubah perilaku dari fungsi lain. Fungsi ini dapat melakukan pengecekan sebelum sebuah fungsi dieksekusi. Contoh, sebuah fungsi hanya dapat dieksekusi oleh alamat akun tertentu

5. Events

Events merupakan *interface* yang memungkinkan pengguna menggunakan fasilitas EVM *logging*. Interface ini digunakan untuk mengetahui *state* dari sebuah *contract* tanpa harus berinteraksi langsung. *Events* biasanya digunakan untuk fungsi-fungsi yang dieksekusi secara *asynchronous*. Fungsi-fungsi yang merubah *state* dari *contract* akan melalui proses mining hingga akhirnya menghasilkan *state* baru.

2.2.11 ReactJS

ReactJS merupakan *library* javascript yang dapat digunakan untuk mendeskripsikan *view* (contohnya elemen dalam HTML) berdasarkan beberapa

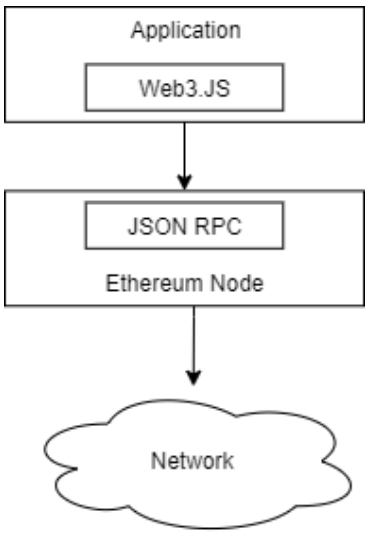
state (keadaan, biasanya dalam bentuk data). Dalam perkembangannya ReactJS ditulis juga dengan nama React [30].

Library ini dibuat oleh *facebook* dan di distribusikan secara gratis. Salah satu contoh yang menonjol dari penerapan ReactJS adalah kemampuannya untuk memperbaharui tampilan di layar monitor secara cepat. Hal ini bisa dibuktikan melalui *facebook*. Pada saat menulis status, membalas komentar, atau menekan tombol like, hasil dari aktivitas tersebut langsung terlihat dilayar monitor saat itu juga.

2.2.12 Web3Js

Web3js memungkinkan untuk berinteraksi dengan *node* Ethereum lokal atau jarak jauh, menggunakan koneksi HTTP atau IPC. Web3js merupakan pustaka yang memungkinkan untuk melakukan tindakan pengiriman seperti mengirim *ether* dari akun satu ke akun lain, membaca dan menulis *smart contract* dan lain sebagainya [31].

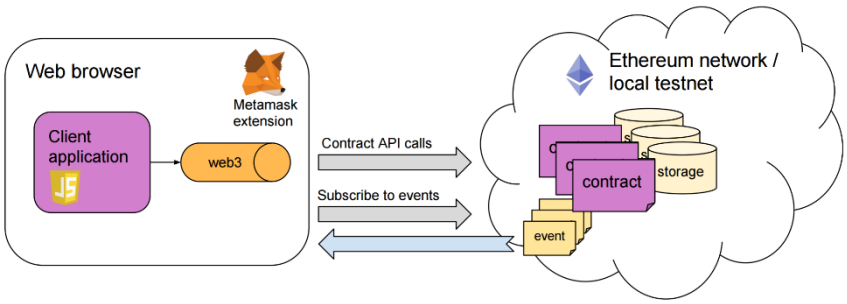
Web3.js dapat digunakan untuk terhubung ke jaringan Ethereum melalui *node* Ethereum yang memungkinkan akses melalui HTTP. Akses tersebut dapat berupa *node* lokal, *node* yang dihosting oleh penyedia *DApp*, atau *gateway* publik seperti Infura, yang mengoperasikan titik akses Ethereum gratis. Salah satu cara umum untuk mengintegrasikan aplikasi *browser* web dengan Ethereum adalah dengan menggunakan ekstensi *browser* Metamask dengan kombinasi dengan Web3js.



Gambar 2.22 Interaksi Web3.JS dengan *Node* Ethereum

2.2.13 Metamask

Metamask adalah sebuah aplikasi dalam *browser* yang memudahkan kita berinteraksi dengan *website* berbasis Ethereum. *Website* berbasis Ethereum biasa dikenal dengan dApps (atau aplikasi terdesentralisasi). Metamask memudahkan kita dalam hal transaksi. Secara sederhana, Metamask adalah dompet Ethereum di dalam *browser*, yang memungkinkan kita menyimpan, membeli, mengirim dan menggunakan Ethereum. MetaMask dapat dipasang di Chrome, Firefox, Opera, dan *browser* Brave yang baru. Menggunakan Metamask dengan kombinasi Web3.js, dalam antarmuka web adalah cara mudah untuk berinteraksi dengan jaringan Ethereum [32].



Gambar 2.23 Interaksi Metamask dengan Web *Browser* dan Ethereum