BAB 2

LANDASAN TEORI

2.1 Autentikasi Produk

Autentikasi produk dari sisi konsumen merupakan sebuah proses untuk membedakan produk asli dan produk palsu, yang menjadi faktor utama penentu perilaku pembelian (Fejes and Wilson, 2013). Autentikasi produk dilakukan sebagai langkah untuk melindungi diri dari penipuan dan kerugian finansial ketika berhadapan dengan produk-produk palsu. Dalam proses ini, konsumen bergantung secara heuristik untuk menilai produk dengan menggunakan berbagai nilai intrinsik dan ekstrinsik terkait produk tersebut. Nilai intrinsik mencakup informasi mengenai atribut fisik dan perfoma yang nyata, sedangkan nilai ekstrinsik mencakup informasi yang diasosiasikan dengan produk tersebut. Nilai-nilai intrinsik dan ekstrinsik yang digunakan pada autentikasi produk dapat dilihat pada Tabel 2.1.

Tabel 2.1 Nilai instrinsik dan ekstrinsik pada autentikasi produk
(Fejes and Wilson, 2013)

Nilai Intrinsik	Nilai Ekstrinsik
• Kemasan • Kualitas instruksi manual • Ejaan nama *brand* • Gambar atau ejaan logo • Kualitas material	• Harga • Lokasi pembelian atau jenis *outlet* • Negara asal pembuat

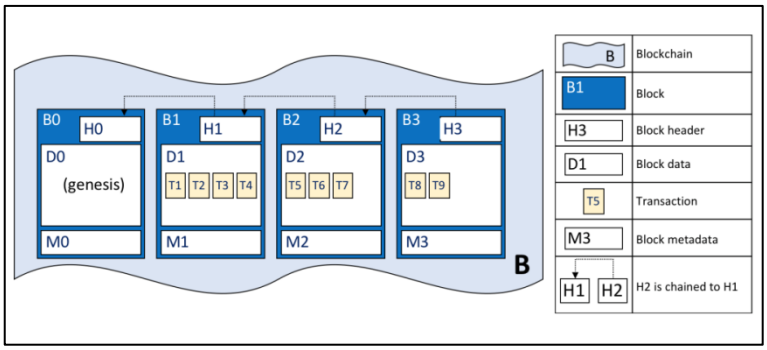
Terdapat berbagai teknologi autentikasi yang dapat digunakan untuk melawan produk palsu, seperti *radio frequency identification* (RFID) dan *quick response* (QR) *code* (Chin, Kim and Choi, 2017). RFID menggunakan gelombang radio untuk menyimpan dan mengambil data dari sebuah *identification chip*, yang

disebut sebagai RFID *tags*. *QR code* merupakan *barcode* 2 dimensi yang mudah untuk dibuat dan dapat menyimpan banyak informasi (maksimal 7,089 karakter). Kelebihan yang dimiliki *QR code* antara lain: data pada *QR code* dapat direstorasi jika sebagian simbol rusak atau kotor, dapat membaca data secara cepat dari segala arah (360 derajat), dapat dicetak pada kertas atau disimpan sebagai gambar pada *smartphones*.

2.2 Blockchain

Blockchain merupakan sebuah *database* terdistribusi yang menyimpan *record* konsensus dari berbagai transaksi ke dalam sebuah rantai blok, yang divalidasi dan dikelola oleh sekumpulan komputer pada sebuah jaringan (Sarmah, 2018). *Blockchain* termasuk salah satu cara untuk mengimplementasikan *database* terdistribusi, namun tidak semua *database* terdistribusi menggunakan *blockchain* (Rutland, 2018). *Blockchain* dapat diimplementasikan pada jaringan terpusat atau terdesentralisasi, yang merujuk pada hak partisipan pada *ledger*. Pada jaringan terpusat, hanya partisipan yang dapat dipercaya dan bereputasi baik yang dapat menambahkan transaksi ke dalam *ledger*. Pada jaringan terdesentralisasi, setiap partisipan dapat bergabung dan menambahkan transaksi ke dalam *ledger*.

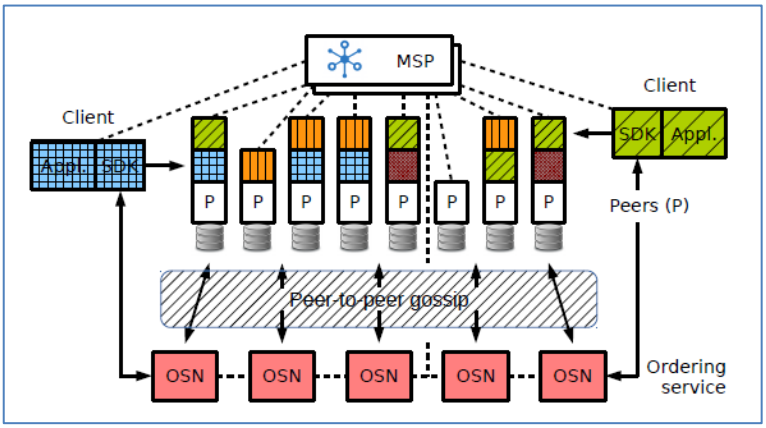
Blockchain terdiri dari susunan blok yang dihubungkan dengan *hash* dari seluruh transaksi pada blok sebelumnya. Hal ini membuat transaksi yang disimpan pada *blockchain* tidak dapat diubah. Satu blok dapat menyimpan beberapa transaksi, kecuali blok genesis yang merupakan blok kosong. Blok genesis menyimpan konfigurasi transaksi yang menjadi *initial state* dari sebuah *channel*. Struktur *blockchain* dapat dilihat pada Gambar 2.1.



Gambar 2.1 Struktur *blockchain*
(Hyperledger Fabric Documentation, 2020)

2.3 Hyperledger Fabric

Berikut adalah konsep-konsep terkait Hyperledger Fabric. Contoh struktur jaringan pada Hyperledger Fabric dapat dilihat pada Gambar 2.2.



Gambar 2.2 Struktur jaringan
(Androulaki *et al.*, 2018)

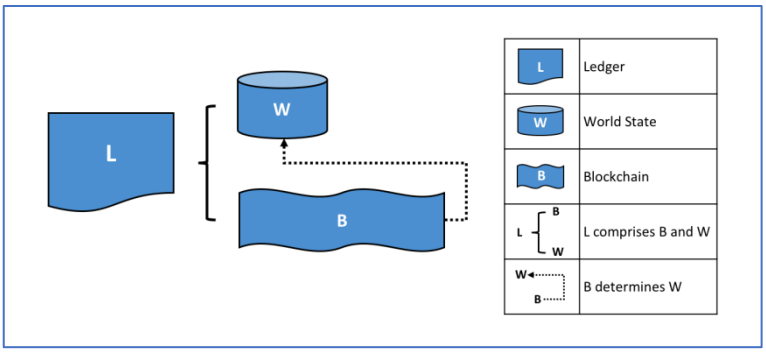
2.3.1 Komponen

Berikut adalah komponen-komponen yang terdapat pada Hyperledger Fabric.

1. Ledger

Ledger terdiri dari 2 bagian, yaitu *world state* dan *blockchain*. *World state* merupakan sebuah *database* yang menampung nilai terkini dari

sekumpulan *ledger states*, yang direpresentasikan dengan pasangan *key-value*. *Blockchain* merupakan *transaction log* yang menyimpan seluruh perubahan pada *ledger states* hingga nilai terkini. Artinya, *world state* merupakan turunan dari *blockchain*. Struktur *ledger* pada Hyperledger Fabric dapat dilihat pada Gambar 2.3.



Gambar 2.3 Struktur *ledger*
(Hyperledger Fabric Documentation, 2020)

2. *Smart Contract*

Smart contract merupakan program yang dapat dieksekusi oleh aplikasi untuk membuat dan menyimpan transaksi pada *ledger*. *Smart contract* disebut juga dengan *chaincode*. Secara programatik, *smart contract* akan mengakses *world states* untuk melakukan operasi *get, put,* atau *delete* pada *ledger states*, dan *blockchain* untuk menambahkan *history* dari transaksi. Perlu dicatat bahwa operasi *delete* hanya menghapus nilai pada *database*, sedangkan *history* dari transaksi tidak dapat diubah.

Smart contract mendefinisikan logika bisnis yang dilakukan diantara pihak yang bertransaksi. Setiap *smart contract* memiliki sebuah *endorsement policy* yang mendefinisikan organisasi mana saja yang harus menyetujui transaksi yang dibuat oleh *smart contract*, agar transaksi tersebut dikatakan valid.

3. *Nodes*

Terdapat 3 jenis *nodes* yang membentuk sebuah jaringan, yaitu:

a. *Clients*

Clients berfungsi untuk mengirimkan *transaction proposal* yang ingin dieksekusi, mengumpulkan *endorsements* menjadi *transaction message*, dan melakukan *broadcast transaction message*.

b. *Peers*

Peers berfungsi untuk mengeksekusi *transaction proposal* dan memvalidasi transaksi. Setiap *peers* mengelola *ledger*, baik *world state* maupun *blockchain*. Tidak semua *peers* mengeksekusi *transaction proposal*, hanya sebagian *peers* yang didefinisikan pada *endorsement policy* yang melakukannya. Sebagian *peers* ini disebut juga dengan *endorsers*.

c. *Ordering Service Nodes*

Ordering Service Nodes berfungsi untuk mengurutkan sekumpulan transaksi secara ketat, membungkusnya ke dalam blok, dan menyebarkan blok kepada *peers* yang terhubung dengannya. *Ordering Service Nodes* disebut juga dengan *orderers*.

4. *Membership Service Provider* (MSP)

MSP menyimpan seluruh identitas *nodes* pada jaringan dan berfungsi untuk memberikan identias *nodes* yang diperlukan untuk autentikasi transaksi, verifikasi integritas transaksi, *sign endorsements*, validasi *endorsements*, dan operasi autentikasi lainnya pada *blockchain*. MSP

digunakan untuk mendefinisikan identitas yang dapat dipercaya pada jaringan dan memberikan akses kontrol kepada partisipan.

5. *Peer-to-Peer Gossip Protocol*

Terdapat 3 fungsi utama dari protokol ini, yaitu:

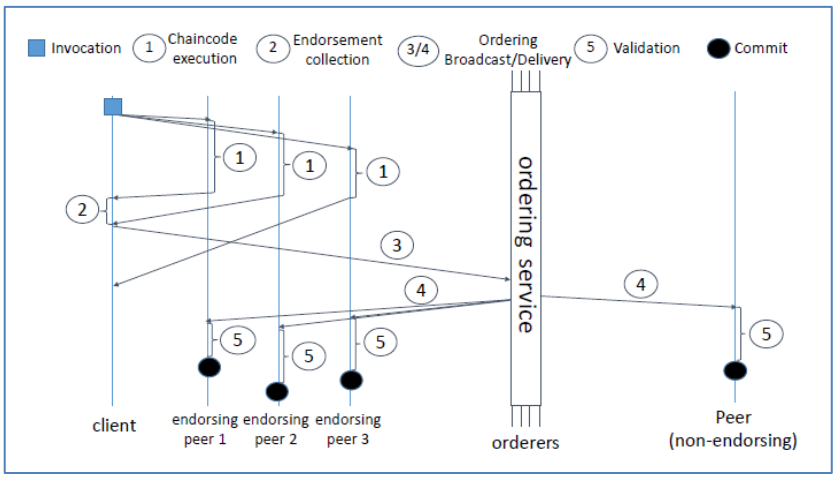
a. Melakukan *peer discovery* dan *channel membership* secara terus menerus untuk mendeteksi *peers* yang aktif dan tidak.

b. Menyebarkan data *ledger* kepada seluruh *peers* dalam sebuah *channel*, sehingga *peers* yang memiliki blok yang hilang dapat menyinkronkan dirinya.

c. Melakukan *peer-to-peer state transfer update* kepada *peer* yang bergabung.

2.3.2 Alur Transaksi

Hyperledger Fabric menerapkan arsitektur *execute-order-validate* dalam alur transaksinya, yang dapat dilihat pada Gambar 2.4 dan dijelaskan sebagai berikut.



Gambar 2.4 Alur transaksi
(Androulaki *et al.*, 2018)

1. *Execution Phase*

Pada tahap ini, *client* mengirimkan *transaction proposal* yang telah diberi *signature* untuk dieksekusi oleh *endorsers*. *Transaction proposal* memiliki sekumpulan parameter input yang dibutuhkan untuk *smart contract* yang dispesifikasikan. Simulasi eksekusi *transaction proposal* dilakukan dengan *world state* setiap *endorsers*, tanpa mengubah nilai yang disimpannya. Setelah simulasi selesai, setiap *endorsers* menghasilkan *transaction response* yang berisi *key-value* yang diubah (*writeset*) dan *key-value* yang dibaca beserta versinya (*readset*).

Transaction response kemudian diberikan *signature* (*endorsement*) dan dikembalikan kepada *client*. *Client* akan mengumpulkan *endorsements* hingga memenuhi *endorsement policy*. Seluruh *endorsements* harus memiliki *writeset* dan *readset* yang sama. Setelah *endorsements* yang dibutuhkan terpenuhi, *client* membuat transaksi dan mengumpulkannya ke *orderers*.

2. *Ordering Phase*

Pada tahap ini, *orderers* mengurutkan seluruh transaksi yang dikumpulkan oleh *clients* secara ketat dan membungkusnya ke dalam blok. Jumlah transaksi pada sebuah blok bergantung pada konfigurasi parameter jumlah dan ukuran transaksi (BatchSize), dan waktu tunggu blok (BatchTimeout). Perlu dicatat bahwa transaksi tidak diurutkan berdasarkan waktu sampai di *orderers*, mengingat banyak transaksi dapat dikumpulkan oleh *client* yang berbeda secara bersamaan. Transaksi yang sama dapat dibungkus pada 2 blok yang berbeda. Blok yang dihasilkan bersifat final dan tidak ada

ledger forks.

3. Validation Phase

Pada tahap ini, *orderers* mengirimkan blok ke *peers* yang terhubung dengannya, sementara *peers* tersebut melakukan *gossip protocol* ke *peers* lainnya untuk mengirimkan blok. *Peers* kemudian melakukan 3 langkah validasi untuk seluruh transaksi pada blok diterima, yaitu:

a. Mengecek *endorsement policy* secara paralel. Jika *endorsement policy* tidak terpenuhi, maka transaksi tersebut akan ditandai tidak valid.

b. Mengecek konflik *read-write* secara sekuensial. Jika versi *readset* tidak sama dengan versi *keys* terkait pada *world state*, maka transaksi tersebut akan ditandai tidak valid.

c. Memperbaharui *ledger*. Hanya transaksi yang valid yang akan memperbaharui *world state*, sedangkan seluruh transaksi yang valid dan tidak valid akan disimpan pada *blockchain*.

2.3.3 Konfigurasi Channel

Channel merupakan sebuah *subnet* privat yang digunakan untuk bertransaksi. Setiap *channel* memiliki blok genesis masing-masing, yang menyimpan konfigurasi *channel* tersebut. *File* konfigurasi disimpan pada configtx.yaml dengan struktur sebagai berikut.

1. Organizations

Organizations mendefinisikan organisasi mana saja yang tergabung dalam sebuah *channel*.

2. *Capabilities*

Capabilities mendefinisikan kapabilitas *channel* agar dapat dijalankan oleh *orderers* atau *peers* yang memiliki versi *binaries* Hyperledger Fabric yang berbeda-beda.

3. *Application*

   Application mendefinisikan *policies* yang mengatur bagaimana persetujuan untuk menggunakan *smart contract* dan pembaharuan konfigurasi *channel*.

4. *Orderer*

   Orderer mendefinisikan jenis algoritma konsensus yang digunakan, BatchSize dan BatchTimeout, dan *policies* yang mengatur *orderers*. BatchSize dan BatchTimeout diatur ketika melakukan evaluasi perfoma, seperti yang dilakukan (Shalaby *et al.*, 2020).

5. *Channel*

   Channel mendefinisikan *policies* yang mengatur konfigurasi *channel* pada tingkat tertinggi. Terdapat 2 jenis *channel* yang digunakan, yaitu *application channel* dan *system channel*. *Application channel* digunakan untuk membagikan ledger dan *smart contract* kepada partisipan. *System channel* digunakan untuk mendefinisikan organisasi yang tergabung dalam *consortium*. Hanya organisasi yang didefinisikan pada *consortium* yang dapat bergabung pada *application channel*.

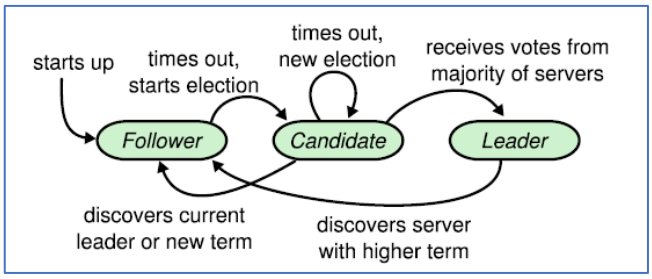
6. *Profiles*

   Profiles mendefinisikan *channel* yang ingin dibuat dengan konfigurasinya.

2.4 Raft

Raft merupakan algoritma konsensus yang bekerja dengan mengelola *replicated log* (Ongaro and Ousterhout, 2019). *Replicated log* merupakan implementasi dari *replicated state machines*, di mana *state machine* pada sekumpulan *server* mengkomputasikan salinan *state* yang identik, dan dapat terus beroperasi meskipun terdapat *server* yang mati. Setiap *server* menyimpan log yang berisi sekumpulan perintah, yang akan dijalankan oleh *state machine* secara berurutan. Pada Hyperledger Fabric, setiap *orderer* memiliki sebuah *finite state machine* (FSM) yang digunakan untuk memastikan urutan log (transaksi) pada setiap *orderer* bersifat deterministik. Raft bersifat *crash fault tolerant* (CFT), jika mayoritas *orderers* yang berjalan (*quorum*) berjumlah $\frac{n}{2} + 1$, di mana *n* adalah jumlah *orderers* yang berpartisipasi (*consenter set*).

Setiap *orderer* memiliki peran sebagai *leader*, *follower*, atau *candidate*, di mana transisi yang dapat terjadi dari setiap *state* dapat dilihat pada Gambar 2.5. Hanya ada satu *orderer* yang berperan sebagai *leader*, sedangkan *orderers* lain berperan sebagai *follower*. *Follower* berfungsi untuk menerima *request* dari *leader* dan *candidate*. *Leader* berfungsi untuk menangani seluruh *request* yang dikirimkan oleh *client* (jika terdapat *request* yang dikirimkan dari *client* ke *follower*, maka *follower* akan mengirimkannya kembali ke *leader*). *Candidate* merupakan kandidat *orderers* yang akan dipilih salah satunya sebagai *leader*. Komunikasi antar *orderers* dilakukan dengan *remote procedure call* (RPC).



Gambar 2.5 Raft *states*
(Ongaro and Ousterhout, 2019)

Berikut adalah 2 konsep utama yang terkait Raft.

1. *Leader Election*

   Raft membagi waktu menjadi *terms*, di mana setiap *terms* dimulai dengan *election*. Seluruh *orderers* bermula dari *state follower*. *Follower* yang tidak menerima komunikasi dalam waktu tunggu tertentu dapat mencalonkan dirinya sebagai *candidate*. Waktu tunggu setiap *follower* (*election timeout*) bersifat *random*, di antara 150 – 300ms. Hanya *candidate* yang memiliki mayoritas *vote* yang berhak menjadi *leader*. Jika *leader* mati atau banyak *candidate* memiliki *vote* yang seimbang (*split votes*), maka akan dilakukan *election term* yang baru.

   Election dimulai dengan salah satu *candidate* menambahkan nilai *term*-nya dengan satu, memberi *vote* pada dirinya sendiri, dan mengirimkan RequestVote RPC kepada *orderers* lain. *Orderers* lain yang belum memberikan *vote* akan memberikan *vote* pada *candidate* tersebut (*first-come-first-served*) dan mengulang kembali *election timeout*-nya. *Leader* akan mengirimkan AppendEntries RPC (*heartbeat*) kepada *follower* secara periodik, untuk mencegah *election baru* dengan mengulang kembali *election timeout* pada setiap *follower*. *Election term* baru akan dilakukan ketika *leader* berhenti mengirimkan *heartbeat*.

2. *Log Replication*

Leader yang telah dipilih akan memulai menerima *request* dari *client*. *Leader* akan menambahkan transaksi sebagai entri log baru dan mengirimkan AppendEntries RPC secara paralel ke setiap *follower*. Sebuah entri log baru akan di-*commit* oleh *leader* ketika mayoritas *follower* telah mereplikasi entri *log* tersebut. Entri log yang di-*commit* berarti entri log tersebut aman untuk dijalankan pada *state machine*. Setelah itu, *leader* akan memberikan *response* kepada *client*, sementara *follower* akan melakukan *commit* entri log.

2.5 ERC-20 Token Standard

ERC-20 merupakan sebuah standar *fungible token*, di mana satu token bernilai sama dengan token lainnya. ERC-20 diimplementasikan sebagai standar API untuk pengembangan token pada *smart contract* (Vogelsteller and Buterin, 2015). Standar token ini bersifat *account-based*, di mana setiap pengguna memiliki *token wallet* masing-masing. Berikut adalah spesifikasi metode standar token ERC-20.

1. name()

   Fungsi ini mengembalikan nama token.

2. symbol()

   Fungsi ini mengembalikan simbol token.

3. decimals()

   Fungsi ini mengembalikan jumlah desimal yang digunakan token.

4. totalSupply()

   Fungsi ini mengembalikan jumlah persediaan seluruh token.

5. balanceOf()

   Fungsi ini mengembalikan jumlah saldo dari akun pemilik.

6. transfer()

   Fungsi ini melakukan pemindahan sejumlah token dari akun pemilik ke akun tujuan.

7. transferFrom()

   Fungsi ini melakukan pemindahan sejumlah token yang diizinkan oleh akun pemilik untuk akun pemakai kepada akun tujuan.

8. approve()

   Fungsi ini mengizinkan sejumlah token dari akun pemilik digunakan oleh akun pemakai.

9. allowance()

   Fungsi ini mengembalikan jumlah token yang diizinkan oleh akun pemilik kepada akun pemakai.

10. mint() (Opsional)

   Fungsi ini menambahkan jumlah persediaan seluruh token, yang hanya dapat dilakukan oleh akun yang diizinkan. Fungsi ini merupakan ekstensi dari ERC-20, yaitu ERC-20 *Mintable* (OpenZeppelin Documentation, 2021).

2.6 Throughput dan Latency

Berikut adalah satuan pengukuran evaluasi performa Hyperledger Fabric yang diuji (Hyperledger Performance and Scale Working Group, 2018).

1. *Transaction Latency*

   Transaction latency (*X*) merupakan waktu yang dibutuhkan oleh sebuah transaksi tersedia pada jaringan dimulai ketika transaksi tersebut dikumpulkan, yang mencakup waktu propagasi dan konsensus.

$$X = (Confirmation\ time\ @\ network\ threshold) - submit\ time \quad \ldots(2.1)$$

2. *Transaction Throughput*

   Transaction throughput (Y) merupakan jumlah transaksi *valid* yang berhasil di-*commit* pada *blockchain* untuk setiap *nodes* dalam rentang waktu tertentu. Satuan pengukuran ini dinyatakan dalam *transactions per second* (TPS).

$$Y = Total\ committed\ transactions\ /\ total\ time\ in\ seconds \quad \ldots(2.2)$$

2.7 Konfigurasi Hyperledger Caliper

Terdapat 2 konfigurasi *file* Hyperledger Caliper yang diperlukan, yaitu *benchmark* dan *workload*. *Benchmark* berisi pengaturan terhadap atribut pengujian yang ingin dilakukan, yang dapat dilihat pada Tabel 2.2.

Tabel 2.2 Pengaturan *benchmark test*
(Hyperledger Caliper Documentation, 2020)

Atribut	Deskripsi
test.name	Nama *benchmark*
test.description	Deskripsi *benchmark*
test.workers	Objek yang berkaitan dengan konfigurasi *worker*
test.workers.type	Jenis *worker*
test.workers.number	Jumlah proses *worker* untuk menjalankan *workload*
test.rounds	*Array* objek yang berkaitan dengan konfigurasi ronde pengujian
test.rounds[i].label	Label ronde pengujian
test.rounds[i].txNumber	Jumlah transaksi yang dikumpulan
test.rounds[i].txDuration	Durasi (dalam detik) Caliper mengumpulkan transaksi secara terus-menerus
test.rounds[i].rateControl	Objek yang berkaitan dengan *rate controller* yang digunakan
test.rounds[i].callback	Path modul *workload*
test.rounds[i].arguments	Objek yang berisi argumen untuk diberikan kepada modul *workload*

Workload berisi modul untuk menyimulasikan transaksi, yang terdiri dari 3 fungsi berikut.

1. init()

   Fungsi ini digunakan untuk menginisialisasi data yang diperlukan sebelum suatu ronde pengujian dimulai.

2. run()

   Fungsi ini digunakan untuk menjalankan suatu ronde pengujian.

3. end()

   Fungsi ini digunakan untuk membersihkan inisialisasi data sebelum ronde pengujian selanjutnya dimulai.