

Disusun Oleh  
Purwono, S.Kom., M.Kom

# PENGANTAR TEKNOLOGI BLOCKCHAIN

Dilengkapi dengan praktikum  
pembuatan konseptual blockchain  
dengan javascript



Penerbit **UHB Press**

# **Pengantar Teknologi Blockchain**

Dilengkapi dengan Praktikum Pembuatan Konseptual  
Blockchain dengan Javascript

Purwono, S.Kom., M.Kom.

**Penerbit UHB Press**

## **Pengantar Teknologi Blockchain**

Dilengkapi dengan Praktikum Pembuatan Konseptual Blockchain dengan Javascript

Oleh:

Purwono, S.Kom., M.Kom

Hak Cipta © 2022 pada penulis,

Copy Editor: Dewira Barlian, S.Mat

Hak Cipta dilindungi oleh undang-undang. Dilarang memperbanyak atau memindahkan Sebagian atau keseluruhan isi buku ini dalam bentuk apapun, secara elektronik maupun mekanis, termasuk mefotokopi, merekam, atau dengan Teknik perekaman lainnya, tanpa izin tertulis dari penerbit.

Diterbitkan oleh Penerbit UHB Press

Jl. Raden Patah No.100, Ledug, Kembaran, Banyumas,  
Jawa Tengah, Telp. (0281) 6843493, Fax. (0281) 6843494,  
Purwokerto 53182

Purwono, S.Kom., M.Kom.

Pengantar Teknologi Blockchain

-edisi Pertama – Purwokerto: UHB Press, 2022

xvi + 152 hlm, 1 Jil: 23 cm

ISBN: 978-623-88102-0-8

## **Sinopsis**

Teknologi blockchain terus berkembang dan menjadi sangat populer akhir-akhir ini. Teknologi ini telah masuk pada berbagai sektor kehidupan dan mempengaruhi kehidupan nyata. Buku ini hadir sebagai pengantar akan teknologi blockchain tersebut mulai dari motivasi dibalik terciptanya blockchain, cara kerja blockchain hingga detail teknis bagaimana teknologi ini bekerja. Untuk memperdalam pemahaman tentang blockchain, penulis selanjutnya membuat sebuah pelatihan sederhana yaitu mengimplementasikan konseptual blockchain dengan bahasa pemrograman javascript. Konsep-konsep cara kerja blockchain dikemas dalam praktikum yang benar-benar memperlihatkan bagaimana teknologi ini bekerja secara rinci dan jelas. Buku ini sangat cocok bagi pemula yang masih kebingungan memahami cara kerja blockchain sebenarnya.

## **Kata Pengantar**

Teknologi *blockchain* terus berkembang dan menjadi populer. Belum banyak buku-buku yang membahas secara detail terkait teknologi ini. Umumnya orang hanya mengenal *blockchain* itu identik dengan mining mata uang digital atau melakukan tradingnya saja, padahal sebetulnya teknologi dibalik *cryptocurrency* ini memiliki manfaat yang lebih banyak.

Teknologi *blockchain* sebetulnya dapat dimanfaatkan untuk berbagai jenis kebutuhan teknologi yang peduli pada keamanan data. *Blockchain* merupakan jaringan terdesentralisasi yang sangat bertolak belakang dengan jaringan terpusat (*centralized network*). Jaringan terpusat memiliki resiko akan penyalahgunaan wewenang otoritas tunggal dalam memanipulasi data sehingga *blockchain* menerapkan konsep terdesentralisasi agar setiap orang memiliki kekuatan yang sama untuk memvalidasi setiap data yang ada di jaringan *blockchain*. Satu saja data yang berubah maka akan segera terdeteksi oleh orang lain dalam jaringan.

Dalam buku ini kami akan menjelaskan tentang konsep dasar bagaimana teknologi *blockchain* ini bekerja. Dimulai dari konsep terdesentralisasi, pengertian blok transaksi, kriptografi, ataupun proses penambangan

(mining). Implementasi dari konsep dasar selanjutnya dibuat dengan simulasi *blockchain* sederhana dengan bahasa *Javascript*. Atas dasar ini, kami mengasumsikan bahwa pembaca setidaknya telah memahami dasar-dasar terkait bahasa pemrograman *javascript* agar dapat mengikuti dengan baik.

Pada seri berikutnya, buku ini juga akan mempelajari bahasa *solidity* yang digunakan dalam pembuatan smart contract di *Blockchain Ethereum*. Bahasa *solidity* mirip dengan *javascript* jadi ketika anda sudah memiliki bekal dasar *javascript* tinggal menyesuaikan dengan pola yang ada di *solidity*. *Smart contract* kemudian disimulasikan dengan Jaringan testing *Blockchain Ethereum* yang dapat dijalankan secara lokal.

Untuk menambah pengetahuan tentang *blockchain*, pada seri selanjutnya kami menghadirkan pembuatan aplikasi rekam medis terdesentralisasi dengan memanfaatkan *smart contract* agar dapat berjalan di jaringan *blockchain ethereum*. Proyek ini bisa digunakan sebagai metode latihan untuk teman-teman yang memang tertarik dan ingin terjun sebagai *developer blockchain ethereum*.

Terakhir, semoga buku ini bisa menambah keilmuan dan inspirasi teman-teman ketika ingin membangun

aplikasi terdesentralisasi dengan blockchain ethereum. Penulis juga menyampaikan terimakasih yang sebesar-besarnya kepada Penerbit UHB Press yang mau menerbitkan karya penulis.

Purwokerto, Juni 2022

**Purwono, S.Kom., M.Kom.**

## Daftar Isi

Sinopsis .....	iii
Kata Pengantar .....	iv
Daftar Isi .....	vii
Daftar Gambar .....	ix
BAB I Berkenalan Dengan Blockchain.....	1
1.1 Pendahuluan .....	1
1.2 Motivasi di Balik Terciptanya Blockchain .....	2
1.2.1 Ketidakpercayaan Pada Otoritas Tunggal .....	2
1.2.2 Solusi Ketidakpercayaan Dengan Desentralisasi .....	4
1.2.3 Blockchain Sebagai Buku Besar Terdistribusi .....	5
1.3 Cara Kerja Blockchain.....	6
1.3.1 Cara Menghubungkan Antar Blok .....	9
1.3.2 Penambangan ( <i>Mining</i> ).....	12
1.3.4 <i>Broadcast Transaction</i> .....	13
1.3.5 Proses <i>Mining</i> .....	14
1.3.6 <i>Proof of Work</i> .....	18
1.4 Detail Teknis pada Blockchain .....	21
1.4.1 Tipe Node dalam Blockchain .....	22
1.4.2 <i>Merkle Tree</i> dan <i>Merkle Root</i> .....	24
1.4.3 Kegunaan <i>Merkle Tree</i> dan <i>Merkle Root</i> .....	26
BAB II Implementasi Blockchain Sendiri dengan Javascript .....	1
2.1 Setup Environment.....	1
2.1.1 Install Node.Js.....	1
2.1.2 Struktur Projek .....	6
2.1.3 Instal Modul SHA256 .....	13
2.2 Block Data Structure .....	15



2.3 Blockchain Object .....	16
2.3.1 <i>Constructor</i> .....	16
2.3.2 Membuat Block Baru .....	18
2.3.3 Mengambil Informasi Block Terbaru .....	20
2.3.4 Membuat Transaksi Baru .....	21
2.3.5 Hash Block .....	23
2.3.6 <i>Proof of Work</i> .....	25
2.3.7 Menjalankan Blockchain .....	29
2.4 Membuat Blockchain API di Javascript.....	36
2.4.1 <i>Setup Environment</i> .....	38
2.4.2 Membangun Blockchain API .....	40
2.4.3 Menjalankan API Blockchain .....	47
2.5 Desentralisasi Jaringan Blockchain di Javascript .....	57
2.5.1 Setup Environment.....	59
2.5.2 Menambahkan network Nodes ke Blockchain .....	64
2.5.3 Membuat End Points .....	65
2.5.4 Sinkronisasi Jaringan Blockchain .....	88
PENUTUP .....	112
DAFTAR PUSTAKA .....	113

## Daftar Gambar

Gambar 1. Jaringan Sentral dan Terdesentralisasi .....	6
Gambar 2. Setiap blockchain dimulai dengan genesis blok .....	7
Gambar 3. Proses Enkripsi dan Dekripsi .....	8
Gambar 4. Rantai Blok dengan Fungsi Hashing.....	11
Gambar 5. Broadcast ke Semua Komputer Penambang .....	13
Gambar 6. Tingkat Kesulitan Jaringan.....	15
Gambar 7. Penambahan Nonce .....	16
Gambar 8. Implementasi Mencari Nonce .....	17
Gambar 9. Konfirmasi Antar Block.....	20
Gambar 10. Isi Sebuah Block .....	21
Gambar 11. Jenis Node dalam Blockchain.....	22
Gambar 12. Merkle root berasal dari Merkle Tree .....	25
Gambar 13. Memvalidasi sebuah transaksi.....	26
Gambar 14. Website Node.js .....	3
Gambar 15. Pilihan File Download Node.js .....	4
Gambar 16. Instalasi Awal Node.js .....	5
Gambar 17. Verifikasi Instalasi node.js dan npm.....	5
Gambar 18. Membuat folder Blockchain.....	7
Gambar 19. Inisialisasi Proyek Blockchain .....	7
Gambar 20. Laman Download Visual Studio Code .....	8
Gambar 21. Menambahkan Folder ke Workspace .....	9
Gambar 22. Memilih Nama Folder Proyek Anda .....	9
Gambar 23. Membuka File Package.json.....	10
Gambar 24. Membuka Terminal Vs Code .....	10
Gambar 25. Terminal Vs Code .....	11
Gambar 26. Mengubah Mode Command Prompt.....	11
Gambar 27. Folder Proyek.....	12
Gambar 28. File baru di folder src dan test.....	13
Gambar 29. Instal Modul SHA256 .....	14
Gambar 30. Modul SHA256 Telah Ditambahkan.....	14
Gambar 31. Ekstensi Thunder Client VS Code .....	37
Gambar 32. Instalasi Thunder Client .....	37
Gambar 33. Thunder Client User Interface.....	38
Gambar 34. Instalasi Express.Js .....	39
Gambar 35. Menjalankan Server Node.Js.....	48

Gambar 36. GET Request Blockchain .....	48
Gambar 37. GET Mine Blockchain .....	50
Gambar 38. POST Transaction Blockchain .....	52
Gambar 39. Menjalankan 5 Node Server .....	79
Gambar 40. New Request ThunderBolt Client.....	80
Gambar 41. Menambahkan Node 2 pada Node 1 .....	81
Gambar 42. Response Penambahan Node Baru .....	81
Gambar 43. Menambahkan Node 3 pada Node 1 .....	82
Gambar 44. Menambahkan Node 4 pada Node 1 .....	82
Gambar 45. Response JSON .....	83
Gambar 46. Cek Jaringan Node 1 .....	83
Gambar 47. Menggunakan Broadcast Node .....	84
Gambar 48. Cek Isi Blockchain Node 1 .....	85
Gambar 49. Cek Isi Blockchain Node 2 .....	86
Gambar 50. Cek Isi Blockchain Node 3 .....	86
Gambar 51. Cek Isi Blockchain Node 4 .....	87
Gambar 52. Cek Isi Blockchain Node 5 .....	87
Gambar 53. Menambahkan jaringan ke dalam node 1.....	101
Gambar 54. Sukses Broadcast Jaringan Baru.....	102
Gambar 55. Proses Mining oleh Node 2.....	104
Gambar 56. Broadcast Transaksi Baru.....	106

# BAB I

## Berkenalan Dengan Blockchain

### 1.1 Pendahuluan

Teknologi blockchain menjadi sangat populer akhir-akhir ini bersamaan dengan bidang teknologi **big data** dan *artificial intelligence* (AI). Kolaborasi antara mereka juga telah diuji coba oleh para ilmuwan dari seluruh dunia. Hal ini terlihat dari banyaknya artikel yang dihasilkan oleh mereka dan berhasil terpublikasi pada jurnal-jurnal internasional bereputasi. Sebelum kita melihat jauh tentang teknologi hebat ini, mari kita perdalam lagi tentang apa itu blockchain.

Banyak orang sepertinya sudah mulai familiar dengan istilah bitcoin, ether, dogecoin, binance, cardano dan banyak lagi jenis *cryptocurrency*. Mereka menyebutnya sebagai portofolio investasi aset digital. Banyak orang yang penasaran dan mulai mencoba investasi di dunia *cryptocurrency*. Beberapa platform terkenal seperti indodax misalnya menawarkan kepada kita dapat melakukan trading mata uang digital tersebut. Kita dapat membeli *cryptocurrency* sebagai aset digital atau portofolio investasi. Banyak juga orang-orang yang berhasil meraup keuntungan ketika mereka melakukan investasi, namun banyak pula yang menjadi rugi

hingga ratusan juta. *Cryptocurrency* ini bersifat fluktuatif, nilai investasinya sangat cepat berubah, kadang naik kadang turun. Hal ini yang menjadikan adanya resiko yang tinggi bagi orang yang tidak pandai dalam mengatur strategi investasi mereka di *cryptocurrency*.

Dalam dunia informatika atau orang sering menyebutnya IT, teknologi blockchain terus berkembang dan dipelajari juga oleh para akademisi informatika. Blockchain kemudian banyak dikembangkan dan diintegrasikan dengan berbagai bidang seperti pada industri kesehatan, keuangan hingga bidang pangan. Blockchain telah menarik perhatian para penggiat teknologi dan masih terus berkembang hingga saat ini.

## **1.2 Motivasi di Balik Terciptanya Blockchain**

### **1.2.1 Ketidakpercayaan Pada Otoritas Tunggal**

Awal mula adanya teknologi ini adalah adanya rasa ketidakpercayaan terhadap otoritas tunggal. Sebagai contoh dalam kehidupan sehari-hari, kita percaya menitipkan uang pada bank atau percaya terhadap pengelolaan negara terhadap pemerintah. Kemudian pertanyaannya adalah bagaimana jika bank atau pemerintah tersebut korup? Bank atau Pemerintah dianggap sebagai otoritas tunggal dimana kita seolah terpaksa mempercayai

bahwa mereka tidak akan melakukan manipulasi data kita atau pencurian aset kita. Rasa ketidakpercayaan ini kemudian muncul sebagai ide dasar dibalik teknologi blockchain.

Contoh lain tentang rasa ketidakpercayaan lainnya adalah terkait isu yang beredar pada masyarakat saat pandemi covid-19 ini. Banyak isu beredar bahwa data-data pasien yang ada di rumah sakit kemudian diubah sehingga terkesan istilah “**dicovidkan**”. Hal ini membuat resah di masyarakat sehingga mereka takut untuk melakukan pengobatan di rumah sakit karena takut diberikan diagnosa palsu yaitu diberi label covid-19. Rumah sakit dalam hal ini dapat dikatakan sebagai otoritas tunggal dan mereka telah diragukan oleh rasa ketidakpercayaan masyarakat dalam penanganan covid-19.

Dalam dunia informatika juga bisa berlaku hal tersebut, misalkan anda memiliki sebuah aplikasi hebat yang sudah dipublikasikan di masyarakat. Banyak data-data privat yang harus dilindungi dan diamankan. Apakah anda yakin dengan penyedia layanan internet anda? atau apakah anda yakin dengan server hosting yang menyimpan data berharga anda? Apakah anda pernah mendengar pencurian data projek aplikasi yang kemudian dijual kembali atau maraknya pembajakan software. Penyedia layanan server biasanya menyewakan space hosting kepada kita kemudian kita

mengupload projek kita kesana, dan kita hanya terpaksa percaya saja walau sangat besar kemungkinan data kita disalah gunakan. Nah, dalam hal ini penyedia layanan server juga bisa dikatakan sebagai otoritas tunggal yang diragukan oleh rasa ketidakpercayaan orang-orang informatika saat mengupload **aplikasi mereka kesana.**

### **1.2.2 Solusi Ketidakpercayaan Dengan Desentralisasi**

Saat banyak orang sudah mulai ragu atau munculnya ketidakpercayaan terhadap otoritas tunggal, atau bingung harus percaya terhadap siapa maka munculah konsep desentralisasi. Agar anda lebih memahami konsep desentralisasi ini kami akan memberikan contoh sederhana dari kehidupan sehari-hari. Sebagai contoh dalam suatu perkumpulan keluarga diadakannya pengumpulan uang kas bersama. Aturan dalam pengumpulan uang kas ini menggunakan konsep desentralisasi. Hal ini dilakukan karena terdapat keluarga jauh yang turut pula mengikuti pengumpulan uang kas ini. Aturan ini dilakukan dengan cara mencatat semua hasil kas yang telah disetorkan dalam jurnal buku besar. Misalnya terdapat 10 kepala keluarga yang ikut, ada yang dari dalam dan luar kota. Pengumpulan dilakukan setiap akhir bulan. Saat dilakukan pengumpulan uang kas, setiap kepala keluarga memiliki jurnal buku besarnya masing-masing kemudian

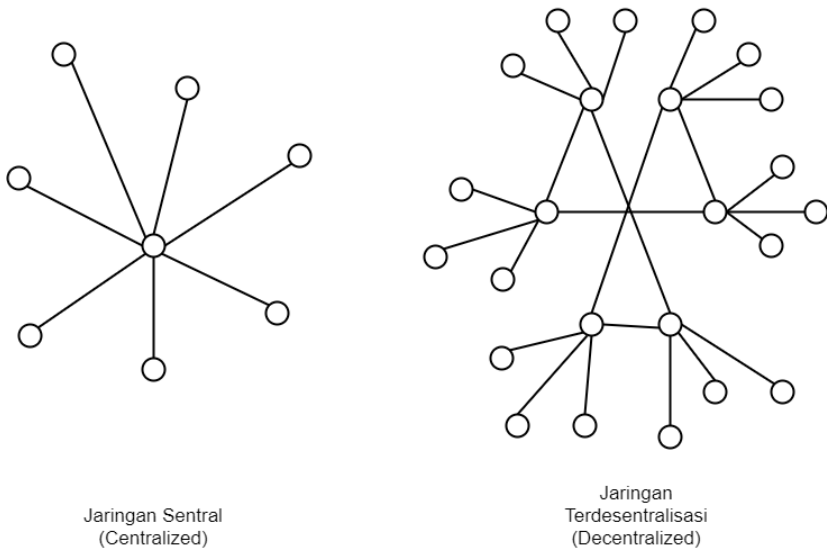
mencatatnya. Jadi setiap kali ada pengumpulan uang kas, maka setiap uang yang masuk dicatat secara bersama-sama. Dengan cara ini setiap kepala keluarga memiliki salinan catatan uang kas yang berhasil dikumpulkan, sehingga ketika ada satu atau dua anggota keluarga yang berupaya mengubah catatan hasil uang kas maka akan diabaikan. Kumpulan keluarga itu akan melakukan konsensus yaitu mencoba melakukan validasi dengan cara bertanya kepada semua anggota keluarga yang ikut mencatat. Catatan yang dianggap benar adalah catatan yang sama dimiliki oleh mayoritas. Misalkan dari 10 orang ada 2 orang yang mencoba merubahnya, namun 8 orang juga memiliki salinan yang asli. Dari sini ke dua orang tersebut dapat dengan mudah diidentifikasi sebagai orang yang berusaha mengubah data sehingga data akan tetap terjaga dan aman.

### **1.2.3 Blockchain Sebagai Buku Besar Terdistribusi**

Berdasarkan konsep desentralisasi yang telah dibahas sebelumnya, kita sudah sedikit memahami bagaimana konsep desentralisasi itu dijalankan. Setiap orang akan melakukan validasi data ketika ada ketidaksesuaian dengan data mayoritas dengan konsep konsensus. Teknologi Blockchain juga menggunakan konsep ini ketika melakukan penyimpanan datanya. Blockchain dapat dikatakan sebagai konsep jaringan desentralisasi atau *peer*



to peer network yang berusaha menyimpan data transaksi dalam bentuk jurnal buku besar (ledger) ke dalam blok-blok data dimana data tersebut didistribusikan kepada semua anggota jaringan. Ilustrasi jaringan terdesentralisasi ini dapat dilihat pada Gambar 1.

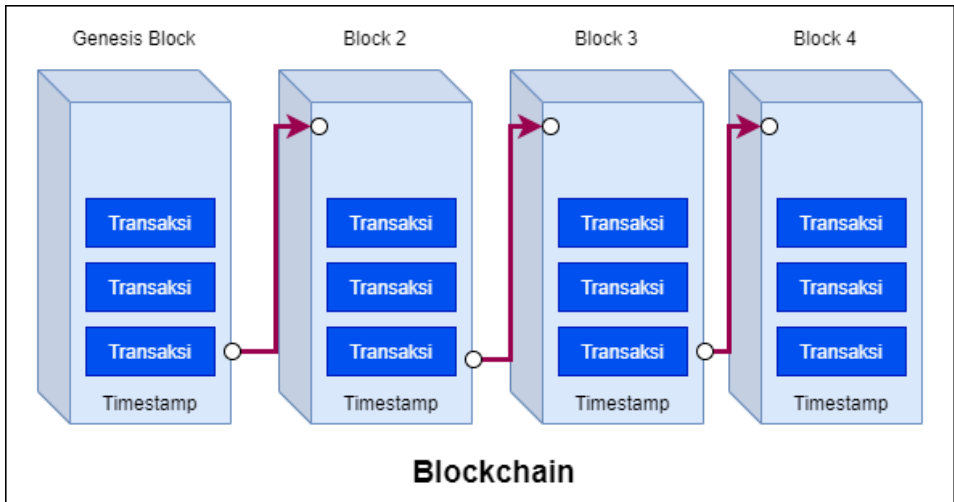


Gambar 1. Jaringan Sentral dan Terdesentralisasi

### 1.3 Cara Kerja Blockchain

Secara teknis, blockchain merupakan kumpulan block yang berisi daftar transaksi yang saling terhubung satu sama lain dengan konsep kriptografi. Block pertama dalam blockchain disebut dengan genesis block. *Genesis* blok merupakan blok istimewa, ia disebut

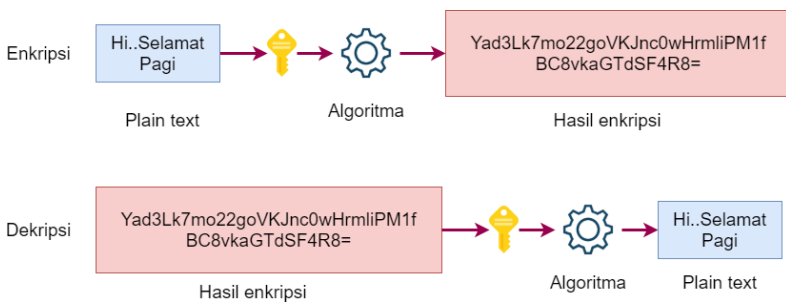
dengan blok ke-0 dan merupakan blok pertama yang pernah ditambang (*mining*) oleh penciptanya. *Genesis* blok menjadi dasar untuk blok-blok transaksi selanjutnya. Untuk lebih memperjelas tentang block-block tersebut, anda dapat melihat pada Gambar 2



Gambar 2. Setiap blockchain dimulai dengan genesis blok

Jika anda belum memahami apa itu kriptografi, kami akan memberikan sedikit ulasan terkait apa itu kriptografi dalam buku ini. Kriptografi merupakan hal wajib yang harus dipahami ketika kita hendak belajar teknologi blockchain. Secara sederhana kriptografi merupakan sebuah cara untuk mengamankan sebuah data atau informasi agar tetap terjaga integritasnya. Kriptografi identik dengan proses enkripsi dan dekripsi. Contoh yang paling umum digunakan adalah ketika anda menggunakan aplikasi pesan instan

seperti whatsapp, messenger, telegram dan sebagainya. ketika anda mengirimkan pesan kepada teman anda, maka dibalik layar yang terjadi adalah adanya proses enkripsi pesan tersebut saat dikirimkan dan kemudian ada proses dekripsi saat pesan itu diterima. Untuk mempermudah anda dalam memahami perbedaan antara enkripsi dan dekripsi dapat dilihat pada Gambar 3.



Gambar 3. Proses Enkripsi dan Dekripsi

Berdasarkan Gambar 3, teks “Hi..Selamat Pagi” yang merupakan sebuah plaintext kemudian dapat diubah kedalam bentuk ciphertext melalui algoritma enkripsi. Plaintext yang awalnya mudah dibaca oleh kita, selanjutnya diubah dalam bentuk ciphertext yang mana akan sulit ditebak. Dalam pesan instan seperti whatsapp pengaplikasian tersebut digunakan dimana pesan yang kita ketik saat diaplikasi masih berformat plaintext sedangkan saat dikirimkan ke orang lain, maka dienkripsi agar pesan kita bersifat rahasia yang artinya adalah hanya kita dan penerima saja yang dapat membacanya. Si penerima menerima *plaintext* kembali setelah

adanya proses dekripsi dimana algoritma akan merubah *ciphertext* menjadi *plaintext* kembali.

### 1.3.1 Cara Menghubungkan Antar Blok

Sebelumnya kita telah memahami secara umum bagaimana sebuah pesan instan melalui tahap enkripsi dan dekripsi agar pesan tersebut dapat selalu terjaga keamanan datanya. Nah, sebelum melanjutkan bagaimana sebuah blok dengan blok lainnya terhubung dengan konsep '*chain*' ini kita harus lebih memahami konsep kunci dalam blockchain yaitu '*hashing*'. Fungsi hashing merupakan salah satu cara untuk merubah bentuk data yang awalnya tetap menjadi berbeda. Ketika sebuah string diubah salah satu karakternya, maka nilai hash yang dihasilkan akan benar-benar berbeda dari sebelumnya. Sebagai contoh adalah pada Gambar 1.3 sebuah string "Hi..Selamat Pagi" berubah bentuk dalam format hash yang terlihat sangat berbeda. Dari sini saja anda akan sulit menyangka bukan bahwa arti sebenarnya dari string tersebut sebetulnya adalah "Hi..Selamat Pagi".

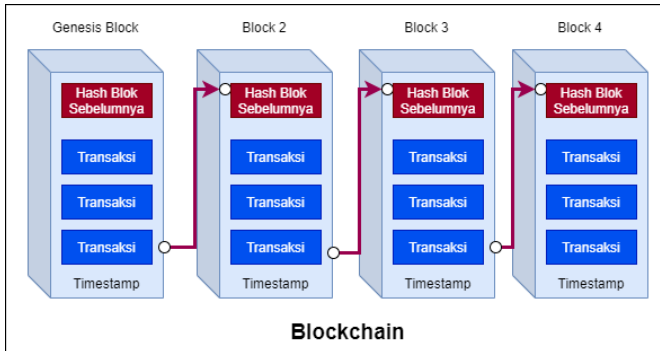
Berikut ini adalah ciri-ciri dari fungsi hash:

- Bersifat *deterministik* - pesan yang sama akan selalu menghasilkan nilai hash yang sama, artinya berapa kali anda

melakukan hash pada suatu string misalnya, maka hasilnya akan tetap sama.

- Bersifat satu arah (*one way process*) - pesan yang telah dilakukan hashing, pesan tersebut akan sulit dikembalikan ke pesan aslinya secara komputasi.
- Tahan benturan (*collision resistant*) - akan sulit menemukan dua pesan dengan hasil hash yang sama.

Setelah anda memahami bagaimana konsep hash ini, sekarang adalah masa dimana kita akan membahas bagaimana blok di blockchain dihubungkan dengan rantai (*chain*). Konten dalam setiap blok di-*hash* dan kemudian disimpan pada blok berikutnya (Gambar 4). Dengan cara tersebut, jika ada transaksi blok yang diubah akan membatalkan hash dari blok saat ini. Hal ini terjadi karena ketika ada transaksi baru yang akan ditambahkan ke blok berikutnya akan selalu terdapat validasi terkait blok hash yang disimpan sebelumnya. Untuk mempermudah pemahaman anda lihatlah Gambar 4.



*Gambar 4. Rantai Blok dengan Fungsi Hashing*

Berdasarkan Gambar 4, dalam satu waktu tertentu (*timestamp*) dapat terjadi lebih dari satu transaksi. Semua transaksi kemudian di hashing dan disimpan pada blok berikutnya. Penyimpanan ini bertujuan untuk menjaga integritas dari data transaksi yang terjadi pada blok sebelumnya. Ketika ada upaya modifikasi pada sebuah blok, maka hash dalam blok tersebut akan bersifat tidak valid, dan ini juga menyebabkan kerusakan pada hash blok yang lainnya di blockchain. Jika seorang peretas ingin merubah satu saja data transaksi di blok, maka agar upayanya tidak diketahui harus dilakukan juga modifikasi pada seluruh blok yang ada di blockchain. Ditambah lagi dengan dia harus melakukan sinkronisasi terhadap seluruh komputer lain yang ada di jaringan. Jika dalam sebuah blockchain terdapat 10.000 komputer, maka ia pun harus merubah pada keseluruhan komputer tersebut. Hal ini yang menjadikan upaya mengubah data dalam blockchain menjadi sangat sulit.

Hingga sampai tahap ini, anda telah memiliki gambaran terkait bagaimana blockchain bekerja dan bagaimana setiap blok dalam blockchain terhubung dengan konsep hashing. Selanjutnya anda akan belajar terkait penambangan (*mining*) yang merupakan topik penting dalam blockchain.

### **1.3.2 Penambangan (*Mining*)**

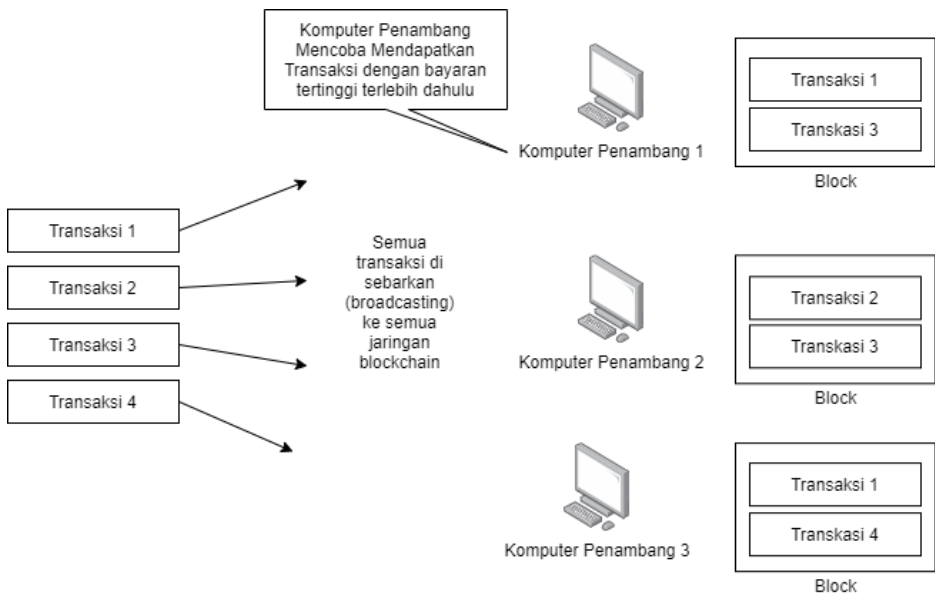
Kata mining menjadi sangat populer beberapa tahun terakhir ini, kita sering mendengar seseorang menjadi kaya ketika melakukan penambangan bitcoin. Sebuah mata uang digital atau dikenal dengan uang crypto (*crypto currency*). Sebetulnya bagaimana sih proses penambangan ini, apakah mirip dengan proses penambangan bahan logam seperti emas? tentunya tidak ya, karena proses ini dilakukan secara digital.

Penambangan (*mining*) sebetulnya merupakan upaya penambahan blok ke dalam blockchain. Dalam blockchain terkenal seperti *Bitcoin* atau *Ethereum*, ada berbagai jenis komputer yang dikenal sebagai node. Komputer di blockchain yang mampu menambahkan blok ke dalam blockchain dikenal sebagai komputer penambang atau node penambang (*miners*). Untuk beberapa jenis node lain, akan dibahas pada sub bab berikutnya, kita akan terlebih dahulu fokus dengan node penambang ini. Kita sekarang tahu kalau *miners* adalah komputer yang mampu melakukan

penambahan blok pada blockchain, lantas bagaimana sih proses *mining* ini?

### 1.3.4 Broadcast Transaction

Ketika ada transaksi, yang sebenarnya terjadi adalah transaksi-transaksi tersebut disiarkan pada seluruh jaringan blockchain. Semua komputer penambang (*miners*) dapat menerimanya namun tentunya pada waktu yang berbeda-beda. Sebagai node yang dapat menerima transaksi, ia akan mencoba menambahkannya ke dalam blok. Setiap node bebas untuk memasukan transaksi apapun yang mereka inginkan ke dalam satu blok.



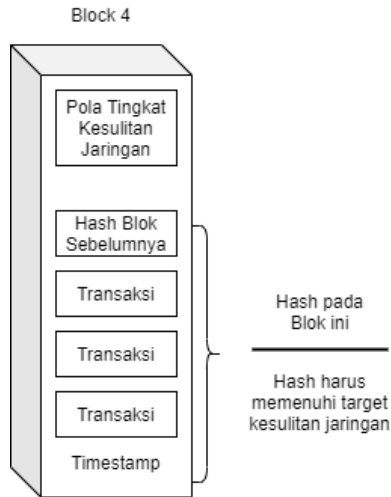
Gambar 5. Broadcast ke Semua Komputer Penambang



Dalam realitanya, tidak sembarangan transaksi dapat ditambahkan dalam suatu blok. Hal ini dipengaruhi oleh berbagai faktor seperti adanya biaya transaksi, ukuran data transaksi, urutan kedatangan dan faktor lainnya. Pada tahap ini setiap transaksi yang telah ditambahkan pada suatu blok masih masuk sebagai transaksi yang bersifat pending atau belum dikonfirmasi oleh semua komputer yang ada di jaringan blockchain. Masalahnya adalah dengan banyaknya komputer penambang di luar sana, siapakah yang dapat menambahkan blok ke blockchain dahulu?

### **1.3.5 Proses *Mining***

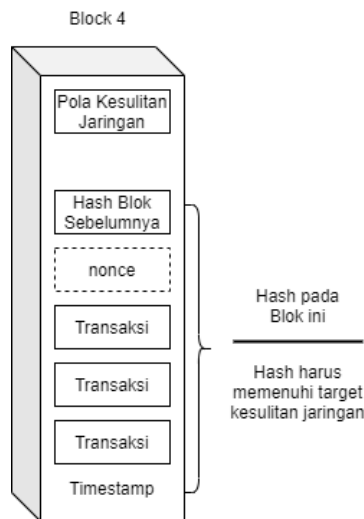
Untuk memperlambat laju penambahan blok ke dalam blockchain, diperlukan sebuah protokol yang namanya adalah konsensus (*consensus*) yang merupakan kesepakatan bersama untuk menentukan target kesulitan jaringan. Setiap node penambang ketika ingin menambahkan blok baru ke dalam blockchain, maka dia harus terlebih dahulu harus melakukan hashing terhadap isi bloknnya dan harus memenuhi kriteria yang ditentukan oleh target kesulitan jaringan.



*Gambar 6. Tingkat Kesulitan Jaringan*

Target kesulitan jaringan ini misalnya, setiap hash harus dimulai dengan lima angka 0 di depan. Oleh karena itu node penambang harus berulang kali mencoba membuat kode hashing dari isi blok dengan memastikan terdapat lima angka 0 di kode hash mereka. Perlu diketahui semakin banyak penambang yang bergabung dengan jaringan maka tingkat kesulitannya meningkat. Sebagai contoh adalah jika saat ada 50 penambang kode hashnya hanya cukup lima bilangan 0 di depan kode hash, nah jika terdapat 400 penambang maka kode hashnya wajib memiliki sepuluh 0 di depan. Ada permasalahan yang ditimbulkan jika kesulitan jaringan tersebut bersifat konsisten yaitu setelah ditemukannya kode hash oleh node penambang, maka kode hash itu dapat terus dipakai untuk

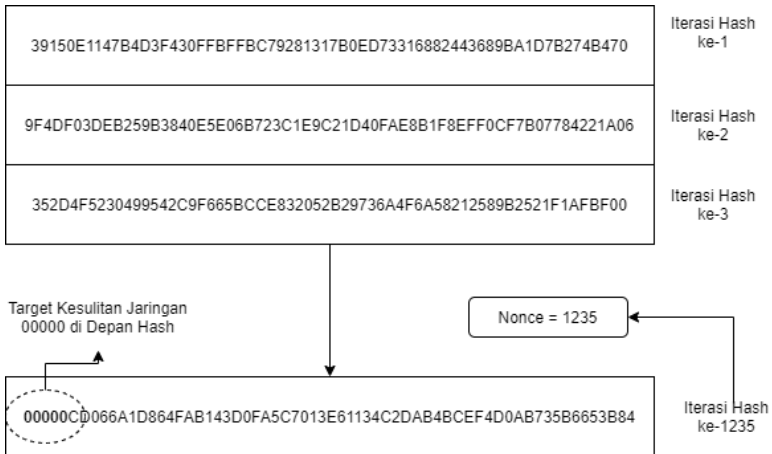
melakukan *mining*. Jadi, perlu dibuat adanya aturan baru yaitu mencari nilai *nonce* (*number only used once*). *Nonce* ini akan ditambahkan ke dalam blok hash yang bertujuan untuk tetap memastikan kesulitan jaringan tetap terjaga. Dengan adanya *nonce*, setiap node penambang selain menghasilkan hash baru untuk blok berikutnya, ia juga harus menghasilkan nilai *nonce*.



Gambar 7. Penambahan Nonce

Nilai *nonce* ini dapat dihasilkan ketika node penambang melakukan hashing dari isi blok. Secara sederhana dapat digambarkan sebagai berikut, jika tingkat kesulitan jaringan adalah harus membuat hash dengan angka 00000 (lima digit angka nol) di depan hashnya maka node penambang akan berusaha dengan komputasi melakukan hash berulang kali hingga berhasil mendapatkan *hash*

dengan 00000 di depannya. Anggap saja sebuah node penambang baru berhasil membuat *hash* dengan lima angka nol di depan pada perulangan ke-1235, maka angka 1235 adalah nilai *nonce* nya. Nah angka ini yang akan disertakan pada hash yang disimpan pada blok berikutnya. Jadi, kesulitan jaringan ini akan tetap terjaga dan terpenuhi. Proses ini terus berlanjut dan berlaku untuk penambang-penambang berikutnya. Ilustrasi pencarian *nonce* ini dapat dilihat pada Gambar 8.



Gambar 8. Implementasi Mencari Nonce

Penambahan blok ke dalam blockchain atau proses mining ini terjadi jika ada node penambang yang berhasil untuk pertama kali memecahkan tingkat kesulitan jaringan berhasil menemukan *nonce* untuk hash blok berikutnya. *Node* penambang yang berhasil ini akan menerima rewards dari jaringan blockchain atas usahanya

memecahkan tingkat kesulitan jaringan dalam bentuk *cryptocurrency*. Misalnya jika blockchainya adalah etereum maka dia akan mendapatkan koin *ether*.

Informasi tentang berhasilnya suatu node penambang dalam memecahkan kesulitan tingkat jaringan ini akan di siarkan ke semua anggota jaringan blockchain dan semua anggota ini akan melakukan validasi apakah benar blok baru tersebut benar bertambah atau tidak. Seketika itu penambang-penambang lain akan berhenti melakukan penambangan dan mencari transaksi lain yang dapat ditambang.

Sebagai informasi bahwa dalam penambangan di Blockchain Bitcoin, seorang penambang pada awalnya mendapatkan 50 BTC dan terus dibagi dua setiap 210.000 blok. Saat buku ini ditulis rewards atas penambangan Bitcoin adalah 6.25 BTC. Sedangkan untuk Ethereum saat ini adalah 2 ETH.

### **1.3.6 Proof of Work**

Proses penambahan blok baru pada blockchain bisa disebut dengan *Proof of Work* (PoW). Kita tahu bahwa untuk melakukan mining ini dapat dikatakan sulit karena membutuhkan waktu yang banyak untuk memecahkan masalah hashing dan menemukan

*nonce*. Namun untuk memverifikasi nya dapat dikatakan dengan mudah karena setiap hash saling terkait satu sama lainnya.

*Proof of Work* membutuhkan sumber daya komputasi yang luar biasa terutama pada kebutuhan GPU (*Graphics Processing Unit*). Proses ini juga membutuhkan jumlah listrik yang besar karena penambang melakukan pekerjaan yang sama berulang kali yaitu ia berusaha menemukan *nonce* untuk memenuhi kesulitan jaringan untuk blok tersebut.

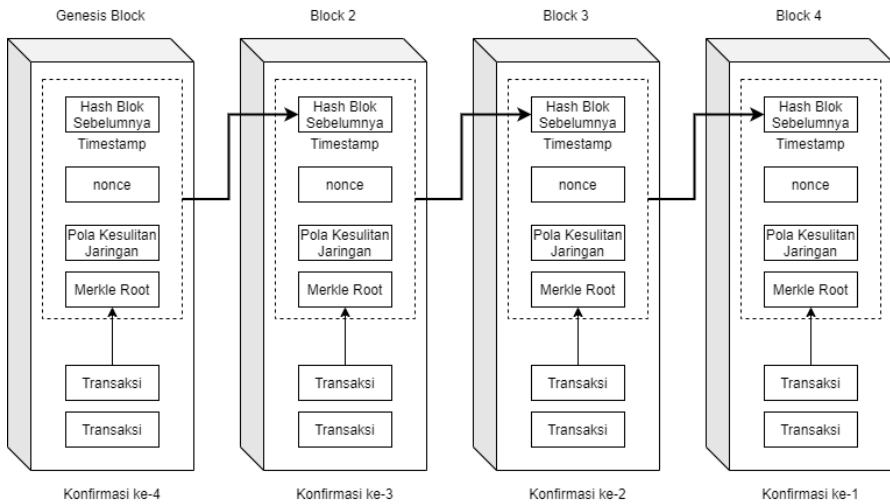
GPU yang kuat diperlukan dalam proses *mining* karena ia dapat mengeksekusi 3200 instruksi 32-bit per jam sedangkan CPU hanya 4 instruksi 32-bit perjam. GPU ini lebih unggul dalam manipulasi sederhana pada kumpulan data yang besar dari pada CPU.

Alasan kenapa banyak orang berinvestasi sebagai *node* penambang adalah karena ia akan mendapatkan rewards atas kerja kerasnya memecahkan masalah tingkat kesulitan jaringan saat blok ditambahkan.

### 1.3.7 Blockchain bersifat *Immutability*

Data yang telah berhasil masuk pada blockchain akan bersifat tidak dapat dihapus dan diubah. Seseorang yang ingin merubah data tersebut, diharuskan pula merubah seluruh blok yang ada dalam blockchain sekaligus mensinkronkan ke semua komputer yang ada di jaringan ini. hal ini tentunya membutuhkan kecepatan dan daya

komputasi yang besar. Setiap blok dalam blockchain akan saling mengkonfirmasi satu sama lainnya, jika ada satu saja blok yang diubah maka akan segera terdeteksi karena mereka saling terhubung dengan fungsi hash yang mana setiap blok menyimpan hash blok transaksi sebelumnya.

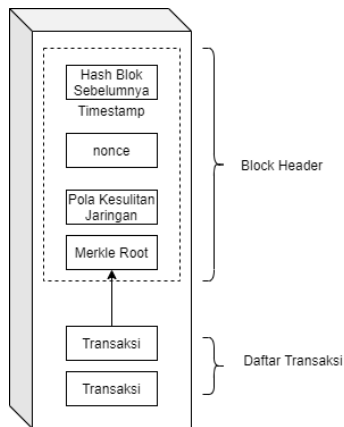


*Gambar 9. Konfirmasi Antar Blok*

Berdasarkan Gambar 9, setiap blok akan mengkonfirmasi apakah hash yang ia miliki benar milik blok sebelumnya. Setiap blok mengkonfirmasi hingga pada *Genesis* blok, sehingga ketika ada perubahan pada salah satu blok akan mudah diidentifikasi pada jaringan blockchain ini.

## 1.4 Detail Teknis pada Blockchain

Pada bagian sebelumnya kita telah bersama-sama mempelajari bahwa sebuah blok berisi *nonce*, *timestamp*, dan beberapa transaksi. Hal itu merupakan pengetahuan yang telah disederhanakan agar anda lebih memahami konsep blockchain.



*Gambar 10. Isi Sebuah Block*

Dalam implementasi sesungguhnya, sebuah blok itu terdiri dari:

- Header Block
- Daftar Transaksi

Header Block sendiri sebetulnya juga terdiri dari beberapa komponen antara lain adalah:

- Hash dari block sebelumnya
- Timestamp

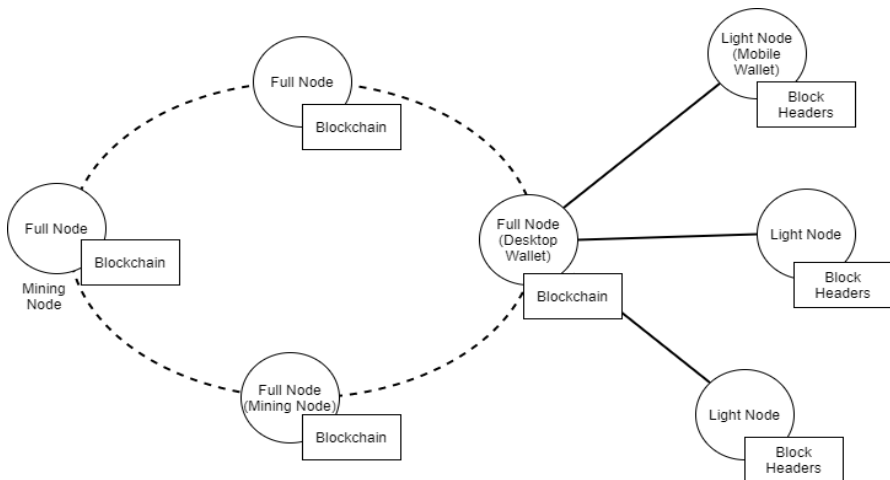


- Merkle Root
- Nonce
- Target Kesulitan Jaringan

Header Block memiliki merkle root, dan bukan transaksi dan secara kolektif transaksi dapat direpresentasikan sebagai merkle root, detail dari merkel root akan diperjelas pada bagian berikutnya.

### 1.4.1 Tipe Node dalam Blockchain

Sebelum lebih lanjut membahas *merkle root*, kita harus terlebih dahulu memahami jenis node dalam jaringan blockchain. Berikut ini adalah ilustrasi jenis node yang umumnya ada di jaringan blockchain.



Gambar 11. Jenis Node dalam Blockchain

Pada bahasan sebelumnya, setiap komputer yang terhubung dengan blockchain disebut dengan *nodes*. Kita juga telah mempelajari salah satu jenis nodes yaitu node penambang dan apa tanggung jawabnya. Ia bekerja untuk menambahkan blok baru pada blockchain dan mencari *nonce* untuk memenuhi target kesulitan jaringan. Node penambang dikenal juga sebagai *full node*.

*Full node* tidak selalu melakukan penambangan, namun *node* penambang harus menjadi *full node*. Tujuan utama dari *full node* adalah memastikan integritas blockchain. *Full node* penambang (*miners*) akan dihargai berupa rewards *coin crypto* saat berhasil menambahkan blok baru ke dalam jaringan blockchain. Contoh jenis *full node* lainnya adalah desktop wallet yang memungkinkan pengguna untuk melakukan transaksi dengan *cryptocurrency*.

Setiap *full node* memiliki salinan seluruh blockchain. *Full node* juga melakukan validasi setiap blok dan transaksi disajikan untuk itu. Selain *full node* juga terdapat *light node*. *Light node* membantu verifikasi menggunakan metode yang disebut dengan SPV (*Simplified payment Verification*). SPV memungkinkan sebuah *node* untuk memverifikasi jika suatu transaksi sudah termasuk dalam satu blok, tanpa perlu mendownload keseluruhan rantai blok. Dengan SPV *light node* terhubung ke *full node* dan mengirimkan transaksi ke ke *full node* untuk verifikasi. *Light node* hanya perlu menyimpan header block dari semua block di blockchain. Sebuah

contoh light node adalah wallet mobile, dimana pengguna dapat melakukan transaksi di perangkat seluler.

Secara ringkas, jenis node yang terdapat pada blockchain adalah sebagai berikut:

**Full Node:**

- Menyalin semua data yang ada di blockchain secara lengkap
- Mampu memverifikasi semua transaksi dari awal
- Memverifikasi block yang baru dibuat dan menambahkannya ke blockchain

**Mining Node** (Harus menjadi Full Node):

- Melakukan Penambangan dengan mencari nonce

**Light Node:**

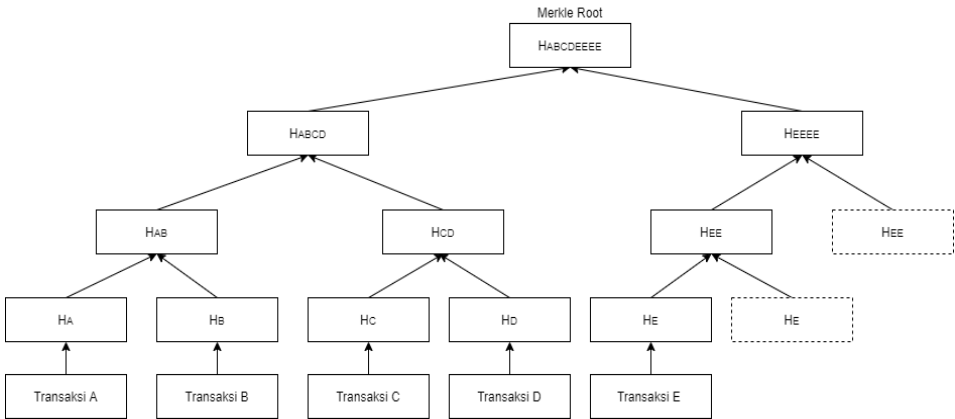
- Mempertahankan header blockchain
- Menggunakan SPV untuk memverifikasi apakah sebuah block dalam blockchain bersifat valid atau tidak.

### 1.4.2 Merkle Tree dan Merkle Root

Kumpulan transaksi dalam satu blok disimpan sebagai merkle tree.

*Merkle tree* merupakan sebuah struktur data berjenis *tree* dimana setiap simpul daun (*leaf node*) adalah *hash* dari transaksi dan

setiap *leaf node* adalah hash kriptografi dari simpul anak (*child node*).



Gambar 12. Merkle root berasal dari Merkle Tree

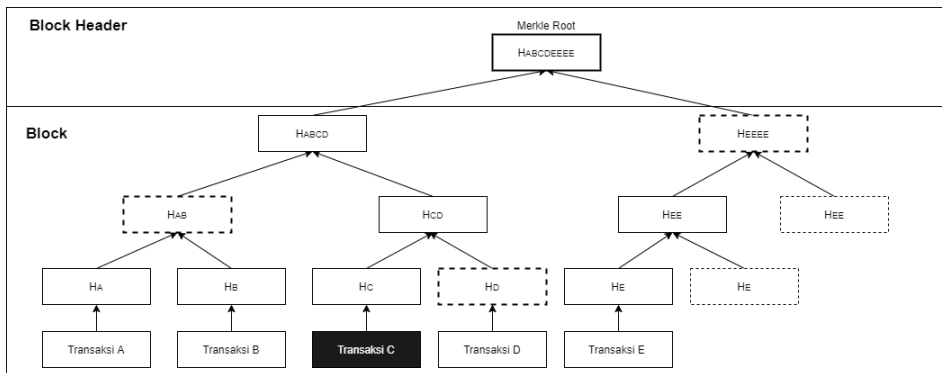
Seperti yang terlihat pada Gambar 12, semua transaksi terlebih dahulu di-hash bersama dengan hash dari node lain. Misalnya saja, hash dari transaksi A yaitu HA digabungkan dengan hash transaksi B yaitu HB yang kemudian diturunkan menjadi HAB. Proses ini terus berulang hingga menghasilkan satu hash saja atau merkle root. Seandainya sebuah hash transaksi tidak memiliki pasangan simpul, maka dia akan dipasangkan dengan dirinya sendiri (HEEEE).

*Merkle root* disimpan pada *header block*, dan sisa transaksinya disimpan pada blok sebagai *merkle tree*. Pada bahasan sebelumnya, telah dibahas bahwa *full node* mengunduh dan menyalin semua blockchain, selain itu terdapat juga jenis *node* lain

yaitu *light node* yang hanya mengunduh *header* blockchain. Karena *light node* tidak mengunduh semua data pada jaringan blockchain, dia mudah dirawat dan dijalankan. Dengan metode SPV, *light node* dapat meminta *full node* untuk memverifikasi suatu transaksi. Contoh *light node* adalah *cryptographic wallet*.

### 1.4.3 Kegunaan *Merkle Tree* dan *Merkle Root*

Dengan menyimpan *merkle root* pada *header block* dan transaksi sebagai *merkle tree* di blok, *light node* dapat dengan mudah memverifikasi apakah transaksi ini milik blok tertentu. Cara kerjanya adalah sebagai berikut.



Gambar 13. Memvalidasi sebuah transaksi

Misalkan *light node* ingin memverifikasi bahwa transaksi C ada di sebuah blok tertentu:

- *Light node* menanyakan pada *full node* untuk hash berikut: HD, HAB, dan HEEEEE
- Karena *light node* dapat menghitung HC, maka node tersebut dapat menghitung HCD dengan HD yang disediakan.
- Dengan HAB yang disediakan, sekarang dapat menghitung HABCD
- Dengan HEEEEE yang disediakan, sekarang dapat menghitung HABCDEEEEE (*merkle root*)
- Karena *light node* memiliki *merkle root* dari block, ia sekarang dapat memeriksa apakah kedua *merkle root* itu cocok. Jika cocok maka transaksi diverifikasi.

Dalam contoh sederhana tersebut, untuk memverifikasi satu transaksi dari lima transaksi, hanya memerlukan tiga *hash* yang diambil oleh *full node*. Secara matematis, untuk  $n$  transaksi dalam satu block, diperlukan  $\log_2 n$  untuk memverifikasi bahwa suatu transaksi ada dalam satu block. Misalnya jika ada 1024 transaksi dalam satu blok, *light node* hanya perlu meminta 10 *hash* untuk memverifikasi keberadaan transaksi di block.

## **BAB II**

### **Implementasi Blockchain Sendiri dengan Javascript**

#### **2.1 Setup Environment**

Sebelum kita melakukan implementasi untuk membuat demo blockchain sendiri dengan javascript, kita harus melakukan setup environment antara lain menginstal node.js dan modul pendukung dalam proyek ini.

##### **2.1.1 Install Node.Js**

Node.js merupakan salah satu perkembangan teknologi yang menjadikan bahasa javascript tidak hanya berjalan pada sisi browser untuk tampilan web namun juga dapat berjalan disisi server. Node.js merupakan sebuah runtime environment untuk javascript. Sifatnya terbuka dan dapat dipergunakan pada berbagai sistem operasi seperti Windows, Macintosh dan Linux. Node.js menjalankan V8 Javascript Engine (Inti dari Google Chrome) yang memiliki performa tinggi. Node.js memiliki banyak sekali modul yang dapat kita gunakan untuk mempercepat pembangunan aplikasi website.

Node.Js memiliki berbagai kelebihan yang menjadikannya populer digunakan dikalangan

developer. Berikut ini adalah kelebihan yang ditawarkan oleh Node.Js.

- **Asynchronous & Event-driven**

Semua API dari Node.js bersifat *asynchronous*, artinya tidak memblokir proses lain sembari menunggu satu proses selesai. Server Node.js akan melanjutkan ke ke pemanggilan API berikutnya lalu memanfaatkan mekanisme event notification untuk mendapatkan respon dari panggilan API sebelumnya.

- **Very Fast**

Eksekusi kode dengan Node.js sangat cepat karena berjalan pada V8 JavaScript Engine dari Google Chrome.

- **Single Threaded but Highly Scalable**

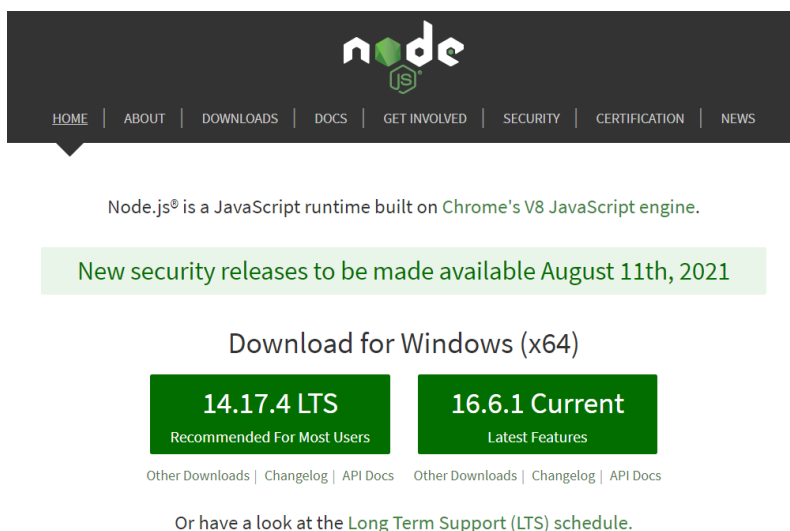
Node.js menggunakan model *single thread* dengan *event looping*. Mekanisme ini membantu server untuk merespon secara *asynchronous* dan menjadikan server lebih scalable dibandingkan server tradisional yang menggunakan banyak thread untuk menangani permintaan.

Dalam proyek ini kita akan memanfaatkan kehebatan Node.js untuk membuat demo aplikasi blockchain.



Banyak sekali modul-modul baik yang sifatnya telah tersedia atau modul eksternal yang dapat mendukung pembuatan demo blockchain dengan javascript ini lebih cepat. Berikut adalah langkah-langkah menginstall Node.js.

1. Untuk menginstal Node.js anda dapat berkunjung ke situs resminya yaitu <http://www.nodejs.org>



*Gambar 14. Website Node.js*

2. Pergi menuju menu Downloads lalu pilih aplikasi Node.js sesuai dengan Sistem Operasi kalian. Pada buku ini saya menggunakan Sistem Operasi Windows 64-bit.

## Downloads

Latest LTS Version: 14.17.4 (includes npm 6.14.14)

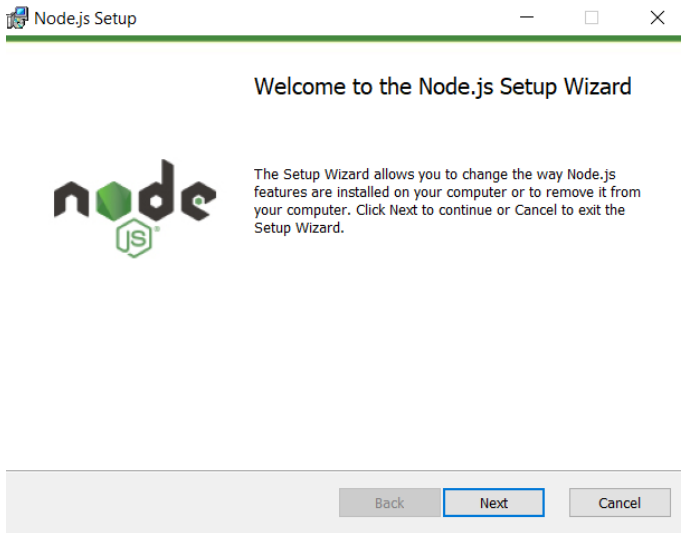
Download the Node.js source code or a pre-built installer for your platform, and start developing today.

The screenshot shows the Node.js Downloads page. It is divided into two main sections: 'LTS Recommended For Most Users' and 'Current Latest Features'. Under 'LTS', there are three options: 'Windows Installer' (node-v14.17.4-x64.msi), 'macOS Installer' (node-v14.17.4.pkg), and 'Source Code' (node-v14.17.4.tar.gz). Under 'Current', there are three options: 'Windows Installer' (node-v14.17.4-x64.msi), 'macOS Installer' (node-v14.17.4.pkg), and 'Source Code' (node-v14.17.4.tar.gz). Below the 'LTS' section, there is a list of download options: Windows Installer (.msi), Windows Binary (.zip), macOS Installer (.pkg), macOS Binary (.tar.gz), Linux Binaries (x64), Linux Binaries (ARM), and Source Code. To the right of this list is a table with columns for 32-bit and 64-bit, and rows for ARMv7 and ARMv8.

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v14.17.4.tar.gz	

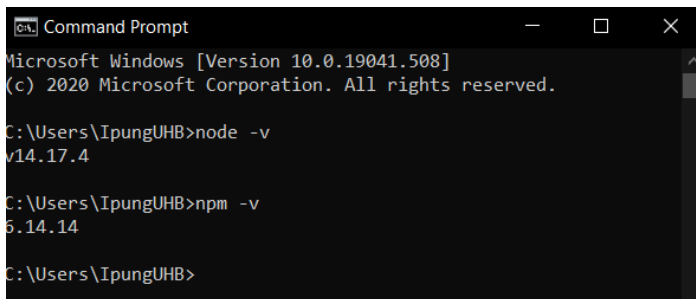
*Gambar 15. Pilihan File Download Node.js*

3. Setelah anda memilih file installer sesuai dengan sistem operasi dan bit-nya, maka file akan terdownload. Pada windows sendiri akan masuk pada folder downloads.
4. Untuk menginstal Node.js dapat dilakukan klik dua kali atau klik kanan pilih **instal**.
5. Pilih **Next** untuk melanjutkan instalasi.



*Gambar 16. Instalasi Awal Node.js*

6. Pilih **next** hingga instalasi Node.Js selesai.
7. Lakukan verifikasi apakah node.js sudah terinstal pada komputer anda dengan cara membuka *command prompt* atau *console* pada komputer anda lalu ketikkan perintah **node -v** dan **npm -v**.



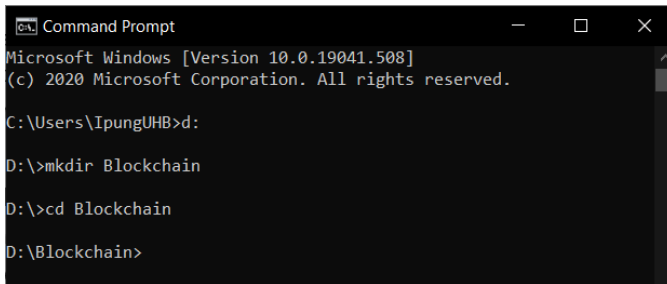
*Gambar 17. Verifikasi Instalasi node.js dan npm*

Jika anda berhasil melakukan instalasi Node.js maka akan muncul versi Node.js sesuai dengan apa yang anda instal. Pada saat buku ini ditulis, saya menggunakan versi 14.17.4 dan npm versi 6.14.14.

### 2.1.2 Struktur Projek

Kita akan membuat struktur projek aplikasi blockchain kita dengan javascript. Setelah berhasil menginstal Node.js, kita akan membuat struktur proyek dengan mode runtime Node.js. Ikuti langkah berikut untuk membuat struktur proyeknya.

1. Bukalah terminal atau *command prompt* pada komputer anda.
2. Arahkan pada folder lokasi kita akan membuat proyek blockchain. Sebagai contoh saya akan menyimpan file proyek pada Drive D. Pada command prompt saya ketikan perintah **:d** untuk beralih pada direktori disk D. Kemudian saya buat folder baru dengan nama "Blockchain" dengan mengetikan perintah **mkdir Blockchain**. Untuk masuk pada folder Blockchain, ketikan perintah **cd Blockchain**.



```
Command Prompt
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\IpungUHB>d:

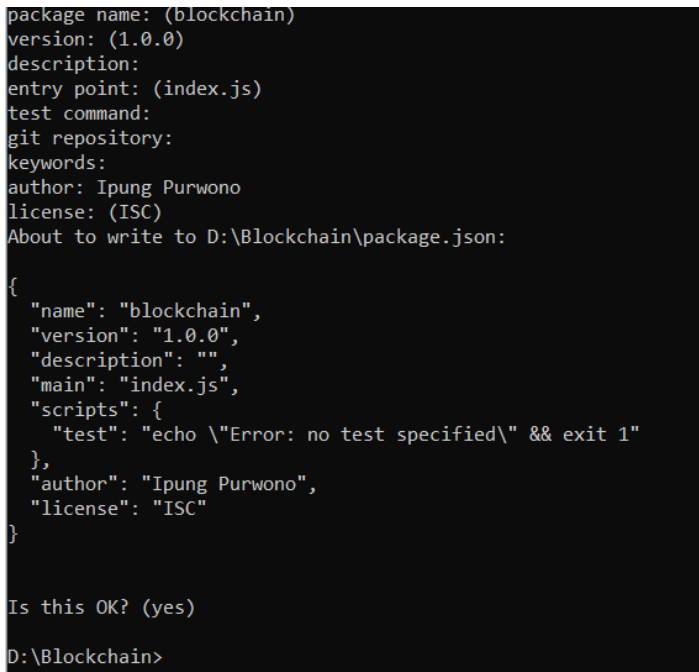
D:\>mkdir Blockchain

D:\>cd Blockchain

D:\Blockchain>
```

*Gambar 18. Membuat folder Blockchain*

3. Masih aktif pada command prompt anda, sekarang ketikkan perintah `npm init`. Isilah data-data proyek anda untuk mempermudah kebutuhan kedepannya nanti.



```
package name: (blockchain)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: Ipung Purwono
license: (ISC)
About to write to D:\Blockchain\package.json:

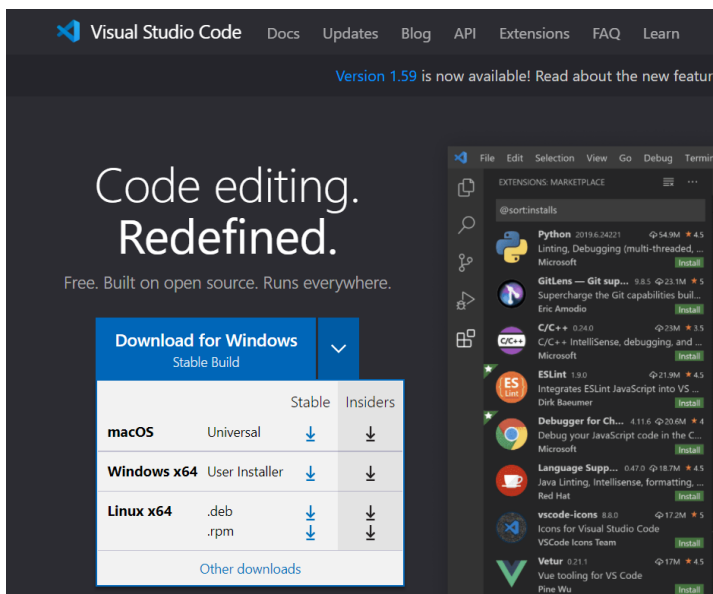
{
  "name": "blockchain",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Ipung Purwono",
  "license": "ISC"
}

Is this OK? (yes)

D:\Blockchain>
```

*Gambar 19. Inisialisasi Proyek Blockchain*

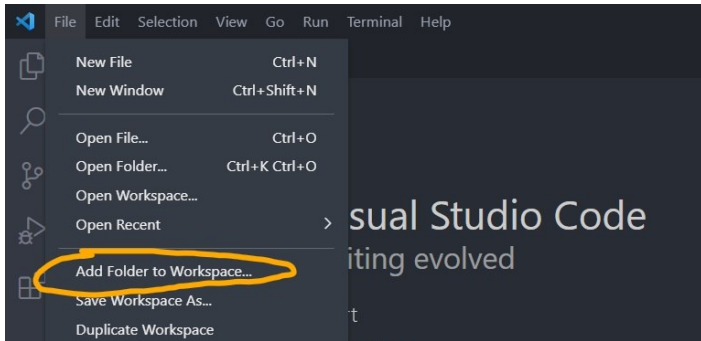
4. Sebelum melanjutkan, pastikan anda memiliki *code editor*. Dalam buku ini penulis menggunakan **visual studio code** sebagai editornya. Jika anda belum menginstal visual studio code, anda bisa menuju situs <http://www.code.vistualstudio.com>. Pilih lah file installer yang sesuai dengan sistem operasi anda. Sebagai contoh saya menggunakan windows 10 64-bit.



Gambar 20. Laman Download Visual Studio Code

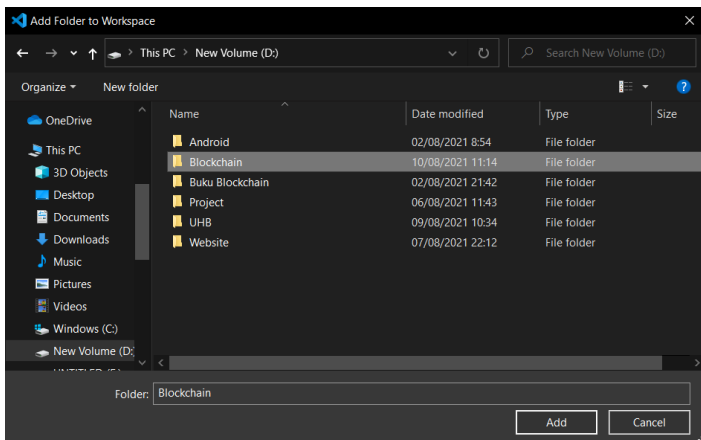
Anda dapat melakukan instalasi seperti pada umumnya, tinggal **next** dan ikuti instruksi instalasi *visual studio code* hingga selesai.

- Setelah anda berhasil menginstal visual studio code, bukalah aplikasi tersebut, lalu pada menu file pilihlah **add folder to project**.



Gambar 21. Menambahkan Folder ke Workspace

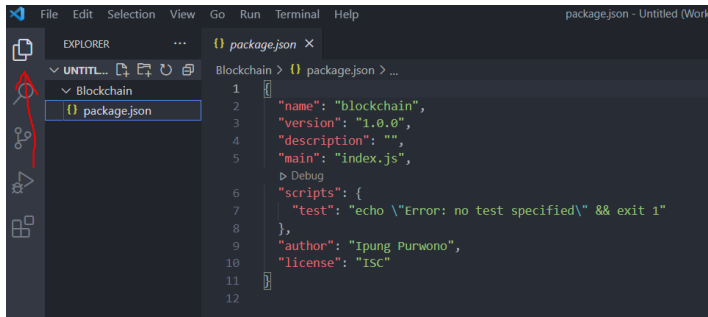
- Pilih folder proyek anda, karena saya membuat folder 'Blockchain' jadi saya memilih folder 'Blockchain' lalu tekan tombol **Add**.



Gambar 22. Memilih Nama Folder Proyek Anda

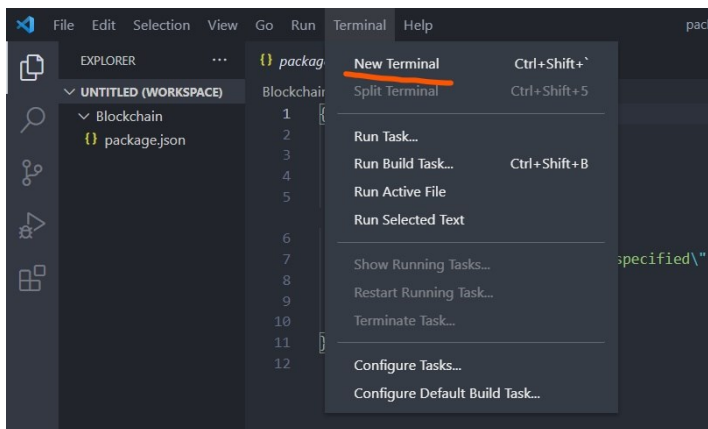
Pilih **Yes** jika muncul dialog untuk trust author.

7. Pilih **icon Explorer** dan cobalah untuk membuka file `package.json`. Disana anda akan melihat isi data dari proyek blockchain anda.



*Gambar 23. Membuka File Package.json*

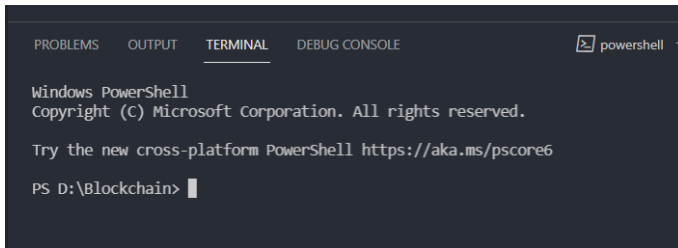
8. Bukalah menu terminal lalu pilih **new terminal**



*Gambar 24. Membuka Terminal Vs Code*

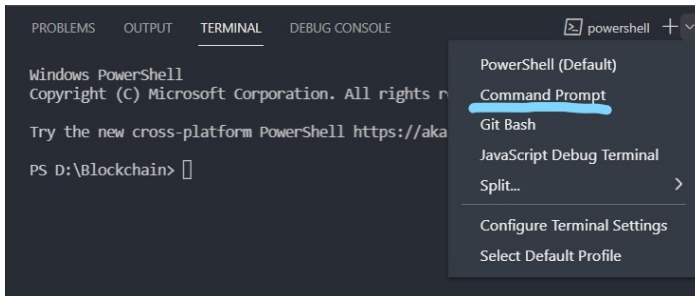
9. Terminal akan terbuka pada panel bawah visual studio code anda.





*Gambar 25. Terminal Vs Code*

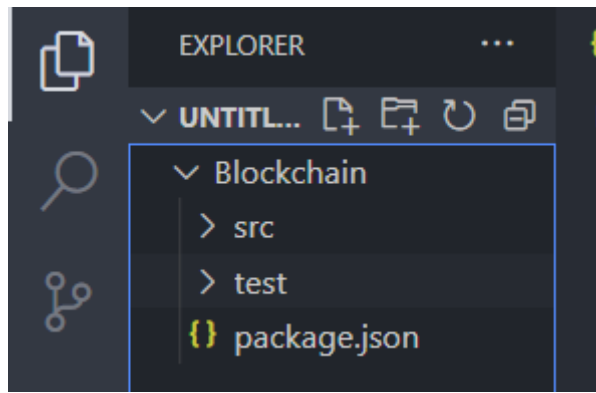
10. Masih pada panel terminal ubahlah mode powershell menjadi **command prompt**.



*Gambar 26. Mengubah Mode Command Prompt*

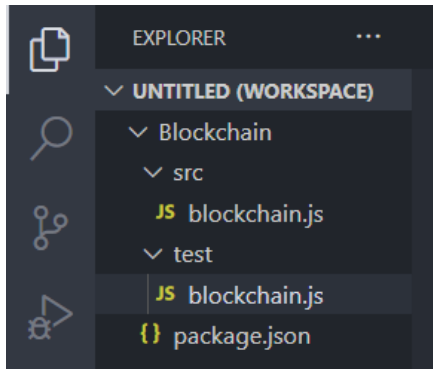
11. Saat ini anda sudah aktif dalam mode command prompt di visual studio code anda. Dengan ini anda akan mudah melakukan instalasi modul-modul node.js yang dibutuhkan pada proyek ini.
12. Anda juga bisa membuat multi terminal dengan memilih **tombol +**. Multi terminal dapat digunakan untuk berbagai kebutuhan seperti satu terminal untuk melakukan instalasi modul dan lainnya bisa digunakan untuk menjalankan server.

13. Buatlah dua buah folder baru pada posisi root dengan nama **src** dan **test**. Anda bisa membuatnya dengan klik kanan pada folder Blockchain lalu pilih new folder kemudian buat dua folder yaitu src dan test.



*Gambar 27. Folder Projek*

14. Tambahkan file baru pada folder src dengan cara klik kanan pada folder src lalu pilih **new file** dan buat file baru dengan nama **blockchain.js**
15. Begitu juga pada folder **test** buatlah sebuah file baru dengan nama **blockchain.js**



*Gambar 28. File baru di folder src dan test*

### 2.1.3 Instal Modul SHA256

Projek ini membutuhkan modul yang akan memenuhi fungsi hash, jadi kami memilih menggunakan modul **SHA256**. Modul ini memiliki algoritma hashing yang akan membantu kita melihat demonstrasi blockchain khususnya saat sebuah block menyimpan kode hash pada block sebelumnya. Untuk itu kita perlu melakukan instalasi modul dengan nama sha256. Instalasi modul ini dapat dilakukan pada terminal visual studio kita yang sudah terbuka sebelumnya. Langkah-langkah instalasi sha256 adalah sebagai berikut:

1. Anda cukup mengetikkan perintah instalasi sha256 dan menekan tombol **enter** pada keyboard anda.

```
npm install sha256
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.

D:\Blockchain>npm install sha256
npm notice created a lockfile as package-lock.json. You should commit this
file.
npm WARN blockchain@1.0.0 No description
npm WARN blockchain@1.0.0 No repository field.

+ sha256@0.2.0
added 3 packages and audited 3 packages in 7.932s
found 0 vulnerabilities

D:\Blockchain>
```

Gambar 29. Instal Modul SHA256

2. Proses instalasi akan selesai dengan ditandai informasi bahwa modul telah berhasil ditambahkan.
3. Modul akan tersimpan pada folder **node\_modules** dan juga terdapat informasi di file **package.json** terkait modul yang berhasil diinstal.

```
EXPLOLERER package.json JS blockchain.js src JS blockchain.js
Blockchain > package.json > ...
1 {
2   "name": "blockchain",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \\\"Error: no test specif
8   },
9   "author": "Ipung Purwono",
10  "license": "ISC",
11  "dependencies": {
12    "sha256": "^0.2.0"
13  }
14 }
15
```

Gambar 30. Modul SHA256 Telah Ditambahkan

## 2.2 Block Data Structure

Block merupakan bagian penting pada blockchain. Setiap data transaksi akan disimpan pada blok dan akan terhubung satu sama lain. Pada Bab 1 telah dijelaskan bahwa setiap blok memiliki struktur data khusus dimana ia akan menyimpan berbagai informasi seperti index block, timestamp, transaksi, nonce, hash dan hash blok sebelumnya. Kerangka blok dalam javascript dapat kita buat dengan code berikut di file **blockchain.js** yang terdapat pada folder **src**.

### Kode 1. Struktur Data Blok

```
1. class Block {
2.     constructor(index, timestamp, nonce, prevBlockHash,
   hash, transactions) {
3.         this.index = index;
4.         this.timestamp = timestamp;
5.         this.transactions = transactions;
6.         this.nonce = nonce;
7.         this.hash = hash;
8.         this.prevBlockHash = prevBlockHash;
9.     }
10. }
```

Keterangan Kode 1:

- Baris 1 kita mendefinisikan sebuah class baru untuk Block
- Baris 2 tambahkan sebuah *constructor* yang artinya adalah metode ini akan dijalankan

pertama kali saat Class objek digunakan. Constructor berisi tentang parameter yang harus dimasukan dalam block yaitu *index block*, *timestamp*, *nonce*, *prevBlockHash*, *hash* dan *transaction*.

- Baris 3 - 8 artinya adalah nilai yang dimasukan oleh user

## **2.3 Blockchain Object**

Block sudah kita buat sebelumnya pada Kode 1. Agar blockchain bekerja, kita harus membuat class baru bernama blockchain. Pada Bab 1 telah dijelaskan bahwa setiap blok akan saling terkait satu sama lain dengan rantai (*chain*). Blockchain juga memiliki genesis block yaitu block pertama yang harus ditambang. Blok juga akan terus bertambah ketika ada penambahan blok baru dari setiap transaksi.

### **2.3.1 Constructor**

Karena blockchain membutuhkan genesis block agar dapat memulai kinerjanya, sekarang kita buat class baru dengan nama Blockchain pada file **blockchain.js** di folder **src**.

## Kode 2. Class Blockchain dan Constructor

```
11. class Blockchain {
12.     constructor() {
13.         this.chain = [];
14.         this.pendingTransactions = [];
15.
16.         this.creatNewBlock(100, '0', 'Genesis block');
17.     }
18. }
```

### Keterangan Kode 2.

- Baris 11 ialah pembuatan *class* baru dengan nama Blockchain
- Baris 12 merupakan sebuah *constructor* atau fungsi yang akan dijalankan pertama kali saat class Blockchain digunakan.
- Baris 13 merupakan sebuah array yang akan menyimpan semua informasi penambangan dari setiap block. Untuk pertama kali **array chain** akan di set dengan array kosong. Array merupakan jenis struktur data yang memungkinkan kita bisa menyimpan berbagai jenis data dalam satu wadah yang sama.
- Baris 14 merupakan **pending transaction** yaitu transaksi-transaksi yang masuk pada sebuah

block dan posisinya adalah blok belum ditambah.

- Baris 16 ialah menjalankan metode untuk membuat genesis blok atau blok yang pertama kali ditambah. Dalam hal ini sebetulnya terdapat 3 parameter yaitu nonce, kode hash sebelumnya dan hash saat ini. Karena ini adalah genesis block penulis kemudian menyederhanakan dengan nilai yaitu **nonce=100, previousBlockHash=0 dan hash='Genesis Block'**.
- Baris 16 terlihat kita memanggil method lain yaitu **createNewBlock** yang akan kita buat selanjutnya.

### 2.3.2 Membuat Block Baru

*Constructor* dalam class Blockchain mewajibkan sistem untuk membuat blok baru dimana saat constructor dijalankan pertama kali adalah membuat genesis block. Hal lain yang harus dibuat adalah adanya metode ketika transaksi-transaksi baru terus bertambah dan kemudian ditambah oleh para miners. Oleh karena itu kita akan membuat sebuah method yaitu **creatNewBlock()**. Isi



kodenya dapat dilihat pada Kode 3.

### Kode 3. Method `creatNewBlock`

```
19. creatNewBlock(nonce, prevBlockHash, hash) {
20.     const newBlock = new Block(
21.         this.chain.length + 1,
22.         Date.now(),
23.         nonce,
24.         prevBlockHash,
25.         hash,
26.         this.pendingTransactions
27.     );
28.
29.     this.pendingTransactions = [];
30.     this.chain.push(newBlock);
31.
32.     return newBlock;
33. }
```

### Keterangan Kode 3.

- Baris 19 merupakan pembuatan method baru dengan nama `create New Block` dengan isi parameter yang digunakan adalah `nonce`, kode hash blok sebelumnya dan hash saat ini.
- Baris 20 membuat sebuah objek baru dengan nama `newBlock`. Objek ini mewarisi (*inheritance*) class `Block` yang telah dibuat sebelumnya (Kode 1). Namun kita perlu menambahkan 3 parameter baru yaitu ***prevBlockHash*** untuk membaca kode hash dari block sebelumnya, hash block saat ini

dan daftar transaksi yang masuk pada sebuah block.

- Baris 29 merupakan sebuah array kosong yang kedepannya akan diisi dengan banyaknya transaksi masuk.
- Baris 30 ialah penambahan data pada array chain yang isinya adalah block yang baru ditambahkan.
- Baris 32 method ini mengembalikan nilai berupa objek newBlock atau block baru.

### 2.3.3 Mengambil Informasi Block Terbaru

Kita perlu membuat sebuah method yang memungkinkan dapat menampilkan informasi blok terbaru. Oleh karena itu kita akan menambahkan method dengan nama **getLatestBlock** yang isinya adalah mengembalikan nilai dari chain posisi index terbaru. Masih pada class Blockchain di file **blockchain.js** di folder src tambahkan method berikut.

Kode 4. Method getLatestBlock

```
34. getLatestBlock() {  
35.     return this.chain[this.chain.length - 1];  
36. }
```

Keterangan Kode 4.

- Baris 34 merupakan pembuatan nama method yaitu **getLatestBlock**
- Baris 35 method ini akan mengembalikan nilai berupa isi array chain terbaru. Caranya adalah kita terlebih dahulu mengukur panjang index chain dengan fungsi length kemudian dikurangi 1 yang artinya adalah itu merupakan index terbaru.

### **2.3.4 Membuat Transaksi Baru**

Setiap waktu tertentu transaksi-transaksi terus bertambah dalam blockchain. Transaksi-transaksi tersebut akan masuk dalam daftar pending transaction atau transaksi yang sudah masuk pada sebuah blok namun blok tersebut belum ditambang atau belum ditambahkan pada jaringan blockchain. Sebagai penyederhanaan kita akan membuat method yang dapat membuat sebuah transaksi di blockchain. Parameter yang digunakan adalah amount atau jumlah transaksi misal 1 BTC atau 1 ETH, sender atau pengirim dan recipient atau penerimanya. Setiap adanya transaksi yang dibuat maka akan masuk pada data

transaksi pending. Kode untuk membuat metode transaksi baru dapat dilihat pada Kode 5.

#### Kode 5. Method `makeNewTransaction`

```
37. makeNewTransaction(amount, sender, recipient) {
38.     const transaction = {
39.         amount: amount,
40.         sender: sender,
41.         recipient: recipient
42.     }
43.
44.     this.pendingTransactions.push(transaction);
45.
46.     console.log(`>>> Transaction:    ${amount}    from
    ${sender} to ${recipient}`);
47.
48.     return this.getLatestBlock().index + 1;
49. }
```

#### Keterangan Kode 5.

- Baris 37 kita membuat method baru dengan nama ***makeNewTransaction***. Parameter yang digunakan ada `amount`, `sender` dan `recipient`.
- baris 38 - 42 kita membuat objek baru dimana objek `transaction` akan berisi key dan value yaitu **`amount`** adalah `amount` yang dimasukkan dalam parameter. Begitu juga dengan `sender` dan `recipient` yang mana, valuenya diambil dari parameter **method**.

- Baris 44 ketika ada transaksi baru maka akan masuk pada *pending transaction*. Hal ini dilakukan dengan cara push pada array **pendingTransaction()**.
- Baris 46 Tampilkan data transaction dalam console terminal visual studio code.
- Baris 48 kembalikan nilai blok terbaru dari fungsi **getLatestBlock + 1** yang telah dibuat pada Kode 4.

### 2.3.5 Hash Block

Setiap blok tentunya menyimpan kode hash sebelumnya sebagai penghubung block satu sama lain. Kita memanfaatkan modul SHA256 yang telah kita instal sebelumnya. Block akan dihash dengan method hashBlock pada Kode 6. Namun sebelumnya kita harus memanggil modul SHA256 pada file **blockchain.js** di bagian atas.

```
const sha256 = require('sha256')
```

Kemudian kita akan membuat kode method hashBlock di dalam class Blockchain yang dapat dilihat pada Kode 6.

### Kode 6. Method hashBlock

```
50. hashBlock(prevBlockHash, currentBlock, nonce) {
51.     const data = prevBlockHash +
      JSON.stringify(currentBlock) + nonce;
52.     const hash = sha256(data);
53.     return hash;
54. }
```

#### Keterangan Kode 6.

- Baris 50 method hashBlock memiliki parameter yaitu **prevBlockHash** yang isinya adalah kode hash dari blok sebelumnya. Selain itu juga terdapat currentBlock atau blok saat ini dan nonce atau nilai yang akan digunakan sekali untuk memenuhi tingkat kesulitan jaringan. Jika anda masih bingung dengan *nonce* silahkan buka kembali pada Bab 1.
- Baris 51 kita membuat variabel baru dengan nama data dimana isinya adalah hash dari blok sebelumnya, blok saat ini yang telah diubah formatnya menjadi string JSON dan angka nonce.
- Baris 52 merupakan kode hash dari variable data dengan memanfaatkan algoritma hashing sha256.

- Baris 53 method mengembalikan nilai hasil dari hash 256 ini.

### **2.3.6 Proof of Work**

Berbicara *Proof of Work (PoW)* yang merupakan salah satu metode penting di Blockchain karena ia adalah kunci keamanan dari data blockchain. PoW akan memastikan semua blok baru atau blok yang telah ditambang oleh para miners itu benar-benar terjadi dan sudah melewati validasi sempurna berupa kesepakatan semua anggota jaringan blockchain. PoW akan mengambil data hash dari blok sebelumnya dan data blok saat ini. Data tersebut kemudian akan menghasilkan kode *hash* baru yang sudah mencapai kesepakatan kesulitan jaringan, misalnya sebuah kesulitan jaringan mewajibkan 5 angka 0 di depan kode hash. Perulangan atau iterasi ini akan menghasilkan nilai *nonce*. Nilai *nonce* kemudian digunakan untuk membuat kode hash baru yang akan disimpan pada blok selanjutnya yang sudah berhasil ditambang oleh para miners. Ketika ada penambahan blok baru maka miners juga wajib memecahkan masalah matematika atau kesulitan jaringan dan wajib menghasilkan nilai *nonce* baru dan akan terus berlanjut seperti itu.

Masih pada class Blockchain di file blockchain.js di folder src, kita akan membuat metode dengan nama **proofOfWork** yang dapat anda lihat pada kode 7.

#### Kode 7. Method proofOfWork

```
55. proofOfWork(prevBlockHash, currentBlockData) {
56.     let nonce = 0;
57.     let hash = this.hashBlock(prevBlockHash,
    currentBlockData, nonce);
58.
59.     while (hash.substring(0, 2) !== '00') {
60.         nonce++;
61.         hash = this.hashBlock(prevBlockHash,
    currentBlockData, nonce);
62.     };
63.
64.     return nonce;
65. }
```

#### Keterangan Kode 6.

- Baris 55 kita membuat method baru yaitu proof of work dengan parameter input adalah kode hash sebelumnya dan data blok saat ini di class Blockchain
- Baris 56 kita membuat variable *nonce*, kita mulai terlebih dahulu dengan nilai 0
- Baris 57 buat sebuah variabel dengan nama hash dimana kita akan menggunakan fungsi **hashBlock** untuk merubah isi dari hash block



sebelumnya, data blok saat ini dan angka nonce ke dalam bentuk hash sha256.

- Baris 59 - 62 lakukan perulangan dengan while untuk mencari nonce. Pada method ini kami menyederhanakan kesulitan jaringan dimana dua digit hash di posisi depan harus bernilai angka 00. Lakukan perulangan hingga benar-benar menemukan 2 angka 0 di depan kode hash baru. Jumlah perulangan hingga berhasil inilah yang disebut nonce. Misalnya kita berhasil menemukan 00 pada saat perulangan ke 1024 maka angka tersebut adalah noncenya.
- Baris 64 selanjutnya kita mengembalikan nilai nonce yang sudah berhasil ditemukan.

Tahap selanjutnya adalah melakukan export module file **blockchain.js** agar dapat dibaca oleh file lain. Jadi, silahkan tambahkan kode berikut pada akhir file blockchain.js

```
module.exports = Blockchain;
```

Kami akan memperlihatkan kode secara keseluruhan pada buku ini agar anda bisa melakukan cek ulang agar tidak ada yang terlewat.

## Kode blockchain.js

```
1. const sha256 = require('sha256');
2.
3. class Block {
4.   constructor(index, timestamp, nonce, prevBlockHash,
   hash, transactions) {
5.     this.index = index;
6.     this.timestamp = timestamp;
7.     this.transactions = transactions;
8.     this.nonce = nonce;
9.     this.hash = hash;
10.    this.prevBlockHash = prevBlockHash;
11.  }
12. }
13.
14. class Blockchain {
15.   constructor() {
16.     this.chain = [];
17.     this.pendingTransactions = [];
18.
19.     this.creatNewBlock(100, '0', 'Genesis block');
20.   }
21.
22.   creatNewBlock(nonce, prevBlockHash, hash) {
23.     const newBlock = new Block(
24.       this.chain.length + 1,
25.       Date.now(),
26.       nonce,
27.       prevBlockHash,
28.       hash,
29.       this.pendingTransactions
30.     );
31.
32.     this.pendingTransactions = [];
33.     this.chain.push(newBlock);
34.
35.     return newBlock;
36.   }
37.
38.   getLatestBlock() {
39.     return this.chain[this.chain.length - 1];
40.   }
41.
42.   makeNewTransaction(amount, sender, recipient) {
43.     const transaction = {
44.       amount: amount,
```

```

45.         sender: sender,
46.         recipient: recipient
47.     }
48.
49.     this.pendingTransactions.push(transaction);
50.
51.     console.log(`>>> Transaction: ${amount} from
    ${sender} to ${recipient}`);
52.
53.     return this.getLatestBlock().index + 1;
54. }
55.
56. hashBlock(prevBlockHash, currentBlock, nonce) {
57.     const data = prevBlockHash +
    JSON.stringify(currentBlock) + nonce;
58.     const hash = sha256(data);
59.     return hash;
60. }
61.
62. proofOfWork(prevBlockHash, currentBlockData) {
63.     let nonce = 0;
64.     let hash = this.hashBlock(prevBlockHash,
    currentBlockData, nonce);
65.
66.     while (hash.substring(0, 2) !== '00') {
67.         nonce++;
68.         hash = this.hashBlock(prevBlockHash,
    currentBlockData, nonce);
69.     };
70.
71.     return nonce;
72. }
73. }
74.
75. module.exports = Blockchain;

```

### 2.3.7 Menjalankan Blockchain

Semua method pada class blockchain telah kita buat, selanjutnya adalah kita akan melakukan testing agar bisa melihat gambaran bagaimana blockchain ini bekerja. Untuk itu kita pindah pada folder test dan akan

mengetikan code baru pada file blockchain.js. Kode ini dapat dilihat pada Kode 7.

### Kode 7. Testing Blockchain

```
1. const Blockchain = require('../src/blockchain');
2. function mine(blockChain) {
3.   console.log('>>> Mining.....');
4.   const latestBlock = blockChain.getLatestBlock();
5.   const prevBlockHash = latestBlock.hash;
6.   const currentBlockData = {
       transactions: blockChain.pendingTransactions,
       index: latestBlock.index + 1
7. }
8.   const nonce = blockChain.proofOfWork(prevBlockHash,
   currentBlockData);
9.   const blockHash = blockChain.hashBlock(prevBlockHash,
   currentBlockData, nonce);
10. // reward for mining
11. blockChain.makeNewTransaction(1, '00000', 'miningNode');
12. console.log('>>> Create new Block:\n',
   blockChain.creatNewBlock(nonce, prevBlockHash, blockHash));
13. }
14. const bitcoin = new Blockchain();
15. console.log('>>> Create new Blockchain:\n', bitcoin);
16. bitcoin.makeNewTransaction(120, 'IPUNG', 'ARIF');
17. mine(bitcoin);
18. bitcoin.makeNewTransaction(1120, 'IPUNG', 'WICAK');
19. bitcoin.makeNewTransaction(300, 'IMAM', 'AGUS');
20. bitcoin.makeNewTransaction(2700, 'SLAMET', 'IMAM');
21. mine(bitcoin);
22.   console.log('>>> Current Blockchain Data:\n',
   bitcoin);
```

### Keterangan Kode 7.

- Baris 1 kita harus melakukan import kode **blockchain.js** pada folder **src** yang sudah di export sebagai sebuah modul.

- Baris 3 kita membuat sebuah fungsi mining sederhana dengan nama **mine**.
- Baris 4 tampilkan proses mining pada console yang mengindikasikan bahwa fungsi ini dijalankan.
- Baris 6 buat sebuah variable **latestBlock** untuk menampung data block terbaru dengan method **getLatestBlock()**
- Baris 7 buat sebuah variabel baru **prevBlockHash** untuk menampung data hash sebelumnya dari variabel **latestBlock** dan ambil nilai hashnya.
- Baris 8 - 11 buat sebuah variabel **currentBlockData** isinya adalah objek dengan key transactions isinya adalah transaksi yang masih pending atau transaksi baru masuk dari blockChain.**pendingTransactions** dan key **index** yang isinya adalah index dari blok terakhir ditambah 1.
- Baris 12 membuat sebuah variable nonce isinya adalah hasil dari penggunaan konsep **proofOfWork**

- Baris 13 membuat variable **blockHas** yang isinya adalah hasil hash dari method **hashBlock**
- Baris 16 jalankan sebuah transaksi baru yang merupakan bentuk rewards atas keberhasilannya melakukan mining kepada penambang. Secara sederhana penambang akan mendapatkan 1 cryptocurrency.
- Baris 18 tampilkan dalam **console log** terkait dengan pembuatan blok baru
- Baris 21 buat sebuah objek baru dari class Blockchain dengan nama **bitcoin**
- Baris 22 tampilkan kembali dalam *console* isi dari objek bitcoin
- Baris 24 jalankan satu transaksi baru
- Baris 26 lakukan **mining** pada data blok yang berisi 2 transaksi sebelumnya
- Baris 28 - 30 buat kembali 3 transaksi baru pada blockchain
- Baris 32 lakukan **mining** kembali pada blok yang berisi 3 transaksi baru
- Baris 34 tampilkan data bitcoin dalam bentuk **json**.

Untuk menjalankan kode ini, anda harus menuju *console* atau terminal visual studio code anda. Saat aktif pada posisi projek folder dengan mode command prompt ketikkan perintah berikut pada terminal.

```
node test/blockchain.js
```

Perintah tersebut menggunakan server **node.js** untuk menjalankan file **blockchain.js** pada folder test. Setelah diketik perintah tersebut tekan tombol enter dan anda akan melihat hasil tes tersebut. Jika berhasil anda akan melihat data transaksi dan proses **mining** yang telah anda lakukan.

#### Hasil Test file blockchain.js pada Console Visual Studio code

```
D:\Blockchain>node test/blockchain.js
>>> Create new Blockchain:
Blockchain {
  chain: [
    Block {
      index: 1,
      timestamp: 1628649691935,
      transactions: [],
      nonce: 100,
      hash: 'Genesis block',
      prevBlockHash: '0'
    }
  ],
  pendingTransactions: []
}
>>> Transaction: 120 from IPUNG to ARIF
```

```

>>> Mining.....
>>> Transaction: 1 from 00000 to miningNode
>>> Create new Block:
Block {
  index: 2,
  timestamp: 1628649691961,
  transactions: [
    { amount: 120, sender: 'IPUNG', recipient: 'ARIF' },
    { amount: 1, sender: '00000', recipient: 'miningNode' }
  ],
  nonce: 171,
  hash:
'00472db606e68812d20cee58db18c720d6b0ece06736a2b789146abbf5b464e3',
  prevBlockHash: 'Genesis block'
}
>>> Transaction: 1120 from IPUNG to WICAK
>>> Transaction: 300 from IMAM to AGUS
>>> Transaction: 2700 from SLAMET to IMAM
>>> Mining.....
>>> Transaction: 1 from 00000 to miningNode
>>> Create new Block:
Block {
  index: 3,
  timestamp: 1628649691972,
  transactions: [
    { amount: 1120, sender: 'IPUNG', recipient: 'WICAK' },
    { amount: 300, sender: 'IMAM', recipient: 'AGUS' },
    { amount: 2700, sender: 'SLAMET', recipient: 'IMAM' },
    { amount: 1, sender: '00000', recipient: 'miningNode' }
  ],
  nonce: 138,
  hash:
'004af97eb734e7af1c7ef4e701d3fc0295511cbdb2ac6ac5616987044a8e8ff4',
  prevBlockHash:
'00472db606e68812d20cee58db18c720d6b0ece06736a2b789146abbf5b464e3'
}
>>> Current Blockchain Data:
Blockchain {
  chain: [
    Block {
      index: 1,
      timestamp: 1628649691935,
      transactions: [],
      nonce: 100,
      hash: 'Genesis block',
      prevBlockHash: '0'
    },
    Block {

```



```

    index: 2,
    timestamp: 1628649691961,
    transactions: [Array],
    nonce: 171,
    hash:
'00472db606e68812d20cee58db18c720d6b0ece06736a2b789146abbf5b464e3',
    prevBlockHash: 'Genesis block'
  },
  Block {
    index: 3,
    timestamp: 1628649691972,
    transactions: [Array],
    nonce: 138,
    hash:
'004af97eb734e7af1c7ef4e701d3fc0295511cbdb2ac6ac5616987044a8e8ff4',
    prevBlockHash:
'00472db606e68812d20cee58db18c720d6b0ece06736a2b789146abbf5b464e3'
  }
],
pendingTransactions: []
}

```

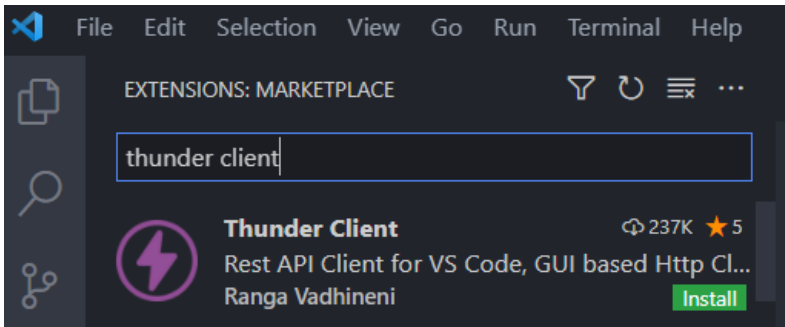
Hasilnya adalah terlihat pertama kali yang terjadi yaitu pembuatan **genesis block**, kemudian ada transaksi pertama dan kedua. Setelah dilakukan *mining* maka blok baru terbentuk. Anda juga akan melihat kode *hash* yang telah terbentuk dan tersimpan pada block baru. Selanjutnya informasi juga di update setelah ada 3 transaksi baru dan dilakukan *mining*. Sekarang block yang ada di blockchain memiliki 3 blok yaitu block pertama merupakan *genesis block*, block kedua yang merupakan block setelah adanya *mining* untuk transaksi 1 dan 2 lalu blok 3 merupakan blok baru yang terbentuk setelah adanya proses *mining* dari 3 transaksi terakhir.

## 2.4 Membuat Blockchain API di Javascript

Kita akan membuat sebuah API sederhana dengan javascript. Kita akan memanfaatkan Node.js untuk mempercepat pekerjaan pembuatan Blockchain API ini. Dengan API kode blockchain akan dengan lebih mudah digunakan khususnya jika kita ingin membuat sebuah tampilan misalnya dalam bentuk website atau aplikasi mobile. Beberapa endpoint url yang akan kita buat pada Blockchain API ini adalah sebagai berikut:

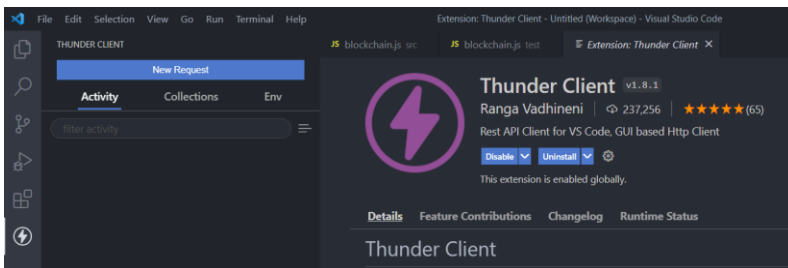
- **GET** /blockchain => endpoint untuk melihat seluruh data blockchain
- **POST** /transaction => endpoint untuk membuat transaksi baru
- **GET** /mine => endpoint untuk melakukan mining atau menambahkan blok baru ke dalam blockchain.

Namun, kita membutuhkan plugins pendukung yaitu thunder client pada visual studio code. Untuk menginstalnya silahkan cari *extensions* dengan nama thunder client pada menu **extension** di visual studio code.



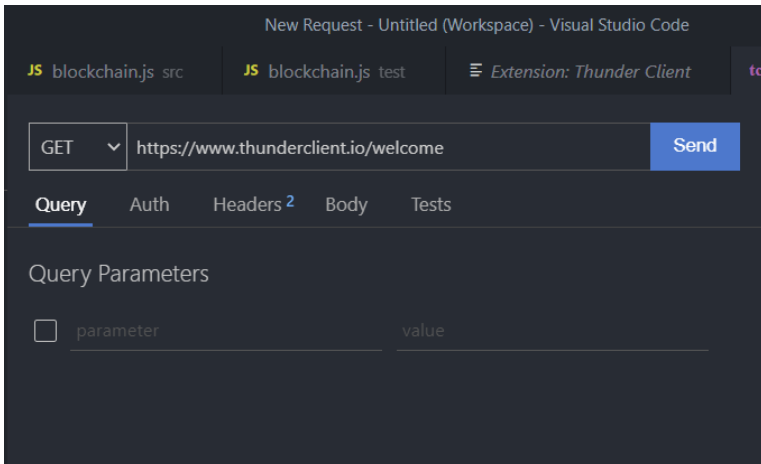
*Gambar 31. Ekstensi Thunder Client VS Code*

Pilih tombol install dan tunggu hingga proses instalasi selesai. Jika anda berhasil menginstalnya maka akan muncul icon thunder client pada sisi kanan visual studio code.



*Gambar 32. Instalasi Thunder Client*

Sebagai percobaan, silahkan tekan menu New Request, maka anda akan melihat sebuah client untuk mencoba endpoint kita ke depan.



*Gambar 33. Thunder Client User Interface*

### **2.4.1 Setup Environment**

Modul lain yang akan kita instal sebagai pendukung node.js untuk membuat API Blockchain antara lain **Express.js**, **body parser** dan **nodemon**. ketiganya dapat dengan mudah diinstal dengan terminal di visual studio code.

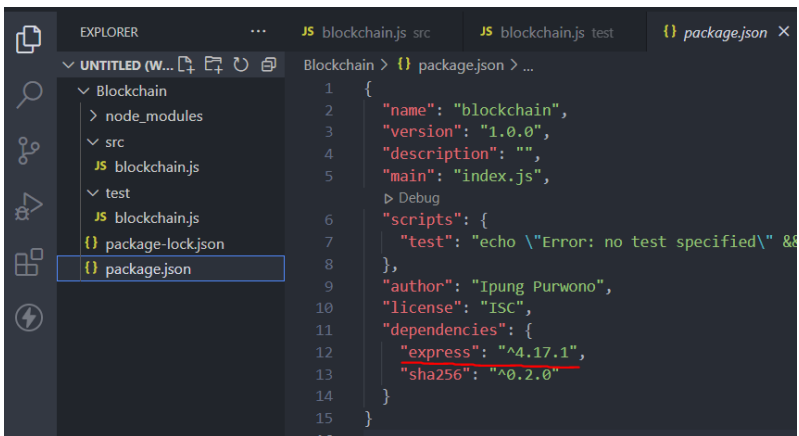
#### 2.4.1.1 Install Express.Js

*Express.js* merupakan salah satu framework populer yang digunakan untuk membuat aplikasi website berbasis node.js. Express js didukung langsung dari mesin Google V8 sehingga performa kinerjanya menjadi lebih maksimal. Kita akan melakukan instalasi express

js. Masih pada terminal visual studio code silahkan ketikkan perintah berikut

```
npm install express
```

Tekan enter dan tunggu hingga proses instalasi selesai. Untuk melakukan cek apakah instalasi berhasil, silahkan buka file package.json dan anda akan melihat versi expressnya. Pada saat buku ini dibuat, kami menggunakan **express versi 4.17.1**



Gambar 34. Instalasi Express.Js

#### 2.4.1.2 Install Nodemon

Selama proses development aplikasi Node.js, kita sebetulnya bisa melakukan restart server secara otomatis ketika ada perubahan yang terjadi pada saat

pengembangan. Oleh karena itu kita membutuhkan modul yaitu **nodemon**. Nodemon akan selalu melakukan cek file-file yang ada di direktori proyek. Ketika ada perubahan, maka nodemon secara otomatis restart server dan aplikasi yang dijalankan saat ini adalah aplikasi terupdate. Untuk menginstal nodemon kita bisa mengetikkan perintah berikut.

```
npm install nodemon
```

Tekan enter dan tunggu hingga proses instalasi selesai. Pada tahap ini ada pengaturan khusus pada file `package.json` agar kita bisa menjalankan server `node.js` secara otomatis. Silahkan buka file **package.json** lalu tambahkan kode berikut pada posisi `script`.

```
"scripts": {  
  "test": "echo \\\"Error: no test specified\\\" && exit 1",  
  "start": "nodemon --watch src -e js src/api.js"  
},
```

### 2.4.2 Membangun Blockchain API

Pembuatan blockchain API kita mulai dengan membuat sebuah file baru pada folder `src` yaitu **api.js**. Jika kalian memperhatikan, pada file `package.json` sebelumnya, kita telah merubah `script` tersebut dan mengarahkannya

pada file `api.js` dengan modul `nodemon`. Bukalah file `api.js` kemudian tambahkan modul `express.js`

#### Kode `api.js`

```
1. const express = require('express');
2. const app = express();
3. app.use(express.json());
```

#### Keterangan Kode `api.js`

- Baris 1 kita memanggil modul `express`
- Baris 2 buat variabel dengan nama `app` dimana `app` ini adalah modul `express`
- Baris 3 gunakan format `json` untuk mempermudah input data parameter pada property **`req.body`** saat api dijalankan.

Masih pada kode `api.js` tambahkan beberapa baris kode dimana kita akan menentukan address node dan memanggil objek `blockchain` dari file **`blockchain.js`** yang telah kita buat sebelumnya.

#### Kode `api.js`

```
4. const nodeAddr = 'ADDR_OWNER';
5.
6. const Blockchain = require('../src/blockchain');
7. const bitcoin = new Blockchain();
```

## Keterangan Kode

- Baris 4 kita membuat variabel **nodeAddr** owner
- Baris 6 memanggil file modul kerangka Blockchain dari file **blockchain.js**
- Baris 7 buat sebuah objek baru dengan nama bitcoin dimana ia mewarisi semua isi dan metode dari class Blockchain

Pada file api.js sekarang kita tambahkan end point yang digunakan untuk mengambil seluruh data blockchain. Dalam API kita bisa menggunakan metode GET request. Alamat yang akan kita gunakan dalam API adalah sebagai berikut <http://localhost:3000/blockchain>. Sekarang silahkan tambahkan fungsi baru pada file api.js

### Kode api.js

```
8. app.get('/blockchain', function (req, res) {
9.     res.send(bitcoin);
10. });
```

## Kerangan Kode:

- Baris 8 dengan metode get dengan merujuk pada alamat **/blockchain** kita akan menampilkan response.



- Baris 9 merupakan respons dari data bitcoin yang notabene adalah objek turunan dari class Blockchain.

Kita juga akan menambahkan end point untuk membuat transaksi baru dimana url ini akan membutuhkan tiga parameter yaitu **amount**, **sender** dan **recipient**. Sekarang tambahkan kembali pada file api.js

#### Kode api.js

```
11. app.post('/transaction', function (req, res) {
12.     const blockIndex = bitcoin.makeNewTransaction(
13.         req.body.amount,
14.         req.body.sender,
15.         req.body.recipient
16.     );
17.
18.     res.json(
19.         {
20.             message: `Transaction is added to block with
index: ${blockIndex}`
21.         }
22.     );
23. });
```

#### Keterangan Kode:

- Baris 11 buat sebuah end point **/transaction** dengan metode POST untuk mengirimkan data pada blockchain, fungsi ini akan memanfaatkan request atau mengambil data dari apa yang

diinputkan user dan response untuk menampilkan hasil ketika transaksi ditambahkan

- Baris 12 buat sebuah objek baru dengan nama `blockIndex` isinya adalah memanggil metode **`makeNewTransaction`** dari objek bitcoin.
- Baris 13 - 15 merupakan penggunaan property javascript yaitu request pada body. Kita akan menginput 3 data yaitu amount, sender dan recipient.
- Baris 18 - 21 merupakan tampilan response berupa pesan dari server jika transaksi berhasil ditambahkan dan dia akan masuk pada blok ke x.

Tambahkan juga endpoint **`/mine`** dengan metode GET untuk melakukan mining pada data blok baru yang akan ditambahkan. Ingat setiap blok memiliki data-data transaksi yang masih bersifat tertunda (*pending*). Setelah dilakukan mining oleh miners baru blok berhasil ditambahkan. endpoint ini digunakan sebagai simulasi untuk melakukan penambangan (*mining*). Pada kode api.js kita tambahkan kode berikut.

#### Kode api.js

```
24. app.get('/mine', function (req, res) {  
25.     const latestBlock = bitcoin.getLatestBlock();
```

```

26.     const prevBlockHash = latestBlock.hash;
27.     const currentBlockData = {
28.         transactions: bitcoin.pendingTransactions,
29.         index: latestBlock.index + 1
30.     }
31.     const nonce = bitcoin.proofOfWork(prevBlockHash,
currentBlockData);
32.     const blockHash = bitcoin.hashBlock(prevBlockHash,
currentBlockData, nonce);
33.
34.     // reward for mining
35.     bitcoin.makeNewTransaction(1, '00000', nodeAddr);
36.
37.     const newBlock = bitcoin.creatNewBlock(nonce,
prevBlockHash, blockHash)
38.     res.json(
39.         {
40.             message: 'Mining new Block successfully!',
41.             newBlock
42.         }
43.     );
44. });

```

#### Keterangan Kode:

- Baris 24 kita buat rute endpoint dengan nama **/mine** metodenya adalah GET
- Baris 25 membuat variabel **latestBlock** yang isinya adalah mengambil data blok terbaru dari metode **getLatestBlock**.
- Baris 26 membuat variable **prevBlockHash** yang isinya adalah hash blok sebelumnya
- Baris 27 membuat sebuah objek baru dengan nama **currentBlockData** isinya adalah data transaksi dari transaksi masuk yang masih

tertunda dan juga index yang sebetulnya adalah blok terakhir ditambah 1

- Baris 31 menjalankan PoW yang menghasilkan nilai **nonce**
- Baris 32 membuat variabel **blockHash** yang isinya adalah hash sha256 dari hash sebelumnya, data blok saat ini dan nonce.
- Baris 35 berikan sebuah **rewards** pada penambang yang disini adalah node address dia akan mendapatkan 1 koin cryptocurrency.
- Baris 37 membuat sebuah objek baru yaitu **newBlock** yang isinya adalah menjalankan pembuatan blok baru dengan metode **creatNewBlock**.
- Baris 38 - 42 merupakan pesan berbentuk JSON yaitu blok baru berhasil ditambahkan setelah adanya mining.

Setelah semua fungsi dibuat untuk membuat route endpoint, kita akan menjalankan server node.js ini pada port 3000. Masih pada file api.js tambahkan kode berikut

#### Kode api.js

```
45. app.listen(3000, function () {  
46.     console.log('> listening on port 3000...');  
47. });
```

#### Keterangan Kode:

- Baris 45 express menjalankan server pada port 3000 di localhost kita
- Baris 46 tampilkan console log bahwa benar server nodejs di jalankan di port 3000

### 2.4.3 Menjalankan API Blockchain

Sekarang saatnya kita menjalankan API blockchain yang telah kita buat. Dari sini kita telah memiliki tiga endpoint yaitu GET \blockchain, POST \transaction dan GET \mine. Untuk menjalankannya kita akan menggunakan nodemon modul. Caranya sangat mudah anda cukup mengetikkan perintah berikut pada console visual studio code anda.

```
npm start
```

Jika berhasil maka anda akan melihat bahwa server dijalankan di port 3000 di localhost.

```
PS D:\Blockchain> npm start

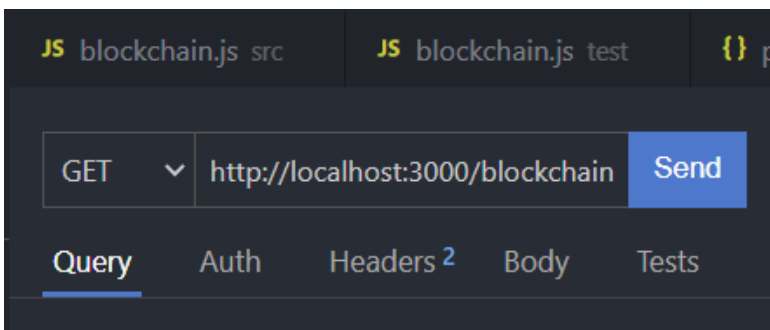
> blockchain@1.0.0 start D:\Blockchain
> nodemon --watch src -e js src/api.js

[nodemon] 2.0.12
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\*
[nodemon] watching extensions: js
[nodemon] starting `node src/api.js`
> listening on port 3000...
█
```

Gambar 35. Menjalankan Server Node.Js

### 2.4.3.1 GET Data Blockchain

Pada visual studio code kita aktifkan ekstensi thunder bolt yang berlogo petir, kemudian pilih new request. Kita masukan alamat endpoint <http://localhost:3000/blockchain> berikut pada thunder bolt. Gunakan metode GET request. Klik send button untuk memproses endpoint ini.



Gambar 36. GET Request Blockchain

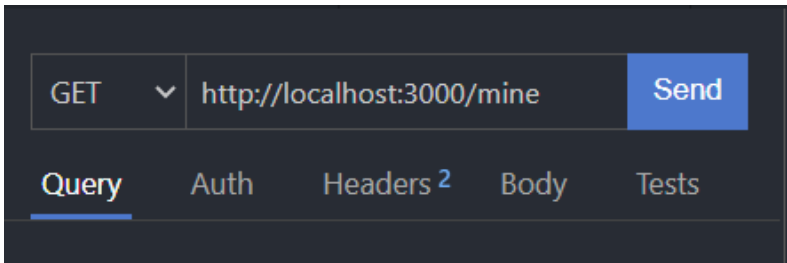
Berdasarkan gambar di atas, ketika memasukkan alamat <http://localhost:3000/blockchain> artinya adalah pada server localhost di port 3000 dengan endpoint GET /blockchain. Setelah kita menekan tombol send maka response server adalah 200 OK dan menampilkan pesan data JSON berikut.

```
1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628664308733,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    }
11.  ],
12.  "pendingTransactions": []
13. }
```

Data JSON ini memperlihatkan bahwa pada rantai blok pertama adalah genesis blok, dia masih belum memiliki data transaksi apapun. Nonce juga masih dibuat default angka 100.

#### 2.4.3.2 GET MINE GENESIS BLOCK

Jalankan *mining* pertama kali dari genesis blok dengan cara memasukkan end point <http://localhost:3000/mine> pada thunderbolt metodenya adalah GET.



Gambar 37. GET Mine Blockchain

Jika anda menekan tombol **Send**, maka response JSON nya adalah sebagai berikut.

```
1. {
2.   "message": "Mining new Block successfully!",
3.   "newBlock": {
4.     "index": 2,
5.     "timestamp": 1628665068320,
6.     "transactions": [
7.       {
8.         "amount": 1,
9.         "sender": "00000",
10.        "recipient": "ADDR_OWNER"
11.       }
12.     ],
13.     "nonce": 44,
14.     "hash":
15.     "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa
16.     23e55dfe964",
17.     "prevBlockHash": "Genesis block"
18.   }
19. }
```

Pesan JSON yang dihasilkan menerangkan bahwa proses mining block berhasil dilakukan. Sekarang blok bertambah menjadi 2 blok. Mining pertama ini memberikan rewards berupa 1 cryptocurrency pada penambang. Anda juga akan melihat kode hash dari



blok sebelumnya di key hash. Nonce yang dihasilkan adalah 44.

Kita buktikan kembali dengan menjalankan endpoint <http://localhost:3000/blockchain> pada thunderbolt di visual studio. Kita akan melihat data JSON berikut.

```
1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628664308733,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    },
11.    {
12.      "index": 2,
13.      "timestamp": 1628665068320,
14.      "transactions": [
15.        {
16.          "amount": 1,
17.          "sender": "00000",
18.          "recipient": "ADDR_OWNER"
19.        }
20.      ],
21.      "nonce": 44,
22.      "hash":
23.      "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa2
24.      3e55dfe964",
25.      "prevBlockHash": "Genesis block"
26.    }
27.  ],
28.  "pendingTransactions": []
29. }
```

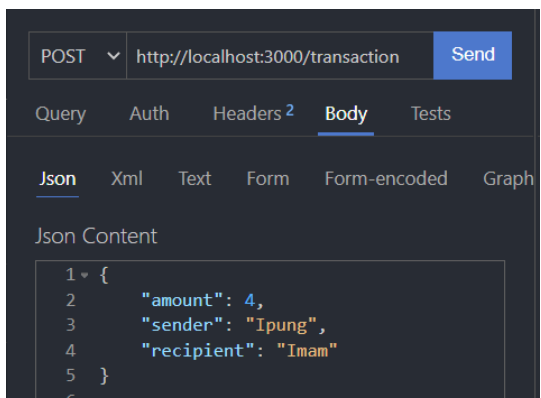
Berdasarkan response JSON tersebut kita melihat ada dua index yaitu index 1 dan 2 artinya sekarang sudah ada dua buah blok yaitu index blok genesis dan blok 2

yang berisi blok baru setelah adanya mining genesis block.

### 2.4.3.3 POST Transaction

Kita coba tambahkan satu data transaksi baru dengan memanggil endpoint yaitu <http://localhost:3000/transaction> metodenya POST pada thunderbolt. Pada sisi body kita pilih mode JSON dan masukan data seperti berikut.

```
{
  "amount": 4,
  "sender": "Ipung",
  "recipient": "Imam"
}
```



Gambar 38. POST Transaction Blockchain

Tekan tombol send dan anda akan melihat pesan response JSON bahwa data transaksi berhasil ditambahkan.

```
{
  "message": "Transaction is added to block with
index: 3"
}
```

Pada pesan tersebut transaksi berhasil ditambahkan pada index ke 3 yang berarti transaksi sudah masuk pada blok ke-3. Jalankan endpoint untuk memeriksa keseluruhan data blockchain dengan memasukan kembali endpoint <http://localhost:3000/blockchain> pada thunderbolt visual studio code. Jika benar, anda akan melihat response yaitu data JSON berupa transaksi tersebut masuk pada pending transactions, artinya transaksi sudah masuk pada blok namun belum ditambang oleh *miner*.

```
"pendingTransactions": [
  {
    "amount": 4,
    "sender": "Ipung",
    "recipient": "Imam"
  }
]
```

Transaksi sebelumnya masuk pada pending transaction. Agar dapat bertambah maka harus

dilakukan mining. Ingat setiap *mining* berhasil maka penambang mendapatkan **rewards** berupa 1 **cryptocurrency**

#### 2.4.3.4 GET mine

Jalankan endpoint mine untuk menambang blok yang berisi transaksi baru. Masukkan alamat <http://localhost:3000/mine> dengan GET. Tekan tombol send dan anda akan mendapatkan pesan JSON seperti berikut.

```
1. {
2.   "message": "Mining new Block successfully!",
3.   "newBlock": {
4.     "index": 3,
5.     "timestamp": 1628666849254,
6.     "transactions": [
7.       {
8.         "amount": 4,
9.         "sender": "Ipung",
10.        "recipient": "Imam"
11.      },
12.      {
13.        "amount": 1,
14.        "sender": "00000",
15.        "recipient": "ADDR_OWNER"
16.      }
17.    ],
18.    "nonce": 53,
19.    "hash":
20.      "00810c97320a926b49e47aefcaea75891dde9d272db17
21.      5f6d5862581e9f50d65",
22.    "prevBlockHash":
23.      "00669af06fa8d395bea4f6325d39f43a3c87788152fc3
24.      050608aa23e55dfe964"
```

Pesan memperlihatkan bahwa proses mining berhasil dengan menghasilkan angka *nonce* 53 dan hash blok baru. Pastikan kembali dengan menjalankan endpoint GET pada <http://localhost:3000/blockchain> untuk melihat apakah ada penambahan blok baru atau tidak.

```
1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628664308733,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    },
11.    {
12.      "index": 2,
13.      "timestamp": 1628665068320,
14.      "transactions": [
15.        {
16.          "amount": 1,
17.          "sender": "00000",
18.          "recipient": "ADDR_OWNER"
19.        }
20.      ],
21.      "nonce": 44,
22.      "hash":
23.      "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa23e5
24.      5dfe964",
25.      "prevBlockHash": "Genesis block"
26.    },
27.    {
28.      "index": 3,
29.      "timestamp": 1628666849254,
30.      "transactions": [
31.        {
32.          "amount": 4,
33.          "sender": "Ipung",
34.          "recipient": "Imam"
35.        },
36.        {
37.          "amount": 1,
```

```

36.         "sender": "00000",
37.         "recipient": "ADDR_OWNER"
38.     }
39. ],
40.     "nonce": 53,
41.     "hash":
    "00810c97320a926b49e47aefcaea75891dde9d272db175f6d5862581e
    9f50d65",
42.     "prevBlockHash":
    "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa23e5
    5dfe964"
43. }
44. ],
45. "pendingTransactions": []
46. }

```

Berdasarkan pesan JSON inu terlihat pada array pending transactions sudah kosong, dan terlihat ada blok baru yaitu blok dengan index 3 artinya sekarang blok sudah bertambah 1 setelah adanya proses mining. Pada baris 26 - 43 muncul blok baru yaitu index 3 isinya adalah data transaksi yang masuk pada blok tersebut. Hal ini terjadi karena blok sudah ditambang.

Anda bisa mencoba-coba menambah transaksi baru dengan skema berikut:

- Tambahkan 2 transaksi baru dengan memanggil endpoint POST \transaction
- Cek data dengan memanggil endpoint GET \blockchain

- Lakukan Mining dengan memanggil endpoint GET \mine
- Cek data kembali dengan memanggil endpoint GET \blockchain

Perhatikan apakah anda bisa melihat blok-blok baru tersebut bertambah atau tidak. Selanjutnya kita akan membuat agar blockchain kita terdesentralisasi. Kita akan menambahkan beberapa node komputer lain agar data dapat didistribusikan pada seluruh anggota jaringan blockchain.

## **2.5 Desentralisasi Jaringan Blockchain di Javascript**

Konsep umum dari teknologi blockchain adalah jaringannya yang terdesentralisasi. Jadi itu akan berbeda dari jaringan yang memiliki otoritas tunggal. Pada bahasan sebelumnya kita hanya memiliki satu node komputer blockchain saja dimana proses transaksi dan penambangan blok baru hanya dilakukan oleh satu komputer. Jika itu dilakukan berarti aplikasi blockchain kita belum menerapkan jaringan terdesentralisasi, oleh karena itu kita akan membuat jaringan ini memiliki beberapa node komputer lain yang terhubung satu sama lain.

Pada section ini kita akan membuat 3 endpoint yaitu:

- **POST** /register-node digunakan untuk menambahkan sebuah node secara spesifik. Sebagai contoh anda adalah node 1 jika anda ingin menambahkan satu persatu node ke dalam jaringan anda misal node 5 dan kemudian anda menambahkan node 4 dan seterusnya.
- **POST** /register-bulk-nodes digunakan untuk menambahkan sebuah node dengan beberapa node sekaligus dalam satu waktu. Sebagai contoh adalah dalam satu waktu anda ingin menambahkan 3 node sekaligus misalnya node 2 ingin menambahkan node 3, node 4 dan node 5.
- **POST** /register-and-broadcast-node digunakan untuk menambahkan sebuah node dan disiarkan kepada seluruh jaringan. Contoh penggunaan endpoint ini adalah jika sebelumnya misalkan node 1 telah memiliki 3 node misal node 2, node 3 dan node 4 kemudian anda ingin menambahkan node 5 dan disiarkan untuk ditambahkan ke node-node di atas. Jika ini dilakukan maka node 5 tidak hanya terdaftar di node 1 saja melainkan juga terdaftar di node 2, 3 dan 4.



## 2.5.1 Setup Environment

Untuk membuat itu semua kita membutuhkan beberapa modul tambahan untuk diinstal dengan npm. Modul-modul tersebut antara lain adalah **UUID** dan Request-Promise.

### 2.5.1.1 Instal UUID

Modul UUID akan digunakan untuk membuat address unik dari setiap node. Untuk menginstal modul ini ketikkan perintah berikut pada terminal visual studio anda lalu tekan enter.

```
npm install uuid
```

Untuk meyakinkan anda apakah modul uuid sudah terinstal dengan baik, seperti biasa cek file package.json anda.

### 2.5.1.2 Instal Request-Promise

Modul *request-promise* merupakan penerapan sederhana pada HTTP Request Client yang mendukung fitur Promise. Dalam javascript promise merupakan sebuah mekanisme standar ECMAScript 2015 yang memungkinkan kita melakukan eksekusi fungsi kode javascript secara asynchronous dan mendapatkan nilai

kembalian (return) dari eksekusi kode tersebut secara tidak langsung, melainkan berupa objek “Promise” yang menjanjikan eksekusi di masa yang akan datang.

Secara sederhana promise sering digunakan dalam proses asinkron yang notabene akan membuat jalan keputusan ketika dalam pelaksanaan proses asinkron ini sukses atau gagal di masa depan. Proses asinkron dalam javascript mengembalikan hasil yang belum bisa diprediksi mana proses-proses yang selesai dijalankan terlebih dahulu. Misalnya ketika kita membuat fungsi untuk mengambil data dari database dengan konsep asinkron dari dua buah endpoint (data provinsi dan data pendidikan). Dalam perjalanannya secara asinkron kita tidak pernah tahu apakah data provinsi atau data pendidikan yang berhasil ditampilkan terlebih dahulu, sehingga kita dapat memanfaatkan Promise ini. Promise akan mengeksekusi kode yang sudah dibuat sebelumnya untuk mengantisipasi hal-hal yang akan terjadi di masa depan. Jika tadi terdapat pengambilan data provinsi dan data pendidikan misalnya, apa yang akan dilakukan sistem jika tiba-tiba saat mengambil data provinsi ternyata server down sehingga gagal di load? Nah promise ini bisa kita manfaatkan dengan memberikan sebuah rencana ketika hal buruk

itu terjadi misal jika koneksi buruk maka jalankan fungsi menampilkan alert bahwa network sedang bermasalah dan sebagainya.

Dalam membangun jaringan desentralisasi ini promise akan kita gunakan ketika menjalankan endpoint broadcast node. Hal ini digunakan karena saat sebuah node ditambahkan dan disiarkan ke dalam jaringan tentunya harus secara bertahap memanfaatkan fungsi pada endpoint register node yang dilakukan satu per satu dalam menambahkan node ke node lain. Untuk menginstalnya silahkan ketikkan perintah berikut kemudian tekan enter untuk memulai proses instalasi.

```
npm install request-promise
```

Anda juga membutuhkan modul request agar request-promise dapat berjalan dengan baik. Untuk menginstalnya silahkan ketikkan perintah berikut kemudian tekan enter untuk memulai proses instalasi.

```
npm install request
```

Sekarang silahkan cek file package.json seharusnya anda sudah berhasil menambahkan uuid, request dan request-promise.

```
"author": "Ipung Purwono",
"license": "ISC",
"dependencies": {
  "body-parser": "^1.19.0",
  "express": "^4.17.1",
  "nodemon": "^2.0.12",
  "request": "^2.88.2",
  "request-promise": "^4.2.6",
  "sha256": "^0.2.0",
  "uuid": "^8.3.2"
}
```

### 2.5.1.3 Membuat Multiple Node

Dalam demo jaringan terdesentralisasi ini kita mencoba menggunakan simulasi 5 node komputer yaitu:

- **node 1 dengan alamat <http://localhost:3001>**
- **node 2 dengan alamat <http://localhost:3002>**
- **node 3 dengan alamat <http://localhost:3003>**
- **node 4 dengan alamat <http://localhost:3004>**
- **node 5 dengan alamat <http://localhost:3005>**

Berarti kita harus menjalankan 5 server sekaligus dengan port berbeda. Untungnya kita dapat dibantu dengan modul node.js. Agar kita dapat menjalankan 5

server sekaligus kita akan mengubah isi script di package.json sebagai berikut.

```
1. "scripts": {
2.   "test": "echo \"Error: no test specified\" && exit 1",
3.   "node1": "nodemon --watch src -e js src/api.js 3001
   http://localhost:3001",
4.   "node2": "nodemon --watch src -e js src/api.js 3002
   http://localhost:3002",
5.   "node3": "nodemon --watch src -e js src/api.js 3003
   http://localhost:3003",
6.   "node4": "nodemon --watch src -e js src/api.js 3004
   http://localhost:3004",
7.   "node5": "nodemon --watch src -e js src/api.js 3005
   http://localhost:3005"
8. },
```

Keterangan kode:

Baris ke 2 - 7 kita menambahkan port berbeda dari 3001 - 3005 yang akan mewakili masing-masing node komputer dalam jaringan terdesentralisasi. Selanjutnya ubah port pada file api.js yang awalnya hanya dapat menjalankan port 3000 saja, tapi sekarang file tersebut dapat digunakan untuk menjalankan server dengan beberapa port. Cari kode berikut pada file api.js

#### Kode api.js

```
app.listen(3000, function () {
  console.log('> listening on port 3000...');
});
```

Kemudian ubah dengan kode berikut

#### Kode api.js

```
const port = process.argv[2];
app.listen(port, function () {
  console.log(`> listening on port ${port}...`);
});
```

Tambahkan juga modul *uuid*, *request* dan *request-promise* pada file *api.js*. Tempatkan pada baris kode di bagian atas sebelum anda mendefinisikan setiap route endpoint.

#### Kode api.js

```
9.   const { v4: uuidv4 } = require('uuid');
10.  const nodeAddr = uuidv4();
11.
12.  const reqPromise = require('request-promise');
```

## 2.5.2 Menambahkan network Nodes ke Blockchain

Buka kembali file **blockchain.js** pada folder *src* anda, kita akan menambahkan *nodeurl* dan *network nodes* dalam konstruktor.

#### Kode blockchain.js

```
const nodeUrl = process.argv[3];
```

Selanjutnya tambahkan `networkNodes` dan `nodeUrl` pada class `Blockchain` di file `blockchain.js`

Kode `blockchain.js`

```
1. class Blockchain {
2.     constructor() {
3.         this.chain = [];
4.         this.pendingTransactions = [];
5.
6.         this.nodeUrl = nodeUrl;
7.         this.networkNodes = [];
8.
9.         this.createNewBlock(100, '0', 'Genesis block');
10.    }
11. ...
12. }
```

Keterangan Kode:

- Baris ke 6 kita tambahkan `nodeUrl` berupa alamat address node komputer
- Baris ke 7 kita masukan array `networkNodes` untuk menampung semua nodes dalam jaringan setelah ditambahkan.

## 2.5.3 Membuat End Points

### 2.5.3.1 Register Node

Buatlah endpoint baru yaitu `/register-node` metodenya adalah `POST`. Fungsinya adalah menambahkan sebuah node url pada jaringan node url lain. Sebagai contoh

node 1 menambahkan node 3 sebagai anggota jaringannya. Pada file **api.js** silahkan tambahkan kode berikut.

#### Kode api.js

```
1. app.post('/register-node', function (req, res) {
2.     const nodeUrl = req.body.nodeUrl;
3.
4.     if ((bitcoin.networkNodes.indexOf(nodeUrl) == -1)
5.         && (bitcoin.nodeUrl !== nodeUrl)) {
6.         bitcoin.networkNodes.push(nodeUrl);
7.
8.         res.json(
9.             {
10.                message: 'A node registers successfully!'
11.            }
12.        );
13.    }
14.    else {
15.        res.json(
16.            {
17.                message: 'This node cannot register!'
18.            }
19.        );
20.    }
21. });
```

#### Keterangan Kode:

- Baris 1 kita membuat sebuah endpoint dengan nama **/register-node** metodenya adalah POST.
- Baris 2 buatlah sebuah variabel `nodeUrl` isinya adalah data yang diinputkan pada property body saat endpoint dijalankan.



- Baris 4 - 6 buat sebuah kondisi jika sebuah index dari array **networkNodes** adalah -1 dan nodeUrl bukan alamat dirinya sendiri maka,tambahkan nodeUrl tersebut pada array networkNodes dengan metode push.
- Baris 8 - 13 response JSON berupa pesan berhasil ditambahkan ke array networkNodes.
- Baris 14 - 19 response JSON berupa pesan jika nodeUrl tidak boleh ditambahkan ke array networkNodes.

### 2.5.3.2 Register Node di Bulk

Buat kembali endpoint dimana kita dapat menambahkan multiple nodes dalam satu waktu. Kita berikan nama endpoint nya adalah **/register-bulk-nodes**. Sebagai contoh kita akan menambahkan node 2, 3 dan 4 dalam satu waktu pada node 1. Fungsi ini digunakan oleh fungsi broadcast nodes. Untuk membuatnya silahkan tambahkan kode berikut pada file api.js.

#### Kode api.js

```
1. app.post('/register-bulk-nodes', function (req, res) {
2.     const networkNodes = req.body.networkNodes;
3.
4.     networkNodes.forEach(nodeUrl => {
5.         if ((bitcoin.networkNodes.indexOf(nodeUrl) == -1)
6.             && (bitcoin.nodeUrl !== nodeUrl)) {
```

```

7.         bitcoin.networkNodes.push(nodeUrl);
8.     }
9. });
10.
11. res.json(
12.     {
13.         message: 'Registering bulk successfully!'
14.     }
15. );
16. });

```

#### Keterangan kode:

- Baris 1 kita membuat route endpoint dengan nama **/register-bulk-nodes**
- Baris 2 variabel `networkNodes` yang berisi data `networkNodes` dari property `request.body`.
- Baris 4 - 9 lakukan perulangan dari multiple nodes yang ditampung oleh variable `networkNodes` dengan `foreach`. Jika index array untuk `networkNodes` `== -1` dan bukan node url dirinya sendiri maka tambahkan dengan metode `push` pada array `networkNodes` berisi multiple nodes pada variabel `networkNodes`.
- Baris 11 - 14 buatlah sebuah pesan JSON bahwa register secara multiple nodes berhasil.

### 2.5.3.3 Register dan Broadcast Node

Tahap pembuatan jaringan terdesentralisasi ditutup dengan pembuatan endpoint **register-and-broadcast-node** menggunakan metode POST. Fungsi ini akan menyiarkan node baru yang ditambahkan dengan memanfaatkan endpoint **/register-bulk-nodes**. Sebagai contoh implementasinya adalah ketika dalam suatu jaringan terdapat 3 node yaitu node 1, 2 dan 3 kemudian kita ingin melakukan broadcast node 5 pada semua jaringan tersebut. Pada file api.js silahkan tambahkan kode berikut.

#### Kode api.js

```
1. app.post('/register-and-broadcast-node', function (req, res)
2.   {
3.     const nodeUrl = req.body.nodeUrl;
4.     if (bitcoin.networkNodes.indexOf(nodeUrl) == -1) {
5.       bitcoin.networkNodes.push(nodeUrl);
6.     }
7.
8.     const registerNodes = [];
9.     bitcoin.networkNodes.forEach(networkNode => {
10.      const requestOptions = {
11.        uri: networkNode + '/register-node',
12.        method: 'POST',
13.        body: { nodeUrl: nodeUrl },
14.        json: true
15.      }
16.
17.      registerNodes.push(reqPromise(requestOptions));
18.    });
19.
20.    Promise.all(registerNodes)
21.      .then(data => {
22.        const bulkRegisterOptions = {
```

```

23.             uri: nodeId + '/register-bulk-nodes',
24.             method: 'POST',
25.             body: {                               networkNodes:
    [...bitcoin.networkNodes, bitcoin.nodeUrl] },
26.             json: true
27.         }
28.     }
29.     return reqPromise(bulkRegisterOptions);
30. }).then(data => {
31.     res.json(
32.         {
33.             message: 'A node registers with network
successfully!'
34.         }
35.     );
36. });
37. });

```

#### Keterangan kode:

- Baris 1 adalah nama endpoint untuk melakukan **broadcast** node pada jaringan.
- Baris 2 merupakan variabel untuk menampung inputan **nodeUrl** yang akan ditambahkan.
- Baris 4 - 6 jika index dari network nodes adalah - 1 maka **nodeUrl** ditambahkan dalam network nodes.
- Baris 8 membuat variabel array yaitu **registerNodes**
- Baris 9 melakukan looping dengan foreach untuk isi dari setiap networkNodes

- Baris 10 -15 merupakan objek baru dengan nama `requestOptions` yang isinya ada uri end point bulk nodes, metodenya `POST`, body nya adalah `nodeUrl` dan support format `JSON`.
- Baris 17 menambahkan data objek dari **`requestOptions`** ke array `registerNodes` dengan memanfaatkan `promise`.
- Baris 20 - 21 lakukan `promise` pada semua data `registerNodes`
- Baris 22 - 28 buat objek baru dengan `bulkRegisterOptions` dengan isinya adalah uri berupa pemanggilan endpoint bulk nodes, metodenya `POST`, body nya adalah isi `networkNodes` dengan format `JSON`.
- Baris 29 berikan kembalian nilai berupa request `promise` untuk objek `bulkRegisterOptions`.
- Baris 30 - 36 tampilkan pesan `JSON` jika broadcast ini berhasil dilakukan.

Secara keseluruhan file `api.js` kodenya adalah sebagai berikut.

<b>Kode api.js</b>
<pre>1. const express = require('express');</pre>

```

2. const app = express();
3. app.use(express.json())
4.
5. const { v4: uuidv4 } = require('uuid');
6. const nodeAddr = uuidv4();
7.
8. const reqPromise = require('request-promise');
9.
10. const Blockchain = require('../src/blockchain');
11. const bitcoin = new Blockchain();
12.
13. app.get('/blockchain', function (req, res) {
14.     res.send(bitcoin);
15. });
16.
17. app.post('/transaction', function (req, res) {
18.     const blockIndex = bitcoin.makeNewTransaction(
19.         req.body.amount,
20.         req.body.sender,
21.         req.body.recipient
22.     );
23.
24.     res.json(
25.         {
26.             message: `Transaction is added to block with
index: ${blockIndex}`
27.         }
28.     );
29. });
30.
31. app.get('/mine', function (req, res) {
32.     const latestBlock = bitcoin.getLatestBlock();
33.     const prevBlockHash = latestBlock.hash;
34.     const currentBlockData = {
35.         transactions: bitcoin.pendingTransactions,
36.         index: latestBlock.index + 1
37.     }
38.     const nonce = bitcoin.proofOfWork(prevBlockHash,
currentBlockData);
39.     const blockHash = bitcoin.hashBlock(prevBlockHash,
currentBlockData, nonce);
40.
41.     // reward for mining
42.     bitcoin.makeNewTransaction(1, '00000', nodeAddr);
43.
44.     const newBlock = bitcoin.creatNewBlock(nonce,
prevBlockHash, blockHash)
45.     res.json(

```

```

46.         {
47.             message: 'Mining new Block successfully!',
48.             newBlock
49.         }
50.     );
51. });
    //ubah port
52. const port = process.argv[2];
53.
54. app.listen(port, function () {
55.     console.log(`> listening on port ${port}...`);
56. });
57.
58. //tambahan end point register-node
59. app.post('/register-node', function (req, res) {
60.     const nodeUrl = req.body.nodeUrl;
61.
62.     if ((bitcoin.networkNodes.indexOf(nodeUrl) == -1)
63.         && (bitcoin.nodeUrl !== nodeUrl)) {
64.         bitcoin.networkNodes.push(nodeUrl);
65.
66.         res.json(
67.             {
68.                 message: 'A node registers successfully!'
69.             }
70.         );
71.     }
72.     else {
73.         res.json(
74.             {
75.                 message: 'This node cannot register!'
76.             }
77.         );
78.     }
79. });
80. //tambahn endpoint register-bulk-nodes
81. app.post('/register-bulk-nodes', function (req, res) {
82.     const networkNodes = req.body.networkNodes;
83.
84.     networkNodes.forEach(nodeUrl => {
85.         if ((bitcoin.networkNodes.indexOf(nodeUrl) == -1)
86.             && (bitcoin.nodeUrl !== nodeUrl)) {
87.             bitcoin.networkNodes.push(nodeUrl);
88.         }
89.     });
90.
91.     res.json(
92.         {

```

```

93.         message: 'Registering bulk successfully!'
94.     }
95. );
96. });
97. //tambahan end-point register-and-broadcast-node
98. app.post('/register-and-broadcast-node', function (req, res)
99. {
100.     const nodeUrl = req.body.nodeUrl;
101.     if (bitcoin.networkNodes.indexOf(nodeUrl) == -1)
102.     {
103.         bitcoin.networkNodes.push(nodeUrl);
104.     }
105.     const registerNodes = [];
106.     bitcoin.networkNodes.forEach(networkNode => {
107.         const requestOptions = {
108.             uri: networkNode + '/register-node',
109.             method: 'POST',
110.             body: { nodeUrl: nodeUrl },
111.             json: true
112.         }
113.     }
114.     registerNodes.push(reqPromise(requestOptions));
115. });
116.
117.     Promise.all(registerNodes)
118.         .then(data => {
119.             const bulkRegisterOptions = {
120.                 uri: nodeUrl + '/register-bulk-
nodes',
121.                 method: 'POST',
122.                 body: { networkNodes:
[...bitcoin.networkNodes, bitcoin.nodeUrl] },
123.                 json: true
124.             }
125.
126.             return reqPromise(bulkRegisterOptions);
127.         }).then(data => {
128.             res.json(
129.                 {
130.                     message: 'A node registers with
network successfully!'
131.                 }
132.             );
133.         });
134. });

```



Selanjutnya kode **blockchain.js** secara lengkap dapat dilihat pada kode berikut ini.

#### Kode blockchain.js

```
1. const sha256 = require('sha256');
2. const nodeUrl = process.argv[3];
3.
4. class Block {
5.   constructor(index, timestamp, nonce, prevBlockHash,
6.     hash, transactions) {
7.     this.index = index;
8.     this.timestamp = timestamp;
9.     this.transactions = transactions;
10.    this.nonce = nonce;
11.    this.hash = hash;
12.    this.prevBlockHash = prevBlockHash;
13.  }
14. }
15. class Blockchain {
16.   constructor() {
17.     this.chain = [];
18.     this.pendingTransactions = [];
19.
20.     this.nodeUrl = nodeUrl;
21.     this.networkNodes = [];
22.
23.     this.createNewBlock(100, '0', 'Genesis block');
24.   }
25.
26.   creatNewBlock(nonce, prevBlockHash, hash) {
27.     const newBlock = new Block(
28.       this.chain.length + 1,
29.       Date.now(),
30.       nonce,
31.       prevBlockHash,
32.       hash,
33.       this.pendingTransactions
34.     );
35.
36.     this.pendingTransactions = [];
37.     this.chain.push(newBlock);
38.
39.     return newBlock;
40.   }
```

```

41.
42.   getLatestBlock() {
43.     return this.chain[this.chain.length - 1];
44.   }
45.
46.   makeNewTransaction(amount, sender, recipient) {
47.     const transaction = {
48.       amount: amount,
49.       sender: sender,
50.       recipient: recipient
51.     }
52.
53.     this.pendingTransactions.push(transaction);
54.
55.     console.log(`>>> Transaction:  ${amount}   from
    ${sender} to ${recipient}`);
56.
57.     return this.getLatestBlock().index + 1;
58.   }
59.
60.   hashBlock(prevBlockHash, currentBlock, nonce) {
61.     const data = prevBlockHash +
    JSON.stringify(currentBlock) + nonce;
62.     const hash = sha256(data);
63.     return hash;
64.   }
65.
66.   proofOfWork(prevBlockHash, currentBlockData) {
67.     let nonce = 0;
68.     let hash = this.hashBlock(prevBlockHash,
    currentBlockData, nonce);
69.
70.     while (hash.substring(0, 2) !== '00') {
71.       nonce++;
72.       hash = this.hashBlock(prevBlockHash,
    currentBlockData, nonce);
73.     };
74.
75.     return nonce;
76.   }
77. }
78.
79. module.exports = Blockchain;

```

Kode file **package.json** secara lengkap dapat dilihat pada kode berikut ini.

#### File package.json

```
1. {
2.   "name": "blockchain",
3.   "version": "1.0.0",
4.   "description": "",
5.   "main": "index.js",
6.   "scripts": {
7.     "test": "echo \"Error: no test specified\" && exit 1",
8.     "node1": "nodemon --watch src -e js src/api.js 3001
9.     http://localhost:3001",
10.    "node2": "nodemon --watch src -e js src/api.js 3002
11.    http://localhost:3002",
12.    "node3": "nodemon --watch src -e js src/api.js 3003
13.    http://localhost:3003",
14.    "node4": "nodemon --watch src -e js src/api.js 3004
15.    http://localhost:3004",
16.    "node5": "nodemon --watch src -e js src/api.js 3005
17.    http://localhost:3005"
18.  },
19.   "author": "Ipung Purwono",
20.   "license": "ISC",
21.   "dependencies": {
22.     "body-parser": "^1.19.0",
23.     "express": "^4.17.1",
24.     "nodemon": "^2.0.12",
25.     "request": "^2.88.2",
26.     "request-promise": "^4.2.6",
27.     "sha256": "^0.2.0",
28.     "uuid": "^8.3.2"
29.   }
30. }
```

#### 2.5.3.4 Menjalankan Blockchain

Tahap uji coba pembuatan jaringan terdesentralisasi ini dilakukan dengan terlebih dahulu mengaktifkan 5 node server.

## Mengaktifkan Node Server

Untuk mengaktifkan kelima node komputer tersebut, kita akan membuka 5 tab terminal pada visual studio code, atau anda juga bisa menggunakan 5 command prompt anda, namun penulis akan menggunakan tab terminal dari visual studio code. Untuk itu dari setiap tap ketikan perintah berikut dna tekan enter.

Terminal 1 Mode CMD

```
npm run node1
```

Terminal 2 Mode CMD

```
npm run node2
```

Terminal 3 Mode CMD

```
npm run node3
```

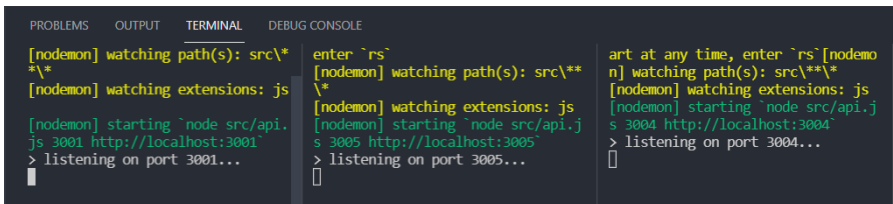
Terminal 4 Mode CMD

```
npm run node4
```

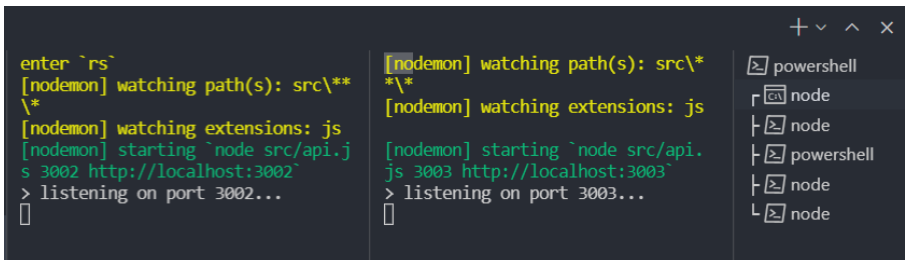
Terminal 5 Mode CMD

```
npm run node5
```

Jlka sudah di run semua servernya, maka anda akan melihat bahwa semua server node telah berjalan.



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
[nodemon] watching path(s): src\**
*\*
[nodemon] watching extensions: js
[nodemon] starting `node src/api.js 3001 http://localhost:3001`
> listening on port 3001...
[enter `rs`
[nodemon] watching path(s): src\**
*\*
[nodemon] watching extensions: js
[nodemon] starting `node src/api.js 3005 http://localhost:3005`
> listening on port 3005...
art at any time, enter `rs`
[nodemon] watching path(s): src\**
*\*
[nodemon] watching extensions: js
[nodemon] starting `node src/api.js 3004 http://localhost:3004`
> listening on port 3004...
```

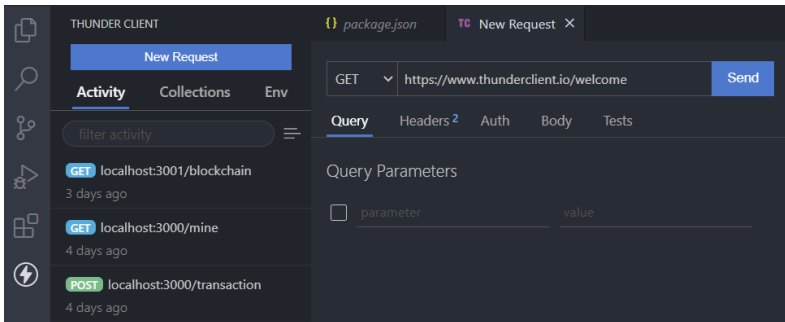


```
enter `rs`
[nodemon] watching path(s): src\**
*\*
[nodemon] watching extensions: js
[nodemon] starting `node src/api.js 3002 http://localhost:3002`
> listening on port 3002...
[nodemon] watching path(s): src\**
*\*
[nodemon] watching extensions: js
[nodemon] starting `node src/api.js 3003 http://localhost:3003`
> listening on port 3003...
powershell
node
node
powershell
node
node
```

Gambar 39. Menjalankan 5 Node Server

## Membuka Thunderbolt Extension

Masih pada visual studio anda silahkan buka kembali ekstensi Thunderbolt. Lalu pilih **New Request**.

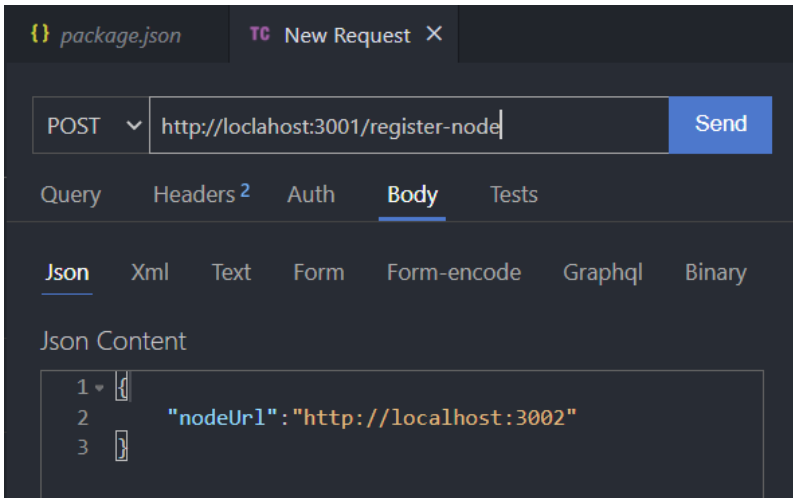


*Gambar 40. New Request ThunderBolt Client*

## Implementasi Register Node

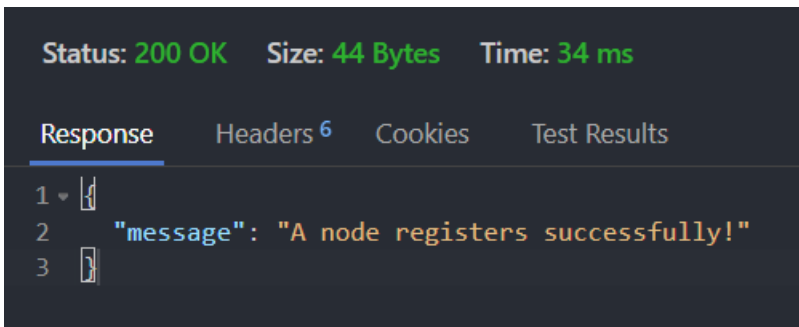
Pada **thunderbolt**, sekarang kita akan coba menambahkan node2, node3 dan node4 pada node1. Untuk menambahkannya, kita akan menggunakan endpoint <http://localhost:3001/register-node>. Penggunaan 3001 karena pada study case kali ini kita akan menambahkan 3 node pada node1 sedangkan node 1 itu memiliki port 3001. Nah masukan pada URL thunderbolt dengan metode POST. Pada kolom body tambahkan kode json berikut.

```
{
  "nodeUrl": "http://localhost:3002"
}
```



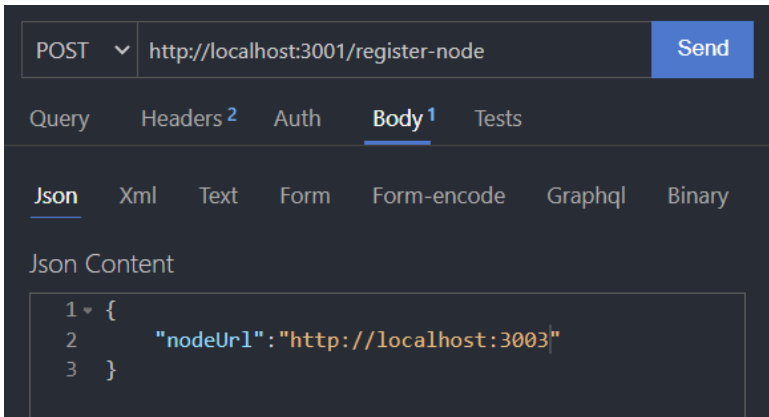
Gambar 41. Menambahkan Node 2 pada Node 1

Pilih tombol **Send**, maka hasil responnya adalah sebagai berikut.



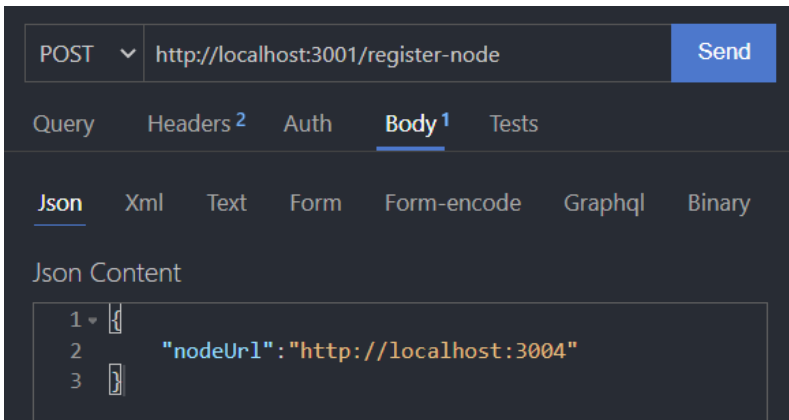
Gambar 42. Response Penambahan Node Baru

Masih pada ekstensi thunderbolt tambahkan juga node 3 kemudian tekan tombol **Send**.



*Gambar 43. Menambahkan Node 3 pada Node 1*

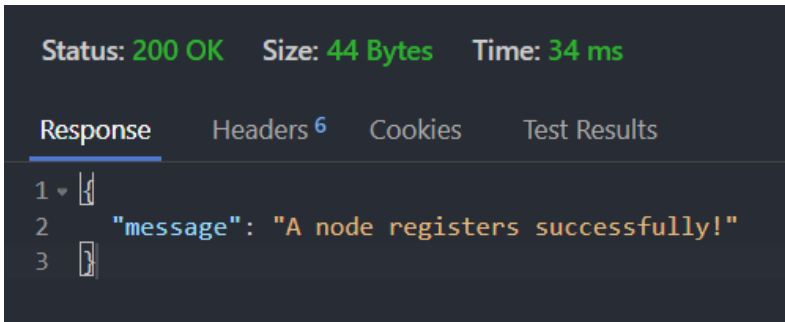
Kita juga akan menambahkan node 4 pada node 1, jadi masih pada ekstensi thunderbolt pada body tambahkan node 4 kemudian tekan **send**.



*Gambar 44. Menambahkan Node 4 pada Node 1*

Maka, respons json yang dihasilkan adalah sebagai berikut.

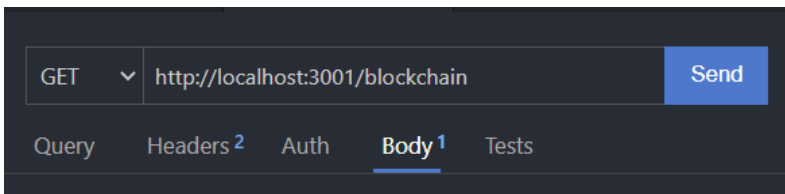




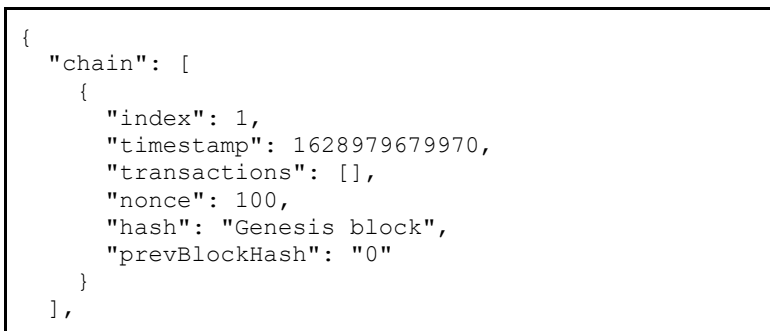
Gambar 45. Response JSON

## Cek Node Terdaftar pada Node 1

Untuk melakukan cek apakah node 2, node 3 dan node 4 sudah terdaftar pada jaringan node 1, kita ketikkan alamat <http://localhost:3001/blockchain> berikut dengan metode GET lalu tekan **Send**.



Gambar 46. Cek Jaringan Node 1

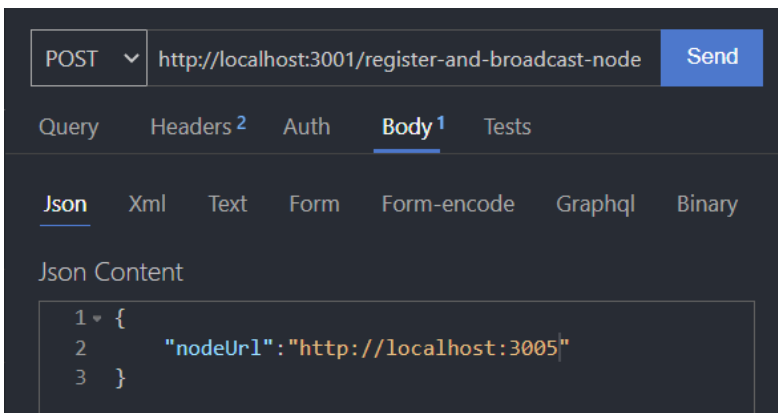


```
"pendingTransactions": [],
"nodeUrl": "http://localhost:3001",
"networkNodes": [
  "http://localhost:3002",
  "http://localhost:3003",
  "http://localhost:3004"
]
```

Lihatlah pada objek `networkNodes`, anda akan melihat bahwa node 2, node 3 dan node 4 sudah masuk pada jaringan node 1.

## Broadcasting Node

Implementasi selanjutnya adalah melakukan broadcast node 5 pada jaringan blockchain kita. Kita juga sudah mengetahui bahwa saat ini node 2, node 3 dan node 4 sudah masuk pada jaringan node 2. Nah kita akan mencoba menjalankan endpoint broadcast node berikut.



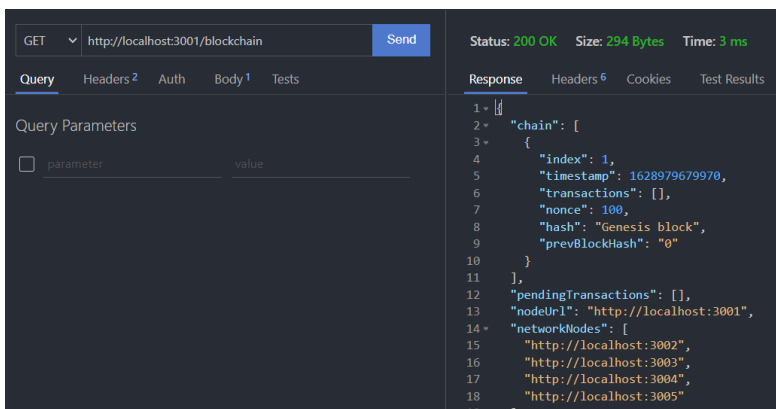
Gambar 47. Menggunakan Broadcast Node

Pada url thunderbolt masukan alamat url yaitu <http://localhost:3001/register-and-broadcast-node> gunakan metode POST. Kemudian pada body tambahkan nodeId untuk node 3005 sesuai pada Gambar 2.34. Setelah itu tekan tombol **send**. Jika benar anda akan mendapatkan response seperti berikut.

```
{
  "message": "A node registers with network successfully!"
}
```

Sekarang untuk mengecek pada setiap isi jaringan dari setiap nodes dapat dilakukan dengan mengetikan endpoint berikut ini.

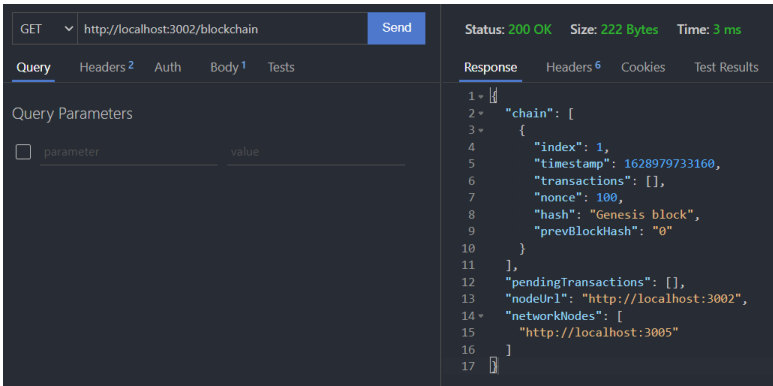
### Cek isi blockchain node 1



Gambar 48. Cek Isi Blockchain Node 1

Berdasarkan gambar di atas, terlihat bahwa sekarang node 5 telah masuk pada network nodes dari node 1.

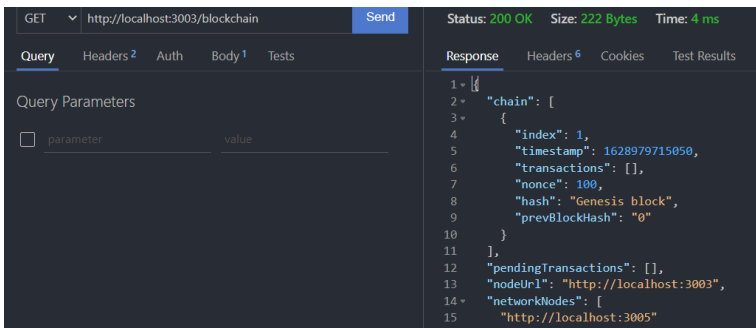
## Cek isi blockchain node 2



Gambar 49. Cek Isi Blockchain Node 2

Berdasarkan gambar, anda akan melihat bahwa node 5 dengan port 3005 sudah masuk pada jaringan node 2 dengan port 3002.

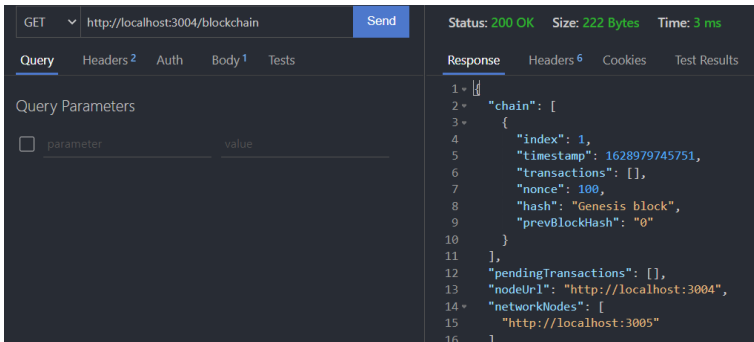
## Cek isi blockchain node 3



Gambar 50. Cek Isi Blockchain Node 3

Berdasarkan gambar, anda akan melihat bahwa node 5 dengan port 3005 sudah masuk pada jaringan node 3 dengan port 3003.

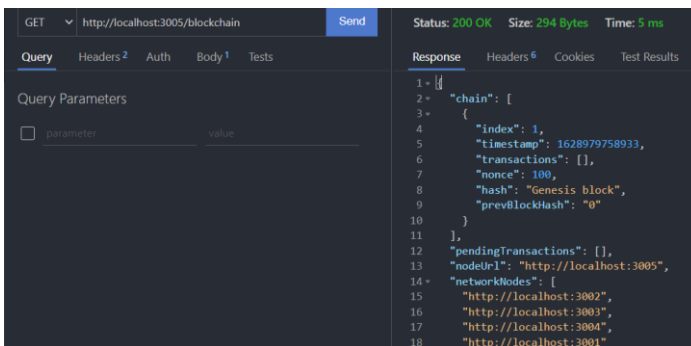
## Cek isi blockchain node 4



Gambar 51. Cek Isi Blockchain Node 4

Berdasarkan gambar, anda akan melihat bahwa node 5 dengan port 3005 sudah masuk pada jaringan node 4 dengan port 3004.

## Cek isi blockchain node 5



Gambar 52. Cek Isi Blockchain Node 5

Berdasarkan gambar, anda akan melihat bahwa node 5 dengan port 3005 sudah memiliki 4 jaringan nodes yaitu node 1, node 2, node 3 dan node 4.

#### **2.5.4 Sinkronisasi Jaringan Blockchain**

Anda sudah memahami bagaimana kita membuat jaringan terdesentralisasi dengan cara membuat beberapa node komputer yang dapat kita jalankan dalam satu waktu. Telah diimplementasikan oleh kita bersama dengan cara membuat 5 nodes yaitu node 1, node 2, node 3 , node 4 dan node 5. Semuanya masuk pada jaringan blockchain node 1.

Ketika jaringan terdesentralisasi sudah dibuat, tugas kita selanjutnya adalah melakukan sinkronisasi semua data pada blockchain. Maksudnya adalah ketika terjadi ada penambahan suatu blok maka semua node komputer memiliki salinan data yang sama. Semua data yang berhasil ditambang akan didistribusikan ke semua anggota jaringan blockchain.

Beberapa langkah yang harus kita lakukan agar semua node komputer dalam jaringan memiliki data salinan yang sama adalah sebagai berikut.

### 2.5.4.1 Mengubah Metode Transaksi

Pada file blockchain.js kita tambahkan **uuid()** dalam file utama. Modul ini akan kita gunakan untuk membuat id transaksi. Pada file blockchain.js tambahkan code untuk import uuid berikut setelah baris nodeUrl.

#### Kode blockchain.js

```
1. const sha256 = require('sha256');
2. const nodeUrl = process.argv[3];
3.
4. //tambahkan uuid untuk id transaksi
5. const { v4: uuidv4 } = require('uuid');
```

Masih pada file blockchain.js silahkan cari fungsi **makeNewTransaction** di class Blockchain, kemudian ubah dengan code berikut.

#### Kode blockchain.js

```
6. makeNewTransaction(amount, sender, recipient) {
7.   const transaction = {
8.     amount: amount,
9.     sender: sender,
10.    recipient: recipient,
11.    id: uuidv4().split('-').join('')
12.  }
13.
14.  return transaction;
15. }
16.
17. //tambahkan kode ini
18. addTransactionToPendingTransactions(transaction) {
19.   this.pendingTransactions.push(transaction);
20.
21.   return this.getLatestBlock().index + 1;
```

```
22. }
```

Keterangan:

- Baris 11 kita tambahkan key id pada objek transaction agar setiap transaksi memiliki id unik.
- Baris 18 - 21 kita tambahkan fungsi baru yaitu *addTransactionToPendingTransaction* dimana setiap transaksi masuk akan masuk pada *pendingTransaction* lalu mengembalikan blok terbaru + 1.

Sekarang kode blockchain.js secara keseluruhan dapat dilihat pada kode berikut ini.

#### Kode blockchain.js

```
1. const sha256 = require('sha256');
2. const nodeUrl = process.argv[3];
3. const { v4: uuidv4 } = require('uuid');
4.
5. class Block {
6.   constructor(index, timestamp, nonce,
7.     prevBlockHash, hash, transactions) {
8.     this.index = index;
9.     this.timestamp = timestamp;
10.    this.transactions = transactions;
11.    this.nonce = nonce;
12.    this.hash = hash;
13.    this.prevBlockHash = prevBlockHash;
14.  }
15. }
16. class Blockchain {
17.   constructor() {
18.     this.chain = [];
19.     this.pendingTransactions = [];
20.   }
```



```

21.         this.nodeUrl = nodeUrl;
22.         this.networkNodes = [];
23.
24.         this.creatNewBlock(100, '0', 'Genesis
    block');
25.     }
26.
27.     creatNewBlock(nonce, prevBlockHash, hash)
    {
28.         const newBlock = new Block(
29.             this.chain.length + 1,
30.             Date.now(),
31.             nonce,
32.             prevBlockHash,
33.             hash,
34.             this.pendingTransactions
35.         );
36.
37.         this.pendingTransactions = [];
38.         this.chain.push(newBlock);
39.
40.         return newBlock;
41.     }
42.
43.     getLatestBlock() {
44.         return this.chain[this.chain.length -
    1];
45.     }
46.
47.     //ubah kode ini
48.     makeNewTransaction(amount, sender,
    recipient) {
49.         const transaction = {
50.             amount: amount,
51.             sender: sender,
52.             recipient: recipient,
53.             id: uuidv4().split('-').join('')
54.         }
55.
56.         return transaction;
57.     }
58.
59.     //tambahkan kode ini
60.     addTransactionToPendingTransactions(transactio
    n) {
61.         this.pendingTransactions.push(transaction);

```

```

62.
63.         return this.getLatestBlock().index +
64.           1;
65.     }
66.     hashBlock(prevBlockHash, currentBlock,
67. nonce) {
68.         const data = prevBlockHash +
69.           JSON.stringify(currentBlock) + nonce;
70.         const hash = sha256(data);
71.         return hash;
72.     }
73.     proofOfWork(prevBlockHash,
74. currentBlockData) {
75.         let nonce = 0;
76.         let hash =
77.           this.hashBlock(prevBlockHash,
78.             currentBlockData, nonce);
79.         while (hash.substring(0, 2) !== '00')
80.           {
81.             nonce++;
82.             hash =
83.               this.hashBlock(prevBlockHash,
84.                 currentBlockData, nonce);
85.           };
86.         return nonce;
87.     }
88. }
89. module.exports = Blockchain;

```

#### 2.5.4.2 Endpoint Transaction

Kita juga akan membuat 3 buah endpoint yaitu transaction, broadcast transaction add block dan mine. Pada file api.js kita ubah end point transaction berikut.

**Kode api.js**

```

1. app.post('/transaction', function (req, res) {
2.     const transaction = req.body;
3.     const blockIndex =
       bitcoin.addToPendingTransactions(transaction);
4.
5.     res.json(
6.         {
7.             message: `Transaction will be
           added to block with index: ${blockIndex}`
8.         }
9.     );
10. });

```

#### Keterangan Kode:

- Baris ke 2 kita buat variabel transaction dimana akan menerima inputan pada properties body berupa transaksi misal sender A mengirim ke recipient B sebesar 1 BTC.
- Baris 3 buat sebuah variabel untuk melihat blockIndex dengan memanggil fungsi yang menambahkan transaksi agar masuk terlebih dahulu pada objek pendingTransaction sebelum ada penambahan.

Agar setiap transaksi dapat tersalin pada semua anggota jaringan blockchain atau setiap node komputer memiliki data yang terdistribusi maka kita akan membuat fungsi berupa endpoint yang bertugas

melakukan itu semua yaitu broadcast. Masih pada file api.js, tambahkan kode berikut.

#### Kode api.js

```
1. app.post('/transaction/broadcast', function (req, res) {
2.     const transaction = bitcoin.makeNewTransaction(
3.         req.body.amount,
4.         req.body.sender,
5.         req.body.recipient
6.     );
7.
8.     bitcoin.addTransactionToPendingTransactions(transaction);
9.
10.    const requests = [];
11.    bitcoin.networkNodes.forEach(networkNode => {
12.        const requestOptions = {
13.            uri: networkNode + '/transaction',
14.            method: 'POST',
15.            body: transaction,
16.            json: true
17.        };
18.        requests.push(reqPromise(requestOptions));
19.    });
20.
21.    Promise.all(requests)
22.        .then(data => {
23.            res.json(
24.                {
25.                    message: `Creating and broadcasting
Transaction successfully!`
26.                }
27.            );
28.        });
29. });
```

#### Keterangan Kode:

- Baris 1 kita membuat endpoint dengan nama /transaction/broadcast

- Baris 2 - 5 objek transaction berisi transaksi baru yang berisi pengirim, penerima dan nilai transaksi.
- Baris 7 jalankan fungsi pendingTransaction agar transaksi masuk pada data pending.
- Baris 10 - 16 lakukan iterasi atau perulangan dengan menyimpan data tersebut pada request dimana akan menggunakan request promise isinya adalah menjalankan endpoint untuk menjalankan transaksi pada semua network node. Misal transaksi terjadi pada node 1, maka semua anggota jaringannya yaitu node 2, node 3 dan node 4 akan mendapatkan request yang sama.
- Baris 21 - 27 menjalankan promise pada semua request jika request berhasil maka dimunculkan pesan bahwa broadcast ini berhasil.

#### 2.5.4.3 Menambahkan fungsi Add Block

Semua data transaksi yang sudah masuk pada data pending akan selamanya masuk pada pending transaction jika tidak ada penambangan oleh miners. Kita akan membuat fungsi untuk menambahkan blok

baru ketika ada penambahan. Pada kode api.js tambahkan kode berikut.

#### Kode api.js

```
1. app.post('/add-block', function (req, res) {
2.   const block = req.body.newBlock;
3.   const latestBlock = bitcoin.getLatestBlock();
4.
5.   if ((latestBlock.hash === block.prevBlockHash)
6.     && (block.index === latestBlock.index + 1)) {
7.     bitcoin.chain.push(block);
8.     bitcoin.pendingTransactions = [];
9.
10.    res.json(
11.      {
12.        message: 'Add new Block successfully!',
13.        newBlock: block
14.      }
15.    );
16.  } else {
17.    res.json(
18.      {
19.        message: 'Cannot add new Block!',
20.        newBlock: block
21.      }
22.    );
23.  }
24. });
```

#### Keterangan Kode:

- Baris 1 kita membuat endpoint untuk menambah blok baru jika ada penambahan
- Baris 2 variable block akan menyimpan data blok baru yang akan diinputkan
- Baris 3 mengambil blok terakhir pada blockchain

- Baris 5 - 22 jika blok terakhir sama dengan blok sebelumnya dan index blok sama dengan index blok terakhir maka tambahkan data blok baru pada blockchain, kemudian kosongkan nilai pending transaction. Jika berhasil berikan pesan bahwa blok baru berhasil ditambahkan dan jika gagal tampilkan pesan kesalahan.

#### 2.5.4.4 Penambangan dan Broadcast

Kita telah membuat fungsi untuk menambahkan blok baru setelah ada penambangan kedalam blockchain. Nah kita akan menyempurnakan end point mine pada file api.js. Ketika ada penambangan dan berhasil maka blok baru akan bertambah. Oleh karena itu kita akan mengubah endpoint /mine. Pada api.js silahkan ubah kode api.js berikut.

##### File api.js

```
1. app.get('/mine', function (req, res) {
2.     const latestBlock = bitcoin.getLatestBlock();
3.     const prevBlockHash = latestBlock.hash;
4.     const currentBlockData = {
5.         transactions: bitcoin.pendingTransactions,
6.         index: latestBlock.index + 1
7.     }
8.     const nonce = bitcoin.proofOfWork(prevBlockHash,
```

```

    currentBlockData);
9.     const blockHash = bitcoin.hashBlock(prevBlockHash,
      currentBlockData, nonce);
10.
11.     const newBlock = bitcoin.creatNewBlock(nonce,
      prevBlockHash, blockHash)
12.
13.     const requests = [];
14.     bitcoin.networkNodes.forEach(networkNode => {
15.         const requestOptions = {
16.             uri: networkNode + '/add-block',
17.             method: 'POST',
18.             body: { newBlock: newBlock },
19.             json: true
20.         };
21.
22.         requests.push(reqPromise(requestOptions));
23.     });
24.
25.     Promise.all(requests)
26.         .then(data => {
27.             // reward for mining
28.             const requestOptions = {
29.                 uri: bitcoin.nodeUrl +
      '/transaction/broadcast',
30.                 method: 'POST',
31.                 body: {
32.                     amount: 1,
33.                     sender: '00000',
34.                     recipient: nodeAddr
35.                 },

```



```

36.         json: true
37.     };
38.
39.         return reqPromise(requestOptions);
40.     })
41.     .then(data => {
42.         res.json(
43.             {
44.                 message: 'Mining & broadcasting new
Block successfully!',
45.                 newBlock: newBlock
46.             }
47.         );
48.     });
49. });

```

#### Keterangan Kode:

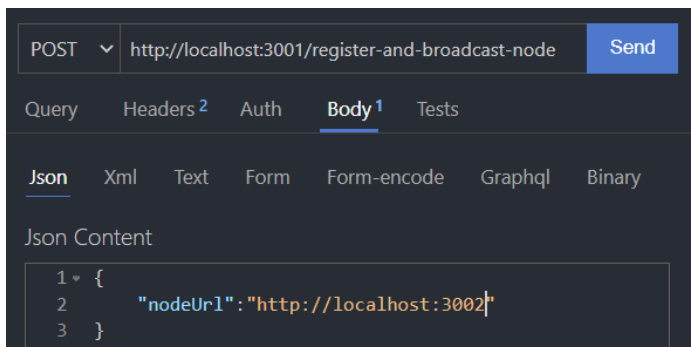
- Baris 2 mengambil data blok terbaru
- Baris 3 mengambil hash dari blok sebelumnya
- Baris 4 objek dari data terbaru yang isinya adalah data transaksi pending dan index dari blok tersebut + 1.
- Baris 8 mengambil nilai nonce dengan PoW
- Baris 9 melakukan hashing dari hash blok sebelumnya dengan data blok saat ini dan nilai nonce.

- Baris 11 buat blok baru dengan fungsi `createNewBlock` yang telah kita buat sebelumnya.
- Baris 13 buat sebuah array request.
- Baris 14 - 23 jalankan broadcast ke seluruh jaringan blockchain ketika ada penambahan blok baru dari hasil proses mining dan simpan pada array request.
- Baris 25 -39 jalankan promise kepada semua data request dimana kita akan memberikan reward dengan mengirimkan sejumlah 1 koin kepada penambang. Broadcast kembali jika pemberian reward ini sukses dengan tujuan bahwa penambangan telah berhasil, dan pada penambang harus mencari blok lain yang harus ditambang.
- Baris 44 berikan pesan bahwa penambangan block telah selesai berhasil disiarkan ke seluruh anggota jaringan.

#### 2.5.4.5 Menjalankan Sinkronisasi Jaringan

Sekarang saatnya kita melakukan uji coba untuk melakukan sinkronisasi jaringan blockchain. Beberapa langkah nya antara lain adalah sebagai berikut:

1. Jalankan semua node server dengan mengaktifkan node 1 hingga node 4 dengan cara menjalankan perintah berikut pada terminal visual studio code atau command prompt;
  - a. `npm run node 1`
  - b. `npm run node 2`
  - c. `npm run node 3`
  - d. `npm run node 4`
  
2. Register dan broadcast node 2, node 3 dan node 4 pada node 1. Dengan menggunakan url endpoint yaitu <http://localhost:3001/register-and-broadcast-node> pada ekstensi thunderbolt visual studio code.



*Gambar 53. Menambahkan jaringan ke dalam node 1*

Lakukan juga pada node 3 dan 4, jika berhasil kalian akan mendapatkan response kode berikut.

```
Status: 200 OK Size: 57 Bytes Time: 38 ms
Response Headers 6 Cookies Test Results
1. [
2. "message": "A node registers with network successfully!"
3. ]
```

Gambar 54. Sukses Broadcast Jaringan Baru

3. Jika anda berhasil melakukan penambahan jaringan dan broadcastnya, maka ketika anda menjalankan endpoint <http://localhost:3001/blockchain> maka menghasilkan response JSON sebagai berikut:

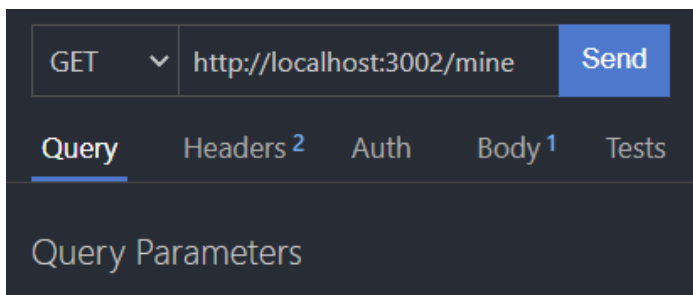
```
1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628996498806,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    }
11.  ],
12.  "pendingTransactions": [],
13.  "nodeUrl": "http://localhost:3001",
14.  "networkNodes": [
15.    "http://localhost:3002",
16.    "http://localhost:3003",
17.    "http://localhost:3004"
18.  ]
19. }
```

Silahkan cek kembali pada endpoint lain misal node 2 dengan alamat url <http://localhost:3002/blockchain> maka akan mendapatkan response JSON serupa. Jangan lupa untuk mengecek node 3 dan node 4.

```
1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628996498596,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    }
11.  ],
12.  "pendingTransactions": [],
13.  "nodeUrl": "http://localhost:3002",
14.  "networkNodes": [
15.    "http://localhost:3001",
16.    "http://localhost:3003",
17.    "http://localhost:3004"
18.  ]
19. }
```

4. Lakukan proses penambangan dari node 2, jadi kami akan simulasikan bahwa node sebagai

penambang. Silahkan ketikkan url <http://localhost:3002> pada thunderbolt dengan metode GET, lalu klik send.



*Gambar 55. Proses Mining oleh Node 2*

Jika berhasil maka response dari proses penambangan oleh node 2 adalah sebagai berikut.

```
1. {
2.   "message": "Mining & broadcasting new Block
   successfully!",
3.   "newBlock": {
4.     "index": 2,
5.     "timestamp": 1628998854216,
6.     "transactions": [],
7.     "nonce": 44,
8.     "hash":
   "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa
   23e55dfe964",
9.     "prevBlockHash": "Genesis block"
10.  }
11. }
```

5. Cek kembali pada url <http://localhost:3001/blockchain>, <http://localh>

[ost:3002/blockchain](http://localhost:3002/blockchain), <http://localhost:3003/blockchain>, dan <http://localhost:3004/blockchain> pada thunderbolt dengan metode GET. Anda akan melihat penambahan blok baru dan rewards bari penambang.

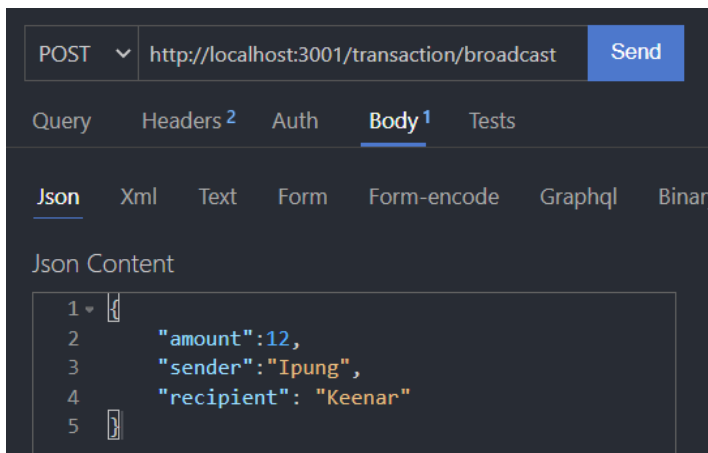
```
1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628996498806,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    },
11.    {
12.      "index": 2,
13.      "timestamp": 1628998854216,
14.      "transactions": [],
15.      "nonce": 44,
16.      "hash":
17.      "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa23e
18.      55dfe964",
19.      "prevBlockHash": "Genesis block"
20.    }
21.  ],
22.  "pendingTransactions": [
23.    {
24.      "amount": 1,
25.      "sender": "00000",
26.      "recipient": "7df444ba-3855-436f-b3a0-
27.      af14d34f71e5",
28.      "id": "ace6bdfb30154aa4a46a2c12e168f910"
29.    }
30.  ],
31.  "nodeUrl": "http://localhost:3001",
32.  "networkNodes": [
33.    "http://localhost:3002",
34.    "http://localhost:3003",
35.    "http://localhost:3004"
36.  ]
37. }
```

Jika diperhatikan sekarang muncul index 2 artinya sudah ada blok baru pada jaringan blockchain. Selain itu juga ada pending transaction berupa reward sebesar 1 koin kepada penambang.

6. Buatlah transaksi baru dengan cara melakukan broadcasting, sebagai contoh saya akan membuat transaksi baru pada node 1 dengan endpoint

<http://localhost:3001/transaction/broadcast>.

Masukan 1 transaksi pada property body. Misalkan kami akan mengirimkan transaksi berikut.



Gambar 56. Broadcast Transaksi Baru



Klik send dan anda akan mendapatkan respon bahwa transaksi berhadil dijalankan.

```
{
  "message": "Creating and broadcasting
Transaction successfully!"
}
```

7. Cek transaksi yang baru masuk tersebut apakah benar masuk pada pending transaction atau tidak. Anda bisa mengeceknya dengan cara membuka url <http://localhost:3002/blockchain> dengan menggunakan GET seharusnya anda mendapatkan respons seperti berikut.

```
1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628996498596,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    },
11.    {
12.      "index": 2,
13.      "timestamp": 1628998854216,
14.      "transactions": [],
15.      "nonce": 44,
16.      "hash":
17.      "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa23e5
18.      5dfe964",
19.      "prevBlockHash": "Genesis block"
```

```

18.   }
19. ],
20. "pendingTransactions": [
21.   {
22.     "amount": 1,
23.     "sender": "00000",
24.     "recipient": "7df444ba-3855-436f-b3a0-af14d34f71e5",
25.     "id": "ace6bdfb30154aa4a46a2c12e168f910"
26.   },
27.   {
28.     "amount": 12,
29.     "sender": "Ipung",
30.     "recipient": "Keenar",
31.     "id": "361411be1de242d39d4cbe057afb1c80"
32.   }
33. ],
34. "nodeUrl": "http://localhost:3002",
35. "networkNodes": [
36.   "http://localhost:3001",
37.   "http://localhost:3003",
38.   "http://localhost:3004"
39. ]
40. }

```

8. Sekarang agar tidak masuk pada pending transaksi kita harus melakukan penambangan. Pada demo kali ini node 4 akan menjadi penambang. Silahkan masukan endpoint berikut pada thunderbolt <http://localhost:3004/mine> lalu tekan send. Jika berhasil maka akan muncul response berikut.

```

1. {
2.   "message": "Mining & broadcasting new Block
   successfully!",
3.   "newBlock": {
4.     "index": 3,
5.     "timestamp": 1628999672031,
6.     "transactions": [

```

```

7.     {
8.         "amount": 1,
9.         "sender": "00000",
10.        "recipient": "7df444ba-3855-436f-b3a0-
af14d34f71e5",
11.        "id": "ace6bdfb30154aa4a46a2c12e168f910"
12.    },
13.    {
14.        "amount": 12,
15.        "sender": "Ipung",
16.        "recipient": "Keenar",
17.        "id": "361411be1de242d39d4cbe057afb1c80"
18.    }
19. ],
20. "nonce": 99,
21. "hash":
"0024994db657aa3db3bf0d98820b233dd9f65c9f5180a2c3be8767453
aaf6406",
22. "prevBlockHash":
"00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa23e5
5dfe964"
23. }
24. }

```

9. Kita akan membuktikan semua data apakah telah disalin pada seluruh anggota jaringan atau belum. Dengan mengetikan alamat <http://localhost:3001/blockchain> atau <http://localhost:3002/blockchain> seharusnya memiliki salinan data yang sama. Pada thunderbolt silahkan masukan alamat tersebut lalu tekan tombol send. Response yang seharusnya terjadi adalah sebagai berikut.

```

1. {
2.   "chain": [
3.     {
4.       "index": 1,
5.       "timestamp": 1628996498806,
6.       "transactions": [],
7.       "nonce": 100,
8.       "hash": "Genesis block",
9.       "prevBlockHash": "0"
10.    },
11.    {
12.      "index": 2,
13.      "timestamp": 1628998854216,
14.      "transactions": [],
15.      "nonce": 44,
16.      "hash":
17.      "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa23e
18.      55dfe964",
19.      "prevBlockHash": "Genesis block"
20.    },
21.    {
22.      "index": 3,
23.      "timestamp": 1628999672031,
24.      "transactions": [
25.        {
26.          "amount": 1,
27.          "sender": "00000",
28.          "recipient": "7df444ba-3855-436f-b3a0-
29.          af14d34f71e5",
30.          "id": "ace6bdfb30154aa4a46a2c12e168f910"
31.        },
32.      ],
33.    },
34.  ],
35. }

```

```
29.     {
30.         "amount": 12,
31.         "sender": "Ipung",
32.         "recipient": "Keenar",
33.         "id": "361411be1de242d39d4cbe057afb1c80"
34.     }
35. ],
36.     "nonce": 99,
37.     "hash":
    "0024994db657aa3db3bf0d98820b233dd9f65c9f5180a2c3be876745
    3aaf6406",
38.     "prevBlockHash":
    "00669af06fa8d395bea4f6325d39f43a3c87788152fc3050608aa23e
    55dfe964"
39. }
40. ],
41. "pendingTransactions": [
42.     {
43.         "amount": 1,
44.         "sender": "00000",
45.         "recipient": "850fce21-87ac-433a-be23-
    5ea0c5d47002",
46.         "id": "d669027582884182a5e9ea2acfd710ee"
47.     }
48. ],
49. "nodeUrl": "http://localhost:3001",
50. "networkNodes": [
51.     "http://localhost:3002",
52.     "http://localhost:3003",
53.     "http://localhost:3004"
54. ]
55. }
```

## **PENUTUP**

Sampai disini kita telah mempelajari tentang pengantar teknologi blockchain. Banyak hal yang sudah disampaikan, mulai dari motivasi dibuatnya blockchain, proses penambangan, sinkronisasi data hingga kita mencoba mengimplementasikannya sendiri konsep tersebut dengan memanfaatkan bahasa pemrograman javascript.

Penulis berharap anda dapat memahami apa yang telah disampaikan dengan baik dan benar. Buku ini merupakan edisi pertama yang akan terus dilanjutkan hingga implementasi nyata menggunakan blockchain seperti ethereum, binance dan sebagainya. Semoga buku ini bermanfaat, Selamat belajar.

## DAFTAR PUSTAKA

- Lee, W. M. (2019). Beginning ethereum smart contracts programming. *With Examples in Python, Solidity and JavaScript*.
- Antonopoulos, A. M., & Wood, G. (2018). *Mastering ethereum: building smart contracts and dapps*. O'reilly Media.
- Bashir, I. (2017). *Mastering blockchain*. Packt Publishing Ltd.
- Dannen, C. (2017). *Introducing Ethereum and solidity* (Vol. 1, pp. 159-160). Berkeley: Apress.
- Vujičić, D., Jagodić, D., & Randić, S. (2018, March). Blockchain technology, bitcoin, and Ethereum: A brief overview. In *2018 17th international symposium infoteh-jahorina (infoteh)* (pp. 1-6). IEEE.
- Riadi, I., Ahmad, T., Sarno, R., Purwono, P., & Ma'arif, A. (2020). Developing Data Integrity in an Electronic Health Record System using Blockchain and InterPlanetary File System (Case Study: COVID-19 Data).
- Segendorf, B. (2014). What is bitcoin. *SverigesRiksbankEconomicReview*, 2014, 2-71.

# PENGANTAR TEKNOLOGI BLOCKCHAIN

Dilengkapi dengan praktikum pembuatan  
konseptual blockchain dengan javascript

Teknologi blockchain terus berkembang dan menjadi populer. Belum banyak buku-buku yang membahas secara detail terkait teknologi ini. Umumnya orang hanya mengenal blockchain itu identik dengan mining mata uang digital atau melakukan tradingnya saja, padahal sebetulnya teknologi dibalik cryptocurrency ini memiliki manfaat yang lebih banyak

Disusun Oleh  
Purwono, S.Kom., M.Kom



Penerbit **UHB Press**

ISBN 978-623-88102-0-8

