

BUILD A

# BLOCKCHAIN FROM SCRATCH

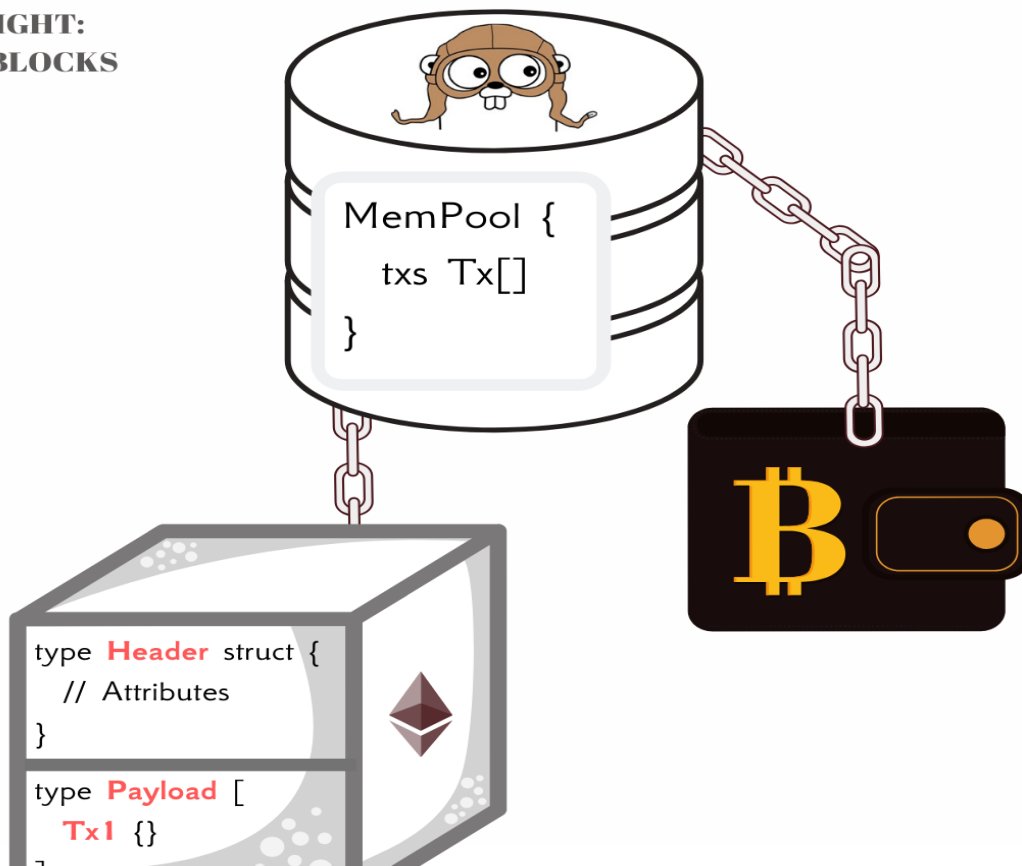
IN GO

Escape Crowded Java/PHP Job Market

Pioneer an Innovative Technology

Expand Your Dev Career

HEIGHT:  
51 BLOCKS



Lukas Lukac

# **Build a Blockchain from Scratch in Go**

Lukas Lukac

© 2020 Lukas Lukac | Version: 1.10 | Last Updated: 30.05.2020 -  
Base Version

# Contents

<b>Foreword</b>	<b>1</b>
<b>Why Start Blockchain Development?</b>	<b>2</b>
<b>Getting Started</b>	<b>6</b>
<b>01   The MVP Database</b>	<b>11</b>
User 1, Andrej	12
<b>02   Mutating Global DB State</b>	<b>16</b>
Dead Party	16
Bonus for BabaYaga	18
<b>03   Monolithic Event vs Transaction</b>	<b>20</b>
Andrej Programming	21
Building a Command-Line-Interface (CLI)	29
<b>04   Humans Are Greedy</b>	<b>43</b>
Typical business greediness	43
<b>05   Why We Need Blockchain</b>	<b>48</b>
BabaYaga Seeks Justice	48
<b>06   L'Hash de Immutable</b>	<b>53</b>
How to Program an Immutable Database?	53
Immutability via Hash Functions	55
Implementing the DB Content Hashing	57

## CONTENTS

<b>07   The Blockchain Programming Model</b>	<b>64</b>
Improving Performance of an Immutable DB	64
Batch + Hash + Linked List $\Rightarrow$ Blocks	66
How adding a TX into a Block works	70
Migrating from TX.db to BLOCKS.db	73
<b>08   Transparent Database</b>	<b>85</b>
Flexible DB Directory	85
Centralized Public HTTP API	91
Deploying TBB Program to AWS	98
Burned-out	102
<b>Unlock All Chapters</b>	<b>105</b>

# Foreword

Hi, you've made a great decision to expand your programming career!

Blockchain is a paradigm shift in software development and architecture. I recommend everyone learn and master it!

Through learning blockchain, you will explore:

- Peer-to-peer systems software architecture
- Event-based architecture
- How servers can communicate autonomously (BTC, ETH, XRP)
- Go programming language ♥
- Encoding and secure hashing
- Asymmetric cryptography and general internet security



I share and document the entire process and progress on:

<https://twitter.com/Web3Coach><sup>1</sup>

---

<sup>1</sup><https://twitter.com/Web3Coach>

# Why Start Blockchain Development?

Two years ago, I was at a crossroads in my career. I was looking for my next programming job after five years of PHP development at travago.

I was deciding between a Java position in NewRelic or a GoLang position at a new blockchain startup. Yep. I went with Go and blockchain. Why? Because it's better for my long-term career. And I recommend you do the same.

If I'd continued in Java development, I'd have kept programming monoliths, occasionally learning something here and there. The majority of the time, I would have been working on autopilot, fixing bugs, and maybe implementing a nice feature every second sprint. At best, it would have been a good, safe, and interesting choice, but a bit slow for my personal taste.

At its worst, I'd be working on broken micro-services :) (Just joking, they are great for scaling individual parts of the project and dividing the ownership between independent teams.)

Aside from the exciting opportunities, there is also a financial aspect to every job. We all have bills to pay. In the Java/PHP/Javascript world, I would compete for a salary raise with 10 million other Java

developers, each with 20 years of experience. Certainly doable, but it's an uphill battle that gets harder by the day.

Web 3.0 is coming. In the last ten years, we've helped build large platforms for mass communication. Life updates? Facebook. Story to tell? Medium. Picture to share? Instagram. Unfortunately, these platforms have a dark side. And this dark side powers their entire business model. Users have given up their data sovereignty and lost attention to advertisements.

But it doesn't have to be like this.

Do you remember when RSS was king, and everyone had a blog? Web3.0 aims to rebuild and improve this vision. Join me and explore the new **tokenized economies** optimizing the value exchange between participants by entirely removing an intermediary. Learn **blockchain immutability**, combine it with **asymmetric cryptography**, and **develop new transparent, open-sourced systems that users can trust** while preventing centralized data breaches.

This book will show you precisely that.

## **Meet the book's main character. Andrej.**

Andrej is a bar owner by night and a software developer by day in a small Slovakian town called Bardejov.

Andrej is tired of:

- **Programming solid, old fashion PHP/Java applications**
- Forgetting how much money his friends and clients owe him for all the unpaid Friday nights vodka shots
- Spending time collecting and counting coins, returning change and generally touching COVID-19 bank bills
- Maintaining different plastic chips, tokens for table football, darts, billiard and poker

Andrej would love to:

- **Have a perfect auditable history of the bar's activities** and sales to make his bar compliant with tax regulations
- **Transform his bar into an autonomous, payment-efficient and safe environment his customers can trust**

"This will be a programming dream!" he tells himself. "I am going to write a simple program and keep all the balances of my clients in a virtual form.

"Every new customer will give me cash, and **I will credit them an equivalent amount of my digital tokens.** The tokens will represent a monetary unit within and outside the bar.



“The users will use the tokens for all bar functionalities from paying for drinks, borrowing and lending them to their friends, and playing table tennis, poker and kicker.

“I will call the tokens: The Blockchain Bar tokens, **TBB!**”

# Getting Started

The book is written in Go, but don't worry - you don't need to have any prior Go experience to start reading the book. It's a very powerful and beginner friendly language and you will pick it up quickly.

## Requirements

I recommend 2+ years of programming experience in Java / PHP / Javascript, or another language similar to Go.

Complete the free [17 lectures of A Tour Of Go](https://tour.golang.org/basics/1)<sup>2</sup> to get familiar with the syntax and basic concepts.

Checkout this nice free [overview of different Go functions](https://blog.learngoprogramming.com/go-functions-overview-anonymous-closures-higher-order-deferred-concurrent-6799008dde7b)<sup>3</sup> from Inanc Gumus.

---

<sup>2</sup> <https://tour.golang.org/basics/1>

<sup>3</sup> <https://blog.learngoprogramming.com/go-functions-overview-anonymous-closures-higher-order-deferred-concurrent-6799008dde7b>

## Why Go?

Because like blockchain, it's a fantastic technology for your overall programming career. Go is a trendy language and better paid than an average Java/PHP position.

Go is optimized for multi-core CPU architecture. You can spawn thousands of light-weight threads (Go-routines) without problems. It's extremely practical for highly parallel and concurrent software such as blockchain networks. By writing your software in Go, you achieve nearly C++ level of performance out of the box without killing yourself for that one time you forgot to free-up memory.

Go also compiles to binary which makes it very portable.

## Setup the project

### Join the private Discord chat

Join the The Blockchain Bar chat server <https://discord.gg/F2e2Qfz><sup>4</sup> and send me the purchased book's GUMROAD LICENSE KEY by a DM to Web3Coach#9926 (my Discord username).

I will add you to a **private students room** where I will be individually supporting you on your new blockchain journey.

### Clone the Github repository

↓ Visit the Github repository and follow the instructions ↓

<https://github.com/web3coach/the-blockchain-bar><sup>5</sup>

---

<sup>4</sup> <https://discord.gg/F2e2Qfz>

<sup>5</sup> <https://github.com/web3coach/the-blockchain-bar>

## Installation

---

### Install Go

Follow the official docs or use your favorite dependency manager to install Go: <https://golang.org/doc/install>

Verify your `$GOPATH` is correctly set before continuing.

### Setup this Repository

Go is bit picky about where you store your repositories.

The convention is to store:

- the source code inside the `$GOPATH/src`
- the compiled program binaries inside the `$GOPATH/bin`

You can `clone` the repository or use `go get` to install it.

### Using Git

```
mkdir -p $GOPATH/src/github.com/web3coach
cd $GOPATH/src/github.com/web3coach

git clone git@github.com:web3coach/the-blockchain-way-of-programming-newsletter-edition.git
```

PS: Make sure you actually clone it inside the `src/github.com/web3coach` directory, not your own, otherwise it won't compile. Go rules.

### Using Go Get

```
go get -u github.com/web3coach/the-blockchain-way-of-programming-newsletter-edition
```

## Getting Started

---

1. Open the eBook at Chapter 1
2. Checkout the first chapter's branch

```
git checkout c1_genesis_json
```

Read, experiment with the code and, most importantly, have fun!

## Getting Unstuck

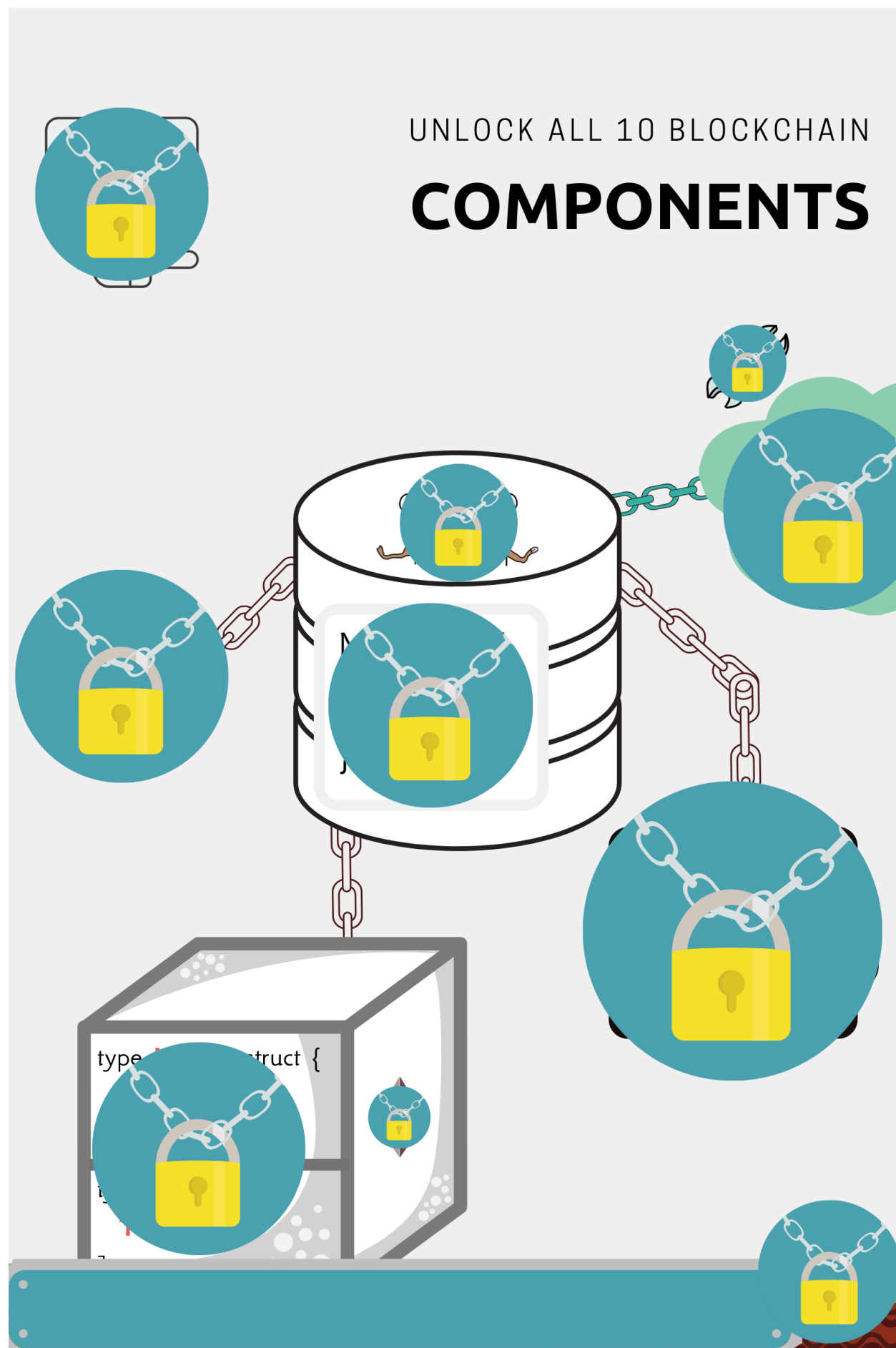
---

Can't understand why is something done in a particular way or crack your way around a specific chapter's topic?

Blockchain is a challenging technology.

As I promised, you have my full support. You are not alone in this. Write me a DM on LinkedIn, and I will help you figure it out and move forward on your new journey :)

---



# 01 | The MVP Database

```
>_ git checkout c1_genesis_json
```

Andrej mastered relational SQL databases in the 90s. He knows how to make advanced data models and how to optimize the SQL queries.

It's time for Andrej to catch-up with innovation and start building Web 3.0 software.

Luckily, after reading “The Lean Startup” book last week, Andrej feels like he shouldn't over-engineer the solution just yet. Hence, he chooses a simple but effective, JSON file for the bar's MVP database.

In the beginning, there was a primitive centralized database.



**Blockchain is a database.**

## User 1, Andrej

*Monday, March 18.*

Andrej generates 1M utility tokens.

In the blockchain world, tokens are units inside the blockchain database. Their real value in dollars or euro fluctuates based on their demand and popularity.

Every blockchain has a **“Genesis”** file. The Genesis file is used to distribute the first tokens to early blockchain participants.

It all starts with a simple, dummy, **genesis.json**.

Andrej creates the file `./database/genesis.json` where he defines that The Blockchain Bar’s database will have 1M tokens and all of them will belong to Andrej:

```
{
  "genesis_time": "2019-03-18T00:00:00.000000000Z",
  "chain_id": "the-blockchain-bar-ledger",
  "balances": {
    "andrej": 1000000
  }
}
```

The tokens need to have a real “utility”, i.e., a use case. Users should be able to pay with them from day 1! Andrej must comply with law regulators (SEC). It is illegal to issue unregistered security.



On the other hand, utility tokens are fine, so he right away prints and sticks a new pricing white paper poster on the bar's door.

Andrej assigns a starting monetary value to his tokens so he can exchange them for euro, dollars or other fiat currency.

1 TBB token = 1€

Item	Price
-----	-----
Vodka shot	1 TBB
Orange juice	5 TBB
Burger	2 TBB
Crystal Head Vodka Bottle	950 TBB

Andrej also decides, **he should be getting 100 tokens per day** for maintaining the database and having such a brilliant disruptive idea.



## Fun Facts

The first genesis Ether (ETH) on Ethereum blockchain was created and distributed to early investors and developers in the same way as Andrej's utility token.

In 2017, during an ICO (initial coin offerings) boom on the Ethereum blockchain network, project founders wrote and presented whitepapers to investors. A whitepaper is a technical document outlining a complex issue and possible solution, meant to educate and elucidate a particular matter. In the world of blockchains, a white paper serves to outline the specifications of how that particular blockchain will look and behave once it is developed.

Blockchain projects raised between €10M to €300M per **whitepaper** idea.

in exchange for money (the ICO "funding"), investor names would be included in the initial "genesis balances", similar to how Andrej did it. Investors' hopes through an ICO are the genesis coins go up in value and that the teams deliver the outlined blockchain.

Naturally, not all whitepaper ideas come to fruition. Massive investments lost to unclear or incomplete ideas are why blockchain received negative coverage in the media throughout these ICOs, and why some still considered it a hype. But the underlying blockchain technology is fantastic and useful, as you will learn further in this book. It's just been abused by some bad actors.



## Summary

Blockchain is a database.

**The token supply, initial user balances, and global blockchain settings you define in a Genesis file.**



## Study Code

Commit: [c6b8eb](#)<sup>6</sup>

---

<sup>6</sup> <https://github.com/web3coach/the-blockchain-bar/commit/c6b8eb3e889550f15be59e659f54970a87b3e94a>

## 02 | Mutating Global DB State

```
>_ git checkout c2_db_changes_txt
```

### Dead Party

*Monday, March 25.*

After a week of work, the bar facilities are ready to accept tokens. Unfortunately, no one shows up, so Andrej orders three shots of vodka for himself and writes the database changes on a piece of paper:

```
andrej-3;    // 3 shots of vodka  
andrej+3;    // technically purchasing from his own bar  
andrej+700;  // Reward for a week of work (7x100 per day)
```

To avoid recalculating the latest state of each customer's balance, Andrej creates a `./database/state.json` file storing the balances in an aggregated format.

New DB state:

```
{  
  "balances": {  
    "andrej": 1000700  
  }  
}
```

## Bonus for BabaYaga

*Tuesday, March 26.*

To bring traffic to his bar, Andrej announces an exclusive 100% bonus for everyone who purchases the TBB tokens in the next 24 hours.

Bing! He gets his first customer called **BabaYaga**. BabaYaga pre-purchases 1000€ worth of tokens, and to celebrate, she immediately spends 1 TBB for a vodka shot. She has a drinking problem.

DB transactions written on a piece of paper:

```
andrej-2000;    // transfer to BabaYaga
babayaga+2000; // pre-purchase with 100% bonus
babayaga-1;
andrej+1;
andrej+100;    // 1 day of sun coming up
```

New DB state:

```
{
  "balances": {
    "andrej": 998801,
    "babayaga": 1999
  }
}
```



## Fun Facts

Blockchain ICO (initial coin offerings based on whitepapers) projects often distribute the genesis tokens with different bonuses, depending on how many of them you buy and how early you do it. Teams offer, on average, 10-40% bonuses to early “participants”.

The word “investor” is avoided, so the law regulators won’t consider the tokens being a security. Projects would reason their main product, blockchain tokens, function as “flying, loyalty points.”

The “participants” later made even 1000% (four zeroes!) on their investment selling to the public through an exchange several months later.



## Summary

Blockchain is a database. The token supply, initial user balances, and global blockchain settings you define in a Genesis file. **The Genesis balances indicate what was the original blockchain state and are never updated afterwards.**

**The database state changes are called Transactions (TX).**



## Study Code

Commit: [def8c1<sup>7</sup>](https://github.com/web3coach/the-blockchain-bar/commit/def8c169292e258525e2f48a075cb26174e52809)

---

<sup>7</sup><https://github.com/web3coach/the-blockchain-bar/commit/def8c169292e258525e2f48a075cb26174e52809>

## 03 | Monolithic Event vs Transaction

```
>_ git checkout c3_state_blockchain_component
```

Developers used to event-sourcing architecture must have immediately recognized the familiar principles behind transactions. They are correct. Blockchain transactions represent a series of events, and the database is a final aggregated, calculated state after replaying all the transactions in a specific sequence.



## Andrej Programming

*Tuesday evening, March 26.*

It's a relaxing Tuesday evening for Andrej. Celebrating his first client, he decides to play some [Starcraft](#)<sup>8</sup> and clean up his local development machine by removing some old pictures. Unfortunately, he prematurely pressed enter when typing a removal command path in terminal `sudo rm -rf /`. Oops.

All his files, including the bar's `genesis.json` and `state.json` are gone.

Andrej, being a senior developer, repeatedly shouted some f\* words very loudly for a few seconds, but he didn't panic! While he didn't have a backup, he had something better — a piece of paper with all the database transactions. The only thing he'd need to do is replay all the transactions one by one, and his database state would get recovered.

Impressed by the advantages of event-based architecture, he decides to extend his MVP database solution. Every bar's activity, such as individual drink purchases, **MUST** be recorded inside the blockchain database.

---

<sup>8</sup><https://www.youtube.com/watch?v=Ff4VlghrTMg&feature=youtu.be&t=516>

Each **customer** will be represented in DB using an **Account** Struct:

```
type Account string
```

Each **Transaction** (TX - a database change) will have the following four attributes: **from**, **to**, **value** and **data**.

The **data** attribute with one possible value (**reward**) captures Andrej's bonus for inventing the blockchain and increases the initial TBB tokens total supply artificially (inflation).

```
type Tx struct {  
    From Account `json:"from"`  
    To    Account `json:"to"`  
    Value uint    `json:"value"`  
    Data  string   `json:"data"`  
}  
  
func (t Tx) IsReward() bool {  
    return t.Data == "reward"  
}
```

The **Genesis DB** will stay as JSON file:

```
{
  "genesis_time": "2019-03-18T00:00:00.000000000Z",
  "chain_id": "the-blockchain-bar-ledger",
  "balances": {
    "andrej": 1000000
  }
}
```

All the transactions, previously written on a piece of paper, will be stored in a local text-file database called **tx.db**, serialized in JSON format and separated by line-break character:

```
{"from": "andrej", "to": "andrej", "value": 3, "data": ""}
{"from": "andrej", "to": "andrej", "value": 700, "data": "reward"}
{"from": "andrej", "to": "babayaga", "value": 2000, "data": ""}
{"from": "andrej", "to": "andrej", "value": 100, "data": "reward"}
{"from": "babayaga", "to": "andrej", "value": 1, "data": ""}
```

The most crucial database component encapsulating all the business logic will be **State**:

```
type State struct {
    Balances    map[Account]uint
    txMempool   []Tx

    dbFile *os.File
}
```

The State struct will know about all user balances and who transferred TBB tokens to whom, and how many were transferred.

It's constructed by reading the initial user balances from genesis.json file:

```
func NewStateFromDisk() (*State, error) {
    // get current working directory
    cwd, err := os.Getwd()
    if err != nil {
        return nil, err
    }

    genFilePath := filepath.Join(cwd, "database", "genesis.json")
    gen, err := loadGenesis(genFilePath)
    if err != nil {
        return nil, err
    }

    balances := make(map[Account]uint)
    for account, balance := range gen.Balances {
        balances[account] = balance
    }
}
```

Afterwards, the genesis State balances are updated by sequentially replaying all the database events from tx.db:

```
txDbFilePath := filepath.Join(cwd, "database", "tx.db")
f, err := os.OpenFile(txDbFilePath, os.O_APPEND|os.O_RDWR, 0600)
if err != nil {
    return nil, err
}

scanner := bufio.NewScanner(f)
state := &State{balances, make([]Tx, 0), f}

// Iterate over each the tx.db file's line
for scanner.Scan() {
    if err := scanner.Err(); err != nil {
        return nil, err
    }

    // Convert JSON encoded TX into an object (struct)
    var tx Tx
    json.Unmarshal(scanner.Bytes(), &tx)

    // Rebuild the state (user balances),
    // as a series of events
    if err := state.apply(tx); err != nil {
        return nil, err
    }
}

return state, nil
}
```

The State component is responsible for:

- **Adding** new transactions to **Mempool**
- **Validating** transactions against the current State (sufficient sender balance)
- **Changing** the state
- **Persisting** transactions to disk
- **Calculating** accounts balances by replaying all transactions since Genesis in a sequence

**Adding** new transactions to Mempool:

```
func (s *State) Add(tx Tx) error {  
    if err := s.apply(tx); err != nil {  
        return err  
    }  
  
    s.txMempool = append(s.txMempool, tx)  
  
    return nil  
}
```

**Persisting** the transactions to disk:

```
func (s *State) Persist() error {  
    // Make a copy of mempool because the s.txMempool will be modified  
    // in the loop below  
    mempool := make([]Tx, len(s.txMempool))  
    copy(mempool, s.txMempool)  
  
    for i := 0; i < len(mempool); i++ {  
        txJson, err := json.Marshal(s.txMempool[i])  
        if err != nil {  
            return err  
        }  
  
        if _, err = s.dbFile.Write(append(txJson, '\n')); err != nil {  
            return err  
        }  
  
        // Remove the TX written to a file from the mempool  
        // Yes... this particular Go syntax is a bit weird  
        s.txMempool = append(s.txMempool[:i], s.txMempool[i+1:]...)  
    }  
  
    return nil  
}
```

**Changing, Validating** the state:

```
func (s *State) apply(tx Tx) error {
    if tx.IsReward() {
        s.Balances[tx.To] += tx.Value
        return nil
    }

    if tx.Value > s.Balances[tx.From] {
        return fmt.Errorf("insufficient balance")
    }

    s.Balances[tx.From] -= tx.Value
    s.Balances[tx.To] += tx.Value

    return nil
}
```



## Building a Command-Line-Interface (CLI)

*Tuesday evening, March 26.*

Andrej wants to have a convenient way to add new transactions to his DB and list the latest balances of his customers. Because Go programs compile to binary, he builds a CLI for his program.

The easiest way to develop CLI based programs in Go is by using the third party `github.com/spf13/cobra` library.

Andrej initializes Go's built-in dependency manager for his project, called `go modules`:

```
>_ $
```

```
cd $GOPATH/src/github.com/web3coach/the-blockchain-bar
```

```
go mod init github.com/web3coach/the-blockchain-bar
```

The `Go modules` command will automatically fetch any library you reference within your Go files.

Andrej creates a new directory called: `cmd` with a subdirectory `tbb`:

```
>_ mkdir -p ./cmd/tbb
```

Inside he creates a `main.go` file, serving as the program's CLI entry point:

```
package main

import (
    "github.com/spf13/cobra"
    "os"
    "fmt"
)

func main() {
    var tbbCmd = &cobra.Command{
        Use:     "tbb",
        Short:   "The Blockchain Bar CLI",
        Run:     func(cmd *cobra.Command, args []string) {
        },
    }

    err := tbbCmd.Execute()
    if err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }
}
```

The Go programs are compiled using the `install` cmd:

```
>_ go install ./cmd/tbb/...

go: finding github.com/spf13/cobra v1.0.0
go: downloading github.com/spf13/cobra v1.0.0
go: extracting github.com/spf13/cobra v1.0.0
```

Go will detect missing libraries and automatically fetch them before compiling the program. Depending on your `$GOPATH` the resulting program will be saved in the `$GOPATH/bin` folder.

```
>_ echo $GOPATH
> /home/web3coach/go

which tbb
> /home/web3coach/go/bin/tbb
```

You can run `tbb` from your terminal now, but it will not do anything because the `Run` function inside the `main.go` file is empty.

The first thing Andrej needs is versioning support for his `tbb` CLI program.

Next to the `main.go` file, he creates a `version.go` command:

```
package main

import (
    "fmt"
    "github.com/spf13/cobra"
)

const Major = "0"
const Minor = "1"
const Fix = "0"
const Verbal = "TX Add && Balances List"

var versionCmd = &cobra.Command{
    Use: "version",
    Short: "Describes version.",
    Run: func(cmd *cobra.Command, args []string) {
        fmt.Printf("Version: %s.%s.%s-beta %s", Major, Minor, Fix, Ver\
bal)
    },
}
```

Compiles and runs it:

```
>_ go install ./cmd/tbb/...
    tbb version
    > Version: 0.1.0-beta TX Add && Balances List
```

Perfect.

Identically to the `version.go` file, he creates a `balances.go` file:

```
func balancesCmd() *cobra.Command {
    var balancesCmd = &cobra.Command{
        Use:     "balances",
        Short:   "Interact with balances (list...).",
        PreRunE: func(cmd *cobra.Command, args []string) error {
            return incorrectUsageErr()
        },
        Run: func(cmd *cobra.Command, args []string) {
        },
    }

    balancesCmd.AddCommand(balancesListCmd)

    return balancesCmd
}
```

The `balances` command will be responsible for loading the latest DB State and printing it to the standard output:

```
var balancesListCmd = &cobra.Command{
    Use:     "list",
    Short:   "Lists all balances.",
    Run: func(cmd *cobra.Command, args []string) {
        state, err := database.NewStateFromDisk()
        if err != nil {
            fmt.Fprintln(os.Stderr, err)
            os.Exit(1)
        }
        defer state.Close()

        fmt.Println("Accounts balances:")
        fmt.Println("_____")
    },
}
```

```
    fmt.Println("")
    for account, balance := range state.Balances {
        fmt.Println(fmt.Sprintf("%s: %d", account, balance))
    }
},
}
```

Andrej verifies if the cmd works as expected. It should print the exact balances defined in the Genesis file because the `tx.db` file is still empty.

```
>_ go install ./cmd/tbb/...

tbb balances list
```

Accounts balances:

---

andrej: 1000000

Works well! Now he only needs a cmd for recording the bar's activity.

Andrej creates `./cmd/tbb/tx.go` cmd:

```
func txCmd() *cobra.Command {
    var txsCmd = &cobra.Command{
        Use:     "tx",
        Short:   "Interact with txs (add...).",
        PreRunE: func(cmd *cobra.Command, args []string) error {
            return incorrectUsageErr()
        },
        Run: func(cmd *cobra.Command, args []string) {
        },
    }

    txsCmd.AddCommand(txAddCmd())

    return txsCmd
}
```

The `tbb tx add cmd` uses `State.Add(tx)` function for persisting the bar's events into the file system:

```
func txAddCmd() *cobra.Command {
    var cmd = &cobra.Command{
        Use: "add",
        Short: "Adds new TX to database.",
        Run: func(cmd *cobra.Command, args []string) {
            from, _ := cmd.Flags().GetString(flagFrom)
            to, _ := cmd.Flags().GetString(flagTo)
            value, _ := cmd.Flags().GetUint(flagValue)

            fromAcc := database.NewAccount(from)
            toAcc := database.NewAccount(to)

            tx := database.NewTx(fromAcc, toAcc, value, "")

            state, err := database.NewStateFromDisk()
            if err != nil {
                fmt.Fprintln(os.Stderr, err)
                os.Exit(1)
            }

            // defer means, at the end of this function execution,
            // execute the following statement (close DB file with all\
TXs)

            defer state.Close()

            // Add the TX to an in-memory array (pool)
            err = state.Add(tx)
            if err != nil {
                fmt.Fprintln(os.Stderr, err)
                os.Exit(1)
            }
        }
    }
}
```



```
        // Flush the mempool TXs to disk
        err = state.Persist()
        if err != nil {
            fmt.Fprintln(os.Stderr, err)
            os.Exit(1)
        }

        fmt.Println("TX successfully added to the ledger.")
    },
}
```

The tbb tx add cmd has 3 mandatory flags: --from, --to and --value.

```
cmd.Flags().String(flagFrom, "", "From what account to send tokens")
cmd.MarkFlagRequired(flagFrom)
```

```
cmd.Flags().String(flagTo, "", "To what account to send tokens")
cmd.MarkFlagRequired(flagTo)
```

```
cmd.Flags().Uint(flagValue, 0, "How many tokens to send")
cmd.MarkFlagRequired(flagValue)
```

```
return cmd
```

The CLI is done!

Andrej migrates all transactions from paper to his new DB:

```
>_ tbb tx add --from=andrej --to=andrej --value=3  
tbb tx add --from=andrej --to=andrej --value=700  
tbb tx add --from=babayaga --to=andrej --value=2000  
tbb tx add --from=andrej --to=andrej --value=100  
--data=reward  
tbb tx add --from=babayaga --to=andrej --value=1
```

Read all TXs from disk and calculate the latest state:

```
>_ tbb balances list
```

Accounts balances:

---

andrej: 998801  
babayaga: 1999

Bar data successfully restored! Phew, what a night!

## About Cobra CLI library

The good thing about the Cobra lib for CLI programming is the additional features it comes with. For example, you can now run: `tbb help cmd` and it will print out all TBB registered sub-commands with instructions on how to use them.

```
tbb help
```

The Blockchain Bar CLI

Usage:

```
tbb [flags]
tbb [command]
```

Available Commands:

```
balances    Interact with balances (list...).
help        Help about any command
tx          Interact with txs (add...).
version     Describes version.
```

Flags:

```
-h, --help  help for tbb
```

Use "`tbb [command] --help`" for more information about a command.



## Fun Facts

Accidentally losing customers' data is a standard Saturday in the corporate world these days. Blockchain fixes this by decentralizing the data storage.

The trick Andrej baked into the program by skipping balance verification for TXs marked as rewards. **Bitcoin and Ethereum work in the same way.** The balance of the Account who **mined a block** increases out of the blue as a subject of total tokens supply inflation affecting the whole chain. The total supply of bitcoins is capped at 21M BTC. You will learn more about “mining” and “blocks” in chapters 7 and 10.

The components **State** and **Mempool** are not unique to this program. Andrej chose the names and designs to match a simplified [go-Ethereum](https://github.com/ethereum/go-ethereum/blob/7b32d2a47017570c44cd7f8a83612a29656c9857/core/tx_pool.go#L211)<sup>9</sup>, model so you have a glance inside the core Ethereum source code.

---

<sup>9</sup>[https://github.com/ethereum/go-ethereum/blob/7b32d2a47017570c44cd7f8a83612a29656c9857/core/tx\\_pool.go#L211](https://github.com/ethereum/go-ethereum/blob/7b32d2a47017570c44cd7f8a83612a29656c9857/core/tx_pool.go#L211)



## Summary

Blockchain is a database. The token supply, initial user balances, and global blockchain settings are defined in a Genesis file. The Genesis balances indicate what the original blockchain state was and are never updated afterwards.

The database state changes are called Transactions (TX). **Transactions are old fashion Events representing actions within the system.**



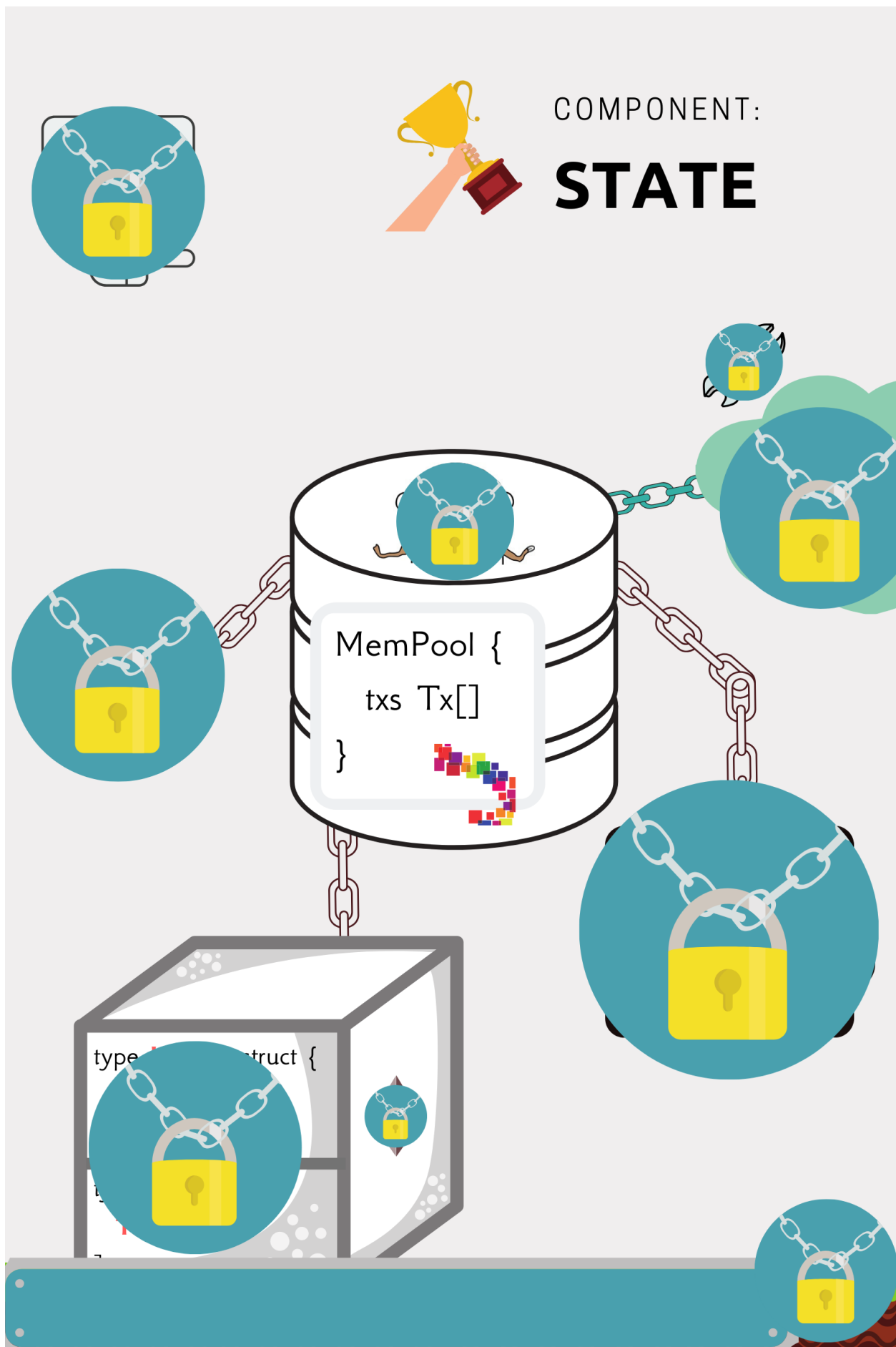
## Study Code

Commit: [5d4b0b](#)<sup>10</sup>

Let's talk about greed.

---

<sup>10</sup><https://github.com/web3coach/the-blockchain-bar/commit/5d4b0b6a001e616109da732fdaf7094f1e1acf85>



## 04 | Humans Are Greedy

```
>_ git checkout c4_caesar_transfer
```

### Typical business greediness

*Wednesday, March 27.*

BabaYaga invested a bit too much. She forgot her flat rent payment was around the corner, and she doesn't have the money. BabaYaga calls her flat owner, **Caesar**.

**BabaYaga:** Hey Caesar, I am sorry, but I don't have the cash to pay you the rent this month...

**Caesar:** Why not?

**BabaYaga:** The Blockchain Bar ICO offered a massive bonus, and I purchased 2000€ worth of tokens for just 1000€. It was a great deal!

**Caesar:** What the hell are you talking about? What is an ICO? What on earth are tokens? Can you pay me in some other way?

**BabaYaga:** Oh, not again. I can give you 1000 TBB tokens worth 1000€, and you can use them in the bar to pay for your drinks! Let me call the bar owner, Andrej, and make the transfer!

**Caesar:** All right... I will take it.

Andrej performs the transfer, **but decides to charge an extra 50 TBB tokens for his troubles.** He doesn't want to, BUT the bar shareholders who invested in him a few years ago are forcing him to generate profit as soon as possible.

BabaYaga won't notice this relatively small fee most likely anyway, Andrej tells himself. In the end, only he has the DB access.

```
>_ // rent payment
    tbb tx add --from=babayaga --to=caesar --value=1000

    // hidden fee charge
    tbb tx add --from=babayaga --to=andrej --value=50

    // new reward for another day of maintaining the DB
    tbb tx add --from=andrej --to=andrej --value=100
    --data=reward
```





## Fun Facts 1/2

The number one blockchain use-case is banking. Many blockchain projects aim to optimize the domestic and international exchange of money across different currency corridors (XRP).

Other projects focus on freedom and self-sovereign identity (SSI) - a digital movement that recognizes an individual should own and control their identity and money without the intervening administrative authorities or other centralized intermediaries. SSI allows people to interact in the digital world with the same freedom and capacity for trust as they do in the offline world. (Bitcoin / Ethereum)

Here are few fun facts why blockchain is a perfect fit for replacing your bank's current banking infrastructure.

—

The good thing about virtual tokens is their fungibility - i.e., their ability to be traded, with each unit being as usable as the next. Performing a transfer from account to account can be done by simply changing the database state. Cryptocurrencies are tradeable 24/7.

You can't trade stocks directly. You need to go through a broker who takes part a percentage of the total transaction as a fee (1-3% to 7% average yearly profit).

An international bank transfer takes between 3-10 business days and can cost as much 5% of the transferred value! If you're sending \$10,000, you may have to pay up to \$500.<sup>11</sup>  
The technology behind for the last 40 years? FTP + CSV files.

---

<sup>11</sup><https://www.ofx.com/en-au/faqs/how-much-does-it-cost-to-send-money-internationally/>



## Fun Facts 2/2

Do you think the stock market is fair? Banks, indexes, and stocks are highly centralized and controlled by governments and private Wall Street groups. Free market? Wall Street controls how much can prices jump/fall in a single day.

As an example, Wall Street halted the trading of “S&P 500 Index” after a 7% drop to protect their investors and hedge funds from losing money from people selling their stocks during March 2020 after COVID news. Afterward, the FED printed trillions of dollars for themselves to support the stock price. If you are a developer who likes to save money and avoid debt, your savings just lost value overnight by a yet unknown percentage.

Many countries are going into negative yields, an unexplored territory with unknown consequences. What does this mean? Soon you will have to pay the bank to keep your savings. Inflation at its best. You are being forced to spend your money to support a system you don't control.



## Summary

**Closed software with centralized access to private data and rules puts only a few people to the position of power. Users don't have a choice, and shareholders are in business to make money.**

Blockchain is a database. The token supply, initial user balances, and global blockchain settings you define in a Genesis file. The Genesis balances indicate what was the original blockchain state and are never updated afterwards.

The database state changes are called Transactions (TX). Transactions are old fashion Events representing actions within the system.



## Study Code

Commit: [00d6ed](https://github.com/web3coach/the-blockchain-bar/commit/00d6ede25b1e54ceb30c0a0314ef99a612db01de)<sup>12</sup>

---

<sup>12</sup><https://github.com/web3coach/the-blockchain-bar/commit/00d6ede25b1e54ceb30c0a0314ef99a612db01de>

# 05 | Why We Need Blockchain

```
>_ git checkout c5_broken_trust
```

## BabaYaga Seeks Justice

*Thursday, March 28.*

BabaYaga enters the bar for her birthday.

**BabaYaga:** Hey, Andrej! Today is my birthday! Get me your most expensive bottle!

**Andrej:** Happy birthday! Here you go: Crystal Head Vodka. But you need to purchase one additional TBB token. The bottle costs 950 tokens, and your balance is 949.

**BabaYaga:** What?! My balance is supposed to be 999 TBB!

**Andrej:** The funds transfer to Caesar you requested last week cost you 50 tokens.

**BabaYaga:** This is unacceptable! I would never agree to such a high fee! You can't do this, Andrej! I trusted your system, but you are as unreliable as every other business owner. Things must change!

**Andrej:** All right, look. You are my most loyal customer, and I didn't want to charge you, but my shareholders forced me. **Let me re-program my system and make it completely transparent and decentralized.** After all, if everyone were able to interact with the bar without going through me, it would significantly improve the bar's efficiency and balance the level of trust!

- Ordering drinks would take seconds instead of minutes
- The customers who forgot their wallets at home could borrow or lend tokens to each other
- I wouldn't have to worry about losing the clients data (again) as everyone would have a copy of it
- **The database would be immutable, so once everyone would agree on a specific state, no one else can change it or maliciously modify the history.** Immutability would help with yearly tax audits as well!
- If shareholders wanted to introduce new fees or raise the current ones, everyone involved in the blockchain system would notice and have to agree with it. The users and business owners would even have to engage in some decentralized governance system together, based on voting, probably. In case of a disagreement, the users walk away with all their data!

**BabaYaga:** Well, it certainly sounds good, but is this even possible?

**Andrej:** Yes, I think so. With a bit of **hashing, linked lists, immutable data structure, distributed replication, and asymmetric cryptography!**

**BabaYaga:** I have no idea what you have just said but go and do your geeky thing, Andrej!

Another day of the system running, another 100 TBB tokens for Andrej for maintaining and improving the blockchain:

```
>_ tbb tx add --from=andrej --to=andrej --value=100  
   --data=reward
```



## Fun Facts

Bitcoin and Ethereum miners also receive rewards every ~15 minutes for running the blockchain servers (nodes) and validating transactions.

Every 15 minutes, one Bitcoin miner receives 12.5 BTC (100k \$ at the moment of writing this page) to cover his servers cost + make some profit.

The Bitcoin network consumes as much electricity as the entire country of Austria. It accounts for 0.29% of the world's annual electricity consumption.

Annually it consumes 76.84 TWh, producing 36.50 Mt CO2 carbon footprint (New Zealand). [Source](#).<sup>13</sup>

Why? You will learn more in Chapter 11, where you will program a Bitcoin mining algorithm from scratch!

PS: Our algorithm will consume a bit less electricity :)

---

<sup>13</sup><https://digiconomist.net/bitcoin-energy-consumption>



## Summary

Closed software with centralized access to private data allows for just a handful of people to have a lot of power. Users don't have a choice, and shareholders are in business to make money.

**Blockchain developers aim to develop protocols where applications' entrepreneurs and users synergize in a transparent, auditable relationship. Specifications of the blockchain system should be well-defined from the beginning and only change if its users support it.**

Blockchain is a database. The token supply, initial user balances, and global blockchain settings are defined in a Genesis file. The Genesis balances indicate what was the original blockchain state and are never updated afterwards.

The database state changes are called Transactions (TX). Transactions are old fashion Events representing actions within the system.



## Study Code

Commit: [642045](https://github.com/web3coach/the-blockchain-bar/commit/64204512f2173eb3f3e136e7e2674a2c456d351f)<sup>14</sup>

---

<sup>14</sup><https://github.com/web3coach/the-blockchain-bar/commit/64204512f2173eb3f3e136e7e2674a2c456d351f>



## 06 | L'Hash de Immutable

```
>_ git checkout c6_immutable_hash
```

The book's technical difficulty starts with this chapter! The concepts will only get more challenging but at the same time, very exciting. Buckle up :)

### How to Program an Immutable Database?

*Friday, March 29.*

If Andrej wants to figure out how to program an immutable DB, he has to realize why other database systems are mutable by design.

He decides to analyze an all-mighty MySQL DB Table:

<b>id</b>	<b>name</b>	<b>balance</b>
1	Andrej	998951
2	BabaYaga	949
3	Caesar	1000

In MySQL DB, anyone with access and a good enough reason can perform a table update such as:

```
UPDATE user_balance SET balance = balance + 100 WHERE id > 1
```

Updating values across different rows is possible because the table rows are independent, mutable, and the latest state is not apparent. What's the latest DB change? Last column changed? Last row inserted? If so, how can Andrej know what row was deleted recently? If the rows and table state were tightly coupled, dependent, a.k.a, updating row 1 would generate a completely new, different table, Andrej would achieve his immutability.

How to tell if any byte in a database has changed?

# Immutability via Hash Functions

*Saturday, March 30.*

Hashing is process of taking a string input of arbitrary length and producing a hash string of fixed length. Any change in input, will result in a new, different hash.

```
package main

import (
    "crypto/sha256"
    "fmt"
)

func main() {
    balancesHash := sha256.Sum256([]byte("| 1 | Andrej | 99895 |"))
    fmt.Printf("%x\n", balancesHash)
    // Output: 6a04bd8e2...f70a3902374f21e089ae7cc3b200751

    // Change balance from 99895 -> 99896

    balancesHashDiff := sha256.Sum256([]byte("| 1 | Andrej | 99896 |"))
    fmt.Printf("%x\n", balancesHashDiff)
    // Output: d04279207...ec6d280f6c7b3e2285758030292d5e1
}
```

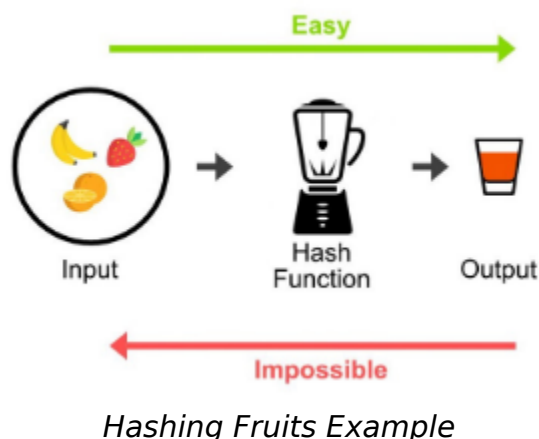
Try it: <https://play.golang.org/p/FTPUa7lhOCE><sup>15</sup>

---

<sup>15</sup><https://play.golang.org/p/FTPUa7lhOCE>

Andrej also requires some level of security for his database, so he decides for a **Cryptographic Hash Function** with the following properties:

- it is **deterministic**;<sup>16</sup> - the same message always results in the same hash
- it is quick to compute the hash value for any given message
- it is infeasible to generate a message from its hash value except by trying all possible messages
- a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
- it is **infeasible**<sup>17</sup> to find two different messages with the same hash value



img src<sup>18</sup>

<sup>16</sup>[https://en.wikipedia.org/wiki/Deterministic\\_algorithm](https://en.wikipedia.org/wiki/Deterministic_algorithm)

<sup>17</sup>[https://en.wikipedia.org/wiki/Computational\\_complexity\\_theory#Intractability](https://en.wikipedia.org/wiki/Computational_complexity_theory#Intractability)

<sup>18</sup><https://twitter.com/cybergibbons/status/1203291585473110016>

## Implementing the DB Content Hashing

*Saturday Evening, March 30.*

Andrej modifies the `Persist()` function to return a new content hash, `Snapshot`, every time a new transaction is persisted.

```
type Snapshot [32]byte
```

The `Snapshot` is produced by this new `sha256` secure hashing function:

```
func (s *State) doSnapshot() error {
    // Re-read the whole file from the first byte
    _, err := s.dbFile.Seek(0, 0)
    if err != nil {
        return err
    }

    txsData, err := ioutil.ReadAll(s.dbFile)
    if err != nil {
        return err
    }
    s.snapshot = sha256.Sum256(txsData)

    return nil
}
```

The `doSnapshot()` is called by the modified `Persist()` function. When a new transaction is written into the `tx.db` file, the `Persist()` hashes the entire file content and returns its 32 bytes “fingerprint”, hash.

```
func (s *State) Persist() (Snapshot, error) {
    mempool := make([]Tx, len(s.txMempool))
    copy(mempool, s.txMempool)

    for i := 0; i < len(mempool); i++ {
        txJson, err := json.Marshal(s.txMempool[i])
        if err != nil {
            return Snapshot{}, err
        }

        fmt.Printf("Persisting new TX to disk:\n")
        fmt.Printf("\t%s\n", txJson)
        if _, err = s.dbFile.Write(append(txJson, '\n')); err != nil {
            return Snapshot{}, err
        }

        err = s.doSnapshot()
        if err != nil {
            return Snapshot{}, err
        }
        fmt.Printf("New DB Snapshot: %x\n", s.snapshot)

        s.txMempool = append(s.txMempool[:i], s.txMempool[i+1:]...)
    }

    return s.snapshot, nil
}
```


---

*Persist() hashes the entire tx.db file*

From this moment, everyone can 100% confidently and securely refer to any particular database state (set of data) with a specific, snapshot hash.

## Practice time.

**1/4** Run the `tbb balances list` cmd and check the balances are matching.

 `tbb balances list`

Account balances at 7d4a360f465d...

id	name	balance
1	Andrej	999251
2	BabaYaga	949
3	Caesar	1000

**2/4** Remove the last 2 rows from `./database/tx.db` and check the balances again.

```
>_ tbb balances list
```

Account balances at 841770dcd3...

id	name	balance
1	Andrej	999051
2	BabaYaga	949
3	Caesar	1000



**3/4** Reward Andrej for last 2 days (from 28th to 30th of March):

## Reward Transaction 1:

```
>_ tbb tx add --from=andrej --to=andrej --value=100  
    --data=reward
```

Persisting new TX to disk:

```
{ "from": "andrej", "to": "andrej", "value": 100, "data": "reward" }
```

New DB Snapshot: ff2470c7043f5a34169b5dd38921ba6825b03b3facb83e426

TX successfully persisted to the ledger.

## Reward Transaction 2:

```
>_ tbb tx add --from=andrej --to=andrej --value=100  
    --data=reward
```

Persisting new TX to disk:

```
{ "from": "andrej", "to": "andrej", "value": 100, "data": "reward" }
```

New DB Snapshot: 7d4a360f468b837b662816bcd52c1869f99327d53ab4a9ca

TX successfully persisted to the ledger.

**4/4** Run the `tbb balances list` cmd and ensure the balances and the snapshot hash is the same as at the beginning.

```
>_ tbb balances list
```

Account balances at 7d4a360f465d...

id	name	balance
1	Andrej	999251
2	BabaYaga	949
3	Caesar	1000

## Done!

Because the cryptographic hash function **sha256**, given the same inputs (current `tx.db` and 2x `tbb tx add`), produces the same output, if you follow the exact steps on your own computer, you will generate the exact same database state and hashes!



## Summary

Closed software with centralized access to private data puts only a few people to the position of power. Users don't have a choice, and shareholders are in business to make money.

Blockchain developers aim to develop protocols where applications' entrepreneurs and users synergize in a transparent, auditable relation. Specifications of the blockchain system should be well defined from the beginning and only change if its users support it.

Blockchain is an **immutable** database. The token supply, initial user balances, and global blockchain settings you define in a Genesis file. The Genesis balances indicate what was the original blockchain state and are never updated afterwards.

The database state changes are called Transactions (TX). Transactions are old fashion Events representing actions within the system.

**The database content is hashed by a secure cryptographic hash function. The blockchain participants use the resulted hash to reference a specific database state.**



## Study Code

Commit: [b99e51](https://github.com/web3coach/the-blockchain-bar/commit/b99e5191b19bc076b98a3869289e4788d0a4a77b)<sup>19</sup>

---

<sup>19</sup> <https://github.com/web3coach/the-blockchain-bar/commit/b99e5191b19bc076b98a3869289e4788d0a4a77b>

# 07 | The Blockchain Programming Model

```
>_ git checkout c7_blockchain_programming_model
```

## Improving Performance of an Immutable DB

*Sunday, March 31.*

Andrej is building not only an immutable database but also a distributed one! He will distribute, replicate all data to every client/stakeholder's computer interacting with the bar **to avoid data centralization** and one, easy to attack, source of truth. This represents a performance challenge.

The current TBB program works fine with a small database size while running on one computer but it suffers two performance bottlenecks:

- Distributing transactions to other computers one by one will be inefficient due to **network latency in distributed systems**
- Creating a snapshot hash of the database and validating it requires to **read and hash the full, potentially several gigabytes large DB from scratch for EVERY NEW TX**

Fortunately, Andrej can solve both issues by implementing a Hashed Linked List in a combination with a standard [Batch Strategy](#)<sup>20</sup>.

**Batching** is a common strategy when working with SQL/NoSQL/Other database systems. The batch strategy consist of “**handling multiple items at once**”. The solution is to encapsulate transactions to **linked “chunks”, “blocks”**.

---

<sup>20</sup>[https://en.wikipedia.org/wiki/Batch\\_processing](https://en.wikipedia.org/wiki/Batch_processing)

## Batch + Hash + Linked List $\Rightarrow$ Blocks

*Monday, April 1.*

Andrej is not joking (see date). He designs the following data structure for encapsulating transactions into batches:

```
type Hash [32]byte

type Block struct {
    Header BlockHeader
    TXs     []Tx // new transactions only (payload)
}

type BlockHeader struct {
    Parent Hash // parent block reference
    Time   uint64
}
```

And he renames Snapshot struct to Hash.

**Block** struct will have 2 attributes, **Header** and **Payload**:

- Payload stores **NEW transactions**
- Header stores block's metadata (**PARENT BLOCK HASH REFERENCE**, time)

Legacy, slow hashing of the entire DB after every new TX:

```
func (s *State) doSnapshot() error {
    // Re-read the whole file from the first byte
    _, err := s.dbFile.Seek(0, 0)
    if err != nil {
        return err
    }

    txsData, err := ioutil.ReadAll(s.dbFile)
    if err != nil {
        return err
    }

    // Give me the hash of the entire DB content
    s.snapshot = sha256.Sum256(txsData)

    return nil
}
```

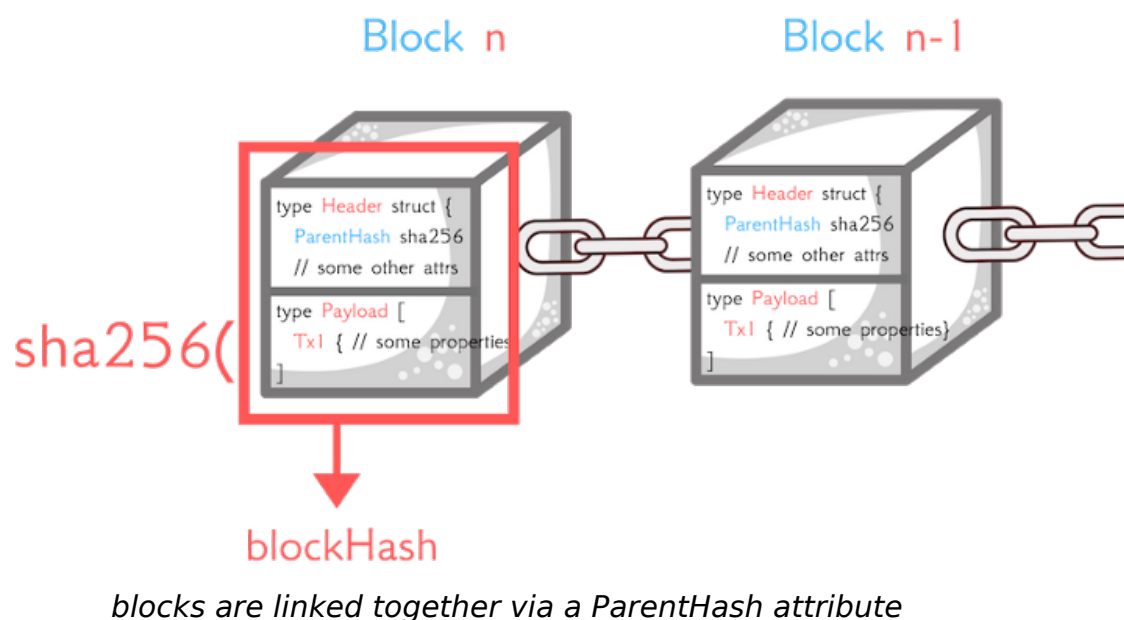
New optimized hashing - where the 32bytes Hash is calculated by only hashing the latest Block encoded as JSON:

```
type Block struct {
    Header BlockHeader // metadata (parent block hash + time)
    TXs     []Tx              // new transactions only (payload)
}

func (b Block) Hash() (Hash, error) {
    blockJson, err := json.Marshal(b)

    return sha256.Sum256(blockJson), nil
}
```

The above implementation can be visually represented as, the **blockchain programming model**:



## Why is the parent block's hash reference necessary?

The ParentHash is being used as a reliable “checkpoint,” representing and referencing the previously hashed database content.

ParentHash improves performance. **Only new data + reference to previous state** needs to be hashed to achieve **immutability**.

E.g., If you attempt to modify a TX value in Block 0, it will result in a new unique Block 0 hash. Hash of Block 1, based on the parent Block 0 reference, would therefore immediately change as well. The cascade effect would affect all the blocks, making the malicious



attacker database invalid - different from the rest of the honest database stakeholders.

The attacker database would be, therefore, excluded from participating in the network. Why? You will find out in Chapter 10 - where you will program a peer-to-peer sync algorithm.

PS: If you are curious in history and in-depth theory of a Linked List data structure, I recommend to checkout this great [Wikipedia explanation](https://en.wikipedia.org/wiki/Linked_list).<sup>21</sup>

---

<sup>21</sup>[https://en.wikipedia.org/wiki/Linked\\_list](https://en.wikipedia.org/wiki/Linked_list)

## How adding a TX into a Block works

*Monday Evening, April 1.*

A new TX struct is constructed from cmd params:

```
func txAddCmd() *cobra.Command {
    var cmd = &cobra.Command{
        Use: "add",
        Short: "Adds new TX to database.",
        Run: func(cmd *cobra.Command, args []string) {
            from, _ := cmd.Flags().GetString(flagFrom)
            to, _ := cmd.Flags().GetString(flagTo)
            value, _ := cmd.Flags().GetUint(flagValue)
            data, _ := cmd.Flags().GetString(flagData)

            tx := database.NewTx(
                database.NewAccount(from),
                database.NewAccount(to),
                value,
                data,
            )
        }
    }
}
```

The TX is then stored in the **State's mempool** (a single TX, or multiple, as necessary):

```
state, err := database.NewStateFromDisk()

state.AddTx(tx)
```

Finally, the `state.Persist()` function to flush the mempool transactions into the `./database/block.db` database file is executed:

```
state.Persist()
```

Internally, the `Persist()` function will create a new `Block`, encode it into a JSON and write it to the DB.

```
func (s *State) Persist() (Hash, error) {
    // Create a new Block with ONLY the new TXs
    block := NewBlock(
        s.latestBlockHash,
        uint64(time.Now().Unix()),
        s.txMempool,
    )

    blockHash, err := block.Hash()
    if err != nil {
        return Hash{}, err
    }

    blockFs := BlockFS{blockHash, block}

    // Encode it into a JSON string
    blockFsJson, err := json.Marshal(blockFs)
    if err != nil {
        return Hash{}, err
    }

    fmt.Printf("Persisting new Block to disk:\n")
    fmt.Printf("\t%s\n", blockFsJson)

    // Write it to the DB file on a new line
    err = s.dbFile.Write(append(blockFsJson, '\n'))
    if err != nil {
        return Hash{}, err
    }
    s.latestBlockHash = blockHash
}
```

```
// Reset the mempool
s.txMempool = []Tx{}

return s.latestBlockHash, nil
}
```

## Migrating from TX.db to BLOCKS.db

*Monday Evening, April 1.*

At the moment, all the transactions are written in the `./database/tx.db` file. To migrate them into blocks, Andrej develops a simple helper command.

He places the cmd into a `./cmd/tbbmigrate/main.go` file:

```
func main() {  
    state, err := database.NewStateFromDisk()  
    if err != nil {  
        fmt.Fprintln(os.Stderr, err)  
        os.Exit(1)  
    }  
    defer state.Close()
```

He encapsulates the first 2 bar's TXs into a Block 0 and persists the block to disk:

```
block0 := database.NewBlock(  
    database.Hash{},  
    uint64(time.Now().Unix()),  
    []database.Tx{  
        database.NewTx("andrej", "andrej", 3, ""),  
        database.NewTx("andrej", "andrej", 700, "reward"),  
    },  
)  
  
state.AddBlock(block0)  
block0hash, _ := state.Persist()  
// Block 0 Hash: 07536e2c559b0a4566b84802...  
// Block 0 Parent Hash: 00000000...
```

Result written to disk:

```
{
  "hash": "07536e2c55ffa629a105f61c8f6c5f1c7289d139e02639436",
  "block": {
    "header": {
      "parent": "0000000000000000000000000000000000000000000000000000000000000000",
      "time": 1556563065
    },
    "payload": [
      {
        "from": "andrej",
        "to": "andrej",
        "value": 3,
        "data": ""
      },
      {
        "from": "andrej",
        "to": "andrej",
        "value": 700,
        "data": "reward"
      }
    ]
  }
}
```

The parent hash is empty because it's the first Block ever created. Andrej inserts the rest of the TX history into a Block 1. Block 1 references Block 0 as its parent by only using the Block 0' hash. It's not necessary to re-hash the entire Block 0 content! Much less CPU cycles wasted, the same level of security and immutability achieved.

```
block1 := database.NewBlock(  
    block0hash,  
    uint64(time.Now().Unix()),  
    []database.Tx{  
        database.NewTx("andrej", "babayaga", 2000, ""),  
        database.NewTx("andrej", "andrej", 100, "reward"),  
        database.NewTx("babayaga", "andrej", 1, ""),  
        database.NewTx("babayaga", "caesar", 1000, ""),  
        database.NewTx("babayaga", "andrej", 50, ""),  
        database.NewTx("andrej", "andrej", 600, "reward"),  
    },  
)  
  
state.AddBlock(block1)  
state.Persist()  
  
// Block 1 Hash: 2efe28463821660834cf8e3cc555...  
// Block 1 Parent Hash: 07536e2c559b0a4566b84802...
```



Disk:

```
{
  "hash": "2efe284638555aeb5aedc911b7b8653add975d0ad3f3ba0c5",
  "block": {
    "header": {
      "parent": "07536e2c55ffa629a105f61c8f6c5f1c7289d139e02639436",
      "time": 1556563065
    },
    "payload": [
      {
        "from": "andrej",
        "to": "babayaga",
        "value": 2000,
        "data": ""
      },
      {
        "from": "andrej",
        "to": "andrej",
        "value": 100,
        "data": "reward"
      },
      ...
    ]
  }
}
```

## Experiment with the migration cmd

 **Practice time.**

**1/3** Remove the existing 2 blocks from `./database/block.db` that Andrej committed in the current branch.

```
>_ cat /dev/null > ./database/block.db
```

**2/3** Compile the code and run the new migration command.

```
>_ go install ./cmd/...  
tbbmigrate
```

**3/3** Observe 2 new JSON encoded blocks being written into the `./database/block.db` as programmed in `./cmd/tbbmigrate/main.go`.

Persisting new Block to disk:

```
{
  "hash": "8a05167d2f...63196c740",
  "block": {
    "header": {
      "parent": "0000000000...000000000",
      "time": 1587486395
    },
    "payload": [
      {
        "from": "andrej",
        "to": "andrej",
        "value": 3,
        "data": ""
      },
      {
        "from": "andrej",
        "to": "andrej",
        "value": 700,
        "data": "reward"
      }
    ]
  }
}
```

Persisting new Block to disk:

```
{
  "hash": "c70775ae5e...b6a6184",
  "block": {
    "header": {
      "parent": "8a05167d2...196c740",
      "time": 1587486395
    }
  }
}
```

```
},
"payload": [
  {
    "from": "andrej",
    "to": "babayaga",
    "value": 2000,
    "data": ""
  },
  {
    "from": "andrej",
    "to": "andrej",
    "value": 100,
    "data": "reward"
  },
  {
    "from": "babayaga",
    "to": "andrej",
    "value": 1,
    "data": ""
  },
  {
    "from": "babayaga",
    "to": "caesar",
    "value": 1000,
    "data": ""
  },
  {
    "from": "babayaga",
    "to": "andrej",
    "value": 50,
    "data": ""
  },
  {
    "from": "andrej",
    "to": "andrej",
    "value": 600,
```

```
        "data": "reward"
    }
]
}
```

PS: Your block hashes will be different because the `block.time` attribute is set on runtime - when you run the `tbbmigrate` cmd.



## Summary

Closed software with centralized access to private data puts only a few people to the position of power. Users don't have a choice, and shareholders are in business to make money.

Blockchain developers aim to develop protocols where applications' entrepreneurs and users synergize in a transparent, auditable relation. Specifications of the blockchain system should be well defined from the beginning and only change if its users support it.

Blockchain is an **immutable** database. The token supply, initial user balances, and global blockchain settings you define in a Genesis file. The Genesis balances indicate what was the original blockchain state and are never updated afterwards.

The database state changes are called Transactions (TX). Transactions are old fashion Events representing actions within the system.

The database content is hashed by a secure cryptographic hash function. The blockchain participants use the resulted hash to reference a specific database state.

**Transactions are grouped into batches for performance reasons. A batch of transactions make a Block. Each block is encoded and hashed using a secure, cryptographic hash function.**

**Block contains Header and Payload. The Header stores various metadata such a time and a reference to the Parent Block (the previous immutable database state). The Payload carries the new database transactions.**

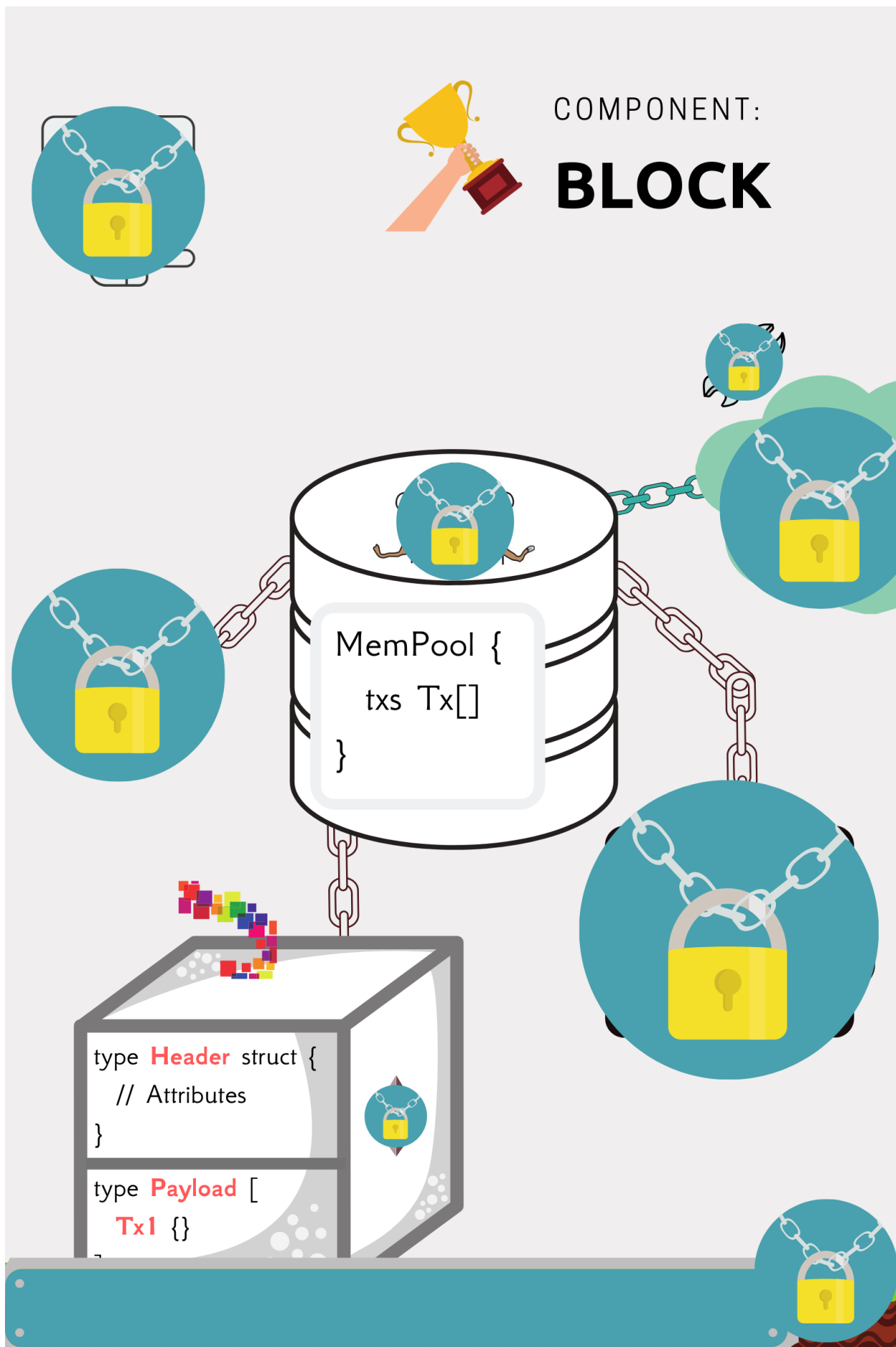


## Study Code

Commit: [239534](#)<sup>22</sup>

---

<sup>22</sup><https://github.com/web3coach/the-blockchain-bar/commit/239534f5eb73480959e65557b8d9d5d14341f38b>





# o8 | Transparent Database

```
>_ git checkout c8_transparent_db
```

## Flexible DB Directory

*Tuesday, April 2.*

The next step for Andrej is to make the DB fully accessible to all customers. To keep things simple, he decides to program a standard HTTP API with few core endpoints for reading customer balances and managing transactions, and to deploy its program to a dedicated server running 24/7. Don't worry, he will secure the API with a SSL certificate later.

But to do so, he must make the application more configurable, especially where the data should be stored.

PS: Andrej will rewrite the HTTP API to a more powerful gRPC version (remote procedure call system) in future chapters. #excitingProject

He introduces a new data directory flag required by all CLI commands.

```
func addDefaultRequiredFlags(cmd *cobra.Command) {
    cmd.Flags().String(
        flagDataDir,
        "",
        "Absolute path where all data will/is stored",
    )
    cmd.MarkFlagRequired(flagDataDir)
}
```

Andrej will need a new file `fs.go` inside the database pkg where he groups together all the file-system related business logic for creating a new `datadir` to store inside the database files.

The `fs.go` has 3 helper functions to determine the DB paths:

```
func getDatabaseDirPath(dataDir string) string {
    return filepath.Join(dataDir, "database")
}

func getGenesisJsonFilePath(dataDir string) string {
    return filepath.Join(getDatabaseDirPath(dataDir), "genesis.json")
}

func getBlocksDbFilePath(dataDir string) string {
    return filepath.Join(getDatabaseDirPath(dataDir), "block.db")
}
```

The first ever `tbb run` program execution should also create a fresh new database with an empty `blocks.db` and a default `genesis.json`.

```
func initDataDirIfNotExists(dataDir string) error {
    if fileExist(getGenesisJsonFilePath(dataDir)) {
        return nil
    }

    dbDir := getDatabaseDirPath(dataDir)
    if err := os.MkdirAll(dbDir, os.ModePerm); err != nil {
        return err
    }

    gen := getGenesisJsonFilePath(dataDir)
    if err := writeGenesisToDisk(gen); err != nil {
        return err
    }

    blocks := getBlocksDbFilePath(dataDir)
    if err := writeEmptyBlocksDbToDisk(blocks); err != nil {
        return err
    }

    return nil
}
```

## Practice time.

**1/3** Initialize a new blockchain dir in a path of your choice. The convention is to use a “hidden” directory in your \$HOME path.

```
>_ tbb balances list --datadir=/home/web3coach/.tbb
```

```
Accounts balances at 0000000000...00000000:
```

---

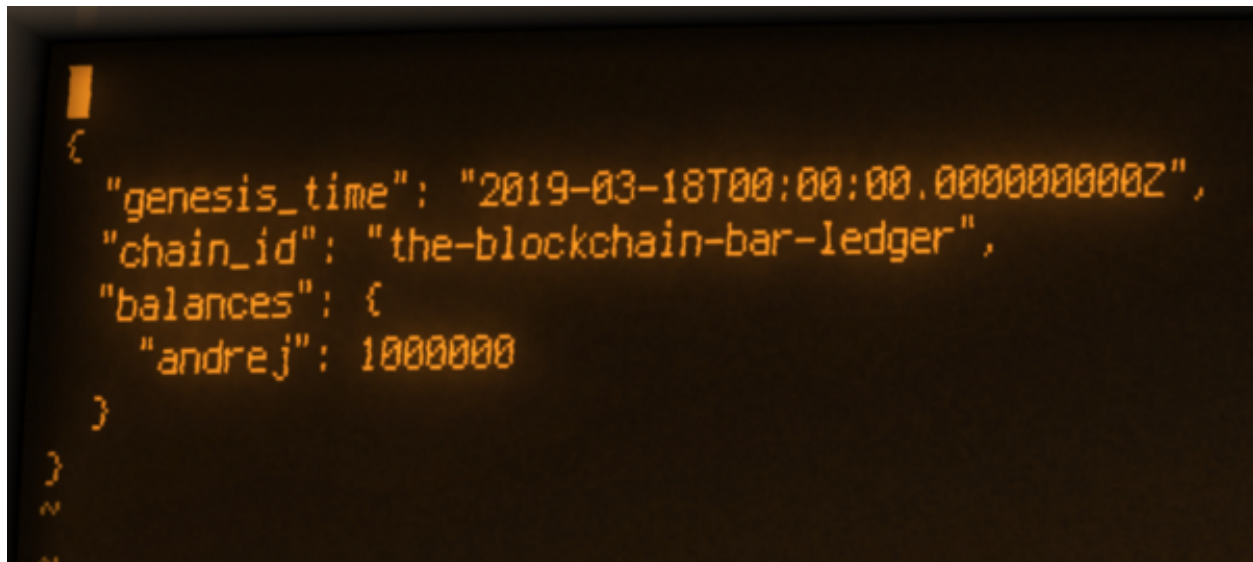
```
andrej: 1000000
```

**2/3** Explore the directory and ensure the blocks.db and genesis.json are present.

```
>_ ls -la /home/web3coach/.tbb/database
```

```
drwxrwxr-x 2 .
drwxrwxr-x 3 ..
-rwxrwxr-x 1 block.db
-rw-r--r-- 1 genesis.json
```

```
>_ vim /home/web3coach/.tbb/database/genesis.json
```



```
{
  "genesis_time": "2019-03-18T00:00:00.000000000Z",
  "chain_id": "the-blockchain-bar-ledger",
  "balances": {
    "andrej": 1000000
  }
}
```

*new datadir genesis*

**3/3** Copy the `blocks.db` from previous chapter into the new dir.

```
>_ export TBBS="$GOPATH/src/github.com/web3coach/the-blockchain-bar"
    export TBB="/home/web3coach/.tbb"

    cp $TBBS/database/block.db $TBB/database/
```

```
> tbb balances list --datadir=/home/web3coach/.tbb
```

Accounts balances at 2efe284638...ad3f3ba0c5:

---

```
babayaga: 949
caesar: 1000
andrej: 999451
```

## Done!

The DB data directory is now configurable, but Andrej is missing another essential part, a public interface to communicate with other blockchain peers.



## Study Code

Commit: [c6be95](#)<sup>23</sup> | Full source code: [v0.5.0](#)<sup>24</sup>

---

<sup>23</sup><https://github.com/web3coach/the-blockchain-bar/commit/c6be95ae7bb3da1bd947e92e41cba4cb23afd5b4>

<sup>24</sup><https://github.com/web3coach/the-blockchain-bar/tree/0.5.0>

## Centralized Public HTTP API

*Wednesday, April 3.*

Andrej needs to convert his CLI program into a HTTP API so he can communicate with other peers and to make his DB public and transparent.

He starts by removing the `tx.go` command because the HTTP Server will be a long running process and will be responsible for maintaining the `State`. He doesn't want to be updating the DB from 2 different input sources and run into collisions and unpredictable behaviour.

Programming an HTTP server in Go is very easy. You don't need any Apache or Nginx.

Andrej programs a new tbb run cmd:

```
package main

func runCmd() *cobra.Command {
    var runCmd = &cobra.Command{
        Use:     "run",
        Short:   "Launches the TBB node and its HTTP API.",
        Run:     func(cmd *cobra.Command, args []string) {
            dataDir, _ := cmd.Flags().GetString(flagDataDir)

            fmt.Println("Launching TBB node and its HTTP API...")

            err := node.Run(dataDir)
            if err != nil {
                fmt.Println(err)
                os.Exit(1)
            }
        },
    }

    addDefaultRequiredFlags(runCmd)

    return runCmd
}
```

He places the HTTP API code inside a new `node/node.go` package’.

Why a node pkg? In the blockchain world, the terminology “client” and “server” are avoided in favor of the “node” concept. Every peer (computer) is a server (backend), a completely valid, independent database.



The `Run()` method loads the `State` from disk and starts listening on a hardcoded (for now) port 8080.

```
const httpPort = 8080

func Run(dataDir string) error {
    state, err := database.NewStateFromDisk(dataDir)
    if err != nil {
        return err
    }
    defer state.Close()

    http.ListenAndServe(fmt.Sprintf(":%d", httpPort), nil)
}
```

This is all you really have to do to launch a basic HTTP server in Go. Impressive right?

Now that he has the server, he attaches 2 new HTTP endpoints to it.

A `http://localhost:8080/balances/list` HTTP GET route for querying account balances:

```
http.HandleFunc("/balances/list", listBalancesHandler)
func listBalancesHandler(w ResponseWriter, r Req, state *state) {
    writeRes(w, BalancesRes{state.LatestBlockHash(), state.Balances})
}
```

Returning a JSON encoded struct with the latest block hash and all bar customers balances:

```

type BalancesRes struct {
    Hash      database.Hash      `json:"block_hash"`
    Balances  map[database.Account]uint `json:"balances"`
}

```

And a `http://localhost:8080/tx/add` HTTP POST route for recording new transactions on the ledger:

```

type TxAddReq struct {
    From  string `json:"from"`
    To    string `json:"to"`
    Value uint   `json:"value"`
    Data  string `json:"data"`
}

http.HandleFunc("/tx/add", txAddHandler)
func txAddHandler(w ResWriter, r Req, state *database.state) {
    req := TxAddReq{}

    // Parse the POST request body
    err := readReq(r, &req)
    if err != nil {
        writeErrRes(w, err)
        return
    }

    tx := database.NewTx(
        database.NewAccount(req.From),
        database.NewAccount(req.To),
        req.Value,
        req.Data,
    )

    // Exactly like from the CLI command.
    // Add a new TX into the Mempool
}

```

```
err = state.AddTx(tx)
if err != nil {
    writeErrRes(w, err)
    return
}

// Flush the Mempool TX to the disk
hash, err := state.Persist()
if err != nil {
    writeErrRes(w, err)
    return
}

writeRes(w, TxAddRes{hash})
}
```

The HTTP support feature added in few minutes using just **one node.go file!!!**

Andrej must love Go 😊

## Practice time.

### 1/3 Boot your new, shinny blockchain node!

```
>_ tbb run --datadir=$HOME/.tbb
```

Launching TBB node and its HTTP API...  
Listening on: 127.0.0.1:8080

### 2/3 Query the customers balances.

```
>_ curl -X GET http://localhost:8080/balances/list
```

```
{  
  "block_hash": "2efe284638...d3f3ba0c5",  
  "balances": {  
    "andrej": 999451,  
    "babayaga": 949,  
    "caesar": 1000  
  }  
}
```

**3/3** Add a new TX via cURL, or a Postman.com with a nice GUI.

**>\_** \$

```
curl --location --request POST 'http://localhost:8080/tx/add' \  
--header 'Content-Type: application/json' \  
--data-raw '{  
    "from": "andrej",  
    "to": "babayaga",  
    "value": 100  
}'  
  
> {"block_hash": "9efa5f844...7e20a187"}
```

**Perfect.** The blockchain still works like a charm.

## Deploying TBB Program to AWS

*Thursday, April 4.*

Andrej can now deploy the program to a dedicated server where a **Supervisor** service will manage the program to keep it running nonstop.

He uploads his local database and the compiled binary to a AWS server:

```
ssh tbb
mkdir -p /home/ec2-user/.tbb/database
exit

scp -i ~/.ssh/tbb_aws.pem $TBB/database/genesis.json ec2-user@ec2-3-12\
7-248-10.eu-central-1.compute.amazonaws.com:/home/ec2-user/.tbb/databa\
se/

scp -i ~/.ssh/tbb_aws.pem $TBB/database/block.db ec2-user@ec2-3-127-24\
8-10.eu-central-1.compute.amazonaws.com:/home/ec2-user/.tbb/database/

scp -i ~/.ssh/tbb_aws.pem $GOPATH/bin/tbb ec2-user@ec2-3-127-248-10.eu\
-central-1.compute.amazonaws.com:/home/ec2-user/

ssh tbb
sudo ln -s /home/ec2-user/tbb /usr/local/bin/tbb

tbb version
> Version: 0.6.1-beta HTTP API
```

Installs and sets up the Supervisor process ([full instructions](#))<sup>25</sup>:

```
sudo amazon-linux-extras install epel
sudo yum install -y supervisor

sudo vim /etc/supervisord.d/tbb.conf
[program:tbb]
command=/usr/local/bin/tbb run --datadir=/home/ec2-user/.tbb
autostart=true
autorestart=true
stderr_logfile=/home/ec2-user/.tbb/tbb.err.log
stdout_logfile=/home/ec2-user/.tbb/tbb.out.log
:wq
// he even figures how to save and exit Vim after several months :)

sudo vim /etc/supervisord.conf
// Configure loading of custom config files
[include]
files = supervisord.d/*.conf

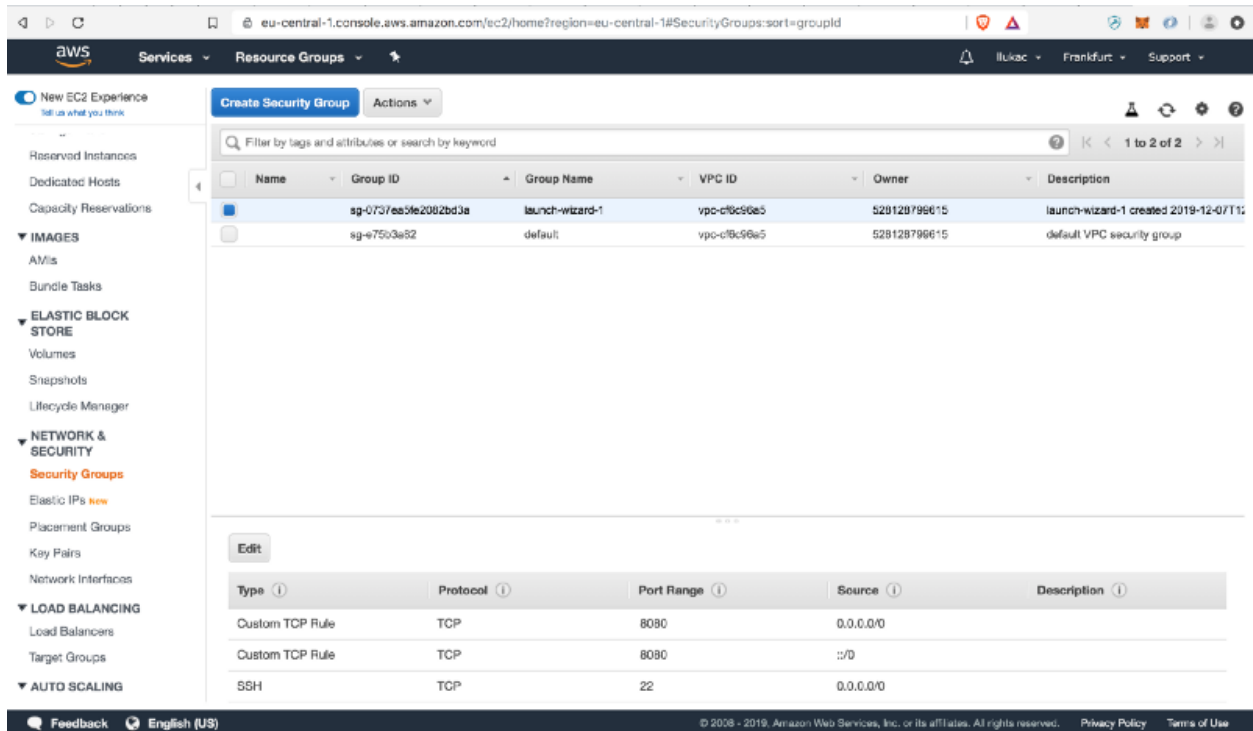
sudo service supervisord start

cat /home/ec2-user/.tbb/tbb.out.log
> Launching TBB node and its HTTP API...
> Listening on HTTP port: 8080
```

---

<sup>25</sup><https://tn710617.github.io/supervisor/>

Opens the HTTP port in AWS network settings:



The screenshot shows the AWS Management Console interface for creating or editing a security group. The left sidebar contains navigation links for various AWS services, including EC2, IAM, and CloudFormation. The main content area displays the 'Create Security Group' page for the 'Launch-wizard-1' security group. The 'Edit' tab is selected, showing a table of rules. The table has columns for Type, Protocol, Port Range, Source, and Description. The rules listed are:

Type	Protocol	Port Range	Source	Description
Custom TCP Rule	TCP	8080	0.0.0.0/0	
Custom TCP Rule	TCP	8080	:::0	
SSH	TCP	22	0.0.0.0/0	

*open AWS HTTP port*

DONE. Andrej's blockchain is now publicly accessible, even though in a very centralized manner, for the time being!



Go ahead, dear reader, query the current blockchain state using cURL.

**>\_** `curl -X GET http://node.tbb.web3.coach:8080/balances/list`

The screenshot shows a REST client interface with a GET request to `http://node.tbb.web3.coach:8080/balances/list`. The response is displayed in JSON format, showing the current blockchain state with a block hash and a list of balances for three users: andrej, babayaga, and caesar.

KEY	VALUE
Key	Value

```
{
  "block_hash": "9319745d5448961fb01dc7f1398761c785e6453bf7e4c47b026c0337d14071b1",
  "balances": {
    "andrej": 1024151,
    "babayaga": 949,
    "caesar": 1000
  }
}
```

*curl TBB balances*

## Burned-out

*Friday, 6th of December 2019.*

Andrej was trying to balance a 9-5 start-up life, teaching blockchain development in his free time and running a blockchain bar for his customers on weekends.

Surprise, surprise, he got burned out.

No such a thing as 10x developer, endless motivation, or 24/7 entrepreneur hustler.

We are animals with complicated emotions and the ability to reason. We are subject to the basic principles of nature and physics.

**Take care of your mental and physical health, my friend!**

Now, Andrej being back and stronger than ever and because blockchain never burns-out, he collects his substantial recovery bonus for the last 247 days.

```
>_ tbb tx add --from=andrej --to=andrej --value=24700  
--data=reward
```



## Summary

Closed software with centralized access to private data puts only a few people to the position of power. Users don't have a choice, and shareholders are in business to make money.

Blockchain developers aim to develop protocols where applications' entrepreneurs and users synergize in a transparent, auditable relation. Specifications of the blockchain system should be well defined from the beginning and only change if its users support it.

Blockchain is an immutable, **transparent** database. The token supply, initial user balances, and global blockchain settings you define in a Genesis file. The Genesis balances indicate what was the original blockchain state and are never updated afterwards.

The database state changes are called Transactions (TX). Transactions are old fashion Events representing actions within the system.

The database content is hashed by a secure cryptographic hash function. The blockchain participants use the resulted hash to reference a specific database state.

Transactions are grouped into batches for performance reasons. A batch of transactions make a Block. Each block is encoded and hashed using a secure, cryptographic hash function.

Block contains Header and Payload. The Header stores various metadata such a time and a reference to the Parent Block (the previous immutable database state). The Payload carries the new database transactions.



## Study Code

Commit: [4f89e4](#)<sup>26</sup> | Full source code: [v0.6.0](#)<sup>27</sup>

---

<sup>26</sup> <https://github.com/web3coach/the-blockchain-bar/commit/4f89e45627b67d2ea3393e2949c8f76939033962>

<sup>27</sup> <https://github.com/web3coach/the-blockchain-bar/releases/tag/0.6.0>

# Unlock All Chapters

You finished the first few chapters! Congratulation!

 20%

**But this was just a quick warm up to give you a taste of how this book is written.** The real fun, heavy programming, a Github repository with all the source code and blockchain knowledge is awaiting you in the full version of the book!



## What's inside the full book version?

- 250+ pages
- Access to a private Discord students room for coding challenges, bug fixing, debugging, questions and support
- 3 years of eBook maintenance, new content, new p2p design patterns
- Private 30 mins blockchain and Go coaching via Zoom
- 1 on 1 direct chat support via Twitter [@Web3Coach](https://twitter.com/Web3Coach) DMs<sup>28</sup> or email
- Cheatsheet, exam

---

<sup>28</sup><https://twitter.com/Web3Coach>

**The full version has 250+ pages:**



## **08 | Transparent Database**

- Flexible DB Directory
- Centralized Public HTTP API
- Deploying TBB Program to AWS
- Burned-out

**Learning:** You program a HTTP server in Go and deploy your program to AWS.



## **09 | It Takes Two Nodes To Tango**

- Distributing the DB to Customers
- Designing a Peer-to-Peer Sync Algorithm

**Learning:** You design a basic peer-to-peer communication algorithm.



## 10 | Programming a Peer-to-Peer DB Sync Algorithm

- Each Block Has a Number
- Tell Me Your State
- Bootstrap Nodes and Peer List
- The Search for a Peer with New Blocks
- Give Me Your Blocks or Life!
- Trust but Verify

**Learning:** You implement a p2p database replication.



## 11 | The Autonomous Database Brain

- The P2P Heaven: The Fastest to Rule Them All
- How does Bitcoin Mining Works?
- Programming Bitcoin Mining Algorithm
- The Slow Elephant in the Bitcoin Room
- How are Bitcoins created?
- Programming Bitcoin Mining Reward in 5 Steps
- Blockchain Releases are Complicated

**Learning:** You make your program fully decentralized and independent.



## 12 | Madam/Sir Your Cryptographic Signature Please

- Hacking a User Balance
- Current State of Web Authentication
- Asymmetric Cryptography in Nutshell
- Blockchain Authentication with go-Ethereum
- Programming a Cryptocurrency Wallet
- MySQL vs Blockchain Authorization
- Madam/Sir, Sign this Database Change
- Programming Signed Transactions
- Digital Signature Replay Attack

**Learning:** You program a decentralized authentication and authorization using asymmetric cryptography.



## 13 | Official TBB Training Ledger

- Create a new Account
- Connect your Node to Students Network
- Request 1000 TBB Testing Tokens

**Learning:** You join an official The Blockchain Bar training network and obtain “production” testing tokens necessary for the next upcoming chapters.





14, 15...

**Bonus:** You will get more bonus chapters in the upcoming months to be always up-to-date with the blockchain ecosystem!



**Buy full version to continue**

<https://gumroad.com/l/build-a-blockchain-from-scratch-in-go><sup>29</sup>

---

<sup>29</sup><https://gumroad.com/l/build-a-blockchain-from-scratch-in-go>