Introducing **piladb** to the Gopher

by Fernando Álvarez

GoMad — Madrid Go UG — 16/02/2017

Summary

- `whoami`

Summary

- `whoami`
- Introduction to **piladb**

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go
- Demo

Summary

- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go
- Demo
- Future plans

Summary

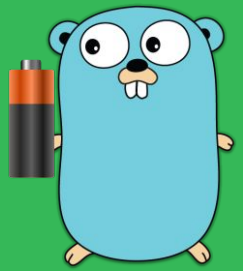
- `whoami`
- Introduction to **piladb**
- Context and use cases
- Functionality and main components
- Internals in Go
- Demo
- Future plans
- Help!

`whoami`

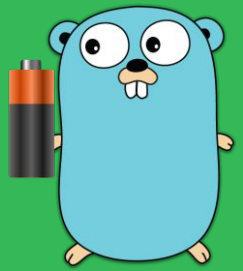
Fernando Álvarez

- Software Engineer from Madrid
- Infrastructure at BeBanjo
- Gopher since 2013
- Open Source through ≅oscillatingworks
- Author of **piladb**
- @fern4lvarez

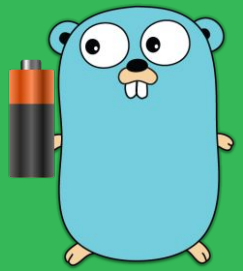
piladb *[pee-lah-dee-bee].*



piladb



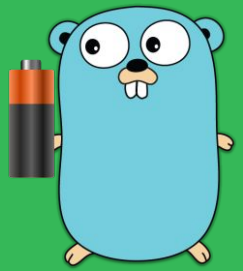
piladb



battery

piladb

charge



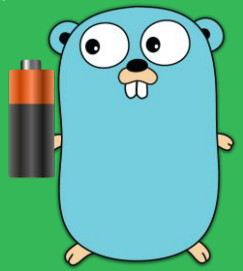
battery

piladb

energy

battery

charge

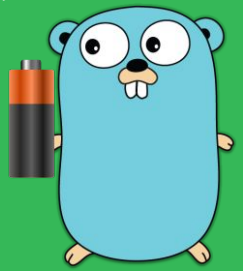


piladb

energy

battery

charge



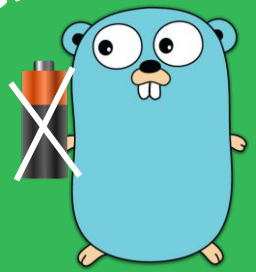
superpowers

piladb

~~energy~~

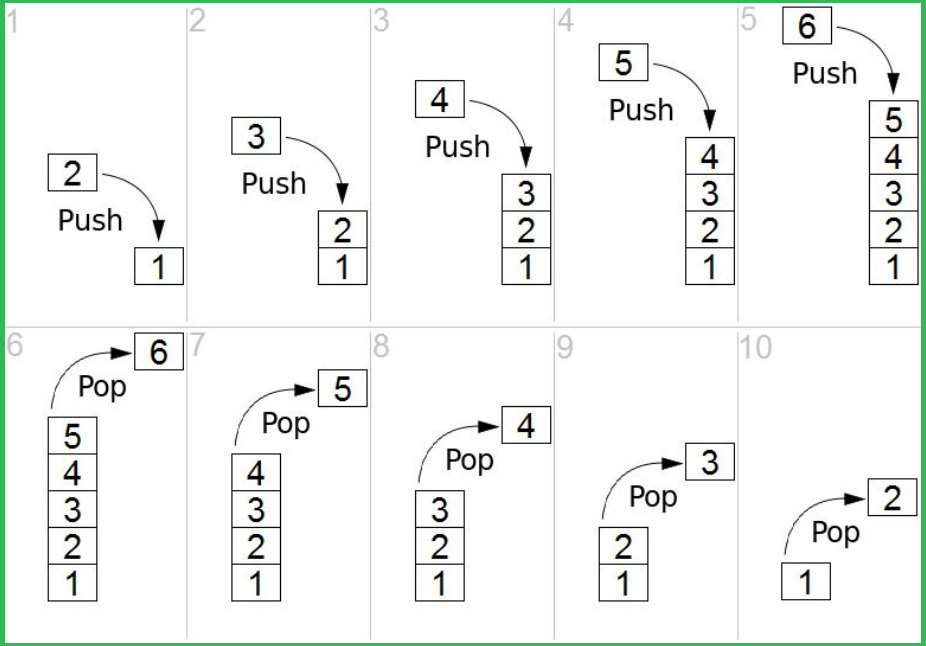
~~battery~~

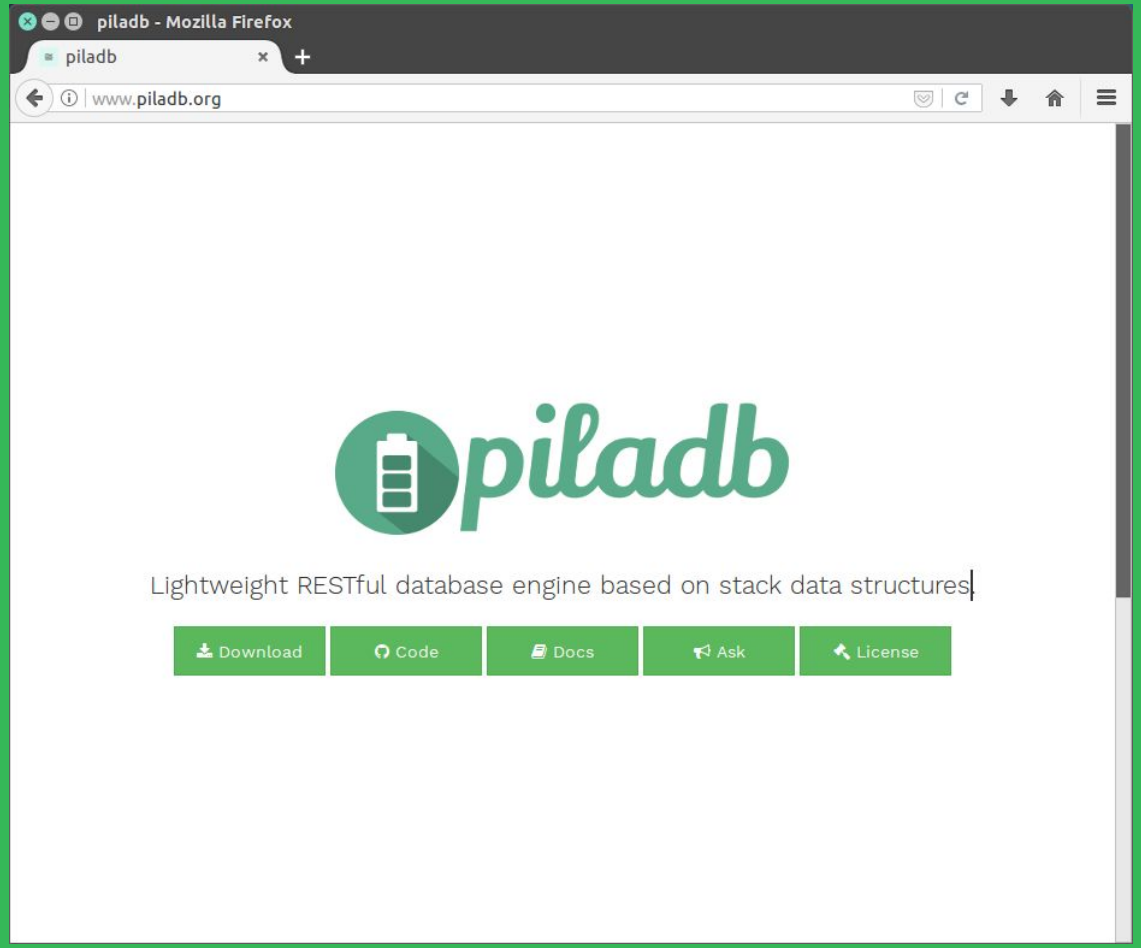
~~charge~~



~~superpowers~~

stack data structure


1
2
Push
1
2
3
Push
2
1
3
4
Push
3
2
1
4
5
Push
4
3
2
1
5
6
Push
5
4
3
2
1
6
6
Pop
5
4
3
2
1
7
5
Pop
4
3
2
1
8
4
Pop
3
2
1
9
3
Pop
2
1
10
2
Pop
1


piladb - Mozilla Firefox
piladb
www.piladb.org
piladb
Lightweight RESTful database engine based on stack data structures
Download
Code
Docs
Ask
License

database engine written in Go

lightweight and fast*

* no benchmarks yet, but feels fast!

```
piladb|master ⇒ find . -name '*.go'| grep -v vendor/ | grep -v "_test.go" | xargs wc -l
   17 ./pkg/stack/stacker.go
   83 ./pkg/stack/stack.go
   35 ./pkg/version/version.go
    8 ./pkg/date/date.go
   31 ./pkg/uuid/uuid.go
  107 ./pila/pila.go
  149 ./pila/stack.go
  138 ./pila/database.go
   65 ./pila/stack_status.go
    9 ./main.go
  395 ./pilad/conn.go
   57 ./pilad/router.go
   24 ./pilad/logo.go
  148 ./pilad/config.go
   68 ./pilad/utils.go
   30 ./pilad/main.go
   57 ./pilad/status.go
   65 ./config/value.go
   53 ./config/config.go
   57 ./config/vars/vars.go
 1596 total
```

Latest release

v0.1.0

0fd38ce

Version 0.1.0

Edit

fern4lvarez released this on 20 Dec 2016 · **12 commits** to master since this release

First release! 🎉

Downloads

piladb0.1.0.darwin-amd64.tar.gz	2.51 MB
piladb0.1.0.darwin-amd64.zip	2.51 MB
piladb0.1.0.linux-amd64.tar.gz	2.54 MB
piladb0.1.0.linux-amd64.zip	2.54 MB
Source code (zip)	
Source code (tar.gz)	

RESTful communication

PUSH, POP, PEEK, SIZE, FLUSH

JSON compatible elements

strings, numbers, objects, arrays, booleans, *null* [1]

[1] https://www.w3schools.com/js/js_json_datatypes.asp

no configuration files

no configuration files

environment variables

CLI parameters

inject via REST API

100% test coverage [1]

[1] https://codecov.io/gh/fern4lvarez/piladb

in-memory store

context and use cases

context

- Project started in October 2015
- More a Proof of Concept, less a solution to a problem
- Solo, side project
- First release in December 2016 (0.1.0)
- Main influences: Redis, Elasticsearch, CouchDB, BoltDB
- Not persistent, not distributed, *yet*

use cases

- Caching system
  - Invalidation using dates
  - All read and write ops are O(1)

use cases

- Caching system
  - Invalidation using dates
  - All read and write ops are O(1)
- Key-Value store with version history
  - Key: name of a Stack, Value: elements of the Stack

use cases

- Caching system
  - Invalidation using dates
  - All read and write ops are O(1)
- Key-Value store with version history
  - Key: name of a Stack, Value: elements of the Stack
- Undo/Redo mechanism

use cases

- Caching system
  - Invalidation using dates
  - All read and write ops are O(1)
- Key-Value store with version history
  - Key: name of a Stack, Value: elements of the Stack
- Undo/Redo mechanism
- Message processing

functionality and main components

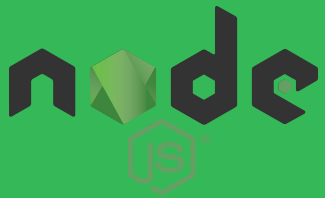
```
~|⇒
~|⇒ pilad
2017/02/15 03:11:25
2017/02/15 03:11:25          d8b 888                888 888
2017/02/15 03:11:25          Y8P 888                888 888
2017/02/15 03:11:25              888                888 888
2017/02/15 03:11:25 88888b.  888 888  8888b.   .d88888 88888b.
2017/02/15 03:11:25 888 "88b 888 888    "88b  d88" 888 888 "88b
2017/02/15 03:11:25 888  888 888 888 .d888888 888  888 888  888
2017/02/15 03:11:25 888 d88P 888 888 888  888 Y88b 888 888 d88P
2017/02/15 03:11:25 88888P"  888 888 "Y888888  "Y88888 88888P"
2017/02/15 03:11:25 888
2017/02/15 03:11:25 888
2017/02/15 03:11:25 888
2017/02/15 03:11:25
2017/02/15 03:11:25 Version: master
2017/02/15 03:11:25 Host:    linux_amd64
2017/02/15 03:11:25 Port:    1205
2017/02/15 03:11:25 PID:     13918
2017/02/15 03:11:25
```

with your favourite HTTP client...


curl://


Insomnia


POSTMAN


node
JS


php

CREATE DATABASE: `PUT /databases?name=MYDATABASE`

```
{
  "id": "22ffa4116b38da7988ebe505f9e129ba",
  "name": "MYDATABASE",
  "number_of_stacks": 0
}
```

CREATE STACK: `PUT /databases/MYDATABASE/stacks?name=MYSTACK`


```
{
  "id": "0c39814bad28d8b2ec5b4d697701c125",
  "name": "MYSTACK",
  "peek": null,
  "size": 0,
  "created_at": "2017-02-15T03:35:22.446530542+01:00",
  "updated_at": "2017-02-15T03:35:22.446530542+01:00",
  "read_at": "2017-02-15T03:35:22.446530542+01:00"
}
```

PUSH: POST /databases/MYDATABASE/stacks/MYSTACK {"element":"this is an element"}


```
{
  "id": "0c39814bad28d8b2ec5b4d697701c125",
  "name": "MYSTACK",
  "peek": "this is an element",
  "size": 1,
  "created_at": "2017-02-15T03:35:22.446530542+01:00",
  "updated_at": "2017-02-15T03:45:30.264123933+01:00",
  "read_at": "2017-02-15T03:46:18.060685036+01:00"
}
```

PUSH x 3: POST /databases/MYDATABASE/stacks/MYSTACK {"element":"this is the 4th element"}


```
{
  "id": "0c39814bad28d8b2ec5b4d697701c125",
  "name": "MYSTACK",
  "peek": "this is the 4th element",
  "size": 4,
  "created_at": "2017-02-15T03:35:22.446530542+01:00",
  "updated_at": "2017-02-15T03:48:47.699231061+01:00",
  "read_at": "2017-02-15T03:48:50.109298271+01:00"
}
```

POP: DELETE /databases/MYDATABASE/stacks/MYSTACK


```
{
  "id": "0c39814bad28d8b2ec5b4d697701c125",
  "name": "MYSTACK",
  "peek": "this is the 3rd element",
  "size": 3,
  "created_at": "2017-02-15T03:35:22.446530542+01:00",
  "updated_at": "2017-02-15T03:50:02.882637946+01:00",
  "read_at": "2017-02-15T03:50:34.287258581+01:00"
}
```

PEEK: GET /databases/MYDATABASE/stacks/MYSTACK?peek

```
{
  "element": "this is the 3rd element"
}
```

FLUSH: `DELETE /databases/MYDATABASE/stacks/MYSTACK?flush`


```
{
  "id": "0c39814bad28d8b2ec5b4d697701c125",
  "name": "MYSTACK",
  "peek": null,
  "size": 0,
  "created_at": "2017-02-15T03:35:22.446530542+01:00",
  "updated_at": "2017-02-15T03:52:27.494852286+01:00",
  "read_at": "2017-02-15T03:52:27.494852286+01:00"
}
```

stack

- LIFO: Last in, First out
- Stores JSON-compatible elements
- Unique name and identifier
- Created by user

database

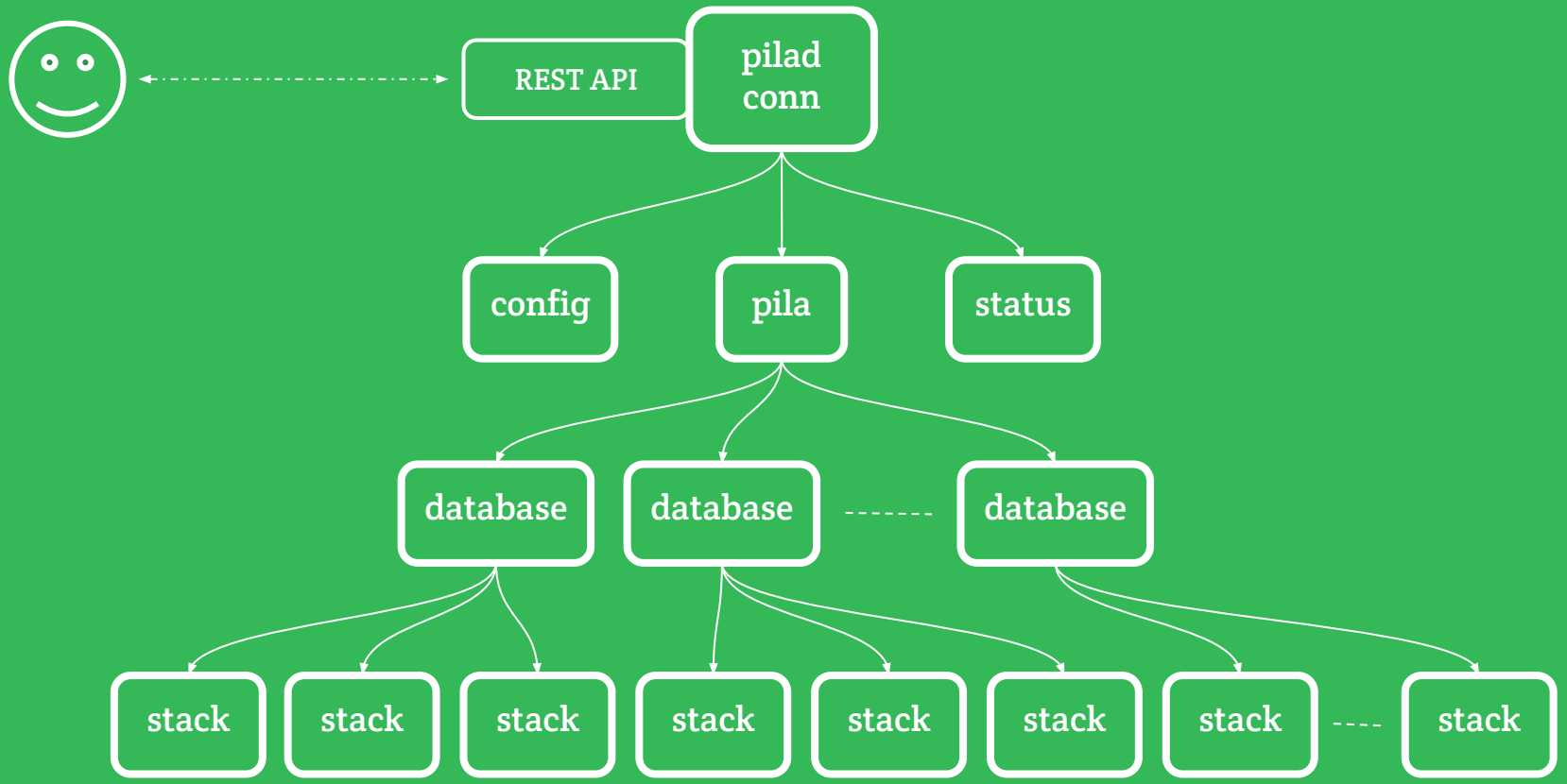
- Contains Stacks
- Unique name and identifier
- Created by user

pila

- Main entity of the engine
- Contains Databases
- Created by pilad on startup
- github.com/fern4lvarez/piladb/pila

pilad

- Daemon that starts piladb engine
- Implements REST API + configuration
- Creates a single instance of the Pila entity


REST API
pilad
conn
config
pila
status
database
database
database
stack
stack
stack
stack
stack
stack
stack
stack

internals in Go

Fernando Álvarez

implementation of a Stack

implementation is decoupled from the type

```go
// Stack represents a stack entity in piladb.
type Stack struct {
	// ID is a unique identifier of the Stack
	ID fmt.Stringer

	// Name of the Stack
	Name string

	// Database associated to the Stack
	Database *Database

	// CreatedAt represents the date when the Stack was created
	CreatedAt time.Time

	// UpdatedAt represents the date when the Stack was updated for the last time.
	// This date must be updated when a Stack is created, and when receives a PUSH,
	// POP, or FLUSH operation.
	// Note that unlike CreatedAt, UpdatedAt is not triggered automatically
	// when one of these events happens, but it needs to be set by hand.
	UpdatedAt time.Time

	// ReadAt represents the date when the Stack was read for the last time.
	// This date must be updated when a Stack is created, accessed, and when it
	// receives a PUSH, POP, or FLUSH operation.
	// Note that unlike CreatedAt, ReadAt is not triggered automatically
	// when one of these events happens, but it needs to be set by hand.
	ReadAt time.Time

	// base represents the Stack data structure
	base stack.Stacker
}
```

./pila/stack.go

interface that contains all Stack operations

```go
package stack

// Stacker represents an interface that contains all the
// required methods to implement a Stack that can be
// used in piladb.
type Stacker interface {
        // Push an element into a Stack
        Push(element interface{})
        // Pop the topmost element of a stack
        Pop() (interface{}, bool)
        // Size returns the size of the Stack
        Size() int
        // Peek returns the topmost element of the Stack
        Peek() interface{}
        // Flush flushes a Stack
        Flush()
}
```

./pkg/stack/stacker.go

use another Stack implementation by touching one line

```go
// NewStack creates a new Stack given a name and a creation date,
// without an association to any Database.
func NewStack(name string, t time.Time) *Stack {
	s := &Stack{}
	s.Name = name
	s.SetID()
	s.CreatedAt = t
	s.base = stack.NewStack()
	return s
}
```

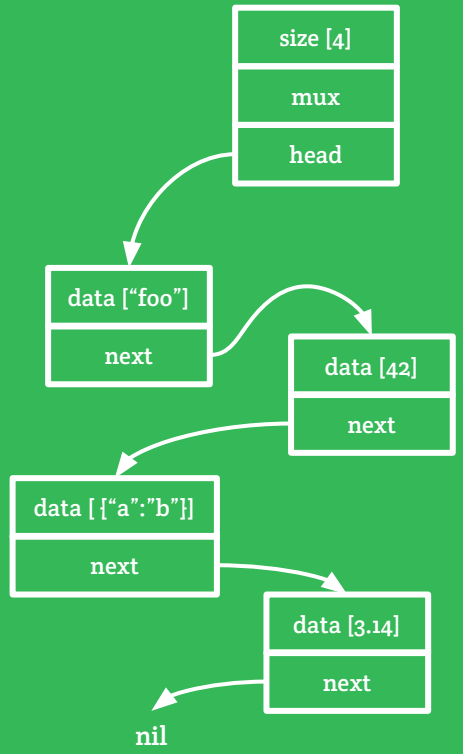
./pila/stack.go

current implementation uses linked lists

implementation using linked lists

```go
// Stack implements the Stacker interface, and represents the stack
// data structure as a linked list, containing a pointer
// to the first Frame as a head and the size of the stack.
// It also contain a mutex to lock and unlock
// the access to the stack at I/O operations.
type Stack struct {
        head *frame
        size int
        mux  sync.Mutex
}

// frame represents an element of the stack. It contains
// data and the link to the next Frame as a pointer.
type frame struct {
        data interface{}
        next *frame
}
```

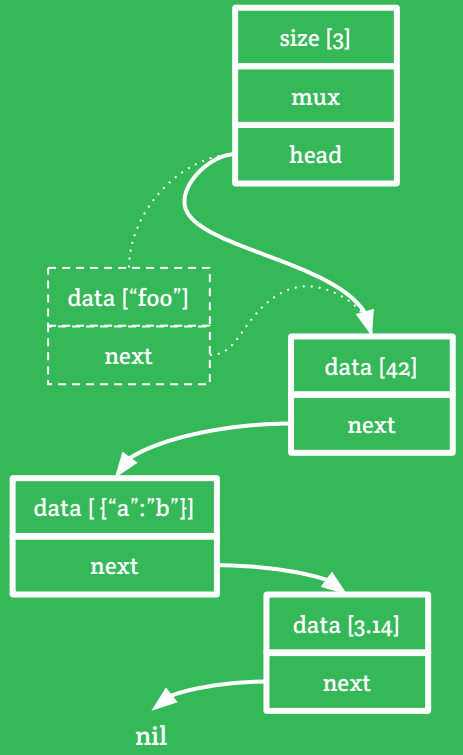
./pkg/stack/stack.go



implementation using linked lists

```go
// Pop removes and returns the element on top of the stack,
// updating its head to the next Frame. If the stack was empty,
// it returns false.
func (s *Stack) Pop() (interface{}, bool) {
	s.mux.Lock()
	defer s.mux.Unlock()

	if s.head == nil {
		return nil, false
	}

	element := s.head.data
	s.head = s.head.next
	s.size--
	return element, true
}
```

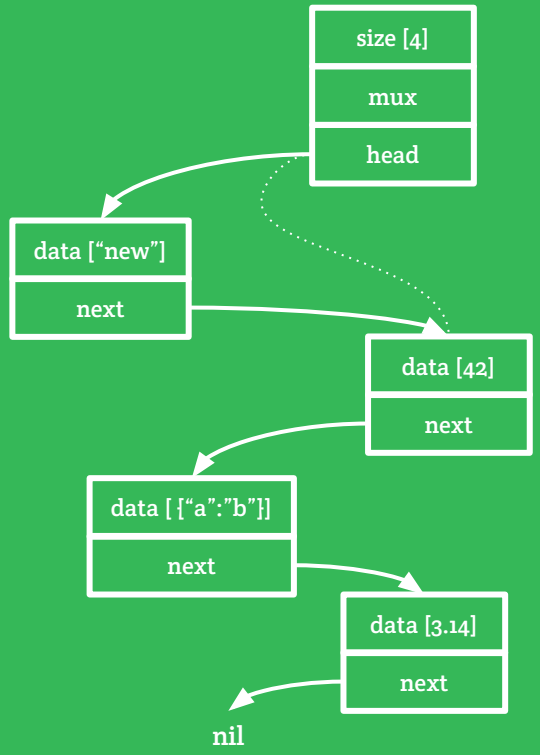
./pkg/stack/stack.go



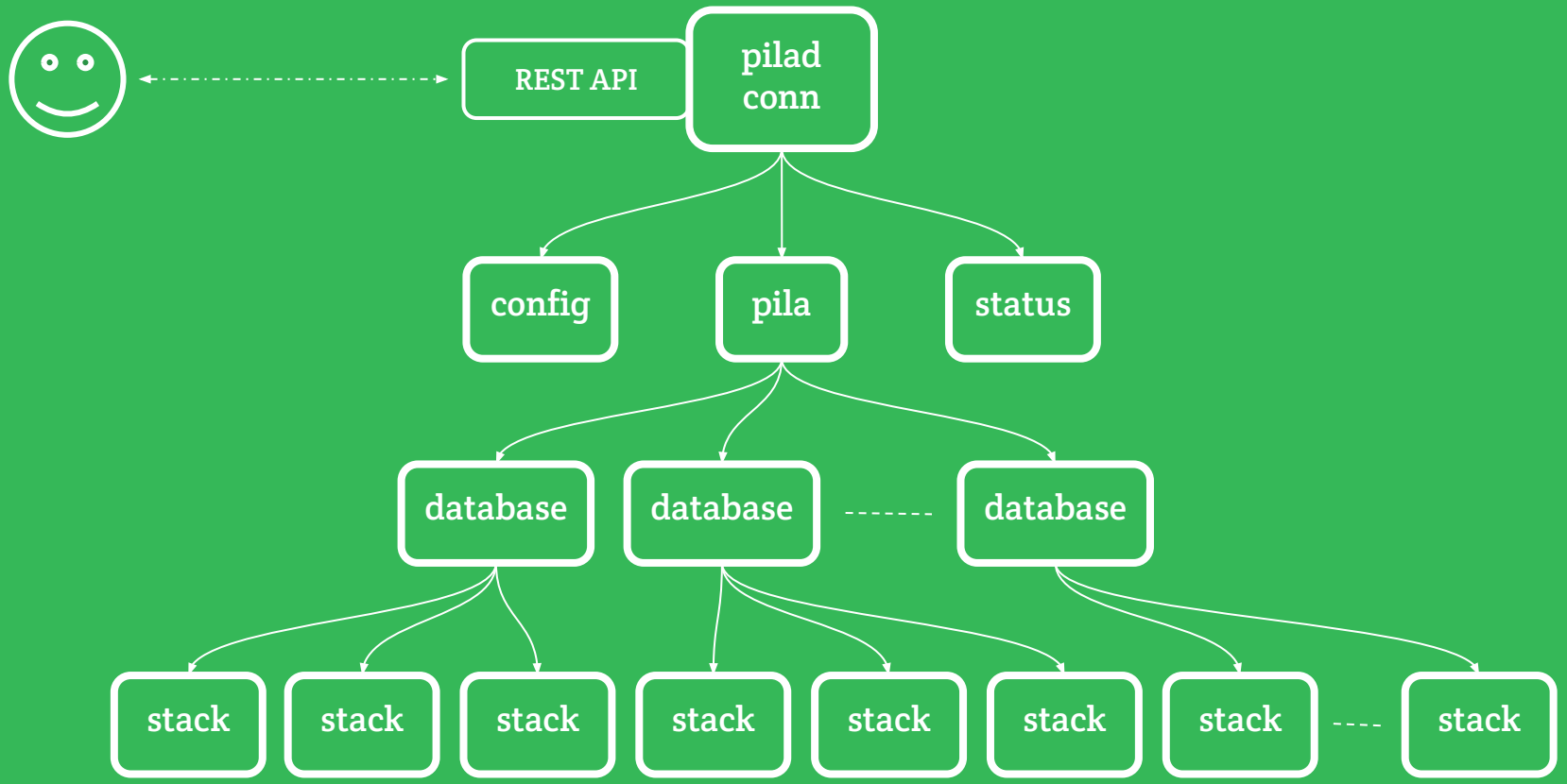
implementation using linked lists

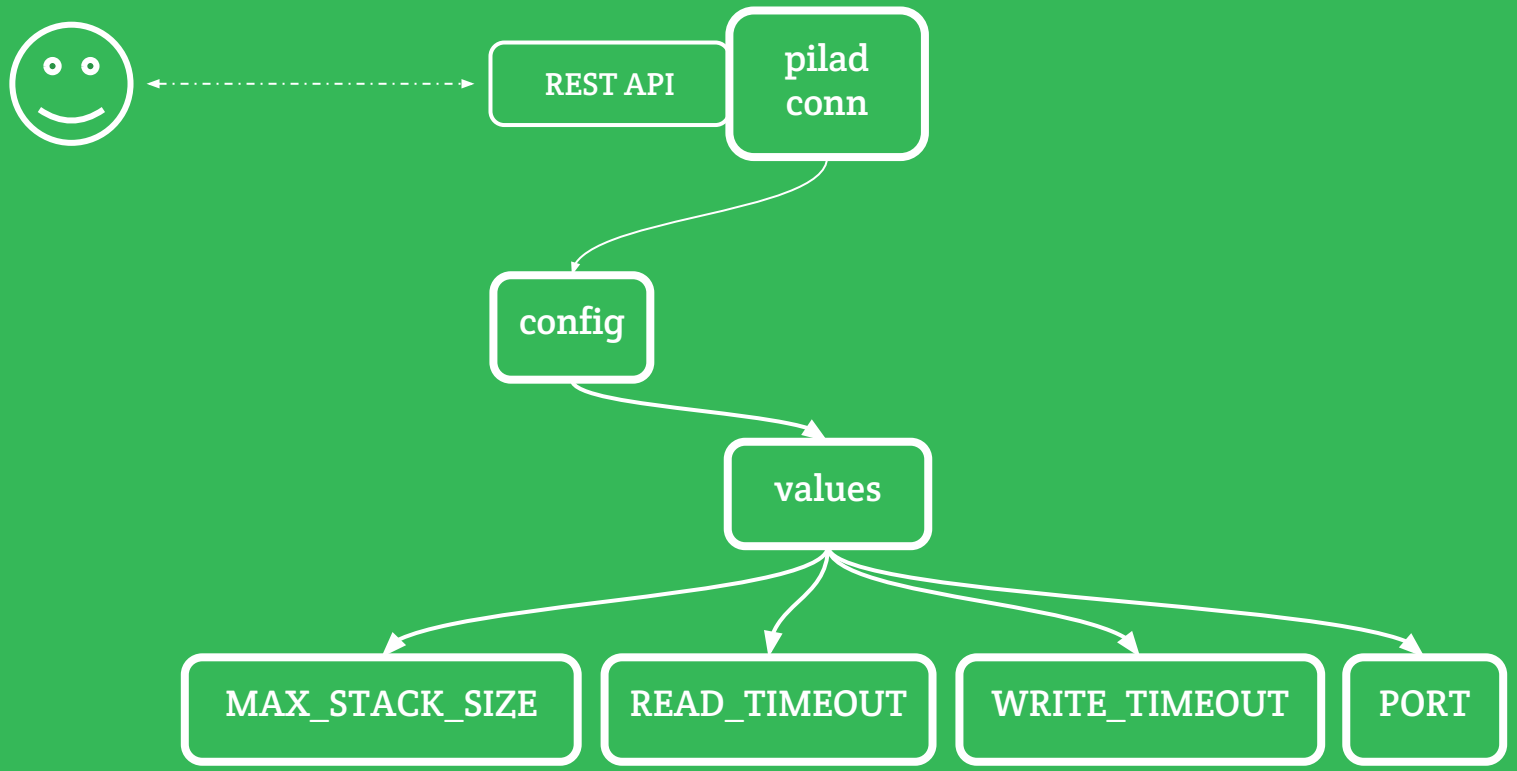
```go
// Push adds a new element on top of the stack, creating
// a new head holding this data and updating its head to
// the previous stack's head.
func (s *Stack) Push(element interface{}) {
	s.mux.Lock()
	defer s.mux.Unlock()

	head := &frame{
		data: element,
		next: s.head,
	}
	s.head = head
	s.size++
}
```

./pkg/stack/stack.go



implementation of Config


REST API
pilad
conn
config
pila
status
database
database
database
stack
stack
stack
stack
stack
stack
stack
stack


REST API
pilad conn
config
values
MAX_STACK_SIZE
READ_TIMEOUT
WRITE_TIMEOUT
PORT

config is like a Pila with a single Database

```go
// Config represents a Database containing all
// configuration values that will be
// updated and consumed by piladb.
type Config struct {
	Values *pila.Database
}
```

```go
// Get gets a config value from a key.
func (c *Config) Get(key string) interface{} {
	s, ok := c.Values.Stacks[uuid.New(CONFIG+key)]
	if !ok {
		return nil
	}
	return s.Peek()
}

// Set sets a config value having a key and the value.
func (c *Config) Set(key string, value interface{}) {
	s, ok := c.Values.Stacks[uuid.New(CONFIG+key)]
	if !ok {
		sID := c.Values.CreateStack(key, time.Now().UTC())
		s, _ = c.Values.Stacks[sID]
	}

	s.Push(value)
}
```

./config/config.go

export a func per config value

```go
// WriteTimeout returns the value of WRITE_TIMEOUT.
// Type: time.Duration, Default: 45
func (c *Config) WriteTimeout() time.Duration {
	writeTimeout := c.Get(vars.WriteTimeout)
	t := intValue(writeTimeout, vars.WriteTimeoutDefault)
	return time.Duration(t)
}

// Port returns the value of PORT.
// Type: int, Default: 1205
func (c *Config) Port() int {
	port := c.Get(vars.Port)
	t := intValue(port, vars.PortDefault)

	if t < 1025 || t > 65536 {
		return vars.PortDefault
	}
	return t
}
```

./config/value.go

set or modify a config var on runtime:

POST /_config/READ_TIMEOUT {"element":15}

```
104                 if r.Method == "POST" {
105                         if r.Body == nil {
106                                 log.Println(r.Method, r.URL, http.StatusBadRequest,
107                                         "no element provided")
108                                 w.WriteHeader(http.StatusBadRequest)
109                                 return
110                         }
111                         err := element.Decode(r.Body)
112                         if err != nil {
113                                 log.Println(r.Method, r.URL, http.StatusBadRequest,
114                                         "error on decoding element:", err)
115                                 w.WriteHeader(http.StatusBadRequest)
116                                 return
117                         }
118
119                         c.Config.Set(vars["key"], element.Value)
120                 }
```

./pilad/config.go

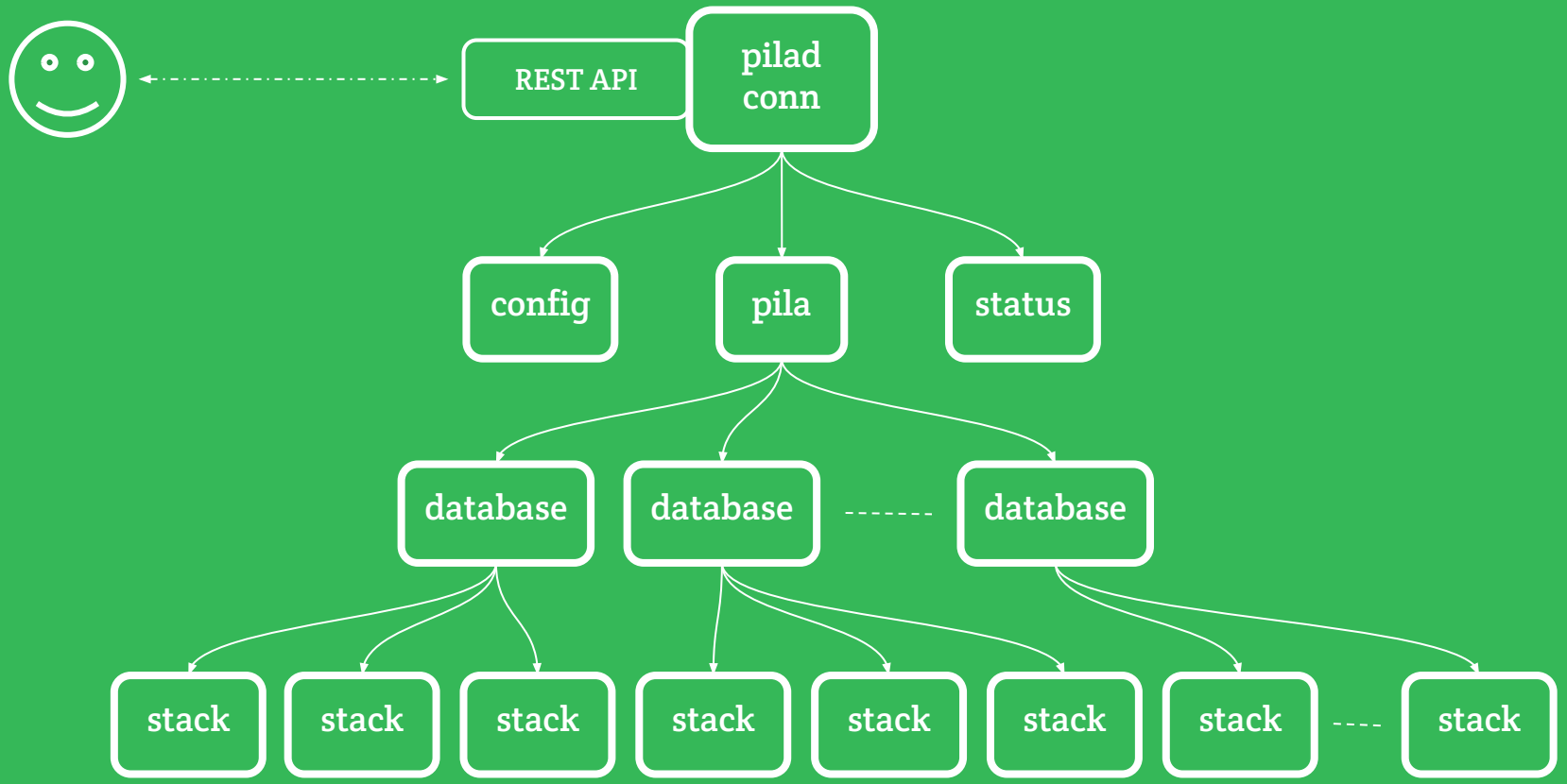
read config values on runtime: PUSH

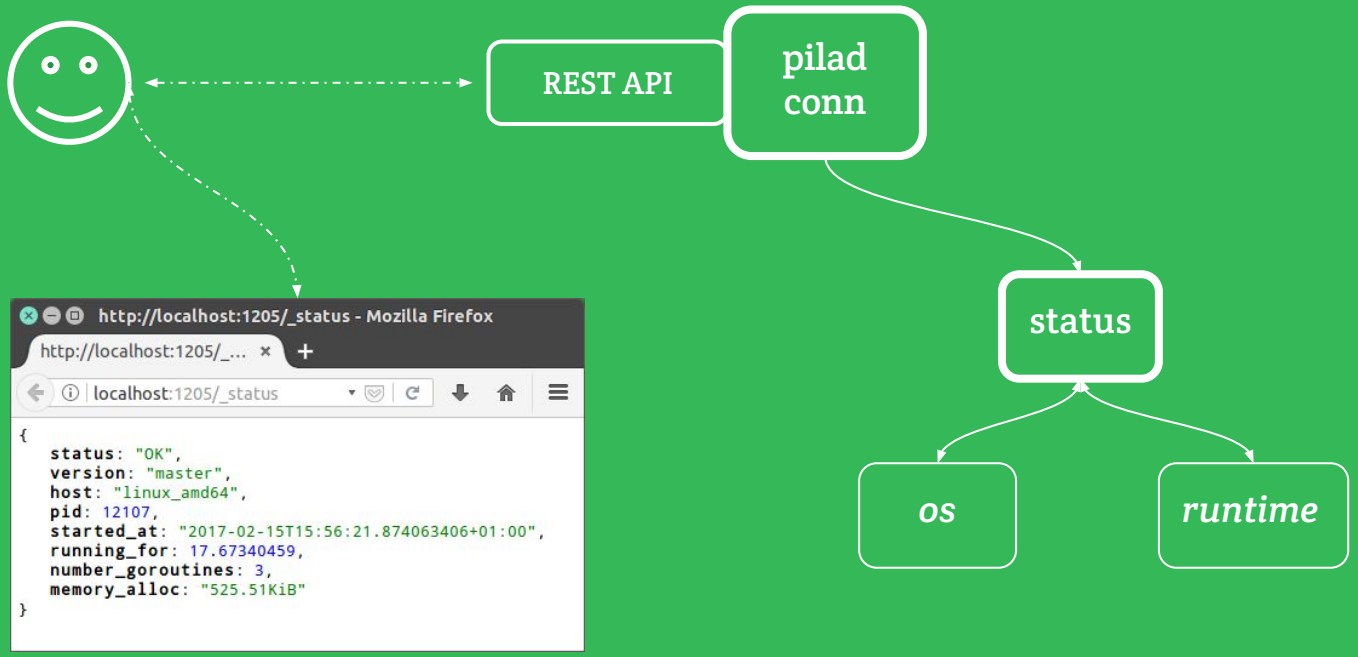
```
246                 case r.Method == "POST":
247                         c.checkMaxStackSize(c.pushStackHandler)(w, r, stack)
248                         return
```

```
136 // checkMaxStackSize checks config value for MaxStackSize and execute the
137 // wrapped handler if check is validated.
138 func (c *Conn) checkMaxStackSize(handler stackHandlerFunc) stackHandlerFunc {
139         return func(w http.ResponseWriter, r *http.Request, stack *pila.Stack) {
140                 if s := c.Config.MaxStackSize(); stack.Size() >= s && s != -1 {
141                         log.Println(r.Method, r.URL, http.StatusNotAcceptable, vars.MaxStackSize, "value reached")
142                         w.WriteHeader(http.StatusNotAcceptable)
143                         return
144                 }
145
146                 handler(w, r, stack)
147         }
148 }
```

./pilad/config.go

implementation of Status


REST API
pilad
conn
config
pila
status
database
database
database
stack
stack
stack
stack
stack
stack
stack
stack


REST API
pilad
conn
status
os
runtime
http://localhost:1205/_status - Mozilla Firefox
http://localhost:1205/_...
localhost:1205/_status
{
status: "OK",
version: "master",
host: "linux_amd64",
pid: 12107,
started_at: "2017-02-15T15:56:21.874063406+01:00",
running_for: 17.67340459,
number_goroutines: 3,
memory_alloc: "525.51KiB"
}

os

- `os.Getpid()`: piladb process ID

runtime

- `runtime.GOOS`: Host operating system
- `runtime.GOARCH`: Host architecture
- `runtime.NumGoroutine()`: Number of existing goroutines
- `runtime.MemStats`: Statistics about memory allocator

```go
// A MemStats records statistics about the memory allocator.
type MemStats struct {
	// General statistics.
	Alloc      uint64 // bytes allocated and not yet freed
	TotalAlloc uint64 // bytes allocated (even if freed)
	Sys        uint64 // bytes obtained from system (sum of XxxSys below)
	Lookups    uint64 // number of pointer lookups
	Mallocs    uint64 // number of mallocs
	Frees      uint64 // number of frees

	// Main allocation heap statistics.
	HeapAlloc    uint64 // bytes allocated and not yet freed (same as Alloc above)
	HeapSys      uint64 // bytes obtained from system
	HeapIdle     uint64 // bytes in idle spans
	HeapInuse    uint64 // bytes in non-idle span
	HeapReleased uint64 // bytes released to the OS
	HeapObjects  uint64 // total number of allocated objects

	// Low-level fixed-size structure allocator statistics.
	//	Inuse is bytes used now.
	//	Sys is bytes obtained from system.
	StackInuse  uint64 // bytes used by stack allocator
	StackSys    uint64
	MSpanInuse  uint64 // mspan structures
	MSpanSys    uint64
	MCacheInuse uint64 // mcache structures
	MCacheSys   uint64
	BuckHashSys uint64 // profiling bucket hash table
	GCSys       uint64 // GC metadata
	OtherSys    uint64 // other system allocations

	// Garbage collector statistics.
	NextGC        uint64 // next collection will happen when HeapAlloc ≥ this amount
	LastGC        uint64 // end time of last collection (nanoseconds since 1970)
	PauseTotalNs  uint64
	PauseNs       [256]uint64 // circular buffer of recent GC pause durations, most recent at [(NumGC+255)%256]
	PauseEnd      [256]uint64 // circular buffer of recent GC pause end times
	NumGC         uint32
	GCCPUFraction float64 // fraction of CPU time used by GC
	EnableGC      bool
	DebugGC       bool

	// Per-size allocation statistics.
	// 61 is NumSizeClasses in the C code.
	BySize [61]struct {
		Size    uint32
		Mallocs uint64
		Frees   uint64
	}
}
```

go/src/runtime/memstats.go

pull MemStats on each request

```go
// Update updates the Status given a current time and memory
// stats.
func (s *Status) Update(now time.Time, mem *runtime.MemStats) {
	s.RunningFor = now.Sub(s.StartedAt).Seconds()
	s.NumberGoroutines = runtime.NumGoroutine()
	s.MemoryAlloc = MemOutput(mem.Alloc)
}
```

./pilad/status.go

```go
// statusHandler writes the piladb status into the response.
func (c *Conn) statusHandler(w http.ResponseWriter, r *http.Request) {
	c.Status.Update(time.Now().UTC(), MemStats())

	w.Header().Set("Content-Type", "application/json")
	log.Println(r.Method, r.URL, http.StatusOK)
	w.Write(c.Status.ToJSON())
}
```

./pilad/conn.go

```go
// MemStats fetches the memory statistics provided
// by the Go stdlib.
func MemStats() *runtime.MemStats {
	var mem runtime.MemStats
	runtime.ReadMemStats(&mem)
	return &mem
}
```

./pilad/utils.go

demo

future plans

0.1.x

- Update to Go 1.8
- More docs!
  - Internals
  - Configuration
  - Go package
- Go client library

0.2.x

- New Stack operations
  - ROTATE, BASE, BLOCK, CONCAT
  - Spoiler! Stacks will be Queues too
- Option to allow pushing to Stack when it is full
  - Bottommost element will be removed
- More clients libraries?
  - Ruby, JS, Python, PHP, Java...

0.3.x

- Version control
  - Using activity logs
- Restore from any point in time
  - Persistence!
- New logo
  - Bye battery!

Future

- Replication
- pilaql (query language)
- Authentication
- TLS
- JSON API
- ... and more!

help!

oscillatingworks / piladb-go

Unwatch 2 | Star 0 | Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

Official piladb client library in Go https://www.piladb.org

Edit

piladb database client go golang http stack Manage topics

2 commits | 1 branch | 0 releases | 1 contributor | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

fern4lvarez committed on GitHub Update README.md — Latest commit 55d867a 2 days ago

.gitignore	Initial commit	2 days ago
LICENSE	Initial commit	2 days ago
README.md	Update README.md	2 days ago

README.md

piladb-go

Official piladb client library in Go

all it takes to get started

```
go get -u github.com/fern4lvarez/piladb/...
cd $GOPATH/src/github.com/fern4lvarez/piladb
make
```

https://www.piladb.org

https://github.com/fern4lvarez/piladb

https://docs.piladb.org

https://www.reddit.com/r/piladb/

https://www.oscillating.works

@oscillatingw

thank you!
questions?