# piladb

# making a small database engine from scratch

by Fernando Álvarez

Codemotion Berlin — 13/10/2017

# Summary

- `whoami`

# Summary

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

# Summary

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

# Summary

- `whoami`
- Introduction to **piladb**
- Inception
- Development

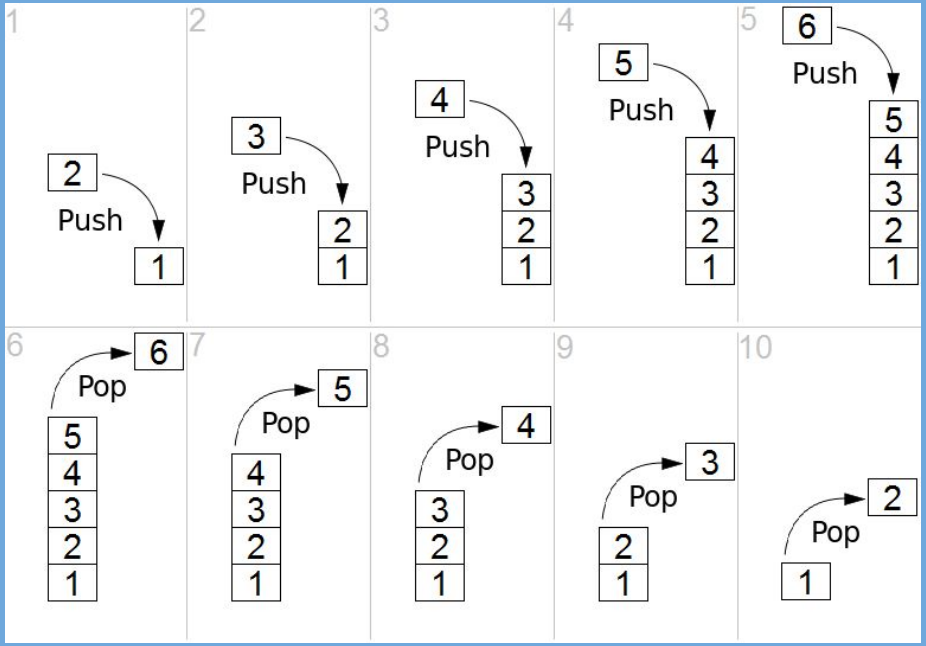
# `whoami`

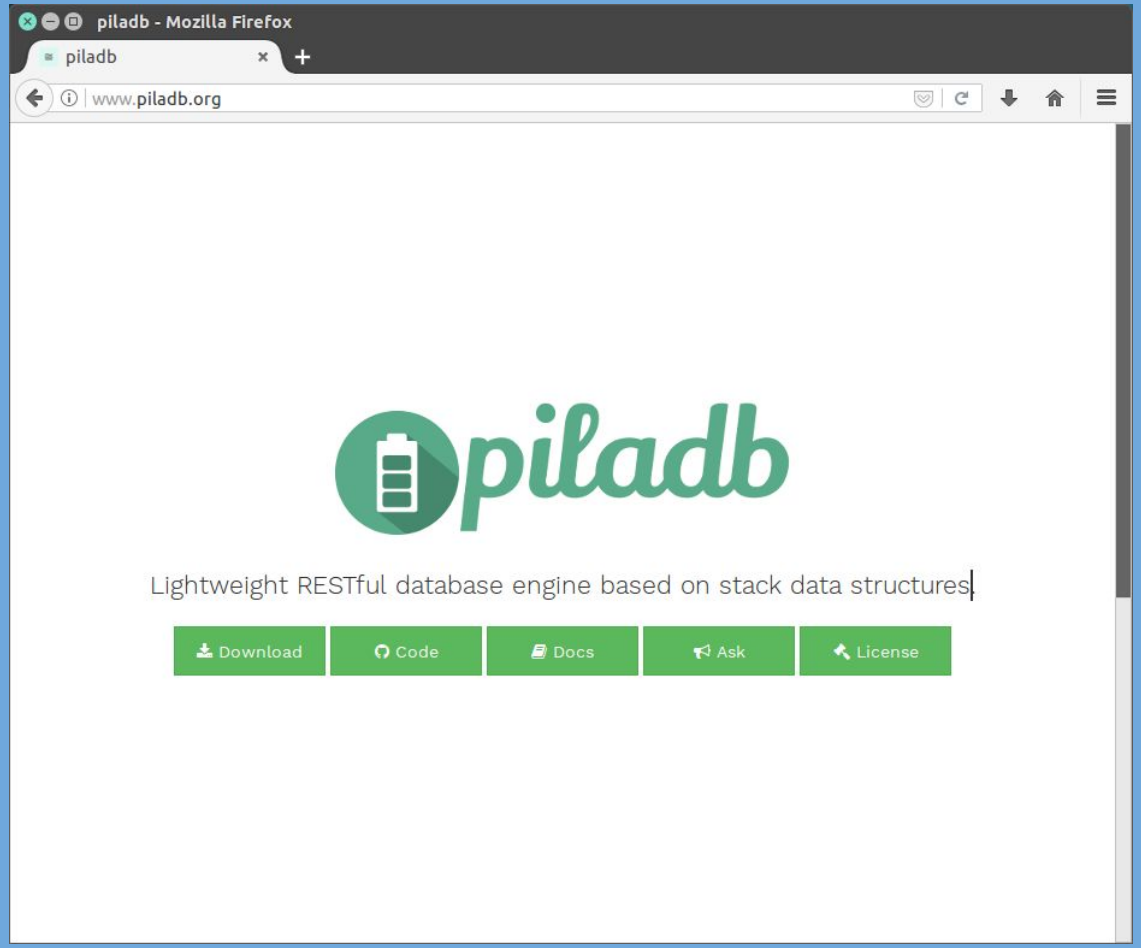
# Fernando Álvarez

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

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

# 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, 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

v0.1.4

57c670a

Unverified

# Version 0.1.4

Edit

**fern4lvarez** released this on Jun 20 · **3 commits** to master since this release

One more release! This is about stability and data .

What's new?

- Build `pilad` with go1.8.3
- Fix data race conditions on Database and Stack types
- New `Stack.UUID()` function to get thread-safe Stack ID
- Introduce new `make race` to find data races in tests

Release inspired by this conversation in Reddit.

See the full diff here: `v0.1.3...v0.1.4`

## Downloads

| | |
|---|---|
| **piladb0.1.4.darwin-amd64.tar.gz** | 2.25 MB |
| **piladb0.1.4.darwin-amd64.zip** | 2.25 MB |
| **piladb0.1.4.linux-amd64.tar.gz** | 2.27 MB |
| **piladb0.1.4.linux-amd64.zip** | 2.27 MB |
| **Source code** (zip) | |
| **Source code** (tar.gz) | |

# RESTful + HTTP 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

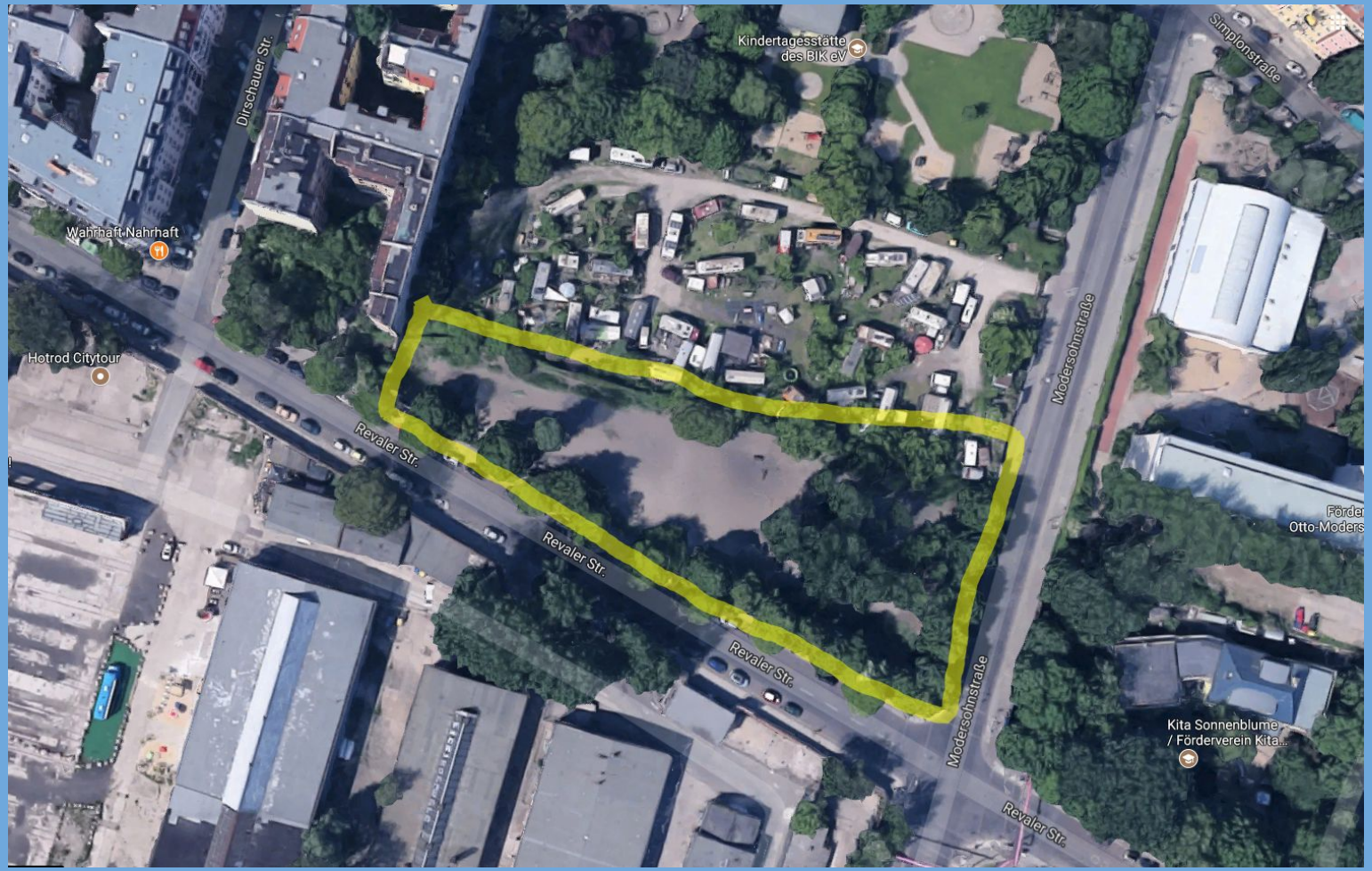
# in-memory store

# 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

```
~|⇒
~|⇒ 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
```

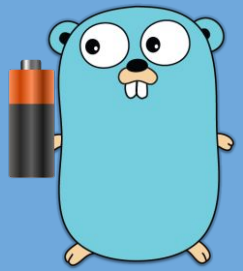
# inception

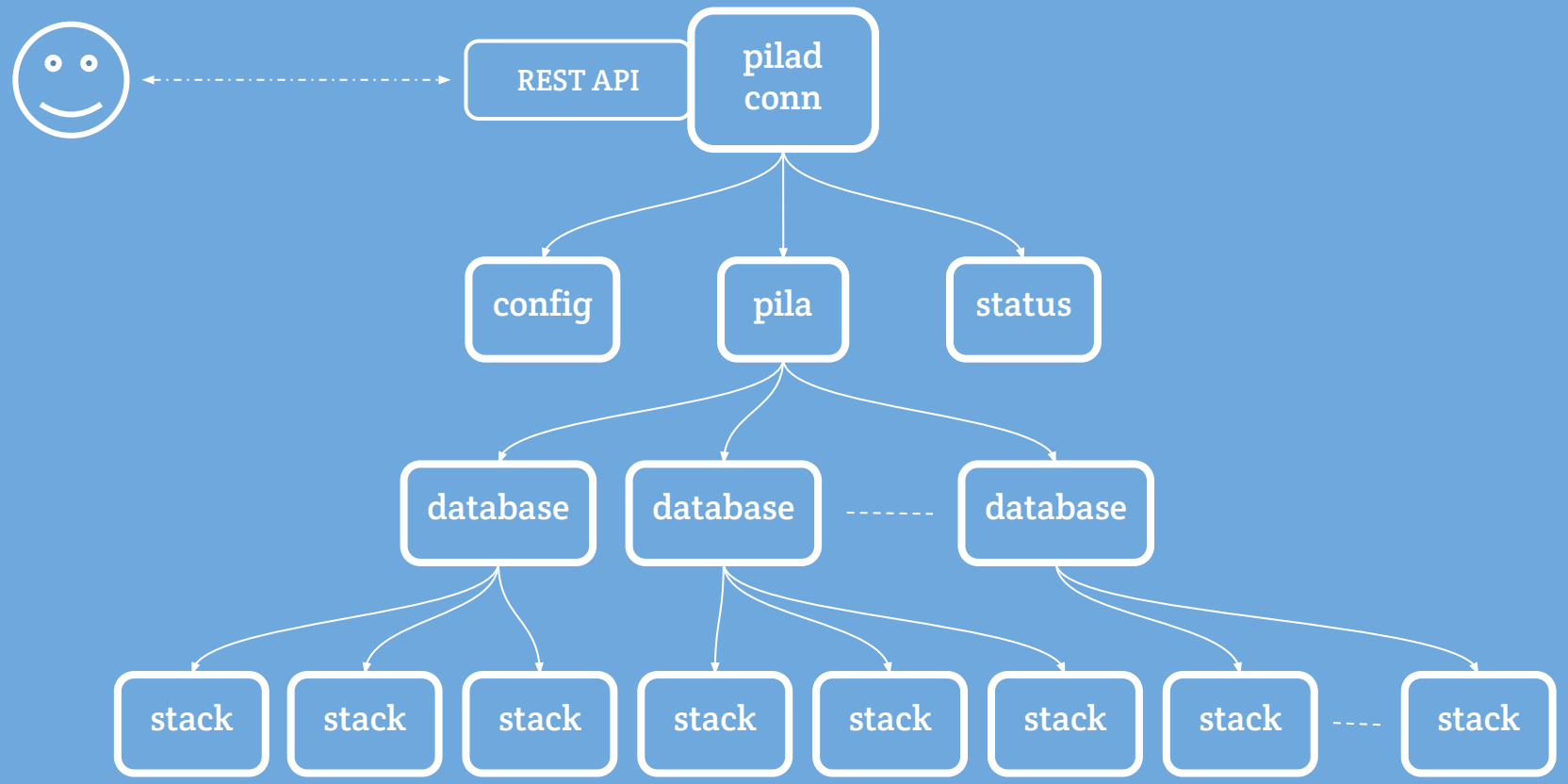

Kindertagesstätte des BIK eV
Simplonstraße
Dirschauer Str.
Wahrhaft Nahrhaft
Hotrod Citytour
Revaler Str.
Revaler Str.
Revaler Str.
Modersohnstraße
Modersohnstraße
Förder
Otto-Moders
Kita Sonnenblume / Förderverein Kita...
Revaler Str.

# premises

- Written in Go
- Usable for everyone, *i.e. not only Go devs*
- Can apply to real world use cases
- Simple, but challenging
- Learn from the experience

# development




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

# 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

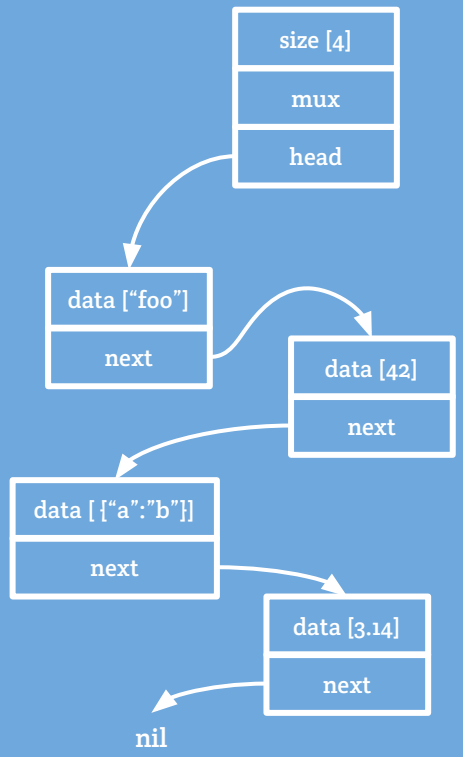
current implementation uses linked lists

# 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



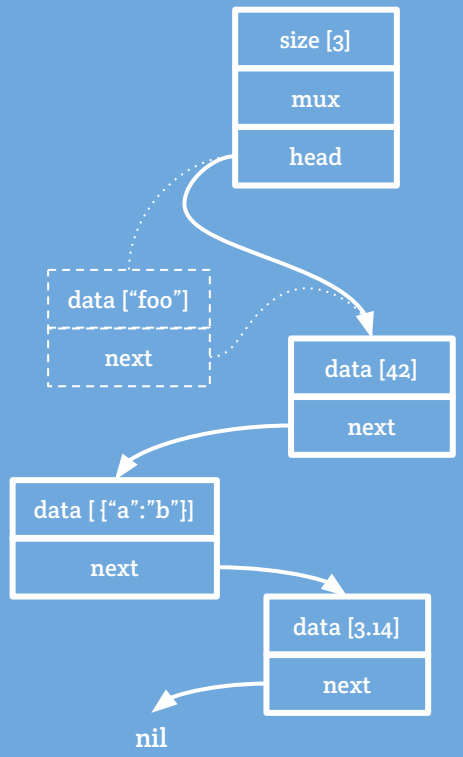
# implementation using linked lists

```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

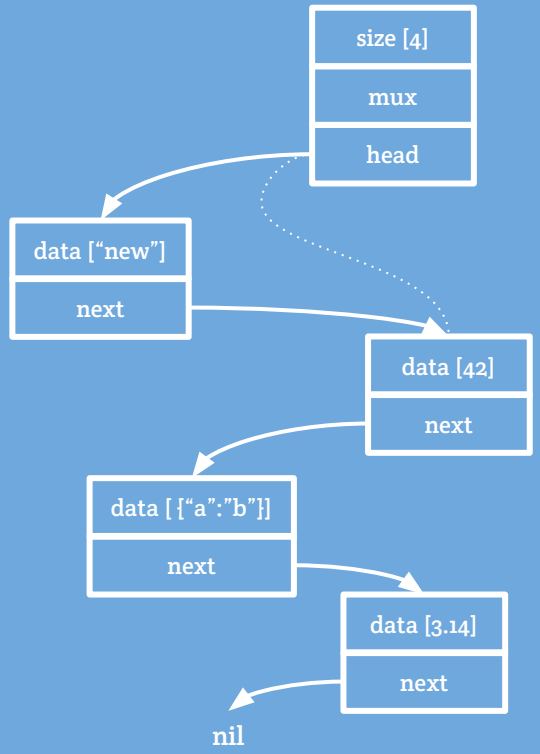


# implementation using linked lists

```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



# trust in `encoding/json`

```go
// Element represents the payload of a Stack element.
type Element struct {
	Value interface{} `json:"element"`
}

// ToJSON converts an Element into JSON.
func (element Element) ToJSON() ([]byte, error) {
	return json.Marshal(element)
}

// Decode decodes json data into an Element.
func (element *Element) Decode(r io.Reader) error {
	elementBuffer := new(bytes.Buffer)
	elementBuffer.ReadFrom(r)

	if !bytes.HasPrefix(elementBuffer.Bytes(), []byte(`{"element"`)) {
		return errors.New("malformed payload, missing element key?")
	}

	decoder := json.NewDecoder(elementBuffer)
	return decoder.Decode(element)
}
```

./pila/stack.go

# `net/http` **handlers**

```go
// pushStackHandler adds an element into a Stack and returns 200 and the element.
func (c *Conn) pushStackHandler(w http.ResponseWriter, r *http.Request, stack *pila.Stack) {
	if r.Body == nil {
		log.Println(r.Method, r.URL, http.StatusBadRequest,
			"no element provided")
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	var element pila.Element
	err := element.Decode(r.Body)
	if err != nil {
		log.Println(r.Method, r.URL, http.StatusBadRequest,
			"error on decoding element:", err)
		w.WriteHeader(http.StatusBadRequest)
		return
	}

	stack.Push(element.Value)
	stack.Update(c.opDate)

	log.Println(r.Method, r.URL, http.StatusOK, element.Value)
	w.Header().Set("Content-Type", "application/json")

	// Do not check error as we consider our element
	// suitable for a JSON encoding.
	b, _ := element.ToJSON()
	w.Write(b)
}
```

./pilad/conn.go

# Summary

- Interconnected data structures
- Encoding/decoding data with `encoding/json`
- `net/http` to connect client with data structures + content

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?