Bienvenue à la Sfeir School
GO 200

School

Déroulement de la formation

C’est quand la pause ?
Quand est-ce qu’on mange ?
Feuille de présence…
Tour de table…

Présentation

[sf≡ir]

Yves DAUTREMAY

Tech lead Java - DevOps

Présentation

[sf≡ir]

Sébastien FRIESS

Developer backend

@sebastienfriess

Présentation

[sf≡ir]

Yohann FACON

Developer backend

Présentation

[sf≡ir]

Vincent DOLEZ

Developer backend / data

@dolez_v

Présentation

[sf≡ir]

Olivier GERARDIN

Architecte Java

@ogerardin

Présentation

[sf≡ir]

Antoine POIVEY

Full Stack Developer

Présentation

[sf≡ir]

Olivier FUXET

Developer backend

@ofuxet

Déroulement de la formation

- Mise en place de l’environnement de développement
- Rappel de la syntaxe de Go
- Développement par branche
- Container, Run et Tests

Environnement de dev

Installation

- Git
- Go : **https://golang.org/dl** & **https://golang.org/doc/install**
- Docker: **https://docs.docker.com/engine/installation**
- Docker-compose: **https://docs.docker.com/compose/install**
- **Visual Studio Code**, **Goland**, **IntelliJ + plugin Go** ou **VimGo**
- JQ: **https://stedolan.github.io/jq/download**
- ou Postman

Workspace Go

- **$GOROOT** : votre installation de Go (pas nécessaire)
- **$GOPATH** : en Go **1.11** $HOME/Go (à définir pour bon fonctionnement du **Makefile**)
  - Votre workspace de base contient
    - **bin** : binaires de vos appli
    - **pkg** : vos objets à linker
    - **src** : toutes vos sources
  - Ajouter les bin Go à votre path : **PATH=$PATH:$GOPATH/bin**
- Il vous faut cloner :
  - **git clone** https://github.com/Sfeir/golang-200
  - dans **$GOPATH/src/github.com/Sfeir/**golang-200
  - vous positionner sur la branche **step01** (git checkout -f step01)

Workspace Go

- Les commandes Go :
  - **build** compile packages and dependencies
  - **fmt** run gofmt on package sources
  - **test** test packages
  - **tool** run specified go tool
  - **get** download and install packages and dependencies
  - **dep** gestion des dépendances (à venir dans le tooling)

Workspace Go

- Le Makefile :
  - make (**all**) ⇒ compilation
  - make **help** ⇒ aide
  - make **test** ⇒ pour lancer les tests
  - make **bench** ⇒ pour lancer les benchmarks
  - make **benchTool** ⇒ pour lancer les benchmarks et pprof
  - make **docker* (Build/BuildMulti/Up/Down/BuildUp/BuildUpMulti)**

Le langage

- Né en 2009 chez Google (après les processeurs multi-coeurs) et OSS
- Binaire compilé autoporteur (début plugin depuis Go 1.8)
- Garbage collector (sub millisecond pour 17 Go de heap)
- Pointeurs 😱
- Goroutines
  - Assimilable à un thread
  - Mais ce n'est **PAS** un thread ⇒ **beaucoup plus léger**
  - Multithreadées sur un pool de thread
- Channels
  - **Do not communicate by sharing memory; share memory by communicating.**
  - Synchronisation
  - Multiplexage (**select**)

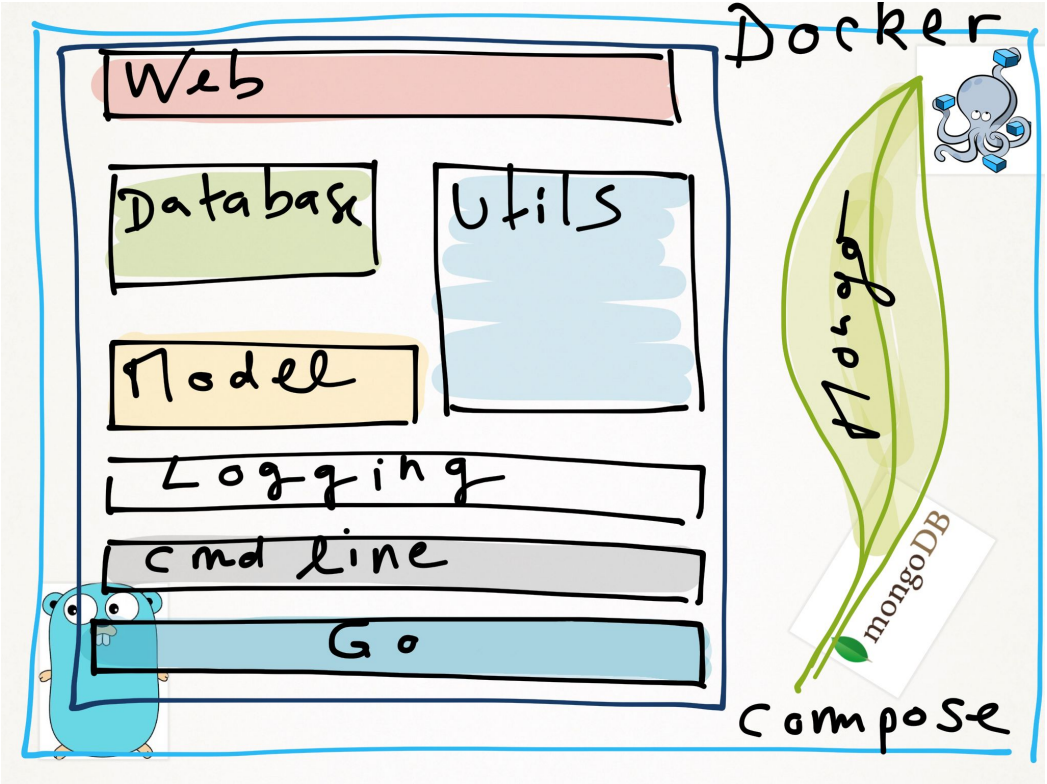
Le langage

- Les Mots-Clés :
  - **Dépendances** : import package
  - **Conditionnelles** : if else switch case fallthrough break default goto select
  - **Itérations** : for range continue
  - **Type** : var func interface struct chan const type map make
  - **Misc** : defer go return panic recover

Objectifs de la formation

- Construire et exécuter mon premier **microservice** CRUD en **Go**
  - un processus lancé en **ligne de commande**
  - qui expose une **API RESTful** en **JSON**
  - qui utilise **MongoDB** pour sa persistance
  - et cherry on the cake : **containerisée** dans docker

Objectifs de la formation



Parsing des arguments

```
git checkout -f step01
```

Parsing des arguments

Les paramètres de lancement de la ligne de commande:

- **port** = 8020
- **logLevel** = "warning"
- **db** = "mongodb://mongo/tasks"
- **dbType** = dao.DAOMockStr
- **migrationPath** = "migration"
- **logFormat** = utils.TextFormatter
- **statisticsDuration** = 20 * time.Second

Parsing des arguments

Go possède sa propre lib de parsing de la ligne de commande :

- https://golang.org/pkg/flag

```
package main

import (
        "fmt"
        "flag"
)

func main() {
        var flagvar int

        flag.IntVar(&flagvar, "intp", 42, "help message for int flag")

        flag.Parse()

        fmt.Println("flagvar has value ", flagvar)
}
```

```
$ go run main.go -intp 314
flagvar has value  314
```

Parsing des arguments

Mais on est barbu·e ou on l'est pas…

- https://github.com/urfave/cli (ex codegangsta)
- **v1** stable (gopkg.in/urfave/cli.v1), **v2** en cours de dev

```
package main

import "github.com/urfave/cli"

var port = 8020

func main() {
        // new app
        app := cli.NewApp()
        app.Name = "mytodolist"
        app.Usage = "mytodolist service launcher"
        // [...]
}
```

Parsing des arguments

```
package main

import "github.com/urfave/cli"

var port = 8020

func main() {
        // [...]
        // command line flags
        app.Flags = []cli.Flag{
                cli.IntFlag{
                        Value: port,
                        Name:  "port",
                        Usage: "Set the listening port of the webserver",
                        Destination: &port,
                        EnvVar: "APP_PORT",
                },
        }
}
```

Parsing des arguments

```
package main

import "github.com/urfave/cli"

var port = 8020

func main() {
        // [...]
        // main action
        // sub action are possible also
        app.Action = func(c *cli.Context) error {
                // parse parameters
                fmt.Printf("port : %d\n", port)
                return nil
        }

        // run the app
        err := app.Run(os.Args)
        if err != nil {
                fmt.Errorf("Run error %q\n", err)
        }
}
```

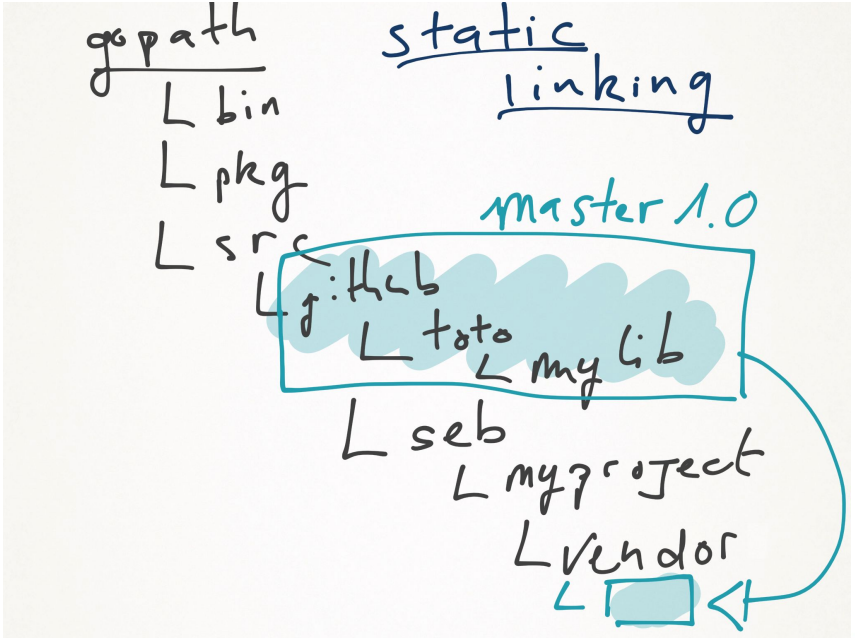
```
$ go run main.go -port 8090

port : 8090

$ go run run main.go -port 8090xy

Incorrect Usage. invalid value
"8090xy" for flag -port:
strconv.ParseInt: parsing
"8090xy": invalid syntax

NAME:
   mytodolist - mytodolist service
launcher

[...]

GLOBAL OPTIONS:
   --port value    Set the listening
port of the webserver (default:
8020)
   --help, -h      show help
   --version, -v   print the version
```

Parsing des arguments - Exercice

/todolist.go

Les dépendances Go

- Go est un langage aux **liens statiques**
- Pour un build **reproductible**
  - Dépendances versionnées avec un proxy d'import : **https://gopkg.in**
  - Disponibilité de toutes **les sources** même celles de ses **dépendances**
- Introduction du répertoire **vendor** :
  - Go 1.5 **GO15VENDOREXPERIMENT=1**
  - Go 1.6 par défaut
  - Plus de 15 outils de vendoring existants (Glide, Godep, Govendor, ...)
  - **dep** officiel intermédiaire
  - **mod** le final



Les dépendances Go

Gestion de dépendance avec l'outil **dep** https://github.com/golang/dep

Processus avec **Dep** from **scratch**

- Je fais mon **dev**
- Je **go get** mes dépendances **GOPATH**
- J'installe **dep**
- **dep init**
- **dep ensure**
- **commit** ou pas vendor

Processus avec **Dep** en **update**

- Je **descends** mon projet avec **Git**
- **dep ensure** ou pas
- **dep ensure -add github.com/foo/bar**
- **dep status**
- **dep ensure -update**

git checkout -f step02

Logging

Attention, les slides suivantes contiennent des scènes pouvant heurter votre sensibilité.

Logging

- Go possède sa propre lib de **logging** mais à l'instar du langage elle est **minimaliste**
  - Pas de niveaux de log 😱
  - Pas de configuration par package
  - Configuration As Code de la sortie (Out, Err)
- La lib alternative la plus répandue : https://github.com/sirupsen/logrus 🦭
  - facile d'utilisation
  - thread safe
  - structuré
  - avec des formateurs
  - des connecteurs / hooks nombreux (elastic, influx, syslog, ...)
  - est "relativement" configurable (formateur, sortie, niveaux AsCode)

Logging

```
package main

import (
	"github.com/sirupsen/logrus"
	"os"
)

func main() {
	logrus.SetFormatter(&logrus.TextFormatter{
		ForceColors:   true,
		FullTimestamp: true,
	})

	logrus.SetOutput(os.Stdout)
	logrus.SetLevel(logrus.DebugLevel)

	logrus.WithField("error", err).
		Warn("error setting log level, using debug as default")

}
```

```
WARN[2017-03-04T23:37:34+01:00] error setting log level, using debug as default  error="log an error"
```

Logging - Exercice

/utils/logger.go

git checkout -f step03

Concurrence / multithreading

- Problématique : compter le nombre de **requêtes par seconde** sur notre API
- Environnement : **multi-threadé** (Go routines)
- Contrainte : pas de **Mutex** pour ne pas plomber les **perfs**
- Solution apportée par Go : **les channel** !

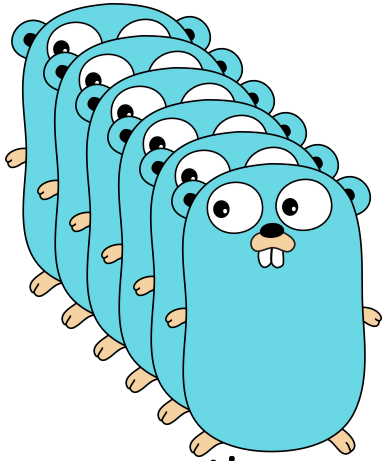
Don't communicate by sharing memory;

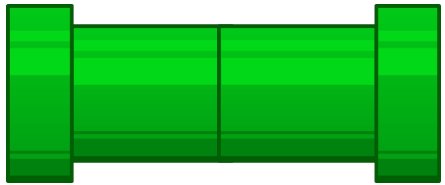
share memory by communicating. (Rob Pike)

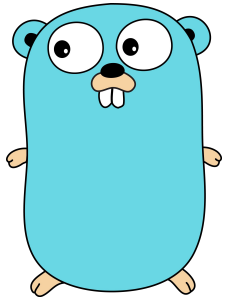
Concurrence / multithreading

Pour illustrer le propos :

⇒ requests ⇒







go routines web

channel avec capacité

go routine counter

Concurrence / multithreading

- On va créer notre première structure (**objet**)

```
// Statistics is the worker to persist the request statistics
type Statistics struct {
        statistics          chan uint8
        counter             uint32
        start               time.Time
        loggingPeriod       time.Duration
}
```

Concurrence / multithreading

- Son **constructeur** associé

```
// NewStatistics creates a new statistics structure and launches its worker routine
func NewStatistics(loggingPeriod time.Duration) *Statistics {
        sw := Statistics{
                statistics:         make(chan uint8, statisticsChannelSize),
                counter:            0,
                start:              time.Now(),
                loggingPeriod:      loggingPeriod,
        }
        go sw.run()
        return &sw
}
```

Concurrence / multithreading

- Une méthode pour la go routine principale d'agrégation

```
func (sw *Statistics) run() {
        ticker := time.NewTicker(sw.loggingPeriod)
        for {
                select {
                case stat := <-sw.statistics:
                        logger.WithField("stat", stat).Debug("new count received")
                        sw.counter += uint32(stat)
                case <-ticker.C:
                        elapsed := time.Since(sw.start)
                        logger.WithField("since", elapsed).WithField("count", sw.counter).Warn("monitoring")
                        // reset
                        sw.counter = 0
                        sw.start = time.Now()
                }
        }
}
```

Concurrence / multithreading - Exercice

/statistics/statistics.go

Tests simples

- Package ***testing*** pour écrire les tests automatisés
- La commande ***go test*** pour exécuter les tests
- Un fichier de test doit se terminer par **_test.go**

```
package main

import "testing"

func TestDummy(t *testing.T) {
   if 4/2 != 2 {
      t.Error("4/2 should be equal to 2")
   }
}
```

```
$ go test

PASS
ok      golang-200/dummy      0.012s
```

Tests simples

- Le test peut être écrit dans le même package que ce que l'on teste ⇒ ***white*** *box tests*
- Ou bien dans le même package suffixé par _test ⇒ ***black*** *box tests*

```
package example

import "testing"


func TestInternalStuff(t *testing.T) {

   // Test non-exported 'abs' function
   if abs(-3) != 3 {
      t.Error("Wrong result")
   }
}
```

```
package example_test

import "testing"


func TestExternalStuff(t *testing.T) {

   // Test exported function
   if example.Abs(-3) != 3 {
      t.Error("Wrong result")
   }
}
```

Tests simples

- Pour tester notre agrégateur de statistique

```
func TestStatistics(t *testing.T) {
	statistics := NewStatistics(2 * statPeriod)

	// other go routine incrementing the counter
	go func() {
		statistics.PlusOne()
	}()

	time.Sleep(statPeriod)

	if statistics.counter != 1 {
		t.Errorf("Wrong count %d ; expected %d", statistics.counter, 1)
	}

	time.Sleep(2 * statPeriod)

	if statistics.counter != 0 {
		t.Errorf("Wrong count %d ; expected %d", statistics.counter, 0)
	}
}
```

Tests simples

- Pour générer le rapport de coverage du code

```
go test -cover -coverprofile=testcover.out
go tool cover -html=testcover.out
```

Tests simples - Exercice

/statistics/statistics_test.go

Si vous appréciez la formation, Envoyez un Tweet !

#sfeirschool #golang
@sfeirstrbg @sfeir
@sebastienfriess

Si vous appréciez la formation, Envoyez un Tweet !

#sfeirschool #golang
@Sfeirlille @sfeir
@sebastienfriess

Si vous appréciez la formation, Envoyez un Tweet !

#sfeirschool #golang
@SfeirLux @sfeir
@ogerardin

git checkout -f step04

Modélisation

Première étape pour la couche d'accès aux données : la modélisation

- Une structure de base définissant ce qu'est une **Task :**

```
// Task is the structure to define a task to be done
type Task struct {
        ID              string      // ID de la Tâche
        Title           string      // Titre
        Description     string      // Description
        Status          int         // Todo, Done, ...
        Priority        int         // High, Medium, Low
        CreationDate    time.Time   // Date de création
        DueDate         time.Time   // Echéance
}
```

Modélisation

Première étape pour la couche d'accès aux données : la modélisation

- Définir des "**énumérés**" **type safe** pour les constantes

```
// TaskPriority is the priority of a task
type TaskPriority int

const (
        // PriorityMinor lower priority
        PriorityMinor TaskPriority = iota
        // PriorityMedium medium priority
        PriorityMedium
        // PriorityHigh high priority
        PriorityHigh
)
```

```
// TaskStatus is the status of a task
type TaskStatus int

const (
        // StatusTodo for incomplete tasks
        StatusTodo TaskStatus = iota
        // StatusInProgress for tasks in progress
        StatusInProgress
        // StatusDone for completed tasks
        StatusDone
)
```

Modélisation

Première étape pour la couche d'accès aux données : la modélisation

- Ajouter des "**annotations**" pour les transformations **BSON** (Mongo) et **JSON** (Web REST)
- Ce qui nous donne la structure finale suivante :

```
// Task is the structure to define a task to be done
type Task struct {
        ID              string          `json:"id,omitempty" bson:"id"`
        Title           string          `json:"title" bson:"title"`
        Description     string          `json:"description" bson:"description"`
        Status          TaskStatus      `json:"status" bson:"status"`
        Priority        TaskPriority    `json:"priority" bson:"priority"`
        CreationDate    time.Time       `json:"creationDate" bson:"creationDate"`
        DueDate         time.Time       `json:"dueDate" bson:"dueDate"`
}
```

💡 Pro tip : Go 1.8 permet le cast de structures identiques ex: `taskDTO = TaskDTO(taskDAO)`

Modélisation

Première étape pour la couche d'accès aux données : la modélisation

- Gestion de l'ID unique par un UUID

```
import (
	"github.com/satori/go.uuid"
)

// NewTask sets a new ID of the Task as a string
func NewTask() *Task {
	return &Task{
		ID:           uuid.NewV4().String(),
		CreationDate: time.Now(),
		Status:       StatusTodo,
		Priority:     PriorityMedium,
	}
}
```

Modélisation

Première étape pour la couche d'accès aux données : la modélisation

- Comparaison avec Equal a cause des dates
- Pour comparer 2 structures en Go toutes ses propriétés doivent être comparables

```
// Equal compares a Task to another
func (t Task) Equal(task Task) bool {
        return t.ID == task.ID &&
                t.Title == task.Title &&
                t.Description == task.Description &&
                t.Status == task.Status &&
                t.Priority == task.Priority &&
                t.CreationDate.Equal(task.CreationDate) &&
                t.DueDate.Equal(task.DueDate)
}
```

Modélisation - Exercice

/model/task.go

git checkout -f step05

Accès aux données

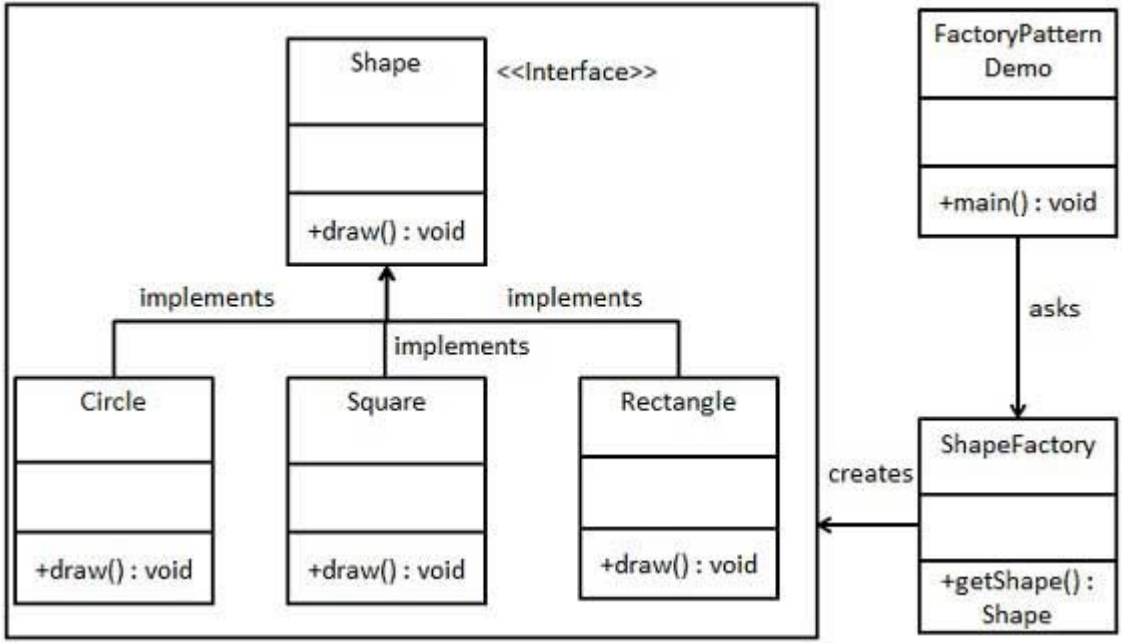
Pour les accès aux données, un bon réflexe est d'utiliser un **pattern Factory (Fabrique pour nos amis canadiens)** qui a pour avantage de :

- Faciliter le **testing** avec des bouchons
- Permettre la **migration** vers un autre type de BDD (ex: SQL vers NoSQL) sans réécriture de toute l'application !

Accès aux données

Le **pattern Fabrique (GoF)** en 3 étapes :

1. Interface DAO
2. Implémentations
3. Fabrique



Accès aux données

1. Interface DAO

```
const (
        // NoPaging used with skip, limit parameters
        NoPaging = -1
)

// TaskDAO is the DAO interface to work with tasks
type TaskDAO interface {
        // GetByID returns a task by its ID
        GetByID(ID string) (*model.Task, error)
        // GetAll returns all tasks with paging capability
        GetAll(start, end int) ([]model.Task, error)
        // GetByTitle returns all tasks by title
        GetByTitle(title string) ([]model.Task, error)
        // GetByStatus returns all tasks by status
        GetByStatus(status model.TaskStatus) ([]model.Task, error)
        // GetByStatusAndPriority returns all tasks by status and priority
        GetByStatusAndPriority(status model.TaskStatus, priority model.TaskPriority) ([]model.Task, error)
        // Save saves the task
        Save(task *model.Task) error
        // Upsert updates or creates a task
        Upsert(task *model.Task) (bool, error)
        // Delete deletes a tasks by its ID
        Delete(ID string) error
}
```

Accès aux données

2. Implémentation : **Mock**

```
// TaskDAOMock is the mocked implementation of the TaskDAO
type TaskDAOMock struct {
	storage map[string]*model.Task
}

// NewTaskDAOMock creates a new TaskDAO with a mocked implementation
func NewTaskDAOMock() TaskDAO {
	daoMock := &TaskDAOMock{
		storage: make(map[string]*model.Task),
	}

	// Adds some fake data
	daoMock.Save(&MockedTask)

	return daoMock
}
```

Accès aux données

2. Implémentation : **Mock**

```
// MockedTask is the task returned by this mocked interface
var MockedTask = model.Task{
        ID:             uuid.NewV4().String(),
        Title:          "Learn Go",
        Description:    "Let's learn the Go programming language.",
        Status:         model.StatusInProgress,
        Priority:       model.PriorityHigh,
        CreationDate:   time.Date(2017, 01, 01, 0, 0, 0, 0, time.UTC),
        DueDate:        time.Date(2017, 03, 23, 0, 0, 0, 0, time.UTC),
}

// Compile time check
var _ TaskDAO = (*TaskDAOMock)(nil)
```

Accès aux données

2. Implémentation : **MongoDB**

```
const (
        collection     = "tasks"
        index          = "id"
)

// TaskDAOMongo is the mongo implementation of the TaskDAO
type TaskDAOMongo struct {
        session *mgo.Session
}

// NewTaskDAOMongo creates a new TaskDAO mongo implementation
func NewTaskDAOMongo(session *mgo.Session) TaskDAO {

        return &TaskDAOMongo{
                session: session,
        }
}
```

Accès aux données

2. Implémentation : **MongoDB**

```
// GetByID returns a task by its ID
func (s *TaskDAOMongo) GetByID(ID string) (*model.Task, error) {

	// check ID
	if _, err := uuid.FromString(ID); err != nil {
		return nil, errors.New("invalid input to UUID")
	}

	// session copy : connection pool
	session := s.session.Copy()
	defer session.Close()

	task := model.Task{}
	c := session.DB("").C(collection)
	err := c.Find(bson.M{"id": ID}).One(&task)
	return &task, err
}
```

Accès aux données

3. Implémentation : **PostgreSQL**

```
// TaskDAOPostgres is the postgres implementation of the TaskDAO
type TaskDAOPostgres struct {
        db *sql.DB
}

// NewTaskDAOPostgres creates a new TaskDAO postgres implementation
func NewTaskDAOPostgres(db *sql.DB) TaskDAO {
        return &TaskDAOPostgres{
                db: db,
        }
}
```

Accès aux données

3. Implémentation : **PostgreSQL**

```
// GetByID returns a task by its ID
func (s *TaskDAOPostgres) GetByID(ID string) (*model.Task, error) {

	// check ID...

	// query db
	rows, err := s.db.Query(`SELECT * FROM todos WHERE uuid=$1`, ID)
	if err != nil {
		return nil, err
	}

	results, err := mapRows(rows)

	if len(results) == 0 {
		return nil, ErrNotFound
	}
	//[...]
}
```

Accès aux données

4. La Fabrique

```
// DBType lists the type of implementation the factory can return
type DBType int

const (
        // DAOMongo is used for Mongo implementation of TaskDAO
        DAOMongo DBType = iota
        // DAOMock is used for mocked implementation of TaskDAO
        DAOMock

        // mongo timeout
        timeout = 5 * time.Second
        // poolSize of mongo connection pool
        poolSize = 35
)

var (
        // ErrorDAONotFound is used for unknown DAO type
        ErrorDAONotFound = errors.New("unknown DAO type")
)
```

Accès aux données

4. La Fabrique

```
// GetTaskDAO returns a TaskDAO according to type and params
func GetTaskDAO(param string, daoType DBType) (TaskDAO, error) {
	switch daoType {
	case DAOMongo:
		// mongo connection
		mgoSession, err := mgo.DialWithTimeout(param, timeout)
		if err != nil {
			return nil, err
		}

		// set 30 sec timeout on session
		mgoSession.SetSyncTimeout(timeout)
		mgoSession.SetSocketTimeout(timeout)
		// set mode
		mgoSession.SetMode(mgo.Monotonic, true)
		mgoSession.SetPoolLimit(poolSize)

		return NewTaskDAOMongo(mgoSession), nil
	case DAOMock:
		return NewTaskDAOMock(), nil
	}
}
```

Accès aux données

4. La Fabrique

```
        case DAOPostgres:
                // postgresql connection
                db, err := sql.Open("postgres", cnxStr)

                // check errors
                if err != nil {
                        return nil, err
                }

                // set max connection in pool
                db.SetMaxOpenConns(poolSize)

                // try to ping host
                if err = db.Ping(); err != nil {
                        return nil, err
                }
                // ...
        }
}
```

Accès aux données

4. La Fabrique

```
// check is db migration is necessary
if len(migrationPath) == 0 {
        return NewTaskDAOPostgres(db), nil
}

//  playing database migration
driver, err := postgres.WithInstance(db, &postgres.Config{})
m, err := migrate.NewWithDatabaseInstance(
        fileScheme+migrationPath,
        "postgres", driver)

if err != nil {
        return nil, err
}

// upgrade database if necessary
err = m.Up()
if err != nil {
        if err != migrate.ErrNoChange {
                return nil, err
        }
}
// ...
```

Accès aux données

Les tests DAO

```
package dao_test

func TestDAOMongo(t *testing.T) {
        // get config
        config := os.Getenv("DB_HOST")

        // build DAO
        daoMongo, err :=
dao.GetTaskDAO(config, dao.DAOMongo)
        if err != nil {
                t.Error(err)
        }

   //...
}
```

```
package dao_test

func TestDAOMock(t *testing.T) {

        daoMock, err := dao.GetTaskDAO("",
dao.DAOMock)
        if err != nil {
                t.Error(err)
        }

   //...
}
```

Accès aux données - Exercice

/dao/task-dao-factory.go

/dao/task-dao-mock.go

/dao/task-dao-mock_test.go

/dao/task-dao-mongo.go

/dao/task-dao-postgres.go

Service web (routeur, middleware et controller)

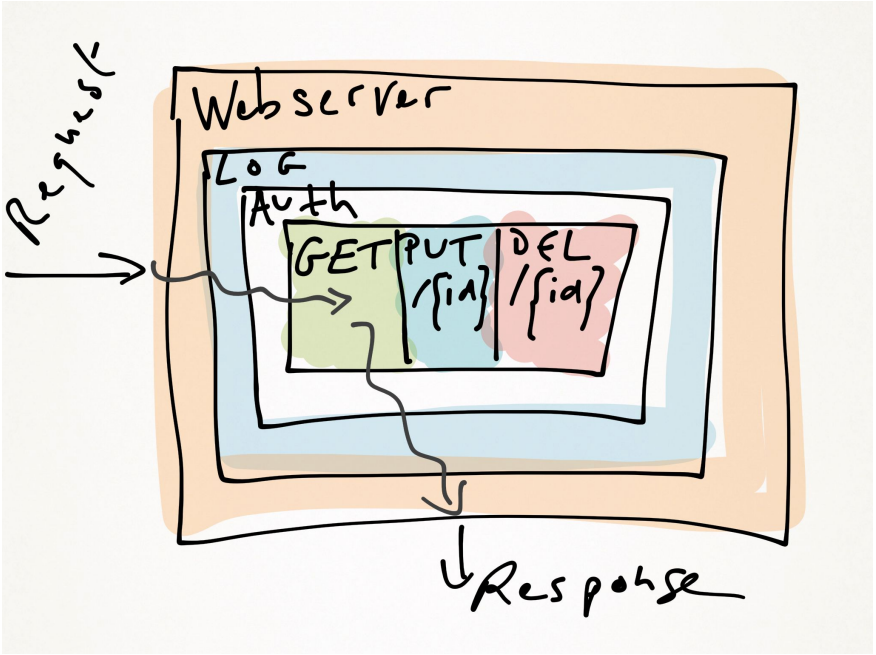
```
git checkout -f step06
```

Service web (routeur, middleware et controller)

Et pour terminer : **Coloriage**

Service web (routeur, middleware et controller)

- **Architecture d'un serveur web**



Service web (routeur, middleware et controller)

- Go arrive avec une bibliothèque web bien fournie : le package **http**
- En **quelques lignes** vous pouvez **monter un serveur web** avec :
  - support HTTP/1 et HTTP/2 (Go 1.6), Push (Go 1.8)
  - HTTPS

```
func main() {
	http.Handle("/", http.FileServer(http.Dir("./")))

	http.HandleFunc("/test", func(w http.ResponseWriter, r *http.Request) {
		w.WriteHeader(http.StatusNotImplemented) // 501
		w.Header().Set("Content-Type", "text/plain; charset=utf-8")
		fmt.Fprintln(w, "Not implemented yet")
	})

	http.ListenAndServe(":8080", nil)
}
```

Service web (routeur, middleware et controller)

- **Handle, HandleFunc : Fonctions** qui travaillent sur une instance de routeur par défaut.

```
// Handle registers the handler for the given pattern
// in the DefaultServeMux.
// The documentation for ServeMux explains how patterns are matched.
func Handle(pattern string, handler Handler) {
	DefaultServeMux.Handle(pattern, handler)
}

// HandleFunc registers the handler function for the given pattern
// in the DefaultServeMux.
// The documentation for ServeMux explains how patterns are matched.
func HandleFunc(pattern string, handler func(ResponseWriter, *Request)) {
	DefaultServeMux.HandleFunc(pattern, handler)
}
```

Service web (routeur, middleware et controller)

- **Handler, HandlerFunc :** Interface et fonctions anonymes qui répondent à des Requêtes.

```
// A Handler responds to an HTTP request.
type Handler interface {
        ServeHTTP(ResponseWriter, *Request)
}

// The HandlerFunc type is an adapter to allow the use of
// ordinary functions as HTTP handlers. If f is a function
// with the appropriate signature, HandlerFunc(f) is a
// Handler that calls f.
type HandlerFunc func(ResponseWriter, *Request)

// ServeHTTP calls f(w, r).
func (f HandlerFunc) ServeHTTP(w ResponseWriter, r *Request) {
        f(w, r)
}
```

Service web (routeur, middleware et controller)

- **Pour faire du web il nous faut :**
  - **Un serveur web**
  - **Un routeur**
  - **Des HandleFunc en face de chaque route**
  - En bonus **des middleware**

- **Pour une utilisation un peu plus fun, on va utiliser les bibliothèques suivantes compatible avec les standards du package http :**
  - https://github.com/urfave/negroni : pour la gestion des middlewares
  - https://github.com/gorilla/mux : un routeur plus “dev friendly”

NB. Il existe aussi **gin** et **martini** mais plus complexes et sans http Go (voire vraiment magiques)

Service web (routeur, middleware et controller)

- **Comment composer des middleware : web/web-server.go**

```
// middleware builder
n := negroni.New()

// add middleware for logging (negroni.Handler)
n.Use(negronilogrus.NewMiddlewareFromLogger(logger.StandardLogger(), "task"))

// add statistics middleware
n.Use(NewStatisticsMiddleware(statisticsDuration))

// add as many middleware as you like

// new router
router := web.NewRouter(web.NewTaskController())

// route handler goes last (http.Handler)
n.UseHandler(router)

// Lance le serveur web
n.Run(":" + strconv.Itoa(port))
```

Service web (routeur, middleware et controller)

- **Negroni Middleware, de quoi parle-t-on ?**
  - Interface **Handler** similaire au Handler http **MAIS** avec un param **next**

```
// Handler handler is an interface that objects can implement to be registered to serve as middleware
// in the Negroni middleware stack.
type Handler interface {
        ServeHTTP(rw http.ResponseWriter, r *http.Request, next http.HandlerFunc)
}

// HandlerFunc is an adapter to allow the use of ordinary functions as Negroni handlers.
type HandlerFunc func(rw http.ResponseWriter, r *http.Request, next http.HandlerFunc)

func (h HandlerFunc) ServeHTTP(rw http.ResponseWriter, r *http.Request, next http.HandlerFunc) {
        h(rw, r, next)
}
```

Service web (routeur, middleware et controller)

- Le middleware de statistique

```
// NewStatisticsMiddleware creates a new statistics middleware
func NewStatisticsMiddleware(duration time.Duration) *StatisticsMiddleware {
        return &StatisticsMiddleware{
                Stat: statistics.NewStatistics(duration),
        }
}


func (sm *StatisticsMiddleware) ServeHTTP(rw http.ResponseWriter, r *http.Request, next http.HandlerFunc) {
        // avant
        sm.Stat.PlusOne()
        next(rw, r)
        // après
}
```

Service web (routeur, middleware et controller)

- Fonctions handler associées aux routes:
  - Allons voir **web/controller.go** et **web/router.go**
  - Déclaration des end points

```
// Get
routes = append(routes, Route{
        Name:        "Get one task",
        Method:      http.MethodGet,
        Pattern:     "/{id}",
        HandlerFunc: controller.Get,
})
```

Service web (routeur, middleware et controller)

- Fonctions handler associées aux routes:
  - Implémentation des HandlerFunc func(ResponseWriter, *Request)
  - pour les utilitaires JSON, Go a ce qu'il faut : voir **web/utils.go**

```
// Get retrieve an entity by id
func (sh *TaskController) Get(w http.ResponseWriter, r *http.Request) {
	// get the task's ID from the URL
	taskID := ParamAsString("id", r)

	// [...]

	logger.WithField("tasks", task).Debug("task found")
	SendJSONOk(w, task)
}
```

Tests web

- Le package **httptest**
- Les tests unitaires avec httptest.ResponseRecorder

```
func TestSomeHandler(t *testing.T) {

	handler := func(w http.ResponseWriter, r *http.Request) {
		io.WriteString(w, "Hello World!")
	}

	req := httptest.NewRequest("GET", "http://example.com/foo", nil)
	w := httptest.NewRecorder()
	handler(w, req)

	resp := w.Result()
	fmt.Println(resp.StatusCode)

	//...
}
```

Tests web

- Les tests end-to-end avec httptest.Server

```
func TestEndToEnd(t *testing.T) {

	ts := httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
		fmt.Fprintln(w, "Hello, client")
	}))
	defer ts.Close()

	res, err := http.Get(ts.URL)
	if err != nil {
		t.Error(err)
	}

	//...
}
```

Subtests

- Les subtests pour mieux **organiser** et **regrouper** les tests

```
func TestFoo(t *testing.T) {
   //
   t.Run("A=1", func(t *testing.T) { ... })
   t.Run("A=2", func(t *testing.T) { ... })
   t.Run("B=1", func(t *testing.T) { ... })
   //
}
```

```
go test -run Foo        # Run top-level tests matching "Foo".
go test -run Foo/A=     # Run subtests of Foo matching "A=".
go test -run /A=1       # Run all subtests of a top-level test matching "A=1".
```

Subtests

- Les subtests peuvent être exécutés en **parallèle**

```
func TestFoo(t *testing.T) {

   t.Run("A=1", func(t *testing.T) {
       t.Parallel()
       ...
   })

   t.Run("A=2", func(t *testing.T) {
       t.Parallel()
       ...
   })

}
```

Exercice

/web/controller.go

/web/statistics-middleware.go

/web/web-server.go

/web/web_test.go

Benchmarks

- Le package **testing** fournit tout ce qu'il faut pour créer des **benchmarks**
- Une fonction de benchmark :
  - commence par **Benchmark**
  - prend en paramètre le nombre d'exécutions à effectuer **b.N**
  - peut être exécutée plusieurs fois avec des valeurs différentes de **b.N**

```
func BenchmarkFib10(b *testing.B) {

    // run the Fib function b.N times
    for n := 0; n < b.N; n++ {
        Fib(10)
    }

}
```

```
$ go test -bench=.

PASS
BenchmarkFib10    5000000               509 ns/op
ok      github.com/Sfeir/fib    3.084s
```

Benchmarks

- **A ne jamais faire** dans un benchmark :
  - changer les paramètres entre deux itérations

```
func BenchmarkFibWrong(b *testing.B) {
	for n := 0; n < b.N; n++ {
		Fib(n)
	}
}
```

  - utiliser le nombre de “runs” comme paramètre

```
func BenchmarkFibWrong2(b *testing.B) {
	Fib(b.N)
}
```

pprof

- Outil de **profiling** *built-in*

```
$ go test -v -bench=. -memprofile=prof.mem
BenchmarkTaskControllerGet-8            200000              5988 ns/op
BenchmarkTaskControllerPost-8           100000             15374 ns/op
BenchmarkHugeMemoryAllocation-8          30000             55522 ns/op
PASS

$ go tool pprof --alloc_space prof.mem
Entering interactive mode (type "help" for commands)
(pprof) top 3
13.81GB of 14.27GB total (96.77%)
Dropped 38 nodes (cum <= 0.07GB)
Showing top 3 nodes out of 29 (cum >= 0.07GB)
      flat  flat%   sum%        cum   cum%
   13.41GB 93.98% 93.98%    13.41GB 93.98%  github.com/Sfeir/golang-200/web.BenchmarkHugeMemoryAllocation
    0.12GB  0.87% 94.85%     0.12GB  0.87%  net/http/httptest.cloneHeader
    0.11GB  0.77% 95.62%     0.11GB  0.77%  net/textproto.MIMEHeader.Set
```

http.pprof

- **Expose** les informations de **profiling** compatibles **pprof**, via une **API HTTP**.
- Il suffit d'importer le package dans son programme : `import _ "net/http/pprof"`
- `$ go tool pprof http://localhost:8080/debug/pprof/heap`
- flame graph disponible également

Container et Run

- **On lance le tout**

  $> make dockerBuildUp(Multi)

  $> make dockerLogs

- **On test**

  $> cd etc

  $> ./apiquery.sh -create

  ou via

[sf≡ir]

Well Done !

👏💪

Ressources

- Site Officiel : https://golang.org
- Tour of Go : https://tour.golang.org/welcome/1
- Vendoring : https://github.com/golang/go/wiki/PackageManagementTools
- Site francophone sur le Go : frenchgo.fr
- livre blanc Comprendre Go de SFEIR

Crédit photo

- http://bakingrecipie.blogspot.fr/2012/04/massive-cherry-cupcake.html
- https://commons.wikimedia.org/wiki/File:Dave_Cheney_at_Golang_UK_2016-2.jpg
- http://i2.wp.com/www.business-angel-france.com/wp-content/uploads/2013/04/kiss.jpg
- https://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Liens

- https://dave.cheney.net/2015/11/05/lets-talk-about-logging