THE DEVELOPER'S CONFERENCE

Clean architecture em Go

Elton Minetto

CTO da Code:Nation - Advisor na Coderockr

O que é Clean Architecture?

→ Independente de frameworks

→ Testável

→ Independente de UI

→ Independente de Database

→ Independente qualquer agente externo

Divide nosso código em 4 camadas

→ **Entities**: representam as entidades das regras de negócio

→ **Use Cases**: as regras de negócio da aplicação

→ **Controller**: adaptam e convertem os dados do formato usado pelas entidades e use cases para agentes externos como bancos de dados , web, etc

→ **Framework & Driver**: frameworks e ferramentas como bancos de dados, frameworks web, etc

Clean architecture em Go

```
ls -lh pkg/entity
-rw-r--r--  1 eminetto  staff   553B Apr 17 16:39 address.go
-rw-r--r--  1 eminetto  staff   1.2K Apr 17 15:51 entity.go
-rw-r--r--  1 eminetto  staff   2.9K Apr 17 15:52 user.go
```

No pacote *entity* estão nossas entidades

```go
package entity

//User data
type User struct {
    ID                ID          `json:"id"`
    Picture           string      `json:"picture"`
    Email             string      `json:"email"`
    Password          string      `json:"password"`
    ValidationHash    string      `json:"validation_hash"`
    ChangePasswordHash string     `json:"change_password_hash"`
    FirstName         string      `json:"first_name"`
    LastName          string      `json:"last_name"`
}
```

```
ls -lh pkg/user
-rw-r--r--  1 eminetto  staff   864B Apr 17 21:03 interface.go
-rw-r--r--  1 eminetto  staff   1.4K Apr 17 16:16 repository_inmem.go
-rw-r--r--  1 eminetto  staff   2.5K Apr 17 16:19 repository_mongodb.go
-rw-r--r--  1 eminetto  staff   3.2K Apr 17 21:03 service.go
-rw-r--r--  1 eminetto  staff   3.1K Apr 17 16:15 service_test.go
```

No arquivo *interface.go* temos a definição das interfaces dos *Use Case* e repositório, onde a entidade vai ser armazenada

```go
package user

import "github.com/thecodenation/highway/pkg/entity"

type Reader interface {
    Find(id entity.ID) (*entity.User, error)
    FindByEmail(email string) (*entity.User, error)
    FindAll() ([]*entity.User, error)
}

type Writer interface {
    Update(user *entity.User) error
    Store(user *entity.User) (entity.ID, error)
}

type Repository interface {
    Reader
    Writer
}

type UseCase interface {
    ForgotPassword(user *entity.User) error
    ChangePassword(user *entity.User, password string) error
    Auth(user *entity.User, password string) error
    IsValid(user *entity.User) bool
    Reader
    Writer
}
```

O *service.go* é a implementação dos *Use Case*

```go
package user

import (
    "github.com/thecodenation/highway/pkg/entity"
    "golang.org/x/crypto/bcrypt"
)

//Service service implementation
type Service struct {
    repo Repository
}

//NewService create new service
func NewService(r Repository) *Service {
    return &Service{
        repo: r,
    }
}

func (s *Service) Auth(user *entity.User, password string) error {
    return bcrypt.CompareHashAndPassword([]byte(user.Password), []byte(password))
}

func (s *Service) Find(id entity.ID) (*entity.User, error) {
    return s.repo.Find(id)
}
//restante do arquivo oculto para facilitar a leitura do slide ;)
```

Nos arquivos *repository_inmem.go* e *repository_mongodb.go* temos implementações da interface que define um repositório, onde as entidades são armazenadas. Neste caso o repositório representa parte da camada *Framework & Driver*

```go
package user
import (
    "os"
    "github.com/juju/mgosession"
    "github.com/thecodenation/highway/pkg/entity"
    mgo "gopkg.in/mgo.v2"
    "gopkg.in/mgo.v2/bson"
)
type MongoRepository struct {
    pool *mgosession.Pool
}
//NewMongoRepository create new repository
func NewMongoRepository(p *mgosession.Pool) *MongoRepository {
    return &MongoRepository{
        pool: p,
    }
}
func (r *MongoRepository) Find(id entity.ID) (*entity.User, error
) {
    result := entity.User{}
    session := r.pool.Session(nil)
    coll := session.DB(os.Getenv("MONGODB_DATABASE")).C("user")
    err := coll.Find(bson.M{"_id": id}).One(&result)
    if err != nil {
        return nil, err
    }
    return &result, nil
}
//restante do arquivo oculto para facilitar a leitura do slide ;)
```

No diretório *api* temos a implementação da camada *Controller* e também de mais uma parte da camada *Framework & Driver*, nos *handlers*

```
cd api ; tree
.
|____handler
| |____user.go
| |____user_test.go
|____doc
| |____api.apib
| |____index.html
|____main.go
```

No trecho a seguir, do *api/main.go* podemos ver como usar o serviço

```
session, err := mgo.Dial(os.Getenv("MONGODB_HOST"))
if err != nil {
    elog.Error(err, elog.ERROR)
}
defer session.Close()
queueService, err := queue.NewAWSService()
if err != nil {
    elog.Error(err, elog.ERROR)
}

mPool := mgosession.NewPool(nil, session, 10)
defer mPool.Close()

userRepo := user.NewMongoRepository(mPool)
userService := user.NewService(userRepo)
```

Podemos facilmente testar nossos pacotes, camada a camada

pkg/user/service_test.go

```go
package user

import (
    "testing"

    "github.com/stretchr/testify/assert"
    "github.com/thecodenation/highway/pkg/entity"
)

func TestIsValidUser(t *testing.T) {
    u := entity.User{
        ID:        entity.NewID(),
        FirstName: "Bill",
        LastName:  "Gates",
    }
    userRepo := NewInmemRepository()
    userService := NewService(userRepo)
    assert.False(t, userService.IsValid(&u))

    u.ValidatedAt = time.Now()
    assert.True(t, userService.IsValid(&u))
}
//restante do arquivo oculto para facilitar a leitura do slide ;)
```

api/handler/user_test.go

```go
package handler

import (
    "encoding/json"
    "fmt"
    "net/http"
    "net/http/httptest"
    "strings"
    "testing"
    "github.com/stretchr/testify/assert"
    "github.com/thecodenation/highway/pkg/entity"
    "github.com/thecodenation/highway/pkg/queue"
    "github.com/thecodenation/highway/pkg/user"
)

func TestUserRegister(t *testing.T) {
    userService := user.NewService(NewInmemRepository())
    h := userAdd(userService, queue.NewInmemService())
    ts := httptest.NewServer(h)
    defer ts.Close()
    payload := fmt.Sprintf(`{
        "email": "ozzy@metal.net",
        "current_role": "God of Rock",
        "nickname": "ozzy"
        }`)
    resp, _ := http.Post(ts.URL+"/v1/user", "application/json", strings.NewReader(payload))
    assert.Equal(t, http.StatusCreated, resp.StatusCode)
    var u *entity.User
    json.NewDecoder(resp.Body).Decode(&u)
    assert.True(t, entity.IsValidID(u.ID.String()))
}
```

Perguntas

http://eltonminetto.net

@eminetto

http://asemanago.com.br