ACTION FLOW

Um framework que faz uso de
Event Driven e Go Routines

GUILHERME ESTEVES

TECH LEAD @ QUEROQUITAR

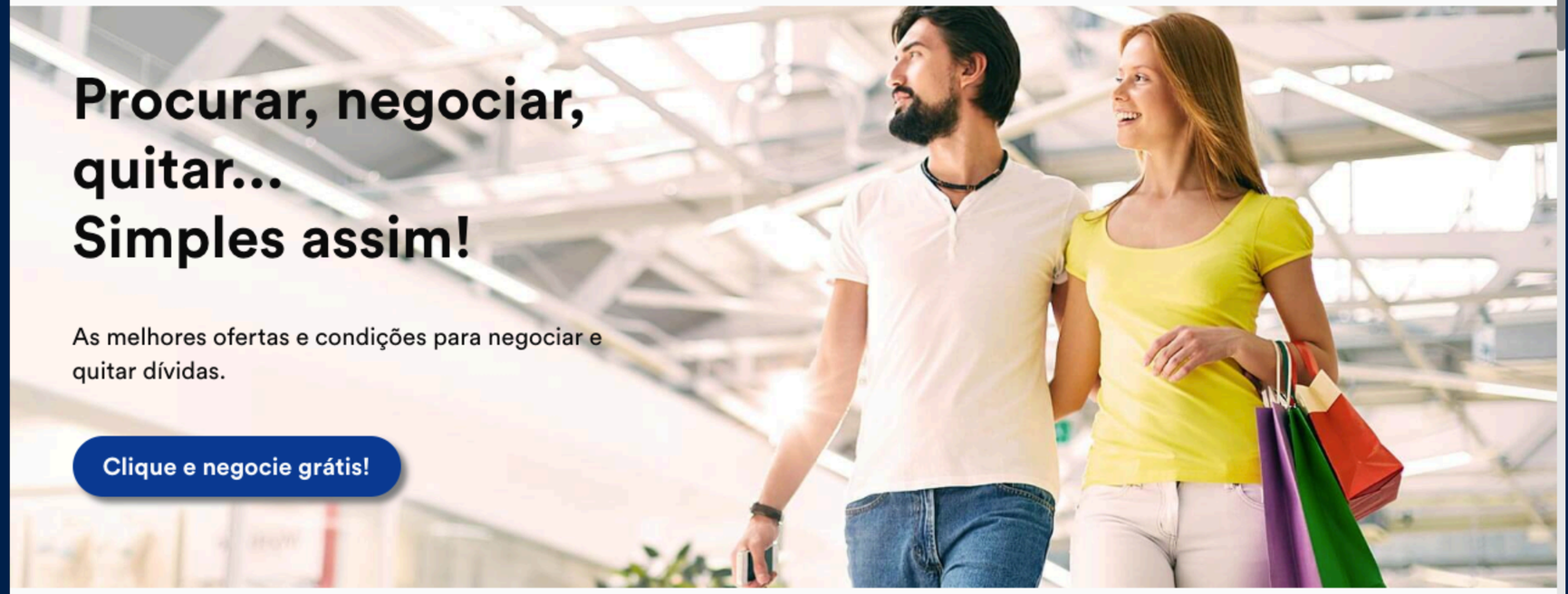
@GUILHERMESTEVES

OLA@GUILHERMESTEVES.DEV

GUILHERMESTEVES.DEV

Quero Quitar!

Início | Quem somos | Como funciona | Blog | Atendimento | Entrar | Cadastrar



Quem somos

Recuperar sonhos, este é o nosso trabalho!

Estamos contrando

PESSOAS INCRÍVEIS

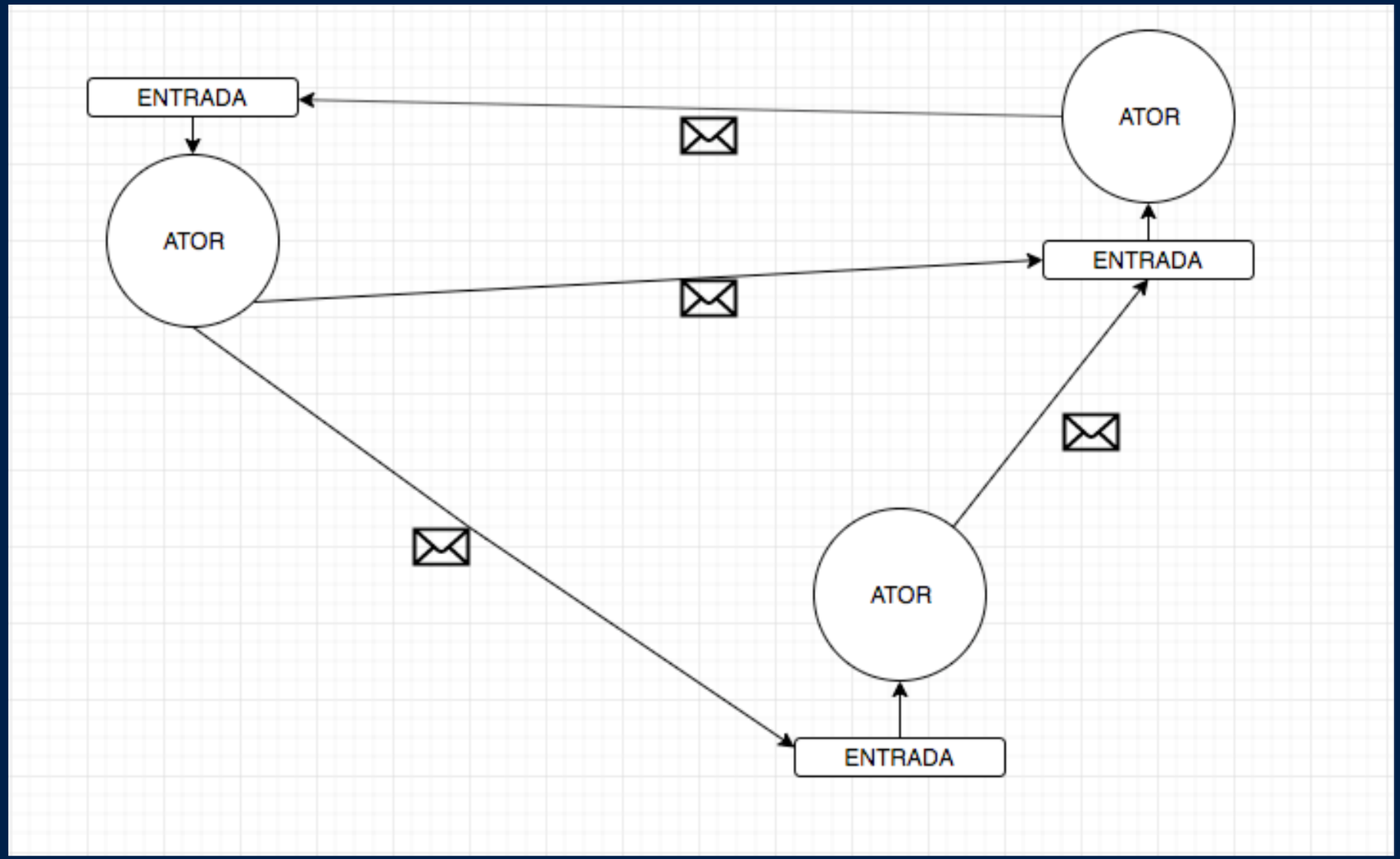
FULL-STACK DEVELOPER

GO & VUE


QUEM CRIOU O ACLOW?
ALEXANDRE MARCO
github.com/AlexMarco7

ATORES

ACTOR MODEL



PROPOSTA

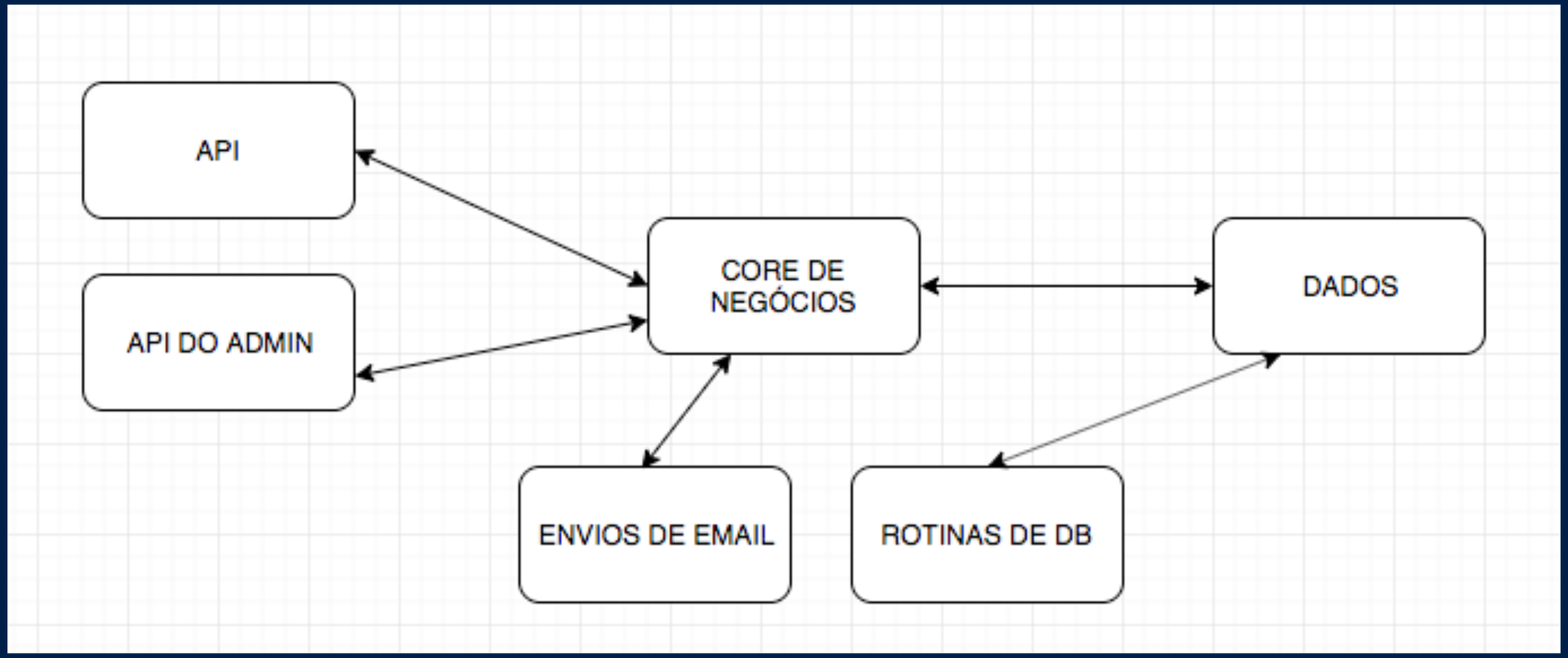
- TODA SOLICITAÇÃO DE UM ATOR DEVE SER UM FLUXO COMPOSTO DE AÇÕES
- AS STRUCTS DEVEM SER ATÔMICAS NO SEU NÍVEL DE ABSTRAÇÃO E FUNCIONALIDADE
- OS NOMES DAS STRUCTS (FLUXOS & AÇÕES) PRECISAM SER SEMÂNTICOS
- A COMUNICAÇÃO ENTRE FLUXOS E AÇÕES PRECISAM SER FEITOS ATRAVÉS DE EVENTOS

ARQUITETURA

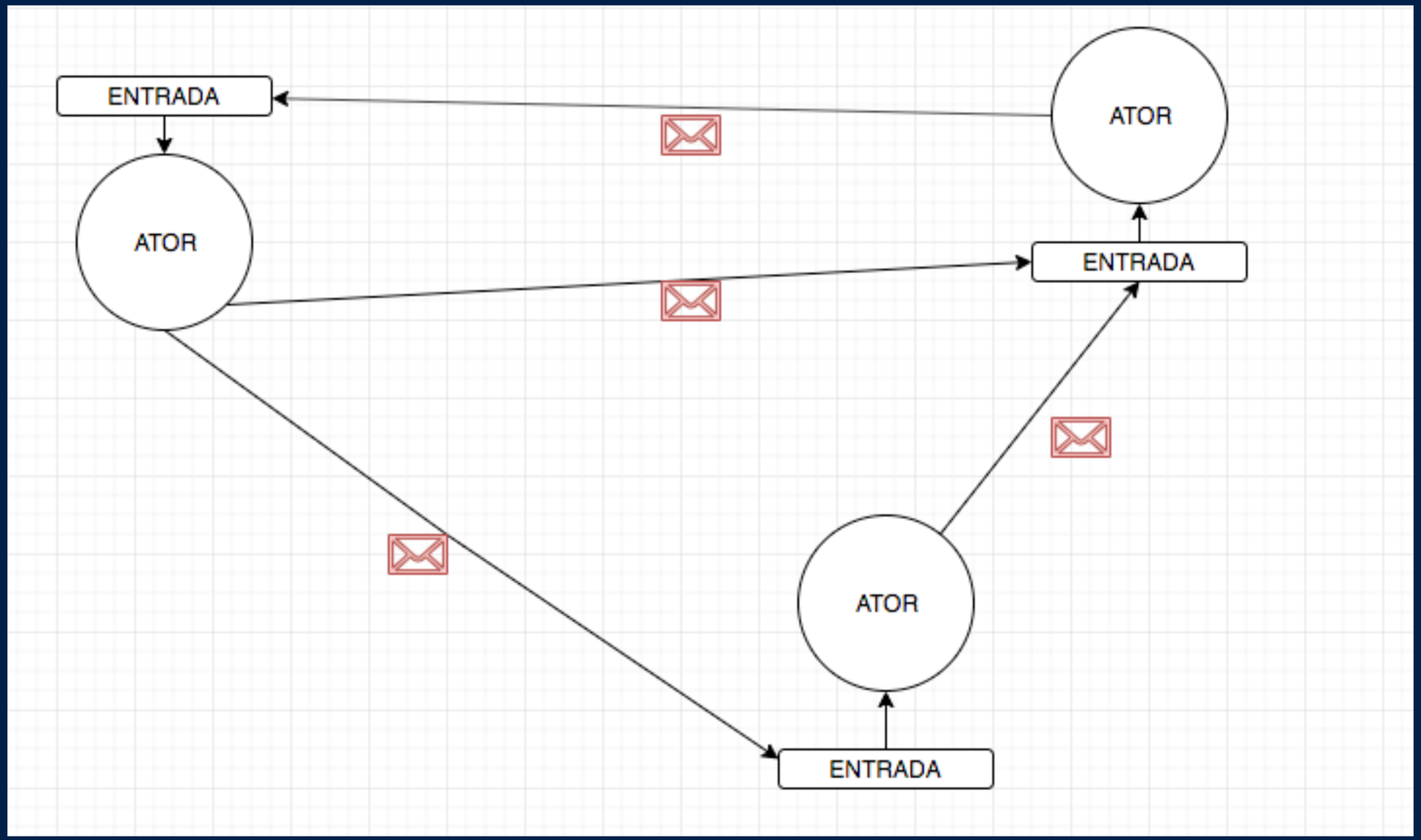
- MÓDULOS
- NÓS
- MENSAGEM

MÓDULOS

MÓDULOS


API
API DO ADMIN
CORE DE NEGÓCIOS
DADOS
ENVIOS DE EMAIL
ROTINAS DE DB

MENSAGENS



MENSAGENS

```
type Message struct {
        Header map[string]interface{}
        Body   interface{}
}
```

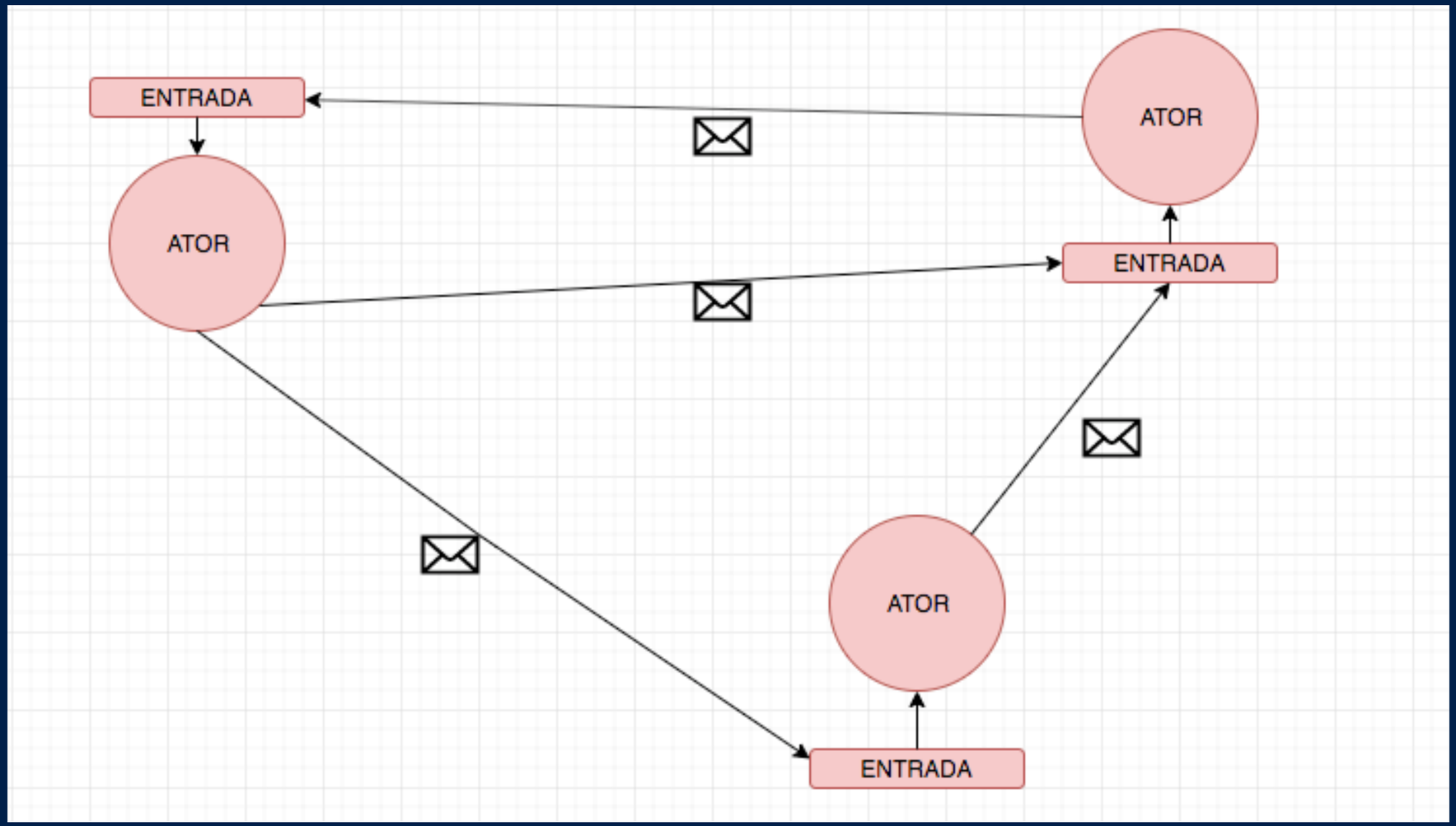
MENSAGENS

```
type ReplyMessage struct {
        Message
        Err error
}
```

MENSAGENS

```
func (n *CreatingTodo) Execute(msg aclow.Message, call aclow.Caller) (aclow.Message, error) {
    body := msg.Body.(message.CreatingToDoResquest)

    reply, _ := call( a: "data@create_todo", aclow.Message{Body: body.Todo.Name})

    if reply.Body == nil {
        return aclow.Message{}, fmt.Errorf( format: "ERROR")
    }

    todo := reply.Body.(model.ToDo)

    return aclow.Message{Body: message.CreatingToDoResponse{
        Todo: &todo,
    }}, nil
}
```

NÓS



NÓS

- **ENDEREÇO, INÍCIO & EXECUÇÃO**
- **CALL & PUBLISH**

NÓS

```
type Node interface {
        Address() []string
        Start(app *App)
        Execute(msg Message, call Caller) (Message, error)
}
```

NÓS

ENDEREÇO

```
func (n *CreatingTodo) Address() []string { return []string{"creating_todo"} }
```

NÓS

INÍCIO

```
func (n *CreatingTodo) Start(app *aclow.App) {
    n.app = app
}
```

NÓS

INÍCIO

```
func (n *SchedullingHello) Start(app *aclow.App) {
    n.app = app

    go func() {
        time.Sleep(time.Second * 20)
        exec := func() {
            app.Publish( address: "core@schedulling_hello", aclow.Message{})
        }
        c := cron.New()
        c.AddFunc( spec: "@every 24h", func() {
            exec()
        })
        go exec()
        c.Start()
    }()
}
```

NÓS

EXECUTE

```go
func (n *CreateTodo) Execute(msg aclow.Message, call aclow.Caller) (aclow.Message, error) {
    client := n.app.Resources["mongo"].(*mongo.Client)
    db := client.Database(config.MongoDbDatabase())

    name := msg.Body.(string)

    todo := model.ToDo{}
    todo.Id = uuid.New().String()
    todo.Name = name
    todo.CreatedAt = time.Now().Format(util.DefaultTimeMask)
    todo.UpdatedAt = time.Now().Format(util.DefaultTimeMask)

    ctx, _ := context.WithTimeout(context.Background(), time.Minute)
    _, err := db.Collection( name: "col_todo").InsertOne(ctx, todo)

    if err != nil {
        return aclow.Message{}, err
    }

    return aclow.Message{Body: todo}, nil
}
```

CALL

```go
func (n *CreatingTodo) Execute(msg aclow.Message, call aclow.Caller) (aclow.Message, error) {
    body := msg.Body.(message.CreatingToDoResquest)

    reply, _ := call( a: "data@create_todo", aclow.Message{Body: body.Todo.Name})

    if reply.Body == nil {
        return aclow.Message{}, fmt.Errorf( format: "ERROR")
    }

    todo := reply.Body.(model.ToDo)

    return aclow.Message{Body: message.CreatingToDoResponse{
        Todo: &todo,
    }}, nil
}
```

PUBLISH

```
func (n *SchedullingHello) Start(app *aclow.App) {
    n.app = app

    go func() {
        time.Sleep(time.Second * 20)
        exec := func() {
            app.Publish( address: "core@schedulling_hello", aclow.Message{})
        }
        c := cron.New()
        c.AddFunc( spec: "@every 24h", func() {
            exec()
        })
        go exec()
        c.Start()
    }()
}
```

NÓS

- SEMPRE PERTENCEM A UM MÓDULO
- É NECESSÁRIO REGISTRAR OS NÓS

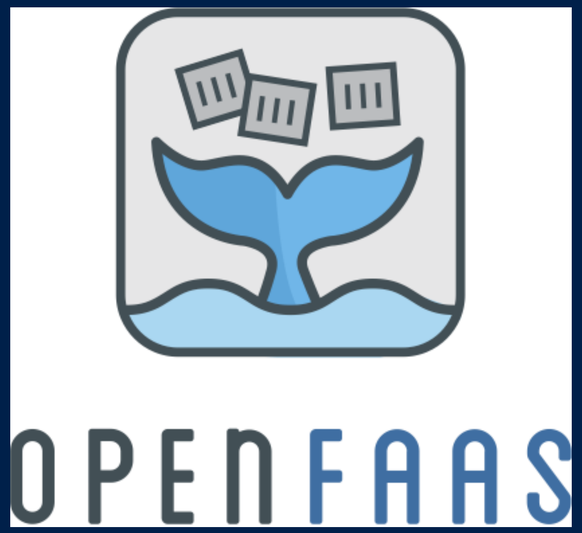
```
func Nodes() []aclow.Node {
    return []aclow.Node{
        &flow.ListingTodo{},
        &flow.CreatingTodo{},
        &flow.LoadingTodo{},
        &flow.UpdatingTodo{},
        &flow.DeletingTodo{},

        &flow.SchedullingHello{},
    }
}
```

MÓDULOS

```go
func main() {
    godotenv.Load()

    startOpt := aclow.StartOptions{
        Local: true,
        Debug: false,
    }

    var app = &aclow.App{}

    app.Start(startOpt)

    router := routing.New()
    app.Resources["router"] = router
    connectOnMongo(app)

    app.RegisterModule( moduleName: "data", data.Nodes())
    app.RegisterModule( moduleName: "core", core.Nodes())

    api.RegisterRoutes(app)

    time.Sleep(time.Second * 2)

    httpServer(app, router)

    app.Wait()

}
```

CONCEITOS FAMILIARES

ONDE EU JÁ VI TUDO ISSO?



NADA MAIS QUE UM

PATTERN MINIMALISTA

FLUXOS

AÇÕES

FUNÇÕES

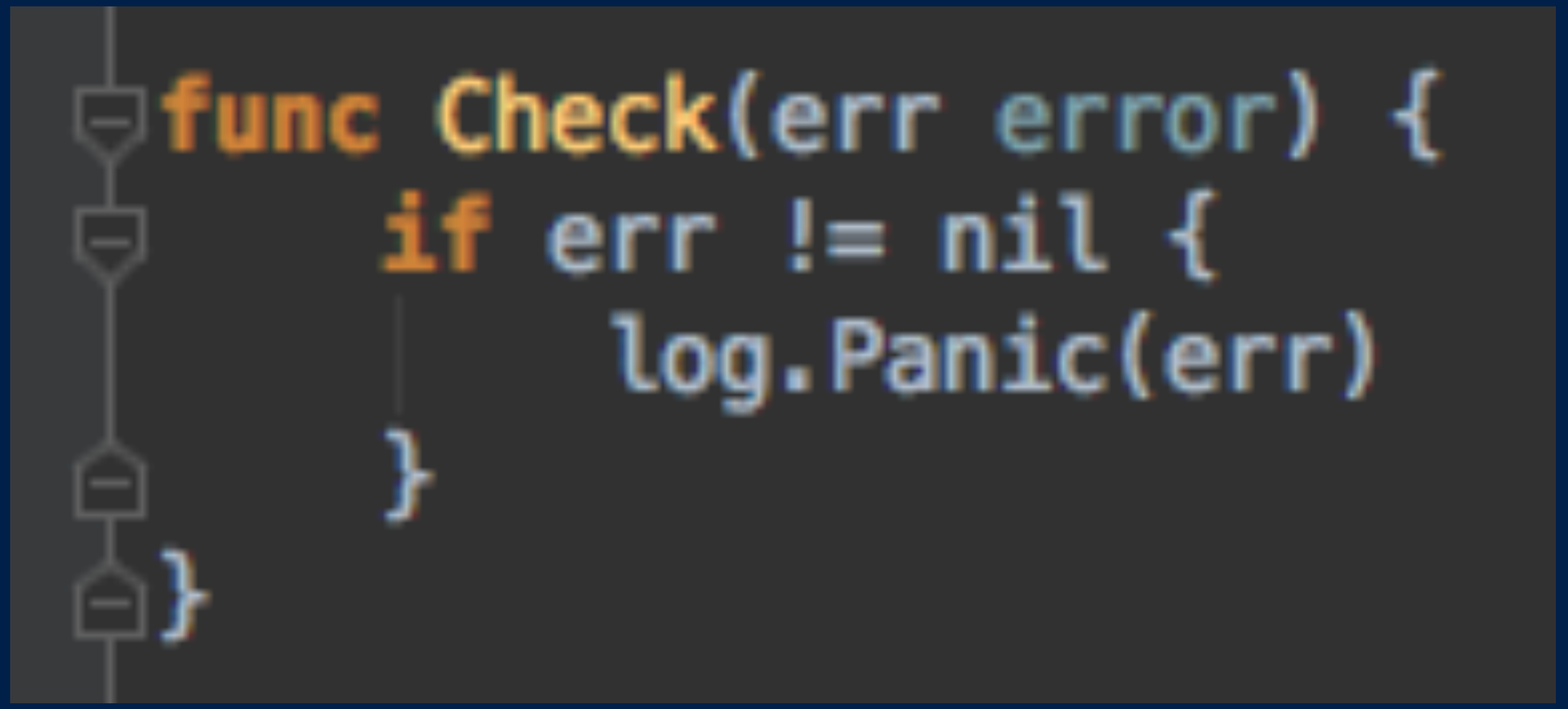
FLUXOS

- FLUXO É UMA SEQUÊNCIA DE UMA OU MAIS AÇÕES
- MONITORA UM ÚNICO EVENTO
- NOME: negócio@criando_usuario (SEMPRE NO GERUNDIO)
- EXEMPLO:
  call("negócio@criando_usuario", aclow.Message{})

AÇÕES

- AÇÃO É UMA ESPECIALIZAÇÃO DE UMA FUNCIONALIDADE
- MONITORA UM ÚNICO EVENTO
- NOME: dados@criar_usuario (SEMPRE NO IMPERATIVO)
- EXEMPLO:
  call("dados@criar_usuario", aclow.Message{})

FUNÇÕES



REVISANDO CONCEITOS

- NOME DE FLUXO É SEMPRE NO GERUNDIO
- NOME DE AÇÃO É SEMPRE NO IMPERATIVO

- MONITORAM UM ÚNICO EVENTO

- UM FLUXO SEMPRE UMA OU MAIS AÇÕES OU UM SUB FLUXO, QUANDO HÁ MUITAS REGRAS

- AÇÕES NÃO CHAMAM AÇÕES NUNCA
- FUNÇÕES PODEM SER CHAMADAS POR AÇÕES OU FLUXOS E VISAM APENAS AUXILIAR


TALK IS CHEAP SHOW ME THE CODE

AGRADECIMENTOS

Project Layout Standards
https://github.com/golang-standards/project-layout/

Alexandre Marco
github.com/AlexMarco7 | linkedin.com/in/alexandre-marco-05668555

Daniel Setton
github.com/dsetton | linkedin.com/in/dsetton

STARS

github.com/AlexMarco7/aclow

github.com/guilhermesteves/go-todo-api


This is it. This is how it ends.

OBRIGADO

@GUILHERMESTEVES