# Challenge to Advanced API Architecture in Go

Seiji Takahashi (@\_\_timakin\_\_) / Gunosy Inc. September 29th, 2017 golang.tokyo#9

### About me

- Seiji Takahashi
- Github: timakin / Twitter: @\_\_timakin\_\_
- Gunosy Inc. Business Development Team
  - Go / Swift
- Just a little bit contributed to Go.

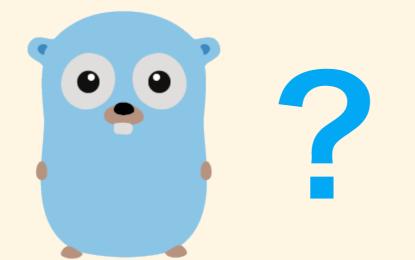


#### Preface

- Finding the sample projects of API server, based on maintainable and feature-rich Go code is so **hard**.
- So I've tried to write an operable API server with plain and standard packages like net/http on myself.
- This is just the result of my best-effort challenge, and not the collective opinion of Go community.

### Agenda

- General problems when you write API server in Go.
- Advanced API architecture in Go, which is adaptable for your production environment.
- The introductions of simple & practical packages you can use in your team tomorrow.



• Use Frameworks?

- Use Frameworks?
  - echo / gin / goji / goa

- Use Frameworks?
  - echo / gin / goji / goa
  - net/http

- Use Frameworks?
  - echo / gin / goji / goa
  - net/http
- Use ORMs?

- Use Frameworks?
  - echo / gin / goji / goa
  - net/http
- Use ORMs?
  - gorm / xorm / gorp / dbr

- Use Frameworks?
  - echo / gin / goji / goa
  - net/http
- Use ORMs?
  - gorm / xorm / gorp / dbr
  - database/sql

- Use Frameworks?
  - echo / gin / goji / goa
  - net/http
- Use ORMs?
  - gorm / xorm / gorp / dbr
  - database/sql
- Which platform?

- Use Frameworks?
  - echo / gin / goji / goa
  - net/http
- Use ORMs?
  - gorm / xorm / gorp / dbr
  - database/sql
- Which platform?
  - AWS / GCP

• It's easy to encounter circular import.

- It's easy to encounter circular import.
- context.Context handling

- It's easy to encounter circular import.
- context.Context handling
- error handling

- It's easy to encounter circular import.
- context.Context handling
- error handling
- Passing middleware objects without context.Context pollution

- It's easy to encounter circular import.
- context.Context handling
- error handling
- Passing middleware objects without context.Context pollution
- Mature Go hackers say "your shouldn't use a framework. Just use net/http.", however, it sounds there are too much stuff to do.

### Any good sample?

marcusolsson/goddd

- <u>marcusolsson/goddd</u>
- Well-capsuled repository with Interface

- marcusolsson/goddd
- Well-capsuled repository with Interface
- encoder / decoder for req / res payload.

- marcusolsson/goddd
- Well-capsuled repository with Interface
- encoder / decoder for req / res payload.
- DDD-based architecture enables you to easily avoid a circular import.

#### How to write API inspired by goddd with plain packages?

### Background

- I wrote the sample project: "govod".
  - Sorry, it's a closed project because it includes some secrets. <a href="mailto:spin">\$\overline\$\$</a>
- Go video on demand API.
- Deploy to Google App Engine
- Features (≒ domains)
  - Authentication
  - Streaming

ex)/api/videos with paging interface

- ├── Gopkg.lock
- Gopkg.toml
- Makefile
- README.md
- REFERENCE.md

#### – app

- ∣ ⊣ dev.yaml
  - ⊣ index.yaml
- ∣ ⊣ main.go
  - └── prd.yaml
- ⊢ circle.yml

#### — src

- ⊣ config
- ⊣ domain
- ⊢ handler
- ⊢ middleware
- repository
- └── vendor
  - ⊢ github.com
  - ⊢ golang.org
  - ⊢ google.golang.org
  - 🖵 gopkg.in

# **Directory tree**

#### Isolate main.go to app directory with app.yaml (GAE config) to avoid the go-app-builder error.

- Gopkg.lock
- Gopkg.toml
- Makefile
- README.md
- REFERENCE.md

#### app

- ├── dev.yaml
- ⊢ index.yaml
- ⊢ main.go
- └── prd.yaml
- circle.yml
- src
  - ⊢ config
  - domain
  - ⊢ handler

  - repository
- └── vendor
  - github.com
  - ⊢ golang.org
  - ⊢ google.golang.org
  - gopkg.in

#### `domain` and `repository` are main directories that have the business logics and data accessors.

<u>middleware</u> `middleware` has the http.Handler implementations.

## **Directory tree**

#### streaming

- decode.go
- dependency.go
- encode.go
- errors.go
- handler.go
- linterface.go Business Logic
  - pager.go
- routing.go
  - service.go

# Inside of domain

- Payload en/decoder
- Error types
- DI object
- Interface of Repository
- Routing
- Paging token parset
- etc...

# func init() { h := initHandlers() http.Handle("/", h)

#### Initializes the router with `gorilla/mux`

# // Routing r := mux.NewRouter()

# Combine middlewares for the simple declaration of http.Handler with `justinas/alice`.

#### You can get stats with kicking the endpoint "/api/stats" (`fukata/golang-stats-api-handler`)

#### // middleware chain

chain := alice.New(
 middleware.AccessControl,
 middleware.Authenticator,

// cpu, memory, gc, etc stats
r.HandleFunc("/api/stats", stats\_api.Handler)

#### Initialize repository, service (business logic), and dependency injector for HTTP Handler.

#### // Streaming Service

streamingRepository := repository.NewStreamingRepository()
streamingService := streaming.NewService(streamingRepository)
streamingDependency := &streaming.Dependency{
 StreamingService: streamingService,

#### }

#### // Authentication Service

```
authRepository := repository.NewAuthRepository()
authService := auth.NewService(authRepository)
authDependency := &auth.Dependency{
    AuthService: authService,
}
```

#### r = streaming.MakeCategoryHandler(streamingDependency, r)

- r = streaming.MakeTopicHandler(streamingDependency, r)
- r = streaming.MakeVideoHandler(streamingDependency, r)
- r = auth.MakeInitHandler(authDependency, r)

## // Bind middlewares h := chain.Then(r)

#### return h

Declare the dependency injected to the custom handler. (If it's not GAE `Dependency` may have `Logger`, or other middlewares...)

Handler with DI enables you to handle middleware without setting them in context.Context.

```
type Dependency struct {
   StreamingService Service
}
type CustomHandler struct {
   Impl func(http.ResponseWriter, *http.Request)
}
func (h CustomHandler) ServeHTTP(w http.ResponseWriter, r *http.Request) {
   vars := mux.Vars(r)
}
```

```
vars := mux.Vars(r)
ctx := appengine.WithContext(r.Context(), r)
ctx, cancel := context.WithTimeout(ctx, handler.TimeOutLimit)
defer cancel()
ctx = handler.SetReqParams(ctx, vars)
cr := r.WithContext(ctx)
h.Impl(w, cr)
```

#### **Registration of the routings with CustomHandler and DI object.**

// MakeVideoHandler ... register handlers for video resources
func MakeVideoHandler(d \*Dependency, r \*mux.Router) \*mux.Router {
 getVideoCollectionHandler := CustomHandler{Impl: d.GetVideoCollectionHandler}
 r.Handle("/api/videos", getVideoCollectionHandler).Methods("GET")
 return r

Request handler with payload decoder/encoder.

Decoder contains the validator. (`go-playground/validator.v9`)

Response payload will be wrapped with `unrolled/render`.

It may results to return the error object given a detail context with `pkg/errors`

```
func (d *Dependency) GetVideoCollectionHandler(w http.ResponseWriter, r *http.Request) {
    reqPayload, err := decodeGetVideoCollectionRequest(r)
    if err != nil {
        res := handler.NewErrorResponse(http.StatusBadRequest, err.Error())
        handler.Redererer.JSON(w, res.Status, res)
        return
    }
```

# Paging token parser which returns pager cursor required by Datastore.

```
var cursor string
if reqPayload.NextPageToken != "" {
    // Parse page token
    cursor, err = ParseGetVideoCollectionPagingToken(reqPayload.NextPageToken)
    if err != nil {
        res := handler.NewErrorResponse(http.StatusBadRequest, err.Error())
        handler.Redererer.JSON(w, res.Status, res)
        return
    }
}
```

# Call StreamingService with context and paging opts, and if it succeeded, return the encoded payload.

```
// Get the video objects
records, npt, err := d.StreamingService.GetVideoCollection(r.Context(), &VideoCollectionPagingOptions{
    Offset: reqPayload.Offset,
   Cursor: cursor,
})
if err != nil {
    if err == ErrResourceNotFound {
        res := handler.NewErrorResponse(http.StatusNotFound, err.Error())
        handler.Redererer.JSON(w, res.Status, res)
        return
    res := handler.NewErrorResponse(http.StatusInternalServerError, err.Error())
    handler.Redererer.JSON(w, res.Status, res)
    return
```

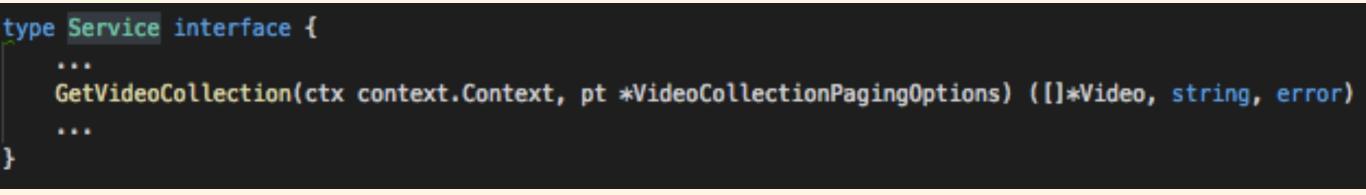
resPayload := encodeGetVideoCollectionResponse(records, npt)

handler.Redererer.JSON(w, http.StatusOK, resPayload)

#### GetVideoCollection will return

- 1. videos
- 2. JWT which contains a pager cursor.
- 3. error

JWT is generated by using `dgrijalva/jwt-go`



# Access to data storage, with repository which implements GetVideos.

```
videos, cursor, err := s.repo.GetVideos(ctx, pt)
if err != nil {
    return nil, "", err
var npt string
if cursor != "" {
    npt, err = BuildVideoCollectionPagingTokenString(cursor)
    if err != nil {
        return nil, "", err
```

return videos, npt, nil

#### Repository is an Interface. It hides which data adapter you depend on. This means you can define MockRepository and replace to them in test code.

# type Repository interface { ... GetVideos(context.Context, \*VideoCollectionPagingOptions) ([]\*Video, string, error) ... }

#### Access to Datastore on Google Cloud Platform. (with `mjibson/goon`) You can switch the adapter to the clients of

MySQL, Postgres, or in-memory database etc...

```
func (repo streamingRepository) GetVideos(ctx context.Context, pt *streaming.V:
    var vs []*streaming.Video
    g := goon.FromContext(ctx)
    query := datastore.NewQuery("Video").Limit(pt.Offset).Order("-CreatedAt")
    if pt.Cursor != "" {
        cursor, err := datastore.DecodeCursor(pt.Cursor)
        if err != nil {
            return nil, "", err
        }
        query = query.Start(cursor)
    }
```

#### Conclusion

- DDD-like architecture is good for Go API development. You can cleverly escape from hell of circular imports.
- **Repository Interface** makes the way to access data **pluggable**.
- In combination with some packages, net/http is surely enough to implement API server. (But there are few samples so it looks hard at first.)

#### Thanks!

- Questions? Come talk to me or contact to the following accounts!
- @\_\_timakin\_\_
- timaki.st@gmail.com

