**KRATOS’s LAYOUT**

├── go.mod   
├── go.sum  
├── LICENSE  
├── README.md  
├── api // Includes .proto API files and the .go files which generated from them.  
│   └── helloworld  
│   ├── errors  
│   │   ├── helloworld.pb.go  
│   │   ├── helloworld.proto  
│   │   └── helloworld\_errors.pb.go  
│   └── v1  
│   ├── greeter.pb.go  
│   ├── greeter.proto  
│   ├── greeter\_grpc.pb.go  
│   └── greeter\_http.pb.go  
├── cmd // The entry point of the kratos app  
│   └── server  
│   ├── main.go  
│   ├── wire.go // wire library is for dependency injection  
│   └── wire\_gen.go  
├── configs // The configuration files for local development.  
│   └── config.yaml  
└── internal // All the codes which are private. Business logics are often exist in there, under "internal" directory for preventing from unwilling import.  
 ├── conf // The structure for configuration parsing, generated from .proto file  
 │   ├── conf.pb.go  
 │   └── conf.proto  
 ├── data // For accessing data sources. This layer is mainly used as the encapsulation of databases, caches etc. The implementation of repo interface which defined in biz layer should be placed here. In order to distinguish from DAO (data access object), the data layer stress on business. Its responsibility is to transform PO to DTO. We dropped the infra layer of DDD.  
 │   ├── README.md  
 │   ├── data.go  
 │   └── greeter.go  
 ├── biz // The layer for composing business logics. It is similar to the domain layer in DDD. The interface of repo are defined in there, following the Dependence Inversion Principle.  
 │   ├── README.md  
 │   ├── biz.go  
 │   └── greeter.go  
 ├──service // The service layer which implements API definition. It is similar to the application layer in DDD. The transformations of DTO to DO and the composing of biz are processed in this layer. We should avoid to write complex business logics here.   
 │ ├── README.md  
 │ ├── greeter.go  
 │ └── service.go  
 └── server // The creation of http and grpc instance  
    ├── grpc.go  
    ├── http.go  
    └── server.go

- **api**: bao gồm các file .proto định nghĩa api và các file .go được generate từ các file .proto đó. (Lưu ý: về thứ tự các api http khi khởi tạo)

*Nên*: đặt tất cả các định nghĩa api tại lớp này và sử dụng protobuf để định nghĩa service và message.

*Không nên*: viết các logic xử lý ở đây.

- **cmd**: chứa tệp entry point của kratos, sử dụng wire để generate DI

(Lưu ý: file wire\_gen.go được generate ra không được chỉnh sửa)

*Nên*: khởi tạo và cấu hình ứng dụng ở đây, thiết lập DI.

*Không nên*: đặt các logic xử lý phức tạp ở đây.

- **configs**: chứa tệp cấu hình môi trường phát triển cục bộ.

- **internal**: chứa các đoạn mã private và các business logics thường được đặt ở lớp internal, tránh những import không mong muốn.

+ **conf**: chứa các cấu trúc để phân tích cấu hình, được sinh ra từ các tệp .proto.

+ **biz**: các business logics được thực hiện ở tầng biz.

*Nên*: định nghĩa các interface của repo, tuân theo nguyên tắc Dependence Inversion Principle (chữ D trong SOLID), đặt các business logic ở đây.

*Không nên*: viết bất kỳ xử lý database ở đây.

+ **data**: truy cập vào các nguồn dữ liệu như database, cache,…

*Nên*: đặt tất cả mã code liên quan đến truy cập dữ liệu và thao tác dữ liệu vào đây; thực hiện các phương thức chuyển đổi từ PO (persistence Object) sang DTO (Data Transfer Object).

*Không nên*: viết các business logics và gọi trực tiếp các api tại đây.

+ **service**: triển khai các định nghĩa api.

*Nên*: Chuyển đổi từ DTO (Data Transfer Object) sang DO (Domain Object) và tổng hợp các phương thức từ tầng biz; viết các logic xử lý đơn giản để chuyển đổi dữ liệu.

*Không nên*: viết các business logics phức tạp và truy cập trực tiếp vào tầng data.

+**server**: tạo các instance của HTTP và gRPC, middleware cũng nên đặt ở đây.

*Nên*: khởi tạo và cấu hình server HTTP và gRPC tại đây.

*Không nên*: đặt các xử lý khác ngoài việc khởi tạo server ở đây.