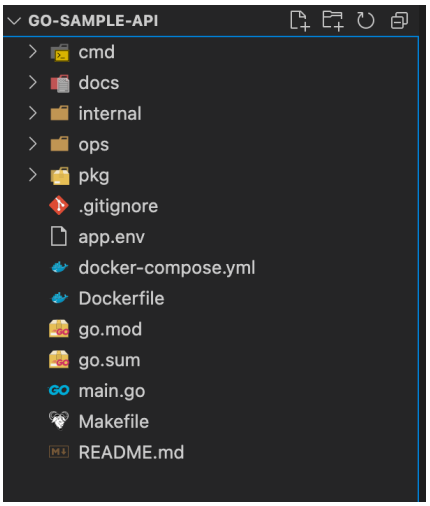
Hi There! =)

Vou compartilhar um pouco aqui sobre o meu repo de API de exemplo com Golang que eu costumo usar como base para "new project" de qualquer outro projeto Golang. Não entenda isso como um padrão de projeto, use-o apenas como "modelo de exemplo", seu cenário irá ditar o que é melhor pra você.



Bom, antes de começar a explicar pacote por pacote, aqui eu segui uma linha de Clean Arch, mas não *by the book* como vocês irão ver. Se voce ainda nao entende Clean Arch, por favor, leia isso aqui antes de prosseguir: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

- "A mas por que você não usou Onion, Hexagonal ou *blablabla*"
- "Onion, Hexagonal, Clean ou seja lá qual for que você esteja pensando neste momento, todas têm o mesmo objetivo: *Baixo acoplamento e Alta coesão.* Se você chegou nisso em um pattern que você desenhou na sua hora de almoço, ótimo! agora é só você convencer seu time a utilizá-lo xD"

- **main.go**
  - Eu gosto de deixar o arquivo main separado de todos os pacotes pois facilita pra quem vê o código pela primeira vez e não sabe por "onde começar". Nele eu só coloco a chamada para o meu arquivo que irá dar start na minha aplicação. Obs: Aqui eu não estou fazendo isso, mas eu poderia receber como parâmetro o nome da app que quero iniciar.
- **cmd**
  - *cmd* é um nome de pacote bem comum que você encontrará em vários outros projetos Golang. Por "padrão", a pasta *cmd* deve conter as suas aplicações, ou seja, cada aplicação que você tem.
    - https://github.com/golang-standards/project-layout
  - Refletindo um pouco sobre o *standard* citado acima e a necessidade de minha aplicação modelo, eu coloquei apenas um arquivo *server.go*, ou seja, este cara fica responsável por fazer as injeções, iniciar meu server, meu router e meu banco. Se eu tivesse necessidade de subir outra aplicação, por exemplo, eu criaria outro arquivo. Vamos supor que eu precise de um worker

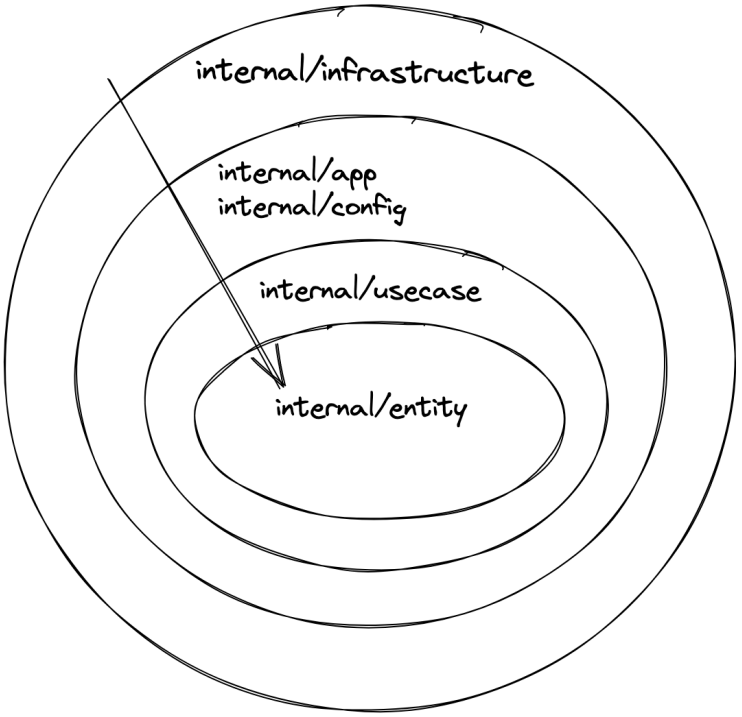
que vá utilizar boa parte do meu código mas quando deployado ele é outra aplicação… neste caso dentro da pasta CMD eu criaria um *worker.go*.

- **docs**
  - Está é uma pasta generica para documentação de sua aplicação, no meu caso, tenho o arquivo do swagger
- **internal**
  - *internal* é um nome de pacote bem comum que você encontrará em vários outros projetos Golang. Por "padrão", a pasta *internal* deve conter aquilo que for privado na sua aplicação, no meu caso, o código todo =).
    - https://github.com/golang-standards/project-layout
- **internal/app**
  - **Handlers** que ficam responsáveis por receber as requisições e direcioná las para o usecase.
  - **Middlewares** que neste exemplo tem só o Cors, mas poderia ter algo como Logs, Prometheus etc.
  - **Validate** que ficam responsáveis por conter as validações de entrada.
- **internal/config**
  - Aqui eu coloco as configurações da aplicação, no meu caso, tenho só um arquivo que faz o Load das variáveis de ambiente que usarei na aplicação.
- **internal/entity**
  - Como o exemplo aqui é bem simples, temos basicamente as *structs* que usaremos para nosso usecase. No Clean Arch essa camada confunde um pouco com o usecase, o Tio Bob explica com mais detalhes essa camada no link que mencionei lá no começo.
- **internal/usecase**
  - O package mais importante, aqui é onde vai o core, as regras de negócio de sua aplicação.
  - Aqui eu gosto de separar por subpastas com suas entidades. Ex: **/account**
    - Perceba que aqui eu tenho dois arquivos, account.go e interface.go
      - account.go basicamente tem a regra de negócio
      - interface.go
        - basicamente tem as assinaturas que são implementadas pelo usecase, ou seja, quem irá chamar você irá chamar através destas assinaturas.
        - basicamente tem as assinaturas que deverão ser implementadas pela camada de repository, banco, API etc.
- **internal/infrastructure**
  - Aqui contém tudo que *é detalhe.* Framework de router é detalhe, configuração de banco de dados é detalhe, framework de server http é detalhe e assim por diante. Isso quer dizer que se eu precisar mudar algo aqui, a mudança deve ser apenas aqui, demais camadas não devem sofrer com isso.
  - **Aqui tem um detalhe importante,** by the book (clean arch), as operações de SQL (select, create, delete e update) deveriam ficar na camada de *Interface & Adapters,* uma vez que ao trocar de banco eu não precisaria mudar as queries, mas na boa, do que eu já vi até agora, isso nunca acontece, por exemplo, mudando de *mysql* para *postgres* você irá perceber

que nas suas queries você usou o *?* e agora com *postgres* tem que ser *$1*. Por isso eu prefiro deixar as queries nesta camada de infraestrutura, numa eventual mudança de banco de dados é quase certo que terei que mexer nelas também.

- **ops**
  - Aqui eu coloco tudo que é arquivo de ops, deployment.yaml, initDB, configmap.yaml etc.
- **pkg**
  - *pkg* é um nome de pacote bem comum que você encontrará em vários outros projetos Golang. Por "padrão", a pasta *pkg* deve conter o que é público na sua aplicação.
    - https://github.com/golang-standards/project-layout
  - Aqui sigo a filosofia de que tudo que é genérico eu coloco lá, por exemplo, formatar data para um formato X, vai para o pkg, validar se saldo é positivo, **não vai para o pkg** pois isso é particular da minha aplicação.

Para facilitar o entendimento de "de-para" dos nomes de pacote para o Clean Arch, vou deixar um desenho bonitão aqui..



Agora ficou fácil né? kkkk

Perceba, a camada interna nunca depende da externa, é o famoso, *Dependency Inversion Principle (DIP),* não sabe o que é? https://medium.com/@kedren.villena/simplifying-dependency-inversion-principle-dip-59228122649a

- Entity nao pode depender de Usecase.

- Usecase nao pode depender de Interface & Adapters
- Interface & Adapter não pode depender de Framework and Drivers.

Como mencionei, independente se é clean arch, hexagonal, onion etc, ambas tem como objetivo deixar a arquitetura com baixo acoplamento, coesa e que consequentemente seja fácil de escrever testes de maneira isolada, testar novos frameworks, mudar a regra de negócio sem impactar tudo.
Não fique preso a nome de pacotes, pois na boa, cada projeto vai ser um nome diferente, se você for criar um projeto novo, aí você dá os nomes que quiser, desde que no final do dia você consiga chegar a uma aplicação fácil de testar, fácil de mudar e fácil de navegar nela.

Referências:

https://eltonminetto.dev/post/2020-06-29-clean-architecture-2anos-depois/
https://manakuro.medium.com/clean-architecture-with-go-bce409427d31
https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/