GO-03 04: Модули и пакеты. Layout проекта

Описание:

По ходу знакомства с пакетами и зависимостями в Go мы создали свой небольшой проект, в котором используются разные типы библиотек и пакетов. Но мы особенно не задумывались над тем, какой должна быть структура в проекте, какие пакеты должны лежать в каких директориях. И вообще не задумывались о том, как выглядит полноценное приложение или микросервис в промышленной среде. Предлагаем исправить эту ситуацию и рассмотреть, как же выглядит типовой проект, написанный на Go.
На самом деле, никто не запрещает вам делать структуру своего проекта как угодно и как вам удобно. В Go это распространенная практика - иметь не стандартизированную структуру. Но все же внутри коммьюнити за время существования Go были выработаны определенные подходы и best practices по организации проекта. Будь то микросервис или большое сложное приложение, например, Kubernetes, в коммьюнити часто используется project-layout.
Это некий boilerplate, шаблон, в котором разные директории имеют предназначения на любой случай жизни :) Используя его, можно упростить некоторые моменты для понимания вашего кода другими разработчиками. В некоторых случаях это позволяет избавиться от путаницы в пакетах и их зависимостях. В дальнейшем мы будем придерживаться именно подобной структуры, поэтому предлагаю сейчас рассмотреть ее подробнее.

```
myproject
└── cmd
    └── myapp
        └── main.go
└── internal
    └── handlers
    └── migrations
    └── routes
└── pkg
    └── helpers
    └── middleware
└── vendor
    └── github.com
    └── gopkg.in
└── api
└── web
```

```
└── configs
└── init
└── scripts
└── build
└── deployments
└── test
└── docs
└── tools
└── examples
└── third_party
└── assets
```

Так выглядит типичный шаблон проекта на Go. Выделенные жирным шрифтом директории - это основа любого проекта.

- cmd - тут находится само приложение, то есть его main пакет, файл main.go. Обычно в пакете main используется минимум бизнес-логики или ее там нет вовсе. Этот пакет служит для конфигурации всего приложения, для правил и аргументов его сборки. Другими словами - это техническая точка входа.
- internal - код, который непосредственно относится к текущему приложению и не будет переиспользован в других приложениях. По сути, это бизнес-логика приложения. В нашем случае это хендлеры, роуты, миграции. Когда мы начнем писать свое полноценное приложение, тут будет находиться основная бизнес-логика. Есть один нюанс. Директория internal имеет свою функциональность. Пакеты, находящиеся в директории internal, не могут быть импортированы через $GOPATH (или еще как-то) в другие приложения, так как подразумевается, что это тот код, который необходим только тут и нигде больше. Он не переиспользуем. Посмотреть подробнее о том, как это работает, можно тут .
- pkg - в этой директории находится код ваших библиотек, которые напрямую не зависят от текущего приложения. Это противоположность директории internal. Сюда нужно помещать пакеты, которые могут быть полезны как в этом приложении, так и в любом другом. Переиспользуемый код, например, хелперы для работы с http, брокерами сообщений, ДБ и т.д. В общем, тут лежит код, который может использоваться в других приложениях. И в нем нет зависимости от текущего приложения.
- vendor - название говорит само за себя. Это директория, которая чаще всего не находится под гитом и тут лежат библиотеки, подтянутые и управляемые менеджером зависимостей. Код из этой директории нельзя изменять. Часто такой директории может не быть в проекте на Go, так как Go использует go mod, а он, как мы помним, по умолчанию не использует режим vendor и может хранить все библиотеки вне зависимости от текущего проекта по пути $GOPATH/pkg/mod. Поэтому проекты, использующие go mod в обычном режиме (а не в режиме vendor), не имеют этой директории внутри проекта.

- api - файлы, так или иначе относящиеся к публичному API. Swagger, Proto и другие подобные схемы.
- web - любая статика для веб-приложений js css html. Также Go шаблоны или компоненты для js фреймворков и т.д.
- config - любые файлы конфигов, необходимые для приложения.
- scripts - скрипты, которые могут быть необходимы для сборки, анализа или установки проекта.
- build - файлы для инструментов сборки или чего-то подобного. Например, конфиги ci/cd или Dockerfile.
- deployments - файлы для непосредственного деплоя приложения в системы оркестрации, например, docker-compose, kubernetes/helm, mesos, terraform, bosh.
- test - тут могут быть расположены любые тесты и вспомогательные файлы для них.
- docs - директория, содержащая, например, описание markdown для вашего приложения. Любая документация или другая полезная информация.
- tools - код вспомогательных инструментов, который не зависит от пакетов в директориях pkg и internal.
- examples - сюда помещаются примеры использования вашего приложения или библиотеки.
- third_party - дополнительные утилиты типа Swagger IU. Или консоль для работы с GraphQL. Или другие инструменты, которые непосредственно не относятся к проекту, но необходимы для использования в нем. Также тут могут находится форкнутые измененные библиотеки.
- assets - картинки, иконки, логотипы и т.д.

Конечно же, использование всех директорий сразу в одном проекте не обязательно. Вы используете только то, что необходимо. Имея такую структуру, вы или любой другой разработчик знаете, куда положить или где можно найти реализацию того или иного функционала в проекте.
Если на данный момент вам непонятны какие-то принципы или предназначение элементов этой структуры, то не переживайте. Более подробно со структурой проекта мы будем знакомиться по ходу дела в этом практикуме. А дополнительную информацию можно посмотреть в репозитории project-layout

Задание:

1. Ознакомьтесь со структурой проекта project-layout.
2. Создайте новый проект(модуль go mod) с названием gopackages-layout.
3. Код из второго задания модуля перенесите в этот проект, в качестве структуры используйте project-layout.
4. Используйте только нужные директории.
5. Запустите его и проверьте, что все работает корректно.

6. Создайте новый репозиторий в группе golang_users_repos/<your_gitlab_id> в gitlab.rebrainme.com/ под названием gopackages-layout и запушьте туда проект с новой структурой.
7. Команду, которой был запущен новый проект, и ее вывод добавьте в ответ.
8. Ссылку на репозиторий добавьте в ответ.