

О спикере

- Работаю в ООО "РОБОВОЙС" на должности Senior Developer
- Занимаюсь интеграциями средств связи
- 6 лет опыта разработки
- 4 года опыта в Golang



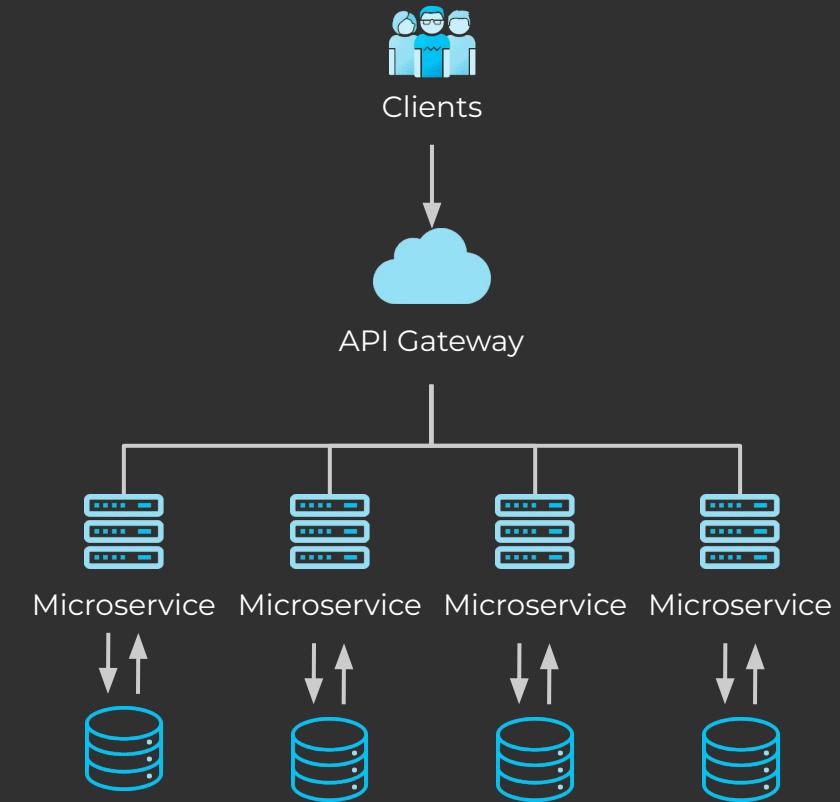
Цель интенсива

Что вы получите:

- Навыки работы в чистой архитектуре
- Навыки вывода логов
- Навыки работы с трассировкой
- Пример написания тестов

Что будет сделано:

- Полноценный микросервис с REST API (контакт сервис)
- Документация к REST API
- Логирование
- Трассировка
- Тесты



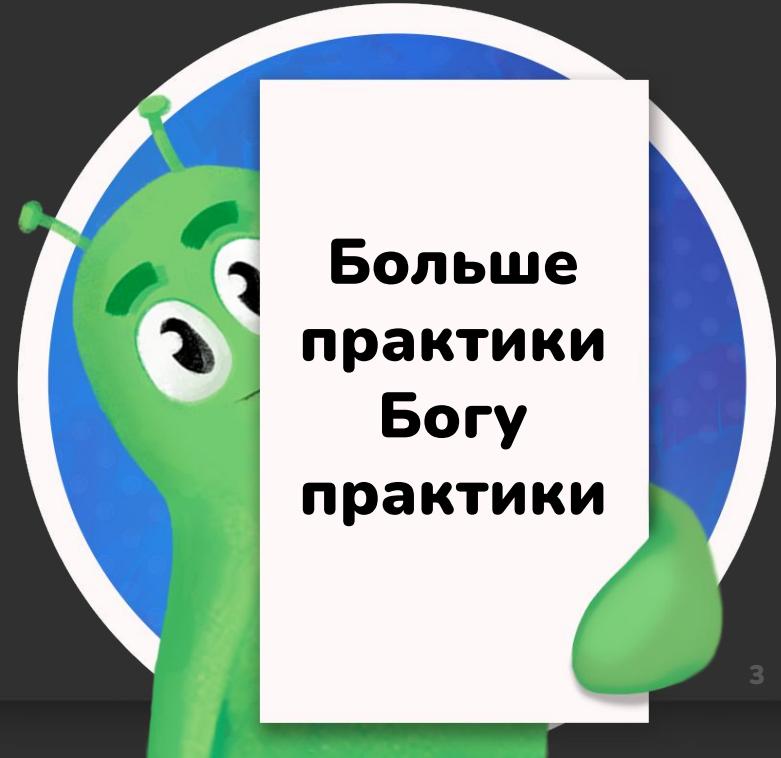
Все практические задания связаны между собой

{REST API}

Вводная

Как будет проходить обучение?

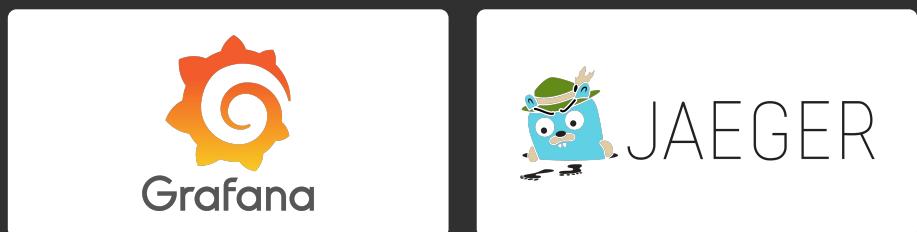
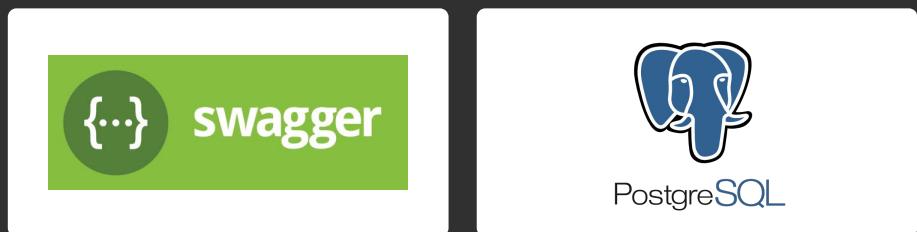
- Презентация, нацеленная на выполнение практики
- Минимальная информация для выполнение практики
- Разбор практики после выполнения заданий



Стек технологий

Как будет проходить практика?

- Требуются:
 - IDE Goland
 - GIT
- Доступы к:
 - GIT
 - Grafana
 - Jaeger
 - PostgreSQL
- Библиотеки
 - [pqx](#)
 - [scany/pgxscan](#)
 - [squirrel](#)
 - [open tracing](#)
 - [mockery](#)
 - [zap](#)
 - [viper](#)
 - [gin](#)



Что будет в курсе?

Структура проекта
(теория и практика)

Observability
(теория и практика)

Чистая архитектура
(теория и практика)

Тестирование
(демонстрация)

Таков путь...

День 1: 1. Создание структуры проекта

2. Чистая архитектура:
 - Структура папок по чистой архитектуре
 - Реализация – Domain
 - Интерфейс – Use Case
 - Интерфейс – Repository
 - Реализация – Delivery
 - Конструкторы слоёв
 - Инициализация слоёв на main

День 2:

- Реализация интерфейса – UseCase
- Реализация интерфейса – Repository
3. Использование – Context
4. Добавить логи
5. Добавить трассировку
6. Тестирование**



Правила

Где задавать вопросы?

В чате Telegram



Где получать ответы?

В чате Telegram в течение прохождения интенсива



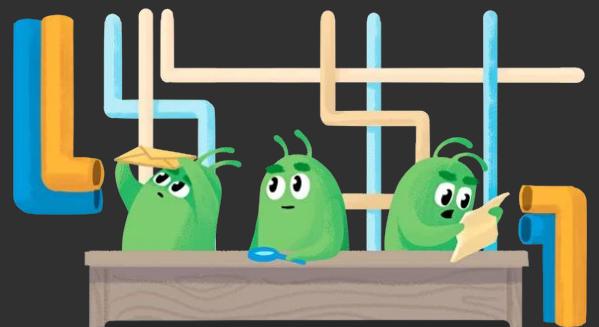
Структура проекта

Зачем нужна структура проекта?

Разбор основных папок /
Просмотр примеров проектов на Go

Иерархия папок

Зачем нужна
структурата проекта?



Зачем нужна структура проекта

- Зачем нужна структура проекта?
- Придерживаетесь ли вы шаблонов при создании новых проектов?
- Есть ли у вас сложности при работе в вашей структуре проекта?

Что если: делать, не думая о структуре?



The screenshot shows a code editor with four tabs open, each displaying a Go file from a project. The tabs are:

- call_manager.go
- contact_manager.go
- campaign_manager.go
- contact_manager_markers.go

The code in the contact_manager.go tab is annotated with various markers and notes:

- Annotations like `func (c *CallManager) MakeCallB` and `func (c *CallManager) ReadContac` are present.
- Notes like `// GetFileList - Get list files` and `// Проверяем атрибуты контакта` are interspersed throughout the code.
- Line numbers are visible on the left side of the code blocks.

The other tabs show similar Go code with their own annotations and line numbers.

P contact.proto ×

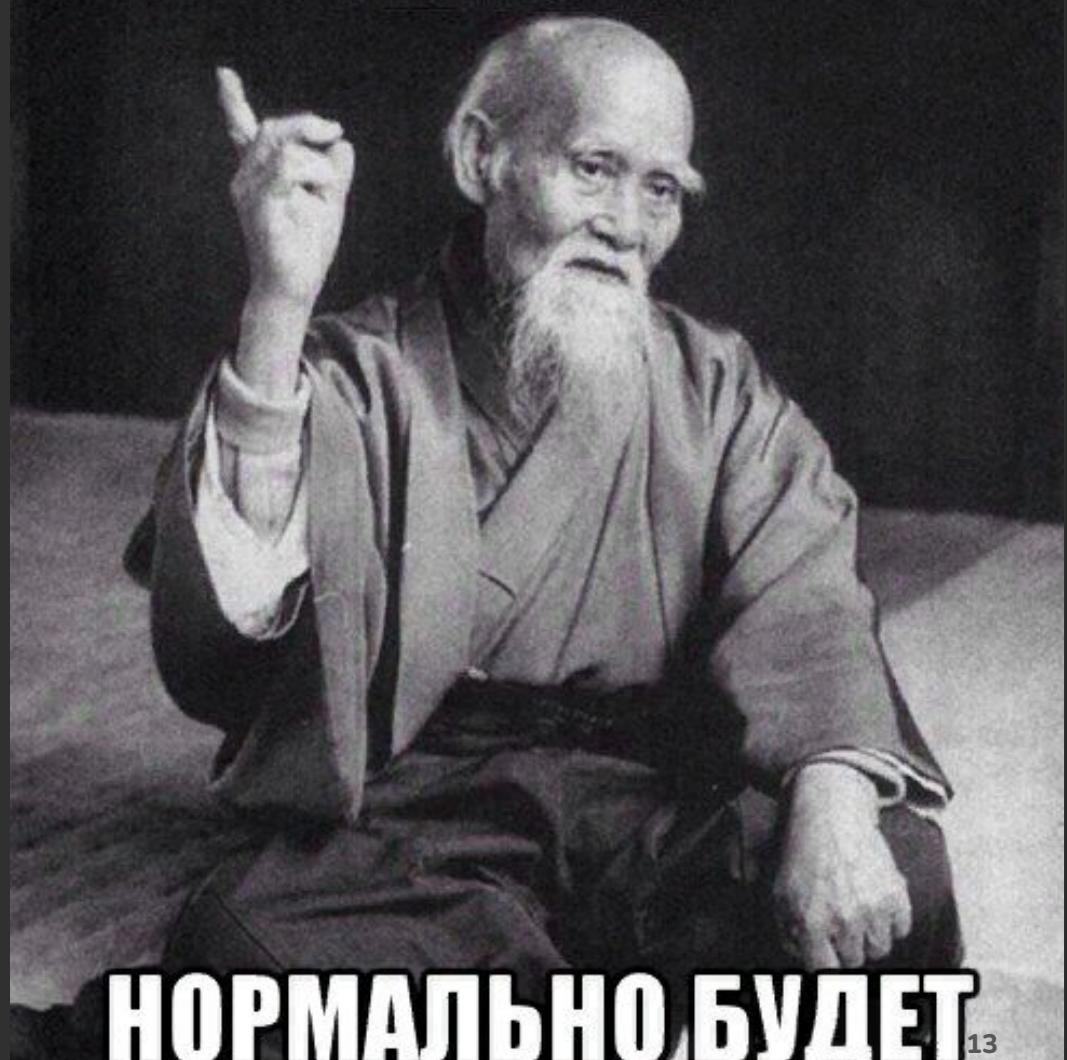
```
1925  
1926  
1927     +service ActionService {...}  
1941  
1942     +service ContactService {...}  
2006  
2007     +service GroupService {...}  
2039  
2040     +service CampaignService {...}  
2106  
2107     // TODO: Delete if it's dead code  
2108     +service ChatbotVersionService {...}  
2124  
2125     +service ScriptService {...}  
2128  
2129     +service CallService {...}  
2182  
2183     +service AMDService {...}  
2190  
2191     +service DialogService {...}  
2200  
2201     // ChatTest Service  
2202     +service ChatTest {...}  
2208  
2209     +service Crm {...}  
2212
```

А как нужно ?

Цели структуры проекта:

1. Ускорение разработки
2. Отражение целей проекта

НОРМАЛЬНО ДЕЛАЙ



НОРМАЛЬНО БУДЕТ

Разбор основных папок



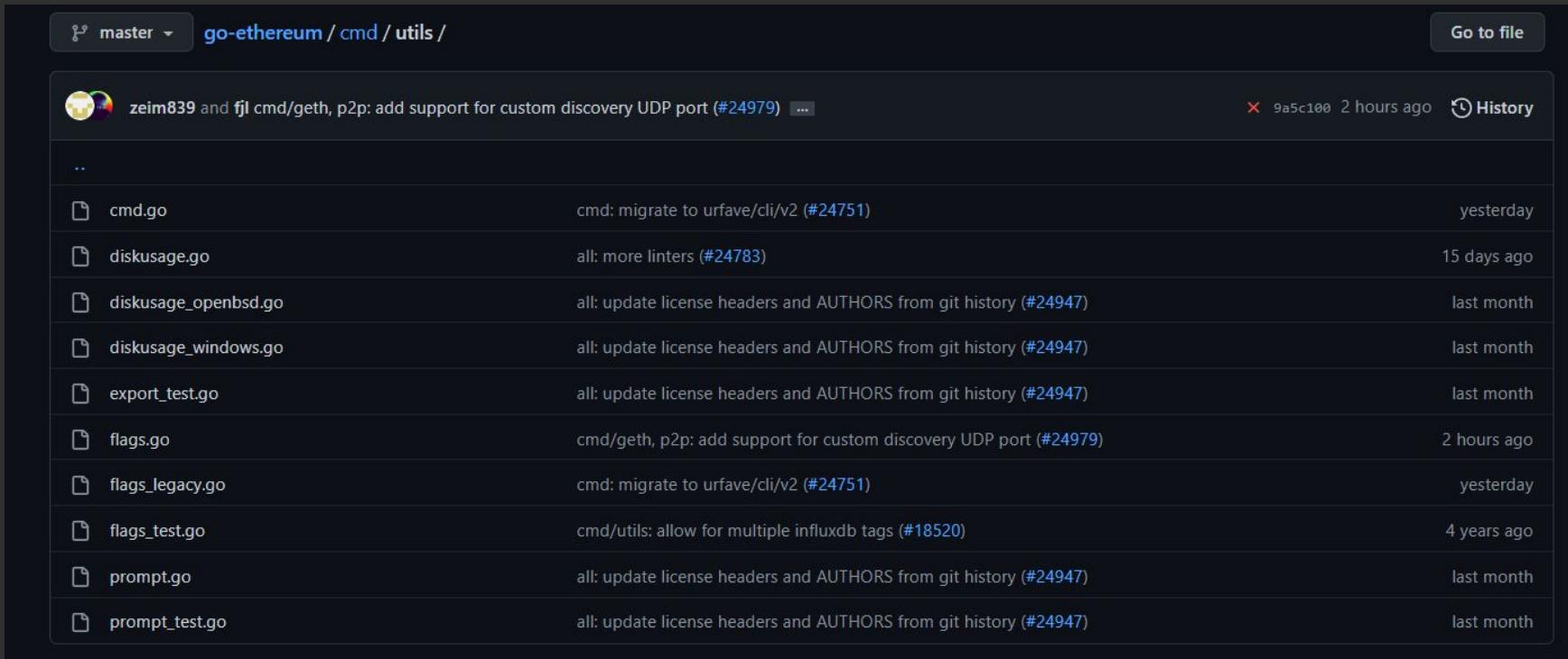
Разбор основных папок

- **cmd** – Основные приложения проекта
- **internal** – Внутренний код приложения и библиотек
- **pkg** – Код библиотек

Дополнительную информацию можно получить в github.com

cmd – основные приложения проекта ([git](#))

- Малые объемы кода
- cmd/app – основное приложение
- cmd/... – вспомогательные приложения



The screenshot shows a GitHub repository interface for the 'go-ethereum' project, specifically the 'cmd' directory. The top navigation bar shows the repository name and the 'master' branch. Below the navigation, there's a search bar and a 'Go to file' button. The main area displays a list of files and their commit history:

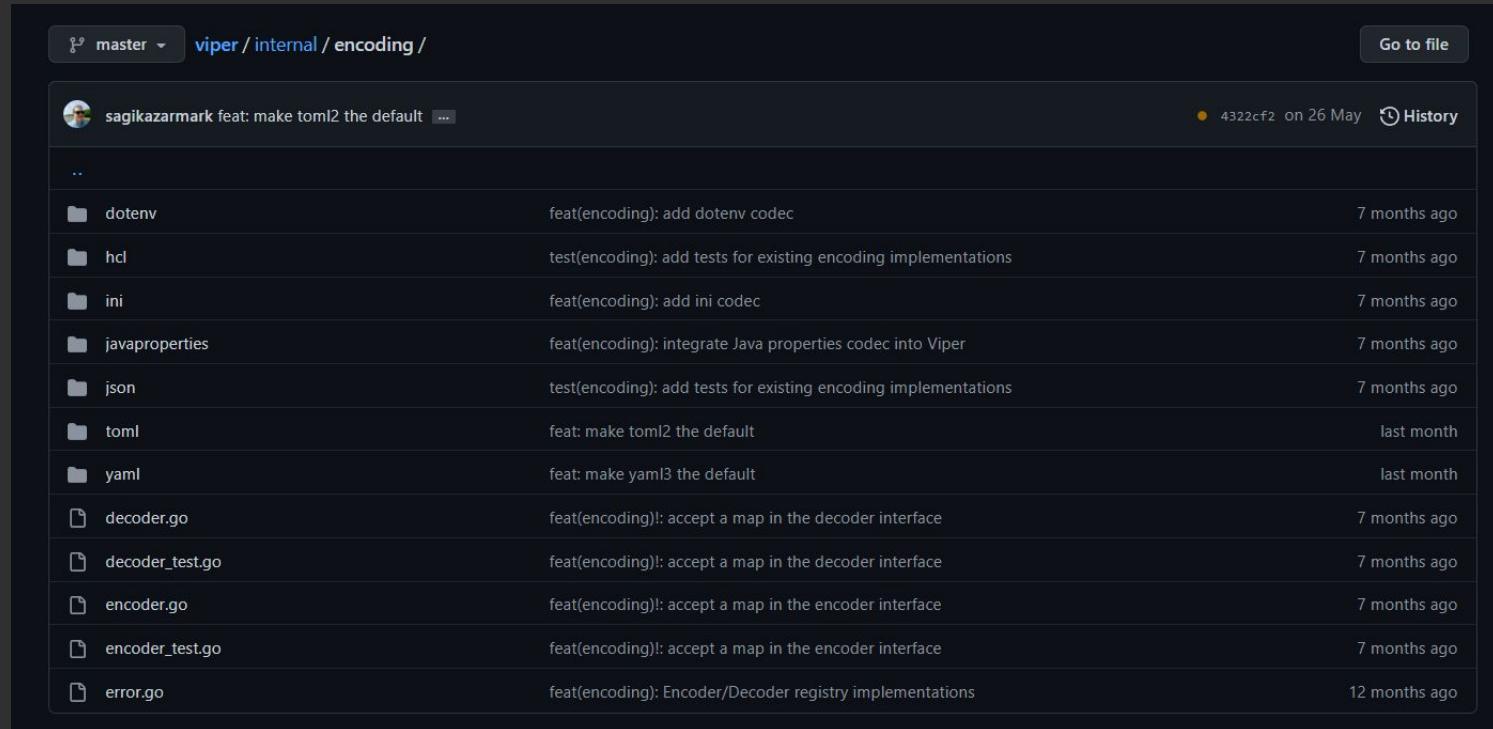
File	Commit Message	Time Ago
cmd.go	cmd: migrate to urfave/cli/v2 (#24751)	yesterday
diskusage.go	all: more linters (#24783)	15 days ago
diskusage_openbsd.go	all: update license headers and AUTHORS from git history (#24947)	last month
diskusage_windows.go	all: update license headers and AUTHORS from git history (#24947)	last month
export_test.go	all: update license headers and AUTHORS from git history (#24947)	last month
flags.go	cmd/geth, p2p: add support for custom discovery UDP port (#24979)	2 hours ago
flags_legacy.go	cmd: migrate to urfave/cli/v2 (#24751)	yesterday
flags_test.go	cmd/utils: allow for multiple influxdb tags (#18520)	4 years ago
prompt.go	all: update license headers and AUTHORS from git history (#24947)	last month
prompt_test.go	all: update license headers and AUTHORS from git history (#24947)	last month

internal – внутренний код приложения и библиотек

Не должен быть применен в других приложениях и библиотеках ([git](#)).

Этот шаблон навязан самим компилятором Golang.

Ознакомьтесь с [release notes](#) Go 1.4



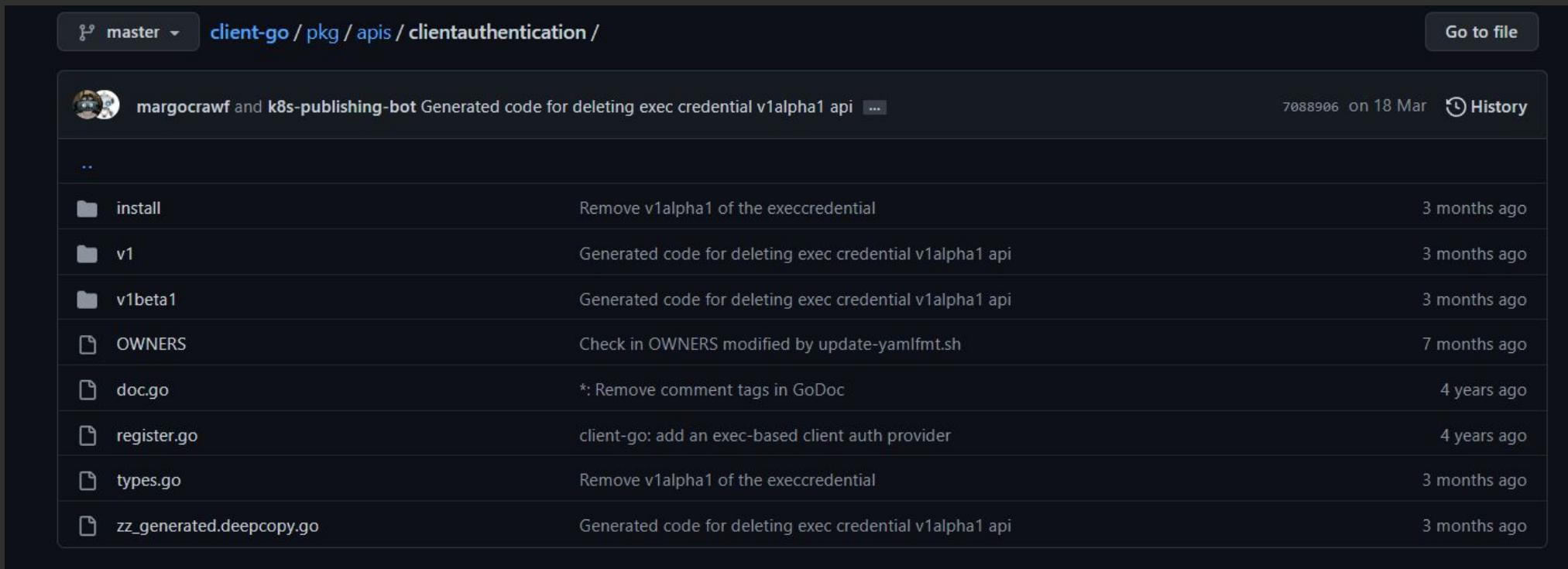
The screenshot shows a GitHub commit history for the `viper/internal/encoding` branch. The commits are listed below:

Commit	Message	Date
feat(encoding): add dotenv codec	sagikazarmark feat: make toml2 the default	7 months ago
test(encoding): add tests for existing encoding implementations		7 months ago
feat(encoding): add ini codec		7 months ago
feat(encoding): integrate Java properties codec into Viper		7 months ago
test(encoding): add tests for existing encoding implementations		7 months ago
feat: make toml2 the default		last month
feat: make yaml3 the default		last month
feat(encoding): accept a map in the decoder interface	decoder.go	7 months ago
feat(encoding): accept a map in the decoder interface	decoder_test.go	7 months ago
feat(encoding): accept a map in the encoder interface	encoder.go	7 months ago
feat(encoding): accept a map in the encoder interface	encoder_test.go	7 months ago
feat(encoding): Encoder/Decoder registry implementations	error.go	12 months ago

pkg – код библиотек

Пригоден для использования в сторонних приложениях ([git](#))

- Автономная работа
- Код в этой директории могут безопасно использовать другие



The screenshot shows a GitHub repository interface. At the top, it displays the repository path: master / client-go / pkg / apis / clientauthentication /. Below this is a list of commits:

Commit	Message	Date
... (ellipsis)		
install	Remove v1alpha1 of the execcredential	3 months ago
v1	Generated code for deleting exec credential v1alpha1 api	3 months ago
v1beta1	Generated code for deleting exec credential v1alpha1 api	3 months ago
OWNERS	Check in OWNERS modified by update-yamlfmt.sh	7 months ago
doc.go	*: Remove comment tags in GoDoc	4 years ago
register.go	client-go: add an exec-based client auth provider	4 years ago
types.go	Remove v1alpha1 of the execcredential	3 months ago
zz_generated.deepcopy.go	Generated code for deleting exec credential v1alpha1 api	3 months ago

Итоги

Что делать если у вас особый случай?

- Нет подходящей папки
- Нет иерархии
- Много ненужных папок
- И вообще, это сложно....



Иерархия папок

- project
 - pkg
 - services
 - serviceName
 - internal
 - cmd/app
 - configs
 - migrations
 - vendor



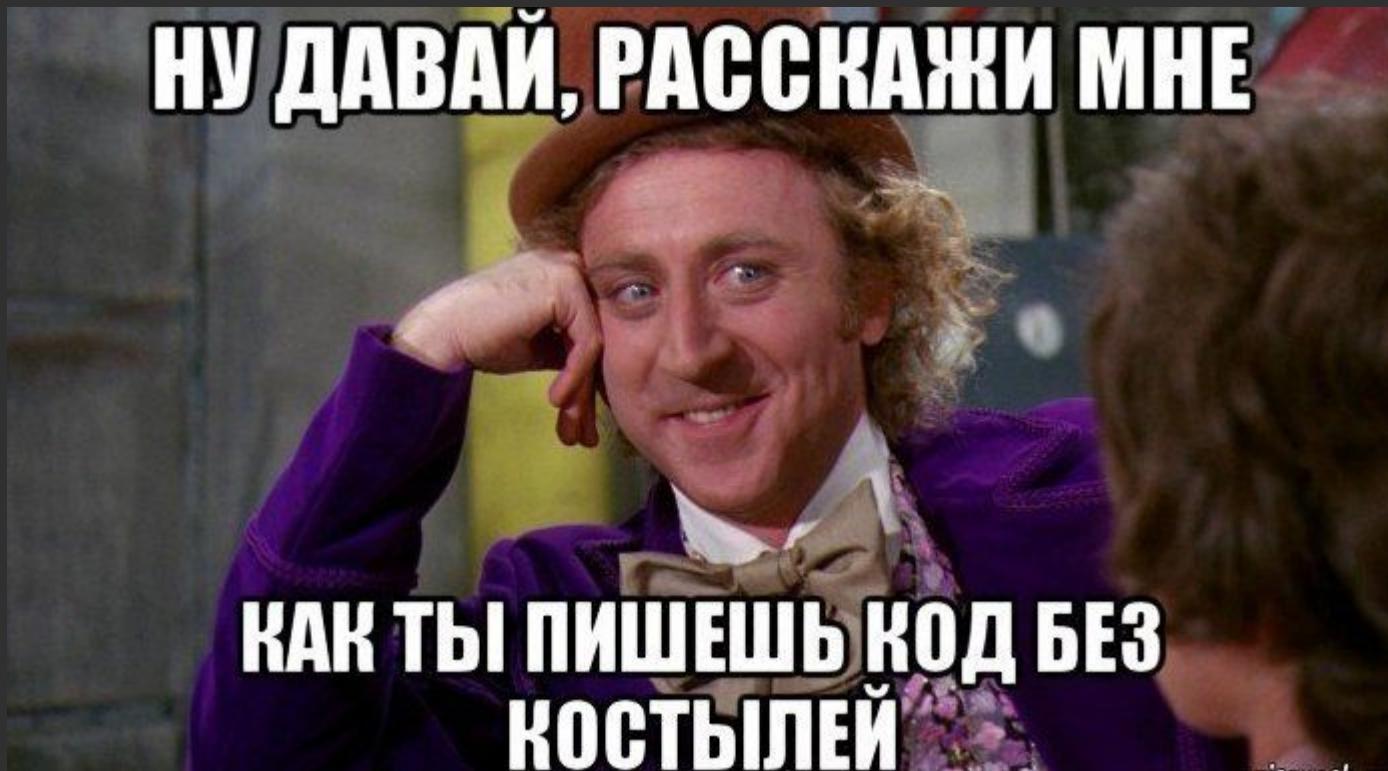
Практика: Создание структуры проекта (15 мин)

Цель: Подключиться к git и создать базовый проект для дальнейшей работы с ним

Задание:

1. Подключиться к git Слёрм
 - Подключиться по адресу https://gitlab.slurm.io/edu/architecture_go_3
 - Доступы для git можно получить в личном кабинете Слёрм
 - Создайте свою ветку в git.
Правила наименования **ФамилияИнициалы-01**. Например, **kolyadkons-01**
2. Создать примитивную структуру проекта
 - Создать папку **pkg**
 - Создать папку **services/contact/internal/cmd/app**
 - Создать **main** функцию с выводом в консоль (папка **app**)
3. Запустить проект
4. Залейте свою ветку в git

Демонстрация



Перерыв 10 мин

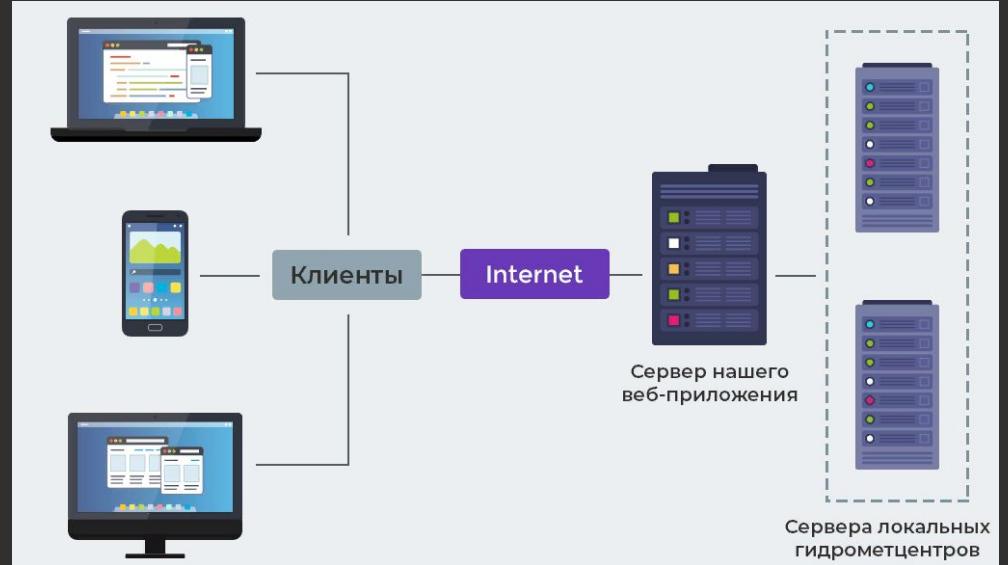
Вопросы

1. Как хорошо/быстро ориентируетесь в структуре нового проекта?
2. Как быстро можете понять, что написано в коде?
3. Практикуете написание тестов?
4. Есть сложности с покрытиями конкретных функций тестами?
5. Боитесь вносить изменения в код, т.к. не понимаете, что может сломаться?

Что объединяет
эти вопросы?



Что такое архитектура?



Архитектура - Компромисс

Чистая архитектура

- Что такое чистая архитектура?
- Зачем её использовать?
- Слои чистой архитектуры
- Правило зависимостей
- Пересечение границ



Что такое чистая архитектура?

Способ организации кода, который **способствует** строгому **разделению ответственности**

При этом между ними передаются только те ресурсы, которые необходимы для выполнения поставленной задачи

Способ разделения логики кода на части (слои):

- бизнес-правила
- обращение к внешним системам
- получение внешних запросов (API)



Зачем использовать чистую архитектуру?

Стандартизация проектов / модулей /
микросервисов

Поддержка старых проектов

Быстрое переключение разработчиков
между проектами

Независимость от сторонних систем

Быстрый старт нового проекта

Тестирование системы.
Скорость и удобство написания тестов

Зачем использовать чистую архитектуру?

Независимость от сторонних систем:

- Фреймворков
- UI
- Базы данных
- Внешних сервисов

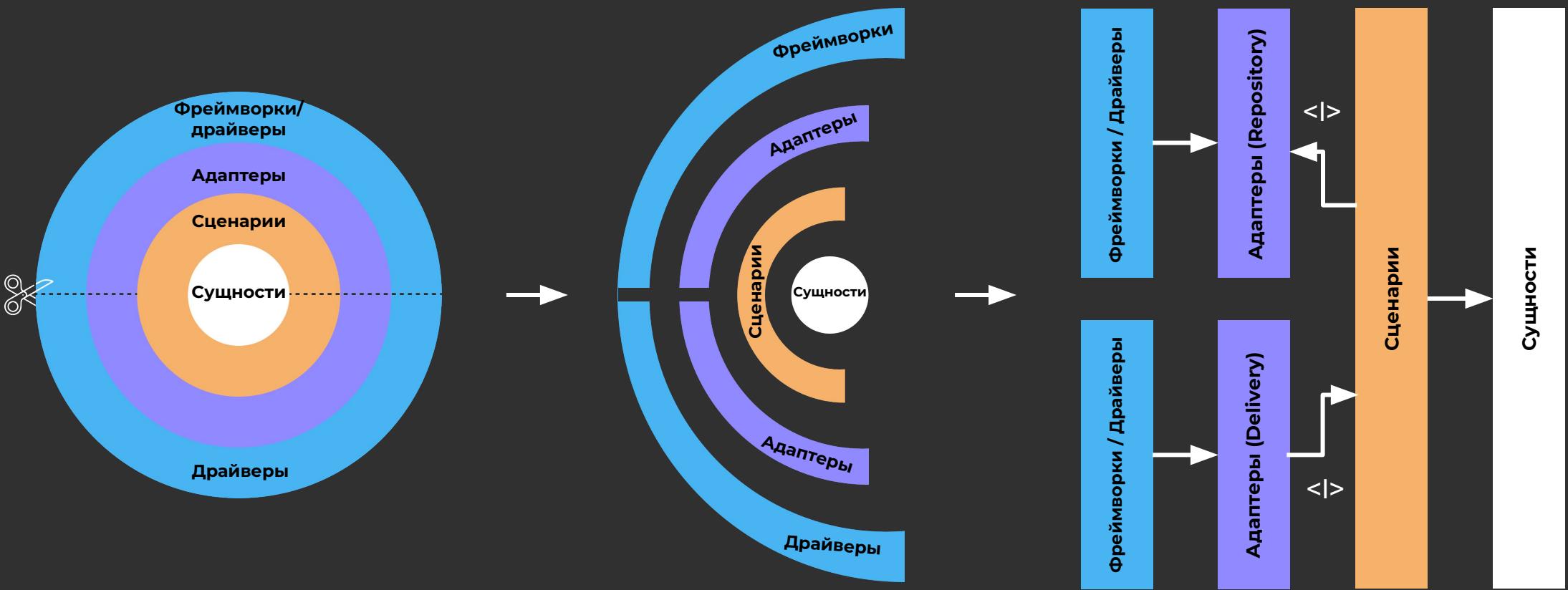
Слои чистой архитектуры

- Domain – Сущности
- Use Case – Сценарии
- Delivery / Repository – Адаптеры
- Drivers – Фреймворки и драйверы



Правило зависимостей

Зависимости в исходном коде должны быть направлены внутрь,
в сторону высокоровневых политик



Структура папок по чистой архитектуре (20 мин)

Ветка с примером прошлого задания: 001-day-1-project-layout

Цель: Создать структуру папок, используя чистую архитектуру

1. Создайте свою ветку в git из ветки kolyadkons-01. Правила наименования ФамилияИнициалы-02
Например, kolyadkons-02
2. В папке **services** создать подпапку **/contact/internal**
3. В **internal** реализовать структуру проекта по чистой архитектуре (создать папки)
 - a. Domain
 - b. Repository
 - c. Delivery
 - d. Use Case
4. Подготовить файлы(папки с файлами) для интерфейсов
 - a. Интерфейс обращения Delivery к Use Case*
 - b. Интерфейс обращения Use Case к Repository*
5. Запустить проект
6. Залейте свою ветку в git

* самостоятельно решите, где размещать интерфейсы.

Демонстрация



Сущности (Domain)

Сущности определяются бизнес-правилами предприятия:

- Реализует все случаи использования системы
 - Объект с методами
 - Структуры данных
 - Функции

Не восприимчив к внешним изменениям:

- Базы данных
- Пользовательским интерфейсом
- Фреймворком

Изменения в работе сущности повлияет:

- Use Case – сценарии
- Delivery / Repository – адаптеры
- Drivers – Фреймворки / драйверы



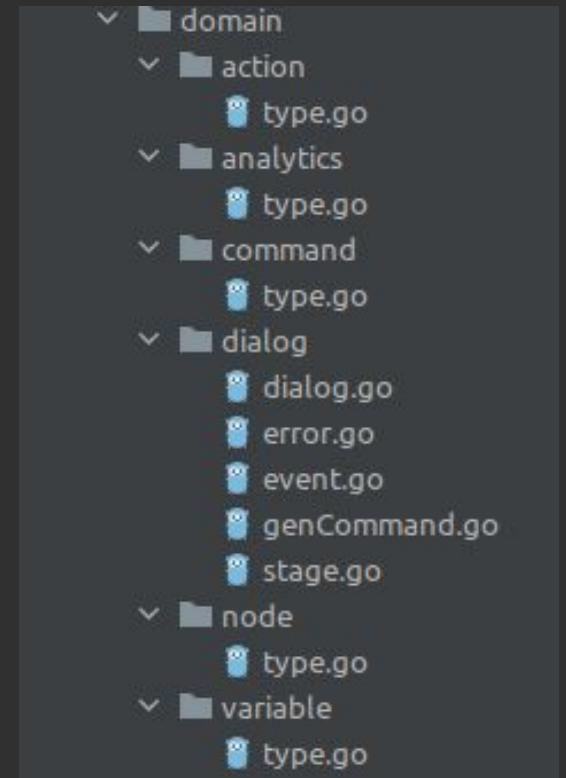
Сущности (Domain)

Примеры:

- Создание структур с которыми взаимодействует непосредственно пользователь
 - Раздел контактов – Контакт
 - Раздел группировки контактов – Группа
 - Раздел пользователя системы – Пользователь
- Проверка обязательности / формата полей
- Бизнес логика
 - При создании контакта, получать по номеру UTC
 - Проверка валидности паспортных данных
 - Заполнение ФИО из атомарных полей

Домен ничего не знает о других уровнях

Он содержит чистую бизнес-логику



Наполнение – Domain (20 мин)

Ветка с примером прошлого задания: 002-day-1-folders-clean-architecture

Цель: Создание структур Контакта, Группы и добавление бизнес логики.

1. Создайте свою ветку в git из ветки kolyadkons-03.
Правила наименования **ФамилияИнициалы-03**. Например, kolyadkons-03
2. В папке services/contact/internal/domain:
 - a. Создать структуру Контакта
 - i. Поля: идентификатор, ФИО, номер телефона
 - ii. Номер телефона – должен иметь только числа (бизнес правило)
 - b. Создать структуру Группы
 - i. Поля: идентификатор, название
 - ii. Название – максимальная длина строки 250 символов (бизнес правило)
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

Обед 60 мин

Сценарии (Use Case)

Use case – детализация, описание действия, которое может совершить пользователь системы

Использует domain, но ничего не знает о внешних слоях

Он не знает, вызывается ли:

- HTTP-запросом
- Обработчиком Pub/Sub
- Командой CLI

Доступы к Repository осуществляются при помощи interface который диктует Use case

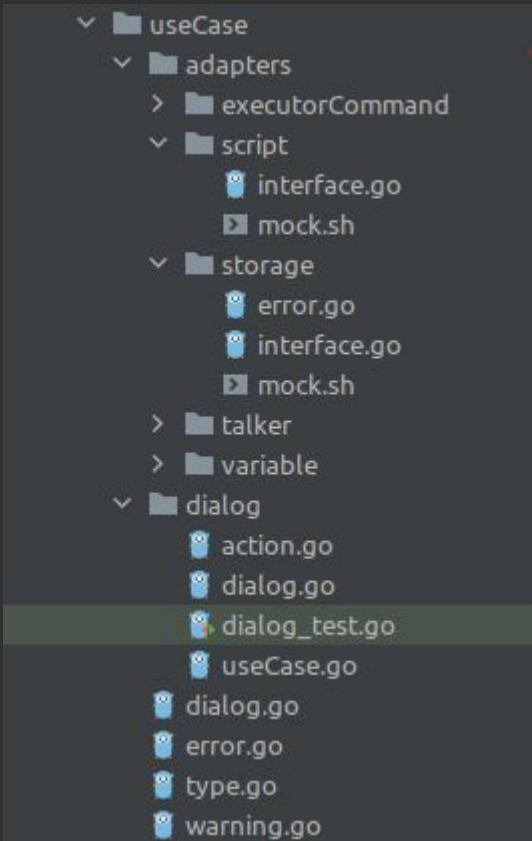
Доступ к Use case осуществляются при помощи interface который диктует Use case



Сценарии (Use Case)

Примеры:

- Обработка команд полученных от Delivery
- Работа с другими Use Case
- Объединение данных с различных источников (Repository)
 - Базы данных
 - Сторонних сервисов
- Отправка команд / событий (Repository)
 - Сохранение данных
 - Генерация событий



Интерфейс – Use Case (40 мин)

Ветка с примером прошлого задания: 003-day-1-domain

Цель: Создать интерфейс для Use Case

1. Создайте свою ветку в git из ветки kolyadkons-03.
Правила наименования **ФамилияИнициалы-04**. Например, kolyadkons-04
2. В папке **services/contact/internal/useCase**:
 - a. Создать интерфейс для Use Case (в файле / папке интерфейсов для Use Case)
 - i. CRUD контакта
 - ii. Создать группу
 - iii. Чтение группы
 - iv. Создание и добавление контакта в группу
3. Запустить проект
4. Залейте свою ветку в git

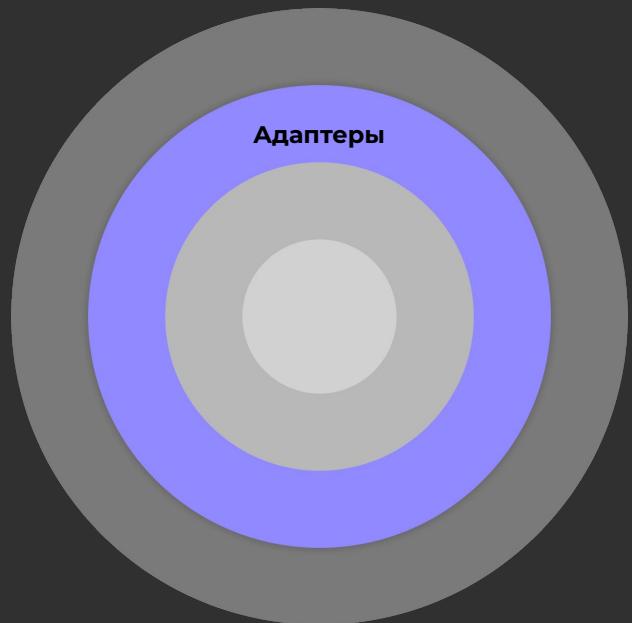


Демонстрация

Интерфейс-Адаптеры

Основная функция уровня интерфейсного адаптера –
получение и отправка данных + преобразование данных

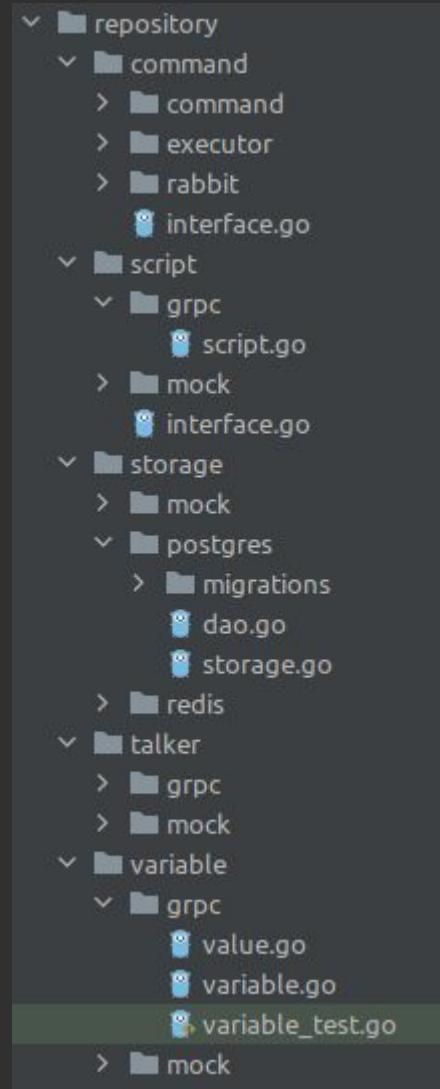
- Получение данных из внешних слоёв
- Отправка данных до внешних слоёв
- Данные из внешних слоёв к удобному формату внутренних слоёв
- Данные из внутренних слоёв к удобному формату внешних слоёв



Интерфейс-Адаптеры (Repository)

- Обеспечивает абстракцию данных через интерфейс
- Использование этого шаблона может помочь добиться слабой связи и сохранить объекты Domain

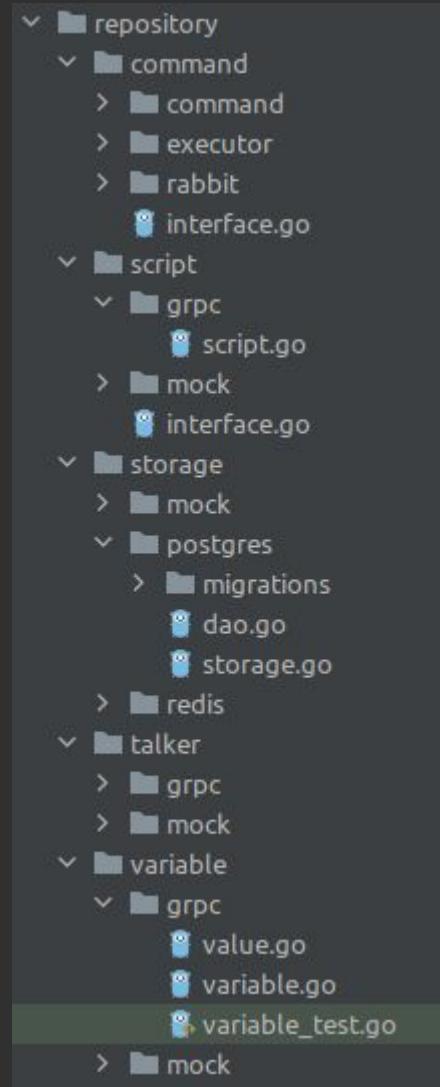
```
15 ① ② type Storage interface {
16    Contact
17    Group
18    Campaign
19  }
20
21 ① ② type Contact interface {
22    ③ CreateContact(ctx robo_context.Context, createdBy uuid.UUID, contacts ...*contact.Contact) ([]*contact.Contact, error)
23    ③ UpdateContact(ctx robo_context.Context, ID, createdBy uuid.UUID, updateFn func(c *contact.Contact) (*contact.Contact, error)) (*contact.Contact, error)
24    ③ DeleteContact(ctx robo_context.Context, createdBy uuid.UUID, filters filter.Filters) ([]*contact.Contact, error)
25
26    ContactReader
27  }
28
29 ① ② type ContactReader interface {
30    ③ ListContact(ctx robo_context.Context, createdBy uuid.UUID, parameter queryParameter.QueryParameter) ([]*contact.Contact, error)
31    ③ ReadContactByID(ctx robo_context.Context, createdBy, ID uuid.UUID) (response *contact.Contact, err error)
32    ③ CountContact(ctx robo_context.Context, createdBy uuid.UUID, filters filter.Filters) (uint64, error)
33    ③ ReadContactByGroupID(ctx robo_context.Context, createdBy, groupID uuid.UUID, parameter queryParameter.QueryParameter) ([]*contact.Contact, error)
34    ③ ListUTCByAllContacts(ctx robo_context.Context, createdBy uuid.UUID) ([]*useCase.GroupContactsByUTC, error)
35
36    ③ FieldListContact(ctx robo_context.Context, createdBy uuid.UUID) ([]*contact.Field, error)
37    ③ GetContactFieldValue(ctx robo_context.Context, contactID uuid.UUID, field, fieldID string) (string, error)
38  }
39
40 ① ② type Group interface {
41    ③ CreateGroup(ctx robo_context.Context, createdBy uuid.UUID, group *group.Group) (*group.Group, error)
42    ③ UpdateGroup(ctx robo_context.Context, createdBy, ID uuid.UUID, updateFn func(group *group.Group) (*group.Group, error)) (*group.Group, error)
43    ③ ListGroup(ctx robo_context.Context, createdBy uuid.UUID, parameter queryParameter.QueryParameter) ([]*group.Group, error)
44    ③ ReadGroupByID(ctx robo_context.Context, createdBy, ID uuid.UUID) (*group.Group, error)
45    ③ DeleteGroup(ctx robo_context.Context, createdBy, ID uuid.UUID) error
46    ③ CountGroup(ctx robo_context.Context, createdBy uuid.UUID, filters filter.Filters) (uint64, error)
47    ContactInGroup
48  }
```



Интерфейс-АдAPTERы (Repository)

Примеры:

- Взаимодействие с БД
- Кэширование данных
- Отправка данных сторонним системам
 - REST
 - gRPC
- Преобразование данных полученных от сторонних систем к формату удобному для Use Case
 - Приведение к структурам Use Case
 - Приведение к структурам Domain
- Преобразование данных полученных Use Case к формату удобному для сторонних систем
 - Приведение к структурам Сторонней системы



Интерфейс – Repository (20 мин)

Ветка с примером прошлого задания: 004-day-1-interface-use-case

Цель: Создать интерфейс для Repository

1. Создайте свою ветку в git из ветки kolyadkons-04.
Правила наименования ФамилияИнициалы-05. Например, kolyadkons-05
2. В папке services/contact/internal/useCase/adapters:
 - Создание интерфейса для repository. Интерфейс репозитория должен быть продиктован требованиями от Use Case (в файле / папке интерфейсов для adapters)
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

Перерыв 10 мин



Интерфейс-Адаптеры (Delivery)

Единственная точка входа для сторонней системы

- Выставление внешнего API
- Работа с методами Use Case

Не может напрямую обращаться к Repository

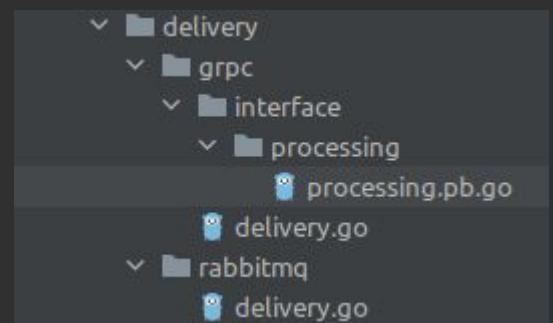
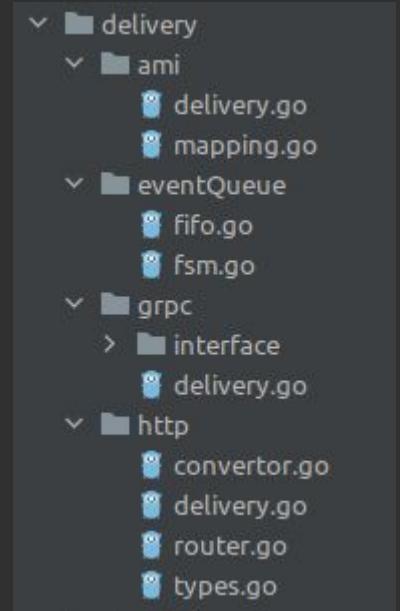


Интерфейс-Адаптеры (Delivery)



Примеры:

- Получение данных от сторонних систем
 - REST
 - gRPC
- Преобразование данных полученных от сторонних к формату удобному для Use Case
 - Приведение к структурам Use Case
 - Приведение к структурам Domain
- Формирование API для взаимодействия



Реализация Delivery (50 мин)

Ветка с примером прошлого задания: 005-day-1-interface-repository

Цель: Реализовать Delivery

1. Создайте свою ветку в git из ветки kolyadkons-5.
Правила наименования **ФамилияИнициалы-6**. Например, kolyadkons-6
2. Добавить routers в папке services/contact/internal/delivery:
 - CRUD для контакта и вызвать interface Use Case
 - Создание для группы и вызвать interface Use Case
 - Чтение для группы и вызвать interface Use Case
 - Создание и добавление контакта в группу и вызвать interface Use Case
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

Фреймворки и драйверы. (Drivers)

Самый внешний уровень модели

- Фреймворки
- Инструменты доступа к базам данных
- Веб-фреймворки



Перерыв 10 мин



Пересечение границ

- Примитивные типы
- Простые структуры
- Data Transfer Objects (DTO)
- Вызовы функций через аргументы



Важно, чтобы через границы передавались простые, изолированные структуры данных

При передаче через границу данные всегда должны принимать форму, наиболее удобную для внутреннего слоя.

Как это всё объединять?

```
package main

import (
    "fmt"
    "net"
    "os"
    "os/signal"
    "syscall"

    "github.com/pkg/errors"
    "github.com/spf13/viper"

    "robovoice/micro-services/dialogProcessor/cmd/app/wire"
    grpcRabbitMQ "robovoice/micro-services/dialogProcessor/internal/processing/delivery/rabbitmq"
    "robovoice/micro-services/dialogProcessor/internal/processing/repository/command/executor"
    messageBroker "robovoice/micro-services/dialogProcessor/internal/processing/repository/command/rabbit"
    grpcScript "robovoice/micro-services/dialogProcessor/internal/processing/repository/script/grpc"
    storage "robovoice/micro-services/dialogProcessor/internal/processing/repository/storage/postgres"
    grpcTalker "robovoice/micro-services/dialogProcessor/internal/processing/repository/talker/grpc"
    grpcVariable "robovoice/micro-services/dialogProcessor/internal/processing/repository/variable/grpc"
    "robovoice/micro-services/dialogProcessor/internal/processing/useCase/dialog"
    "robovoice/pkg/di/robo_context"
    log "robovoice/pkg/di/robo_logger"
    "robovoice/pkg/messageBroker/rabbitMQ"
    "robovoice/proto"
    protoClumsyTalker "robovoice/protobuf/clumsyTalker"
    "robovoice/protobuf/contactServiceV2/contact"
    protoScriptAction "robovoice/protobuf/scriptService/action"
    protoMarker "robovoice/protobuf/scriptService/marker"
    protoScript "robovoice/protobuf/scriptService/script"
```

Пакет cmd
Функция main



Создать конструкторы слоёв. (15 мин)

Ветка с примером прошлого задания: 006-day-1-implemented-delivery

Цель: Создать конструктор для слоёв Use Case и Repository

1. Создайте свою ветку в `git` из ветки `kolyadkons-06`.

Правила наименования **ФамилияИнициалы-07**. Например, `kolyadkons-07`

2. В папке `services/contact/internal/repository`:

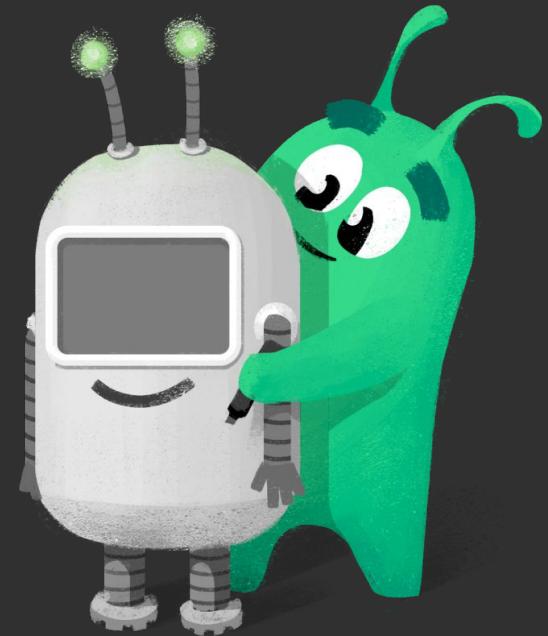
- Создать конструктор для `Repository`
- Создать заглушки для интерфейсов

3. В папке `services/contact/internal/useCase`:

- Создать конструктор для `UseCase`
- Создать заглушки для интерфейсов

4. Запустить проект

5. Залейте свою ветку в `git`



Демонстрация

Инициализация слоёв на main (20 мин)

Ветка с примером прошлого задания: 007-day-1-layers-constructor

Цель: Подключение слоёв в main

1. Создайте свою ветку в git из ветки kolyadkons-07.
Правила наименования **ФамилияИнициалы-08**. Например, kolyadkons-08
2. В папке pkg/
 - Подключение базы Postgres ([pgxpool](#))
3. В папке services/contact/cmd/app:
 - Добавить в main вызовы созданных конструкторов и запустить вечный цикл для прослушивания REST запросов
4. Запустить проект
5. Залейте свою ветку в git



Демонстрация

Вопросы

