

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

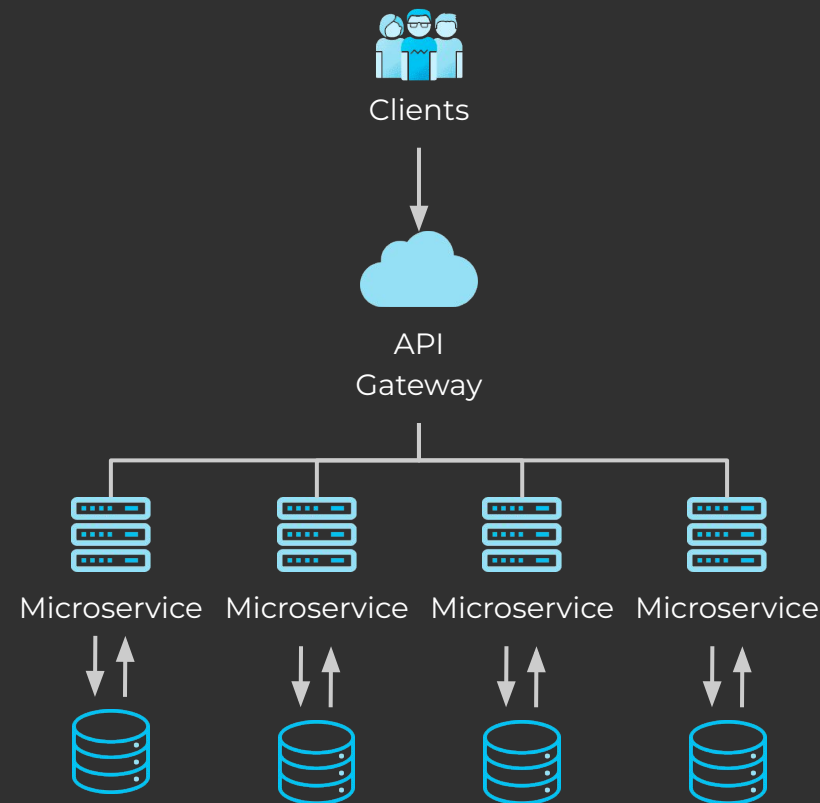
Что вы получите:

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

Что будет будет сделано:

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

Все практические задания связаны между собой

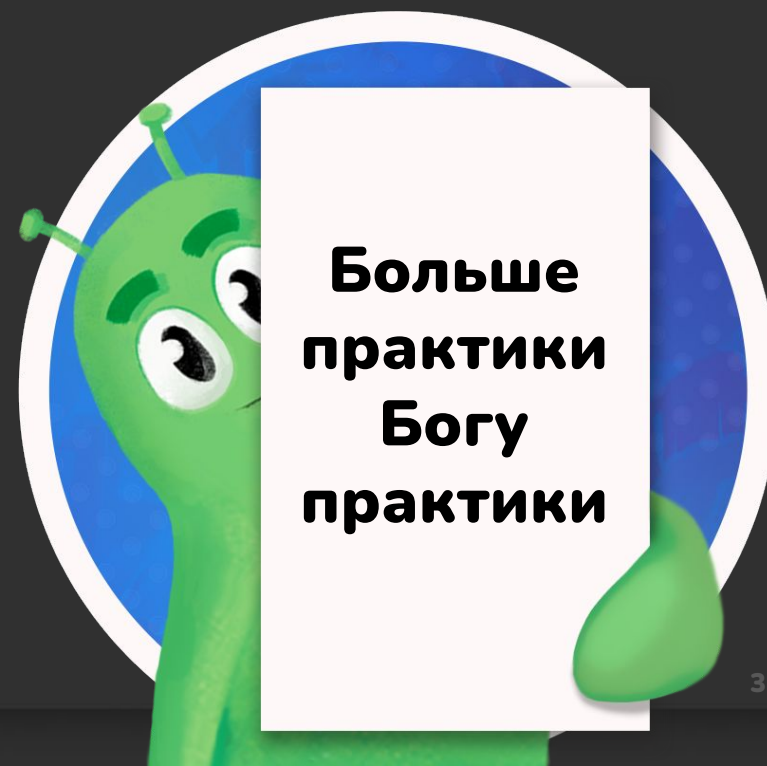


{REST API}

Вводная

Как будет проходить теория?

- Презентация, нацеленная на выполнение практики
- Минимальная информация для выполнения практики

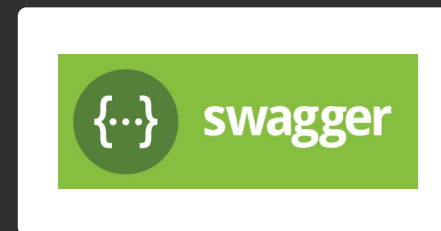


Стек технологий

Как будет проходить практика?

- **Требуются:**
 - IDE Goland
 - Swagger
 - GIT
- **Доступы к:**
 - GIT
 - Grafana
 - Jaeger
 - PostgreSQL

Все практические задания связаны между собой. Каждое новое практическое задание расширяет возможности предыдущего.



Таков путь...

День 1:

1. Создание структуры проекта

День 2:

2. Подключение к БД
3. Чистая архитектура:
 - 3.1. Структура папок по чистой архитектуре
 - 3.2. Наполнение – Domain
 - 3.3. Интерфейс – Use Case
 - 3.4. Интерфейс – Repository
 - 3.5. Конструкторы слоёв
 - 3.6. Инициализация слоёв на main
 - 3.7. Реализация интерфейса – Use Case

День 3:

- 3.8. Реализация интерфейса – Repository
- 3.9. Реализация – Delivery
4. Использование – Context
5. Добавить логи
6. Добавить трассировку
7. Тестирование**



Что будет в курсе?

Структура проекта

(теория и практика)

Observability

(теория и практика)

Чистая архитектура

(теория и практика)

Тестирование

(демонстрация)



3

Дня

8

Часов
теории

16

Часов
практики

Правила

Где задавать вопросы?

1. В чате Telegram

Где получать ответы?

1. В конце каждого дня или смыслового блока
2. В чате Telegram в течение прохождения интенсива

Как будут выбираться вопросы?

Будут выбраны самые интересные или часто задаваемые вопросы, ставьте реакции.



Структура проекта

Зачем нужна структура проекта?

Разбор основных папок /
Просмотр примеров проектов на Go

Иерархия папок

Зачем нужна
структура проекта?



Зачем нужна структура проекта

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

Что если:

делать, не думая о структуре?



Project

- bpmlntegrationService
- consul
- contactService
 - action
 - amd
 - call
 - campaign
 - chat
 - consul
 - grpc_connectors
 - helpers
 - internal
 - lib
 - cache
 - crm
 - database
 - calculation_datetime.go
 - call_helpers.go
 - call_manager.go
 - campaign_manager.go
 - campaign_manager_target.go
 - campaign_manager_target_test.go
 - chatbot_versions_manager.go
 - contact_helpers.go
 - contact_manager.go
 - contact_manager_attribute.go
 - contact_manager_attribute_test.go
 - contact_manager_import_info.go
 - contact_manager_markers.go
 - contact_manager_media_test.go
 - contact_manager_test.go
 - DB.go
 - ftp.go
 - group_helpers.go
 - group_manager.go
 - helpers.go
 - helpers_test.go
 - service.go
 - log
 - marker
 - proto
 - proto_build
 - redis

call_manager.go

1170

1179

1266

1267

1325

1326

1535

1536

1669

1670

1686

1687

1806

1807

1918

1919

1957

1958

1959

1980

1981

2044

2045

2049

2050

2139

2140

2296

2297

2327

2328

2337

2338

2375

func (c *CallManager) MakeCallLib

func (c *CallManager) ReadContact

func (c *CallManager) DeleteCall

func (c *CallManager) ReadHung

func (c *CallManager) ReadCall

func (c *CallManager) GetAudioF

func (c *CallManager) GetFileList(pathToFiles

func (c *CallManager) Recognize

type CallSetting struct {...}

func (c *CallManager) GetCallIn

func (c *CallManager) ReadCallS

func (c *CallManager) GetCallsE

type IntervalWhere struct {...}

func (c *CallManager) GetCalls

CallDateInfo.String() string

contact_manager.go

1109

1165

1166

1241

1242

1254

1255

1269

1270

1289

1290

1302

1303

1335

1336

1376

1377

1433

1434

1461

1462

1481

1482

1501

1502

1697

1698

1774

1775

1998

1999

2000

2025

func (c *ContactManager) GetScriptMarkers(c

func (c *ContactManager) AddCalledOutContac

func (c *ContactManager) ReadCalledOutConta

func (c *ContactManager) SetMarkerHTTP(ctx

func (c *ContactManager) GetMarkerHTTP(ctx

func (c *ContactManager) ReadUserApiSetting

func (c *ContactManager) CreateUserApiSett

func (c *ContactManager) GetLastCreatedUpda

func (c *ContactManager) GetCountContacts(c

func (c *ContactManager) existAMD(ctx robo

func (c *ContactManager) scriptIsPublished(c

func (c *ContactManager) CreateInvoke(c co

func (c *ContactManager) CheckContactIds(ct

func checkContactAttributes(attributes *pro

func (c *ContactManager) GetLastCreatedUpda

func (c *ContactManager) GetCountContacts(c

func (c *ContactManager) existAMD(ctx robo

func (c *ContactManager) scriptIsPublished(c

func (c *ContactManager) CreateInvoke(c co

func (c *ContactManager) CheckContactIds(ct

func checkContactAttributes(attributes *pro

func (c *ContactManager) GetLastCreatedUpda

func (c *ContactManager) GetCountContacts(c

func (c *ContactManager) existAMD(ctx robo

func (c *ContactManager) scriptIsPublished(c

func (c *ContactManager) CreateInvoke(c co

func (c *ContactManager) CheckContactIds(ct

func checkContactAttributes(attributes *pro

campaign_manager.go

1882

1883

2030

2031

2070

2071

2090

2091

2232

2233

2234

2274

2275

2330

2331

2448

2449

2494

2495

2591

2592

2700

2701

2727

2728

2803

2804

2837

2838

2907

2908

3016

3017

3049

func (c *CampaignManager) GetContact

func (c *CampaignManager) GetCallIds

type DialogHistory struct {...}

func (c *CampaignManager) ReadDialog

func (c *CampaignManager) readLastCal

func (c *CampaignManager) handleDial

func (c *CampaignManager) ReadDialog

func (c *CampaignManager) handleShor

func (c *CampaignManager) CopyContac

func addContactToCampaignByGroup(tx

func (c *CampaignManager) GetFileNam

func (c *CampaignManager) GetLastCra

func (c *CampaignManager) GetCountC

func (c *CampaignManager) ReadStat

func (c *CampaignManager) StartCampa

func (c *CampaignManager) StopCampa

func (c *CampaignManager) ReadStat(ctx context.Cont

contact_manager_markers.go

1

2

3

25

26

27

214

215

290

291

376

377

431

432

433

476

477

548

549

792

793

794

795

805

806

929

930

968

969

979

980

1014

1015

1080

package lib

import ...

// Deprecated: Moved to scriptService

func (c *ContactManager) CreateMarke

func (c *ContactManager) ReadMarker

func (c *ContactManager) UpdateMarke

func (c *ContactManager) DeleteMarke

// Deprecated: Moved to scriptService

func (c *ContactManager) GetMarkerTY

func (c *ContactManager) GetSystemCo

func (c *ContactManager) GetMarkers

type HashMarkers map[string]*proto.M

func (m HashMarkers) String() string

func (c *ContactManager) FillMarkers

func GetColumnAndTableName(ctx conte

func CheckColumnAndTableName(ctx con

func CheckColumnTableSchemaName(colu

func (c *ContactManager) ReadUMarker

func (c *ContactManager) CreateMarker(ctx context.Co

```
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
```

Демонстрация плохого кода

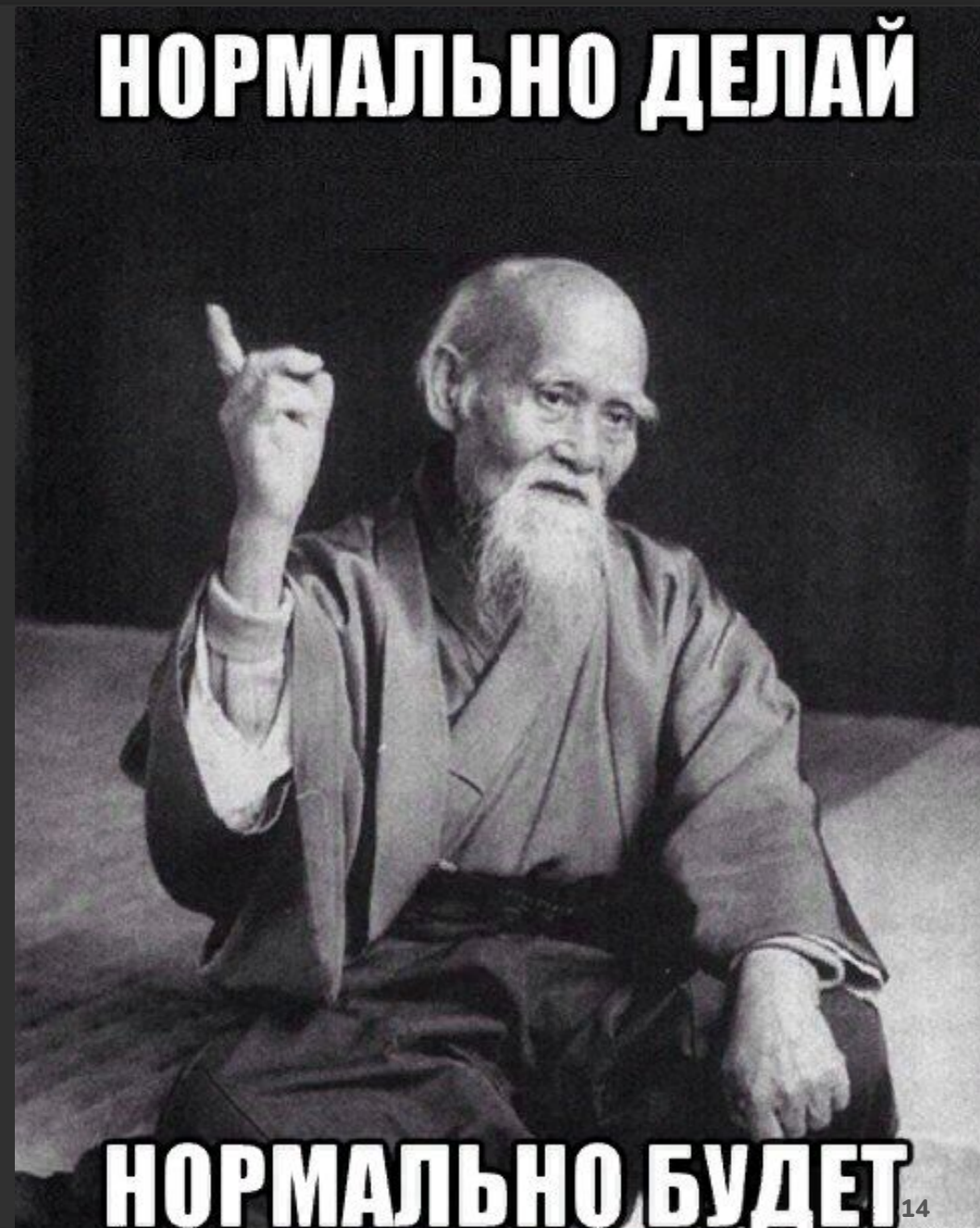
(Ветка: 001_bad_code)



А как нужно ?

Цели структуры проекта:

1. Ускорение разработки
2. Отражение целей проекта



Разбор основных папок



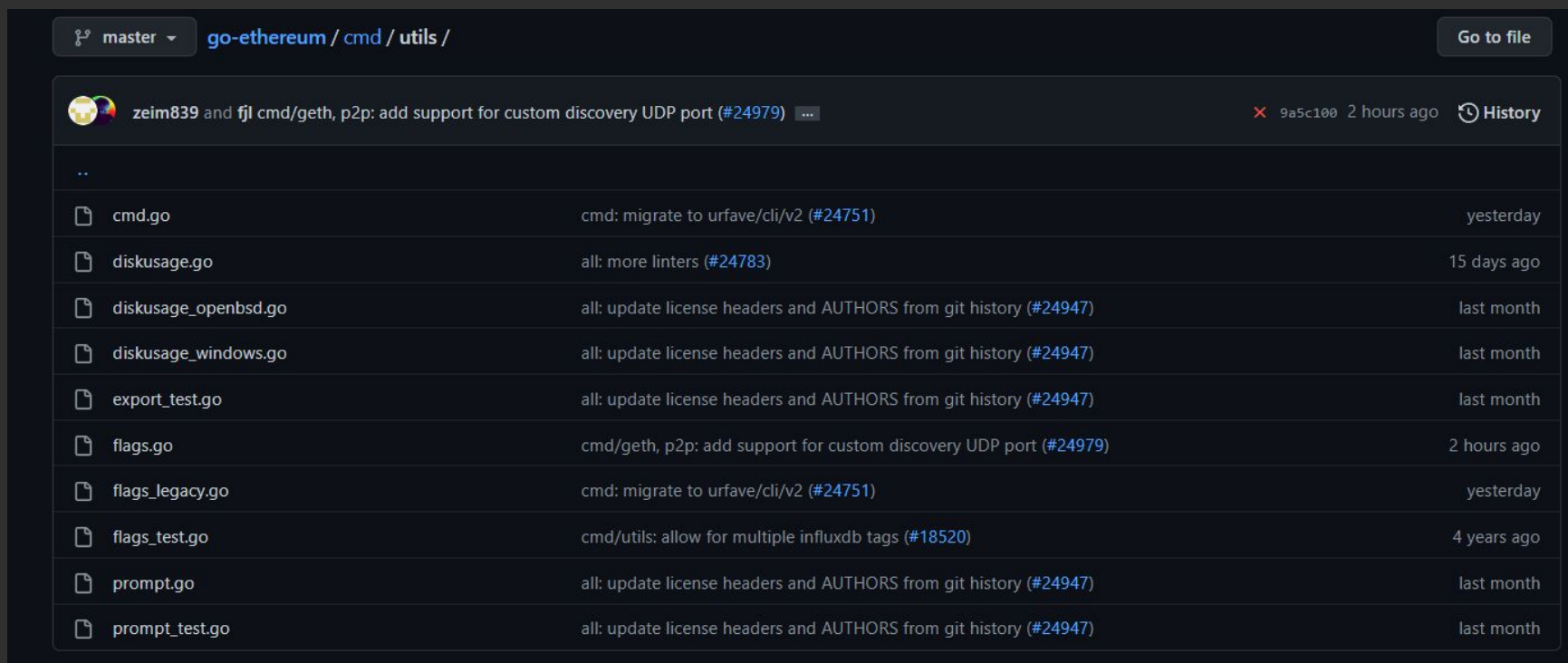
Разбор основных папок

- **cmd** – Основные приложения проекта
- **internal** – Внутренний код приложения и библиотек
- **pkg** – Код библиотек
- **vendor** – Зависимости приложений
- **docs / api** – Документация
- **deployments / deploy** – Шаблоны и файлы конфигураций
- **configs** – Шаблоны файлов конфигураций и файлы настроек по-умолчанию

Дополнительную информацию можно получить в github.com

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

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



The screenshot shows the GitHub interface for the repository 'go-ethereum / cmd / utils'. At the top, there's a navigation bar with 'master' selected and a 'Go to file' button. Below this, a commit message is displayed: 'zeim839 and fjl cmd/geth, p2p: add support for custom discovery UDP port (#24979)' with a commit hash '9a5c100' and a timestamp '2 hours ago'. A 'History' button is also present. The main part of the image is a table listing recent commits.

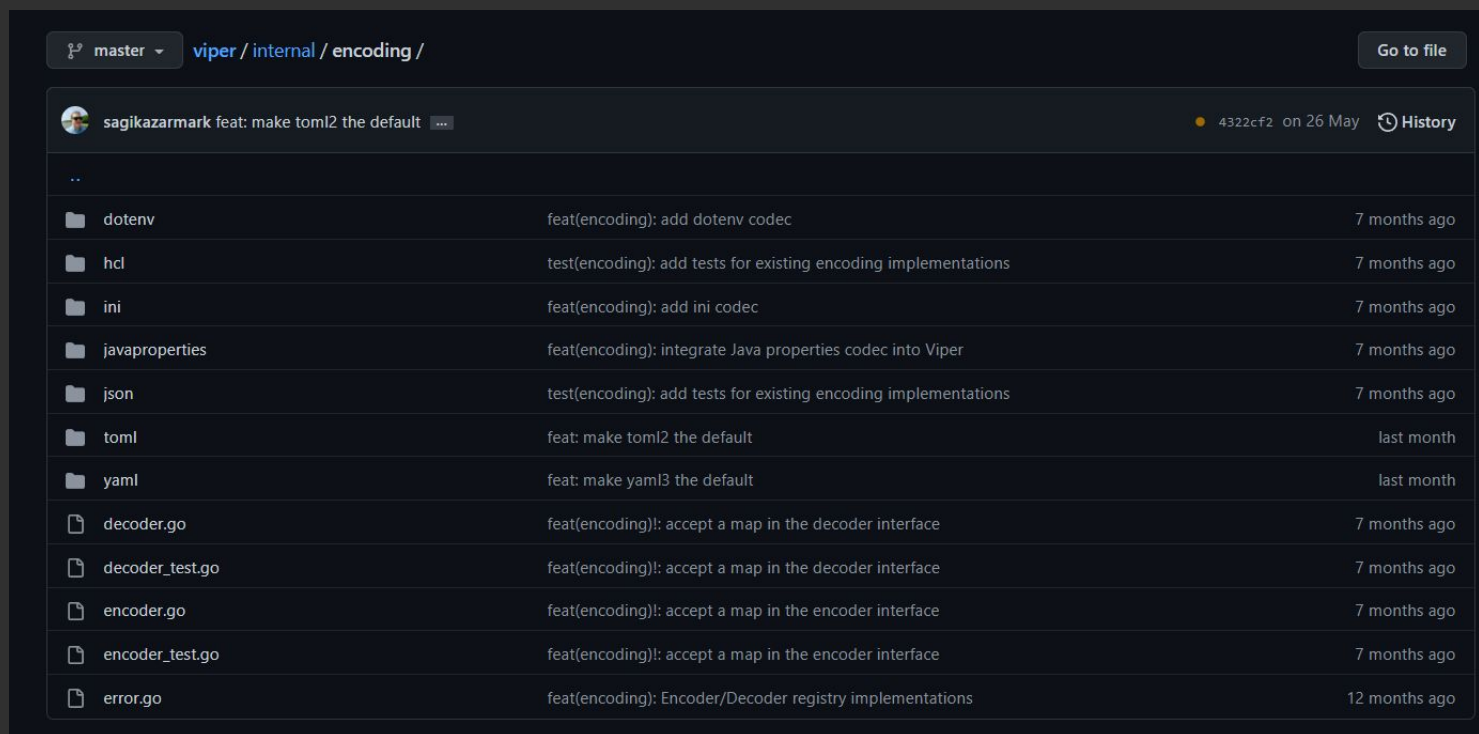
File	Commit Message	Time
..		
cmd.go	cmd: migrate to urfave/cli/v2 (#24751)	yesterday
diskusage.go	all: more linters (#24783)	15 days ago
diskusage_openbsd.go	all: update license headers and AUTHORS from git history (#24947)	last month
diskusage_windows.go	all: update license headers and AUTHORS from 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

Это код, который не должен быть применен в других приложениях и библиотеках ([git](#)).

Стоит отметить, что этот шаблон навязан самим компилятором Golang.

Ознакомьтесь с [release notes](#) Go 1.4



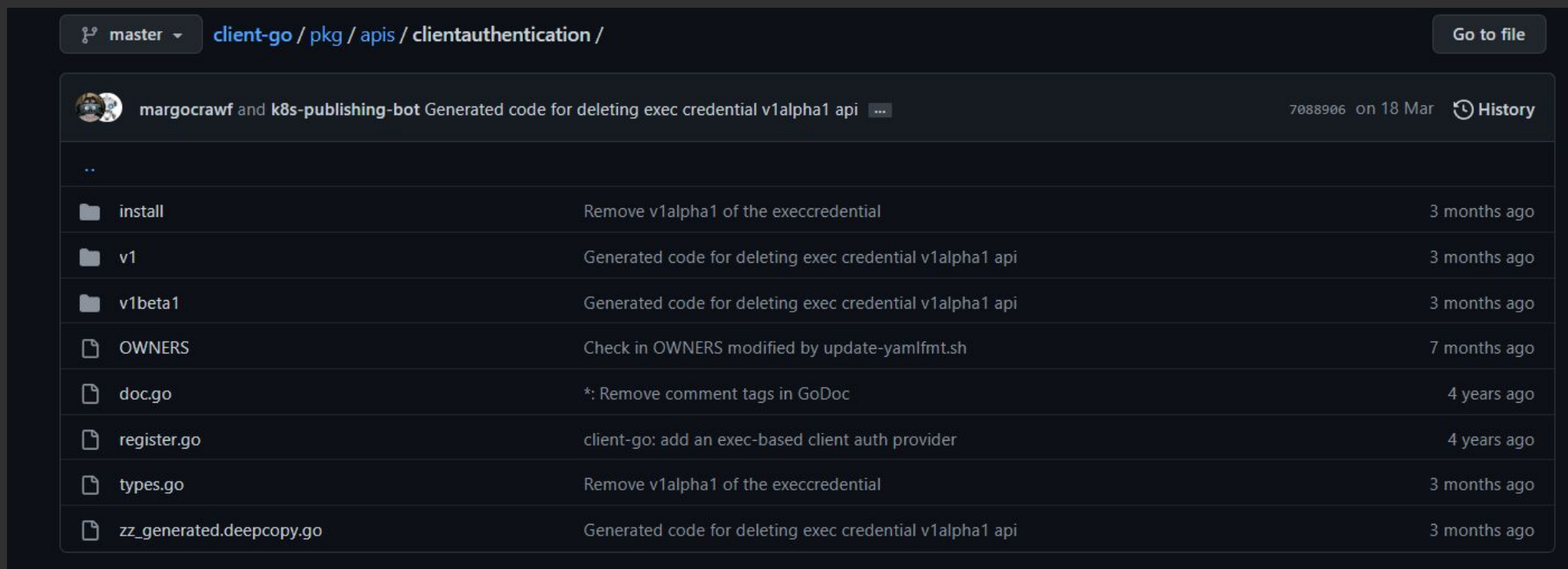
The screenshot shows the GitHub interface for the repository 'viper/internal/encoding'. At the top, there's a breadcrumb 'viper / internal / encoding /' and a 'Go to file' button. Below this, a commit summary for 'sagikazarmark feat: make toml2 the default' is shown, including a commit hash '4322cf2' and the date 'on 26 May'. The main part of the image is a table listing recent commits and their associated files.

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

pkg – Код библиотек

Это код, который пригодный для использования в сторонних приложениях ([git](#))

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



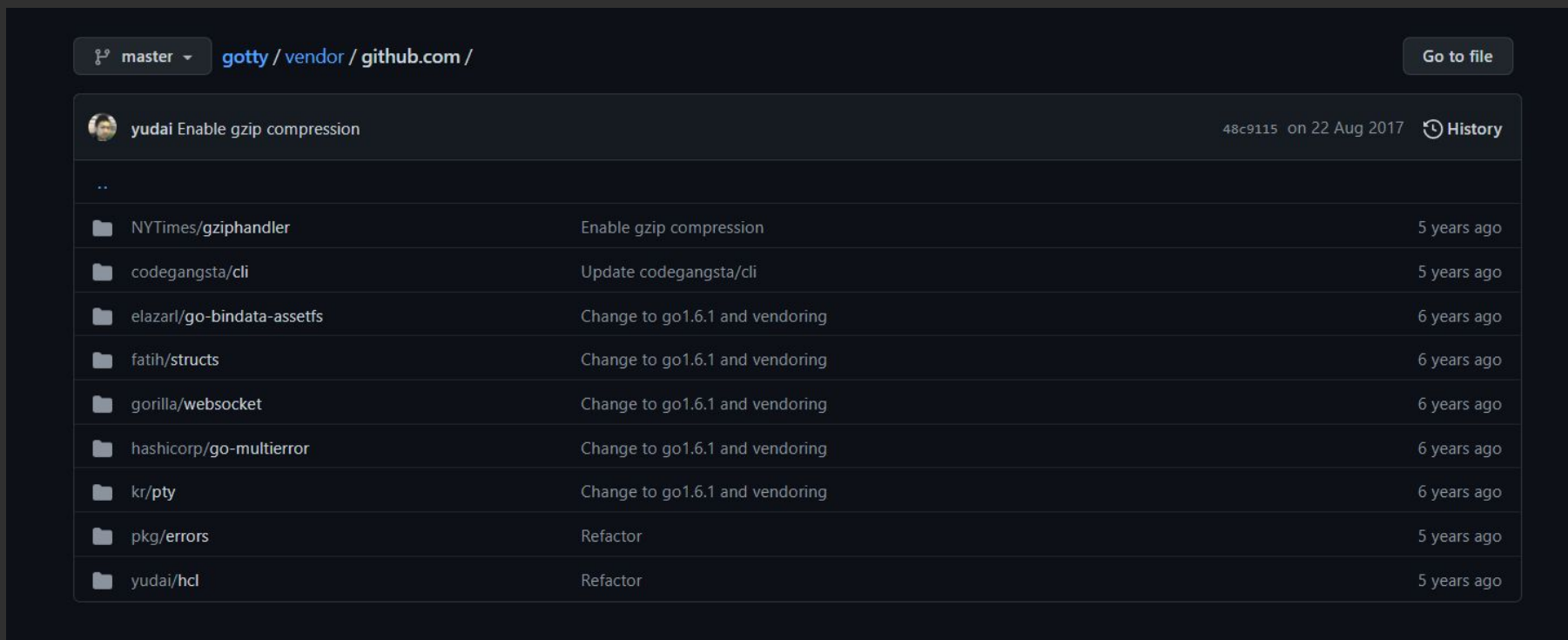
The screenshot shows a GitHub repository page for the path `client-go / pkg / apis / clientauthentication /`. The commit history table lists the following entries:

File	Commit Message	Time Ago
..		
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

vendor – Зависимости приложений

Управляемый вручную или с использованием ([git](#))

Например, использовать go mod



master gotty / vendor / github.com / Go to file

yudai Enable gzip compression 48c9115 on 22 Aug 2017 History

..		
NYTimes/gziphandler	Enable gzip compression	5 years ago
codegangsta/cli	Update codegangsta/cli	5 years ago
elazarl/go-bindata-assetsfs	Change to go1.6.1 and vendoring	6 years ago
fatih/structs	Change to go1.6.1 and vendoring	6 years ago
gorilla/websocket	Change to go1.6.1 and vendoring	6 years ago
hashicorp/go-multierror	Change to go1.6.1 and vendoring	6 years ago
kr/pty	Change to go1.6.1 and vendoring	6 years ago
pkg/errors	Refactor	5 years ago
yudai/hcl	Refactor	5 years ago

docs / api – Документация

- **docs** – Проектные и пользовательские документы ([git](#))
- **api** – Спецификации OpenAPI/Swagger, файлы схем JSON, файлы определения протоколов. ([git](#))

The screenshot shows a GitHub file view for the file `2021-08-22-split-postmortem.md` in the `docs/postmortems` directory of the `go-ethereum` repository. The file is 11.1 KB and contains 266 lines of text. The content of the file is as follows:

Minority split 2021-08-27 post mortem

This is a post-mortem concerning the minority split that occurred on Ethereum mainnet on block 13107518, at which a minority chain split occurred.

Timeline

- 2021-08-17: Guido Vranken submitted a bounty report. Investigation started, root cause identified, patch variations discussed.
- 2021-08-18: Made public announcement over twitter about upcoming security release upcoming Tuesday. Downstream projects were also notified about the upcoming patch-release.
- 2021-08-24: Released v1.10.8 containing the fix on Tuesday morning (CET). Erigon released v2021.08.04.
- 2021-08-27: At 12:50:07 UTC, issue exploited. Analysis started roughly 30m later,

Bounty report

2021-08-17 RETURNDATA corruption via datacopy

On 2021-08-17, Guido Vranken submitted a report to bounty@ethereum.org. This coincided with a geth-meetup in Berlin, so the geth team could fairly quickly analyse the issue.

He submitted a proof of concept which called the `dataCopy` precompile, where the input slice and output slice were overlapping but shifted. Doing a `copy` where the `src` and `dest` overlaps is not a problem in itself, however, the `returnData` slice was *also* using the same memory as a backing-array.

Technical details

During CALL-variants, `geth` does not copy the input. This was changed at one point, to avoid a DoS attack reported by Hubert Ritzdorf, to

The screenshot shows a GitHub file view for the file `message.pb.go` in the `api/internal/proto` directory of the `go-micro` repository. The file is 412 Bytes and contains 28 lines of code. The content of the file is as follows:

```
1 package proto
2
3 type Message struct {
4     data []byte
5 }
6
7 func (m *Message) ProtoMessage() {}
8
9 func (m *Message) Reset() {
10     *m = Message{}
11 }
12
13 func (m *Message) String() string {
14     return string(m.data)
15 }
16
17 func (m *Message) Marshal() ([]byte, error) {
18     return m.data, nil
19 }
20
21 func (m *Message) Unmarshal(data []byte) error {
22     m.data = data
23     return nil
24 }
25
26 func NewMessage(data []byte) *Message {
27     return &Message{data}
28 }
```

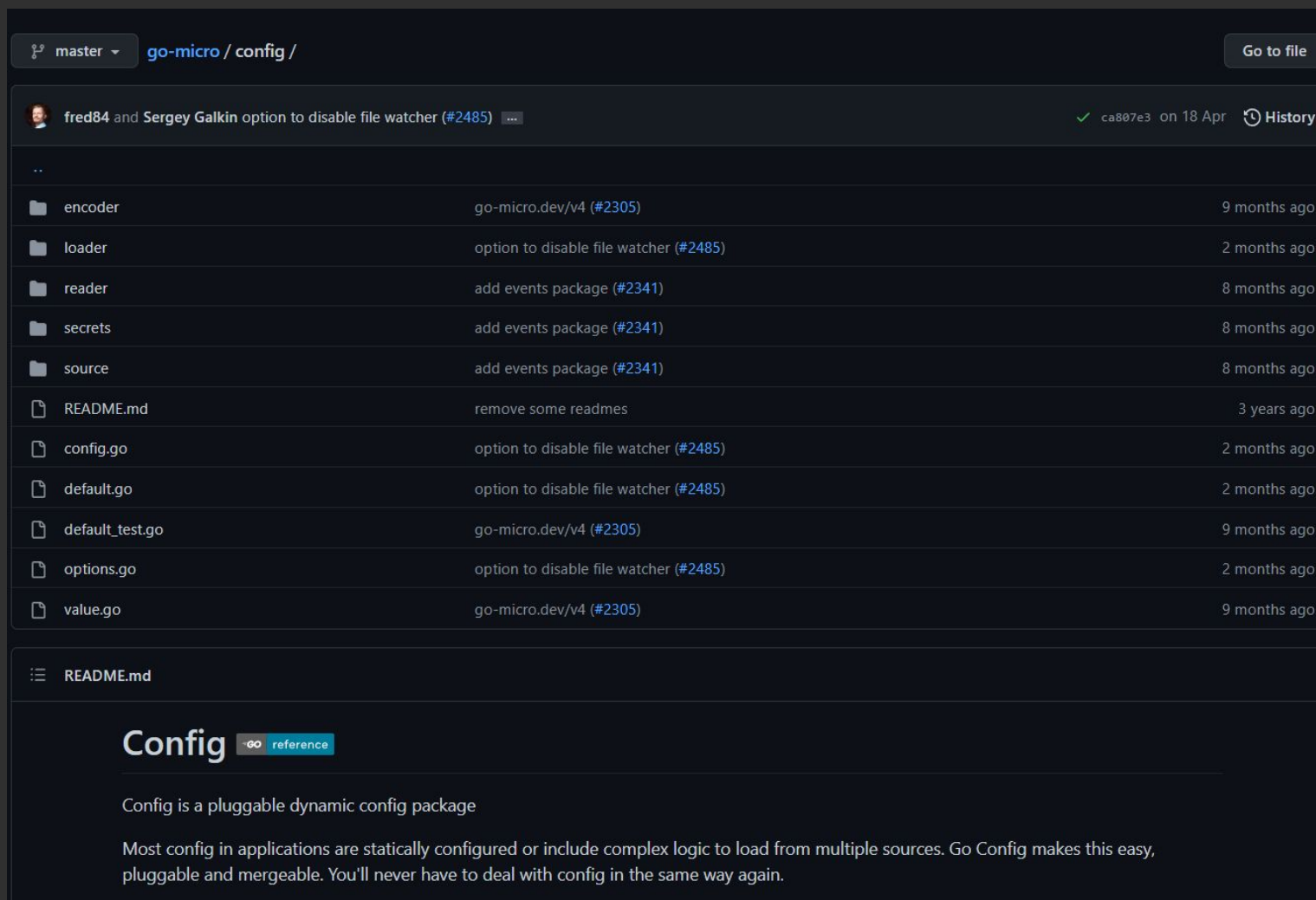
deployments / deploy – Шаблоны и файлы конфигураций

(docker-compose, kubernetes/helm, mesos, terraform, bosh)

- ▼ deploy
 - ▼ docker-compose
 - > application
 - > infrastructure
 - > dockerfiles
 - ▼ gitlab
 - > jobs
 - > script_ci
 - > Makefile
 - > protobuf
 - > scripts

```
service.yml
1  version: "3.7"
2
3  networks:
4    net:
5      driver: overlay
6
7  services:
8    contact:
9      image: ${tag_contactservice}
10     deploy:
11       mode: replicated
12       placement: <1 key>
13       replicas: ${REPLICA}
14       resources:
15         limits:
16           cpus: "0.25"
17       restart_policy: <3 keys>
18       update_config: <3 keys>
19       labels: <13 items>
20       volumes: <4 items>
21       extra_hosts:
22         - "speechcat:${SPEECHCAT_IP}"
23       healthcheck: <5 keys>
24       environment: <7 keys>
25       logging: <2 keys>
26     networks:
27       - net
```

configs – Шаблоны файлов конфигураций и файлы настроек по-умолчанию ([git](#))



The screenshot displays the GitHub interface for the `go-micro / config` repository. At the top, the branch `master` is selected, and the path `go-micro / config /` is shown. A `Go to file` button is present. Below the repository name, a commit by `fred84` and `Sergey Galkin` is highlighted, titled "option to disable file watcher (#2485)".

The file list includes:

- `..`
- `encoder` (go-micro.dev/v4 (#2305), 9 months ago)
- `loader` (option to disable file watcher (#2485), 2 months ago)
- `reader` (add events package (#2341), 8 months ago)
- `secrets` (add events package (#2341), 8 months ago)
- `source` (add events package (#2341), 8 months ago)
- `README.md` (remove some readmes, 3 years ago)
- `config.go` (option to disable file watcher (#2485), 2 months ago)
- `default.go` (option to disable file watcher (#2485), 2 months ago)
- `default_test.go` (go-micro.dev/v4 (#2305), 9 months ago)
- `options.go` (option to disable file watcher (#2485), 2 months ago)
- `value.go` (go-micro.dev/v4 (#2305), 9 months ago)

The `README.md` file is expanded, showing the following content:

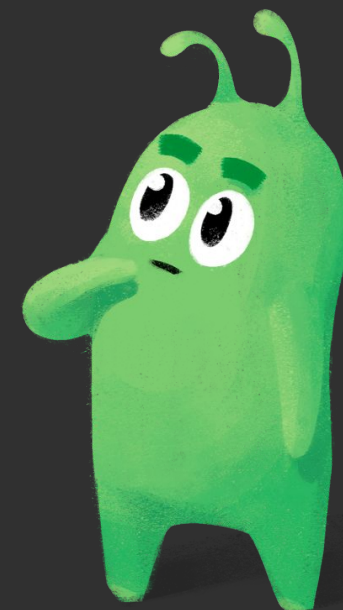
Config go reference

Config is a pluggable dynamic config package

Most config in applications are statically configured or include complex logic to load from multiple sources. Go Config makes this easy, pluggable and mergeable. You'll never have to deal with config in the same way again.

Ответьте на вопросы

- *Как у вас организована структура?*
- *Проблемы при работе с структурой проекта?*
- *Есть вопросы по папкам?*
- *Есть шаблон при создании нового проекта?*



ИТОГИ

Что делать если у вас особый случай?

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

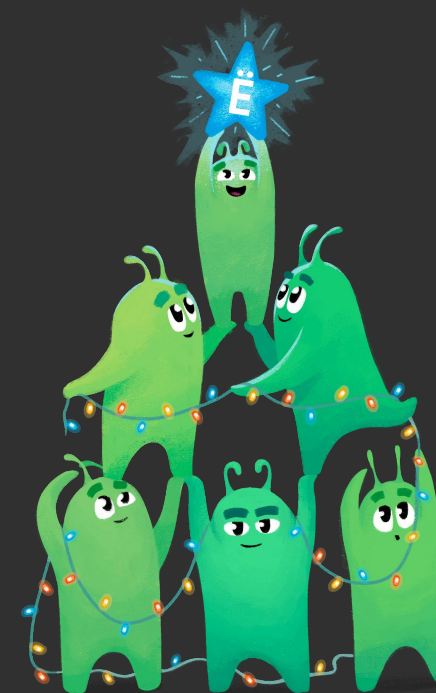


Перерыв 10 мин



Иерархия папок

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



Практика: Создание структуры проекта (40 мин)

Цель: Подключиться к [git](#) и создать базовый проект для дальнейшей работы с ним

Задание:

1. Подключиться к [git](#) Слёрм
 - a. Подключиться по адресу <https://gitlab.slurm.io/a.iskakov/forgoproject>
 - b. Доступы для git можно получить в личном кабинете Слёрм
 - c. Создайте свою ветку в git. Правила наименования [ФамилияИнициалы-01](#)
Например, [kolyadkons-01](#)
2. Создать примитивную структуру проекта
 - a. Создать папку pkg
 - b. Создать папку services/contact/cmd
 - i. Создать пакет app
 - ii. Создать main функцию с выводом в консоль
3. Запустить проект
4. Залейте свою ветку в git

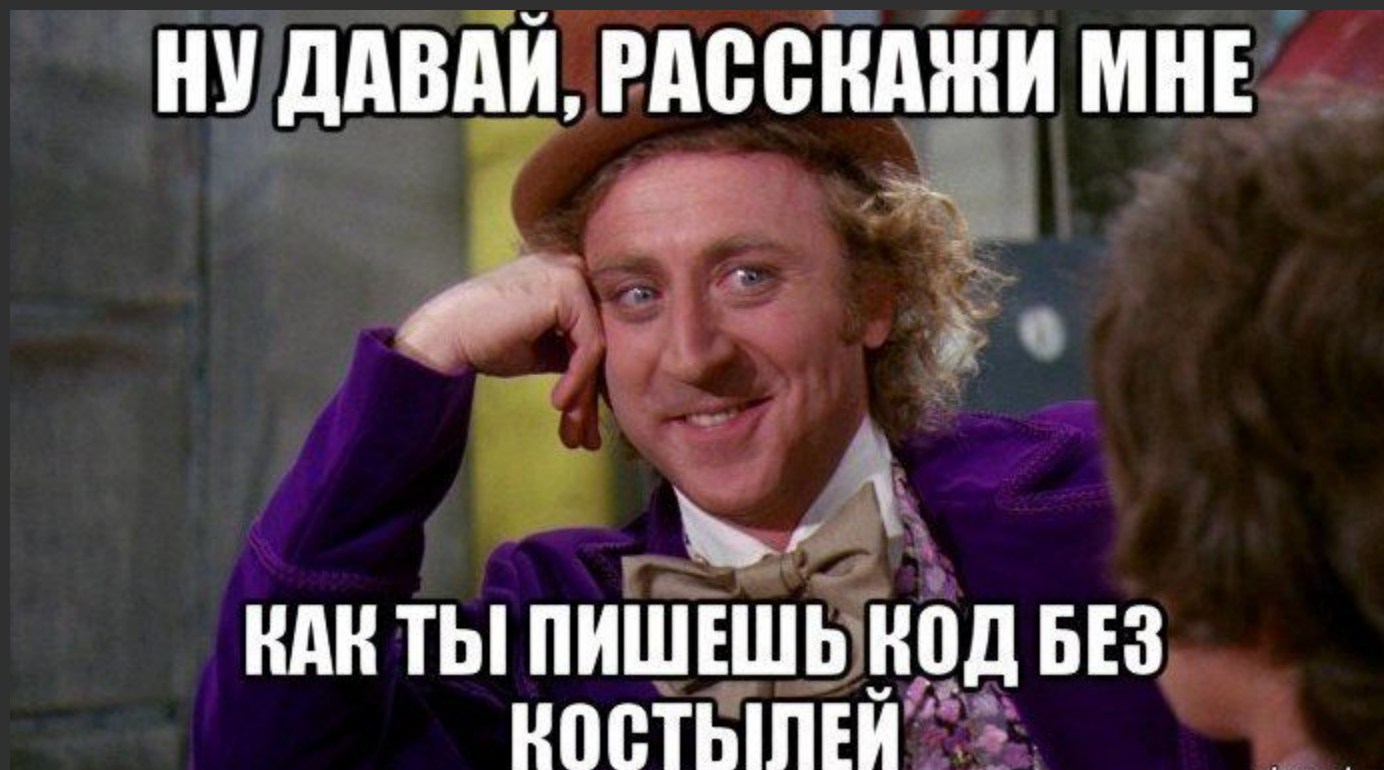
Результаты

Успели ли выполнить задание? (опрос)

- ☐ Да
- ☐ Успел на 75%
- ☐ Успел на 50%
- ☐ Успел на 25%
- ☐ Нет



Демонстрация



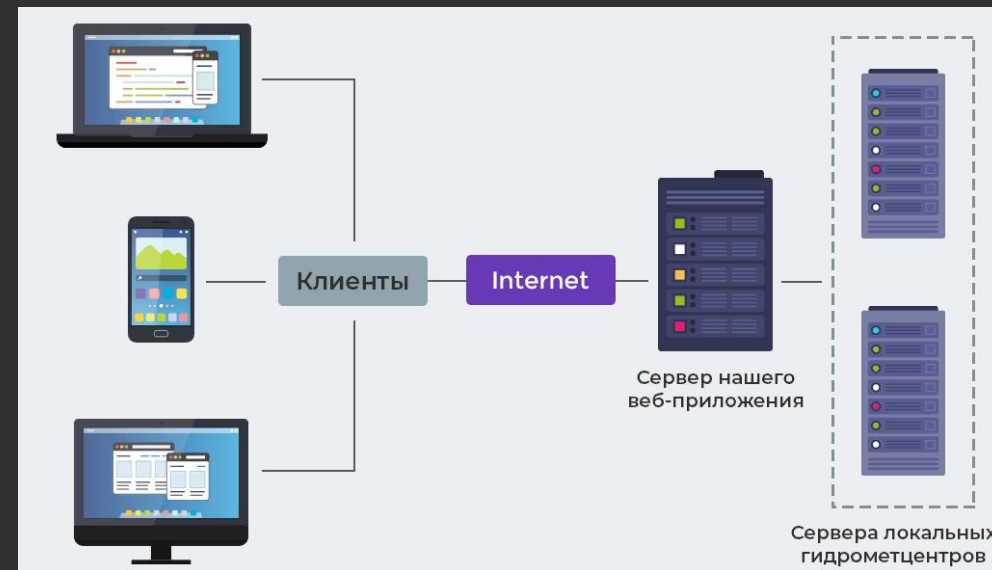
Обед 60 мин

Чистая архитектура

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



Что такое архитектура?



Архитектура - Компромисс

Блиц-опрос

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?

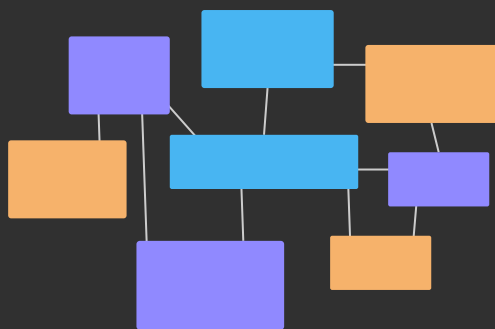


Low coupling, high cohesion

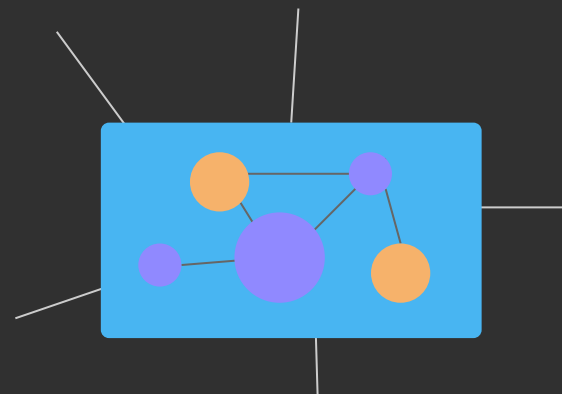
Качественный дизайн архитектуры обладает:

- **Low Coupling** – слабое связывание
- **High Cohesion** – высокое сцепления

Это значит, что программный компонент имеет мало внешних связей и отвечает за решение близких по смыслу задач



Coupling

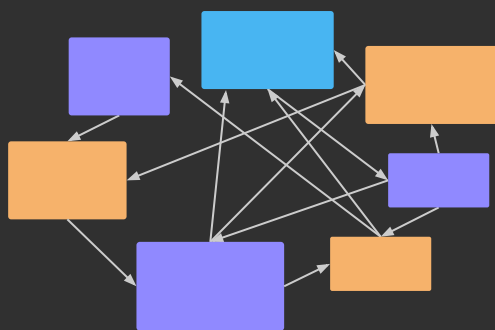


Cohesion

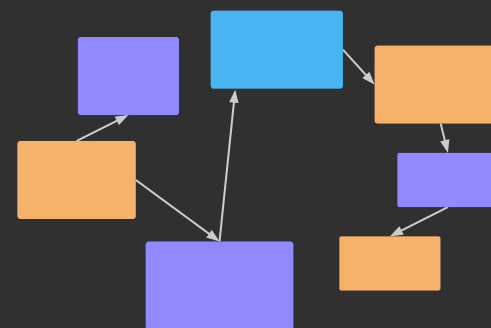
Coupling – Связанность

Coupling – насколько взаимозависимы разные подпрограммы или модули

- **Low coupling** – соответствует общим показателям хорошей читаемости и сопровождаемости
 - Малое число зависимостей между классами / подсистемами
 - Слабая зависимость одного класса / подсистемы от изменений в другом классе / подсистеме
 - Высокая степень повторного использования подсистем
- **High coupling** – затрудняет понимание логики модулей, их модификацию, автономное тестирование, а также переиспользование по отдельности



High Coupling



Low Coupling

Coupling – Связанность

- **Content – содержимого**

- Модуль изменяет / полагается на внутренние особенности другого модуля
- Изменение работы второго модуля приведет к переписыванию первого

- **Common – через общее**

- Два модуля работают с общими данными (глобальной переменной)
- Изменение разделяемого ресурса приведет к изменению всех работающих с ним модулей

- **External – через внешнее**

- Два модуля используют навязанный извне формат данных (протокол связи)

- **Control – по управлению**

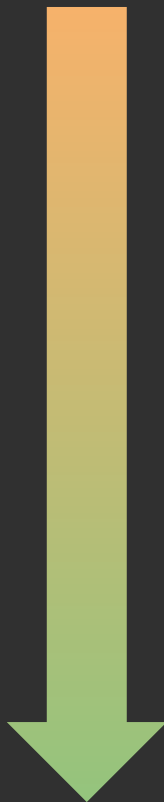
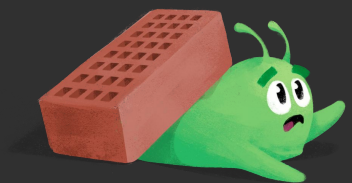
- Один модуль управляет поведением другого
- Присутствует передача информации о том, что и как делать

Coupling – Связанность

- **Data-structured – структурированным по данным**
 - Модули используют одну и ту же структуру, но каждый использует только ее части
 - Изменение структуры может привести к изменению модуля
- **Data – через данных**
 - Модули совместно используют данные, например, через параметры
- **Message – по сообщениям**
 - Модули общаются только через передачу параметров или сообщений
- **No coupling – отсутствие связанности**
 - Модули вообще никак не взаимодействуют

Coupling – Связанность

Типы моделей связанности



Содержимого

Через общее

Через внешнее

По управлению

Структурированным по данным

Через данных

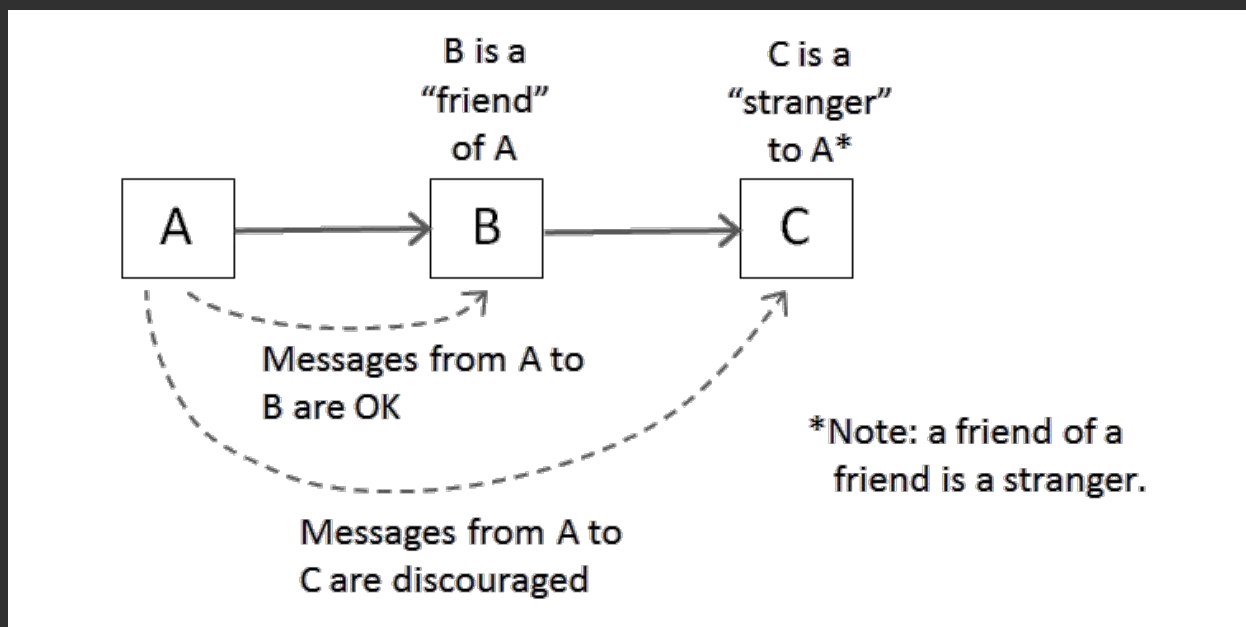
По сообщениям

Отсутствие связанности

Принцип наименьшего знания (Закон Деметры)

Объект A не должен иметь возможность получить непосредственный доступ к объекту C, если у объекта A есть доступ к объекту B и у объекта B есть доступ к объекту C

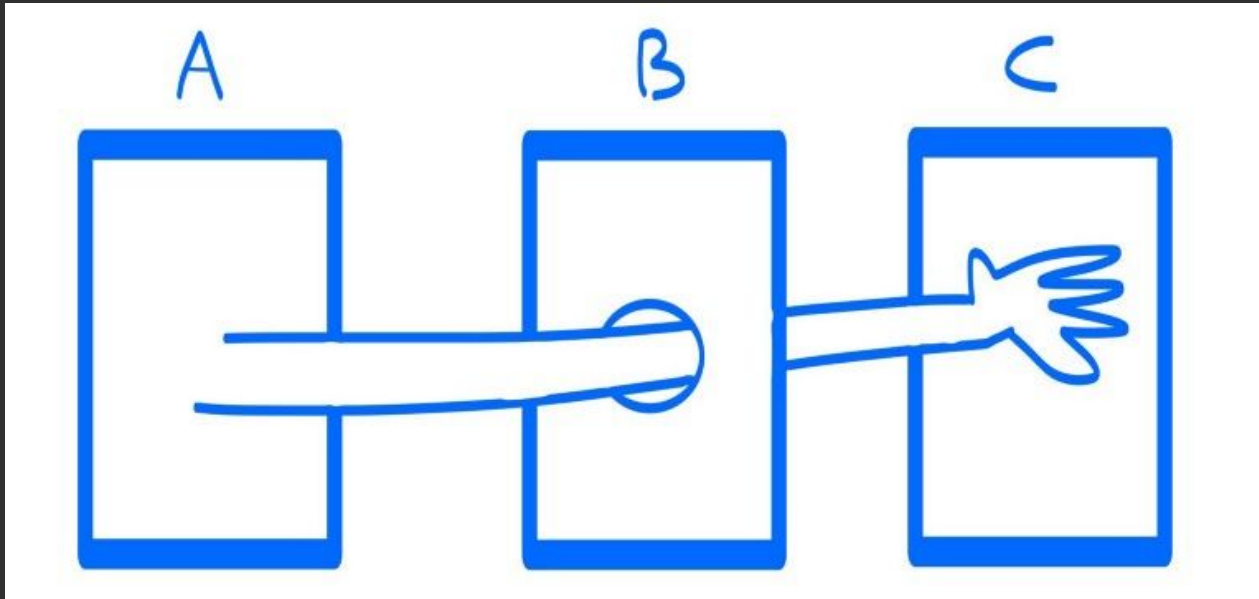
- Должен знать о модулях, которые «непосредственно» относятся к этому модулю
- Обращаться только к непосредственным «друзьям»
- Не взаимодействовать с незнакомцами



Принцип наименьшего знания (Закон Деметры)

Преимущества – код, разработанный с соблюдением данного закона, делает написание тестов более простым, а разработанное ПО менее сложно при поддержке и имеет большие возможности повторного использования кода

Недостатки – требуется создание большого количества малых методов-адаптеров для передачи вызовов метода к внутренним компонентам



Cohesion – Сцепление

Cohesion – насколько сильно взаимосвязаны элементов внутри модуля

- **Low cohesion** – много разнородных функций или несвязанных между собой обязанностей
 - Трудность понимания
 - Сложность при повторном использовании
 - Сложность поддержки
 - Ненадежность, постоянная подверженность изменениям
- **High cohesion** – обязанности хорошо согласованы между собой и он не выполняет огромных объемов работы

Cohesion – Сцепление

- **Coincidental – случайная**

- Части модуля сгруппированы “от фонаря”
- Единственное, что их объединяет — сам модуль

- **Logical – логическая**

- Части модуля логически относятся к одной проблеме
- Части могут различаться по своей природе

- **Temporal – временная**

- Части модуля обычно используются в программе в одно время, рядом

- **Procedural – процедурная**

- Части модуля всегда используются в определенном порядке

Cohesion – Сцепление

- **Communication – по взаимодействию**

- Части модуля работают над одним и теми же данными

- **Sequential – по последовательности действий**

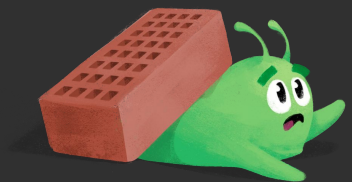
- Результат работы одной части модуля является исходными данными для другой

- **Functional – функциональная**

- Части модуля направлены на решение одной четкой задачи, за которую отвечает модуль

Cohesion – Сцепление

Типы моделей сцепления



Случайная

Логическая

Временная

Процедурная

По взаимодействию

По последовательности действий

Функциональная



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

Независимость от сторонних систем

Фреймворк – программная платформа

Kit – набор пакетов, которые обеспечивают комплексный, надежный и создания микросервисов для организаций любого размера



Gin – REST API для серверной части

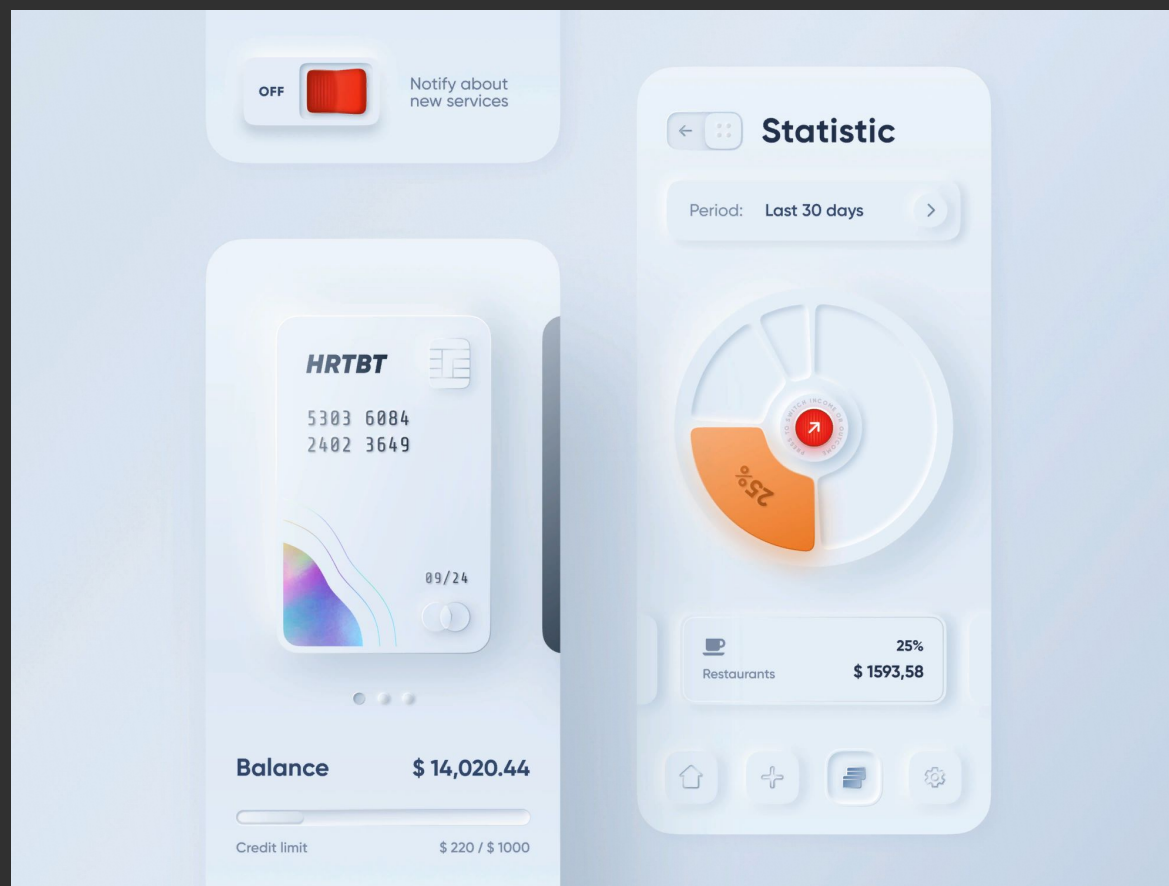


Go Gin
Web Framework

Независимость от сторонних систем

UI – элементы продукта,
делающих его притягательным:

- цвет
- стиль кнопок
- графика
- анимация
- типографика
- инфографика
- виджеты
- поведение
- отклики кнопок
- и так далее



Независимость от сторонних систем

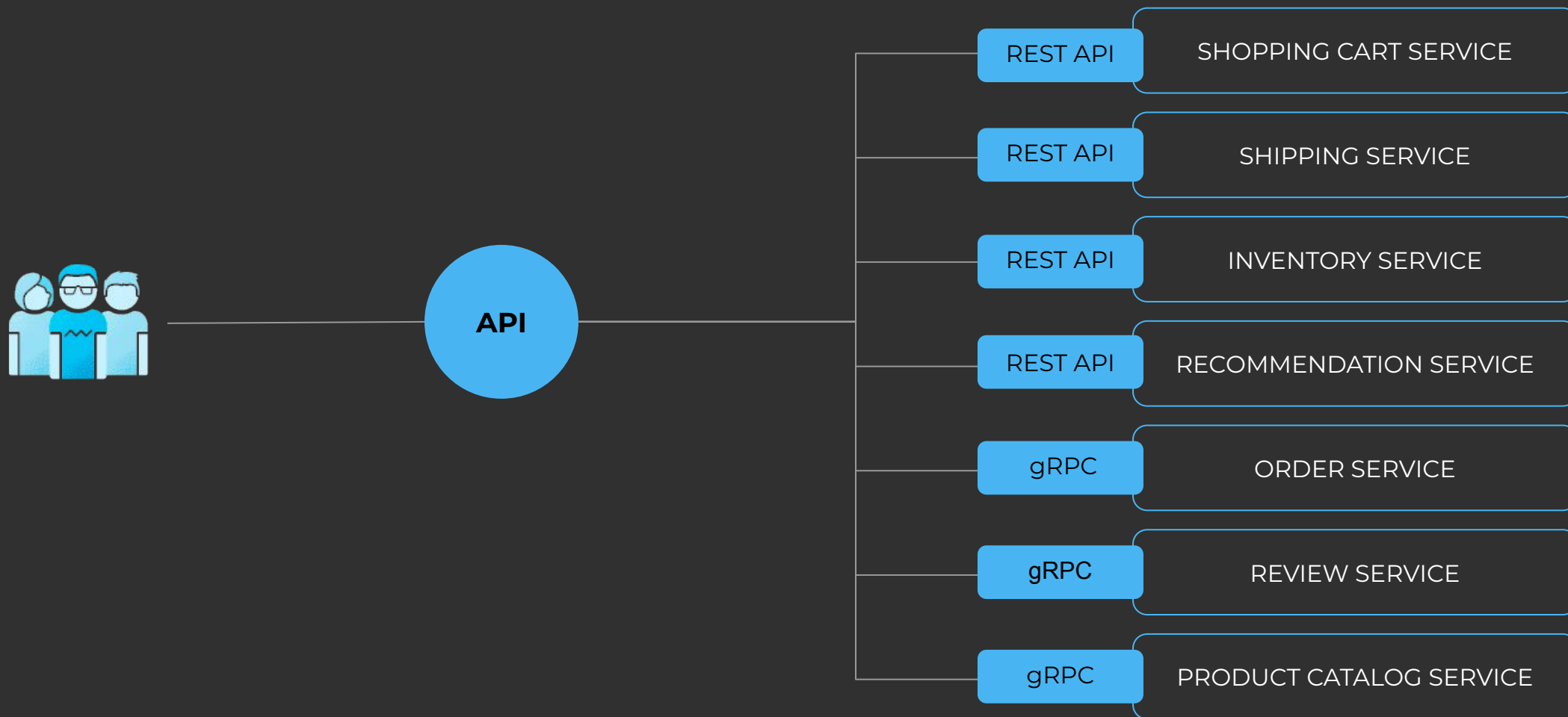
Базы данных – совокупность данных, хранимых в соответствии со схемой данных

- PostgreSQL
- ClickHouse
- MySQL
- Redis



Независимость от сторонних систем

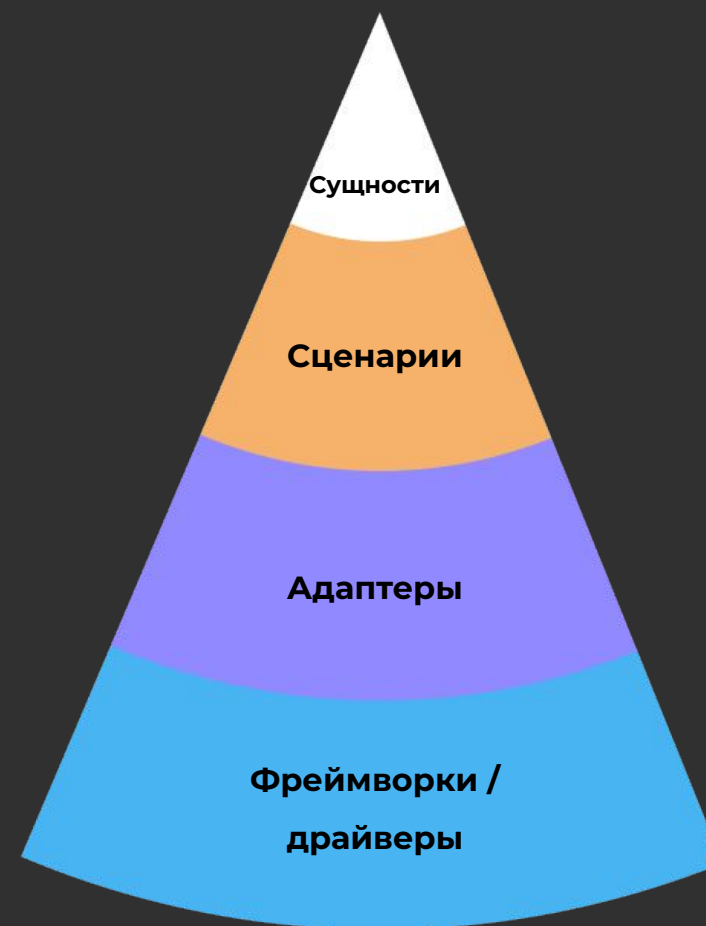
Внешних сервисов – любые сторонние системы



Перерыв 10 мин

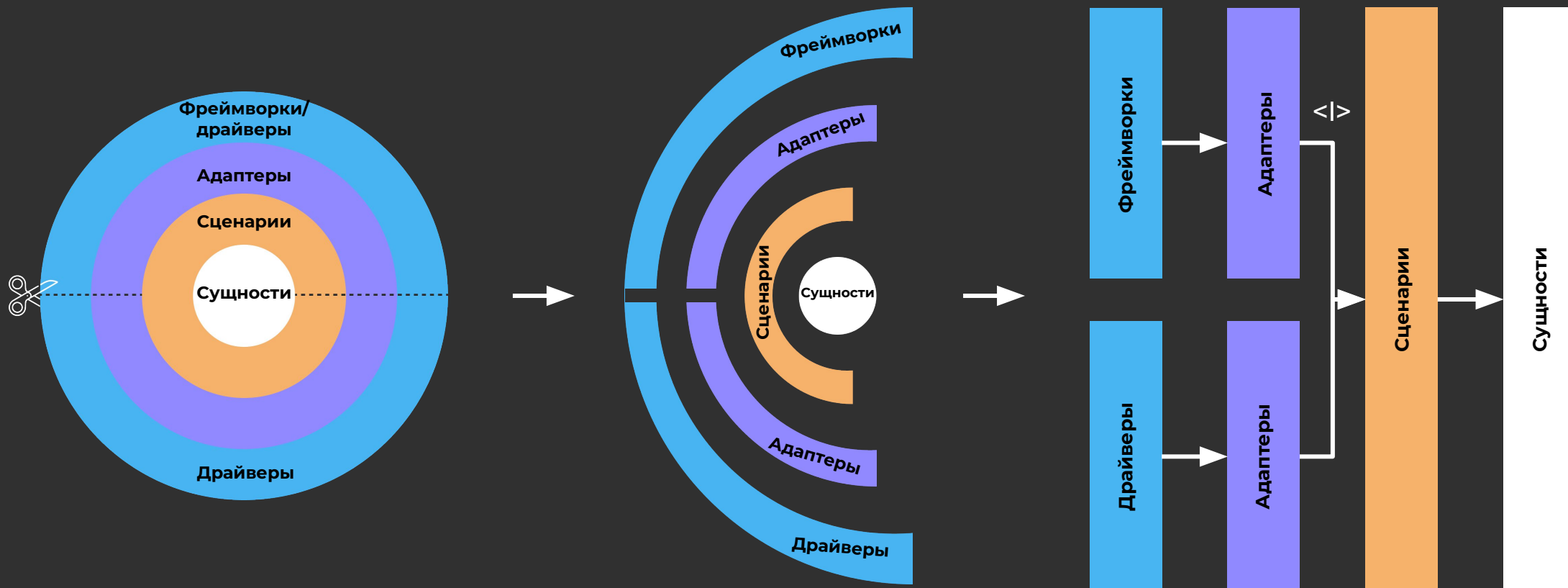
Слои чистой архитектуры

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



Правило зависимостей

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



Сущности (Domain)

Сущности определяются бизнес-правилами предприятия:

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

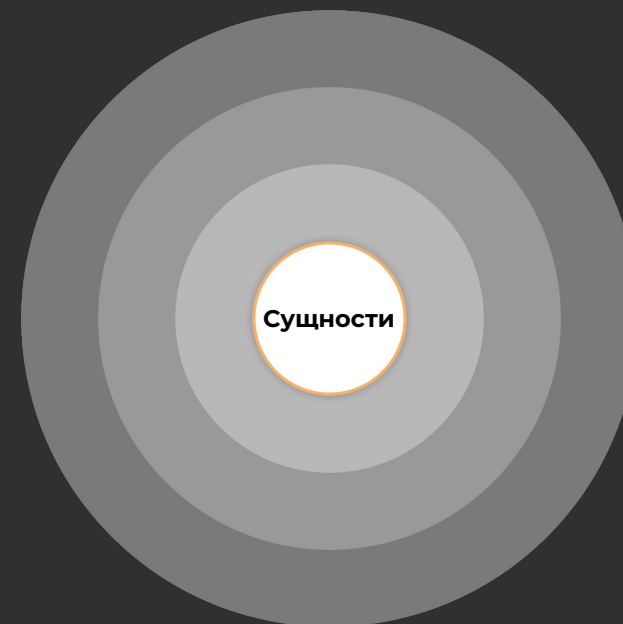
Не восприимчив к внешним изменениям:

- Базы данных
- Пользовательским интерфейсом
- Фреймворком

Изменения в работе сущности повлияет:

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

Изменения поведения приложения затронут код в данном слое



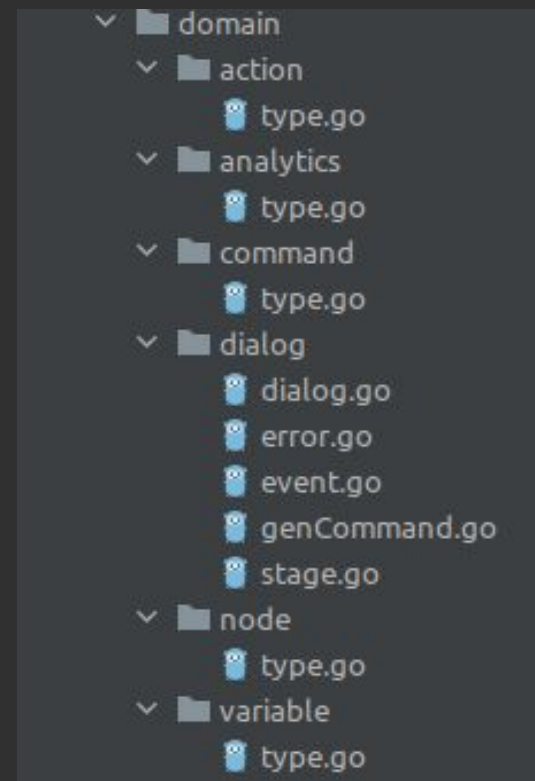
Сущности (Domain)

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
 - Заполнение ФИО из атомарных полей

Домен ничего не знает о других уровнях

Он содержит чистую бизнес-логику



Сценарии (Use Case)

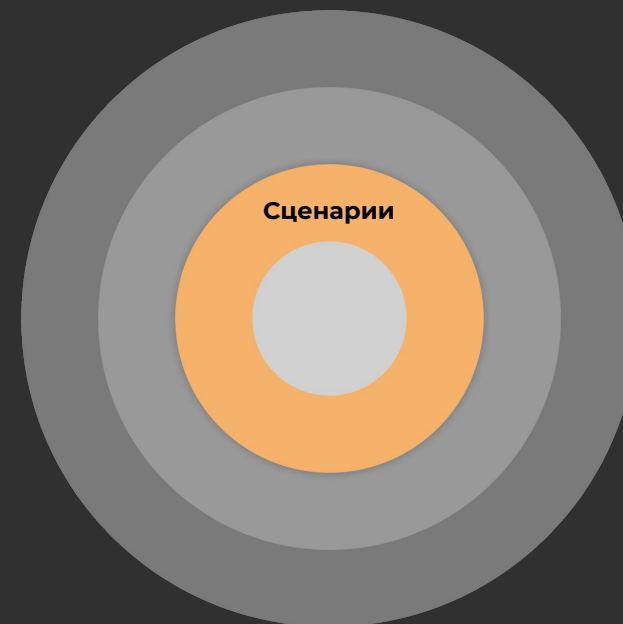
Use case – это детализация, описание действия, которое может совершить пользователь системы

Использует domain, но ничего не знает о внешних слоях

Он понятия не имеет, вызывается ли:

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

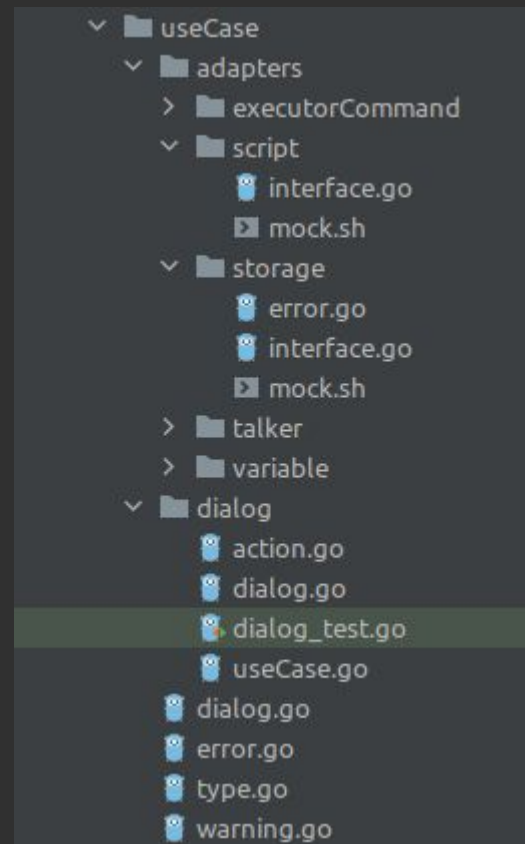
Доступы к Repository / Delivery осуществляются при помощи interface который диктует Use case



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-Адаптеры

Основная функция уровня интерфейсного адаптера –
преобразование данных

- Данные из внешних слоёв к удобному формату Use Case
- Данные из Use Case к удобному формату для внешнего слоя



Интерфейс-Адаптеры (Repository Pattern)

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

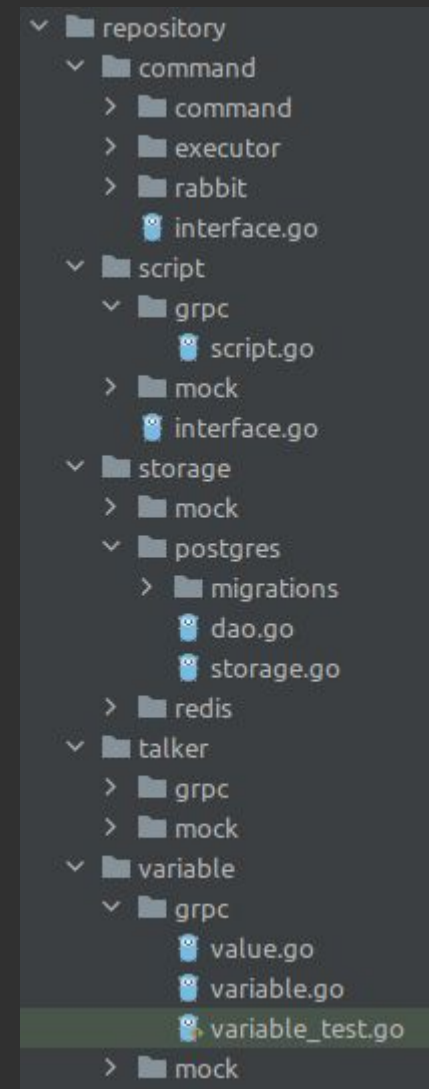
```
15 type Storage interface {
16     Contact
17     Group
18     Campaign
19 }
20
21 type Contact interface {
22     CreateContact(ctx robo_context.Context, createdBy uuid.UUID, contacts ...*contact.Contact) ([]*contact.Contact, error)
23     UpdateContact(ctx robo_context.Context, ID, createdBy uuid.UUID,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

```
repository
├── command
│   ├── command
│   ├── executor
│   └── rabbit
│       └── interface.go
├── script
│   ├── grpc
│   │   └── script.go
│   ├── mock
│   │   └── interface.go
├── storage
│   ├── mock
│   ├── postgres
│   │   ├── migrations
│   │   ├── dao.go
│   │   └── storage.go
│   ├── redis
├── talker
│   ├── grpc
│   ├── mock
├── variable
│   ├── grpc
│   │   ├── value.go
│   │   ├── variable.go
│   │   └── variable_test.go
│   └── mock
```


Интерфейс-Адаптеры (Repository)

Примеры:

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



Интерфейс-Адаптеры (Delivery)

Единственная точка входа для сторонней системы

- Выставление внешнего API
- Работа с методами Use Case

Не может напрямую обращаться к Repository

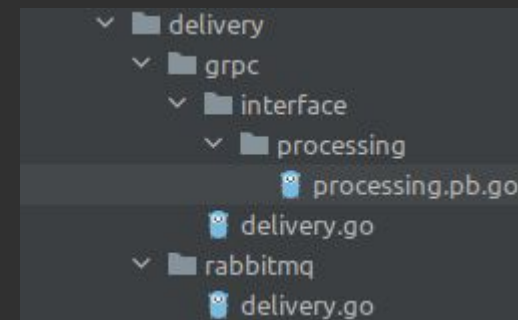
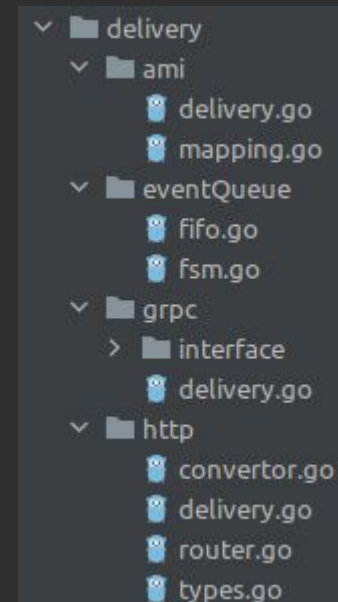


Интерфейс-Адаптеры (Delivery)



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



Фреймворки и драйверы. (Drivers)

Самый внешний уровень модели

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



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 круга.

Как это всё объединять?

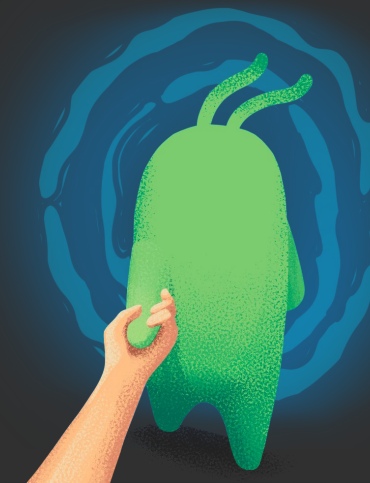
```
package main

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



Теория всё? Можно уже кодить?

Примеры кода на чистой архитектуре:

1. [go-clean-template](#)
2. [go-clean-arch](#)
3. [go-clean-architecture](#)



Блиц-опрос

1. Что делать, если у вас *Domain* сущность имеет данные из двух разных систем?
2. Как быть, если вам нужно объединить данные из разных БД?
3. Как решить проблему транзакционности при вставке в несколько таблиц одно БД?
(Создание контакта и добавление его в группу)

Таков путь...

День 1: 1. Создание структуры проекта

День 2: 2. Подключение к БД
3. Чистая архитектура:
3.1. Структура папок по чистой архитектуре
3.2. Наполнение – Domain
3.3. Интерфейс – Use Case
3.4. Интерфейс – Repository
3.5. Конструкторы слоёв
3.6. Инициализация слоёв на main
3.7. Реализация интерфейса – Use Case

День 3: 3.8. Реализация интерфейса – Repository
3.9. Реализация – Delivery
4. Использование – Context
5. Добавить логи
6. Добавить трассировку
7. Тестирование**



Вопросы

