

День 2: Структура проекта и «Чистая архитектура»

Спикер:
Николай Колядко



Вопросы



Таков путь...

- День 1:**
1. Создание структуры проекта
 2. Чистая архитектура:
 1. Структура папок по чистой архитектуре
 2. Реализация – Domain
 3. Интерфейс – Use Case
 4. Интерфейс – Repository
 5. Реализация – Delivery
 6. Конструкторы слоёв
 7. Инициализация слоёв на main
- День 2:**
8. Реализация интерфейса – UseCase
 9. Реализация интерфейса – Repository
 3. Использование – Context
 4. Добавить логи
 5. Добавить трассировку
 6. Тестирование**



Реализация интерфейса Use Case (30 мин)

Ветка с примером прошлого задания: 008-day-1-init-main

Цель: Реализовать интерфейс Use Case

1. Создайте свою ветку в git из ветки kolyadkons-08.
Правила наименования ФамилияИнициалы-09. Например, kolyadkons-09
2. Реализовать интерфейсы в папке `services/contact/internal/useCase`:
 - CRUD контакта
 - Создание группы
 - Чтение группы
 - Создание и добавление контакта в группу
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

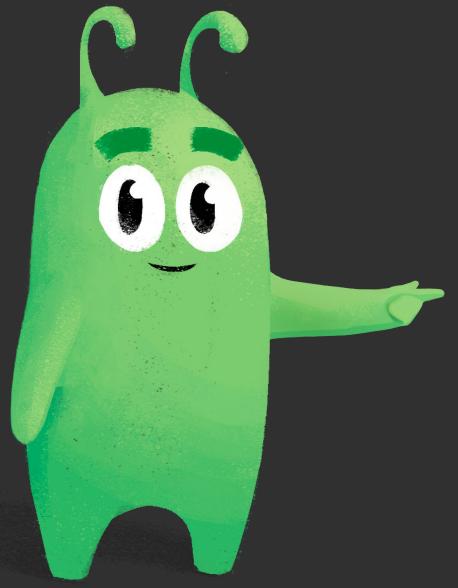
Перерыв 10 мин

Реализация интерфейса Repository (40 мин)

Ветка с примером прошлого задания: 009-day-2-implemented-use-case

Цель: Реализовать интерфейс Repository для контакта

1. Создайте свою ветку в git из ветки kolyadkons-09.
Правила наименования **ФамилияИнициалы-10**. Например, kolyadkons-10
2. В папке `services/contact/internal/repository` : (squirrel / pgxscan)
 - CRUD контакта
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

Реализация интерфейса Repository (50 мин)

Ветка с примером прошлого задания: 009-day-2-implemented-use-case

Цель: Реализовать интерфейс Repository для группы

1. Создайте свою ветку в git из ветки kolyadkons-09.
Правила наименования **ФамилияИнициалы-10**. Например, kolyadkons-10
2. В папке `services/contact/internal/repository` : (squirrel / pgxscan)
 - Создание группы
 - Чтение группы
 - Создание и добавление контакта в группу
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

На этом чистая архитектура ВСЁ



Обед 60 мин

Context

Переносит через границы и между процессами:

- Сроки выполнения
- Сигналы отмены
- Значения относящиеся к области запроса



Context interface

- **Deadline** – возвращает время, когда будет отменен
- **Done** – возвращает канал. Будет закрыт после завершения текущей работы или отмены контекста
- **Err** – возвращает причину завершения
- **Value** – получает значение, соответствующее ключу

Дополнительную информацию можно получить в github.com

Работа с Context

Передается слой за слоем, поэтому, когда на верхнем уровне возникает ошибка и требуется завершение, на нижнем уровне может получить уведомление и обработать

```
contact.go
1 package http
2
3 import ...
23
24 var mappingSorts = query.SortsOptions{...}
32
33 //...
46 func (d *Delivery) CreateContact(c *gin.Context) {...}
106
107 //...
120 func (d *Delivery) UpdateContact(c *gin.Context) {...}
182
183 //...
194 func (d *Delivery) DeleteContact(c *gin.Context) {...}
210
211 //...
225 func (d *Delivery) ListContact(c *gin.Context) {...}
268
269 //...
281 func (d *Delivery) ReadContactByID(c *gin.Context) {...}
300

useCase/interface.go
1 package useCase
2
3 import ...
11
12 type Contact interface {
13     Create(ctx context.Context, contacts ...*contact.Contact)
14     Update(ctx context.Context, contactUpdate contact.Update)
15     Delete(ctx context.Context, ID uuid.UUID /*Тут можно передавать фильтр*/)
16
17     ContactReader
18 }
19
20 type ContactReader interface {
21     List(ctx context.Context, parameter queryParameter)
22     ReadByID(ctx context.Context, ID uuid.UUID) (response, error)
23     Count(ctx context.Context) /*Тут можно передавать фильтр*/
24 }
25
26 type Group interface {
27     Create(ctx context.Context, groupCreate *group.Group)
28     Update(ctx context.Context, groupUpdate *group.Group)
29     Delete(ctx context.Context, ID uuid.UUID /*Тут можно передавать фильтр*/)
30
31     GroupReader
32 }

storage/interface.go
1 package storage
2
3 import ...
11
12 type Storage interface {
13     Contact
14     Group
15 }
16
17 type Contact interface {
18     CreateContact(ctx context.Context, contacts ...*contact.Contact)
19     UpdateContact(ctx context.Context, ID uuid.UUID, updateFn)
20     DeleteContact(ctx context.Context, ID uuid.UUID) error
21
22     ContactReader
23 }
24
25 type ContactReader interface {
26     ListContact(ctx context.Context, parameter queryParameter)
27     ReadContactByID(ctx context.Context, ID uuid.UUID) (response, error)
28     CountContact(ctx context.Context) /*Тут можно передавать фильтр*/
29 }
30
31 type Group interface {
32     CreateGroup(ctx context.Context, group *group.Group) (*group.Group, error)
33 }
```

Работа с Context

Передача данных внутри контекста:

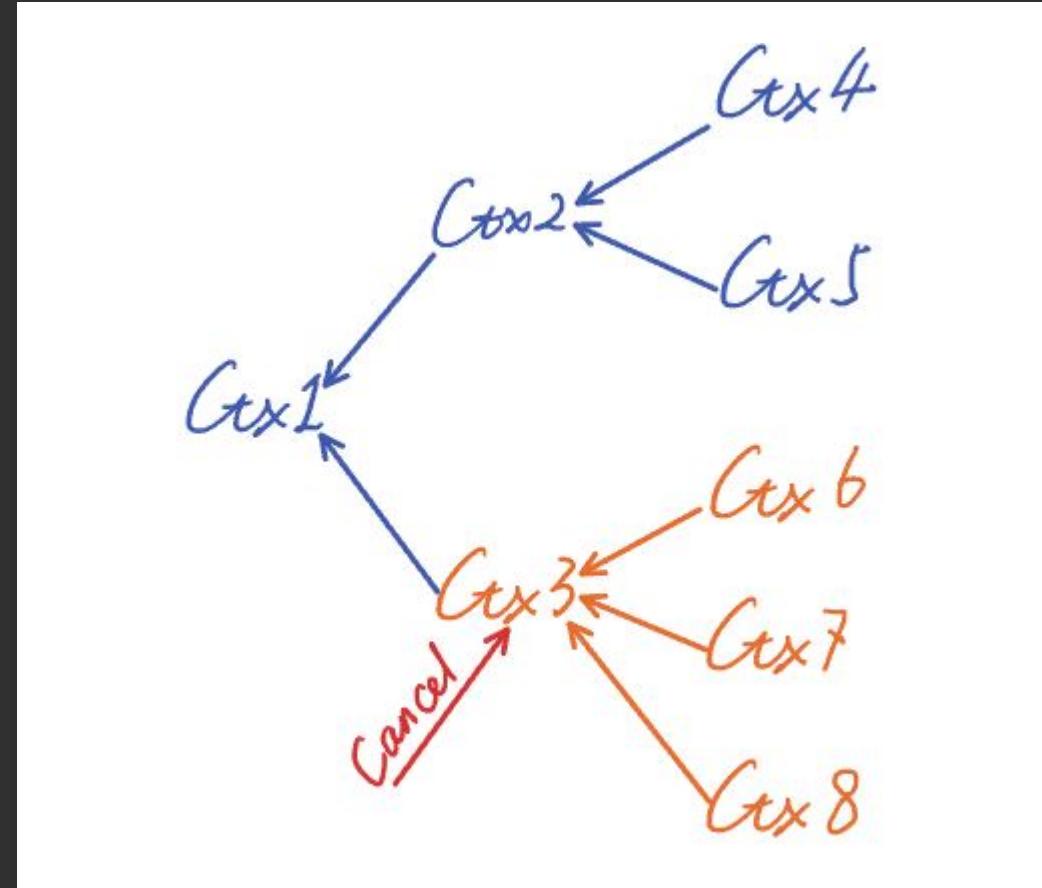
- Сохранение по ключу
- Рекурсивное получение переменных по ключу

Не рекомендуется использовать **для передачи данных** необходимых **для выполнения** функций

Варианты использования – cancel

- Установка таймера на выполнение
 - WithTimeout
 - WithDeadline
- Отмена по внутренней логике
- Отмена со стороны

Имеет решающее значение для **повышения стабильности и устойчивости системы**

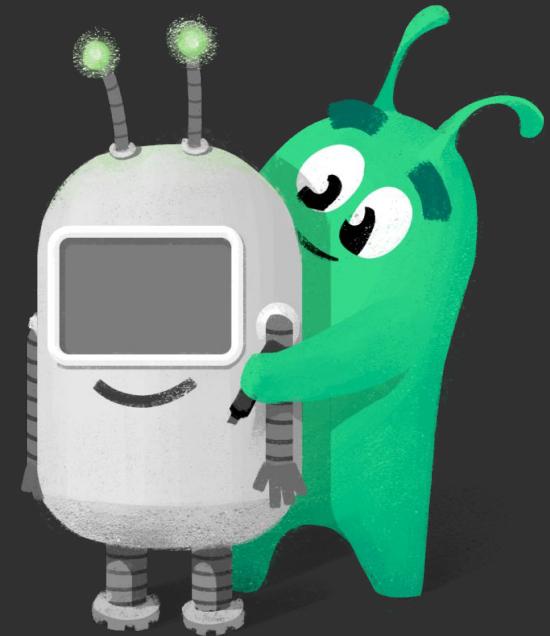


Использование Context (20 мин)

Ветка с примером прошлого задания: 010-day-2-implemented-repository

Цель: Добавить использование Context в методах

1. Создайте свою ветку в git из ветки kolyadkons-11.
Правила наименования **ФамилияИнициалы-11**. Например, kolyadkons-11
2. В папке `services/contact/internal/`:
 - Внутри каждого метода создать контекст на Delivery (context)
 - Добавить проброску контекста через слои между всеми
 - Delivery -> UseCase -> Repository
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

Observability – наблюдаемость

Показатель того, насколько эффективно можно определить внутреннее состояние системы по ее выходным данным (телеметрии)

Три кита:

- Логи
- Трассировки
- Метрики

Наблюдаемость системы зависит от телеметрии, получаемой с помощью измерительных приборов, поступающих от конечных точек и служб

Цель – понять, что происходит в используемых средах, чтобы вы могли обнаруживать и устранять проблемы



Observability – наблюдаемость

Помогает понимать и отвечать на конкретные вопросы о том, что происходит в системах

Позволяет понять, что работает медленно или не работает и что необходимо сделать для повышения производительности

Команды могут получать оповещения о проблемах и активно устранять их до того, как они повлияют на пользователей

Собирать и анализировать данные о наблюдаемости, у вас появится бесценное представление о влиянии ваших цифровых сервисов на бизнес

Оптимизировать конверсии

Проверять соответствие выпусков ПО бизнес-целям

Приоритеты бизнес-решений на основе того, что наиболее важно

Анализирует данные о пользовательском опыте, вы можете обнаружить проблемы раньше и разработать лучший пользовательский опыт на основе реальной

Observability – Логи

Логи – структурированные или неструктурированные текстовые записи о незаметных событиях, произошедших в определенное время



Observability – Трассировки

Трассировка – отображает активность транзакции или запроса по мере прохождения через приложения и показывает, как подключаются службы, включая сведения на уровне кода

Например, последовательность вызова функций

Трассировка является косвенным способом оценки логики приложения



Observability – Трассировки

Метрики – значения, представленные в виде подсчетов или показателей, которые часто вычисляются или агрегируются за определенный период времени

Часто числовое значение за определенный период времени

Показатели могут быть получены из различных источников:

- Инфраструктура
- Хосты
- Службы
- Облачные платформы
- Внешние источники

Информация предоставляется через API или Events



Преимущества наблюдаемости

Мониторинг
производительности
приложений

Мониторинг
инфраструктуры

Бизнес-
аналитика



DevSecOps
и SRE

Опыт конечного
пользователя

Проблемы наблюдаемости

- **Изолированные данные:** множество агентов, разрозненные источники данных и разрозненные инструменты мониторинга затрудняют понимание взаимозависимостей между приложениями
- **Объем, скорость, разнообразие и сложность:** практически невозможно получить ответы из огромного количества необработанных данных
- **Ручное инструментирование и настройка:** Когда ИТ-ресурсы вынуждены вручную настраивать и изменять код для каждого нового типа компонента или агента
- **Отсутствие предварительной подготовки:** Даже при нагрузочном тестировании в стадии предварительной подготовки разработчики по-прежнему не имеют возможности наблюдать или понимать, как реальные пользователи влияют на приложения и инфраструктуру, прежде чем они запустят код в производство

Вопросы



Добавить логи (30 мин)

Ветка с примером прошлого задания: 011-day-2-context

Цель: Добавить использование Context в методах

1. Создайте свою ветку в git из ветки kolyadkons-11.
Правила наименования **ФамилияИнициалы-12**. Например, kolyadkons-12
2. В папке services/contact/internal:
 - Проверка существующего контакта. ([zap](#))
 - При получении контакта. Если контакт не найдет по ID – вернуть ошибку (404)
 - При возникновении ошибку логировать в консоль первоначальную ошибку (указать стек вызова)
3. Запустить проект
4. Залейте свою ветку в git

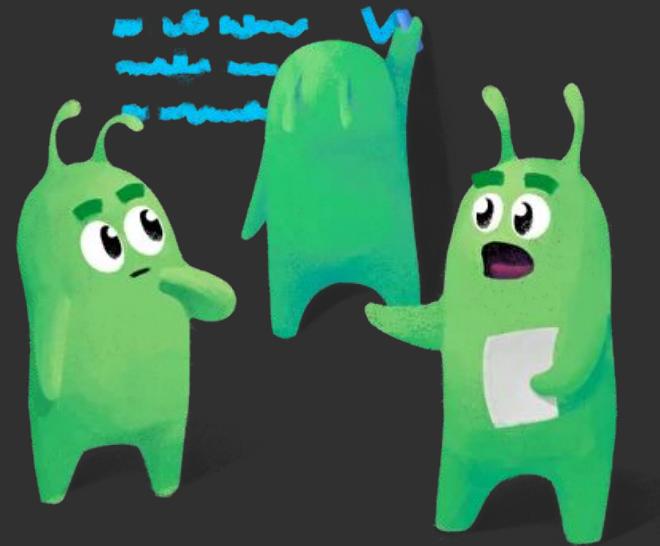


Демонстрация

Добавить трассировку (40 мин)

Ветка с примером прошлого задания: 012-day-2-log

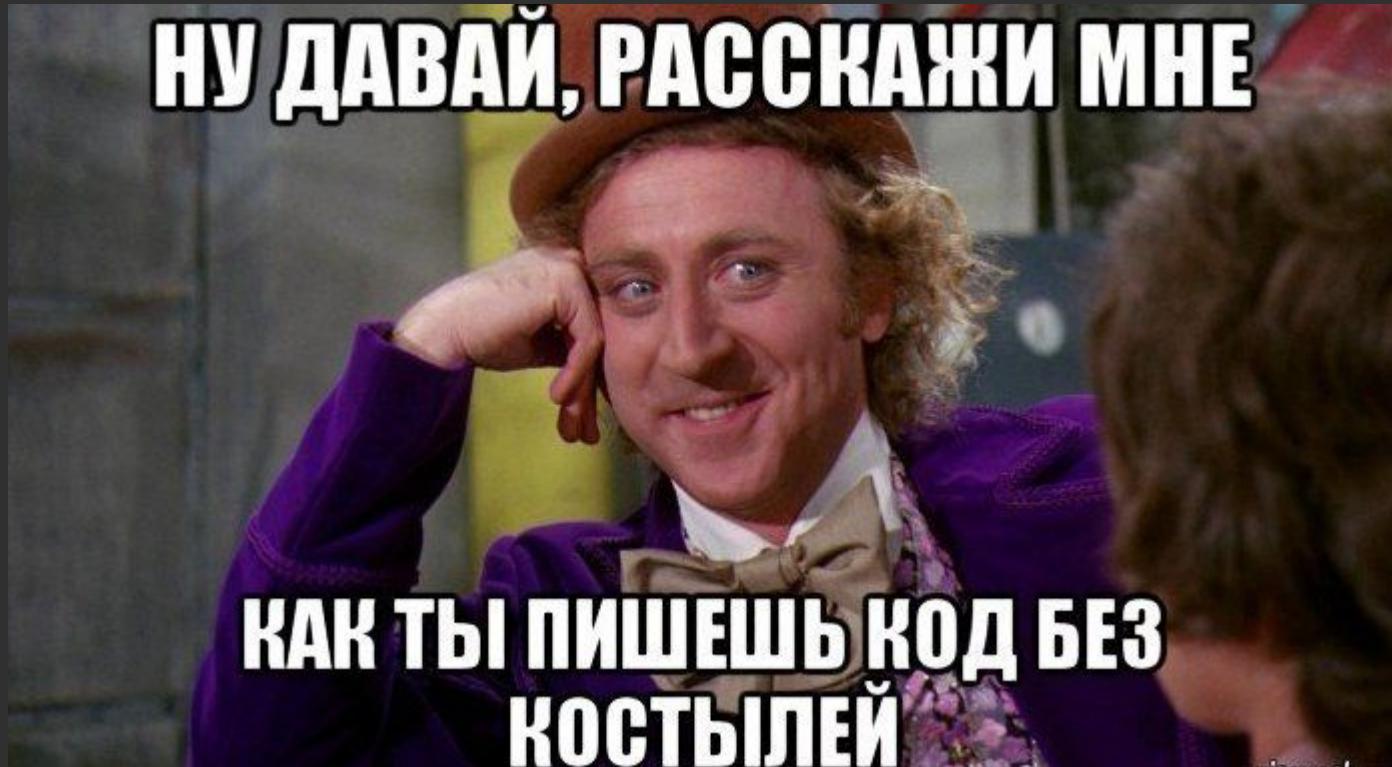
1. Создайте свою ветку в git из ветки kolyadkons-12. Правила наименования
ФамилияИнициалы-13
Например, kolyadkons-13
2. В папке **services/contact/internal** :
 - Пробросить трассировку. Delivery -> UseCase -> Repository ([opentracing-go](#))
 - Создание контакта
3. Запустить проект
4. Залейте свою ветку в git



Демонстрация

Демонстрация Тестирование

Ветка с примером прошлого задания: 013-day-2-tracing

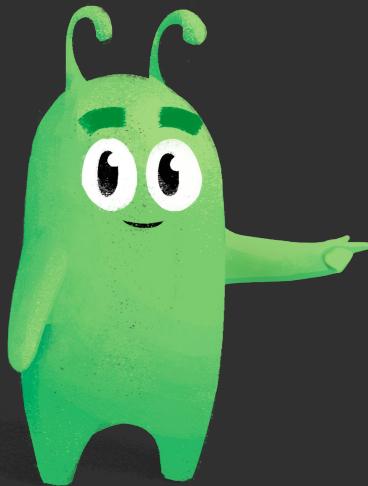


Конец пути ...

- День 1:**
1. Создание структуры проекта
 2. Чистая архитектура:
 - о Структура папок по чистой архитектуре
 - о Реализация – Domain
 - о Интерфейс – Use Case
 - о Интерфейс – Repository
 - о Реализация – Delivery
 - о Конструкторы слоёв
 - о Инициализация слоёв на main
- День 2:**
- о Реализация интерфейса – UseCase
 - о Реализация интерфейса – Repository
 3. Использование – Context
 4. Добавить логи
 5. Добавить трассировку
 6. Тестирование**



Вопросы



Что самое полезное для вас (Чат)

Что из этого Вы можете уже использовать в работе?

Какую оценку поставите себе по итогом интенсива?

Оправдались ли ожидания?

Всё

Весь проект в ветке:

015-day-2-end

Литература:

- Learning Domain-Driven Design
- Go with domain

