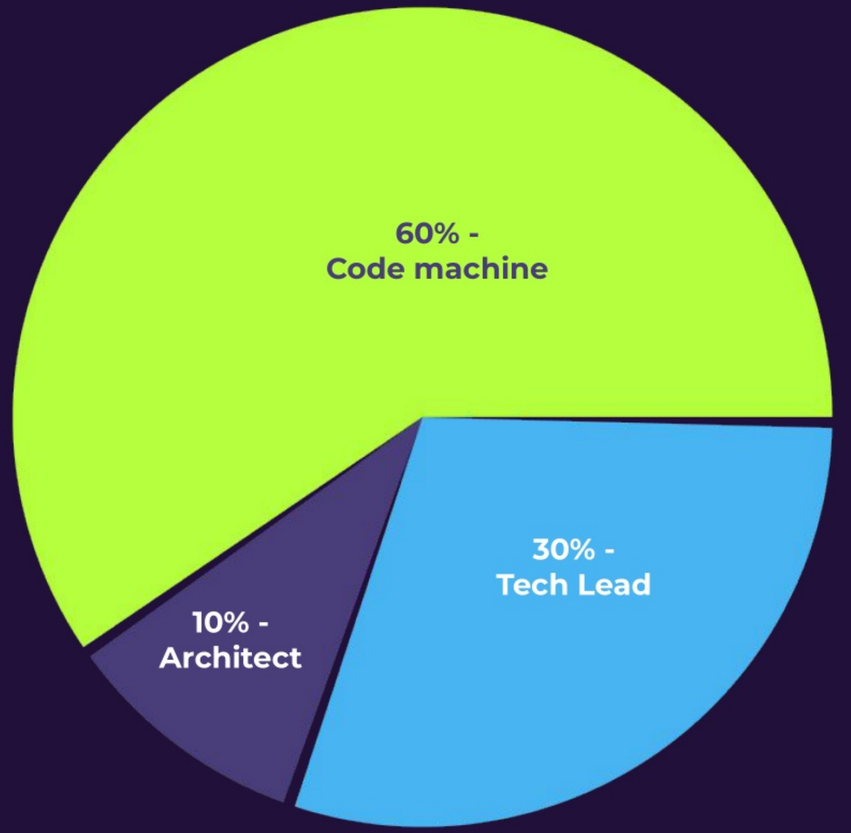
Какой опыт содержит этот интенсив

- Изучаем мощь фреймворка, пишем не только хороший, но и правильный/ поддерживаемый/ расширяемый код
- Осознаем какие проблемы решают заложенные во фреймворк подходы, понимаем как координировать совместную работу над кодовой базой
- Заглядываем в будущее, представляем место решений в экосистеме, а не изолированно



День 1

- Познакомимся
- Поговорим об API
- Напишем ну очень простое приложение с API
- Сделаем это приложение чуть сложнее, но сильно поддерживаемое
- Научимся работать на уровне между запросом и его обработчиком
- Подключим все это к СУБД и создадим базу данных миграцией
- Упакуем в образ и запустим его в контейнере

Тема #1 - Как не наломать дров в создании API

А что за API-то?

Почему? Зачем?
А главное для чего?

API повсюду, мы просто об этом не думаем

- Библиотека
  - Модуль
    - Сервис
      - Бизнес-домен
        - SaaS-решение
          - Интеграция SaaS-решений

Library Based API

Web Service API

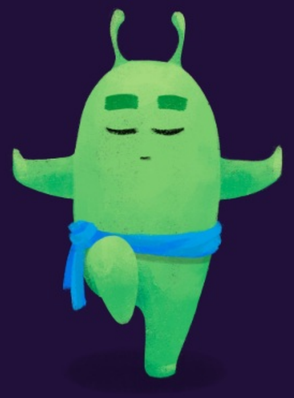
Class Based API

```
def do_useful_things(*args, *kwargs) -> Any

class VeryImportantThing():
    def init(*args, *kwargs)

    def process_this(*args, *kwargs)

    def do_again(*args, *kwargs)

POST /{resource_id}/{task_id}/run
```

Проблема #1

✔ API - возможность для клиента взаимодействовать с вашей системой в понятных ему категориях

✔ API - отражение реального мира в модели вашего приложения без лишнего

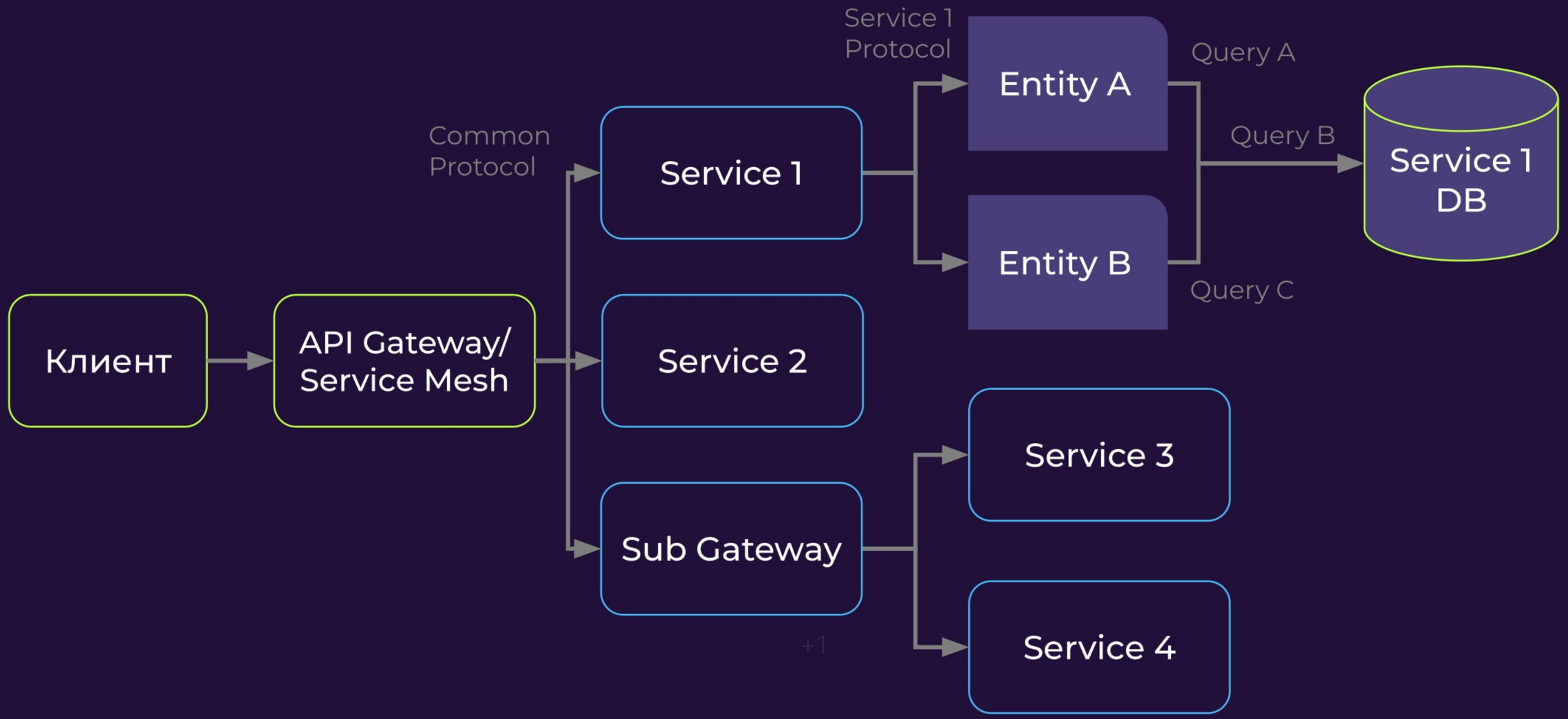


❌ API не интерфейс к данным

❌ API не описание вашей инфраструктуры

❌ API даже не структура вашего бизнеса

API как трансформация технического в бизнес



Пример из жизни

Как инженеры видели self-service аналитику

Задача: посчитать услуги в EdTech

Ресурсы в API:

- Вводное занятие
- Тьюторское занятие
- Групповое занятие
- Лабораторная работа
- Самостоятельная работа

Пример из жизни

Как мы реализовали self-service аналитику

Задача: посчитать услуги в EdTech

В EdTech проводят уроки:

- Вводное занятие - Урок (с методистом и одним учеником)
- Тьюторское занятие - Урок (с учителем и учеником)
- Групповое занятие - Урок (с учителем несколькими учениками)
- Лабораторная работа - Урок (без учителя и несколькими учениками)
- Самостоятельная работа - Урок (без учителя и одним учеником)

Я пишу веб-сервис

REST, или REST-like, или RPC, или GraphQL?

Все зависит от ваших нужд

Экстенсивным развитием появляются протоколы
Интенсивным развитием появляются версии протоколов и RFC

- Много микросервисов с единственной ответственностью - gRPC
- Много простых манипуляций на ресурсами - REST
- Много сложных манипуляций над многокомпонетными ресурсами - GraphQL
- Много чтений и навигации по сложным ресурсам - Custom protocol over REST (E.g. OData)
- Трекинг выполнения асинхронной задачи - Web Sockets

И с этим можно спорить!
Главное, делать как можно проще для клиента
И не мешать все в одну кашу

Окей, допустим это будет REST

Как задизайнить хэндлеры?

Используйте силу прикладных протоколов

REST - протокол поверх HTTP и у него есть Verbs

- **HEAD** - Проверить существование объекта без его получения
- **GET** - Получить объект
- **PUT** - Создать или заменить объект
- **DELETE** - Удалить объект
- **PATCH** - Изменить объект
- **POST** - Вызвать операцию над объектом - очень страшный метод, подумайте трижды

Verbs определяют, что произойдет с объектом (кроме POST)

Напишите в чат, что не так с POST

Дайте клиенту свободу решений

Но задокументируйте логику вашего приложения

- Обозначьте и полно опишите ресурсы
- Предоставьте список возможных операций над ресурсом и их последствия
- Технически отделите ресурсы от их представления в БД и СУБД
- По возможности имплементируйте максимально короткие ссылки на объект (помните, что состояние протокола хранит клиент)

✔ GET /group/{group_id}/students
✔ GET /student/{student_id}
✔ GET /group/{group_id}

❌ GET /school/{school_id}/faculty/{faculty_id}/group/{group_id}/students
❌ GET /user/{user_id}?is_student=true
❌ GET /faculty-of-group/{group_id}

Написали, обработали

Что ответить?

+1

Отвечайте четко и отвечайте правду

REST - протокол поверх HTTP и у него есть коды ответа

- 200 - Операция осуществлена успешно
- 201 - Объект создан
- 202 - Заявка принята, но результата пока нет
- 302 - Ресурс найден, но находится по другому URI
- 401 - Для доступа нужно авторизироваться
- 402 - Доступ предоставляется на платной основе
- 404 - Объект не найден
- 409 - Объект уже существует
- 422 - Объект имеет логическую ошибку в свойствах

Не делайте кастомные коды в body
Помните, чем проще и нативнее, тем лучше

Держите в курсе изменений

Возвращайте измененный объект после мутирующих операций PUT/PATCH/POST

Как правило, этот объект попросят следующей же операцией, а оверхед на соединение может составлять 99%

Каков итог?

Чек-лист начала разработки API

- Ожидания клиента заранее известны, отражены в API с первым приоритетом и задокументированы
- API достаточно абстрактно, чтобы не повторять вашу инфраструктуру/ процессы/ структуру компании
- У клиента есть свобода в вариантах использования вашего API как конструктора операций в вашем приложении
- Заранее определен набор тест-кейсов вашего API
- Решения нативны по отношению к протоколу и соблюдают конвенцию протокола
- Клиент всегда получает актуальную и своевременную информацию в ответе от приложения
- Параметры запроса в API валидируются приложением

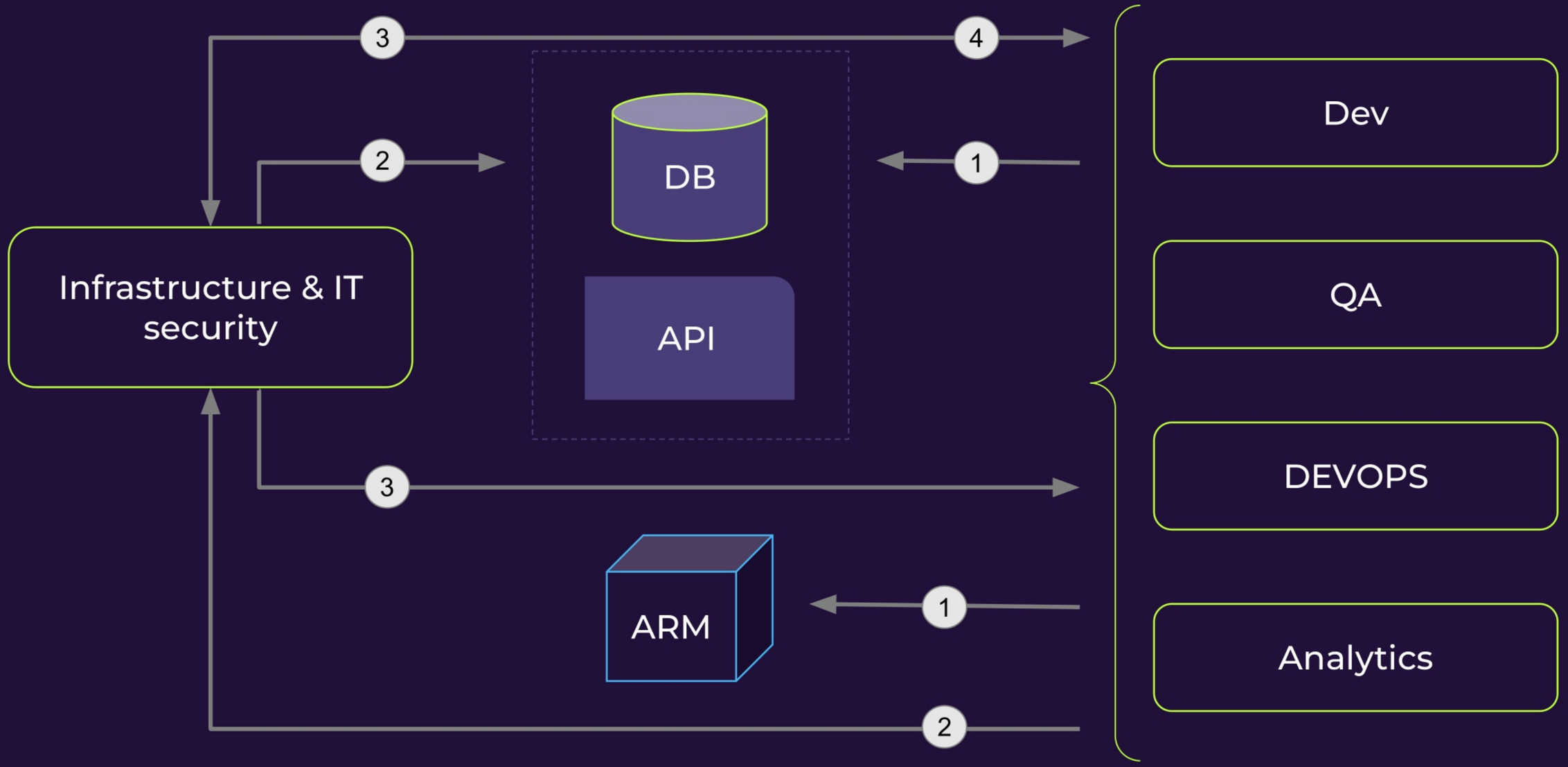
Тема #2 - Как создать API

Создаем скелет приложения

Система контроля выданных ресурсов

Сходим к менеджеру инфры

Структура процесса



Требования заказчика

Важно видеть:

- Список ресурсов в целом и по свободности
- Список сотрудников с их ресурсами
- Закрепленные за сотрудником ресурсы
- Кто имеет доступ к ресурсу

Нужно уметь:

- Добавить, редактировать, удалять профайл сотрудника
- Добавить, редактировать, удалять ресурс
- Делать крутой финт с выдачей сотруднику нескольких ресурсов одним действием

Уточним у команд

Описание картины мира

- Заказчик - сотрудник компании - учетный номер, ФИО, не обязательно грейд
- Ресурсы - доступ к вычислительным мощностям или выдача АРМ - идентификатор ресурса, тип ресурса, признак мультидоступа
- Вычислительные мощности - база данных, сервис или АРМ

Устным распоряжением ССО членам успешных команд может быть выдан доступ к *генератору пиццы* (относится к АРМ), однажды получивший доступ к генератору пиццы, не теряет его до увольнения

Чек-лист начала разработки API

✔ Ожидания клиента заранее известны, отражены в API с первым приоритетом и задокументированы

✔ API достаточно абстрактно, чтобы не повторять вашу инфраструктуру/ процессы/ структуру компании

✔ У клиента есть свобода в вариантах использования вашего API как конструктора операций в вашем приложении

✔ Заранее определен набор тест-кейсов вашего API

❓ Решения нативны по отношению к протоколу и соблюдают конвенцию протокола

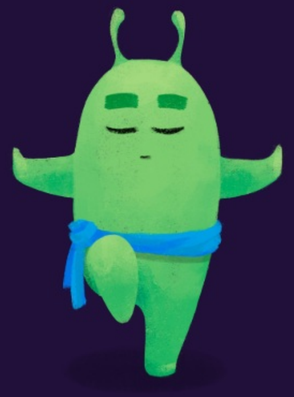
❓ Клиент всегда получает актуальную и своевременную информацию в ответе от приложения

❓ API поддерживает операции сортировки, фильтрации и постраничного вывода

❓ Параметры запроса в API валидируются приложением

Заголовок

РыбаРыбаРыбаРыбаРыба
РыбаРыбаРыба



РыбаРыбаРыбаРыбаРыба
РыбаРыбаРыба

Заголовок

РыбаРыбаРыбаРыбаРыбаРыбаРыбаРыбаРыбаРыбаР
ыбаРыбаРыбаРыбаРыбаРыбаРыба
РыбаРыбаРыбаРыбаРыбаРыбаРыбаРыба
РыбаРыбаРыбаРыбаРыбаРыбаРыбаРыба
РыбаРыбаРыбаРыбаРыбаРыбаРыбаРыбаРыбаР
ыбаРыбаРыбаРыбаРыбаРыбаРыбаРыбаРыбаРы
баРыбаРыбаРыбаРыба
РыбаРыбаРыбаРыбаРыбаРыбаРыбаРыба
РыбаРыбаРыбаРыбаРыбаРыбаРыбаРыба