ЗАДАЧА

1. Развернуть сервис на Golang, Postgres, Clickhouse, Nats (альтернатива kafka), Redis
2. Описать модели данных и миграций
3. В миграциях Postgres
    a. Проставить primary-key и индексы на указанные поля
    b. При добавлении записи в таблицу устанавливать приоритет как макс приоритет в таблице +1. Приоритеты начинаются с 1
    c. При накатке миграций добавить одну запись в Projects таблицу по умолчанию
        i. id = serial
        ii. name = Первая запись
4. Реализовать CRUD методы на GET-POST-PATCH-DELETE данных в таблице GOODS в Postgres
5. При редактировании данных в Postgres ставить блокировку на чтение записи и оборачивать все в транзакцию. Валидируем поля при редактировании.
6. При редактировании данных в GOODS инвалидируем данные в REDIS
7. Если записи нет (проверяем на PATCH-DELETE), выдаем ошибку (статус 404)
    a. code = 3
    b. message = “**errors.good.notFound**“
    c. details = {}
8. При GET запросе данных из Postgres кешировать данные в Redis на минуту. Пытаемся получить данные сперва из Redis, если их нет, идем в БД и кладем их в REDIS
9. При добавлении, редактировании или удалении записи в Postgres писать лог в Clickhouse через очередь Nats (альтернатива kafka). Логи писать пачками в Clickhouse
10. При обращении в БД использовать чистый SQL
11. Обернуть приложение в докер

МИГРАЦИИ POSTGRES

1. **Синий цвет** - primary key
2. * - индекс параметры
3. C - создание
4. U - обновление
5. R - обязательное поле или нет при редактировании

PROJECTS

Атрибут	Тип	Описание	Пример	C	U	R
id *	int	id записи	1	+		+
name	string	название	Запись 1	+	+	+
created_at	timestamp	дата и время	timestamp	now		+

Добавляем при старте приложения сразу одну запись в таблицу

GOODS

Атрибут	Тип	Описание	Пример	C	U	R
id *	int	id записи	1	+		+
project_id *	int	id кампании	1	+		+
name *	string	название	Запись 1	+	+	+
description	string	описание	Описание		+	
priority	int	приоритет	1	max+1	+	+
removed	bool	статус удаления	false	false	+	+
created_at	timestamp	дата и время	timestamp	now		+

МИГРАЦИИ CLICKHOUSE

Атрибут	Тип	Описание	Пример
Id *	int	идентификатор	1
ProjectId *	int	идентификатор	1
Name *	string	название	Запись 1
Description	string	описание	Описание
Priority	int	приоритет	1
Removed	bool	статус удаления	false
EventTime	timestamp	дата и время	timestamp

REST Методы

POST /good/create

request

```
TYPE: Post
Header: empty
URL: projectId=int

Payload: {
  "name": "string"
}
```

response

```
{
  "id": int,
  "projectId": int,
  "name": "string",
  "description": "string",
  "priority": int,
  "removed": false,
  "createdAt": "timestamp"
}
```

PATCH /good/update

request

```
TYPE: Patch
Header: empty
URL: id=int & projectId=int // проверяем, что запись есть

Payload: {
  "name": "string", // проверяем, чтобы не было пустым
  "description": "string" // необязательное поле
}
```

response

```
{
  "id": int,
  "projectId": int,
  "name": "string",
  "description": "string",
  "priority": int,
  "removed": false,
  "createdAt": "timestamp"
}
```

- инвалидируем данные в Redis
- отправляем лог в Clickhouse

DELETE /good/remove

request

```
TYPE: DELETE
Header: empty
URL: id=int & projectId=int // проверяем, что запись есть

Payload: {}
```

response

```
{
  "id": int,
  "campaignId": int,
  "removed": true
}
```

- инвалидируем данные в Redis
- отправляем лог в Clickhouse

GET /goods/list

request

```
TYPE: GET
Header: empty
URL: limit=int & offset=int

Payload: {}
```

response

```
{
  "meta": {
    "total": int, // сколько всего записей
    "removed": int, // сколько записей со статусом Removed = true
    "limit": int, // какое ограничение стоит на вывод объетов (например 20шт)
    "offset": int // от какой позиции выводить данные в списке
  },
  "goods": [{
    "id": int,
    "projectId": int,
    "name": "string",
    "description": "string",
    "priority": int,
    "removed": false,
    "createdAt": "timestamp"
  }]
}
```

- если записей нет, возвращаем пустой массив
- **limit** - сколько вывести позиций в списке (например 20 позиций). Если не указан, считаем по умолчанию 10
- **offset** - от какой позиции вывести записи в кол-ве limit (например от 20 позиции вывести 20 позиций, соответственно с 20 по 40). Если не указан, считаем 1

PATCH /good/reprioritiize

request

```
TYPE: PATCH
Header: empty
URL: id=int & projectId=int // проверяем, что запись есть

Payload: {
  "newPriority": int
}
```

response

```
{
  "priorities": [{ // отдаем все приоритеты, которые были изменены
      "id": int,
      "priority": int
  }]
}
```

- Обновляем приоритет у текущей записи и у все, кто после. Приоритет начинается с 1
- инвалидируем данные в Redis
- отправляем лог в Clickhouse

Меняем позицию у текущей сущности и у сущностей после этой (прибавляем +1)