

# ЗАДАЧА

1. Развернуть сервис на Golang, Postgres, Clickhouse, Nats, Redis, Docker
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.common.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. 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
<b>id *</b>	int	id записи	1	+		+
<b>name</b>	string	название	Запись 1	+	+	+
<b>created_at</b>	timestamp	дата и время	timestamp	now		+

Добавляем при старте приложения сразу одну запись в таблицу

## GOODS

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

## МИГРАЦИИ CLICKHOUSE

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

## REST Методы

### POST /good/create

request

```
1 TYPE: Post
2 Header: empty
3 URL: projectId=int
4
5 Payload: {
6   "name": "string"
7 }
```

response

```
1 {
2   "id": int,
3   "projectId": int,
4   "name": "string",
5   "description": "string",
6   "priority": int,
7   "removed": false,
8   "createdAt": "timestamp"
9 }
```

## PATCH /good/update

request

```
1  TYPE: Patch
2  Header: empty
3  URL: id=int & projectId=int // проверяем, что запись есть
4
5  Payload: {
6    "name": "string", // проверяем, чтобы не было пустым
7    "description": "string" // необязательное поле
8  }
```

response

```
1  {
2    "id": int,
3    "projectId": int,
4    "name": "string",
5    "description": "string",
6    "priority": int,
7    "removed": false,
8    "createdAt": "timestamp"
9  }
```

- инвалидируем данные в Redis
- отправляем лог в Clickhouse

## DELETE /good/remove

request

```
1 TYPE: DELETE
2 Header: empty
3 URL: id=int & projectId=int // проверяем, что запись есть
4
5 Payload: {}
```

response

```
1 {
2   "id": int,
3   "campaignId": int,
4   "removed": true
5 }
```

- инвалидируем данные в Redis
- отправляем лог в Clickhouse

## GET /goods/list

request

```
1 TYPE: GET
2 Header: empty
3 URL: limit=int & offset=int
4
5 Payload: {}
```

response

```
1 {
2   "meta": {
3     "total": int, // сколько всего записей
4     "removed": int, // сколько записей со статусом Removed = true
5     "limit": int, // какое ограничение стоит на вывод объектов (например 20шт)
6     "offset": int // от какой позиции выводить данные в списке
7   },
8   "goods": [{
9     "id": int,
10    "projectId": int,
11    "name": "string",
12    "description": "string",
13    "priority": int,
14    "removed": false,
15    "createdAt": "timestamp"
16  }]
17 }
```

- 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

```
1 TYPE: PATCH
2 Header: empty
3 URL: id=int & projectId=int // проверяем, что запись есть
4
5 Payload: {
6   "newPriority": int
7 }
```

response

```
1 {
2   "priorities": [{ // отдаем все приоритеты, которые были изменены
3     "id": int,
4     "priority": int
5   }]
6 }
```

- 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)