# IN Migration Kafka концепт

## Каким образом производиться миграций через кафку?

Миграция через кафку производиться таким способом, что в кафке создаётся топик с каким-то названием которого не встречается в кафке на данный момент, далее в этот топик скидываются данные различных Бизнес Объектов(БО) в виде строк, которые содержит в себе JSON определенного ниже описанного формата, дальше этот топик регистрируется в системе и после регистраций в системе, система обращается в данный топик и начинает загружать данные в себе базу данных.

[Здесь](001-in-migration-kafka-example.md) описано это детальней.

### Структура JSON, каким оно должно быть.

Структура JSON должен соответствовать ниже приведённому примеру.

{  
"recordId": "12345",  
"externalId": "bfsseVCjGC225bM@",  
"id": "dgL9~4keReK7Z1qi",  
"boCode": "BO",  
"fields": [  
 {  
 "code": "textField",  
 "apiValue": "someValue"  
 },  
 {  
 "code": "textField2",  
 "apiValue": "someValue2"  
 }  
],  
"boFields": [  
 {  
 "fieldCode": "boTextField",  
 "toBoCode": "BO2",  
 "toFieldCode": "textFieldOfBO2",  
 "apiValue": "015252634519"  
 },  
 {  
 "fieldCode": "boTextField",  
 "boiInput": {  
 "recordId": "12345",  
 "externalId": "bfsseVCjGC225bM@",  
 "id": "dgL9~4keReK7Z1qi",  
 "boCode": "BO",  
 "fields": [  
 {  
 "code": "textField",  
 "apiValue": "someValue"  
 },  
 {  
 "code": "textField2",  
 "apiValue": "someValue2"  
 }  
 ],  
 "boFields": {...}  
 }  
 }  
],  
"state": "ACTUAL"  
}

### Подробнее о полях

recordId:Строка - Идентификатор данной записи, он должен быть уникальным в рамках всего топика   
 он используется для того чтобы запись обрабатывалась всего одни раз его затем,  
 система его запоминает и при повторной записи система его игнорирует.   
  
id:Строка - Идентификатор инстанции бизнес-объекта внутренний для платформы MyBPM. Если он указан, то  
 ищется запись по этому идентификатору. Если находиться, то происходит обновление данных.  
 Иначе генерируется ошибка. Если этот идентификатор не указан, то используется идентификатор  
 externalId  
  
externalId:Строка - Идентификатор инстанции БО, который используется во внешней системе (которая поставляет данные).  
 Если внутренний идентификатор не используется, то используется данный идентификатор для поиска  
 инстанции. В таком случае если инстанция находиться, то происходит обновление данных. Если не  
 находиться, то создаётся новая инстанция.  
  
boCode:Строка - Код бизнес объекта в которой, должны записаться данные.  
  
fields:Массив - Поля со значениями для мигрированния .  
 code:Строка - Код поля в бизнес объекте .  
 apiValue:Строка - Значение из REST API.  
  
boFields:Массив - Бизнес-поля для миграции вложенных бизнес-объектов   
 Стандартный подход:{  
 fieldCode:Строка - Код поля, который принадлежит с кодом boCode.  
 toBoCode:Строка - Код бизнес объекта на которое указывает данное поле.  
 toFieldCode:Строка - Код поля у бизнес объекта toBoCode. Это поле должно быть уникальное   
 apiValue:Строка - Значение поле toFieldCode у бизнес объекта с кодом toBoCode.  
 }  
 Рекурсивный подход:{  
 fieldCode:Строка - Код поля, который принадлежит с кодом boCode.  
 boiInput:Строка - вложенный JSON другого BoiInput.   
 }  
  
state - Состояние записи могут быть (ACTUAL, REMOVED, ARCHIVED, TEST), может отсутствовать,   
 если записи нету то state станет ACTUAL, а если запись есть то ничего не измениться   
 ACTUAL:Строка - Это актуальная запись которая доступна для работы.  
 REMOVED:Строка - Удалённая запись.  
 ARCHIVED:Строка - Архивированная запись.  
 TEST:Строка - Запись предназначена для режима тестирования

#### [Почему существует две структуры миграции вложенных бизнес-объектов и чем они отличаются друг от друга?](../025_kafka_migration/005-in-migration-kafka-bo-fields-example.md)

# IN Migration Kafka примеры

### Пример мигрирование поле

Термин: Бизнес-объект - БО

В дальнейшем уроке буду использовать данный термин для обозначения Бизнес-объекта

Давайте создадим бизнес-объект Клиент с полями ИИН, ИМЯ, ФАМИЛИЯ. ИИН - уникальное поле.

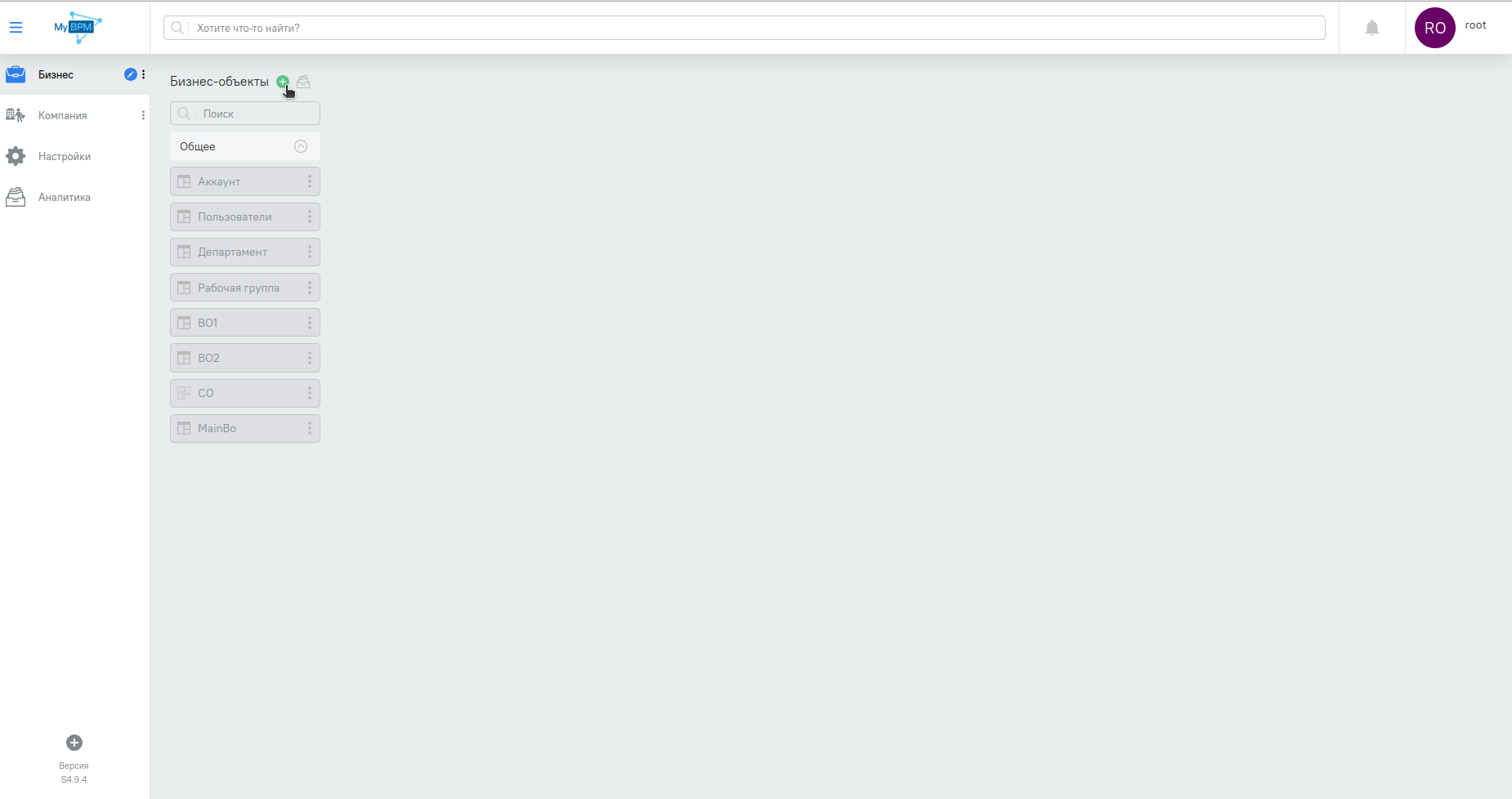
Для начало надо нажать на карандашик которая находиться в правом верхнем углу (серый карандаш).



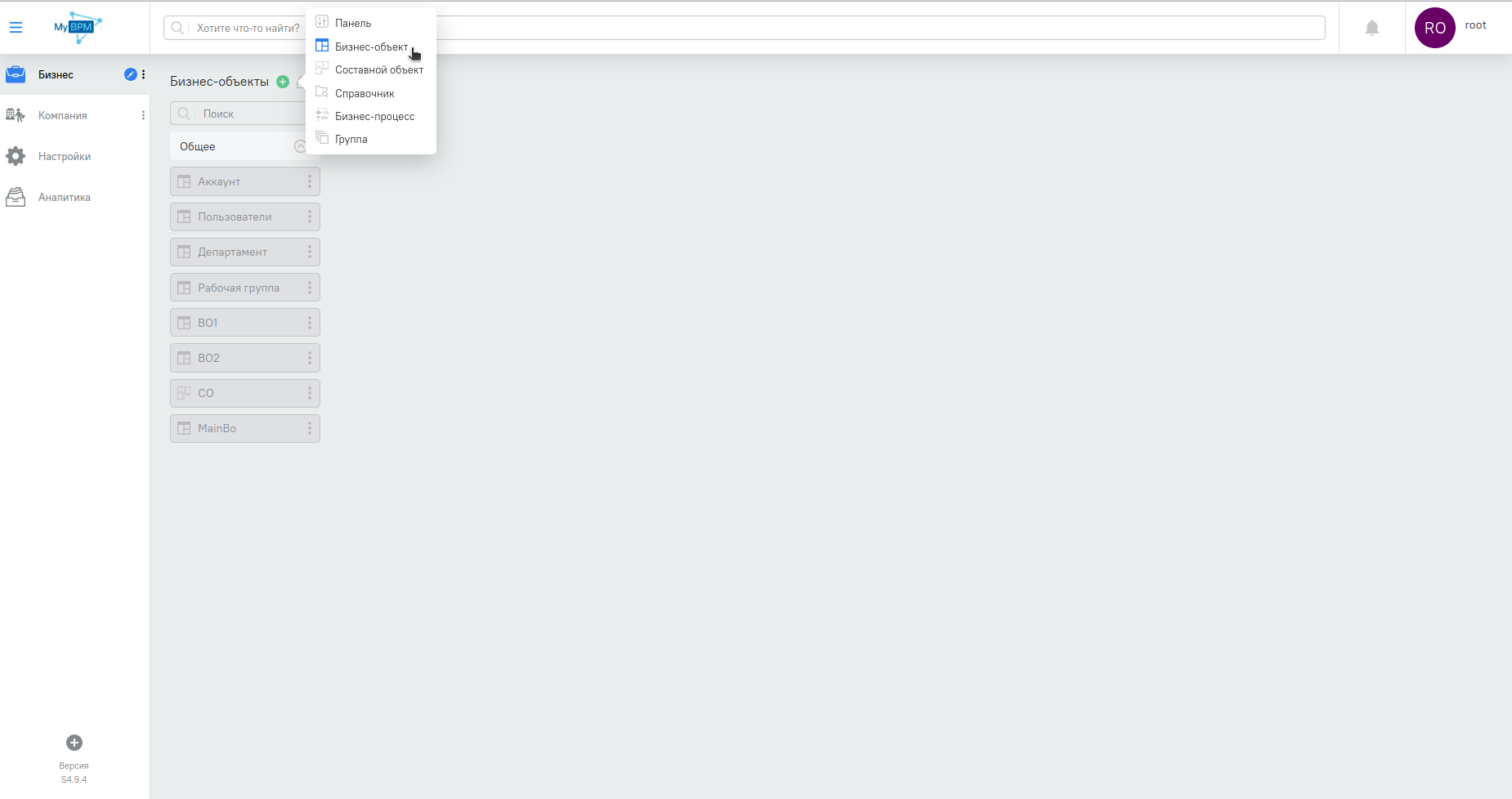
После нажатия карандаш становиться синим, это означает что вы находитесь в режиме редактирования, данном режиме можно создать бизнес объект



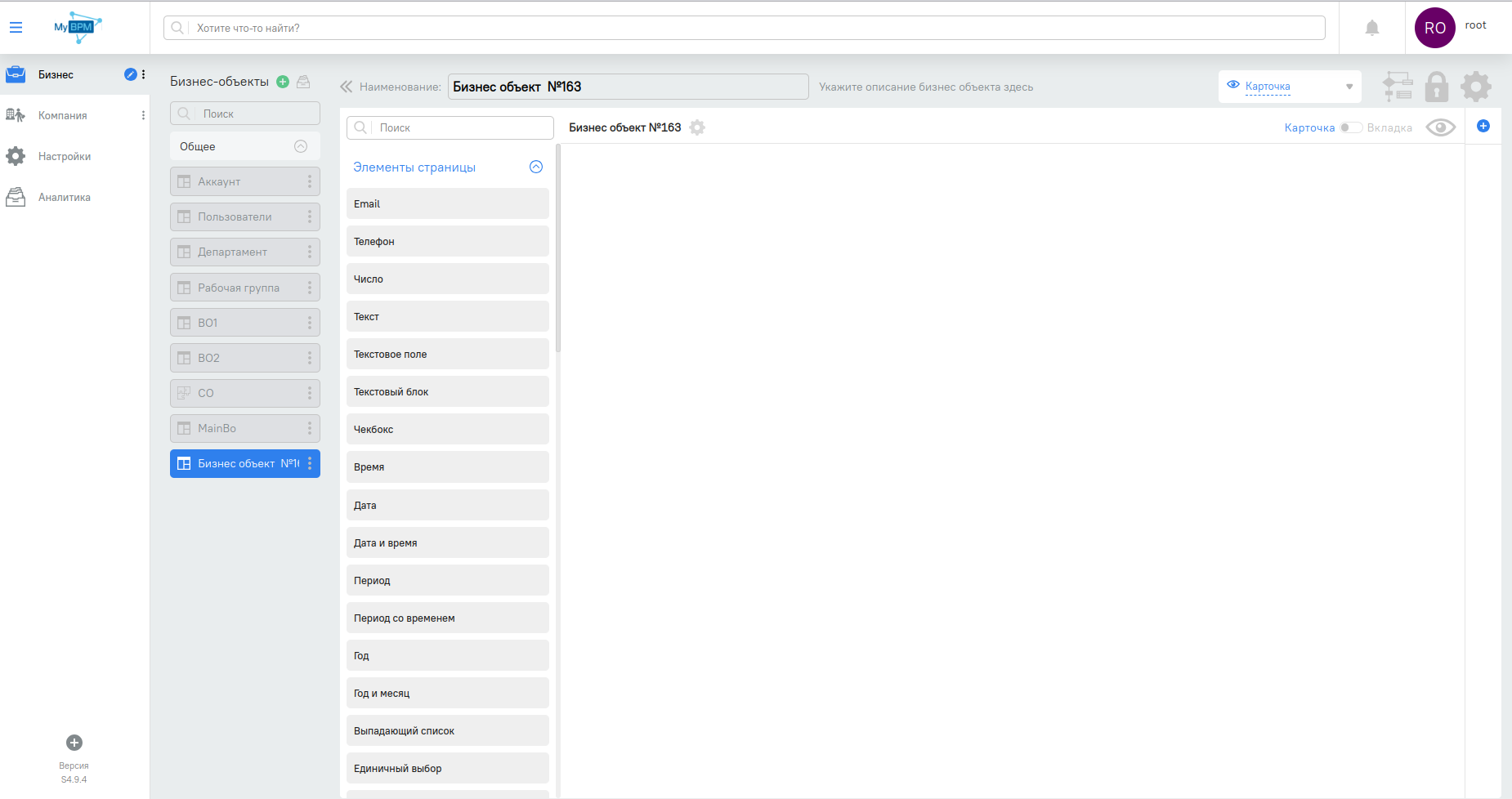
Убедившись что карандаш стал синим, нажимаем на зелёный крест, который добавляет новый бизнес объекты и прочее



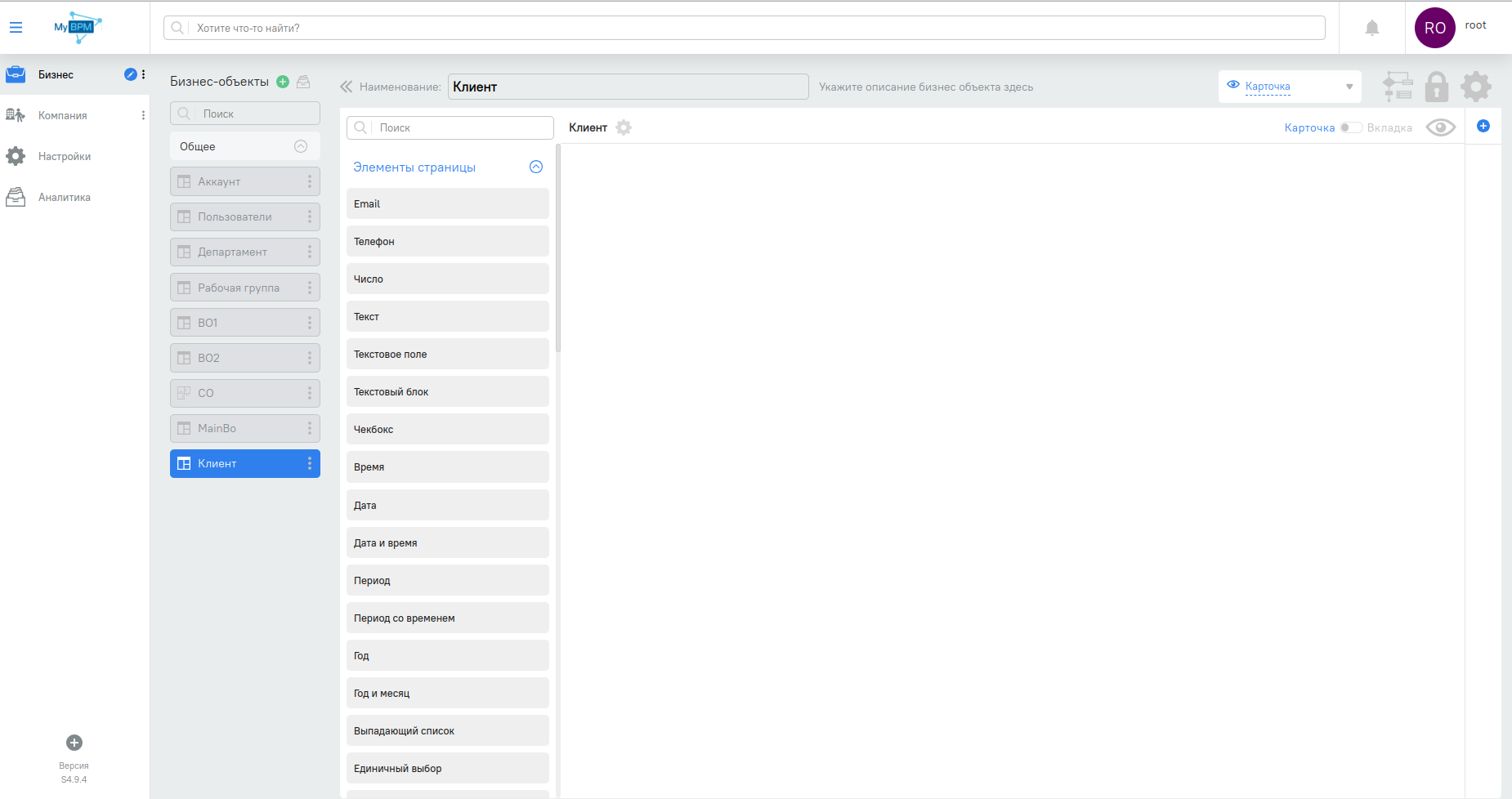
Появиться окошка и нажимаем на Бизнес-объект



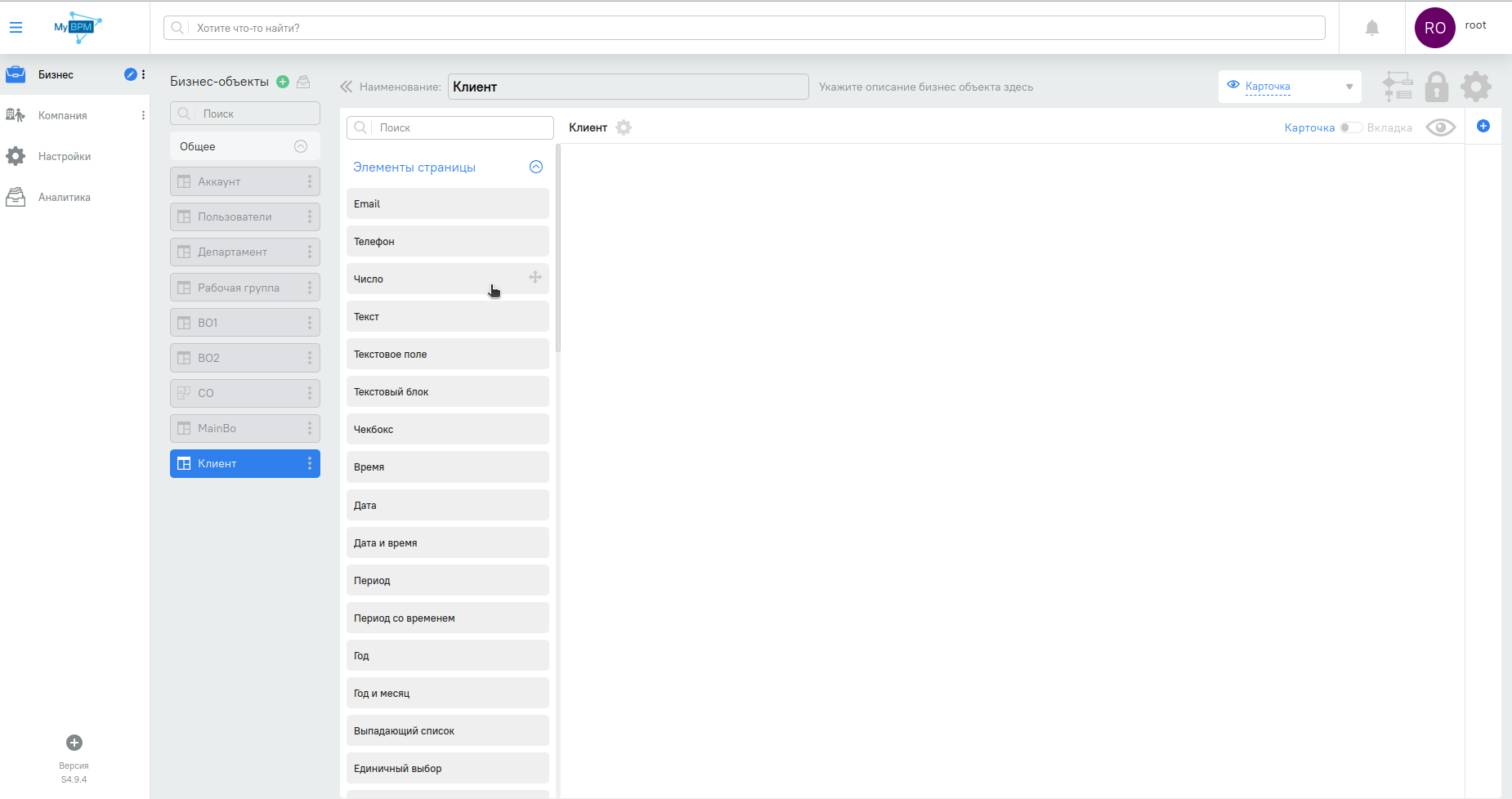
Появиться пустой БО



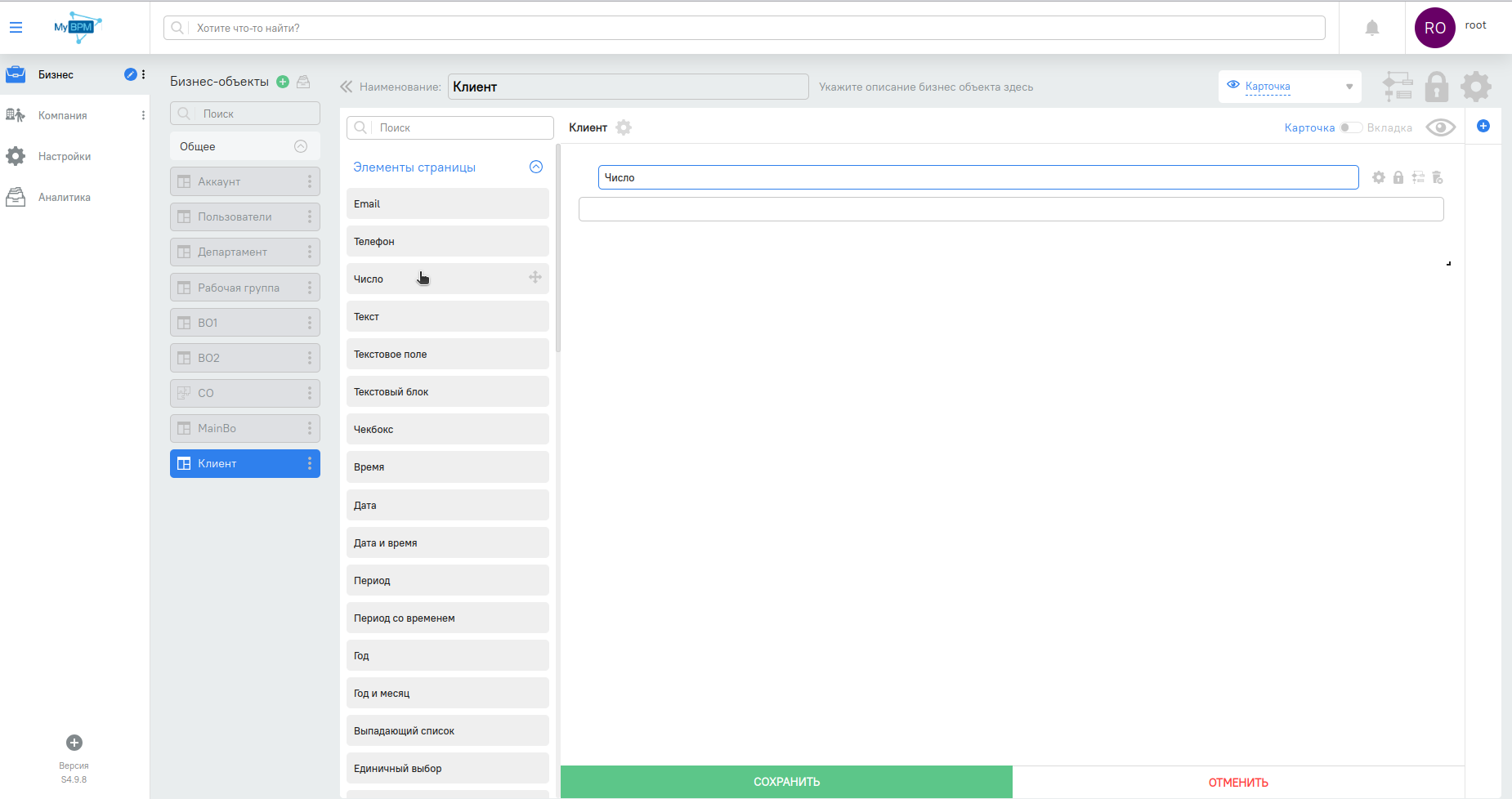
Давайте переименуем название БО на Клиент-а, для этого надо нажать на Наименование



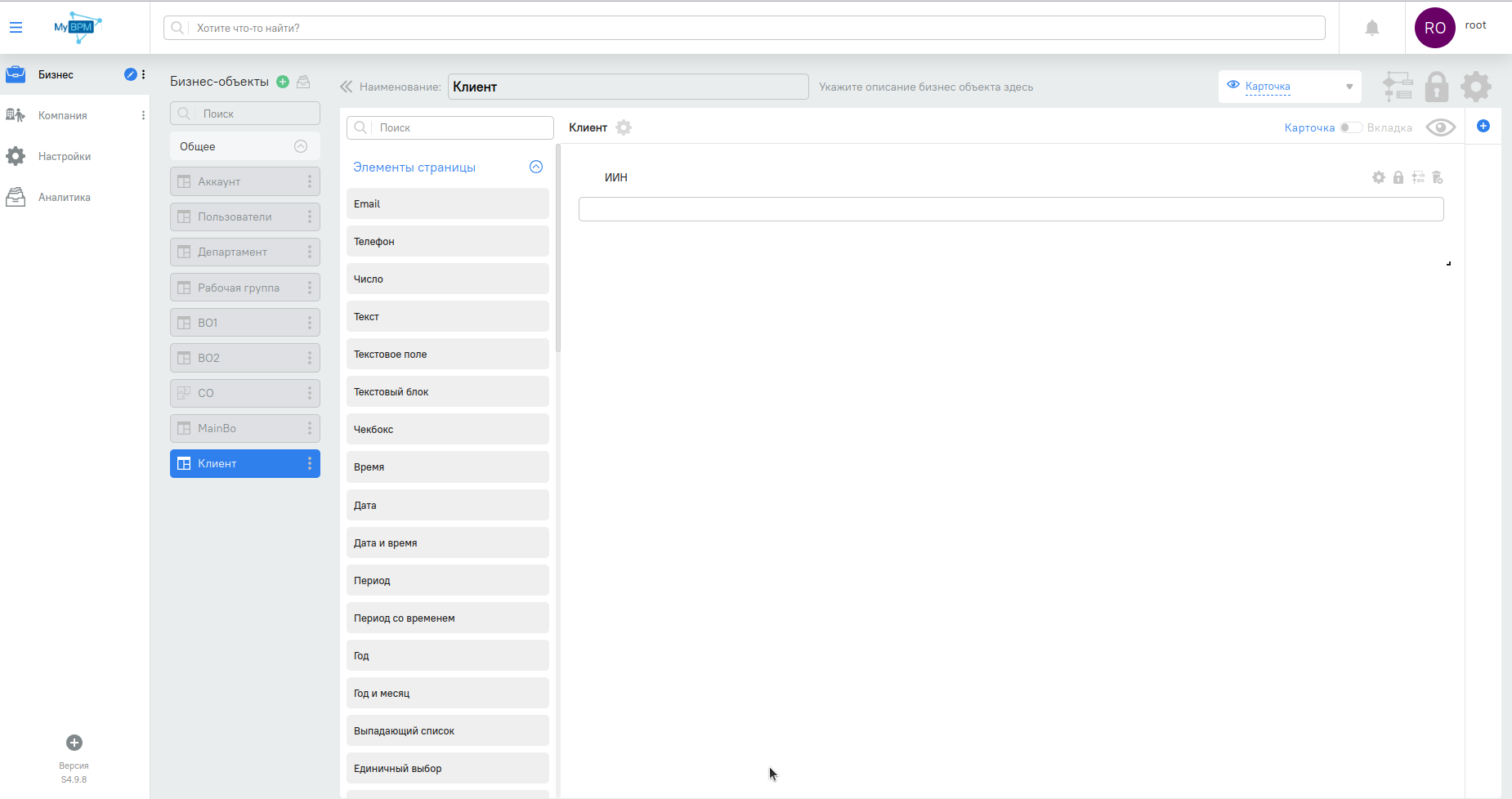
Осталось добавить поля как мы обговаривали ИИН, ИМЯ, ФАМИЛИЯ давайте добавим. В Элементах-страницы находиться поля Число, двойным нажатием на данное поле оно зарегистрирует в БО.



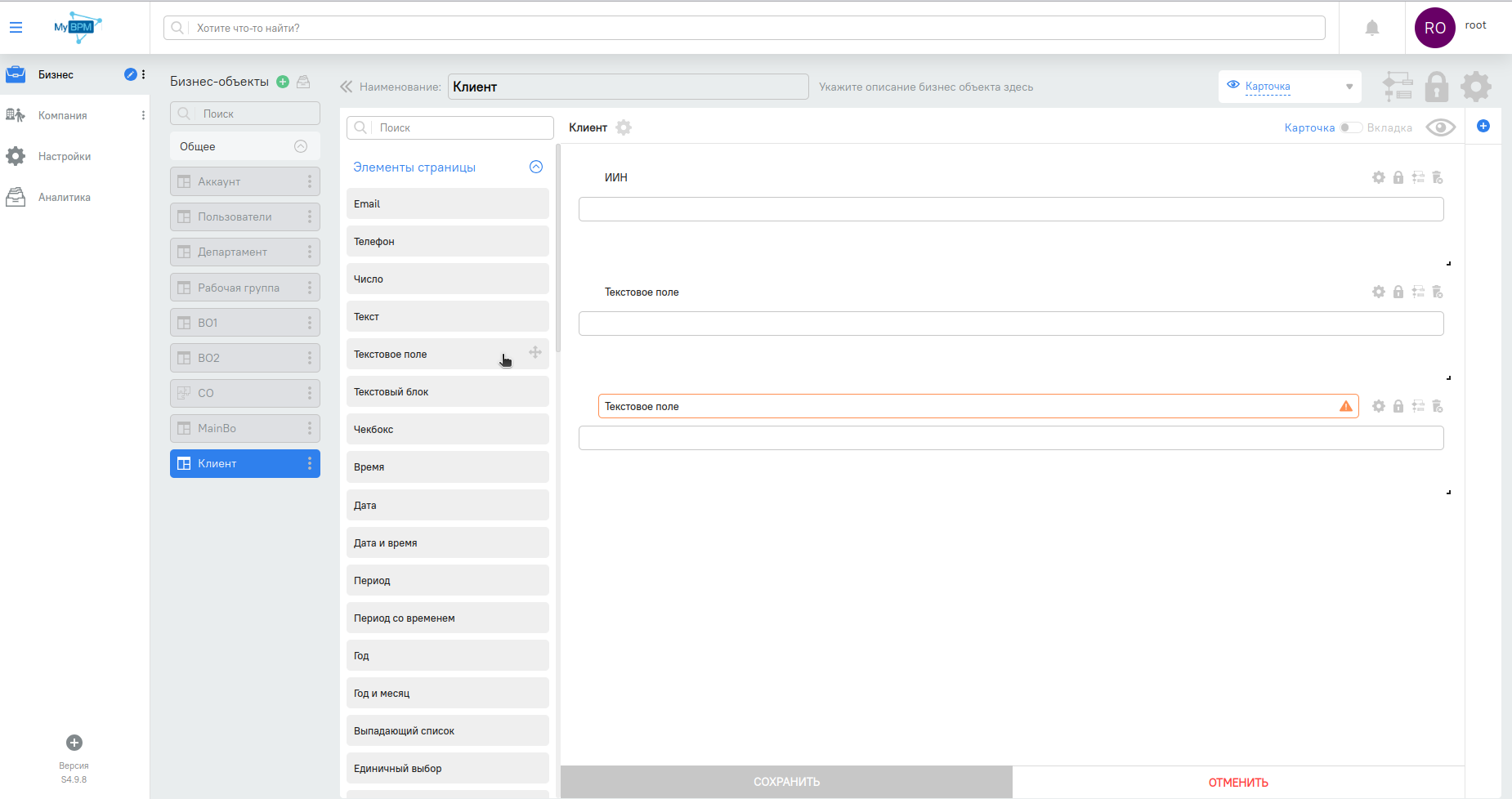
Результат:



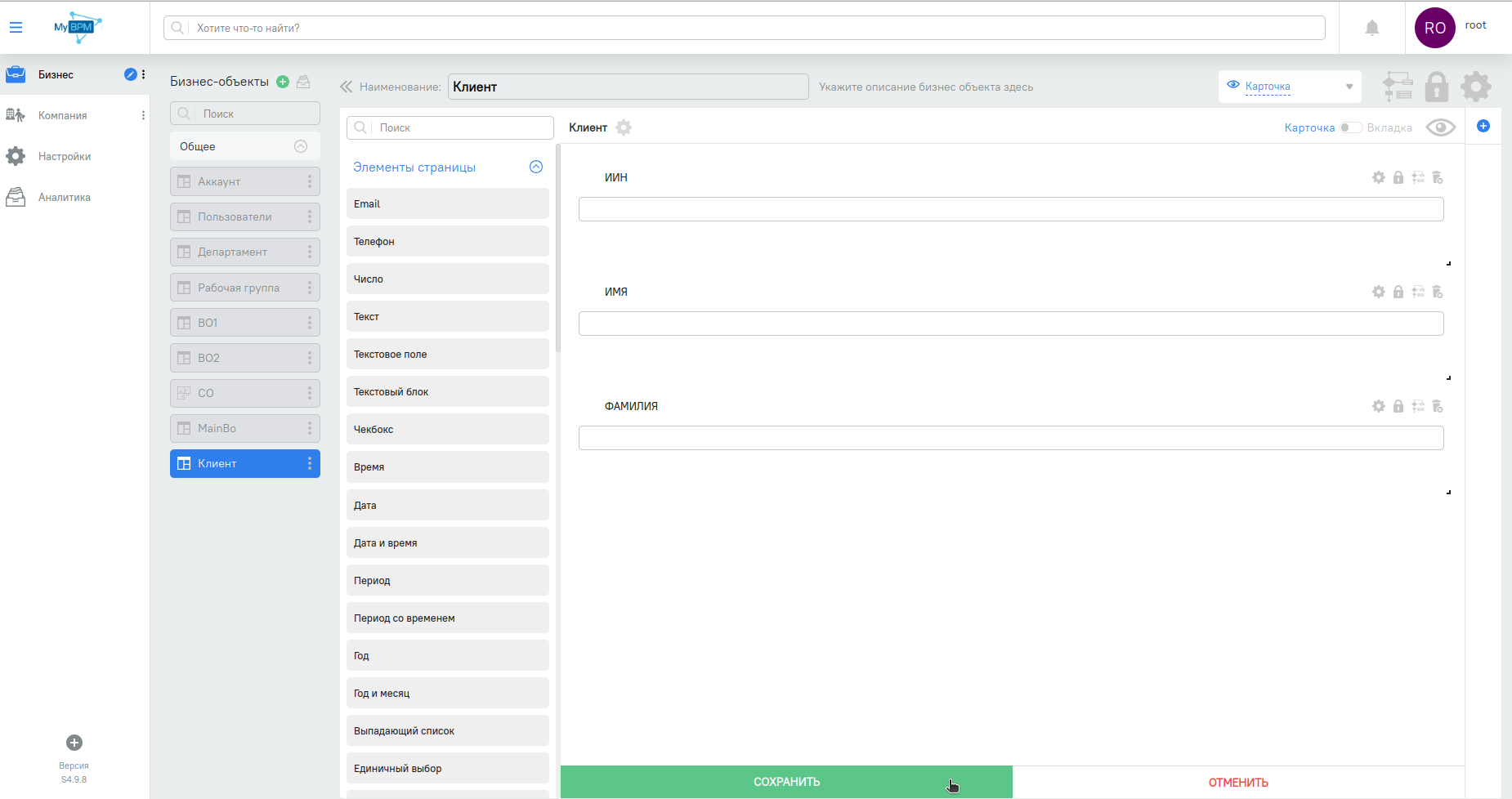
Надо переименовать название добавленного поле на ИИН и сохранить.



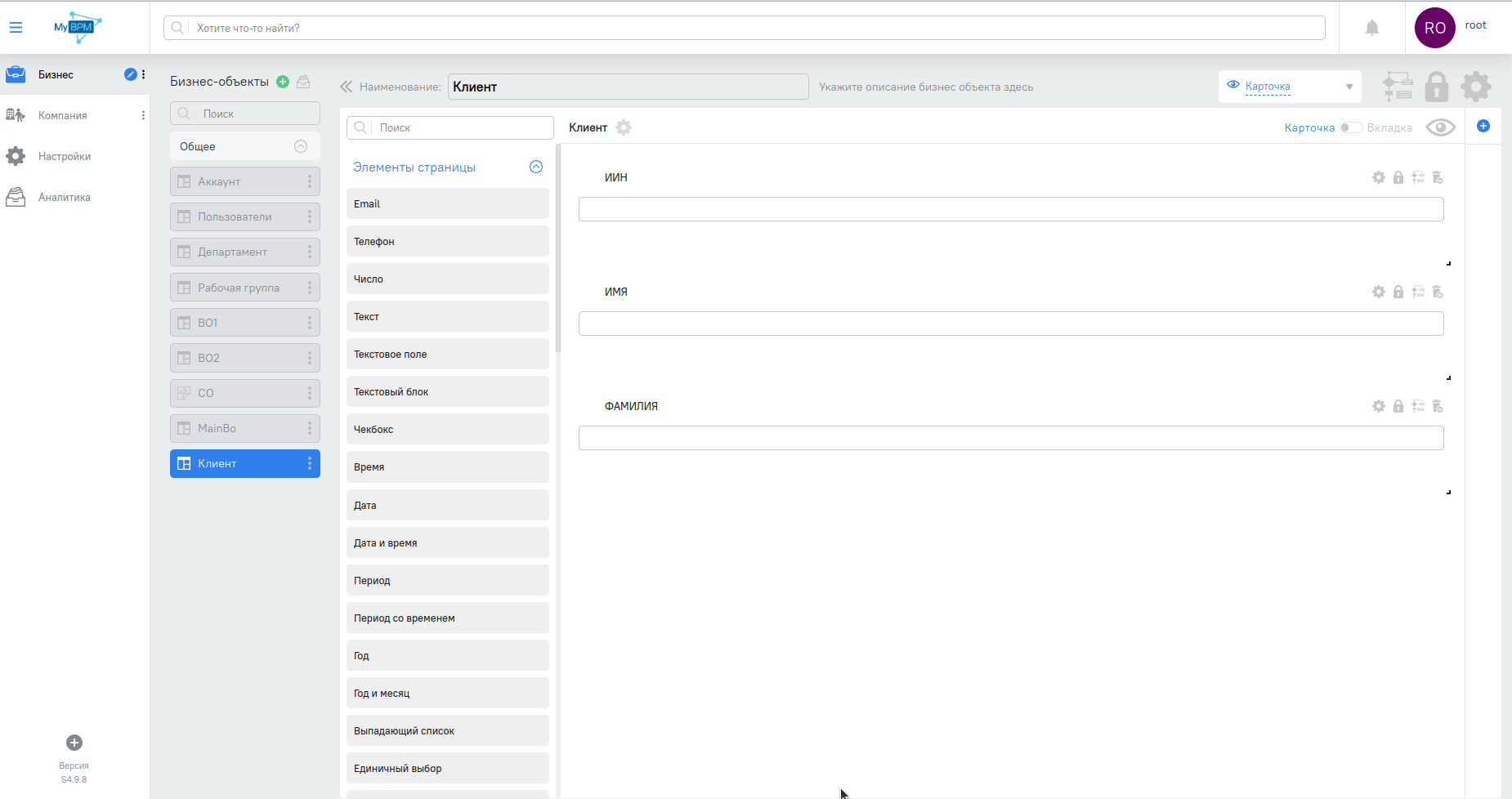
Теперь добавим 2 тестовых поле для Имени и Фамилий.



Надо переименовать название добавленных полей на ИМЯ, ФАМИЛИЯ и сохранить.

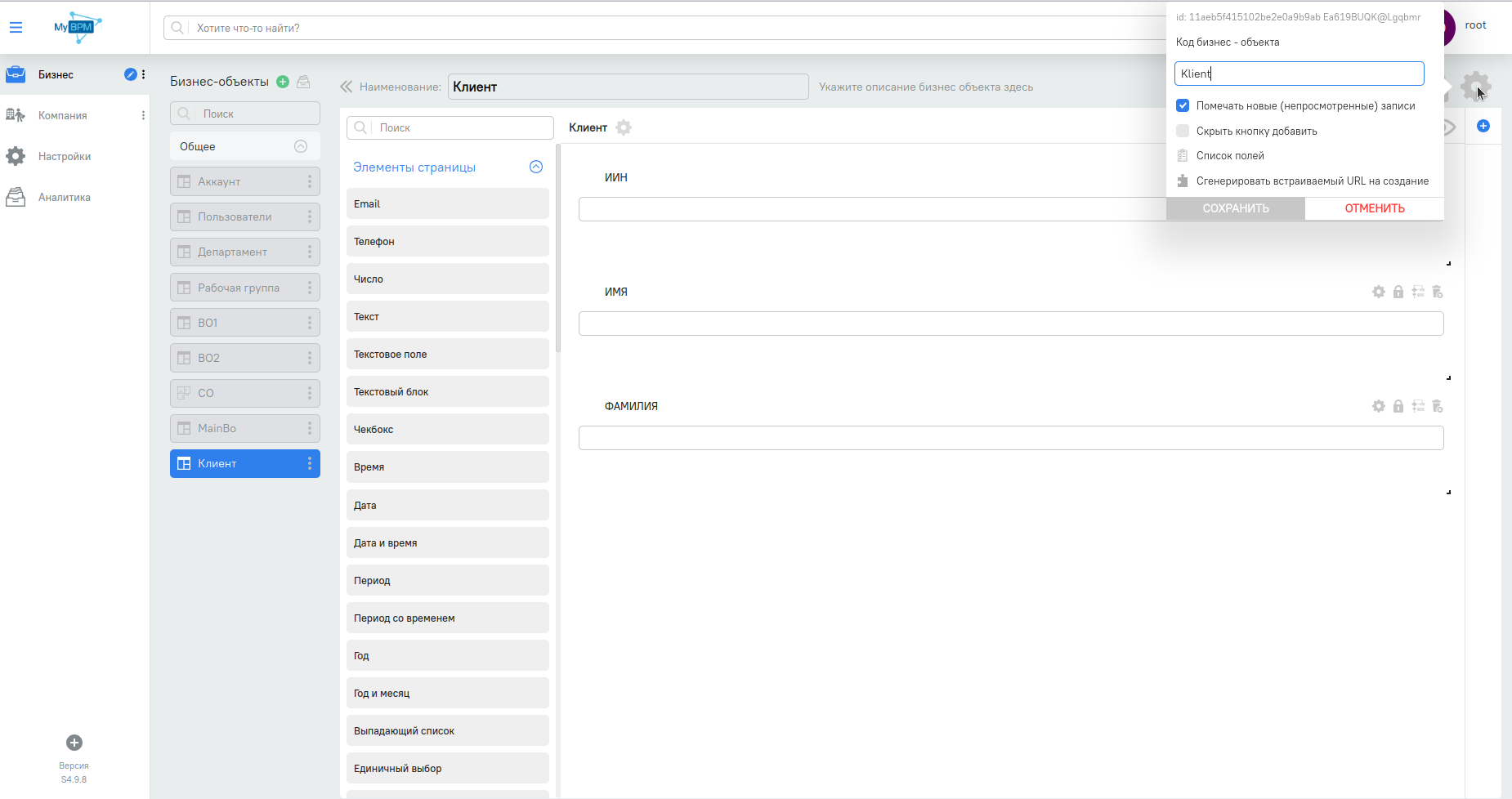


Сохраняем изменения нажатием на кнопку СОХРАНИТЬ.

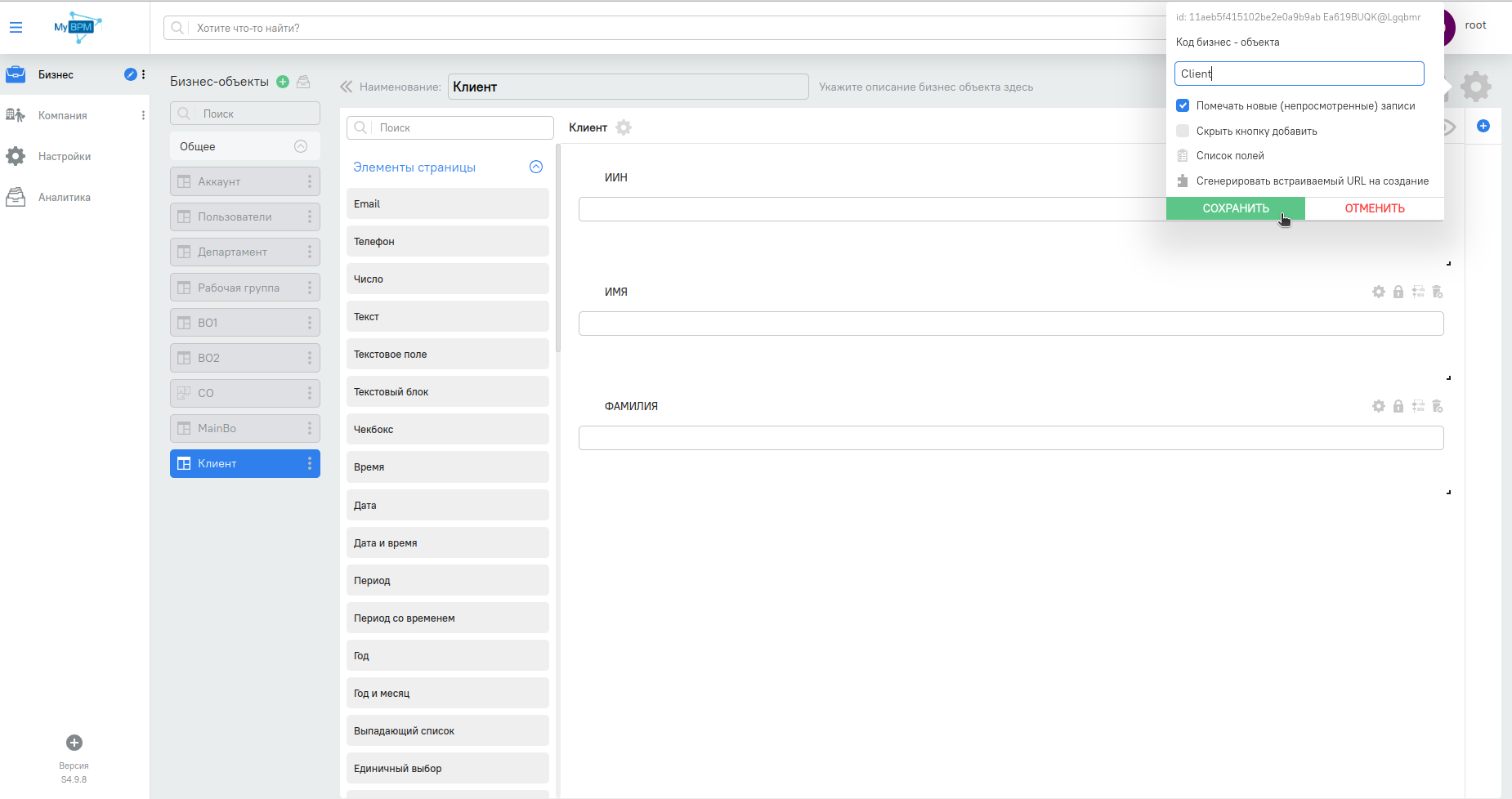


После надо изменить код БО, и полей.

С начало изменим код самого БО, для этого надо нажать на шестиугольник которая находиться в правом верхнем углу.



Проставляем код который мы хотим, допустим будет "Client".

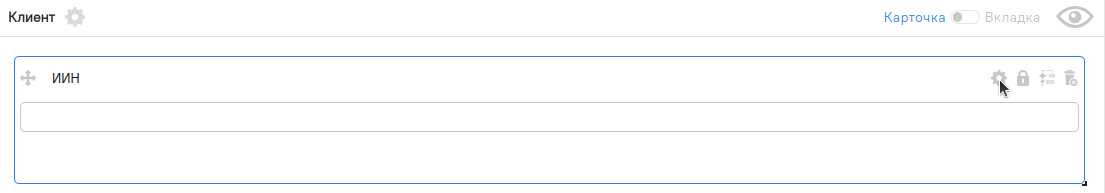


Загорелась зеленая кнопка СОХРАНИТЬ и нажимаем на него.

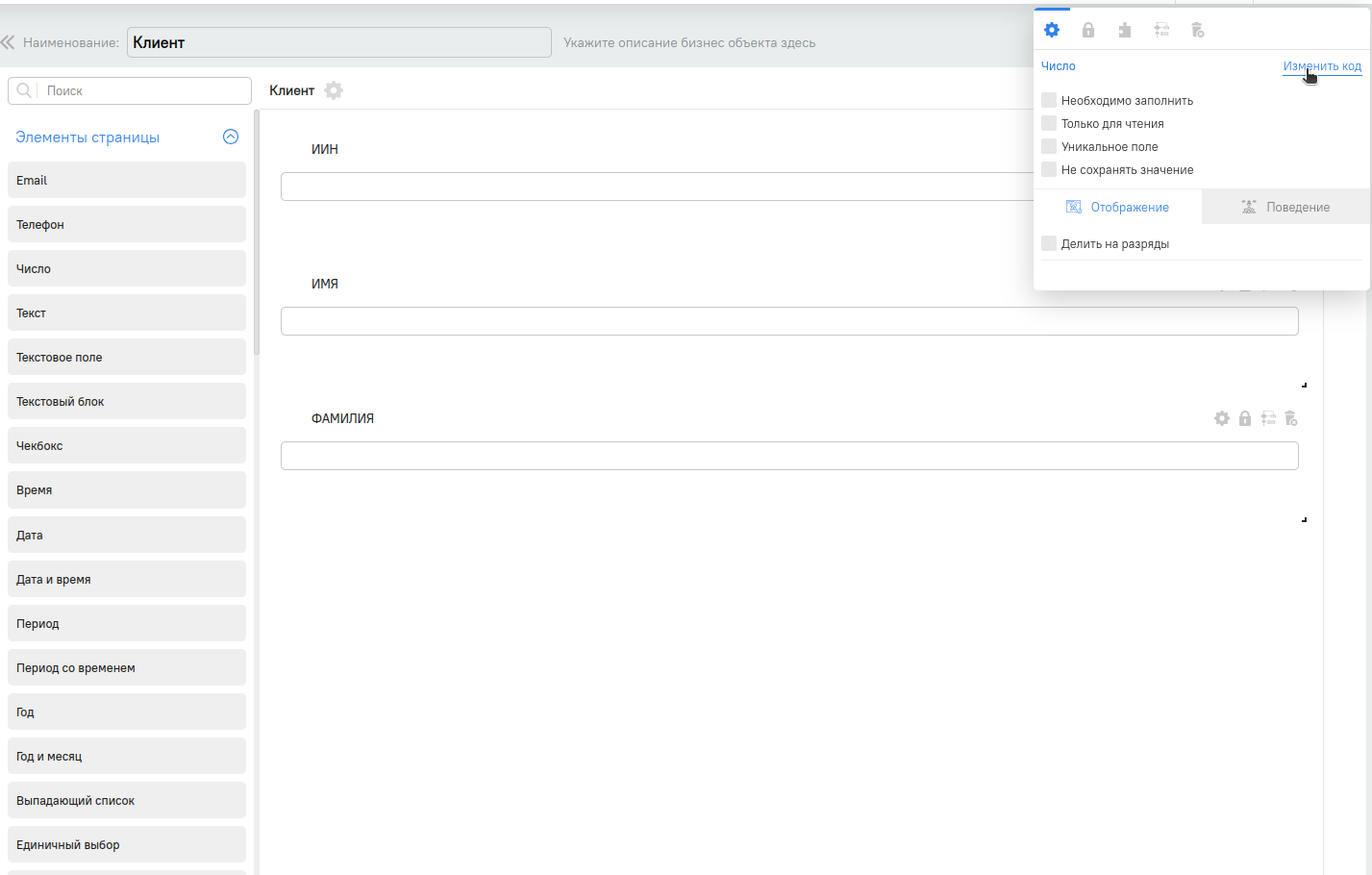
На этом моменте мы изменили код у БО на "Client".

Осталось изменить коды полей данного БО, у каждого поле есть свой настройки находиться правее у поля.

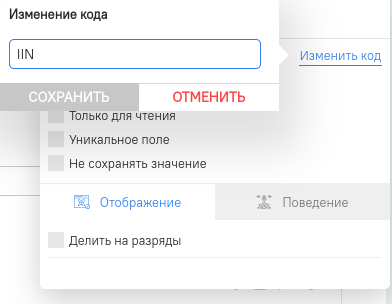
Нажимаем на шестерёнку.



После нажатия должна появиться окошка настроек.



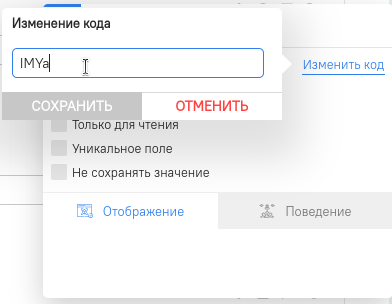
Далее надо кликнуть изменить код.



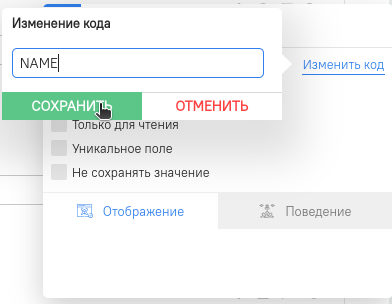
Код данного поле "IIN" такой код поля нас вполне устраивает.

Точно также надо проделать с остальными полями.

Давайте поменяем код у поля ИМЯ, для этого также нажимаем на шестерёнку у плоя ИМЯ затем кликаем на "Изменить код".



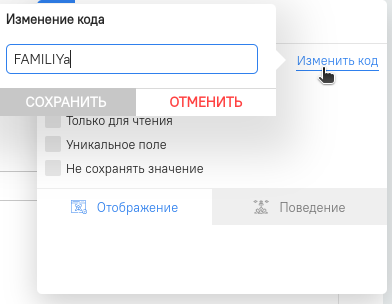
Код данного поле "IMYa" такой код поля нас не устраивает, меняем на "NAME"".



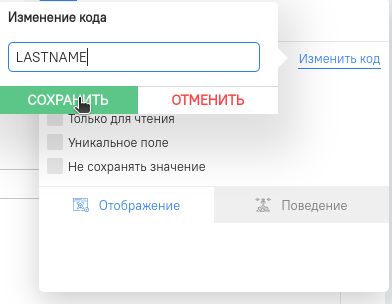
Изменили теперь сохраняем.

Осталось только поле ФАМИЛИЯ давайте приступим.

Для этого точно так же как и у остальных полей нажимаем на шестерёнку у поля ФАМИЛИЯ, далее кликаем изменить код.



Код данного поле "FAMILIYA" такой код поля нас не устраивает, меняем на "LASTNAME".



После того как изменили код на "LASTNAME" нажимаем сохранить.

Поздравляю мы полностью подготовили БО Клиент для миграций через кафку

## Давайте сформируем JSON для БО Клиент.

Как мы уже знаем,

У БО Клиент код равен "Client",

Код поля ИИН равен "IIN",

Код поля ИМЯ равен "NAME",

Код поля ФАМИЛИЯ равен "LASTNAME".

в recordId передаем рандомное значение.  
 в externalId передаем иднетификатор из вашей системы, если таковы имеются.  
 в boCode передаем код нашего БО которое мы определили "Client".  
  
 в fields имеет 2 значения "code", "apiValue".  
 в code передаем код поле ИМЯ, как мы уже определили код поле равен "NAME"  
 в apiValue передаем имя "Сергей".  
 Так как fields это массив передаем код и значения для полей ФАМИЛИЯ и ИИН,  
 как описано выше.  
 в state можем прописать такие значения как "ACTUAL","REMOVED", "ARCHIVED", "TEST".

### Формируем JSON.

{"recordId": "random-value",  
 "externalId": "bfsseVCjGC225bM@",  
 "boCode": "Clinet",  
 "fields": [  
 {  
 "code": "NAME",  
 "apiValue": "Сергей"  
 },  
 {  
 "code" : "LASTNAME",  
 "apiValue" : "Иванов"  
 },  
 {  
 "code" : "IIN",  
 "apiValue" : "17"  
 }  
 ],  
 "state": "ACTUAL"}

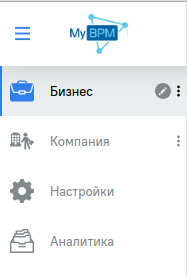
### Надо создать топик для кафки

Отправляем сформированный JSON в топик которую вы создали

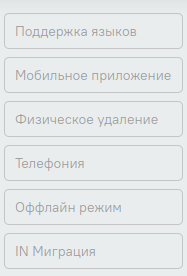
### Осталось сохранить ваш топик в настройках MyBPM.

### Как сохранить топики в настройках MyBPM?

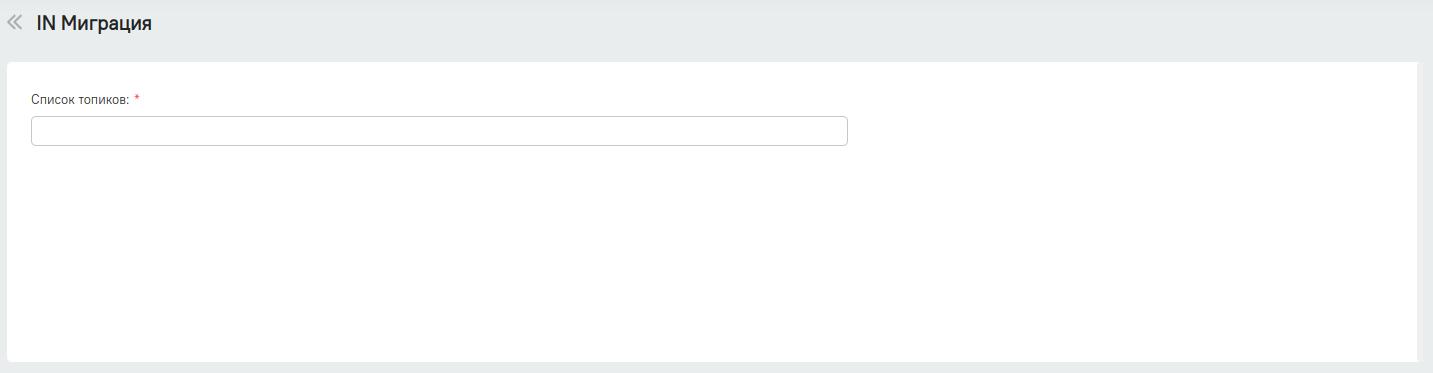
1) В меню кликнуть на **Настройки**.



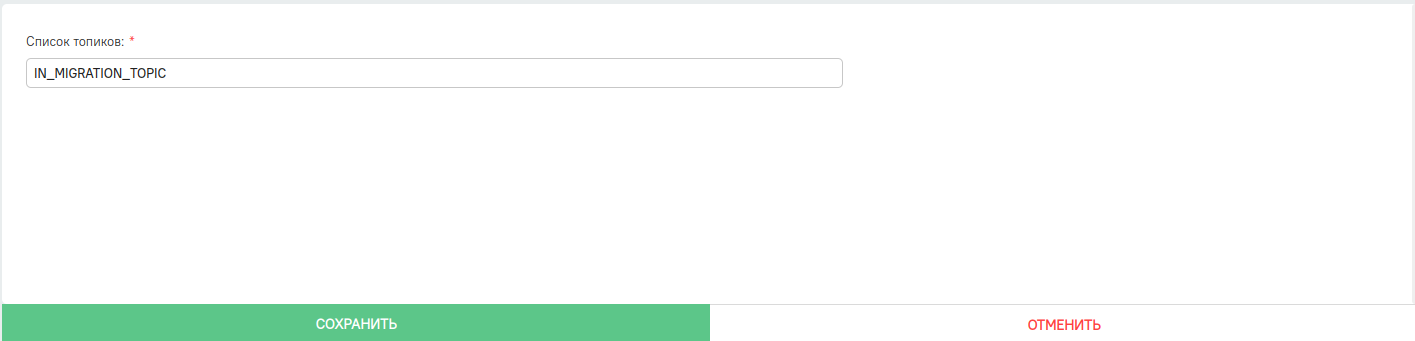
2) Далее появиться окошка настроек.



3) Кликнуть на IN Миграция.



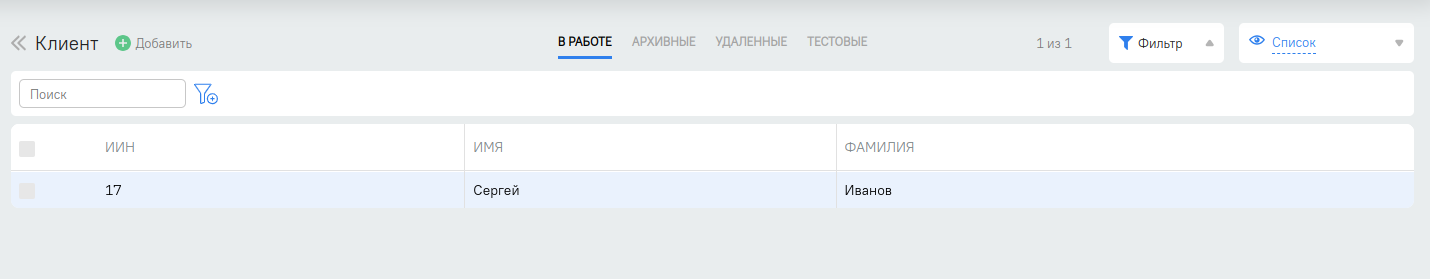
4) Заполнить поле ***Список топиков*** топиками которые вы уже за ранее создали.



5) После заполнение нажмите сохранить.

#### Система уже подписалась на ваши топики и готов мигрировать

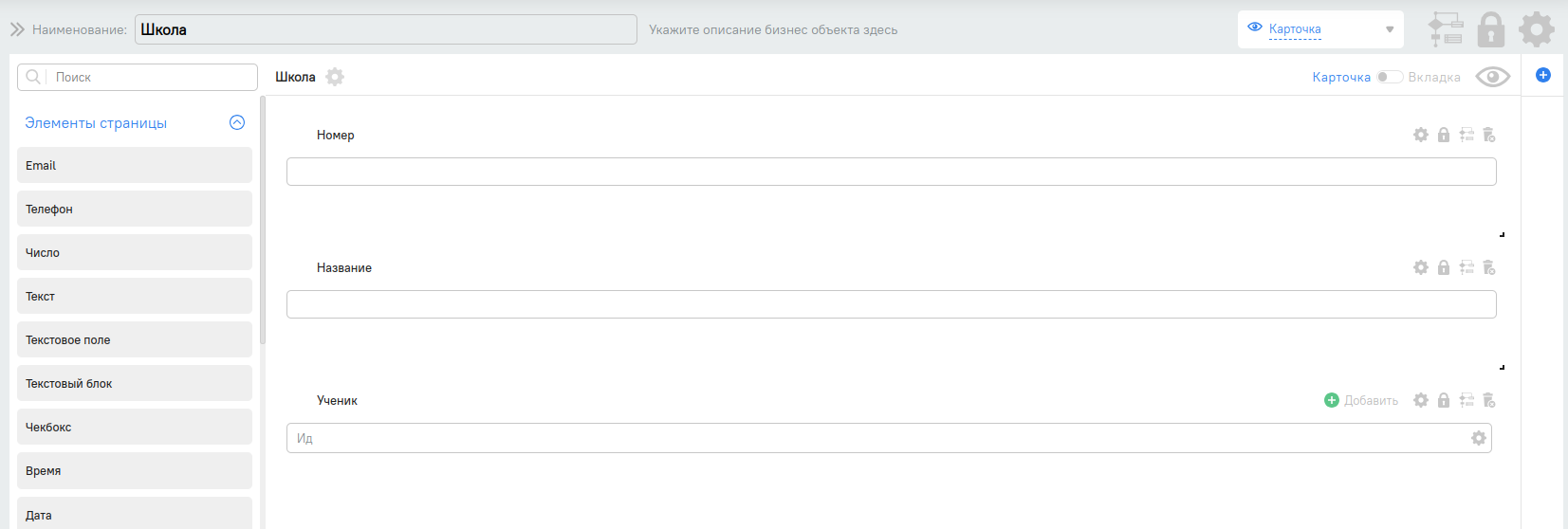
### Результат, появиться новая запись Клиента с ИИН 17, ИМЯ Сергей, ФАМИЛИЯ Иванов.



### Как мигрировать вложенный бизнес объект?

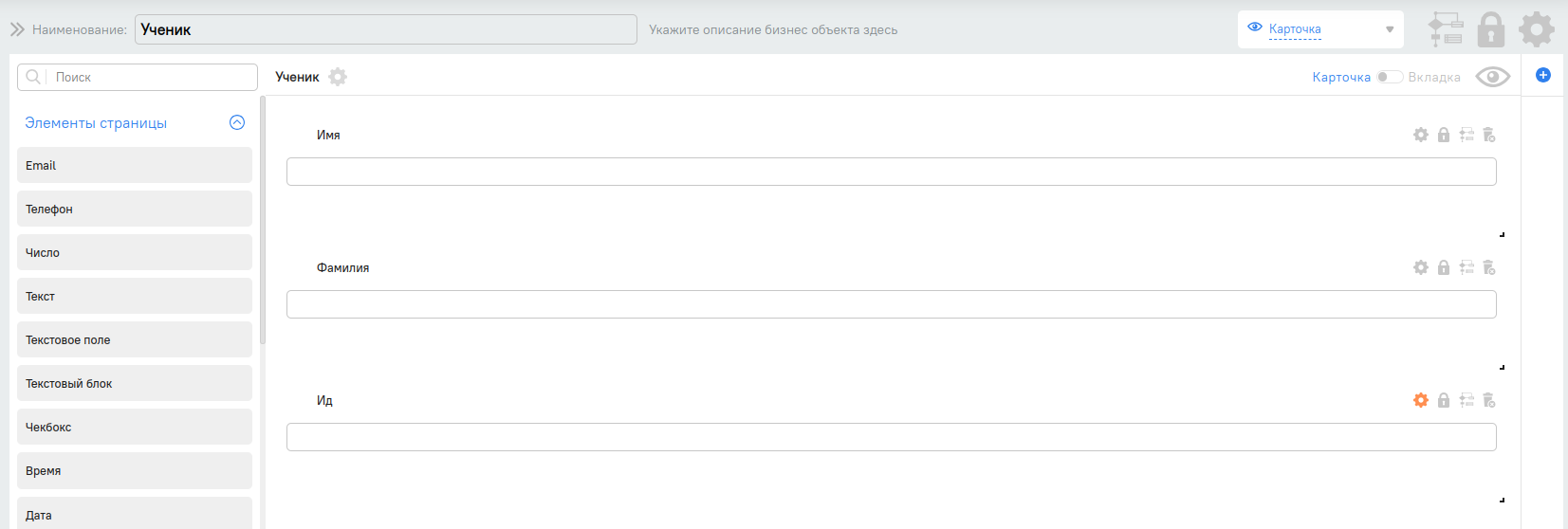
Допустим у нас есть БО Школа и Ученики. Школа может иметь много учеников, получается ONE TO MANY connection.

Структура Школы (Номер, Название, Ученик)



Структура Ученика (Имя, Фамилия, Ид)

Ид уникальное поле с помощью данного поле будем мигрировать учеников в БО Школу



С начало надо смигрировать ученика.

Описанном выше пример с клиентом, мы уже знаем как мигрировать простые Бо здесь точно так же.

Надо получить коды полей и Бо.

Код Бо ученика я поставил - "student"

Код поле Имя - "name"

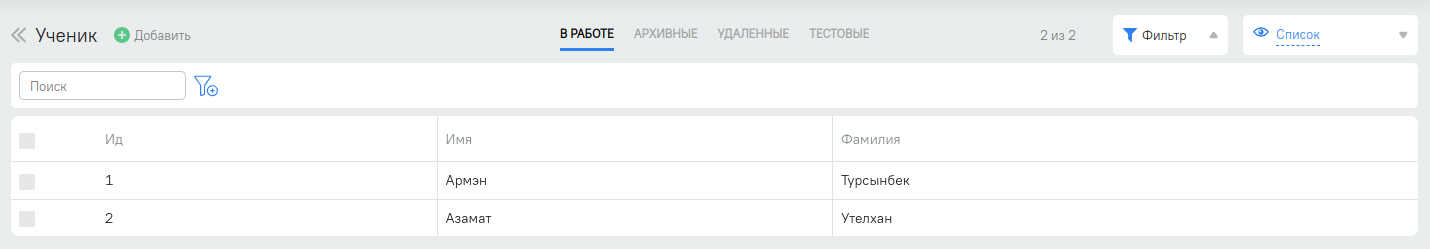
Код поле Фамилия - "surname"

Код поле Ид - "student\_id"

Осталось сформировать JSON и отправить в топик миграций.

var jsonStudent1 = {  
 "recordId": "random-value",  
 "externalId": "7fq@eZ1kpg0iwHVV",  
 "boCode": "student",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Армэн"  
 },  
 {  
 "code" : "surname",  
 "apiValue" : "Турсынбек"  
 },  
 {  
 "code" : "student\_id",  
 "apiValue" : "1"  
 }  
 ],  
 "state": "ACTUAL"  
}  
  
var jsonStudent2 = {  
 "recordId": "random-value",  
 "externalId": "Iqn5iMZcg~M1I5o0",  
 "boCode": "student",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Азамат"  
 },  
 {  
 "code" : "surname",  
 "apiValue" : "Утелхан"  
 },  
 {  
 "code" : "student\_id",  
 "apiValue" : "2"  
 }  
 ],  
 "state": "ACTUAL"  
}

Отправляем сформированный JSON в топик миграций если вы сделали все правильно должно появится две инстанций ученика.



Далее надо смигрировать школу, формируем JSON.

Код Бо Школа - "school"

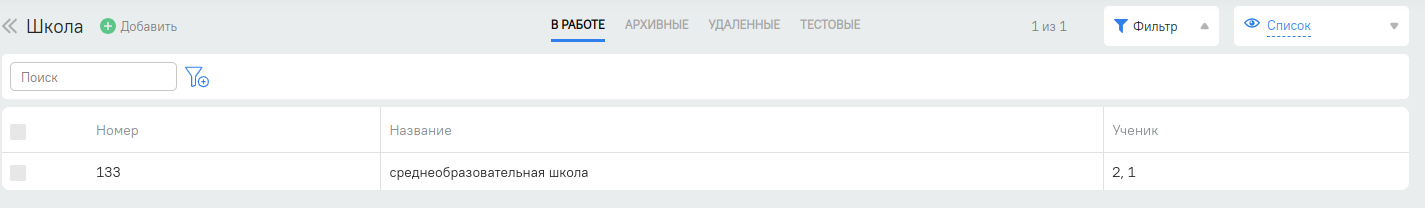
Код поле Номер - "number"

Код поле Название - "name"

Код поле Ученик - "student"

var schoolJson = {  
 "recordId": "jklhdfsksdfjkg",  
 "externalId": "MVLFsYhLv5eVbkXE",  
 "boCode": "school",  
 "fields": [  
 {  
 "code": "number",  
 "apiValue": "133"  
 },  
 {  
 "code": "name",  
 "apiValue": "среднеобразовательная школа"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "student",  
 "toBoCode": "student",  
 "toFieldCode": "student\_id",  
 "apiValue": "1"  
 },  
 {  
 "fieldCode": "student",  
 "toBoCode": "student",  
 "toFieldCode": "student\_id",  
 "apiValue": "2"  
 }  
 ],  
 "state": "ACTUAL"  
}

### Результат создалось запись школы с двумя учениками.



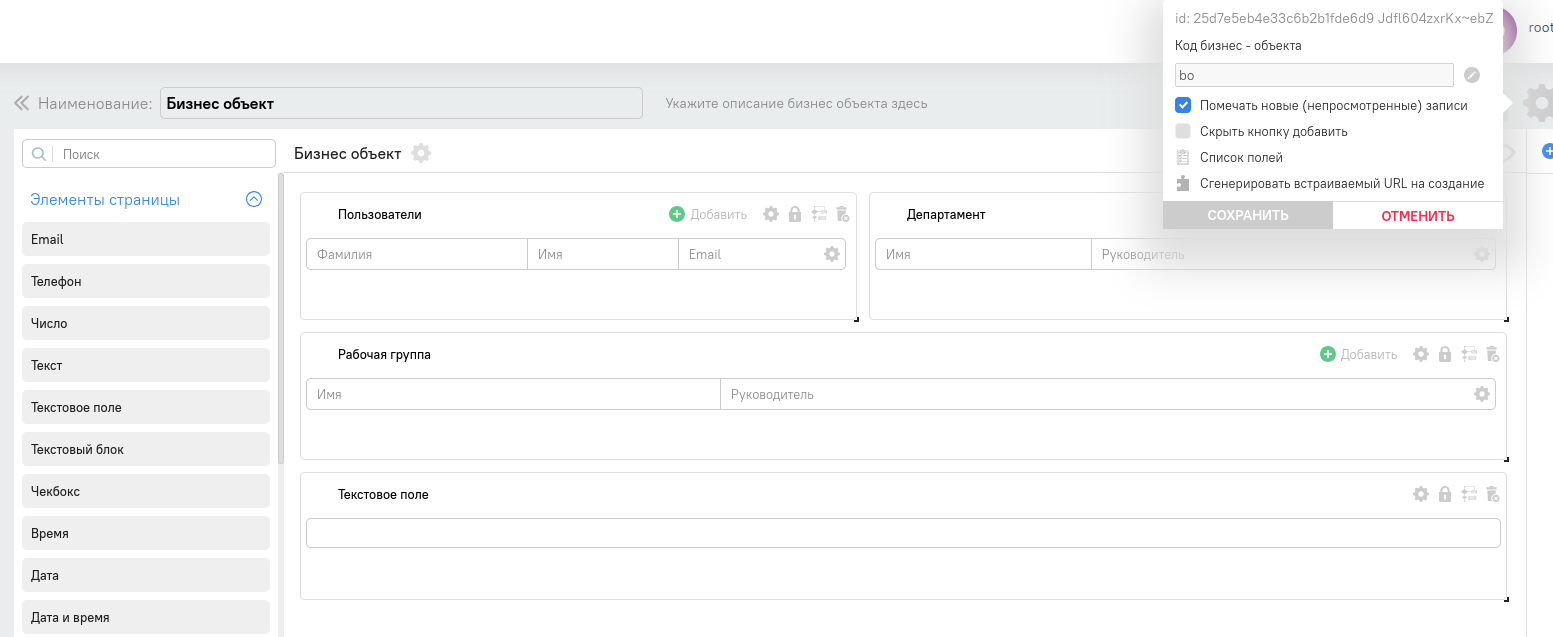
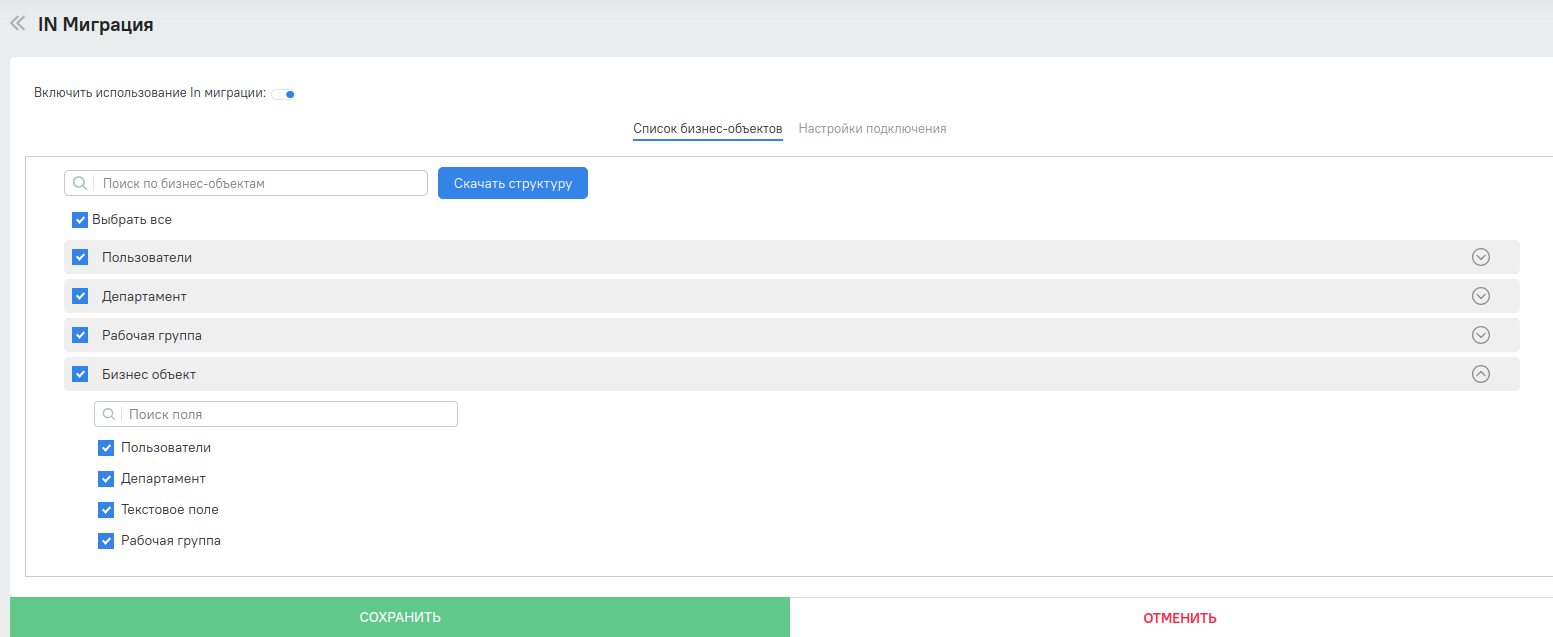
# IN Migration Kafka для Системных БО

Миграция через кафку работает для следующих системных БО:

1. Пользователи
2. Рабочая группа
3. Департамент

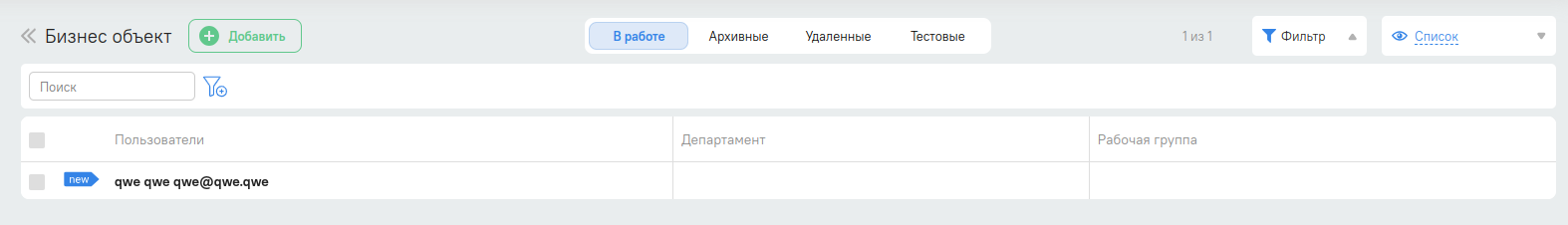
## Примеры использования:

### Подготовка

1. Создадим следующее БО с полями: Текстовое поле и Вложенное БО (Пользователи, Департамент, Рабочая группа): 
2. Выберем эти поля с вложенными системными БО в настройках IN Миграции: 
3. Создадим пользователя c любым email: 

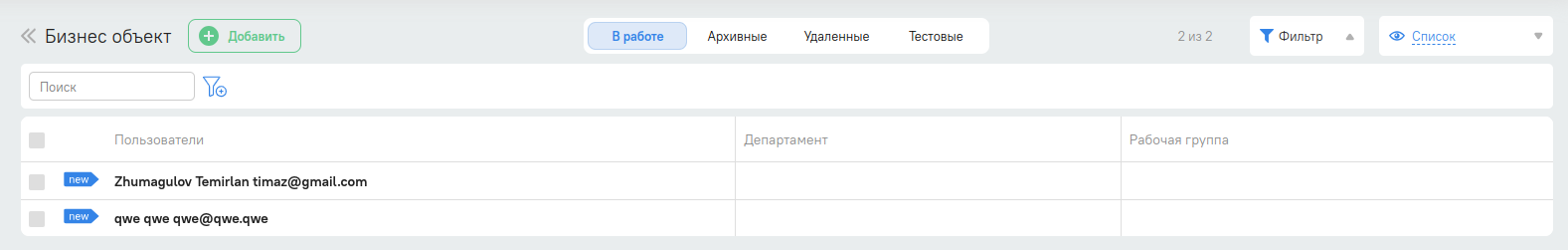
### 1. Создание инстанции БО с вложенным полем БО "Пользователи" с добавлением существующего пользователя по уникальному полю "email"

{  
 "externalId": "bfsseVCjGC225bM@",  
 "id": "dgL9~4keReK7Z1qi",  
 "boCode": "BO",  
 "boFields": [  
 {  
 "fieldCode": "Polzovateli",  
 "toFieldCode": "email",  
 "apiValue": "qwe@qwe.qwe"  
 }  
 ],  
 "state": "ACTUAL"  
}

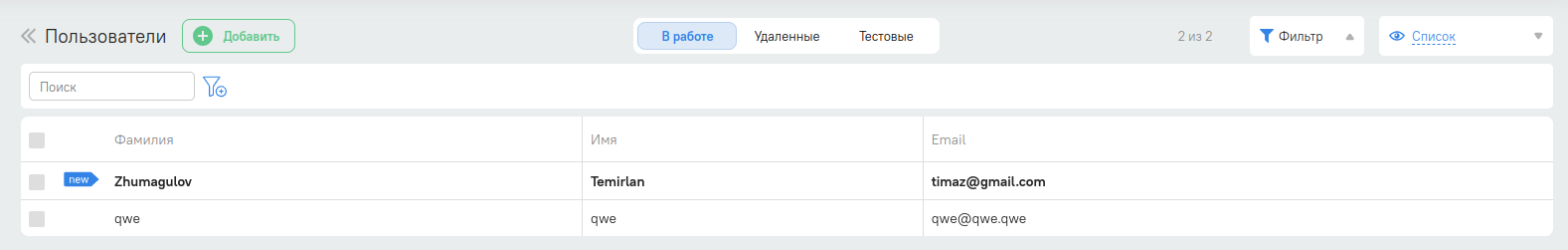


### 2. Миграция инстанции БО вложенной инстанцией БО "Пользователи" который привязан рекурсивно

{  
 "externalId": "cfsseVCjGC225bM@",  
 "id": "cgL9~4keReK7Z1qi",  
 "boCode": "BO",  
 "boFields": [  
 {  
 "fieldCode": "Polzovateli",  
 "boiInput": {  
 "id": "~7TZ9zwn9P3Tj7Xx",  
 "externalId": "fgL9~4keReK7Z1qi",  
 "boCode": "Person",   
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Temirlan"  
 },  
 {  
 "code": "surname",  
 "apiValue": "Zhumagulov"  
 },  
 {  
 "code": "email",  
 "apiValue": "timaz@gmail.com"  
 },  
 {  
 "code": "status",  
 "apiValue": "ACTIVE"  
 },  
 {  
 "code": "accessLevel",  
 "apiValue": "ACCESS\_ALLOWED"  
 }  
 ],  
 "state": "ACTUAL"  
 }  
 }  
 ],  
 "state": "ACTUAL"  
}

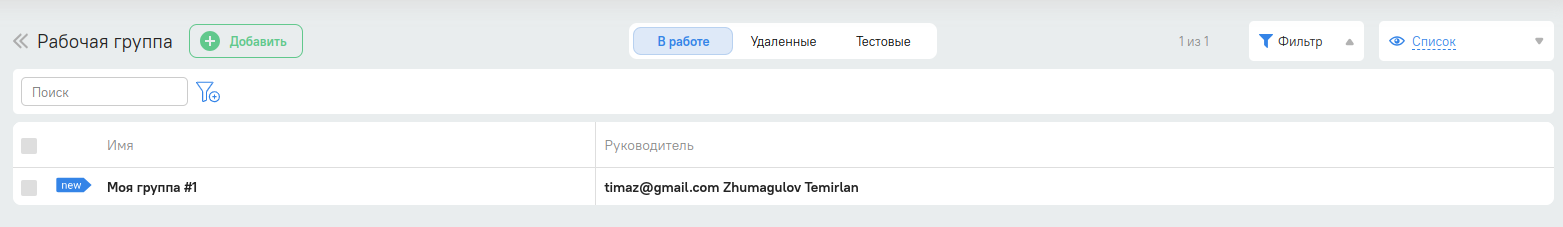
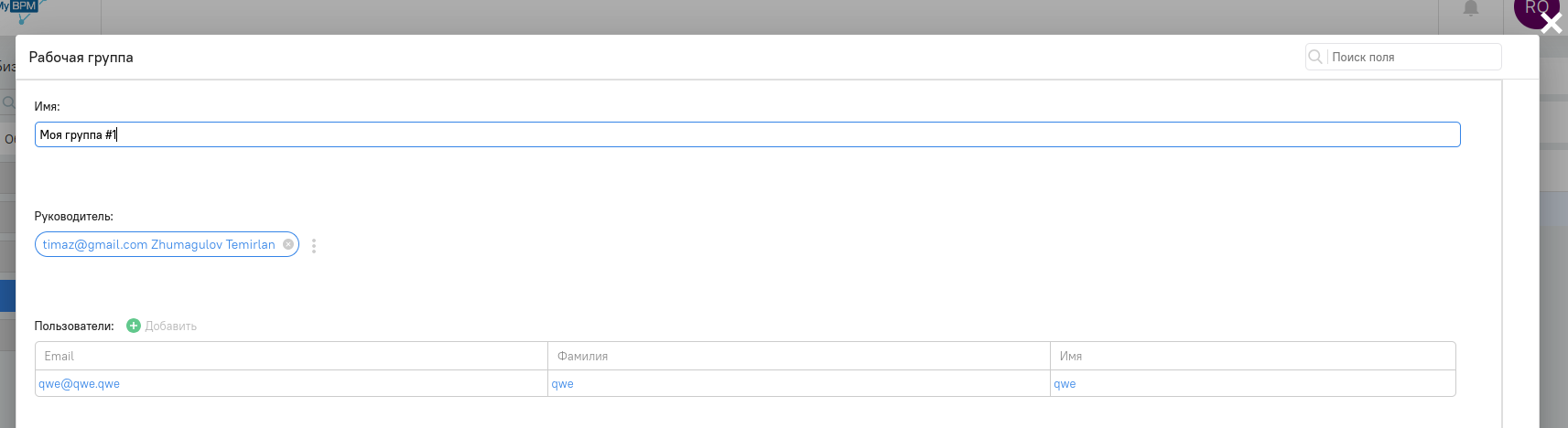


Как видно на примере К БО полю "Пользователи" добавился пользователь, который был создан через миграцию:



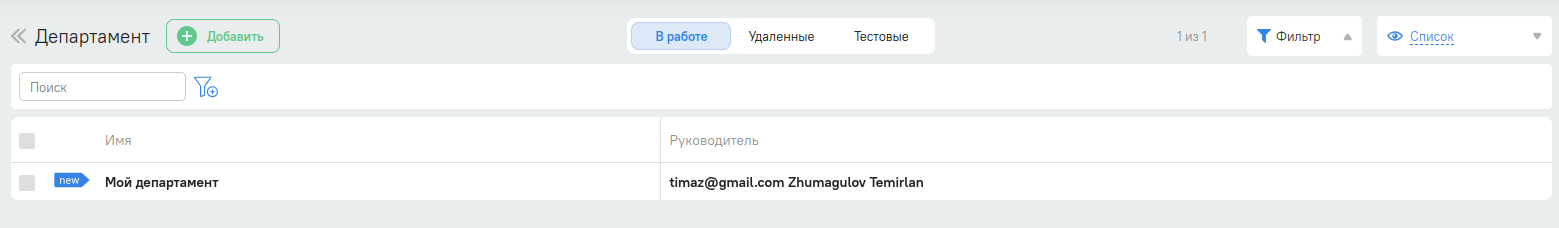
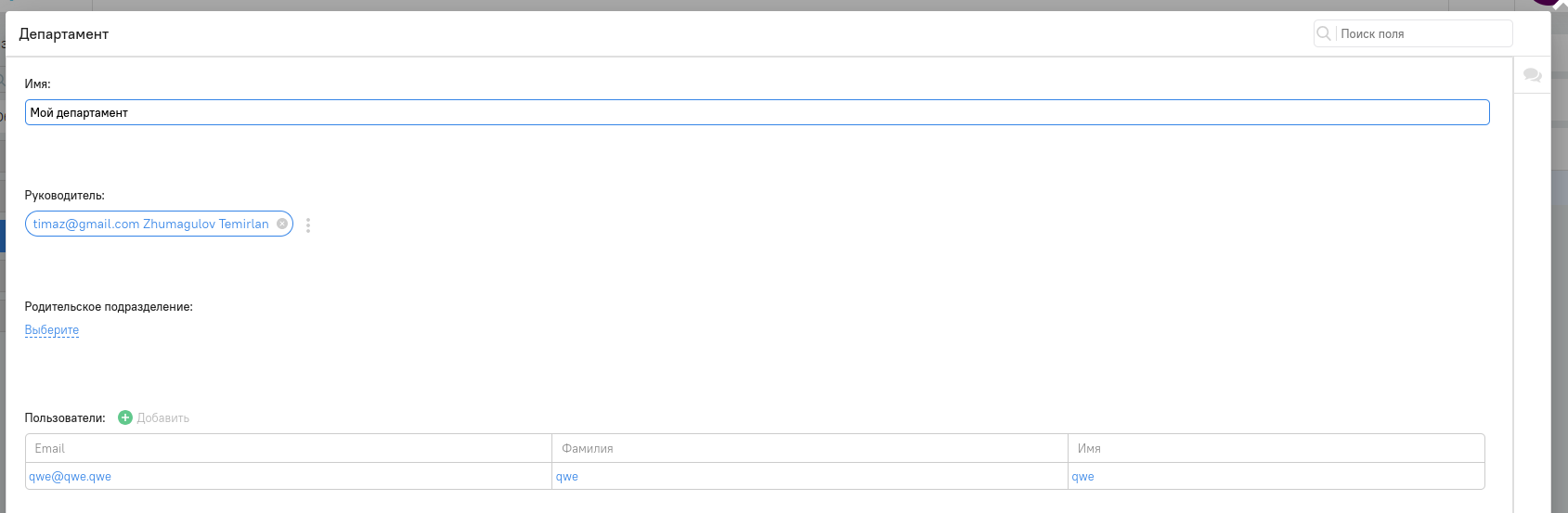
### 3. Миграция инстанции рабочей группы

{  
 "externalId": "zfsseVCjGC225bM@",  
 "id": "zgL9~4keReK7Z1qi",  
 "boCode": "PersonGroup",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Моя группа #1"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "header",  
 "toFieldCode": "email",  
 "apiValue": "timaz@gmail.com"  
 },  
 {  
 "fieldCode": "users",  
 "toFieldCode": "email",  
 "apiValue": "qwe@qwe.qwe"  
 }  
 ],  
 "state": "ACTUAL"  
 }

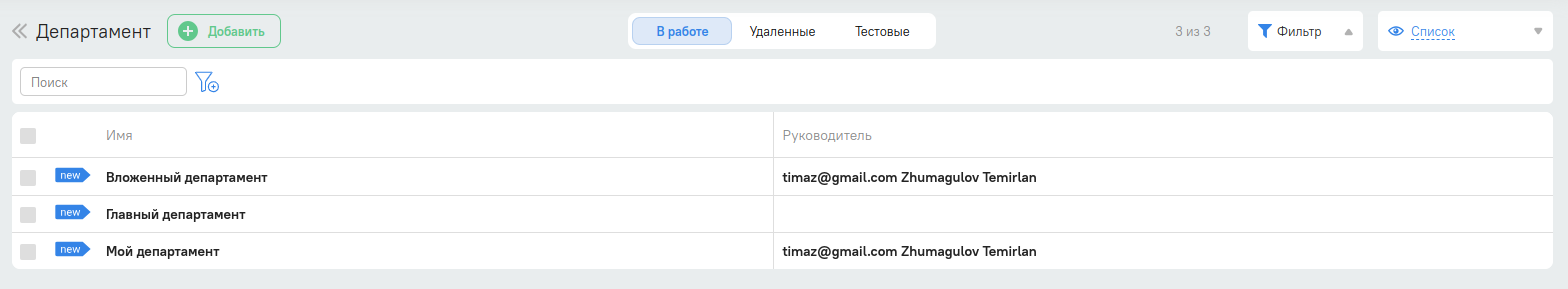
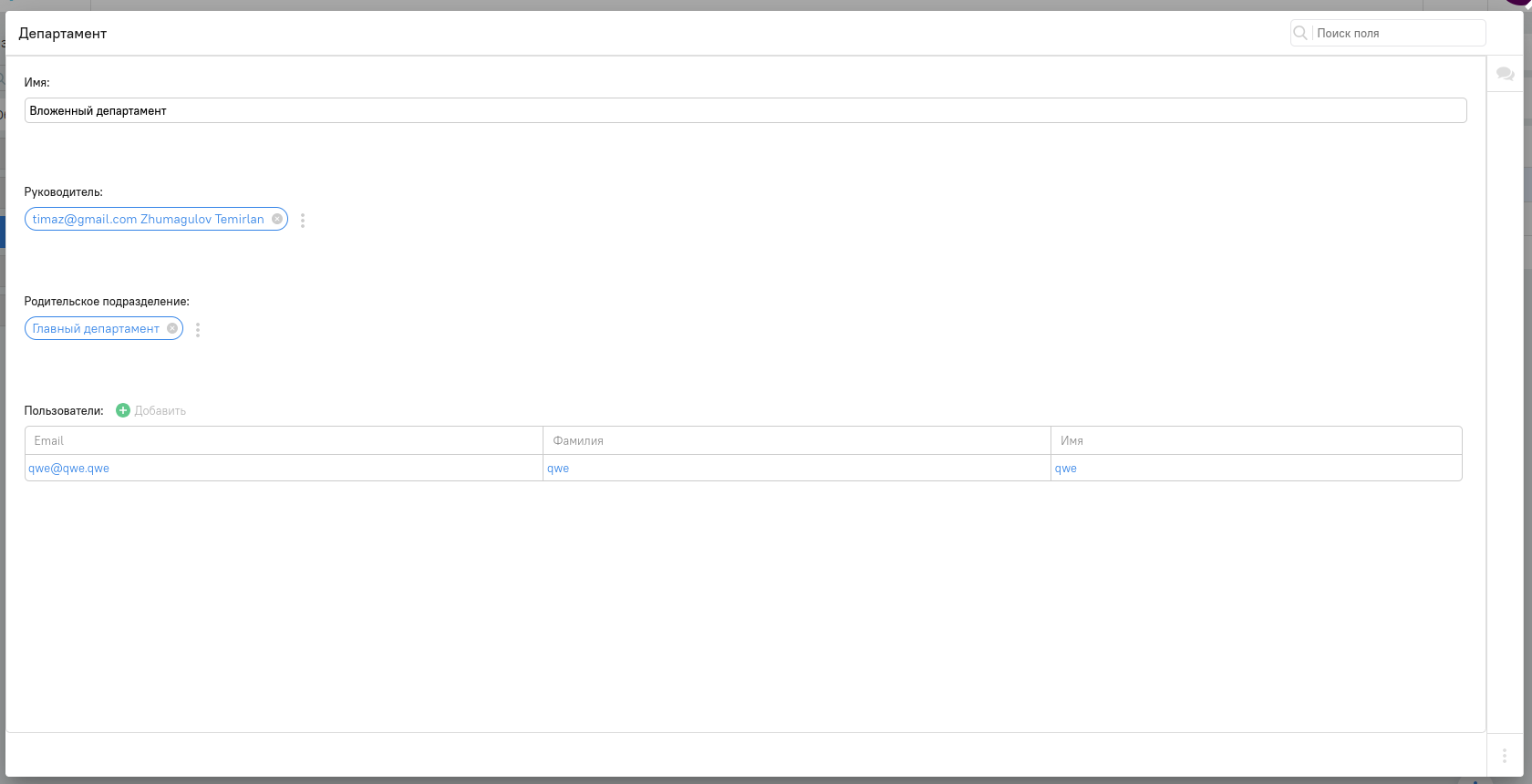
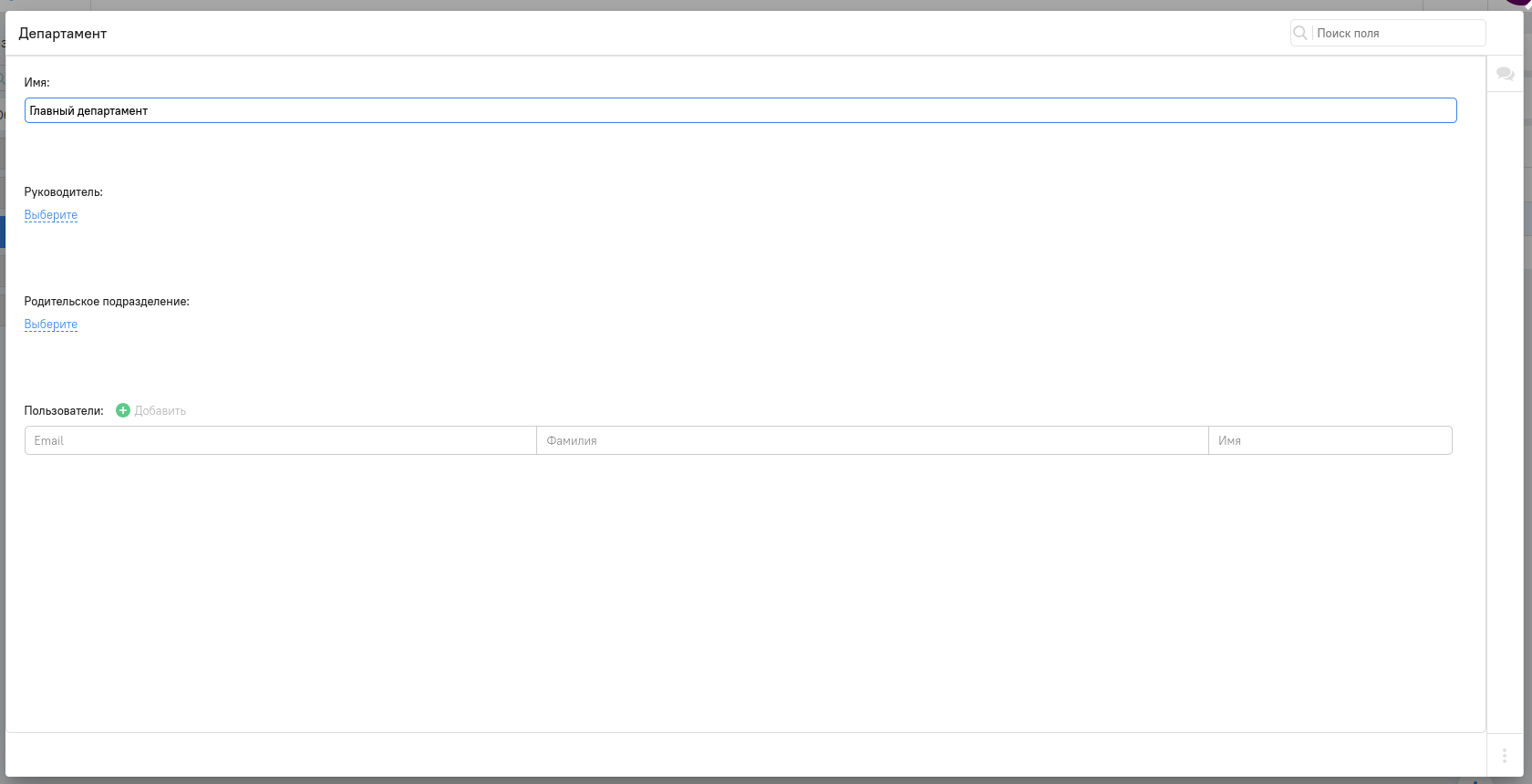
### 4. Миграция инстанции Департамента

{  
 "externalId": "hfsseVCjGC225bM@",  
 "id": "hgL9~4keReK7Z1qi",  
 "boCode": "Department",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Мой департамент"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "header",  
 "toFieldCode": "email",  
 "apiValue": "timaz@gmail.com"  
 },  
 {  
 "fieldCode": "person",  
 "toFieldCode": "email",  
 "apiValue": "qwe@qwe.qwe"  
 }  
 ],  
 "state": "ACTUAL"  
 }

### 5. Миграция инстанции Департамента с родительским подразделением вложенным рекурсивно

{  
 "externalId": "kfsseVCjGC225bM@",  
 "id": "kgL9~4keReK7Z1qi",  
 "boCode": "Department",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Вложенный департамент"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "header",  
 "toFieldCode": "email",  
 "apiValue": "timaz@gmail.com"  
 },  
 {  
 "fieldCode": "person",  
 "toFieldCode": "email",  
 "apiValue": "qwe@qwe.qwe"  
 },  
 {  
 "fieldCode": "parent",  
 "boiInput": {  
 "externalId": "pfsseVCjGC225bM@",  
 "id": "pgL9~4keReK7Z1qi",  
 "boCode": "Department",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Главный департамент"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "header",  
 "toFieldCode": "email",  
 "apiValue": "alikhan@gmail.com"  
 },  
 {  
 "fieldCode": "person",  
 "toFieldCode": "email",  
 "apiValue": "yerkebulan@gmail.com"  
 }  
 ],  
 "state": "ACTUAL"  
 }  
 }  
 ],  
 "state": "ACTUAL"  
 }

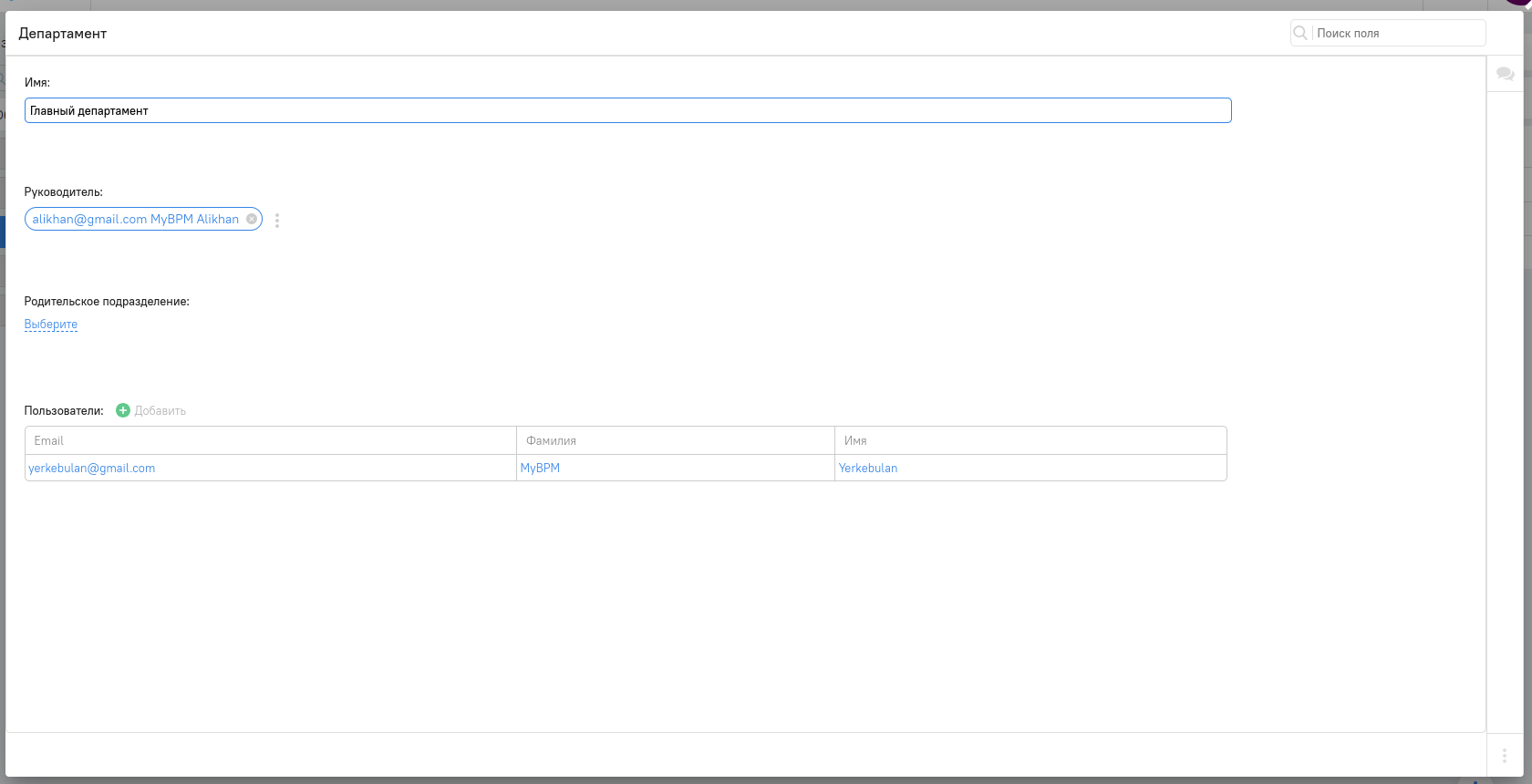
  

Однако пользователей с email-ами **(**[**alikhan@gmail.com**](mailto:alikhan@gmail.com)**,** [**yerkebulan@gmail.com**](mailto:yerkebulan@gmail.com)**)** в системе нет, поэтому мы их добавим через миграцию:

{  
 "id": "~8TZ9zwn9P3Tj7Xx",  
 "externalId": "~gL9~4keReK7Z1qi",  
 "boCode": "Person",   
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Alikhan"  
 },  
 {  
 "code": "surname",  
 "apiValue": "MyBPM"  
 },  
 {  
 "code": "email",  
 "apiValue": "alikhan@gmail.com"  
 },  
 {  
 "code": "status",  
 "apiValue": "ACTIVE"  
 },  
 {  
 "code": "accessLevel",  
 "apiValue": "ACCESS\_ALLOWED"  
 }  
 ],  
 "state": "ACTUAL"  
}

После отправки сообщения, отправляем это сообщение для добавления пользователя ([yerkebulan@gmail.com](mailto:yerkebulan@gmail.com)):

{  
 "id": "~9TZ9zwn9P3Tj7Xx",  
 "externalId": "~mL9~4keReK7Z1qi",  
 "boCode": "Person",   
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Yerkebulan"  
 },  
 {  
 "code": "surname",  
 "apiValue": "MyBPM"  
 },  
 {  
 "code": "email",  
 "apiValue": "yerkebulan@gmail.com"  
 },  
 {  
 "code": "status",  
 "apiValue": "ACTIVE"  
 },  
 {  
 "code": "accessLevel",  
 "apiValue": "ACCESS\_ALLOWED"  
 }  
 ],  
 "state": "ACTUAL"  
}



## Особенности

* В отличии от общих БО, инстанции системных БО не могут быть пустыми, то есть иметь 0 изменении по полям

# OUT Migration Kafka концепт

## Каким образом производится внешняя миграция через Apache Kafka?

Внешняя миграция через кафку производится таким способом, что в кафке создаётся топик с каким-то названием, дальше этот топик регистрируется в системе и после регистрации, система обращается в данный топик и начинает записывать различные события (создание новой инстанции, обновление инстанции и тд.) в виде JSON определенного ниже формата по выбранным бизнес объектам (БО) в настройках out миграции,

Детальнее это показано [здесь](001-out-migration-kafka-example.md)

## Важные нюансы миграции:

1. Уникальные поля передаются всегда, вне зависимости выбрали ли вы их в настройках или нет, и вне взависимости есть ли по ним какие-то изменения или нет. Если по ним нет изменении, они передаются как **unchangedUniqueFields** или **unchangedUniqueBoFields**, если изменения есть то они передаются как обычно в **fields** или **boFields**. Но учтите, что в случае если у уникального поля нет значения или если она системная, то она не передается.
2. Важно понимать, что система не имеет возможности отличить создание инстанции БО от её изменения, поэтому информации этом в выгружаемых JSON-данных не поступает. Поэтому в данными стоит работать в режиме

* UPSERT или   
  INSERT-ON-DUPLICATE-UPDATE
* Режим UPSERT предполагает, что вначале нужно проверить наличие инстанции, если таковая есть, то обновить её. Иначе - создать новую.
* Режим INSERT-ON-DUPLICATE-UPDATE предполагает, что всегда нужно создавать новую запись. Предварительно нужно предусмотреть чтобы идентификатор инстанции является уникальным полем (например первичным ключом). При этом стоит ожидать ошибку, что такая запись уже есть. И если эта ошибка появилась, то подавить её и выполнить обновление инстанции.

### Структура JSON, каким оно должно быть.

{  
 "recordId": "12345",  
 "externalId": "bfsseVCjGC225bM@",  
 "id": "dgL9~4keReK7Z1qi",  
 "boCode": "BO",  
 "fields": [  
 {  
 "code": "textField",  
 "apiValue": "someValue"  
 },  
 {  
 "code": "textField2",  
 "apiValue": "someValue2"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "boTextField",  
 "toBoCode": "BO2",  
 "toFieldCode": "textFieldOfBO2",  
 "apiValue": "015252634519"  
 },  
 {  
 "fieldCode": "boTextField",  
 "toBoCode": "BO3",  
 "toFieldCode": "textFieldOfBO3",  
 "apiValue": "015256324519"  
 }  
 ],  
 "unchangedUniqueFields": [  
 {  
 "code": "uniqueDynamicField",  
 "apiValue": "someValue"  
 }  
 ],  
 "unchangedUniqueBoFields": [  
 {  
 "fieldCode": "uniqueBoField",  
 "toBoCode": "BO2",  
 "toFieldCode": "textFieldOfBO2",  
 "apiValue": "015252634519"  
 }  
 ],  
 "state": "ACTUAL"  
}

### Подробнее о полях

recordId:Строка - Идентификатор данной записи, он должен быть уникальным в рамках всего топика.   
 Он используется для того чтобы запись обрабатывалась всего одни раз его затем,  
 система его запоминает и при повторной записи система его игнорирует.  
  
externalId:Строка - Идентификатор внешней системы, которая загрузила данные на платформу MyBPM. Если инстанция была  
 создана внутри платформы, то данный идентификатор будет пустым.  
  
id:Строка - Идентификатор инстанции БО - внутренний для платформы MyBPM.  
  
boCode:Строка - Код бизнес объекта в которой, должны записаться данные.  
  
fields:Массив - Поля со значениями для мигрированния .  
 code:Строка - Код поля в бизнес объекте .  
 apiValue:Строка - Значение из REST API.  
  
boFields:Массив - Бизнес поля для миграций (вложенный бизнес объект).  
 fieldCode:Строка - Код поля, который принадлежит с кодом boCode.  
 toBoCode:Строка - Код бизнес объекта на которое указывает данное поле.  
 toFieldCode:Строка - Код поля у бизнес объекта toBoCode. Это поле должно быть уникальное   
 apiValue:Строка - Значение поле toFieldCode у бизнес объекта с кодом toBoCode.   
  
unchangedUniqueFields:Массив - Объекты как в fields, но только для неизмененных уникальных полей.  
 code:Строка - Код поля в бизнес объекте .  
 apiValue:Строка - Значение из REST API.  
  
unchangedUniqueBoFields:Массив - Объекты как в boFields, но только для неизмененных уникальных полей.  
 fieldCode:Строка - Код поля, который принадлежит с кодом boCode.  
 toBoCode:Строка - Код бизнес объекта на которое указывает данное поле.  
 toFieldCode:Строка - Код поля у бизнес объекта toBoCode. Это поле должно быть уникальное   
 apiValue:Строка - Значение поле toFieldCode у бизнес объекта с кодом toBoCode.  
  
state:Строка - Состояние записи могут быть (ACTUAL, REMOVED, ARCHIVED, TEST), может отсутствовать,   
 если записи нету то state станет ACTUAL, а если запись есть то ничего не измениться   
 ACTUAL - Это актуальная запись которая доступна для работы.  
 REMOVED - Удалённая запись.  
 ARCHIVED - Архивированная запись.  
 TEST - Запись предназначена для режима тестирования

# OUT Migration Kafka пример

### Пример как пройсходит миграция поле

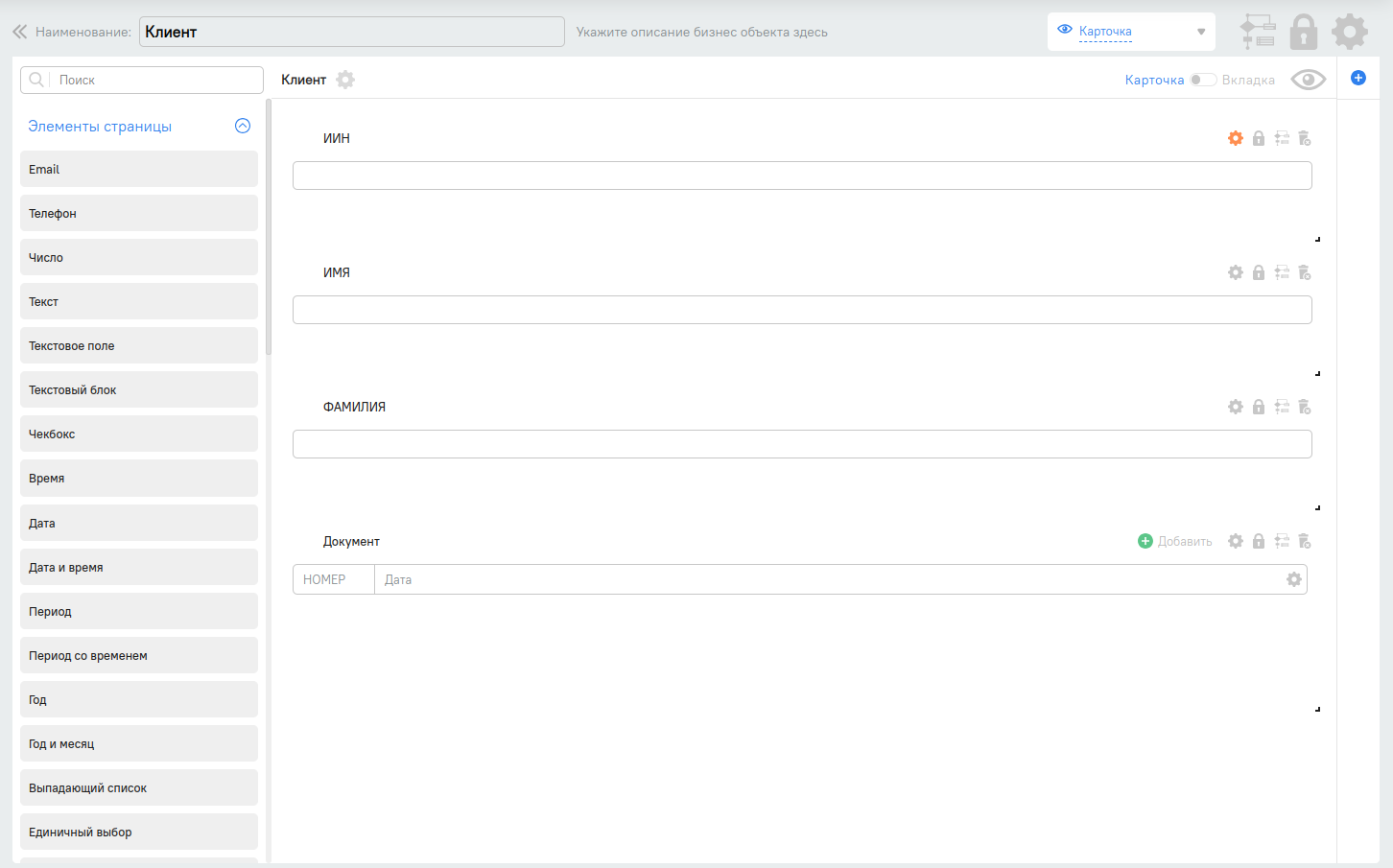
#### Самое начало

Термин: БО - Бизнес Объект.

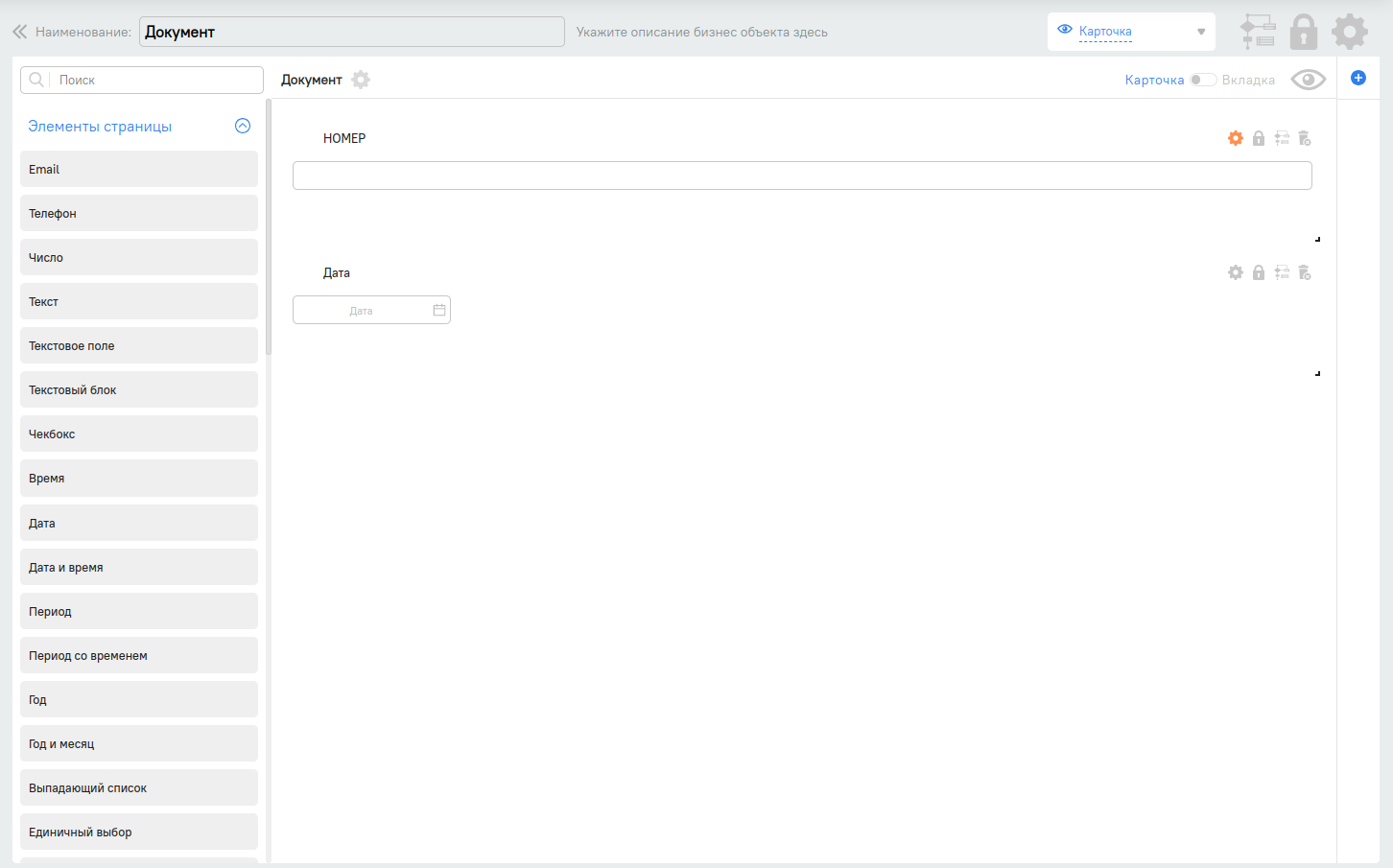
Допустим в системе есть БО Клиент с полями ИИН, ИМЯ, ФАМИЛИЯ, ДОКУМЕНТ.  
ИИН - уникальное поле,ДОКУМЕНТ является вложенным БО.  
Поля БО Документ - НОМЕР, ДАТА. НОМЕР - уникальное поле.

По внешней миграций хотим получить все события (создание, обновление, удаление инстанций БО и тд.)

Структура БО Клиент

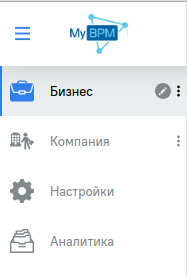


Структура БО Документ

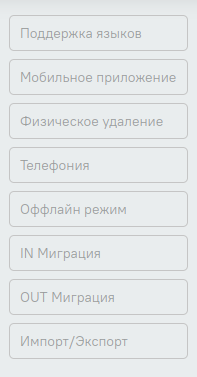


Для того чтобы миграция началась надо настройть внешнюю миграцию.

Для этого переходим в раздел Настройки в главном меню.

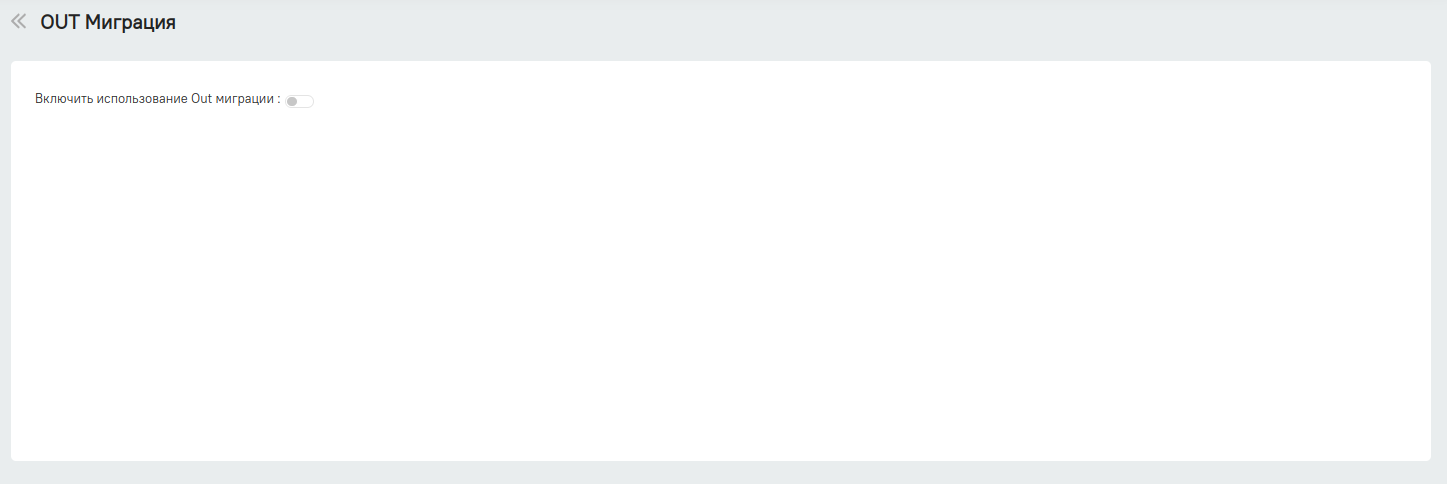


Далее кликаем на Настройки

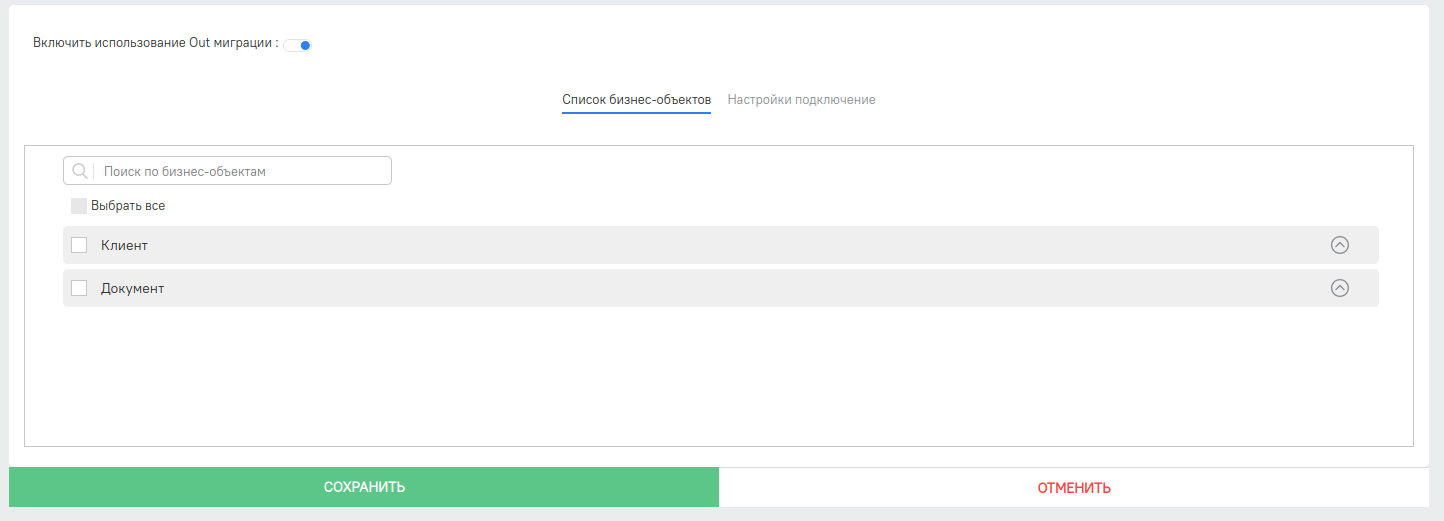


Видим раздел OUT Миграция кликаем.

Кликнув видим пустую настройку.

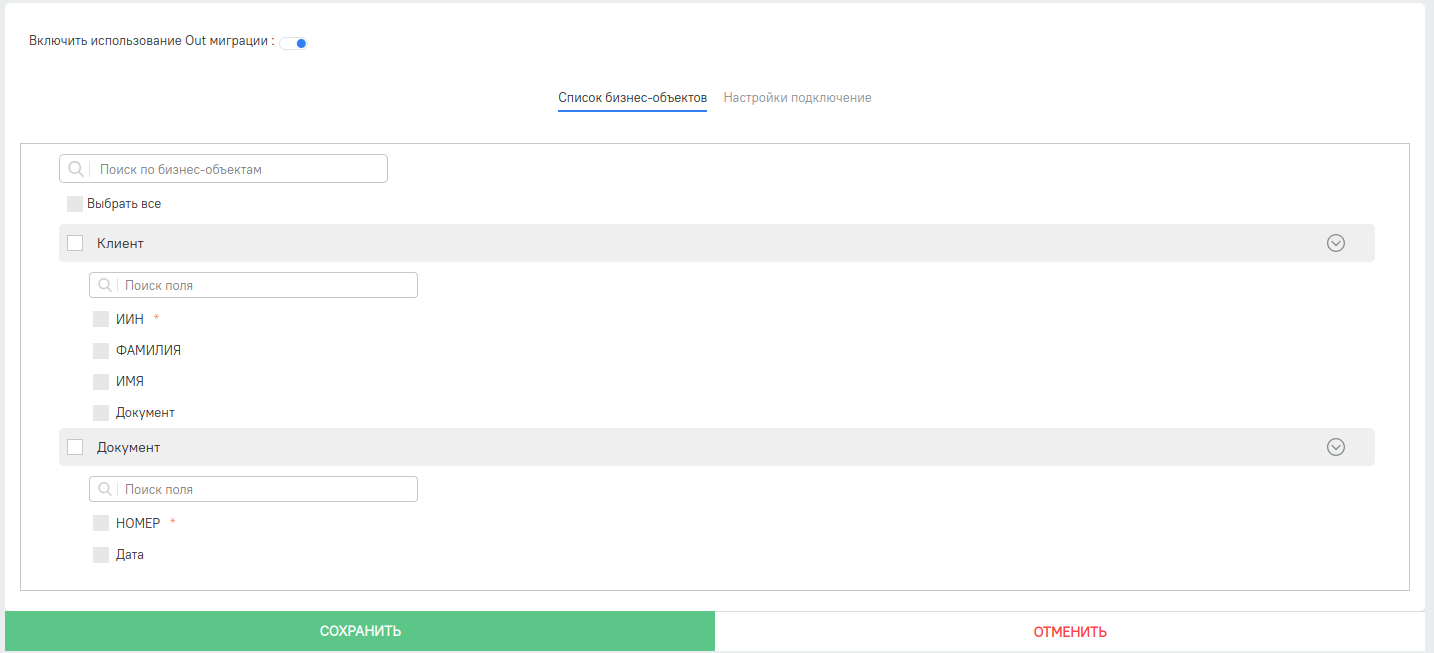


Нажимаем на чекбокс, чтобы активировать настройку.

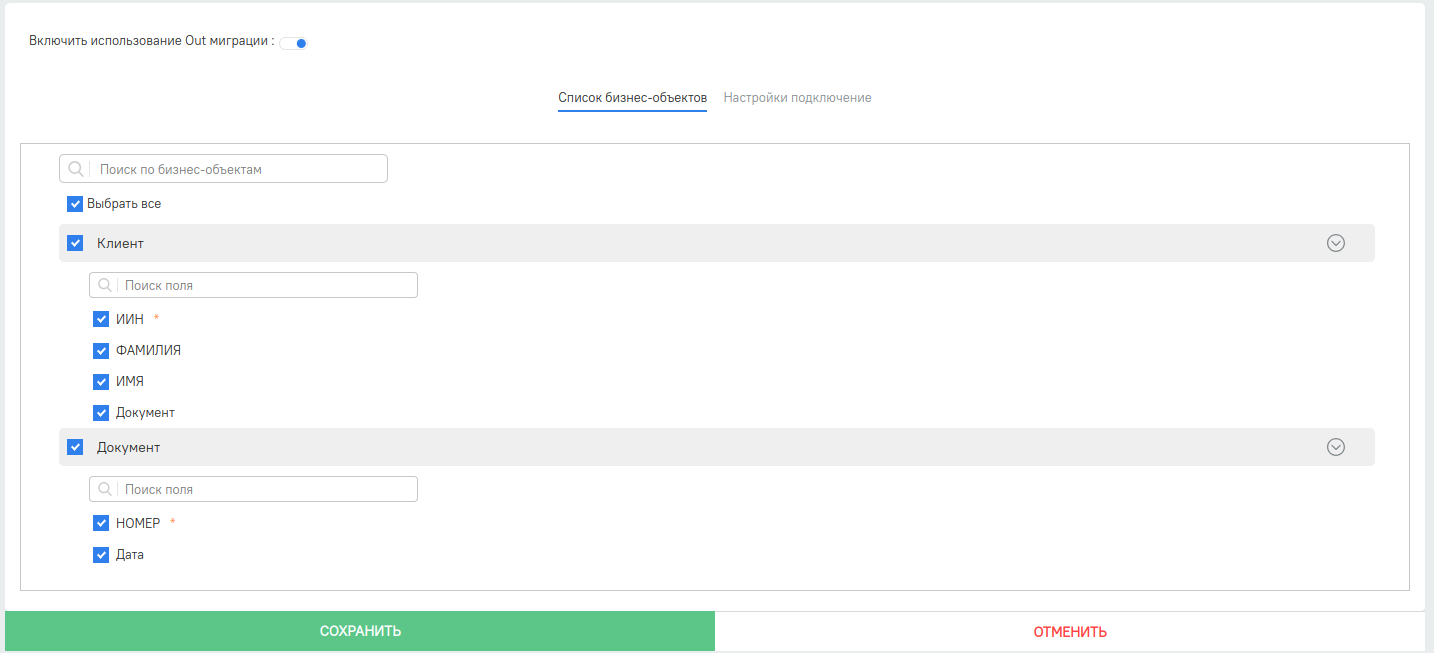


Видим два БО Клиент и Документ.

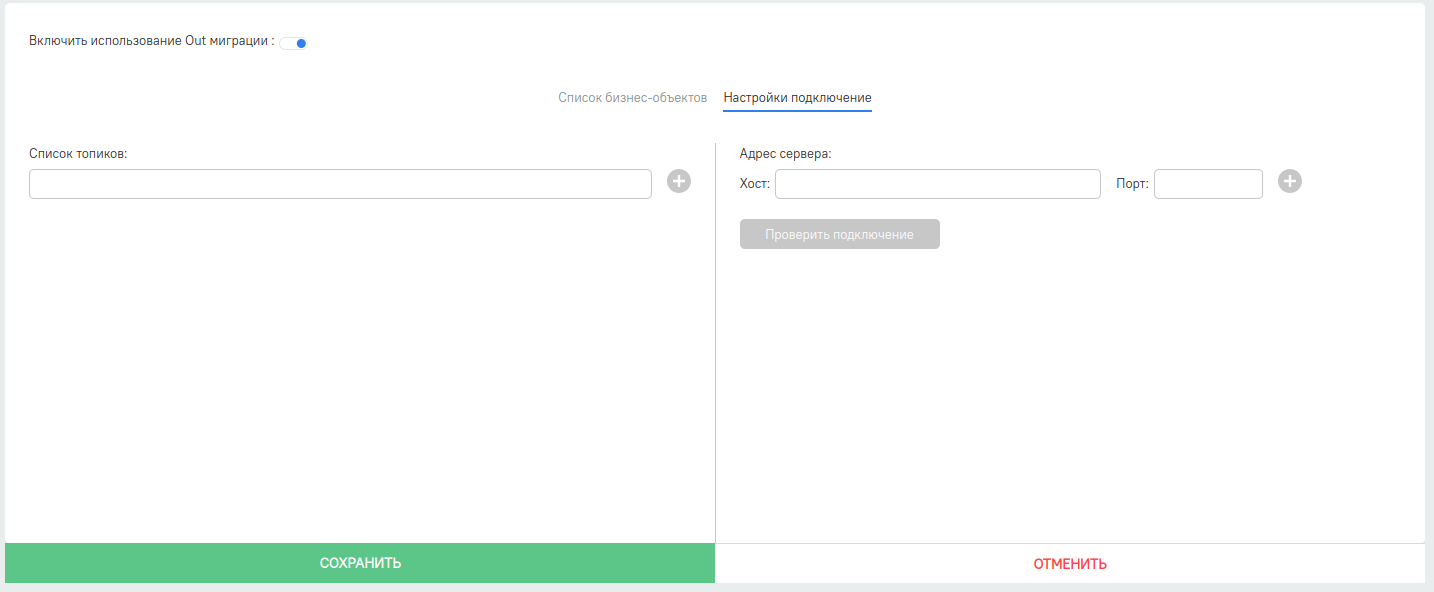
Раскрыв БО-ты можем указать поля, которые будут участвовать в миграции



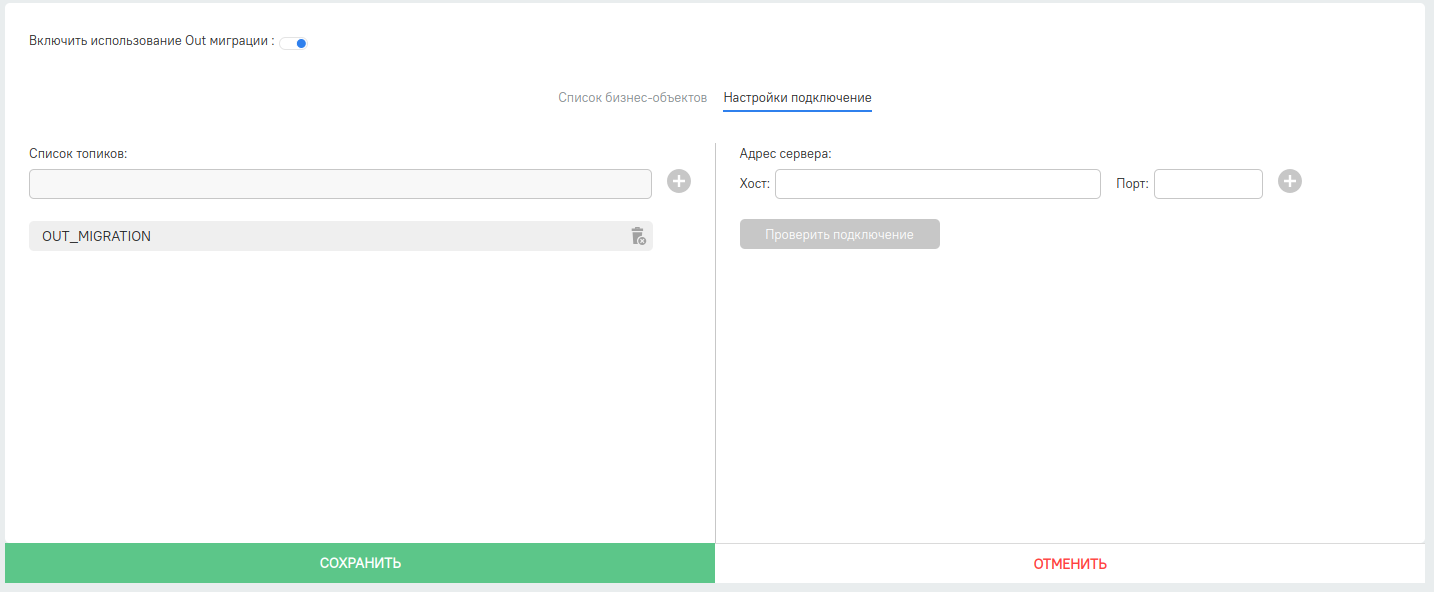
Дале выбираем поля. Примечание: поля с красными звёздочками - это уникальные поля, они нужны для миграций вложенных БО.



Переходим в "Настройки подключение".



Здесь необходимо указать название топика, в который будут записываться данные.

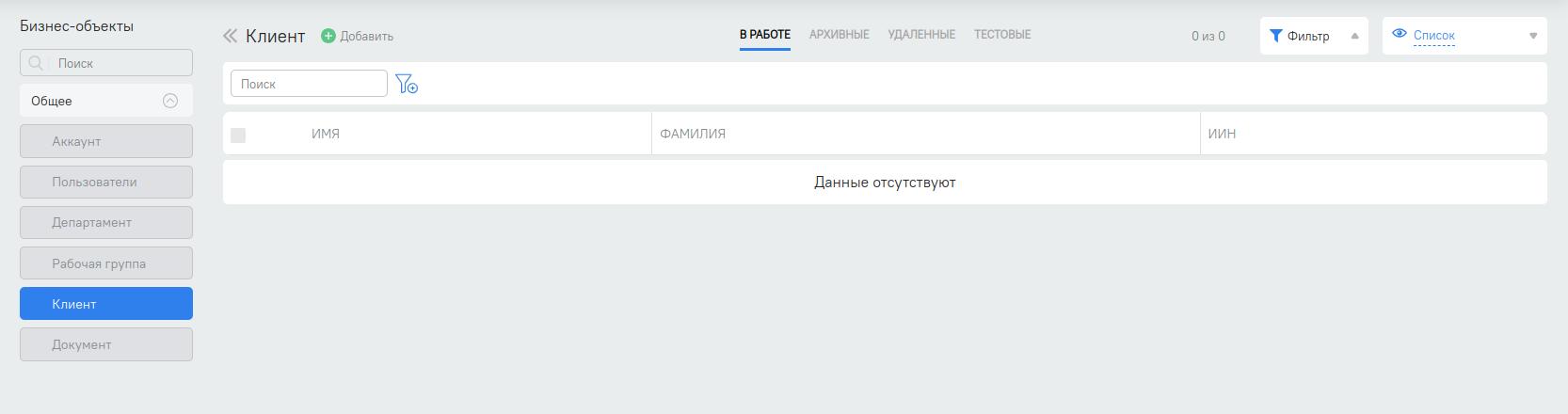


Сохраняем.

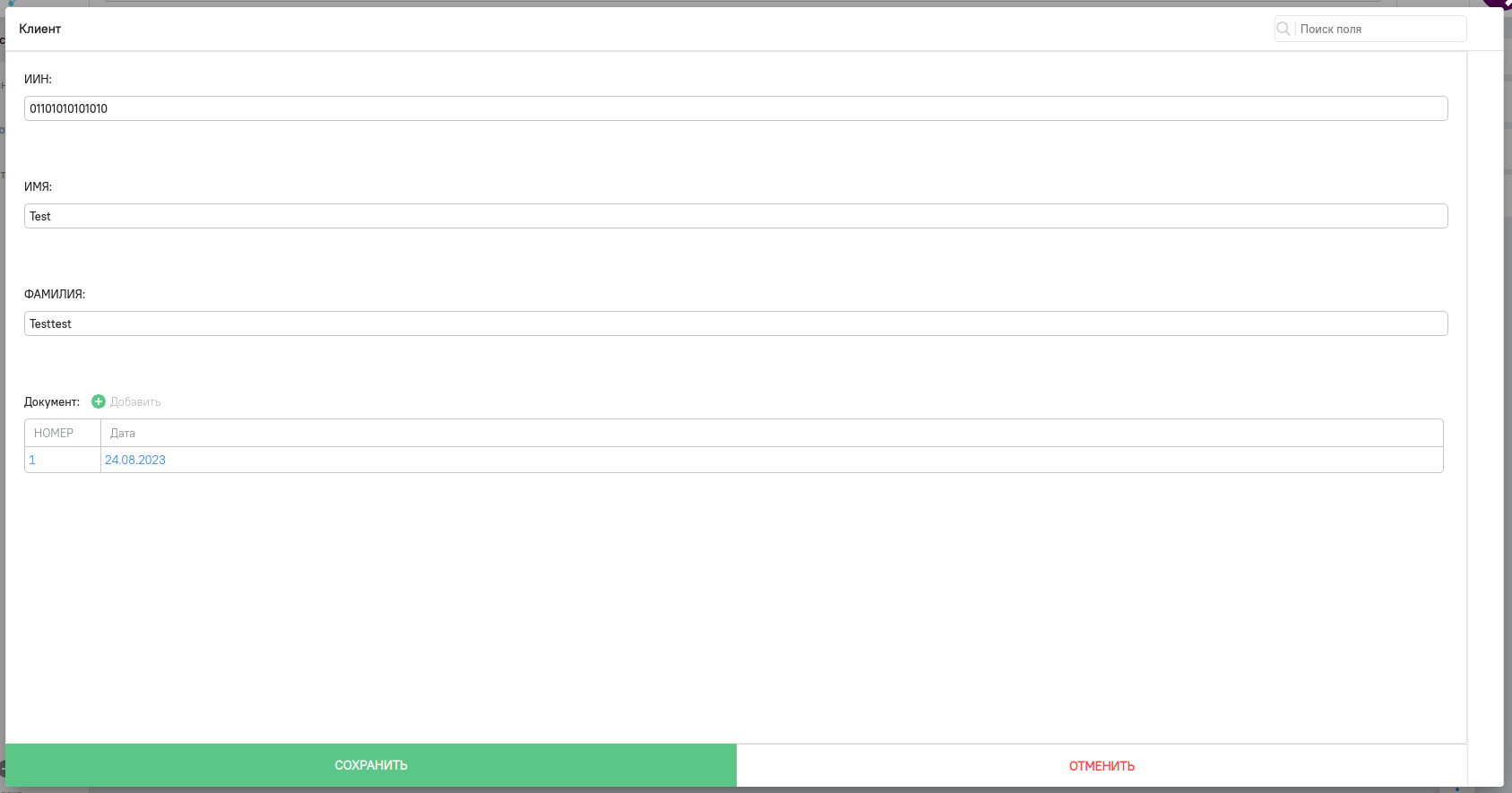
Прекрасно, мы всё настроили.

## Давайте создадим инстанции БО клиент.

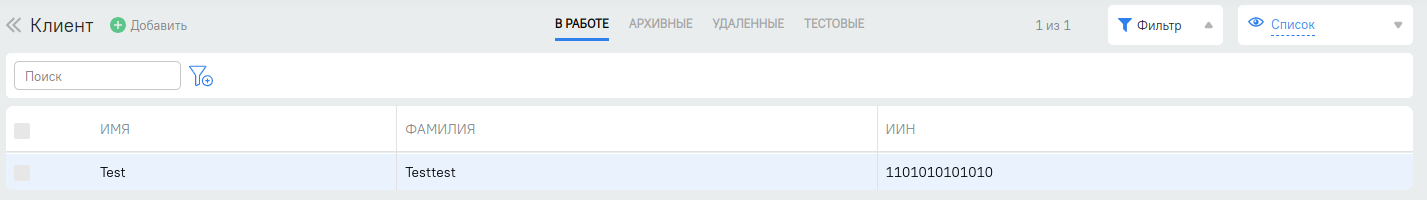
Для этого переходим в реестр БО клиент и нажимаем на кнопку добавить.



Заполняем поля.



Сохраняем и видим в реестре есть запись.



## Результат, в топике OUT\_MIGRATION появилась запись формата JSON.

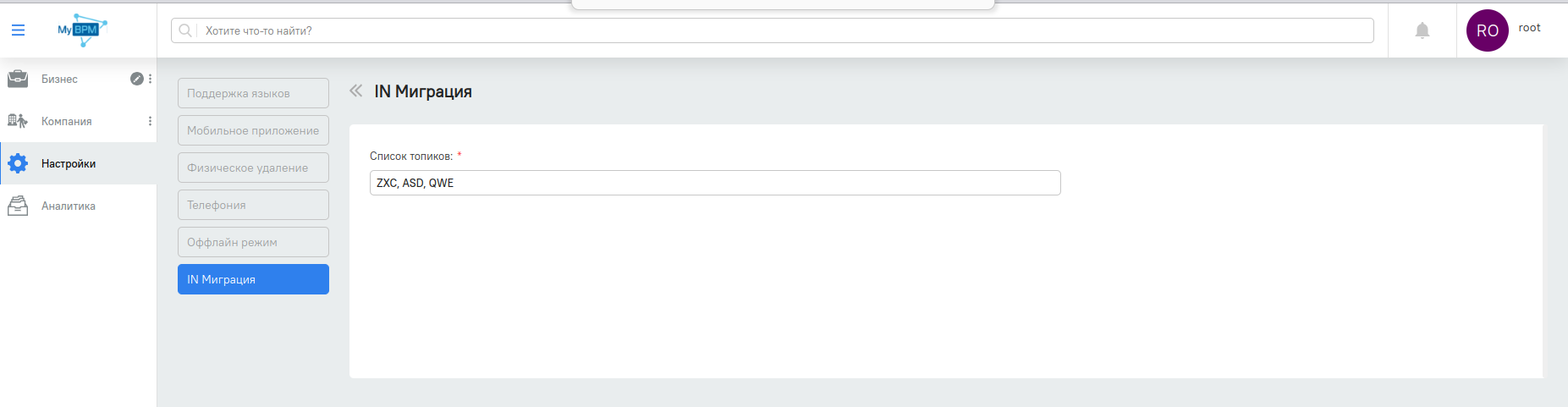
{  
 "recordId":null,  
 "externalId":null,  
 "id":"yKjoIid~C@R1h2Nf",  
 "boCode":"client",  
 "fields":[  
 {  
 "code":"lastname",  
 "apiValue":"Testtest",  
 "updatedAt":1692865330851  
 },  
 {  
 "code":"name",  
 "apiValue":"Test",  
 "updatedAt":1692865330851  
 },  
 {  
 "code":"iin",  
 "apiValue":"1101010101010",  
 "updatedAt":1692865330851  
 }  
 ],  
 "boFields":[  
 {  
 "fieldCode":"doc",  
 "toBoCode":null,  
 "toFieldCode":"number",  
 "apiValue":"1",  
 "updatedAt":1692865330851  
 }  
 ],  
 "state":"ACTUAL"  
}

# IN Migration Kafka Introduction

## Самое начало

/kz/greetgo/mybpm/model\_kafka\_mongo/mongo/company/CompanyDto.java

Нужно заполнить с каких топиков загружать данные, ниже приведён пример.



## Какими данными работает миграция?

Миграция работает данными в формате **JSON** класса **BoiInput.java**

Класс **BoiInput.java** предназначен для работы миграция через кафку

kz/greetgo/mybpm/register/impl/migration/in\_kafka/model/BoiInput.java

## Структура класса BoiInput.java

public String recordId;  
  
public String externalId;  
  
public String id;  
  
public String boCode;  
  
public List<BoiInputField> fields;  
  
public List<BoiInputFieldBo> boFields;  
  
public State state;

## Поля BoiInput.java для чего они

public String recordId

**recordId** - Идентификатор данной входящей записи

Предназначен для предотвращения повторной записи данных, можно не определять, но не желательно.

public String externalId

**externalId** - Внешний идентификатор инстанций, равен ***BoiDto.externalId()***

Предназначен для определения инстанций **BoiDto.java**

public String id

**Id** - Идентификатор инстанций, равен ***BoiDto.strId()***

Предназначен для определения инстанций **BoiDto.java**

public String boCode

**boCode** - Код бизнес объекта в которой должна вставиться входящая запись, равен ***BoDto.code()***

public List<BoiInputField> fields

**Fields** - список полей со значения которые участвует в миграций

#### [Подробнее о Fields](002-in-migration-kafka-fields.md)

public List<BoiInputFieldBo> boFields;

**boFields** - список ссылочных полей со значения которые участвует в миграций

#### [Подробнее о BoFields](003-in-migration-kafka-bo-fields.md)

public State state

**state** - состояние инстанций (ACTUAL, REMOVED, ARCHIVED, TEST)

# IN Migration Kafka BoiInputField

## Структура класса BoiInputField.java

public String code;  
  
public String apiValue;

public String code

**Code** - код поля в которой должна записаться инстанция, равен ***BoFieldDto.code()***

public String apiValue

**apiValue** - Значение из RestAPI

Для работы с apiValue используется Интерфейс BoFieldTypeProvider.java

Директория где можете посмотреть код BoFieldTypeProvider.java

kz/greetgo/mybpm/register/impl/bo/etc/BoFieldTypeProvider.java

# IN Migration Kafka BoiInputFieldBo

## Условия для того чтобы инстанций записалась в систему, поле должно быть уникальным!!!

### Стандартная структура класса BoiInputFieldBo.java

public String fieldCode;  
  
public String toBoCode;  
  
public String toFieldCode;  
  
public String apiValue;

### Рекурсивная структура класса BoiInputFieldBo.java

public String fieldCode;  
  
public BoiInput boiInput;

#### [Почему существует две структуры и чем они отличаются друг от друга?](005-in-migration-kafka-bo-fields-example.md)

### Описание полей

public String fieldCode

**fieldCode** - код поля в которой должна записаться инстанция, равен ***BoFieldDto.code()***

public BoiInput boiInput;

**boiInput** - вложенный JSON другого [BoiInput](001-in-migration-kafka-start.md), который позволяет добавлять или удалять ссылки на другие бизнес-объекты в поле BO/CO и одновременно создавать, читать, изменять или удалять эти вложенные бизнес-объекты.

Это поле используется в сочетании с fieldCode, другие поля не нужны.

public String toBoCode

**toBoCode** - код Бизнес Объекта в которой должна записаться инстанций, равен ***BoDto.code()***

Используется для миграций СО, для определения в какой именно БО должна записаться инстанция

***Можно не определять если это БО***

public String toFieldCode

**toFieldCode** - код поля вложенного Бо

public String apiValue

**apiValue** - Значение из RestAPI, Примечание значения должно быть уникальное

Для работы с apiValue используется Интерфейс BoFieldTypeProvider.java

Директория где можете посмотреть код BoFieldTypeProvider.java

kz/greetgo/mybpm/register/impl/bo/etc/BoFieldTypeProvider.java

# IN Migration Kafka Additional

## Временное хранение смигрированных данных

Каждая мигрированная запись храниться в монго коллекций

mongo.mybpm\_aux.KafkaMigrationRecord

Это предотвращает перезапись данных.

Класс, который используется в этой, коллекций:

public class KafkaMigrationRecordDto

Расположения данного класса:

kz/greetgo/mybpm/model\_kafka\_mongo/migration/KafkaMigrationRecordDto.java

## Schedulers of InMigrationKafka

Все шедулеры связанные с миграцией через кафку находиться в классе:

public class InMigrationKafkaScheduler

Удаление временных записей из монго коллекций

@Scheduled("#")  
 @FromConfig("repeat every 5 hour")  
 public void deleteRecords() {  
 inMigrationKafka.deleteKafkaMigrationRecord();  
 }

Повторяются каждые 5 часов, и удаляет просроченные ***KafkaMigrationRecordDto***. Время простроченных записей можно определить в конфиге

Запуск дополнительной миграций для не найденных значение ссылочных БО и СО

@Scheduled("#")  
 @FromConfig("repeat every 10 min")  
 public void applyRefBoi() {  
 inMigrationKafkaRefBoi.migrateFromTempTable();  
 }

Запускается каждые 10 минут если находит значение то, смигрирует эту запись и сделает запись не актуальным во временной таблице. Примечание эти временные данные хранятся в ПОСТГРЕСЕ

Удаляет не актуальные записи в 23:30, под не актуальными записями имеется ввиду уже смигрированные данные которые описан выше

@Scheduled("#")  
@FromConfig("23:30")  
public void deleteUnActualRecords() {  
inMigrationKafkaRefBoi.deleteUnActualRecords();  
}

## Config of InMigrationKafka

@Description("Конфиг используемые во время миграции в через кафку")  
public interface InMigrationKafkaConfig {  
  
@Description("Количество часов до очищения временных записей")  
@DefaultIntValue(5)  
int recordCleanHours();  
  
@Description("Интервал в минутах для обновление не просаженных ссылочных записей")  
@DefaultIntValue(10)  
long interval();  
  
@Description("Количество минут до удаления старых записей из Базы")  
@DefaultIntValue(10080)  
long oldRecordDeleteMinutes();  
}

# IN Migration Kafka BoiInputFieldBo Example

Существует два способа IN миграции вложенных бизнес-объектов через Kafka: **стандартный** и **рекурсивный**.

* Стандартный способ позволяет переносить по одной записи за раз, и для производительности лучше сначала перенести вложенный bo, а затем основной bo.
* Рекурсивный способ позволяет эффективно переносить множество записей одновременно, но при этом необходимо создать древовидную структуру.

Ниже вы можете увидеть примеры каждого подхода:

## Стандартный способ

1) Добавляем школу

{  
 "recordId": "random-value",  
 "externalId": "MVLFsYhLv5eVbkXE",  
 "boCode": "school",  
 "fields": [  
 {  
 "code": "number",  
 "apiValue": "133"  
 },  
 {  
 "code": "name",  
 "apiValue": "среднеобразовательная школа"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "class",  
 "toBoCode": "class",  
 "toFieldCode": "class\_id",  
 "apiValue": "1"  
 }  
 ],  
 "state": "ACTUAL"  
}

2) Добавляем класс

{  
 "recordId": "random-value",  
 "externalId": "MVLFsYhLv5eVbkXE",  
 "boCode": "class",  
 "fields": [  
 {  
 "code": "number",  
 "apiValue": "9G"  
 },  
 {  
 "code": "name",  
 "apiValue": "Класс №"  
 },  
 {  
 "code": "class\_id",  
 "apiValue": "1"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "student",  
 "toBoCode": "student",  
 "toFieldCode": "student\_id",  
 "apiValue": "1"  
 },  
 {  
 "fieldCode": "student",  
 "toBoCode": "student",  
 "toFieldCode": "student\_id",  
 "apiValue": "2"  
 }  
 ],  
 "state": "ACTUAL"  
}

3) Добавляем студентов

{  
 "recordId": "random-value",  
 "externalId": "7fq@eZ1kpg0iwHVV",  
 "boCode": "student",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Армэн"  
 },  
 {  
 "code" : "surname",  
 "apiValue" : "Турсынбек"  
 },  
 {  
 "code" : "student\_id",  
 "apiValue" : "1"  
 }  
 ],  
 "state": "ACTUAL"  
}

{  
 "recordId": "random-value",  
 "externalId": "Iqn5iMZcg~M1I5o0",  
 "boCode": "student",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Азамат"  
 },  
 {  
 "code" : "surname",  
 "apiValue" : "Утелхан"  
 },  
 {  
 "code" : "student\_id",  
 "apiValue" : "2"  
 }  
 ],  
 "state": "ACTUAL"  
}

## Рекурсивный способ

1. Добавляем Школу -> Вкладываем класс -> Вкладываем учеников

{  
 "recordId": "random-value",  
 "externalId": "MVLFsYhLv5eVbkXE",  
 "boCode": "school",  
 "fields": [  
 {  
 "code": "number",  
 "apiValue": "133"  
 },  
 {  
 "code": "name",  
 "apiValue": "среднеобразовательная школа"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "class",  
 "boiInput": {  
 "recordId": "random-value",  
 "externalId": "MVLFsYhLv5eVbkXE",  
 "boCode": "class",  
 "fields": [  
 {  
 "code": "number",  
 "apiValue": "9G"  
 },  
 {  
 "code": "name",  
 "apiValue": "Класс №"  
 },  
 {  
 "code": "class\_id",  
 "apiValue": "1"  
 }  
 ],  
 "boFields": [  
 {  
 "fieldCode": "student",  
 "boiInput": {  
 "recordId": "random-value",  
 "externalId": "7fq@eZ1kpg0iwHVV",  
 "boCode": "student",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Армэн"  
 },  
 {  
 "code": "surname",  
 "apiValue": "Турсынбек"  
 },  
 {  
 "code": "student\_id",  
 "apiValue": "1"  
 }  
 ],  
 "boFields": [],  
 "state": "ACTUAL"  
 }  
 },  
 {  
 "fieldCode": "student",  
 "boiInput": {  
 "recordId": "random-value",  
 "externalId": "Iqn5iMZcg~M1I5o0",  
 "boCode": "student",  
 "fields": [  
 {  
 "code": "name",  
 "apiValue": "Азамат"  
 },  
 {  
 "code": "surname",  
 "apiValue": "Утелхан"  
 },  
 {  
 "code": "student\_id",  
 "apiValue": "2"  
 }  
 ],  
 "state": "ACTUAL"  
 }  
 }  
 ],  
 "state": "ACTUAL"  
 }  
 }  
 ],  
 "state": "ACTUAL"  
}

# IN Migration

#### [Как настроить IN миграцию для старта](001-in-migration-start.md)

#### [Процесс IN миграции](002-in-migration-process.md)

#### [Дополнительные настройки IN миграции](003-in-migration-additional.md)

# IN Migration Start

## Самое начало

kz/greetgo/mybpm/model\_kafka\_mongo/mongo/bo/field/BoFieldDto.java

Нужно проставить needLoadFromInTables=true в поля, которые должны быть мигрированы

## Генерация файла с DDL для формирования in\_таблиц

LaunchGenerateMigrationFile\_TMP  
LaunchGenerateMigrationFile  
MigrationController

## Чтобы миграция запускалась нужно в конфиге

/mybpm/configs/InMigrationPostgresConfig.txt

Поставить true у параметра hasInMigration

## Здесь находятся конфиги распределения сообщений кафки по топикам

/mybpm/direct-configs/KafkaDistributionBoOverTopic.yaml

## Чтобы миграция запустилась нужно проставить код компании, в которой происходит миграция, в конфиге:

/mybpm/configs/InMigrationConfig.txt

В параметре companyCode

## Загрузка тестовых данных в in\_таблицы

kz.greetgo.mybpm\_util\_light.probes.gen.InsertManyData

## Загрузка тестовых данных для массового теста миграции

kz.greetgo.mybpm.register.impl.migration.in.InMigrationStarterImplTest.prepareDataForMigrationStart()

## Есть 2 метода запустить миграцию

1. kz/greetgo/mybpm/register/controller/migration/MigrationController.java  
2. kz/greetgo/mybpm/register/scheduler/InMigrationScheduler.java

1. Через контроллер (только для теста)
2. Через scheduler (на проде нужно настраивать только через это)

## Конфиги нужные для миграции:

1. kz/greetgo/mybpm/register/config/in\_migration/InMigrationConfig.java  
2. kz/greetgo/mybpm/register/config/in\_migration/InMigrationDataSourceConfig.java  
3. kz/greetgo/mybpm/register/config/in\_migration/InMigrationPostgresConfig.java

## Классы, где смотреть код:

1. kz/greetgo/mybpm/register/impl/migration/in/InMigrationStarterImpl.java  
2. kz/greetgo/mybpm/register/impl/migration/in/InMigrationImpl.java  
3. kz/greetgo/mybpm/register/consumer/controller/in\_migration/InMigrationConsumer.java  
4. kz/greetgo/mybpm/register/impl/migration/in/InMigrationWorkerImpl.java  
5. kz/greetgo/mybpm/register/impl/migration/in/InMigrationKafkaRegisterImpl.java

# IN Migration Process

In миграция работает с помощью кафки.

Состоит из 3 частей:

1) Забор данных из in таблиц и отправка в кафку (топик = IN\_MIGRATION) 2) Получения данных из кафки и подготовка boi для отправки в другой топик кафки (BOI) 3) Дальше consumer-ы топика BOI делают свою часть.

## Часть 1. Забор данных из in таблиц и отправка в кафку

* После того как при помощи *DDL* были созданы in таблицы.
* И после того как была настроена связка с базой где находится in таблицы.
* И после того как был настроен InMigrationScheduler или вызван контроллер MigrationController.

InMigrationStarterImpl начинает искать таблицу **process\_tracking**, которая отвечает за реальный запуск миграции.

Таблица **process\_tracking** - служит для контроля обмена данными между нашей системой и системой заказчика (которая заполняет in таблицы). То есть она состоит из 2 частей.

* Сторона заказчика создает новую запись в таблице и заполняет свои информационные поля. Как только они заканчивают заполнения всех in таблиц они дают нам сигнал об окончании.
* Наша сторона ждет появления этого сигнала, и как дожидается, дальше начинается миграция in. После того как мы закончили забор данных из in таблиц в кафку, мы даем сигнал об окончании забора данных из in таблиц. Сторона заказчика видя сигнал с нашей стороны обратно начинает свою часть. И так по кругу.

**ВАЖНО!** Нужно договорится со стороной заказчика чтоб они после получения сигнала от нас не очищали in таблицы сразу же. Эти in таблицы дальше используются во **2 части миграции**. Необходимо увидеть время, когда количество lag-ов в кафке (топик = IN\_MIGRATION, groupId = in\_migration) уйдет в 0. Это время и есть время через которое необходимо чистить in таблицы для следующей загрузки.

Описание колонок в таблице **process\_tracking**: 1) id - просто autoincrement id 2) oper\_date - опер день, за который мы получаем данные 3) start\_load\_time - начало загрузки данных в in таблицы 4) end\_load\_time - конец загрузки данных в in таблицы 5) load\_status - **сигнал** об окончании загрузки данных в in таблицы 6) start\_ins\_inkafka\_time - начало забора данных из in таблиц 7) end\_ins\_inkafka\_time - конец забора данных из in таблиц 8) ins\_inkafka\_status - **сигнал** об окончании забора данных из in таблиц и отправки в кафку 9) occupied\_by\_process - id миграции 10) migrate\_cur\_date - дата миграции 11) periodic\_update\_time - поле нужное для борьбы с постоянным рестартом api (замечался в проекте jusan collection) 12) inserted\_at - дата и время создании записи

Как только наша система удовлетворена таблицей **process\_tracking** мы переходим к InMigrationImpl и начинаем забирать данные с in таблиц и отправлять их в кафку.

## Часть 2. Получения данных из кафки и подготовка boi для отправки в другой топик кафки (BOI)

Consumer in миграции принимает объекты

1. KafkaInMigration  
2. KafkaInMigrationRefChecker

KafkaInMigration - хранит данные одной записи в одной in таблице.

KafkaInMigrationRefChecker - специальный объект, который нужен для дополнительной проверки связок между инстанциями.

Но самый главный здесь это KafkaInMigration. Ниже расписан каждый этап прохождения одной записи через InMigrationKafkaRegisterImpl:

1) Достаем все данные, которые были залиты в in таблицы одной записи. Тут имеется ввиду сама таблица этой записи и также все данные из связанных таблиц.

2) Если у этой записи нет никаких данных для мигрирования, то она скипается.

3) При помощи externalId и boId проверяем есть ли уже по ним какая-нибудь инстанция. Если есть, то KafkaBoi.isCreate = false иначе KafkaBoi.isCreate = true

4) Проверяем нет ли признака removed=true в in таблице. Если есть, то мы ставим kafkaBoi.actual(false) и заканчиваем здесь проход по миграции для этой записи.

5) Заполняем в kafka.changes новые данные. С начало простые поля, а дальше поля типа BO и CO. Если мы ожидали какой-то объект в поля BO или CO а его еще нет в системе, то мы делаем insert в таблицу ожидания связок boi\_reference.

6) Делаем kafkaBoi.apply если хоть одно поле этой записи изменилось в этой миграции, иначе мы просто садим в монго inMigrationUpdatedAt=new Date() чтоб не засорять кафку пустыми данными.

7) При ошибке MongoErrorDuplicateKey находим инстанцию по который был конфликт и делаем слияние данных с существующей и новой записью. Все новые данные обновляем в существующей инстанции и делаем kafkaBoi.apply. И также дальше работаем уже по существующей инстанции.

8) Если мы **НЕ** словили MongoErrorDuplicateKey, то сохраняем свою boiId в BoiInMigrationRefдля того, чтоб далее нас находили по externalId и boId

9) В конце запись начинает искать себя в таблице boi\_reference чтоб узнать кто его ожидает. Если есть такие инстанции, то мы добавляем себя к ним.

## Часть 3. Дальше consumer-ы топика BOI делают свою часть.

Это уже отдельная от in миграции часть. Тут происходит добавление в elasticSearch и тд работы.

## Дополнительные замечания.

Не нужно забывать что при мигрирования системных BO как:

Person  
Department  
PersonGroup

Нужно в обязательном порядке мигрировать хоть одно системное поле. Иначе не появится запись.

Пример:

При этом случае Person появится в mybpm\_boi, но в коллекции Person будет пусто. Что означает что этого Person не существует в системе.

**ВАЖНО!** Company не мигрирует через in миграцию.

# IN Migration Additional Settings

## Где смотреть логи

/var/log/mybpm/migration/in\_migration.log

## Где смотреть трэйсы

2. /var/log/mybpm/traces/inMigration.log

Чтобы увидеть трэйсы нужно будет в zoo navigator найти inMigrationTracer и в его **level** проставить TRACE вместо **OFF**

## Информативные таблицы миграции в базе aux1

### 1. in\_migration.boi\_reference

Таблица нужна для связок между полями типа BO или CO. Она работает по логике:

"Если я по таблице связок получаю связанный объект, но его еще нет в системе, то он сохраняется в эту таблицу. Получается что каждый элемент который проходит миграцию, проверяет не ждет ли его кто-то из ранее прошедших миграцию"

### 2. in\_migration.copy\_table\_info

Хранит информацию про **copy** таблицы. Нужен во избежания ошибок при создании **copy таблиц**

### 3. in\_migration.errors

Хранит данные об ошибках при миграции, в частности нужен был для отлавливания неожиданных ошибок

### 4. in\_migration.from\_kafka\_to\_mongo\_tracking

Хранит данные о том, сколько данных пришли в **InMigrationKafkaRegisterImpl.comeKafkaInMigration** и что с ними в итоге стало.

### 5. in\_migration.log

Хранит данные об ошибках при миграции, в частности нужен был для отлавливания ожидаемых ошибок

### 6. in\_migration.timings

Хранит данные про время выполнения самых долго работающих методов в **InMigrationKafkaRegisterImpl**

## Mongo DTO, которая связывает внешние in\_id с внутренние boiId

kz/greetgo/mybpm/model\_kafka\_mongo/mongo/bo/BoiInMigrationRefDto.java

## Логика очистки информативных таблиц находится в классе:

kz/greetgo/mybpm/register/impl/migration/in/InMigrationCleanerImpl.java

Кроме **copy** таблиц. Copy таблицы создаются и удаляются в классе

kz/greetgo/mybpm/register/impl/migration/in/InMigrationImpl.java

# Ускорение входящей миграции

## Введение

В этом файле прописаны некоторые советы по ускорению миграции.

## Предварительные требования

Перед началом работы необходимо удостовериться что POD-ы Кафки и Java настроены корректно ресурсы.

Java Удостоверьтесь, что в yaml файле mybpm-api, правильно настроен environment JAVA\_TOOL\_OPTIONS.

Значение в "-Xms" и "-Xmx" должны быть одинаковы и составлять не более 75% от общей памяти выделенной на сам под. 50% самый безопасный и оптимальный вариант.

Пример. Если на POD выделено 10Gb, то этому параметру можно выделить не более 7680M или 7Gb (75%). Но лучше поставить 5120M (50%)

* { name: JAVA\_TOOL\_OPTIONS, value: "-Xms7680M -Xmx7680M" }
* or
* { name: JAVA\_TOOL\_OPTIONS, value: "-Xms5120M -Xmx5120M" }

Kafka Удостоверьтесь, что в yaml файле kafka, правильно настроен environment KAFKA\_HEAP\_OPTS.

Правила такие же как для Java, но лимит 50% от памяти POD-а

Пример. Если на POD выделено 10Gb. То этому параметру нужно выделить 5120M или 5Gb (50%).

* { name: JAVA\_TOOL\_OPTIONS, value: "-Xms5120M -Xmx5120M" }

## Основная часть

Самый простой способ увеличить скорость миграции - увеличить количество консьмеров (consumers), потребляющие сообщения из топика IN\_MIGRATION

Кол-во консьюмеров насраивается в зукипере (ZooKeeper) в директории consumers. Чтобы найти эту директорию:

1. Перейдите в ZooNavigator (адрес, на котором он разворачивается, смотрите в docker-compose файле для локалки это - <http://localhost:10010>)
2. Перейдите в директорию mybpm
3. Далее перейдите в директорию consumers
4. Здесь лежат конфигурации всех коньсьюмеров mybpm-а, кол-во консьюмеров для миграции можно изменить в файле InMigration.workerCount

Адрес файла в zoonavigator-е "/mybpm/consumers/InMigration.workerCount"

### Правила по установке консьюмеров

1. Кол-во консьюмеров никогда не должно превышать количество партиций (partitions) топика (в данном случае топика IN\_MIGRATION)
2. Кол-во консьюмеров должно быть делителем числа партиции топика. Например, если в топике IN\_MIGRATION 48 партиции, то возможное значение для консьюмеров: 1, 2, 4, 6, 8, 12, 24, 48.
3. Обратите внимание кол-во консьюмеров напрямую зависит от выделенных ресурсов системе. Если памяти недостаточно кафке или Java, то даже увеличив кол-во консьюмеров, вы не ускорите миграцию.
4. Лучший выбор постепенно увеличивать количество по следющему алгоритму:
   * По дефолту стоит 1 консьюмер, нужно увеличить до 2. Проверить нагрузку на сервер во время миграции.
   * Если нагрузка в рамках допустимого и миграции стала быстрее, то можно увеличить кол-во консьюмеров еще раз, учитывая вышеприведенные правила (при 48 партициях, следующие значения будут: 4, 6, 8, 12, 24, 48)
   * Повторять до тех пор, пока не найдете оптимальное значение (скорость перестала увеличиваться или нагрузка слишком большая)

### Дополнительные способы

Увеличить скорость миграции можно также отключив трэкинг времени исполнения методов. Сделать это можно в конфиге в зукипере. Для этого нужно найти InMigrationConfig.

Адрес конфига в zoonavigator-е "/mybpm/configs/InMigrationConfig.txt"

В этом конфиге для параметра "trackTimings" нужно поставить значение false. Это даст небольшой прирост скорости.

## Дополнительная информация

Чтобы посмотреть среднюю скорость миграцию, нужно выполнить SQL-команду:

SELECT (SUM(inserted\_data\_counter) / extract(epoch from MAX(finished\_at) - MIN(started\_at))) as avg\_speed FROM in\_migration.from\_kafka\_to\_mongo\_tracking WHERE started\_at >= <время начало миграции>;

Этот запрос вернет кол-во записей в секунду.

# Настройка миграции на платформе MyBPM

Функционал MyBPM для формирования структуры внешней Базы Данных SQL с автоматической выгрузкой и загрузкой данных.

Порядок действий:

1. В режиме редактирования, в необходимом Бизнес-объекте выбираем нужное вам поле для выгрузки.

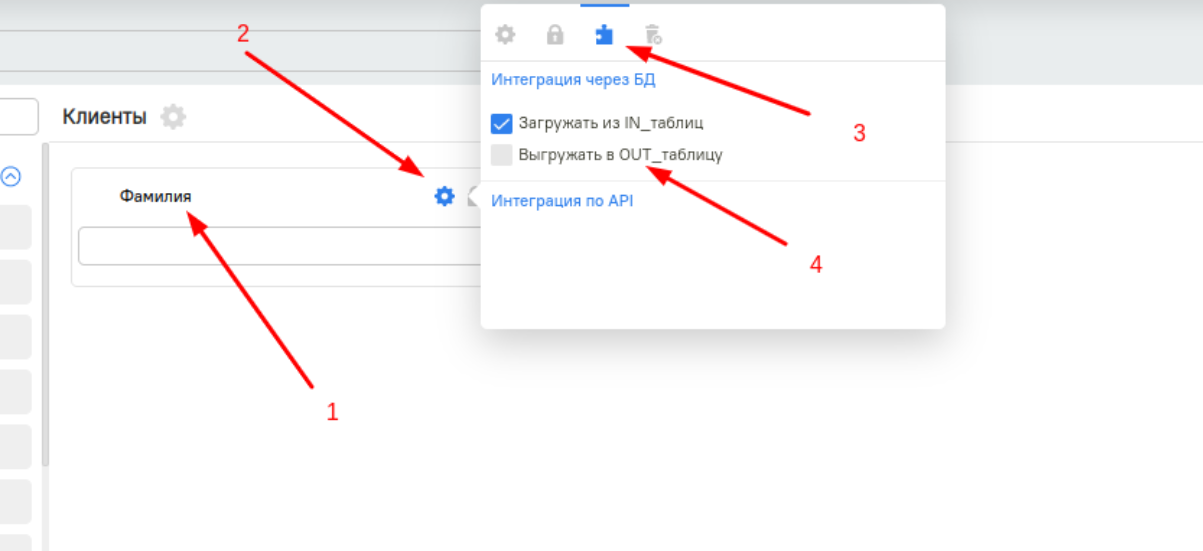
2. Нажимаем на иконку настроек поля в виде шестеренки в углу поля.

3. В модальном окне в верхних вкладках выбираем пункт интеграций.

4. В разделе “Интеграция через БД” отмечаем необходимую опцию:

- для загрузки из БД данных в MyBPM выбираем пункт “Загружать из IN\_таблиц”,  
- для выгрузки данных из MyBPM в БД выбираем пункт “Выгружать в OUT\_таблицу”.

5. Сохраняем изменения в БО нажатием кнопки “Сохранить”.



Настройка признака in/out

После всех настроек в MyBPM полей интеграции:

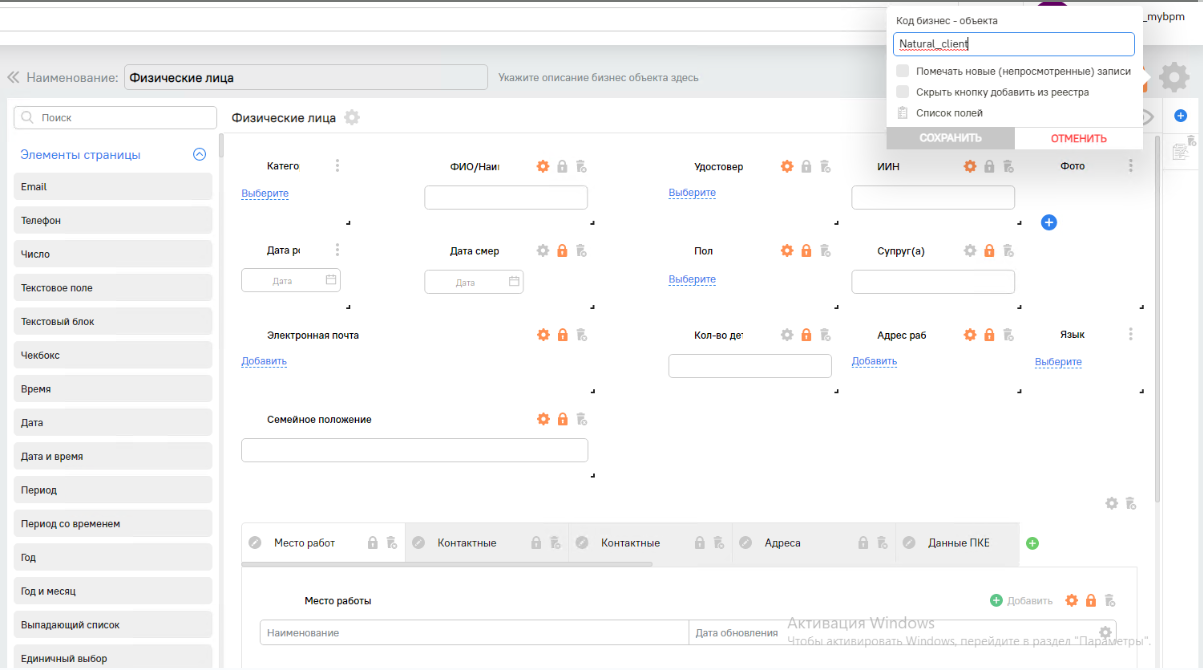
* Для загрузки данных в MyBPM формируется структура для SQL в формате DDL файла. Файл предоставляет разработчик MyBPM. Данный файл необходимо запустить в вашей внешней БД PostgreSQL, который создает всю структуру таблиц. Дальнейшее обновление данных должно соответствовать этой структуре. Предоставляются данные для подключения к БД.
* Для выгрузки данных из MyBPM тоже предоставляются данные для подключения к БД.

**!Важно**: При изменении полей для интеграции, формируется новая структура БД. Для того чтобы выгрузка произошла корректно, необходимо запустить описанный процесс заново.

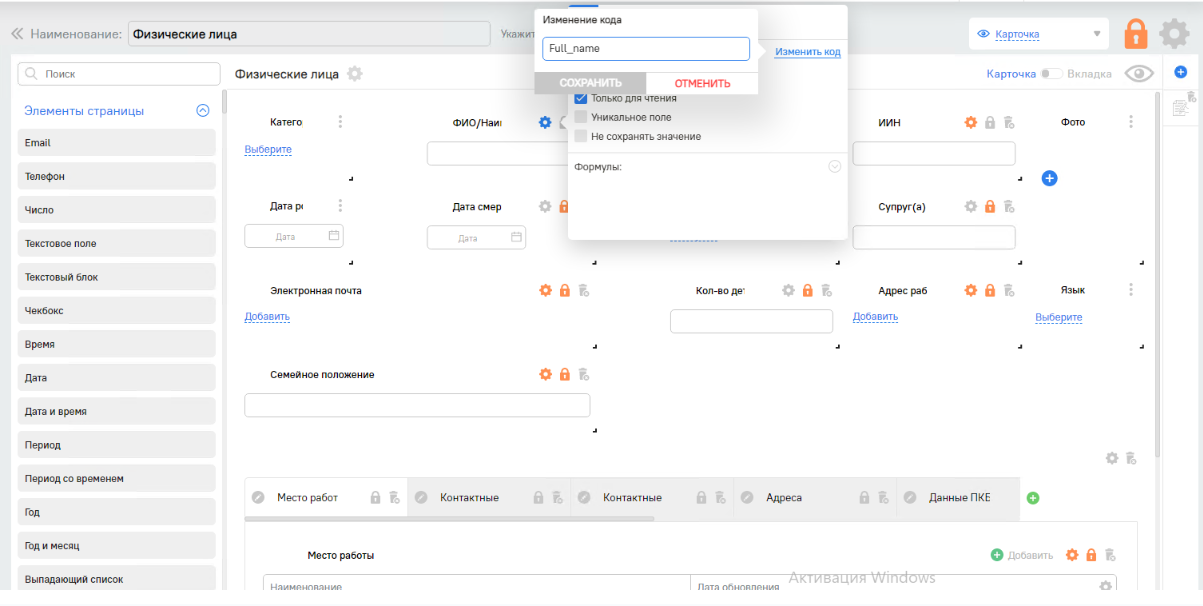
Структура формируется по правилам:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Структура БО MyBPM | Структура таблиц в БД | Описание |
| 1 | Код бизнес-объекта | Название таблицы с приставкой |  |
| 2 | Код поля | Название колонки |  |
| 3 | Содержимое поля в MyBPM | Значения поля |  |
| 4 | Тип поля | Тип переменной | Ассоциирован с типами SQL согласно логике полей MyBPM |
| 5 | Вложенный БО | Внешний таблица со связанными id | Таблица содержит Id вложенного объекта, и Id принимающего |
| 6 | Вложенный Композитный объект | Внешние таблицы со связанными id | Количество таблиц соответствует количеству вложенных БО в Композитный объект. Для каждого вложенного объекта формируется своя таблица связки |

## Пример формирования структуры таблицы выгрузки IN из MyBPM.



Код бизнес-объекта



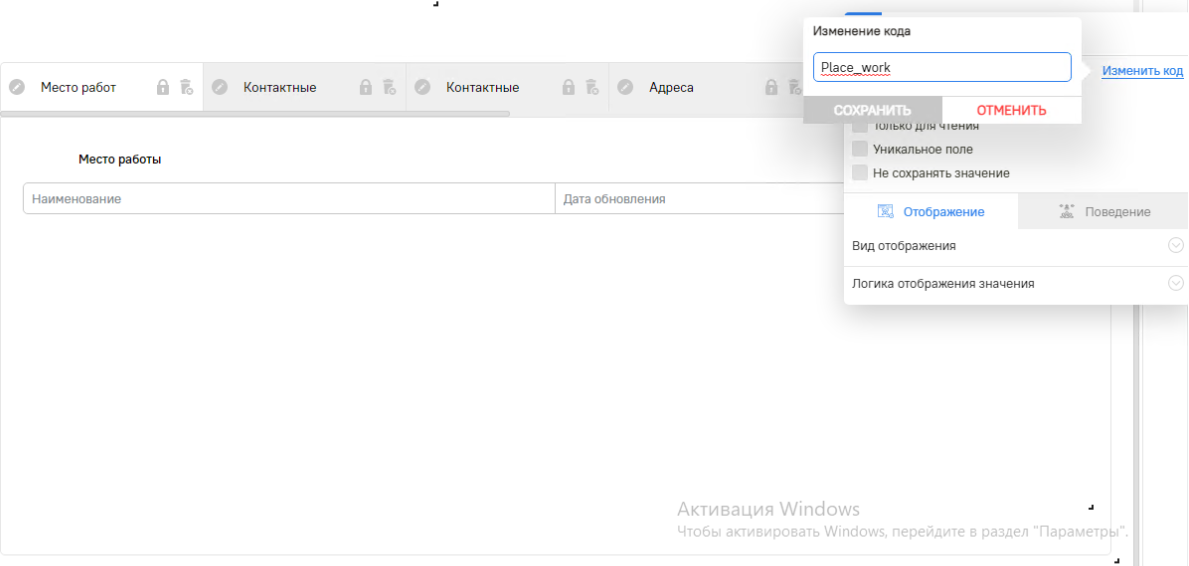
Код поля

В результате получаем таблицу с названием по коду БО и приставкой in.  
create table public.in\_Natural\_client (  
id bigserial primary key,  
in\_id varchar(100),  
out\_id varchar(32),  
inserted\_date timestamp with time zone default clock\_timestamp(),  
in\_status varchar(100) default '' not null,  
occupied\_id bigint,  
occupied\_at timestamp with time zone,  
removed bool default false,  
IIN text,  
Category varchar(32),  
Date\_Birth timestamp with time zone,  
Date\_death timestamp with time zone,  
SUSN\_type bool,  
Relationship\_status text,  
Client\_dep\_colvir\_ID numeric(24,4),  
Gender varchar(32),  
Amount\_children numeric(24,4),  
Client\_code text,  
Full\_name text,  
Global\_client\_ID numeric(24,4)  
);

Структура таблицы в описании:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя колонки в базе | Имя колонки MyBPM | Тип | Описание |
|  | Global\_client\_ID | int(8) | Global ID клиента |
|  | Category | Справочник - Category | Категория |
|  | Full\_name | VARCHAR(256) | ФИО |
|  | IIN | VARCHAR(256) | ИИН |
|  | Identity | Таблица - Identity | Удостоверение |
|  | Date\_Birth | Date | Дата рождения |
|  | Date\_death | Date | Дата смерти |
|  | Relationship\_status | VARCHAR(256) | Семейное положение |
|  | Email | Таблица - Email | Электронная почта |
|  | Gender | Справочник - Gender | Пол |
|  | Amount\_children | NUMERIC | Кол-во детей |
|  | SUSN\_type | Bool | Тип СУСН |
|  | Client\_dep\_colvirID | Numeric | Идентификатор подразделения клиента в колвире |

## Пример с вложенным БО

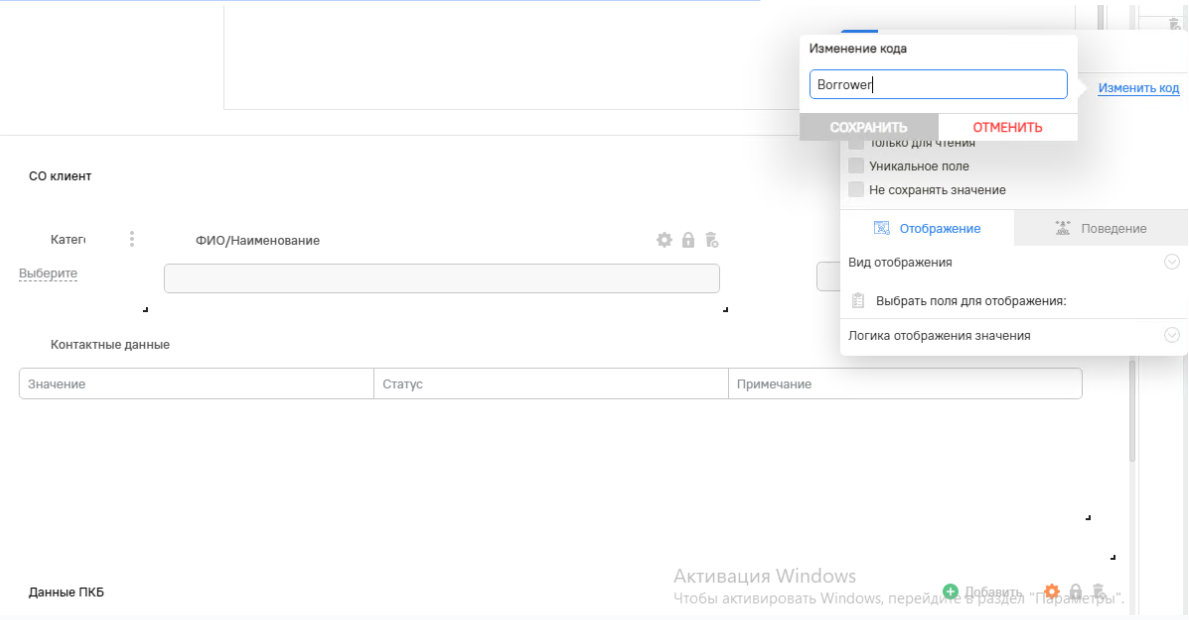


Код вложенного БО

Сформированная таблица с ключами на существующие таблицы, вложенные в БО. Где Natural\_client\_id ссылается на id в таблице in\_Natural\_client, то есть системный авто инкриминированный id, а Place\_work\_id ссылается на id в таблице in\_ Place\_work.

create table public.in\_Natural\_client\_Place\_work (  
 Natural\_client\_id varchar(100),  
 Place\_work\_id varchar(100),  
 in\_status varchar(100) default '' not null,  
 inserted\_date timestamp with time zone default clock\_timestamp(),  
 primary key (Natural\_client\_id, Place\_work\_id)  
);

## Пример с вложенным Композитным объектом



Код вложенного КО

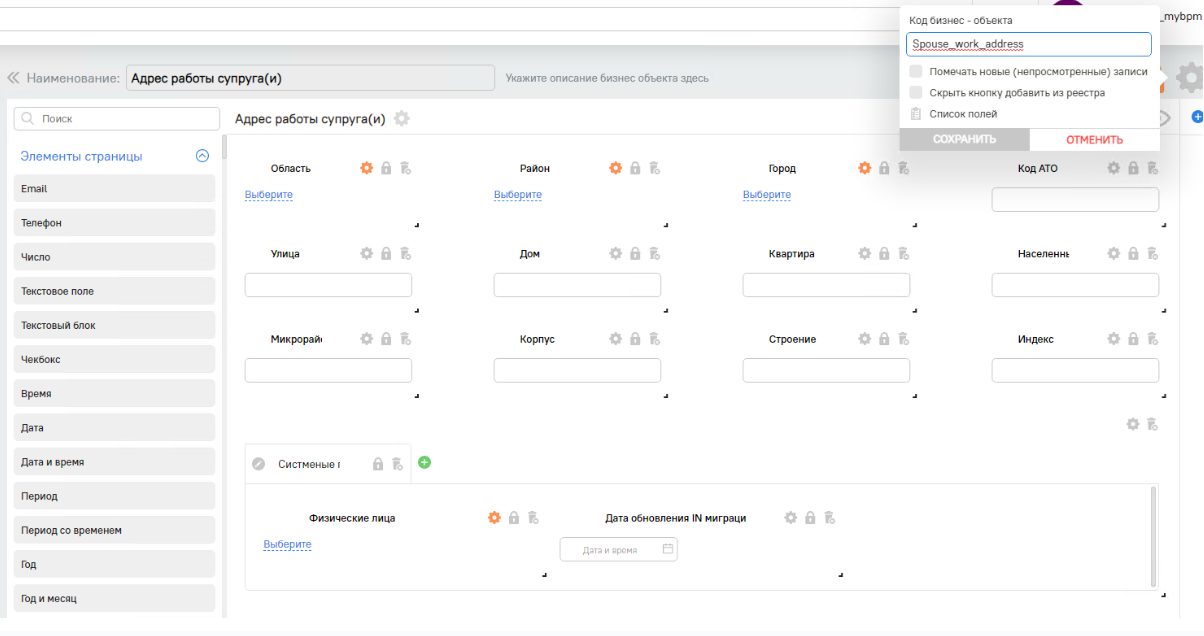
В данном случае вложенный КО ссылается на три БО - физические лица, юридические лица, ИП. Где Contracts\_id ссылается на id таблицы in\_Contracts, а Borrower\_id на id в таблицах in\_Natural\_client, in\_Legal\_client, in\_IP\_client

Таблица связка с таблицей физического лица.  
create table public.in\_Contracts\_Borrower\_Natural\_client (  
 Contracts\_id varchar(100),  
 Borrower\_id varchar(100),  
 in\_status varchar(100) default '' not null,  
 inserted\_date timestamp with time zone default clock\_timestamp(),  
 primary key (Contracts\_id, Billing\_PR\_Natural\_client\_id)  
);  
Таблица связка с таблицей юридического лица.  
create table public.in\_Contracts\_Borrower\_Legal\_client (  
 Contracts\_id varchar(100),  
 Borrower\_id varchar(100),  
 in\_status varchar(100) default '' not null,  
 inserted\_date timestamp with time zone default clock\_timestamp(),  
 primary key (Contracts\_id, Billing\_PR\_Natural\_client\_id)  
);  
Таблица связка с таблицей ИП.  
create table public.in\_Contracts\_Borrower\_IP\_client (  
 Contracts\_id varchar(100),  
 Borrower\_id varchar(100),  
 in\_status varchar(100) default '' not null,  
 inserted\_date timestamp with time zone default clock\_timestamp(),  
 primary key (Contracts\_id, Billing\_PR\_Natural\_client\_id)  
);

## Пример формирования OUT таблицы выгрузки данных из MyBPM

Для начала проверяется наличие схемы, если такая схема не существует, то создается по принципу:

* приставки out;
* кода компании в MyBPM;
* версии в коде сервера (code: OUT\_MIGRATION\_VERSION);
* версии в переменной окружения (yaml файл code:MYBPM\_ OUT\_MIGRATION\_VERSION); Пример: out\_jusan\_1\_8;



Код БО



Код поля

Создается таблица с названием соответственно названию БО с кодами полей.

create table Spouse\_work\_address (  
 id bigserial primary key,  
 system\_created\_at timestamp with time zone,  
 system\_creator\_fio varchar(100),  
 system\_id varchar(100) unique,  
 inserted\_at timestamp with time zone default clock\_timestamp(),  
 updated\_at timestamp with time zone,  
 Street text,  
 ATO\_code text,  
 Structure text,  
 corps text,  
 Microdistrict text,  
 Locality text,  
 Apartment text,  
 House text,  
 Index text  
);

Структура таблицы в описании:

|  |  |  |  |
| --- | --- | --- | --- |
| Имя колонки в DWH | Имя колонки MyBPM | Тип | Описание |
|  | Client\_id (==Global\_client\_ID) | int(8) | Global ID клиента |
|  | Address\_type | Справочник - Address\_type | Тип адреса |
|  | District | Справочник - District | Район |
|  | City | Справочник - City | Город |
|  | Region | Справочник - Region | Область |
|  | Street | VARCHAR(256) | Улица |
|  | House | VARCHAR(256) | Дом |
|  | Apartment | VARCHAR(256) | Квартира |
|  | ATO\_code | VARCHAR(256) | Код АТО |
|  | Locality | VARCHAR(256) | Населенный пункт |
|  | Microdistrict | VARCHAR(256) | Микрорайон |
|  | corps | VARCHAR(256) | Корпус |
|  | Structure | VARCHAR(256) | Строение |
|  | Index | VARCHAR(256) | Индекс |

## Системные поля

Поля являются системными:

* id bigserial primary key,
* system\_created\_at timestamp with time zone,
* system\_creator\_fio varchar(100),
* system\_id varchar(100) unique,
* inserted\_at timestamp with time zone default clock\_timestamp(),
* updated\_at timestamp with time zone.

# Как настроить OUT миграцию для старта

## Самое начало

kz/greetgo/mybpm/model\_kafka\_mongo/mongo/bo/field/BoFieldDto.java

Нужно проставить needUploadToOutTable=true в поля, которые должны быть мигрированы

## Генерация файла с DDL для просмотра таблиц, которые попадут в out миграцию

MigrationController

## Чтобы миграция запускалась нужно в конфиге

/mybpm/configs/OutMigrationPostgresConfig.txt

Поставить true у параметра hasOutMigration

## Дополнительно можно проставить в env yaml файле значение как в примере ниже для:

1) управления схемой, куда будут мигрировать данные с системы 2) управлением OutMigrationConsumer GroupId

`- { name: MYBPM\_OUT\_MIGRATION\_VERSION, value: "1" }`

## В отличие от in миграции, out работает с помощью кафки, и ловит изменения

KafkaBoi

## Конфиги нужные для миграции:

1. kz/greetgo/mybpm/register/config/out\_migration/OutMigrationDataSourceConfig.java  
2. kz/greetgo/mybpm/register/config/out\_migration/OutMigrationPostgresConfig.java

## Классы, где смотреть код:

1. kz/greetgo/mybpm/register/consumer/controller/out\_migration/OutMigrationConsumer.java  
2. kz/greetgo/mybpm/register/impl/migration/out/OutMigrationImpl.java  
3. kz/greetgo/mybpm/register/impl/migration/out/OutMigrationWorkerImpl.java

# Процесс OUT миграции

Out миграция принимает объект

KafkaBoi

И при помощи boId достается companyCode.

Out таблицы создаются в отдельной схеме в базе. Название схемы - out\_companyCode\_{KafkaTopic.OUT\_MIGRATION\_VERSION}\_{env.MYBPM\_OUT\_MIGRATION\_VERSION}

При первом разе идет проверка существует ли схема для out таблиц.

* Если их нет, то система создает out таблицы и запоминает их структуру.
* Если они есть, то мы берем структуру таблиц не создавая таблицы.

Дальше по boiId мы достаем

BoiDto

И при помощи структуры out таблиц делаем insert on conflict в нужные таблицы.

Название таблиц - boCode. Название колонок - fieldCode

Если мы имеем в БО поле типа БО или СО, то создается таблица которая их связывает.

Пример: 1) *Course* имеет поле *Teacher(fieldType=BO)*. В таком случае для Course будет таблица *Course* и \_Course\_Teacher\_

2) *Contract* имеет поле *Borrower(fieldType=CO)*. *Borrower* здесь поле, которое хранит 3 разных БО. Здесь будет таблица *Contract* и \_Contract\_Borrower\_. Из-за того, что Contract\_Borrower имеет тип СО, он хранит id-шки 3 разных таблиц.

**ВАЖНО!** Для того, чтоб добавить новые поля или БО-шки для out миграции необходима менять:

KafkaTopic.OUT\_MIGRATION\_VERSION (обязательно инкремент)

Или более **предпочтительный вариант**:

В environment-е MYBPM\_OUT\_MIGRATION\_VERSION (обязательно инкремент)