Работа с Платформой Б3

ДЛЯ РАЗРАБОТЧИКОВ

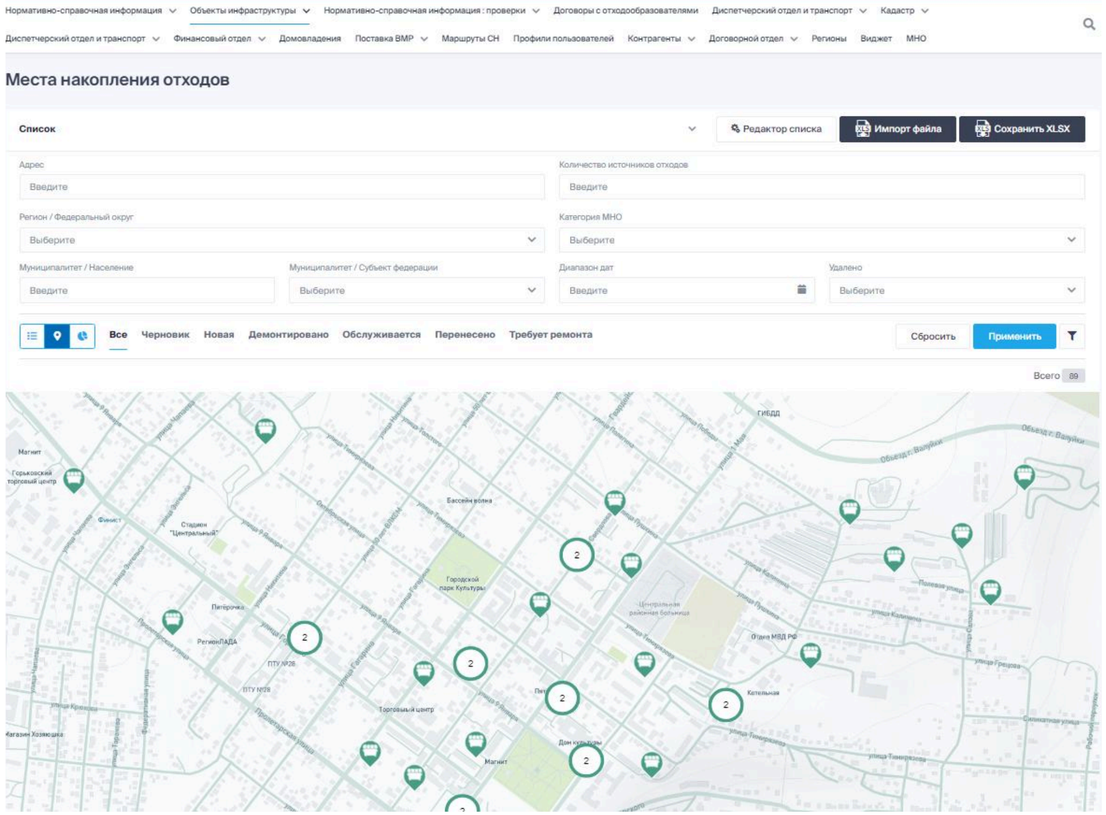
БОЛЬШАЯ ТРОЙКА

1. Введение

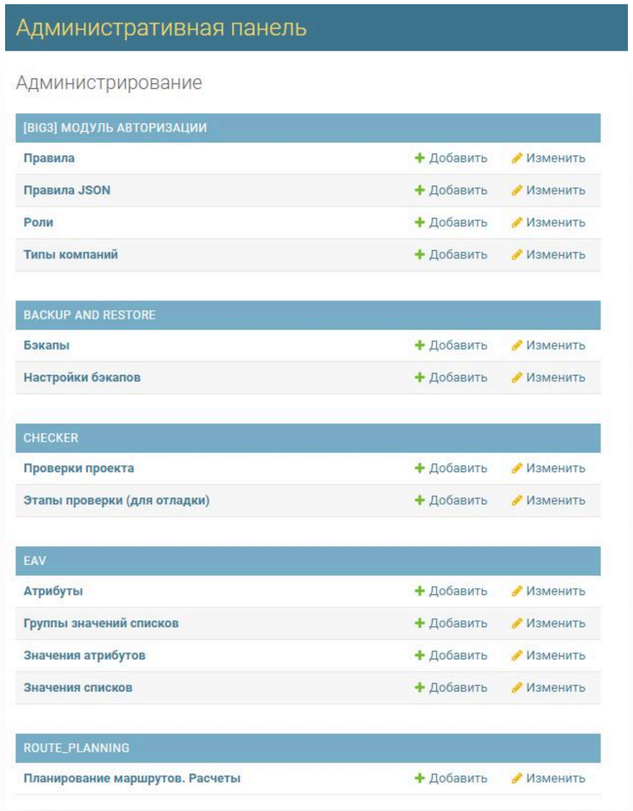
1.1. Что такое Платформа БЗ?

Платформа БЗ - это фреймворк, на базе которого разрабатываются государственные информационные системы и enterprise-системы. Платформа состоит из базовых и опциональных модулей.

Веб-интерфейс (Личный кабинет пользователя)



Админ-панель



1.2. Документация на Платформу БЗ

- **Техническая документация** по Платформе БЗ собирается из .md-файлов в Gitlab и расположена здесь: http://docs.dev.big3.ru/ (доступ предоставляется по запросу)
- **Руководство оператора**, в котором можно найти подробные инструкции на основной функционал Платформы БЗ, расположено здесь: https://platform.big3.ru/
  А также доступно для скачивания по прямой ссылке.

1.3. Как развернуть проект локально

Про развертывание можно узнать здесь:

- Развертывание бэкенда
- Развертывание фронтенда

1.4. Как настроить простую форму для тестов

Для создания простой формы для тестирования разрабатываемого функционала необходимо:

- Создать набор данных.
- Настроить форму списка по созданному набору данных и добавить поля на форму.
- При необходимости создать форму элемента (детальную страницу) и настроить её.
- Создать пункт меню.
- При работе с операциями, а также при возникновении проблем с правами доступа, настроить необходимые права.

Для тестирования разрабатываемого функционала также могут быть полезны **собственные dev-инструменты Платформы БЗ**. Все они располагаются в Личном кабинете в профиле пользователя с ролью “Администратор”.

- Dev: Настройки приложения - /profile/dev/settings
- Dev: Демо UI KIT - /profile/dev/ui-demos/main
- Dev: Тосты и ошибки - /profile/dev/ui-demos/toast

1.5. Использование формул в Платформе БЗ

В Платформе БЗ реализована возможность **обработки данных формы с помощью формул**. Например, можно вывести значение поля в зависимости от выполнения условий; вычислить значение поля, используя различные арифметические, статистические функции; можно скрыть поле, сделать его обязательным, изменить его внешний вид в зависимости от значений других полей и т.п.

В формулах можно обращаться к **различным данным из форм и объектов** (например, использовать значение другого поля формы или получить данные пользователя). Для обращения к таким данным используются **контексты данных**.

Разделы документации по теме:

- Контексты данных
- Формулы
- Основные операторы, доступные в формулах
- Доступные методы в формулах
- Доступные методы в объекте FormcyMoment

Для удобной проверки работоспособности формул реализована **песочница формул (ЛК → Профиль → Dev: Инструменты)**. Песочница доступна только для роли “Администратор”.

2. Бэкенд. Создание модели данных

2.1. Основные особенности моделей

Для моделей существует несколько **родительских классов**. Новые модели необходимо наследовать от родительских.

- **BaseModel** – базовый класс модели с привязкой к контрагенту.
  Пример: class TestModel(BaseModel)
- **GeoPointModel** - класс для объектов типа точка, который должен содержать координаты и привязку к муниципалитетам и регионам.
  Пример: class TestModel(BaseModel, GeoPointModel)
- **GeoObject** – класс для объектов типа полигон, содержит поля для 3 уровней детализации геометрии.
  Пример: class TestModel(GeoObject)
- **EavModelHistory** – класс для хранения истории изменений внутри модели. Позволяет хранить историю не только для полей модели, но и для eav-атрибутов. В настройках типа представления появится отметка в параметре «Хранить историю». По умолчанию хранение истории отключено.
  Пример: class TestModel(EavModelHistory)
- **BaseModelReport** – класс для подключения операций импорта и экспорта в Excel. Уже содержится в BaseModel, может быть подключен к модели Clickhouse.

Модели Clickhouse

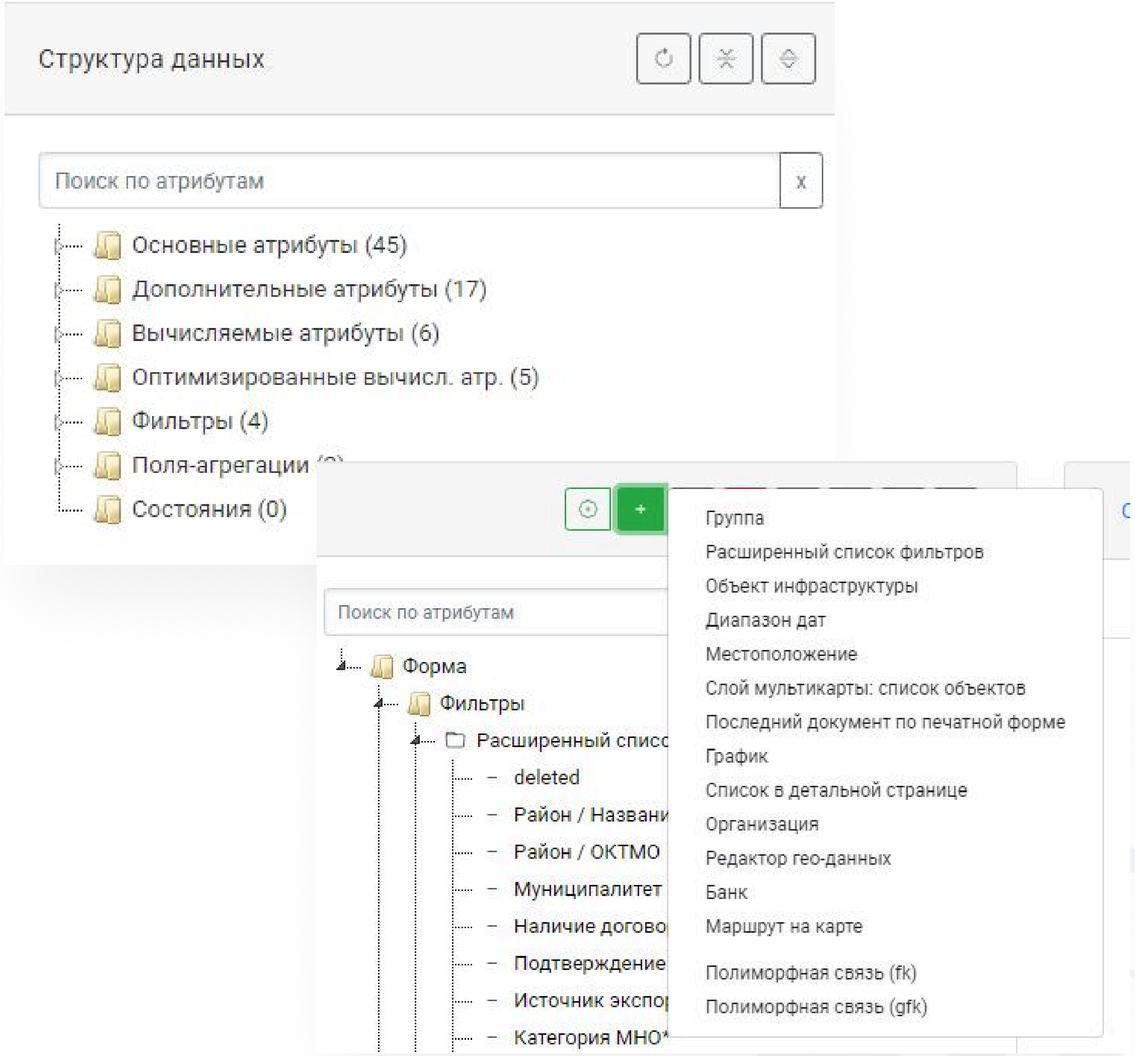
- При создании всегда необходимо наследоваться от ExtendedClickHouseModel.
- Поля в модели использовать из clickhouse\fields.py.
- В остальном необходимо руководствоваться документацией библиотеки django_clickhouse.

2.2. Виды атрибутов

В Платформе Б3 используются различные виды атрибутов. Атрибуты разбиты по видам в блоке “Структура данных” в админ-панели при создании/изменении формы:

- **Основные атрибуты** – это атрибуты модели.
- **Дополнительные атрибуты** - json-атрибуты.
- **Вычисляемые атрибуты только для чтения** – это функции в модели, помеченные декоратором @b3_readonly_attribute. Не изменяемые.
- **Оптимизированные вычисляемые атрибуты** - @b3_annotation_attribute декоратор в функции.
- **Фильтры** - динамические фильтры (b3_filter_attribute).
- **Комплексные атрибуты** - это комплексные компоненты. Они добавляются при создании/изменении формы в среднем блоке (в структуре формы).

Подробное описание по каждому виду атрибутов можно найти в документации по соответствующим ключевым словам.



2.3. Переопределение моделей

В коде Платформы БЗ выделено ядро, отраслевые решения и конкретные проекты. Это позволяет **переопределять модели внутри проекта или отраслевого решения**.

Например, модель контрагента (Participant) находится в ядре Платформы БЗ. Т.е. данная модель будет присутствовать в каждом новом проекте, реализованном на Платформе БЗ. Но для каждого проекта нужны различные дополнительные функции и поля в этой модели. Если их добавить на Платформе БЗ, то они появятся во всех проектах.

Поэтому разработчик в рамках конкретного проекта **может написать класс, в котором будет находиться расширение для определенной модели** (операции, атрибуты, переходы, метод Save), **и зарегистрировать новую модель**.

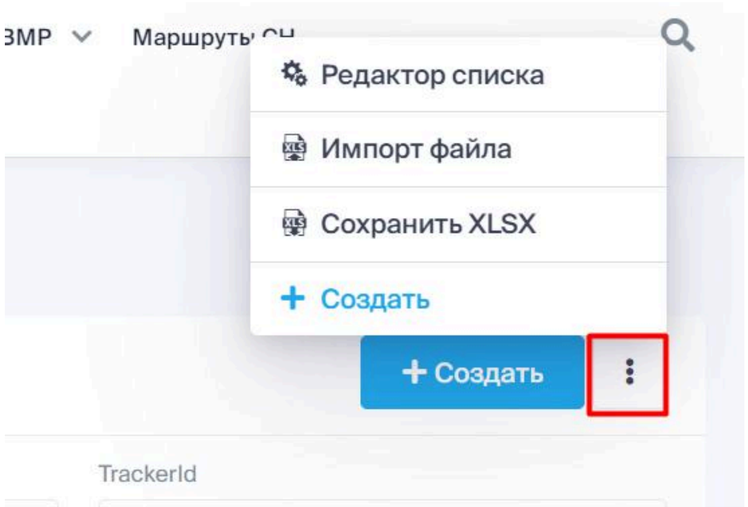
Пример создания и регистрации нового класса:

```
from django.db import models
from django.db.models import Count, ExpressionWrapper
from eav.models import Attribute
from form_builder.annotations import fb_expression_wrapper
from form_builder.decorator import fb_dynamic_attribute, fb_readonly_attribute
from onboarding.models import Course
from workflow.custom_logic import register_custom_atrs
from workflow.decorators import operation, transition

class TestCourse:
    @b3_filter_attribute("Только свои")
    def course_filter(self, queryset, value):
        if value:
            queryset = queryset.filter(category_id=value)
        return queryset

    @b3_readonly_attribute("Наличие разрешения", Attribute.TYPE_BOOLEAN)
    def has_permission(self):
        return 'Yes or no - I dont know'

    @operation(title='Подписать', operation_type='all',
access_rules_from='post')
    def sign(self):
        return 'TestCourse_operation'

    @classmethod
    @fb_annotation_attribute("TestCourse_fb_expression_wrapper", "Накопительный
итог", Attribute.TYPE_FLOAT)
    def my_courses(cls):
        expression = ExpressionWrapper(Count('category'),
output_field=models.FloatField())
        return expression

    @transition(title='Отменить формирование отчёта')
    def cancel_generation(self):
        return 'TestCourse_transition'

register_custom_atrs(TestCourse, Course)
```

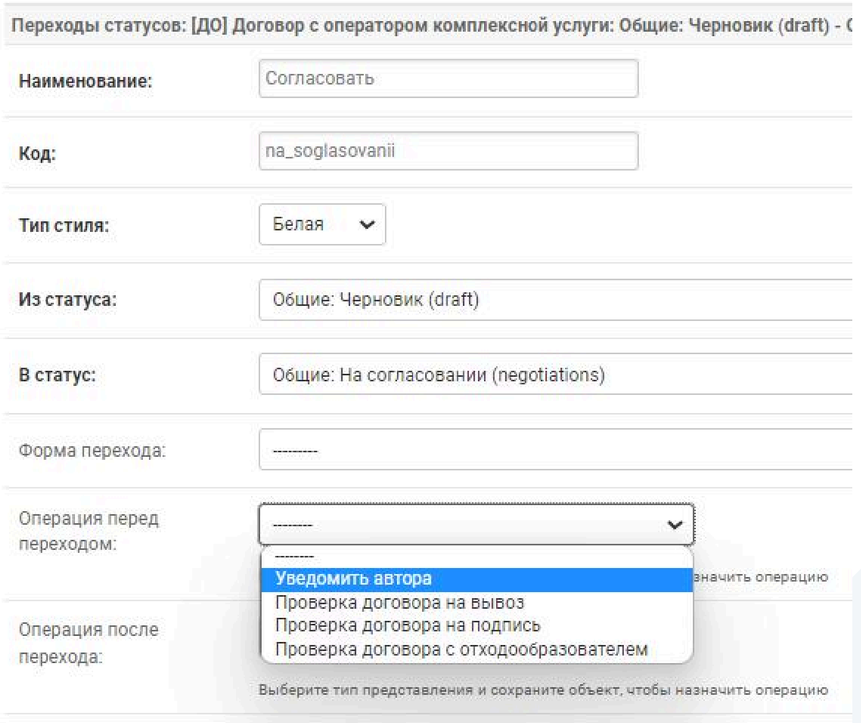
2.4. Операции

Операции над моделями - это действия на форме, которые производятся с объектом или с несколькими объектами сразу. У каждой операции есть тип:
- **all** - действие осуществляется со всем списком целиком;
- **partial** - действие осуществляется с выбранными элементами списка;
- **one** - действие осуществляется с единственным элементом списка;
- **detail** - действие осуществляется с единственной сущностью;
- **custom** - действие осуществляется только при возникновении определённых событий.

Подробное описание см. здесь.



Операции при переходах статусов - это действия, которые выполняются над объектами перед или после изменения статуса (нажатия кнопки перехода статуса).



3. Фронтенд

3.1. Основные фронтенд-репозитории Платформы Б3

1. Репозиторий Big3 Angular Shared - https://git.big3.ru/analytics/big3-angular-shared
Содержит переиспользуемые компоненты и модули:

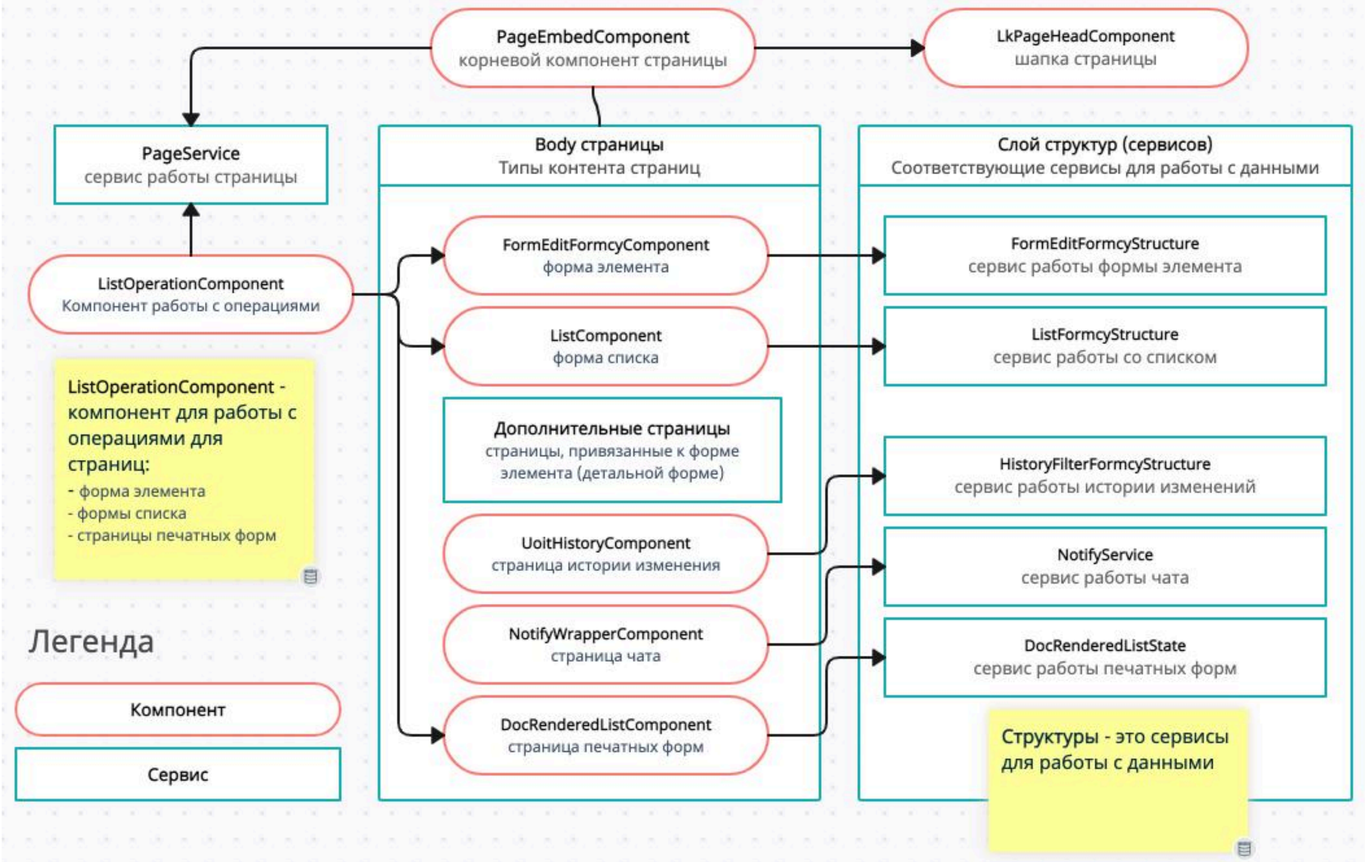
- components – компоненты, изолированные от formcy-конструктора. Используются внутри formcy-компонентов или страницах, не связанных с Конструктором.
- modules
  - **formcy** - core-модуль Конструктора.
  - **formcy-uoit** - расширение core-модуля Конструктора для работы с формами, списками, формами перехода и т.д.
- ui-kit
  - SCSS-файлы стилей компонентов
  - SCSS-переменные для тем (на основе subapp)
  - утилиты для работы с переменными цвета в .ts и .html файлах

2. Фронтенд-приложение находится в папке **front** основного репозитория и подключает библиотеку **Big3 Angular Shared** как сабмодуль.

3.2. Структура страницы

- **PageEmbedComponent** - основной компонент страницы, который отображает:
  - PageHead
  - PageBody - соответствующий тип страницы
    - Форма списка
    - Форма элемента
    - Печатная форма
    - Чат
    - История изменений

- **PageService** - основной сервис страницы, который получает и преобразовывает конфиг формы для дальнейшей инициализации:
  - Инициализирует шапку страницы (табы переходов на related страницы, заголовок страницы, работа по переходам по табам);
  - Создает хлебные крошки;
  - Формирует id для компонентов "Печатные формы", "Чат";
  - Формирует srcTpl формы списка/элемента, т.е формирует body страницы;
  - Формирует сабменю (временно не работает).



3.3. Структура формы списка

Основным компонентом списка является **listComponent**, который может использоваться как:

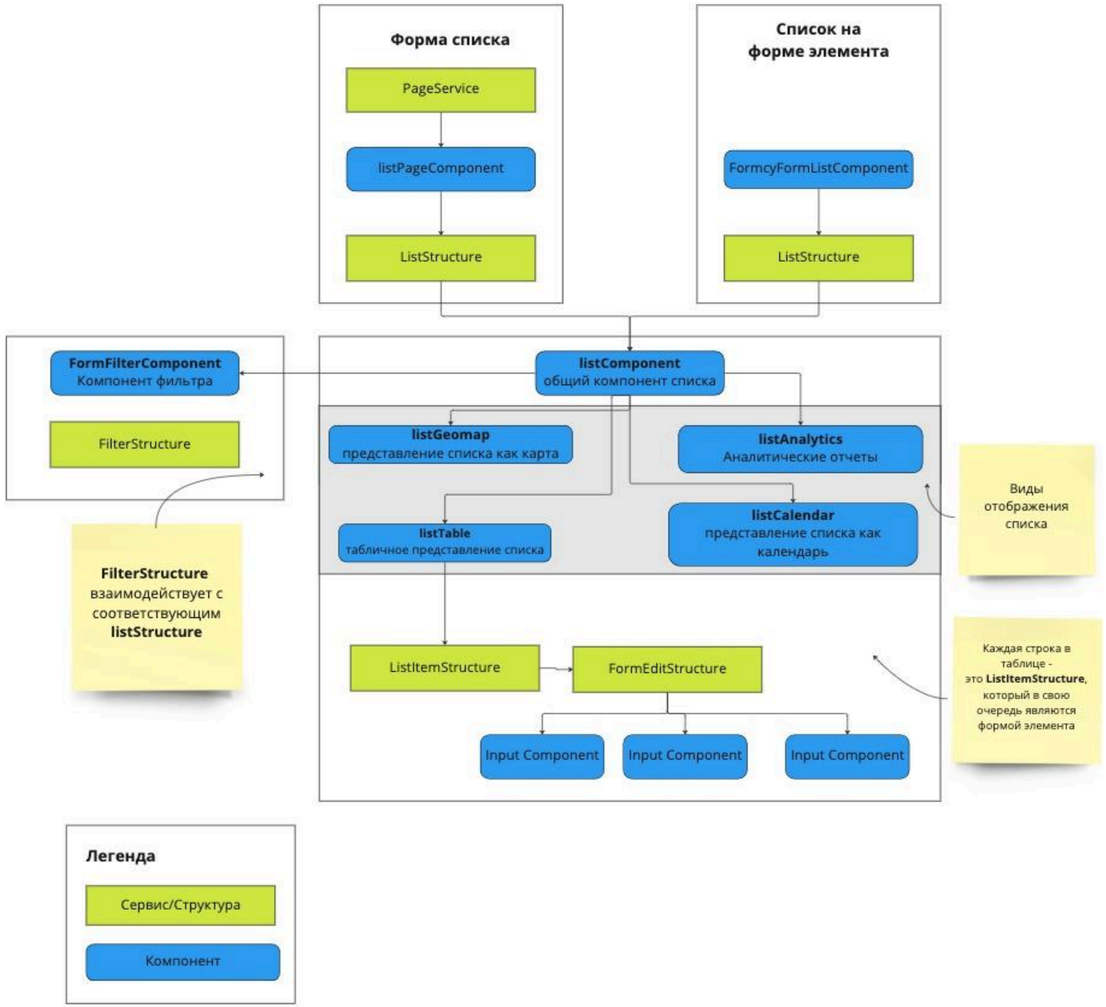
- вид формы списка (родительский компонент listPageComponent);
- компонент для формы элемента (родительский компонент FormcyFormListComponent).

Родительский компонент создает на основе данных с API **ListStructure** - основную структуру (сервис) для работы:

- с данными;
- взаимодействия с прочими структурами (FilterStructure, listItemStructure).

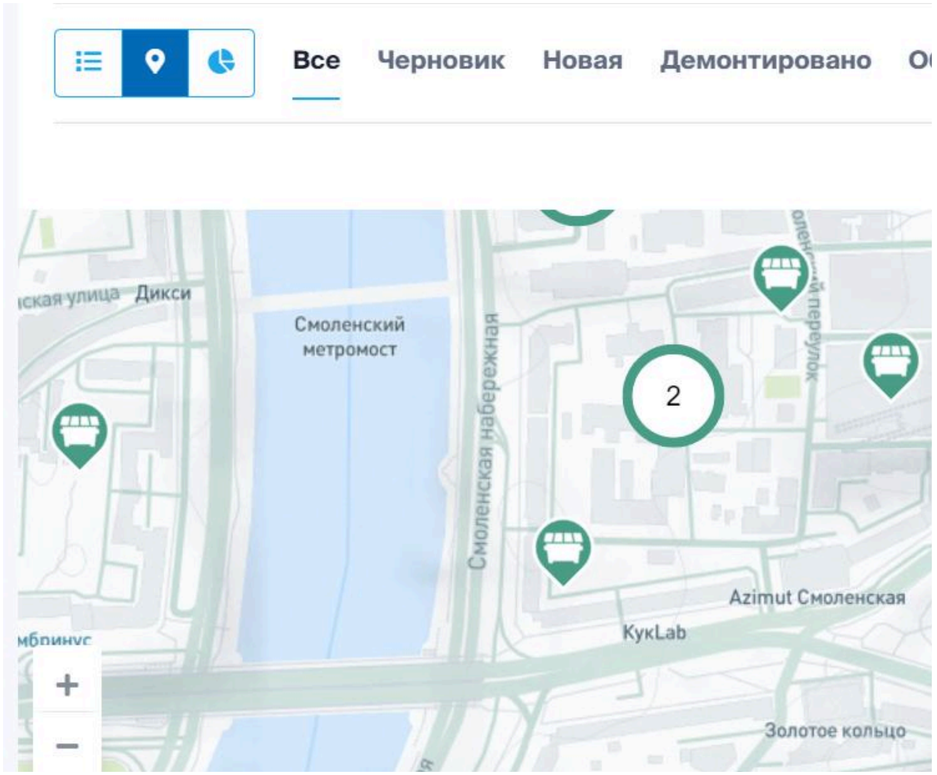
Форма списка имеет **4 вида отображения** данных:

- таблица (listTable);
- карта (listGeomap);
- календарь (listCalendar);
- аналитика (listAnalytics).

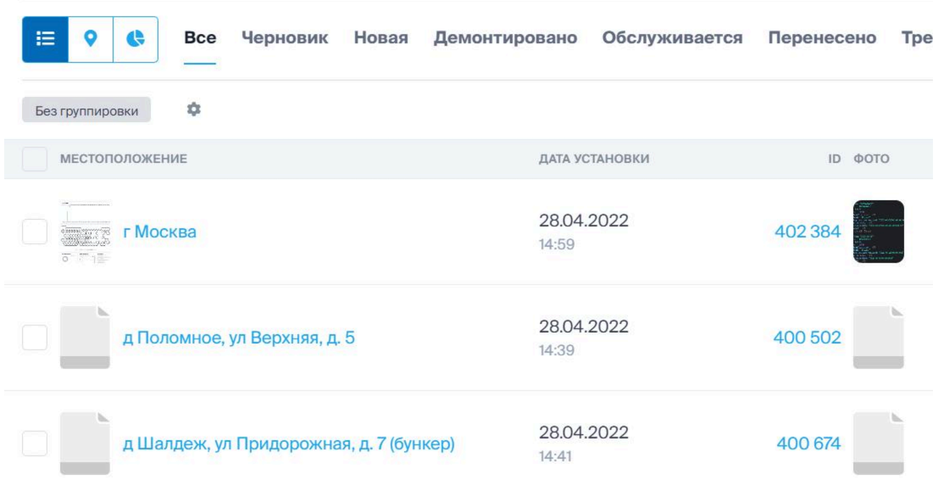


3.3.1. Структура формы списка. Виды отображения

Список-календарь и список-карта наиболее простые виды отображения, т.к. используются только для отображения данных с API.



Список-таблица состоит из listItemStructure (каждая строка таблицы). Т.е. строка таблицы - форма элемента, а ячейка таблицы - компонент формы элемента. Соответственно форма элемента вляется простейшей формой редактирования.



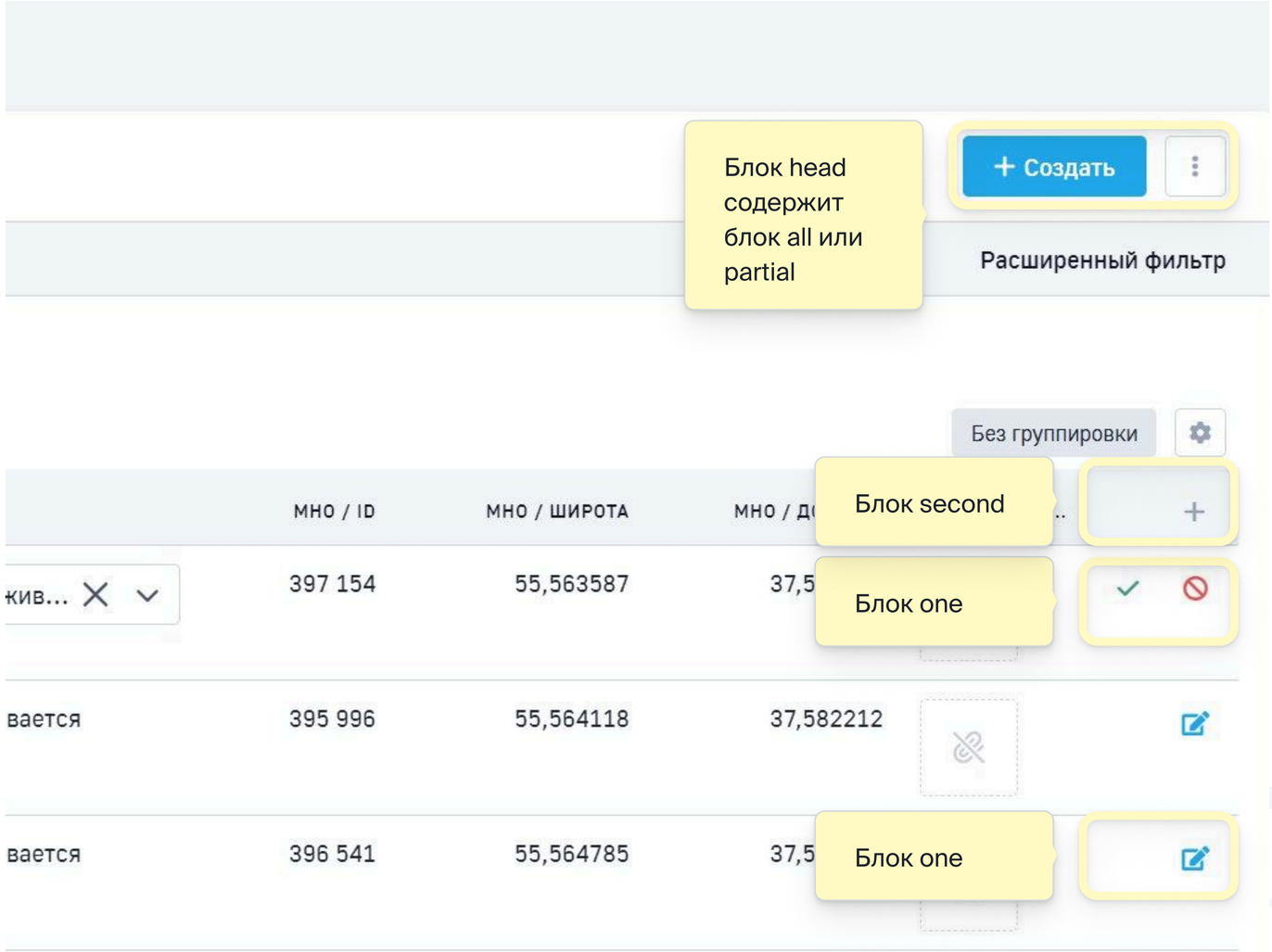
3.3.2. Структура формы списка. Блоки кнопок

В шапке **всех страниц содержится блок кнопок head**, который в зависимости от условий на конкретной странице может содержать кнопки какого-то блока этой страницы.

Форма списка содержит блоки **all, partial, one, second и empty**. В блок **head** попадают кнопки блока partial, если выбран хотя бы один элемент списка. Иначе в head попадают кнопки блока all.

Блок **empty** отображается тогда, когда список пустой. Обычно в этом блоке предлагается полезное для такого случая действие.

Подробное описание блоков кнопок можно найти в документации Платформы БЗ.



3.4. Структура формы элемента

Для отображения формы элемента используется **FormEditFormcyComponent**, но в отличии от формы списка **структура формы элемента** (FormEditStructure) создается в **pageService**.

FormEditStructure в pageService создается и инициализируется только работа формул. Это необходимо для того, чтобы выполнять формулы из конфигурации для чата, печатных форм, списка и создания.

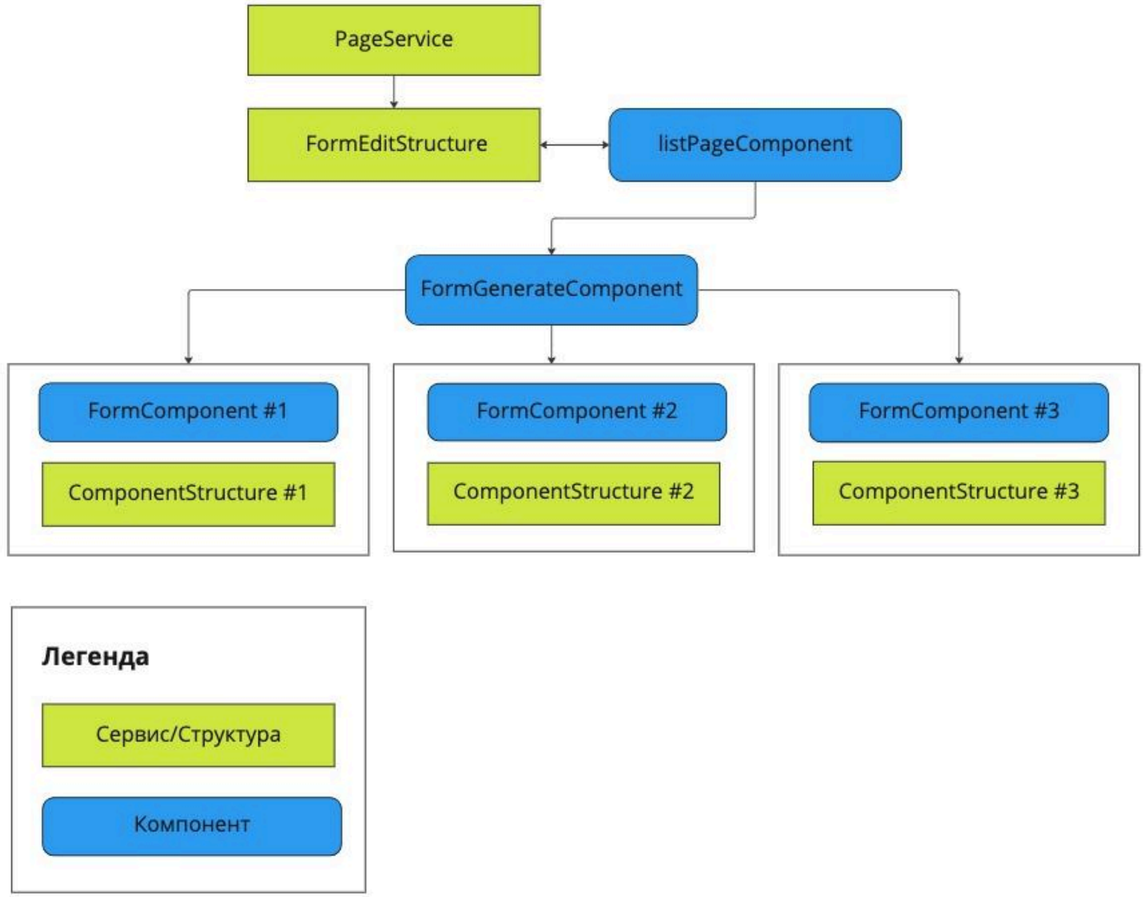
Пример:

```
objectApi: "wf__waste_site__waste_site/{data.id}"
chatRoomFilterKey: "waste_site_object__{data.id}"
docScanFilterKey: "wastesite__{data.id}"
```

Таким образом **конфигурация формы может кэшироваться**, так как "упрощенная" инициализация FormEditStructure позволяет выполнять формулы для всей страницы на основе данных.

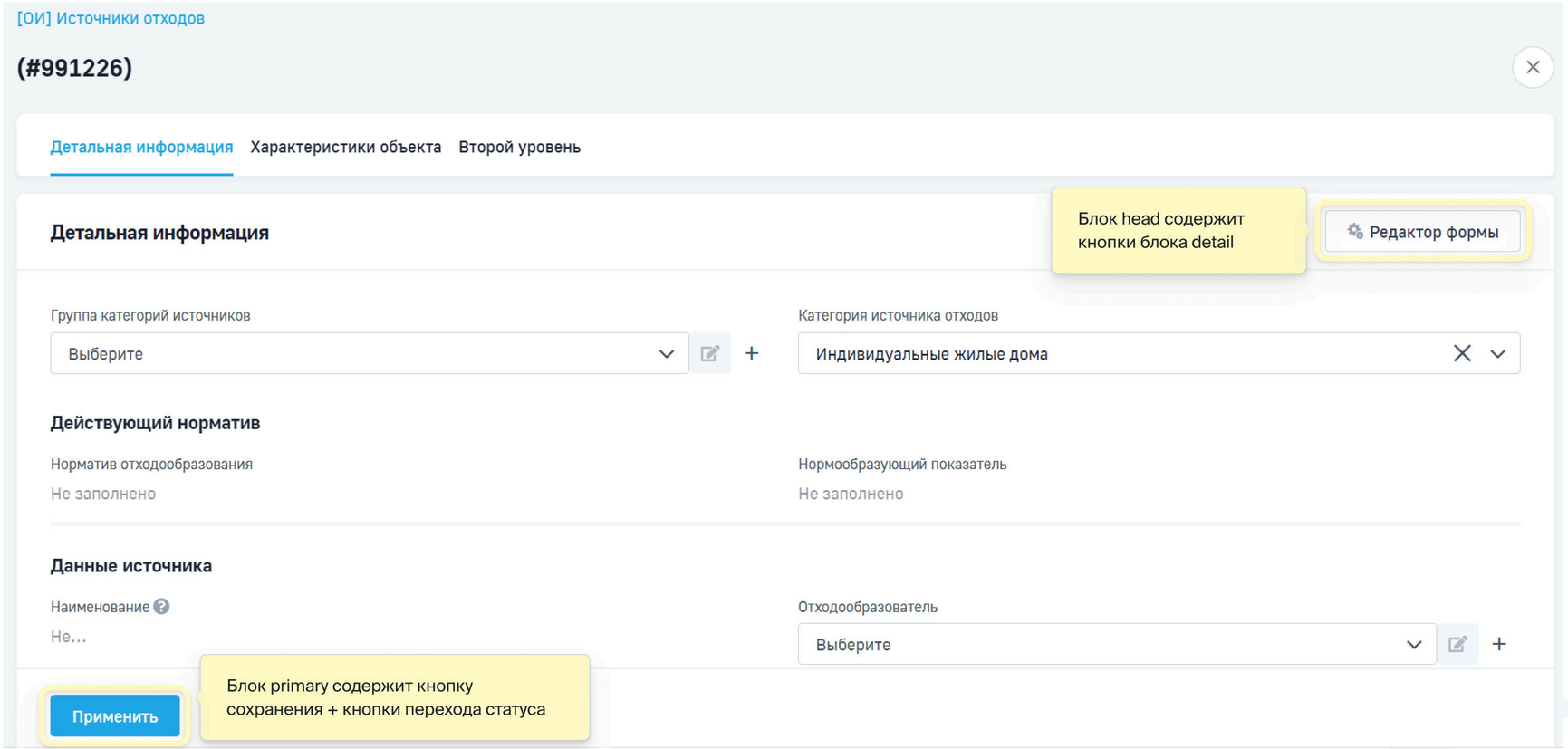
При этом инициализация **FormEditStructure не происходит в корневой форме списка**, т.к. она не имеет родительской формы элемента.

Далее, если это форма элемента, то **инициализируется шаблон формы в компоненте FormEditFormcyComponent** для отображения элементов формы.



3.4.1. Структура формы элемента. Блоки кнопок

Форма элемента содержит блоки **detail и primary**. В блок **head** попадают кнопки блока detail.

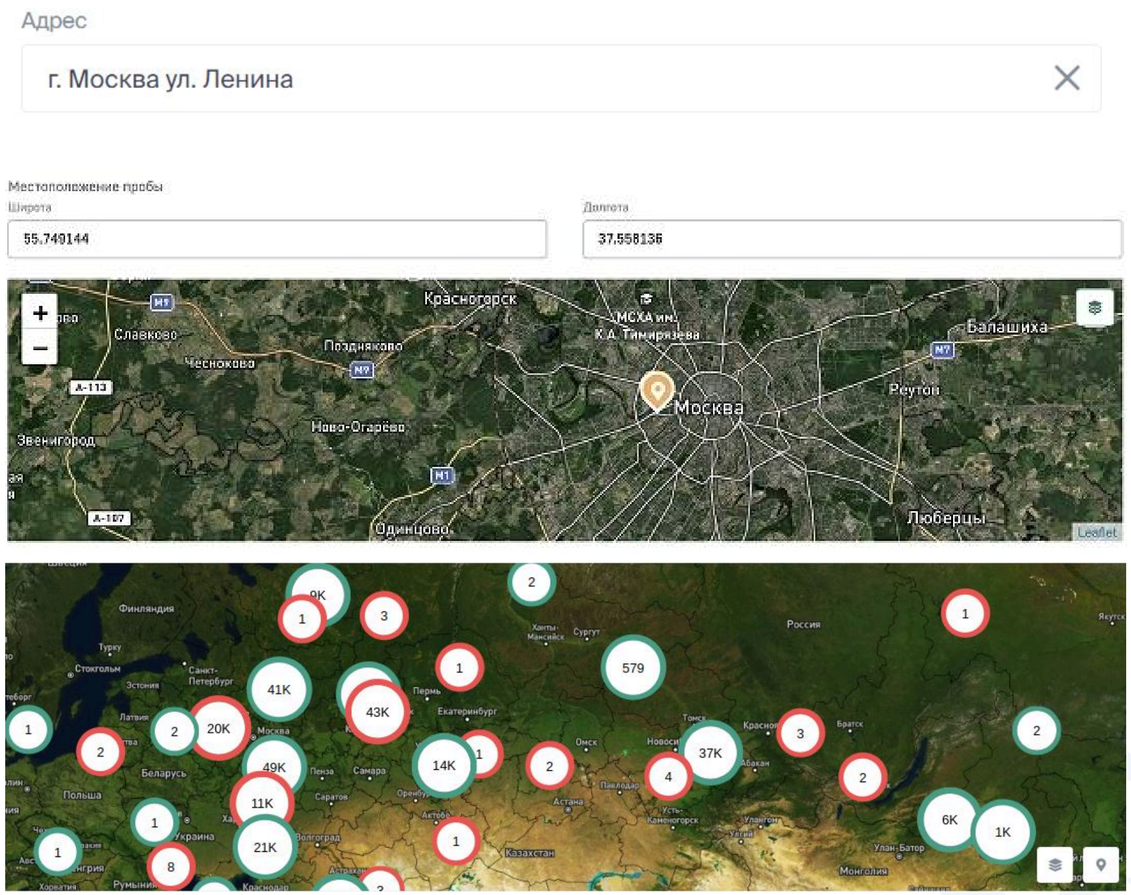


4. Создание нового компонента на Платформе Б3

4.1. Виды компонентов

На Платформе БЗ есть **4 вида компонентов**:

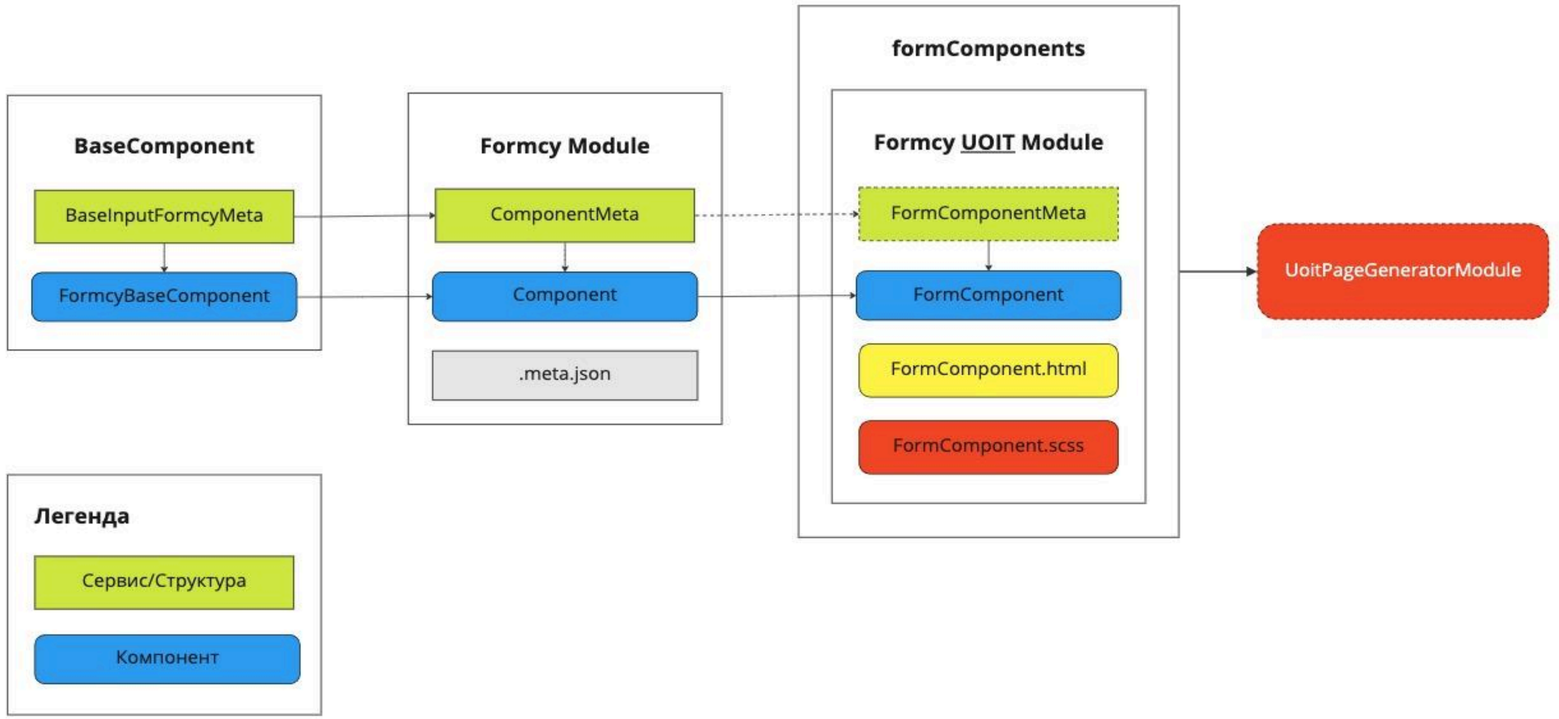
- **Компонент, привязанный к данным** (наследуемый от BaseInputFormcyMeta). Например, Text, Date, SelectObject.
  Это компонент, у которого есть valuePropName (иногда несколько), и который доступен только для поддерживаемого типа данных. Например, компонент Date/DateTime можно выбрать только для типа данных date, datetime.
- **Компонент группы** (наследуемый от BaseGroupFormcyMeta). Например, Group.
  Содержит в себе другие группы и/или массив компонентов для объединения и отображения группы полей.
- **Комплексный компонент** (наследуемый от BaseFormcyMeta/ BaseGroupFormcyMeta). Например, GeoPoint, Bank.
  Явно привязан к данным модели данных и добавляется в админ-панели через контекстное меню. Это самостоятельный компонент, у которого в .meta.json можно создавать сколько угодно связей с данными и работать с ними по аналогии с ValuePropName.
- **Комплексный компонент 2.0** (наследуемый от BaseFormcyMeta/ BaseGroupFormcyMeta). Например, timelineChartMulti, Organization.
  Обычно наследуется от компонента группы, и является родительским управляющим компонентом для вложенных. Т.е. в этом компоненте можно изменить отображение, поведение или связать событиями несколько полей.



4.2. Структура компонента

У всех компонентов есть следующие составляющие:

- **.meta.json** - конфиг компонента для отображения в админ-панели.
- **metaComponent** - структура компонента, отвечающая за работу с данными (преобразование, обновление, передача в обьект значений формы (objectData)).
- **component** - angular-компонент, реализующий функциональную и визуальную составляющую элемента формы.



4.3. Создание компонента на бэкенде

- Новые компоненты должны наследоваться от **form_builder.forms.BaseAttrForm** и **form_builder.forms.DecorationAttrForm**.

Например:

```
class DateRangeAttrForm(BaseAttrForm, DecorationAttrForm):
      """ Диапазон дат """
```

- В классе Meta необходимо **определить переменные:**
  - **icon** - иконка, которая будет показываться в Конструкторе;
  - **verbose_name** - наименование компонента;
  - **component** - техническое наименование компонента.

Например:

```
class Meta(BaseAttrForm.Meta):
      component = FrontConfigCommon.COMPONENT_DATE_RANGE
      verbose_name = 'Диапазон дат'
      icon = 'bi-calendar-range'
      represent_template = 'Диапазон дат ({field_from}-{field_to})'
```

- Также необходимо посмотреть **методы в родительских классах и переопределить их в случае необходимости.**
  Например, такие методы как: init, get_config_front, get_init_val.

4.4. Создание компонента на фронтенде

1. В core-модуле formcy создать 3 файла:

- .meta.json - файл конфигурации;
- <componentName>.formcy.component.ts - базовый компонент с наследованием от FormcyBaseComponent;
- <componentName>.formcy.meta.ts - структура для работы с данными с наследованием от BaseInputFormcyMeta.

2. В модуле formcy-uoit создать 3 файла:

- <componentName>form-formcy.component.ts - компонент с наследованием от <componentName>.formcy.component.ts;
- <componentName>form-formcy.component.scss - стили компонента;
- <componentName>form-formcy.component.html - html-шаблон компонента;

В результате получается обычный компонент Angular, который занимается отображением компонента, и у которого есть слой данных meta.

3. Добавить компонент в formComponents и formComponentsMap для последущего импорта в UoitPageGeneratorModule.

4. Добавить компонент в enum FFComponentName:

```
 {
     alias: "<название компонента из .meta.json поля code>"
 }
```

5. Пересобрать бэкенд с новым @big3-shared.

5.1. Типовой .meta.json

```
{ "name": "Галочка",
   "code": "Checkbox",
   "additionalParams": [
    {
      "name": "<название доп. параметра, которое будет
отображаться в админ-панели>",
      "code": "<code доп. параметра, который придет с
бэкенда>",
      "type": "<boolean | list | number | string>"

      "listData": [{
         "name": "<name>",
         "code": "<code>"
       }]
    },
 ]
}
```

Ура! Ты изучил всю вводную информацию!