



Работа с Платформой Vig3

ДЛЯ РАЗРАБОТЧИКОВ



1. Введение

Процессы. Документация. Начало работы



1.1. Что такое Платформа?

Платформа Vig3 представляет собой решение, на базе которого разрабатываются проекты в сфере экологии: для учета и мониторинга обращения с отходами, мониторинга состояния и загрязнения окружающей среды, атмосферного воздуха и т. п. Платформа состоит из 2х основных частей:

Веб-интерфейс (Личный кабинет пользователя)

Админ-панель (Конструктор Платформы)



1.2. Документация

Документация по Платформе хранится в 2 местах:

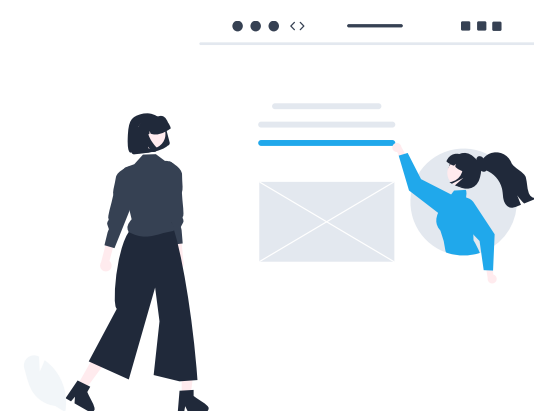
- ✓ **Общая информация**, а также основные инструкции по работе с Платформой хранятся в Wiki: <https://wiki.yandex.ru/platform/> (доступ только под email `***@big3.ru`)
- ✓ **Техническая документация** по Платформе собирается из .md-файлов в Gitlab и расположена здесь: <http://docs.dev.big3.ru/> (доступ можно найти в статье "[Полезные ссылки](#)")

Полезные статьи в Wiki, которые пригодятся в работе:

- ✓ [Полезные ссылки](#)
- ✓ [Используемые стенды](#)
- ✓ [Команда Платформы](#) (к кому можно обратиться с вопросами)
- ✓ [Code style, Best practices](#)



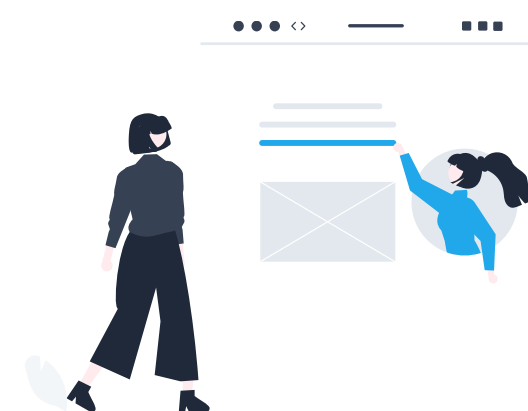
1.3. Собрания команды



Дейли

С 11.00 до 12.00 Мск.
Обсуждение текущих вопросов по задачам.

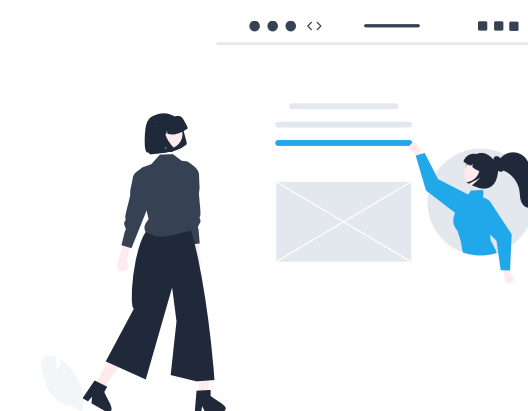
На дейли мы обсуждаем вопросы текущего дня. Срочные вопросы можно обсудить и в чате команды, но на дейли это делается быстрее и в режиме диалога.



Планирование спринта

По пн 1 раз в 2 недели с 10.00 Мск.
Обсуждение и оценка задач нового спринта.

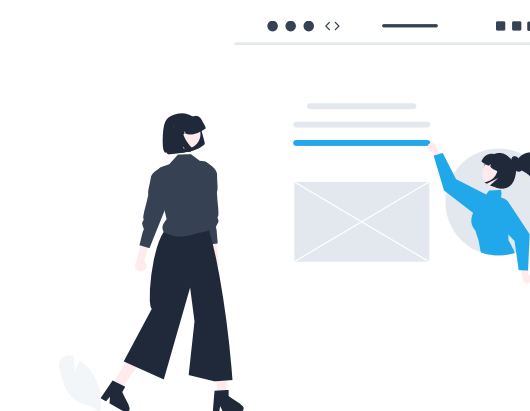
У каждого разработчика одинаковый набор очков истории на спринт - 48 sp. Задачи бэклога оцениваются в sp на покере планирования. Митинг заканчивается, когда все участники команды набрали себе задач на полный спринт.



Демо-показ спринта

С 13.00 до 16.00 Мск.
Показ разработанных задач прошедшего спринта. На основе показа составляется [Краткий обзор по ключевым изменениям в спринте](#).

На демо мы показываем результаты нашей работы. Каждый участник демонстрирует другим свои результаты спринта. Часто мы приглашаем на демо участников других команд и других коллег.



Ретро

Раз в 1-2 спринта.
Обсуждение проблем в процессах, предложения по улучшению работы в команде.



1.4. Как отдать задачу на тест и что делать с протестированной задачей

Для ведения задач на Платформе используется [Яндекс.Трекер](#).

- ✓ Разработка ведется **спринтами по 2 недели**. Задачи оцениваются на планировании в начале спринта.
- ✓ Задачи разрабатываются **в порядке приоритета** - от блокеров к задачам с низким приоритетом.
- ✓ После проведения оценки можно начинать работу над задачей. Полный **процесс движения задач в трекере** описан в [wiki-статье](#). При работе над задачей не забывайте обновлять в ней статусы.



Перед началом работы над задачей также необходимо изучить [процесс разработки и модель ветвления](#).

После выполнения задачи необходимо отправить ее на **тестирование и ревью**. Процесс подготовки задачи и отправки приведен в [wiki-статье](#).

Разработчик заканчивает работу над задачей, когда она переходит в **статус “Протестировано”**. Руководитель проекта закрывает ее после того, как она попала на боевой стенд или в версию.



1.5. Видео-онбординг

В этих видео собрана основная информация по Платформе:

- ✓ [Онбординг в Платформу. Часть 1:](#) архитектура, основные компоненты Платформы, знакомство с Конструктором.
- ✓ [Онбординг в Платформу. Часть 2:](#) ролевая модель в Платформе, основные эндпоинты, аналитические панели.
- ✓ [Онбординг в Платформу. Часть 3:](#) базовый и расширенный функционал Конструктора, типы атрибутов.
- ✓ [Онбординг в окружение фронтенда:](#) стенды фронтенда, взаимодействие фронт-стендов с бэк-стендами. Настройки окружения фронт-стенда.



1.6. Как развернуть проект локально

Про развертывание можно узнать здесь:

- ✓ [Развертывание бэкенда](#)
- ✓ [Сборка фронт-стенда локально \(Angular Build/Serve, Docker Compose\)](#)



1.7. Как настроить простую форму для тестов

Для создания простой формы для тестирования разрабатываемого функционала необходимо:

- ✓ [Создать тип представления и добавить дополнительные атрибуты при необходимости.](#)
- ✓ [Настроить форму списка по созданному типу представления: добавить поля на форму, а также вывести ее в Личном кабинете.](#)
- ✓ [При необходимости создать форму элемента \(детальную страницу\) и настроить её.](#)
- ✓ [При работе со статусами также необходимо создать бизнес-процесс со статусами.](#)
- ✓ [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**. Все они располагаются в Личном кабинете в профиле пользователя с ролью “Администратор”.

- ✓ [Dev: Настройки приложения](#)
- ✓ [Dev: Демо UI KIT](#)
- ✓ [Dev: Тосты и ошибки](#)



1.8. Использование формул в Конструкторе

В Конструкторе 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**.

- ✓ [Общее описание реализации формул и доступных контекстов данных](#)
- ✓ [Основные операторы, доступные в формулах](#)
- ✓ [Доступные методы в формулах](#)
- ✓ [Доступные методы в объекте FormsyMoment \(объект, который связан с какой-то конкретной датой, временем и таймзоной\)](#)

Для удобной проверки работоспособности формул реализована **песочница формул (ЛК → Профиль → Dev: Инструменты)**. Песочница доступна только для роли “Администратор”. Краткую инструкцию по использованию песочницы см. [здесь](#).



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 используются различные виды атрибутов. Атрибуты разбиты по видам в Конструкторе в блоке “Структура данных” при создании/изменении формы:

- ✓ **Основные атрибуты** – это атрибуты модели.
- ✓ **Дополнительные атрибуты** - еав-атрибуты. Подробное описание см. [здесь](#).
- ✓ **Вычисляемые атрибуты** – это функции в модели, помеченные декоратором `@fb_readonly_attribute`. Не изменяемые. Подробное описание см. [здесь](#).
- ✓ **Оптимизированные вычисляемые атрибуты** - `@fb_expression_wrapper` декоратор в функции. Подробное описание см. [здесь](#).
- ✓ **Фильтры** - динамические фильтры. Подробное описание см. [здесь](#).
- ✓ **Комплексные атрибуты** - это комплексные компоненты. Они добавляются в Конструкторе при создании/изменении формы в среднем блоке (в структуре формы). Подробное описание про отдельные компоненты см. [здесь](#).

Структура данных

Поиск по атрибутам

- Основные атрибуты (45)
- Дополнительные атрибуты (17)
- Вычисляемые атрибуты (6)
- Оптимизированные вычисл. атр. (5)
- Фильтры (4)
- Поля-агрегации (0)
- Состояния (0)

Поиск по атрибутам

- Форма
 - Фильтры
 - Расширенный список
 - deleted
 - Район / Названи
 - Район / ОКТМО
 - Муниципалитет
 - Наличие догово
 - Подтверждение
 - Источник экспор
 - Категория МНО*

Группа

- Расширенный список фильтров
- Объект инфраструктуры
- Диапазон дат
- Местоположение
- Слой мультикарты: список объектов
- Последний документ по печатной форме
- График
- Список в детальной странице
- Организация
- Редактор гео-данных
- Банк
- Маршрут на карте
- Полиморфная связь (fk)
- Полиморфная связь (gfk)



2.3. Переопределение моделей

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

Например, модель контрагента (Participant) находится в ядре Платформы. Т.е. данная модель будет присутствовать в каждом новом проекте, реализованном на Платформе. Но для каждого проекта нужны различные дополнительные функции и поля в этой модели. Если их добавить на Платформе, 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:
    @fb_dynamic_attribute("Только свои")
    def course_filter(self, queryset, value):
        if value:
            queryset = queryset.filter(category_id=value)
        return queryset

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

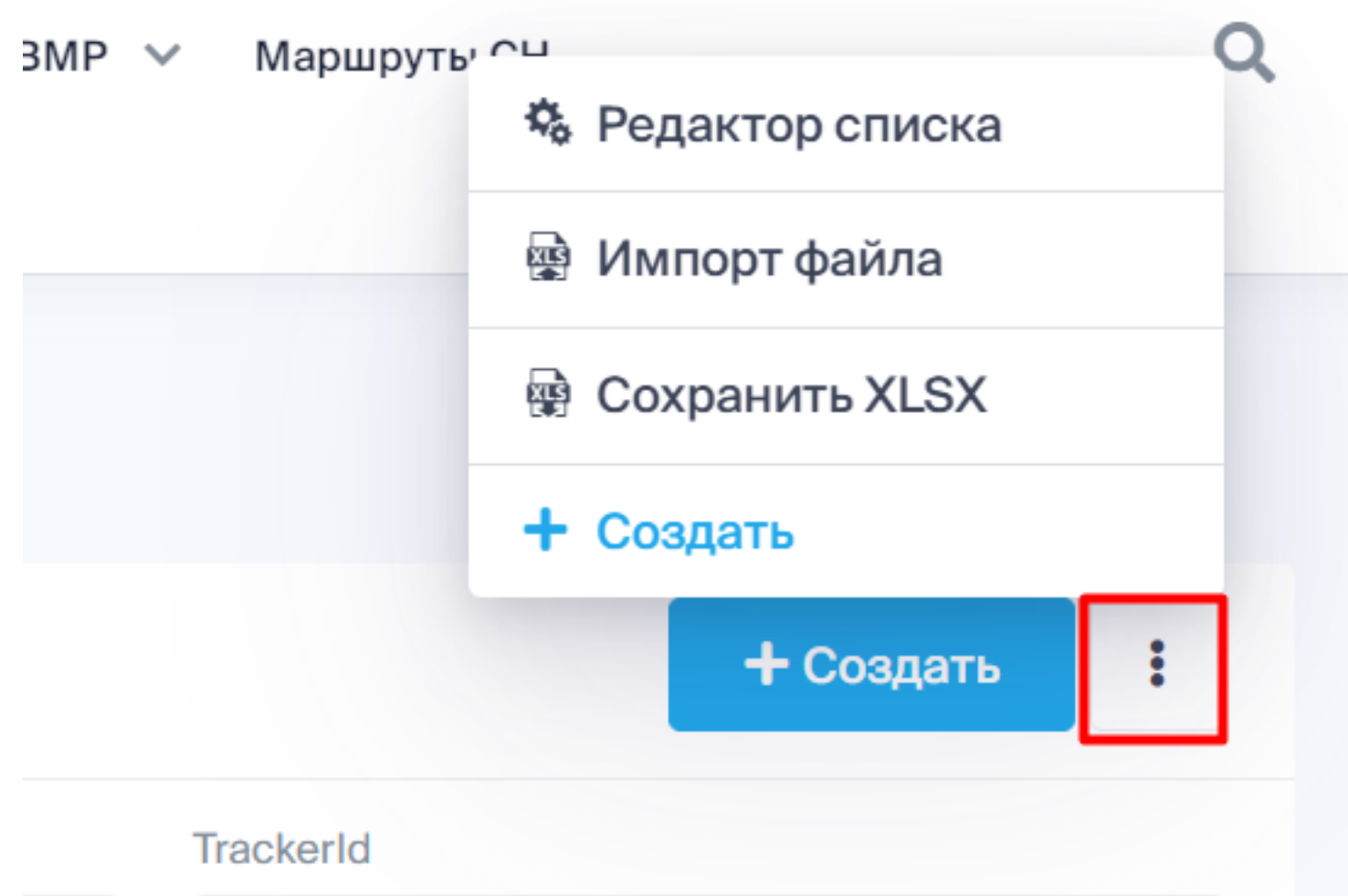
    @transition(title='Отменить формирование отчёта')
    def cancel_generation(self):
        return 'TestCourse_transition'

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).
 Подробное описание см. [здесь](#).

Переходы статусов: [Д0] Договор с оператором комплексной услуги: Общие: Черновик (draft) - С

Наименование:	Согласовать
Код:	na_soglasovani
Тип стиля:	Белая ▾
Из статуса:	Общие: Черновик (draft)
В статус:	Общие: На согласовании (negotiations)
Форма перехода:	-----
Операция перед переходом:	<div style="border: 1px solid gray; padding: 2px;"> <div style="background-color: #e0e0e0; padding: 2px;">----- ▾</div> <div style="background-color: #e0e0e0; padding: 2px;">-----</div> <div style="background-color: #007bff; color: white; padding: 2px;">Уведомить автора</div> <div style="background-color: #e0e0e0; padding: 2px;">Проверка договора на вывоз</div> <div style="background-color: #e0e0e0; padding: 2px;">Проверка договора на подпись</div> <div style="background-color: #e0e0e0; padding: 2px;">Проверка договора с отходообразователем</div> </div>
Операция после перехода:	-----

Выберите тип представления и сохраните объект, чтобы назначить операцию



3. Фронтенд



3.1. Основные фронтенд-модули Платформы

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. Репозиторий Личного кабинета - <https://git.big3.ru/analytics/personal-area>

3. Репозиторий Оперативного контроля - <https://git.big3.ru/analytics/analytics-front>

Это отдельная система оперативного контроля за очисткой контейнерных площадок, которая взаимодействует с Платформой.



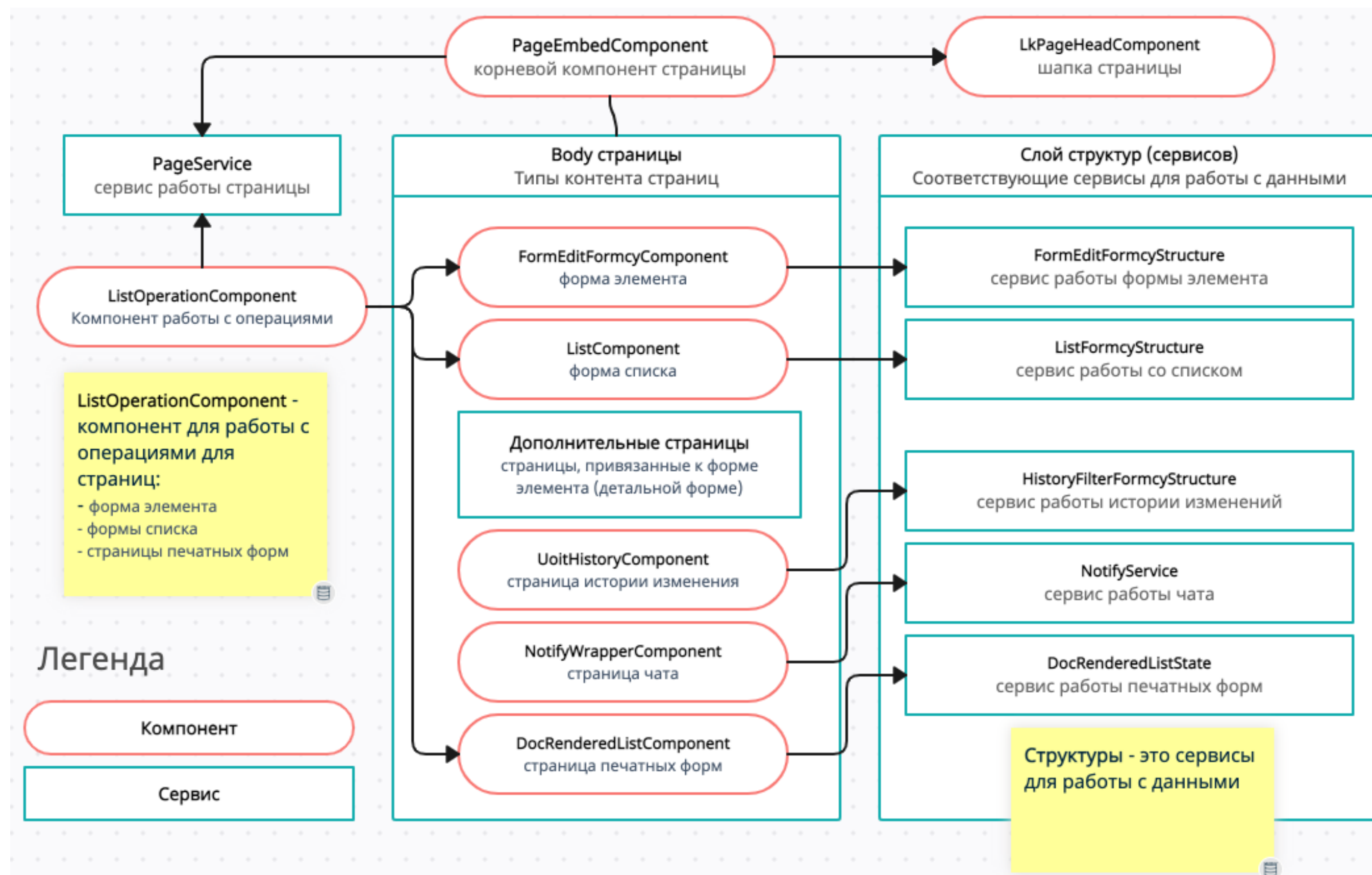
3.2. Структура страницы

✓ **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:

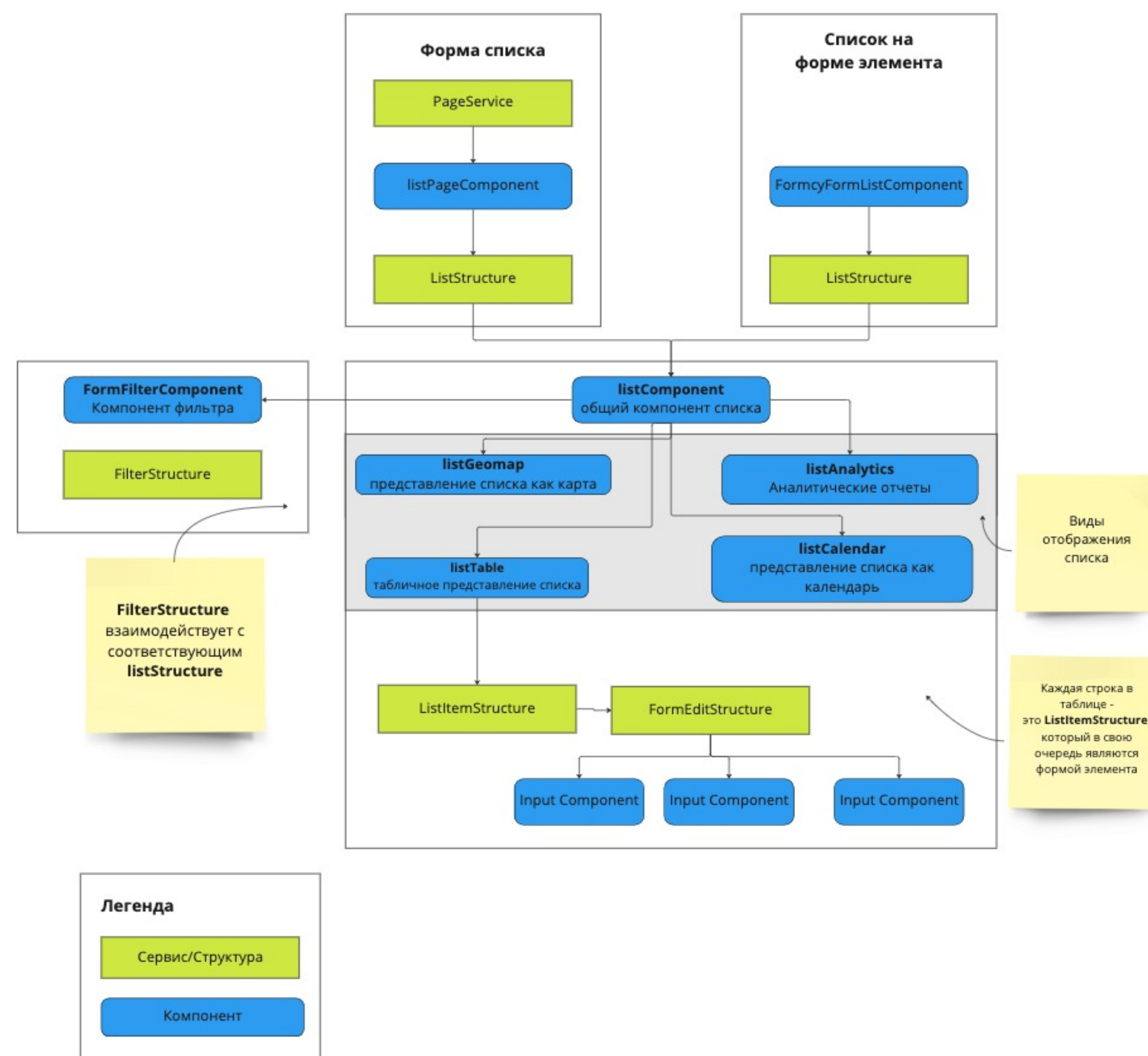
- ✓ вид формы списка (родительский компонент `listPageComponent`);
- ✓ компонент для формы элемента (родительский компонент `FormcyFormListComponent`).

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:

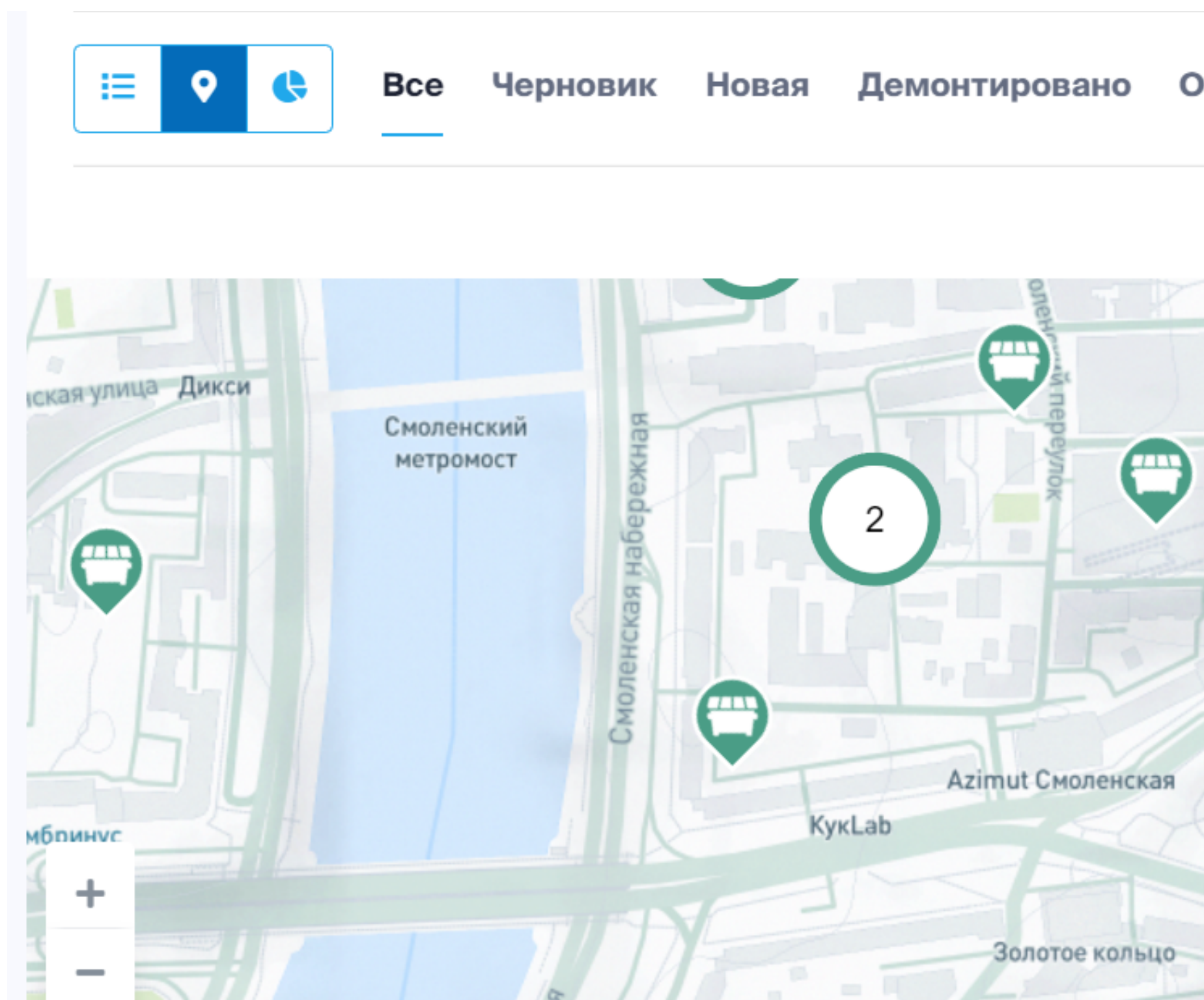
- ✓ таблица (`listTable`);
- ✓ карта (`listGeomap`);
- ✓ календарь (`listCalendar`);
- ✓ аналитика (`listAnalytics`).





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 является простейшей формой редактирования.

<input type="checkbox"/>	МЕСТОПОЛОЖЕНИЕ	ДАТА УСТАНОВКИ	ID	ФОТО
<input type="checkbox"/>	г Москва	28.04.2022 14:59	402 384	
<input type="checkbox"/>	д Полонное, ул Верхняя, д. 5	28.04.2022 14:39	400 502	
<input type="checkbox"/>	д Шалдеж, ул Придорожная, д. 7 (бункер)	28.04.2022 14:41	400 674	



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 см. [здесь](#).

Категории источников отходов

Список

Блок "head": содержит блок "all" или "partial"

Введите

Сбросить Применить

Всего 32 10

checkbox	НАЗВАНИЕ	РОДИТЕЛЬСКАЯ КАТЕГОРИЯ	Блок "second"
<input type="checkbox"/>	test	test17	+ [edit]
<input type="checkbox"/>	test17	тест 111	X [check] [close]
<input type="checkbox"/>	канкен	test16	Блок "one" [edit]
<input type="checkbox"/>	тест 111	-	[edit]
<input type="checkbox"/>	test16	test16	[edit]

Сбросить Применить

Всего 0 10

Блок "empty"

Список пуст

+ Создать



3.4. Структура формы элемента

Для отображения формы элемента используется **FormEditFormscyComponent**, 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.

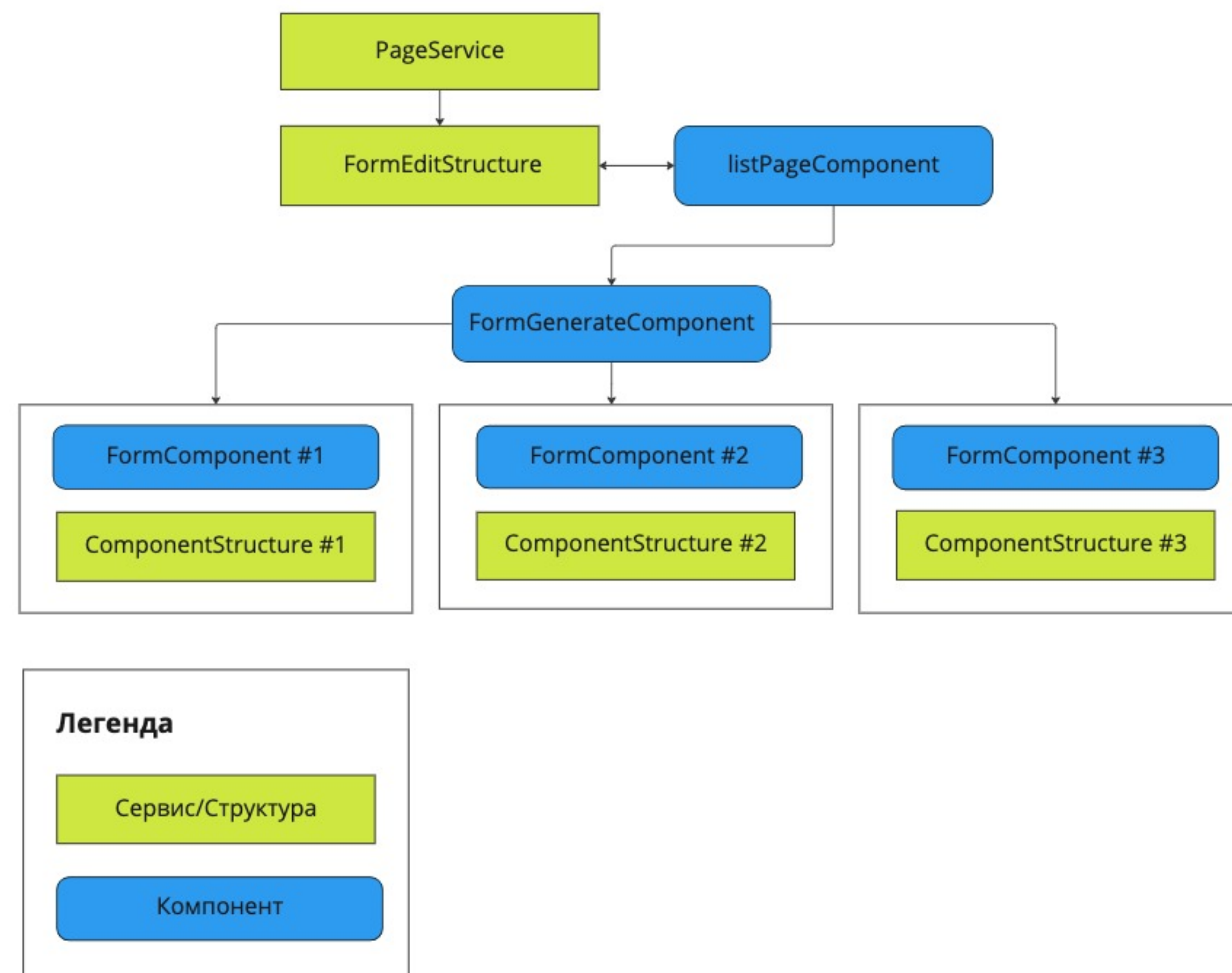
Пример:

```
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 FormEditFormscyComponent** 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**.

Подробное описание блоков кнопок см. [здесь](#).

Договоры с отходаобразователями

№ 9-8-13251 от 23.08.2022

Детальная информация | Позиции договора | Вложения | Начисления | Оплата

Редактор формы | Выставление/перевыставление начисления

Статус документа: ЧЕРНОВИК | ID: 1247 | Номер: 9-8-13251 | Региональный оператор *: ГУП "ЭКОТЕХПРОМ"

ОПЕРАТОР КОМПЛЕКСНОЙ УСЛУГИ

Оператор комплексной услуги: ООО "СПЕЦТРАНС" | Актуальный агентский договор: № 32110565588 от 01.10.2021

Подписант оператора: Самсонов Станислав Андреевич | Почта для заявок: E-Mail: disp@spt.moscow

КЛИЕНТ

Категория *: Юридические лица | Потребитель выбранной ранее категории *

Применить | Согласовано регоператором | На удаление

Блок "head": содержит кнопки блока "detail"

Блок "primary": содержит кнопку сохранения + кнопки перехода статуса



4. Создание нового компонента на Платформе



4.1. Виды компонентов

На Платформе есть **4 вида компонентов**:

- ✓ **Компонент, привязанный к данным** (наследуемый от BaseInputFormscyMeta). 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 BaseGroupFormscyMeta). Например, Group. Содержит в себе другие группы и/или массив компонентов для объединения и отображения группы полей.
- ✓ **Комплексный компонент** (наследуемый от BaseFormscyMeta/ BaseGroupFormscyMeta). 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 BaseFormscyMeta/ BaseGroupFormscyMeta). 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.

Адрес

г. Москва ул. Ленина



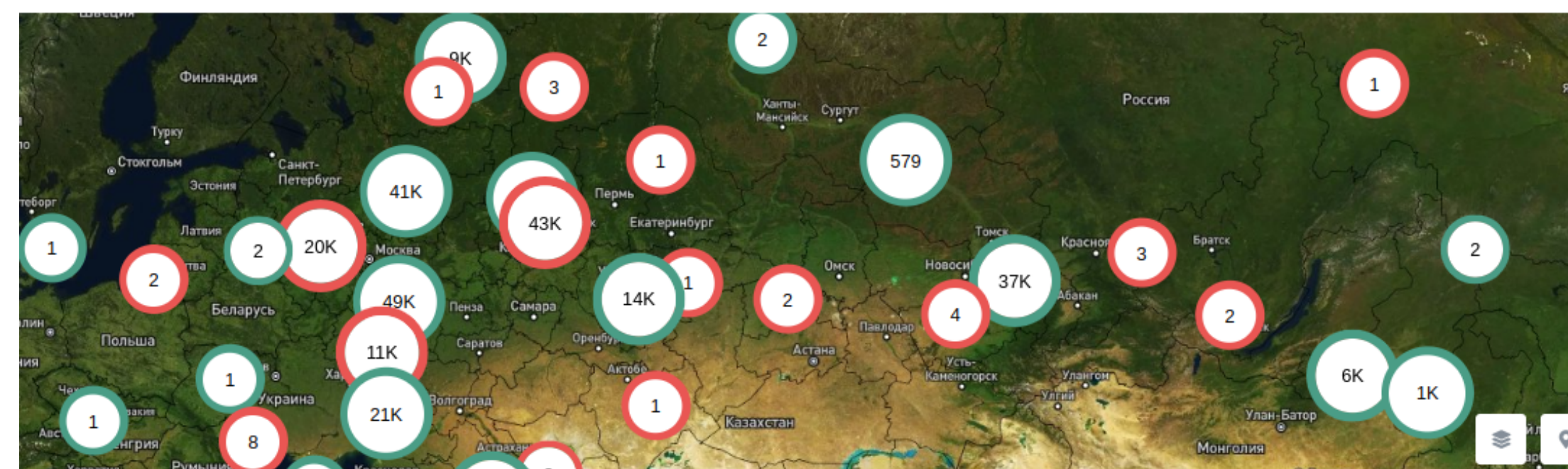
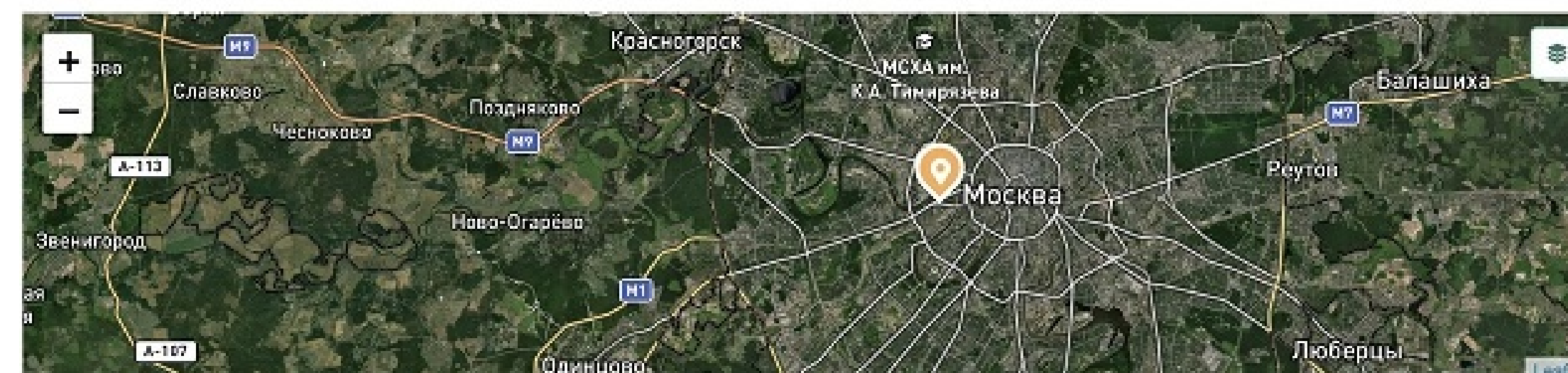
Местоположение пробы

Широта

55.749144

Долгота

37.558136

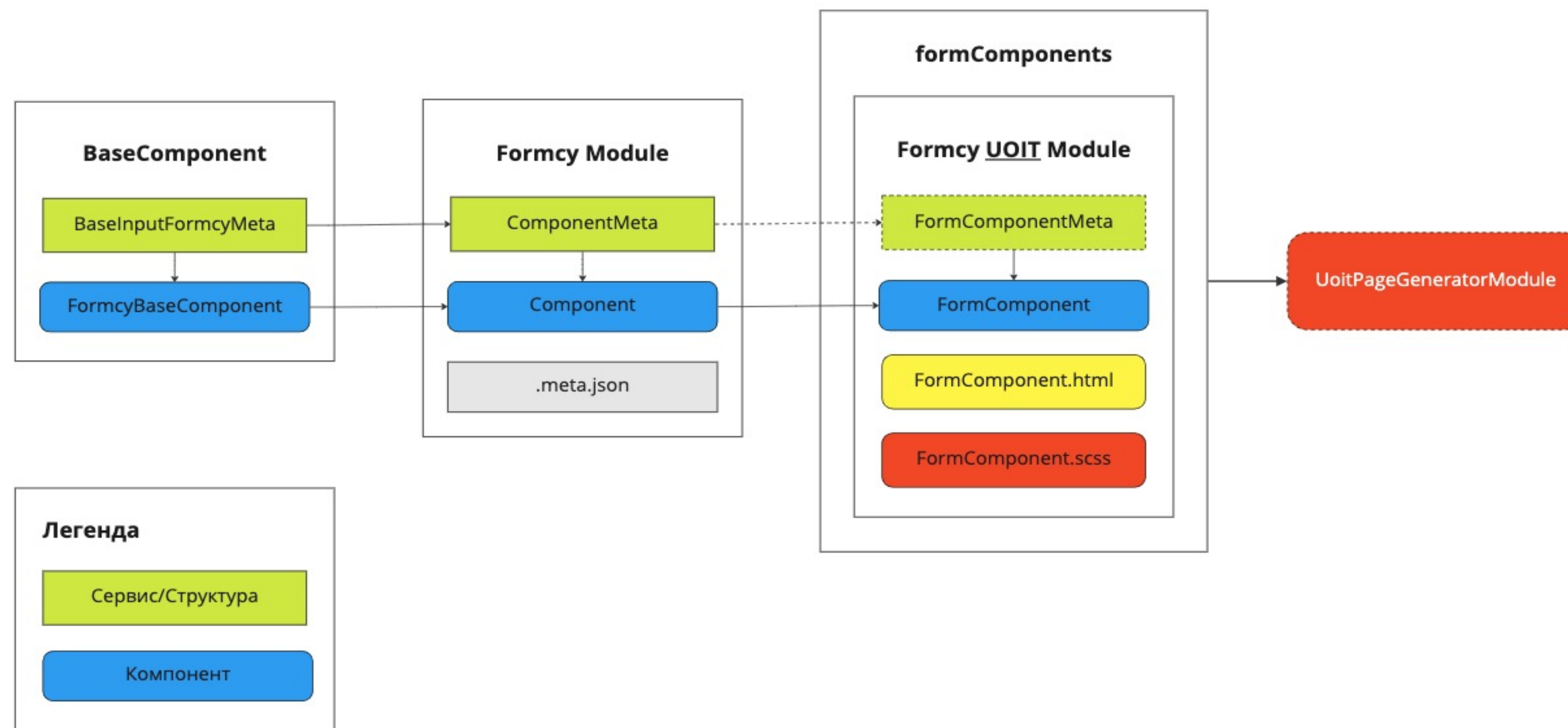




4.2. Структура компонента

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 объект 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 последующего 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!
Теперь можешь проверить свои знания по Платформе Big3

[Пройти тест](#) *

*доступ только под email ***@big3.ru