COLLABORATORS

	TITLE :	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		16 января 2018 г.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Оглавление

Список иллюстраций

Глава 1

Руководство по интеграции

Ревизия VCS: 8349

1.1 Введение

Данный документ представляет собой описание основных способов интеграции с ARTA Synergy.

1.2 Способы интеграции с ARTA Synergy

1. Синхронизация/интеграция server-side
    1. Событийная
    2. Прямая
2. Модуль, влияющий на ход исполнения маршрута
    1. Блокирующий процесс
    2. SQL-запрос
3. Приложение, работающее от имени пользователя по API
4. Ссылки на модули системы
5. WEB-модуль, встроенный в ARTA Synergy
6. Дополнительный обработчик для стандартного процесса ARTA Synergy
7. Внешние модули-компоненты (BMK)
    1. Добавление BMK
    2. События для BMK
8. Интеграция с Elasticsearch и Kibana для реализации аналитических дашбордов

1.2.1 Синхронизация/интеграция server-side

Существует два основных подхода для интеграции с Synergy:

- Прямая интеграция — интеграционные модули разрабатываются с использованием API Synergy и интегрируемых систем. Синхронизация данных между системами и координация обмена между ними остаётся за разработчиком интеграционного модуля
- Событийная интеграция — когда какая-либо из подсистем Synergy генерирует различные события, связанные с какими-либо данными. Обработчики этих событий (на стороне Synergy) при необходимо преобразовывают данные событий и передают их интегрируемой системе через какой-либо транспортный уровень

1.2.1.1 Прямая

ARTA Synergy предоставляет API для доступа к своим функциям с помощью rest сервисов. Описание методов API можно посмотреть в javadoc. Авторизация для всех методов API — `Basic HTTP`, подробнее о способах авторизации см. Способы авторизации в ARTA Synergy

1.2.1.2 Событийная

Под «событием» мы будем подразумевать сообщение о каком-либо изменении в Arta Synergy, содержащее тип события и минимально необходимые для получения связанной с событием информации либо воздействия на Synergy данные. Обработчик события (или событий) — программный модуль, читающий сообщения о событиях из `JMS Queue` или `JMS Topic` и осуществляющий, при необходимости, доступ к экземпляру Synergy, сгенерировавшему сообщение, с помощью API Synergy.

Обработчик событий является отдельным от Arta Synergy приложением, которое может работать как на том же сервере приложений, что и Arta Synergy, так и на удалённом.

Кроме этого, обработчик события может иметь собственные конфигурационные файлы, необходимые для реализации целевого назначения.

Обработчик событий может обрабатывать как конкретное событие (`например, event.registers.formdata.add`), так и класс событий (например, `event.registers.*`).

Обработка события может происходить в 3 этапа:

1. Получение события
2. Получение и преобразование необходимых обработчику данных
3. Передача сформированного пакета данных далее (опционально)

ARTA Synergy генерирует событие в случае, если для этого события настроены обработчики. Обработчики событий настраиваются в конфигурационном файле `${jboss.home}/standalone/configuration/arta/api-observation-configuration.xml`.

Сообщение, помещаемое в очередь `JMS`, представляет собой экземпляр `javax.jms.TextMessage`. Тело сообщения зависит от типа события, его описание можно посмотреть ниже среди описаний типов событий. Каждое событие содержит свойство `api_event`, указывающее на тип события, вызвавшего его (содержимое тега `<event>event.registers.formdata.add</event>` в конфигурационно файле).

Например:

```
<configuration>
    <listener>
        <queue>java:jboss/queues/Synergy/UsersQueue</queue>
        <event>event.users.*</event>
    </listener>
    <listener>
        <queue>java:jboss/queues/Synergy/RegisterCreateDocQueue</queue>
        <event>event.registers.formdata.add</event>
    </listener>
</configuration>
```

В этом примере настроены обработчики:

1. `java:jboss/queues/Synergy/UsersQueue` для всех событий класса `event.users.*`, т.е. всех событий, связанных с пользователятми: `event.users.account.change`, `event.users.formdata.change`, `event.users.account.add` и т.д.
2. `java:jboss/queues/Synergy/RegisterCreateDocQueue` для события добавления записи реестра `event.registers.formdata.add`.

Рассмотрим, например, код обработчика очереди `UsersQueue`:

```
public class UsersMessagesListener implements MessageListener {

    public void onMessage(Message message) {
        //Получаем идентификатор пользователя, для которого
        //сгенерировано событие
        String userID = ((TextMessage) message).getText();
        //Получаем тип события
        String eventType = message.getStringProperty("api_event");

        //Выполнение действие по получению дополнительных данных через API
        //и прочих операций, зависящих от условий решаемой задачи
    }
}
```

Ниже описаны типы событий, которые могут быть сгенерированы ARTA Synergy.

Для события `[event.orgstructure.department.formdata.change]` - идентификатор подразделения, для события `[event.orgstructure.position.formdata.change]` - идентификатор должности, для события `[event.users.formdata.change]` - идентификатор пользователя будет передаваться как основной параметр, остальные как свойства. Получить их можно следующим образом:

```
public void onMessage(Message message) {
    //Получение идентификатора пользователя/должности/подразделения (В зависимости от  ↩
        события на которое подписаны)
    String userID = ((TextMessage) message).getText();
    //Получаем идентификатор формы
    String formUUID = message.getStringProperty("formUUID");
}
```

1.2.1.2.1 События ARTA Synergy

1.2.1.2.1.1 События пользователей

Данные события генерируются для каждого из нижеописанных случаев изменения *данных пользовате*

- `[event.users.account.change]` Изменение данных полей *первичной карточки* пользователя, т.е. параметров его учётной записи:
  - Фамилия
  - Имя
  - Отчество
  - Логин
  - Код для показателей
  - e-mail
  - JID
  - Личная папка пользователя
- `[event.users.formdata.change]` Изменение данных *карточек пользователей* на основе *форм*, ассоциированных с ним посредством функциональности «Отдел кадров». Для данного события передаются следующие данные:
  - `userID` - `ID` пользователя
  - `formUUID` - `ID` формы карточки пользователя
  - `dataUUID` - `ID` данных по форме
- `[event.users.account.add]` Добавление новой записи учётной записи пользователя (и связанными с ней файлами по формам «отдела кадров»)
- `[event.users.account.delete]` Удаление (пометка «удалённые») учётной записи пользователя (и связанных с ней файлов по формам «отдела кадров»)
- `[event.users.contactdata.change]` Изменение «контактных данных» пользователя — изменение/до записей раздела «Контакты» профиля пользователя (модуль «Сотрудники») следующих типов:
  - Skype
  - Рабочий телефон
  - XMPP
  - Адрес
  - Мобильный телефон
  - Почта
  - Телефон

Для всех событий типа `event.users.*` передаваемые данные — `ID` пользователя Synergy.

1.2.1.2.1.2 События должностей

Данные события генерируются для каждого из нижеописанных случаев с *должностями*:

- `[event.orgstructure.position.add]` Добавление новой должности
- `[event.orgstructure.position.change]` Изменение данных должности - добавление/изменение/уда следующей информации:
  - Общее:
    * Название должности (на трех языках)
    * Код для показателей
    * Подразделение
    * Шифр
    * Необходимое количество штатных единиц
    * Тип назначения целей

    * Номер
  - Управление модулями
  - Показатели - статус активности

- `[event.orgstructure.position.formdata.change]` Изменение данных *карточки должности* на основе *формы*, ассоциированной с ней посредством функциональности «Отдел кадров». Для данного события передаются следующие данные:
  - `positionID` - `ID` должности
  - `assistantID` - `ID` заместителя, передается только при изменении данных карточки заместителя
  - `formUUID` - `ID` формы карточки должности
  - `dataUUID` - `ID` данных по форме
- `[event.orgstructure.position.delete]` Удаление должности

Для всех событий типа `event.orgstructure.position.*` передаваемые данные - `ID` должности `Synergy`.

Итерационные задачи по реализации: 0281, 0493

1.2.1.2.1.3 События подразделений

Данные события генерируются для каждого из нижеописанных случаев с *подразделениями*:

- `[event.orgstructure.department.add]` Добавление нового департамента
- `[event.orgstructure.department.change]` Изменение данных подразделения - добавление/изменен следующей информации:
  - Общее:
    * Информация о подразделении:
      · Название (на трех языках)
      · Номер
      · Код для показателей
      · Родительское подразделение (для всех узлов, кроме корневого)
      · Удаленный филиал
    * Информация о руководителе подразделения:
      · Название должности (на трех языках)
      · Тип назначения целей
      · Руководитель
      · И.О. руководителя
    * Заместители:
      · Название (на трех языках)
      · Номер
      · Пользователь
      · Подразделения, в которых данный пользователь будет выполнять обязанности заместителя
  - Управление модулями
  - Показатели - статус активности
  - Права на дела:
    * Наследовать права от родительского подразделения
    * Дело
    * Тип документа

Замечание

Ввиду особенностей реализации при сохранении подразделения отдельно сохраняется его карточка, отдельно - заместители. Таким образом, в данном случае событие `event.orgstructure.department.change` будет отправлено дважды, а при изменении заместителей через метод API `rest/api/positions/assistant/save` - единожды.

- `[event.orgstructure.department.formdata.change]` Изменение данных *карточки подразделения* на основе *формы*, ассоциированной с ней посредством функциональности «Отдел кадров». Для данного события передаются следующие данные:
  - `departmentID` - `ID` подразделения
  - `formUUID` - `ID` формы карточки подразделения
  - `dataUUID` - `ID` данных по форме
- `[event.orgstructure.department.delete]` Удаление подразделения

Для всех событий типа `event.orgstructure.department.*` передаваемые данные - `ID` подразделения `Synergy`.

Итерационные задачи по реализации: 0281, 0493

1.2.1.2.1.4 События реестров

Событие для реестра не генерируются самостоятельно и не имеют предопределенных названий. Для того, чтобы для реестра было сгенерировано событие, необходимо в процесс активации / изменения / удаления реестра добавить процесс «Событие реестра» и указать в поле «Название» его название.

Название события должно начинаться со строки `event.registries.formdata.`. Для различных событий и для различных реестров могут быть указаны разные либо одинаковые названия событий в зависимости от целей решаемой задачи.

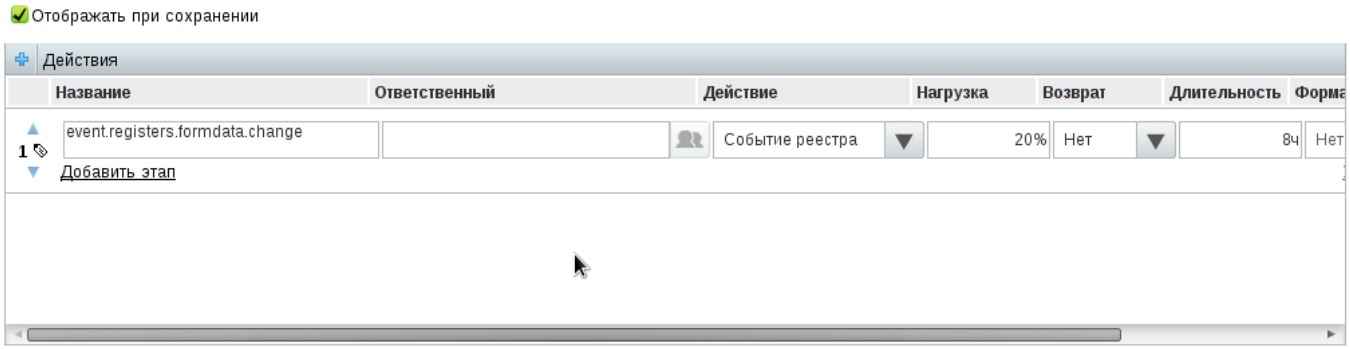


Рис. 1.1: Процесс «Событие реестра»

Подробнее о настройке и использовании реестров можно узнать в Руководстве Методолога, раздел Реестры, и Руководстве Пользователя, раздел Реестры.

1.2.1.2.1.5 События адресной книги

Объекты адресной книги (люди, организации) могут генерировать следующие события:

- [event.addressbook.contact.add] Добавление нового контакта адресной книги
- [event.addressbook.contact.change] Изменение данных контакта адресной книги: добавление / изменение / удаление записей карточки контакта. Событие генерируется при изменении:
  - данных в дополнительной карточке
  - данных в стандартной карточке:
    * «Люди»: ФИО, дата рождения, изображение, телефон, мобильный, e-mail, адрес, IM, URL, место работы, примечание, ключевые слова, поля дополнительной карточки, а также доступност контакта
    * «Организации»: название, изображение, сайт, адрес, телефон, мобильный, e-mail, поля дополнительной карточки, а также доступность контакта
- [event.addressbook.contact.change] Изменение данных контакта адресной книги
- [event.addressbook.contact.delete] Удаление контакта адресной книги

Для всех событий типа event.addressbook.contact.* передаваемые данные - это ID контакта адресной книги Synergy.

Итерационные задачи по реализации: AB1.

1.2.1.2.1.6 События работ

- [event.workflow.work.create] Создание работы
- [event.workflow.work.change] Изменение следующих параметров работы:
  - название
  - нагрузка
  - приоритет
  - сроки
  - ключевые слова
  - повторение
  - форма завершения
  - прогресс
- [event.workflow.work.completion] Фактическое завершение работы
- [event.workflow.work.expired] Работа просрочилась
- [event.workflow.work.delete] Удаление работы

> **Примечание**
>
> При добавлении/изменении/удалении комментария к работе, аналогичное событие для документов в очередь не добавляется.

Для события [event.workflow.work.expired] используется системная настройка «Интервал проверки работ на просроченность (в минутах)» (Конфигуратор -> Настройки системы -> Параметры уведомлен

Минимальный набор передаваемых данных для всех событий типа event.workflow.work.*:

- идентификатор работы

Если работа запускается по реестру, то также обязательно передаются:

- идентификатор данных по форме записи реестра (свойство с ключом `dataUUID`)
- идентификатор документа реестра (свойство с ключом `documentID`)

В случае, если генерируется любое событие по работе, порожденной мероприятием проекта, в свойствах сообщения (ключ ArrangementID) передается идентификатор этого мероприятия (`ParentID`).

Для работ по процессу «работа по форме» (вызванного как непосредственно из маршрута реестра, так и из шаблона маршрута), кроме всего прочего, также передаются данные из дополнительных полей, настроенных непосредственно в самом процессе «работа по форме».

Итерационные задачи по реализации: AS25, AS26

1.2.1.2.1.7 События по проектам

- `[event.projects.arrangement.create]` Создание мероприятия проекта
- `[event.projects.arrangement.delete]` Удаление мероприятия проекта
- `[event.projects.arrangement.restore]` Восстановление мероприятия проекта
- `[event.projects.arrangement.responsibles]` Фактическое изменение списка ответственных за мероприятие проекта

Примечание: событие изменения списка ответственных не включает случаи, когда выбранному ответственному создается работа-запрос стать ответственным и когда он отказывает в этом запросе. Таким образом, учитывается только фактическое назначение ответственного за мероприятие проекта.

- `[event.projects.arrangement.progress]` Выставление прогресса мероприятия

Минимальный набор передаваемых данных для всех событий типа `event.projects.arrangement.*` - это идентификатор мероприятия проекта.

Итерационные задачи по реализации: AS29, AS30

1.2.1.2.1.8 События по документам

- `[event.docflow.document.register]` Регистрация документа в журнале

Минимальный набор передаваемых данных в сообщении для события `[event.docflow.document.register]` - идентификатор документа.

В свойствах сообщения (ключ `registerID`) передаётся идентификатор журнала.

Итерационные задачи по реализации: AS35.

1.2.1.2.1.9 События по формам

- `[event.form.formdata.change]` Создание и сохранение данных по форме

Минимальный набор передаваемых данных в сообщении для события `[event.form.formdata.change]`:

- dataUUID - идентификатор данных по форме;

- formID - идентификатор формы;
- isNew - сохранены ли данные:
  - true - новые;
  - false - существующие.

В свойствах сообщения (ключ `dataUUID`) также передаётся идентификатор данных по форме.

1.2.1.2.1.10 События комментариев

Данные события генерируются для каждого из нижеописанных случаев:

Комментарии к работе

- `[event.comment.work.add]` Добавление нового комментария к работе
- `[event.comment.work.change]` Изменение комментария к работе
- `[event.comment.work.delete]` Удаление комментария к работе

> **Примечание**
> При добавлении/изменении/удалении комментария к работе, аналогичное событие для документов в очередь не добавляется.

Комментарии к документу

- `[event.comment.document.add]` Добавление нового комментария к документу
- `[event.comment.document.change]` Изменение комментария к документу
- `[event.comment.document.delete]` Удаление комментария к документу

Личные комментарии

- `[event.comment.personal.add]` Добавление нового личного комментария
- `[event.comment.personal.change]` Изменение личного комментария
- `[event.comment.personal.delete]` Удаление личного комментария

Комментарии к проекту/мероприятию

- `[event.comment.action.add]` Добавление нового комментария к мероприятию проекта
- `[event.comment.action.change]` Изменение комментария к мероприятию проекта
- `[event.comment.action.delete]` Удаление комментария к мероприятию проекта

Для всех событий типа `event.comment.*` передаваемые данные зависят от типа комментария и выглядят следующим образом:

- Комментарий к работе:
  - идентификатор комментария (свойство с ключом `message text`)
  - идентификатор автора комментария (свойство с ключом `userID`)
  - идентификатор документа (свойство с ключом `documentID`)
  - идентификатор работы (свойство с ключом `actionID`)

- Комментарий к документу:
  - идентификатор комментария (свойство с ключом `message text`)
  - идентификатор автора комментария (свойство с ключом `userID`)
  - идентификатор документа (свойство с ключом `documentID`)
- Личный комментарий:
  - идентификатор комментария (свойство с ключом `message text`)
  - идентификатор автора комментария (свойство с ключом `userID`)
  - идентификатор документа (свойство с ключом `documentID`)
  - идентификатор работы (свойство с ключом `actionID`)
- Комментарий к мероприятию:
  - идентификатор комментария (свойство с ключом `message text`)
  - идентификатор автора комментария (свойство с ключом `userID`)
  - идентификатор проекта (свойство с ключом `projectID`)
  - идентификатор мероприятия (свойство с ключом `actionID`)

Примечание
В случае, если объектом события является комментарий к проекту, то параметры projectID и actionID будут равны.

Итерационные задачи по реализации: 0345

1.2.1.2.1.11 Генерация произвольных событий

В ARTA Synergy имеется метод API, позволяет генерировать произвольные события.

URL метода: `rest/api/events/create`. Типа запроса: `POST`.

Метод принимает следующие обязательные параметры:

- `eventName` - название события (строка);
- `eventMsg` - произвольный `json` (строка).

В случае успешного выполнения метода сервер вернет сообщение «Событие успешно сгенерировано».

Пример:

Событие, генерируемое мобильным клиентом по координатам `GPS`:

```
eventName=event.ext.gps&eventMsg={"lat":333.333,"lon":222.222}
```

Итерационные задачи по реализации: API41.

1.2.2 Модуль, влияющий на ход исполнения маршрута

1.2.2.1 Блокирующий процесс

Блокирующий процесс предназначен для того, чтобы предоставить возможность в маршрут активации реестра вставить асинхронный вызов внешнего модуля. Основное отличие блокирующего процесса от события реестра (см. События реестров) заключается в том, что:

- при использовании блокирующего процесса маршрут реестра дожидается ответа о результате выполнения операции внешним модулем
- блокирующий процесс может завершиться положительно или отрицательно, что повлияет на дальнейшую работу маршрута (Если блокирующий процесс завершится отрицательно — процесс остановится, если положительно — то продолжит работу дальше)

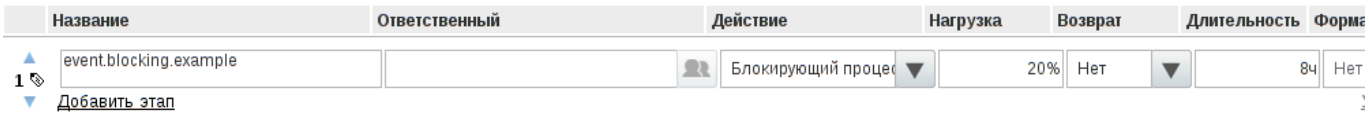
Модуль, реализующий блокирующий процесс, должен представлять собой отдельное приложение, задеплоенное на jboss в соответствии с правилами, описанными в разделе Как задеплоить интеграционное приложение.

Запускается код модуля блокирующего процесса через очередь. При старте этапа маршрута, содержащего блокирующий процесс, в очередь добавляется сообщение, которое должен обработать модуль.

Конфигурация блокирующего процесса:

Для того, чтобы добавить блокирующий процесс, необходимо выполнить следующие действия:

1. Добавить процесс с в маршрут реестра в конфигураторе:



Название процесса должно начинаться с `event.blocking.` и далее строка, характеризующая суть блокирующего процесса.

1. Создать очередь JMS для блокирующего процесса. Для этого необходимо в конфигурационный файл `jboss` (в стандартной установке это `/opt/synergy/jboss/standalone/configuration/standalone-onesynergy.xml`) в секцию `<subsystem xmlns="urn:jboss:domain:messaging:1.2">` добавить:

```
<jms-queue name="ExampleQueue">
    <entry name="java:jboss/queues/Integration/ExampleQueue"/>
    <durable>true</durable>
</jms-queue>
```

1. Связать очередь и процесс через конфигурационный файл `{$jboss.home}/standalone/configuration/arta/api-observation-configuration.xml`, добавив в него следующее:

```
<listener>
     <queue>java:jboss/queues/Integration/ExampleQueue</queue>
     <event>event.blocking.example</event>
</listener>
```

Обратите внимание, что название блокирующего процесса, указанное в маршруте в конфигураторе должно быть равно значению тега в конфигурационном файле `api-observation-configuration.xml` (в данном примере: `event.blocking.example`) и название очереди должно совпадать со значением тега `queue` конфигурационного файла `api-observation-configuration.xml` (в данном примере: `java:jboss/queues/Integration/ExampleQueue`)

Сообщение, передаваемое в очередь, является экземпляром `TextMessage`. Содержимым сообщения является объект JSON с полями:

1. dataUUID — идентификатор данных по форме записи реестра
2. executionID — идентификатор блокирующего процесса
3. documentID — идентификатор документа реестра

После того, как модуль обратится к внешней системе и выполнит необходимые действия, он должен вызвать метод API Synergy для того, чтобы возвратить результат выполнения процесса и продолжить работу маршрута. Для того, чтобы это сделать, необходимо вызвать метод API `kz.arta.synergy.samples.processes.blocking.ProcessesService#signalProcess`.

> **Примечание:**
>
> Сигнал блокирующему процессу (API - signalProcess) для его разблокировки/блокировки нужно отправлять после того, как этот процесс был запущен, то есть после того как транзакция с запуском процесса была завершена. Для этого, перед отправкой сигнала, проверяйте на наличие такого процесса в БД. В противном случае, блокирующий процесс может завершиться в транзакции, но в маршруте нет.
>
> В примере кода ниже разблокировка маршрута осуществляется в методе `onMessage`. Если время выполнения действия значительно или зависит от внешних факторов (например, доступность интегрируемой системы, или необходимость ввода пользователем данных в интегрируемой системе), то разблокировка маршрута может произойти позже, в любой другой момент времени из другого метода, а сам метод onMessage должен завершиться без ошибок, «запомнив» переданные параметры.

```
package kz.arta.synergy.samples.processes.blocking;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import org.codehaus.jackson.JsonFactory;
import org.codehaus.jackson.JsonParser;
import org.codehaus.jackson.JsonToken;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

/**
 * <p><Пример блокирующего процесса</p>
 */
public class BlockingQueueListener implements MessageListener {

    public void onMessage(Message message) {

        String dataUUID = null;
        String executionID = null;
        String documentID = null;

        if (!(message instanceof TextMessage)){
            return;
        }

        try {

            JsonFactory factory = new JsonFactory();
```

```
            JsonParser parser = factory.createJsonParser(((TextMessage) message).getText()) ↩
                ;
            JsonToken token = null;

            while ((token = parser.nextToken()) != null) {
                if (token == JsonToken.FIELD_NAME) {
                    String fieldName = parser.getText();
                    parser.nextToken();
                    String value = parser.getText();
                    if (fieldName.equals("dataUUID")){
                        dataUUID = value;
                    } else if (fieldName.equals("executionID")){
                        executionID = value;
                    } else if (fieldName.equals("documentID")){
                        documentID = value;
                    }
                }
            }

        //Выполнение каких-либо действий
        ….......

            //Разблокировка маршрута

            String address = "http://127.0.0.1:8080/Synergy";
            String login = "1";
            String password = "1";
            String signal = "got_agree";
            boolean isSuccess = false;

            try {
                URL url = new URL(address + "/rest/api/processes/signal?signal=" + signal + ↩
                     "&executionID=" + executionID + "&param1=resolution&value1=signal_is_"  ↩
                    + signal);

                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                conn.setRequestMethod("GET");
                conn.setRequestProperty("Accept", "application/json; charset=utf-8");

                String encoded = Base64.encode((login + ":" + password).getBytes());
                conn.setRequestProperty("Authorization", "Basic " + encoded);

                String output;
                StringBuffer result = new StringBuffer();

                BufferedReader br = new BufferedReader(new InputStreamReader((conn. ↩
                    getInputStream())));

                while ((output = br.readLine()) != null) {
                    result.append(output);
                }

                conn.disconnect();

                JsonFactory factory = new JsonFactory();
                JsonParser parser = factory.createJsonParser(result.toString());
                JsonToken token = null;

                while ((token = parser.nextToken()) != null) {
                    if (token == JsonToken.FIELD_NAME) {
                        String fieldName = parser.getText();
                        token = parser.nextToken();
```

```
                        if (fieldName.equals("errorCode") && parser.getText().equals("0")){
                            isSuccess = true;
                        }
                    }
                }
            } catch (Exception exc){
                logger.error(exc.getMessage(), exc);
            }

        } catch (Exception exc){
            logger.error(exc.getMessage(), exc);
        }
    }
}
```

1.2.2.2 SQL-запрос

1.2.3 Приложение, работающее от имени пользователя по API

Пример: мобильное приложение

1.2.4 Ссылки на модули системы и их внутренние элементы

Ссылки на модули и различные объекты Synergy можно использовать как внутри основного web-приложения (в этом случае предпочтительно использовать относительные ссылки, чтобы не перезагружать страницу), так и во внешних системах.

Общий вид ссылок:

http[s]://**host***[:port]*/**Application**?*param1=value1*&*param2=value2*#*param3=value3*&*param4=value*

где

- **host** - доменное имя или ip-адрес сервера Synergy
- *port* - порт
- **Application**:
  - `Synergy` - основное приложение
  - `Configurator` - Конфигуратор
  - `SynergyAdmin` - административное приложение
- *param1*, *param2* - параметры абсолютной ссылки
- *param3*, *param4* - параметры относительной ссылки

Параметры абсолютной ссылки - это, как правило:

- `locale` - локаль загружаемой системы
- `nocache` - специальный параметр, предотвращающий случайное кэширование

остальные параметры можно передавать как параметры относительной ссылки.

Ниже для краткости будем приводить образец относительной ссылки

1.2.4.1 Ссылка на модуль системы

```
#submodule=module_id
```

где `module_id`:

- `workflow` - Потоки работ
- `calendar` - Ежедневник
- `repository` - Хранилище
- `plans` - Проекты
- `pointers` - Цели и показатели
- `employees` - Сотрудники

При переходе по ссылке откроется указанный модуль.

1.2.4.1.1 Ссылка на документ и файл в нём

```
#submodule=common&file_identifier=some_file_id&action=open_document&document_identifier= ↩
    some_doc_id
```

При переходе по такой ссылке откроется указанный документ с основным файлом, а если указан `file_identifier` - то откроется документ с этим файлом.

1.2.4.1.2 Ссылка на проект и мероприятие в нем

```
#submodule=plans&action=open_action&action_identifier=some_action_id&project_identifier= ↩
    some_project_id
```

При переходе по такой ссылке откроется указанный проект, а если указан `action_identifier` - то в проекте будет выделено это мероприятие.

1.2.4.1.3 Ссылка на профиль пользователя

```
#submodule=employees&innermodule=structure&action=open_user&user_identifier=some_user_id
```

При переходе по такой ссылке будет открыт модуль «Сотрудники», а в нем - профиль указанного пользователя

1.2.4.1.4 Отключение всего пользовательского клиентского скриптинга

Если в абсолютной ссылке указать параметр `noCustomScripting`, то все пользовательские ВМК, скрипты в формах и пользовательских компонентах будут отключены. Это можно использовать для отладки пользовательских компонентов, ВМК и скриптов на форме.

1.2.5 WEB-модуль, встроенный в ARTA Synergy

Web-приложение внешнего модуля открывается в `iframe` в окне основного приложения. При этом рабочая область внешнего модуля занимает всю область страницы, кроме панели меню и панели задач:

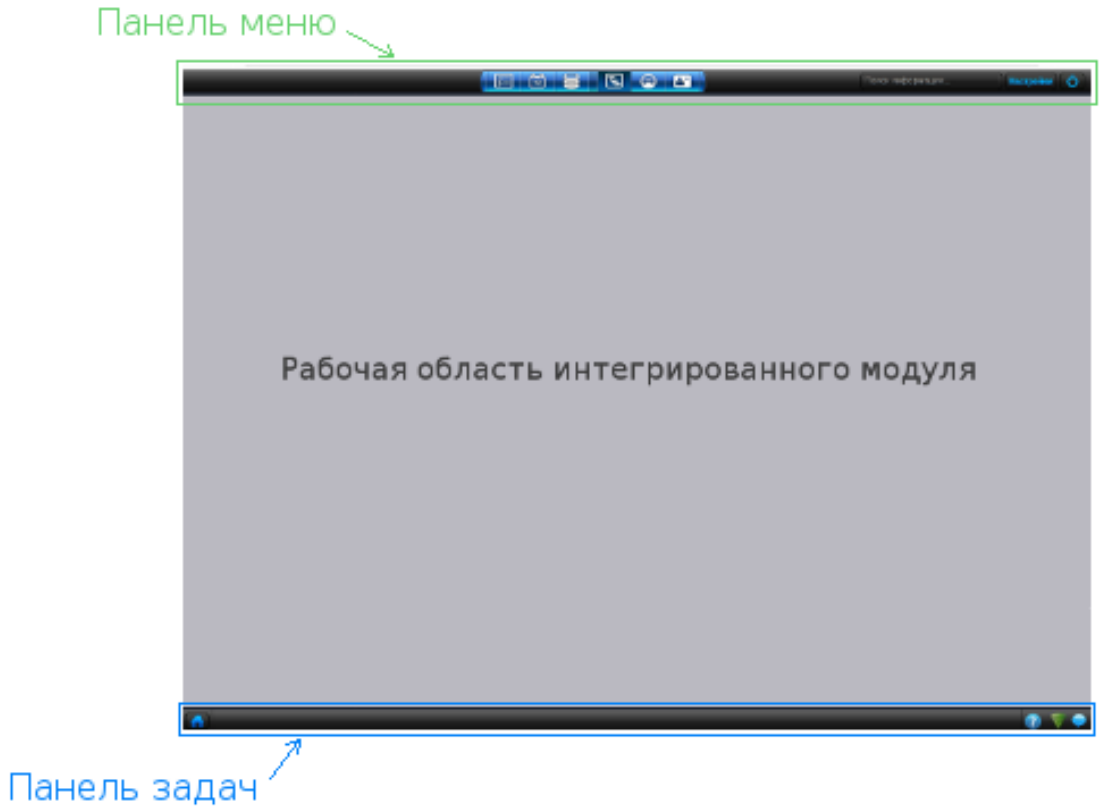


Рис. 1.2: Веб-модуль

На данный момент существует пользовательский интерфейс для добавления нового модуля. Для этого нужно перейти в Конфигуратор -> Настройки системы -> Управление модулями -> Внешние модули и нажать на кнопку «Добавить».

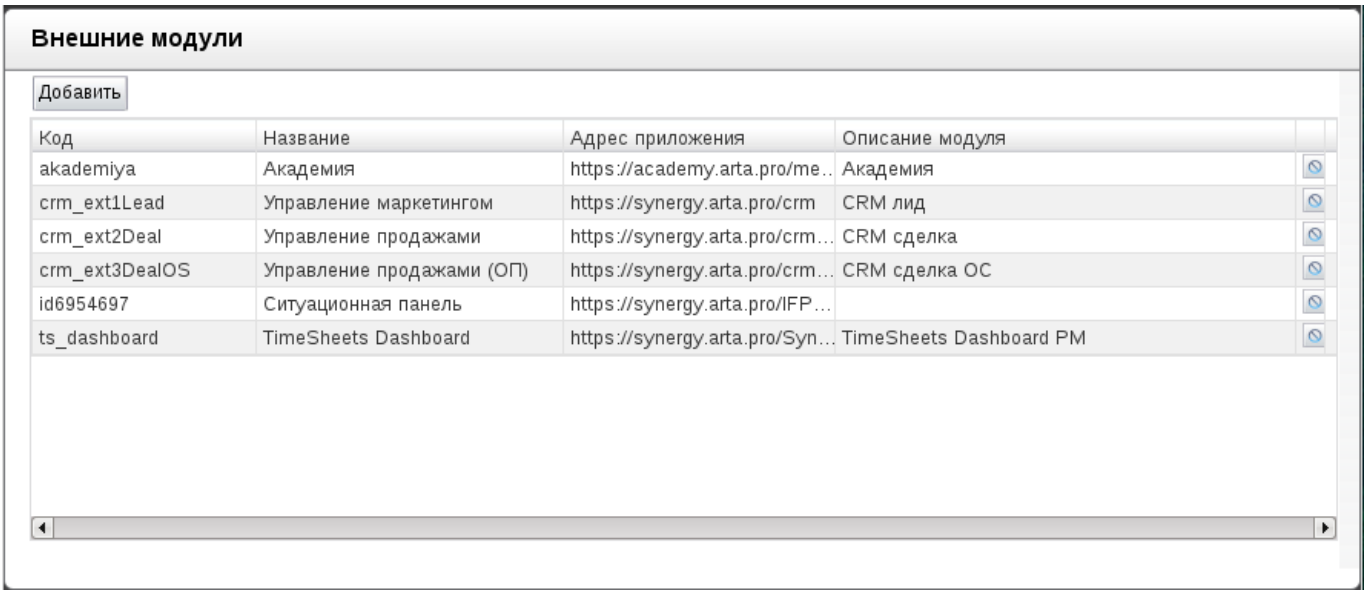


Рис. 1.3: Внешние модули

В открывшемся окне нужно заполнить следующие поля:

- «Идентификатор» - поле должно содержать уникальное значение.
- «Название на русском/казахском/английском языке» - название модуля в соответствующем интерфейсе.
- «Адрес приложения» - поле для ввода URL.
- «Описание модуля» - поле для описания данного модуля.
- «Иконка» - задает иконку модуля в пользовательской подсистеме (по умолчанию внешний модуль имеет стандартную иконку). Для того, чтобы изменить стандартную иконку, нужно кликнуть по кнопке «Выберите файл» и в диалоге выбора файла указать файл формата PNG, размер которого не превышает 28x26.

Рис. 1.4: Добавление нового внешнего модуля

Однако, если необходимо, это можно сделать с помощью SQL-запроса в таблицу `outer_modules`, вставив запись со следующими полями:

- `id` — идентификатор модуля, должен совпадать с идентификатором вашего проекта в репозитории проектов
- `nameru`, `namekz`, `nameen` — название модуля на русском, казахском и английском языках соответствен
- `url` — адрес приложения
- `description` — описание модуля
- `active` — активен ли модуль, 1/0.

Для реализации SSO приложений, ARTA Synergy при загрузке интегрированного модуля будет в строку URL добавлять три параметра:

1. `locale` — локаль авторизованного пользователя
2. `sso_hash` — hash-сумма для идентификации пользователя.
3. `host` — адрес, с которого загружено приложение Synergy

Например, если URL приложения `http://host:port/plans_module`, то запрашиваться будет URL `http://host:port/plans_module?locale=locale_value&sso_hash=sso_hash_value`

Интегрированный модуль должен будет будет получить из URL параметр `sso_hash` и запросить по API у ARTA Synergy информацию об авторизованном пользователе (идентификатор, имя). Если метод API возвращает информацию о пользователе, это подтверждает, что данный пользователь действительно авторизован с данного хоста, в данном браузере.

Далее строка `sso_hash` будет использована для Сессионной авторизации и вызова API Arta Synergy.

В ARTA Synergy реализована возможность обращения к ее модулям по относительной ссылке. Такая же возможность существует для внешних web-модулей. Переход по ссылке вида:

#submodule=outer&outerModuleID=*значение_из_таблицы_outer_modules*&*прочие_параметры_по_желанию_модуля*

активирует в Synergy заданный модуль и передаст ему заданные в url-е параметры (параметры locale, sso_hash, host также будут переданы, несмотря на то, что они отсутствуют в ссылке).

Часто возникает необходимость в этой ссылке передать ссылку на текущий документ. Для этого можно добавить в ссылку параметр, значение которого будет равно ${docID} — эта строка в web-интерфейсе просмотрщика форм Synergy будет заменена на идентификатор данного документа.

1.2.6 Дополнительный обработчик для стандартного процесса ARTA Synergy

Цель данного вида интеграции — дать возможность проверить возможность запуска стандартного процесса и, при необходимости, прервать его.

Стандартный функционал платформы ARTA Synergy дает возможность запретить отправку документов на согласование, утверждение, если количество уровней оргструктуры между отправителем и получателем превышает некоторое настроенное значение. Но в некоторых компаниях существуют более сложные правила, ограничивающие возможность отправки документов/работ. В этих случаях необходима разработка данного обработчика.

Обработчик может быть применён к процессам:

- «работа» (assignment-single)
- «согласование» (agreement-single)
- «утверждение» (approval-single)
- «ознакомление» (acquaintance-single)
- «отправка документа» (send-document)
- «общий процесс при запуске по формам» (common-process-by-form)
- «отправка документа по форме» (send-document-by-form)

Обработчик представляет собой Java-класс, реализующий интерфейс

kz.arta.synergy.integration.api.bp.StartHandlerIF.

Данный интерфейс находится в библиотеке integration-api.jar, которую можно взять здесь:

svn://scm.forge.arta.local/svnroot/synergy/trunk/build/artifacts/integration-api.jar

> *Примечание.*
>
> Начиная с итерации 66 данная библиотека входит в стандартный пакет arta-synergy-synergy. Она лежит в папке /opt/synergy/jboss/standalone/deployments/Synergy.ear/lib.

Интерфейс содержит два метода:

- makeDecision — проверяет возможно ли выполнение процесса
- getResolution — возвращает текст, который должен быть записан в ход исполнения

Более подробную информацию о полях методов можно посмотреть в javadoc к этим методам, которые доступны в integration-api.jar (библиотека содержит и скомпилированные классы, и исходный код).

Установка обработчика для процесса осуществляется с помощью конфигурационного файла ${jboss.server.config.dir}/arta/process-handlers-configuration.xml, имеющего следующий форма

```
<?xml version="1.0" encoding="UTF-8"?>
<process-handlers-configuration
  xmlns="http://www.arta.kz/xml/ns/ai"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.arta.kz/xml/ns/arta
  http://www.arta.kz/xml/ns/ai/process-handlers-configuration .xsd">
 <handlers>
     <handler>
         <process>assignment-single</process>
         <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler- ↩
             providers>
     </handler>
     <handler>
         <process>agreement-single</process>
         <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler- ↩
             providers>
     </handler>
     <handler>
         <process>approval-single</process>
         <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler- ↩
             providers>
     </handler>
     <handler>
         <process>acquaintance-single</process>
         <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler- ↩
             providers>
     </handler>
     <handler>
         <process>send-document</process>
         <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler- ↩
             providers>
     </handler>
 </handlers>
</process-handlers-configuration>
```

Обработчики выполняются последовательно до тех пор, пока метод `makeDecision` одного из них не вернет `false`, после этого процесс прерывается.

Библиотеку, содержащую обработчик необходимо скопировать в папку `/opt/synergy/jboss/standalone/deployments/Synergy.ear/lib`.

После копирования библиотеки обработчика и изменения файла `process-handlers-configuration.xml` `JBoss` необходимо перезапустить.

Примечание. Процесс `common-process-by-form` запускает процессы `agreement-single`, `approval-single`, `acquaintance-single`, `assignment-single` (подпроцессы). Поэтому, если обработчик будет запрещать выполнение подпроцесса и при этом разрешать выполнение процесса `common-process-by-form`, то подпроцессы все равно будут прерваны. Аналогично, если выполнение `common-process-by-form` разрешено, а выполнение подпроцесса запрещено, подпроцессы будут прерваны.

Пример использования

С использованием этого способа интеграции был реализован внешний модуль, ограничивающий перепоручение и отправку каких-либо работ на согласование пользователям определенных групп.

Для установки внешнего модуля из репозитория необходимо установить пакет `arta-synergy-ext-sendcontrol`.

Далее на остановленном `JBoss` в конфигурационном файле `${jboss.server.config.dir}/arta/process-handlers-configuration.xml` необходимо прописать следующие обработчики процесса:

```
<?xml version="1.0" encoding="UTF-8"?>
<process-handlers-configuration
```

```
  xmlns="http://www.arta.kz/xml/ns/ai"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.arta.kz/xml/ns/arta
  http://www.arta.kz/xml/ns/ai/process-handlers-configuration .xsd">
 <handlers>
     <handler>
         <process>assignment-single</process>
         <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler- ↩
             providers>
     </handler>
     <handler>
         <process>agreement-single</process>
         <handler-providers>kz.arta.synergy.ext.blocking.SendControlHandler</handler- ↩
             providers>
     </handler>
 </handlers>
</process-handlers-configuration>
```

Установка групп (каким группам пользователей Synergy) разрешать либо блокировать процессы осуществляется с помощью конфигурационного файла `${jboss.server.config.dir}/arta/ext/send-control.xml`. Пример настройки:

```
<?xml version="1.0" encoding="UTF-8"?>
<send-control>

    <!--
        Идентификаторы групп, членам которых разрешено отправлять
        что-либо из блоков `deny`
    -->
    <from>
        <allow>35</allow>
    </from>

    <!--
        Идентификаторы групп, членам которых могут отправлять что-либо
        только пользователи групп из блоков `allow`, если таковые есть.
        В противном случае, никто ничего этим пользователям отправить
        не сможет.
    -->
    <to>
        <deny comment="Вы не можете отправлять что-либо на согласование данному  ↩
            пользователю">111</deny>
    </to>
</send-control>
```

1.2.6.1 Исходный код SendControl:

- public link
- private link > *Примечание* > > В этом примере настроено, что перепоручение и отправка на согласование > возможны только от пользователей группы «35» пользователям группы «111». > При попытке выполнения указанных действий пользователям, не входящим в > группу «111», система отобразит указанную ошибку.

1.2.7 Внешние модули-компоненты (BMK)

Интеграция с помощью внешнего-модуля компонента предназначена для добавления или замены каких-либо элементов пользовательского web-интерфейса ARTA Synergy. Для этого необходимо

описать пользовательский компонент, который и будет служить внешним модулем-компонентом, а затем указать для него место размещения и способ вставки.

1.2.7.1 Добавление ВМК

Для настройки пользовательских компонентов методологу нужно во вкладке «Процессы» конфигурато выбрать пункт «Пользовательские компоненты».

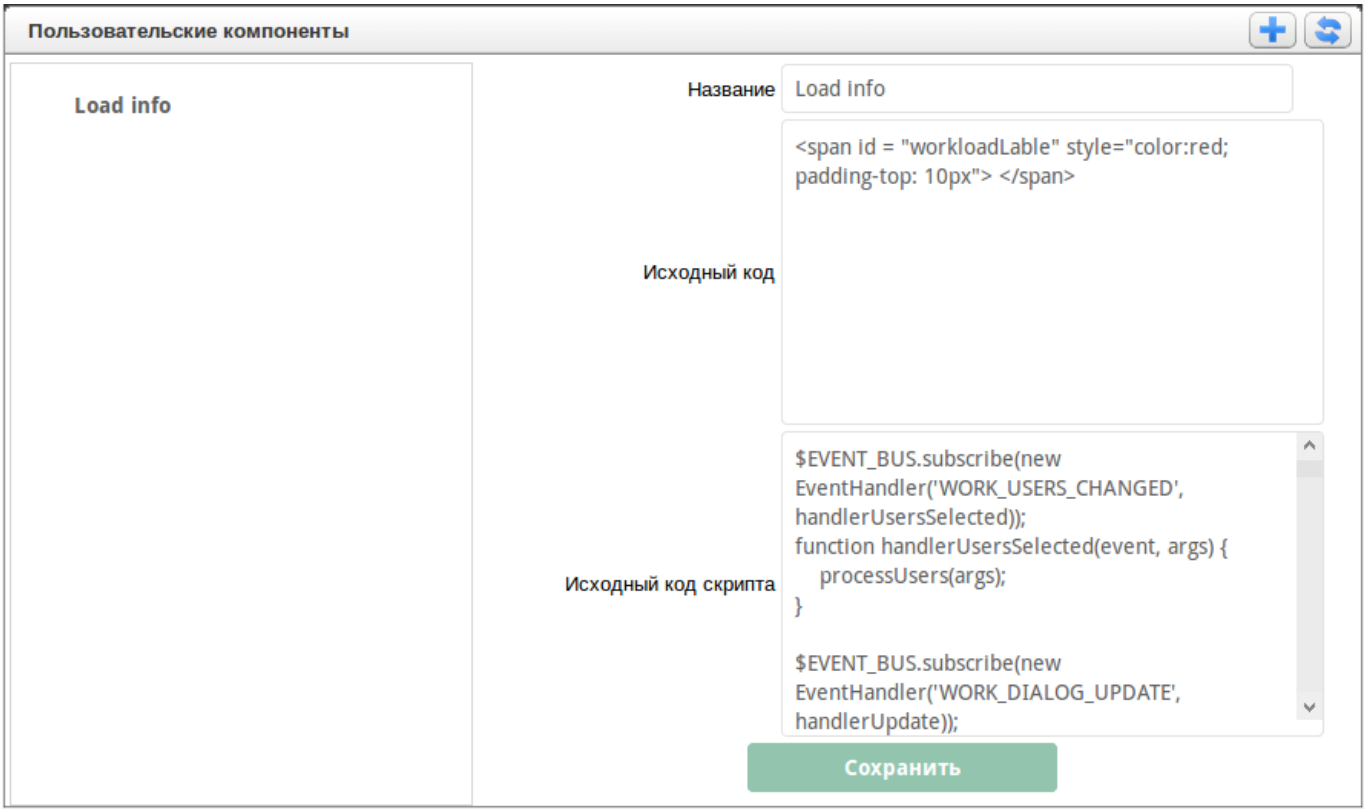


Рис. 1.5: Пользовательские компоненты

Настройка пользовательских компонентов включает в себя настройку следующих полей:

- «Название» - название пользовательского компонента, является обязательным полем;
- «Исходный код» - код компонента, написанный на `HTML/CSS`. Можно также добавить сюда и код на `javascript`, но делать этого не рекомендуется, т.к. для `js` вызывать функции отсюда будет сложнее. Для написания скрипта следует использовать поле «Исходный код скрипта»;
- «Исходный код скрипта» - код скрипта, написанный на `javascript`, с помощью которого можно настроить содержимое компонента, обратившись к существующим в системе API, описание которых можно посмотреть в javadoc и воспользовавшись возможностью обработать необходимые клиентские события. Для этого нужно вызвать функцию `$EVENT_BUS.subscribe()` и указать ей в качестве параметров необходимое событие и функцию, которую это событие обрабатывает.

  ```
  $EVENT_BUS.subscribe(new EventHandler ('WORK_USERS_CHANGED', handlerUsersSel
  ected));
  ```

  Также объект `EVENT_BUS` позволяет отписываться от событий и создавать новые. Для возможности указания новых событий необходимо обратиться к центру решений.

«Строгий режим» JavaScript:

Начиная с версии Synergy 3.14, все пользовательские скрипты выполняются с добавлением директивы `use strict`. Эта директива означает, что соответствующий ей код будет выполнятся в так называемом **«строгом режиме»**, поддерживающем стандарт `JavaScript` ECMAScript5.

Примечание:

Если код скрипта содержит конструкции, не соответствующие стандарту ES5, то они не будут выполняться. Это не является ошибкой Synergy.

Использование пользовательских компонентов как полей формы

В общем случае пользовательские компоненты не участвуют в местах, где могут использоваться какие-либо другие компоненты формы:

- отсутствует в полях формы реестра;
- отсутствует в фильтрах реестра;
- не учитывается при поиске в реестре;
- не участвует в сортировке реестра;
- отсутствует во всех видах сопоставлений;
- не учитывается во всех спец.процессах (например, в условном переходе).

Для того, чтобы пользовательский компонент присутствовал в сопоставлениях, необходимо, чтобы в его модели были реализованы методы `setAsfData` и `getAsfData` (см. Создание нового компонента), а также чтобы возвращалась структура компонента вида:

```
{
    "id": "id",
    "type": "type",
    "value": "value",
    "key": "key"
}
```

Для снятия остальных ограничений достаточно реализации метода `getAsfData` и корректности возвращаемой структуры.

Пример:

В качестве примера расмотрим реализацию пользовательского компонента «Load info», который показывает перегруженных сотрудников при создании работы (данный компонент входит в стандартну поставку Synergy).

Итерационные задачи по реализации: 0193: Компонент предупреждение о перегрузке

Исходный код содержит следующий простой `HTML` код:

```
<!-- Определим добавляемый элемент, указав для него необходимый стиль. -->
<span id = "workloadLable" style="color:red; padding-top: 10px"> </span>
<!-- За содержимое элемента будет отвечать исходный код скрипта.       -->
```

Исходный код скрипта устроен более сложным образом. Рассмотрим реализацию функции получающе данные из `api`:

```
function ajaxUserWL(ar){
    // Функция filtrAr() исключает из массива повторения пользователей.
    ar = filtreAr(ar);

    // Используем массив, который подходит под формат `json`, определенный в `api`.
    // Данный формат необходимо всегда уточнять в javadoc.
    var req = [];
    for (var i = 0; i < ar.length; i++){
        var r = {userID: ar[i].userId,
            startDate: getCurrentDateFormated(),
            finishDate: getCurrentDateFormated()};
        req.push(r);
    }

    // Вызываем функцию, которая для пользователей из массива req, вызовет
    // api, возвращающую их загруженность. Перегруженные пользователи
    // добавляются в массив res и формируют содержимое HTML тага.
    jQuery.ajax({

        // Вызов необходимого api.
        url: 'rest/person/workload/m',
        type: 'post',
        data: JSON.stringify(req),
        contentType: 'application/json',
        dataType: 'json',

        // Функция осуществляет проверку на перегруженность и помещает таких
        // сотрудников в массив res
        success: function (data) {
            res = [];
            for (var i = 0; i < ar.length; i++) {
                var user = data[ar[i].userId];
                if (user == null) {
                    continue;
                }
                for (var j = 0; j < user.length; j++) {

                    // Проверка на перегруженность.
                    if (parseFloat(user[j].value) > 100) {
                        res.push(ar[i]);
                    }
                }
            }

            // Функция определяет формат, в котором будет выводиться инофрмация.
            overloadUsers(res);
        }
    });
}
```

Итерационные задачи по реализации: 0195: API получения нагрузки

Замечание:

Полную реализацию компонента можно посмотреть в настройках пользовательских компонентов, выбрав компонент «Load info».

Замечание:

Для API, находящихся по URL rest/%api/method%, авторизация будет происходить по куки (как в браузере), поэтому его можно вызывать авторизованному в Synergy пользователю.

Для API, находящихся по другим адресам URL, необходимо настроить процесс авторизации (см. Способы авторизации). Сделать это можно например так: вызвать API по адресу `https://%логин%:%пароль%@%адрес/метода/api%`

Для того, чтобы выбрать, где использовать пользовательский компонент, методологу необходимо во вкладке «Процессы» выбрать пункт «Внешние модули-компоненты» и добавить новый внешний модуль-компонент.

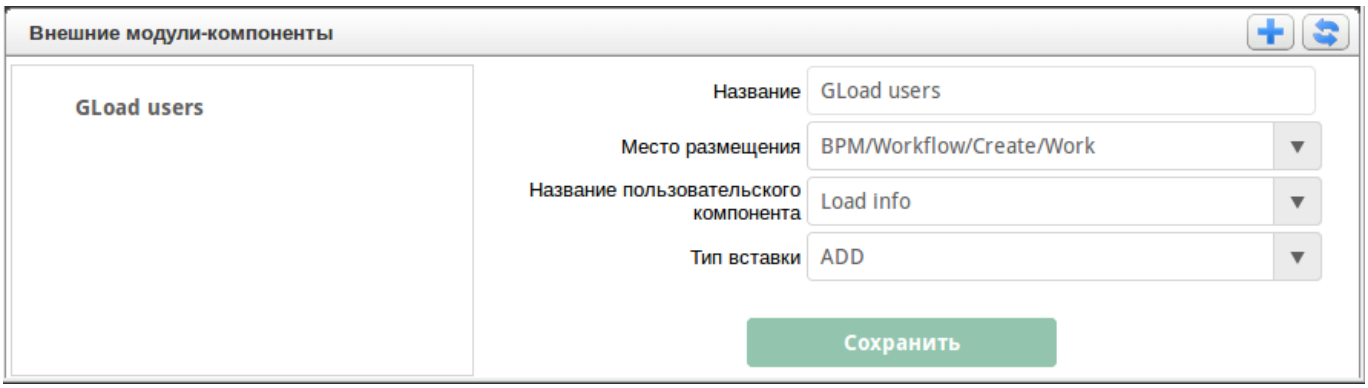


Рис. 1.6: Внешние модули-компоненты

Настройка внешнего модуля-компонента включает в себя настройку следующих полей:

- «Название» - название внешнего модуля-компонента, является обязательным полем;
- «Место размещения» - выбрать один из доступных вариантов для указания места, в котором будет находиться пользовательский компонент. На данный момент доступны следующие альтернати
  - «onLoad» - пользовательский компонент будет выводиться при загрузке приложения;
  - «Depricated» - место, используемое для вывода пользовательских компонентов в старых версиях, не рекомендуется к применению;
  - «BPM/Workflow/Create/Work» - пользовательский компонент будет выводиться в диалоговом окне создания работы;
  - «Shell/TopPanel/Right» - пользовательский компонент будет выводиться на верхней панели оболочки Synergy, левее поля «Поиск».

  Для размещения пользовательского компонента в новом месте необходимо его запрашивать у центра решений (в качестве доработки API).
- «Название пользовательского компонента» - выбрать один из доступных вариантов пользовательски компонентов, настроенных ранее;
- «Тип вставки» - выбрать один из доступных вариантов:
  - «ADD» - добавляет тег пользовательского компонента к тегу места размещения;
  - «REPLACE» - удаляются потомки тега места размещения и в него помещается пользовательский тег (замена всех потомков тега места размещения).

1.2.7.2 События для BMK

Для работы BMK в Synergy реализована отправка событий в JS EventBus.

Пример использования события из нового проигрывателя форм в JS:

```
AS.FORMS.bus.on(AS.FORMS.EVENT_TYPE.formShow, function (event, model, view) {
    var calendarModel = model.getModelWithId('cmp-1'); //Передается идентификатор компонента
    calendarModel.on(AS.FORMS.EVENT_TYPE.valueChange, function () {
        var value = calendarModel.getValue();
        console.log(value);
    });
});
```

Ниже описаны параметры для каждого события.

1.2.7.2.1 WORK_USERS_CHANGED

WORK_USERS_CHANGED - изменены исполнители работы в диалоге создания работы.

- в качестве аргумента передается список выбранных пользователей в виде массива JSON с полями: *userId, lastname, firstname, patronymic*
- *добавлено в версии 3.0*

1.2.7.2.2 WORK_DIALOG_UPDATE

WORK_DIALOG_UPDATE - обновлен диалог создания работы.

- в качестве аргумента передается список выбранных пользователей в виде массива JSON с полями: *userId, lastname, firstname, patronymic*
- *добавлено в версии 3.0*

1.2.7.2.3 SETTINGS_LOADED

SETTINGS_LOADED - загружены настройки приложения.

- в качестве аргумента передается *null*
- *добавлено в версии 3.3*

1.2.7.2.4 DEPARTMENT_ENTITY_CHANGED

DEPARTMENT_ENTITY_CHANGED - подразделение обновлено в компоненте формы.

- в качестве аргумента передается JSON с полями *id* и *values*
- values - массив JSON с полями: *value* - отображаемое имя, *key* - идентификатор пользователя
- *добавлено в версии 3.6*

1.2.7.2.5 FORM_LOADED

FORM_LOADED - компоненты формы прорисованы

- в качестве аргумента передаётся пустой JSON объект с полем dataID.
- *добавлено в версии 3.6*

Пример использования события в JS:

```
$EVENT_BUS.subscribe(new EventHandler('FORM_LOADED', handlerUsersSelected));
        function handlerUsersSelected(event, args) {
            console.log("event FORM_LOADED");
            var component = jQuery('img#image_for_portlet');
            if (component === null || component.length === 0) {
                console.log("no success");
                return;
            }
            console.log("success!!");
            console.log(component.parentNode);
            console.log(component);
            var mySpan = document.createElement("div");
            mySpan.innerHTML = "<div id='portlet_div' ><iframe frameborder = "0" height =  ↩
                "100%" id = "portlet1" src = "http://192.168.2.119:8080/widget/web/guest/ ↩
                home/-/meetups" width = "100%" > </iframe ></div > ";
            component.replaceWith(mySpan.innerHTML);
        }
            var checkAgain = function() {
                setTimeout(handlerUsersSelected, 5000);
            };
```

1.2.7.2.6 REGISTRY_SELECTED

REGISTRY_SELECTED - нажата кнопка «Создать», когда в навигаторе выделен активный реестр.

- в качестве аргумента передаётся JSON объект вида:

```
registryCode:"reg_code"
registryId:"9034810f-5f18-44b9-948a-8f78a5f1ec9d"
```

- *добавлено в версии 3.14*

Кроме того, атрибуты `registryCode` и `registryId` содержатся в элементе списка реестров, например:

```
<table cellspacing="0"
       cellpadding="0"
       synergytest="RegistryTreeElement"
       registryid="82356e07-a859-49cc-8adf-896c32725810"
       registrycode="Заявление_о_приеме_на_работу_(вариант_2,_на_двух_языках)"
       style="display: inline;"
       class="commonLabelBold">
<colgroup> <col> </colgroup>
<tbody>
    <tr>
        <td>007 Заявление о приеме на работу на период</td>
        <td style="white-space: nowrap;"></td>
    </tr>
</tbody>
</table>
```

1.2.7.2.7 USER_CHOOSER_CREATED

USER_CHOOSER_CREATED - обновлено тэговое поле выбора пользователя в диалоге.

- в качестве аргумента передаются два параметра: событие и объект, содержащий ссылку на компонент выбор пользователя (в поле args).
- *добавлено в версии 3.12 Tengri*

Пример обработчика события добавления компонента выбора пользователя, который выводит идентификатор компонента в консоль браузера:

```
function onUserChooserCreated(evt, chooser){
        var id = getUserChooserId(chooser.args);
        console.log(id);
}
```

Оперировать компонентом выбора пользователя можно следующими функциями:

- `getUserChooserId(chooserComponent)` - получение идентификатора компонента выбора пользовател
- `getUserChooserShowAll(chooserComponent)` - получение настройки «Разрешить добавлять соисполн к работам, не являющихся подчиненными»
- `setUserChooserSelectedIds(chooserComponent, arrayOfUsersIds)` - выбрать переданных пользоват
- `getUserChooserSelectedIds(chooserComponent)` - получить идентификаторы выбранных пользовате

Идентификаторы компонентов выбора пользователя:

1. Диалог создания работы создания работы:
   - исполнитель - `editWorkUserChooser`
   - ответственные - `editWorkResponsibleUserChooser`
   - автор - `editWorkAuthorUserChooser`
2. Отправить-> переслать:
   - адресаты - `sendDocumentUserChooser`
3. Отправить -> перепоручить:
   - исполнитель - `assignmentSendUserChooser`
   - ответственные - `assignmentSendResponsibleUserChooser`
4. Отправить на согласование, утверждение, ознакомление:
   - адресаты - `sendWorkUserChooser`
5. Отправить по маршруту:
   - ответственный - `editRouteSendDialog`
6. все остальные компоненты будут иметь идентификатор `userChooser`.

В каждый DOM элемент компонента выбора пользователя добавляется атрибут `userchooser` со значением идентификатора компонента выбора пользователя.

1.2.8 Скриптинг в формах: модели, свойства и методы

Основные требования и некоторые детали задачи описаны в соответствующей постановке: «Скриптинг в формах»

В данном же документе описываются все используемые для скриптинга модели, свойства и методы.

Примеры использования скриптинга см. здесь.

Используемые технологии и библиотеки:

- `jQuery`
- `Underscore` - утилиты
- `Backbone` - UI компоненты
- `Marionette` - UI компоненты
- `jQuery ui` - UI компоненты
- `math.js` - поддержка математики больших чисел
- `base64.js` - добавляет методы btoa, atob для браузеров, в которых их нет (в частности IE 9, 10)
- `XregExp.js` - поддержка более сложных регулярных выражений

Схемы работы проигрывателя:

Рис. 1.7: Схема 1, общая схема работы проигрывателя

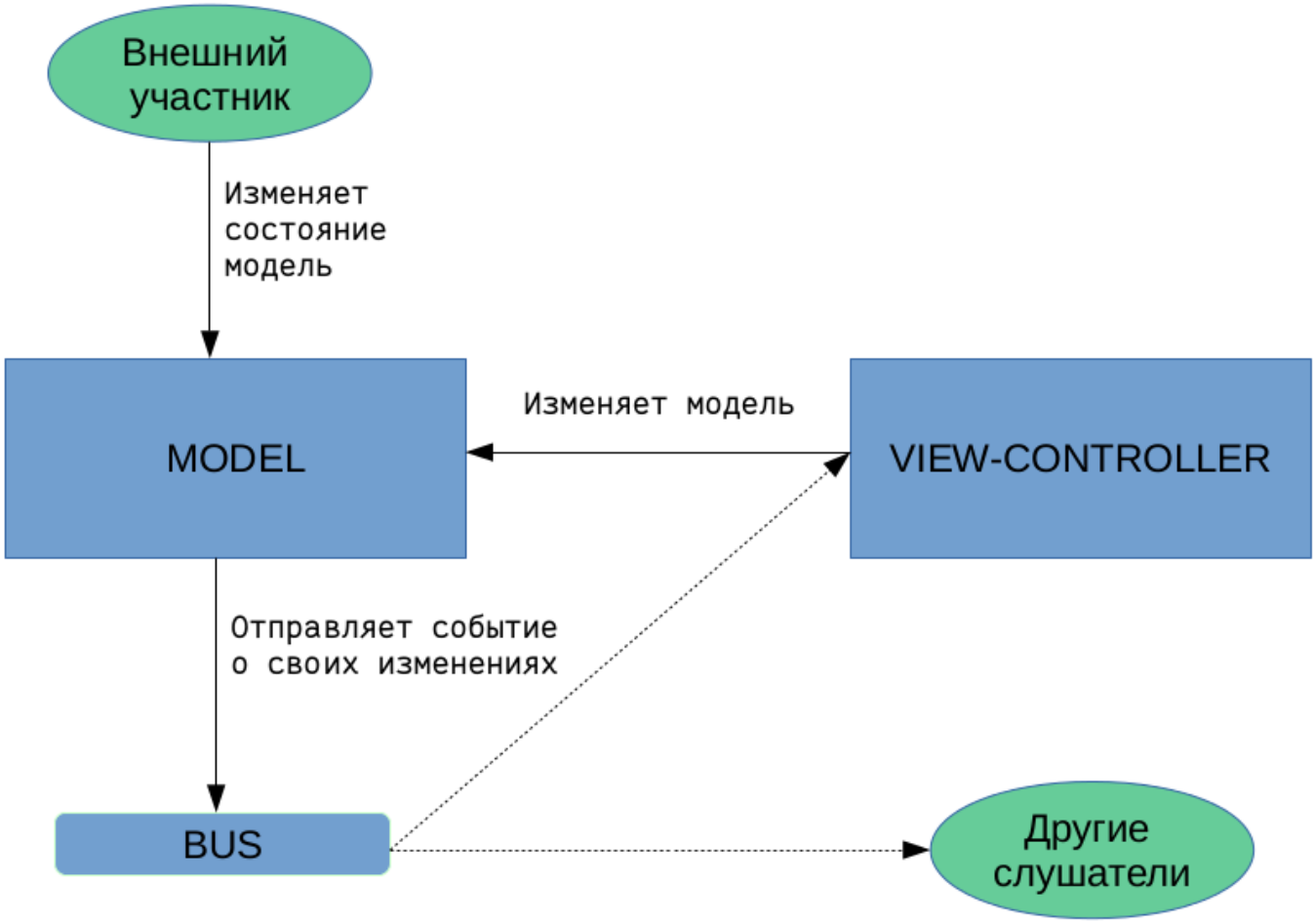


Рис. 1.8: Схема 2, процесс изменения модели

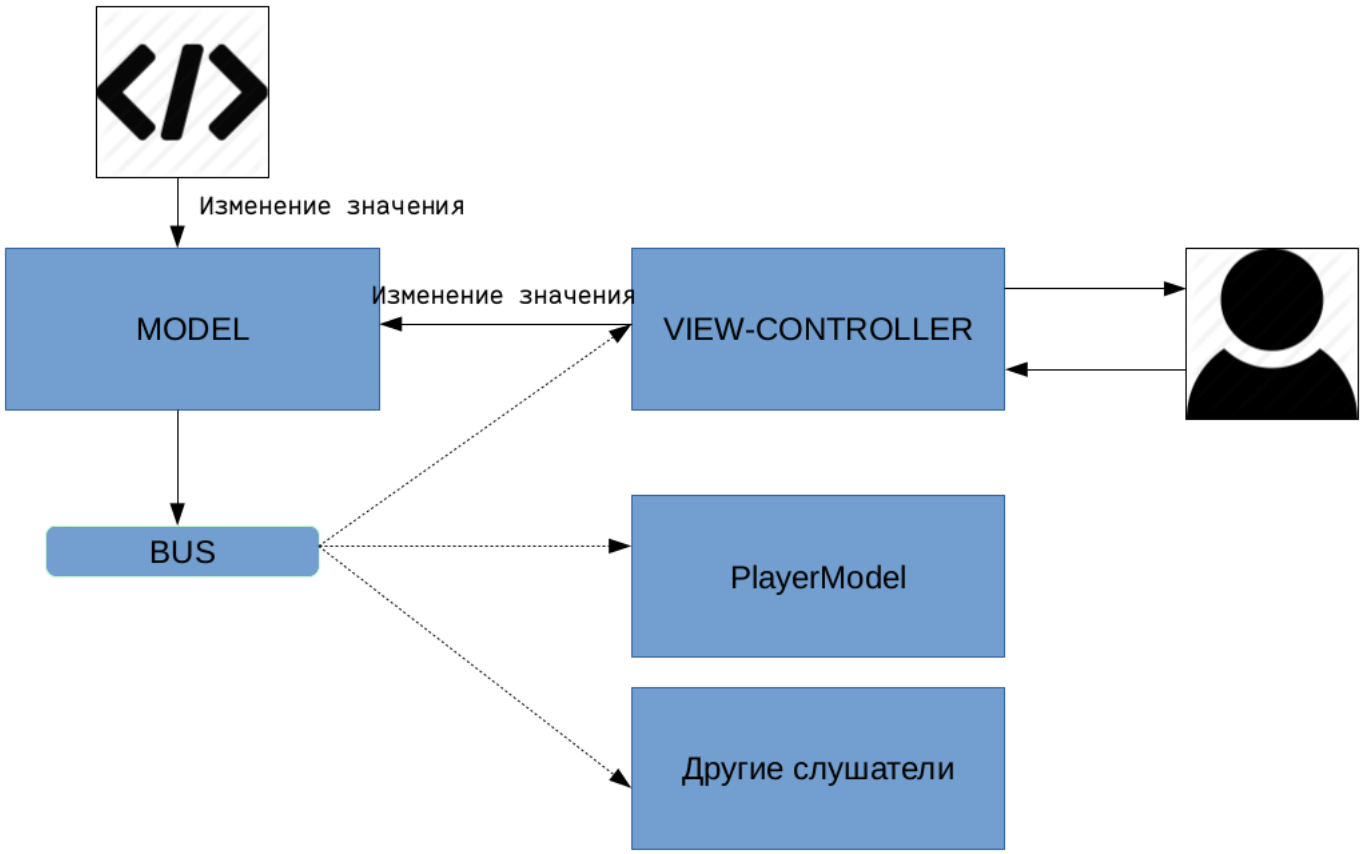


Рис. 1.9: Схема 3, процесс изменения значения компонента

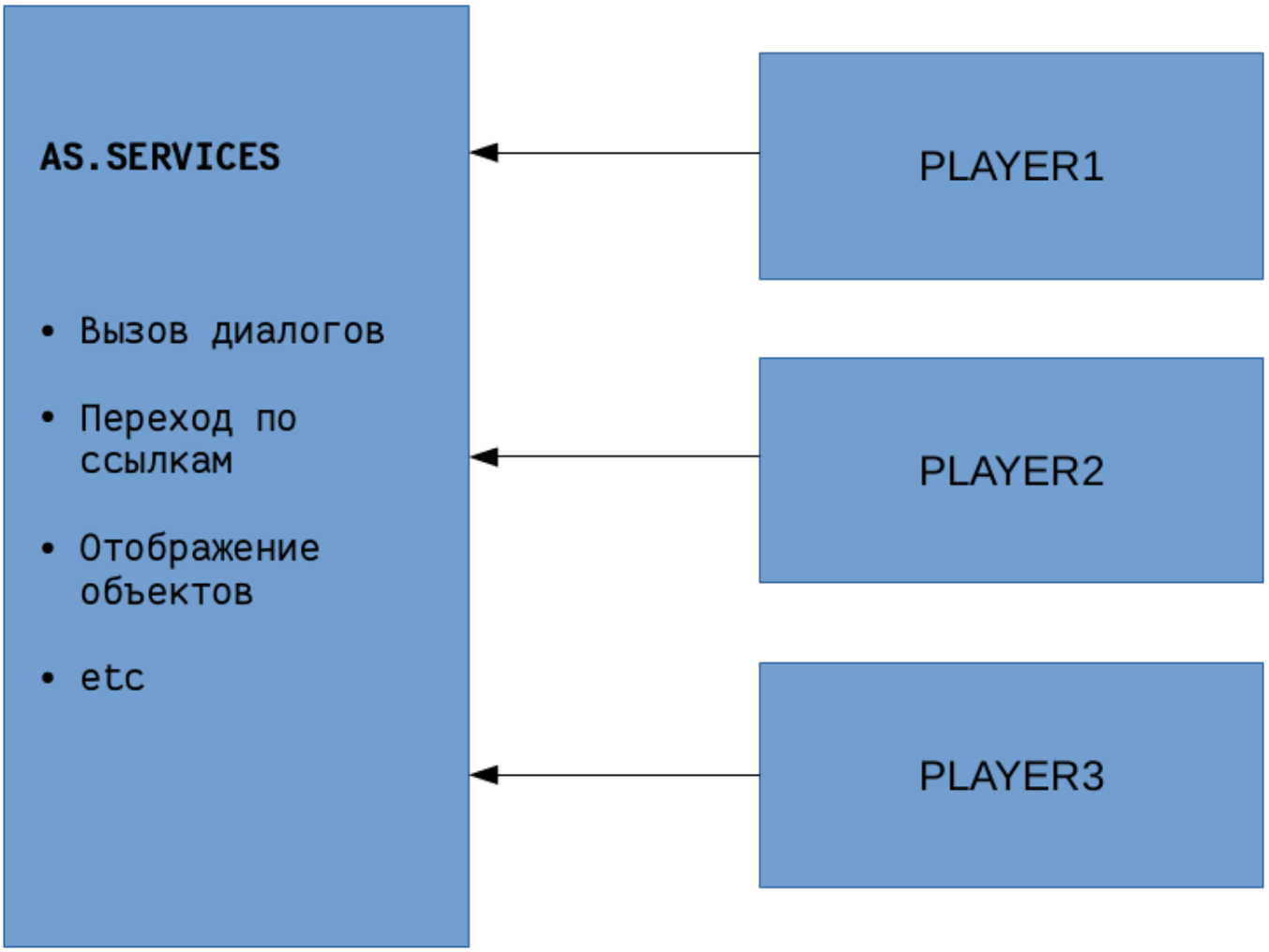


Рис. 1.10: Схема 4, взаимодействие со средой

Общедоступные объекты:

- `AS` - общее пространство имен
- `AS.FORMS` - формы
- `AS.COMPONENTS` - компоненты, которые могут быть использованы отдельно от проигрывателя форм
- `AS.SERVICES` - сервисы
- `AS.LOGGER` - логгер
- `AS.OPTIONS` - настройки
- `AS.DICTIONARIES` - кэш справочников

Типы событий:

- `formShow :'formShow'` - отображение формы (построение новой формы)
- `formDestroy :"formDestroy"` - метод destroy модели проигрывателя
- `dataLoad:'dataLoad'` - подгрузка данных

- `valueChange:'valueChange'` - изменение значения модели
- `markInvalid :'markInvalid'` - необходимость отображению подсветить невалидное значение
- `tableRowAdd:'tableRowAdd'` - добавление ряда в таблице
- `tableRowDelete:'tableRowDelete'` - удаление ряда в таблице
- `dictionaryLoad :'dictionaryLoad'` - подгрузка справочника
- `tagEdit :'tagEdit'` - редактирование тега
- `tagDelete :'tagDelete'` - удаление тега
- `loadComponentAdditionalInfo :'loadComponentAdditionalInfo'` - подгрузка дополнительной информации компонента (сейчас используется для пользовательского компонента)

Особенности реализации:

В области видимости скрипта (добавленный как к компоненту, так и пользовательскому компоненту) имеются следующие переменные:

- `model` - модель текущего компонента;
- `view` - отображение текущего компонента;
- `editable` - режим (просмотр / редактирование);
- `model.playerModel` - модель проигрывателя;
- `view.playerView` - отображение проигрывателя.

Скрипт (добавленный как к компоненту, так и пользовательскому компоненту) выполняется каждый раз при смене режима проигрывателя (просмотр - редактирование). При этом *модель* компонента остается та же, а *отображение* компонента каждый раз пересоздается. Поэтому при написании скриптов следует учесть следующее: если добавляются или переписываются методы модели, либо происходит подписывание на события другой модели, то рекомендуется использовать следующее:

```
if (!model.inited)
  {
    //манипулирование моделями
    model.inited = true;
  }
```

«Строгий режим» JavaScript:

Начиная с версии Synergy 3.14, все пользовательские скрипты выполняются с добавлением директивы `use strict`. Эта директива означает, что соответствующий ей код будет выполнятся в так называемом **«строгом режиме»**, поддерживающем стандарт `JavaScript` ECMAScript5.

> **Примечание:**
>
> Если код скрипта содержит конструкции, не соответствующие стандарту ES5, то они не будут выполняться. Это не является ошибкой Synergy.

Ссылки для быстрого перехода по документу:

- `Создание экземпляра проигрывателя и его методы`
- Базовые модели и отображения
- Модели и отображения, которые не имеют специфичных свойств или переопределения методов

- Модели и отображения, которые имеют специфичные свойства или методы
  - Модели и отображения проигрывателя
  - Модели и отображения компонентов
    * «Страница» и «Таблица»
    * «Числовое поле»
    * «Выпадающий список», «Выбор вариантов», «Переключатель вариантов»
    * «Дата/время»
    * «Файл»
    * «Ссылка»
    * «Пользователь»
    * «Должность»
    * «Подразделение»
    * «Период повторения»
    * «Ссылка на проект»
    * «Ссылка на реестр»
    * «Ссылка на адресную книгу»
    * «Ссылка на файл в хранилище»
- Методы поля ввода с тегами
- Методы AS.SERVICES
- Утилиты, полезные функции
  - Утилита при вызове методов API Synergy
  - Утилиты для работы с asfData и asfDefinition
  - Утилиты для работы с датами
  - Утилиты для работы с компонентами
- Создание нового компонента

1.2.8.1 Создание экземпляра проигрывателя и его методы

Для того, чтобы создать экземпляр проигрывателя, необходимо вызвать метод `AS.FORMS.createPlayer()`.

Методы проигрывателя:

Наименование	**Аргумент**	**Описание**
showFormData	formUid(*)	Отображение последней версии формы по ее идентификатору
showFormData	formUid(*), version	Отображение указанной версии формы по ее идентификатору
showFormData	formUid(*), version, dataUid	Отображение формы по сведениям, содержащимся в последней версии данных по форме (параметры formUid и version игнорируются)
showFormData	formUid(*), version, dataUid, dataVersion	Отображение формы по сведениям, содержащимся в указанной версии данных по форме (параметры formUid и version игнорируются)

Наименование	Аргумент	Описание
showFormByCode	formCode(*), version	Отображение формы по ее коду без создания соответствующего экземляра asfData.
dataLoaded	definition(*), data	Построение формы согласно definition и заполнение asfData значением параметра data
saveFormData	hander(*)	Сохранение данных формы, после которого выполняется функция handler с параметром asfDataUUID
destroy		Удаления экземпляра проигрывателя

Примечания:

- (*) - параметр обязателен.
- dataLoaded - вспомогательный метод, вызывающийся при выполнении методов showFormData и showFormByCode.

1.2.8.2 Базовые модели и отображения

Базовая модель.

Наименование	Тип	Описание
asfProperty	asfProperty	Определение компонента из описания формы
playerModel	AS.FORMS.PlayerModel	Модель проигрывателя

Базовые свойства модели.

Наименование	Аргумент	Описание
getErrors		Получение ошибок заполнения поля согласно настройкам asfProperty
isEmpty		Пустое ли значение
getLocale		Получение локали, настроенной в компоненте
isValid		Валидно ли текущее значение модели согласно настройкам asfProperty
fireChangeEvents		Вызов событий изменения значения формы

Наименование	Аргумент	Описание
getHTMLValue		Получение HTML-представления текстового значения поля со стилями
doSetValue	newValue	Вставить новое значение и отправить уведомление всем подписанным объектам, в т.ч. отображению
getValue		Получение значения
getTextValue		Получение текстового значения
getAsfData	Int blockNumber	Получение `asfData` с указанным номером блока (если это компонент статической таблицы, то передавать эту переменную не нужно)
setAsfData	asfData	Вставить значение `asfData`

Примечание.

`getErrors` возвращает массив ошибок. Если массив пустой, значит ошибки отсутствуют, иначе - его элементы имеют поля:

- `errorCode` - код ошибки;
- `id` - идентификатор компонента.

Возможные коды ошибок:

- `emptyValue`
- `wrongValue`
- `deletedValue`
- `valueTooHigh`
- `valueTooSmall`

Для одного и того же компонента может быть как несколько ошибок, так может и не быть ни одной.

Базовое отображение.

Наименование	Тип	Описание
model	Наследует AS.FORMS.Model	Модель
container	div	Контейнер, в котором будет отрисовываться отображение
input	input	Поле ввода для некоторых компонентов: • текстовое поле • числовое поле • многострочный текст

Наименование	Тип	Описание
playerView		Отображение проигрывателя

Базовые методы отображения.

Наименование	Аргумент	Описание
unmarkInvalid		Убрать пометку неправильно заполненного поля
markInvalid		Пометить поле как неправильно заполненное
checkValid		Проверить корректность текущего значения (если поле заполнено неверно, то вызовется метод `markInvalid`)
setEnabled	boolean enabled	Сделать доступным или недоступным для редактирования
setVisible	boolean visible	Сделать видимым или невидимым
updateValueFromModel		Обновить отображение согласно значению модели

Примечание.

Если скрывается последний компонент в строке, то вся строка принимает минимальную высоту 18 px.

1.2.8.3 Модели и отображения, которые не имеют специфичных свойств или переопределен методов

- `AS.FORMS.DocAttributeModel` - модель компонента «Свойства документа»;
- `AS.FORMS.DocAttributeView` - отображение компонента «Свойства документа»;
- `AS.FORMS.DocLinkModel` - модель компонента «Ссылка на документ»;
- `AS.FORMS.DocLinkTextView` - отображение компонента «Ссылка на документ» в режиме редактирова
- `AS.FORMS.DocLinkView` - отображение компонента «Ссылка на документ» в режиме просмотра;
- `AS.FORMS.ImageModel` - модель компонента «Изображение»;
- `AS.FORMS.ImageView` - отображение компонента «Изображение»;
- `AS.FORMS.LabelModel` - модель компонента «Неизменяемый текст»;
- `AS.FORMS.LabelView` - отображение компонента «Неизменяемый текст»;
- `AS.FORMS.SimpleModel` - модель компонентов «Многострочное поле», «Номер», «HTD-редактор», «Ход выполнения», «Лист резолюций», «Лист подписей»;
- `AS.FORMS.TextBoxModel` - модель текстового поля.

1.2.8.4 Модели и отображения, которые имеют специфичные свойства или методы

Модели и отображения проигрывателя.

- `AS.FORMS.PlayerModel` - модель проигрывателя

Свойства:

Наименование	Тип	Описание
models	Массив AS.FORMS.TableModel	Массив моделей страниц
errorDataLoad	boolean	`true` - если описание формы содержит ошибки
formId	string	Идентификатор формы
asfDataId	string	Идентификатор данных
nodeId	string	Идентификатор ноды данных
hasChanges	boolean	Имеются ли изменения
formName	string	Наименование формы
formats	string	Форматы печати
defaultPrintFormat	string	Формат печати по умолчанию
hasMobile	boolean	Имеется ли мобильное представление
hasPrintable	boolean	Имеется ли печатное представление
formCode	string	Код формы

Методы:

Наименование	Аргумент	Описание
buildModelsDefinition	definition	Построить форму по описанию
getModelWithId	cmpId, tableId, tableBlockIndex	Получение модели компонента с указанным идентификатором в указанной таблице и указанном блоке, если идентификатор таблицы не указан, то ищется модель компонента на странице или в статических таблицах

- `AS.FORMS.PlayerView` - отображение проигрывателя

Свойства:

Наименование	Тип	Описание
view	Массив AS.FORMS.TableStaticView	Массив отображений страниц

Наименование	Тип	Описание
editable	boolean	`true` - если описание формы содержит ошибки

Методы:

Наименование	Аргумент	Описание
setEditable	Boolean editable	Устанавливает режим отображения чтение/запись
getViewWithId	cmpId, tableId, tableBlockIndex	Получение отображения компонента с указанным идентификатором в указанной таблице и указанном блоке, если идентификатор таблицы не указан, то ищется модель компонента на странице или в статических таблицах
appendTo	HTML element	Отображает проигрыватель в заданном компоненте
destroy		

Модели и отображения компонентов.

- «Страница» и «Таблица»
- «Числовое поле»
- «Выпадающий список», «Выбор вариантов», «Переключатель вариантов»
- «Дата/время»
- «Файл»
- «Ссылка»
- «Пользователь»
- «Должность»
- «Подразделение»
- «Период повторения»
- «Ссылка на проект»
- «Ссылка на реестр»
- «Ссылка на адресную книгу»
- «Ссылка на файл в хранилище»

1.2.8.4.1 Модели и отображения компонентов «Страница» и «Таблица»

- `AS.FORMS.TableModel` - модель компонентов «Страница» и «Таблица»

Методы:

Наименование	Аргумент	Описание
createRow		Добавляет блок таблицы
removeRow	Номер блока	Удаляет блок таблицы
getTextValue		Возвращает форматированное значение свертки
isHaveHeader		Есть ли заголовок
isPage		Является ли страницей
isStatic		Является ли статической таблицей
isParagraph		Включена ли свертка

- `AS.FORMS.TableStaticView` - отображение статической таблицы

Методы:

Наименование	Аргумент	Описание
getRowsCount		Возвращает количество рядов таблицы
setColumnVisible	Int columnNumber boolean visible	Делает столбец таблицы видимым / невидимым
getInvisibleColumns		Возвращает список невидимых столбцов
getViewWithId	cmpId, tableId, tableBlockIndex	Получение отображения компонента с указанным идентификатором в указанной таблице и указанном блоке, если идентификатор таблицы не указан, то ищется модель компонента на странице или в статических таблицах

- `AS.FORMS.TableDynamicView` - отображение динамической таблицы

Методы:

Наименование	Аргумент	Описание
setEnabled		Разрешает либо запрещает пользователю добавлять и удалять блоки, при этом программная возможность остается
setColumnVisible	Int columnNumber boolean visible	Делает столбец таблицы видимым / невидимым

Наименование	Аргумент	Описание
getInvisibleColumns		Возвращает список невидимых столбцов
getViewWithId	cmpId, tableId, tableBlockIndex	Получение отображения компонента с указанным идентификатором в указанной таблице и указанном блоке, если идентификатор таблицы не указан, то ищется модель компонента на странице или в статических таблицах
mergeCell	• `Int Row` • `int column` • `int rowsCount` • `int columnCount`	Объединяет ячейки в блоке
splitCell	Row, column	Разъединяет ячейки
getRowsCount		Возвращает количество рядов таблицы
getBlocksCount		Возвращает количество блоков таблицы

- `AS.FORMS.TableParagraphView` - отображение динамической таблицы в свертке

1.2.8.4.2 Модели и отображения компонента «Числовое поле»

- `AS.FORMS.NumericModel` - модель компонента «Числовое поле»

Формат `asfData`:

1.2.8.4.2.1 Формат данных компонента «Числовое поле»

```
{
  "id": "cmp-qdg8xo",
  "type": "numericinput",
  "value": "2 222,00", //текстовое представление
  "key": "2222.00" //числовое представление
}
```

- `AS.FORMS.NumericInputView` - отображение компонента «Числовое поле»

1.2.8.4.3 Модели и отображения компонентов выбора

- `AS.FORMS.ComboBoxModel` - модель компонентов «Выпадающий список», «Выбор вариантов», «Переключатель вариантов»

Формат `asfData` выпадающего списка:

1.2.8.4.3.1 Формат данных компонента «Выпадающий список»

```
{
  "id": "cmp-7gj2xv",
  "type": "listbox",
  "value": "наименование выбранного элемента",
  "key": "значение выбранного элемента"
}
```

Формат `asfData` выбора вариантов:

1.2.8.4.3.2 Формат данных компонента «Выбор вариантов»

```
{
  "id": "cmp-7gj2xv",
  "type": "check",
  "value": ["значение элемента 1", "значение элемента 2", "значение элемента 3"],
  "key": ["наименование элемента 1", "наименование элемента 2", "наименование элемента 3]
}
```

Формат `asfData` переключателя вариантов:

1.2.8.4.3.3 Формат данных компонента «Переключатель вариантов»

```
{
  "id": "cmp-7gj2xv",
  "type": "radio",
  "value": "значение выбранного элемента",
  "key": "наименование выбранного элемента"
}
```

Свойства:

Наименование	**Тип**	**Описание**
listElements	[{value : «value1», key : «key1»}, ...]	Массив всех элементов компонента не зависимо от фильтра
listCurrentElements	[{value : «value1», key : «key1»}, ...]	Массив элементов компонента согласно фильтру, если таковой имеется, либо массив всех элементов

Методы:

Наименование	**Аргумент**	**Описание**
updateModelData		Обновляет данные текущих элементов компонента согласно фильтрам
getTextValues		Получение массива выбранных текстовых значений

Наименование	Аргумент	Описание
getValue		Возвращает массив выбранных идентификаторов
setValue	Идентификатор либо массив идентификаторов	Вставляет значение

1.2.8.4.4 Модели и отображения компонента «Дата/время»

- `AS.FORMS.DateModel` - модель компонента «Дата/время»

Формат `asfData`:

1.2.8.4.4.1 Формат данных компонента «Дата/время»

```
{
  "id": "cmp-pfa3r5",
  "type": "date",
  "value": "2016-маусым-15",
  "key": "2016-06-15 12:07:00"
}
```

Методы:

Наименование	Аргумент	Описание
getValue		Возвращает строковое представление даты в формате «yyyy-MM-dd HH:mm:ss», например 2016-05-25 16:16:16
setValue	Принимает строковое значение даты в формате «yyyy-MM-dd HH:mm:ss», например 2016-05-25 16:16:16	Вставляет значение

> Примечание.
>
> Значение даты хранится как строка в формате `yyyy-MM-dd HH:mm:ss`. Пустое значение хранится как `____-__-__ __:__:__` или как `null`.

- `AS.FORMS.DateView` - отображение компонента «Дата/время»

1.2.8.4.5 Модели и отображения компонента «Файл»

- `AS.FORMS.FileModel` - модель компонента «Файл»

Формат `asfData`:

1.2.8.4.5.1 Формат данных компонента «Файл»

```
{
  "id": "cmp-d84aev",
  "type": "file",
  "value": "Файл.png", //имя файла
  "key": "14064b88-633c-4748-b9c6-9fbf8c8a86e6" //идентификатор файла в хранилище
}
```

Методы:

Наименование	Аргумент	Описание
getValue		Возвращает `null` либо данные, которые имеют следующие обязательные поля: • `identifier` • `name` • `path` • `mime` • `fromStorage` (true/false)
setValue	Объект с обязательными полями: • `identifier` • `name` • `path` • `mime` • `fromStorage` (true/false)	Вставляет значение

• `AS.FORMS.FileView` - отображение компонента «Файл»

1.2.8.4.6 Модели и отображения компонента «Ссылка»

• `AS.FORMS.LinkModel` - модель компонента «Ссылка»

Формат `asfData`:

1.2.8.4.6.1 Формат данных компонента «Ссылка»

```
{
  "id": "cmp-7u4wuv",
  "type": "link",
  "value": "#submodule=common&action=open_document&document_identifier=${docID}",
  "key": "Документ; false"
}
```

> Примечание.
>
> Значение параметра `key` состоит из надписи к ссылке и через «; » (с пробелом) опцию, открывать ли ссылку в новом окне.

Методы:

Наименование	Аргумент	Описание
getValue		Возвращает описание ссылки: • value:«url» • key:«label;openInNew»
setValue	Объект с полями: • value:«url» • key:«label;openInNew» Например: • value:«http://ya.ru» • key:«OPEN YA;true»	Вставляет значение
getTextValue		Возвращает подпись ссылки
isOpenInNew		Открывать ли ссылку в новом окне
setValueFromInput	Объект с полями: • url • title • openInNew	Обновляет значение

• AS.FORMS.LinkView - отображение компонента «Ссылка»

1.2.8.4.7 Модели и отображения компонента «Пользователь»

• AS.FORMS.UserLinkModel - модель компонента «Пользователь»

Формат asfData:

1.2.8.4.7.1 Формат данных компонента «Пользователи»

Формат данных для единственного значения:

```
{
  "id": "cmp-7gj2xv",
  "type": "entity",
  "value": "Фамилия Имя Отчество",
  "key": "1ba41729-871f-4575-88d8-5a6c3de6297a",
  "formatVersion": "V1",
  "manualTags": {"identifier" : "введенный вручную тэг"}
}
```

Формат данных для множественного значения:

```
{
  "id": "cmp-7gj2xv",
  "type": "entity",
  "value": "formattedName1, formattedName2, formattedName3",
```

```
  "key": "identifier1;identifier2;identifier3",
  "formatVersion": "V1",
  "manualTags": {"identifier1" : "введенный вручную тэг"}
}
```

Примечание.

`manualTags` - это пользователи, для которых были изменены названия вручную.

`key` - это список id выбранных пользователей, разделенных «;».

Идентификаторы могут иметь приставки:

- без приставки - пользователь;
- `g` - группа (g-идентификатор_группы);
- `text` - произвольный текст (text-номер_просто_число);
- `contact` - контакт адресной книги (contact-идентификатор_контакта).

Методы:

Наименование	**Аргумент**	**Описание**

getValue		Возвращает массив данных, которые содержат обязательные поля:
		• `personID` - идентификатор пользователя
		• `personName` - название пользователя
		а также необязательные:
		• `positionName` - название должности пользователя (если существует)
		• `customFields`:
		{
		– `calendarColor` - цвет статуса
		– `calendarStatusLabel` - текст статуса (например «Введен ручную»)
		}

getSelectedIds		Возвращает массив строк - идентификаторов выбранных пользователей

Наименование	Аргумент	Описание

setValue	Объект с обязательными полями:
	• personID - идентификатор пользователя
	• personName - название пользователя
	а также необязательные:
	• positionName - название должности пользователя (если существует)
	• customFields:
	{
	– calendarColor - цвет статуса
	– calendarStatusLabel - текст статуса (например «Введен ручную»)
	}	

• AS.FORMS.UserLinkView - отображение компонента «Пользователь»

1.2.8.4.8 Модели и отображения компонента «Должность»

• AS.FORMS.PositionLinkModel - модель компонента «Должность»

Формат asfData:

1.2.8.4.8.1 Формат данных компонента «Должности»

```
{
  "id": "cmp-rgi3pr",
  "type": "entity",
  "value": "название должности",
  "key": "f1af818e-cd5d-4390-9e96-b26dd148e42d",
  "formatVersion": "V1",
  "manualTags": {"identifier" : "введенный вручную тэг"}
}
```

Примечание.

manualTags - это должности, для которых были изменены названия вручную.

Методы:

Наименование	Аргумент	Описание
getValue		Возвращает массив данных, которые содержат обязательные поля: • `elementID` - идентификатор должности • `elementName` - название должности а также необязательные: • `departmentName` - название подразделения, которому принадлежит должность • `status` - текст статуса (например, «Введен ручную») • `statusColor` - цвет статуса
getSelectedIds		Возвращает массив строк - идентификаторов выбранных должностей
setValue	Объект с обязательными полями: • `elementID` - идентификатор должности • `elementName` - название должности а также необязательные: • `departmentName` - название подразделения, которому принадлежит должность • `status` - текст статуса (например, «Введен ручную») • `statusColor` - цвет статуса	

• `AS.FORMS.PositionLinkView` - отображение компонента «Должность»

1.2.8.4.9 Модели и отображения компонента «Подразделение»

• `AS.FORMS.DepartmentLinkModel` - модель компонента «Подразделение»

Формат `asfData`:

1.2.8.4.9.1 Формат данных компонента «Подразделения»

Формат данных для единственного значения:

```
{
  "id": "cmp-7gj2xv",
```

```
  "type": "entity",
  "value": "название подразделения",
  "key": "4a76ae9b-a460-431a-9edc-c9bf2966f2fb",
  "formatVersion": "V1",
  "manualTags": {"identifier" : "введенный вручную тэг"}
}
```

Формат данных для множественного значения:

```
{
  "id": "cmp-7gj2xv",
  "type": "entity",
  "value": "name1;; name1;; name3;; name4",
  "key": "identifier1;identifier2;identifier3",
  "formatVersion": "V1",
  "manualTags": {"identifier" : "введенный вручную тэг"}
}
```

Примечание.

`manualTags` - это подразделения, для которых были изменены названия вручную.

`key` - это список id выбранных подразделений, разделенных «;».

Для множественного значения в качестве разделителя наименований `value` используется «;; » (с пробелом после точек с запятой).

Методы:

Наименование	**Аргумент**	**Описание**
getValue		Возвращает массив данных, которые содержат обязательные поля: • `departmentId` - идентификатор подразделения • `departmentName` - название подразделения а также необязательные: • `parentName` - название подразделения, которому принадлежит должность • `hasChildren` - имеются ли дочерние подразделения • `status` - текст статуса (например, «Введен ручную») • `statusColor` - цвет статуса
getSelectedIds		Возвращает массив строк - идентификаторов выбранных подразделений

Наименование	Аргумент	Описание
setValue	Объект с обязательными полями: • `departmentId` - идентификатор подразделения • `elementName` - название подразделения а также необязательные: • `parentName` - название подразделения, которому принадлежит должность • `hasChildren` - имеются ли дочерние подразделения • `status` - текст статуса (например, «Введен ручную») • `statusColor` - цвет статуса	

• `AS.FORMS.DepartmentLinkView` - отображение компонента «Подразделение»

1.2.8.4.10 Модели и отображения компонента «Период повторения»

• `AS.FORMS.RepeatPeriodModel` - модель компонента «Период повторения»

Формат `asfData`:

1.2.8.4.10.1 Формат данных компонента «Период повторения»

Формат данных для значения «Нет»:

```
{
  "id": "cmp-mqk3ik",
  "type": "repeater",
  "value": "Нет",
  "key": "0"
}
```

Формат данных для значения «По дням недели»:

```
{
  "id": "cmp-mqk3ik",
  "type": "repeater",
  "value": "По дням недели: Понедельник, Воскресенье",
  "key": "1|1.0;7.0"
}
```

> Примечание.
>
> Значение параметра `value` состоит из типа значения («По дням недели») и через «: » список полных названий дней недели, разделенных «, ».
>
> Значение параметра `key` состоит из типа значения (1 - это по дням недели) и через «|» список значений, разделенных «;», каждое значение в формате `порядковый_номер_дня_недели.0`.

Формат данных для значения «По дням месяца»:

```
{
  "id": "cmp-mqk3ik",
  "type": "repeater",
  "value": "По дням месяца: 12, 17, 23",
  "key": "2|12.0;17.0;23.0"
}
```

Примечание.

Значение параметра `value` состоит из типа значения («По дням месяца») и через «: » список дней месяца, разделенных «, ».

Значение параметра `key` состоит из типа значения (2 - это по дням месяца) и через «|» список значений, разделенных «;», каждое значение в формате `день_месяца.0`.

Формат данных для значения «Ежегодно»:

```
{
  "id": "cmp-mqk3ik",
  "type": "repeater",
  "value": "Ежегодно: 4.1, 5.11, 7.12, 9.30",
  "key": "4|1.4; 11.5; 12.7; 30.9"
}
```

Примечание.

Значение параметра `value` состоит из типа значения («Ежегодно») и через «: » список дней года, разделенных «, », каждое значение в формате `номер_месяца.номер_дня`.

Значение параметра `key` состоит из типа значения (4 - это ежегодно) и через «|» список значений, разделенных «; », каждое значение в формате `номер_дня.номер_месяца`.

Свойства:

Наименование	**Тип**	**Описание**
type	Int	Тип периода (0 - нет, 1 - по дням недели, 2 - по дням месяца, 4 - ежегодно

Методы:

Наименование	**Аргумент**	**Описание**
getValue		Возвращает массив элементов согласно типу, например [«4.1», «5.11», «7.12», «9.30»]
setValue	Строка следующего типа: 4|1.4;11.5;12.7;30.9	Вставляет значение

Наименование	Аргумент	Описание
setValueFromInput	`Int type array <String> selectedDays`	Обновляет значение модели `type` • `type` - тип периода • `selectedDays` - массив строк согласно типу, например [«4.1», «5.11», «7.12», «9.30»]
getTypeText		Возвращает текстовую расшифровку выбранного типа
getValueString	• `value` строка типа «7.12» согласно формату и типу • `type` - тип периода • `full` - полное представление (используется для режима просмотра)	Возвращает строчное представление данных

- `AS.FORMS.RepeatPeriodView` - отображение компонента «Период повторения»

1.2.8.4.11 Модели и отображения компонента «Ссылка на проект»

- `AS.FORMS.ProjectLinkModel` - модель компонента «Ссылка на проект»

Формат `asfData`:

1.2.8.4.11.1 Формат данных компонента «Ссылка на проект/портфель»

```
{
  "id": "cmp-7gj2xv",
  "type": "projectlink",
  "value": "Проект: %название%", //для портфеля "Портфель: %название%"
  "key": "00e24c3d-d3de-423a-9b86-5d501975b922",
  "valueID": "00e24c3d-d3de-423a-9b86-5d501975b922"
}
```

Методы:

Наименование	Аргумент	Описание
getValue		Возвращает идентификатор выбранного проекта или портфеля
setValue	Идентификатор проекта или портфеля, либо `null`	Вставляет значение

Наименование	Аргумент	Описание
setValueFromInput	Объект с полями: • actionID - идентификатор • name - название • elementType - тип элемента (256 - проект, 128 - портфель)	Обновляет значение компонента

• AS.FORMS.ProjectLinkView - отображение компонента «Ссылка на проект»

1.2.8.4.12 Модели и отображения компонента «Ссылка на реестр»

• AS.FORMS.RegistryLinkModel - модель компонента «Ссылка на реестр»

Формат asfData:

1.2.8.4.12.1 Формат данных компонента «Ссылка на реестр»

```
{
  "id": "cmp-6vrrkp",
  "type": "reglink",
  "value": "1111-GPON", //значащее содержимое через «-»
  "key": "a1b478c0-2d33-11e6-b327-3085a93a6496",
  "valueID": "a1b478c0-2d33-11e6-b327-3085a93a6496",
  "username": "Admin Admin Admin",
  "userID": "1"
}
```

Методы:

Наименование	Аргумент	Описание
getValue		Возвращает идентификатор выбранного документа реестра
setValue	Идентификатор документа реестра	Вставляет значение
getRegistryID		Возвращает идентификатор реестра

• AS.FORMS.RegistryLinkView - отображение компонента «Ссылка на реестр»

1.2.8.4.13 Модели и отображения компонента «Ссылка на адресную книгу»

• AS.FORMS.AddressLinkModel - модель компонента «Ссылка на адресную книгу»

Формат asfData:

1.2.8.4.13.1 Формат данных компонента «Ссылка на адресную книгу»

```
{
  "id": "cmp-7gj2xv",
  "type": "personlink",
  "value": "Фамилия Имя Отчество (Организация)", //для организации "Организация (Адрес)"
  "key": "0:idenitifier",
  "valueID": "0:identifier"
}
```

Примечание.

Цифра, предваряющая идентификатор, означает тип контакта: 0 - люди, 1 - организация.

Методы:

Наименование	**Аргумент**	**Описание**
getValue		Возвращает идентификатор выбранной записи
setValue	Идентификатор записи	Вставляет значение
setValueFromInput	Объект с полями: • `newValue` - идентификатор • `newTextValue` - подпись • `newType` - тип контакта (0 - люди, 1 - организация)	Обновляет значение компонента

- `AS.FORMS.AddressLinkView` - отображение компонента «Ссылка на адресную книгу»

1.2.8.4.14 Модели и отображения компонента «Ссылка на файл в хранилище»

- `AS.FORMS.FileLinkModel` - модель компонента «Ссылка на файл в хранилище»

Формат `asfData`:

1.2.8.4.14.1 Формат данных компонента «Ссылка на файл в Хранилище»

```
{
  "id": "cmp-5hpcpy",
  "type": "filelink",
  "value": "Файл.pdf", //название файла
  "key": "36afa2d9-cd78-4638-8132-773a69ff0c55" //идентификатор файла
}
```

Методы:

Наименование	Аргумент	Описание
getValue		Возвращает `null` либо данные, которые имеют следующие обязательные поля: • `identifier` • `name` • `icon`
setValue	Объект с обязательными полями: • `identifier` • `name` • `icon`	Вставляет значение

• `AS.FORMS.FileLinkView` - отображение компонента «Ссылка на файл в хранилище»

1.2.8.5 Методы поля ввода с тегами

События:

Наименование	Аргумент	Описание
tagEdit	• `event` - событие • `tag` - объект, подпись которого была изменена • `tagArea` - поле ввода с тегами	Возникает при изменении подписи у элемента поля ввода с тегами
tagDelete	• `event` - событие • `tag` - объект, который был удален • `tagArea` - поле ввода с тегами	Возникает при удалении элемента поля ввода с тегами

Методы:

Наименование	Аргумент	Описание
Конструктор	• `css` - объект с css свойствами, обычно передается ширина поля ввода с тегами • `width` - число, максимальная ширина поля ввода; если передается 0, то рассчитывается исходя из ширины css • `multivalue` - может ли принимать множественные значения • `editContent` - доступно ли редактирование тегов объектов • `canAddRandomText` - можно ли добавлять в качестве элементов произвольно набранный пользователем текст • `suggestHandler` - обработчик, который вызывается для отображения пользователю подсказки; если передать null, то у пользователя не будет встроенного inline поиска; в обработчик передается один параметр - введенный пользователем текст • `createRandom` - обработчик, который добавляет введенный пользователем тег; в метод передается название тега	
focus		Переводит фокус на поле ввода
addValue	• `value` - добавляемое значение • `updateSuggestionInput` перерисовать ли поле ввода с тегами	Добавляет значение
markValidity	• `valid` - валидно ли поле	Помечает поле правильно либо неправильно заполненным; неправильно заполненное поле обводится красным цветом и подсвечивается
getValues		Возвращает список текущих значений поля ввода

Наименование	Аргумент	Описание
setValues	• `values` - массив значений, может содержать произвольные объекты, у которых обязательно должно быть свойства `tagName` (подпись тега)	Устанавливает значение поля ввода с тегами
setVisible	`visible`	Определяет видимость компонента
setEditable	`editable`	Доступен ли компонент для редактирования
getWidget		Получение компонента, который можно добавить в дом

1.2.8.6 Методы AS.SERVICES

Методы:

Наименование	Аргумент	Описание
showDropDown	values - массив значений [{value : «value1», title : «title1», selected : true}, [{value : «value2», title : «title2»}, [{value : «value3», title : «title3»}] anchor - якорный компонент, к которому следует привязать popup minWidth - минимальная ширина handler - callback	Отображает выпадающий список В callback передается `value` выбранного значения
showDatePicker	value - объект типа Date anchor - якорный компонент, к которому следует привязать popup input - поле ввода даты (можно передать null), используется для передачи фокуса handler : callback	Отображает компонент выбора даты

Наименование	Аргумент	Описание

showUserChooserDialog	values, multiSelectable, isGroupSelectable, showWithoutPosition, filterPositionID, filterDepartmentID, locale, handler	Отображает диалог выбора пользователей
		В callback передается массив, содержащий объекты со следующими полями:
		• `personID` - идентификатор пользователя
		• `personName` - название пользователя
		а также необязательные:
		• `positionName` - название должности пользователя (если существует)
		• `customFields`:
		{
		– `calendarColor` - цвет статуса
		– `calendarStatusLabel` - текст статуса (например «Введен ручную»)
		}

showPositionChooserDialog	values, multiSelect, filterUserId, filterDepartmentId, showVacant, locale, handler	Отображает диалог выбора должностей В callback передается массив, содержащий объекты со следующими полями: • `elementID` - идентификатор должности • `elementName` - название должности а также необязательные: • `departmentName` - название подразделения, которому принадлежит должность • `status`: текст статуса (например, «Введен ручную») • `statusColor` - цвет статуса

Наименование	Аргумент	Описание
showDepartmentChooserDialog	values, multiSelectable, filterUserID, filterPositionID, filterDepartmentID, filterChildDepartmentID, locale, handler	Отображает диалог выбора департаментов В callback передается массив, содержащий объекты со следующими полями: • `departmentId` - идентификатор подразделения • `departmentName` - название подразделения а также необязательные: • `parentName` - название подразделения, которому принадлежит должность • `hasChildren` - имеются ли дочерние подразделения • `status` - текст статуса (например, «Введен ручную») • `statusColor` - цвет статуса
showProjectLinkDialog	handler	Отображает диалог выбора проекта/портфеля В callback передается массив, содержащий объекты со следующими полями: • `actionID` - идентификатор • `name` - название • `elementType` - тип элемента (256 - проект, 128 - портфель)
showRegisterLinkDialog	registry, handler	Отображает диалог выбора записи реестра В callback передается идентификатор выбранного документа
showWaitWindow		Отображает окно ожидания Каждая платформа реализует свое окно, внутри Synergy это реализовано, для встраивания в другой портал необходимо данный метод переопределить
hideWaitWindow		Скрывает окно ожидания Каждая платформа реализует свое окно, внутри Synergy это реализовано, для встраивания в другой портал необходимо данный метод переопределить

Наименование	Аргумент	Описание
unAuthorized	Url - URLметода API, при вызове которого появилась ошибка 401	Скрывает окно ожидания Каждая платформа реализует свое окно, внутри Synergy это реализовано, для встраивания в другой портал необходимо данный метод переопределить

1.2.8.7 Утилита при вызове методов API Synergy

Методы AS.FORMS.ApiUtils:

Наименование	Аргумент	Описание
addAuthHeader		Добавляет заголовок с параметрами авторизации
getFullUrl	UrlPart - «rest/api/asforms/form_ext?formID=»	Формирует частичный (при наличии) либо полный URL до метода API
simpleAsyncGet	UrlPart - «rest/api/asforms/form_ext?formID=» callback - куда вернуть результат dataType - тип данных data - данные errorHandler — callback в случае ошибки (при ошибке 401 вызовется метод AS.SERVICES.unAuthorized)	Выполняет GET-запрос к API Synergy Автоматически добавит заголовки
simpleAsyncPost	UrlPart - «rest/api/asforms/form_ext?formID=» callback - куда вернуть результат dataType - тип данных data - данные contentType - тип содержания errorHandler — callback в случае ошибки (при ошибке 401 вызовется метод AS.SERVICES.unAuthorized)	Выполняет GET-запрос к API Synergy Автоматически добавит заголовки

1.2.8.8 Утилиты для работы с asfData и asfDefinition

Методы AS.FORMS.ASFDataUtils:

Наименование	Аргумент	Описание
getComponentLocale	asfProperty	Получение локали, настроенной в компоненте
isReadOnly	model	Является ли компонент не редактируемым согласно настроек

Наименование	Аргумент	Описание
getBaseAsfData	• `asfProperty` - настройки компонента из описания формы • `blockNumber` • `value` • `key`	Получение базового объекта для формирования данных для сохранения в Synergy. Получив таким образом объект, можно добавить к нему собственные свойства (см. пример ниже)

1.2.8.9 Утилиты для работы с датами

Методы AS.FORMS.DateUtils:

Наименование	Аргумент	Описание
isEmpty	DateStr	Соответствует ли переданная строка пустому значению, например 201-- будет считаться пустым значением
parseDate	DateStr	Возвращает объект дата соответствующей строке, если строка не соответствует формату, то null
formatDate	Date, format	Форматирует дату согласно формату
getMonthName	number	Получение полного названия месяца по номеру
getMonthShortName	number	Получение сокращенного названия месяца по номеру
getMonthPossessiveName	number	Получение названия месяца в родительном падеже
getWeekDayName	number	Получение названия дня недели по номеру
getWeekDayShortName	number	Получение сокращенного названия дня недели по номеру

1.2.8.10 Утилиты для работы с компонентами

Методы AS.FORMS.ComponentUtils:

Наименование	Аргумент	Описание
createModel	Asfproperty, playerModel	Создание модели компонента
createView	Model, playerView, editable	Создание отображения компонента
createContainer		Создание контейнера компонента

Наименование	Аргумент	Описание
makeBus	object	Делает из любого переданного объекта шину событий, добавляет методы: • `trigger(event, [аргументы ивента])` - вызвать событие • `on(event, handler)` - подписаться на событие • `off(event, handler)` - отписаться от события
createFunction	Code - код, prefixCode - код приставка	Создание функции из строк кода

1.2.8.11 Создание нового компонента

Перед созданием нового компонента необходимо определиться со следующими вопросами:

1. Какие данные он будет хранить?
2. Какие ошибки валидации данного компонента существуют?
3. Как компонент должен выглядеть в режиме просмотра, редактирования, неправильно заполненны
   в режиме редактирования?

Ответив на эти вопросы, можно приступить к написанию компонента.

Предположим, нужно хранить в качестве значения компонента 3 поля:

- `identifier` - идентификатор выбранного значения;
- `name` - наименование;
- `info` - дополнительная информация.

Таким образом, в переменной `value` *модели* будет объект, содержащий эти 3 поля. Например:

```
value = { identifier : 1, name : «Наименование», info : «Дополнительная информация» }
```

Данный объект необходимо передавать в метод модели `setValue`, а получать в методе модели `getValue`.

Чтобы эти данные сохранялись в файл по форме и поднимались при последующем открытии, необходимо реализовать 2 метода модели:

- `getAsfData(blockNumber)`
- `setAsfData(asfData)`

Необходимо учесть, что поля сохраняемого объекта `asfData` могут иметь лишь следующий перечень наименований:

- `value` - обычно это текстовое значение компонента;
- `key` - обычно это значение компонента;
- `valueID` - дополнительный идентификатор;
- `username` - имя пользователя;

- `userID` - идентификатор пользователя;
- `values` - массив строк;
- `key` - массив строк.

Все эти поля необязательны, но объект, сохраняемый в файле по форме, может иметь только такие свойства.

Пример реализации этих методов:

```
model.getAsfData = function(blockNumber){
    if(model.getValue()) {
        /*следующий метод сформирует правильную запись для сохранения в файле по форме
           при этом:
            model.getValue().title – запишется в поле value
            model.getValue().value – запишется в поле key*/
        var result = AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber,  ↩
            model.getValue().name , model.getValue().identifier);
        /* дописываем необходимую информацию в поле valueID*/
        result.valueID = model.getValue().info ;
        return result;
    } else {
        return AS.FORMS.ASFDataUtils.getBaseAsfData(model.asfProperty, blockNumber);
    }
};

model.setAsfData = function(asfData){
    if(!asfData || !asfData.value) {
        return;
    }
    /*читаем данные из объекта из файла по форме: дополнительная информация была сохранена  ↩
       в поле valueID и теперь читаем из него*/
    var value = { value : asfData.key, title : asfData.value, info : asfData.valueID};

    model.setValue(value);
};
```

Далее необходимо определить список специальных ошибок. Например, пользователь может выбирать значениями с некоторыми идентификаторами . Для этого необходимо переопределить метод модели `getSpecialErrors`.

```
model.getSpecialErrors = function() {
    if(model.getValue()) {
        if(model.getValue().identifier == '0') {
            return {id : model.asfProperty.id, errorCode : AS.FORMS.INPUT_ERROR_TYPE. ↩
               wrongValue};
        }
    }
};
```

В данном примере проверяется, является ли значение идентификатора равным 0. Если да, то это значит, что компонент неправильно заполнен и возвращается ошибка. Synergy при этом будет показывать, что данные заполнены некорректно.

Работа с моделью теперь завершена.

Далее будем работать с *отображением*.

Предположим, что на вопрос №3 даны следующие ответы:

1. В режиме просмотра компонент должен представлять собой просто подпись.
2. В режиме редактирования - это недоступное для редактирования поле ввода и кнопка, по нажатию на которую открывается диалог выбора записи.
3. Неправильно заполненное поле должно подсвечивать красным кнопку компонента.

Необходимо инициализировать отображение, в зависимости от режима (просмотр или редактирование

В области видимости есть переменная `editable`:

- `editable =false` соответствует режиму просмотра;
- `editable =true` соответствует режиму редактирования.

Для режима просмотра достаточно иметь `div`, куда будет вставлено тестовое описание поля, и реализовать метод `updateValueFromModel`.

Пример:

```
var textView = jQuery('<div>', {class : "asf-label"});
view.container.append(textView);
view.updateValueFromModel = function(){
    textView.html(model.getValue().name);
};
```

Создается `div`, который добавляется в контейнер компонента. При любом изменении модели автоматически вызовется метод `updateValueFromModel` и значение изменится.

Для режима редактирования необходимо создать `div`, кнопку, а также реализовать методы `upd ateValueFromModel`, `markInvalid`, `unmarkInvalid`.

Пример:

```
/**
 * инициализируем внешний вид компонента, див и кнопку
 */
var inputView = jQuery('<div>', {class : "asf-label", style : "width:calc(100% - 30px)"});
var button = jQuery('<button>', {class : "asf-browseButton"});
view.container.append(inputView);
view.container.append(button);
/**
 * обновляем значение отображения в зависимости от значения модели
 */
view.updateValueFromModel = function(){
    inputView.html(model.getValue().name);
};
/**
 * метод помечает поле как неправильно заполненное
 */
view.markInvalid = function(){
   button.css("background-color", "#aa3344");
};
/**
 * метод убирает пометку неправильно заполненного поля
 */
view.unmarkInvalid = function(){
    button.css("background-color", "");
};
/**
 * вызываем какой-нибудь диалог выбора пользователя
```

```
 */
button.onclick(function(){
    var value = {identifier : "1", name : "name "+math.rand(), info : "additional info"};  ↩
          // тут прописывается какая-нибудь логика получения этого значения
    model.setValue(value);
});
```

1.3 Установка и применение JavaScript интерпретатора

1.3.1 Введение

JavaScript интерпретатор (далее просто «интерпретатор») - внешний модуль, предоставляющий возможность написать server-side скрипт на JavaScript с использованием объектов платформы. На данный момент использовать интерпретатор можно для обработки в блокирующем процессе. Поддерживаемые объекты: формы, личные карточки. С помощью интерпретатора есть возможность решать такие задачи, как арифметические действия с числовыми полями, с датами, производить необходимые расчеты в динамических таблиц и т.д.

Данная документация актуальна для версии 2.63 и выше.

1.3.2 Автоматический способ установки

В данном разделе описывается автоматический способ установки интерпретатора используя средства управления программными пакетами операционных систем, базирующихся на GNU/Debian Linux. В случае возникновения проблем во время установки пакета Вы можете воспользоваться устаревшим ручным способом.

1. Установите пакет *arta-synergy-interpreter*:

   ```
   aptitude install arta-synergy-interpreter
   ```

   В процессе установки будут запрошены:

   - пароль для доступа к MySQL, т.к. установка требует модификации этой базы данных;
   - логин и пароль пользователя, который будет использован для создания отчетов об ошибках скриптов (по умолчанию используется «1»).

1. После установки и перезагрузите JBoss.

Интерпретатор будет доступен по адресу:

```
http://адрес_сервера:http_порт_сервера/interpreter/
```

1.3.3 Ручной способ установки

1. Создать базу данных *mysql* (с кодировкой UTF-8):

   ```
   mysql -uroot -proot
   CREATE DATABASE interpreter CHARACTER SET utf8 ;
   use interpreter
   ```

2. Выполнить в базе данных скрипт:

```
create table INTERPRETER_PROCESS (ID BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
NAME VARCHAR(512),
DESCRIPTION VARCHAR(2048),
CODE VARCHAR(10000),
LOGIN VARCHAR(512),
PASSWORD VARCHAR(512),
AUTH_KEY VARCHAR(2048),
DEFAULT_MESSAGE VARCHAR(512),
PRIMARY KEY(ID));
```

3. Настроить на нее datasource сервера приложений с JNDI-именем `java:jboss/datasources/int-ds`:

    ```
    nano /opt/synergy/jboss/standalone/configuration/standalone-onesynergy.xml
    ```

    В секцию `<subsystem xmlns="urn:jboss:domain:datasources:1.1">` подсекции `<datasources>` добавить:

```
<xa-datasource jndi-name="java:jboss/datasources/int-ds" pool-name="interpreter"  ↩
    enabled="true" use-ccm="false">
            <xa-datasource-property name="URL">
            jdbc:mysql://127.0.0.1:3306/interpreter?useUnicode=true&amp; ↩
                characterEncoding=utf8
            </xa-datasource-property>
            <driver>com.mysql</driver>
            <xa-pool>
                <min-pool-size>20</min-pool-size>
                <max-pool-size>200</max-pool-size>
                <is-same-rm-override>false</is-same-rm-override>
                <interleaving>false</interleaving>
                <pad-xid>false</pad-xid>
                <wrap-xa-resource>false</wrap-xa-resource>
            </xa-pool>
            <security>
                <user-name>root</user-name>
                <password>root</password>
            </security>
            <validation>
            <validate-on-match>false</validate-on-match>
                <background-validation>false</background-validation>
            </validation>
            <statement>
                <share-prepared-statements>false</share-prepared-statements>
            </statement>
</xa-datasource>
```

4. Создать на сервере приложений очередь с JNDI-именем `java:jboss/queues/Integration/InterpreterQueue`:

    ```
    nano /opt/synergy/jboss/standalone/configuration/standalone-onesynergy.xml
    ```

    В секцию `<jms-destinations>` добавить:

```
<jms-queue name="InterpreterQueue">
    <entry name="java:jboss/queues/Integration/InterpreterQueue"/>
    <durable>true</durable>
</jms-queue>
```

5. Связать очередь и процесс через конфигурационный файл

```
nano  /opt/synergy/jboss/standalone/configuration/arta/api-observation-con
figuration.xml
```

Примечание
Если файл отсутствует, то необходимо его создать и добавить строки `<listeners>` `</listeners>`. Между ними нужно разместить очередь универсального слушателя очередей, указанную ниже. Также необходимо проверить права на файл (должно быть `jboss:synergy`).

```
<listener>
    <queue>java:jboss/queues/Integration/InterpreterQueue</queue>
    <event>event.blocking.interpreter.*</event>
</listener>
```

6. Установить на сервер ear-файл и скопировать его в директорию `/opt/synergy/jboss/stand alone/deployments`:

```
cd  /opt/synergy/jboss/standalone/deployments
cp -r  /путь до файла/synergy-interpreter-ear1.8.ear ./
chown -R jboss:synergy synergy-interpreter-ear1.8.ear
```

7. Перезапустите сервер.

1.3.3.1 После установки

1. Создать файл *interpreter.properties*:

```
nano  /opt/synergy/jboss/standalone/configuration/arta/interpreter/interpr
eter.properties
```

Примечание
Если каталога *interpreter* не существует, необходимо создать его с помощью команды *mkdir*.

Настроить права для данного файла:

```
chown -R jboss:synergy interpreter.properties
```

Добавить в него строки:

```
synergy.address=http://localhost:8080/Synergy
interpreter.prefix=event.blocking.interpreter.
script.not.found.send.ok=false
script.not.found.default.msg=Скрипт не найден
script.not.found.default.login=1
script.not.found.default.pwd=1
```

2. Включить режим логирования *TRACE*. Для этого откройте файл конфигурации

```
nano /opt/synergy/jboss/standalone/configuration/standalone-onesynergy.xml
```

В секцию `<profile>` подсекции `<subsystem xmlns="urn:jboss:domain:logging:1.1">` добавьте:

```
<periodic-rotating-file-handler name="iterpreter-handler">
    <formatter>
        <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t) %s%E%n"/>
    </formatter>
    <file relative-to="jboss.server.log.dir" path="interpreter.log"/>
        <suffix value=".yyyy-MM-dd"/>
    <append value="true"/>
</periodic-rotating-file-handler>
```

А также:

```
<logger category="kz.arta.ext.interpreter">
    <level name="TRACE"/>
        <handlers>
            <handler name="iterpreter-handler"/>
        </handlers>
</logger>
```

3. Перезапустите сервер.

1.3.4 Защита

Модуль «Интерпретатор» не имеет встроенной защиты от несанкционированного входа - защитить его можно внешним образом, например, используя `nginx`.

Ниже приведём пример с установкой защиты от входа в модуль при помощи web-сервера nginx, его модулей `http_auth_request_module`, `headers-more-nginx-module` и метода REST API Synergy `rest/api/auth/{role}`. Будем предполагать, что используется стандартный конфигурационный файл для `nginx`, поставляемый вместе с Synergy, `synergy-base`.

1.3.4.1 Вводная часть

Веб-сервер `nginx` встроенными средствами позволяет ограничивать доступ к серверу или какому-либо `location`-у с проверкой имени пользователя и пароля по протоколу «HTTP Basic Authentication», однако стандартный модуль `ngx_http_auth_basic_module` позволяет задать только статические пары `логин:пароль` в парольном файле. Мы же хотим использовать данные учётных записей Synergy и в этом нам поможет модуль `ngx_http_auth_request_module`. Этот модуль ограничивает доступ путём выполнения подзапроса со всеми заголовками оригинального запроса. Если кодом ответа на подзапрос будет `2xx`, то аутентификация будет считаться пройденной, в случае, если подзапрос возвращает `401`-й код ошибки, в ответ на оригинальный запрос будет передан заголовок `WWW-Authenticate` из подзапроса.

Специально для подобных случаев в Synergy предусмотрен метод API `rest/api/auth/{role}`, где вместо `{role}` можно передать `user`, `administrator` или `methodologist`. В случае, если пользовательск данные авторизации, переданные в заголовке `Authorization`, соответствуют пользователю, который

1. имеет доступ в систему и
2. обладает указанной ролью,

метод вернёт код `200`, в обратном случае - `403`, а при отсутствии заголовка `Authorization` - `401`.

Модуль `headers-more-nginx-module` понадобится нам для того, чтобы заменить содержимое заголовка `WWW-Authenticate`, которое передаёт API Synergy - в целях упрощения интеграции внешнего проигрывателя там сейчас передаётся `None` вместо `Basic`, а стандартная директива `nginx`, `add_header`, не срабатывает при `401` коде ответа от прокси.

1.3.4.2 Настройка

Для начала необходимо установить пакет `nginx-extras`. Возможен конфликт с пакетом `nginx-full` (если он у вас установлен) - в этом случае смело заменяйте последний на `nginx-extras` - он содержит всё то же самое, что и `nginx-full` + дополнительные модули.

```
# aptitude install nginx-extras
```

После установки вам необходимо добавить в конфигурационный файл `synergy-base` следующие директивы:

```
# editor /etc/nginx/sites-enabled/synergy-base

[ ... ]

server {
    server_name synergy.arta.pro; #DO NOT CHANGE. use dpkg-reconfigure arta-synergy-synergy

    [ ... ]

    # Новый location, используемый для аутентификации
    location = /auth-sd {
        proxy_pass              http://127.0.0.1:8080/Synergy/rest/api/auth/methodologist;
        more_set_headers -s 401 'WWW-Authenticate: Basic';
        proxy_pass_request_body off;
        proxy_set_header        Content-Length "";
        proxy_set_header        X-Original-URI $request_uri;
    }

    # И в секцию, которая соответствует Интерпретатору
    location /interpreter {
        auth_request            /auth-sd;
        proxy_pass              http://127.0.0.1:8080/interpreter;

        [ ... ]

    }

    [ ... ]

}
```

На этом настройка закончена, перезагрузим конфигурацию `nginx`:

```
# /etc/init.d/nginx reload
```

Теперь для доступа к `/interpreter` необходимо ввести логин и пароль активной учётной записи Synergy с правами «Synergy Developer» («суперметодолог»).

1.3.5 Интерфейс модуля

При переходе к модулю «Интерпретатор» открывается следующее окно:

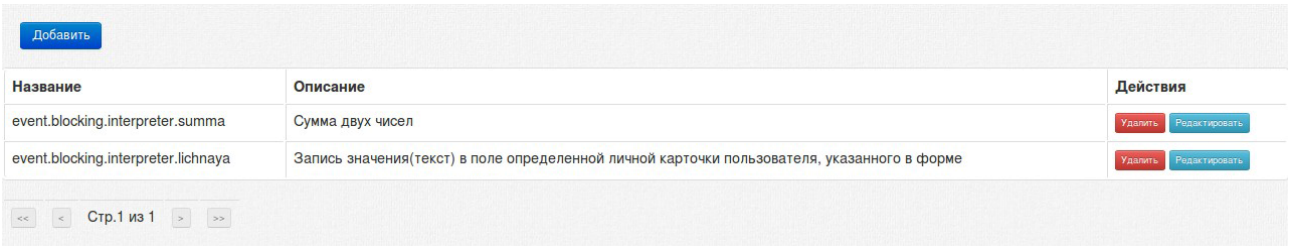


Рис. 1.11: Интерфейс модуля

В данном окне отображается список скриптов, которые можно редактировать и удалять, нажав соответствующую кнопку. Чтобы вставить новый скрипт, необходимо нажать кнопку «Добавить». Открывается окно:

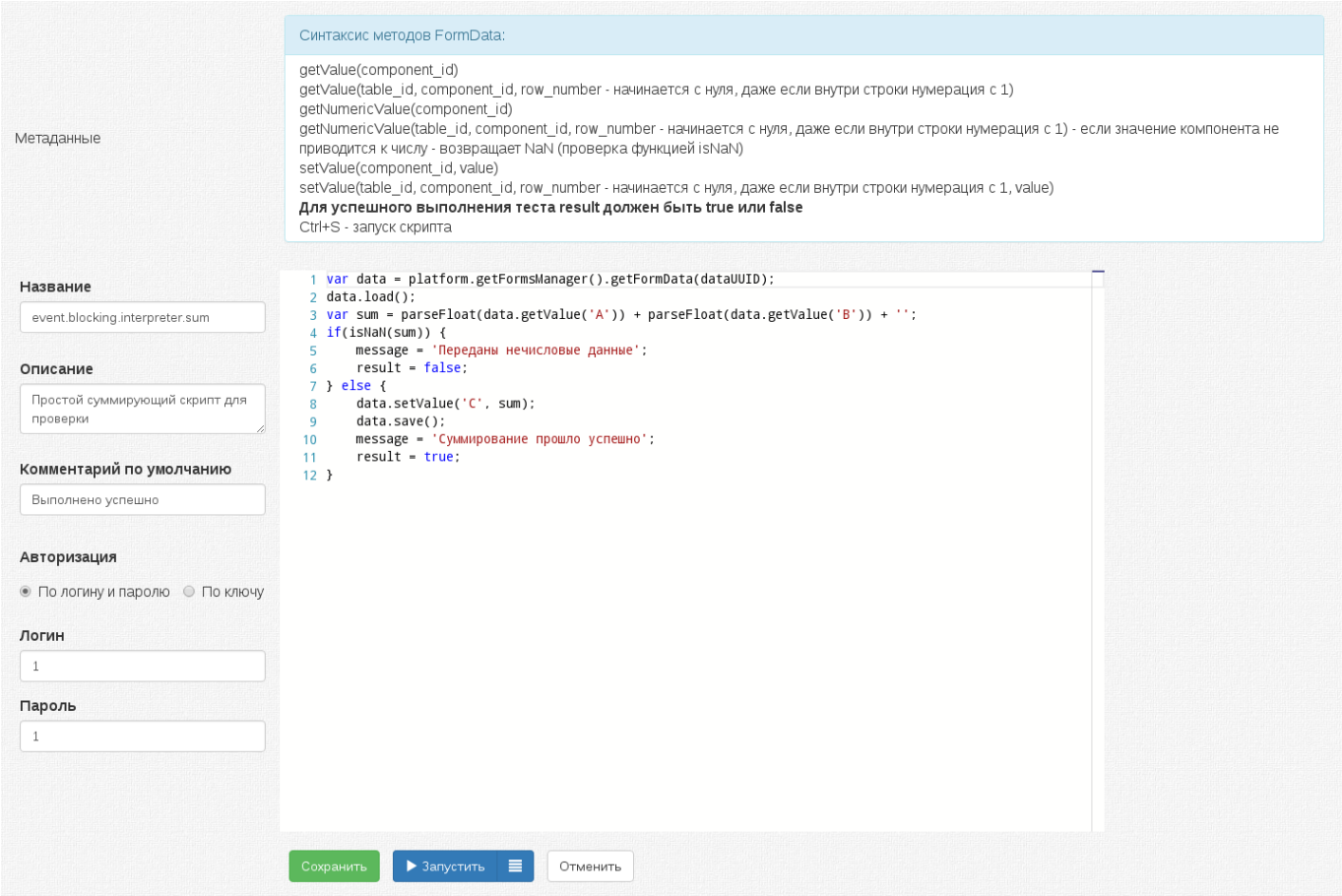


Рис. 1.12: Интерфейс модуля

Данное окно делится на две области: метаданные и код. Метаданные:

- Название — название скрипта в формате `event.blocking.interpreter.%название_скрипта%`;
- Описание;
- Комментарий по умолчанию;
- Авторизация.

В окне кода прописывается сам скрипт.

Скрипт может обращаться к параметрам авторизации, которые указаны в интерпретаторе, с помощью строковых переменных `login`, `password` и `key`.

> *Не забудьте сохранить написанный скрипт!*

1.3.6 Запуск скрипта

Написанный скрипт можно запустить непосредственно из интерпретатора с помощью кнопки «Запустить» или нажатием клавиш Ctrl+S. При первом запуске скрипта открывается окно «Конфигура выполнения скрипта», где можно ввести значения параметров `dataUUID`, `documentID` и `execu tionID` - эти параметры передаются скрипту интерпретатора при его запуске в блокирующем процессе Synergy:

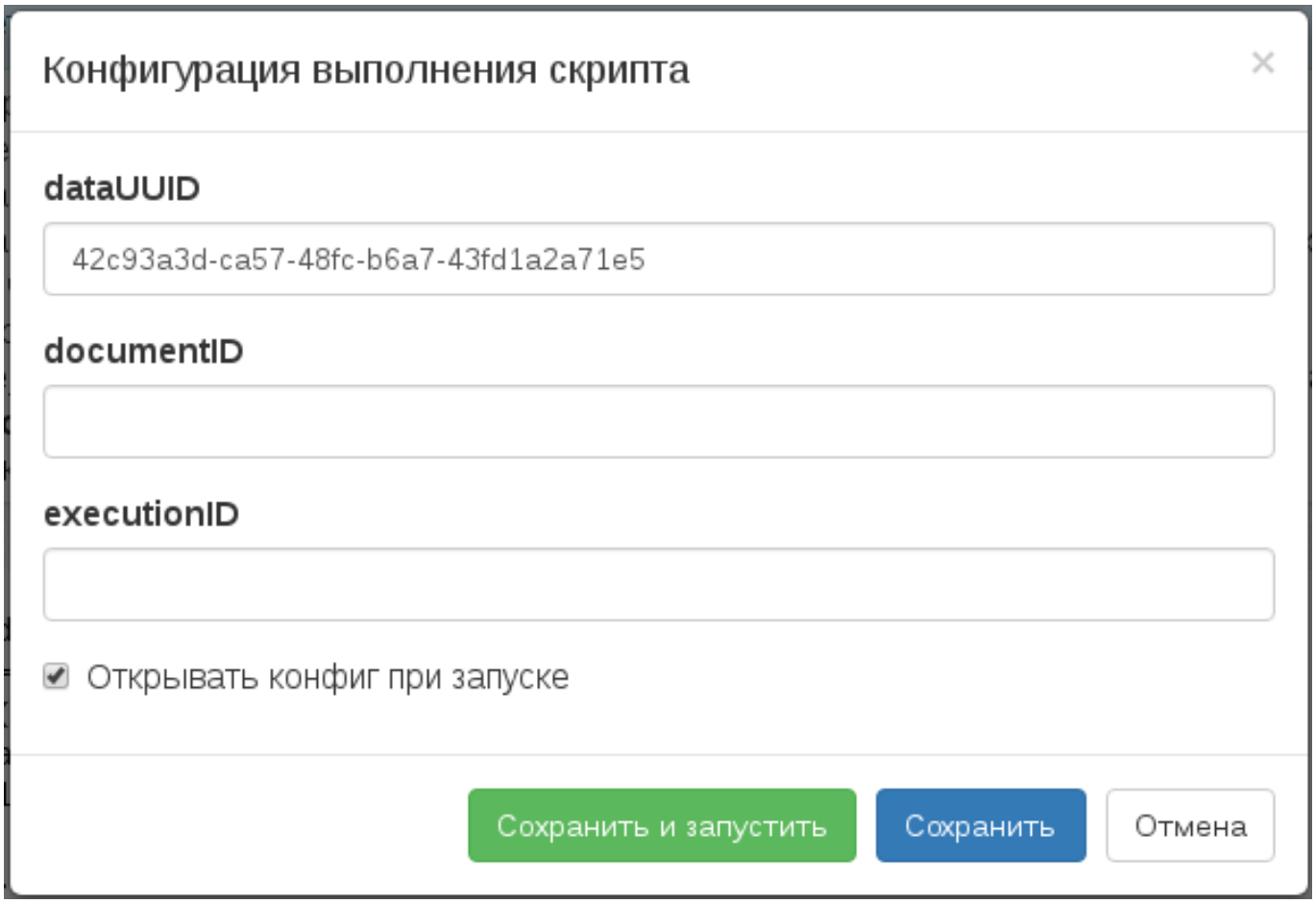


Рис. 1.13: Конфигурация выполнения скрипта

Примечание:

Формально эти параметры не обязательны для запуска скрипта, но выполнение скрипта без них может приводить к ошибкам.

Если флаг «Открывать конфиг при запуске» установлен, то это окно будет отображаться при каждом запуске скрипта, иначе - по нажатию на кнопку  .

Результат выполнения скрипта отображается в виде всплывающего сообщения в нижней правой части экрана:

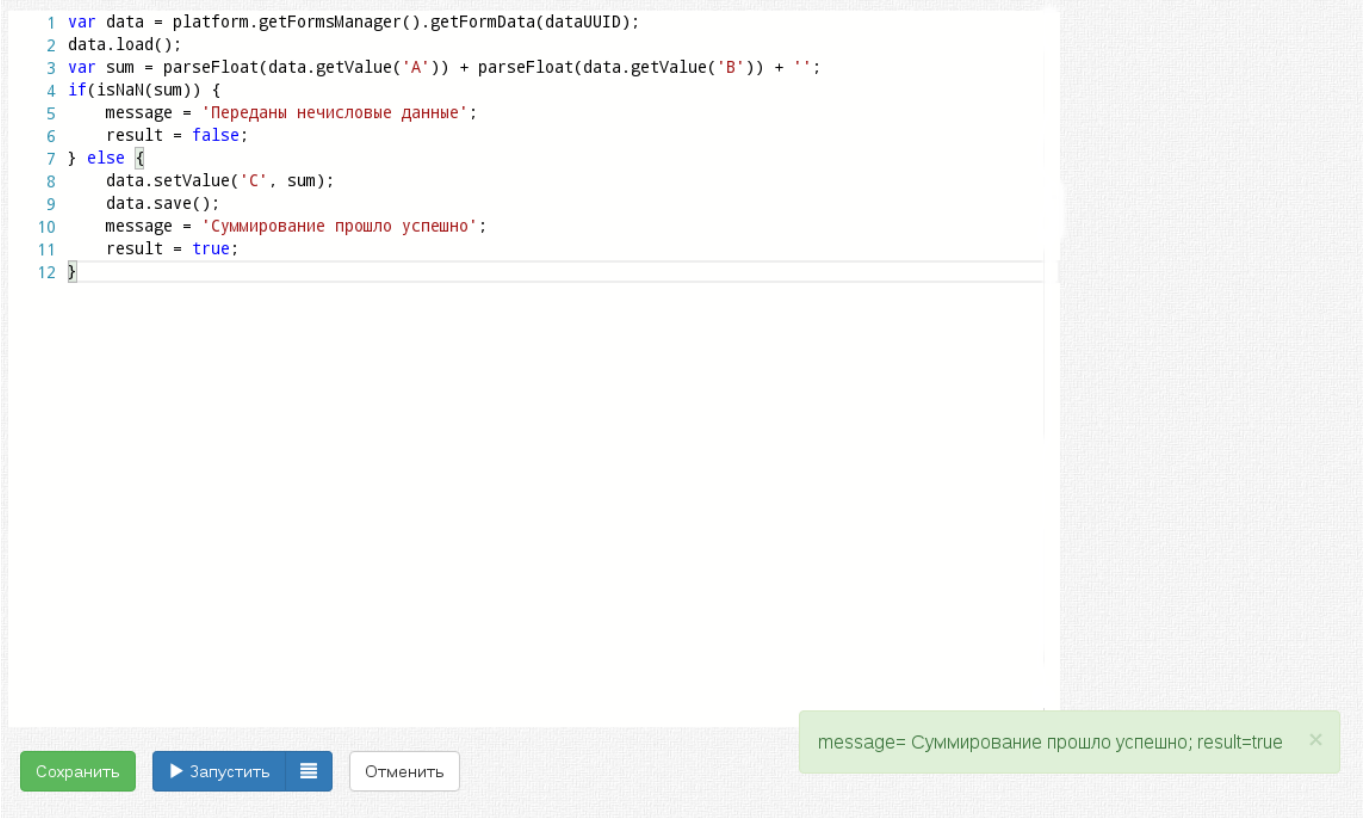


Рис. 1.14: Успешное завершение выполнения скрипта

Если при выполнении возникли ошибки, то они также отображаются во всплывающем сообщении:

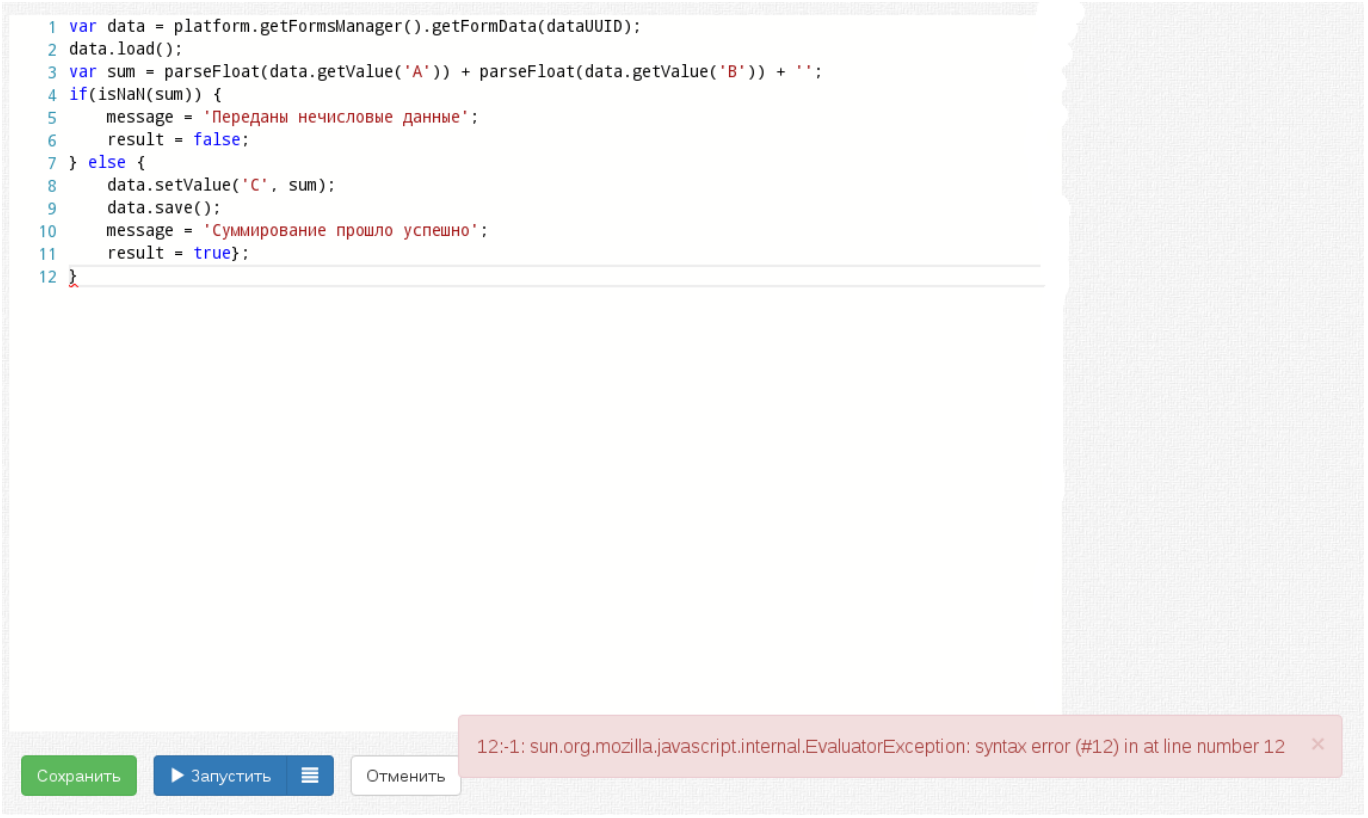


Рис. 1.15: Ошибка при выполнении скрипта

1.3.7 Объекты ARTA Synergy

Встроенный в java интерпретатор позволяет передавать Java объекты JavaScript-у, поэтому модуль интерпретатор предоставляет пользователю:

1. Объект платформы с названием *platform* или *synergy*;
2. Все строки, пришедшие как входные данные в блокирующий процесс;
3. *documentData* — объект *FormData* с подгруженными данными процесса.

Таким образом, код для суммирования двух полей формы и записи значения в третье поле документа будет выглядеть так:

```
summa = documentData.getValue(«a») + documentData.getValue(«b»);
documentData.setValue(«c», summa);
documentData.save();
```

Запись одного поля документа в поле личной карточки, при условии, что на форме есть Объект Synergy:

```
var cardData = synergy.getCardsManager().getUserCard(«идентификатор_формы»,
documentData.getValue(«user_chooser_id»));
cardData.setValue(«field», documentData.getValue(«document_field»));
cardData.save();
```

Пример работы с динамической таблицей:

```
for (i = 0; i < getRowsCount("id_таблицы"); i ++){
sum = sum + getValue("id_таблицы", "id_компонента", i);
}
```

1.3.7.1 Платформа

Назначение: Отвечает за создание экземпляров других объектов. Класс: `kz.arta.ext.interpreter.platform.Platform`.

Методы:

- `getFormsManager()` возвращает «Менеджер данных по формам»;
- `getCardsManager()` возвращает «Менеджер личных карточек».

1.3.7.2 Менеджер данных по формам

Назначение: Поиск и получение данных по формам. Класс: `kz.arta.ext.interpreter.forms.search.FormsManager`.

Методы:

- `getFormData(идентификатор_данных)` возвращает объект FormData.

1.3.7.3 Менеджер личных карточек

Назначение: Поиск и получение личных карточек пользователей. Класс: `kz.arta.ext.interpreter.forms.cards.CardsManager`.

Методы:

- `getUserCard(идентификатор_формы, идентификатор_пользователя)`

1.3.7.4 Файл по форме

Назначение: Подгрузка и сохранение данных по форме. Класс: `kz.arta.ext.interpreter.forms.data.FormData`.

Методы:

- `getValue(code)`
- `setValue(code, value)`
- `load()`
- `save()`
- `getRowsCount("id_таблицы")`
- `getValue("id_таблицы", "id_компонента", номер_строки)`
- `setValue("id_таблицы", "id_компонента", номер_строки, значение)`

1.3.8 Использование API методов

На данный момент итерпретатор позволяет обращаться ко всем доступным методам API ARTA Synergy. Для этого нужно прописывать запросы необходимых методов непосредственно в скрипт.

Пример 1. *POST-запрос API-метода*

```
// Создаём объект POST-запроса
var post = new org.apache.commons.httpclient.methods.PostMethod("http://192.168.4.6:8080/ ↩
    Synergy/rest/api/storage/copy");
// Добавляем параметры согласно спецификации метода "rest/api/storage/copy"
post.addParameter("fileID", fileReportID);
post.addParameter("documentID", documentID);
// Создаём HTTP-клиент и авторизационные данные
var client = new org.apache.commons.httpclient.HttpClient();
var creds = new org.apache.commons.httpclient.UsernamePasswordCredentials(synergyUser,  ↩
    synergyPass);
// Задаём клиенту способ авторизации и передаём авторизационные данные
client.getParams().setAuthenticationPreemptive(true);
client.getState().setCredentials(org.apache.commons.httpclient.auth.AuthScope.ANY, creds);
// Настраиваем заголовки запроса
post.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
// Выполняем метод
var status = client.executeMethod(post);
// Обязательно закрываем соединение
post.releaseConnection();
var result = true;
```

Пример 2. *GET-запрос API-метода*

```
// Блок аналогичен расположенному выше
var get = new org.apache.commons.httpclient.methods.GetMethod("http://127.0.0.1:8080/ ↩
    Synergy/rest/api/departments/list");
var client = new org.apache.commons.httpclient.HttpClient();
var creds = new org.apache.commons.httpclient.UsernamePasswordCredentials(synergyUser,  ↩
    synergyPass);
client.getParams().setAuthenticationPreemptive(true);
client.getState().setCredentials(org.apache.commons.httpclient.auth.AuthScope.ANY, creds);
get.setRequestHeader("Content-type", "application/json");
var status = client.executeMethod(get);

// Получаем HTTP-код возврата и преобразуем его в строку
// далее используем по своему усмотрению
var message = "" + status;
// Обязательно закрываем соединение
get.releaseConnection();
var result = true;
```

Пример 3. *GET-запрос API-метода*

```
// Блок аналогичен тому расположенному выше
var get = new org.apache.commons.httpclient.methods.GetMethod("http://127.0.0.1:8080/ ↩
    Synergy/rest/api/departments/list");
var client = new org.apache.commons.httpclient.HttpClient();
var creds = new org.apache.commons.httpclient.UsernamePasswordCredentials("ivanov", "1");
client.getParams().setAuthenticationPreemptive(true);
client.getState().setCredentials(org.apache.commons.httpclient.auth.AuthScope.ANY, creds);
get.setRequestHeader("Content-type", "application/json");
var status = client.executeMethod(get);

// Возвращает тело запроса HTTP, если такое есть, как String
var responseBody = get.getResponseBodyAsString();
```

```
var json = eval("(" + responseBody + ")");

var message = "" + status + " " + json[0].departmentID;

get.releaseConnection();

var result = true;
//
var responseBody = get.getResponseBodyAsString();
var json = eval("(" + responseBody + ")");
var message = "" + status + " " + json[0].documentID;
```

1.3.9 Авторизация

Так как API ARTA Synergy работает только с авторизацией модуль интерпретатор должен предоставлят возможность настраивать для каждого скрипта параметры авторизации:

- Логин и пароль пользователя, от имени которого должен работать скрипт;
- Ключ (для авторизации по ключам).

1.3.10 Завершение процесса

Блокирующий процесс может завершиться как успешно так и неуспешно. В обоих случаях необходимо передавать комментарий, говорящий о результате завершения процесса.

Модуль должен предоставлять возможность в скрипте указать как должен завершиться процесс и с каким комментарием. Для этого необходимо при завершении скрипта взять из него значения переменных:

- *result* - результат:
  - *true* (по умолчанию) — успешно завершено;
  - *false* — не успешно завершено.
- *message* — комментарий завершения; значение по-умолчанию вводится в метаданных скрипта.

1.3.11 Примеры скриптов

Пример 1. *Сумма двух чисел внутри одной формы*

> **Примечание**
>
> Компоненты должны быть числовыми.

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
var summa = form.getNumericValue("cmp-a") + form.getNumericValue("cmp-b");
form.setValue("cmp-c", summa);
form.setValue("cmp-d", summa);
form.save();
var result=true;
var message = "OK";
```

Пример 2. *Запись значения(текст) в поле определенной личной карточки пользователя, указанного в форме*

Примечание

Личную карточку после отработки процесса нужно обновить.

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
var card= platform.getCardsManager().getUserCard('97ec70b2-a5b1-455d-86de-28555323298d',  ↩
    form.getValue("userID"));
card.load();
card.setValue("cmp-8", 'Привет, мир!');
card.save();
var result = true;
var message = "Успешно завершено";
```

Пример 3. *Запись суммы двух компонентов формы в поле определенной личной карточки пользователя, указанного в форме*

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
var card= platform.getCardsManager().getUserCard('97ec70b2-a5b1-455d-86de-28555323298d',  ↩
    form.getValue("userID"));
card.load();
card.setValue("cmp-8", form.getNumericValue("cmp-a")+form.getNumericValue("cmp-b"));
card.save();
var result = true;
var message = "Успешно завершено";
```

Пример 4. *Разница между датой в форме и конкретным числом (количество полных дней)*

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();

var d1 = form.getValue("date-a");
var year = parseInt(d1.substring(0,4));
var month = parseInt(d1.substring(5,7).replace('0', ''))-1;
var day = parseInt(d1.substring(8,10));
var hh = parseInt(d1.substring(11,13));
var mi = parseInt(d1.substring(14,16));
var sec = parseInt(d1.substring(17));
var date_1 = new Date(year, month, day, hh, mi, sec);

var date_2 = Date.parse("October 4, 2014 19:28:34 GMT");
form.setValue("cmp-2",date_2);
var dif = (date_1.getTime()-date_2)/86400000
form.setValue("cmp-1", dif);
form.setValue("cmp-2",Math.floor(dif));

form.save();
var result=true;
var message = "Привет, мир!";
```

Пример 5. *Разница между двумя датами в личной карточке (количество полных дней), запись результата в поле формы*

```
var form = platform.getFormsManager().getFormData (dataUUID); form.load();
var card= platform.getCardsManager().getUserCard('97ec70b2-a5b1-455d-86de-28555323298d',  ↩
    form.getValue("userID"));
card.load();
var d1 = card.getValue("date_a");
var year_a = parseInt(d1.substring(0,4));
var month_a = parseInt(d1.substring(5,7).replace('0', ''))-1;
var day_a = parseInt(d1.substring(8,10));
```

```
var hh_a = parseInt(d1.substring(11,13));
var mi_a = parseInt(d1.substring(14,16));
var sec_a = parseInt(d1.substring(17));
var date_1 = new Date(year_a, month_a, day_a, hh_a, mi_a, sec_a);

var d2 = card.getValue("date_b");
var year_b = parseInt(d2.substring(0,4));
var month_b = parseInt(d2.substring(5,7).replace('0', ''))-1;
var day_b = parseInt(d2.substring(8,10));
var hh_b = parseInt(d2.substring(11,13));
var mi_b = parseInt(d2.substring(14,16));
var sec_b = parseInt(d2.substring(17));
var date_2 = new Date(year_b, month_b, day_b, hh_b, mi_b, sec_b);

var dif = (date_2.getTime()-date_1.getTime())/86400000;
form.setValue("cmp-a",Math.floor(dif));
form.save();
var result=true;
var message = "Привет, мир!";
```

Пример 6. *Количество строк в дин. таблице и сумма значений компонентов дин.таблицы*

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
form.setValue("cmp-c", form.getRowsCount("table"));
var sum=0;
for (i = 0; i < form.getRowsCount("table"); i ++)
{
sum = sum + form.getNumericValue("table", "cmp-table", i);
}
form.setValue("cmp-b",sum);
form.setValue("cmp-8",sum);
form.save();
var result=true;
var message = "OK";
```

Пример 7. *Код, который будет завершать процесс успешно, если значение переменной sum_one больше 100000, а неуспешно в обратном случае будет выглядеть так:*

```
var form = platform.getFormsManager().getFormData (dataUUID);
form.load();
form.save();
if (form.getNumericValue("sum_one") < 100000){
     result = true;
     message = "Успешно отправлено по маршруту";
} else {
     result = false;
     message = "Заявка не выполнена. Сумма превышает стандарт" ;
}
```

Больше примеров смотрите в Cookbook.

1.4 Руководство по работе с аналитическими дашбордами

Аналитические дашборды - это набор диаграмм, отображающих состояние данных в различных представлениях и разрезах. Они предназначены для упрощения работы по их оценке, обработке, прогнозированию дальнейшего состояния и принятию решений.

1.4.1 Введение

Для интеграции аналитических дашбордов в Synergy используется комбинация инструментов **Elasticsearch** и **Kibana**.

Elasticsearch (ES) - это мощный инструмент для полнотекстового поиска и анализа данных. Он позволяет быстро загружать, выполнять поиск и анализировать большие объемы данных. Однако ES не имеет специальной визуальной оболочки, и его использование возможно с помощью набора специальных API.

Kibana - это платформа для анализа и визуализации данных. Kibana обрабатывает данные, загруженны в ES, и работает только параллельно с ним. Если работа с ES предполагает использование специальног синтаксиса команд, то Kibana позволяет обрабатывать те же данные с помощью визуального интерфейса. При этом Kibana содержит интерпретатор, позволяющий использование всех возможност и специальных команд ES.

Индексация и обработка исходных данных Synergy производится с помощью ES, дальнейший анализ и визуализация - с помощью Kibana.

В настоящем документе будут рассмотрены только некоторые из возможностей этих инструментов, непосредственно относящиеся к задаче визуализации данных. Для подробного изучения всех их возможностей и способов использования рекомендуем обращаться к официальной документации:

- Elasticsearch;
- Kibana.

1.4.2 Подключение Elasticsearch и Kibana

1.4.2.1 Системные требования

Для реализации диаграмм используются продукты Elasticsearch (индексация данных и поиск) и Kibana (визуализация данных). Наибольшие системные ресурсы занимает Elasticsearch (ES). Для его работы рекомендуется использовать отдельный сервер. Наиболее критичным ресурсом для ES является оперативная память: **минимальный допустимый размер - 8Gb**, рекомендуемый - от 16 до 64 Gb.

Для хранения индексов рекомендуется выделять отдельный диск или RAID-массив, причем желательн использовать SSD.

Актуальные системные требования перечислены здесь.

1.4.2.2 Подключение пакетов Elasticsearch и Kibana

Пакеты Elasticsearch и Kibana, подготовленные для интеграции и Synergy, а также пакет установки Java 8 располагаются в репозитории `unstable`. Для корректной установки убедитесь, что в файле `/etc/apt/sources.list` прописаны и не закомменированы следующие строки:

```
deb http://deb.arta.kz/tengri unstable main contrib non-free
```

Обновите репозиторий, выполнив команду:

```
aptitude update
```

1.4.2.2.1 Установка Java

Рекомендуется предварительно установить Java. Для работы ES необходима 8 версия Java.

Установка из подключенных репозиториев

Версией Java по умолчанию должна стать 8-я, поэтому выполняем в следующем порядке:

```
aptitude install oracle-java8-installer
```

Для того, чтобы проверить, что Java по умолчанию 8-я, выполняем команду:

```
java -version
```

Вывод должен быть таким:

```
java version "1.8.0_111"
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)
```

Если Java по умолчанию получила другую версию, выводим список установленных версий, выполнив команду:

```
update-java-alternatives --list
```

Список установленных версий будет выведен в следующем виде:

```
java-1.7.0-openjdk-amd64 1071 /usr/lib/jvm/java-1.7.0-openjdk-amd64
java-8-oracle 1081 /usr/lib/jvm/java-8-oracle
```

Переключим версию на нужную, выполнив команду:

```
update-java-alternatives --set java-8-oracle
```

Установка из ранее скачанного архива аналогична описанной в Руководстве администратора.

1.4.2.2.2 Установка и настройка Elasticsearch

В консоли сервера, предназначенного для работы ES, выполните команду:

```
aptitude install elasticsearch
```

Будет установлен пакет ES.

Запуск ES осуществляется командой:

```
/etc/init.d/elasticsearch start
```

Команды остановки, перезапуска и проверки статуса ES аналогичны используемым для jboss:

- `stop` - остановка;
- `restart` - перезапуск (комбинация команд `stop` и `start`);
- `status` - проверка текущего статуса ES.

По умолчанию ES доступен по адресу `localhost:9200`. Изменить эту настройку можно в файле `/etc/elasticsearch/elasticsearch.yml`.

Проверить запуск ES можно, перейдя в браузере по адресу `localhost:9200` либо выполнив команду:

```
curl localhost:9200
```

Вывод должен быть таким:

```
{
  "name" : "RFSWkzt",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "r67YbmerQvyNHdxlzDIt3A",
  "version" : {
    "number" : "5.1.2",
    "build_hash" : "c8c4c16",
    "build_date" : "2017-01-11T20:18:39.146Z",
    "build_snapshot" : false,
    "lucene_version" : "6.3.0"
  },
  "tagline" : "You Know, for Search"
}
```

Для комплексной установки Java8 и Elasticsearch необходимо установить общий пакет:

`aptitude install arta-synergy-indexator-elasticsearch`

Этот пакет по зависимостям установит пакеты `oracle-java8-installer` и `elasticsearch`, а также установит версию Java по умолчанию и настроит конфигурационные файлы.

1.4.2.2.3 Установка и настройка Kibana

В консоли сервера выполните команду:

```
aptitude install kibana
```

По умолчанию Kibana запускается по адресу `localhost:5601`, адрес используемого ES - `localhost:9200`. Изменить эти настройки можно в конфигурационном файле Kibana: `/etc/kibana/kibana.yml`. Если необходимо, чтобы Kibana была доступна по локальной сети, нужно изменить параметр `server.host`, указав для него IP-адрес сервера Kibana и раскомментировав соответствующую строку:

```
    # Kibana is served by a back end server. This setting specifies the port to use.
    #server.port: 5601

    # Specifies the address to which the Kibana server will bind. IP addresses and host  ↩
        names are both valid values.
    # The default is 'localhost', which usually means remote machines will not be able to  ↩
        connect.
    # To allow connections from remote users, set this parameter to a non-loopback address.
    server.host: "192.168.1.79"

    # Enables you to specify a path to mount Kibana at if you are running behind a proxy.  ↩
        This only affects
    # the URLs generated by Kibana, your proxy is expected to remove the basePath value  ↩
        before forwarding requests
    # to Kibana. This setting cannot end in a slash.
    #server.basePath: ""

    # The maximum payload size in bytes for incoming server requests.
    #server.maxPayloadBytes: 1048576

    # The Kibana server's name.  This is used for display purposes.
    #server.name: "your-hostname"
```

```
    # The URL of the Elasticsearch instance to use for all your queries.
    #elasticsearch.url: "http://localhost:9200"

    # When this setting's value is true Kibana uses the hostname specified in the server. ↩
        host
    # setting. When the value of this setting is false, Kibana uses the hostname of the  ↩
        host
    # that connects to this Kibana instance.
    #elasticsearch.preserveHost: true
```

Примечание:

Обратите внимание, что Kibana **не имеет встроенных средств для контроля доступа**: при переходе по адресу любой пользователь имеет полные права на запись, редактирование и удаление данных. Если требуется обеспечение защиты, предлагаем использовать средства **`nginx reverse proxy`**.

Запуск Kibana осуществляется командой:

```
/etc/init.d/kibana start
```

Команды остановки, перезапуска и проверки статуса Kibana аналогичны используемым для jboss и ES.

Примечание:

Во время запуска и работы Kibana обязательно должен быть запущен ES, иначе возникнет ошибка:

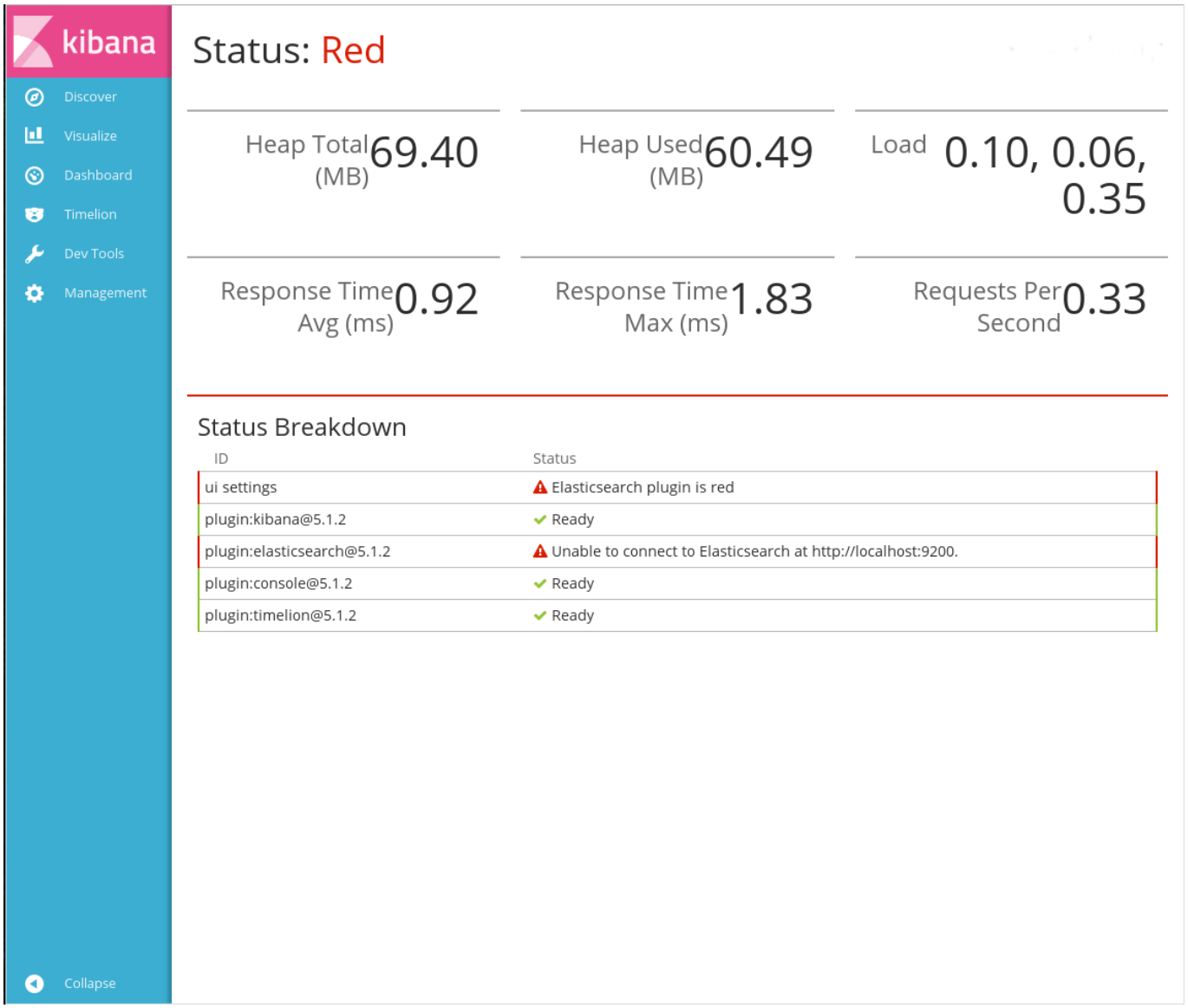


Рис. 1.16: Ошибка «Status: RED»

1.4.2.3 Индексация данных форм

Запись данных в индекс производится после сохранения данных по форме. Первичная загрузка данных в ES осуществляется с помощью процесса индексации данных форм (Административное приложение -> Обслуживание системы -> Управление индексом форм). В это время для каждой формы и каждого компонента этой формы в Synergy создается несколько индексов. Каждый из этих индексов будет отображен в Kibana со своим кодом, как используемое поле.

1.4.2.3.1 Названия индексов и alias-ы

- Для всех данных по форме, принадлежащих реестру с идентификатором `someRegistryID`, создаётся индекс с именем `<index-prefix>-r-someRegistryID`.
- Для всех данных по форме с идентификатором `someFormID` создаётся индекс с именем `<index-prefix>-f-someFormID`.

- Если итоговая длина названия индекса (как для форм, так и для реестров) превысит 255 байт, оно будет обрезано до 255 байт.

Таким образом, для каждого реестра и для каждой формы, по которым есть данные, будет создано по индексу. Если по форме создан реестр, а также созданы какие-то данные вне реестра, то в этом случае будет создано два индекса:

1. `<index-prefix>-r-IdOfRegistryWithOtherForm`
2. `<index-prefix>-f-otherID`.

Замечание:

Здесь описано, как получить названия всех имеющихся индексов в Elasticsearch.

Кроме этого, для удобства использования и возможности переноса конфигурации для каждого из вышеперечисленных индексов создаётся alias. Имена alias-ов формируются так:

- Для данных реестров: `<alias-prefix>-r-нормализованный_код_реестра`
- Для данных форм без реестров: `<alias-prefix>-f-нормализованный_код_формы` где `нормализованый_код_реестра` и `нормализованный_код_формы` - коды, соответственно, реестра и формы, в которых специальные символы , ., [,], {, }, (,), +, -, ?, ^, $, | заменяются на _.

Предупреждение

При этом возможна ситуация, когда нормализованные коды разных реестров совпадут и alias будет создан на на все соответствующие индексы. Эта маловероятное затруднение может быть решено изменением кодов соответствующих реестров или форм. В случае, если это невозможно, необходимые alias-ы можно создать вручную.

При изменении кода реестра или формы в Synergy имя соответствующего alias-а также изменяется.

`<index-prefix>` и `<alias-prefix>` настраиваются в конфигурационном файле `arta/elasticConfiguration.xml`

1.4.2.3.2 Структура документа в индексе

Одна единица данных в индексе Elasticsearch называется *Документ*. Документ содержит поля определённых типов. Каждый документ в текущем индексе соответствует одной единице данных по форме (=файлу по форме, записи реестра) и содержит следующие поля:

- `asfDataId` - идентификатор данных по форме, тип `keyword`;
- `formId` - идентификатор формы, тип `keyword`;
- `formCode` - код формы, тип `keyword`;
- `registryId` - идентификатор реестра, тип `keyword` (содержит значение `-1` для данных по форме, не связанных с реестром);
- `documentId` - идентификатор документа Synergy, тип `keyword`;
- `status` - статус записи реестра, тип `number`:

  - 0 - «Подготовка» (NO_ROUTE) - это значение также устанавливается для данных по форме, не связанных с реестром
  - 1 - «В процессе» (STATE_NOT_FINISHED)
  - 2 - «Активная» (STATE_SUCCESSFUL)
  - 3 - «Неуспешная» (STATE_UNSUCCESSFUL)
- deleted - признак удаления записи реестра (0 - не удалено, 1 - удалено), тип number (0 для данных по форме, не связанных с реестром);
- created - дата и время создания данных по форме, тип date;
- modified - дата и время изменения данных по форме, тип date.

Далее следуют поля, соответствующие компонентам формы:

- Для каждого компонента формы создаётся несколько полей документа в индексе.
- Название полей, соответствующих компоненту формы, формируется так: идентификаторкомпонентаформывнижнемрегистре_key_постфикс и идентификаторкомпонентаформывнижнемрегистре_value_постфикс (данные для которых берутся, соответственно, из полей key и value данных по форме).
- Для каждого поля *_key и *_value создаются поля с нижеперечисленными постфиксами.
- Для компонентов, находящихся внутри динамической таблицы, а также компонентов с мультивыбор («Объекты Synergy»), значения записываются в массив для всех постфиксов с учетом типов компонентов.
- Для компонентов, имеющих key и value, создается общее поле *_object (Object).

*Постфиксы для полей *_key:*

- _exact - поле содержит значение key, приведенное к нижнему регистру, тип keyword;
- _sort - поле содержит точное значение key, тип keyword;
- _number - поле содержит значение key, приведенное к числу, тип number;

  Примечание:

  Если поле key в документе пусто, в данное поле будет записано максимальное значение для типа long: 9 223 372 036 854 775 807

- _date - поле содержит значение key, приведенное к дате; поле присутствует только для компонентов Synergy типа «Дата/время»; тип date;
- _double - поле содержит значение key, приведенное к числу, тип double;

  Примечание:

  Данное поле создается только в том случае, если из значения key удалось выделить число (т.е. есть хотя бы один документ, использующий это поле, содержит числовое значение);

- пустой постфикс - поле содержит n-граммы значения key, через пробел, тип text.

*Постфиксы для полей *_value:*

- _exact - поле содержит значение value, приведенное к нижнему регистру, тип keyword;
- _sort - поле содержит точное значение value, тип keyword;
- _number - поле содержит значение value, приведенное к числу, тип number;

  Примечание:

  Если поле key в документе пусто, в данное поле будет записано максимальное значение для типа long: 9 223 372 036 854 775 807

- _prefix - поле содержит возможные префиксы из значения value, через пробел, тип text;
- _postfix - поле содержит возможные постфиксы из значения value, через пробел, тип text;
- пустой постфикс - поле содержит n-граммы значения value, через пробел, тип text.

1.4.2.3.3 Индексы изменения данных (историчные индексы)

Индексы изменения данных создаются только для тех форм и реестров, коды которых подпадают под шаблоны (секции в конфигурационном файле arta/elasticConfiguration.xml, см. описание выше).

Имя индекса <index-prefix>-rh-someRegistryID и <index-prefix>-fh-someFormID, для реестров и форм, соответственно. Alias-ы: rh-нормализованный_код_реестра и fh-нормализованный_код_формы.

Отличие индексов изменения данных от текущих индексов - на каждое изменение данных по форме создаётся новый документ в индексе. Кроме этого, для компонентов формы создаются только поля со следующими постфиксами:

- Для *_key:
  - _exact
  - _number
  - _date
  - _double
- Для *_value:
  - _exact
  - _number

Типы данных и условия создания полей такие же, как и в текущем индексе.

1.4.3 Визуализация данных в Kibana

1.4.3.1 Шаблоны индексов

Для использования индексов Elasticsearch в диаграммах Kibana необходимо указать эти индексы, используя **шаблоны индексов** (Index patterns). Они представляют собой маску имени, которой должны соответствовать индексы, входящие в этот шаблон.

Подробно о шаблонах индексов написано в официальном руководстве по Kibana

Например, если необходимо создать шаблон для индексов myindex-1, myindex-2, myindex-3 и myindex-abc, требуется создать шаблон индекса myindex-*, где символ * означает подстановку произвольного набора символов.

Примечание:

Поскольку имена индексов данных форм составляются на основе кодов соответствующих компонентов форм, рекомендуется присваивать этим компонентам коды с учетом некоторого значащего префикса так, чтобы используемые данные можно было объединить в группу по маске имени.

В случае, если изменение кодов компонентов не представляется возможным, можно создать шаблон индекса с маской “*“. Этот шаблон будет содержать все индексы Elasticsearch.

Другой способ объединения данных по форме в единый шаблон индекса - создание шаблона для отдельного реестра или формы. Например, если в диаграмме необходимо использовать данные реестра `someRegistryID`, нужно создать шаблон индекса с названием `r-someRegistryID`. Аналогично, для использования данных формы (в случае, если Synergy не содержит реестра для этой формы) с кодом `someFormID` нужно создать шаблон индекса `f-someFormID`.

Создание шаблонов индексов осуществляется в разделе *Management - Index Patterns*. Для создания нового шаблона нужно нажать на кнопку **+ Add New**. Откроется окно создания нового индекса:

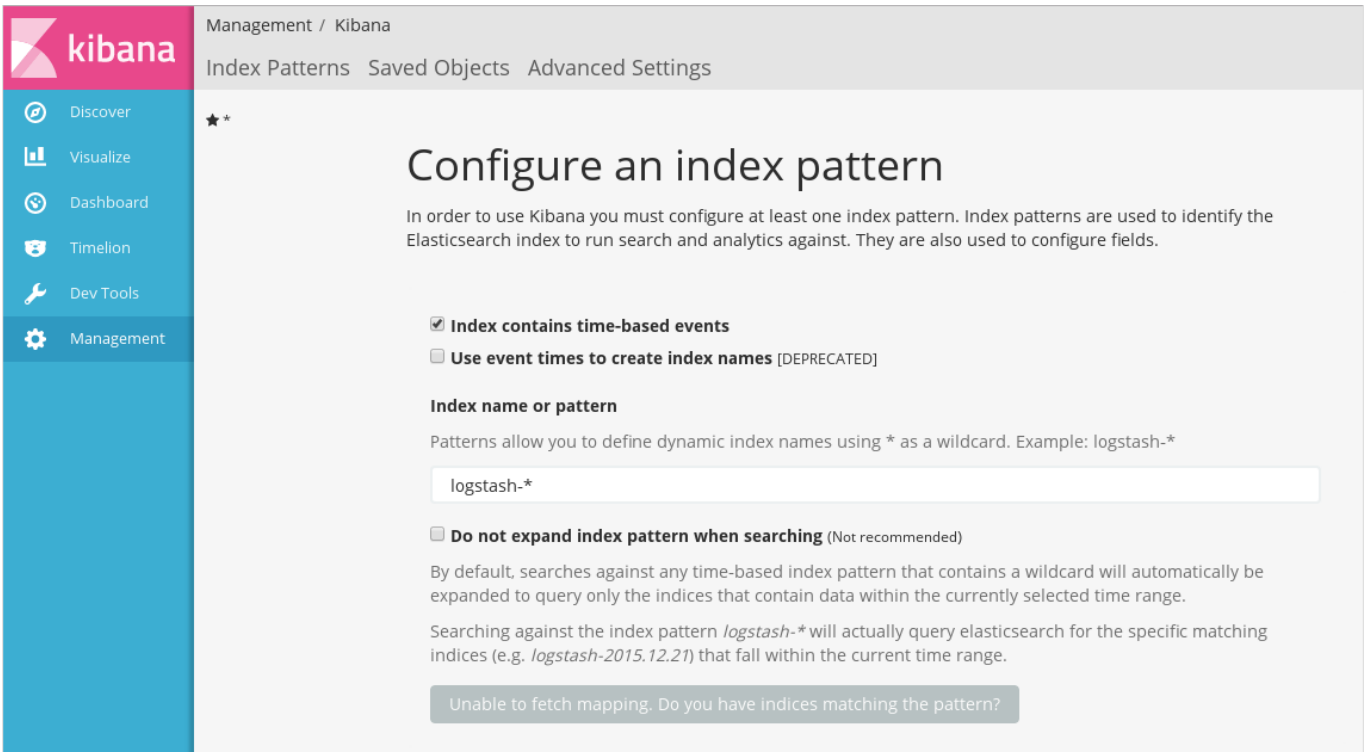


Рис. 1.17: Создание нового шаблона индексов

Установленный чекбокс **Index contains time-based events** означает, что данные, которые входят в шаблон, содержат временны**е** данные.

Примечание:

Не рекомендуется оставлять этот чекбокс включенным, если не планируется визуализация данных во времени - например, отслеживать нагрузку на сервер в настоящий момент. Без особой настройки диаграммы, использующие такие поля, будут отображать только данные, соответствующие текущему моменту времени.

В поле **Index name or pattern** необходимо ввести имя шаблона индекса:

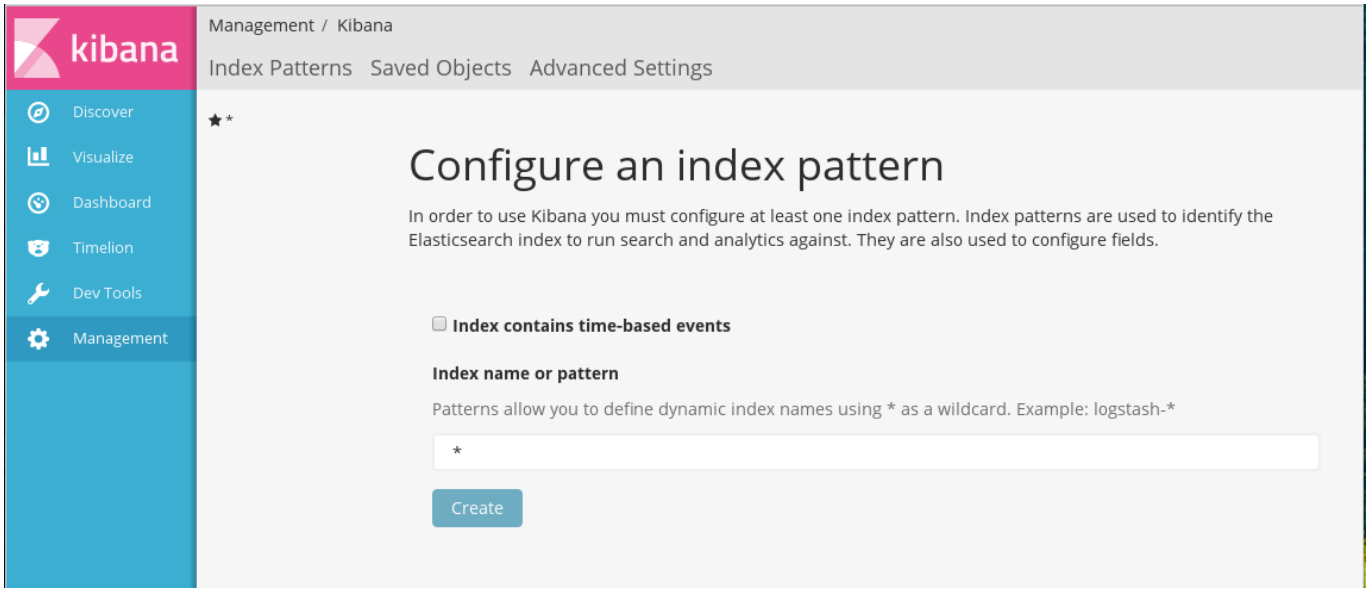


Рис. 1.18: Создание нового шаблона индексов без временных данных

В случае, есди Elasticsearch содержит индексы с именами, соответсвующими указанному шаблону, отобразится доступная кнопка **Create**. Для создания шаблона индекса нужно нажать на эту кнопку.

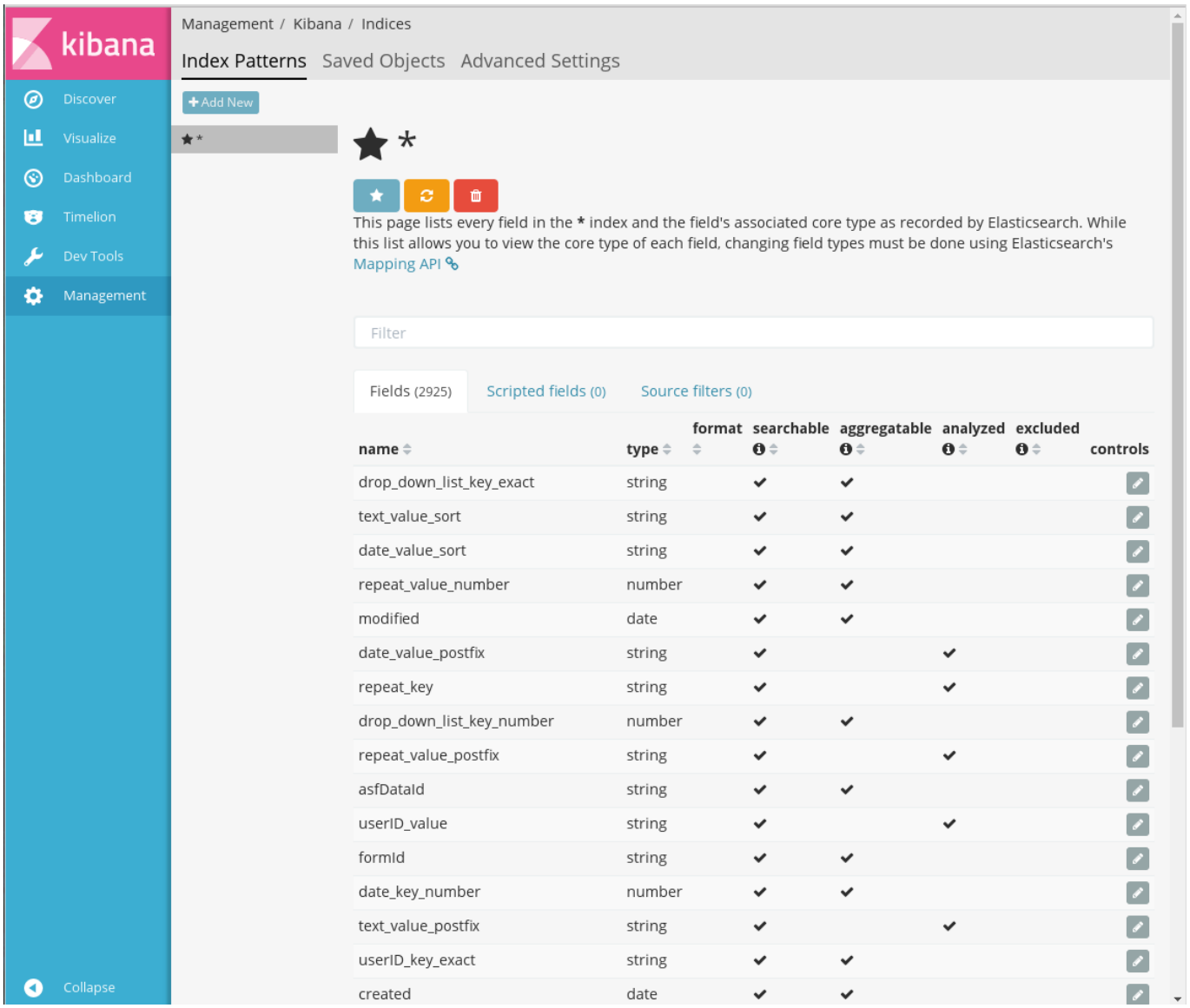


Рис. 1.19: Созданный шаблон индексов

После создания шаблона отображается таблица со списком индексов, входящих в этот шаблон, и их свойствами. Эти свойства зависят от типов индексов. Типы, с которыми индексируются данные форм Synergy, описаны в разделе Индексация данных форм.

1.4.3.2 Создание диаграмм

Kibana позволяет создавать следующие типы диаграмм:

1. **Area chart** - предназначена для отображения общего изменения данных во времени, когда выявление суммарного значения всех данных важнее, чем сравнение любых двух или более последовательностей. Например, полезна для отображения использвания ресурсов сервера.
2. **Data table** - отображение данных как разультата агрегации в виде таблицы.
3. **Line chart** - используется для отображения данных в виде линий (графиков). В отличие от Area charts, удобна для сравнения последовательностей между собой.

4. **Markdown widget** - вставка произвольной информации, используя синтаксис языка Markdown.
5. **Metric** - отображение одного числа - результата агрегации числовых данных.
6. **Pie Chart** - предназначена для отображения вклада нескольких частей в некоторый общий результат. Может принимать вид круговой (pie) или кольцевой (donut) диаграммы.
7. **Tag cloud** - отображение данных таким образом, чтобы их размер зависел от некоторого числового показателя этих данных (например, количества упоминаний).
8. **Tile map** - специфический тип диаграмм, использующий агрегацию географических данных (тип поля `geo_point`) для их отображения на карте.
9. **Timeseries** - специфический тип диаграмм, визуализирующий временные ряды.
10. **Vertical bar chart** - наиболее универсальная диаграмма, отображающая числовые показатели произвольных полей в виде вертикальной гистограммы.

Здесь будут рассмотрены некоторые наиболее универсальные из этих диаграмм. Для ознакомления с работой остальных типов рекомендуем обратиться к официальному руководству по Kibana

> **Примечание:**
>
> В диаграммах возможно использование только агрегируемых типов полей. К ним относятся все числовые типы, а также типы `date`, `keyword`, `geo_shape` и другие. Агрегируемые поля отмечены галочкой в графе «Aggregatable» (на странице Managenent - Index Patterns).

1.4.3.2.1 Общая часть

Все диаграммы создаются в разделе **Visualize**:

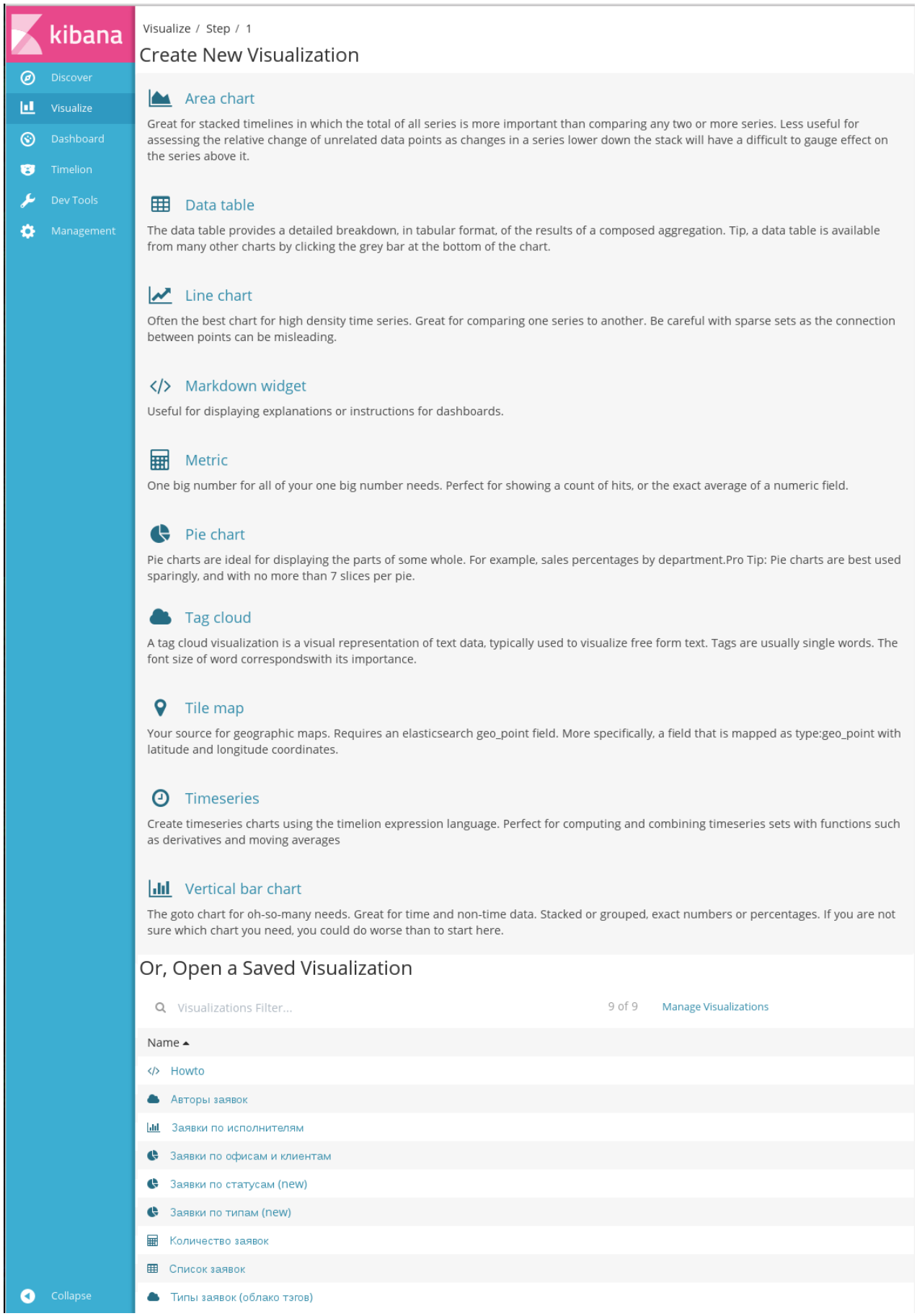


Рис. 1.20: Kibana, раздел Visualize

В общем случае, процесс создания диаграмм состоит из трех шагов:

1. Выбор типа диаграммы.
2. Выбор источника данных (шаблона индекса). В одной диаграмме возможно использование только одного шаблона, поэтому для использования в одной диаграмме данных документов по нескольким формам, необходимо использовать alias-ы.

   Этот шаг отсутствует для диаграммы *Markdown widget*.

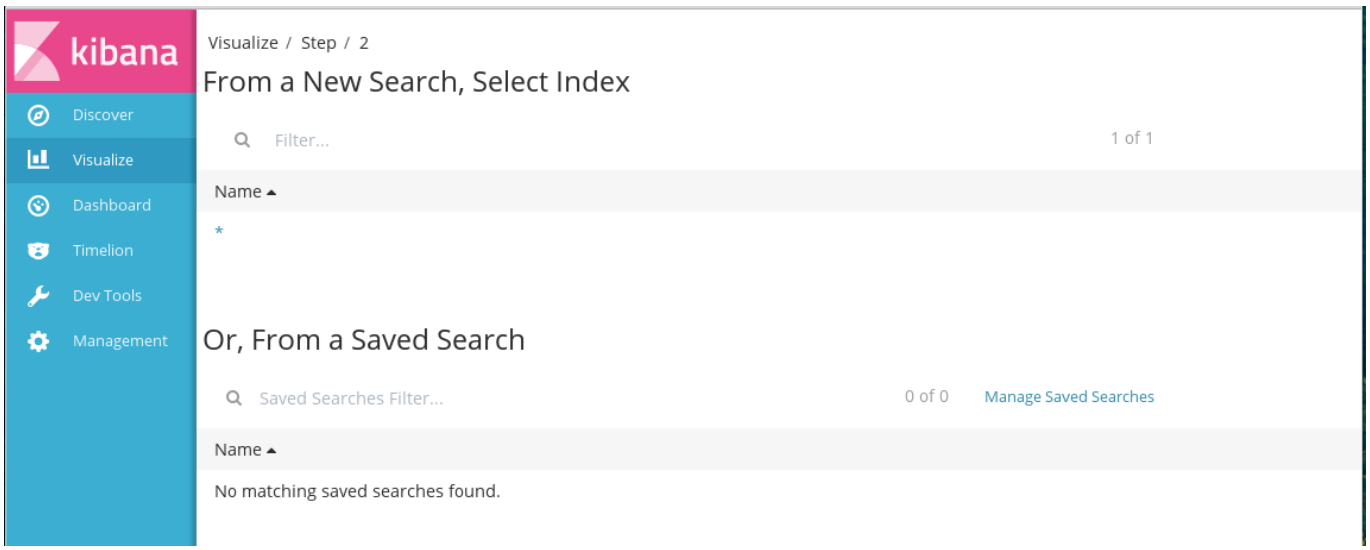


Рис. 1.21: Выбор источника данных

   В качестве источника данных может выступать шаблон индекса или результат поиска по данным (сохраненный или новый).

1. Настройка отображаемых данных:

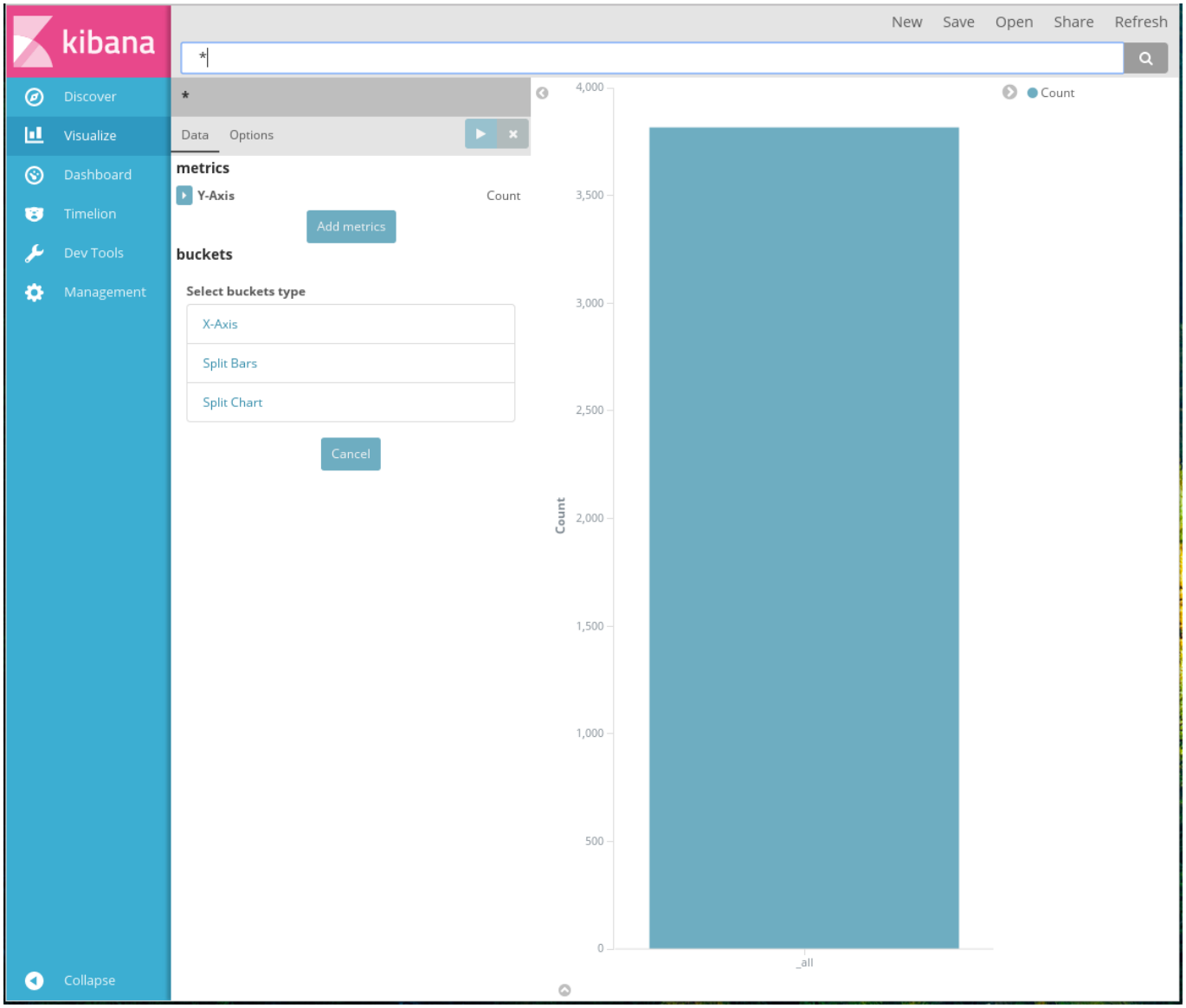


Рис. 1.22: Настройка отображаемых данных

Окно настройки данных имеет стандартный вид:

В верхней части располагается панель меню с пунктами:

- *New* - создать новую диаграмму, переход к шагу 1;
- *Save* - сохранить диаграмму;
- *Open* - открыть существующую диаграмму;
- *Share* - предоставить доступ к сохраненной диаграмме;
- *Refresh* - обновить данные.

Ниже панели меню расположено поле поиска данных, использующее синтаксис **Lucene**. Данное поле используется для фильтрации данных, отображаемых в диаграмме. Например, для отображения только неудаленных документов в этой панели нужно ввести запрос:

```
deleted:0
```

где `deleted` - поле, генерируемое во время индексации данных форм и хранящее значение 0, если данные в Synergy не были удалены.

Основная рабочая область окна делится на две части:

- настройка данных: выбор полей и способа их агрегации;
- просмотр результатов: отображение результата обработки выбранных данных.

В части настройки данных, в разделе *Metrics*, необходимо выбрать способ агрегации числовых данных, отображаемых в диаграмме. В поле *Aggregation* выбирается способ агрегации, а в поле *Custom label* вводится отображаемое название параметра.

В разделе *Buckets* необходимо выбрать используемые данные, числовые параметры которых будут отображены на диаграмме. Так же, как и в разделе *Metrics*, здесь в поле *Aggregation* выбирается способ агрегации, а в поле *Custom label* вводится отображаемое название параметра. Отличие от раздела *Metrics* состоит в том, что раздел *Buckets* позволяет группировать произвольные типы данных при выборе соответствующего типа агрегации.

Наиболее универсальным способом агрегации, используемом во всех примерах ниже, является *Term*. Этот способ позволяет агрегировать данные как строки, аналогично функции `GROUP BY` в SQL. Подробно об остальных типах агрегации можно ознакомиться в официальном руководстве по Kibana.

При выборе этого типа агрегации дополнительно отображаются поля:

- *Field* - выпадающий список, содержащий все поля, входящие в текущий шаблон индекса, для которых доступна агрегация.
- *Order By* - параметр сортировки данных - по метрике из раздела *Metrics*, по отдельной метрике (*Custom metric*) или по содержимому текущего поля (*Term*).
- *Order* - направление сортировки:
- *Descending* - по убыванию;
- *Ascending* - по возрастанию.
- *Size* - количество отображаемых элементов - отображаются указанное количество элементов, располагающиеся в начале списка отсортированных указанным образом данных.
- *Custom label* - отображаемое название параметра.

Для каждого используемого параметра, независимо от того, используется ли он в разделах *Metrics* или *Buckets*, доступна дополнительная настройка, отображаемая при нажатии на лейбл **Advanced**:

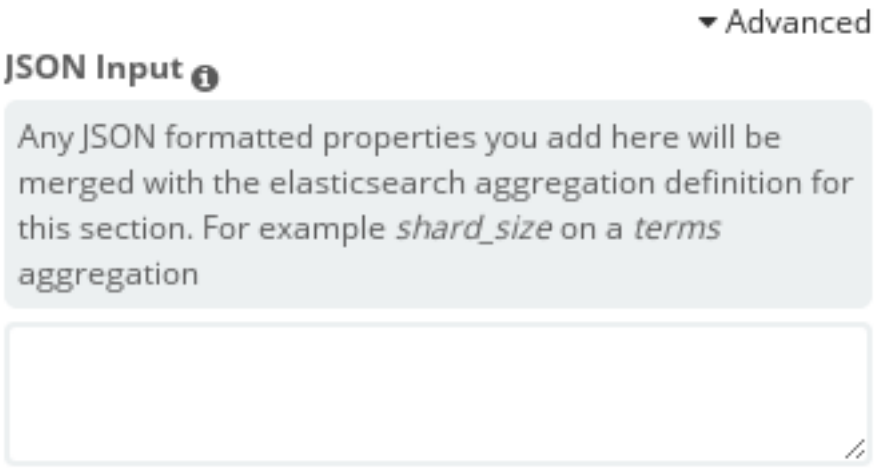


Рис. 1.23: JSON Input

Она представляет собой текстовое поле, в которое можно добавить специальные свойства в формате `JSON`, например:

```
{ "script" :"doc['grade'].value * 1.2" }
```

В настоящем документе процесс создания диаграмм и дашбордов будет рассмотрен на примере анализа данных формы «Заявка»:

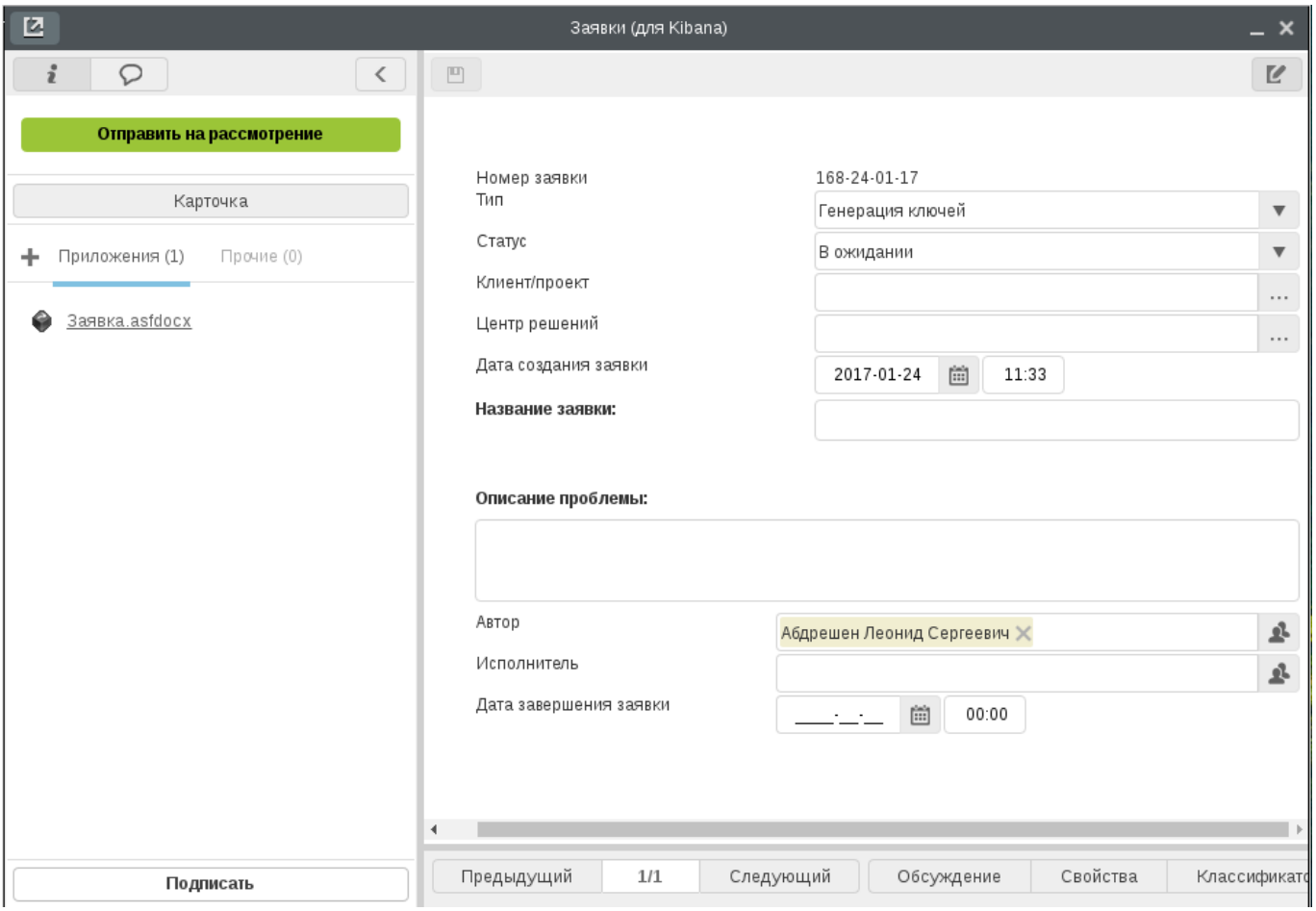


Рис. 1.24: Форма «Заявка»

1.4.3.2.2 Pie chart

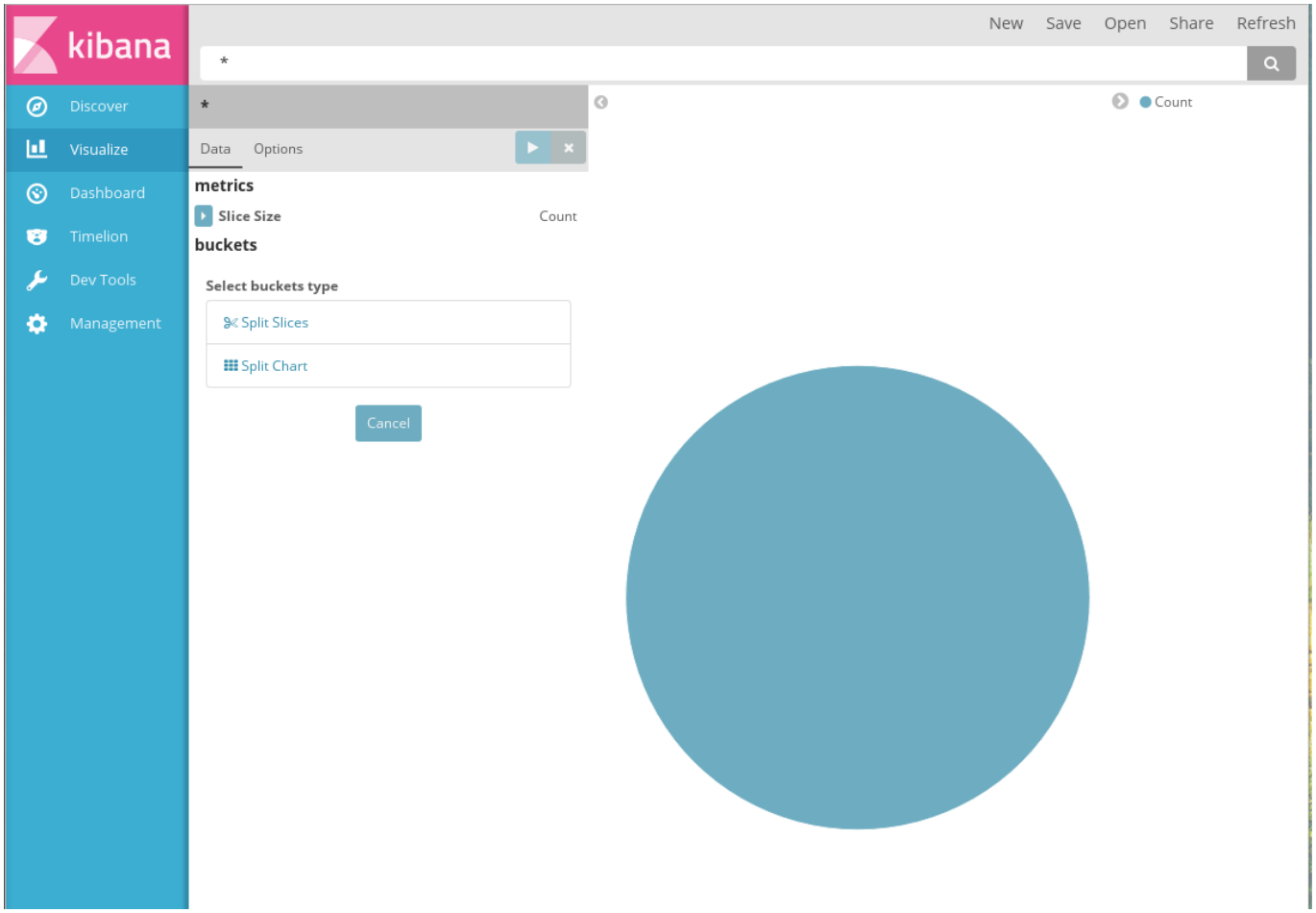


Рис. 1.25: Создание диаграммы Pie chart

Для диаграммы Pie chart возможно два способа организации используемых параметров:

- **Split Slices**: параметр будет отображен на диаграмме как новый уровень секторов:

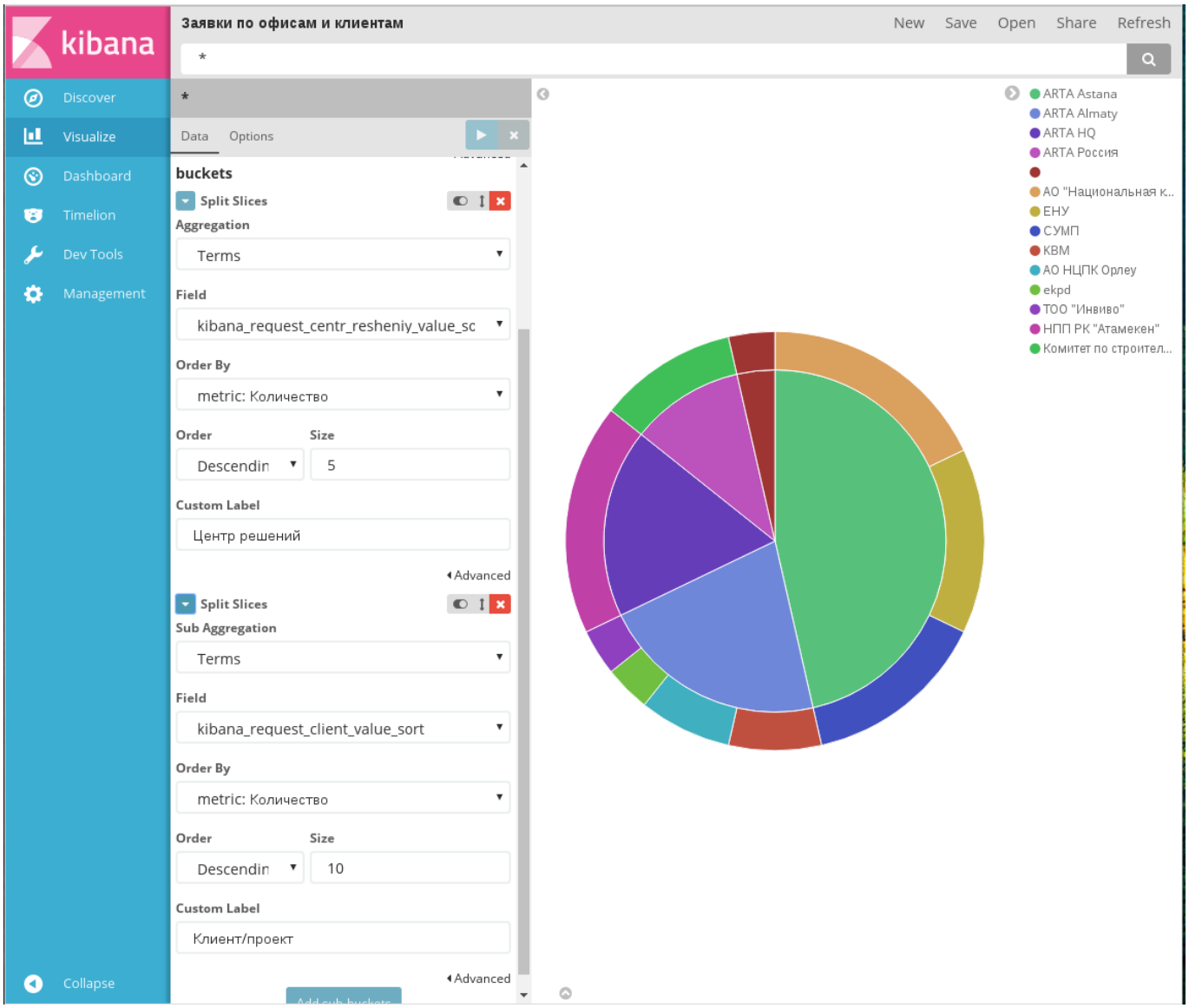


Рис. 1.26: Заявки по офисам и клиентам

На этом примере в разделе *Buckets* в качестве первого параметра использовано поле «Центр решений» - результаты этой агрегации на диаграмме отображены во внутреннем круге.

В качестве второго параметра используется поле «Клиент/проект», и результаты этой агрегации отображаются во внешнем круге.

> **Примечание:**
>
> Такая последовательность была выбрана в силу специфики входных данных: известно, что один центр решений занимается несколькими проектами, но одним проектом занимается ровно один центр решений.

Итоговая диаграмма позволяет оценить распределение объема заявок как по центрам решений, так и по отдельным проектам.

- **Split Chart**: для нового параметра будет построена отдельная диаграмма:

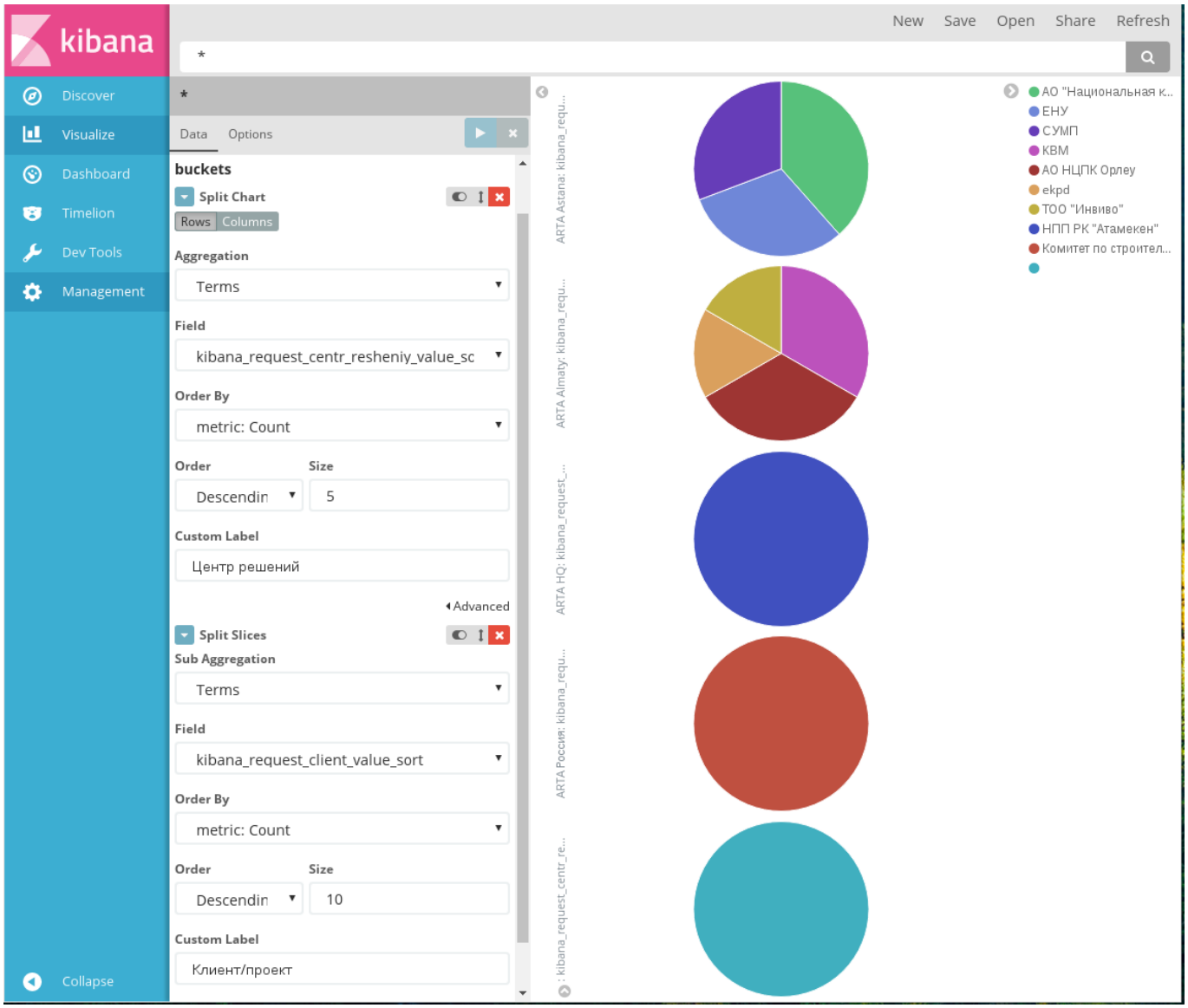


Рис. 1.27: Заявки по офисам и клиентам

Здесь для каждого центра решений (поле выбрано первым параметром с типом *Split Chart*) отрисована отдельная диаграмма, в которой показано распределение заявок по проектам этих центров (второй параметр с типом *Split Slices*). Видно, что три центра решений оставляли заявки только по одному из своих проектов.

Примечание:

Kibana допускает использование *Split Chart* только в сочетании с *Split Slices*, причем в этом случае параметр со *Split Chart* обязательно должен располагаться выше, чем параметр со *Split Slices* (сначала разделить данные по отдельным диаграммам, а потом разделять данные внутри каждой диаграммы).

Добавить новый параметр можно, нажав на кнопку **Add sub-buckets**.

Вкладка **Options** для этого типа диаграмм содержит три параметра:

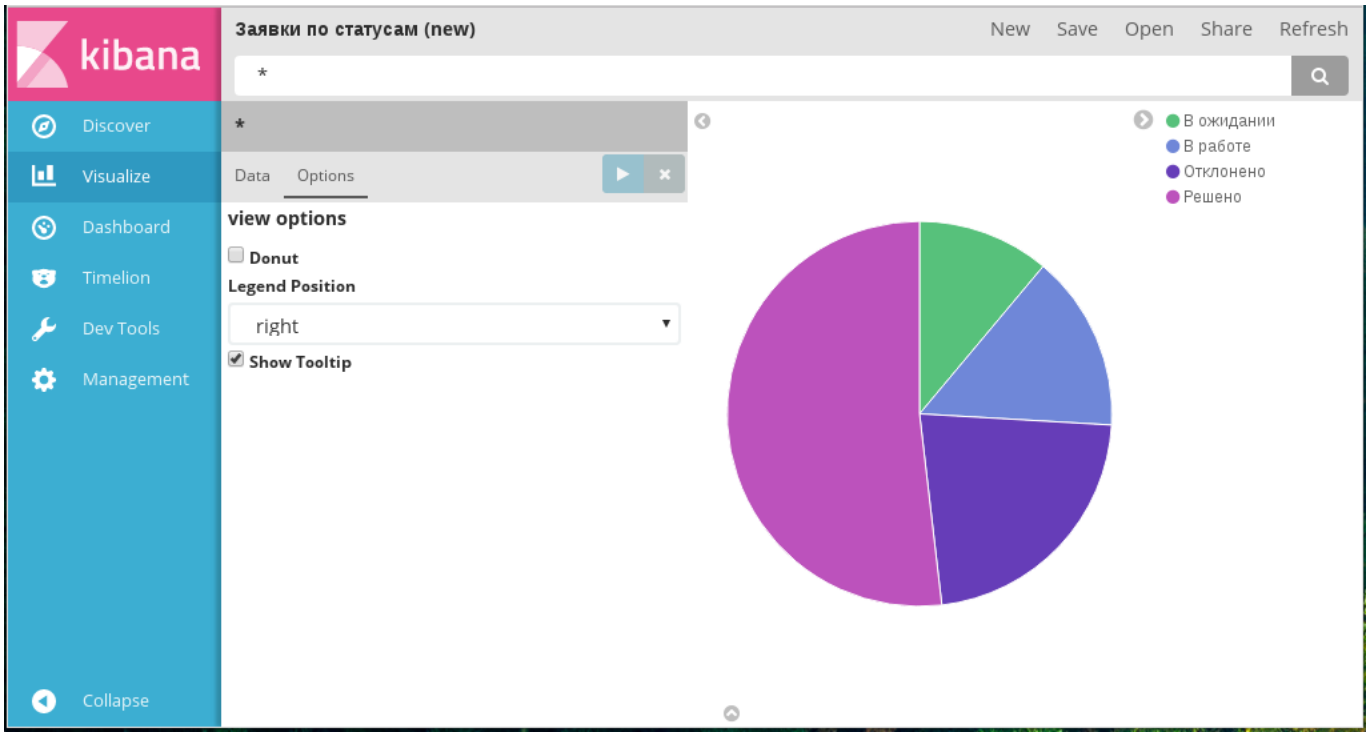


Рис. 1.28: Вкладка «Опции» диаграммы Pie chart

- Вид диаграммы: если чекбокс *Donut* включен, диаграмма принимает вид кольцевой, если отключен - круговой (по умолчанию).
- Расположение легенды: по умолчанию справа от диаграммы.
- Показывать всплывающие подсказки при наведении на часть диаграммы: отображаются, если включен чекбокс *Show Tooltip*.

1.4.3.2.3 Data table

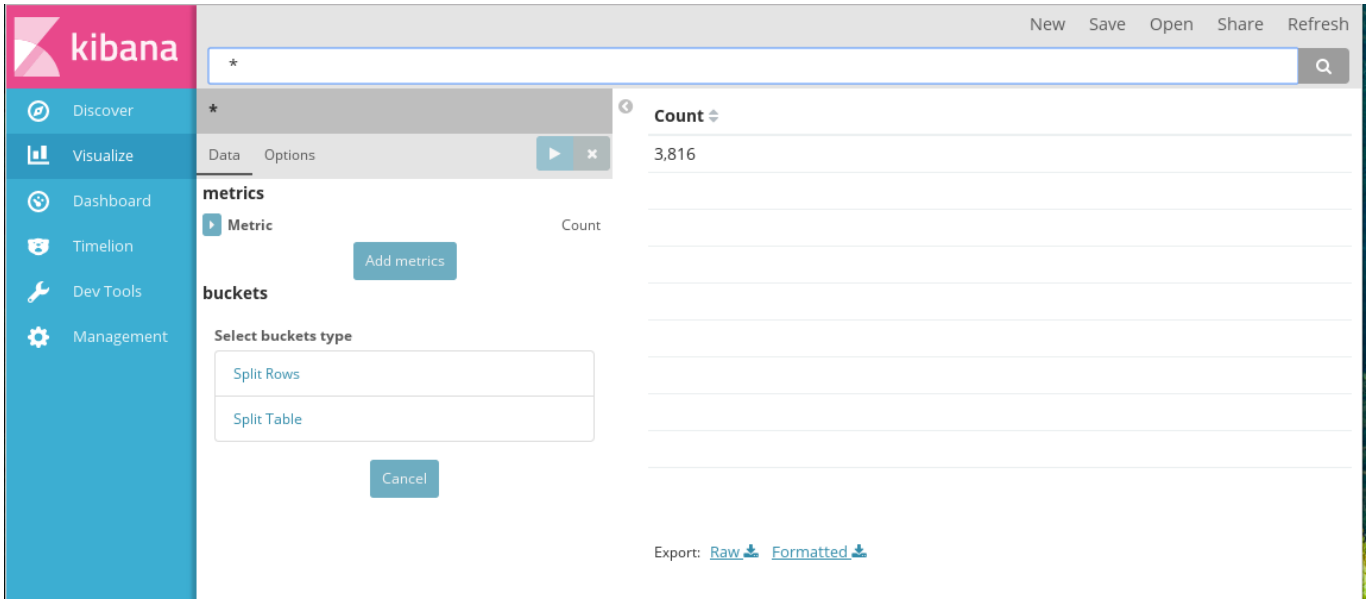


Рис. 1.29: Создание диаграммы Data table

В разделе *Metrics* необходимо указать одну или несколько метрик, по которым будут агрегироваться данные в таблице.

В разделе *Buckets* необходимо указать используемые параметры и способы их агрегации. Для этого типа диаграмм также существует два способа организации входных параметров:

- *Split Rows* для добавления параметра как нового столбца к текущей таблице;
- *Split Table* для добавления параметра как отдельной таблицы.

Функциональность этих способов полностью аналогична *Split Slices* и *Split Chart* для круговой диаграммы.

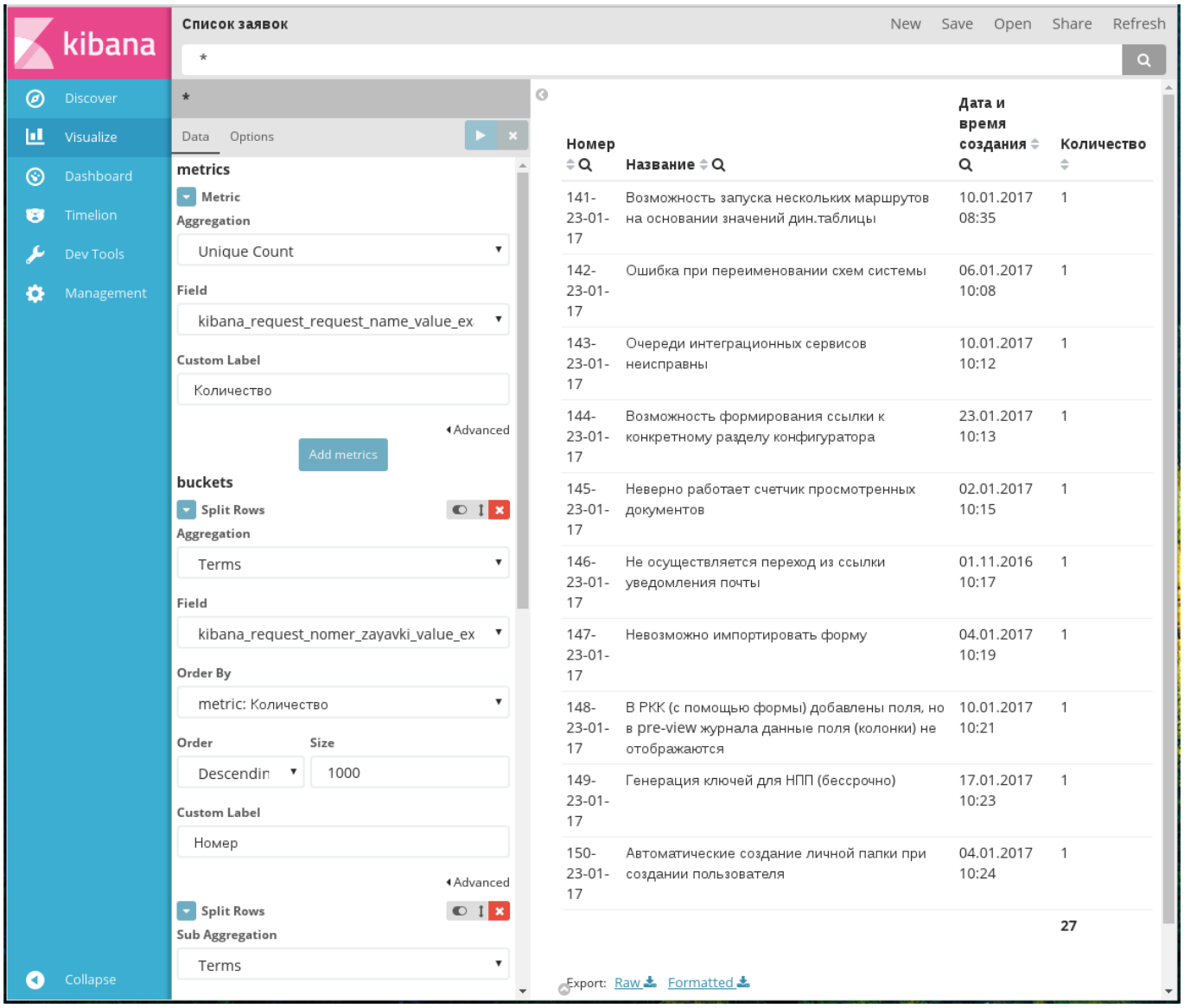


Рис. 1.30: Настроены отображаемые данные в таблице

В диаграмме, указанной на рисунке выше, все параметры были добавлены как *Split Rows*. Для каждого параметра в разделе *Buckets* используется агрегация *Terms*.

Вкладка **Options** для этого типа диаграмм содержит следующие параметры:

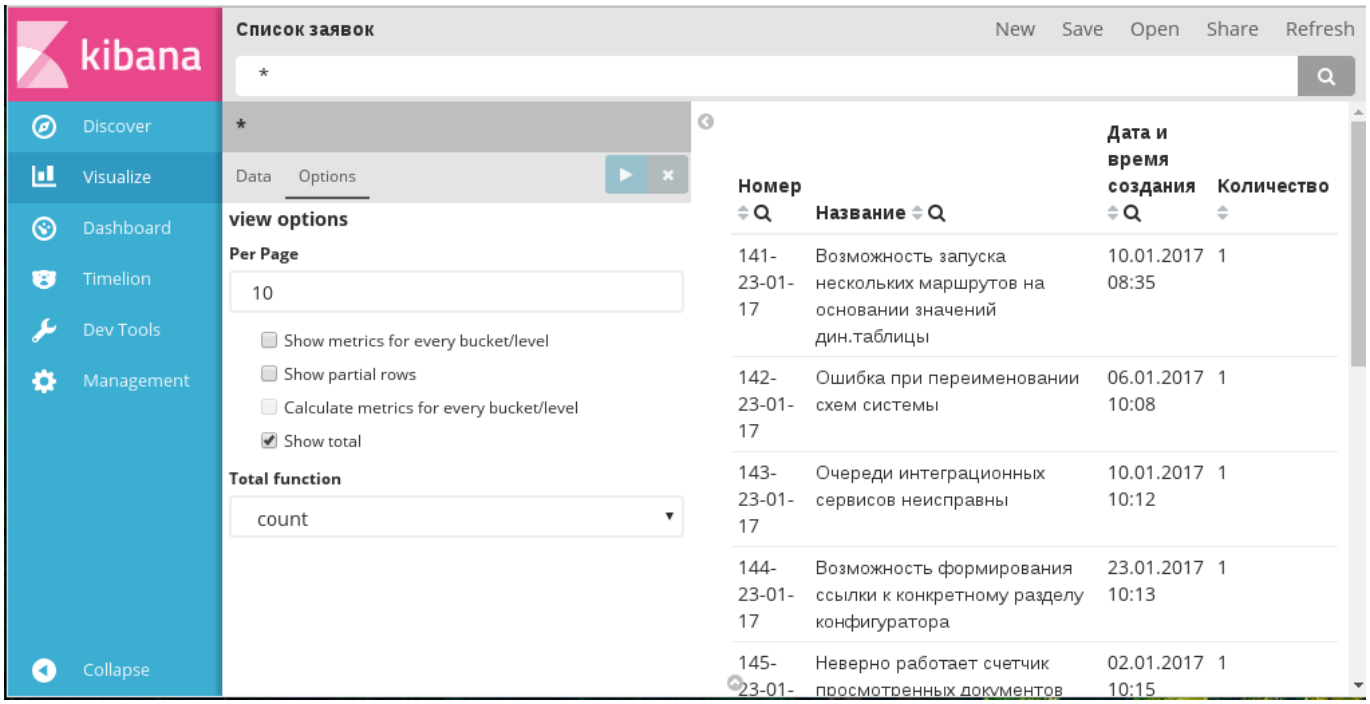


Рис. 1.31: Вкладка «Опции» диаграммы Data table

- Количество отображаемых строк на странице: по умолчанию отображается 10 строк. В случае, если все записи не помещаются на одну страницу, в нижней части таблицы отображается переключатель страниц.
- Отображать метрики для каждой группы/уровня: если чекбокс включен, то для каждого столбца (в случае *Split Rows*) или каждой таблицы (в случае *Split Table*) будет добавлен столбец с результатом агрегации из раздела *Metrics*.
- Отображать частичные строки: если чекбокс включен, то в таблицу будут включены строки с данными, отсутствующими для выбранных индексов (полей). По умолчанию в таблице отображаются только полностью заполненные строки.
- Считать метрики для каждой группы/уровня: чекбокс, недоступный для ручной установки. Его значение зависит от параметра «Отображать метрики для каждой группы/уровня».
- Отображать итоговые значения: если чекбокс включен, то на каждой странице таблицы для каждой отображаемой метрики будет указано итоговое значение этой метрики для всех данных таблицы.
- Функция для итогов: выбор функции для подсчета итоговых значений метрик. Параметр доступен только в том случае, если установлен чекбокс «Отображать итоговые значения».

1.4.3.2.4 Vertical bar chart

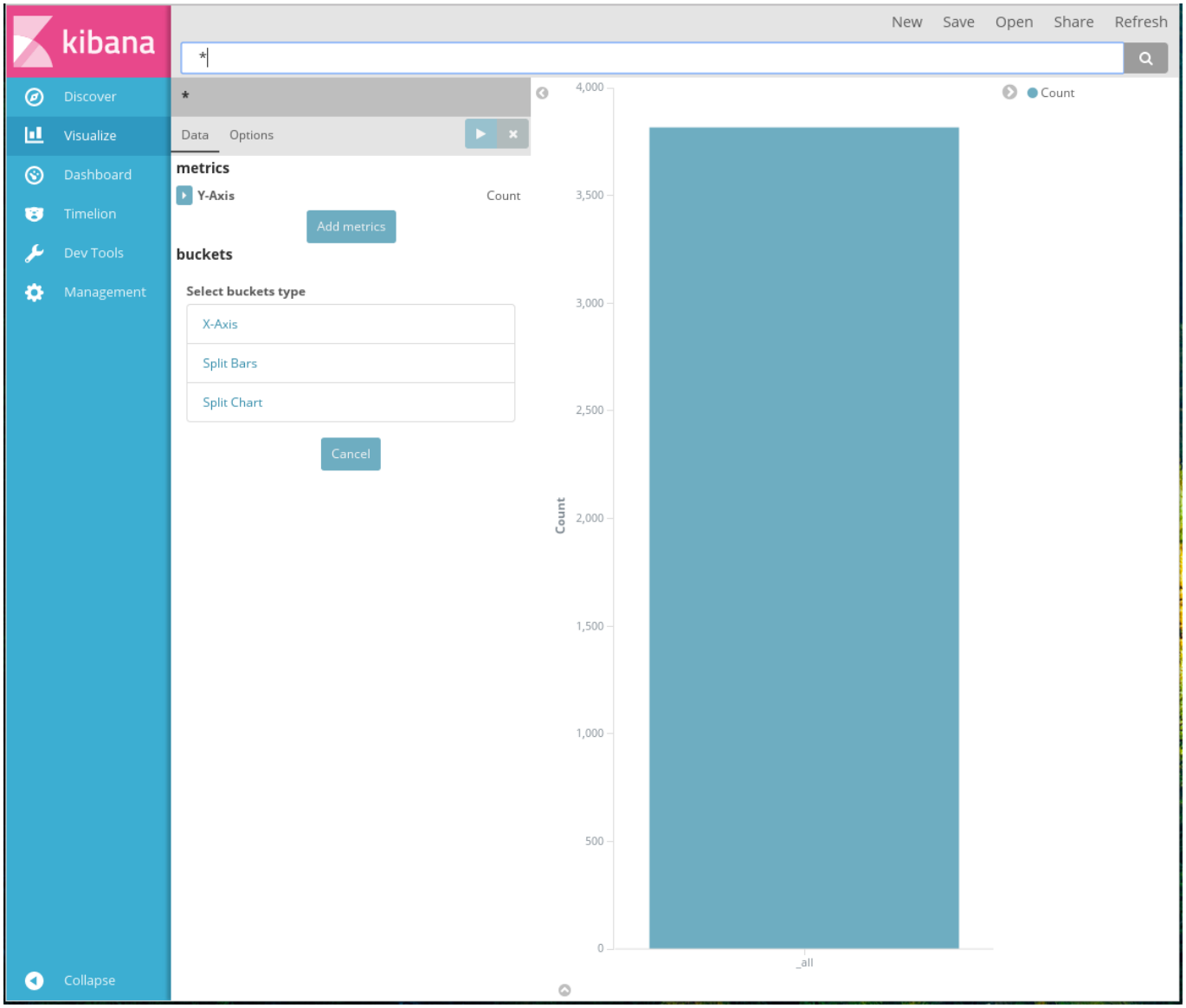


Рис. 1.32: Создание диаграммы Vertical bar chart

В диаграмме этого типа по оси Y располагаются метрики (параметры в *Metrics*), с по оси X - группы (параметры в *Buckets*).

Доступно указание нескольких метрик на оси Y и не больше одной группы каждого типа:

- *X-Axis*
- *Split Bars*
- *Split Chart*

Функциональность *Split Bars* и *Split Chart* полностью аналогична *Split Slices* и *Split Chart* для круговой диаграммы.

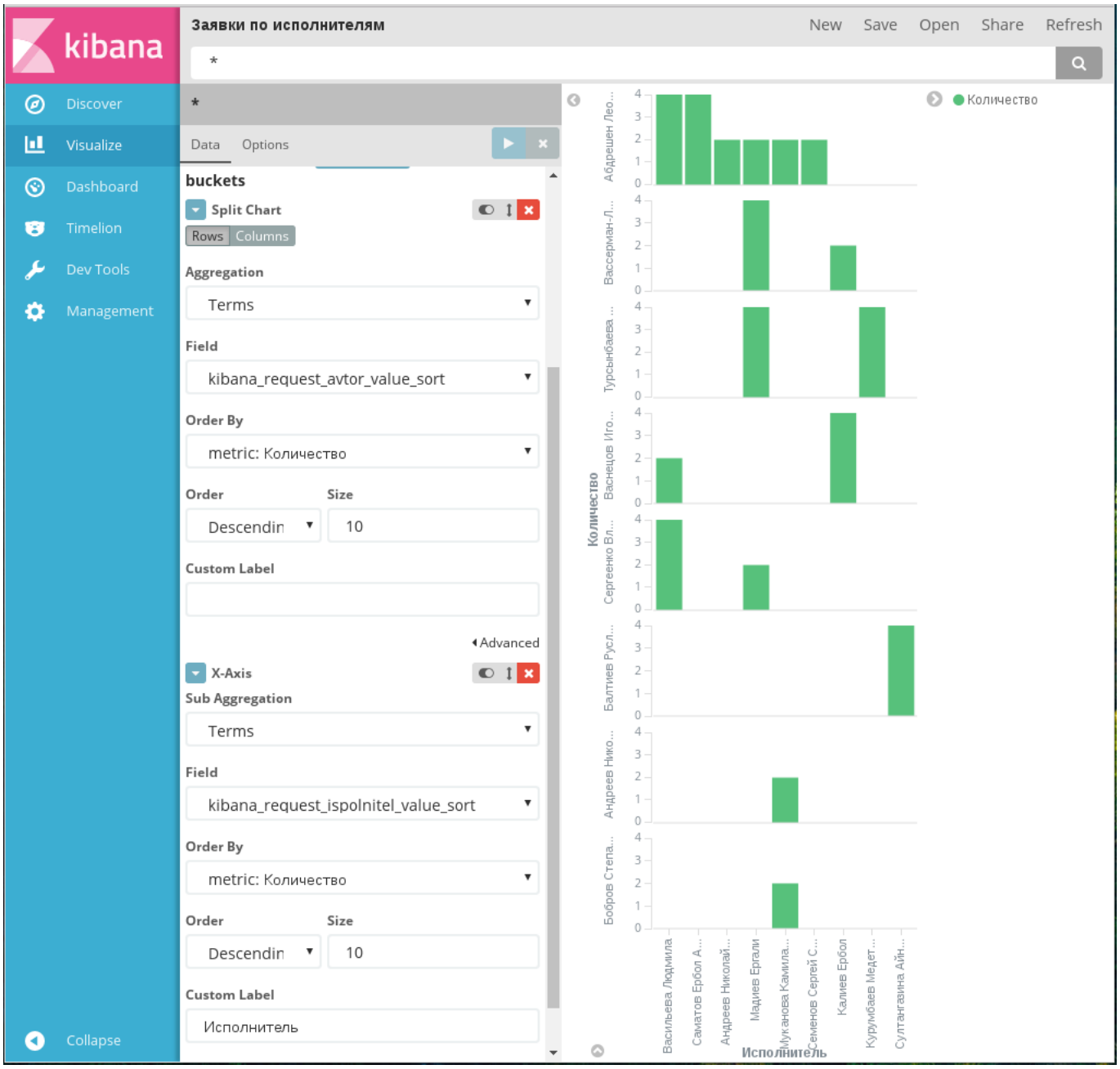


Рис. 1.33: Диаграмма Vertical bar chart, пример 1

На примере 1 показан результат разделения параметров по диаграммам: отображается количество заявок, поданных разными авторами и выполненных разными исполнителями. Для этого исполнители расположены по оси X, а для каждого автора заявок отрисована отдельная диаграмма.

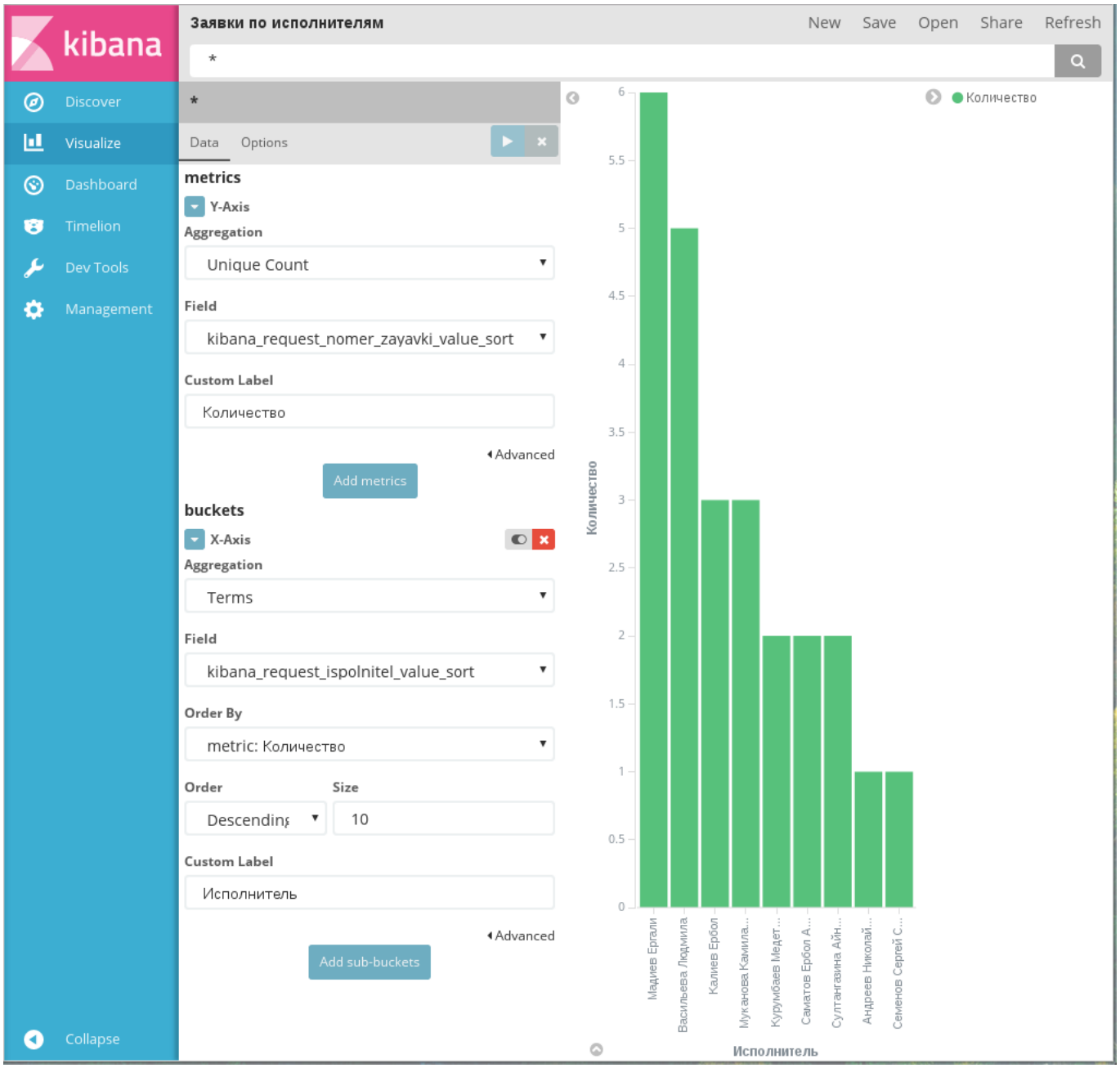


Рис. 1.34: Диаграмма Vertical bar chart, пример 2

На примере 2 показана простая гистограмма, визуализирующая количество заявок, выполненных разными исполнителями.

1.4.3.2.5 Markdown widget

Специфичный тип диаграммы, который не имеет раздела *Data*. В левой части рабочей области располагается поле ввода текста с использованием синтаксиса языка Markdown, в правой части отображается результат разметки текста:

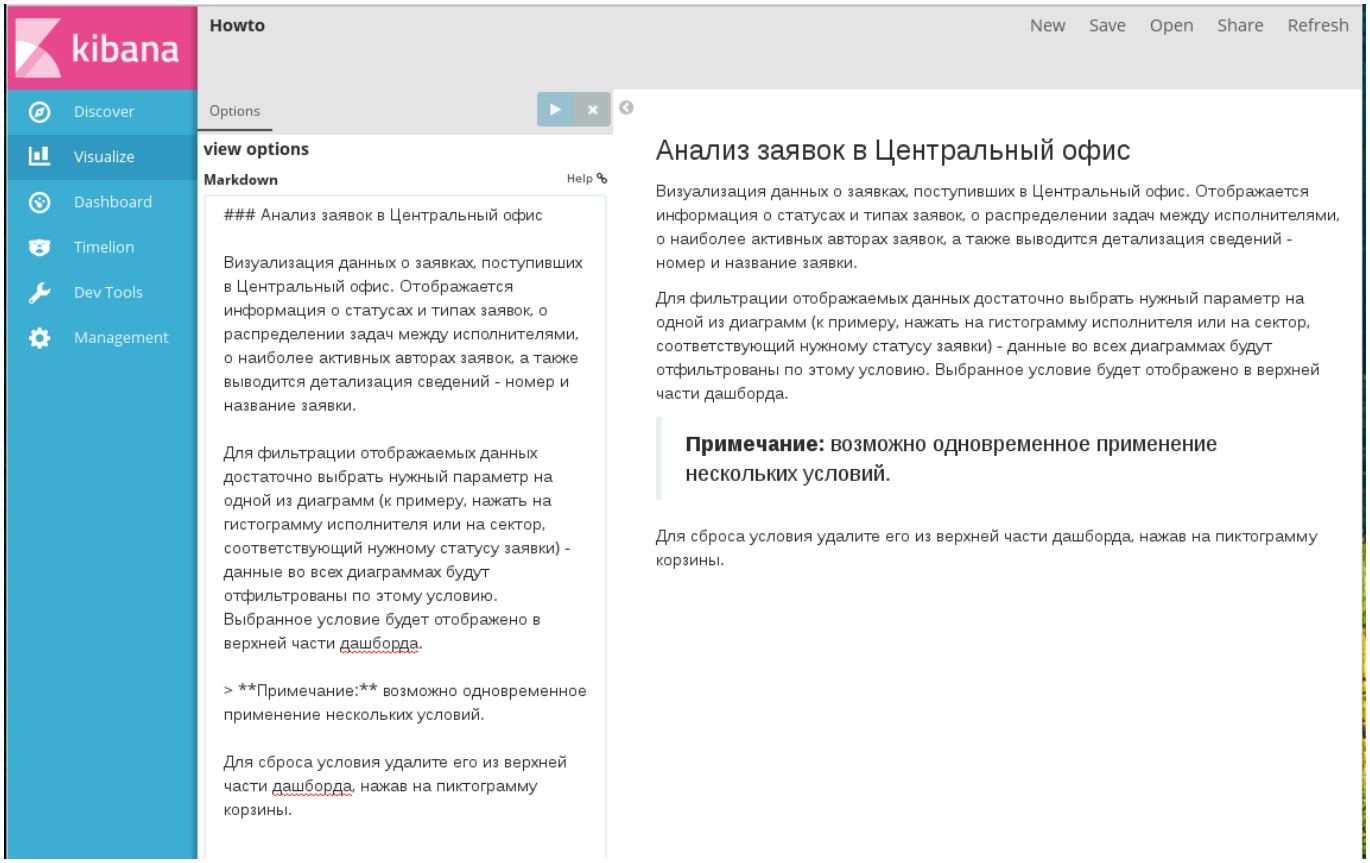


Рис. 1.35: Создание Markdown widget

Эта диаграмма не имеет никаких особых настроек.

1.4.3.2.6 Metric

Диаграмма *Metric* работает только с числовыми данными, поэтому для нее доступны только агрегации типа *Metrics*:

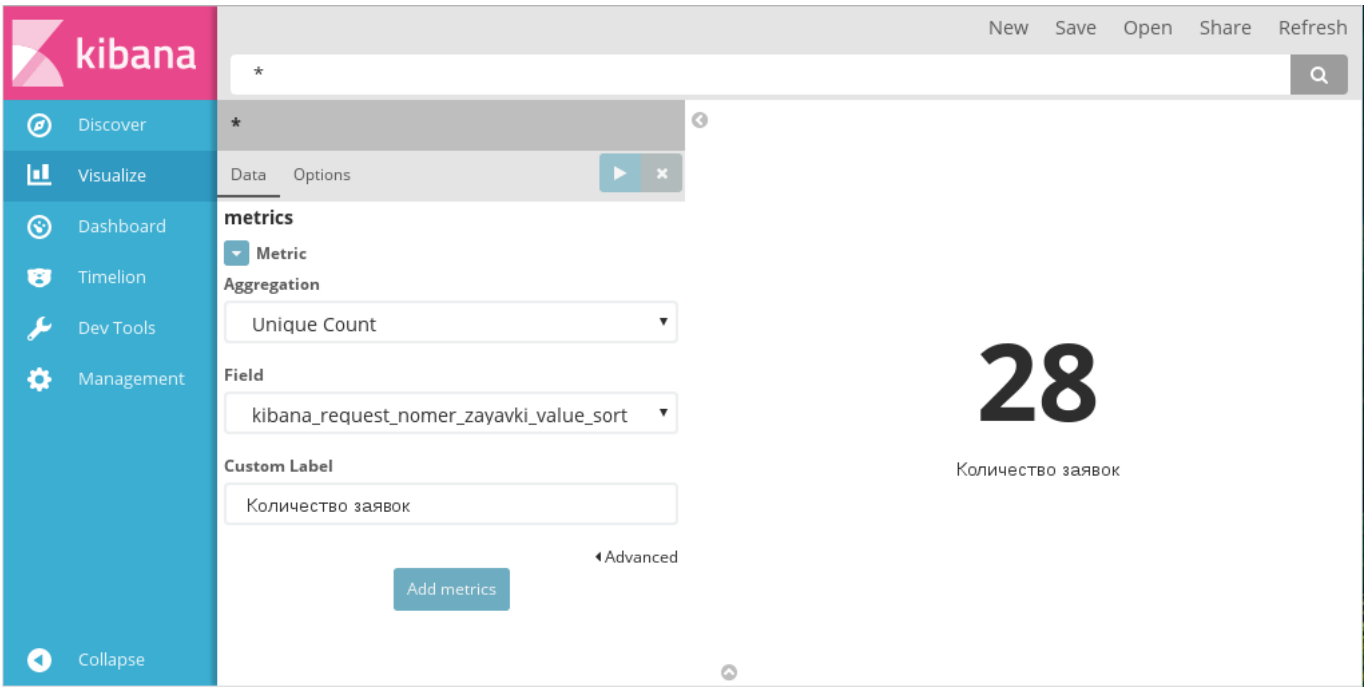


Рис. 1.36: Создание Metrics

Добавить новую метрику можно, нажав на кнопку **Add metrics**. Новая метрика будет добавлена как новое отображаемое число.

Вкладка **Options** для этого типа диаграмм содержит только один параметр - размер шрифта:

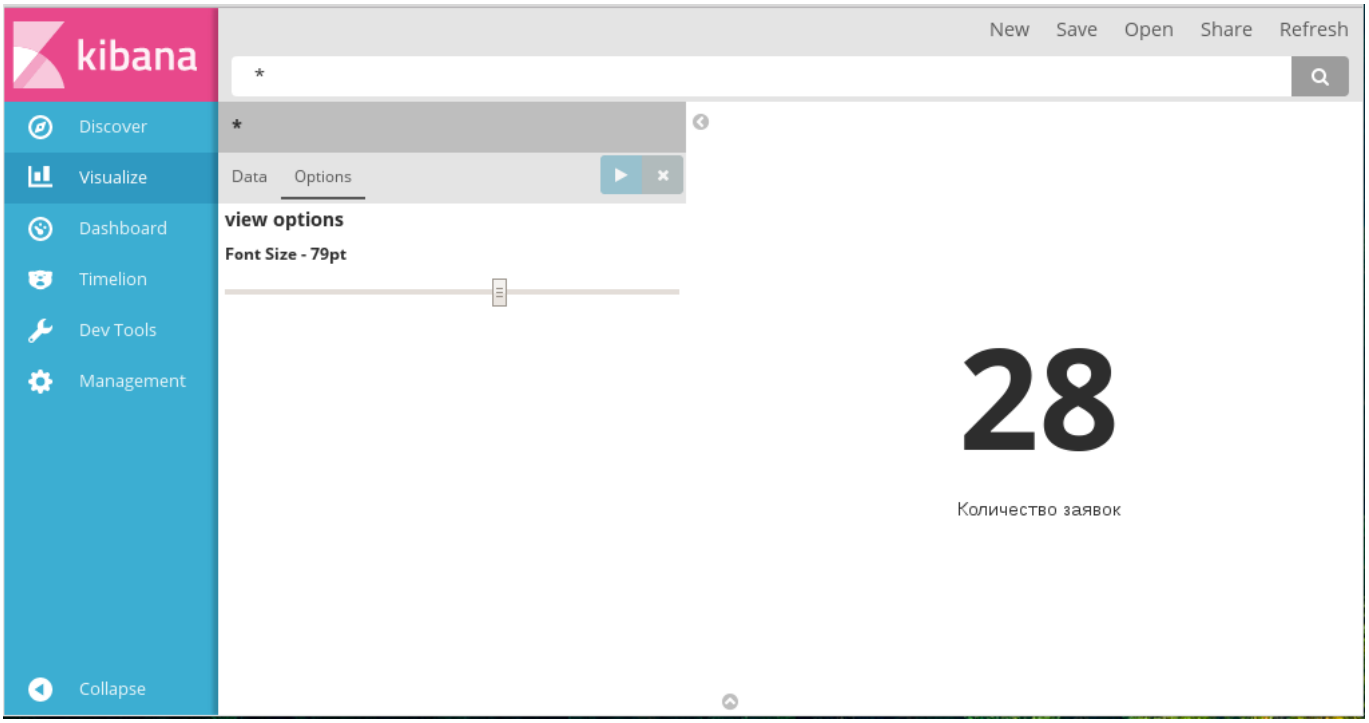


Рис. 1.37: Вкладка «Опции» диаграммы Metrics

1.4.3.2.7 Tag cloud

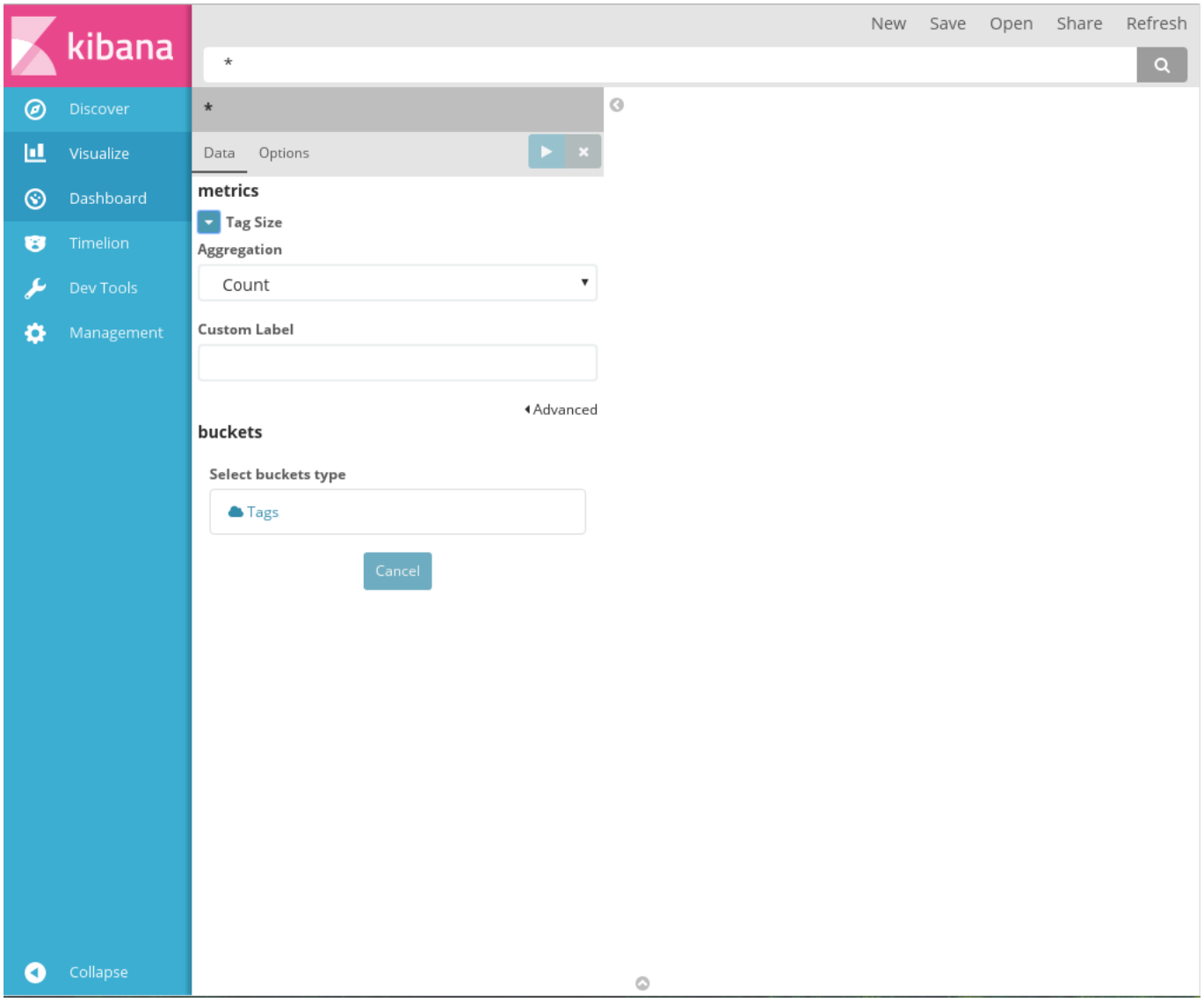


Рис. 1.38: Создание Tag cloud

В диаграмме Tag cloud возможно использование только одной агрегации *Metrics* и только одного, специального способа организации данных в *Buckets* - *Tags*. Добавление новых метрик или новых групп недоступно.

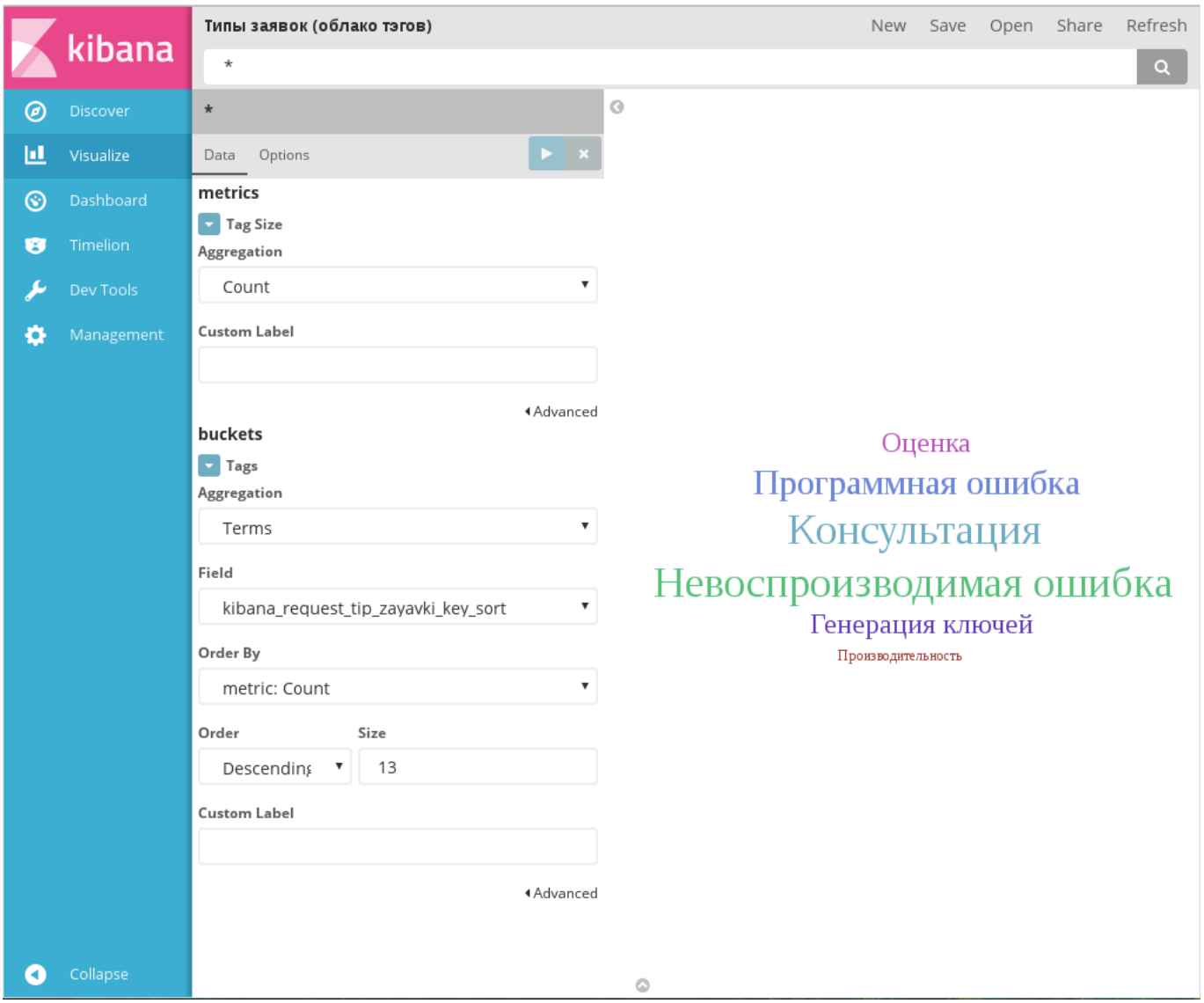


Рис. 1.39: Пример диаграммы Tag cloud

Вкладка **Options** содержит следующие параметры:

- зависимость размера текста от числовой метрики: линейная, логарифмическая или квадратичная;
- ориентация тэгов: горизонтальная, вертикальная или произвольная;
- границы размеров шрифта в тэгах;
- отображать название используемых параметров: чекбокс, по умолчанию выключен.

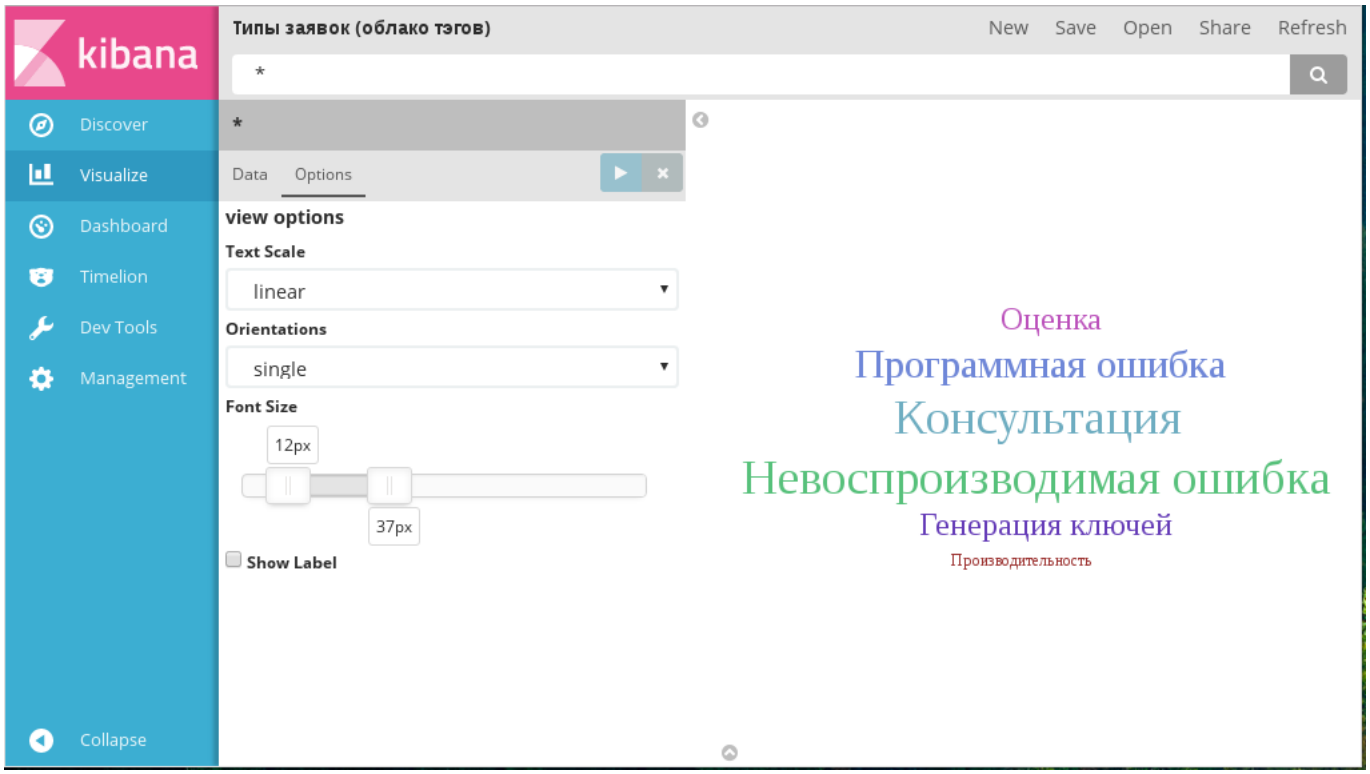


Рис. 1.40: Вкладка «опции» диаграммы Tag cloud

1.4.3.3 Создание дашбордов

Дашборд представляет собой панель, на которой располагаются ранее созданные диаграммы, с широкими возможностями настроек отображения, обновления и публикации.

Создание дашбордов производится в разделе **Dashboard**:

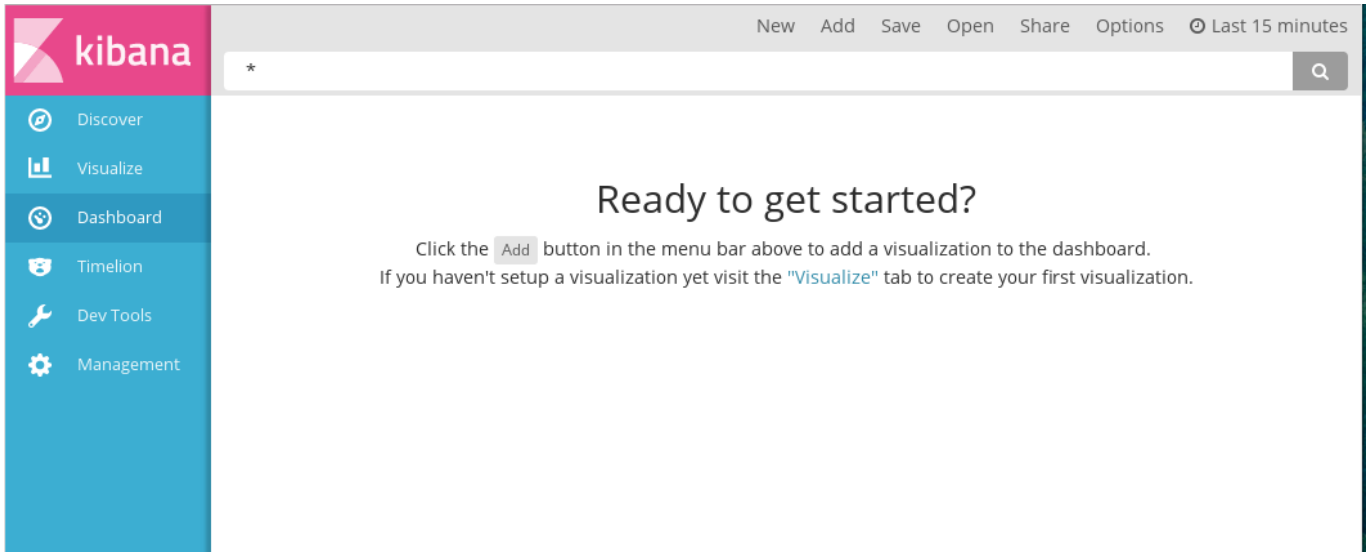


Рис. 1.41: Kibana, раздел Dashboard

Панель меню этого раздела содержит пункты:

- **New** - переход к строке поиска и созданию нового фильтра.
- **Add** - добавить новый дашборд, содержит перечень сохраненных диаграмм и результатов поиска:

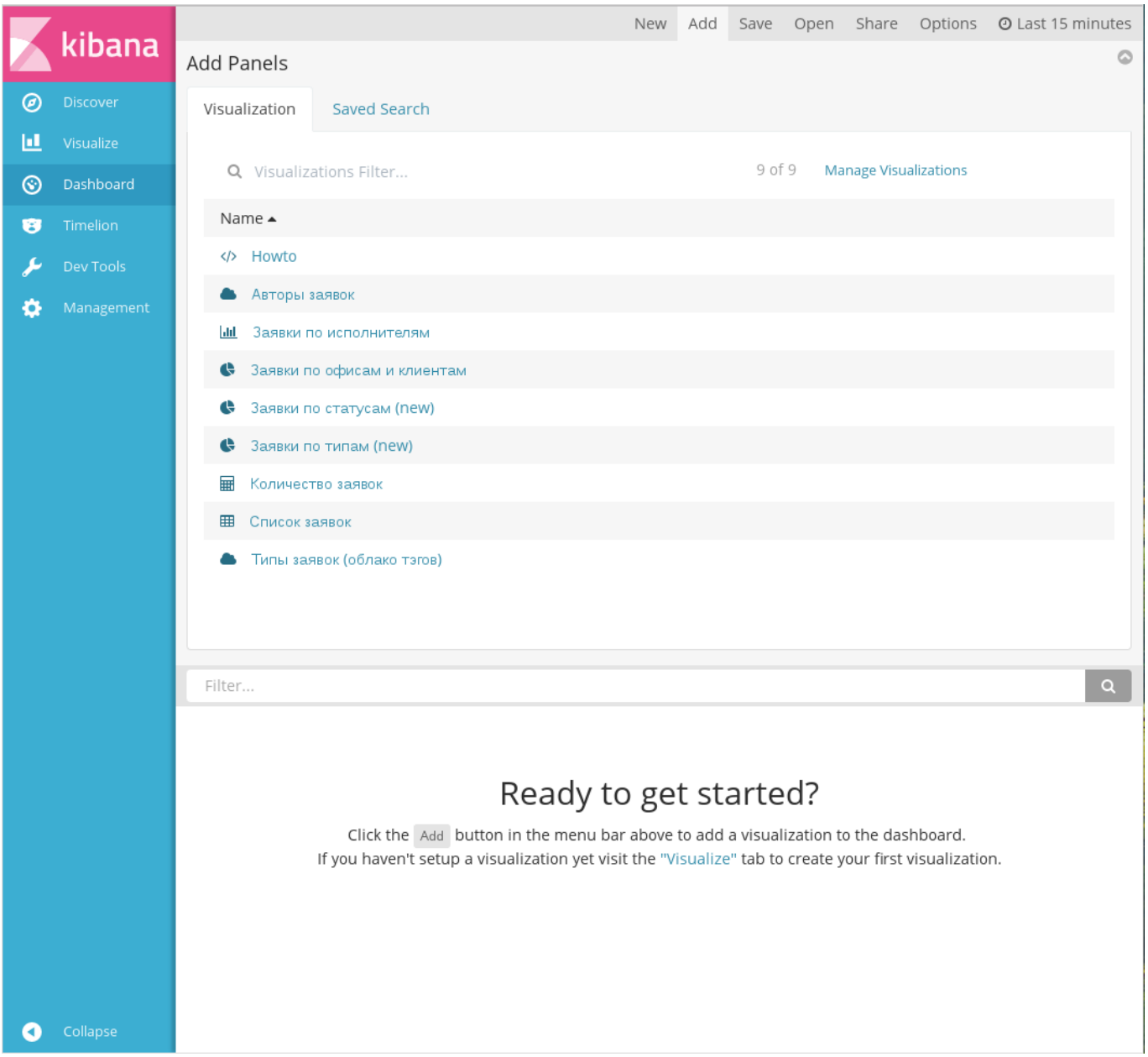


Рис. 1.42: Добавление диаграммы на дашборд

Каждая диаграмма в списке сопровождается пиктограммой, указывающей на тип диаграммы.

- **Save** - сохранить текущий дашборд.
- **Open** - открыть ранее сохраненный дашборд.
- **Share** - настройки публикации дашборда. Доступны только для сохраненного дашборда.

- **Options** - настройка внешнего вида дашборда, содержит единственный чекбокс «Использовать темную тему», по умолчанию выключен.
- **Time range** - настройка режима отображения данных для диаграмм. В самой панели отображается настроенный период. По умолчанию отображаются данные за последние 15 минут. Данная настройка актуальна, если есть необходимость отображения данных в режиме реального времени. Доступна возможность быстрой настройки периода (за сегодня, за эту неделю, за последний год), указания абсолютной (дата и время в формате `YYYY-MM-DD HH:mm:ss.SSS`) или относительной (например, последние 25 минут) настройки.

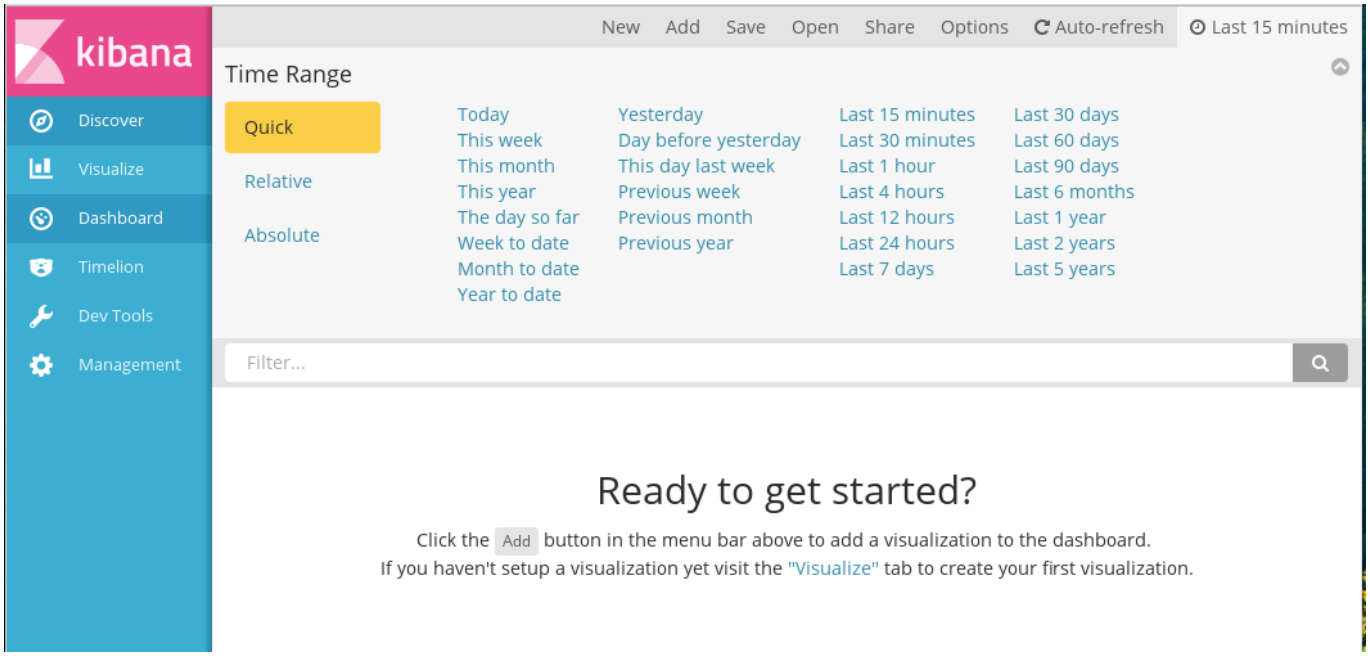


Рис. 1.43: Настройки периода отображения

При переходе к этой настройке в панели меню появляется дополнительный пункт **Auto-refresh**. Он предназначен для настройки интервала обновления диаграмм:

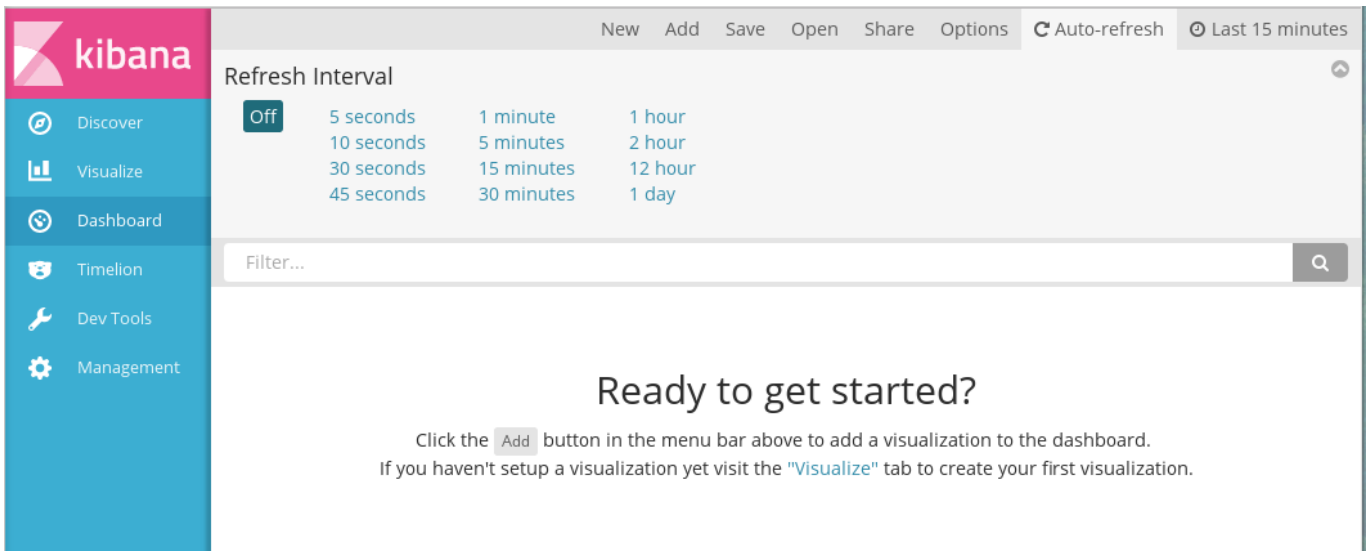


Рис. 1.44: Настройки периода обновления диаграмм

Данная настройка актуальна, если данные, на основе которых построены диаграммы, регулярно обновляются: например, в терминах Synergy, если необходимо отображать актуальные данные реестров, в которых регулярно появляются новые записи.

По умолчанию автообновление выключено.

Для всех диаграмм на дашборде возможно одновременное применение условий для отображаемых данных. Для этого нужно ввести условие в панель поиска, располагающуюся ниже панели меню. Функциональность этой панели для раздела **Dashboards** аналогично панели в разделе **Visualize**.

1.4.3.3.1 Добавление и настройка диаграмм

Для добавления ранее сохраненной диаграммы на дашборд необходимо выбрать пункт меню **Add**. Отобразится список доступных диаграмм (илл. «Добавление диаграммы на дашборд» выше). Необходимо кликнуть на нужную диаграмму - она будет добавлена на дашборд:

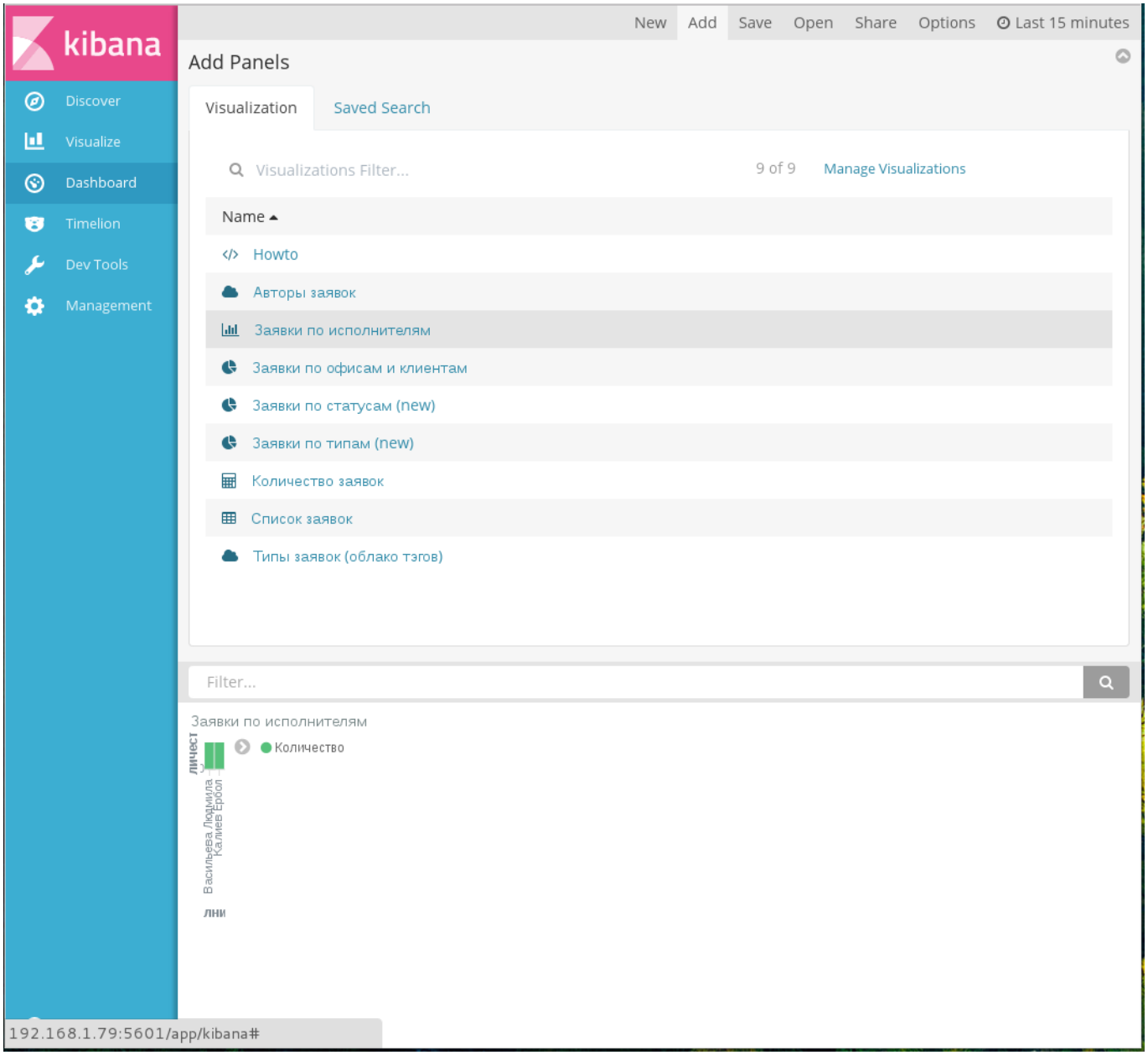


Рис. 1.45: Добавлена панель диаграммы на дашборд

Размер отображаемой диаграммы можно изменить, потянув за левый нижний угол панели диаграммы:

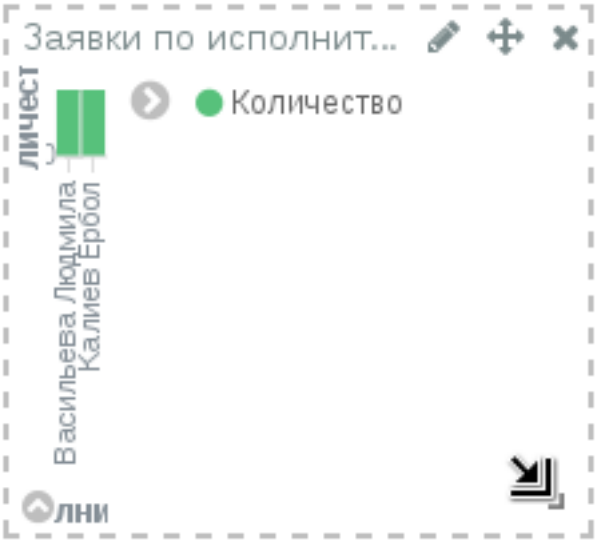


Рис. 1.46: Изменение размера панели диаграммы

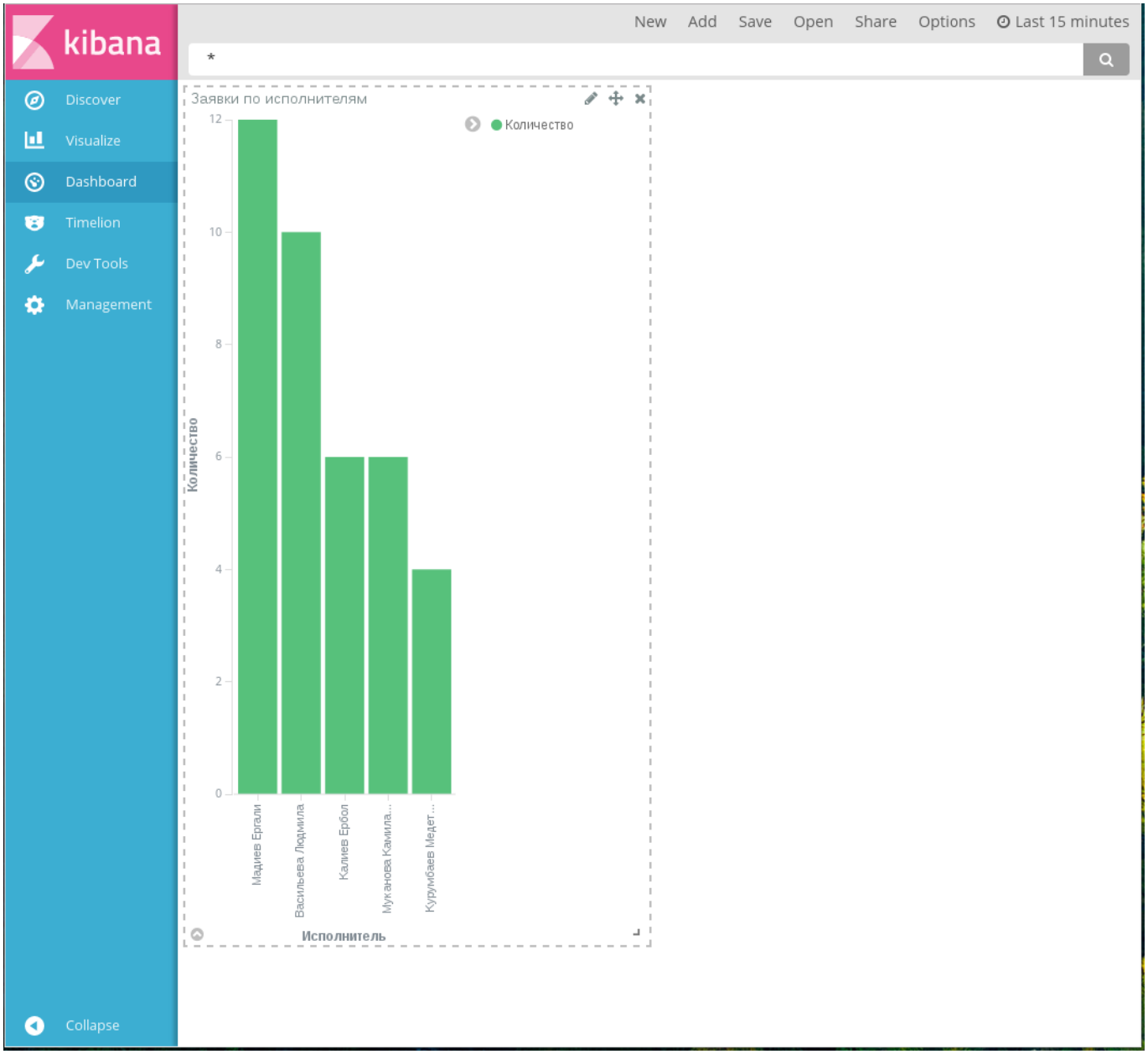
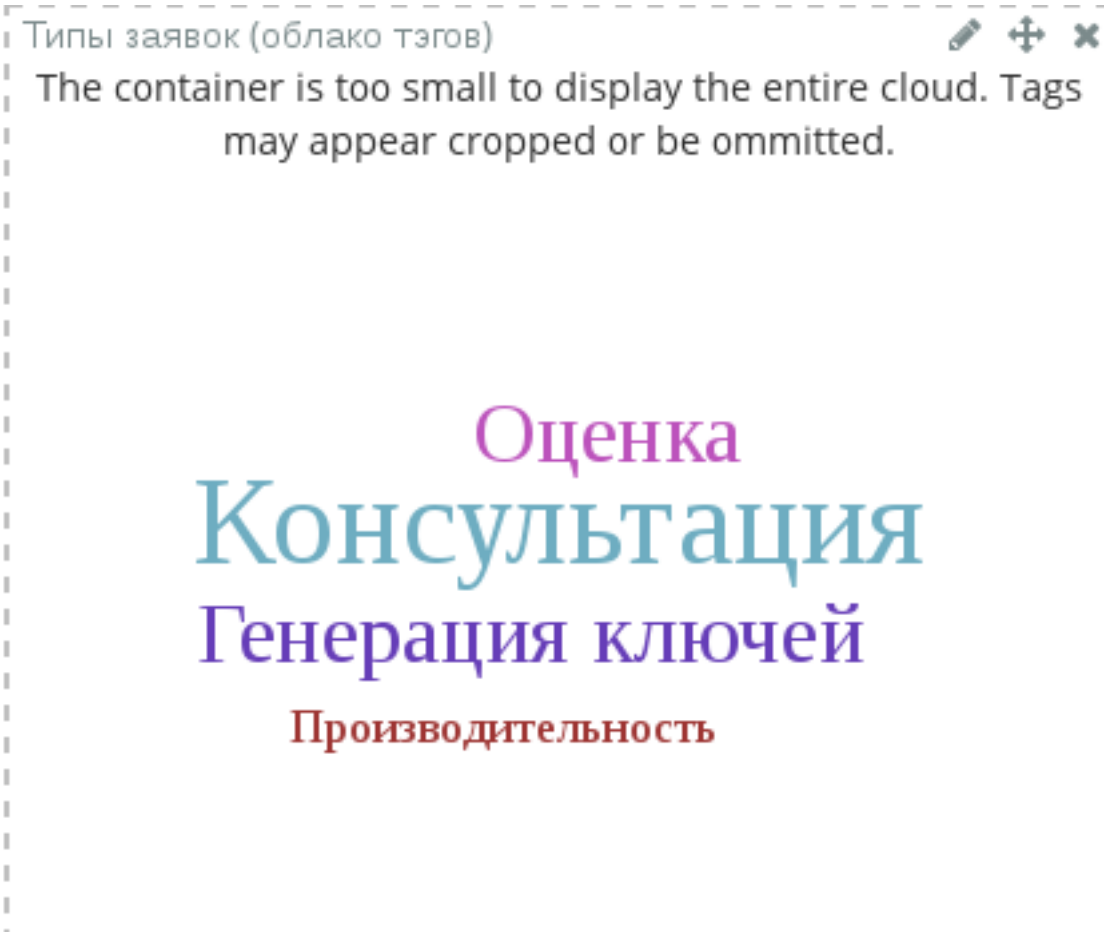


Рис. 1.47: Изменен размер панели диаграммы

В случае, если все данные диаграммы не помещаются на панели, в нее будет добавлен внутренний скролл.

Примечание:

Если на определенном размере панели диаграммы «Облако тэгов» не помещаются все данные, в ней будут отображены только наиболее популярные тэги (столько, сколько возможно уместить на указанном размере панели), и будет отображен текст, предупреждающий, что для отображения всех данных необходимо увеличить размер панели диаграммы:



На панели диаграммы отображаются пиктограммы управления:

- - изменить диаграмму (переход к настройкам отображаемых данных диаграммы в разделе **Visualize**);
- - переместить панель диаграммы на дашборде;
- - удалить панель диаграммы с дашборда;
- - изменить размер панели диаграммы;
- - отобразать/свернуть источники данных в виде таблицы, запроса или исходных данных Elasticsearch, а также статистику запроса данных для этой диаграммы;
- - отобразить/свернуть легенду.

Количество диаграмм, располагаемых на дашборде, не ограничено, наложение диаграмм друг на друга не допускается.

Пример готового дашборда:

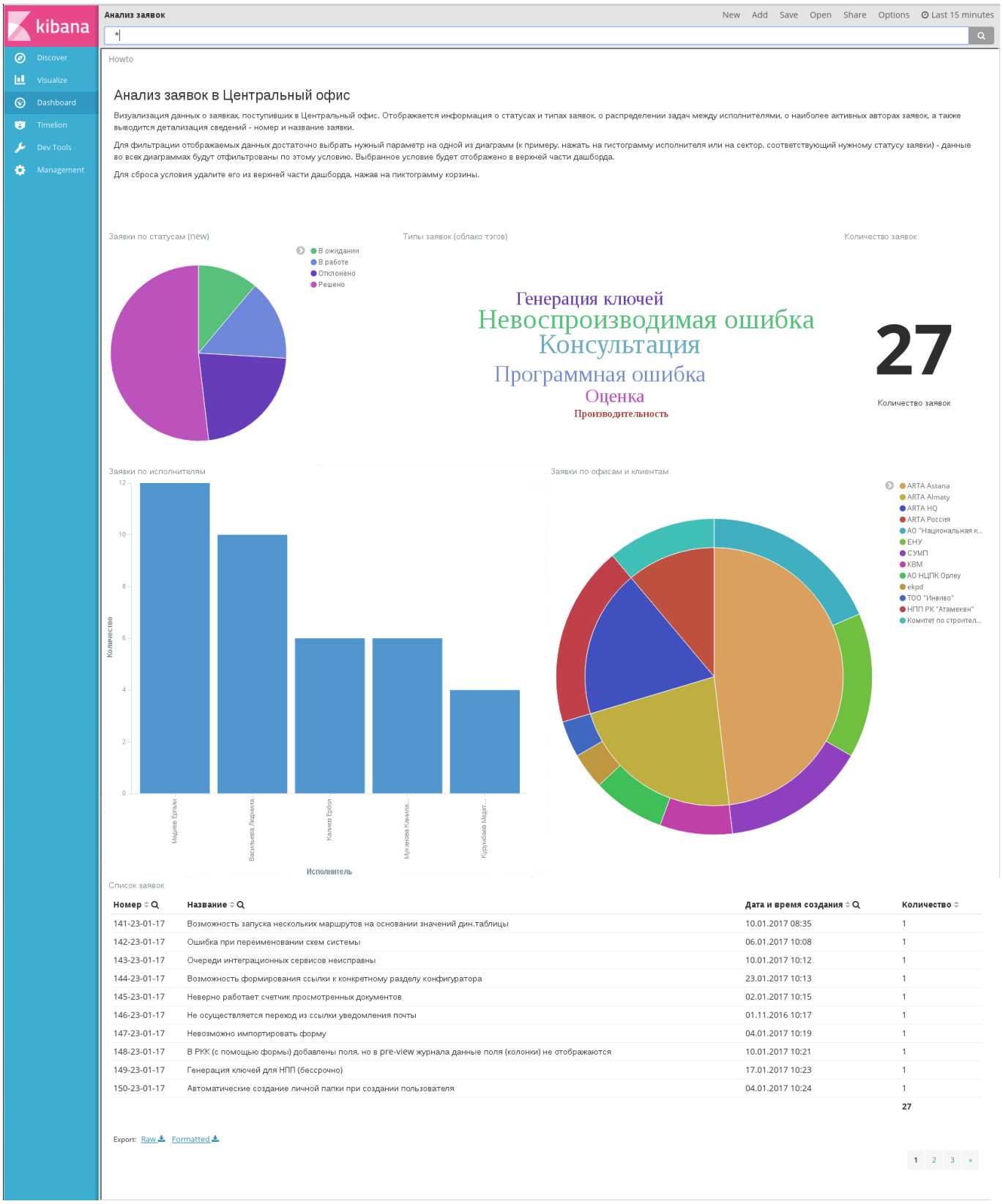


Рис. 1.48: Пример готового дашборда в режиме редактирования

Типы использованных примеров диаграмм (перечислены сверху вниз, слева направо):

1. *Markdown widget*
2. *Pie chart*
3. *Tag cloud*
4. *Metric*
5. *Vertical bar chart*
6. *Pie chart*
7. *Data table*

Примечание:

Применение фильтров ко всем диаграммам на дашборде дает корректные результаты только в том случае, если коды используемых полей полностью совпадают (в том числе постфиксы). В случае, если необходимо отображение данных из нескольких форм, имеющих сквозные параметры (например, параметр «Статус»), необходимо, чтобы коды компонентов, соответствующих этому параметру, совпадали на всех формах, а в диаграммах использовалось одно и то же поле с учетом постфикса.

1.4.3.3.2 Публикация дашборда

Kibana предоставляет способы публикации дашборда как интерактивной диаграммы или как снимка его состояния на момент публикации (`shapshot`). Публикация производится в меню **Share**:

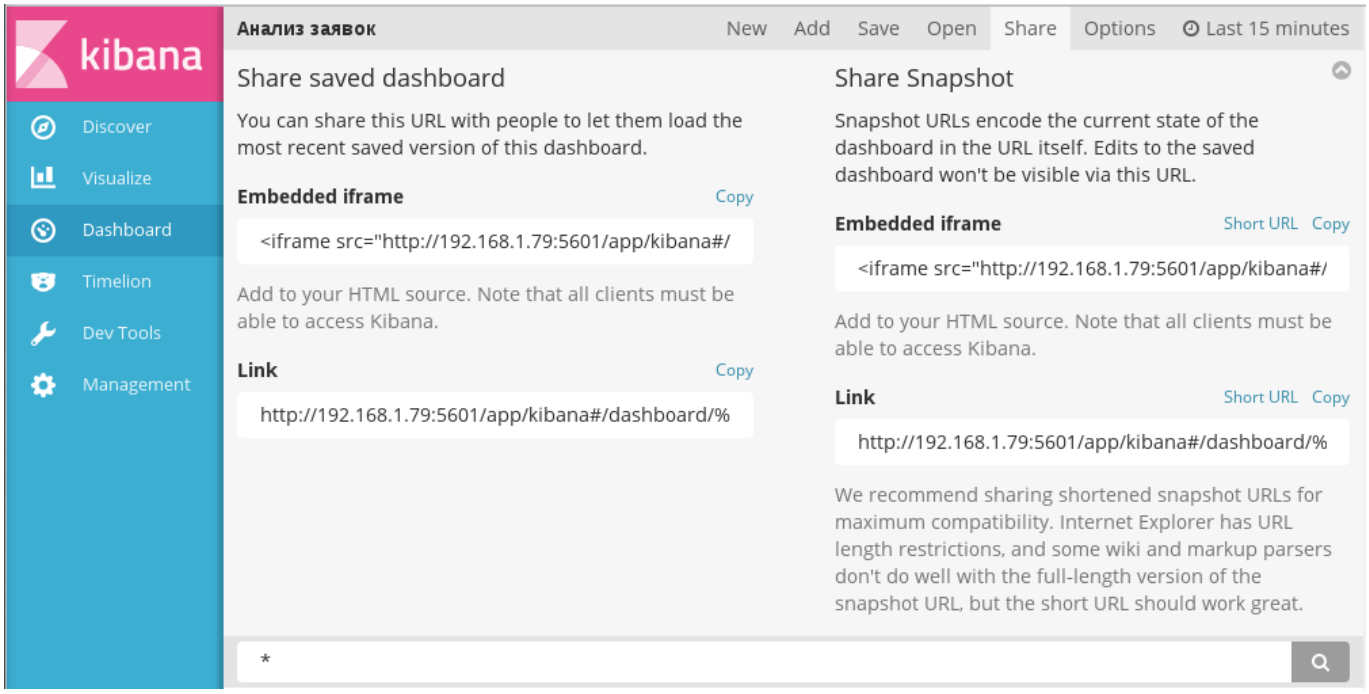


Рис. 1.49: Пункт меню «Share»

Встраивание как дашборда, так и его снимка возможно двумя способами:

1. как фрейма `html` - код для вставки содержится в поле **Embedded iframe**;

2. как ссылки - URL страницы содержится в поле **Link**.

   Примечание:

   По URL, автоматически генерируемому Kibana, пользователям предоставляется дашборд в режиме редактирования, с правом доступа ко всем разделам Kibana. Для того, чтобы предоставить пользователям доступ к дашборду только в режиме просмотра, необходимо в URL ссылки добавить параметр:

   ```
   &embed=true
   ```

Один из способов публикации дашборда в Synergy - добавление его как внешнего модуля. При этом каждый дашборд должен быть оформлен как отдельный внешний модуль. В качестве адреса приложения необходимо использовать URL дашборда.

Другой способ - включение фрейма с дашбордом в пользовательский компонент. В этом случае в качестве HTML-кода необходимо использовать код из поля *Embedded iframe*.

Примечание:

По умолчанию в код фрейма включены границы 800x600 пикселей. Для того, чтобы дашборд занимал все доступное место, необходимо изменить эти параметры:

```
height="100%" width="100%"
```

Параметр `embed=true`, означающий доступ к дашборду только в режиме просмотра, включается Kibana по умолчанию.

Примечание:

Обратите внимание, что для того, чтобы дашборд был доступен пользователю, у него должен быть доступ к серверу, на котором запущена Kibana.

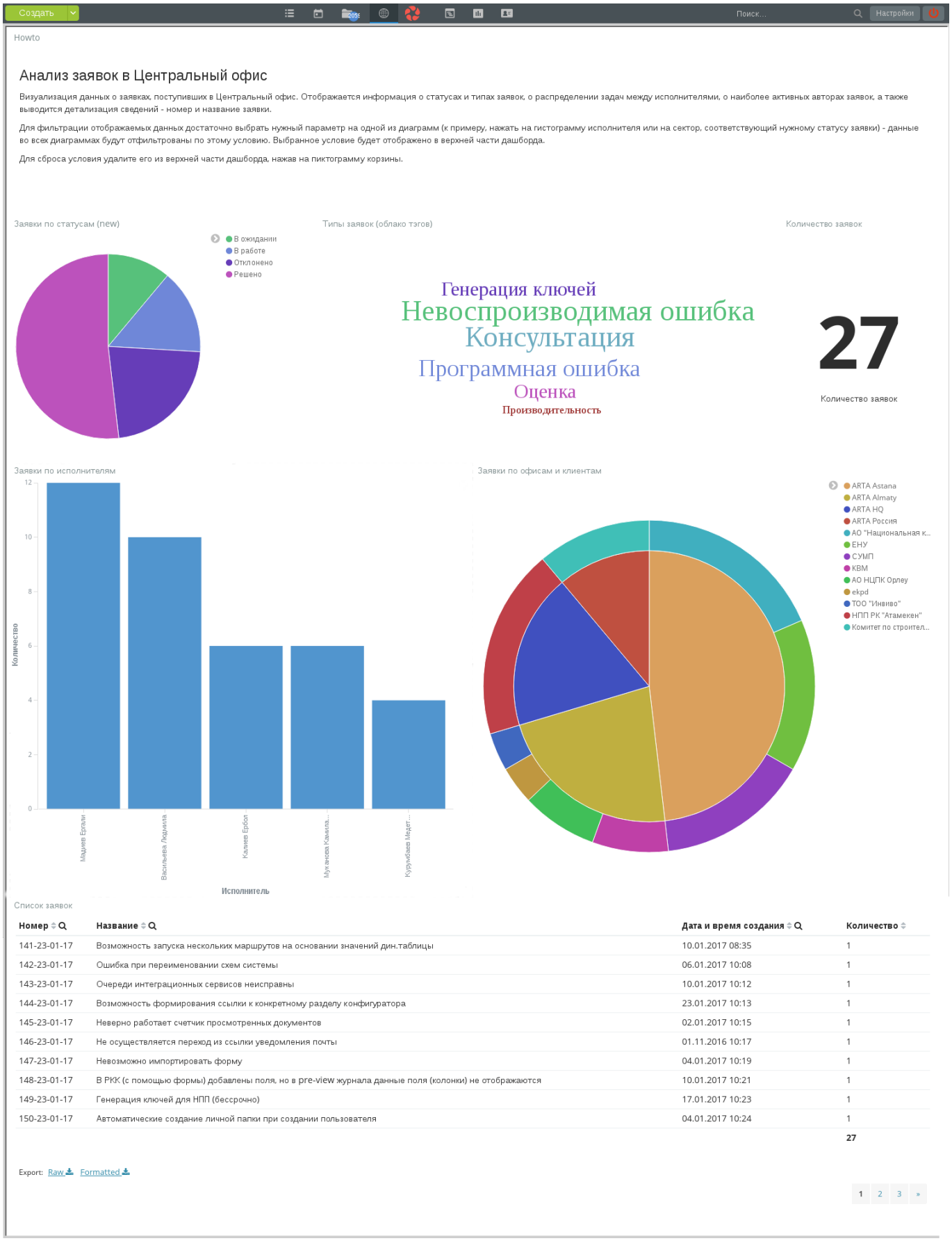
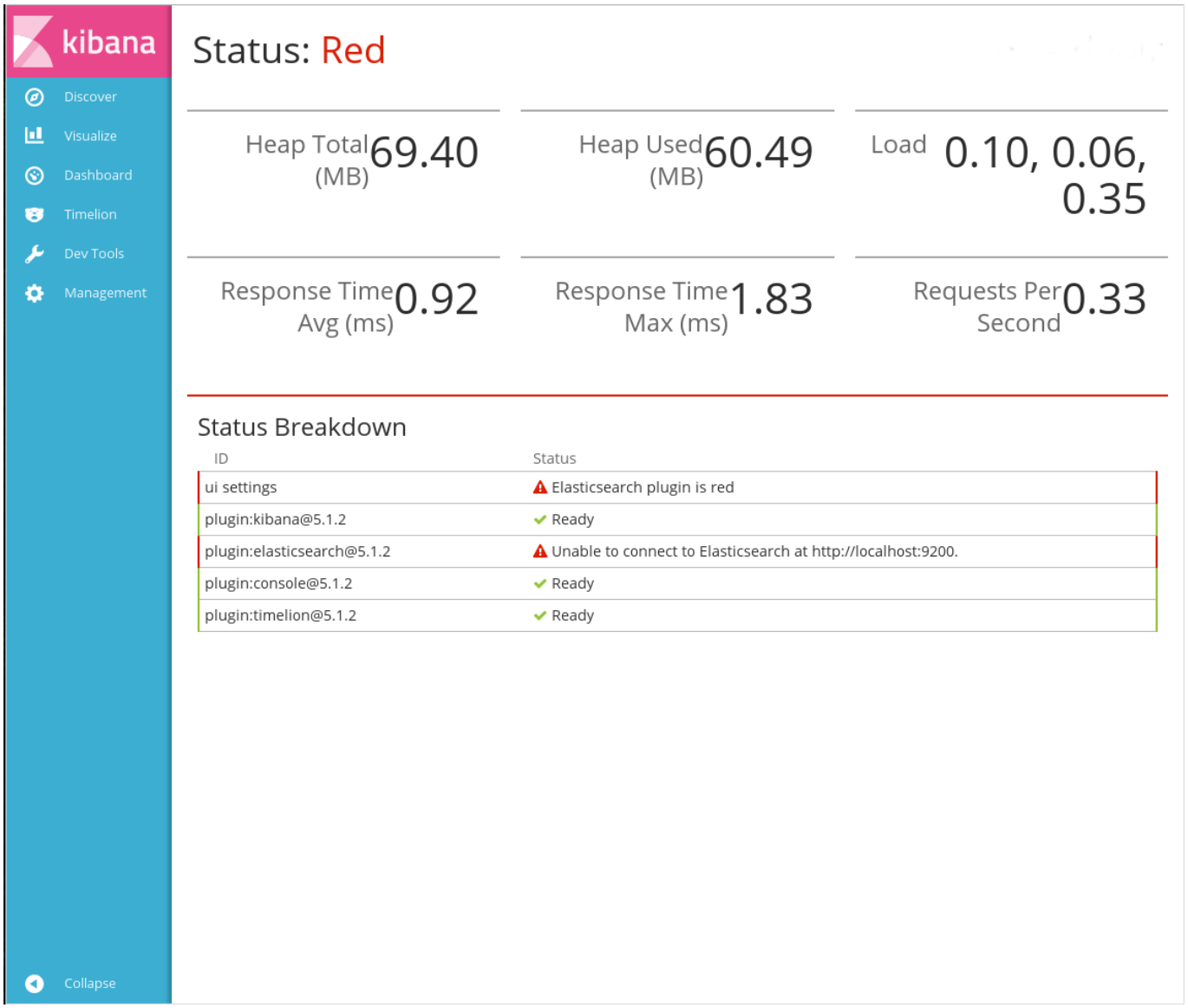


Рис. 1.50: Пример дашборда, опубликованного как внешний модуль

1.4.3.4 Возможные проблемы и способы их решения

1. **Status: Red**



Ошибка связана с невозможностью доступа к сервису Easticsearch (ES). При ее возникновении сначала необходимо проверить статус ES. Для этого в консоли сервера, на котором запущен ES, выполните команду:

```
#/etc/init.d/elasticsearsh status
```

Результатом выполнения команды должно быть сообщение:

```
[ ok ] elasticsearch is running.
```

Другой способ - проверить статус ES непосредственно:

```
#curl localhost:9200
```

`localhost:9200` - это адрес ES по умолчанию.

Вывод должен быть таким:

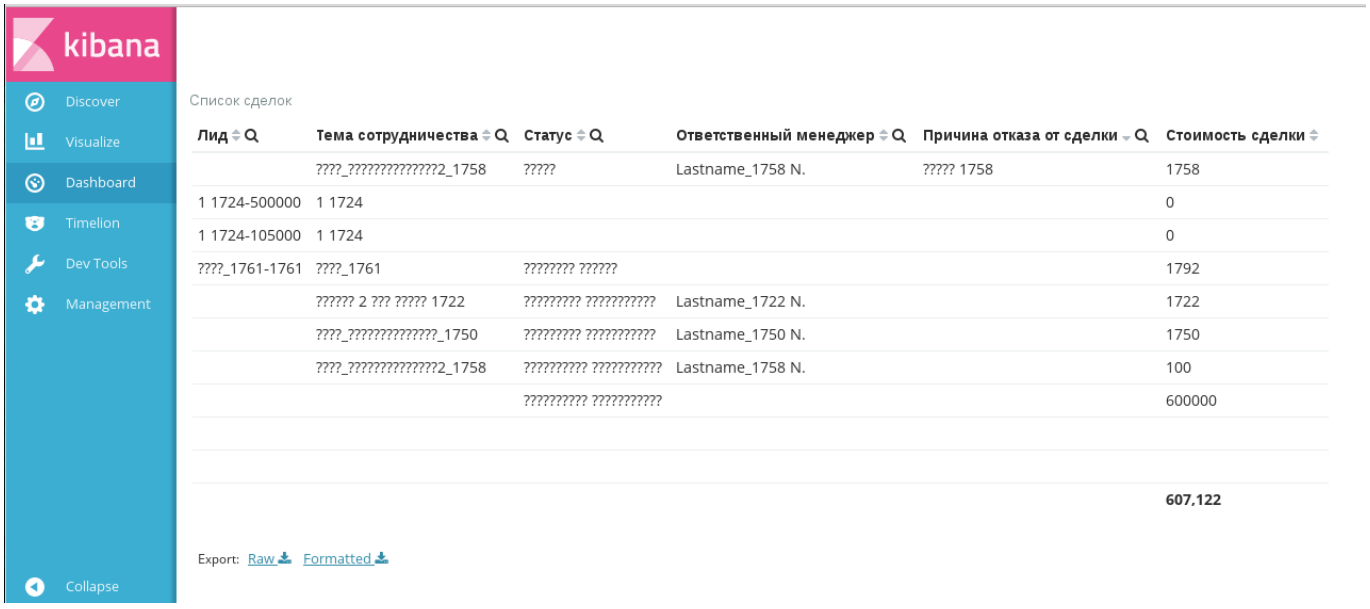
```
{
  "name" : "RFSWkzt",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "r67YbmerQvyNHdxlzDIt3A",
  "version" : {
    "number" : "5.1.2",
    "build_hash" : "c8c4c16",
    "build_date" : "2017-01-11T20:18:39.146Z",
    "build_snapshot" : false,
    "lucene_version" : "6.3.0"
  },
  "tagline" : "You Know, for Search"
}
```

Если при ошибке **Status: Red** результат выполнения этих команд совпадает с ожидаемым, это значит, что сервис Elasticsearch запустился, но еще не успел обработать все данных в индексах. Ошибка может возникать, если в ES загружен большой объем данных. В этом случае рекомендуется дать ES время на полную загрузку (до 30 минут).

Если спустя время статус Kibana не изменился, или в результате выполнения команды `curl` появляется сообщение о невозможности подсоединения к серверу, значит, необходимо перезапустить ES, выполнив команду:

```
#etc/init.d/elasticsearsh restart
```

1. **Русскоязычные данные импортировались в ES как «????»**



При возникновении такой проблемы рекомендуется:

2.1. Остановить запущенные сервисы, выполнив команды:

```
#/etc/init.d/arta-synergy-jboss stop

#/etc/init.d/kibana stop

#/etc/init.d/elasticsearch stop
```

2.2. Перейти к настройке локали сервера:

```
#dpkg-reconfigure locales
```

2.3. В качестве локали и локали по умолчанию установить локаль `en_US.UTF-8`.

2.4. Запустить Synergy и Elasticsearch:

```
#/etc/init.d/arta-synergy-jboss start

#/etc/init.d/elasticsearch start
```

2.5. Выполнить полную переиндексацию данных форм в административном приложении Synergy (Управление индексом форм).

2.6. Запустить Kibana:

```
#/etc/init.d/kibana start
```

1. **Записи в реестре не отображаются в Synergy, но видны в результатах поиска по реестру и в данных Kibana**

3.1. Остановить все сервисы:

```
#/etc/init.d/arta-synergy-jboss stop

#/etc/init.d/kibana stop

#/etc/init.d/elasticsearch stop
```

3.2. Удалить существующие индексы ES:

```
rm -r /var/lib/elasticsearch/nodes
```

3.3. Запустить Synergy и Elasticsearch:

```
#/etc/init.d/arta-synergy-jboss start

#/etc/init.d/elasticsearch start
```

3.4. Выполнить полную переиндексацию данных форм в административном приложении Synergy (Управление индексом форм).

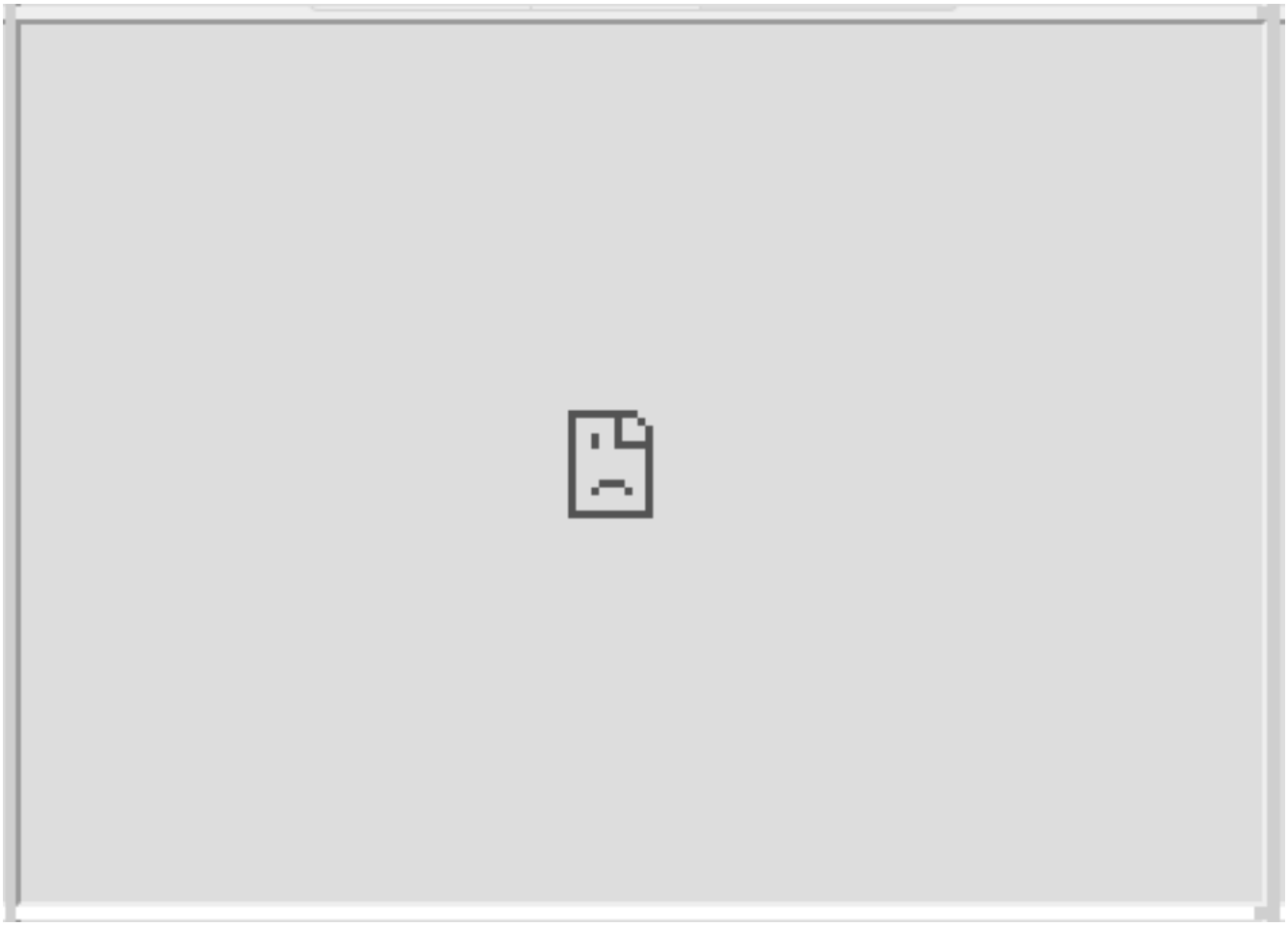
3.5. Запустить Kibana:

```
#/etc/init.d/kibana start
```

1. **При публикации дашбордов/диаграмм пользователи видят слева панель Kibana**

   Диаграмма/дашборд были опубликованы в режиме редактирования. Чтобы избежать такой проблемы, необходимо в параметры URL-ссылки или HTML-фрейма добавить параметр `embed=true`. Этот параметр означает, что данные должны публиковаться в режиме просмотра.

1. **При настроенной ссылке на дашборд/коду вставки HTML диаграммы не отображаются.**



   Проблема возникает в том случае, если отключена Kibana. Необходимо проверить ее состояние:

   - в браузере перейти по адресу `<server.host>:<server.port>`, где `<server.host>` - адрес сервера, на котором запущена Kibana, а `<server.port>` - номер порта (по умолчанию используется порт 5601)

   - для проверки статуса из консоли сервера выполните команду:

     ```
     #/etc/init.d/kibana status
     ```

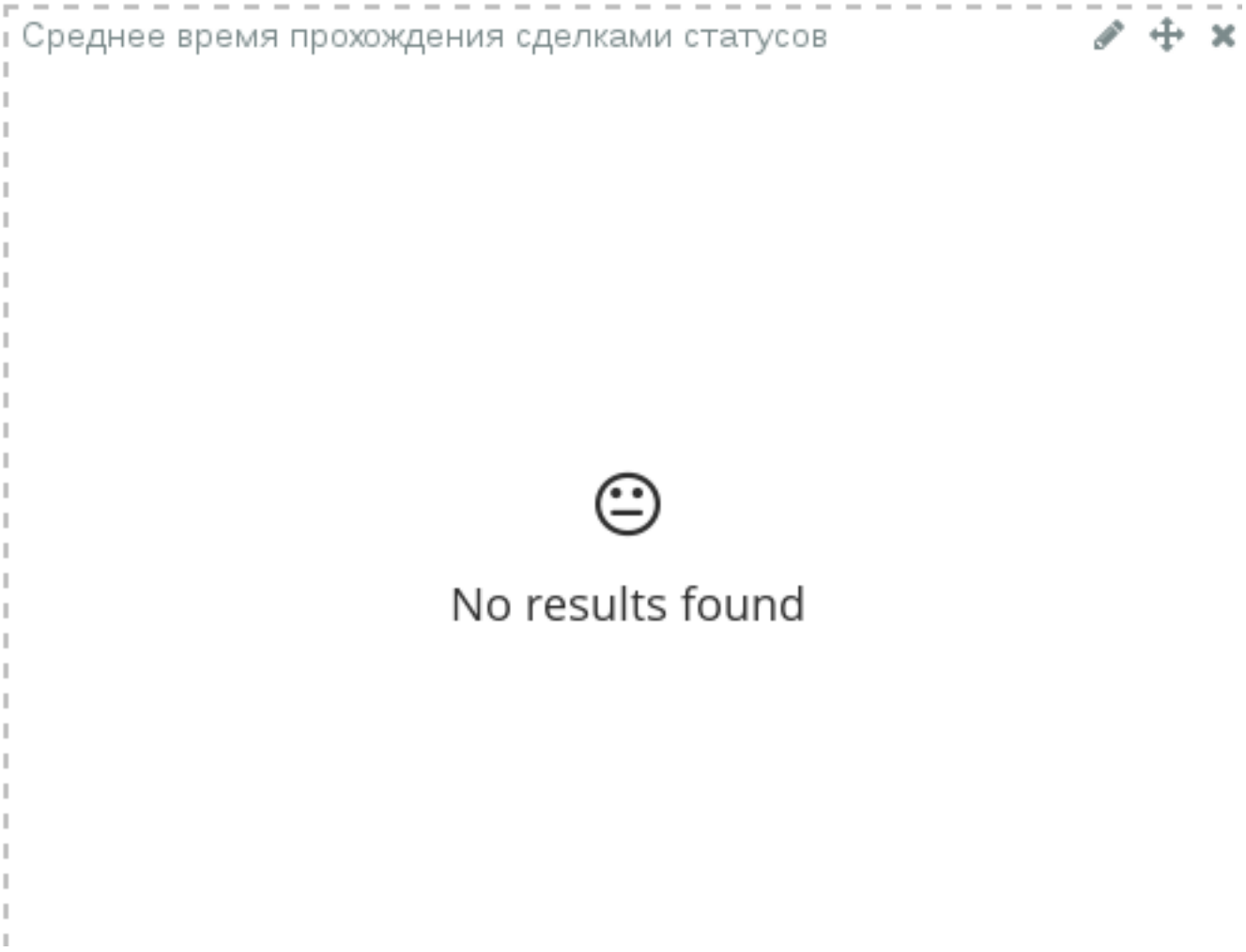
     Результатом выполнения должно быть сообщение:

```
kibana is running
```

В случае, если Kibana не запущена или вернула ошибку, необходимо ее перезапустить, выполнив команду:

```
#/etc/init.d/kibana restart
```

1. **Вместо диаграммы отображается сообщение «No results found»**



Kibana отображает в диаграммах только заполненные исходные данные. Диаграмма может не отображаться в двух случаях:

6.1. К диаграмме было применено условие, результаты которого не используются в диаграмме.

6.2. В исходных данных (в формах и реестрах Synergy) нет ни одного документа, данные из которого должны отобразиться в диаграмме.

Данное поведение не является ошибкой ни Kibana, ни Synergy, и при обновлении данных диаграммы отобразятся автоматически.

1. **Диаграмма ссылается на недоступное поле**

Проблема может возникнуть при импорте диаграмм из внешних источников (в том числе при установке бизнес-приложений на базе Synergy, использующих Kibana), и чаще всего связана с отсутствием в шаблоне индекса числового поля с постфиксом `_double`. Проверить это можно, перейдя к настройке диаграммы: в агрегациях по отсутствующим полям отображается ошибка.

Индекс для поля `_double` создается только в том случае, если из содержимого поля удалось выделить число. То есть если во всех документах поле не заполнено, то и индекс с типом `double` для него создан не будет.

Для того, чтобы исправить проблему, нужно хотя бы в одном документе по форме заполнить числовое поле, на отсутствие которого ссылается Kibana, после чего необходимо обновить шаблон индекса для диграммы, нажав на кнопку «Обновить» (раздел **Management** - **Index patterns**).

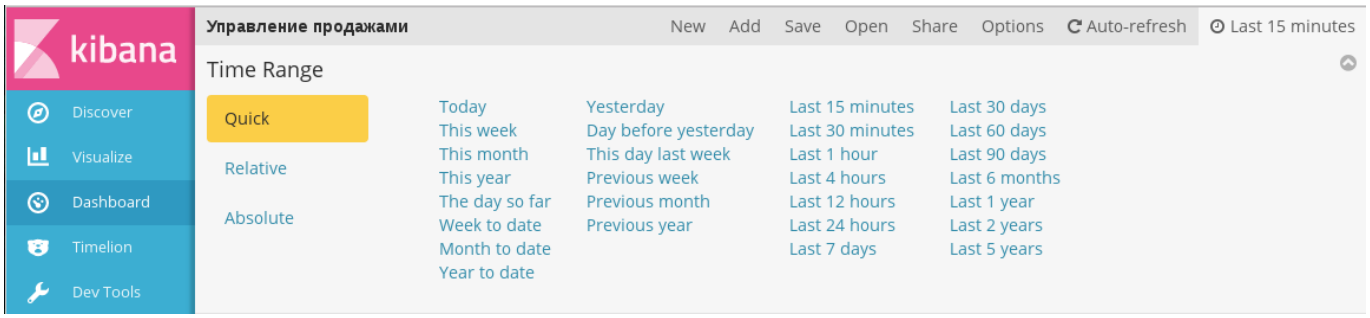
Для того, чтобы предотвратить возникновение такой ошибки, рекомендуется для каждого поля, которое будет использовано в диаграммах Kibana как числовое, сохранять значение по умолчанию в редакторе форм.

1. **При обновлении данных в Synergy не обновляются соответствующие диаграммы**

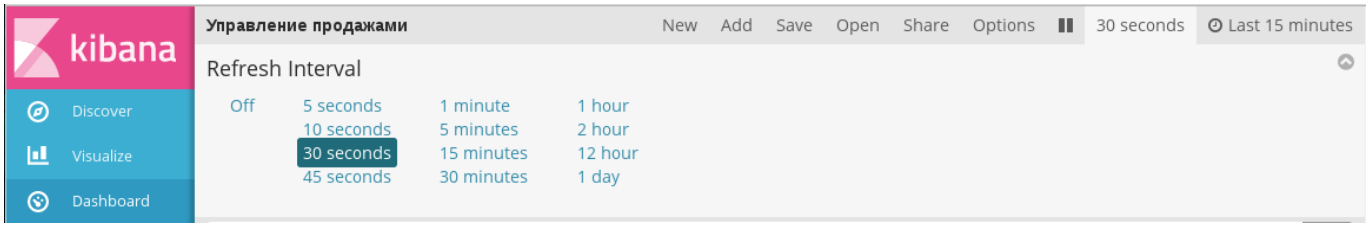
   Проблема возникает из-за отсутствия или неправильной настройки периода обновления данных. Проверить эти настройки можно в Kibana:

   8.1. Открыть дашборд, требующий настройки автообновления.

   8.2. В панели меню выбрать настройки периода времени:



   8.3. Выбрать пункт **Auto-refresh**:



   8.4. Настроить **Refresh Interval** - периодичность обновления данных.

   Примечание:

Не рекомендуется устанавливать периодичность обновления меньше, чем 30 секунд, поскольку на стороне пользователя может возникнуть проблема фризов (секундных подергиваний или застываний изображения).

1. **После проведения индексации форм в Kibana отсутствуют данные форм**

   Возможно, не был переключен индексатор данных форм. По умолчанию в Synergy для индексации данных используется система Lucene. Переключение индексаторов между Lucene и ES осуществляется в файле `/opt/synergy/jboss/standalone/configuration/arta/esb/formIndex.xml`. Необходимо убедиться, что содержимое первого тэга `<handler>`, соответствующее Lucene, закомментировано, и раскомментировать содержимое тэга после `<elastic>` (относящееся к ES):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<system xmlns="http://www.arta.kz/xml/ns/ai"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.arta.kz/xml/ns/ai http://www.arta.kz/xml/ns/ai/ai. ↩
            xsd">

    <name>synergy</name>
    <clusterName>synergy</clusterName>
    <host>127.0.0.1</host>
    <port>9001</port>
    <master>false</master>
    <local>false</local>
    <seed>false</seed>

    <handlers>
        <!--handler>
            <name>indexForm</name>
            <classname>kz.arta.synergy.indexator.forms.IndexFormHandler</classname>
            <max-number>30</max-number>
        </handler>
        <handler>
            <name>deleteForm</name>
            <classname>kz.arta.synergy.indexator.forms.DeleteIndexFormHandler</classname>
            <max-number>10</max-number>
        </handler>
        <handler>
            <name>searchForms</name>
            <classname>kz.arta.synergy.indexator.forms.SearchFormHandler</classname>
            <max-number>30</max-number>
        </handler>
        <handler>
            <name>searchRegistries</name>
            <classname>kz.arta.synergy.indexator.forms.SearchRegistryHandler</classname>
            <max-number>30</max-number>
        </handler>
        <handler>
            <name>indexInfo</name>
            <classname>kz.arta.synergy.indexator.forms.IndexInfoHandler</classname>
            <max-number>1</max-number>
            <properties>configuration.path=/opt/synergy/jboss/standalone/configuration/arta ↩
                /formIndex.xml</properties>
        </handler-->

        <!--elastic-->

        <handler>
            <name>indexForm</name>
```

```
            <classname>kz.arta.synergy.indexator.elastic.ElasticIndexFormHandler</classname ↩
                >
            <max-number>30</max-number>
        </handler>
        <handler>
            <name>deleteForm</name>
            <classname>kz.arta.synergy.indexator.elastic.ElasticDeleteIndexFormHandler</ ↩
                classname>
            <max-number>10</max-number>
        </handler>
        <handler>
            <name>searchForms</name>
            <classname>kz.arta.synergy.indexator.elastic.ElasticSearchFormHandler</ ↩
                classname>
            <max-number>30</max-number>
        </handler>
        <handler>
            <name>searchRegistries</name>
            <classname>kz.arta.synergy.indexator.elastic.ElasticSearchRegistryHandler</ ↩
                classname>
            <max-number>30</max-number>
        </handler>
        <handler>
            <name>indexInfo</name>
            <classname>kz.arta.synergy.indexator.elastic.ElasticIndexInfoHandler</classname ↩
                >
            <max-number>1</max-number>
            <properties>configuration.path=/opt/synergy/jboss/standalone/configuration/arta ↩
                /elasticConfiguration.xml</properties>
        </handler>

    </handlers>
</system>
```

1. **Не запускается Elasticsearch**

   10.1. Необходимо проверить, что ES действительно не запустился, поскольку возможна ситуация, что он еще не успел провести первичную обработку данных (см. пункт 1).

   Проверка статуса ES может быть осуществлена двумя способами:

   - в консоли сервера, на котором запущен ES, выполните команду:

     ```
     #/etc/init.d/elasticsearsh status
     ```

     Результатом выполнения команды должно быть сообщение:

     ```
     [ ok ] elasticsearch is running.
     ```

   - проверьте статус ES непосредственно:

     ```
     #curl localhost:9200
     ```

     `localhost:9200` - это адрес ES по умолчанию.

     Вывод должен быть таким:

```
{
  "name" : "RFSWkzt",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "r67YbmerQvyNHdxlzDIt3A",
  "version" : {
    "number" : "5.1.2",
    "build_hash" : "c8c4c16",
    "build_date" : "2017-01-11T20:18:39.146Z",
    "build_snapshot" : false,
    "lucene_version" : "6.3.0"
  },
  "tagline" : "You Know, for Search"
}
```

10.2. Если вывод отличается, проверьте указание переменной `JAVA_HOME` в файле `/etc/default/elasticsearch`:

```
################################
# Elasticsearch
################################

# Elasticsearch home directory
#ES_HOME=/usr/share/elasticsearch

# Elasticsearch Java path
JAVA_HOME=/usr/lib/jvm/java-8-oracle

# Elasticsearch configuration directory
#CONF_DIR=/etc/elasticsearch

# Elasticsearch data directory
#DATA_DIR=/var/lib/elasticsearch

# Elasticsearch logs directory
#LOG_DIR=/var/log/elasticsearch

# Elasticsearch PID directory
#PID_DIR=/var/run/elasticsearch

# Additional Java OPTS
#ES_JAVA_OPTS=

# Configure restart on package upgrade (true, every other setting will lead to not ↩
    restarting)
#RESTART_ON_UPGRADE=true
```

Примечание:

В качестве `JAVA_HOME` используется полный путь к папке `bin` используемой версии Java. Строка с переменной должна быть раскомментирована.

10.3. Перезапустите ES, выполнив команду:

```
#etc/init.d/elasticsearsh restart
```

1.4.4 Использование диаграмм

Все диаграммы Kibana, за исключением диаграмм Metric и Markdown widget, полностью интерактивны Возможно «проваливание» по клику на любую часть диаграммы: при этом условие, соответствующее этой части, будет применено ко всем диаграммам на дашборде.

Рассмотрим использование диаграмм на примере ранее показанного дашборда «Анализ заявок в центральный офис» (илл. «Пример дашборда, опубликованного как внешний модуль»).

Подбор диаграмм на дашборде позволяет такие действия:

- просмотр всех заявок определенного статуса или типа;
- анализ загруженности и качества работы исполнителей;
- просмотр статуса заявок от выбранного центра решений или от выбранного клиента/проекта, и так далее.

В качестве примера детально рассмотрим действие **«Просмотр всех заявок в статусе „Отклонено“**

Для просмотра сведений по отклоненным заявкам необходимо на диаграмме «Заявки по статусам» кликнуть на сектор, соответствующий статусу «Отклонено». Условие «Статус заявки» = «Отклонено» автоматически применится ко всем диаграммам (кроме Markdown widget):

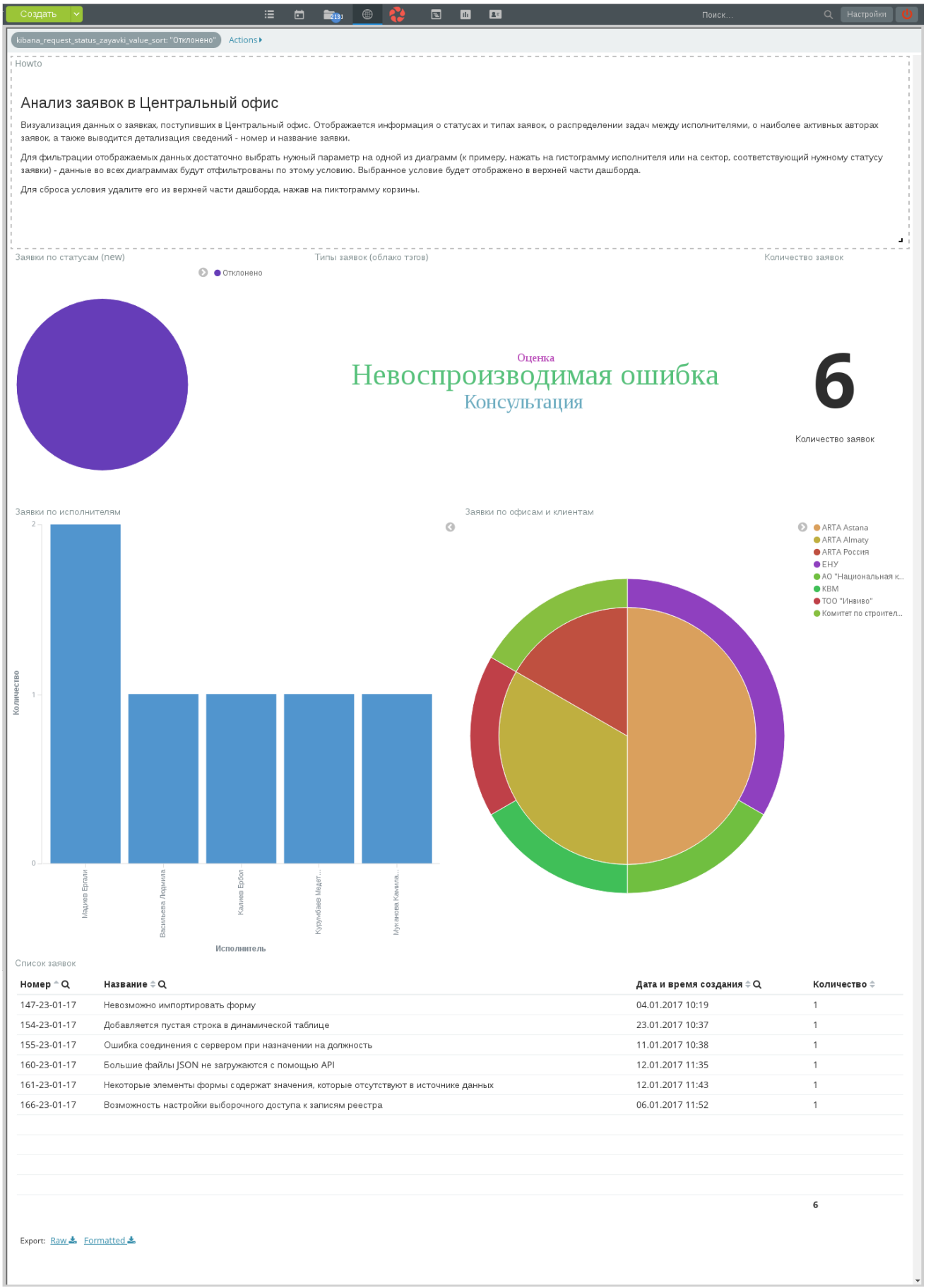


Рис. 1.51: Условие на статус применено

Произошло «проваливание»: все диаграммы отображают результаты только по заявкам, имеющим статус «Отклонено». На примере видно, что всего было отклонено 6 заявок, больше всего из них имели тип «Невоспроизводимая ошибка». Также видно, какие центры решений подавали эти заявки, кто из исполнителей их отклонял. В нижней части дашборда таблица содержит перечень всех отклоненных заявок.

В верхней части дашборда отобразилась плашка примененного фильтра в формате `%название поля%:"%значение%"`. При наведении мыши на эту плашку отображаются пиктограммы возможных действий с фильтром:

- включить/выключить фильтр;
- закрепить фильтр;
- отображать только результаты фильтрации / отображать все результаты;
- удалить фильтр;
- редактировать запрос для фильтра (синтаксис Elasticsearch).

Для того, чтобы применить еще одно условие (например, увидеть отклоненную заявку с типом «Оценка», достаточно в диаграмме *Tag cloud* кликнуть на лейбл с этим типом. Новое условие применится автоматически:

Рис. 1.52: Условие на тип применено

Видно, что единственная отклоненная заявка с типом «Оценка» касалась возможности настройки выборочного доступа к записям реестра.

Новый фильтр отображен в верхней части дашборда. Кроме того, в верхней части доступно меню **Actions**, позволяющее действия одновременно над всеми фильтрами.

> **Примечание:**
>
> Все фильтры применяются только для текущего пользователя и только на время текущего подключения. Каждый переход к внешнему модулю с диаграммами означает новое подключение к Kibana, и при этом все ранее сохраненные условия будут сброшены.

1.5 Способы авторизации в ARTA Synergy

API ARTA Synergy доступно только авторизованным пользователям. Тип авторизации — `BASIC HTTP`. Методы API выполняются от имени того пользователя, который авторизован. Имеются следующие типы авторизации:

1.5.1 Авторизация по логину и паролю

Авторизация пользователя по его логину и паролю приемлема в тех случаях, когда приложение может знать текущий логин и пароль пользователя, например:

- Приложение предоставляет альтернативный интерфейс к некоторым модулям Synergy (мобильное приложение, десктопный клиент для хранилища)
- Приложение представляет собой `server-side` утилиту для синхронизации, для которого создан выделенный пользователь, и его логин и пароль хранятся в конфигурационном файле на сервере.

Для реализации данного типа авторизации надо передать в запросе заголовок `Authorization` со значением

`Basic` + Base64(`login` + `:` + `password`)

Например:

Логин	`Administrator`
Пароль	`123456`
Значение заголовка	`Basic QWRtaW5pc3RyYXRvcjoxMjM0NTY=`

1.5.2 Сессионная авторизация

Сессионная авторизации используется для встроенных WEB-модулей. При сессионной авторизации также используется тип — `BASIC HTTP`, но в качестве логина пользователя необходимо использовать значение `$session` и в качестве пароля — полученное значение `sso_hash`.

Таким образом заголовок `Authorization` должен иметь значение:

`Basic` + Base64(`$session:` + `sso_hash`)

Например:

Значение sso_hash	D3RONfC52dtJO5XgDyn5qUMv
Значение заголовка	Basic JHNlc3Npb246RDNST05mQzUyZHRKTzVYZ0R5bjVxVU12

Кроме того, получить параметры авторизации можно с помощью переменной окружения основного WEB-приложения Synergy:

- $CURRENT_USER - представляет собой JSON-объект следующего вида:

```
{
  "id": "Идентификатор текущего пользователя",
  "sso_hash": "hash-сумма для идентификации пользователя",
  "surname": "Фамилия текущего пользователя",
  "name": "Имя текущего пользователя",
  "patronymic": "Отчество текущего пользователя"
}
```

1.5.3 Авторизация по ключам

Модуль, который хочет авторизоваться от имени какого-либо пользователя таким способом, должен сгенерировать для него ключевую пару, обеспечив сохранность закрытого ключа. Затем модуль сохраняет получивший открытый ключ для пользователя в Synergy, используя следующий вызов API:

kz.arta.synergy.server.api.rest.person.PersonService#generateUserAuthKey

Этот вызов назначает ключ тому пользователю, от имени которого выполняется.

Параметр user_token_expire_interval регулирует интервал устаревания ключей авторизации. Пример настройки интервала:

```
    insert into options (id, value) values ('user_token_expire_interval', '5256000'); -- 10 ↩
        лет
```

> Интервал устаревания ключа указывается в минутах. Значение по умолчанию 0, то есть если ранее для данного пользователя был сгенерирован другой ключ, то предыдущий автоматически становится недействительным. Создать ключ можно только для существующего WEB-модуля, так как для этого требуется идентификатор приложения. Если у вас нет необходимости разрабатывать WEB модуль, но есть необходимость в использовании авторизации по ключам, можно создать такой модуль на уровне БД и отключить его использование в административном приложении SynergyAdmin для всех элементов оргструктуры

Использование этого ключа для авторизации аналогично использованию сессионного ключа. Тип авторизации Basic HTTP, в качестве логина пользователя надо использовать строку «$key», в качестве пароля — полученный с помощью API ключ.

Таким образом заголовок Authorization должен иметь значение:

Basic + Base64($key: + значение_ключа)

Например:

Значение ключа	MS03Y2Q0ZGU3YS0zYjRkLTQ2NjgtYWIyOC0zZDI1YzgxZGNmOGZfMjAxMy0xMC0zMSAxNzo0Mg==
Значение заголовка	Basic JGtleTpNUzAzWTJRMFpHVTNZUzB6WWpSa0xUUTJOamd0WVdJeU9DMHpaREkxWXpneFpHTm1PR1pmTWpBeE15MHhNQzB6TVNBeE56bzBNZz09

1.6 Как задеплоить интеграционное приложение

ARTA Synergy работает на сервере приложений JBoss AS7. Интеграционное приложение может представлять собой `jar`-файл либо `war`-файл либо их комбинацию.

Если приложение является одиночным файлом, его можно задеплоить, скопировав в директорию `${jboss.home}/standalone/deployments`. Если приложение состоит из нескольких файлов, необходим создать `*.ear` приложение.

Если приложение имеет зависимости на внешние библиотеки и они находятся в модулях JBoss-а (`${jboss.home}/modules`), необходимо использовать их, прочие зависимости — помещать внутрь приложения.

> **В целях безопасности работы приложения Synergy и сервера приложений категорическ запрещается помещать артефакты интеграционного модуля в приложение Syn ergy.ear и изменять состав модулей (`${jboss.home}/modules`).**

1.7 Стандартные решения интеграционных задач

1.7.1 Внедрение портлета liferay в Synergy

С помощью разработки BMK есть возможность отображения портлетов в Synergy. Для того чтобы отобразить портлет необходимо:

- Установить java 1.8.
- Развернуть инстанс liferay. Для этого скачиваем со страницы https://www.liferay.com/downloads Liferay Portal CE Bundled with Wildfly. Разворачиваем архив в папку /opt. Переходим в папку /opt/liferay-portal-7.0-ce-ga1/wildfly-10.0.0/bin. Запускаем jboss ./standalone.sh -b 0.0.0.0
- Открываем браузер и подключаемся к порту 8080 (http://localhost:8080). Вначале предстоит пройти установки, предложенные Basic Configuration. Принимаем настройки указанные по умолчанию
- Создаем файл system-ext.properties в папке /opt/liferay-portal-7.0-ce-ga1/wildfly-10.0.0/standalone/depl INF/classes/
- Прописываем в файле строку http.header.secure.x.frame.options=false, чтобы была возможность отображения портлетов на других сайтах (системные настройки liferay)
- Перезапускаем jboss
- Заходим на страницу http://localhost:8080 и на ней добавляем предлагаемые liferay портлеты.
- У каждого портлета в настройках прописан его адрес. Например:

```
<script src="http://192.168.2.119:8080/o/frontend-js-web/liferay/widget.js" type="text/ ↩
    javascript"></script>
<script type="text/javascript">
Liferay.Widget({ url: 'http://192.168.2.119:8080/widget/web/guest/home/-/contacts'});
</script>
```

- Этот адрес используем для внедрения портлета в синерджи.

Пример 1:

Пишем пользовательский компонент в конфигураторе, который отобразит ссылку рядом с полем поиска в синерджи, по нажатию на которую отобразится портлет в диалоговом окне.

Исходный код:

```
<a href="#" onclick="openbox('Wrapp');return false;" >Портлет</a>
<div id="Wrapp" style="display:none">
<div id='tt'><div class="close" onclick="openbox('Wrapp')">x</div>
<iframe frameborder="0" height="100%" id="portlet1" src="http://192.168.2.119:8080/widget/ ↩
    web/guest/home/-/com_liferay_password_generator_web_portlet_PasswordGeneratorPortlet"  ↩
    width="100%"></iframe>
  </div></div>
<style type="text/css">
#Wrapp,#Wrapp2 {
 top:0;
  left:0;
 position:fixed;
 background-color: rgba(0, 0, 0, 0.8);
  opacity: inherit;
 width:100%;
 height:100%;
 z-index:10000;
}
#tt,#tt2 {
  position:relative;
  background-color:#fff;
  width:500px;
  padding:12px;
  height:400px;
  margin:7% auto auto auto;
  border:red solid 1px;
}
div.close {
  cursor:pointer;
  position:absolute;
  font-weight:700;
  text-shadow:#000 1px 1px 0;
  color:red;
  right:7px;
  top:2px;
}
</style>
```

Исходный код скрипта:

```
 function openbox(id,tt) {
    var div = document.getElementById(id);
    var tt_div = document.getElementById(tt);
    if(div.style.display == 'block') {
        div.style.display = 'none';
    }
    else {
        div.style.display = 'block';
    }
  }
```

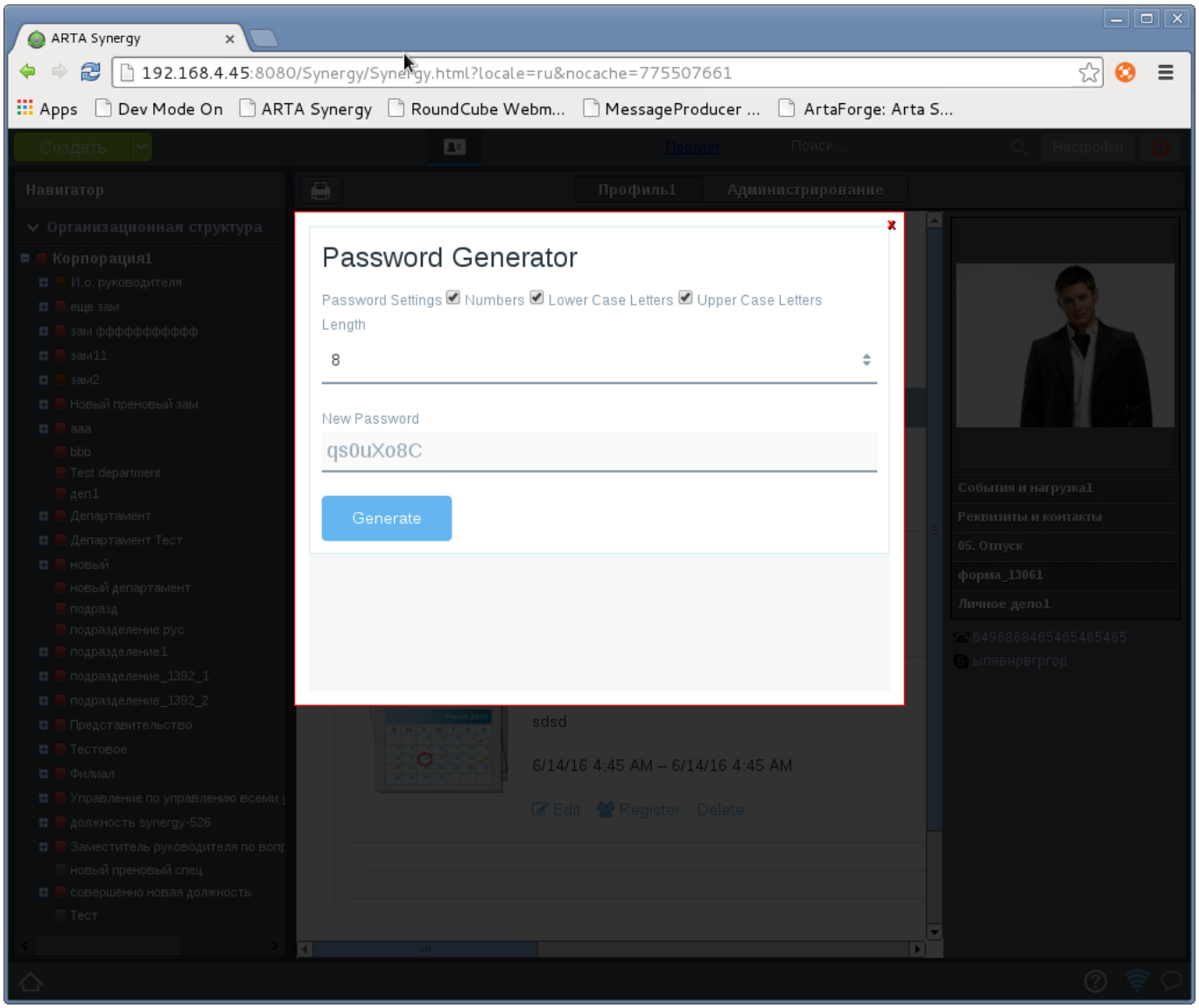


Рис. 1.53: Отображение портлета в диалоговом окне

Пример 2:

Заменяем компоненты формы изображение с идентификатором image_for_portlet.

Исходный код:

```
<div></div>
<style type="text/css">
#portlet_div {
  position:relative;
  width:640px;
  padding:12px;
  height:448px;
}
</style>
```

Исходный код скрипта:

```
(function () {
```

```
            console.log("appended");

            var checkAgain = function() {
                setTimeout(changeUserName, 5000);
            };

            var changeUserName = function () {
                var component = jQuery('img#image_for_portlet');
                if (component === null || component.length === 0) {
                    console.log("no success");
                    checkAgain();
                    return;
                }
                console.log("success!!");
                console.log(component.parentNode);
                console.log(component);
                var mySpan = document.createElement("div");
                mySpan.innerHTML = "<div id='portlet_div' ><iframe frameborder = "0" height ↩
                     = "100%" id = "portlet1" src = "http://192.168.2.119:8080/widget/web/ ↩
                    guest/home/-/meetups" width = "100%" > </iframe ></div > ";
                component.replaceWith(mySpan.innerHTML);
        checkAgain();
            };
            changeUserName();
        })();
```

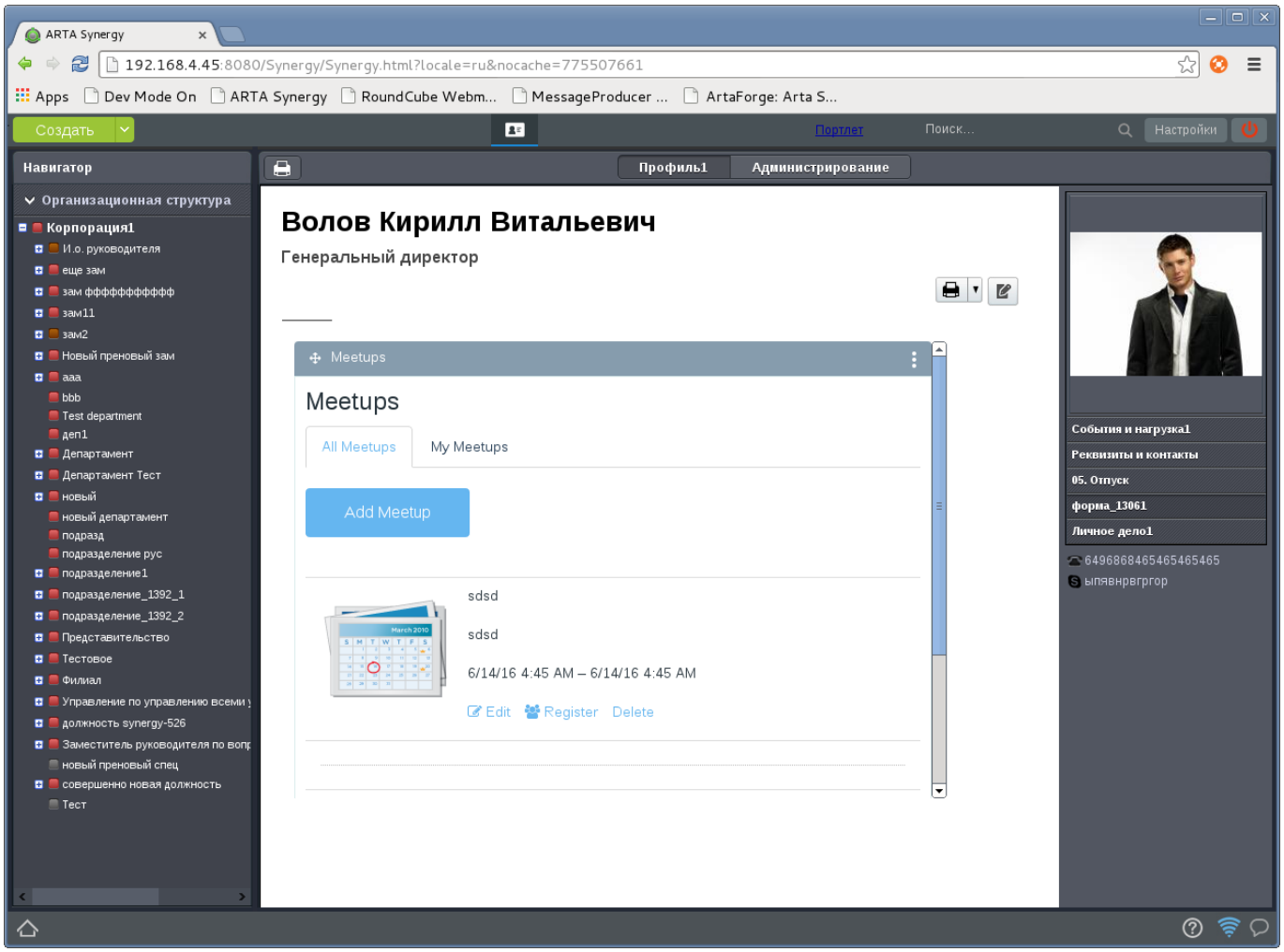


Рис. 1.54: Отображение портлета на форме Synergy