Kolmogorov.ai | Predicate

Model and Data Quality

ООО Дата Сапиенс

v.1.1.0 | 01.11.2023

Table of contents

1. Home

Predicate [pr d k t] - Инструмент класса Metrics Store.

Позволяет организовать библиотеку произвольных метрик и их визуализаций, упрощает задачу регламентного построения рабочих панелей мониторинга бизнес решений и обработки инцидентов качества.

Скачать документацию в формате PDF

2. Install

2.1 Введение

Predicate позволяет тестировать модели и проверять качество данных рассчитывая метрики по регламенту и формируя отчеты.

Основной сервис бэкенда - FastAPI сервер, который принимает запросы по адресу, описанному в ingress сервиса. Часть задач выполняется асинхронно с использованием Celery. Данные хранятся в БД Postgres. Для каждого проекта генерируется DAG Airflow, который позволяет мониторить данные, метрики и модели по регламенту.

Взаимодействие пользователя с приложением осуществляется через web-интерфейс. Доступ к функционалу открывается после авторизации.

Установка

Первичная установка программного обеспечения производится командой разработки и внедрения ООО «Дата Сапиенс» на сервер, предоставляемый заказчиком.

Обновление

Проверка, загрузка и установка обновлений программного обеспечения требует подключения к Интернету и выполняется вручную на сервере администратором ПО. Для обновления комонента ПО необходимо удалить его и установить повторно, после чего снова запустить Kolmogorov.ai **Predicate**.

Переустановка

Любую доступную версию Kolmogorov.ai **Predicate** можно загрузить и установить с помощью контейнеров, поставляемых разработчиком. Перед переустановкой допускается удаление существующего ПО.

2.2 Администратору

2.2.1 Пререквизиты

Kubernetes

Должен быть развернуть кластер Kubernetes и установлен инструмент для управления Kubernetes приложениями Helm

Установка Kubernetes на minikube

Рекомендации от команды DevOps

Keycloak

Сервис для авторизации и аутентификации Keycloak.

Рекомендуется использовать чарт codecentric

Gitlab

В Gitlab необходимо создать два репозитория:

SHARE

Репозиторий Share нужен для хранения файлов метрик и трансформов. Путь до репозитория указывается в переменной окружения `PREDICATE_GIT_REPO_SHARE`, ветка, в которой хранятся файлы, соотвествующие конкретному стенду, указывается в переменной окружения `PREDICATE_GIT_SHARE_BRANCH`.

DAGS

Для хранения DAGs проектов, сгенерированных Predicate.

Путь до репозитория указывается в переменной окружения `PREDICATE_GIT_DAGS_BRANCH`, ветка, в которой хранятся файлы соотвествующие конкретному стенду указывается в переменной окружения `PREDICATE_GIT_SHARE_BRANCH`.

Airflow

Airflow нужен для исполнения DAGs проектов, сгенерированных Predicate, по расписанию или разово, если расписание не указано.

Рекомендуется для установки Airflow использовать этот чарт

При установке отредактируйте файл `values.yaml`:

- укажите нужный `ingress`
- укажите репозиторий Gitlab и ветку, в которой будет лежать DAGs: `airflow-service` -> `dags` -> `gitSync`

Minio

S3 хранилище, в котором сохраняются временные файлы и результаты метрик при исполнении DAG.

Инструкция по установке Minio

2.2.2 Серверная часть

Серверная часть

Predicate позволяет тестировать модели и проверять качество данных рассчитывая метрики по регламенту и формируя отчеты.

Основной сервис бэкенда - FastAPI сервер, который принимает запросы по адресу, описанному в ingress сервиса. Часть задач выполняется асинхронно с использованием Celery. Данные хранятся в БД Postgres. Для каждого проекта генерируется DAG Airflow, который позволяет мониторить данные, метрики и модели по регламенту.

Устройство чарта серверной части

Чарт содержит в себе следующие сервисы:

PREDICATE-BACKEND

Основной сервис бекенда. Внутри контейнера разворачивается FastAPI сервер, который принимает запросы по адресу, описанному в ingress

Содержит init-container `init-db`, применяющий миграции Alembic к базе данных, URL которой указан в переменной `PREDICATE_DB_URL`.

Внутри этого сервиса в values.yaml описываются переменные окружения, которые потом собираются в секрет predicate-backend-secret.

SERVICE-CELERY

Сервис запускает воркер для Celery

SERVICE-GITSYNC

Сервис для синхронизации состояния репозитория git-share и volume в кластере

SERVICE-FLOWER

Flower для мониторинга и администрирования заданий Celery

POSTGRESQL

Основная база данных Predicate на базе Postgres

REDIS

Брокер сообщений для celery задач

Переменные окружения

Название	Описание	Тип данных	Значение по умолчанию
TZ	Часовой пояс	str	Europe/Moscow
PREDICATE_ENV_NAME	Название среды	str	
PREDICATE_ROOT_PATH	root_path для сервера Predicate на базе FastAPI	str	/api
PREDICATE_URL	Адрес Predicate. Если значения переменных берутся не из Vault, то можно использовать шаблон "https://{{ .Values.ingress.host }}.{{ .Values.ingress.baseDomain }}" в качестве значения	str	
PREDICATE_PORT	Порт, на котором разворачивается сервер Predicate	int	8080
PREDICATE_DB_URL	Адрес Postgres базы данных в формате "postgresql://:@:/" В качестве рекомендуется использовать название сервиса базы predicate-postgresql	str	postgresql://:@predicate-postgresql:5432/postgres
PREDICATE_DB_POOL_SIZE	Количество одновременных подключений к базе	int	30
PREDICATE_DB_CONNECT_TIMEOUT	Время ожидания подключения к базе данных(в секундах)	int	10
PREDICATE_AIRFLOW_URL	Адрес Airflow	str	
PREDICATE_AIRFLOW_USER	Логин пользователя Airflow	str	
PREDICATE_AIRFLOW_PASSWORD	Пароль пользователя Airflow	str	
PREDICATE_AIRFLOW_RETRIES	Количество попыток выполить задачу Airflow в случае ошибки	int	1
PREDICATE_AIRFLOW_RETRY_DELAY_MIN	Время между попытками в минутах	int	3
AIRFLOW_DATA_RETRIES	Количество попыток выполить задачу по загрузке данных при помощи datasource Airflow в случае ошибки	int	10
AIRFLOW_DATA_RETRY_DELAY_MIN	Время между попытками в минутах	int	5
PREDICATE_SMTP_HOST	Адрес SMTP-сервера для отправки уведомлений на почту	str	smtp.gmail.com
PREDICATE_SMTP_PORT	Порт SMTP-сервера	int	465
PREDICATE_SMTP_USER	Адрес email пользователя для отправки писем через SMTP-сервер	str	
PREDICATE_SMTP_PASSWORD	Пароль или токен пользователя для отправки писем через SMTP-сервер. токен можно получить через Google App Passwords	str	
PREDICATE_KEYCLOAK_USERNAME	Логин администратора Keycloak	str	

Название	Описание	Тип данных	Значение по умолчанию
PREDICATE_KEYCLOAK_PASSWORD	Пароль пользователя Keycloak	str	
PREDICATE_KEYCLOAK_TOKEN_URL	Адрес Keycloak для получения токена в формате https:///auth/realms//protocol/openid-connect/token	str	
PREDICATE_KEYCLOAK_AUTH_URL	Адрес Keycloak для авторизации в формате https:///auth/admin	str	
PREDICATE_KEYCLOAK_REALM	Название realm Keycloak	str	
PREDICATE_KEYCLOAK_CLIENT	Название приложения(client) Keycloak	str	
PREDICATE_EXECUTION_IMAGE_PULL_POLICY	ImagePullPolicy для образов	str	Always
PREDICATE_EXECUTION_BASE_IMAGE	Образ, который используется для исполнения метрик и трансформов в сгенерированном DAG	str	
PREDICATE_K8S_REGCRED	Название image_pull_secrets	str	regcred
PREDICATE_K8S_SECRET_NAME	Имя секрета, который используется в KubernetesPodOperator в DAG проекта. Должен совпадать с секретом, который опередяется в values.yaml в основном сервисе чарта	str	predicate-backend-secret
PREDICATE_K8S_PVC_DATAFRAME	Имя PVC для датафреймов	str	
PREDICATE_K8S_PVC_DATAFRAME_PATH	Путь, по которому монтируется в контейнер PVC с датафреймами	str	
PREDICATE_K8S_PVC_TRANSFORM	Имя PVC для трансформов	str	
PREDICATE_K8S_PVC_TRANSFORM_PATH	Путь, по которому монтируется в контейнер PVC с трансформами	str	
PREDICATE_K8S_PVC_SHARE	Имя PVC для кода метрик	str	
PREDICATE_K8S_PVC_SHARE_PATH	Путь, по которому монтируется в контейнер PVC с кодом метрик	str	
PREDICATE_SHARE_PATH	Путь в PVC до папки с метриками и трасформами	str	
PREDICATE_S3_URL	Адрес S3 хранилища	str	
PREDICATE_S3_USERNAME	Логин для доступа к S3 хранилищу	str	
PREDICATE_S3_PASSWORD	Пароль для доступа к S3 хранилищу	str	
PREDICATE_S3_RESULT_BUCKET	Имя S3 bucket для сохранения результатов метрик	str	
PREDICATE_S3_TMP_BUCKET	Имя S3 bucket для сохранения временных файлов (код метрик, код трансформов, pickle)	str	
PREDICATE_S3_IGNORE_SECURE	Проверка SSL сертификатов при подключении к S3	bool	False

Название	Описание	Тип данных	Значение по умолчанию
PREDICATE_GIT_URL	Адрес Gitlab	str	
PREDICATE_GIT_LOGIN	Логин пользователя для доступа к Gitlab	str	
PREDICATE_GIT_SECRET	Токен пользователя для доступа к Gitlab	str	
PREDICATE_GIT_USER_NAME	Имя пользователя Gitlab	str	
PREDICATE_GIT_USER_EMAIL	Почта пользователя Gitlab	str	
PREDICATE_GIT_REPO_DAGS	Путь, по которому находится репозиторий с DAGs Если репозиторй располагается по адресу https://git.angara.cloud/kolmogorov/predicate/dags, то значение переменной будет kolmogorov/predicate/dags	str	
PREDICATE_GIT_REPO_SHARE	Путь, по которому находится репозиторий с метриками Если репозиторй располагается по адресу https://git.angara.cloud/kolmogorov/predicate/share, то значение переменной будет kolmogorov/predicate/share	str	
PREDICATE_CELERY_BROKER_URL	Адрес брокера сообщений Redis в формате redis://:@: В качестве рекомендуется использовать название сервиса predicate-redis-master. Значение по умолчанию 6370	str	
PREDICATE_CELERY_RESULT_BACKEND_URL	Адрес БД с результатами выполнения Celery задач формат "db+postgresql+psycopg2://postgres:postgres@predicate-postgresql:5432/postgres"	str	
PREDICATE_CELERY_FLOWER_API_URL	Адрес сервиса Flower. В качестве URL рекомендутеся использоваться название сервиса predicate-flower	str	http://predicate-flower:5555/api
GIT_SYNC_USERNAME	Логин пользователя для доступа к Gitlab	str	
GIT_SYNC_PASSWORD	Токен пользователя для доступа к Gitlab	str	
GIT_SYNC_REPO	Адрес репозитория Gitlab	str	
GIT_SYNC_REV	Git revision (тег или хэш) для проверки	str	HEAD
GIT_SYNC_GIT_CONFIG	Дополнительные найстройки конфига git в формате'key1:val1,key2:val2'	str	http.sslVerify:false

Название	Описание	Тип данных	Значение по умолчанию
GIT_SYNC_MAX_FAILURES	Количество последовательных сбоев, допустимое перед прерыванием синронизации	int	
GIT_SYNC_PERIOD	Сколько времени нужно ждать между попытками синхронизации. Этот параметр должен быть не менее 10 мс.	str	
GIT_SYNC_DEPTH	Число коммитов, которое будет скопировано в историю репозитория-клона. По умолчнаию копируются все коммиты	int	
PREDICATE_K8S_NAMESPACE	Namespace, который пользуется в KubernetesPodOperator в DAG	str	predicate-{{ .Values.env }}
PREDICATE_K8S_BASE_IMAGE	Основной образ Predicate для исполнения тасок в DAG, совпадает с образом сервиса service в чарте. Для некоторых тасок указывается другой образ(см. `PREDICATE_EXECUTION_BASE_IMAGE`)	str	{{ .Values.image.repository }} {{ .Values.image.tag }}
PREDICATE_GIT_DAGS_BRANCH	Ветка репозитория DAGs, из который берется DAGs для текущего стенда. Обычное совпадает с названием среды	str	{{ .Values.env }}
PREDICATE_GIT_SHARE_BRANCH	Ветка репозитория метрик, из который берется метрики для текущего стенда. Обычное совпадает с названием среды	str	{{ .Values.env }}
GIT_SYNC_BRANCH	Ветка репозитория, с которой синхронизируется текущий стенд. Обычное совпадает с названием среды	str	{{ .Values.env }}

Установка чарта бекенд сервиса Predicate

1. Отредактируйте файл `values.yaml`
   Подставьте актуальный репозиторий и тег образа в следующих сервисах:
   - `service` > `image`
   - `service-celery` -> `image`
   - `service-flower` -> `image`
   Подставьте актуальный `host` и `baseDomain` в следующих сервисах
   - `service` -> `ingress`
   - `service-flower` -> `ingress`
2. Если установка релиза происходит первый раз, то раскомментировать строчку `"--metrics"` в `service` -> `init` -> `containers` -> `args`, чтобы заполнить таблицу Metrics базовыми метриками
3. Запустите команду для установки релиза

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

Обновление

Для обновления чарта запустите команду:

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

Удаление

Для удаления чарта запустите команду:

```
helm delete --namespace=<namespace> <realisename>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза,

2.2.3 Клиент

Клиент

Web-интерфейс Predicate

Устройство чарта

Чарт содержит в себе один сервис: `predicate-frontend` с файлами web-приложения.

Сервис имеет два init-контейнера:

1. `generate-configs`

   Настраивает перменные окружения `PREDICATE_API` и `DATASOURCE_REGISTRY_API`

2. `nginx-config`

   Запускает nginx-сервер

Переменные окружения

Название	Описание	Тип данных	Значение по умолчанию
PREDICATE_API	Адрес API, обычно совпадает с адресом интерфейса с префиксом /api	str	
DATASOURCE_REGISTRY_API	Адрес API Datasource	str	
KEYCLOAK_REALM	Имя realm Keycloak. Обычное совпадает с realm, указанным в бекенде	str	
KEYCLOAK_AUTH_SERVER_URL	Адрес Keycloak для авторизации в формате https:///auth Обычное совпадает с указанным в бекенде	str	
KEYCLOAK_RESOURCE	Название приложения(client) Keycloak	str	
TZ	Часовой пояс	str	Europe/Moscow

Установка чарта сервиса Predicate

1. Отредактируйте файл `values.yaml`
   - Укажите актуальный репозиторий и тег образа
   - Укажите актуальный `host` и `baseDomain`
   - Укажите актулаьные переменные окружения
2. Запустите команду

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

Обновление

Для обновления чарта запустите команду:

```
helm upgrade
--namespace=<namespace> \
--install \
--atomic \
<realisename> <path to chart>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза, `path to chart` - путь до папки чарта или до архива

Удаление

Для удаления чарта запустите команду:

```
helm delete --namespace=<namespace> <realisename>
```

где `namespace` - неймспейс k8s, в котором устанавливается релиз, `realisename` - название релиза,

2.2.4 Datasource

1. Общее описание

Сервис представляет собой REST API для получения информации о таблицах и их столбцах из источника данных (путь до точки входа - `klmg_datasource/_bin_/datasource:server`). Также имеется скрипт для скачивания данных из источника данных (путь до скрипта - `klmg_datasource/_bin_/datasource:download`).

На данный момент поддерживаются следующие источники данных: ***S3, PostgreSQL, Oracle, Snowflake, Hive (с подключением через Kerberos) и Axiom (другой продукт kolmogorov.ai)***. Также можно добавлять свои источники данных. Для этого нужно создать папку с именем источника данных папке `klmg_datasource/sources` и переопределить функции для работы с этим источником данных.

В ***Makefile*** находятся основные команды:

- `check_code` - проверка кода линтерами (flake8, black, isort) и статическая проверка типов (mypy);
- `test` - (нужен python3.9) запуск юнит-тестов;
- `install_env` - локальная установка виртуального окружения;
- `docker/build/{source_name}` - сборка docker образа;
- `docker/push/{source_name}` - пуш docker образа в docker registry;
- `helm/install/{source_name}` - установка helm chart в kubernetes\openshift\minikube;
- `helm/delete/{source_name}` - удаление helm chart из kubernetes\openshift\minikube;

Источник данных S3 дополнительно использует Celery для загрузки файлов в S3 и отображения прогресса загрузки.

2. Деплой сервиса

`secret_name` - уникальный ключ datasource. Должен совпадать:

- в поле `datasource_key` в таблице `Datsource` в БД Postgres Predicate. `datasource_key` - ключ, который указывается при создании нового источника данных.
- названии секрета `{secret_name}-secret`, который устанавливается в K8S, и указывается в чарте
- в `uriPrefix` сервиса в чарте: `/datasource/api/{secret_name}(/|$)(.*)`
- в переменной окружения `DATASOURCE_ROOT_PATH` в чарте: `/datasource/api/{secret_name}`

Если нужно создать два источника данных одного типа, то они должны отличаться ключом, например, S3 и S3-2

1. Соберите docker образ и запуште в docker registry

```
make cr={cr} tag={tag} docker/build/{source_name} && make cr={cr} tag={tag} docker/push/{source_name}
```

где `tag` - имя тега docker-образа, `cr` - репозиторий образа, по умолчанию *registry.git.angara.cloud:443/kolmogorov/datasource*

2. Добавьте секрет в kubernetes\openshift\minikube

```
kubectl apply -f ./_secrets_/{secret_name}-secret.yaml -n {namespace}
```

3. Проверьте значения в `values.{source_name}.yaml`:
4. актуальный `image` -> `repository` и `image` -> `tag`
5. source_name в `uriPrefix`
6. акутальный `ingress` -> `host`
7. source_name в `extra_vars` -> `secret`
8. source_name в `extra_vars` -> `DATASOURCE_ROOT_PATH`
9. Установите helm chart в kubernetes\openshift\minikube

```
make helm/install/{source_name}
```

Добавление datasource в Predicate

1. Во вкладке *Источники данных* в web-интерфейсе Predicate нажмите кнопку *Создать*
2. В поле `Ключ` укажите `source_name`
3. Укажите название
4. В поле `Образ` укажите образ, совпадающий с образом указанным в чарте

3. User Guide

3.1 О Predicate

3.1.1 Назначение и функционал

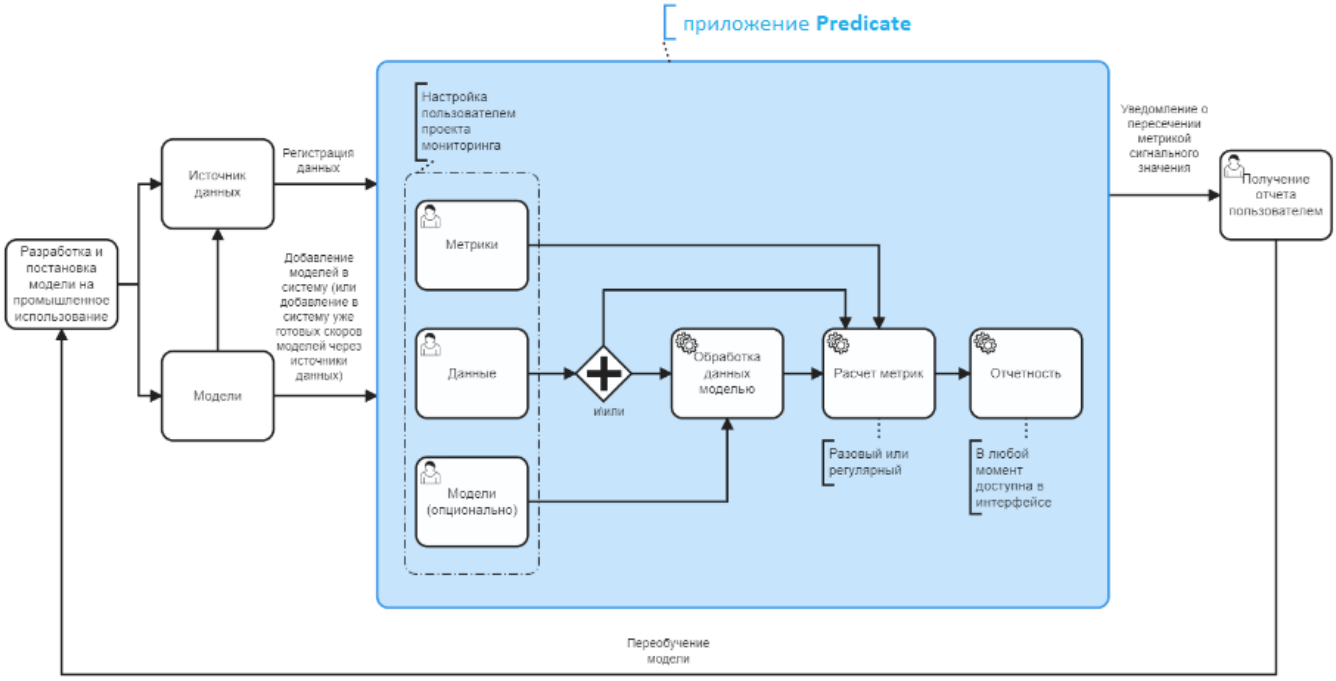
Predicate [pr d k t] - Инструмент класса *Metrics Store*.

Позволяет организовать библиотеку произвольных метрик и их визуализаций, упрощает задачу регламентного построения рабочих панелей мониторинга бизнес-решений и обработки инцидентов качества.

Приложение Kolmogorov.ai **Predicate** может применяться как инструмент класса *Model Quality Management* при автоматизации процессов тестирования, мониторинга и валидации аналитических моделей и наборов данных в системах принятия решений различных индустрий:

- финансовый сектор;
- телеком, ритейл;
- производство.

Роль Predicate в промышленном применении модели



Функционал

В приложении Kolmogorov.ai **Predicate** предусмотрены следующие **пользовательские сценарии**:

- постановка модели на мониторинг с автоматическим созданием дашборда;
- регистрация нового объекта данных;
- загрузка новой метрики или теста;
- загрузка модели или преобразования данных;
- создание шаблона проекта мониторинга;
- создание отчета о валидации.

3.1.2 Глоссарий

Источник данных – подключение к некоторому хранилищу данных (файлов / таблиц БД).

Данные – датасет с сохраненными настроенными метаданными (источник подключения, описание, разметка и т.д.). Примеры объектов данных: один файл csv, одна таблица SQL БД.

Разметка данных – описание свойств столбцов зарегистрированного в системе объекта данных.

Метрика – исполняемый над данными тест (скрипт), выводящий результат в виде графика и (или) скалярного значения.

Модель – предиктивная аналитическая модель.

Преобразование – исполняемый над данными скрипт, выводящий результат в виде датасета с числом столбцов, большим или равным числу столбцов в исходном датасете. Применение модели к датасету, содержащему набор признаков, задается в системе как преобразование, добавляющее к исходному датасету столбец с предсказаниями.

Проект – проект мониторинга. Состоит из набора данных, метрик и (опционально) моделей с настроенным пользователем режимом исполнения.

Регламент – расписание запусков проекта мониторинга.

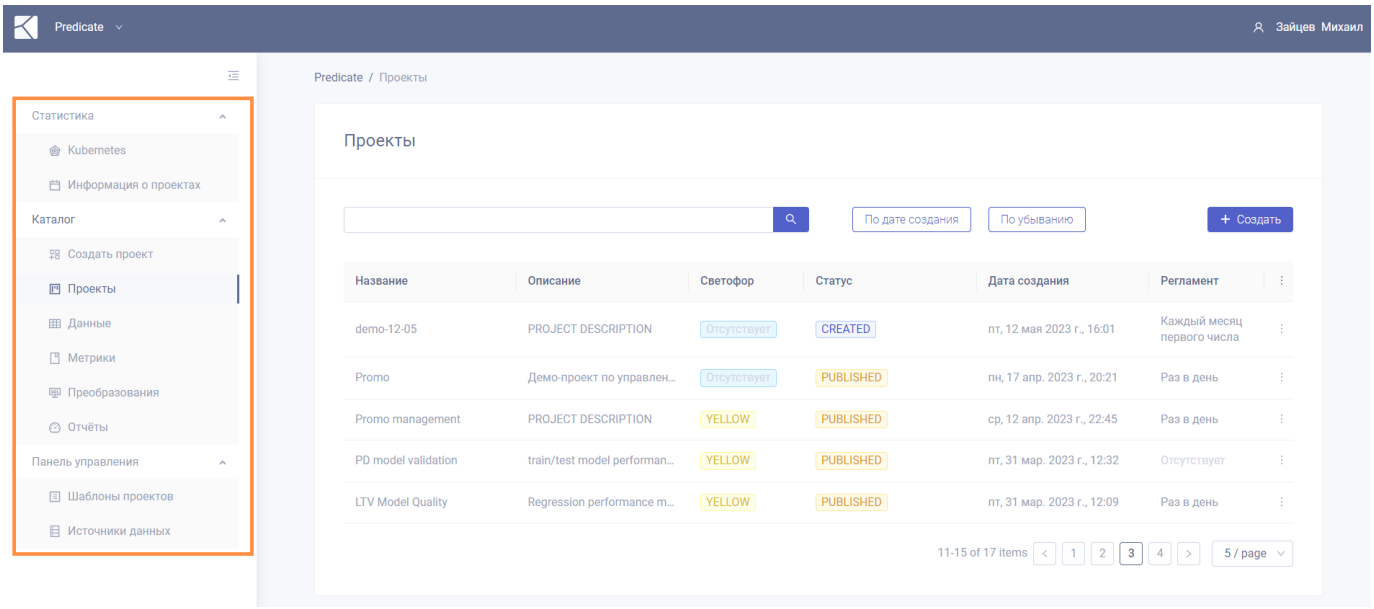
Дашборд – раздел проекта с результатами расчета метрик.

Пользовательский дашборд (отчет) – совокупность текстовых панелей и графиков, созданная пользователем на основе результатов расчета проектов мониторинга. Один пользовательский отчет может включать результаты нескольких проектов мониторинга.

Шаблон проекта – преднастроенный сохраненный набор метрик, их параметров и (опционально) настроек регламента.

3.1.3 Обзор основного меню

Основное меню приложения расположено в левой части экрана:



Описание разделов меню и функций, доступных из этих разделов, приведено ниже.

Статистика

KUBERNETES

Информация о загруженности namespace в OpenShift, в котором будут исполняться проекты Predicate.

- Память: оперативная память, которую использует приложение Predicate (используемая память / квота на кластере Kubernetes)
- CPU: число процессорных ядер (используется / квота)

> **Примечание**
>
> В статистику входят только сервисы, которые относятся к приложению Predicate, и не входит Airflow (который обычно является основным потребителем CPU).

ИНФОРМАЦИЯ О ПРОЕКТАХ

Информация о расписании запуска проектов в указанный период, а также о проблемах исполнения проектов за указанный период.

Каталог

СОЗДАТЬ ПРОЕКТ

Функция создания проекта мониторинга качества модели/данных.

Подразумевает выбор пользователем данных для расчета, необходимых метрик и периодичности мониторинга, а также (опционально) настройку светофора проекта.

ПРОЕКТЫ

- Каталог имеющихся в системе (созданных ранее) проектов мониторинга качества моделей/данных.

  Для каждого проекта доступны:
- основная информация (идентификатор, статус, дата создания и др.);
- список входных параметров (используемые данные и метрики);
- расписание запусков;
- результаты выполнения проекта (автоматически формируемый дашборд и логи).
- Переход к форме создания нового проекта.

ДАННЫЕ

- Каталог датасетов, доступных для проведения расчетов в проектах мониторинга.
- Переход к форме регистрации нового датасета.

МЕТРИКИ

- Каталог метрик (тестов), доступных для расчета в проектах мониторинга.
- Переход к форме регистрации новой метрики.

ПРЕОБРАЗОВАНИЯ

- Каталог преобразований, доступных для применения к зарегистрированным в системе датасетам.
- Переход к форме регистрации нового преобразования.

ОТЧЕТЫ

Дашборды пользовательского дизайна собираются из доступных в системе результатов проектов мониторинга (графиков и скалярных значений метрик). Позволяют включать блоки с текстовой информацией.

Доступна выгрузка отчета в файл pdf.

Панель управления

ШАБЛОНЫ ПРОЕКТОВ

Шаблон проекта представляет из себя преднастроенный набор метрик. При работе с шаблоном проекта для запуска расчетов пользователю остается выбрать только набор данных.

На странице доступны:

- Каталог шаблонов проектов мониторинга.
- Переход к форме создания нового шаблона.

ИСТОЧНИКИ ДАННЫХ

Источник данных - подключение к существующему хранилищу данных, позволяющее получить доступ к находящимся в хранилище таблицам данных для дальнейшего формирования датасетов, которые будут использоваться при создании проектов мониторинга.

На странице доступны:

- Сведения о подключенных к системе хранилищах данных.
- Переход к форме подключения нового источника данных.

3.1.4 Ролевая модель Predicate

Сущности, к которым предоставляется ролевой доступ

Присвоение ролевого доступа доступно для всех каталогизируемых сущностей Predicate, а именно:

- проектов
- метрик
- данных
- преобразований
- отчетов
- шаблонов проектов
- источников данных

Роли и уровень доступа

В Predicate поддерживаются следующие роли :

Название роли	Уровень доступа
reader	Доступ на чтение всей информации об объекте
editor	reader + редактирование объекта (кроме доступов и удаления)
maintainer	editor + возможность предоставления доступа к объекту (кроме owner)
owner	maintainer + возможность предоставления доступа owner и удаления объекта

Роли Predicate являются вторичными ролями относительно **ролей KeyCloak**. Если роль на KeyCloak предоставляет или отнимает доступ к определенной группе объектов, то роли Predicate для этих объектов не будут иметь значения. (KeyCloak - внешний сервис, используемый приложением Predicate для авторизации и аутентификации.)

Пользователю может быть присвоена только одна роль на одном объекте.

Если роль присвоена группе KeyCloak, то доступ получат все пользователи, добавленные в эту группу.

Пользователь, который создал объект, по умолчанию наделяется ролью owner для этого объекта.

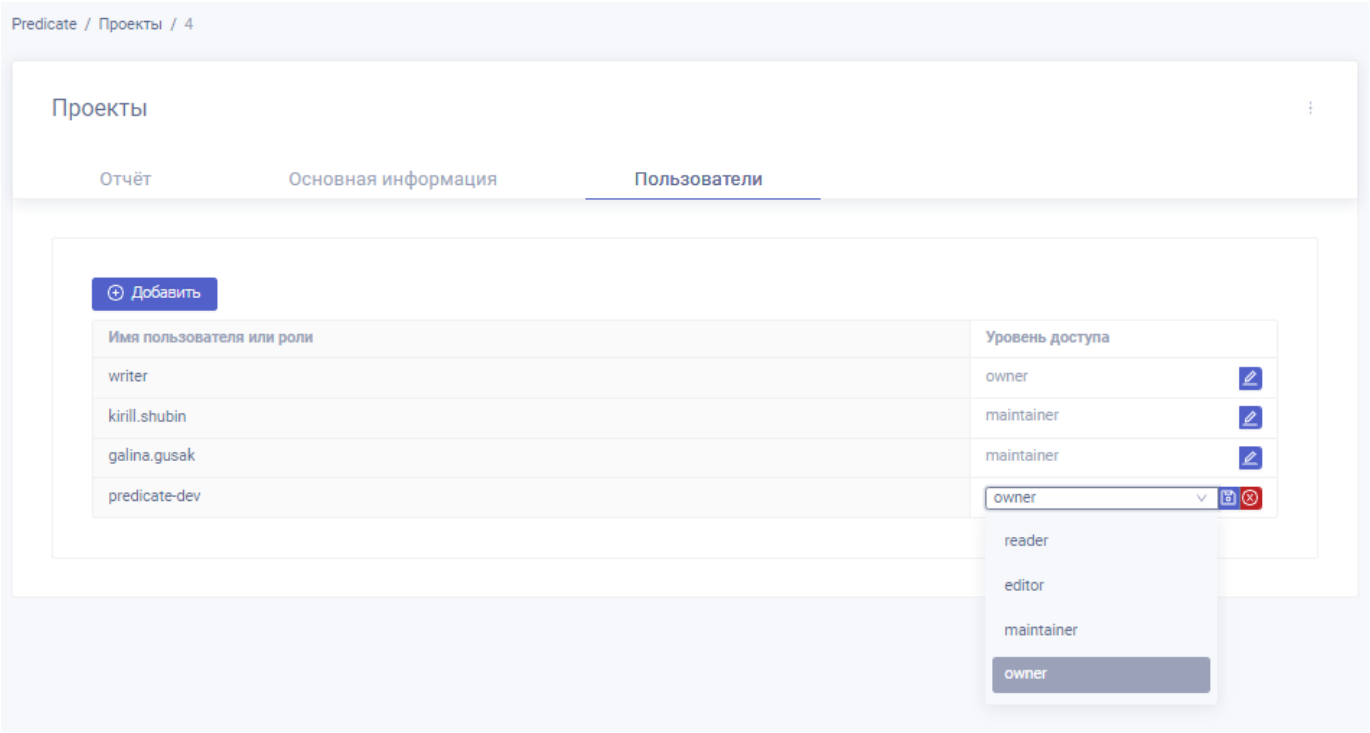
Просмотр / предоставление / изменение уровня доступа к объекту

После создания объекта (или шаблона) в Predicate по умолчанию доступ к нему имеет только один пользователь - тот, кто создал объект.
Чтобы другие пользователи также смогли просматривать информацию об объекте и/или иным способом работать с ним, необходимо отдельным действием предоставить им доступ.

Просмотр списка пользователей, имеющих доступ к объекту, а также **предоставление доступа** новым пользователям / группам пользователей и **изменение уровней доступа** производится на странице с информацией об объекте на *вкладке "Пользователи"*. Последовательность действий одинакова для всех каталогизируемых сущностей.

1. В каталоге в строке с названием нужного объекта наведите курсор на символ меню (три точки) и нажмите "Просмотр". Откроется страница с информацией об объекте.
2. На верхней панели выберите вкладку "Пользователи".
3. Чтобы *добавить* новых пользователей / группы в открывшемся окне нажмите "Добавить" и выберите необходимые имена учетных записей и подходящие роли из выпадающих списков.
4. Для *редактирования* роли или *удаления* доступа для пользователя, уже имеющего доступ к объекту, нажмите значок редактирования справа в строке с именем нужного пользователя. После того как редактирование завершено, нажмите значок сохранения.

Страница редактирования доступов для объекта типа "Проект":



3.2 Объекты

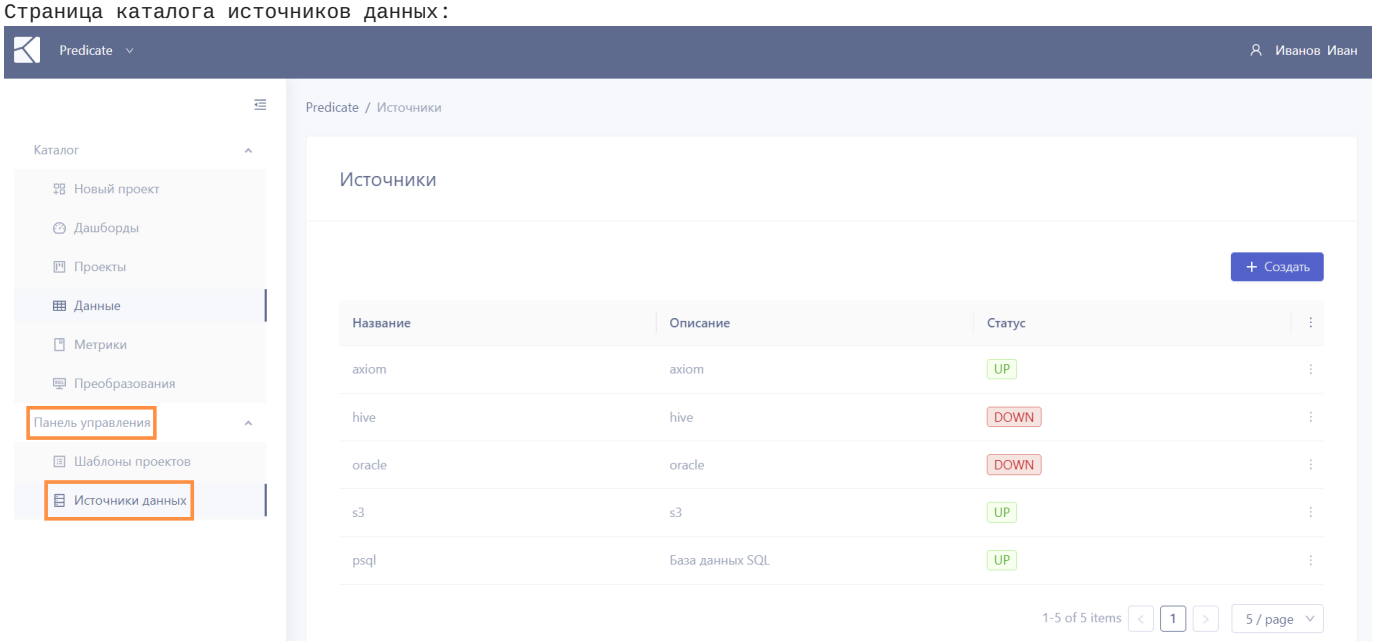
3.2.1 Источники данных

Predicate - Источники данных

ПЕРЕХОД К КАТАЛОГУ ИСТОЧНИКОВ ДАННЫХ

Переход к каталогу подключенных к системе источников данных осуществляется из основного меню: *Панель управления > Источники данных*.

Страница каталога источников данных:



ВОЗМОЖНЫЕ СТАТУСЫ ИСТОЧНИКА ДАННЫХ

UP - источник подключен и доступен.
DOWN - источник зарегистрирован, но не доступен.

УПРАВЛЕНИЕ КАТАЛОГОМ

В верхней части экранной формы доступны кнопки **сортировки каталога** по дате подключения или названию источника, в убывающем или возрастающем порядке.
По умолчанию источники в каталоге упорядочены по дате подключения к системе - от новых к более ранним.

Также доступен **поиск**.
Введите подстроку в поле поиска и нажмите `Enter` или символ лупы. Регистр не важен, поиск происходит по названию, описанию, ключу источников, а также по тэгам.

Каталог может содержать более одной страницы. Кнопки переключения между страницами, а также кнопка настройки количества объектов на странице, доступны в правом нижнем углу экранной формы.

ПРОСМОТР И СОЗДАНИЕ ОБЪЕКТОВ

Для **просмотра информации об источнике** наведите курсор на символ меню (три точки) справа в строке соответствующего источника и нажмите "Просмотр".

Для **подключения нового источника** нажмите кнопку "Создать" в правой верхней части окна каталога источников.
Подробнее - см. раздел "подключение нового источника данных".

Поддерживаемые источники данных

Источник данных – подключение к существующему хранилищу данных, позволяющее получить доступ к находящимся в хранилище таблицам данных для дальнейшего формирования датасетов, которые будут использоваться при создании проектов мониторинга.

Загрузка данных для проектов мониторинга происходит из реляционных источников, зарегистрированных в системе.

Продукт поддерживает следующие типы подключения:

- хранилище **s3** (хранение файлов типа .csv)
- базы данных **PostgreSQL**, **Oracle**, **Hive**
- feature store **Kolmogorov.Axiom**

Для использования в проекте мониторинга, датасет из источника данных должен быть зарегистрирован в системе согласно алгоритму регистрации датасета.

Примечание

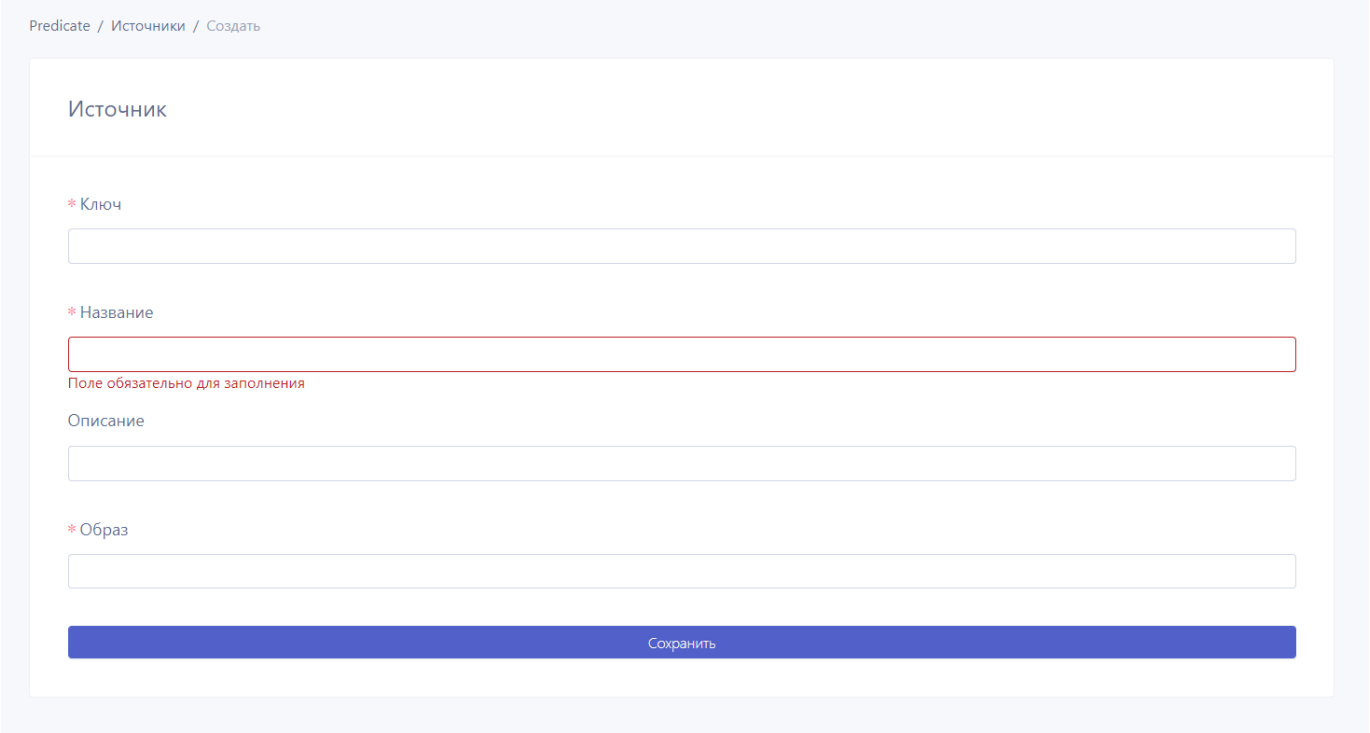
При создании проекта мониторинга на данных из динамично пополняемой таблицы PostgreSQL возможно настроить **границы по времени** для среза данных, используемого при расчете метрики.

Примечание

Загрузка данных с рабочих машин пользователей возможна в хранилища s3 в формате .csv.

Подключение нового источника данных

Регистрация нового источника данных доступна через интерфейс: *Панель управления > Источники данных > Создать*.



Параметры регистрации:

- Ключ (обязательное поле) - тип подключения (s3 / psql / oracle / hive / datasource-axiom).
- Название (обязательное поле) - название источника данных в системе.
- Описание (необязательное поле) - описание источника данных.
- Образ (обязательное поле) - используемый для источника образ (на его основе разворачивается api-сервер для источника, позволяющий скачивать из источника данные при исполнении проекта).

Пользователь сможет сохранить данные об источнике только после заполнения всех обязательных полей.

Предусловия: источник данных должен быть развернут и иметь доступ к приложению Predicate.

Результат регистрации: параметры источника данных становятся доступны через *Панель управления > Источники данных*. В списке источников отображается статус успешности подключения.

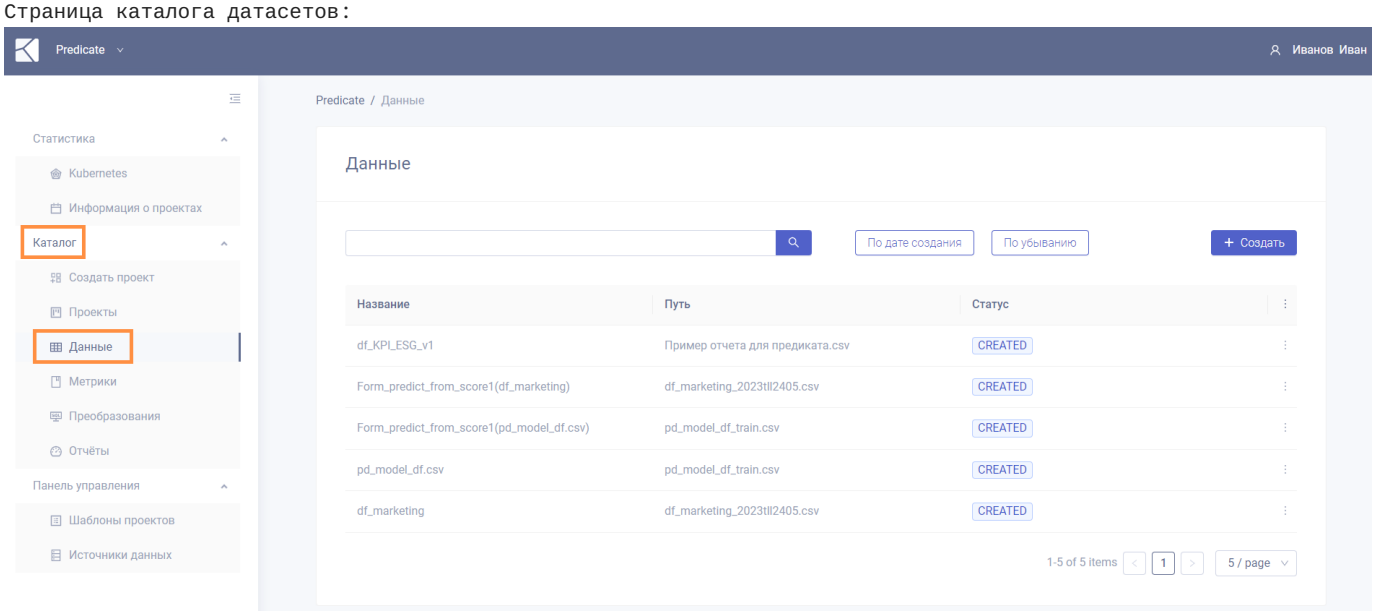
3.2.2 Данные

Predicate - Данные

ПЕРЕХОД К КАТАЛОГУ ДАТАСЕТОВ

Переход к каталогу зарегистрированных в системе датасетов осуществляется из основного меню: *Каталог* > *Данные*.

Страница каталога датасетов:



ИНФОРМАЦИЯ, ОТОБРАЖАЕМАЯ В КАТАЛОГЕ

- **Название** - название датасета в системе. Имя параметра вида *Название_преобразования(название датасета)* указывает на то, что датасет содержит столбцы, полученные путем преобразования другого датасета.
- **Путь** - указывает путь до датасета в источнике. Расширение .csv указывает на то, что датасет хранится в виде файла в s3 хранилище. Отсутствие расширения означает что датасету соответствует таблица в базе данных.
- **Статус** - статус успешности регистрации датасета.

ВОЗМОЖНЫЕ СТАТУСЫ ДАННЫХ

CREATED - датасет зарегистрирован (соответствующая запись создана в базе)

УПРАВЛЕНИЕ КАТАЛОГОМ

В верхней части экранной формы доступны кнопки **сортировки каталога** по дате регистрации или названию объекта данных, в убывающем или возрастающем порядке.
По умолчанию датасеты в каталоге упорядочены по дате регистрации в системе - от новых к более ранним.

Также доступен **поиск**.
Введите подстроку в поле поиска и нажмите `Enter` или символ лупы. Регистр не важен, поиск происходит по названию датасетов, названию, описанию, ключу соответствующих им источников данных, а также тэгам.

Каталог может содержать более одной страницы. Кнопки переключения между страницами, а также кнопка настройки количества объектов на странице, доступны в правом нижнем углу экранной формы.

ПРОСМОТР И СОЗДАНИЕ ОБЪЕКТОВ

Для **просмотра информации о датасете** наведите курсор на символ меню (три точки) справа в строке соответствующего датасета и нажмите "Просмотр".

Для **регистрации нового датасета** нажмите кнопку "Создать" в правой верхней части окна каталога датасетов. Подробнее – см. раздел "регистрация нового датасета".

Регистрация нового датасета

В данном разделе описан процесс регистрации датасета из источника, подключенного к системе. Подробнее про источники данных см.: "источники данных".

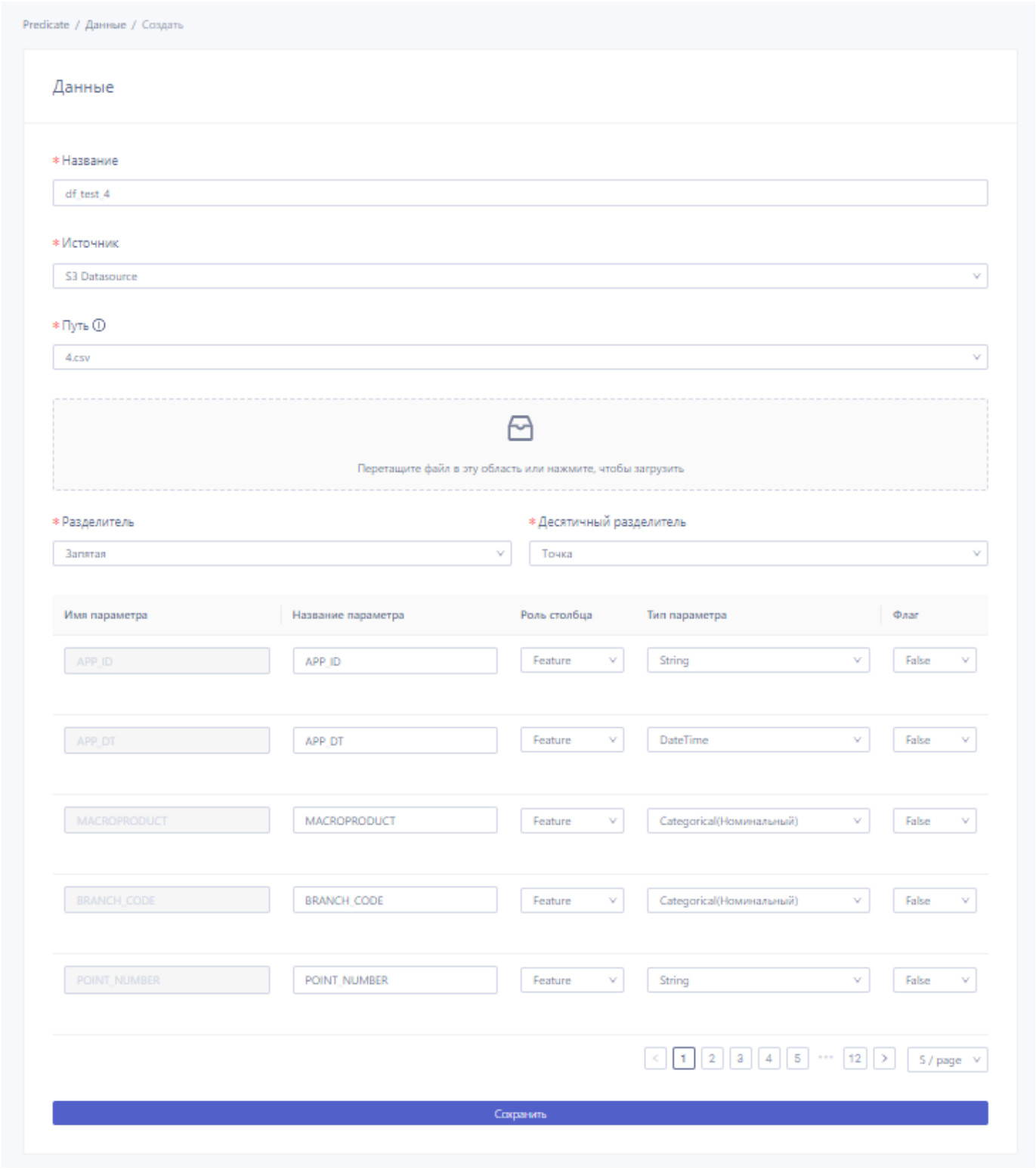
> **Важно**
>
> Для всех описанных ниже вариантов регистрации предусмотрено **ограничение** на объем датасета - **10 Гб**.

Регистрация данных из файла .csv

Загрузка данных с рабочих машин пользователя возможна в хранилища s3 в формате .csv.

Выберите функцию “Данные” в основном меню приложения. В открывшемся окне каталога данных нажмите кнопку "Создать".

Появится форма регистрации нового датасета:



Введите желаемое название датасета и тип источника данных:

- s3 – для файлов .csv

Загрузите файл с расширением .csv нажав на соответствующую область страницы.

Ограничение

Название, присваиваемое датасету при регистрации, и название загружаемого csv-файла не должны повторять названия уже имеющихся в системе объектов соответствующего типа.

Выберите символы-разделители:

- для столбцов в файле - "Разделитель";
- для отделения целой и десятичной частей числа - "Десятичный разделитель".

Загруженные данные необходимо разметить по их типу и по роли для датасета. Описание всех типов и ролей приведено на странице "разметка датасета".

Совет

Если в датасете есть столбец с датой/временем и предполагаются регламентные запуски проектов, измените значение в поле "Флаг" для этого столбца на True. Подробнее см.: "разметка датасета".

После завершения разметки нажмите "Сохранить".

Если загрузка прошла успешно, появится страница с подробной информацией о зарегистрированном датасете.

В дальнейшем к просмотру информации об этом датасете можно вернуться по схеме: *Каталог > Данные > [Выбор строки с названием датасета] > Просмотр (переход из меню в правой части строки)*

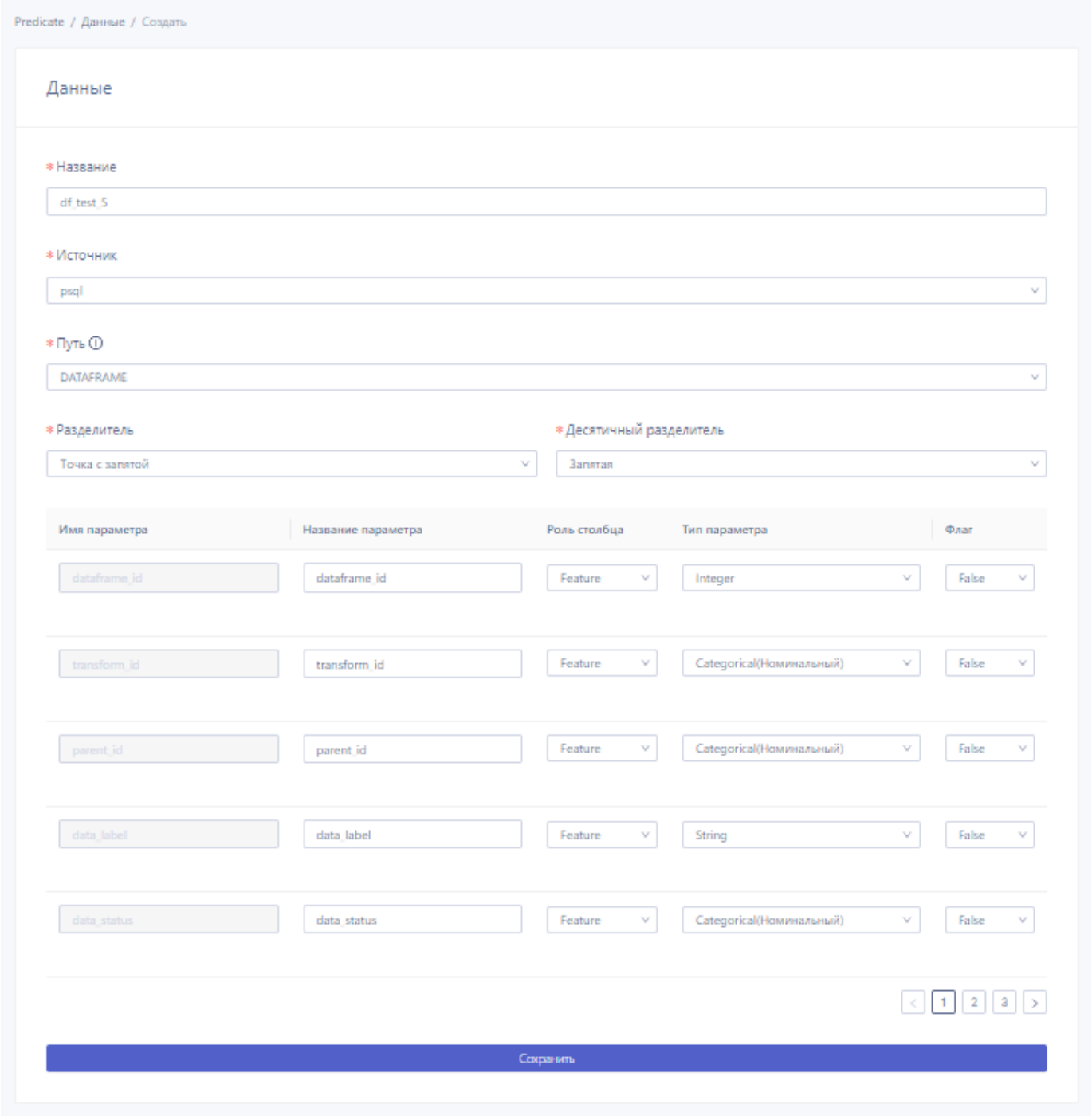
В каталоге данных отображается статус успешности регистрации датасета.

Регистрация данных из базы SQL

При регистрации таблицы базы данных SQL в качестве объекта данных в приложении Predicate возможно регулярное автоматическое обновление зарегистрированного датасета за счет изменения/добавления данных в таблицу любым стандартным для работы с БД способом.

Выберите функцию “Данные” в основном меню приложения. В открывшемся окне каталога данных нажмите кнопку "Создать".

Появится форма регистрации датасета:



Введите желаемое название датасета и тип источника данных:

- psql - для базы SQL

Ограничение

Название, присваиваемое датасету при регистрации, не должно повторять названия уже зарегистрированных в системе датасетов.

Заполните поле "Путь" выбрав из выпадающего списка таблиц БД название необходимой таблицы.

Выберите в поле "Десятичный разделитель" символ для отделения целой и десятичной частей числа, используемый в БД. В поле "Разделитель" оставьте значение по умолчанию.

Загруженные данные необходимо разметить по их типу и по роли для датасета. Описание всех типов и ролей приведено на странице "разметка датасета".

Совет

Если в датасете есть столбец с датой/временем и предполагаются регламентные запуски проектов, измените значение в поле "Флаг" для этого столбца на True. Подробнее см.: "разметка датасета".

После завершения разметки нажмите "Сохранить".

Если загрузка прошла успешно, появится страница с подробной информацией о зарегистрированном датасете.

В дальнейшем к просмотру информации об этом датасете можно вернуться по схеме: *Каталог > Данные > [Выбор строки с названием датасета] > Просмотр (переход из меню в правой части строки)*

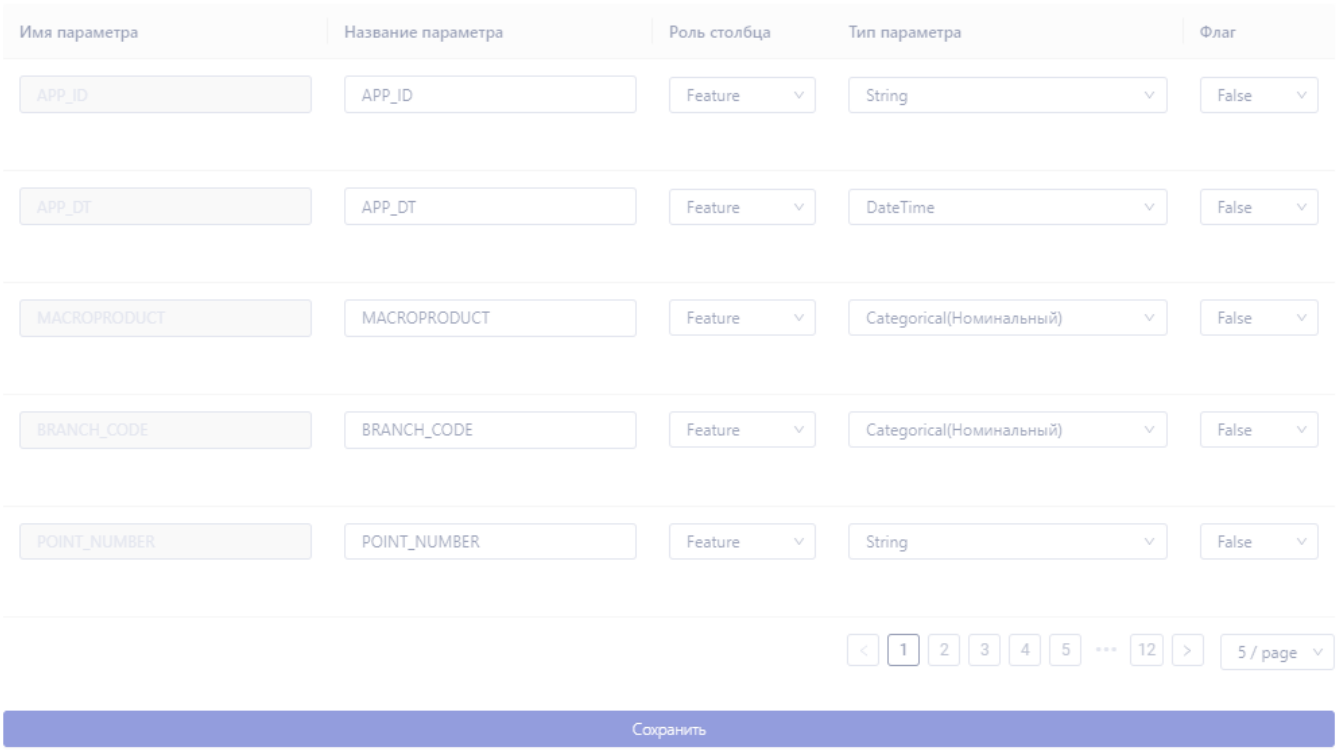
В каталоге данных отображается статус успешности регистрации датасета.

Разметка датасета

Разметка относится к отдельным **столбцам** датасета. Ниже приведено описание параметров разметки.

> **Важно**
>
> Разметка столбцов производится при регистрации датасета и не может быть изменена после завершения процесса регистрации.

ФОРМА РАЗМЕТКИ ДАТАСЕТА



Имя параметра

Название столбца в исходной таблице в источнике данных. Распознается автоматически, не может быть изменено пользователем.

Название параметра

Пользовательское имя столбца. По умолчанию совпадает с названием столбца в исходной таблице, но может быть изменено пользователем.

Это название используется при обращении к столбцу датасета в проектах мониторинга.

Роль столбца

Возможные роли столбцов:

№	Название	Описание
1	ID	Столбец с идентификатором записи в источнике.
2	DateTime	Дата и время. Например, переменная для указания времени сбора данных.
3	Feature	Переменная, используемая как входные данные для модели. Например, пол, возраст, заработок и так далее.
4	Prediction	Предсказания модели.
5	Target	Целевая переменная модели.
6	NonUse	Данные, которые не будут использоваться при работе с этим источником.

По умолчанию всем столбцам присваивается роль Feature.

Изменить роль можно выбрав соответствующее значение из выпадающего списка.

Роль столбца, указанная при разметке, влияет на отображение данного столбца в выпадающем списке возможных параметров метрик при создании проекта. Если в коде метрики для **параметра типа column** задано ограничение по роли (подробнее о методе with_role см. здесь), для передачи в метрику будут доступны только столбцы с соответствующей ролью.

Учитывая вышесказанное стоит внимательно относиться к разметке датасета. В частности, для столбцов, содержащих: дату и время, предсказания модели, целевые метки обязательно указывать соответствующие роли - DateTime, Prediction и Target соответственно.

Тип параметра

Возможные типы параметров:

№	Название	Описание
1	Integer	Целочисленный тип.
2	Float	Числа с плавающей точкой.
3	String	Текстовая информация.
4	DateTime	Дата и время.
5	Date	Дата.
6	Categorical (Номинальный)	Значения, группирующие информацию по категориям. Например, "м" и "ж" для пола.
7	Categorical (Порядковый)	Значения, группирующие информацию по упорядоченным категориям. Например, разряды рейтинговой шкалы.
8	Vector	Каждая запись представляет собой вектор (массив) значений.

Тип данных столбца распознается автоматически. Значение типа может быть изменено пользователем (доступен выбор из выпадающего списка).

Флаг

Флаг - отметка для регламента: True\False.

Столбец с датой, по которому предполагается фильтрация данных (настройка скользящего окна) при регламентных запусках проекта мониторинга, должен быть отмечен флагом **True** в разметке регистрируемого датасета.

Регламентный запуск проекта доступен и для датасетов без столбца с флагом True (в том числе для датасетов вообще без столбца с датой). В этом случае при каждом запуске проекта для расчета берутся все записи, имеющиеся в объекте данных на текущий момент.

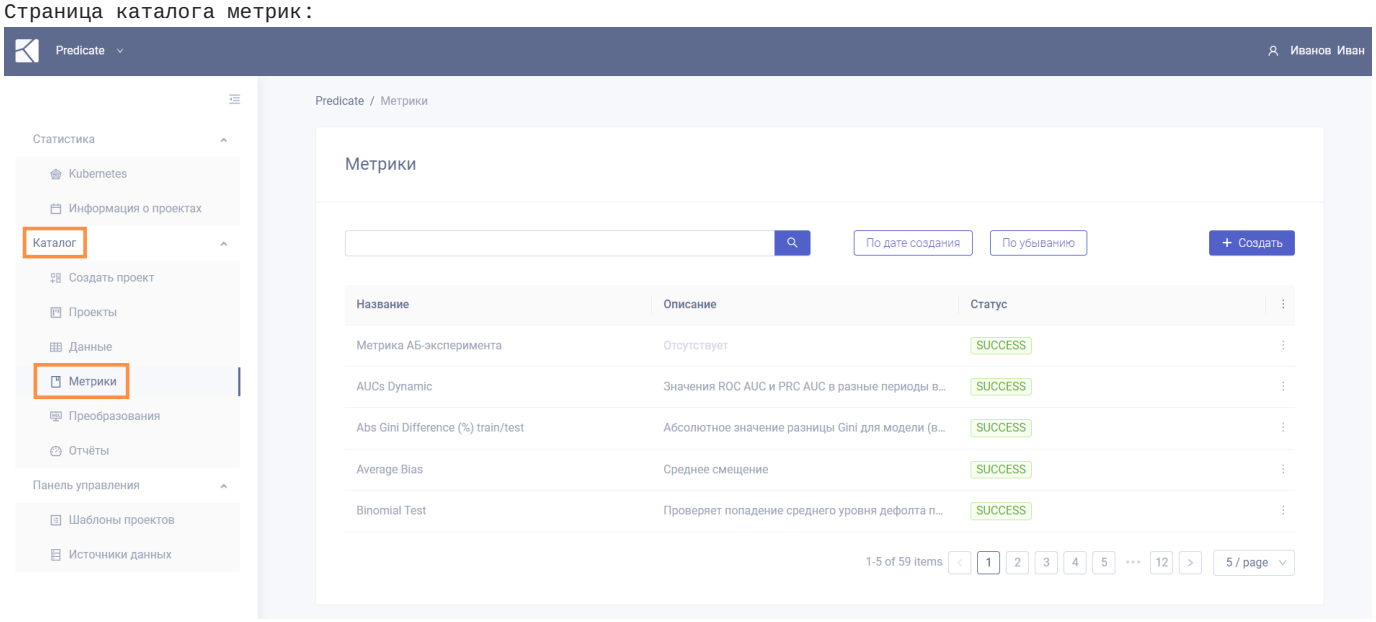
3.2.3 Метрики

Predicate - Метрики

ПЕРЕХОД К КАТАЛОГУ МЕТРИК

Переход к каталогу имеющихся в системе метрик/тестов осуществляется из основного меню: *Каталог > Метрики*.

Страница каталога метрик:



ВОЗМОЖНЫЕ СТАТУСЫ МЕТРИКИ

CREATED - запись о метрике с соответствующим названием создана в базе. Реализуется, например, когда файл с кодом метрики помещен в репозиторий "напрямую", а не через интерфейс приложения.
SUCCESS - файл с кодом метрики успешно загружен в Git-репозиторий.
FAILED - ошибка при загрузке файла с кодом метрики в Git.

УПРАВЛЕНИЕ КАТАЛОГОМ

В верхней части экранной формы доступны кнопки **сортировки каталога** по дате загрузки или названию метрики, в убывающем или возрастающем порядке.

Исходные метрики, загруженные в систему при установке, упорядочены в алфавитном порядке по названию.

Метрики, загружаемые пользователем через интерфейс приложения, попадают в конец каталога. Таким образом по умолчанию пользовательские метрики находятся в конце каталога и упорядочены по дате загрузки в систему - от новых к более ранним.

Также доступен **поиск**.
Введите подстроку в поле поиска и нажмите `Enter` или символ лупы. Регистр не важен, поиск происходит по названию и описанию метрик, названию классов метрик в исходном коде, а также тэгам.

Каталог может содержать более одной страницы. Кнопки переключения между страницами, а также кнопка настройки количества объектов на странице, доступны в правом нижнем углу экранной формы.

ПРОСМОТР И СОЗДАНИЕ ОБЪЕКТОВ

Для **просмотра информации о метрике** наведите курсор на символ меню (три точки) справа в строке соответствующей метрики и нажмите "Просмотр".

Для **регистрации новой метрики** нажмите кнопку "Создать" в правой верхней части окна каталога метрик. Подробнее - см. раздел "регистрация новой метрики".

Регистрация новой метрики

ПОДГОТОВКА ФАЙЛА МЕТРИКИ

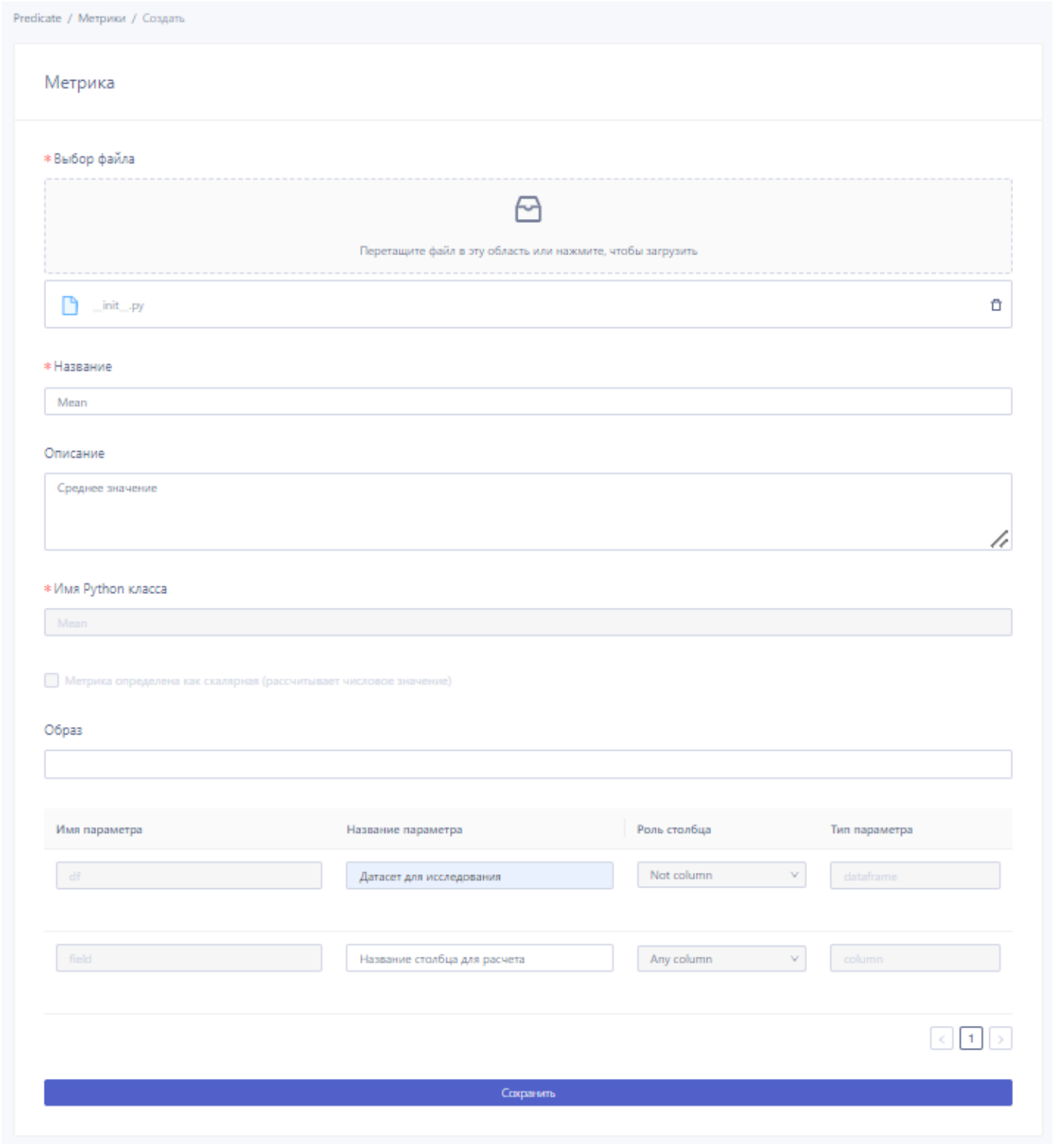
Загрузка метрики или теста в систему осуществляется из файла с расширением `.py`.

Подробнее о составлении кода метрики см. на странице "структура файла метрики".

ПЕРЕХОД К ФОРМЕ РЕГИСТРАЦИИ МЕТРИКИ

Для загрузки новой метрики перейдите в раздел “Метрики” основного меню приложения и на открывшейся странице каталога метрик и тестов нажмите кнопку "Создать".

Откроется форма загрузки новой метрики:



ЗАГРУЗКА ФАЙЛА МЕТРИКИ

В открывшейся форме необходимо загрузить python-файл с кодом метрики и заполнить поля:

- **Название** (обязательное поле) - название метрики в системе;
- **Описание** (необязательное поле) - описание метрики;
- **Имя Python класса** (обязательное поле) - название класса (id) метрики, при корректном скрипте заполняется автоматически;
- **Образ** (необязательное поле) - рекомендуется оставить настройки по умолчанию.

РАЗМЕТКА ВХОДНЫХ ПАРАМЕТРОВ МЕТРИКИ

При корректной обработке .py-скрипта открывается предзаполненная таблица с разметкой для входных параметров метрики.

- **Имя параметра** – заполняется автоматически, совпадает с названием параметра в .py-скрипте.
- **Название параметра** – заполняется автоматически, соответствует описанию параметра в блоке описания класса метрики. Допускается изменение этого поля на этапе регистрации метрики в системе.
- **Роль столбца** – для параметров типа column – роль в датасете (Feature / Target / Prediction / ID / Datetime / Any column), для параметров других типов – Not column. Заполняется автоматически согласно значению, переданному в метод .with_role() при объявлении параметра типа column в коде метрики (возможные для передачи значения перечислены в таблице 1 ниже). Если значение не было передано – столбцу присваивается роль Any column.
- **Тип параметра** – тип входного параметра. Заполняется автоматически согласно типу параметра, указанному в коде метрики. Поддерживаемые типы входных параметров приведены ниже в таблице 2.

Таблица 1. Роли столбцов, которые могут быть переданы в метод .with_role() в коде метрики.

№	Название	Описание
1	DATETIME	Столбец с датой и(или) временем
2	FEATURE	Столбец, содержащий признак
3	ID	Столбец с id записей
4	PREDICTION	Столбец с предсказаниями модели
5	TARGET	Столбец с целевой переменной

Таблица 2. Предусмотренные в системе **типы входных параметров метрик**.

№	Название	Описание
1	dataframe	Датасет для исследования. Каждая метрика должна иметь хотя бы один параметр типа dataframe.
2	column	Название столбца из dataframe, над которым производится расчет. Метрика может принимать на вход один или несколько параметров типа column.
3	multi-column	Список названий столбцов для расчета. Используется при необходимости рассчитать одну и ту же метрику над разными столбцами.
4	range	Список из двух вещественных чисел – границ светофора для значения метрики.
5	int-value	Натуральное число.
6	float-value	Вещественное число.
7	bool	Параметр-флаг. Если он равен True, реализуется один сценарий исполнения метрики; если False – другой сценарий.
8	date	Дата.
9	time	Время.
10	dropdown	Выпадающий список из нескольких значений.
11	multi-dropdown	Выпадающий список с возможностью множественного выбора.

Подробнее: >>> **Типы параметров метрик и их использование** <<<

Для завершения процесса загрузки метрики необходимо нажать кнопку "Сохранить".

ПРОСМОТР СОЗДАННОЙ МЕТРИКИ

После того как метрика создана, соответствующая ей запись появится в списке тестов, зарегистрированных в системе (*Каталог > Метрики*).

При успешном статусе загрузки кода метрика становится доступной для создания проектов с ее использованием.

Структура файла метрики

Код метрики должен быть написан на Python в соответствии с приведенной ниже структурой и сохранен в файле с расширением `.py`.

Примеры кода метрики доступны на отдельной странице.

Далее перечислены правила конструирования класса метрики.

1. ФОРМАТ

Файл с исходным кодом для метрики должен быть создан по следующему пути: **metrics**/`metric_key`/__init__.py

В файле `__init__.py` должен быть класс, унаследованный от строго одного из двух возможных типов:

- `ScalarMetric` для метрик, результат которых является скалярным значением - просто числом (например, MAE, RMSE и т.п.)
- `Metric` для метрик, результат которых не представляется в виде одного числа (например, confusion matrix)

Импортируются базовые классы строго следующим образом:

```
from klmg_predicate_types.metric import Metric, ScalarMetric
```

Имя класса метрики должно быть идентичным `metric_key`.

Пример: создаётся скалярная метрика MAPE (Mean Absolute Percentage Error).
Путь файла с её исходным кодом в этом репозитории должен быть: `metrics/MAPE/__init__.py`
Каркас для логики в этом файле должен выглядеть так:

```
from klmg_predicate_types.metric import ScalarMetric


class MAPE(ScalarMetric):
    ...
```

Далее **логика расчёта метрики задаётся путём определения в теле класса ряда методов**.

2. МЕТОДЫ

2.1. set_params

Данный метод должен определяться первым в классе.
Точка входа в исполнение метрики, в этот метод будут передаваться значения параметров из командной строки.
Базовая сигнатура:

```
def set_params(self, *, **kwargs: typing.Any) -> None:
    ...
```

Определение метода `set_params` в классе метрики должно соответствовать следующим правилам:

- метод принимает только keyword аргументы - параметры метрики
- каждый параметр должен иметь аннотацию его типа с использованием **библиотеки типов**
- значение параметра по умолчанию не должно нарушать логику валидации указанного типа параметра
- для извлечения значений параметров в теле метода нужно использовать атрибут `value`, имеющийся у каждого параметра
- передавать значения параметров в следующие методы нужно через атрибуты экземпляра класса (`self`), не используя при этом зарезервированные имена: `label` и `chart` для всех метрик, и ещё дополнительно `scalar` и `signal` для скалярных

>>> **Типы параметров метрик и их использование** <<<

Для загрузки кода метрики через UI Predicate на текущий момент необходимо, чтобы обращение к типам параметров всегда было в виде `param.<тип>`. Поэтому импортировать модуль с типами параметров нужно строго следующим образом:

```
from klmg_predicate_types.metric import param
```

Продолжая пример с MAPE:

```
from klmg_predicate_types.metric import param, ScalarMetric


class MAPE(ScalarMetric):
    def set_params(
        self,
        *,
        df: param.Dataframe,
        score: param.Column,
        target: param.Column,
        signal_config: param.Signal.s(higher_is_better=False, domain=(0, 1)) = {
            "threshold_yellow": 0.3,
            "threshold_red": 0.4,
        },
    ):
        self.score = score.value
        self.target = target.value
        self.df = df.value.astype({self.score: "float", self.target: "float"})
        self.signal_config = signal_config.value
```

Таким образом, MAPE принимает на вход:

- `pandas` датафрэйм с данными для расчёта
- имя колонки с предсказанием
- имя колонки с целевой переменной (реальным значениями)
- настройки простого светофора

У настроек простого сфетофора в данном примере есть значение по умолчанию.

В теле метода экземпляру метрики (`self`) присваиваются значения параметров (получаемые через атрибут `value`) для использования в остальных методах.

Конфигурация higher_is_better

Часто по метрике однозначно понятно, что лучше - большее или меньшее значение. Например, MAPE - это ошибка, значит меньше - лучше. Для ROC AUC, напротив, предпочтительно большее значение.
Чтобы **зафиксировать направление индикатора скалярного значения** следует в определении параметра типа `Signal` передать аргументу `higher_is_better` подходящее булево значение: `True` если лучше большие значения метрики, иначе - `False`. В качестве примера см. определение параметра `signal_config` метрики MAPE выше. Далее можно указать значения по умолчанию для желтой и красной границ светофора.

Если предпочтительность больших или меньших значений не определена однозначно (например, для метрики Mean), можно **конфигурировать направление индикатора скалярного значения через параметры метрики**.
Следует добавить параметр типа `Bool`, а затем вызвать метод `self.set_evaluation` и передать ему в качестве аргумента `higher_is_better` значение этого параметра (обязательно как именованный аргумент):

```
from klmg_predicate_types.metric import param, ScalarMetric


class Mean(ScalarMetric):
        def set_params(
            self,
            *,
            df: param.Dataframe,
            field: param.Column,
            higher_is_better: param.Bool,
            signal_config: param.Signal.s(higher_is_better=None),
        ):
            self.field = field.value
            self.df = df.value.astype({self.field: "float"})
            self.signal_config = signal_config.value
            self.set_evaluation(higher_is_better=higher_is_better.value)
```

В примере выше метрика Mean принимает на вход:

- `pandas` датафрэйм с данными для расчёта
- имя колонки для расчёта
- булево значение
- настройки простого светофора

У настроек простого сфетофора в данном случае нет значений по умолчанию, направление индикатора определяется полученным булевым значением:

- если `True` - лучшим считается большее скалярное значение метрики, красная граница сфетофора меньше желтой;
- если `False` - лучшим считается меньшее скалярное значение метрики, красная граница светофора больше желтой.

2.2. scalar_value (только для скалярных метрик)

В этом методе скалярной метрики вычисляется числовое значение результата. Его базовая сигнатура:

```
def scalar_value(self) -> typing.Union[int, float]:
    ...
```

- возвращаемое этим методом значение присваивается атрибуту `self.scalar`
- атрибут `self.scalar` будет доступен в следующих методах

Продолжая пример с MAPE:

```
from sklearn.metrics import mean_absolute_percentage_error


class MAPE(ScalarMetric):
    ...

    def scalar_value(self):
        df = self.df.dropna()[abs(self.df[self.target]) > 0]

        return mean_absolute_percentage_error(
            y_pred=df[self.score],
            y_true=df[self.target],
        )
```

Для нескалярных метрик (унаследованных от `Metric`) этот метод определять не нужно. Если определить, использоваться он не будет.

2.3. traces

Этот метод возвращает **последовательность** графиков, инициализируемых с помощью контроллеров из **библиотеки типов**. Его базовая сигнатура:

```
def traces(self) -> typing.Sequence[klmg_predicate_types.plotly.TraceController]:
    ...
```

>>> **Типы контроллеров и примеры визуализации** <<<

В примере про MAPE нужен один элемент визуализации - индикатор скалярного значения со сравнением относительно красной границы светофора. Базовый класс `ScalarMetric` уже содержит в себе встроенный атрибут `indicator`, содержащий именно такой объект для визуализации:

```
class MAPE(ScalarMetric):
    ...

    def traces(self):
        return (
            self.indicator,
        )
```

Без использования встроенного атрибута это выглядело бы примерно так:

```
from klmg_predicate_types.plotly.controller import Indicator
```

```
class MAPE(ScalarMetric):
    ...

    def traces(self):
        return (
            Indicator(
                value=self.scalar,
                mode="number+delta",
                reference=self.signal_config.threshold_red,
                higher_is_better=False,
            ),
        )
```

Это чтобы показать, как пользоваться контроллерами в общем виде. Но для индикатора скалярного значения метрики следует использовать её встроенный атрибут `indicator`.

2.4. custom_layout (опционально)

Этот метод может возвращать пользовательский layout для Plotly. Его базовое определение:

```
def custom_layout(self) -> Optional[Dict[str, Any]]:
    return None
```

Переопределение этого метода в классе метрики позволяет настроить layout для всех traces, возвращаемых соответствующим методом. В примере про MAPE кастомизируется только заголовок графика:

```
class MAPE(ScalarMetric):
    ...

    def custom_layout(self):
        return {
            "title": f"<b>Mean Absolute Percentage Error (MAPE)</b><br><br>"
            f"score column: {self.score}<br>target column: {self.target}"
        }
```

3. КЛАССОВЫЕ АТРИБУТЫ

3.1. label (опционально)

Можно определить классовый атрибут `label`, чтобы явно указать название метрики. При наличии это поле будет использоваться во время создания метрики через UI Predicate.

4. ПОРЯДОК ИСПОЛНЕНИЯ МЕТОДОВ

В рантайме методы метрики исполняются в том же порядке, в котором они перечислены выше.

1. `set_params`
2. `scalar_value` (если метрика скалярная, иначе нет такого метода)
3. `traces`
4. `custom_layout`

Это чтобы понимать, из какого метода можно присвоить атрибуты экземпляру класса и потом получить их значения в других методах.

Параметры метрик Predicate

Модуль: `klmg_predicate_types.metric.param`

Содержит типы для парсинга и валидации значений параметров метрик Predicate.
Это классы, обладающие свойствами `pydantic` модели.

1. Поддерживаемые типы параметров

Экземпляр любого из типов создаётся с передачей значения параметра метрики через keyword `value`.

Соотвественно, забирать значение можно также по атрибуту `value`.

1.1. Простые типы

Пример:

```
>>> import pandas as pd
>>> from klmg_predicate_types.metric import param
>>> df = pd.Dataframe({"x": [0, 1, 2, 3], "y": [0, 1, 4, 9]})
>>> df_test = param.Dataframe(value=df)
>>> df_test.value.head()
   x  y
0  0  0
1  1  1
2  2  4
3  3  9
```

Если инициализировать значением неверного типа или формата:

```
>>> df_test = param.Dataframe(value="это_строка_а_не_датафрэйм")
```

```
pydantic.error_wrappers.ValidationError: 1 validation error for Dataframe
value
  instance of DataFrame expected
```

Сводная таблица простых типов:

label	class	тип `value`
dataframe	`DataFrame`	`pandas.DataFrame`
multi-column	`MultiColumn`	`typing.Sequence[str]`
bool	`Bool`	`bool`
date	`Date`	`datetime`
time	`Time`	`time`

1.2. Конфигурируемые типы

Конфигурируемые типы (некоторые обязательно) настраиваются на сужение диапазона принимаемых значений.

Пример:

```
>>> chosen_option = param.Dropdown(value="option")
>>> chosen_option.value
'option'
```

Код выше отработает корректно.

```
>>> ConfiguredDropdown = param.Dropdown.of("option_1", "option_2")
>>> chosen_option = ConfiguredDropdown(value="option_3")
```

```
pydantic.error_wrappers.ValidationError: 1 validation error for Dropdown
value
  unexpected value; permitted: 'option_1', 'option_2'
```

А тут будет возвращена ошибка, так как для параметра явно заданы допустимые значения.

Сводная таблица конфигурируемых типов:

label	class	тип `value` без конфигурации	обязательная конфигурация	сигнатура конфигурации
column	`Column`	`str`	нет	`Column.with_role(role: str = column_role.ANY_COLUMN)`
float-value	`FloatValue`	`float`	нет	`FloatValue.between(left: typing.Optional[float] = None, right: typing.Optional[float] = None`
int-value	`IntValue`	`int`	нет	`IntValue.between(left: typing.Optional[int] = None, right: typing.Optional[int] = None)`
range	`Range`	`typing.Tuple[float, float]`	нет	`Range.between(left: typing.Optional[float] = None, right: typing.Optional[float] = None)`
dropdown	`Dropdown`	`typing.Union[str, float, int]`	***да**	`Dropdown.of(*options: typing.Union[str, float, int])`
multi-dropdown	`MultiDropdown`	`typing.Sequence[typing.Union[str, float, int]]`	***да**	`MultiDropdown.of(*options: typing.Union[str, float, int])`
signal	`Signal`	объект `threshold_yellow: float` `threshold_red: float`	***да**	`Signal.s(higher_is_better: typing.Optional[bool], domain: typing.Optional[typing.Tuple[float, float]] = None`

2. Вспомогательные методы

2.1. Получение label

Позволяет не прописывать в явном виде значения меток разных типов в разных местах приложения, а потом где-нибудь забыть поменять.

Сигнатура:

```
cls.get_label() -> str
```

Пример:

```
>>> label = param.FloatValue.get_label()
>>> type(label)
<class 'str'>
>>> label
'float-value'
```

2.2. Конструктор контекста

Для удобного создания контекста дефолтных сущностей.

Сигнатура:

```
cls.make_context(default: typing.Any) -> typing.Dict[str, typing.Any]
```

Значение `default` валидируется в соответствии с аннотацией типа `value` (и в соответствии с конфигурацией для конфигурируемых типов).

Пример с простым типом:

```
>>> context = param.Column.make_context(default="y_true")
>>> type(context)
<class 'dict'>
>>> context
{'default': 'y_true'}
```

Пример с конфигурируемым типом:

```
>>> Signal = param.Signal.s(higher_is_better=False, domain=(0, 1))
>>> context = Signal.make_context()
>>> type(context)
<class 'dict'>
>>> context
{'config': {'higher_is_better': False, 'domain': (0.0, 1.0)}}
```

Графики Plotly

Модуль: `klmg_predicate_types.plotly.controller`

Содержит контроллеры для создания графиков Plotly, используемых для **визуализации результатов метрик** в Predicate. Каждый инициализированный контроллер имеет атрибут `data`, содержащий объект с данными для одного графика (то что называется trace в терминологии Plotly).

Входные данные, поступающие в контроллер, валидируются с помощью `pydantic`. Пример валидации:

```
>>> scatter = controller.Scatter(x=[0, 1, 2, 3], y=[0, 1, 4, 9, 16, 25])
```

```
pydantic.error_wrappers.ValidationError: 1 validation error for SCATTER
__root__
  Lengths of "x" (4) and "y" (6) sequences must be equal.
```

Поддерживаемые типы графиков

1. Bar

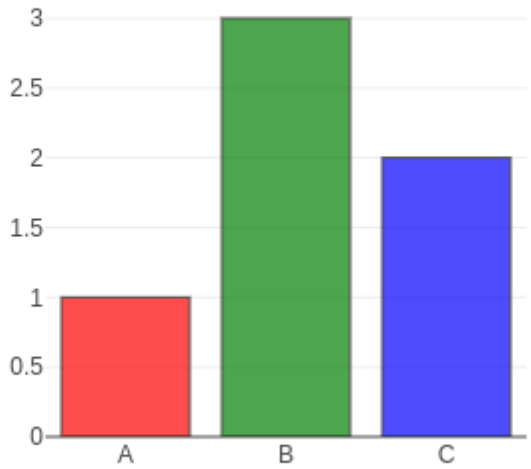
Сигнатура:

```
controller.Bar(
    values: typing.Sequence[float],
    labels: typing.Sequence[str],
    color: typing.Optional[typing.Union[typing.Sequence[str], str]] = None,
    name: typing.Optional[str] = None,
)
```

Пример:

```
import json
from klmg_predicate_types.plotly import controller

bar = controller.Bar(
    values=[1, 3, 2],
    labels=["A", "B", "C"],
    color=["red", "green", "blue"],
)

with open("data.json", "w") as handle:
    json.dump(bar.data, handle)
```

```
const fs = require('fs');
const { Plotly } = require("node-kernel");
const layout = {"height": 400, "width": 400};

let text = fs.readFileSync("data.json");
let data = JSON.parse(text.toString());
Plotly.newPlot("plot", [data], layout);
```



2. Heatmap

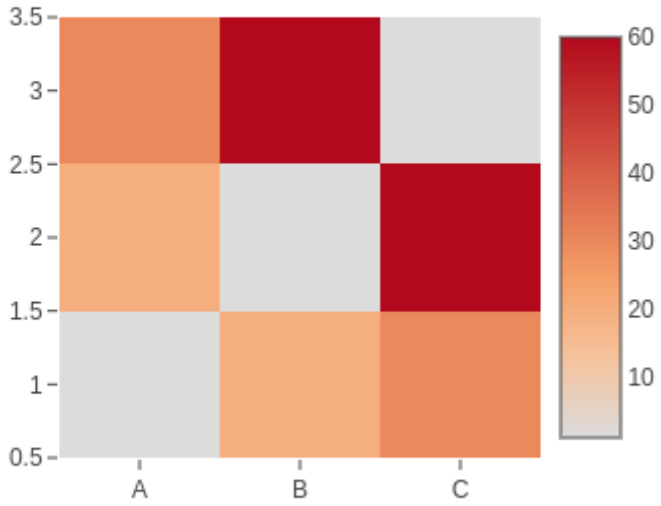
Сигнатура:

```
controller.Heatmap(
    matrix: typing.Sequence[typing.Sequence[float]],
    labels_x: typing.Optional[typing.Sequence[typing.Union[float, str]]] = None,
    labels_y: typing.Optional[typing.Sequence[typing.Union[float, str]]] = None,
)
```

Пример:

```
heatmap = controller.Heatmap(
    matrix=[
        [1, 20, 30],
        [20, 1, 60],
        [30, 60, 1],
    ],
    labels_x=["A", "B", "C"],
    labels_y=[1, 2, 3],
)

with open("data.json", "w") as handle:
    json.dump(heatmap.data, handle)
```

```
text = fs.readFileSync("data.json");
data = JSON.parse(text.toString());
Plotly.newPlot("plot", [data], layout);
```



3. Indicator

Сигнатура:

```
controller.Indicator(
    value: float,
    mode: typing.Literal["number", "number+delta"] = "number",
    reference: typing.Optional[float] = None,
    higher_is_better: typing.Optional[bool] = True,
)
```

Пример:

```
indicator = controller.Indicator(
    value=0.75,
    mode="number+delta",
    reference=0.5,
    higher_is_better=False,
)

with open("data.json", "w") as handle:
    json.dump(indicator.data, handle)
```

```
text = fs.readFileSync("data.json");
data = JSON.parse(text.toString());
Plotly.newPlot("plot", [data], layout);
```

4. Scatter

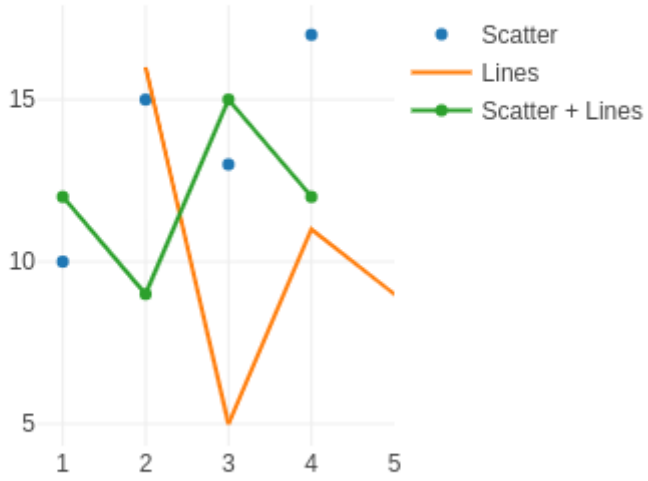
Сигнатура:

```
controller.Scatter(
    x: typing.Union[typing.Sequence[float], typing.Sequence[str]],
    y: typing.Sequence[float],
    name: typing.Optional[str] = None,
    mode: typing.Literal["lines", "markers", "lines+markers"] = "lines",
    line_color: typing.Optional[str] = None,
    line_width: typing.Optional[float] = None,
    line_dash: typing.Optional[typing.Literal["solid", "dashdot"]] = None,
    marker_color: typing.Optional[str] = None,
    marker_size: typing.Optional[float] = None,
)
```

Пример:

```
sc1 = controller.Scatter(
  x=[1, 2, 3, 4],
  y=[10, 15, 13, 17],
  mode="markers",
  name="Scatter",
)

sc2 = controller.Scatter(
  x=[2, 3, 4, 5],
  y=[16, 5, 11, 9],
  mode="lines",
  name="Lines",
)

sc3 = controller.Scatter(
  x=[1, 2, 3, 4],
  y=[12, 9, 15, 12],
  mode="lines+markers",
  name="Scatter + Lines",
)

data = [sc1.data, sc2.data, sc3.data]

with open("data.json", "w") as handle:
    json.dump(data, handle)
```

```
text = fs.readFileSync("data.json");
data = JSON.parse(text.toString());
Plotly.newPlot("plot", data, layout);
```



5. Custom

Сигнатура:

```
controller.Custom(**kwargs: typing.Any)
```

Позволяет создать пользовательский тип с любой структурой объекта данных для графика.
Принимает keyword аргументы с любым уровнем вложенности.

Примеры кода метрик

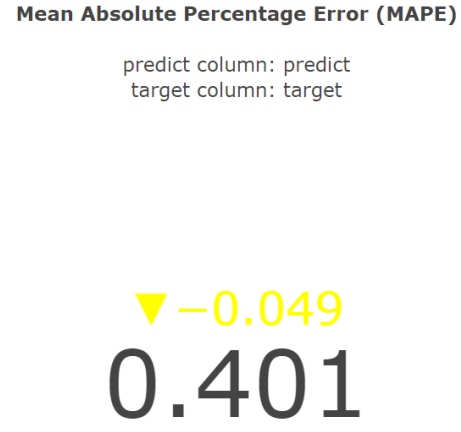
Просмотр python-кода имеющихся в системе метрик доступен из интерфейса приложения:

1. На странице каталога метрик (Каталог > Метрики) выберите нужную метрику, наведите курсор на символ меню (три точки) в правой части соответствующей строки и нажмите “Просмотр”.
2. Откроется экранная форма с информацией о метрике. Прокрутите страницу вниз и нажмите кнопку "Открыть код" в правом нижнем углу экранной формы.

Пример скалярной метрики с выводом типа 'indicator + delta'

Наследование от `ScalarMetric`. Указаны границы светофора. "Дельта" рассчитывается как значение разницы между полученным значением метрики и красной границей светофора.

```
from typing import Any, Dict, Iterable, Optional

from klmg_predicate_types.constant import column_role
from klmg_predicate_types.metric import ScalarMetric, param
from klmg_predicate_types.plotly import TraceController
from sklearn.metrics import mean_absolute_percentage_error


class MAPE(ScalarMetric):
    """
    Средняя абсолютная ошибка в процентах

    df: Датасет для исследования
    predict: Название столбца с предсказаниями модели
    target: Название столбца с целевой переменной
    signal_config: Параметры светофора
    """

    label = "Mean Absolute Percentage Error"

    def set_params(
        self,
        *,
        df: param.Dataframe,
        predict: param.Column.with_role(column_role.PREDICTION),
        target: param.Column.with_role(column_role.TARGET),
        signal_config: param.Signal.s(higher_is_better=False, domain=(0, 1)) = {
            "threshold_yellow": 0.3,
            "threshold_red": 0.45,
        },
    ):
        self.predict = predict.value
        self.target = target.value
        self.df = df.value.astype({self.predict: "float", self.target: "float"})
        self.signal_config = signal_config.value

    def scalar_value(self):
        if self.df.empty:
            raise Exception("Dataframe is empty")
        df = self.df.loc[:, [self.target, self.predict]].dropna()[abs(self.df[self.target]) > 0]

        return mean_absolute_percentage_error(
            y_pred=df[self.predict],
            y_true=df[self.target],
        )

    def traces(self) -> Iterable[TraceController]:
        return (self.indicator,)

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": f"<b>Mean Absolute Percentage Error (MAPE)</b><br><br>"
            f"predict column: {self.predict}<br>target column: {self.target}"
        }
```



Пример скалярной метрики с выводом типа 'indicator без дельты'

Наследование от `ScalarMetric`. Направление индикатора регулируется через параметр логического типа `higher_is_better`. В данном примере цветовой сигнал светофора не выводится вместе со скалярным значением метрики (`mode="number"` для `Indicator` в методе `traces`), однако границы светофора будут отображены в разделе Dynamic дашборда проекта.

```
from typing import Any, Dict, Iterable, Optional

import pandas as pd
from klmg_predicate_types.metric import ScalarMetric, param
from klmg_predicate_types.plotly import TraceController
from klmg_predicate_types.plotly.controller import Indicator


class Min(ScalarMetric):
    """
    Минимальное значение в выбранном столбце

    df: Датасет для исследования
    field: Название столбца для расчета
    higher_is_better: Большее скалярное значение метрики - лучше
    signal_config: Параметры светофора
    """

    label = "Min"

    def set_params(
        self,
        *,
        df: param.Dataframe,
        field: param.Column,
        higher_is_better: param.Bool,
        signal_config: param.Signal.s(None),
    ):
        self.field = field.value
        self.df = df.value.astype({self.field: "float"})
        self.signal_config = signal_config.value
        self.set_evaluation(higher_is_better=higher_is_better.value)

    def scalar_value(self):
        if self.df.empty:
            raise Exception("Dataframe is empty")
        if not pd.api.types.is_numeric_dtype(self.df[self.field]):
            raise Exception("Not numeric type")
        metric_result = float(self.df[self.field].min(skipna=True))
        return metric_result

    def traces(self) -> Iterable[TraceController]:
        return (Indicator(value=self.scalar, mode="number"),)
```

```
    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {"title": f'<b>Min</b> for column "{self.field}"'}
```

Min for column "y_pred"

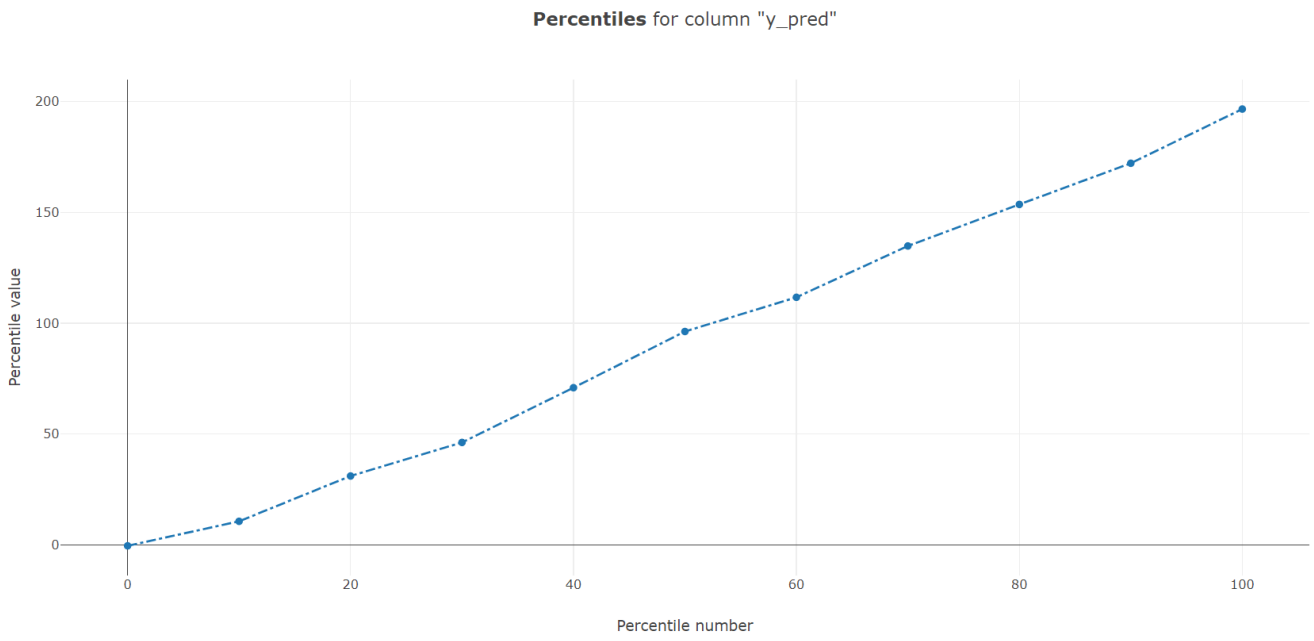
−0.466

Пример кода метрики с выводом типа 'line'

Наследование от `Metric`.

```
from typing import Any, Dict, Iterable, Optional

import numpy as np
from klmg_predicate_types.metric import Metric, param
from klmg_predicate_types.plotly import TraceController
from klmg_predicate_types.plotly.controller import Scatter


class Percentiles(Metric):
    """
    Линейный график в осях номер-значение перцентиля для выбранного столбца

    df: Датасет для исследования
    field: Название столбца для расчета
    """

    label = "Percentiles"

    def set_params(
        self,
        *,
        df: param.Dataframe,
        field: param.Column,
    ) -> None:
        self.field = field.value
        self.df = df.value.astype({self.field: "float"})

    def traces(self) -> Iterable[TraceController]:
        if self.df.empty:
            raise Exception("Dataframe is empty")
        q = np.arange(0, 110, 10)
        x_values = q.tolist()
        y_values = np.percentile(self.df[self.field].dropna(), q).tolist()

        sc1 = Scatter(
            x=x_values,
            y=y_values,
            mode="lines+markers",
            line_dash="dashdot",
            marker_size=7,
        )

        return (sc1,)

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": f'<b>Percentiles</b> for column "{self.field}"',
            "xaxis": {"title": "Percentile number", "side": "left"},
```

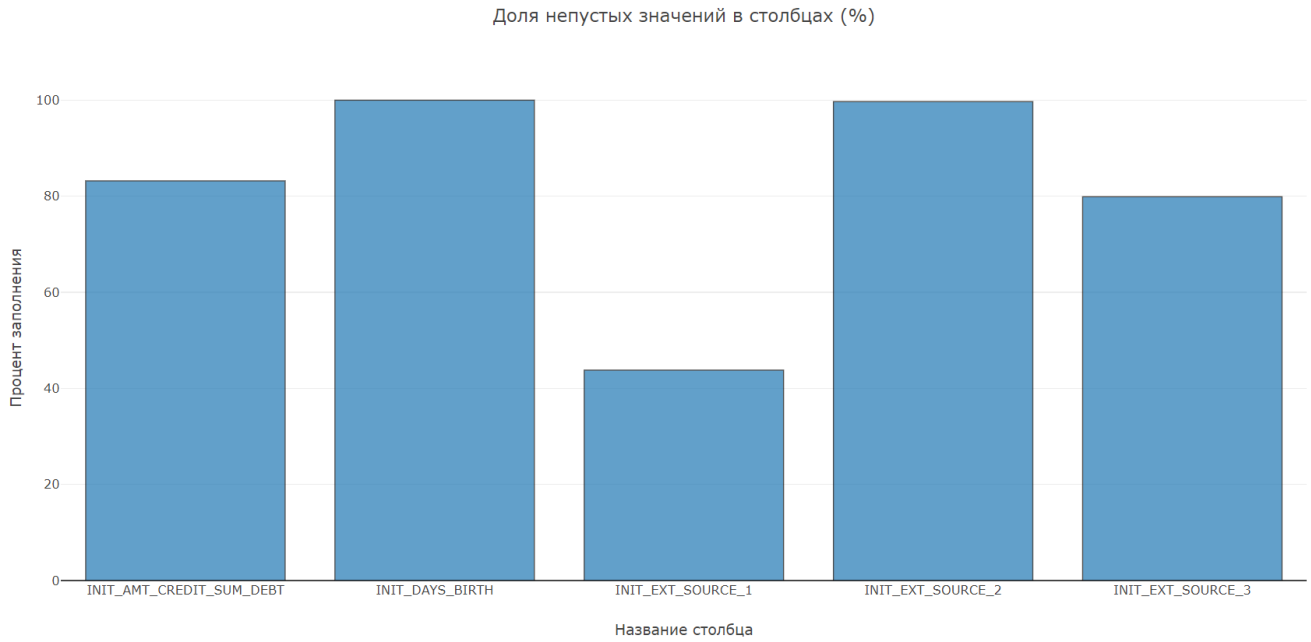
```
            "yaxis": {"title": "Percentile value", "side": "left"},
        }
```



Пример кода метрики с выводом типа 'bar'

Наследование от `Metric`.

```
from typing import Any, Dict, Optional, Sequence

from klmg_predicate_types.metric import Metric, param
from klmg_predicate_types.plotly import TraceController
from klmg_predicate_types.plotly.controller import Bar


class FillPct(Metric):
    """
    Barchart с процентом не-null значений в выбранных столбцах

    df: датасет для исследования
    fields_to_test: массив названий столбцов для расчета
    """

    label = "Fill Percent for Features"

    def set_params(
        self,
        *,
        df: param.Dataframe,
        fields_to_test: param.MultiColumn,
    ) -> None:
        self.df = df.value
        self.fields_to_test = fields_to_test.value

    def traces(self) -> Sequence[TraceController]:
        self.df = self.df[self.fields_to_test]

        rep = round(100 - (self.df.isna().sum() / self.df.shape[0] * 100), 1)
        labels = rep.index.tolist()
        values = rep.tolist()

        return (Bar(values=values, labels=labels),)

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": "Доля непустых значений в столбцах (%)",
            "xaxis": {"title": "Название столбца", "side": "left"},
            "yaxis": {"title": "Процент заполнения", "side": "left"},
        }
```

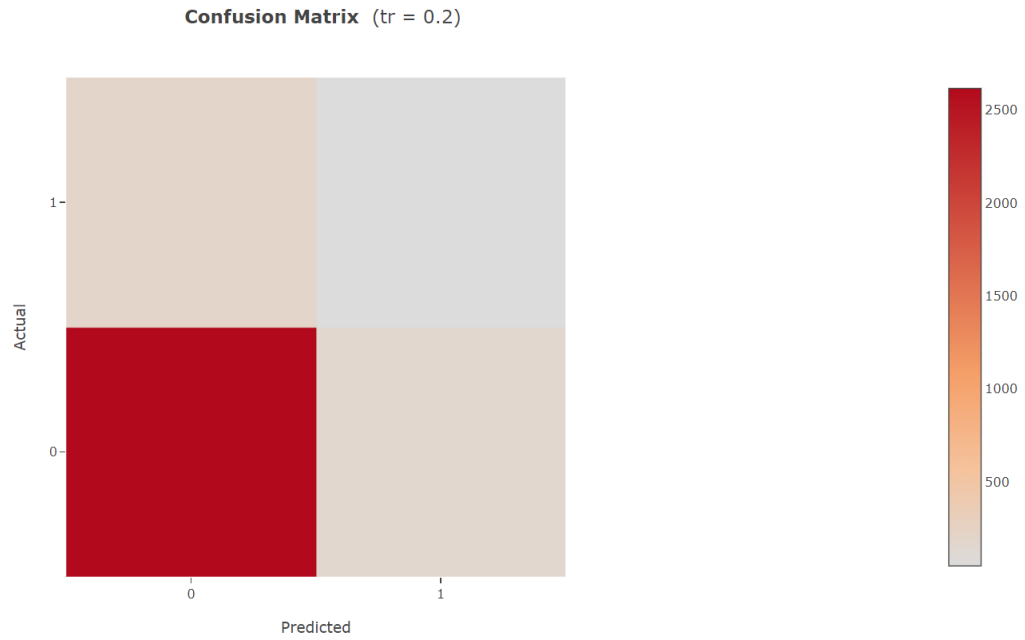


Пример кода метрики с выводом типа 'heatmap'

Наследование от `Metric`.

```
from typing import Any, Dict, Optional, Sequence

import pandas as pd
from klmg_predicate_types import column_role
from klmg_predicate_types.metric import Metric, param
from klmg_predicate_types.plotly import TraceController
from klmg_predicate_types.plotly.controller import Heatmap


class ConfusionMatrix(Metric):
    """
    Возвращает матрицу ошибок

    df: датасет для исследования
    score: название столбца с предсказаниями модели (вероятностными или бинарными)
    target: название столбца с целевой переменной
    tr: порог отсечения классов
    """

    label = "Confusion Matrix"

    def set_params(
        self,
        *,
        df: param.Dataframe,
        score: param.Column.with_role(column_role.PREDICTION),
        target: param.Column.with_role(column_role.TARGET),
        tr: param.FloatValue.between(left=0, right=1) = 0.5,
    ) -> None:
        self.score = score.value
        self.target = target.value
        self.df = df.value.astype({self.score: "float", self.target: "float"})
        self.tr = tr.value

    def traces(self) -> Sequence[TraceController]:
        y_test = self.df[self.target]
        preds = self.df[self.score].apply(lambda x: 1 if x > self.tr else 0)

        M = pd.crosstab(y_test, preds, dropna=True, rownames=["Actual"], colnames=["Predicted"])

        return (
            Heatmap(
                matrix=M.to_numpy().tolist(),
                labels_x=M.columns.tolist(),
                labels_y=M.index.tolist(),
            ),
        )

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": {"text": f"<b>Confusion Matrix</b>  (tr = {self.tr})", "x": 0.5},
            "xaxis": {
```

```
                "title": "Predicted",
                "side": "left",
                "scaleanchor": "y",
                "scaleratio": 1,
                "constrain": "domain",
                "type": "category",
            },
            "yaxis": {"title": "Actual", "side": "left", "type": "category"},
        }
```

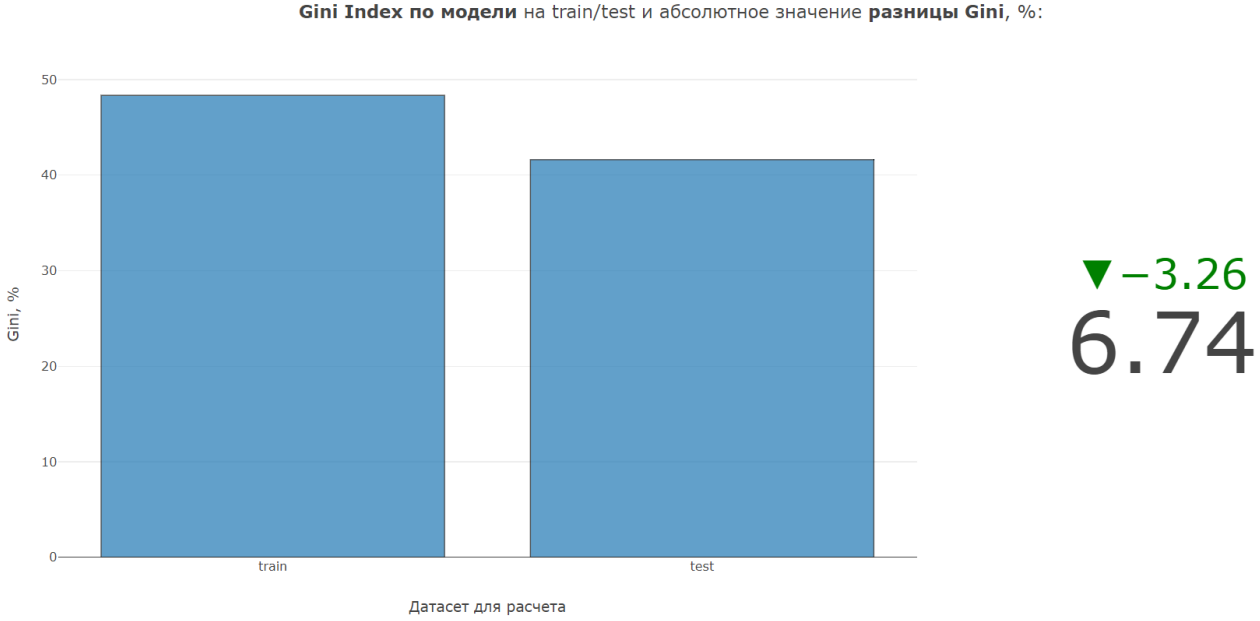


Пример кода метрики с выводом типа 'bar' + 'indicator'

Наследование от `Metric`.

```
from typing import Any, Dict, Optional, Sequence

from klmg_predicate_types import column_role
from klmg_predicate_types.metric import Metric, param
from klmg_predicate_types.plotly import TraceController
from klmg_predicate_types.plotly.controller import Bar, Indicator
from sklearn.metrics import roc_auc_score


class Gini_diff_model(Metric):
    """
    Bar -- Значения Gini Index по модели на train и test
    Indicator -- Абсолютное значение разницы Gini между train и test

    df_train: датасет для обучения
    df_test: датасет для теста
    score: название столбца с предсказаниями модели в виде вероятностей
    target: название столбца с целевой переменной
    """

    label = "Abs Gini Difference (%) train/test"

    def set_params(
        self,
        *,
        df_train: param.Dataframe,
        df_test: param.Dataframe,
        score: param.Column.with_role(column_role.PREDICTION),
        target: param.Column.with_role(column_role.TARGET),
        signal_bounds: param.Range = (5, 10),
    ) -> None:
        self.score = score.value
        self.target = target.value
        self.df_train = df_train.value.astype({self.target: "float", self.score: "float"})
        self.df_test = df_test.value.astype({self.target: "float", self.score: "float"})
        self.signal_bounds = signal_bounds.value

    def traces(self) -> Sequence[TraceController]:
        temp_train = self.df_train.loc[:, [self.target, self.score]].dropna()
        temp_test = self.df_test.loc[:, [self.target, self.score]].dropna()
```

```
        result_train = round(
            100 * (2 * roc_auc_score(temp_train[self.target], temp_train[self.score]) - 1), 2
        )
        result_test = round(
            100 * (2 * roc_auc_score(temp_test[self.target], temp_test[self.score]) - 1), 2
        )

        model_result = round(abs(result_test - result_train), 2)

        bar = Bar(
            values=[result_train, result_test],
            labels=["train", "test"],
        )
        indic = Indicator(
            value=model_result,
            mode="number+delta",
            reference=sorted(self.signal_bounds)[1],
            higher_is_better=False,
        )

        return (
            bar,
            indic,
        )

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": "<b>Gini Index по модели</b> на train/test"
            " и абсолютное значение <b>разницы Gini</b>, %:",
            "xaxis": {"title": "Датасет для расчета", "side": "left"},
            "yaxis": {"title": "Gini, %", "side": "left"},
        }
```



Пример кода метрики с выводом типа 'custom'

Наследование от `Metric`.

```
from typing import Any, Dict, Optional, Sequence

from klmg_predicate_types.metric import Metric, param
from klmg_predicate_types.plotly import TraceController
from klmg_predicate_types.plotly.controller import Custom
from scipy.stats import shapiro


class ShapiroWilk(Metric):
    """
    Тест Шапиро-Уилка (проверяет нормальность распределения данных)

    df: датасет для исследования
    field: название столбца для расчета
    signal_bounds: границы светофора
    """

    label = "Shapiro-Wilk Test"

    def set_params(
        self,
```

```
        *,
        df: param.Dataframe,
        field: param.Column,
        signal_bounds: param.Range.between(left=0, right=100) = (1, 5),
    ) -> None:
        self.df = df.value
        self.field = field.value
        self.signal_bounds = signal_bounds.value

    def traces(self) -> Sequence[TraceController]:
        shapiro_result = shapiro(self.df[self.field].dropna().astype("float"))
        p_value = float(shapiro_result[1] * 100)

        signal = (
            "red"
            if p_value <= sorted(self.signal_bounds)[0]
            else "green"
            if p_value > sorted(self.signal_bounds)[1]
            else "yellow"
        )

        histogr = Custom(
            type="histogram",
            x=self.df[self.field].dropna().astype("float").tolist(),
            nbinsx=30,
            marker={"opacity": 0.8},
            xaxis="x1",
            yaxis="y1",
        )
        indic = Custom(
            type="indicator",
            mode="number+delta",
            value=p_value,
            delta={
                "position": "top",
                "reference": self.signal_bounds[0],
                "increasing": {"color": signal},
                "decreasing": {"color": signal},
            },
            title={"text": "<b>p-value</b>, %:"},
            domain={"x": [0, 0.4], "y": [0, 1]},
        )

        return (
            histogr,
            indic,
        )

    def custom_layout(self) -> Optional[Dict[str, Any]]:
        return {
            "title": {
                "text": f"<b>Shapiro-Wilk Test</b><br><br>column: {self.field}"
                f"<br>H_0: Данные из нормального распределения",
                "x": 0.1,
            },
            "xaxis": {
                "title": f"{self.field} values",
                "side": "left",
                "constrain": "domain",
                "domain": [0.5, 1],
            },
            "yaxis": {
                "title": "number of values",
                "side": "left",
                "domain": [0, 0.9],
            },
            "grid": {"rows": 1, "columns": 2, "pattern": "independent"},
        }
```

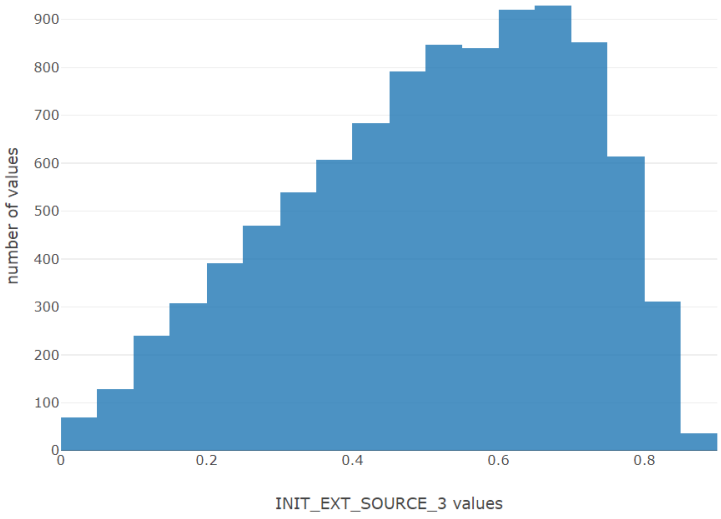
Shapiro-Wilk Test

column: INIT_EXT_SOURCE_3
H_0: Данные из нормального распределения

p-value, %:

▼ $-1e+38\times10^{-38}$

4.32×10^{-38}



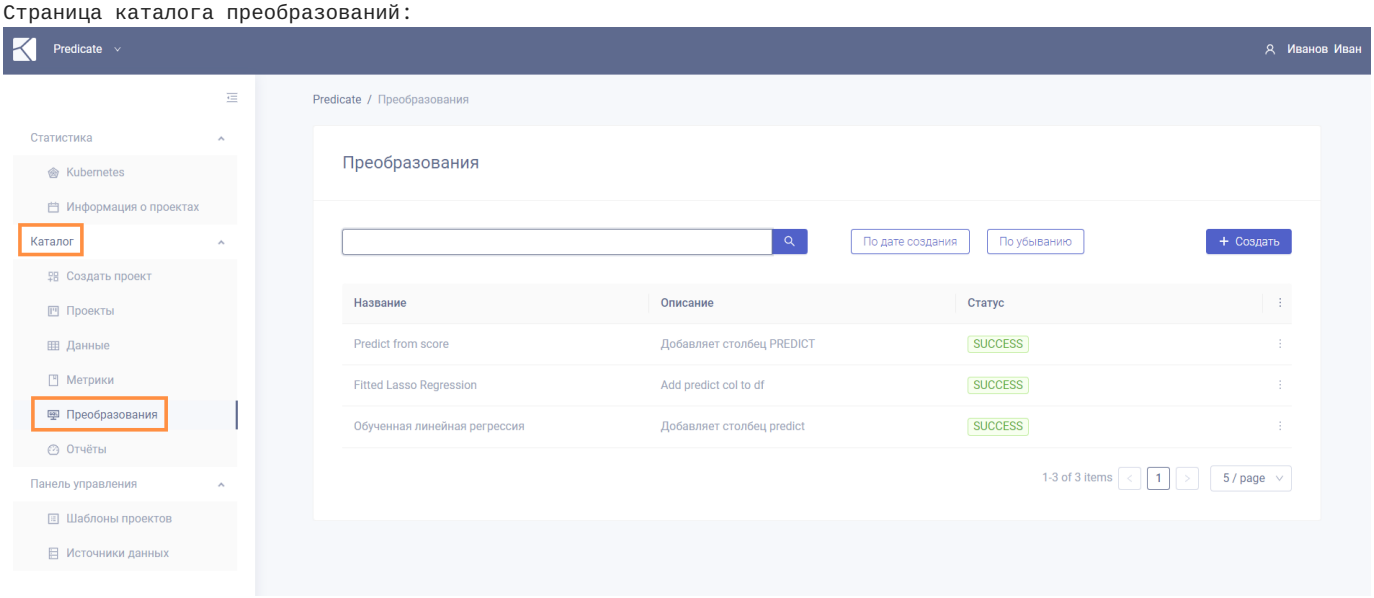
3.2.4 Преобразования

Predicate - Преобразования

ПЕРЕХОД К КАТАЛОГУ ПРЕОБРАЗОВАНИЙ

Переход к каталогу имеющихся в системе преобразований осуществляется из основного меню: *Каталог* > *Преобразования*.

Страница каталога преобразований:



ВОЗМОЖНЫЕ СТАТУСЫ ПРЕОБРАЗОВАНИЯ

CREATED - запись о преобразовании с соответствующим названием создана в базе. Реализуется, например, когда файлы преобразования помещены в репозиторий "напрямую", а не через интерфейс приложения.
SUCCESS - python и pickle файлы преобразования успешно загружены в Git-репозиторий.
FAILED - ошибка при загрузке файлов преобразования в Git.

УПРАВЛЕНИЕ КАТАЛОГОМ

В верхней части экранной формы доступны кнопки **сортировки каталога** по дате регистрации или названию преобразования, в убывающем или возрастающем порядке.
По умолчанию преобразования в каталоге упорядочены по дате загрузки в систему - от новых к более ранним.

Также доступен **поиск**.
Введите подстроку в поле поиска и нажмите `Enter` или символ лупы. Регистр не важен, поиск происходит по названию и описанию преобразований, названию классов преобразований в исходном коде, а также тэгам.

Каталог может содержать более одной страницы. Кнопки переключения между страницами, а также кнопка настройки количества объектов на странице, доступны в правом нижнем углу экранной формы.

ПРОСМОТР И СОЗДАНИЕ ОБЪЕКТОВ

Для **просмотра информации о преобразовании** наведите курсор на символ меню (три точки) справа в строке соответствующего преобразования и нажмите "Просмотр".

Для **регистрации нового преобразования** нажмите кнопку "Создать" в правой верхней части окна каталога преобразований. Подробнее - см. раздел "регистрация нового преобразования".

Регистрация нового преобразования

Predicate поддерживает работу с преобразованиями, написаными на языках:

- Python
- R-script

Для регистрации преобразования необходимо сначала подготовить файлы, содержащие логику преобразования, а затем загрузить их в Predicate через web-интерфейс.

ПОДГОТОВКА ФАЙЛОВ ПРЕОБРАЗОВАНИЯ

Для регистрации **Python-преобразования** необходимо подготовить заранее **два файла**:

- файл с кодом на Python, содержащий *определение класса преобразования* (.py-файл);
- pickle-файл с *экземпляром класса преобразования*, определенного в .py-файле.

Для регистрации **R-преобразования** необходимо подготовить заранее **один файл**:

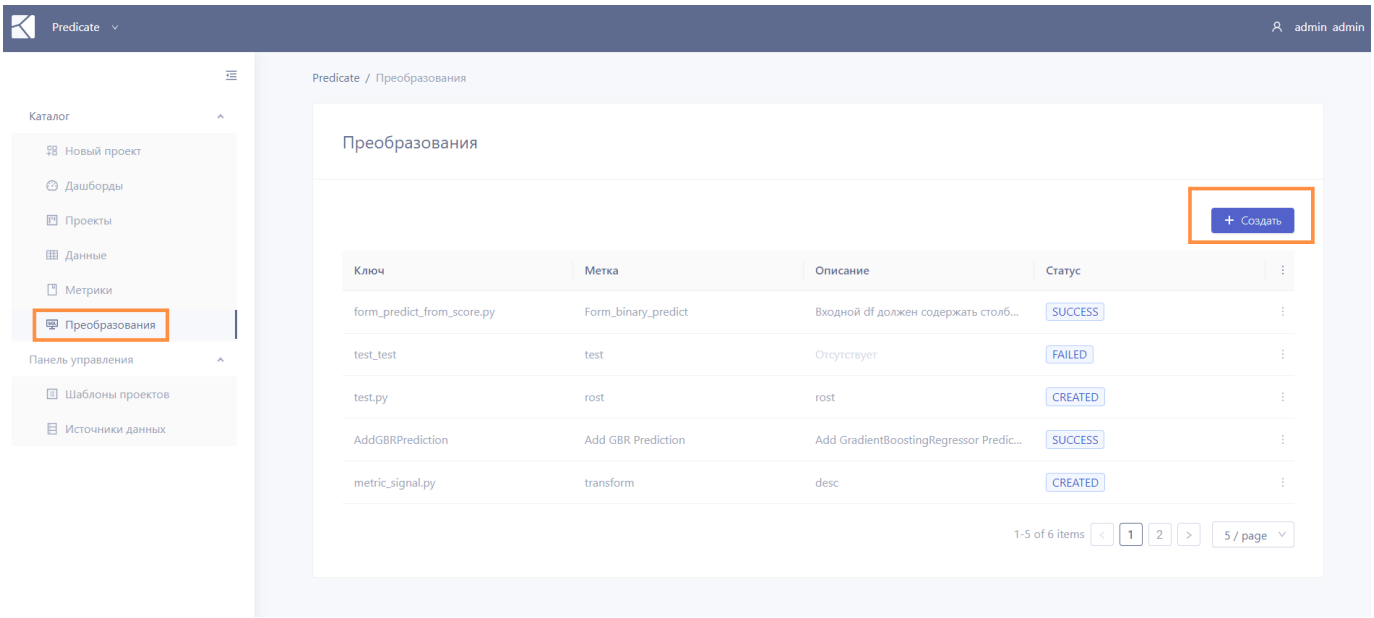
- pickle-файл с *экземпляром объекта преобразования*.

Инструкции по структуре файлов и примеры кода для Python-преобразований и для R-преобразований приведены на странице "Файлы преобразования" в соответствующих разделах.

После того как файлы подготовлены, выполните следующие действия.

ПЕРЕХОД К ФОРМЕ РЕГИСТРАЦИИ ПРЕОБРАЗОВАНИЯ

Перейдите в раздел “Преобразования” основного меню приложения и на открывшейся странице каталога преобразований нажмите кнопку "Создать".



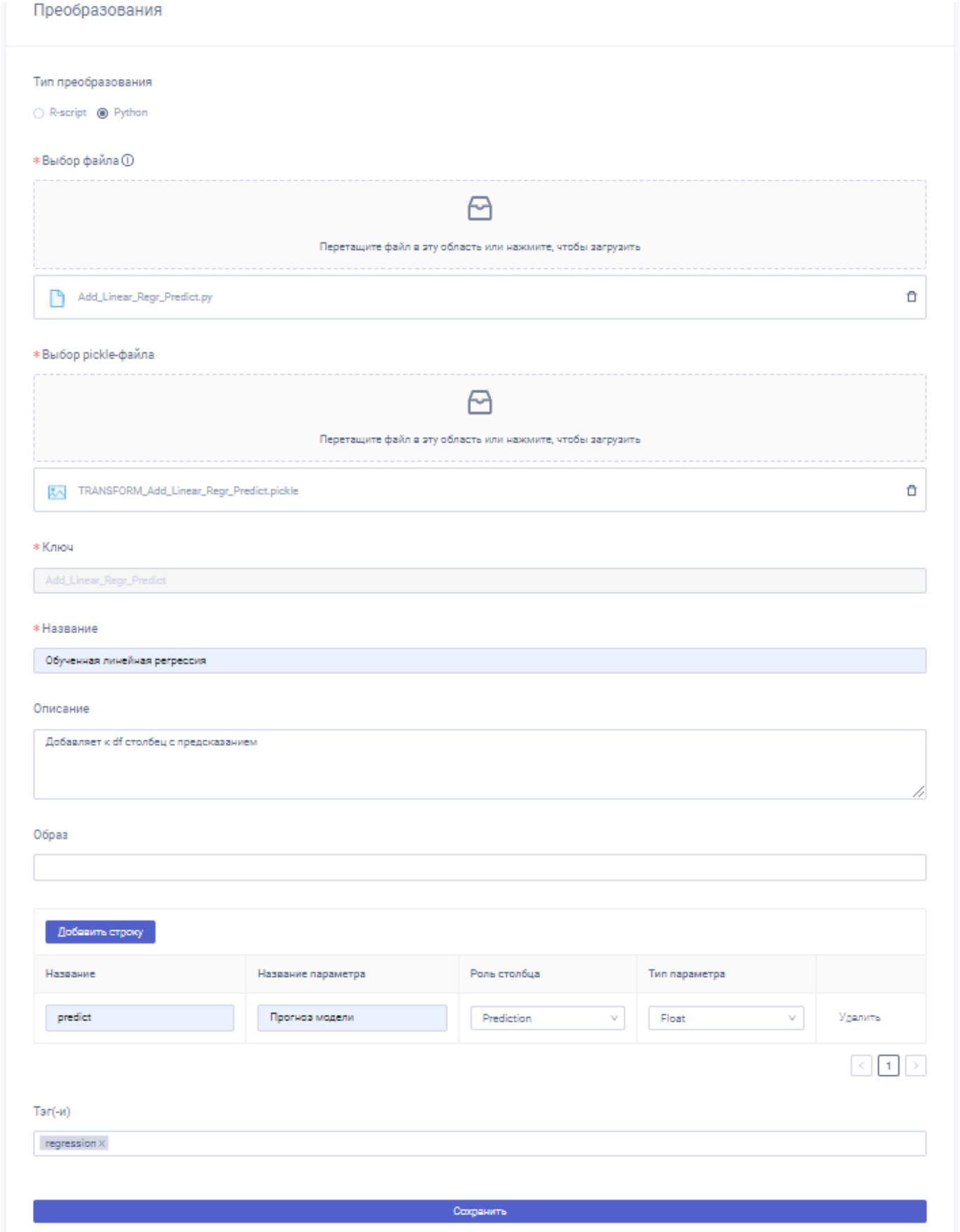
ЗАГРУЗКА ФАЙЛОВ ПРЕОБРАЗОВАНИЯ

В открывшейся форме регистрации нового преобразования сначала выберите тип преобразования (Python / R-script).

Далее загрузите необходимые файлы:

- для Python-преобразования сначала загрузите **.py-файл**, после чего станет доступно поле для загрузки pickle-файла, в которое нужно загрузить соответствующий файл;
- для R-преобразования загрузите pickle-файл в соответствующее поле.

Форма регистрации преобразования:



Заполните поля:

- Ключ (обязательное поле) – название класса (id) преобразования, при корректном скрипте заполняется автоматически;
- Название (обязательное поле) – название преобразования в системе;
- Описание (необязательное поле) – описание преобразования;
- Образ (необязательное поле) – рекомендуется оставить настройки по умолчанию;

РАЗМЕТКА СТОЛБЦОВ, ДОБАВЛЯЕМЫХ ПРЕОБРАЗОВАНИЕМ

При корректной обработке .py-скрипта открываются дополнительные поля для регистрации столбцов, добавляемых преобразованием к датасету.

Для каждого из добавляемых столбцов предлагается задать разметку:

- название (должно повторять название соответствующего столбца в .py-файле);
- алиас;
- роль;
- тип.

Описание всех типов и ролей приведено на странице "разметка датасета".

Добавьте к преобразованию **тэги** выбрав одно или несколько значений из выпадающего списка.

Завершите процесс загрузки преобразования нажав кнопку "Сохранить".

ПРОСМОТР СОЗДАННОГО ПРЕОБРАЗОВАНИЯ

После того как преобразование создано, оно появится в списке преобразований, зарегистрированных в системе (*Каталог > Преобразования*), и будет доступно для применения к зарегистрированным в системе датасетам.

Файлы преобразования

PYTHON-ПРЕОБРАЗОВАНИЕ

Python- и pickle- файлы

- Python-файл содержит *определение класса* преобразования. Например, модели.
- Pickle-файл содержит *экземпляр класса* преобразования, определенного в python-файле. Например, модели, обученной на неком наборе данных.

При корректном скрипте название класса преобразования автоматически отобразится в поле "Ключ" формы регистрации преобразования.

Шаблон python-файла преобразования

```
import необходимые_импорты

# определение класса преобразования
class Название_класса_преобразования:

    # инициализация входных параметров (опционально)
    def __init__(self, параметры):
         self.параметр1 = ...
         self.параметр2 = ...
         ...

    # метод, содержащий экземпляр класса преобразования,
    # который Predicate применяет к входному df (обязательно)
    def transform(self, df: pd.DataFrame):
        transformed = df.copy()

        /логика преобразования/

        transformed['новый_столбец_1'] = ...
        transformed['новый_столбец_2'] = ...

        return transformed
```

Создание pickle-файла преобразования

В pickle-файл необходимо сохранить экземпляр определенного в python-файле класса преобразования. В экземпляр в качестве параметра можно передать, например, предварительно обученную модель.

```
# создаем экземпляр класса преобразования
obj = Название_класса_преобразования(...)    # на месте ... может быть обученная модель


# сохраняем его в pickle-файл
with open('TRANSFORM_Название_класса_преобразования.pickle', 'wb') as f:
    pickle.dump(obj, f)
```

Примеры кода

Пример кода преобразования Add_Linear_Regr_Predict, которое добавляет к полученному на вход датафрейму - набору признаков - столбец с предсказаниями заранее обученной линейной регрессионной модели для данных значений признаков.

.py-файл преобразования (*Add_Linear_Regr_Predict.py*):

```
import pandas as pd

class Add_Linear_Regr_Predict:
    def __init__(self, unpickled_fitted_model):
        self.model = unpickled_fitted_model

    def transform(self, df: pd.DataFrame):
        transformed = df.copy()
        transformed["predict"] = self.model.predict(transformed)

        return transformed
```

Следующий ниже код необходимо исполнить для создания **pickle-файла**:

```
import pickle
import pandas as pd
from sklearn import linear_model
```

```
# определение класса преобразования (то же, что в .py-файле)
class Add_Linear_Regr_Predict:
    def __init__(self, unpickled_fitted_model):
        self.model = unpickled_fitted_model

    def transform(self, df: pd.DataFrame):
        transformed = df.copy()
        transformed["predict"] = self.model.predict(transformed)

        return transformed
# конец дублирования кода из .py-файла


# СОЗДАНИЕ И ОБУЧЕНИЕ МОДЕЛИ

file_name = 'train_data.csv'                # файл с обучающими данными
df = pd.read_csv(file_name, decimal='.', delimiter=';')
X = df.drop(columns=['target']).values
y = df['target'].values

model = linear_model.LinearRegression()      # создаем модель
fitted_model = model.fit(X, y)               # обучаем ее на данных из файла


# создаем ЭКЗЕМПЛЯР КЛАССА ПРЕОБРАЗОВАНИЯ, в который передаем обученную модель
obj = Add_Linear_Regr_Predict(fitted_model)

# сохраняем его в pickle-файл
with open('TRANSFORM_Add_Linear_Regr_Predict.pickle', 'wb') as f:
    pickle.dump(obj, f)
```

В результате исполнения данного кода сформируется pickle-файл *TRANSFORM_Add_Linear_Regr_Predict.pickle*, который необходимо загрузить в приложение вместе с python-файлом *Add_Linear_Regr_Predict.py*.

R-ПРЕОБРАЗОВАНИЕ

Pickle-файл

Содержит сериализованный *экземпляр объекта* преобразования.
Например, модели, обученной на неком наборе данных.

Пример кода

Приведенный ниже код создает pickle-файл, содержащий объект преобразования, добавляющего к полученному на вход датасету столбец weight.

В Predicate при регистрации R-преобразования надо будет загрузить через интерфейс полученный pickle-файл *TRANSFORM_R_AddWeightColumn.pickle*.

```
# Импортируем библиотеку и
# создаем таблицу model_obj, содержащую столбец weight со значениями от 1 до 6

library(data.table)
model_obj <- data.table(
  weight = 1:6
)

# Объявляем объект преобразования, который принимает на вход model
# (model может быть произвольным R-объектом (таблицей, функцией (моделью), др.),
#  в данном примере model - это таблица)

AddWeightColumn <- function(model){             # создаем объект преобразования с названием AddWeightColumn
    # attributes
    self.model <- model                         # создаем переменную self.model и присваиваем ей значение model
    structure(class = "Transform", list(
        # methods
        transform = function(dataframe){        # в обязательный метод transform передаем df и инструкции по его преобразованию
          dataframe$weight <- self.model        # к исходному df добавляем столбец с названием weight, в который записываем значение из self.model
          return(dataframe)
        }
    ))
}


# Создаем экземпляр объекта преобразования, в который передаем
# определенный ранее объект model_obj (таблицу со столбцом weight)

my_object_new <- AddWeightColumn(model_obj)

# Создаем pickle-файл и записываем в него
# сериализованный экземпляр объекта преобразования

outCon <- file("TRANSFORM_R_AddWeightColumn.pickle", "w")
```

```
mychars <- rawToChar(serialize(my_object_new, NULL, ascii=T))
cat(mychars, file=outCon); close(outCon)
```

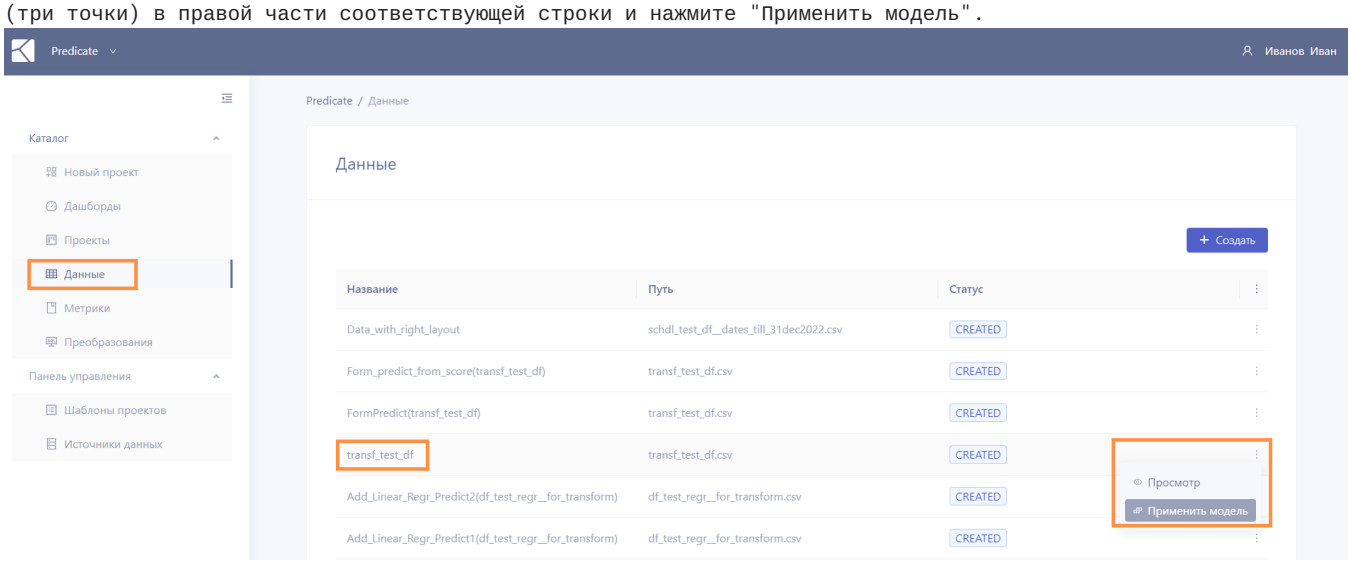
Применение преобразования к данным

Преобразование проводится над данными - получает на вход датасет и возвращает датасет. В выходном датасете число столбцов больше или равно числа столбцов во входном датасете.

В приложении доступно исполнение зарегистрированных в системе преобразований (Каталог > Преобразования) над зарегистрированными в системе датасетами (Каталог > Данные).

Для того, чтобы **применить преобразование**:

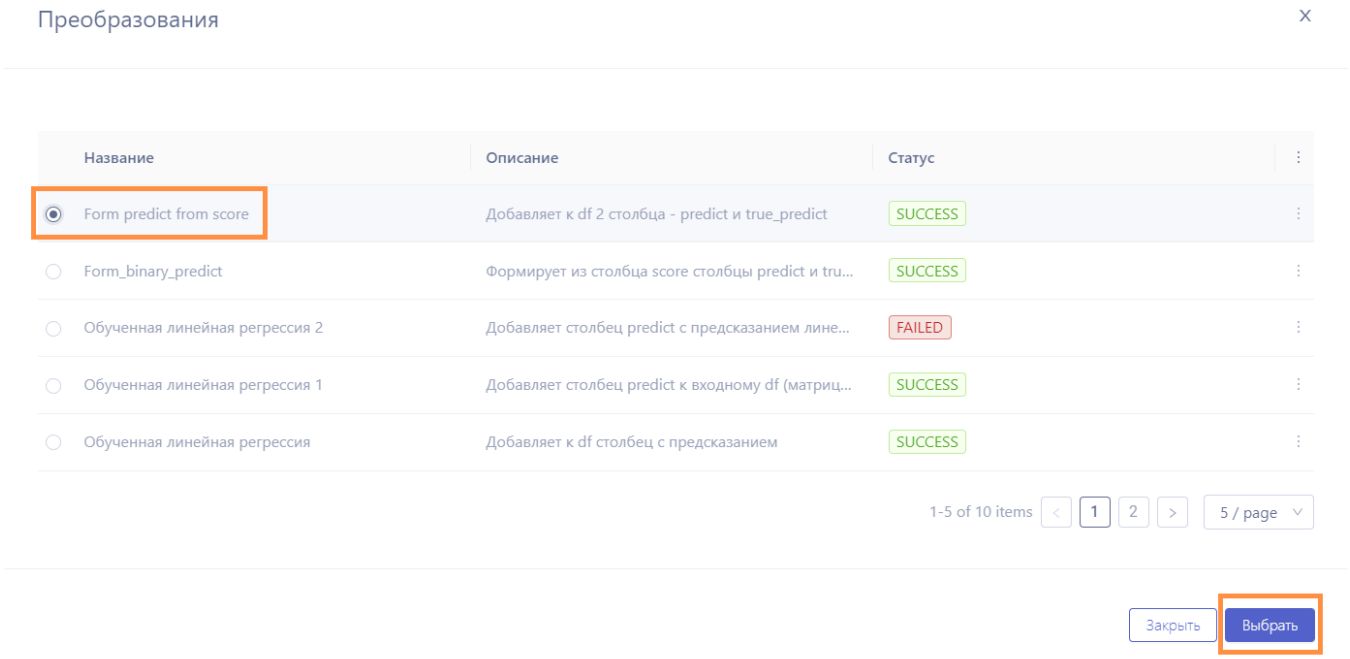
- Перейдите в каталог датасетов (Каталог > Данные), выберите нужный **датасет**, наведите курсор на символ меню (три точки) в правой части соответствующей строки и нажмите "Применить модель".



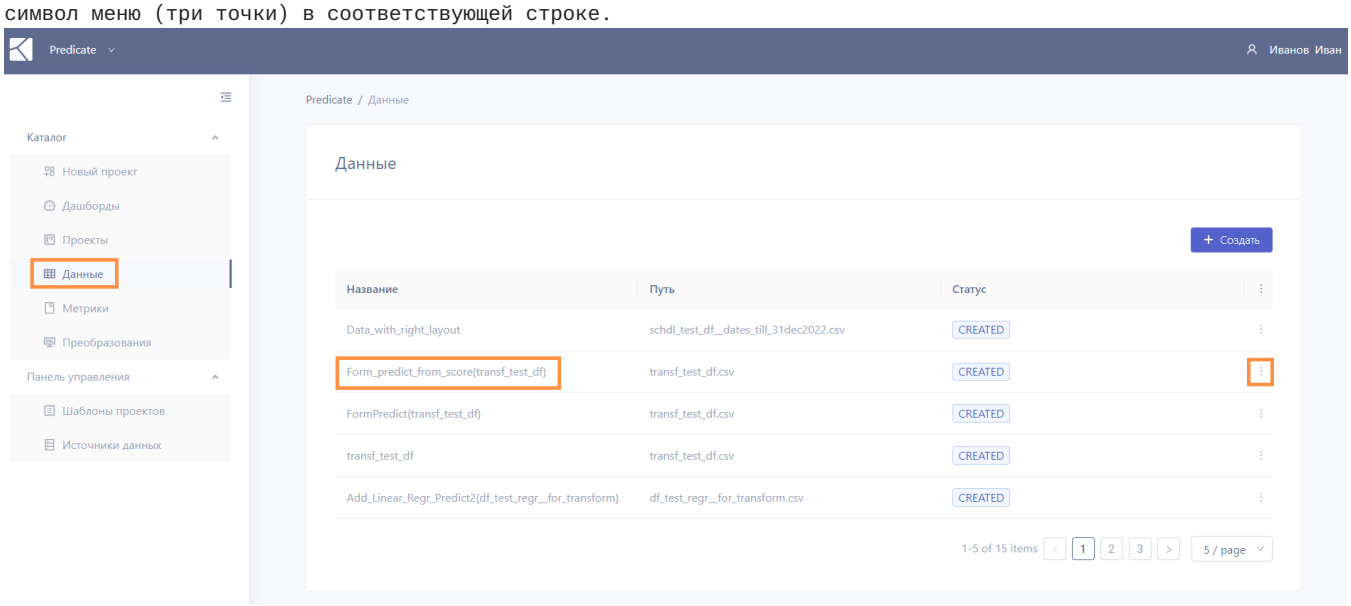
Примечание

Опция "применить модель" доступна для датасетов, ссылающихся на источник данных (s3, psql или др.), и недоступна для датасетов, сформированных путем преобразования датасетов первого типа.

- В открывшемся окне каталога преобразований отметьте нужное преобразование и нажмите "Выбрать".



- В каталог данных (Каталог > Данные) автоматически добавится строка, соответствующая новому, преобразованному, датасету. Название преобразованного датасета формируется по схеме: *название_преобразования(название_исходного_датасета)*. Информацию о новом датасете можно посмотреть нажав на символ меню (три точки) в соответствующей строке.



- Исходный датасет останется в каталоге в неизменном виде.

В результате **преобразованный датасет** становится доступным для использования в проектах мониторинга наряду с другими зарегистрированными датасетами.

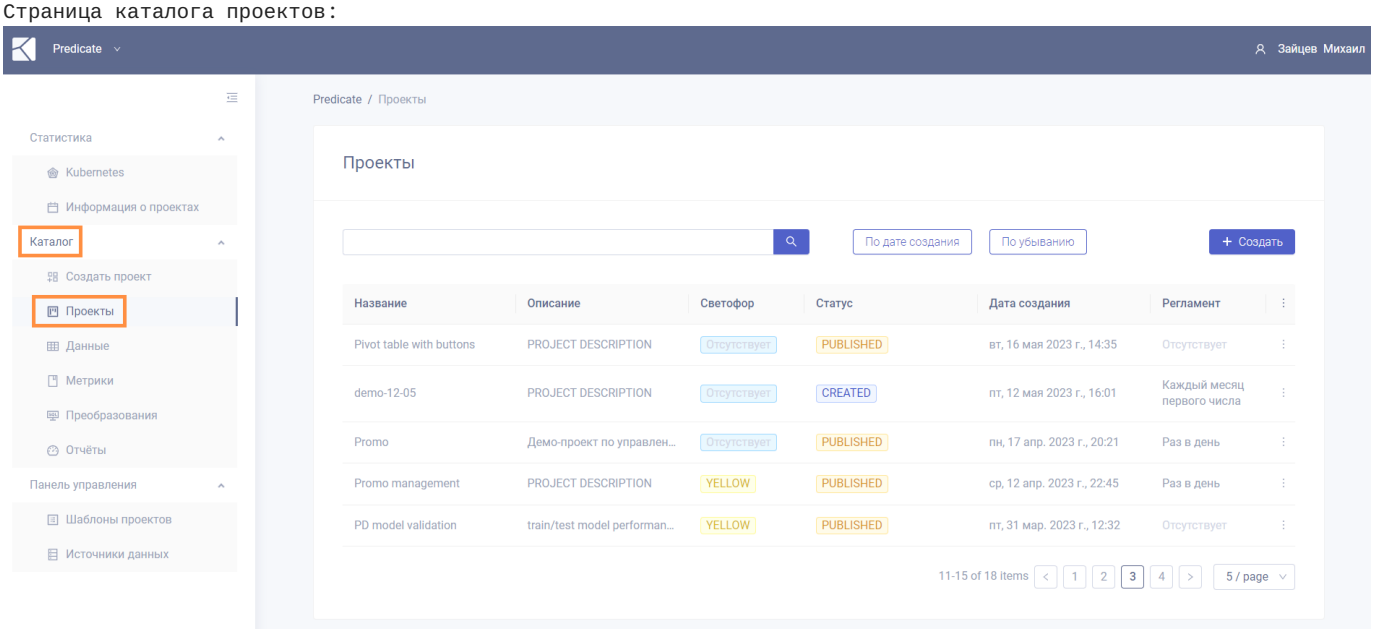
3.2.5 Проекты

Predicate - Проекты

ПЕРЕХОД К КАТАЛОГУ ПРОЕКТОВ

Переход к каталогу имеющихся в системе проектов мониторинга моделей/данных осуществляется из основного меню: *Каталог > Проекты*.

Страница каталога проектов:



ИНФОРМАЦИЯ, ОТОБРАЖАЕМАЯ В КАТАЛОГЕ

- **Название** - название проекта. Указывается пользователем при создании проекта. Должно быть уникальным.
- **Описание** - описание проекта. Указывается пользователем при создании проекта.
- **Светофор** - результат расчета правила над скалярными метриками проекта.
- **Статус** - показывает успешность прохождения стадий формирования проекта в системе.
- **Дата создания** - дата создания проекта (может не совпадать с датой первого запуска проекта).
- **Регламент** - частота регулярных автоматических запусков проекта.

ВОЗМОЖНЫЕ ЗНАЧЕНИЯ СВЕТОФОРА ПРОЕКТА

GREEN - согласно правилу, проект завершился с хорошим результатом
YELLOW - согласно правилу, проект завершился с удовлетворительным результатом
RED - согласно правилу, проект завершился с неудовлетворительным результатом
Отсутствует - значения светофора нет (либо в проекте не использовался светофор, либо правило еще не было рассчитано)
ERROR - ошибка при расчете светофора (например, не была посчитана одна из метрик, используемых в правиле)

Подробнее о светофорах - см. раздел "светофор проекта".

ВОЗМОЖНЫЕ СТАТУСЫ ПРОЕКТА

CREATED - проект создан (данные заполнены корректно, соответствующая запись сохранена в базе).
PUBLISHED - DAG Airflow сформирован и загружен в Git.
FAILED - ошибка при формировании или загрузке DAG Airflow в Git.
SUCCESS - последний запуск проекта прошел успешно.

УПРАВЛЕНИЕ КАТАЛОГОМ

В верхней части экранной формы доступны кнопки **сортировки каталога** по дате или названию проекта, в убывающем или возрастающем порядке.
По умолчанию проекты в каталоге упорядочены по дате создания - от новых к более ранним.

Также доступен **поиск**.
Введите подстроку в поле поиска и нажмите `Enter` или символ лупы. Регистр не важен, поиск происходит по названию и описанию проектов, а также по тэгам.

Каталог может содержать более одной страницы. Кнопки переключения между страницами, а также кнопка настройки количества объектов на странице, доступны в правом нижнем углу экранной формы.

ПРОСМОТР И СОЗДАНИЕ ОБЪЕКТОВ

Для **просмотра общей информации о проекте** и **результатов расчета метрик проекта** наведите курсор на символ меню (три точки) справа в строке соответствующего проекта и нажмите "Просмотр". Подробнее о результатах проекта - см. "просмотр результатов мониторинга".

Для **создания нового проекта** нажмите кнопку "Создать" в правой верхней части окна каталога проектов. Подробнее - см. раздел "создание нового проекта".

Если необходимо создать проект из сохраненного ранее шаблона - действуйте согласно инструкции из раздела "создание проекта из шаблона".

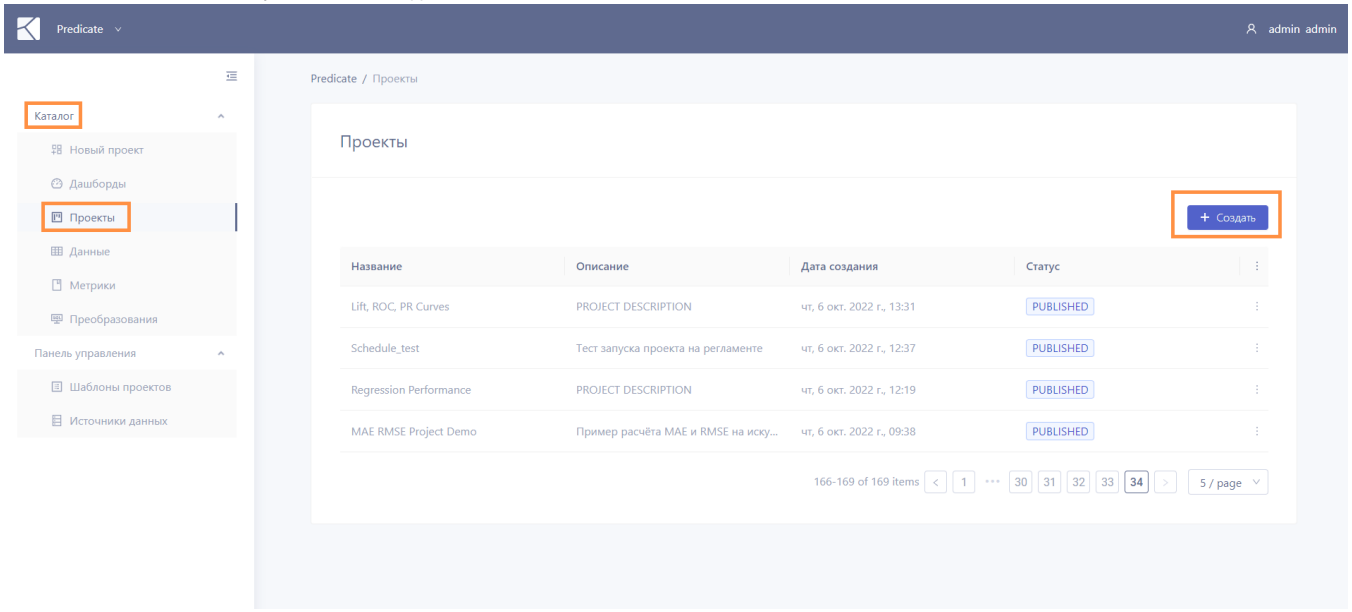
Создание нового проекта

Порядок создания проекта для единоразового или многократного мониторинга работы модели.

ПЕРЕХОД К ФОРМЕ СОЗДАНИЯ ПРОЕКТА

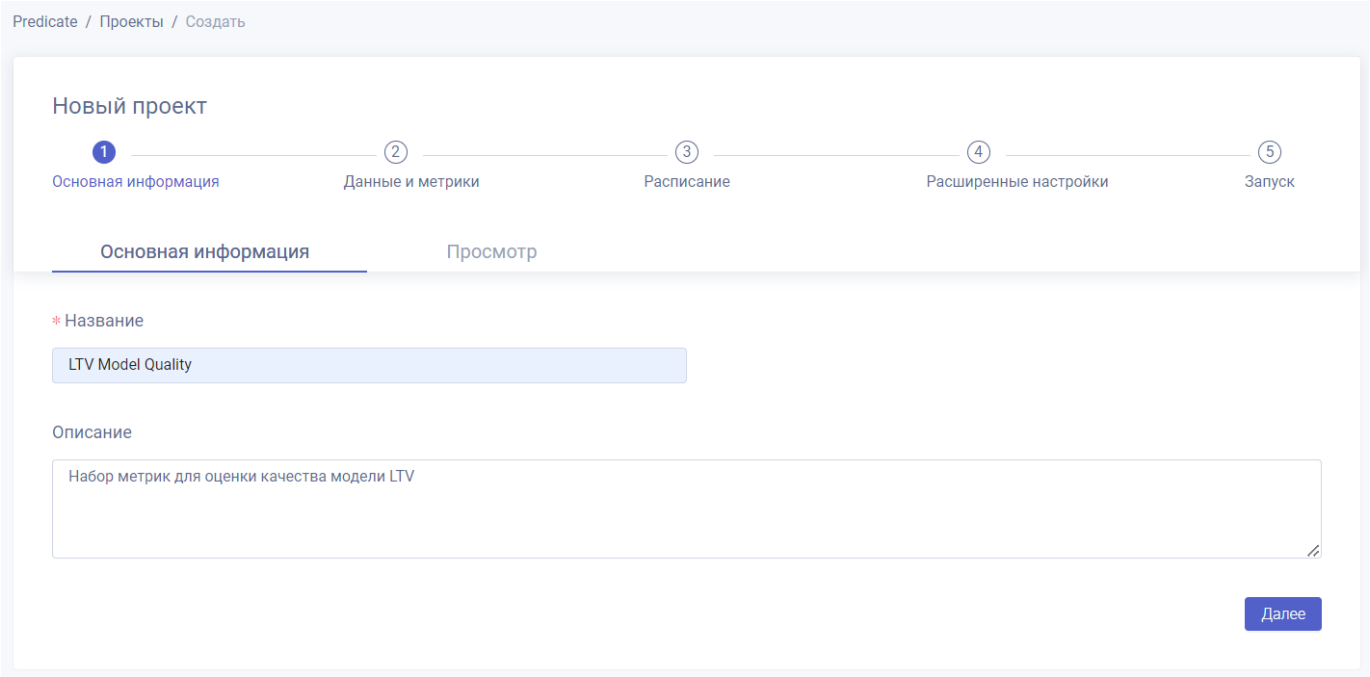
Переход к форме создания нового проекта осуществляется из основного меню приложения одним из следующих способов.

- Способ 1: *Каталог > Новый проект*
- Способ 2: *Каталог > Проекты > Создать*



- Алгоритм создания проекта из шаблона описан на отдельной странице.

В результате откроется форма создания нового проекта:



Создание проекта состоит из последовательного набора этапов. Названия этапов отображаются в верхней части страницы. Для пройденных и текущего этапов название и номер выделены цветом.

Промежуточный просмотр заполненных параметров проекта доступен на вкладке "Просмотр".

Для перехода на следующий этап необходимо заполнить обязательные поля текущего этапа и нажать кнопку "Далее". Для возврата на предыдущий этап доступна кнопка "Назад".

ЭТАП "ОСНОВНАЯ ИНФОРМАЦИЯ"

В форме создания проекта на этапе **"Основная информация"** введите:

- название проекта (обязательное поле);
- его описание (опционально).

Для перехода к следующему этапу нажмите кнопку “Далее”.

ЭТАП "ДАННЫЕ И МЕТРИКИ"

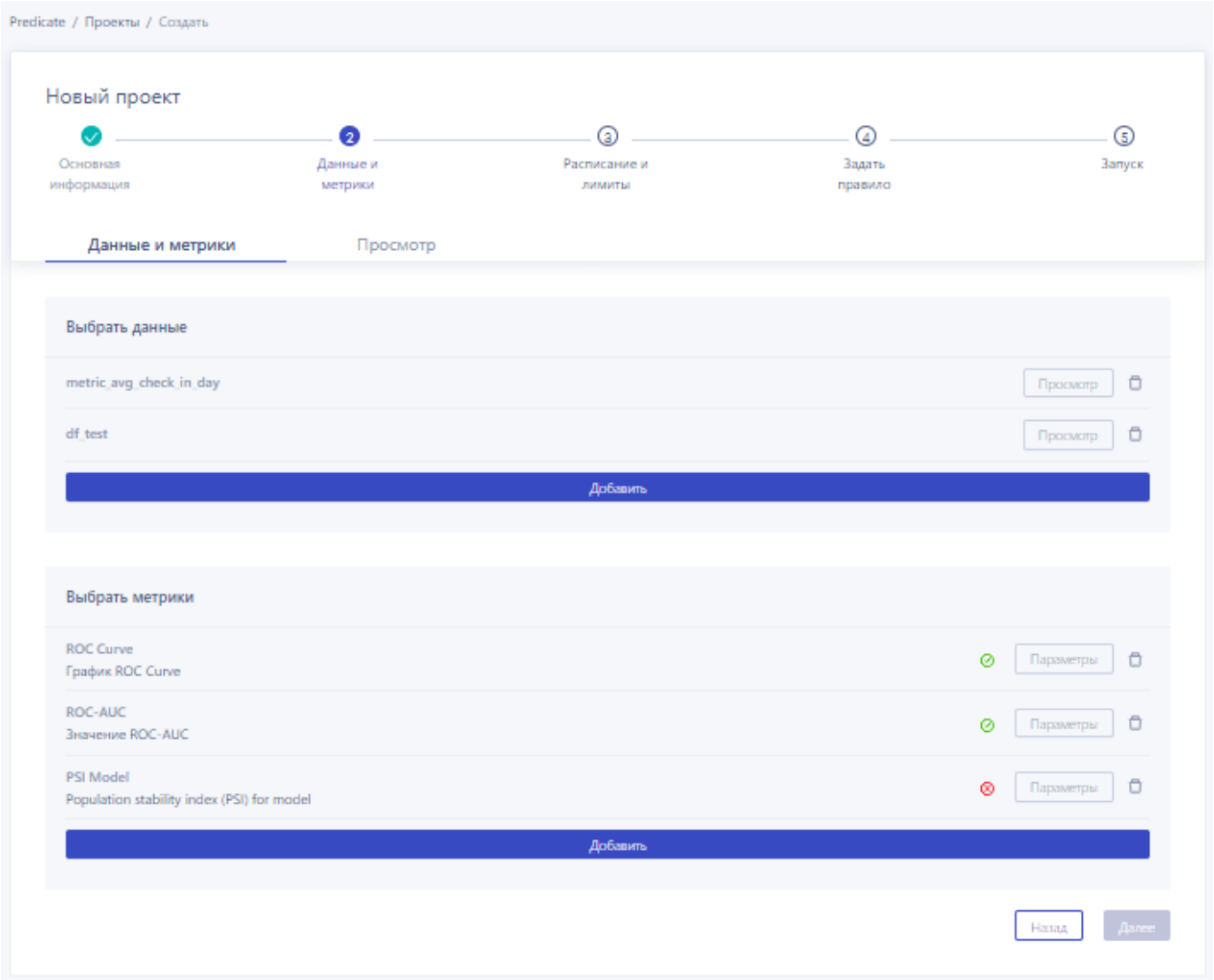
На этапе **"Данные и метрики"**:

Выбор данных

- Выберите **данные** из каталога нажав кнопку "Добавить". В открывшемся окне поставьте галочку рядом с названием необходимого датасета и нажмите кнопку "Выбрать". Допускается выбор нескольких датасетов в рамках проекта. Обратите внимание, что каталог доступных датасетов может состоять из нескольких страниц.
- Если необходимо зарегистрировать новые датасеты, это можно сделать согласно инструкции из раздела "регистрация датасета".

Выбор метрик

Выберите **метрики** из каталога нажав кнопку "Добавить". В открывшемся окне отметьте необходимые метрики нажатием кнопки "+" в столбце "Добавить метрику". Допускается выбор нескольких метрик, а также выбор метрики более одного раза в рамках проекта. Завершите процесс формирования набора метрик проекта нажатием кнопки "Выбрать". Обратите внимание, что каталог метрик может состоять из нескольких страниц.



Задание параметров метрик

Далее необходимо задать **параметры** каждой из выбранных **метрик**. Для этого нажмите кнопку "Параметры" справа от названия метрики. В открывшемся окне выберите из выпадающих списков необходимые параметры. В первую очередь производится выбор датасета, далее - столбцов для расчета и других параметров. Нажмите кнопку "Сохранить". Если в метрику переданы все необходимые параметры, напротив ее названия появится зеленая галочка. Когда все параметры всех метрик выбраны, нажмите кнопку "Далее".

ЭТАП "РАСПИСАНИЕ"

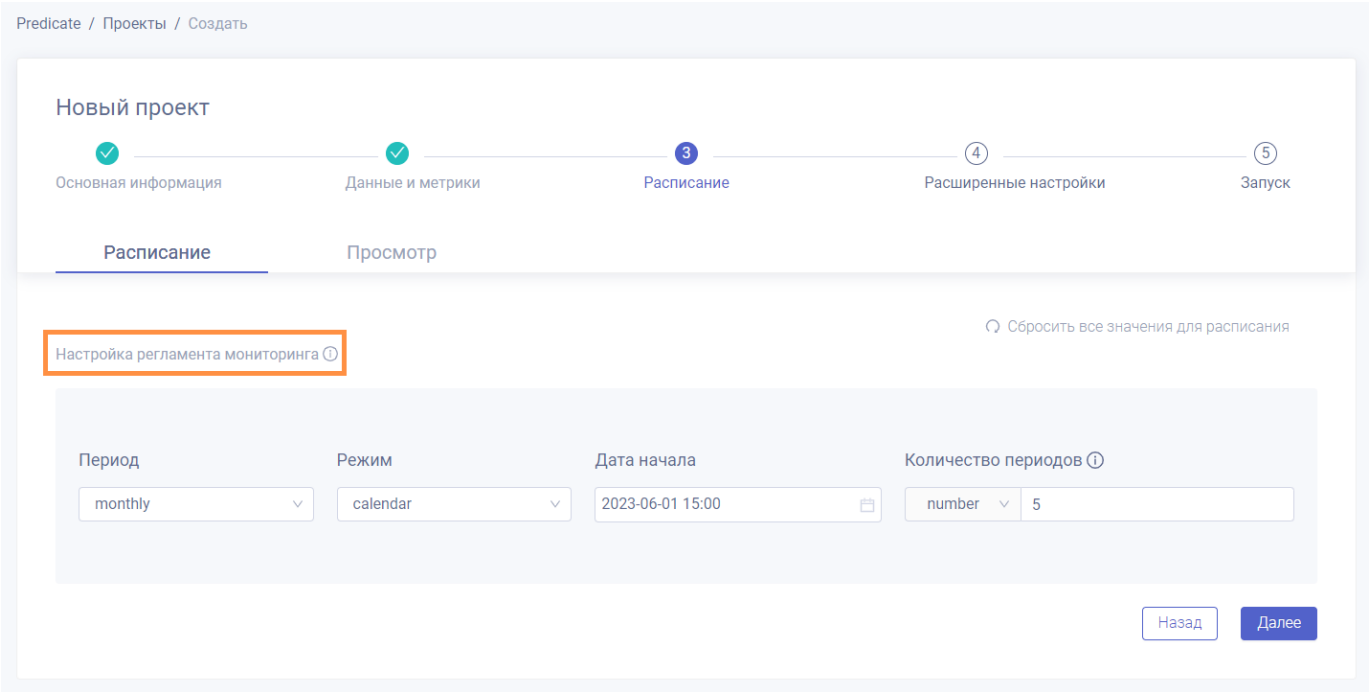
На этапе **“Расписание”** настраивается регламент регулярных запусков проекта.

Единоразовый запуск

> **Важно**
>
> Если планируется **единоразовый запуск** проекта, пропустите этот этап оставив все поля пустыми и нажав кнопку "Далее".
> Расчет начнется сразу после завершения создания проекта.

Регулярные запуски

Для настройки **расписания запусков** заполните форму:



Период (обязательно) - периодичность запуска проекта:

- daily - ежедневно;
- weekly - еженедельно;
- monthly - ежемесячно.

Режим (обязательно) - режим забора данных из источника:

- calendar - период отсчитывается от даты запуска;
- most recent - ближайший законченный полный период (неделя с пн по вс / месяц с 1 по 31 число / ...).

Дата начала (обязательно) - дата начала регулярного мониторинга. От этой даты "назад" отсчитываются периоды для забора данных.

Количество периодов (обязательно) - количество периодов для забора данных (ширина скользящего окна):

- число - данные за это число периодов будут учтены в расчетах;
- all - выбрать все данные в таблице.

Пример

Пусть датой начала проекта выбрано 20 октября 2022 (четверг), период - еженедельно, количество периодов - 2. Тогда в случае выбора режима *calendar* расчеты будут проводиться на данных за 6-20 октября 2022 (2 недели с чт по чт), а при выборе режима *most recent* - на данных за 3-17 октября 2022 (2 недели с пн по пн).

Важно

Настройка **скользящего окна** для забора данных при регламентных запусках проекта доступна только для датасетов со столбцом, содержащим **дату записи** и отмеченным **флагом** `True` в разметке датасета.

Для датасетов без столбца с флагом `True` (в том числе для датасетов вообще без столбца с датой) регламентный запуск проекта также возможен. В этом случае при каждом запуске для расчета берутся **все записи**, имеющиеся в объекте данных на текущий момент.

ЭТАП "РАСШИРЕННЫЕ НАСТРОЙКИ"

Дополнительные параметры проекта для более точной его настройки:

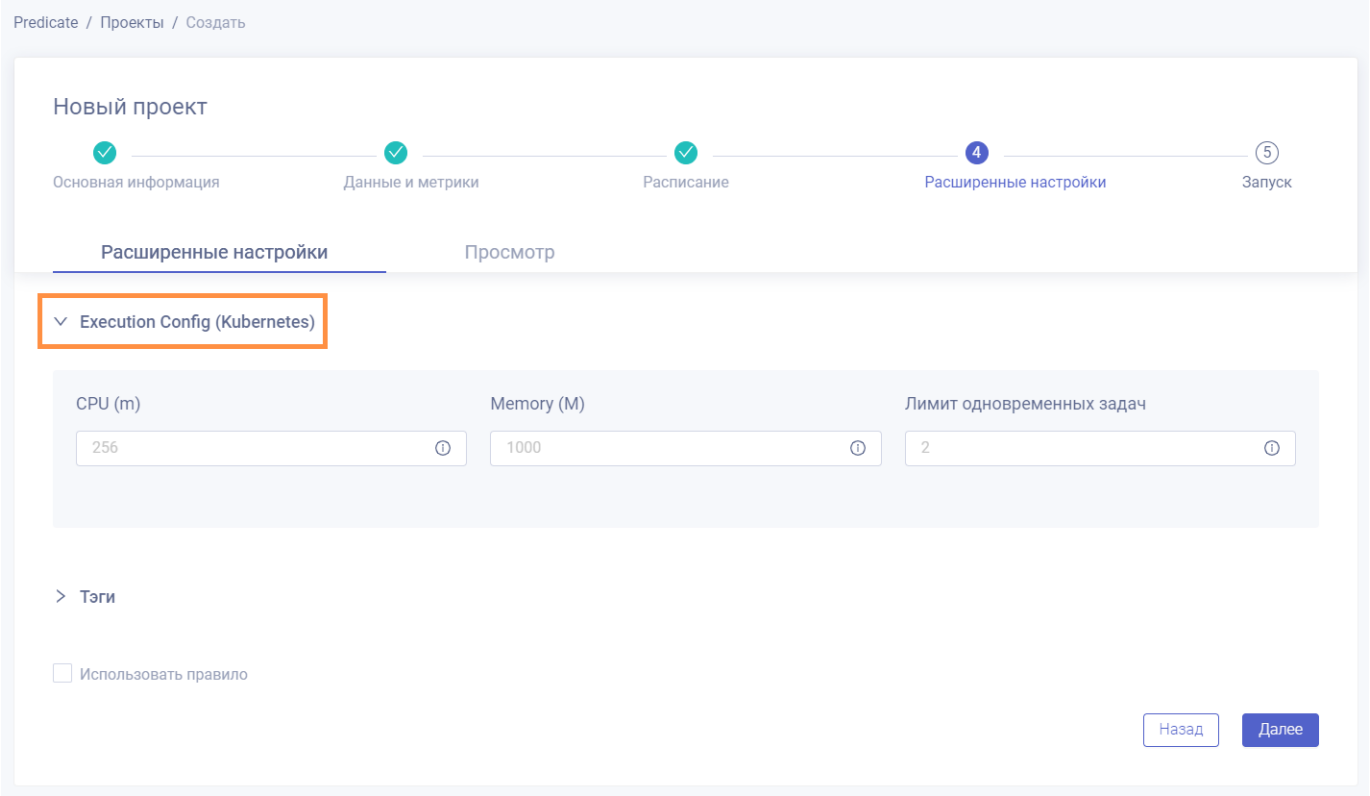
- ограничение ресурсов, выделяемых под проект;
- тэги;
- правило вычисления светофора проекта;
- настройка уведомлений.

Важно

Чтобы запустить проект с параметрами по умолчанию пропустите этот этап нажав кнопку "Далее".

Execution Config (Kubernetes)

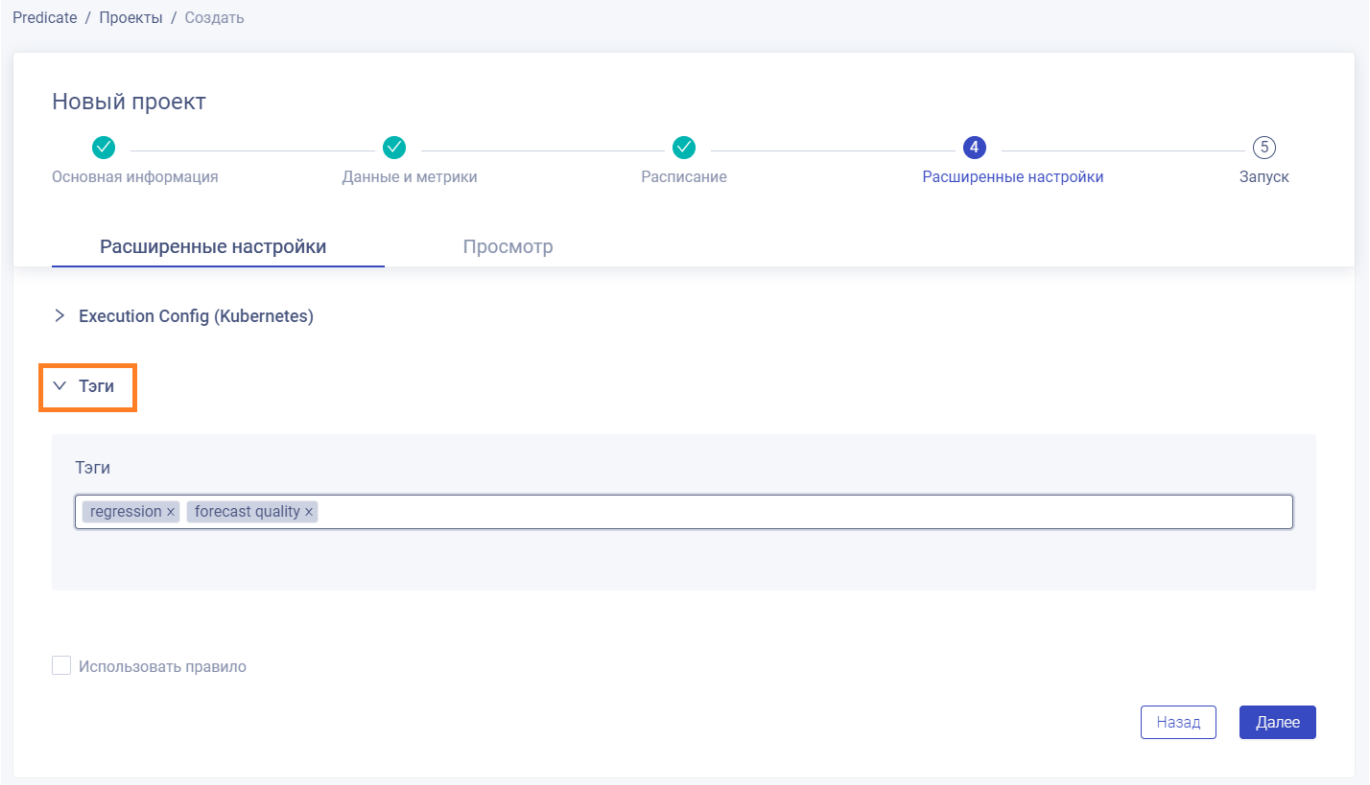
Для изменения стандартного **ограничения ресурсов**, выделяемых под проект, разверните и заполните форму "Execution Config":



- *CPU* (опционально) - лимиты CPU под одну параллельную задачу исполнения проекта, измеряется в милли-ядрах. Рекомендуется указывать значение, равное `<число строк в таблице>/600` (примерная формула, полученная опытным путем);
- *Memory* (опционально) - лимиты памяти под проект, измеряются в мегабайтах. Рекомендуется указывать значение, равное `<число строк в таблице>/180` (примерная формула, полученная опытным путем).

Тэги

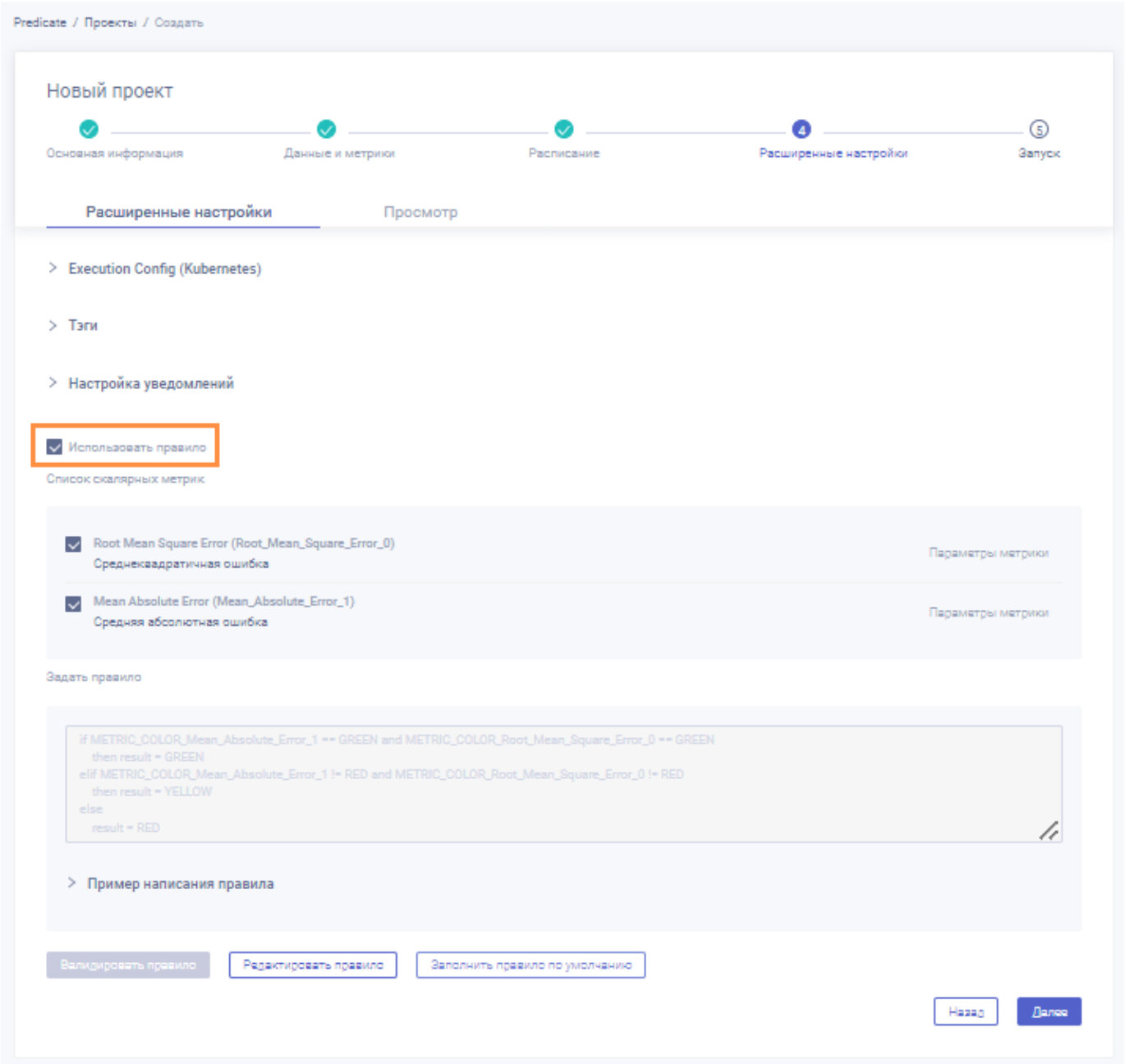
Для добавления **тэгов** к проекту разверните соответствующую форму и выберите тэги (один или несколько) из выпадающего списка. В каталоге проектов будет доступен поиск по тэгам.



Использовать правило

Чек-бокс “Использовать правило” следует выбрать когда необходимо настроить правило вычисления светофора проекта.

При активации "Использовать правило" появится форма настройки правила:



Сначала выберите **скалярные метрики**, которые будут использоваться в правиле, из предложенного списка.

Далее составьте **правило** одним из следующих способов:

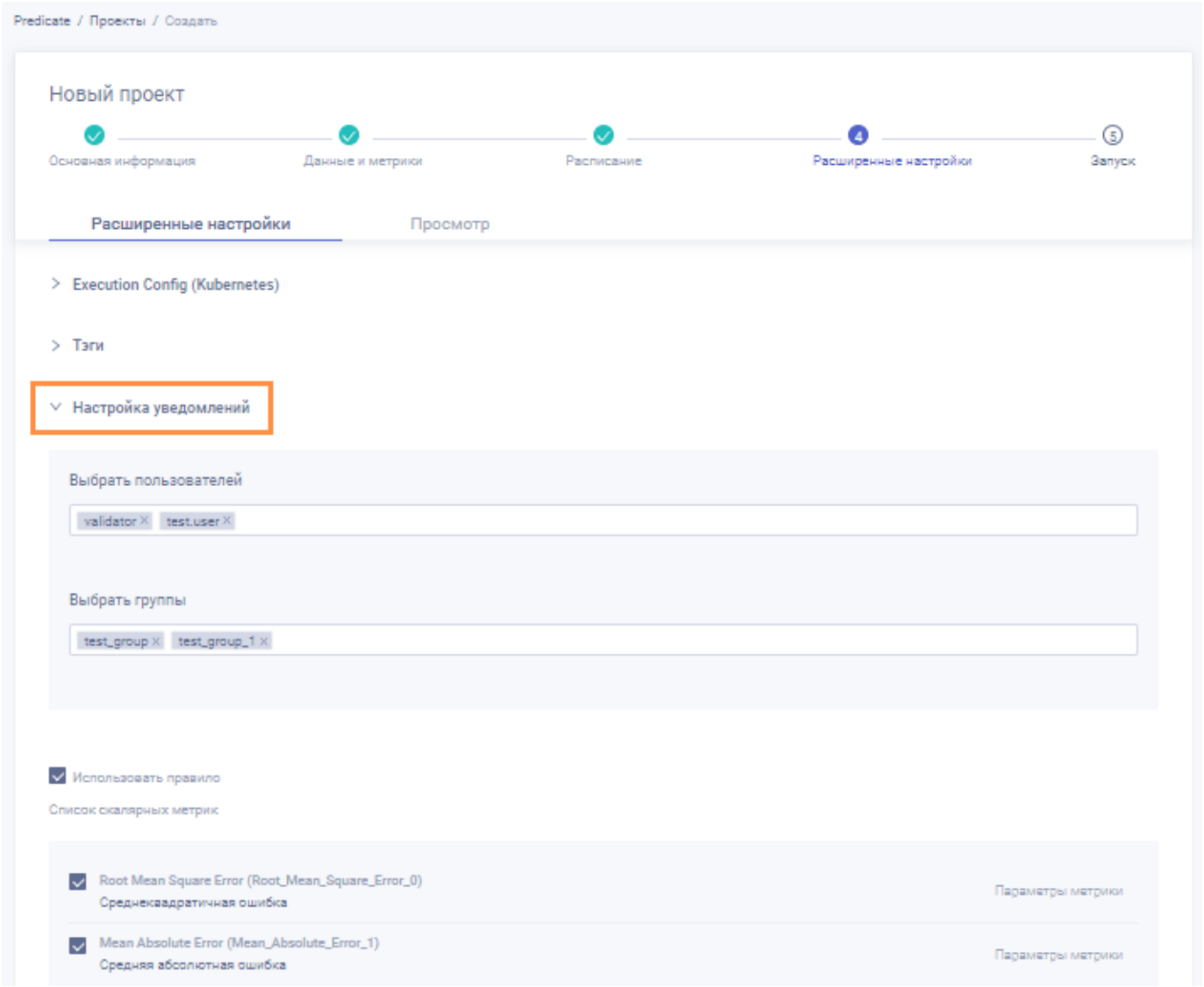
- заполните поле правила по умолчанию нажав соответствующую кнопку
- самостоятельно напишите правило согласно инструкции и нажмите кнопку “Валидировать правило”. В случае ошибки валидации проверьте корректность составленного правила, внесите исправления и повторно нажмите "Валидировать правило".

Пример правила можно посмотреть развернув соответствующую панель под полем ввода правила.

Настройка уведомлений

Настройка **уведомления пользователей** по результатам расчета светофора проекта.

Опция доступна только в случае выбора чек-бокса “Использовать правило”. Форма настройки уведомлений:



Для отправки уведомлений можно выбрать как **конкретных пользователей** по ФИО, так и **группу пользователей** (например, валидаторов), всем членам которой будет направляться уведомление в случае неудачного завершения проекта (красный сигнал светофора или ошибка при расчете правила).

После того как все дополнительные параметры проекта настроены завершите этап "Расширенные настройки" нажав кнопку "Далее".

ЭТАП "ЗАПУСК"

На этапе **“Запуск”** следует проверить данные создаваемого проекта.

Если все данные соответствуют необходимым, завершите процесс создания проекта нажатием кнопки “Создать”.

ПРОСМОТР СОЗДАННОГО ПРОЕКТА

После того как проект создан, в каталоге проектов (*Каталог > Проекты*) появляется соответствующая этому проекту строка и отображается статус успешности создания проекта.

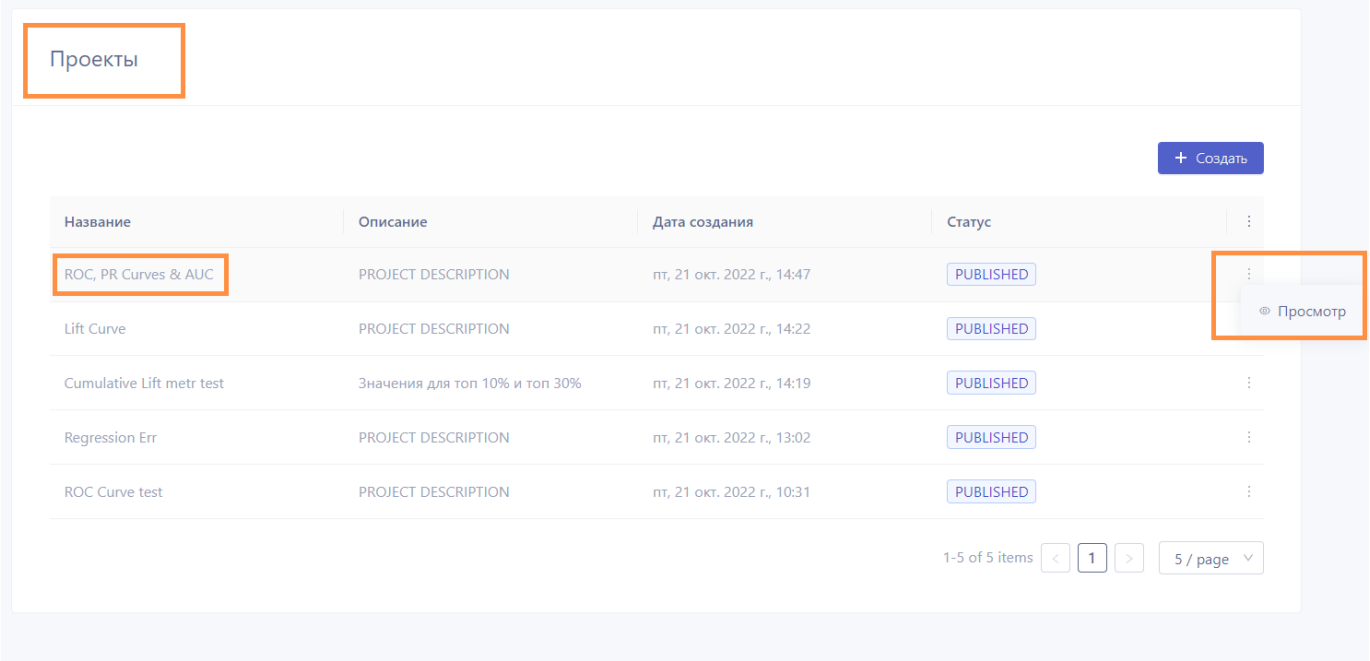
Инструкция по просмотру отчета с результатами проекта представлена в разделе "результаты мониторинга".

Просмотр результатов мониторинга

Просмотр дашборда с результатами исполнения проекта.

ПЕРЕХОД НА СТРАНИЦУ ПРОЕКТА

На странице каталога проектов (Каталог > Проекты) выберите нужный проект, наведите курсор на символ меню (три точки) в правой части соответствующей строки и нажмите “Просмотр”.



Откроется страница проекта.

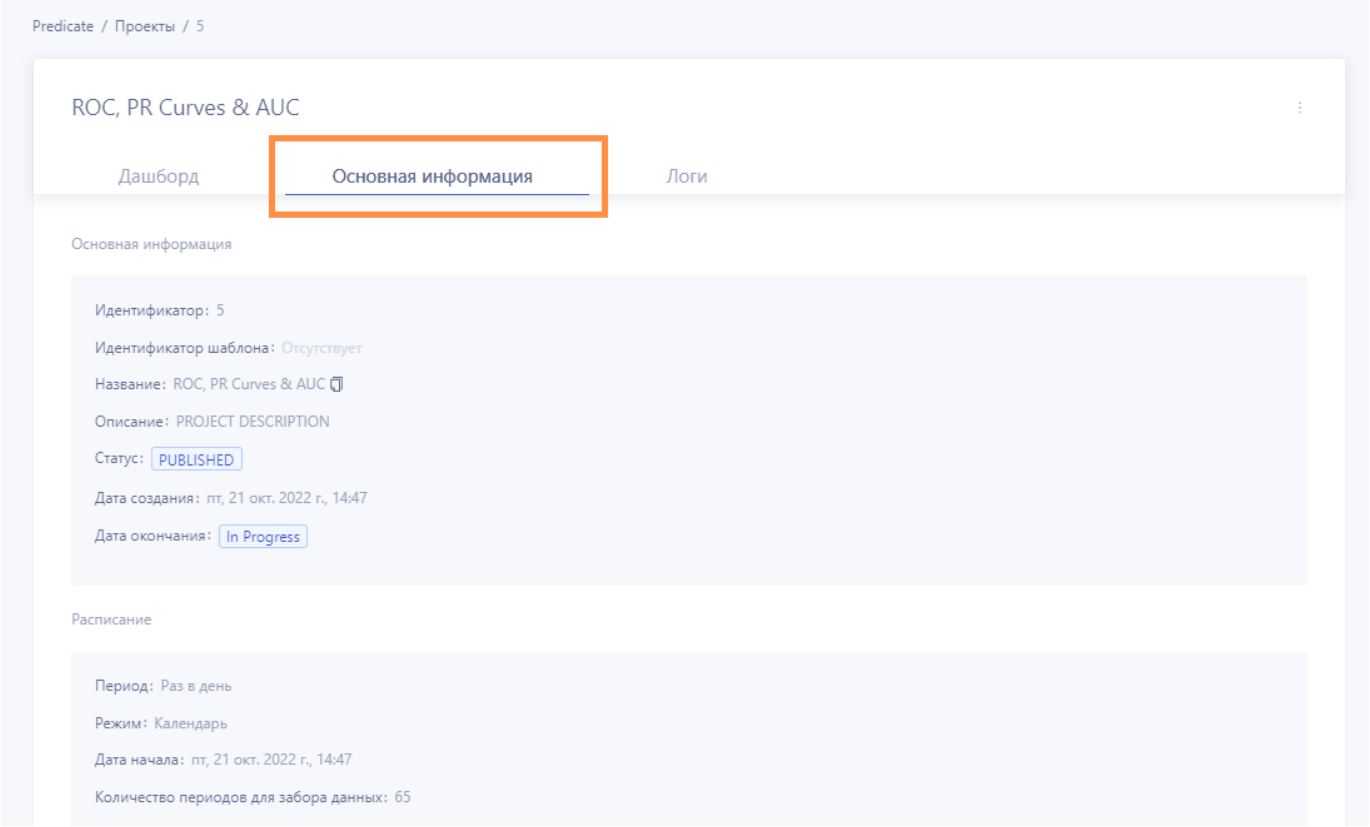
СТРАНИЦА ПРОЕКТА

Содержит вкладки:

- вкладка “Основная информация” - информация о параметрах проекта;
- вкладка “Дашборд” - визуальный отчет о мониторинге.

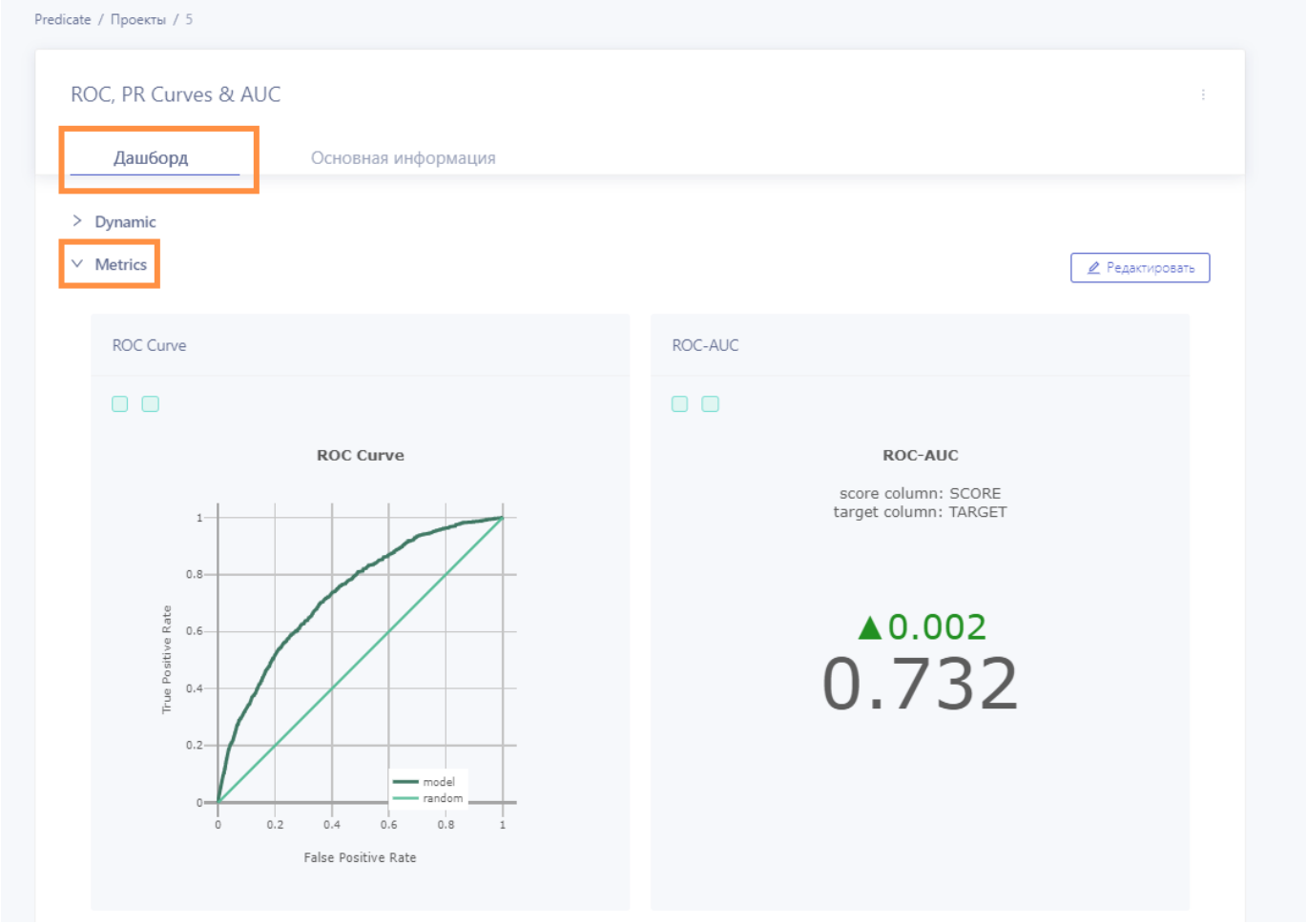
Информация о проекте

Вкладка "Основная информация":



Дашборд

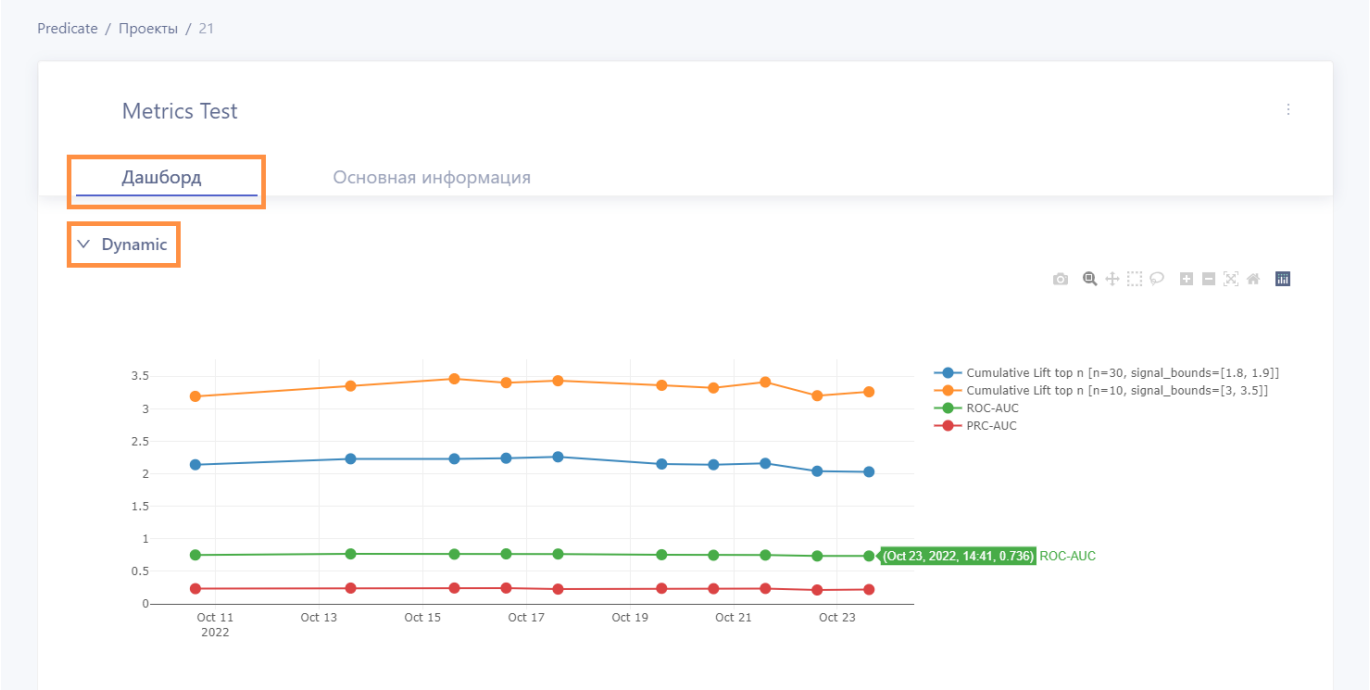
Вкладка "Дашборд", раздел "Metrics":



Если нажать кнопку "Редактировать" в правом верхнем углу экранной формы, становится доступным изменение размеров и перемещение карточек метрик. После завершения редактирования дашборда нажмите "Сохранить".

Обновляемые дашборды позволяют отслеживать результаты исполнения проектов согласно настроенному расписанию. В разделе "Dynamic" вкладки "Дашборд" отображается динамика изменения скалярных метрик от запуска к запуску проекта.

Вкладка "Дашборд", раздел "Dynamic":



Светофор проекта

Светофор проекта рассчитывается на основе значений и цветовых сигналов **скалярных метрик** проекта.

Значения, которые может принимать светофор проекта: `GREEN`, `YELLOW`, `RED`. У проекта может отсутствовать светофор.

Значение (цвет) светофора проекта определяется в соответствии с **правилом**, которое составляет пользователь. Правило задает логику преобразования совокупности значений и/или цветовых сигналов скалярных метрик проекта в итоговый сигнал (красный/желтый/зеленый).

Пример правила для расчета светофора проекта

Если все скалярные метрики проекта "зеленые", то зеленый; иначе, если все "не красные", то желтый; иначе - красный.

Набор скалярных метрик проекта, участвующих в расчете сфетофора, и правило расчета задаются при создании проекта мониторинга на этапе "Расширенные настройки".

Значение светофора проекта отображается в каталоге проектов и соответствует результату последнего запуска проекта.

Значение светофора может **изменяться от запуска к запуску проекта**, отражая таким образом изменения отслеживаемых метрик.

КАК НАПИСАТЬ ПРАВИЛО

Алиасы

На этапе "Расширенные настройки" при создании проекта нужно выбрать чек-бокс "Использовать правило" и отметить галочками метрики, которые предполагается использовать в правиле.

В скобках после названия метрики указан **алиас**, автоматически присвоенный метрике в данном проекте.

В правиле к метрике следует обращаться по алиасу.

Значения метрик

Можно обращаться как к **числовому значению** метрики, так и к значению простого светофора метрики (**цвету сигнала**).

Для обращения к значению нужно использовать префикс `METRIC_VALUE_` перед алиасом, для обращения к цвету сигнала нужно использовать префикс `METRIC_COLOR_` перед алиасом.

Пример

Есть метрика с алиасом MAE.
Для обращения к значению нужно написать `METRIC_VALUE_MAE`.
Для обращения к цвету простого светофора нужно написать `METRIC_COLOR_MAE`.

Синтаксис правила

Структура выражения

If условие **then** действие [**elif** условие **then** действие] **else** действие

Условием может быть обычное логическое выражение со скобками, and/or, сравнениями >,<,==,!= и ссылками на значения метрик (`METRIC_COLOR` / `METRIC_VALUE`).

Действием может быть результат либо еще одно if-выражение, каждое if-выражение(включая изначальное) может содержать сколько угодно (можно 0) elif и должно закрываться else.

Результат, он же светофор проекта, может быть задан как `result = GREEN`, `result = YELLOW`, `result = RED`.

Примеры правил

Простой пример

Алиасы доступных метрик: MAE, MEAN

```
if METRIC_COLOR_MAE == GREEN and METRIC_VALUE_MEAN > 0
    then result = GREEN
else result = RED
```

Сложный пример

Алиасы доступных метрик: MAE, MSE, MEAN

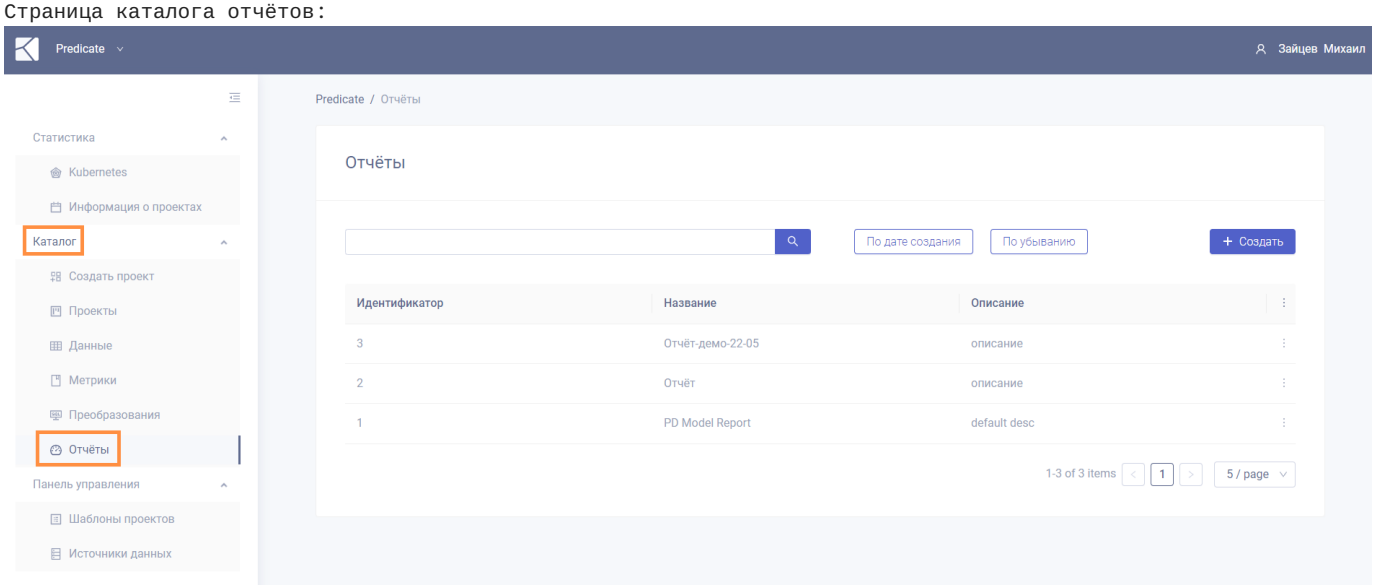
```
if METRIC_COLOR_MAE == GREEN or (METRIC_COLOR_MSE == GREEN and METRIC_COLOR_MEAN == GREEN)
    then
        if METRIC_VALUE_MSE < 10
            then result = GREEN
        elif METIC_COLOR_MSE == YELLOW
            then result = YELLOW
        else result = RED
else result = RED
```

3.2.6 Отчёты

Predicate - Отчёты

Переход к каталогу сохраненных в системе отчётов (пользовательских дашбордов) осуществляется из основного меню: *Каталог > Отчёты*.

Страница каталога отчётов:



УПРАВЛЕНИЕ КАТАЛОГОМ

В верхней части экранной формы доступны кнопки **сортировки каталога** по дате создания или названию отчёта, в убывающем или возрастающем порядке.
По умолчанию отчёты в каталоге упорядочены по дате регистрации в системе - от новых к более ранним.

Также доступен **поиск**.
Введите подстроку в поле поиска и нажмите `Enter` или символ лупы. Регистр не важен, поиск происходит по названию и описанию отчётов.

Каталог может содержать более одной страницы. Кнопки переключения между страницами, а также кнопка настройки количества объектов на странице, доступны в правом нижнем углу экранной формы.

ПРОСМОТР, СОЗДАНИЕ, РЕДАКТИРОВАНИЕ И ВЫГРУЗКА В PDF ОТЧЕТОВ

1. Для **просмотра отчёта** наведите курсор на символ меню (три точки) справа в строке соответствующего отчёта и нажмите "Просмотр". Произойдет переход на страницу отчёта.
2. Для **создания нового отчёта** нажмите кнопку "Создать" в правой верхней части окна каталога отчётов. Подробнее - см. раздел "создание отчёта".
3. Для **редактирования отчёта** перейдите на страницу отчёта (см. п.1). На странице отчёта наведите курсор на символ меню (три точки) в правом верхнем углу экрана и нажмите "Редактировать отчёт".
4. Для **выгрузки отчёта в PDF** перейдите на страницу отчёта (см. п.1). На странице дашборда наведите курсор на символ меню (три точки) в правом верхнем углу экрана и нажмите "Выгрузить в PDF".

ОБНОВЛЕНИЕ ИНФОРМАЦИИ В ОТЧЁТЕ

Отчёт не статичен, карточки с графическими и скалярными результатами расчета метрик *обновляются при перезапуске исходных проектов*. Таким образом, отчёт всегда отражает состояние на момент последних запусков исходных проектов.
Зафиксировать состояние отчёта можно выгрузкой его в pdf.

Создание отчёта

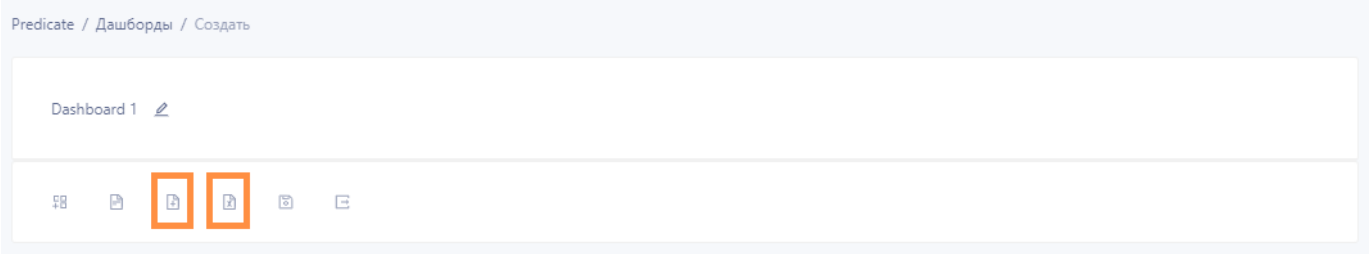
Для начала работы с конструктором пользовательских дашбордов необходимо перейти из главного меню на страницу создания отчёта: Каталог > Отчёты > Создать.

Элементами отчёта являются **графики** и **текстовые панели**. Положение на странице и размеры элементов можно менять мышкой в процессе конструирования отчёта.

Отчёт может состоять из произвольного числа страниц.

ДОБАВЛЕНИЕ И УДАЛЕНИЕ СТРАНИЦ ОТЧЁТА

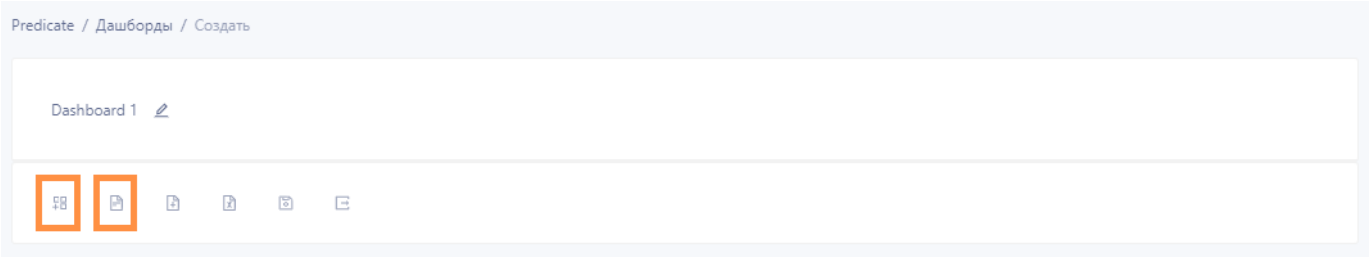
Кнопки добавления/удаления страниц отчёта расположены в средней части панели инструментов:



- кнопка добавления страницы в конец отчёта (слева)
- кнопка удаления текущей страницы (справа)

ДОБАВЛЕНИЕ ЭЛЕМЕНТА

Кнопки добавления элементов расположены в левой части панели инструментов:



- кнопка добавления карточки графика (слева)
- кнопка добавления текстовой панели (справа)

После нажатия одной из названных кнопок в конце текущей страницы отчёта появится новая карточка. Порядок, расположение и размер карточек можно позже изменить мышкой.

Автоматический перенос элементов между страницами не предусмотрен. Если необходимо добавить элемент на страницу, отличную от текущей, сначала явно перейдите на нужную страницу.

НАСТРОЙКА КАРТОЧКИ ГРАФИКА

1. Для начала настройки наведите курсор на символ меню (три точки) в правом верхнем углу карточки и нажмите "Edit".
2. Добавьте на карточку нужные графики из открывшегося каталога.
3. Отредактируйте название карточки.
4. Нажмите кнопку "Сохранить".

Окно редактирования карточки графика:

НАСТРОЙКА ТЕКСТОВОЙ ПАНЕЛИ

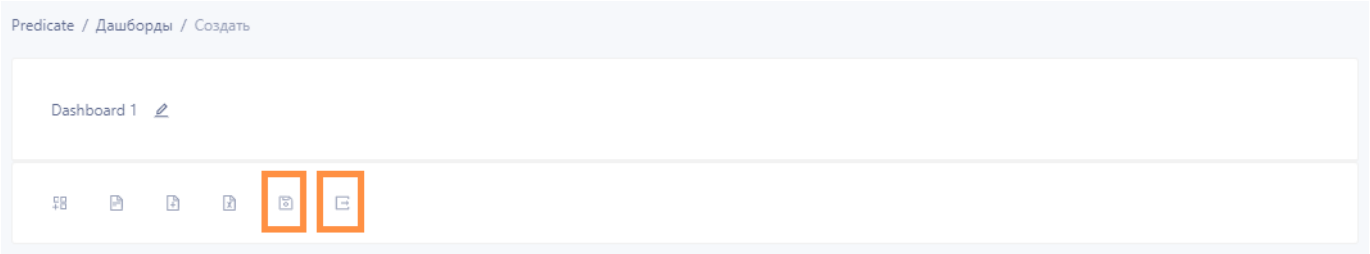
Нажмите на значок редактирования и введите текст, который должен отображаться на текстовой панели.

УДАЛЕНИЕ ЭЛЕМЕНТА

Для удаления карточки наведите курсор на символ меню (три точки) в ее правом верхнем углу и нажмите "Delete".

СОХРАНЕНИЕ ОТЧЁТА

Кнопки сохранения и выгрузки отчёта расположены в правой части панели инструментов:



- кнопка сохранения отчёта в приложении Predicate (слева);
- кнопка выгрузки отчёта в формате pdf на компьютер пользователя (справа).

Завершите создание отчёта нажатием одной из названных кнопок.

Сохранение отчёта в файл pdf

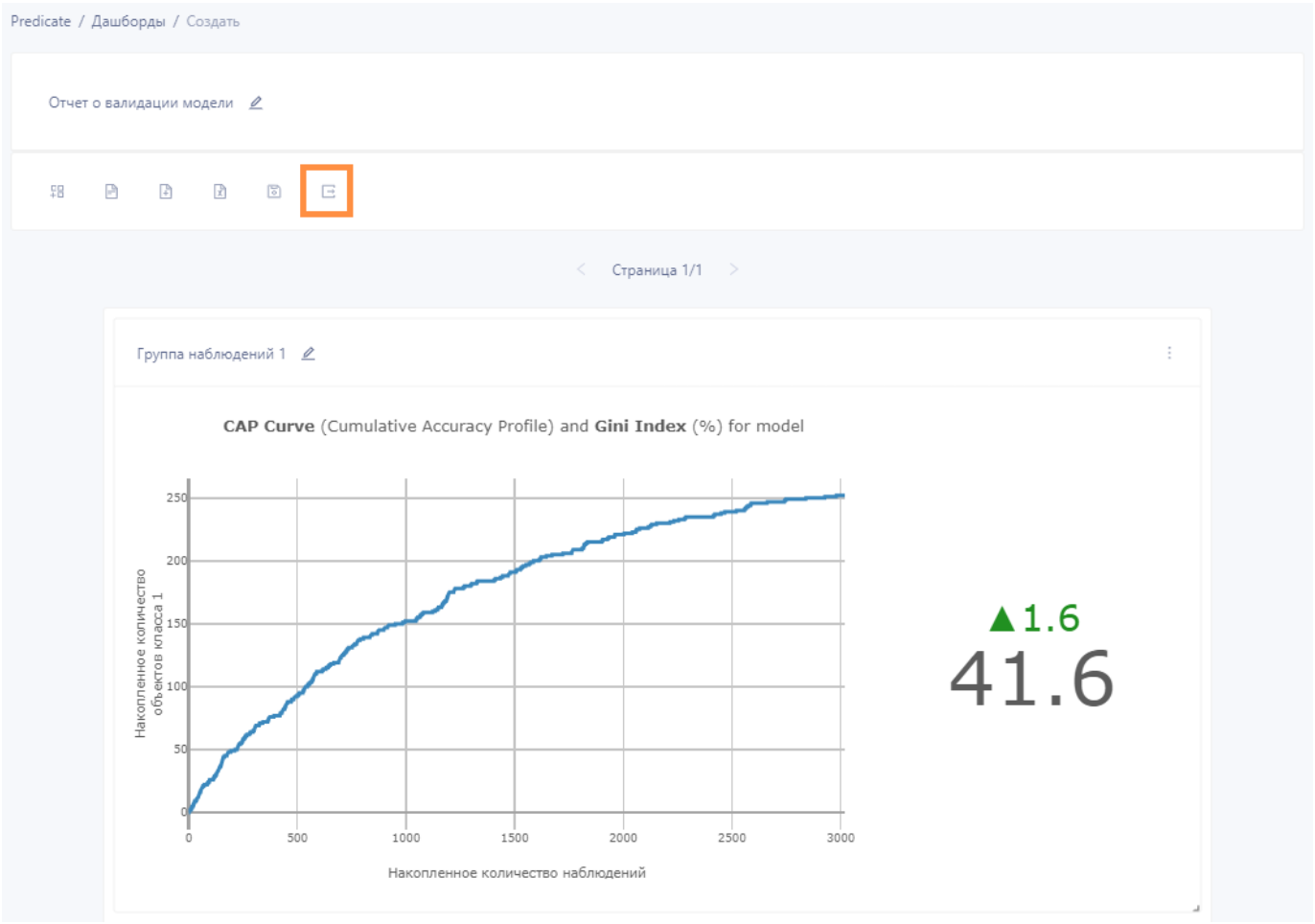
Выгрузка отчёта (пользовательского дашборда) в файл pdf возможна двумя способами:

- при создании отчета (в этом случае возможна выгрузка без промежуточного сохранения в каталоге отчётов);
- выгрузка отчёта, ранее сохраненного в каталоге отчетов.

ВЫГРУЗКА В PDF ПРИ СОЗДАНИИ ОТЧЕТА

1. Создайте отчет согласно инструкции.
2. Сохраните готовый отчёт на компьютер пользователя в формате pdf нажатием кнопки выгрузки в файл pdf, которая находится справа на панели инструментов конструктора отчёта.

Страница конструирования отчёта:



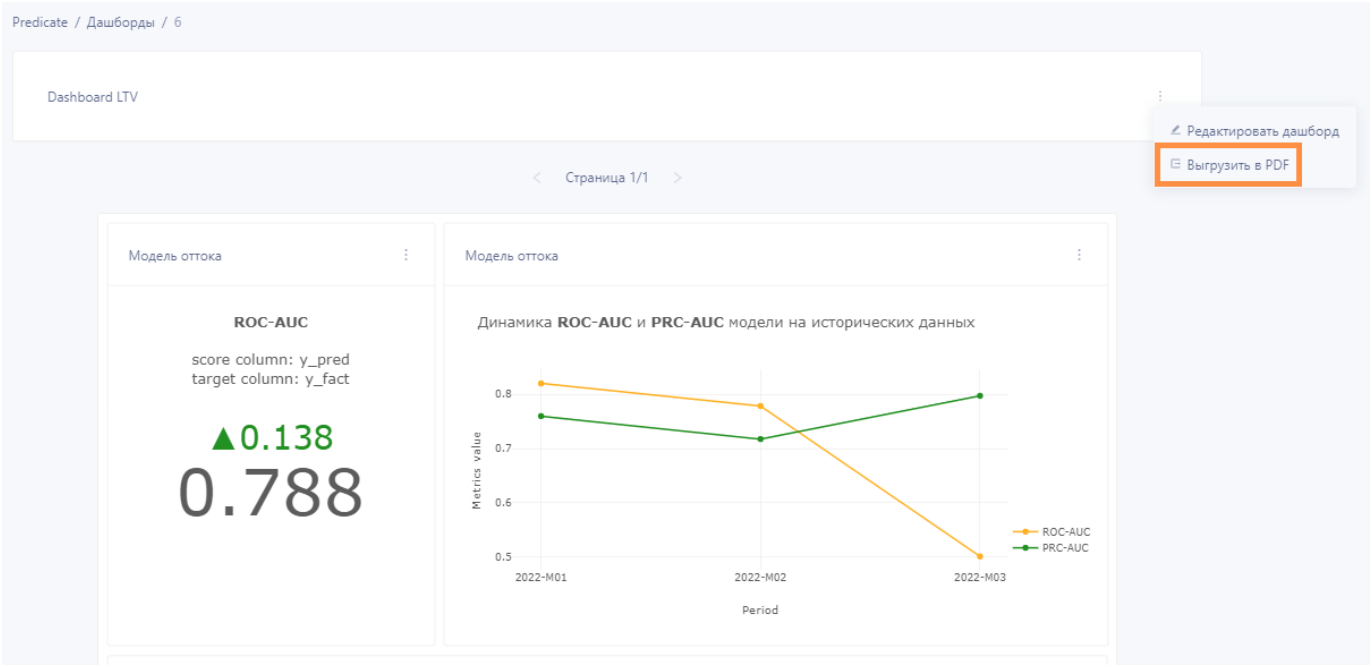
После выгрузки отчёта описанным способом можно сохранить отчёт в каталоге (нажав кнопку с символом дискеты на панели инструментов), чтобы в дальнейшем продолжить редактирование.

Если покинуть страницу создания отчёта не нажав кнопку сохранения, созданный черновик отчета будет удален.

ВЫГРУЗКА В PDF ОТЧЁТА ИЗ КАТАЛОГА

1. Из каталога отчётов перейдите на страницу необходимого отчёта. (Для этого в каталоге наведите курсор на символ меню (три точки) справа в строке соответствующего отчета и нажмите "Просмотр".)
2. На странице отчёта наведите курсор на символ меню (три точки) в правом верхнем углу экрана и нажмите "Выгрузить в PDF".

Страница отчёта:



3.3 Шаблоны

3.3.1 Шаблоны проектов

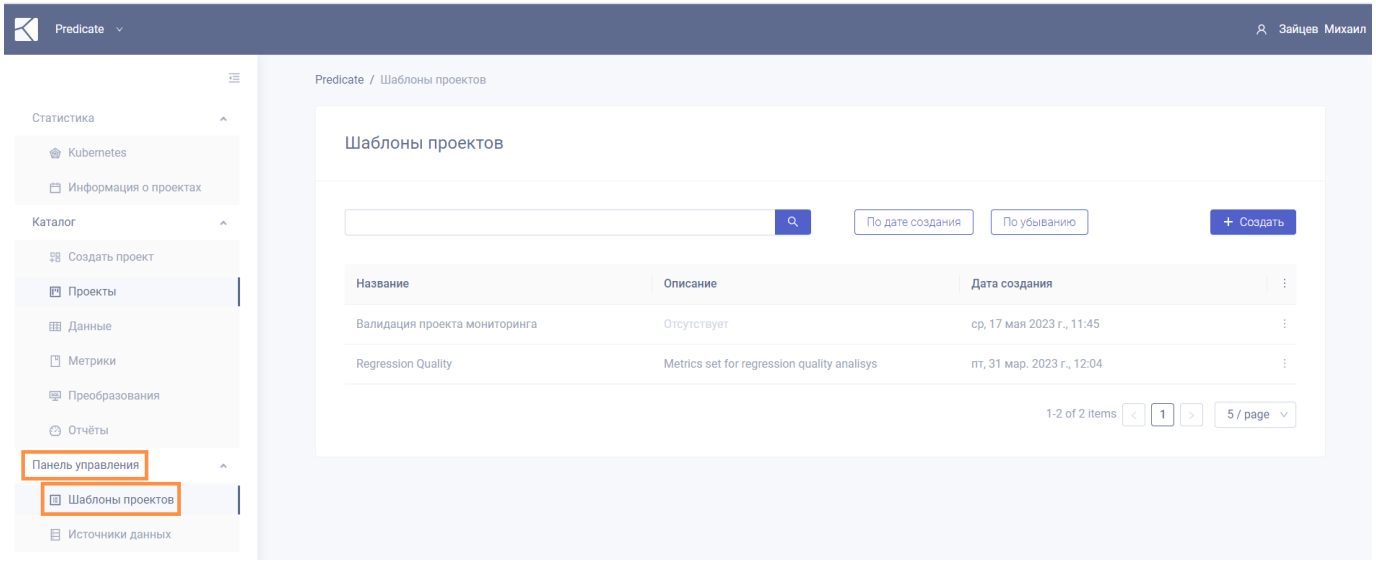
Predicate - Шаблоны проектов

Шаблон проекта представляет из себя преднастроенный набор метрик. Опционально в шаблоне может быть сохранено преднастроенное расписание мониторинга. При работе с шаблоном проекта для запуска расчетов пользователю остается выбрать только набор данных.

ПЕРЕХОД К КАТАЛОГУ ШАБЛОНОВ

Переход к каталогу имеющихся в системе шаблонов проектов осуществляется из основного меню: *Панель управления* > *Шаблоны проектов*.

Страница каталога шаблонов проектов:



УПРАВЛЕНИЕ КАТАЛОГОМ

В верхней части экранной формы доступны кнопки **сортировки каталога** по дате создания или названию шаблона, в убывающем или возрастающем порядке.
По умолчанию шаблоны в каталоге упорядочены по дате регистрации в системе - от новых к более ранним.

Также доступен **поиск**.
Введите подстроку в поле поиска и нажмите `Enter` или символ лупы. Регистр не важен, поиск происходит по названию и описанию шаблонов, а также тэгам.

Каталог может содержать более одной страницы. Кнопки переключения между страницами, а также кнопка настройки количества объектов на странице, доступны в правом нижнем углу экранной формы.

ПРОСМОТР И СОЗДАНИЕ ОБЪЕКТОВ

Для **просмотра информации о шаблоне** наведите курсор на символ меню (три точки) справа в строке соответствующего шаблона и нажмите "Просмотр".

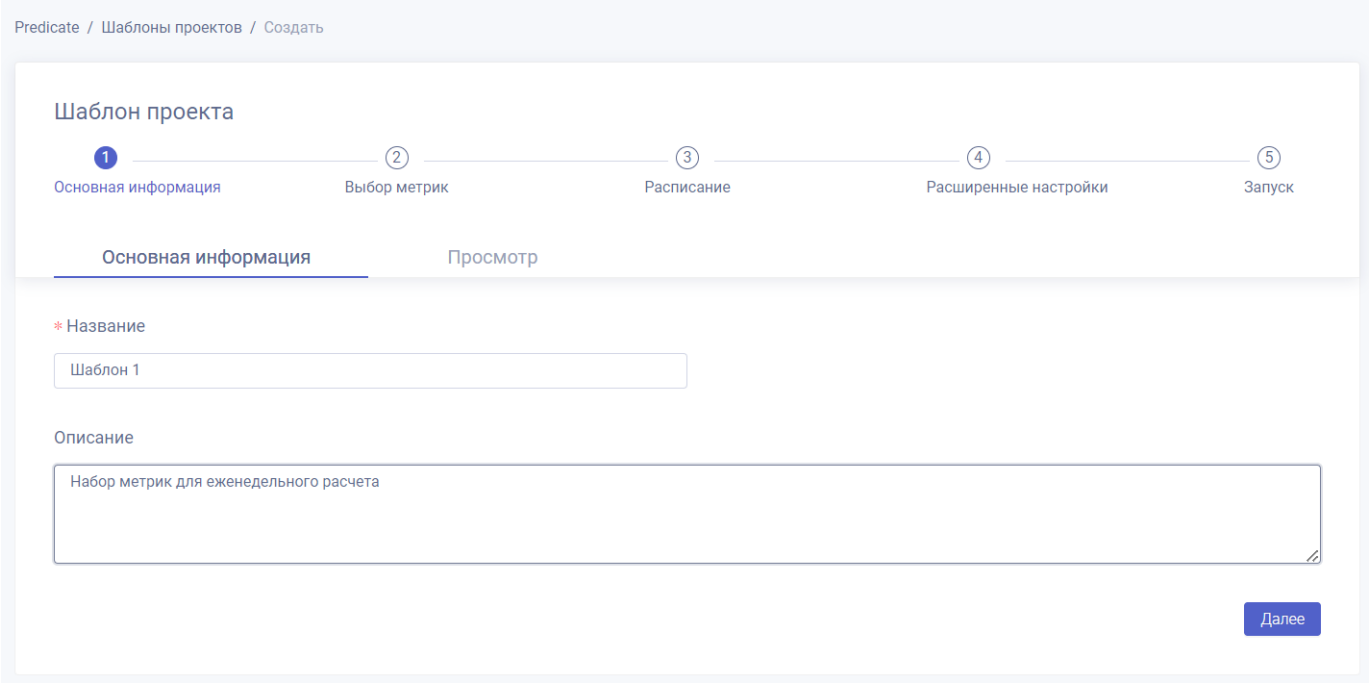
Для **создания нового шаблона** нажмите кнопку "Создать" в правой верхней части окна каталога шаблонов. Подробнее - см. раздел "создание нового шаблона".

Создание нового шаблона

ПЕРЕХОД К ФОРМЕ СОЗДАНИЯ ШАБЛОНА ПРОЕКТА

Для создания нового шаблона перейдите в раздел “Шаблоны проектов” основного меню приложения и на открывшейся странице каталога шаблонов нажмите кнопку “Создать”.

Откроется форма создания шаблона проекта:



Процесс создания шаблона проекта состоит из 5 этапов и почти полностью аналогичен процессу создания нового проекта.

ЭТАП "ОСНОВНАЯ ИНФОРМАЦИЯ"

На этапе "**Основная информация**" введите название (обязательно) и описание (опционально) шаблона. Нажмите "Далее".

ЭТАП "ВЫБОР МЕТРИК"

Список метрик шаблона

На этапе "**Выбор метрик**" нажмите "Добавить". В открывшемся окне каталога метрик отметьте необходимые метрики нажав на "+" справа от названия метрики.
Допускается выбор метрики более одного раза для одного шаблона.

Обратите внимание, что каталог метрик может состоять из нескольких страниц.

Когда все необходимые метрики отмечены, нажмите кнопку "Выбрать".

Сохранение параметров метрик

Сохранение значений параметров метрик при создании шаблона проекта является опциональным.

Если значение параметра метрики не задано при создании шаблона, его можно будет задать позже, при создании проекта из шаблона.

Окно настройки параметров открывается нажатием кнопки "**Параметры**" справа от названия метрики.

> **Важно**
>
> В шаблонах **допускается** сохранение значений параметров типов: **range**, **bool**, **int-value**, **float-value**, **dropdown**, **multi-dropdown**.
>
> **НЕ допускается** сохранение в шаблонах значений параметров типов: **dataframe**, **column**, **date**, **time**. Параметры этих типов задаются при создании проекта.

Завершите этап нажав "Далее".

ЭТАП "РАСПИСАНИЕ"

На этапе "**Расписание**" выберите необходимые параметры регламентных запусков, или пропустите этот этап.

Во втором случае расписание можно будет настроить позже, при создании проекта из шаблона.

Для перехода к следующему этапу нажмите "Далее".

ЭТАП "РАСШИРЕННЫЕ НАСТРОЙКИ"

Этап "Расширенные настройки" является опциональным.

В разделе Execution Config отрегулируйте количество ресурсов, которое будет выделяться под проект при каждом из запусков.

В поле настройки тэгов выберите из дропдауна один или несколько тэгов, которые будут прикреплены к шаблону. В каталоге шаблонов будет доступен поиск по тэгам.

ЭТАП "ЗАПУСК"

На этапе "**Запуск**" проверьте сохраненную информацию.

Если необходимо внести исправления, вернитесь на предыдущий этап с помощью кнопки "Назад".

Если все указано верно, завершите создание шаблона нажатием кнопки "Сохранить".

ПРОСМОТР СОЗДАННОГО ШАБЛОНА

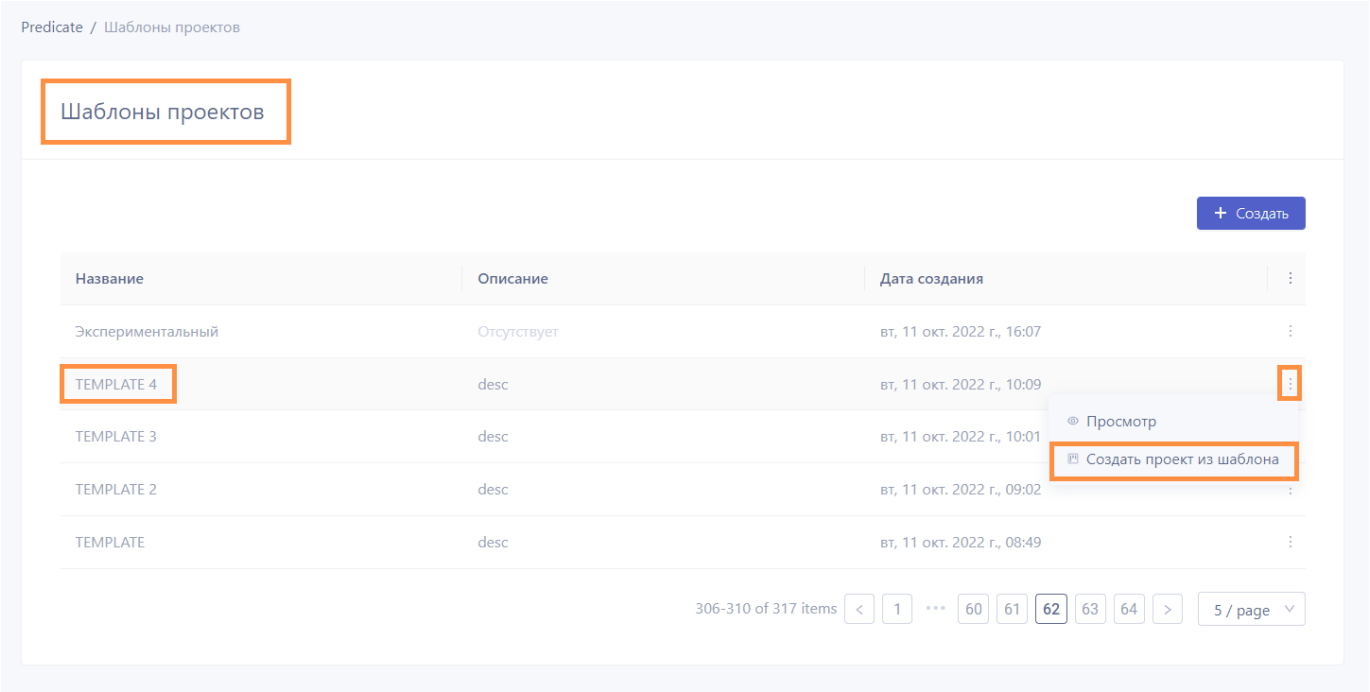
После того как шаблон проекта сохранен, он становится доступен в каталоге шаблонов: *Панель управления > Шаблоны проектов*.

Алгоритм создания проекта мониторинга из шаблона описан в разделе "создание проекта из шаблона".

Создание проекта из шаблона

Для создания проекта мониторинга из сохраненного ранее шаблона:

На странице каталога шаблонов (*Панель управления* > *Шаблоны проектов*) выберите нужный шаблон, наведите курсор на символ меню (три точки) в правой части соответствующей строки и нажмите “Создать проект из шаблона”.



Откроется форма создания нового проекта, часть полей которой будут заполнены автоматически в соответствии с настройками выбранного шаблона. Оставшиеся поля следует заполнить согласно инструкции (начиная с п.2) из раздела "создание нового проекта".

3.4 Работа в приложении

3.4.1 Начало работы

Доступ пользователя к приложению Kolmogorov.ai **Predicate** осуществляется через веб-интерфейс браузера.

Для начала работы в приложении необходимо авторизоваться.

Системные требования

Для использования приложения необходим один из перечисленных браузеров:

развернуть список браузеров

- Firefox 63.0.1 или выше
- Chrome 70.0.3538.67 или выше
- Safari 12 или выше
- Opera 56.0.3051.99 или выше
- Edge 80.0 или выше

А также подключение к корпоративной сети с доступом к серверу, на котором установлено ПО.

Требования к квалификации пользователя

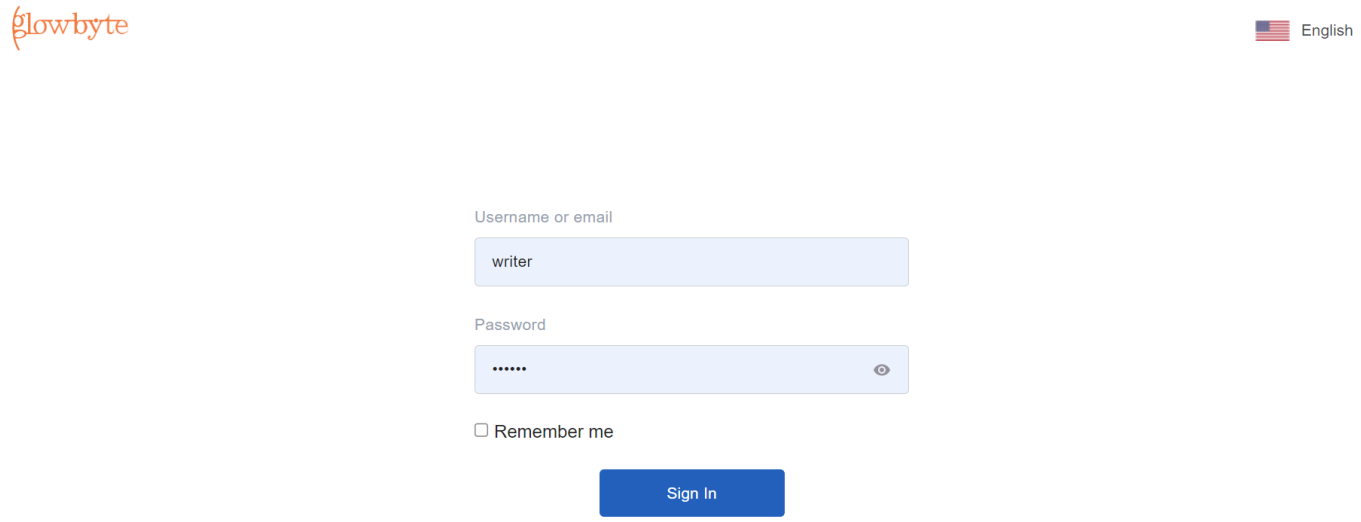
- Пользователь приложения **Predicate** должен обладать базовыми навыками компьютерной грамотности, а также теоретическими знаниями о принципах работы заложенных в систему тестов и метрик.
- Для регистрации источников данных и создания проектов мониторинга пользователь приложения **Predicate** должен знать структуру используемых источников данных.

Авторизация

ВХОД В ПРИЛОЖЕНИЕ

Для входа в приложение **Predicate** в адресной строке браузера введите URL: https://predicate-demo.k8s.datasapience.ru/project

Откроется страница авторизации:



Введите логин (Username) и пароль (Password) в соответствующие поля.
Узнать логин и пароль можно у администратора системы.

Если необходимо сохранить данные авторизации, установите флаг в поле «Remember me».

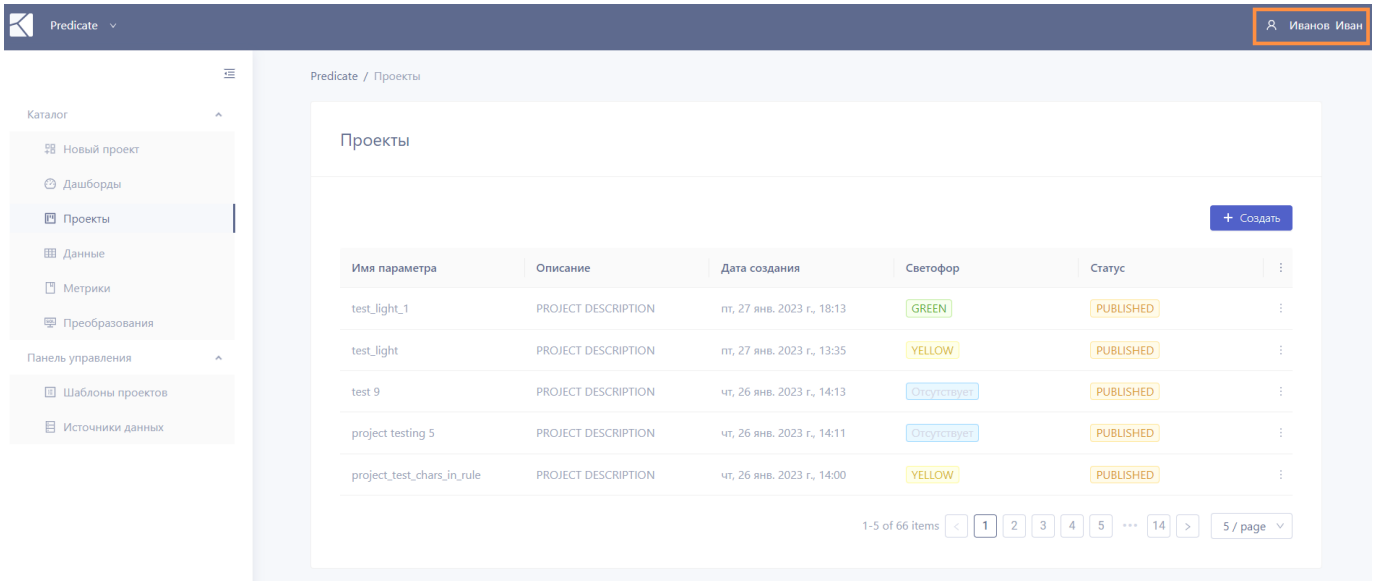
Завершите авторизацию нажатием кнопки «Sign In».

УРОВЕНЬ ДОСТУПА

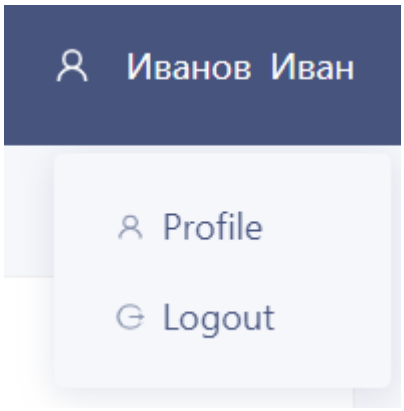
В интерфейсе пользователю видны и доступны для работы все объекты, которые он создал самостоятельно, а также те объекты, к которым ему был специально предоставлен доступ. Подробнее см. раздел "ролевая модель".

ПРОСМОТР ПРОФИЛЯ ПОЛЬЗОВАТЕЛЯ

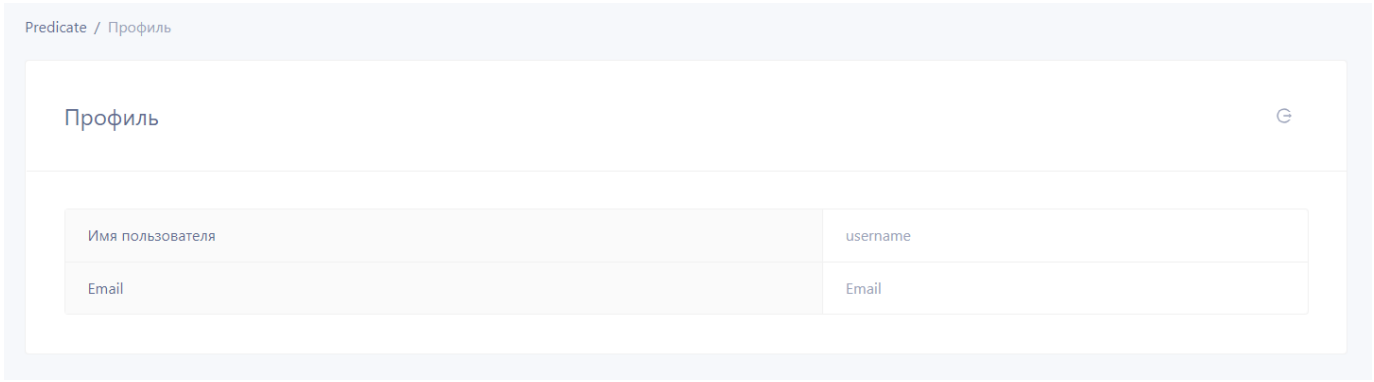
После входа в систему имя пользователя отображается в правой верхней части экрана.



При наведении курсора на имя пользователя всплывает меню:



Выберите «Profile» для просмотра информации о профиле пользователя (имени и электронной почте):



ВЫХОД ИЗ ПРОФИЛЯ И СМЕНА ПОЛЬЗОВАТЕЛЯ

Для выхода из профиля наведите курсор на имя пользователя и выберите во всплывающем меню опцию "Logout".

После выхода из профиля произойдет перенаправление на страницу авторизации, откуда можно войти в систему под другим именем пользователя.

3.4.2 Сценарий: создание проекта мониторинга

Функция "**проект мониторинга**" приложения Predicate позволяет проводить расчет метрик над выбранными данными. Результатом работы проекта мониторинга является автоматически формируемый дашборд, включающий скалярные значения метрик и графическую информацию.

Для создания проекта необходимо выполнить следующие шаги:

1. **Подготовить данные.**
   Для этого необходимо выбрать датасет из каталога или зарегистрировать в приложении новый объект данных одним из следующих способов:
   - загрузив файл .csv;
   - выбрав один из имеющихся в s3-хранилище файлов .csv;
   - выбрав таблицу в базе данных SQL.
2. **Определиться с метриками.**
   Доступен как выбор метрик из каталога, так и создание авторской метрики пользователем.
   В случае создания авторской метрики необходимо сначала написать код метрики на Python в соответствии с заданной структурой, а затем загрузить файл с кодом метрики в систему через интерфейс приложения.
3. **Создать проект мониторинга через интерфейс приложения** согласно инструкции - последовательно заполнив необходимые поля экранных форм на каждом из этапов регистрации проекта.

После создания проект мониторинга появляется в каталоге.
Просмотр результатов проекта доступен со страницы каталога проектов в любое время после завершения расчетов.

4. Developer Guide

4.1 Введение

Приложение Kolmogorov.ai **Predicate** позволяет тестировать модели и проверять качество данных рассчитывая метрики по регламенту и формируя отчеты.

4.1.1 Технологии

Backend приложения разработан на языке *Python* с использованием фреймворка *FastAPI*, библиотек *SqlAlchemy*, *Alembic*, *Pydantic*, *Celery*. В качестве базы данных backend используется *Postgres*.

Web-интерфейс разработан на языке *JavaScript* с использованием фреймворка *React* и дизайн-системы *Ant*.

Компонентная архитектура приложения

4.1.2 Сервисы

- Predicate-backend - основной сервис бэкенда
- Service-celery - запускает воркер для celery
- Service-gitsync - сервис для синхронизации состояния репозитория git-share и volume в кластере
- Service-flower - мониторинг и администрирование заданий celery
- Postgresql - база данных Predicate
- Redis - брокер сообщений для celery задач
- Predicate-web - основной сервис фронтенда

Команды запуска сервисов находятся в модуле bin.

4.1.3 Predicate API

Swagger-документация: скачать в формате JSON

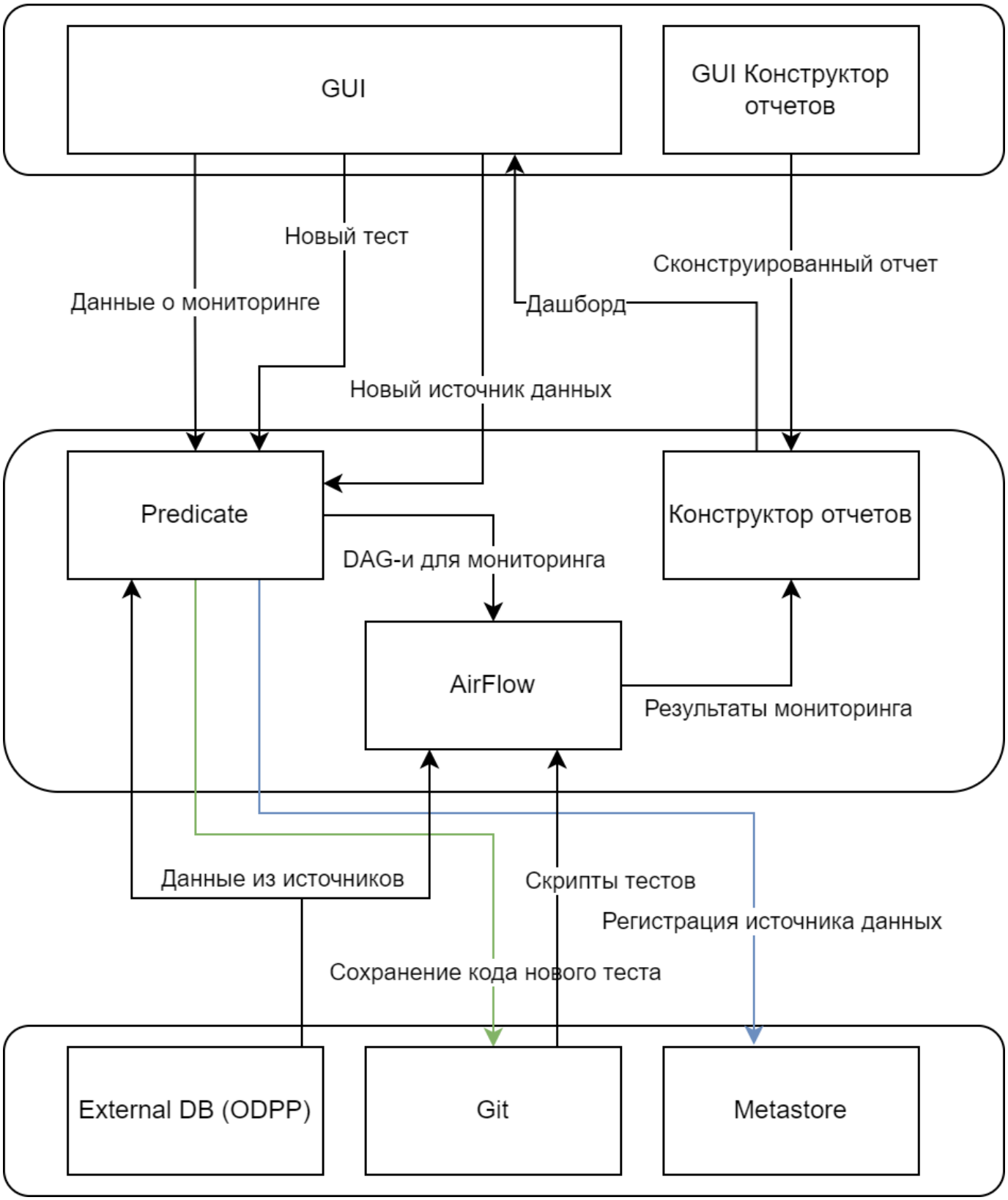
4.1.4 Объектная модель Predicate

- ORM-объекты
- Pydantic-схемы
- Схема базы данных

4.2 Архитектура приложения Predicate

Приложение Predicate состоит из трех слоев:

1. Пользовательский интерфейс;
2. Бизнес-уровень;
3. Уровень встроенных баз данных и подключения к внешним базам данных.


GUI
GUI Конструктор отчетов
Новый тест
Данные о мониторинге
Дашборд
Сконструированный отчет
Новый источник данных
Predicate
Конструктор отчетов
DAG-и для мониторинга
AirFlow
Результаты мониторинга
Данные из источников
Скрипты тестов
Регистрация источника данных
Сохранение кода нового теста
External DB (ODPP)
Git
Metastore

4.3 Пререквизиты

Компоненты, которые не входят в дефолтную поставку приложения Predicate, однако необходимы для его работы.

Об установке перечисленных ниже компонент можно прочитать в разделе Install.

4.3.1 Kubernetes

Должен быть развернуть **кластер Kubernetes** и установлен инструмент для управления Kubernetes приложениями **Helm**.

4.3.2 Keycloak

Keycloak - сервис для авторизации и аутентификации.

4.3.3 Gitlab

Репозитории Gitlab используются для хранения файлов метрик, преобразований (моделей) и DAG проектов Predicate.

4.3.4 Airflow

Airflow нужен для исполнения DAGs проектов, сгенерированных Predicate, по расписанию или разово, если расписание не указано.

4.3.5 Minio

S3 хранилище для временных файлов и результатов метрик при исполнении DAG.

4.3.6 Datasources

Источники данных для проектов Predicate.
На данный момент поддерживаются следующие типы источников: **S3**, **PostgreSQL**, **Oracle**, **Snowflake**, **Hive** (с подключением через Kerberos) и **Axiom** (feature store от Kolmogorov.ai).

4.4 База данных

4.4.1 База данных Predicate

Для хранения данных backend приложения используется Postgres.

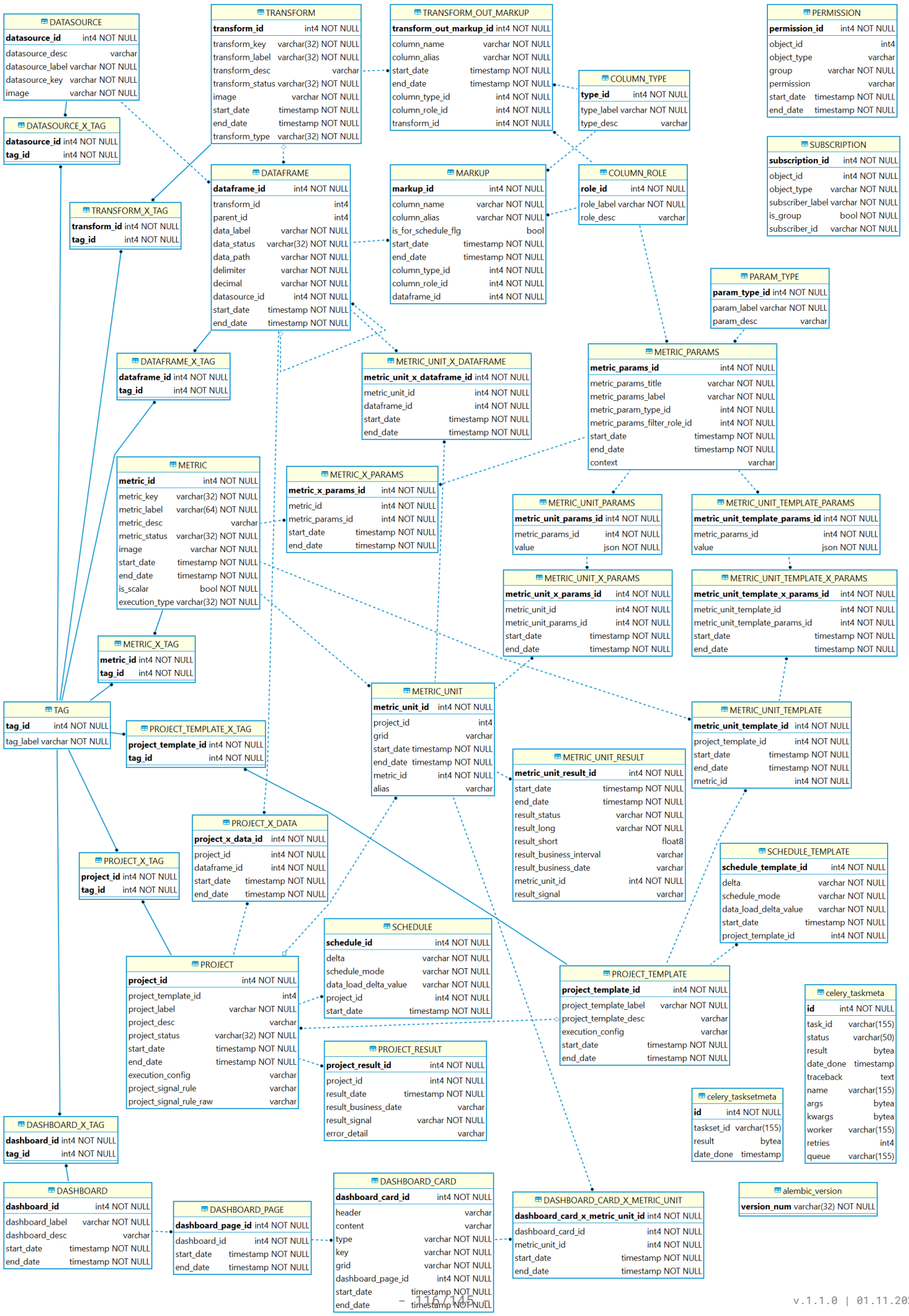
ER-диаграмма базы данных представлена в разделе "Схема".

Описания таблиц объединены в группы по принадлежности к основным сущностям Predicate и доступны в разделе "Таблицы" на соответствующих страницах:

- PROJECT - таблицы **проекта**, результатов метрик проекта, расписания, светофора;
- DATAFRAME - таблицы **датасета** и разметки датасета;
- METRIC - таблицы **метрики**, параметров метрики;
- TRANSFORM - таблицы **преобразования** и разметки столбцов, добавляемых преобразованием;
- DASHBOARD - таблицы **отчета**, страницы отчета, карточки с графиком или текстом;
- PROJECT_TEMPLATE - таблицы с общей информацией о **шаблоне проекта**, набором метрик и шаблоном расписания;
- DATASOURCE - таблицы с информацией о подключении к **источнику данных**;
- Вспомогательные таблицы - связаны с несколькими сущностями или системой в целом: тип и роль столбца данных, теги (TAG), подписки (SUBSCRIPTION), ролевой доступ (PERMISSION), celery, alembic version.

Код Python-классов, соответствующих объектам базы данных, содержится в модуле `predicate/models`.

4.4.2 Схема базы данных Predicate


DATASOURCE
datasource_id int4 NOT NULL
datasource_desc varchar
datasource_label varchar NOT NULL
datasource_key varchar NOT NULL
image varchar NOT NULL
DATASOURCE_X_TAG
datasource_id int4 NOT NULL
tag_id int4 NOT NULL
TRANSFORM
transform_id int4 NOT NULL
transform_key varchar(32) NOT NULL
transform_label varchar(32) NOT NULL
transform_desc varchar
transform_status varchar(32) NOT NULL
image varchar NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
transform_type varchar(32) NOT NULL
TRANSFORM_OUT_MARKUP
transform_out_markup_id int4 NOT NULL
column_name varchar NOT NULL
column_alias varchar NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
column_type_id int4 NOT NULL
column_role_id int4 NOT NULL
transform_id int4 NOT NULL
COLUMN_TYPE
type_id int4 NOT NULL
type_label varchar NOT NULL
type_desc varchar
PERMISSION
permission_id int4 NOT NULL
object_id int4
object_type varchar
group varchar NOT NULL
permission varchar
start_date timestamp NOT NULL
end_date timestamp NOT NULL
SUBSCRIPTION
subscription_id int4 NOT NULL
object_id int4 NOT NULL
object_type varchar NOT NULL
subscriber_label varchar NOT NULL
is_group bool NOT NULL
subscriber_id varchar NOT NULL
DATAFRAME
dataframe_id int4 NOT NULL
transform_id int4
parent_id int4
data_label varchar NOT NULL
data_status varchar(32) NOT NULL
data_path varchar NOT NULL
delimiter varchar NOT NULL
decimal varchar NOT NULL
datasource_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
MARKUP
markup_id int4 NOT NULL
column_name varchar NOT NULL
column_alias varchar NOT NULL
is_for_schedule_flg bool
start_date timestamp NOT NULL
end_date timestamp NOT NULL
column_type_id int4 NOT NULL
column_role_id int4 NOT NULL
dataframe_id int4 NOT NULL
COLUMN_ROLE
role_id int4 NOT NULL
role_label varchar NOT NULL
role_desc varchar
TRANSFORM_X_TAG
transform_id int4 NOT NULL
tag_id int4 NOT NULL
PARAM_TYPE
param_type_id int4 NOT NULL
param_label varchar NOT NULL
param_desc varchar
DATAFRAME_X_TAG
dataframe_id int4 NOT NULL
tag_id int4 NOT NULL
METRIC_UNIT_X_DATAFRAME
metric_unit_x_dataframe_id int4 NOT NULL
metric_unit_id int4 NOT NULL
dataframe_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
METRIC_PARAMS
metric_params_id int4 NOT NULL
metric_params_title varchar NOT NULL
metric_params_label varchar NOT NULL
metric_param_type_id int4 NOT NULL
metric_params_filter_role_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
context varchar
METRIC
metric_id int4 NOT NULL
metric_key varchar(32) NOT NULL
metric_label varchar(64) NOT NULL
metric_desc varchar
metric_status varchar(32) NOT NULL
image varchar NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
is_scalar bool NOT NULL
execution_type varchar(32) NOT NULL
METRIC_X_PARAMS
metric_x_params_id int4 NOT NULL
metric_id int4 NOT NULL
metric_params_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
METRIC_UNIT_PARAMS
metric_unit_params_id int4 NOT NULL
metric_params_id int4 NOT NULL
value json NOT NULL
METRIC_UNIT_TEMPLATE_PARAMS
metric_unit_template_params_id int4 NOT NULL
metric_params_id int4 NOT NULL
value json NOT NULL
METRIC_UNIT_X_PARAMS
metric_unit_x_params_id int4 NOT NULL
metric_unit_id int4 NOT NULL
metric_unit_params_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
METRIC_UNIT_TEMPLATE_X_PARAMS
metric_unit_template_x_params_id int4 NOT NULL
metric_unit_template_id int4 NOT NULL
metric_unit_template_params_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
METRIC_X_TAG
metric_id int4 NOT NULL
tag_id int4 NOT NULL
METRIC_UNIT
metric_unit_id int4 NOT NULL
project_id int4
grid varchar
start_date timestamp NOT NULL
end_date timestamp NOT NULL
metric_id int4 NOT NULL
alias varchar
TAG
tag_id int4 NOT NULL
tag_label varchar NOT NULL
PROJECT_TEMPLATE_X_TAG
project_template_id int4 NOT NULL
tag_id int4 NOT NULL
METRIC_UNIT_TEMPLATE
metric_unit_template_id int4 NOT NULL
project_template_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
metric_id int4 NOT NULL
METRIC_UNIT_RESULT
metric_unit_result_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
result_status varchar NOT NULL
result_long varchar NOT NULL
result_short float8
result_business_interval varchar
result_business_date varchar
metric_unit_id int4 NOT NULL
result_signal varchar
PROJECT_X_DATA
project_x_data_id int4 NOT NULL
project_id int4 NOT NULL
dataframe_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
SCHEDULE_TEMPLATE
schedule_template_id int4 NOT NULL
delta varchar NOT NULL
schedule_mode varchar NOT NULL
data_load_delta_value varchar NOT NULL
start_date timestamp NOT NULL
project_template_id int4 NOT NULL
PROJECT_X_TAG
project_id int4 NOT NULL
tag_id int4 NOT NULL
SCHEDULE
schedule_id int4 NOT NULL
delta varchar NOT NULL
schedule_mode varchar NOT NULL
data_load_delta_value varchar NOT NULL
project_id int4 NOT NULL
start_date timestamp NOT NULL
PROJECT
project_id int4 NOT NULL
project_template_id int4
project_label varchar NOT NULL
project_desc varchar
project_status varchar(32) NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
execution_config varchar
project_signal_rule varchar
project_signal_rule_raw varchar
PROJECT_TEMPLATE
project_template_id int4 NOT NULL
project_template_label varchar NOT NULL
project_template_desc varchar
execution_config varchar
start_date timestamp NOT NULL
end_date timestamp NOT NULL
celery_taskmeta
id int4 NOT NULL
task_id varchar(155)
status varchar(50)
result bytea
date_done timestamp
traceback text
name varchar(155)
args bytea
kwargs bytea
worker varchar(155)
retries int4
queue varchar(155)
PROJECT_RESULT
project_result_id int4 NOT NULL
project_id int4 NOT NULL
result_date timestamp NOT NULL
result_business_date varchar
result_signal varchar NOT NULL
error_detail varchar
celery_tasksetmeta
id int4 NOT NULL
taskset_id varchar(155)
result bytea
date_done timestamp
DASHBOARD_X_TAG
dashboard_id int4 NOT NULL
tag_id int4 NOT NULL
DASHBOARD
dashboard_id int4 NOT NULL
dashboard_label varchar NOT NULL
dashboard_desc varchar
start_date timestamp NOT NULL
end_date timestamp NOT NULL
DASHBOARD_PAGE
dashboard_page_id int4 NOT NULL
dashboard_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
DASHBOARD_CARD
dashboard_card_id int4 NOT NULL
header varchar
content varchar
type varchar NOT NULL
key varchar NOT NULL
grid varchar NOT NULL
dashboard_page_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
DASHBOARD_CARD_X_METRIC_UNIT
dashboard_card_x_metric_unit_id int4 NOT NULL
dashboard_card_id int4 NOT NULL
metric_unit_id int4 NOT NULL
start_date timestamp NOT NULL
end_date timestamp NOT NULL
alembic_version
version_num varchar(32) NOT NULL

Перейти к описанию базы

4.4.3 Таблицы

Проект

Таблицы, связанные с сущностью **проект**.

PROJECT

Информация об имеющихся проектах мониторинга. Основная сущность

Column	Type	Constraint	Comment
project_id	int	pk	Meta
project_template_id	int	fk	id шаблона проекта
project_label	str	uniq	Label
project_desc	str	-	Описание
project_status	str	default='CREATED'	Статус проекта
start_date	timestamp	default=datetime.now	Meta
end_date	timestamp	default=5999/12/31	Meta

SCHEDULE

Информация о расписании исполнения проектов

Column	Type	Constraint	Comment
schedule_id	int	pk	Meta
project_id	int	fk	id проекта
delta	str	-	Частота работы регламента (monthly/ weekly/daily)
start_date	timestamp	default=datetime.now	Календарная дата начала работы регламента
schedule_mode	str	uniq	Режим забора данных из источника (most recent/calendar)
data_load_delta_value	str	-	Число дельт в рассматриваемом окне данных (1/2/3/all)

PROJECT_X_DATA

Many-to-many table. Информация о датасетах, участвующих в проектах

PROJECT_X_TAG

Ассоциативная таблица проекты - теги

METRIC_UNIT

Информация о метриках, участвующих в проектах

Column	Type	Constraint	Comment
metric_unit_id	int	pk	Meta
project_id	int	fk	id PROJECT
metric_id	int	fk	id METRIC
grid	str	-	Строковое представление для карточки на дашборде
start_date	timestamp	default=datetime.now	Meta
end_date	timestamp	default=5999/12/31	Meta

METRIC_UNIT_PARAMS

Значения параметров для расчета метрики

Column	Type	Constraint	Comment
metric_unit_params_id	int	pk	Meta
metric_unit_params_id	int	fk	id METRIC_PARAMS
value	json	-	Значение параметра метрики

METRIC_UNIT_PARAMS

Значения параметров для метрик, участвующих в проектах

METRIC_UNIT_X_PARAMS

Ассоциативная таблица: метрики проекта - значения параметров для этих метрик

METRIC_UNIT_X_DATAFRAME

Датасеты, используемые для расчета метрик в проекте

METRIC_UNIT_RESULT

Результат расчета метрики в рамках проекта

Column	Type	Constraint	Comment
metric_unit_result_id	int	pk	Meta
result_status	str	-	Статус исполнения metric_unit (1/-1)
result_short	str	-	Скалярный результат (40 / 40, 60, 20.09)
result_long	srt	-	Имя json файла графика в s3
result_business_interval	srt	-	Интервал среза данных, при использовании расписания ([2022-07-01 00:00:00, 2022-08-30 00:00:00])
result_business_date	srt	-	Правая граница интервала, при использовании расписания (2022-08-30 00:00:00)
start_date	timestamp	default=datetime.now	Meta
end_date	timestamp	default=5999/12/31	Meta

PROJECT_RESULT

Результирующие значения светофоров проектов

Датасет

Таблицы, связанные с сущностью **датасет**.

DATAFRAME

Зарегистрированные в системе объекты данных (сущности из источников данных - таблицы, csv). Основная сущность

Column	Type	Constraint	Comment
dataframe_id	int	pk	Meta
datasource_id	int	fk	id источника данных
project_label	str	uniq	Label
project_status	str(32)	default='CREATED'	Статус проекта
data_path	str	-	Имя таблицы/файла
decimal	str	-	Десятичный разделитель для csv
delimiter	str	-	Разделитель колонок csv
start_date	timestamp	default=datetime.now	Meta
end_date	timestamp	default=5999/12/31	Meta

MARKUP

Разметка колонок для dataframe

Column	Type	Constraint	Comment
markup_id	int	pk	Meta
dataframe_id	int	fk	id DATAFRAME
column_role_id	int	fk	id COLUMN_ROLE
column_type_id	int	fk	id COLUMN_TYPE
column_name	str		Имя колонки
column_alias	str	default='CREATED'	Алиас колонки
is_for_schedule_flg	bool	-	*Одна* из колонок, выбранная для исполнения расписания
start_date	timestamp	default=datetime.now	Meta
end_date	timestamp	default=5999/12/31	Meta

DATAFRAME_X_TAG

Ассоциативная таблица источники данных - теги

Метрика

Таблицы, связанные с сущностью **метрика**.

METRIC

Описание метрики качества модели/данных. Основная сущность

Column	Type	Constraint	Comment
metric_id	int	pk	Meta
metric_key	str(32)	-	id PROJECT_TEMPLATE
metric_label	str(64)	uniq	Label
metric_desc	str	-	Описание
metric_status	str(32)	default='CREATED'	Статус проекта
image	str	-	Docker image, на котором исполняется метрика
start_date	timestamp	default=datetime.now	Meta
end_date	timestamp	default=5999/12/31	Meta

METRIC_PARAMS

Информация о параметрах метрик. Основная сущность

Column	Type	Constraint	Comment
metric_params_id	int	pk	Meta
metric_params_type_id	str	fk	id PARAM_TYPE
metric_filter_role_id	str	fk	id COLUMN_ROLE
metric_params_title	str	-	Название параметра
metric_status_lable	str	default='CREATED'	Описание параметра
image	str	-	Docker image, на котором исполняется метрика
start_date	timestamp	default=datetime.now	Meta
end_date	timestamp	default=5999/12/31	Meta

PARAM_TYPE

Справочная таблица типов параметров метрик

Column	Type	Constraint	Comment
param_id	int	pk	Meta
param_label	str	uniq	Label
param_desc	str	-	Описание

METRIC_X_PARAMS

Информация о входных параметрах, необходимых для работы метрик. Many-to-many table

METRIC_X_TAG

Информация о тегах, присвоенных метрикам

METRIC_UNIT

Примечание

Таблицы, в названии которых присутствует `METRIC_UNIT` содержат информацию о метриках, участвующих в проектах мониторинга. Информаця об этих таблицах доступна на странице PROJECT.

Преобразование

Таблицы, связанные с сущностью **преобразование**.

TRANSFORM

Общая информация о преобразованиях (моделях)

TRANSFORM_OUT_MARKUP

Информация о разметке столбцов, добавляемых преобразованиями ко входным датасетам

TRANSFORM_X_TAG

Ассоциативная таблица преобразования – теги

Отчет (пользовательский дашборд)

Таблицы, связанные с сущностью **отчет**.

DASHBOARD

Информация об имеющихся отчетах. Основная сущность

DASHBOARD_PAGE

Информация о страницах отчета

DASHBOARD_CARD

Информация о карточках графиков и текстовых блоках, размещенных на страницах отчета

DASHBOARD_CARD_X_METRIC_UNIT

Информация о тех метриках, рассчитанных в рамках проектов мониторинга, которые отображаются на карточках графиков в отчетах

DASHBOARD_X_TAG

Ассоциативная таблица отчеты – теги

Шаблон проекта

Таблицы, которые используются для создания **шаблона проекта** (сущность PROJECT_TEMPLATE). Представляют собой копии соответствующих таблиц сущности PROJECT.

PROJECT_TEMPLATE

Информация о шаблоне проекта. Основная сущность

SCHEDULE_TEMPLATE

Информация о сохраненных настройках расписания

METRIC_UNIT_TEMPLATE

Информация о метриках, сохраненных в шаблоне

METRIC_UNIT_TEMPLATE_PARAMS

Предсохраненные значения параметров для метрик шаблона

METRIC_UNIT_TEMPLATE_X_PARAMS

Ассоциативная таблица: метрики шаблона - сохраненные значения параметров для этих метрик

Источник данных

Таблицы, связанные с сущностью **источник данных**.

DATASOURCE

Информация об источниках данных. Основная сущность

Column	Type	Constraint	Comment
datasource_id	int	pk	Meta
datasource_label	str	uniq	Label
datasource_desc	str	-	Описание
project_key	str	uniq	??Тип источника данных (s3/psql/oracle...)
image	str	-	Docker image, на котором основывается источник

DATASOURCE_X_TAG

Ассоциативная таблица источники данных - теги

Вспомогательные таблицы

Таблицы, связанные с несколькими сущностями сразу или системой в целом.

COLUMN_ROLE

Справочная таблица ролей столбцов данных (dataframe, metric, transform)

Column	Type	Constraint	Comment
role_id	int	pk	Meta
role_label	str	uniq	Label
role_desc	str	-	Описание

COLUMN_TYPE

Справочная таблица типов столбцов данных (dataframe, transform)

Column	Type	Constraint	Comment
type_id	int	pk	Meta
type_label	str	uniq	Label
type_desc	str	-	Описание

TAG

Справочная таблица тегов (project, dataframe, metric, transform, dashboard, datasource, project_template)

PERMISSION

Таблица ролей

Column	Type	Constraint	Comment
permission_id	int	pk	Meta
object_id	int	-	id сущности базы
object_type	str	-	Класс сущности из базы
group	str	-	Группа ролей
permission	str	-	Уровень доступа

SUBSCRIPTION

Таблица с информацией о подписках на уведомления.

ALEMBIC_VERSION

Таблица с информацией об id последней версии базы данных.

CELERY_TASKMETA

Служебная таблица с информацией о фоновых задачах. Не входит в объектную модель Predicate, автоматически формируется библиотекой celery.

CELERY_TASKSETMETA

Служебная таблица с информацией о группах фоновых задач. Не входит в объектную модель Predicate, автоматически формируется библиотекой celery.

4.5 Backend

4.5.1 Predicate Backend

Основной сервис бэкенда - FastAPI сервер, который принимает запросы по адресу, описанному в ingress сервиса. Часть задач выполняется асинхронно с использованием Celery. Данные хранятся в БД Postgres.

Для каждого проекта генерируется DAG Airflow, который позволяет мониторить данные, метрики и модели по регламенту.

Код серверной части приложения хранится в модуле `predicate`.

`predicate/predicate`

Основной бэкенд, включает подмодули:

- `api` - API сервер на базе FastAPI
- `bin` - CLI (command line interface) приложения. Точка входа в Predicate
- `celery` - управление фоновыми задачами
- `core` - конфигурация приложения и оберток внешних сервисов
- `crud` - классы с определением типовых методов взаимодействия с базой данных
- `models` - ORM классы базы данных на основе SqlAlchemy
- `schemas` - проверка и расширение описаний объектов, проходящих через API
- `utility` - вспомогательные функции, переиспользуемые в нескольких блоках backend

`predicate/packages`

Зависимости, исходный код которых поставляется вместе с приложением Predicate:

- `fastapi-utils` - вспомогательные инструменты для FastAPI
- `git-share` - библиотека стандартных метрик Predicate
- `git-utils` - интерфейс для работы с удаленными git-репозиториями
- `klmg-predicate-types` - библиотека типов Predicate

`predicate/plugins`

`project-subscription` - плагин для отправки уведомлений по результатам работы проектов

`predicate/tests`

Unit-тесты

predicate/Makefile

В `Makefile` находятся основные команды:

- `server/dev` - запуск FastAPI сервера
- `celery_worker` - запуск Celery-воркера
- `celery_flower` - запуск Flower для мониторинга и администрирования заданий Celery
- `check_code` - проверка кода линтерами (flake8, black, isort) и статическая проверка типов (mypy);
- `docker/build` - сборка docker образа;
- `docker/push` - пуш docker образа в docker registry;
- `helm/install` - установка helm chart в kubernetes\openshift\minikube;
- `helm/delete` - удаление helm chart из kubernetes\openshift\minikube;

4.5.2 predicate

api

Путь: `predicate/predicate/api`

API сервер на базе фреймворка FastAPI.

Содержит функционал, связанный с endpoint-ами которые доступны клиентам для обращения к ним по протоколу http для взаимодействия с объектной моделью Predicate.

bin

Путь: `predicate/predicate/bin`

Подмодуль содержит описание команд для CLI (command line interface) приложения.
Entry Point приложения Predicate.

- `bin/start/start.py` - команды инициализации (запуска API и отрисовки интерфейса Swagger)
- `bin/start/celery` - команды запуска сайдкаров, связанных с фреймворком celery
- `bin/database/alembic/versions` - миграции, описывают изменение состояния базы данных
- `bin/execution` - код, связанный с исполнением проектов мониторинга (формирование и запуск DAG Airflow)

celery

Путь: `predicate/predicate/celery`

Подмодуль для управления фоновыми задачами.

core

Путь: `predicate/predicate/core`

Подмодуль для конфигурации приложения и оберток для внешних сервисов.

crud

Путь: `predicate/predicate/crud`

Подмодуль содержит классы, в которых определяются типовые методы взаимодействия с базой данных. Служит прослойкой между endpoint-ами API и ORM (object relational mapping).

models

Путь: `predicate/predicate/models`

Подмодуль содержит реализацию ORM классов базы данных.

ORM - object relational mapping - интерфейс, который программным сущностям сопоставляет сущности из базы данных. Для реализации ORM-подхода в Predicate используется библиотека SqlAlchemy.

- `models/tables/base` - описание классов объектов
- `models/tables` - здесь в ру-файлах хранятся наследуемые классы

schemas

Путь: `predicate/predicate/schemas`

См. также: `klmg_predicate_types/api`

Схемы - это классы, которые выполняют функции определния структуры данных и проверки входящих данных на сотвествие этой структуре.

Схемы используются для методов API, где нужно передавать или возвращать данные, соответсвующие объектам базы данных.

ТИПЫ СХЕМ

- **Response schemas** - регламентируют формат, в котором объекты базы возвращаются при ответе методов API. Данные схемы не содержат дополнительных кастомных проверок.
- **Request schemas** - регламентируют формат входных данных POST методов API для создания объектов и PATCH методов для обновления объектов.

СХЕМЫ PREDICATE

- `predicate/predicate/schemas` - request schemas для POST и PATCH методов API. Расширение описаний объектов, которые проходят через endpoints. К классам объектов добавляются методы, которые создают (POST) или обновляют (PATCH) записи в базе.

Остальные схемы, используемые в приложении, определяются в модуле `api` библиотеки klmg-predicate-types.

utility

Путь: `predicate/predicate/utility`

Вспомогательные функции, которые переиспользуются в нескольких блоках backend-части приложения.

4.5.3 packages

fastapi-utils

Путь: `predicate/packages/fastapi-utils`

Библиотека вспомогательных инструментов для FastAPI.

git-share

Путь: `predicate/packages/git-share`

Библиотека стандартных метрик, входящих в дефолтную поставку приложения Predicate.

git-utils

Путь: `predicate/packages/git-utils`

Интерфейс для работы с удаленными git-репозиториями.

klmg-predicate-types

Путь: `predicate/packages/klmg-predicate-types`

Библиотека типов для Predicate.

- `klmg_predicate_types/api` - request и response схемы для методов API
- `klmg_predicate_types/metric/param` - типы для парсинга и валидации значений параметров метрик Predicate
- `klmg_predicate_types/plotly` - контроллеры для отрисовки графиков метрик

4.5.4 plugins

Путь: `predicate/plugins`

`predicate/plugins/project-subscription` - плагин для уведомлений по результатам работы светофоров проектов мониторинга

4.5.5 Unit-тесты

Путь: `predicate/tests`

4.6 Web-интерфейс

Web-интерфейс приложения Predicate разработан на языке JavaScript с использованием фреймворка React и дизайн-системы Ant.