|  |  |  |
| --- | --- | --- |
| logo |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
| **СКД -Система контроля данных**  Техническое задание  на разработку программного обеспечения |

|  |  |
| --- | --- |
| Разработчики | Андрей Телятник |
| Организация | АО "АТС", Россия, Москва, Краснопресненская набережная, д. 12, подъезд 7 |
| **Москва 2017** | |

[1 История изменений 3](#_Toc502264217)

[2 Краткая характеристика системы 3](#_Toc502264218)

[2.1 Общая информация 3](#_Toc502264219)

[2.1.1 Виды проверок 3](#_Toc502264220)

[2.2 Параметры системы 4](#_Toc502264221)

[3 Функциональные требования 4](#_Toc502264222)

[3.1.1 Считывание программного кода проверок из репозитория 4](#_Toc502264223)

[3.1.2 Получение заданий на выполнение проверки 5](#_Toc502264224)

[3.1.3 Выполнение проверок 6](#_Toc502264225)

[3.1.3.1 SQL запросы 6](#_Toc502264226)

[3.1.3.2 YML запросы 7](#_Toc502264227)

[3.1.3.3 Python-скрипты 7](#_Toc502264228)

[3.1.4 API получения результатов проверок 9](#_Toc502264229)

[3.1.4.1 API endpoint «последние выполненные проверки» 9](#_Toc502264230)

[3.1.4.2 Пример реализации API 9](#_Toc502264231)

[3.1.5 Web-интерфейс просмотра результатов проверок 9](#_Toc502264232)

[3.1.5.1 Получение результатов проверок 10](#_Toc502264233)

[3.1.5.2 Отображение списка проверок 10](#_Toc502264234)

[3.1.5.3 Фильтрация списка 11](#_Toc502264235)

[3.1.5.4 Выбор отчета для ознакомления 11](#_Toc502264236)

[3.1.5.5 Ознакомление с отчетом 11](#_Toc502264237)

[4 Приложение 1 11](#_Toc502264238)

# История изменений

# Краткая характеристика системы

Система Контроля данных (далее СКД) является инфраструктурной информационной системой в АО «Администратор Торговой Системы» и предназначена для снижения вероятности возникновения ошибок в расчетных системах.

В СКД заводятся проверки, которые проверяют или какой-то конретный этап расчета, или некоторые общие соотношения (например, баланс), которые должны выполняться по итогам расчета.

По итогам выполнения проверки:

* может быть получен результат «пройдено» / «ошибка»
* может быть сформирован отчет (excel файл) с данными, которые проверяет оператор расчетной системы.

Проверки выполняются после завершения расчета в боевой базе на реальных данных. Таким образом, эти проверки дополняют unit-тесты, которые выполняются над искусственными данными.

Система СКД по своему характеру должна работать независимо от расчетных систем.

Независимо от результата проверки, модуль проверок не вмешивается расчетную часть (не останавливает и не блокирует работу расчетных систем). Решение о корректности расчета всегда остается за оператором системы.

Основные задачи системы:

а) автоматическое выполнение пользовательских проверок после завершения операций в расчетных системах;

б) предоставление статусов и результатов проверок в интерфейс расчетной системы.

**Глоссарий**

**СКД, SKD -**  Система Контроля данных АО «АТС»

**ИС** – информационная система

## Общая информация

В системе предусмотрены следующие типы проверок:

* SQL запросы
* YML запросы (SQL запросы с метаданными)
* Python скрипты

Сигнал для запуска проверок приходит от ИС в виде пакета данных json. В пакете указывается:

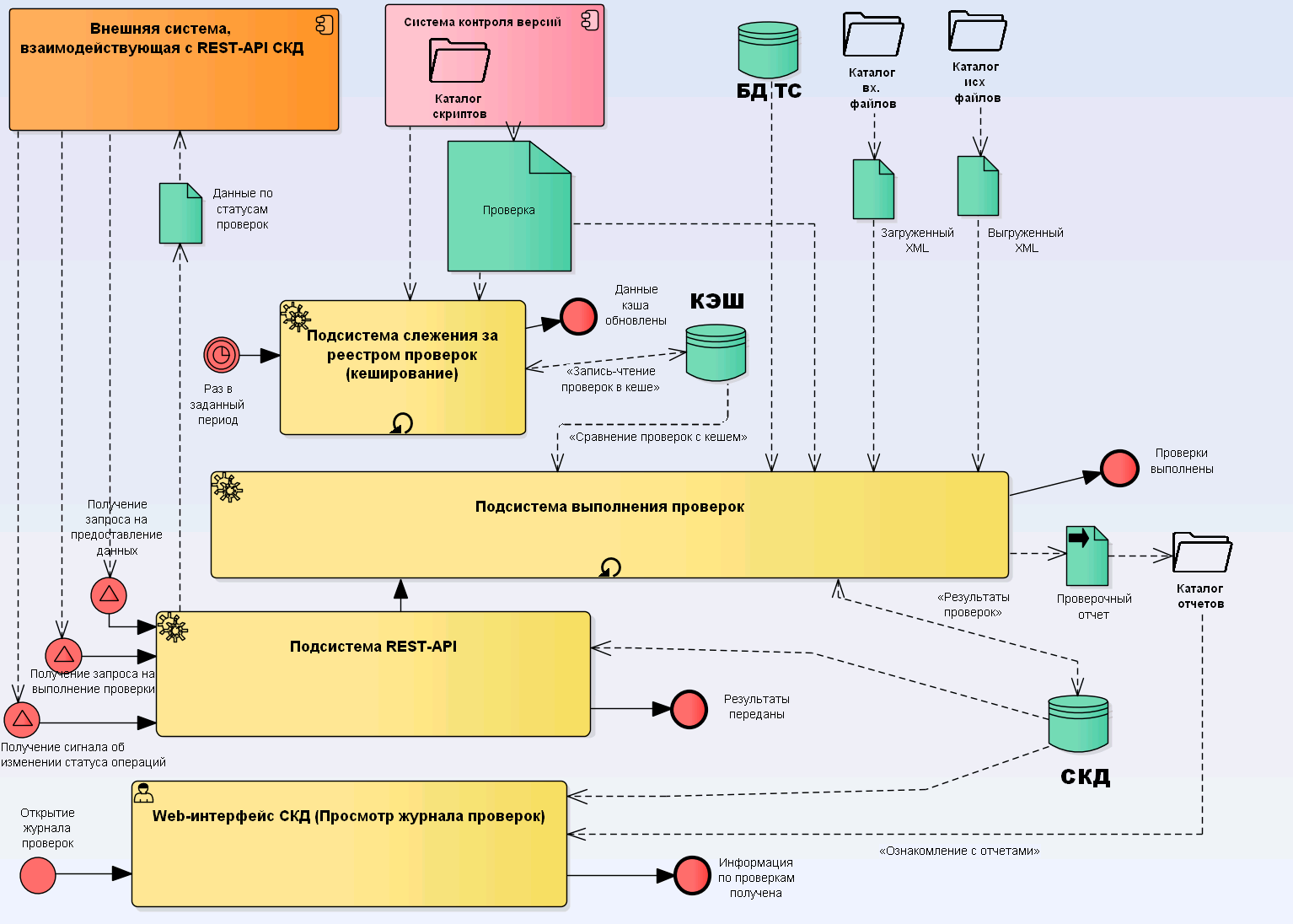
* параметры, идентифицирующие завершенную операцию, которую необходимо проверить (номер, код операции и код системы, идентификаторы зависимых операций)
* параметры подключения к БД
* прочие параметры, необходимые для выполнения проверки

Все проверки, где явно не указано иное, требуют ровно одно подключение к БД.

Проверки делятся на *контрольные* (имеют результат «пройдено» / «ошибка»), и *информационные* (только выгружают отчет с данными, которые проверяет оператор расчетной системы).

СКД предоставляет внешним системам информацию о статусе выполнения проверок и о результатах проверок.

Для работы СКД использует внутреннюю БД под управлением MongoDB.



## Параметры системы

|  |  |
| --- | --- |
| **Название** | **Тип данных** |
| Путь к репозиторию | *Словарь: код системы-> путь к репозиторию*  dict: string->string |
| Периодичность обновления кэша проверок | int (секунд) |
| Каталог для записи файлов с результатами проверок | string |
| Максимальное количество одновременных проверок, отправляемых на выполнение | int |

# Функциональные требования

**Основные функции**

* Считывание программного кода проверок из репозитория
* Получение заданий на выполнение проверки (от API графа операций или любой другой системы или пользователя);
* Выполнение проверок
* Предоставление внешним системами информации о статусе выполнения проверок и о результатах проверок
* Предоставление сводной информации о статусе выполнения проверок в web-интерфейсе

### Считывание программного кода проверок из репозитория

Проверки представляют собой файлы определенных типов (.sql, .yml, .py) по принципу «один файл – одна проверка» (подробнее см. п. 3.1.3).

В СКД проверки хранятся в следующей иерархии:

* 1 уровень «СистемаA»
  + 2 уровень «ОперацияА»
* проверка 1.sql
* проверка 2.yml
* проверка 2.py
  + 2 уровень «ОперацияB»
* проверка 1.sql
* проверка 2.yml
* проверка 2.py
* 1 уровень «СистемаB»
  + 2 уровень «ОперацияA»
* проверка 1.sql
  + 2 уровень «ОперацияB»
* проверка 1.py
  + 2 уровень «ОперацияC»
* проверка 42.yml
* 1 уровень «СистемаC»

Источником проверок для СКД являются git-репозитории, путь к которым следует указывать в конфигурационном файле СКД (см. 2.2). При этом первым уровнем вышеуказанной иерархии (система) будет являться наименование репозитория (ключ репозитория– код Системы), вторым (операции) – папки в корне репозитория (наименование – кодом Операции). Репозитории могут содержать файлы, не входящие в иерархию, они игнорируются.

Из схемы выше видно, что проверки разного типа могут иметь одно наименование в рамках одной Операции.

Таким образом, уникальным ключем (или путем) проверки является путь в иерархии: «код системы/код Операции/проверка.тип».

СКД синхронизирует master-ветки репозиториев в режиме онлайн (периодичность задается в настройках системы, см. п. 2.2) и помещает проверки в свою БД (на примере реализации на MongoDB в виде документов):

{

"system" : "TS",

"operation" : "\_AUX",

"name" : "current\_session",

"extension" : "py",

"key\_path" : "TS/\_AUX/current\_session.py",

"content" : "исходный код проверки"

}

### Получение заданий на выполнение проверки

СКД позволяет инициализировать проверки по конкретной операции (в рамках конкретной системы), либо по одной или нескольким проверкам в рамках конкретной операции. Таким образом, в задании на выполнение проверок должны содержаться:

* system: код системы;
* operation : код операции;
* массив имен требуемых проверок (пустой массив, если требуется выполнить все проверки операции).

Информация об источнике данных для проверок также передается в задании:

* информация о подключении к одному или двум стендам (в зависимости от выполняемых проверок). Возможно подключение к БД Oracle или Postgres с указанием соответствующего подкласса Connection и строки подключения (см. пример ниже).
* ops: словарь дополнительных переменных (bind variables).

Заданию должен быть присвоен некий ключ (атрибут key) произвольной формы, например, для идентификации задания в ряду других аналогичных заданий, но выполняемых на другом стенде.

Задание на проведение проверки приходит в виде POST запроса в формате JSON:

{

"system": "TSIII",

"operation": "CONSCALC",

"key": {

"smth": "some\_value",

"user": "konstantinov",

"stand\_name": "ts\_black"

},

"sources": [

{

"connection\_string": "facts/\*\*\*@ts\_black", //строка подключения к БД Oracle

"class\_name": "OracleConnection", // говорим, что соединяемся с БД Oracle

"ops": {

"CONSCALC": 1442,

"LOADPOWDEM": 56601,

"CALENDAR": 56401,

"LOADPREVDR": 62801

}

}, {

"connection\_string": "host=vm-facts-db port=5432 dbname=facts user=facts password=\*\*\*",

"class\_name": "PostgresConnection",

"ops": {

"CONSCALC": 1,

"LOADPOWDEM": 2,

"CALENDAR": 3,

"LOADPREVDR": 4

}

}

],

}

При получении задания СКД инициализирует выполнение требуемых проверок и сохраняет его в БД с фиксацией времени начала выполнения проверок и времени окончания выполнения последней проверки (переданные пароли к БД заменяются на «\*\*\*»).

### Выполнение проверок

Перед началом выполения проверки в БД фиксируется время запуска проверки, а также ей присваивается ключ, переданный в задании. По завершении фиксируется о временя окончания выполнения проверки, а также результат проверки (см. ниже) и путь к файлу с отчетом (см. ниже).

Результат проверки – в общем случае некоторое произвольное скалярное или JSON-подобное значение, сохраняемое в БД. Результат True/False интерпретируется соответственно как «пройдено» / «ошибка», при получении любого другого результата проверка считается информационной.

Отчет – xlsx или другой файл, формируемый при выполнении проверки, в БД сохраняется только путь к этому файлу.

В системе предусмотрены следующие типы проверок:

* SQL запросы
* YML запросы (SQL запросы с метаданными)
* Python скрипты

Для YML тип проверки задается в метаданных:

* + если <type> = LOGICAL, то проверка контрольная, результат выполнения определяется также, как для SQL проверок
  + если <type> = INFO или COMPARE, проверка информационная

#### SQL запросы

SQL запросы оформляются в виде .sql-файла, содержащего ровно один запрос. Использование PL/SQL не допускается. В запросе можно использовать привязку переменных (bind variables), передаваемых в задании для СКД. Подробнее о передаче параметров см. 3.1.2.

Результатом проверки является True (ошибка) в случае, если запрос вернул хотя бы одну строку, False (пройдено) в противном случае. Результат выполнения запроса сохраняется в xlsx-файл с наименованиями столбцов в первой строке (даже если результат не содержит ни одной строки, все равно выгружается файл с наименованиями столбцов).

Пример: **select \* from table  
 where field = :value**

#### YML запросы

YML запросы оформляются в виде YML файла, при этом обзательными полями являются <type>, принимающее одно из трех значений [LOGICAL|INFO|COMPARE], и <query>, содержащее SQL-запрос. Все проверки данного типа сохраняют результат выполнения запроса в xlsx-файл с наименованиями столбцов в первой строке.

<type> = LOGICAL (контрольная проверка) – результатом проверки является значение логического типа, True (ошибка) в случае, если запрос вернул хотя бы одну строку, False (пройдено) в противном случае (аналогично п.2.1.1)

<type> = INFO (информационная проверка) – результатом проверки является значение None. Т.е. полезным результатом такой проверки является результат выполнения SQL-запроса, который в дальнейшем просматривается пользователем ИС.

<type> = COMPARE – данному типу проверки требуются два подключения к БД. Проверка производит сравнение результатов выполнения запроса в обеих БД. Процедура сравнения выглядит так:

**select** 'a', q.\*

**from** <query>@FIRST\_DB q

minus

**select** 'a', q.\*

**from** <query>@SECOND\_DB q

**union** **all**

**select** 'b', q.\*

**from** <query>@SECOND\_DB q

minus

**select** 'b', q.\*

**from** <query>@FIRST\_DB q

Результатом проверки является количество записей данного запроса (кол-во отличий). Поскольку результат в данном случае не логическое значение, проверка информационная.

#### Python-скрипты

Python-скрипты представляют собой .py-файл, содержащий единственную функцию (или корутину) обернутую некоторыми декораторами, определяющими входные аргументы функции.

Допускается только одно имя функции – run\_check.

Возможные декораторы:

1. output\_file\_descriptor – принимает аргументы check, task, ext, bin.
   1. check и task объявляются вне функции, передаются всегда.
   2. ext – расширение выходного файла, необязательный аргумент. Если аргумент опущен, то расширение по умолчанию xlsx.
   3. bin – признак бинарного выходного файла, необязательный аргумент. Если аргумент опушен, то значение по умолчанию False. Сам файловый дескриптор – стандартный дескриптор Python, если run\_check – функция, или дескриптор модуля aiofiles, в случае если run\_check – корутина. Если в функции не используется данный декоратор – то используется неявное сохранение результатов.

Неявное сохранение результатов срабатывает в случае, если результат работы функции (корутины) представляет собой одно из следующих: <list[tuple]>, < list[tuple] и результат>, < результат и list[tuple]> (где list[tuple] – согласно терминологии Python, < результат > – логическое значение или None); в таком случае < list[tuple]> интерпретируется как данные для отчета, которые сохраняются в файл xlsx, <результат> интерпретируется как результат проверки (сохраняется в атрибут проверки result).

В ином случае xlsx-файл не формируется, а возвращаемое значение функции интерпретируется как результат проверки (сохраняется в атрибут проверки result).

1. single\_connection – принимает аргументы check и task, объявляемые вне функции. Возвращает экземпляр класса DBConnection и словарь переданных в задании переменных.
2. double\_connection – то же, но в двух экземплярах.

Шаблоны скрипта представлены ниже:

– шаблон проверки-корутины с одним соединением к БД и явной записью в файл

@output\_file\_descriptor(check, task, ext: str, bin: bool)  
@single\_connection(check, task)  
**async def** run\_check(file\_descriptor, connection, source):  
 *"""  
 Метаданные в формате YAML  
 """* **await** file.write(**'some information'**)  
 data = **await** connection.async\_exec\_script(**'''  
 select \* from table  
 where field = :value  
 '''**, value=source[**'value'**], get\_field\_names=**True**)  
 **for** row **in** data:  
 **await** file.write(**','**.join(row))  
 **return False**

– шаблон проверки с двумя соединениями к БД и неявным сохранением результатов

@double\_connection(check, task)  
**def** run\_check(connection1, source1, connection2, source2):  
 *"""  
 Метаданные в формате YAML  
 """* result = []  
 query1 = **'select id, code from table1 where field = :value'** query2 = **'''select sum(numbers) from table2  
 where id = :id and filter = :value'''  
 for** id, code **in** connection1.script\_cursor(query1, value=source1[**'value'**]):  
 [(data,)] = connection2.exec\_script(query2,  
 id=id,  
 value=source2[**'filter'**]  
 )  
 result.append((code, data))  
 **return** result, len(result) > 20

Для Python проверок могут задаваться метаданные, в виде YML документа, вставленного в docstring функции.

### API получения результатов проверок

Для получения данных из СКД предусматриваются четыре точки входа:

1. Шаблоны проверок
2. Задания
3. Проверки
4. Последние выполненные проверки

Первые три точки представляют собой по сути прямые каналы на коллекции или таблицы в БД. На п. 4 следует остановиться подробнее.

#### API endpoint «последние выполненные проверки»

Данная точка входа предусматривает использование техники «long polling». Ответ на запрос возвращается только в случае изменения его содержимого относительно последнего полученного клиентом ответа. Для этого предполагается использовать хэширование тела ответа, отправляемого пользователю. В последующих запросах клиент включает хэш ответа в тело запроса, что гарантирует ему получение ответа только в случае его изменения. Эта техника позволит мгновенно получать изменения в статусе проверок в режиме онлайн без необходимости бомбардирования сервера запросами со стороны клиента.

Если клиент опустит в запросе хэш предыдущего ответа, то СКД сразу же вернет текущее состояние ответа, а также его хэш.

#### Пример реализации API

Подробности конкретной реализации API сервера, использующего БД MongoDB см. в Приложении 1.

### Web-интерфейс просмотра результатов проверок



Система должна иметь web-интерфейс для отображения статуса и истории выполнения проверок.

Интерфейс представляет собой таблицу в которую выводится список всех выолняемых и выполненных проверок со следующими атрибутами:

* Название проверки
* Стенд
* Название операции (код операции?) *названия нет в задании*
* Статус выполнения
* Время старта и завершения, время выполнения
* Тип проверки
* Ссылку для скачивания отчета, если он был сформирован

В таблице должна быть доступна сортировка и фильтр по любой колонке.

# Приложение 1

В представленном примере API используется БД MongoDB. Имеется три коллекции:

* cache (шаблоны проверок)
* tasks (задания)
* checks (проверки)

К каждой коллекции предоставляется прямой доступ с использованием синтаксиса pymongo (имеется в виду особенность, касающаяся сортировки), во многом аналогичного синтаксису MongoDB:

|  |  |
| --- | --- |
| Запрос на языке MongoDB | POST запрос для API СКД на /rest/cache |
| db.cache  .find(<query>, <project>)  .sort(<sort>)  .skip(<skip>)  .limit(<limit>) | {"query": <query>,  "sort": <sort>,  "skip": <skip>,  "limit": <limit>,  "project": <project>  } |

Так покрываются №1-3 из п. 3.1.4. № 4 из вышеуказанного пункта покрывается endpoint-ом /rest/get\_last\_checks и имеет следующую сигнатуру:

{

"query": <query>,

"response\_hash": <hash\_value> // необязательное поле

}

При указанном непустом значении response\_hash срабатывает механизм long polling, ожидающий изменения ответа на запрос пользователя (путем сравнения hash-значения нового ответа с переданными в запросе). Иначе немедленно возвращается список последних проведенных проверок. В поле query обязательно должно присутствовать поле system.