

РУКОВОДСТВО РАЗРАБОТЧИКА ПРОЕКТОВ.

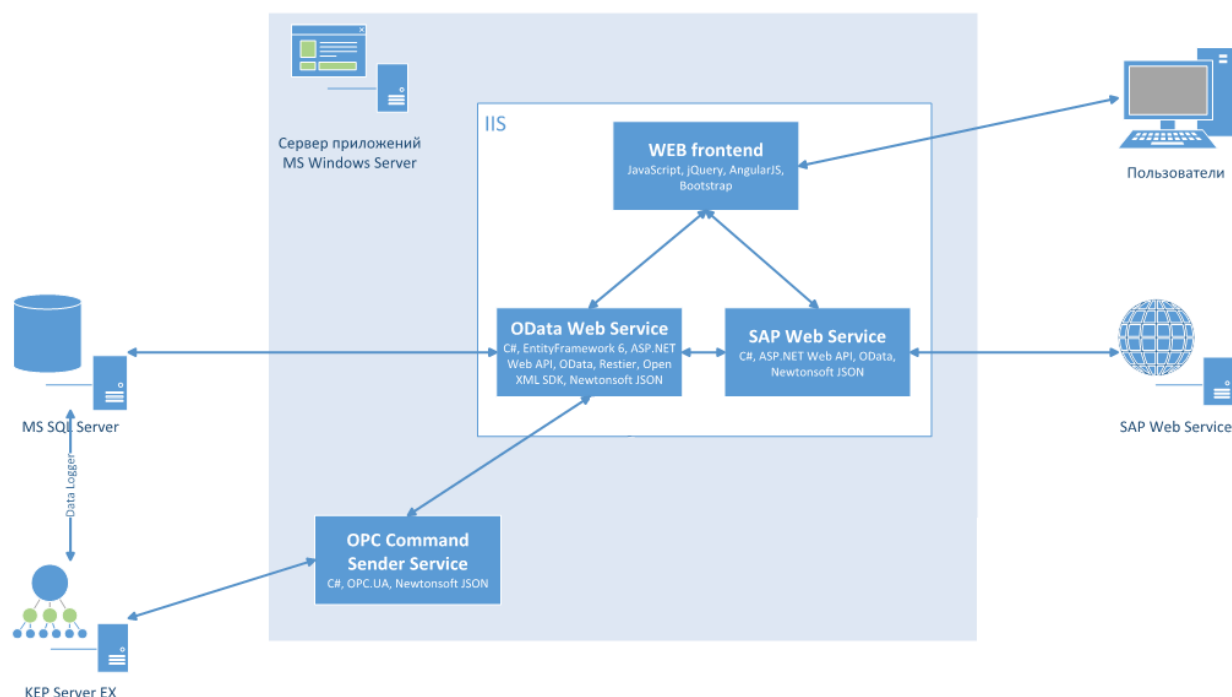
Содержание

1.	Общие положения.....	4
1.1.	Архитектура системы.	4
1.2.	Разделение БД на схемы.	4
1.3.	Структура проекта в GitHub	5
1.4.	Разработка нового проекта, выкладывание коммитов.	7
1.5.	Ветки DDL, DML, SDL в структуре isa95_database.	8
1.6.	Работа системы непрерывной интеграции Jenkins.	9
1.7.	Группы пользователей.	10
2.	База Данных	11
2.1.	Правила именования объектов.....	11
2.1.1.	Процедуры	11
2.1.1.1.	Процедуры вставки	11
2.1.1.2.	Процедуры редактирования	11
2.1.1.3.	Процедуры удаления	11
2.1.1.4.	Процедура вычитки	11
2.1.1.5.	Процедуры с output параметрами.....	11
2.1.2.	Функции.....	11
2.1.3.	Представления.....	11
2.1.4.	Sequence	11
2.2.	Типы данных	11
2.2.1.	Дата+время	11
2.2.2.	HierarchyID.....	12
2.3.	Работа с Primary Key полями	12
2.4.	Правила генерации пользовательских сообщение об ошибках	12
2.5.	Правила внесения изменений в структуры БД.....	13
2.5.1.	Правила именования скриптов	13
2.5.2.	Правила написания скриптов	13
2.5.3.	Применение скриптов.....	13
3.	Структура стандарта ISA-95.....	15
3.1.	Схема ISA95_MATERIAL.....	15
3.2.	Схема ISA95_EQUIPMENT.	17
3.2.1.	Ролевая модель оборудования.....	18
3.2.2.	Организация технического обслуживания оборудования.	19
3.2.3.	Диагностика оборудования.	20
3.3.	Схема ISA95_PERSONNEL.....	21
3.4.	Схема ISA95_OPERATION_DEFINITION	23
3.5.	Схема ISA95_WORK_DEFINITION.....	23

3.6.	Схема ISA95_WORKFLOW_SPECIFICATION.....	24
4.	Работа с веткой FrontEnd.	25
4.1.	Общая информация.....	25
4.2.	Добавление новой роли.	25
4.3.	Получение данных.....	26
4.4.	Запись/редактирование данных.....	27
4.5.	Локализация.	28
4.6.	Коммиты.....	28
5.	Структура View и процедур для Entity, используемых в диаграммах	30
5.1.	Entity для списка диаграмм	30
5.2.	Entity для списка нодов.....	30
5.3.	Entity для списка связей.....	31
6.	Настройка сайта	33

1. Общие положения.

1.1. Архитектура системы.



Архитектура системы представлена на рисунке и состоит из следующих элементов:

1. MS SQL Server, на котором развернута БД ISA95_PRODUCTION. При настройке прав доступа к данным используется встроенный инструмент MS SQL (роли MS SQL, группы ActiveDirectory).
2. Сервер приложений MS WindowsServer, на котором работают веб-сервисы и приложения, обеспечивающие основной функционал системы:
 - a. OData Web Service - веб-сервис, разработанный по REST (Representational state transfer – это стиль архитектуры программного обеспечения для распределенных систем) для организации единого интерфейса между клиентом и сервером, с поддержкой протокола Open Data Protocol (OData), версии 4.
 - b. SAP Web Service – обеспечивает обмен данными с SAP.
 - c. OPC Command Sender Service – Windows-сервис, обеспечивающий организацию взаимодействия с оборудованием нижнего уровня с использованием технологии OPC (например, отсылка команд контроллерам).
 - d. Web Frontend – интерфейс между пользователем и сервером.
3. OPC Server (KEPServerEX) – унифицированное средство сбора данных с первичных устройств по интерфейсу OPC и записи их в БД.

1.2. Разделение БД на схемы.

Для БД KRR-PA-ISA95_PRODUCTION созданы следующие схемы по умолчанию:

ISA95_MATERIAL (ANSI/ISA-95.00.02-2010 п.5.4),
ISA95_PHYSICAL_ASSET (ANSI/ISA-95.00.02-2010 п.5.3),

ISA95_EQUIPMENT (ANSI/ISA-95.00.02-2010 п.5.2),
ISA95_PERSONNEL (ANSI/ISA-95.00.02-2010 п.5.1),
ISA95_OPERATION_DEFINITION (ANSI/ISA-95.00.02-2010 п.6.1),
ISA95_WORK_DEFINITION (ANSI/ISA-95.00.04-2012 п.7),
ISA95_OPERATION_SCHEDULE (ANSI/ISA-95.00.02-2010 п.6.2),
ISA95_WORK_SCHEDULE (ANSI/ISA-95.00.04-2012 п.8),
ISA95_OPERATION_PERFORMANCE (ANSI/ISA-95.00.02-2010 п.6.3),
ISA95_WORK_PERFORMANCE (ANSI/ISA-95.00.04-2012 п.9).

ISA95EXT_PACKAGING,
ISA95EXT_LOCATION – схемы, расширяющие существующий стандарт;

Кроме указанных схем создаются схемы под каждый новый проект, к примеру:

AMKR_ROLLING_LABEL_PRINT,
AMKR_WEIGHING,
AMKR_OXYGENACCOUNTING,
AMKR_TRANSPORT,

Схемы реализующие другие стандарты – и их пересечение с ISA95:
AML (<https://www.automationml.org/>),
MIMOSA (<http://www.mimosa.org/>).

1.3. Структура проекта в GitHub

В GitHub организована следующая структура для разрабатываемых проектов.

Структура веток:

- Isa95_database
- Odata_unified_frontend
- Print_service
- Opc_isa_service
- Documentation
- Odata_unified_svc
- Odata_sap_svc

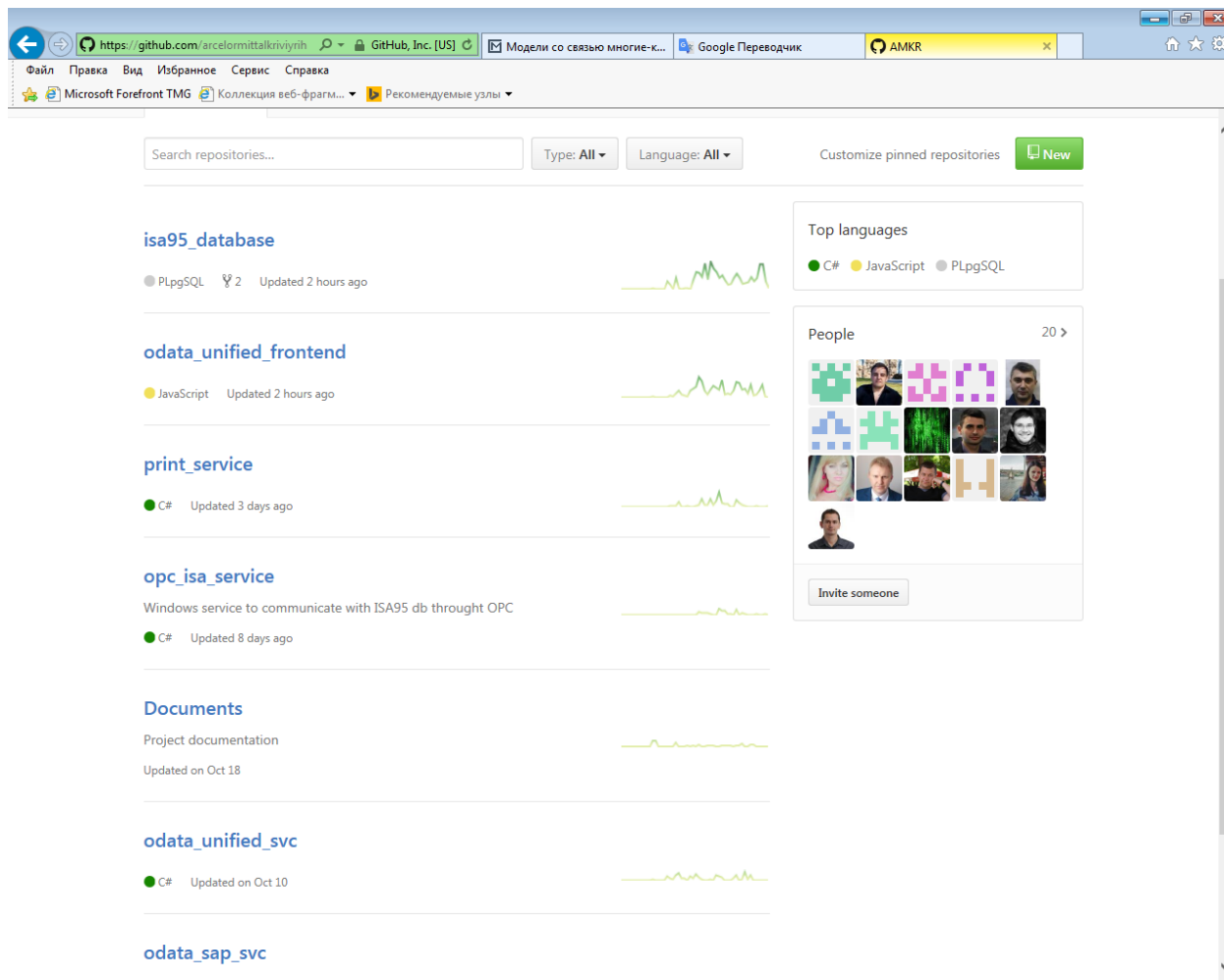


Рис.№2. Общая структура GitHub.

В случае начала разработки нового проекта в структуре БД [KRR-PA-ISA95_PRODUCTION] будет создана новая схема согласно названию проекта и в ветках `Isa95_database` и `Odata_unified_frontend` в GitHub будут добавлены дополнительные ветки, совпадающие по названию со схемой БД.

К примеру, для проекта «АСУ Кислород» создана схема `AMKR_OXYGENACCOUNTING` и ветки `Isa95_database\OxygenAccounting`; `Odata_unified_frontend\OxygenAccounting`; в GitHub. Рис.№2.

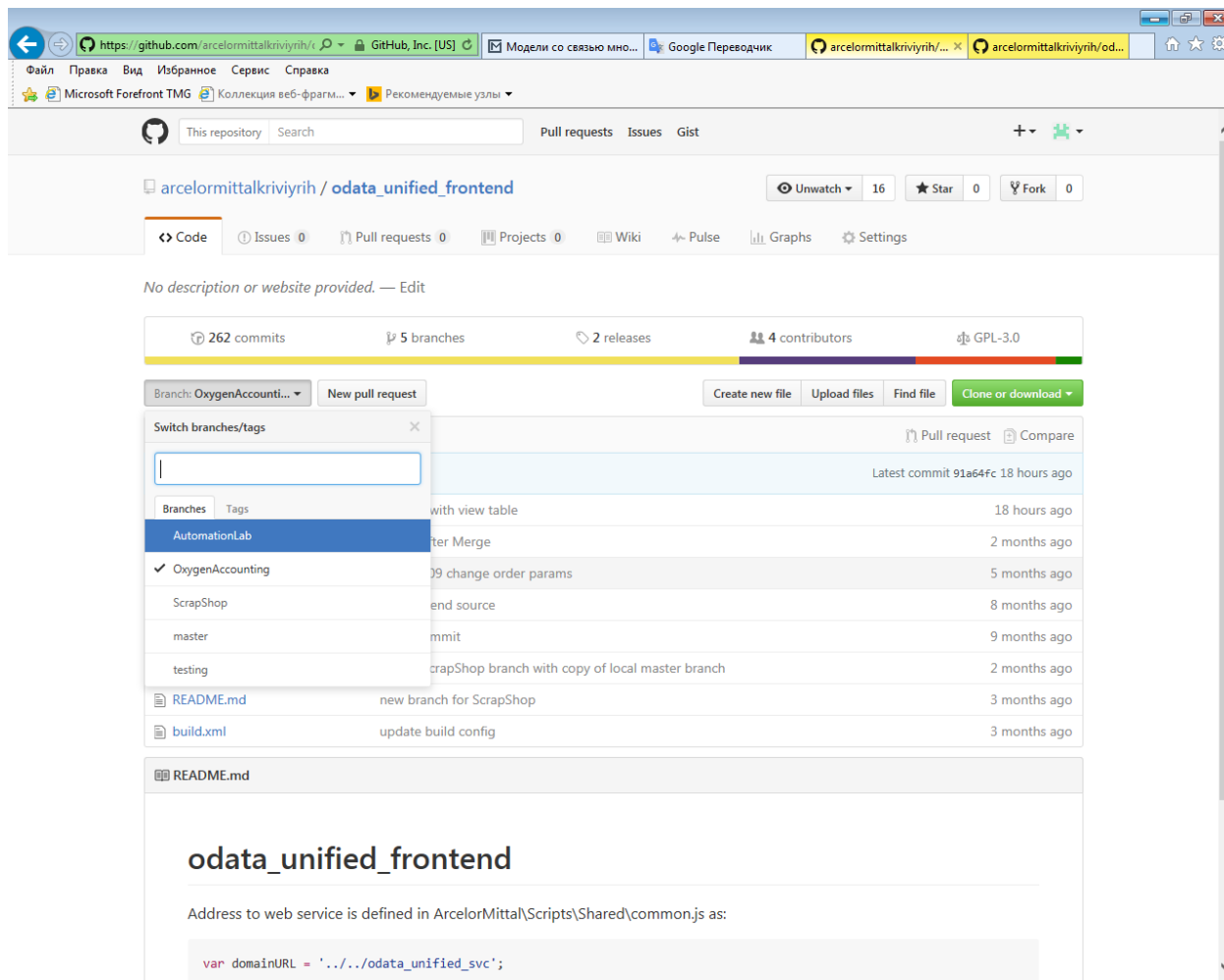


Рис.№2. Разделение проектов по веткам.

1.4. Разработка нового проекта, выкладывание коммитов.

Для разработки проекта необходимо в MSSQL тестового сервера создать backup БД и развернуть с новым названием на тестовом сервере.

Разработанные успешные решения необходимо выкладывать в ветку **testing** для проведения полного тестирования, - после чего, ветки таких проектов будут сливаться в ветку **master** по вашему запросу ответственным тестировщиком и выкладываться на продуктив. Итог – в ветку **master** попадают проекты, успешно оттестированные. И только ветка мастер едет в продуктив.

Также в GitHub необходимо использовать метки (tag), для отделения значимых этапов проекта (релиз, начало тестирование, конец тестирования и т.д.)

Предупреждения:

Коммитить в ветку мастер запрещено, данные коммиты будут удаляться ответственным тестировщиком.

При тестировании проектов необходимо учитывать, что еженедельно по воскресениям в 8:30 инициируется разворачивание BackUp с продуктивного сервера и при этом возможна потеря некоторых данных сформированных при проведении тестирования таких как данные полученные с датчиков, данные ручного ввода.

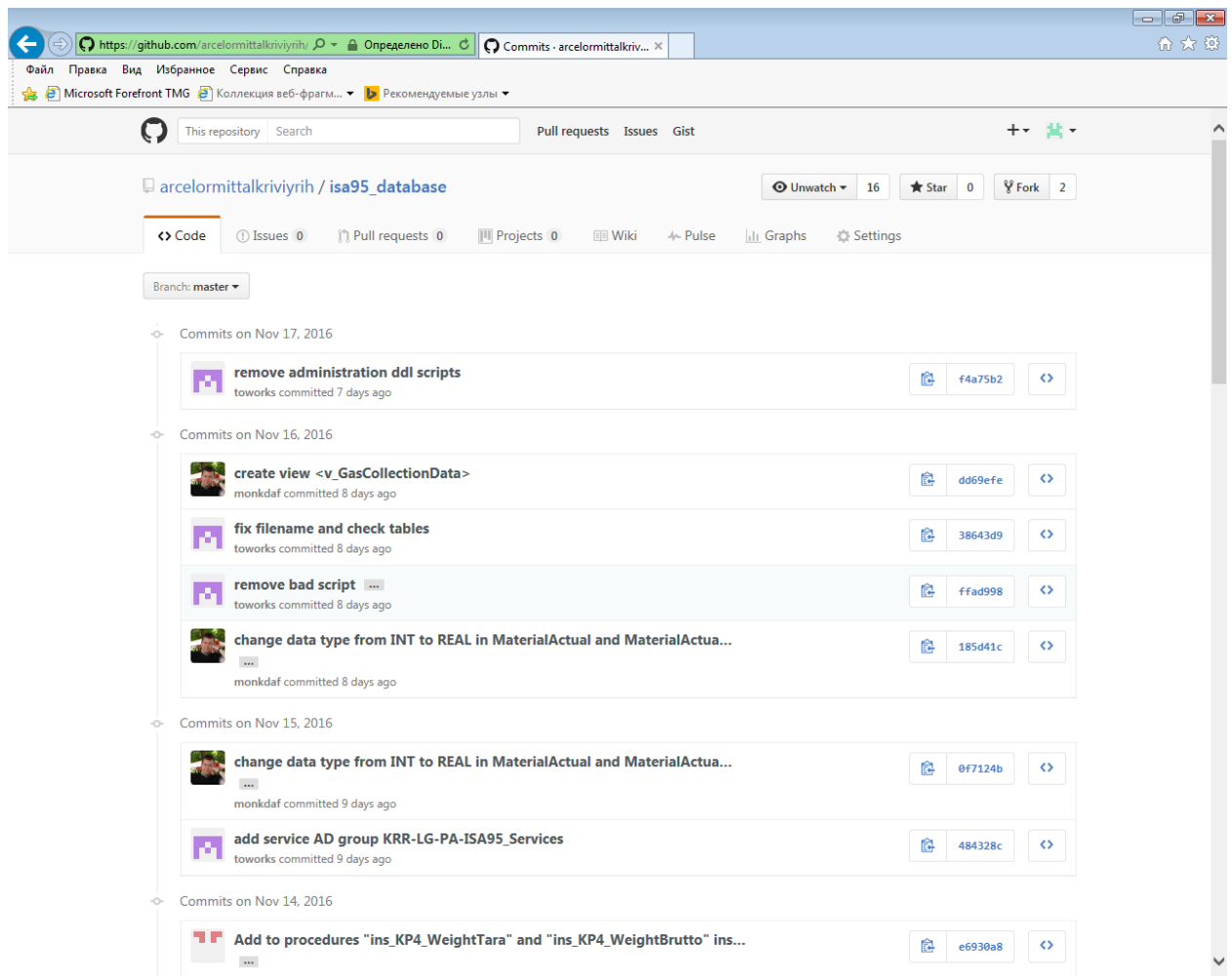


Рис.№3. Просмотр выполненных коммитов.

1.5. Ветки DDL, DML, SDL в структуре isa95_database.

В структуре ветки isa95_database/service_packs/

Созданы следующие ветки:

DDL - для коммитов выполняющих инструкции по модификации структуры данных CREATE, ALTER и т.д.

DML - для коммитов выполняющих инструкции INSERT, UPDATE, DELETE, MERGE.

SDL – для коммитов выполняющих инструкцию SELECT.

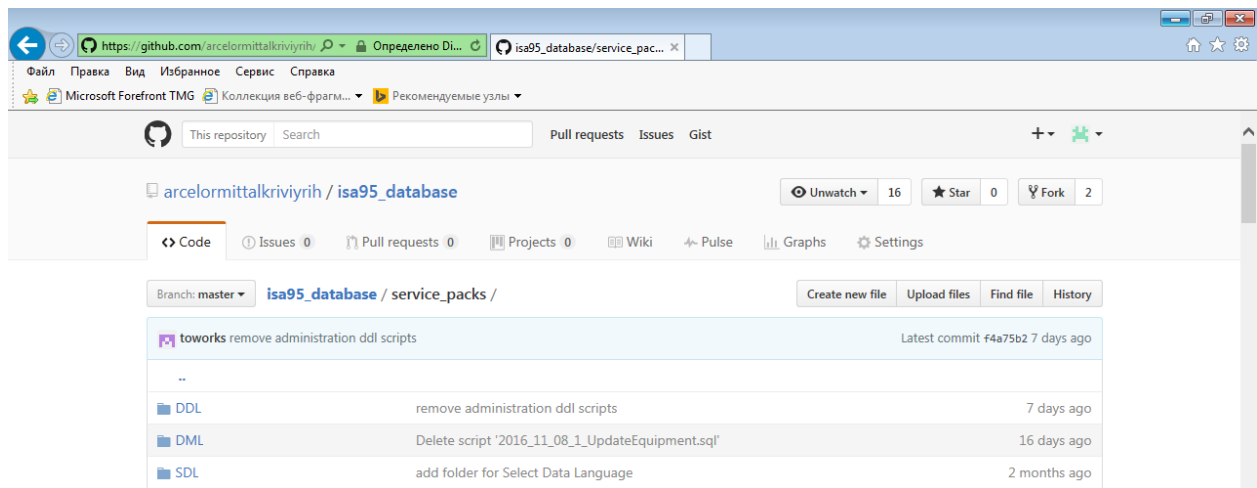


Рис. №4. Ветки DDL, DML, SDL.

1.6. Работа системы непрерывной интеграции Jenkins.

Ветка **krr-app-palbp01** (для коммитов в продуктивную среду) запускаются вручную и настроена на ветку **master**

Ветка **krr-tst-palbp01** (для коммитов в тестовую среду) запускаются автоматически каждый рабочий день в 8:30 и настроена на ветку **testing**

Ветка **isa95_database_backup krr-tst-palbp01** разворачивания BackUp настроена на запуск по воскресеньям в 8:30 с последующим перезапуском всех проектов.

Страница сгенерирована: 14.12.2016 10:58:33 EET [REST API](#) [Jenkins ver. 2.0](#)

Рис. №5. Структура Jenkins.

1.7. Группы пользователей.

Архитектура приложений использующих в своей основе принципы стандарта ИСА 95 должна быть построена с применением ролей MSSQL.

В рамках одного проекта необходимо более четко разделить пользователей по группам (например, операторы / тестировщики / разработчики и т.д.) и дать разрешение всем этим группам на одну роль в таблице KPPRoles БД [KRR-PA-ISA95_PRODUCTION] к примеру роль Analytics.

2. База Данных

2.1. Правила именования объектов

2.1.1. Процедуры

2.1.1.1. Процедуры вставки

Процедуры вставки данных начинаются с префикса «ins_» далее имя таблицы (например ins_Equipment);

2.1.1.2. Процедуры редактирования

Процедуры редактирования данных начинаются с префикса «upd_» далее имя таблицы (например upd_Equipment);

2.1.1.3. Процедуры удаления

Процедуры удаления данных начинаются с префикса «del_» далее имя таблицы (например del_Equipment);

2.1.1.4. Процедура вычитки

Процедуры вычитки данных начинаются с префикса «get_» далее имя таблицы (например get_Equipment);

2.1.1.5. Процедуры с output параметрами

Если процедура имеет output параметр, то имя параметра должно заканчиваться на Out.

2.1.2. Функции

Имя функций начинаются с префикса «get_»

2.1.3. Представления

Представления начинаются с префикса «v_» далее имя основной таблицы;

MS Entity Framework ставит требование что бы в таблицах и вью всегда был PK.

Поскольку в вью такое не возможно, то принимаем такое правило:

- первое поле всегда это наш PK.
- имя поля должно быть ID
- если по логике запроса первое поле уникально, то и все хорошо.
- если по логике вью первое поле не уникально, то нужно создавать фиктивный PK, использовать newID() для заполнения этого фиктивного PK.

2.1.4. Sequence

Sequence начинаются с префикса «gen_» далее имя таблицы для которой он предназначен;

2.2. Типы данных

2.2.1. Дата+время

Для полей дата+время использовать тип datetimeoffset, а не datetime, поскольку RESTier не отличает datetime от date.

Если использование типа datetimeoffset невозможно (например KEP Server Data Logger не поддерживает тип datetimeoffset), то можно использовать тип datetime, но для вычитки данных с таблицы с этим полем, необходимо таблицу оформить в виде представления где поле с datetime преобразовать в datetimeoffset следующим образом:

```
ToDateTimeOffset([datetime_field],0) as datetime_field
```

2.2.2. HierarchyID

Нельзя использовать тип HierarchyID, поскольку он не поддерживается MS Entity Framework. Для построения иерархии используется классический метод с ParentID и связью таблицы самой на себя. Для ParentID используется поле с именем, совпадающим с именем таблицы. Например, для таблицы Equipment поле, содержащее идентификатор родительской строки, также называется Equipment

	ID	Description	EquipmentLevel	Equipment
1	100	АрселорМиттал Кривой Рог	Enterprise	NULL
2	101	Металлургическое производство	Site	100
3	102	Коксохимическое производство	Site	100
4	103	Вспомогательное производство	Site	100
5	104	Агломерационный департамент	Area	101
6	105	Аглоцех-1	Area	104
7	106	Аглоцех-2	Area	104
8	107	Аглоцех-3	Area	104
9	108	Доменный цех-1	Area	104
10	109	Доменный цех-2	Area	104
11	110	Аглоцех МП	Area	104
12	111	Сталеплавильный департамент	Area	101

2.3. Работа с Primary Key полями

В БД KRR-PA-ISA95_PRODUCTION для таблиц создан уникальный номер (sequence) в качестве первичного ключа. Поэтому во всех разрабатываемых приложениях необходимо планировать использование Sequence при Insert, т.е. категорически запрещается указывать ID при вставке данных.

Если ключевое поле таблицы имеет тип int, то его рекомендуется заполнять, используя Sequence или Identity. При этом обязательно что бы Sequence был прописан как DEFAULT VALUE для поля :

Identity: ID **INTEGER IDENTITY** (1,1) **PRIMARY KEY**

Sequence: **ALTER TABLE** [dbo].[TableName] **ADD DEFAULT** (NEXT VALUE FOR [dbo].[gen_SequenceName]) **FOR** [ID]

2.4. Правила генерации пользовательских сообщений об ошибках

Для генерации пользовательского сообщения об ошибке необходимо использовать метод THROW. Использовать RAISERROR запрещено (обладает некоторыми недостатками: невозможность сослаться на точную строку возникновения ошибки, не может повторно инициировать исключение).

Пример простого статического сообщения:

```
THROW 60001, N'LENGTH param required', 1;
```

Сообщение с параметрами:

```
SET @err_message = N'Заказ [' + CAST(@COMM_ORDER AS NVARCHAR) + N'] уже существует';
THROW 60010, @err_message, 1;
```

2.5. Правила внесения изменений в структуры БД

Все изменения в структуре БД должны быть оформлены в виде SQL скриптов. Каждый такой скрипт должен содержать необходимый и достаточный набор кода для внесения изменений. Если изменение структуры требует изменение данных, то скрипт должен содержать и код по модификации этих данных.

Все скрипты должны сохраняться в папке `service_packs` в репозитории https://github.com/arcelormittalkriviyrh/isa95_database

Порядок выполнения скриптов – согласно сортировки по имени файла.

Примечание: Файлы скриптов должны быть в кодировке UTF-8.

2.5.1. Правила именования скриптов

`<year>_<month>_<day>_<n>_<short_name>.sql`

Где:

`year` – год создания скрипта

`month` – месяц создания скрипта

`day` – день создания скрипта

`n` – номер скрипта за день

`short_name` – короткое имя скрипта, кратко описывающее суть изменения

Имя скрипта не должно содержать пробелов.

Например: `2016_04_10_1_add_equipement_table.sql`

2.5.2. Правила написания скриптов

- Каждый скрипт должен заканчиваться вызовом `GO`
- В начале скрипта необходимо указать его автора и краткие комментарии по его выполнению
- Если скрипт вставляет данные, то перед вставкой необходимо выполнять проверку наличия этих данных. Скрипт может быть вызван несколько раз (например, в скрипте была ошибка), поэтому во избежание проблемы с дубликатами данных, необходимо проверять что таких данных нет в таблице
- Если скрипт изменяет структуры, то перед этим необходимо проверять что эти изменения уже не были выполнены, например, проверять что добавляемое поле в таблицу уже еще не существует.

2.5.3. Применение скриптов

Скрипты, сохраненные в репозитории, автоматически выполняются системой Jenkins при вызове команды на сборку новой версии.

Успешно примененные скрипты регистрируются в таблице `ServicePacksFiles`.

Если скрипт не выполнен успешно, то ошибки его выполнения необходимо смотреть в логах сборки соответствующего билда в Jenkins

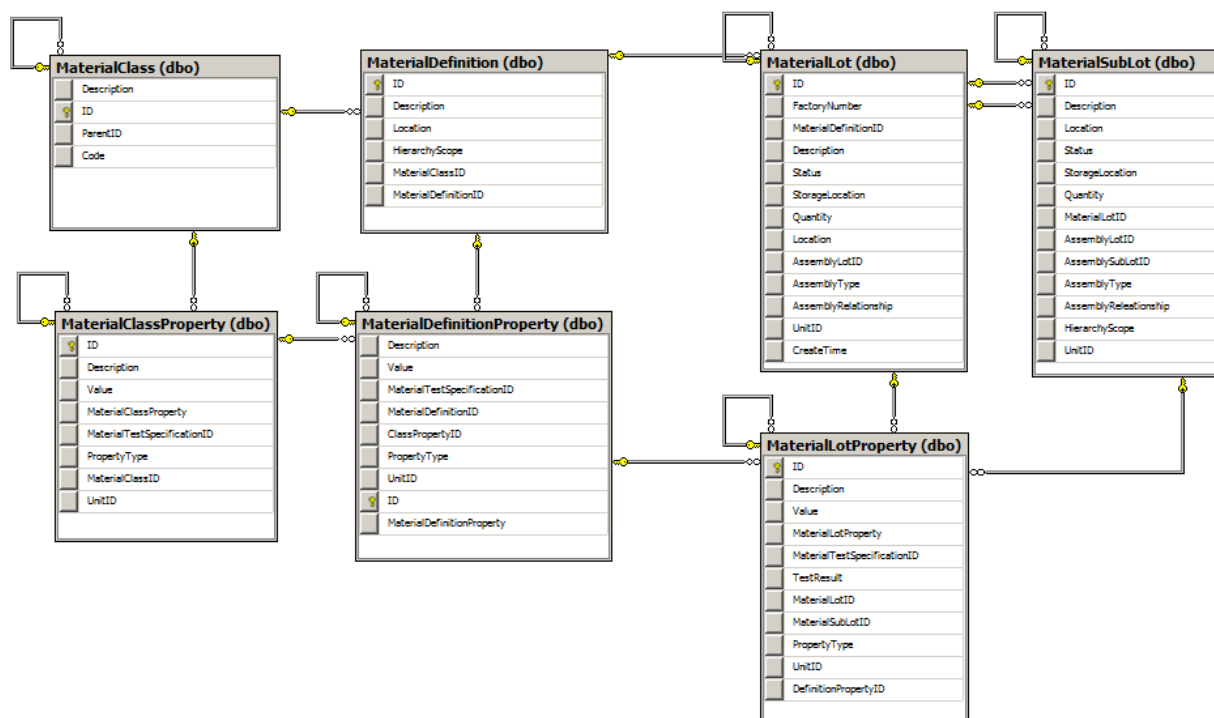
После исправления скрипта, его необходимо обновить в репозитории и вызвать сборку новой версии повторно.

Внимание: Если скрипт уже был успешно выполнен, то его запрещено изменять, поскольку эти изменения не будут применены при следующей сборке.

3. Структура стандарта ISA-95.

3.1. Схема ISA95_ MATERIAL

Схема используется для хранения данных о материалах, которые необходимы или имеются в наличии.



MaterialClass – таблица, содержащая классы материалов:

	ID	ParentID	Code	Description
1	1	NULL	PROFILE	Профиль
2	2	NULL	BINDING	Увязка
3	3	NULL	STEEL GRADE	Марка стали
4	4	NULL	LIQUID STEEL	Жидкая сталь
5	5	NULL	INGOT	Заготовка BL
6	6	NULL	BILLET	Заготовка МНПЗ
7	7	NULL	L	Длина пореза
8	8	NULL	Order Importance	Коеф.важности заказа
9	9	NULL	Scrap	Лом

MaterialClassProperty – таблица со свойствами классов материалов:

	ID	Description	Value	MaterialClassProperty	PropertyType	MaterialClassID	UnitID
1	1	Погонная масса [кг/м]	MA	NULL	NULL	1	NULL
2	2	Длина [м]	LE	NULL	NULL	1	NULL
3	3	Допуск плюс [%]	MP	NULL	NULL	1	NULL
4	4	Допуск минус [%]	MM	NULL	NULL	1	NULL
5	9	Диаметр увязки	BINDING_DIA	NULL	NULL	2	NULL
6	10	Количество увязок	BINDING_QTY	NULL	NULL	2	NULL
7	11	Коеф. учета веса увязок	BINDING_WEIGHT_COEF	NULL	NULL	2	NULL
8	12	Вид лома	Scrap type	NULL	NULL	9	NULL

Material Definition – таблица, содержащая определения материалов:

	ID	Description	Location	HierarchyScope	MaterialClassID	MaterialDefinitionID
1	1	Арматура 8	NULL	NULL	1	NULL
2	2	Арматура 10	NULL	NULL	1	NULL
3	3	Арматура 12	NULL	NULL	1	NULL
4	4	Арматура 14	NULL	NULL	1	NULL
5	69	Увязка 6,5/9	NULL	NULL	2	NULL
6	70	Увязка 7/9	NULL	NULL	2	NULL
7	71	Увязка 7,5/9	NULL	NULL	2	NULL
8	72	Увязка 8/9	NULL	NULL	2	NULL
9	73	Увязка 8,5/9	NULL	NULL	2	NULL
10	74	Увязка 6,5/8	NULL	NULL	2	NULL

MatetialDefinitionProperty:

	ID	Description	Value	MaterialDefinitionID	ClassPropertyID	PropertyType	UnitID	MaterialDefinitionProperty
1	27	NULL	0.395	1	1	NULL	NULL	NULL
2	28	NULL	12	1	2	NULL	NULL	NULL
3	29	NULL	8	1	3	NULL	NULL	NULL
4	30	NULL	8	1	4	NULL	NULL	NULL
5	31	NULL	0.617	2	1	NULL	NULL	NULL
6	32	NULL	12	2	2	NULL	NULL	NULL
7	33	NULL	5	2	3	NULL	NULL	NULL
8	34	NULL	5	2	4	NULL	NULL	NULL

Таблица MaterialDefinition и MaterialDefinitionProperty содержат определение материала и его свойства. Примером такого материала может служить 'Марка Стали' для выплавки с ее специфическими свойствами.

Таблица MaterialLot содержит информацию о партии материала.

Таблица MaterialSubLot содержит информацию о подпартии материала.

Таблицы MaterialActual и MaterialActualProperty содержат информацию о весе брутто, нетто материала, о фактических свойствах данного материала.

3.2. Схема ISA95_EQUIPMENT.

Схема содержит информацию о конкретных единицах оборудования и его классах.

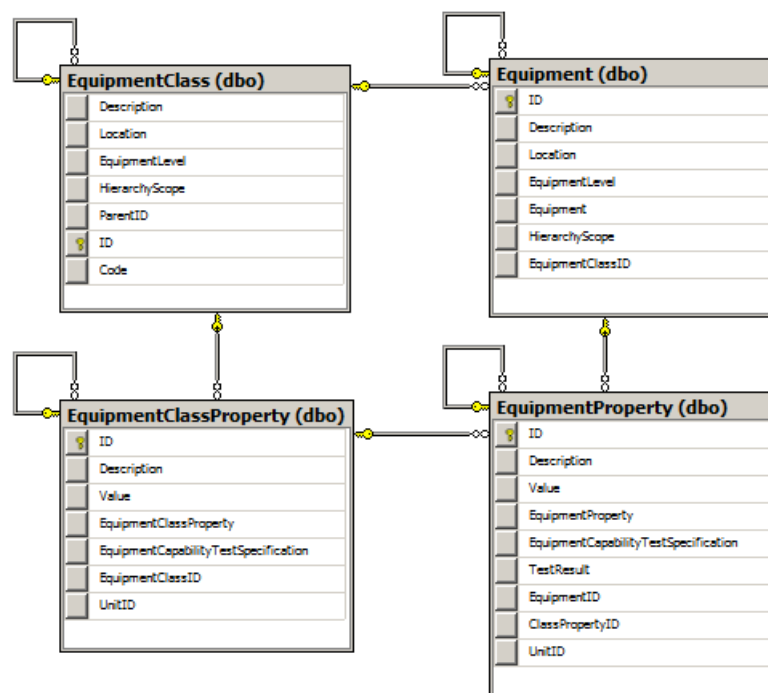
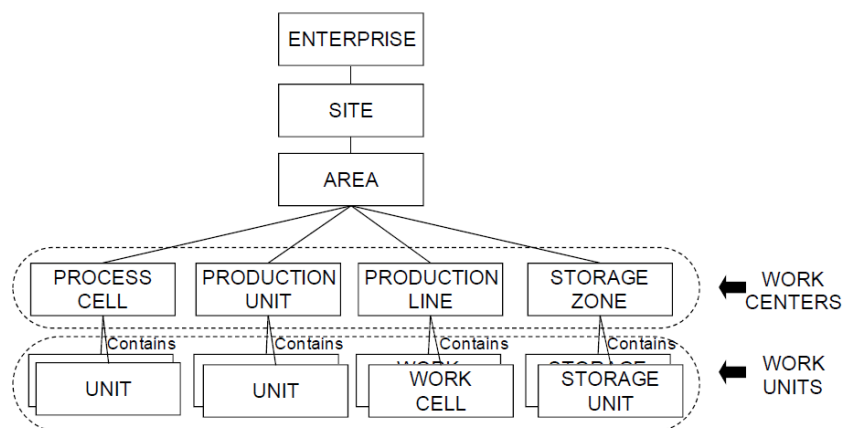


Таблица 'EquipmentClass' содержит определение класса оборудования и свойства оборудования.

Таблица 'Equipment' используется для хранения данных об оборудовании.

3.2.1. Ролевая модель оборудования.

Базовые активы в производственном процессе обычно представляются в иерархической форме:



Предприятие(ENTERPRISE) – это комплекс, который состоит из одной или нескольких производственных площадок и может содержать в себе производственные участки.

Производственная площадка (SITE) – определенная предприятием группа объектов, объединенная по физическому, территориальному или логическому принципу. Может включать производственные участки, технологические линии, производственные модули и агрегаты. Производственную площадку обычно идентифицируют по ее географическому положению и основным производственным возможностям.

Производственный участок (AREA) – физическая, территориальная или логическая группа объектов, определяемая в рамках производственной площадки. Сюда могут входить гибкие производственные модули, производственные агрегаты и технологические линии.

Производственный участок состоит из одного или нескольких низкоуровневых элементов, обеспечивающих выполнение функций производства.

Типы рабочих мест (WORK CENTERS) – технологические модули (PROCESS CELL), единичное производственное оборудование (PRODUCTION UNIT), производственные линии (PRODUCTION LINE), или зоны хранения (STORAGE ZONE).

Пример построения модели иерархии и разделение согласно EquipmentLevel:

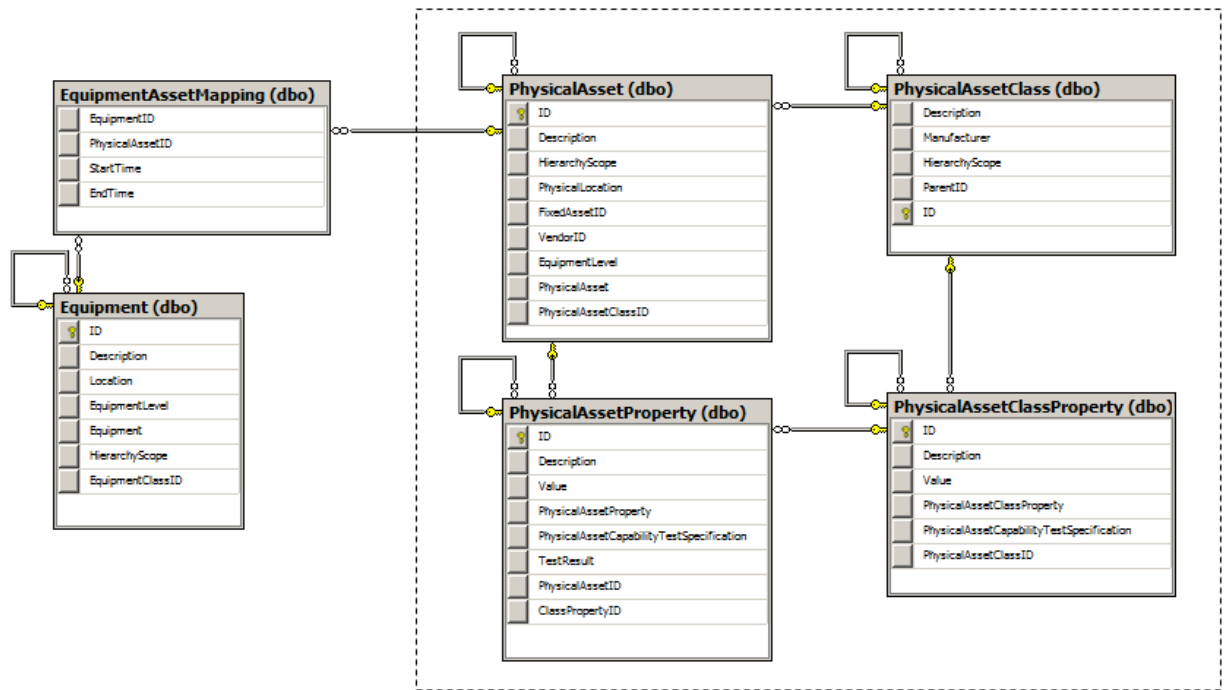
ID	Description	Location	EquipmentLevel	Equipment	HierarchyScope	EquipmentClassID
1	100 АрселорМиттал Кривой Рог	NULL	Enterprise	NULL	NULL	NULL
2	104 Агломерный департамент	NULL	Area	100	NULL	21
3	105 Аглоцех-1	NULL	Area	104	NULL	9
4	106 Аглоцех-2	NULL	Area	104	NULL	9
5	107 Аглоцех-3	NULL	Area	104	NULL	9
6	108 Доменный цех-1	NULL	Area	104	NULL	9
7	109 Доменный цех-2	NULL	Area	104	NULL	9
8	110 Аглоцех МП	NULL	Area	104	NULL	9
9	111 Сталеплавильный департамент	NULL	Area	100	NULL	21
10	112 Конвертерный цех	NULL	Area	111	NULL	9
11	113 Мартеновский цех	NULL	Area	111	NULL	9
12	114 Огнеупорно-известковый цех	NULL	Area	111	NULL	9
13	115 Копровой цех	NULL	Area	111	NULL	9
14	116 Прокатный департамент	NULL	Area	100	NULL	21
15	122 Вальцетокарный цех	NULL	Area	116	NULL	9
16	123 Транспортный департамент	NULL	Area	100	NULL	21
17	124 ЦДСР	NULL	Area	100	NULL	21
18	125 РМЦ-3	NULL	Area	124	NULL	9
19	130 Миксерное отделение	NULL	Process Cell	112	NULL	11
20	131 Миксерный блок 1	NULL	Process Cell	130	NULL	11
21	132 Миксерный блок 2	NULL	Process Cell	130	NULL	11
22	133 Конвертерное отделение	NULL	Process Cell	112	NULL	11
23	134 Конвертерный блок 1	NULL	Process Cell	133	NULL	11

Запрос успешно выполнен. krr-tst-pahwi02 (11.0 RTM) EUROPE\evshvets (132) KRR-PA-ISA95_PRODUCTION 00:00:00 217 строк

Схема ISA95_PHYSICAL_ASSET

PhysicalAsset (Физический актив) – понятие, напрямую связанное с оборудованием. Физический актив – конструкционный элемент оборудования.

Информационная схема PhysicalAsset предназначена для описания структуры оборудования согласно проектной документации. Она заполняется элементами оборудования, когда в проекте предусматривается указание периода технического обслуживания, срока эксплуатации оборудования, сроков заказа оборудования; в противном случае схема не заполняется.



Элементы PhysicalAsset связаны со схемой Equipment через таблицу EquipmentAssetMapping. Через нее указывается, какую роль играет оборудование в реальных производственных процессах.

Через таблицу ResourceRelationshipNetwork определяется связь с покупными запчастями и узлами (MaterialDefinition, MaterialLot).

3.2.2. Организация технического обслуживания оборудования.

Для решения задачи технического обслуживания, оборудование в [PhysicalAssetClass] разделено на классы, с возможностью построения иерархии объединения с помощью ParentID согласно следующего примера:

	Description	Manufacturer	HierarchyScope	ParentID	ID
69	Датчики расхода	NULL	NULL	10000	10018
70	Датчик расхода по перепаду давления	NULL	NULL	10018	10019
71	Датчик перепада давления PMD55	Endress+Hauser	NULL	10019	10020
72	PMD55-AA22AA67DGCBJA2A-AI	Endress+Hauser	NULL	10020	10021
73	PMD55-AA22AA67DGCBJA2C-AIHB	Endress+Hauser	NULL	10020	10022
74	PMD55-AA22AA67DGCBJA2A-AIHB	Endress+Hauser	NULL	10020	10023
75	PMD55-AA22AA67DGCBJA2A-AIHB	Endress+Hauser	NULL	10020	10024
76	PMD55-AA22AA67DGCBJA2A-AIHB	Endress+Hauser	NULL	10020	10025

Связь между конкретным датчиком в Physical Asset и производственной линией/цехом из Equipment, к которому он принадлежит осуществляется через таблицу EquipmentAssetMapping.

К примеру, имеем датчик перепада давления, принадлежащий установке «Печь Ковш ОНРС»:

[Equipment]

ID	Description	Location	EquipmentLevel	Equipment	HierarchyScope	EquipmentClassID
1	100 АрселорМиттал Кривой Рог	NULL	Enterprise	NULL	NULL	NULL
2	111 Сталеплавильный департамент	NULL	Site	100	NULL	21
3	112 Конверторный цех	NULL	Area	111	NULL	9
4	143 ОНРС	NULL	Production Line	112	NULL	11
5	144 УПК	NULL	Work Cell	143	NULL	11
6	145 МНПЗ	NULL	Work Cell	143	NULL	11

В таблице [PhysicalAsset] любой физический актив должен быть потомком от оборудования, указанного в таблице Equipment и имеющего EquipmentLevel. Пример на рисунке ниже, где датчик температуры согласно иерархии объединения, принадлежит конкретному Work Cell из Equipment:

ID	Description	HierarchyScope	PhysicalLocation	FixedAssetID	VendorID	EquipmentLevel	PhysicalAsset	PhysicalAssetClassID
1	9999 УПК	NULL	NULL	NULL	NULL	Work Cell	NULL	NULL
2	10000 Датчик температуры	NULL	NULL	NULL	NULL	NULL	9999	10014

Для связи между [Equipment] и [PhysicalAsset] используется [EquipmentAssetMapping], поэтому запись в таблице должна иметь следующий вид:

[EquipmentAssetMapping]

EquipmentID	PhysicalAssetID	StartTime	EndTime
144	9999	04.01.2017	

Техническое обслуживание подразумевает следующие основные понятия, которые должны отражены в следующих таблицах:

№	Наименование	Задействована схема	Таблица
1	период тех. обслуживания	PhysicalAsset	PhysicalAssetProperty
2	срок замены оборудования	PhysicalAsset	PhysicalAssetProperty
3	наличие ЗИП на складе	Material	MaterialActual
4	Ввод/Вывод из эксплуатации	PhysicalAsset	PhysicalAssetProperty

Схемы PhysicalAsset и схему Material необходимо связывать с помощью ResourceRelationshipNetwork (Сетевой модели связи ресурсов).

3.2.3. Диагностика оборудования.

Для организации мониторинга состояния оборудования, объекты в таблице PhysicalAsset, относящиеся к:

- IT оборудования,
- датчикам,
- PLC,
- определённым типам электроустройств типа частотного привода,
- ПК

должны иметь свойство status (In Work, Failure и т.д.) в таблице PhysicalAssetProperty. Данное свойство будет показывать их состояние в текущий момент времени, что позволит производить мониторинг готовности оборудования.

При этом возникает необходимость периодического обновления свойства status, путем выполнения команды Ping, слежения за временем обновления параметров датчиков или проверки валидности измеряемых параметров.

Пример реализации WEB интерфейса по работе с системой диагностики показан на рисунке ниже.

ArcelorMittal Маркетолог Маркировщик **Специалист АСУТП** Аналитик Специалист Цеха

EUROPElevshvets RU ▾

Биржи Версия **Диагностика** Заказы Логотипы Материалы Оборудование Персонал Связь с SAP

Шаблоны Бирок Отчёты по газу Тренды по газу gasCollection.gasBalance

☒ Система печати бирок включена
 Выберите сторону:
 СПЦ-1 \ MC 250-1 \ Левая ▾

СПЦ-1 \ MC 250-1 \ Левая

Название оборудования	Статус оборудования
PLC	OK
UPS	OK
Карман 1	OK
Карман 2	OK
Карман 3	OK
Карман 4	OK
Принтер MC 250-1 левая	OK

3.3. Схема ISA95_PERSONNEL

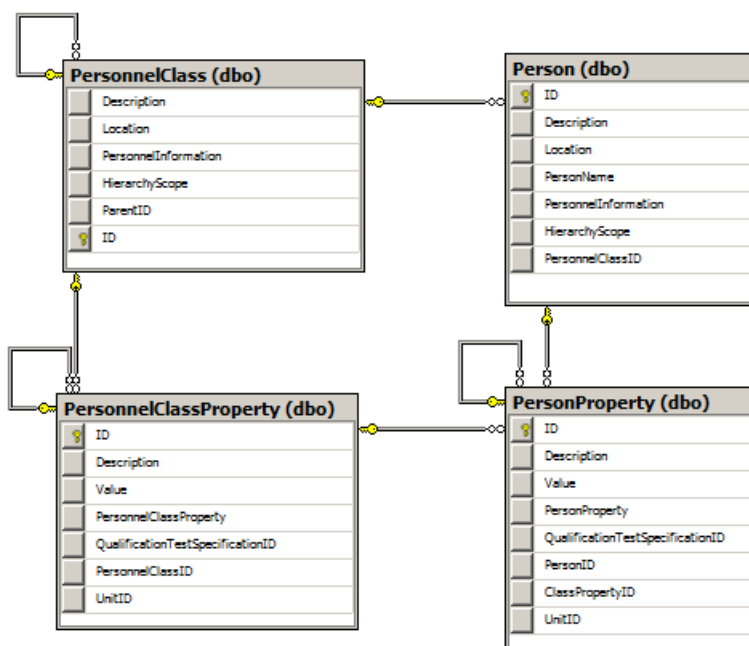


Таблица Person содержит информацию о персонале, выполняющем определенную работу и принадлежащему к определенной профессии.

Таблица PersonnelClass описывает группы лиц к определенному классу.

Примеры классов Весовщик, Оператор дистрибутора и т.д.

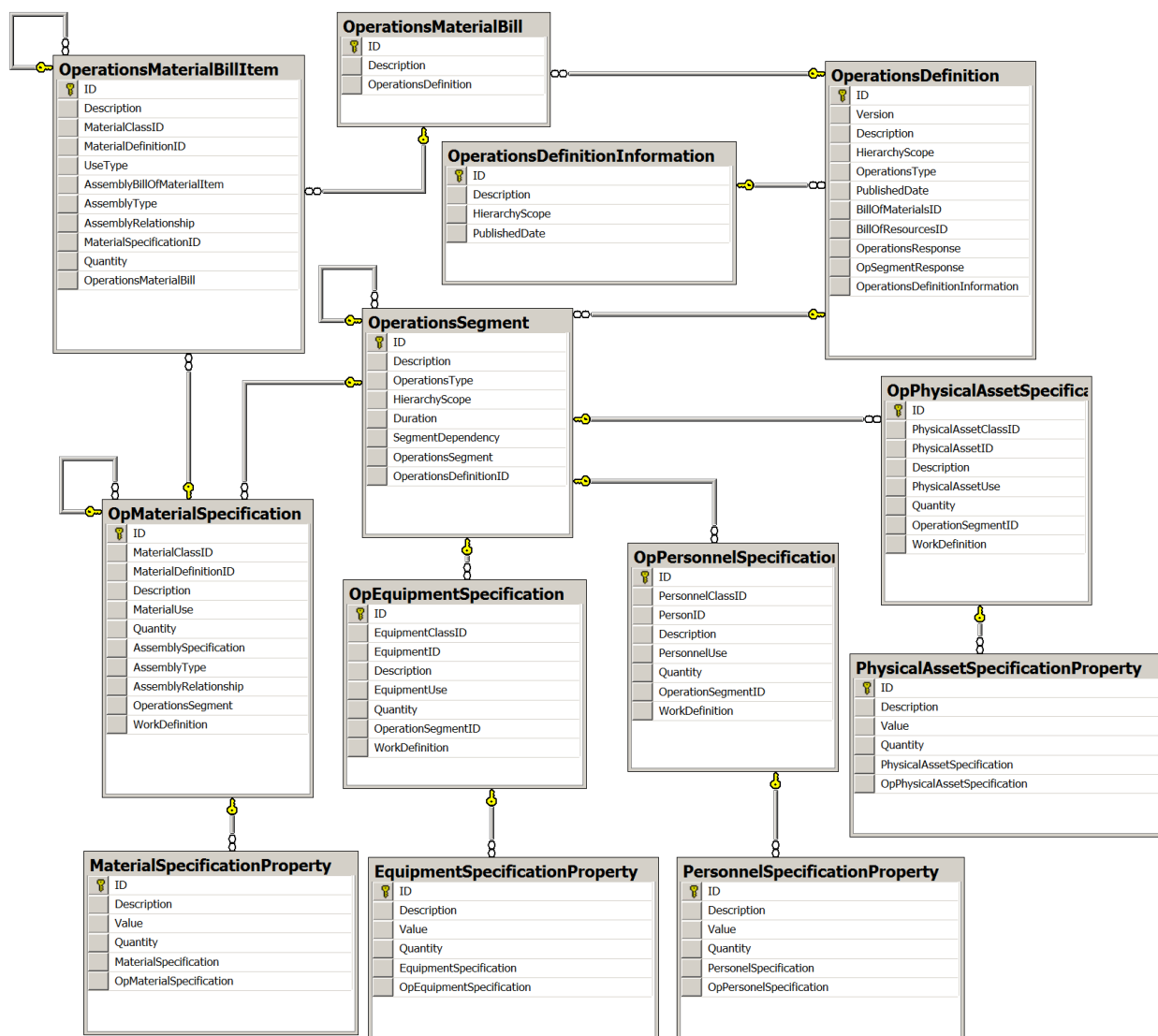
	ID	Description	Location	PersonnelInformation	HierarchyScope	ParentID
1	1	Маркировщик	NULL	NULL	NULL	NULL
2	3	Специалист АСУТП	NULL	NULL	NULL	NULL
3	4	Маркетолог	NULL	NULL	NULL	NULL
4	5	Весовщик	NULL	NULL	NULL	NULL

Таблица PersonnelClassProperty описывает свойства группы. К примеру свойство «График Работы» 042 или 170.

3.4. Схема ISA95_OPERATION_DEFINITION

Данная схема касается создания производственных операций к такой операции можно отнести производственную операцию по размещению заказа на производство определенной партии продукции с заданными характеристиками, установленным весом партии и с определённым сроком производства.

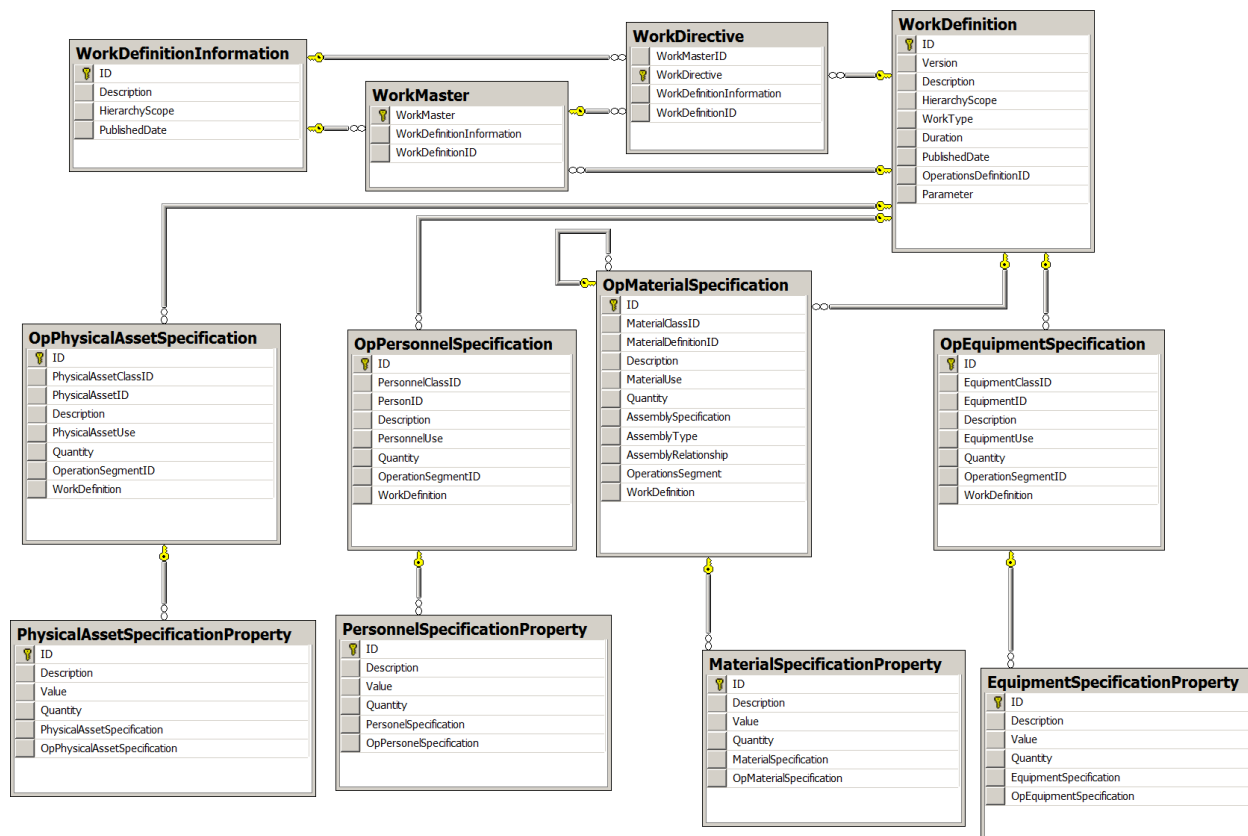
Производственная операция по выпуску партии продукции подразумевает заказ производственных мощностей, материалов Bill of Material и ресурсов Bill of Resources необходимых для выпуска данной продукции, поэтому данная схема затрагивает таблицы групп Material, Personal, Equipment, Physical Asset.



3.5. Схема ISA95_WORK_DEFINITION.

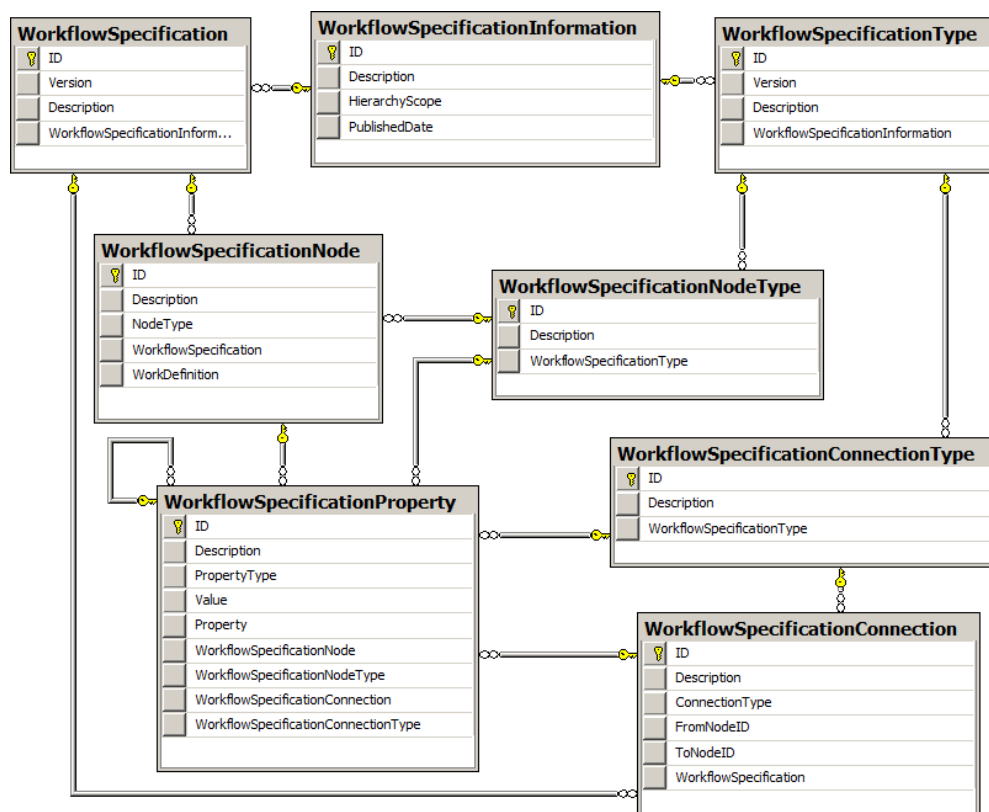
Согласно схемы Work Definition Производственные операции, которые описаны в главе 3.5 разделяются по цехам изготовителям, задействованным в изготовлении данной партии продукции.

В зависимости от количества цехов, задействованных в производственной операции, создается такое же количество записей в таблице Work Definition, которые в дальнейшем подразделяются на определенные работы Work из которых состоит рабочий процесс цеха такие как: прокатка, упаковка, стрипперование, загрузка вагона, которые в свою очередь делятся на элементарные операции Job: подъем, опускание, снятие со стенда, взвешивание одной порции и т.д.



3.6. Схема ISA95_WORKFLOW_SPECIFICATION.

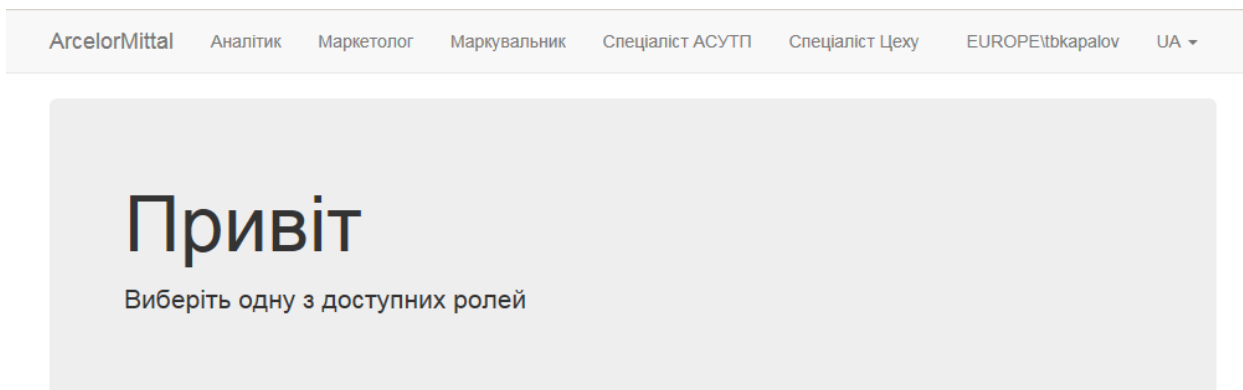
С помощью схемы ISA95_WORKFLOW_SPECIFICATION можно построить последовательность выполнения рабочего процесса, состоящего из Work и Job описанных в главе 3.6. В данной схеме совокупность операций рабочего процесса расставляется по порядку в виде последовательности действий.



4. Работа с веткой FrontEnd.

4.1. Общая информация.

FrontEnd – веб-приложение, реализующее интерфейс между пользователем и бизнес-логикой сервера. Основан на фреймворке AngularJS. В зависимости от роли пользователю в приложении доступны соответствующие вкладки. Доступ к данным БД реализуется при помощи REST сервера odata_unified_svc, поддерживающем OData протокол.



4.2. Добавление новой роли.

Для того что бы добавить новую роль в систему, которая будет отображаться как еще одна вкладка, необходимо добавить запись в таблицу KPPRoles в БД KRR-PA-ISA95_PRODUCTION.

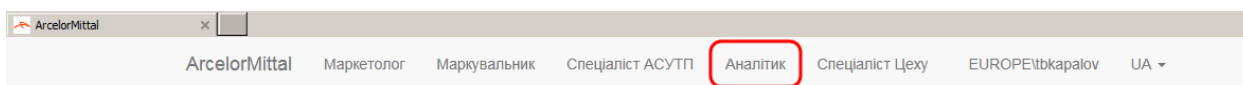


Таблица содержит имя роли в системе и имя Active Directory группы что соответствует этой роли. Роль может содержать несколько AD групп пользователей.

	ID	RoleName	ADRoleName
1	1	Market	europe\KRR-LG-PA-LabelPm_Market
2	2	PA	europe\KRR-LG-PA-LabelPm_PA
3	3	WorkshopSpecs	europe\KRR-LG-PA-LabelPm_WorkshopSpecs
4	4	Marker	europe\KRR-LG-PA-LabelPm_Marker
5	6	WeightAnalytics	EUROPE\KRR-LG-PA-Weight Analytics
6	7	AutomationLab	EUROPE\KRR-LG-SQL-PAHWL-DATP_AutomationLab

Для того что бы определить содержимое закладки для новой роли необходимо:

A. В проекте зайти в папку Scripts -> App, далее в файл Index.js

B. В этом файле нужно создать Angular state, который соответствует роли. Например, для роли WeightAnalytics существует такой state:

```
.state('app.WeightAnalytics', {  
    url: '/weightanalytics',  
    templateUrl: 'Static/weightanalytics/index.html',  
    controller: 'WeightAnalyticsCtrl',  
    onEnter: function ($state, roles) {  
        if (!vmIsAuthorized('WeightAnalytics', roles))  
            $state.go('app.error', { code: 'unauthorized' });  
    }  
})
```

С. После создания стейта мы создаем файл js, который будет соответствовать этой роли. Создается он в той же папке Scripts -> App (например, WeightAnalytics.js). В этом файле мы создаем ангулярский контроллер, который будет обрабатывать эту роль, и внутри которого будет описываться ее бизнес-логика.

Пример создания контроллера для роли WeightAnalytics:

```
app.controller(WaitAnalyticsCtrl, ['$scope', 'indexService', '$state', 'roles',
function ($scope, indexService, $state, roles) {

//business logic here
}]);
```

Название контроллера может быть любым, но оно должно соответствовать тому, которое прописано в конфигурации стейта. Чтобы не путаться, контроллер называем также, как роль, но дописываем к названию Ctrl. Получается WeightAnalytics + Ctrl = WaitAnalyticsCtrl

D. После создания js файла его нужно включить в сборщик проекта для слияния с остальным кодом. Для этого мы идем в проекте в папку App_Start, там заходим в файл BundleConfig.cs . там мы находим секцию bundles.Add(new Bundle("~/bundles/js/app").Include(...)) и вписываем в нее путь к нашему созданному файлу по аналогии с остальными

E. Создаем HTML шаблон страницы роли. Для этого мы идем в папку Static. Создаем там подпапку с названием роли (регистр неважен, но лучше в нижнем) и внутри этой папки создаем файл index.html

4.3. Получение данных.

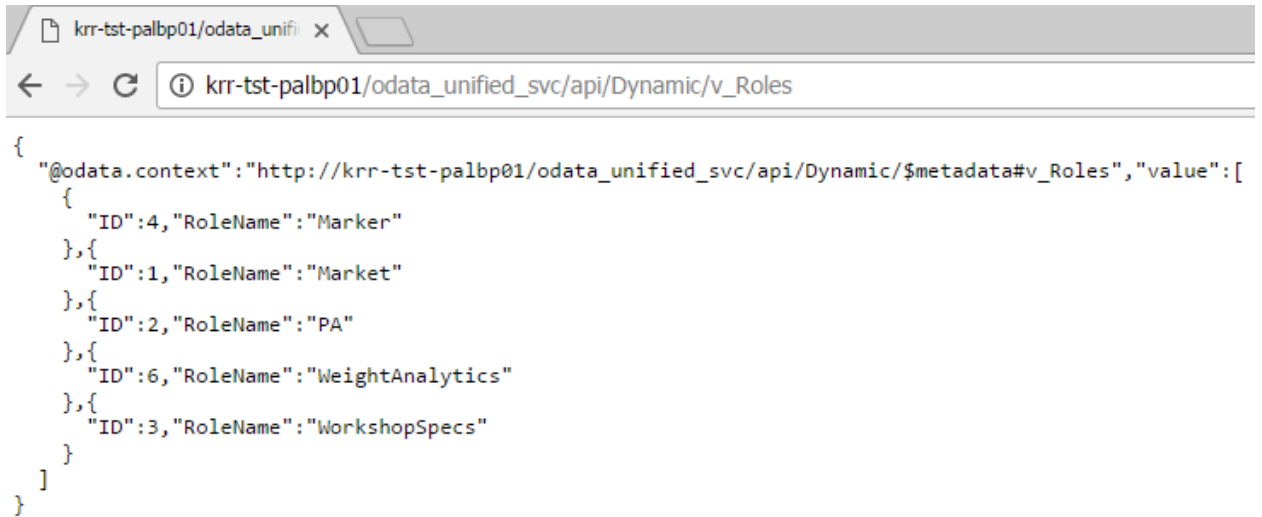
Получение данных из БД во FrontEnd заключается в посылке XMLHttpRequest запроса из Frontend'a сервису odata_unified_svc, который в свою очередь возвращает ответ в JSON. формате. На клиентской стороне для этого служит ангуляровский сервис indexService и его метод getInfo. Общий вид его использования следующий:

```
function vmGetSomeData() {
    indexService.getInfo('v_SomeView')
    .then(function (response) {
        var SomeData = response.data.value;
    });
};
```

Т.е. функция vmGetSomeData() использует сервис для получения в ответе response данных из SQL представления v_SomeView. Также в запросе можно использовать OData синтаксис для фильтрации и сортировки данных:

```
indexService.getInfo("v_SomeView?$filter=Column_1 eq
{0}&$orderby=Column_2".format(param))
```

Данные в JSON можно получить и непосредственно в браузере, обратившись к odata_unified_svc. Например, данные из представления v_Roles, в которой содержатся доступные пользователю роли:



The screenshot shows a web browser window with the address bar displaying the URL: `krr-tst-palbp01/odata_unified_svc/api/Dynamic/v_Roles`. The page content shows a JSON response from an OData service. The response is an array of objects, each representing a role. The objects contain an 'ID' and a 'RoleName'.

```
{
  "@odata.context": "http://krr-tst-palbp01/odata_unified_svc/api/Dynamic/$metadata#v_Roles", "value": [
    {
      "ID": 4, "RoleName": "Marker"
    }, {
      "ID": 1, "RoleName": "Market"
    }, {
      "ID": 2, "RoleName": "PA"
    }, {
      "ID": 6, "RoleName": "WeightAnalytics"
    }, {
      "ID": 3, "RoleName": "WorkshopSpecs"
    }
  ]
}
```

Для отображения данных в табличном виде можно использовать уже существующий компонент jsGrid:

```
$('#table_html_id').jsGrid({
  height: "500px",
  width: "720px",

  sorting: false,
  paging: true,
  editing: true,
  filtering: true,
  autoload: true,
  pageLoading: true,
  inserting: true,
  pageIndex: 1,
  pageSize: 10,
  rowClick: function (args) {}

}).jsGrid('initOdata', {
  serviceUrl: serviceUrl,
  table: 'SQL_table_name',

  fields: [{
    id: 'ID',
    name: 'ID',
    title: 'ID',
    readonly: true,
    type: 'number',
    order: 1
  }, {
    id: 'Description',
    name: 'Description',
    title: $translate.instant('pa.grid.material.description'),
    order: 2
  }]
}).jsGrid('loadOdata', {
  defaultFilter: 'ID eq -1',
  order: 'Description'
});
```

4.4. Запись/редактирование данных.

Редактировать данные непосредственно в таблицах позволяет вышеописанный компонент jsGrid (если разрешена функция вставки/редактирования).

Для обращения к SQL процедурам служит метод `sendInfo` сервиса `indexService`:

```
indexService.sendInfo('SQL_procedure_name', {
    param_1: 100500,
    param_2: 'param',
    param_3: 'value'
}).then(function (response) {
    var r = response;
});
```

Он посылает POST запрос сервису `odata_unified_svc` с указанными параметрами.

4.5. Локализация.

Для поддержки языков существует сервис `$translate` и файл `Scripts -> App -> Resources.js`, в котором содержатся переменные и их переводы на поддерживаемые языки. Использование сервиса в html коде и ангуляровском контроллере следующее:

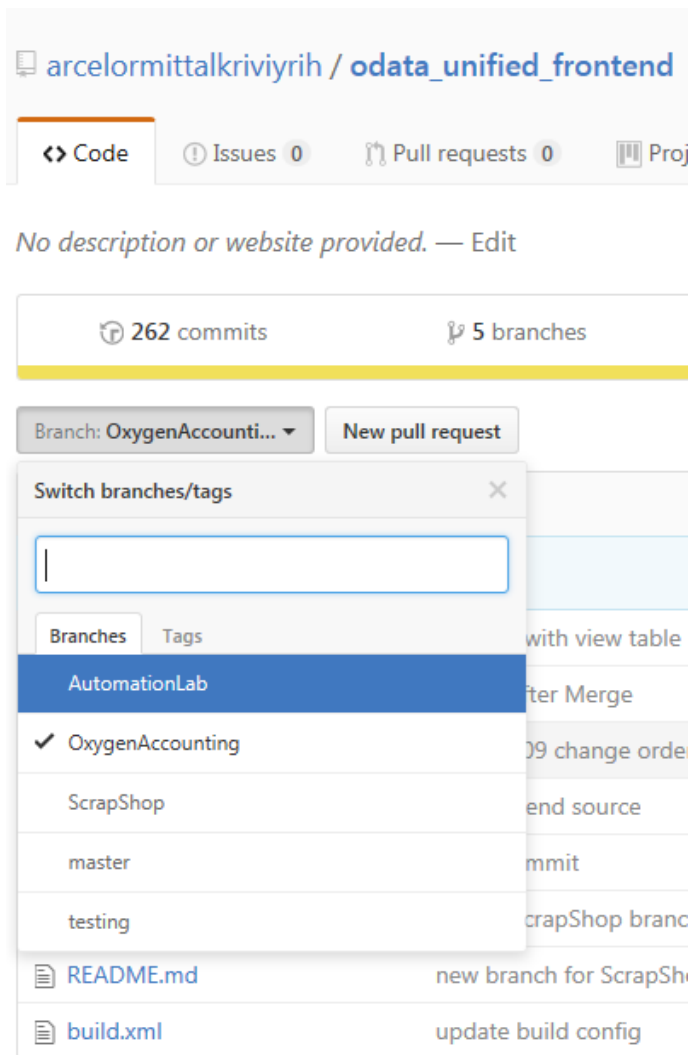
```
{{ mode.Description | translate }}
```

```
var nodename = $translate.instant('tree.nodeName');
```

4.6. Коммиты.

В репозитории `odate_unified_frontend` существует несколько веток:

- основная **master** (сборка этой ветки разворачивается на продуктивном сервере **krr-app-palbp01**);
- тестовая **testing** (сборка – на тестовом сервере **krr-tst-palbp01**);
- отдельные ветки проектов (например, **ScrapShop**, **AutomationLab**).



Вся разработка ведется в ветках своих проектов. При успешном тестировании своей ветки, она сливается в ветку **testing**, сборка которой разворачивается и проверяется на тестовом сервере. После того, как все удостоверятся, что на **krr-tst-palbp01** все части проекта работают без замечаний, ветка **testing** сливается ответственным лицом в **master**, а оттуда – на продуктивный сервер **krr-app-palbp01**.

5. Структура View и процедур для Entity, используемых в диаграммах

5.1. Entity для списка диаграмм

View для списка диаграмм должна иметь следующие поля:

```
ID (int, not null)
Description (nvarchar(50), null)
json (nvarchar(max), null)
```

ID - Идентификатор диаграммы

json - json диаграммы (заполняется компонентом)

Description - Название диаграммы

Процедура удаления диаграммы должна называться [Имя Entity(Имя View)]_delete например v_Diagram_delete.

А также иметь следующие параметры:

ID INT

Процедура редактирования должна называться [Имя Entity(Имя View)]_update например v_Diagram_update

А также иметь следующие параметры:

ID INT,
json NVARCHAR(MAX),
Description NVARCHAR(50)

Процедура добавления должна называться [Имя Entity(Имя View)]_insert например v_Diagram_insert

А также иметь следующие параметры:

json NVARCHAR(MAX),
Description NVARCHAR(50)

5.2. Entity для списка нодов

View для списка нодов должна иметь следующие поля:

```
ID (int, not null)
Description (nvarchar(50), null)
DiagramID (int, null)
json (nvarchar(max), null)
```

ID - Идентификатор нода

json - json нода (заполняется компонентом)

DiagramID - Идентификатор диаграммы

Description - Имя нода

Процедура удаления должна называться

[Имя Entity(Имя View)]_delete например v_DiagramNode_delete

А также иметь следующие параметры:

ID INT

Процедура редактирования должна называться [Имя Entity(Имя View)]_update например v_DiagramNode_update

А также иметь следующие параметры:

ID INT,
json NVARCHAR(MAX),
DiagramID INT,
Description NVARCHAR(50)

Процедура добавления должна называться [Имя Entity(Имя View)]_insert например v_DiagramNode_insert

А также иметь следующие параметры:

DiagramID INT,
json NVARCHAR(MAX),
Description NVARCHAR(50)

5.3. Entity для списка связей

View для списка связей должна иметь следующие поля:

ID (int, not null)
Description (nvarchar(50), null)
FromNodeID (int, not null)
ToNodeID (int, not null)
DiagramID (int, null)
json (nvarchar(max), null)

ID - Идентификатор связи
json - json связи (заполняется компонентом)
FromNodeID - Идентификатор нода начала связи
ToNodeID - Идентификатор нода окончания связи
DiagramID - Идентификатор диаграммы
Description - Имя связи

Процедура удаления должна называться

[Имя Entity(Имя View)]_delete например v_DiagramConnection_delete

А также иметь следующие параметры:

ID INT

Процедура редактирования должна называться [Имя Entity(Имя View)]_update например v_DiagramConnection_update

А также иметь следующие параметры:

ID INT,
json NVARCHAR(MAX),
FromNodeID INT,
ToNodeID INT,

DiagramID	INT,
Description	NVARCHAR(50)

Процедура добавления должна называться [Имя Entity(Имя View)]_insert например v_DiagramConnection_insert а также иметь следующие параметры:

FromNodeID	INT,
ToNodeID	INT,
DiagramID	INT,
json	NVARCHAR(MAX),
Description	NVARCHAR(50)

6. Настройка сайта

Настройки сайта хранятся в файле Scripts\Shared\common.js

Параметр	Описание	Пример
domainURL	Адрес OData сервиса	../../odata_unified_svc
serviceUrl	Путь к API	domainURL + '/api/Dynamic/'
sapUrl	Адрес сервиса получения информации о заказах из SAP	../../odata_sap_svc/GetSAPInfo?orderNo=
scalesRefresh	Время обновления экрана маркировщицы весовыми данными, мс	1000
workRequestRefresh	Время обновления экрана маркировщицы в спец режимах (отбраковка, сортировка...), мс	1000

Примечание: Для папки где находится веб сервис, необходимо что бы были права доступа на чтение, выполнение и просмотр содержимого папок для группы Пользователи (т.е. все пользователи):

