Григорий Долгий

2021

РЕШЕНИЕ TIS\_ESPC\_FORK

Общее описание

ПАО «ЧМК»



# ОБЩИЕ СВЕДЕНИЯ

Данное решение является модернизацией существующей системы ТИС ЭСПЦ. Целью модернизации является:

* Замена существующей версии (10.2.19.193)
* Предоставление более удобного интерфейса для работы с информацией
* Переход на сравнительно новые технологии веб разработки для увеличения надежности, простоты поддерживания и изменения

## Требования

* Ввиду применения современных технологий поддерживаются только современные браузеры Chrome , Firefox , новый Edge 
* Старые браузеры не способны правильно отобразить страницы (к ним относятся старый Edge  и Internet Explorer ).

## Как войти

* Новая версия (тестовый релиз) доступен по адресу <http://10.2.19.193:82>
* Если все в порядке, то на входе должна отобразиться страница авторизации:

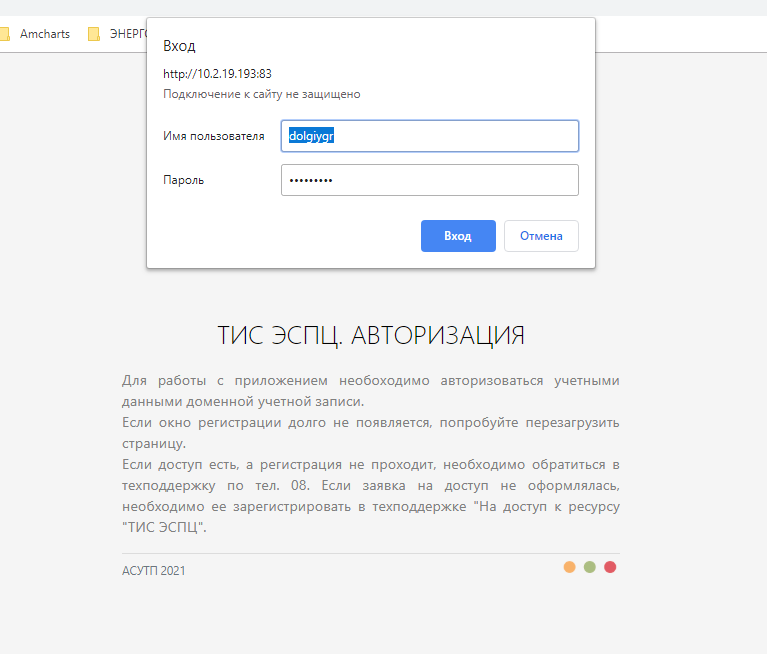


Рисунок 1 – Окно входа в систему

* Вход осуществляется вводом доменных учетных данных
* При успешном входе должна отобразиться стартовая страница:

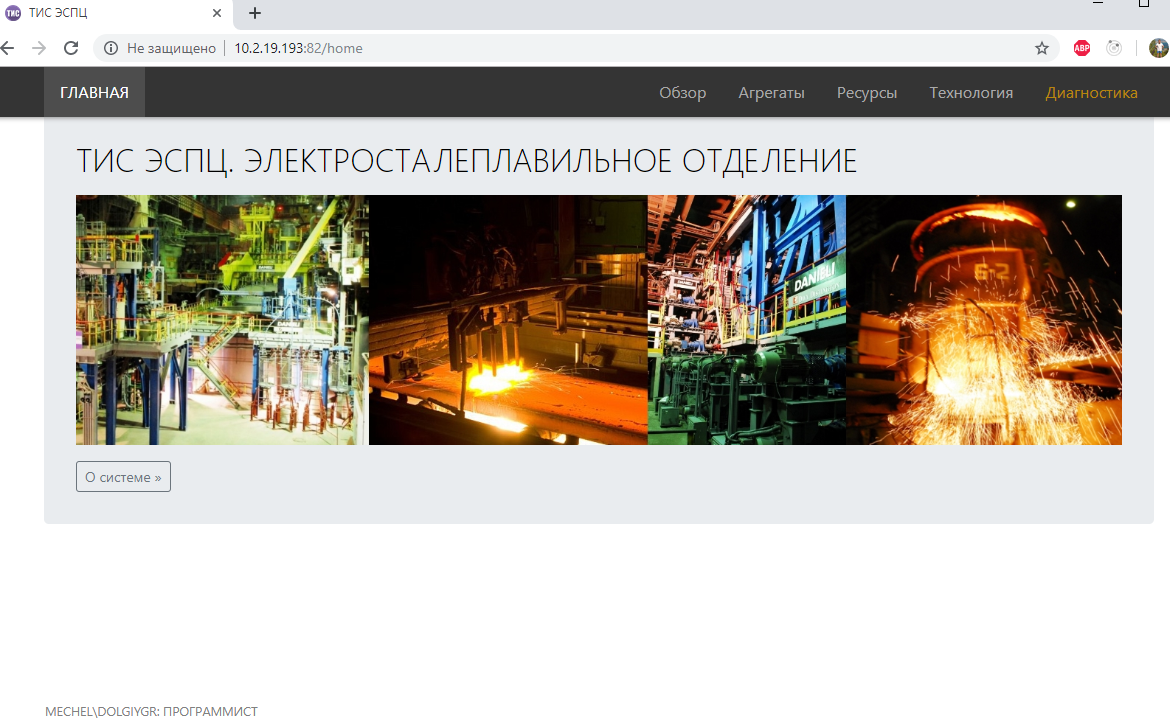


Рисунок 2 – Внешний вид стартовой страницы приложения

# АРХИТЕКТУРА РЕШЕНИЯ

## Выбор средств разработки

Ввиду того, что площадкой развертывания ТИС является Windows Server 2008, применение современных решений от Microsoft не представляется возможным. Речь идет о фреймворке .Net Core с открытым исходным кодом и доступном для скачивания с оф. сайта Microsoft. Поэтому, с целью уйти от устаревших и сложных в поддерживании и модернизации Windows Forms, на которых построен существующий ТИС ЭСПЦ, принято решение переписать решение используя современные технологии.

Так, серверная часть решения выполнена в виде MVC-приложения с применением API-контроллеров, а клиентская часть – с применением современной библиотеки React. Для редактирования серверной части приложения использована среда разработки VisualStudio 2019. Для редактирования клиентской части – VisualStudio Code.

Готовое решение состоит из двух отдельных проектов, разворачиваемых на разных портах сервера в виде отдельных сайтов. Клиентская часть на порту 82, а серверная на 83. В процессе работы клиентская часть выполняет **http** запросы к серверной части. А серверная часть отвечает на запросы только с одного указанного порта. Подробнее про публикацию можно почитать в разделе ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ.

## Структура решения

В решении создан проект с одноименным названием построенный по шаблону «Веб приложение ASP – Веб API».

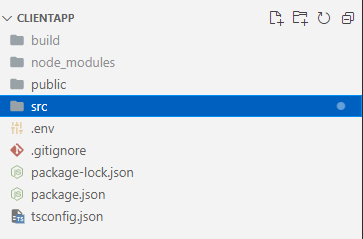
Выбранный шаблон дает площадку для создания необходимых API контроллеров для получения данных из БД или других серверных источников и отправки данных на клиент.

Для поддержания стандартной структуры папок в проекте в корне создана папка **ClientApp** по аналогии с проектами .Net Core, содержащая клиентскую часть приложения.

Таким образом, используя 2 программы для разработки приложения (VisualStudio – для серверной части и открыв папку ClientApp в VisualStudio Code) удобно редактировать решение и пользоваться всеми преимуществами среды VisualStudio Code.

Папка Controllers содержит необходимые API-контроллеры. В конфигурационном файле приложения содержится настройка обработчика ролей и строки подключения к базам данных.

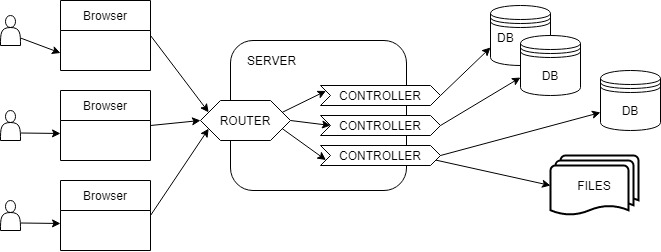
## Структура клиентской части

Клиентская часть выполнена проектом шаблона **create-react-app** с модификацией **Typescript**. Структура корневых папок проекта представлена на рисунке справа.

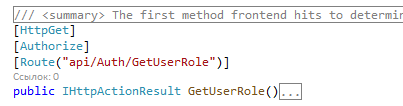
Весь полезный код компонентов расположен в папке **src**. Файл **package.json** содержит список подключенных библиотек. Файл **tsconfig.json** содержит настройки компилятора typescript. В папке **build** располагается релиз клиентской части, создаваемый в процессе компиляции проекта. Папка **node\_modules** является стандартной для фреймворка **webpack** и содержит исходные коды всех использованных javascript библиотек.

# СЕРВЕРНАЯ ЧАСТЬ РЕШЕНИЯ

## Принцип действия схемы REST-API

Преимущество подхода REST-API сводится к возможности размещения сервера и баз данных в разных местах. При этом сервер выполняет роль единой входной точки для доступа к данным. А его маршрутизация позволяет задействовать нужный контроллер.

## API-контроллеры

API‑контроллер возвращает информацию в формате JSON, удобном для работе на клиентской части приложения. Контроллер является логически обособленной точкой для получения информации. Так, по названию контроллера можно определить какого рода данные он может предоставить.

Контроллеры располагаются в папке **Controllers**. Представляют собой классы, наследованные от базового **ApiController**.

Методы контроллеров представляют точки доступа по протоколу HTTP. Так, над каждым методом указан путь, на который нужно обратиться, чтобы получить необходимые данные. На рисунке выше, например, это путь «api/Auth/GetUserRole».

Все запросы к API-контроллерам начинаются с приставки «api».

Котроллеры отвечают за правильную обработку запросов и предоставление нужных данных. В зависимости от реализации того или иного метода, контроллер может обратиться как к базе данных, так и к локальному источнику данных (файлы, изображения и пр.). Для обращения к базе данных используется ряд классов, реализующих шаблон Repository. Таким образом, работа с базой данных полностью делегирована моделям репозиториев, стандартизуя решение и отделяя работу с базой от логики работы самого контроллера.

Все использованные модели описывающие данные в виде классов располагаются в папке **Models**.

## Модели

Для реализации объектного подхода доступа к данным, каждая сущность описана в виде отдельного класса. Это базовые сущности работы с данными. Эти сущности расположены в подпапке **DTOs**.

Остальные папки содержат обработчики данных, которые используют базовые модели из папки DTOs.

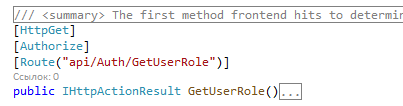
Папка **Auth** содержит обработчик ролей пользователей. В папке **Diagnostics** расположены обработчики диагностических данных. Папка **Repositories** вмещает репозитории для работы с базами данных.

## Защита доступа к данным

Серверная часть приложения защищена с помощью механизмов аутентификации и авторизации.

### Аутентификация

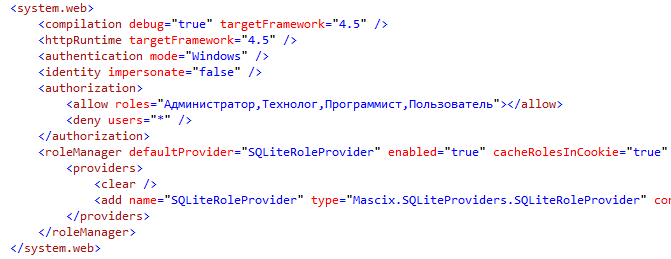
При входе на сайт предлагается ввести свои доменные учетные данные. Этот способ аутентификации удобен тем, что обработка введенных данных делегируется контроллеру домена и данные для регистрации в домене и на сайте ТИС одни и те же. При этом, если доменные учетные данные введены не верно, использование сайта невозможно.

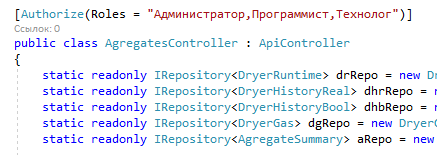
Каждый метод контроллера либо сам контроллер в целом снабжен тегом [Authorize], который подразумевает прохождение процедуры аутентификации перед выполнением кода. Если аутентификация не пройдена, то сервер вернет ошибку.

Механизм аутентификации реализован стандартными средствами IIS сервера на котором развернут релиз приложения. Подробнее можно прочитать в разделе ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ.

### Авторизация

Для разграничения прав пользователей необходим функционал ролей пользователей. Этот функционал реализован свободно распространяемым классом **SQLiteRoleProvider** из набора **Mascix**.

Подключение функционала класса осуществляется в конфигурационном файле проекта **Web.config** в секции **system.web**.

Функционал класса **SQLiteRoleProvider** позволяет назначить каждому пользователю определенную роль. Роли отличаются правами доступа к ресурсам. Пример ограничения всего контроллера на использование определенными ролями представлен на рисунке справа.

Таблицы соответствия ролей пользователей их доменным учетным записям находятся в базе данных **SQLite**, расположенной в папке App\_Data проекта. Администраторы могут редактировать ролевой доступ пользователей посредством удобного веб-интерфейса на сайте приложения.

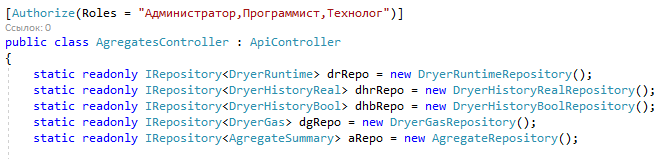
## Auth-контроллер

Контроллер предоставляет точки входа для управления ролевой моделью приложения. Средствами данного контроллера выполняются стандартные методы по редактированию ролевого доступа пользователей.

В данном приложении реализацией этого механизма служит класс **SQLiteRoleProvider**, который реализует базовый класс и которому делегирована работа с ролями. База данных ролевой модели находится в папке App\_Data проекта в файле **auth.db3**. В нем содержатся список ролей и таблица назначения ролей пользователям.

Так как для работы с ролями необходимо знать доменное имя пользователя, все методы этого контроллера снабжены тегом [Authorize], который перед доступом к методу гарантирует прохождение пользователем доменной авторизации.

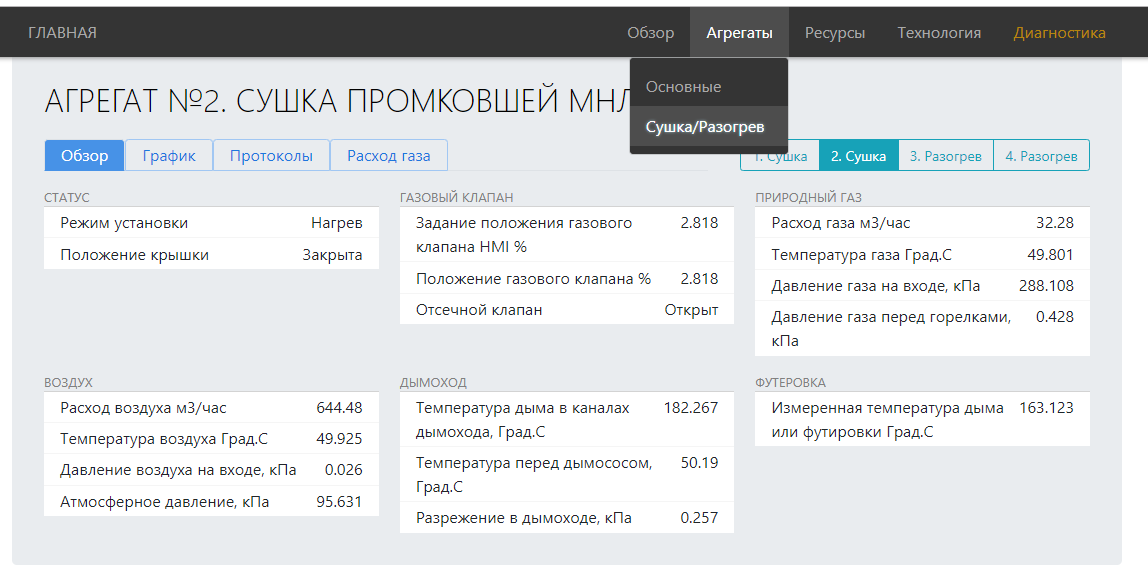
## Agregates-контроллер

Контроллер предоставляет точки входа для получения оперативной и исторической информации по агрегатам ЭСПЦ.

На рисунке слева показаны репозитории, которым он делегирует работу с базами данных. Каждый репозиторий реализует интерфейс IRepository и предоставляет как минимум один метод получения списка запрошенных параметров.

Все репозитории кроме **AgregateRepository** осуществляют выборку для представления «Сушка/Разогрев». Информация для этого представления берется из базы данных **tis\_runtime**. В эту базу данных пишут две службы, написанные на Delphi. Названия служб **FromDryPLC** и **FromHeaterPLC**.

### DryerRuntimeRepository

Предоставляет мгновенные данные для агрегатов сушки и разогрева. На рисунке слева показано представление, которое наполняется информацией, полученной от этого репозитория.

То есть на всех агрегатах «1.Сушка, 2.Сушка, 3.Разогрев, 4.Разогрев» вкладка (слева) «Обзор» наполняется этим репозиторием.

Репозиторий обращается к серверу MSSQL на сервере ТИС (10.2.19.193), базе данных **tis\_runtime**, таблицам с суффиксом **runtime**. Строка подключения к базе данных находится в конфигурационном файле проекта под именем **sqlTIS**.

### DryerHistoryRealRepository

Предоставляет исторические данные по всем агрегатам сушки/разогрева для вкладки **График**.

Репозиторий обращается к серверу MSSQL на сервере ТИС (10.2.19.193), базе данных **tis\_runtime**, таблицам с суффиксом **history**. Строка подключения к базе данных находится в конфигурационном файле проекта под именем **sqlTIS**.

### DryerHistoryBoolRepository

Предоставляет исторические данные по всем агрегатам сушки/разогрева для вкладки **Протоколы**. Возвращает информацию о произошедших событиях в пределах запрошенного интервала.

Репозиторий обращается к серверу MSSQL на сервере ТИС (10.2.19.193), базе данных **tis\_runtime**, хранимым процедурам с суффиксом **protocol**. Строка подключения к базе данных находится в конфигурационном файле проекта под именем **sqlTIS**.

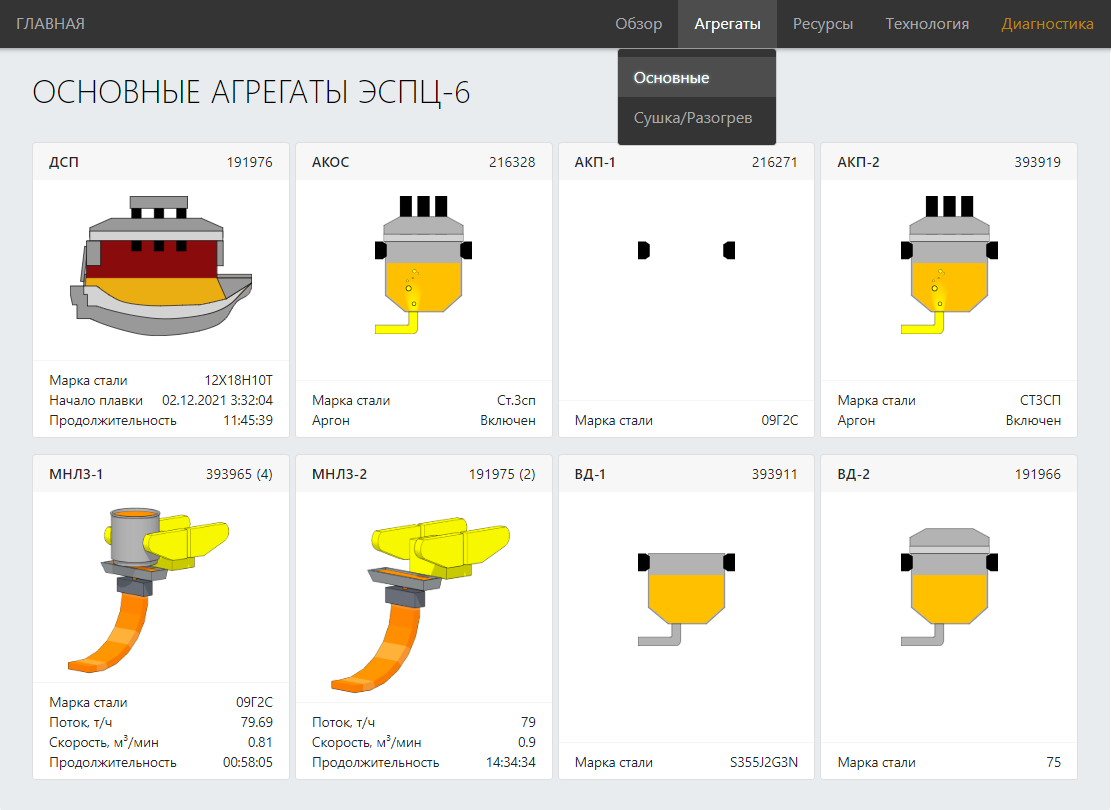
### DryerGasRepository

Предоставляет исторические данные по всем агрегатам сушки/разогрева для вкладки **Расход газа**.

Репозиторий обращается к серверу MSSQL на сервере ТИС (10.2.19.193), базе данных **tis\_runtime**, таблицам с корнем **GAS**. Строка подключения к базе данных находится в конфигурационном файле проекта под именем **sqlTIS**.

### AgregateRepository

Предоставляет мгновенную информацию о состоянии основных агрегатов ЭСПЦ. На рисунке ниже показано представление, за информацию на котором отвечает этот репозиторий.

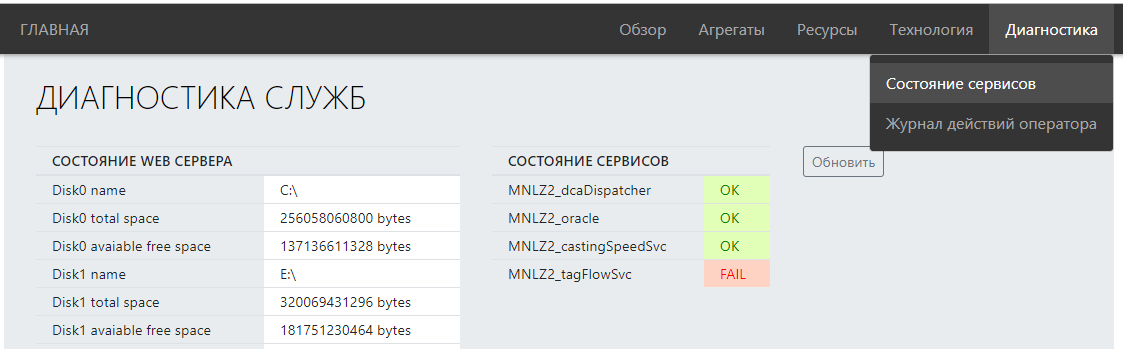
Репозиторий обращается к серверу MySQL на сервере диспетчера комбината (10.2.59.150), базе данных **obzor\_db**, таблицам **dev\_catalog** и **dev\_state**.

Строка подключения к базе данных находится в конфигурационном файле проекта под именем **mysql**.

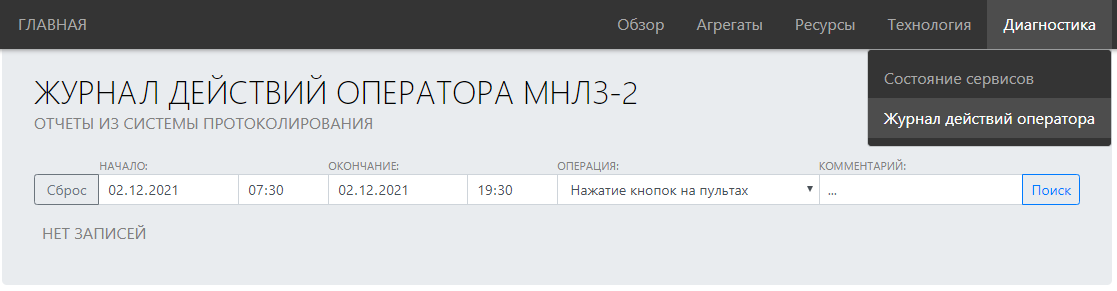
## Diagnostic-котроллер

Контроллер предоставляет диагностическую информацию о работе служб, состоянии веб-сервера и действиях операторов на пультах управления. Использует один репозиторий для доступа к данным.

### Диагностическая информация

Без обращения к базам данных контроллер анализирует требуемые службы и отображает оперативную информацию в веб представлении показанном на рисунке слева.

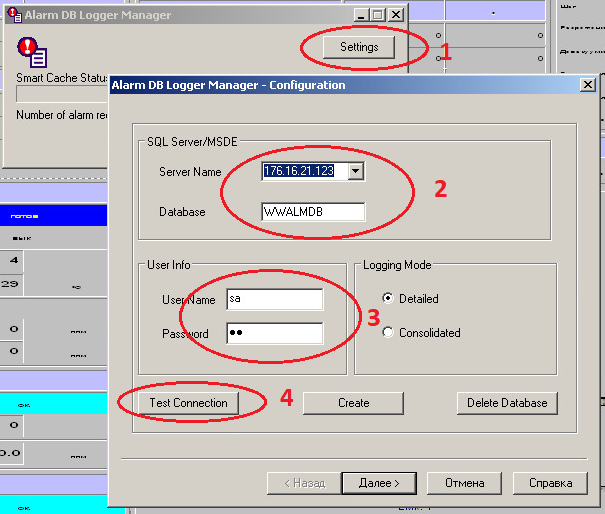
### Информация о нажатии кнопок операторами

Для получения информации контроллер делегирует общение с базами данных репозиторию **OperatorRepository**.

Репозиторий отвечает за представление показанное на рисунке справа.

Информация запрашивается из нескольких источников.

#### Информация по МНЛЗ

Для предоставления сведений о действиях операторов МНЛЗ репозиторий использует базу данных сервера MSSQL на сервере ТИС (10.2.19.193), базе данных **WWALMDB**. В эту базу удаленно пишут **Intouch** клиенты, развернутые на АРМах МНЛЗ.

Для настройки удаленного протоколирования необходимо зайти на нужную машину с клиентом Intouch, вызвать приложение **AlarmDbLoggerManager** и ввести учетные данные удаленной базы данных в последовательности, описанной на рисунке слева.

После ввода учетных данных необходимо проверить соединение нажав Test Connection.

Так, для настройки логгирования действий операторов на трех постах МНЛЗ (пост 10 АРМ1, АРМ2 и Гега), необходимо зайти на каждую машину и указать удаленную базу данных WWALMDB на сервере ТИС (10.2.19.193), пароль и UserName.

Репозиторий предоставляет данные таблицы **Events**. Строка подключения к базе данных находится в конфигурационном файле проекта под именем **sqlCCM.**

#### Информация по АКП и ВД

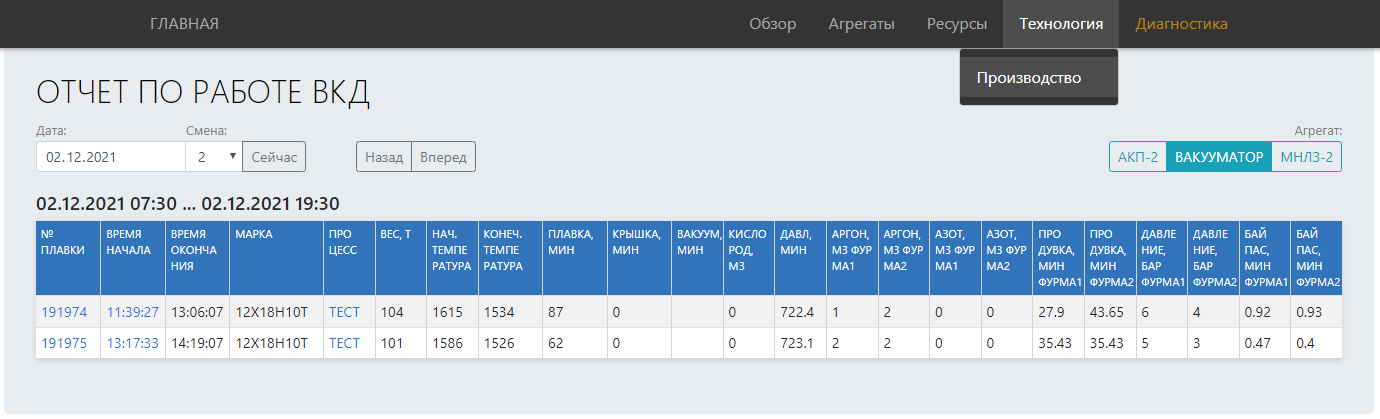
Собирается отдельно с удаленного сервера базы данных MSSQL развернутого на диагностической виртуальной машине сервера ЭСПЦ (10.2.19.36). В виртуальной среде ЭСПЦ эта машина показана на рисунке справа.

Репозиторий предоставляет данные таблицы **Events**. Строка подключения к базе данных находится в конфигурационном файле проекта под именем **sqlVOD**.

## Production-контроллер

Контроллер предоставляет точки входа для получения оперативной и исторической информации по технологическим процессам ЭСПЦ.

На рисунке выше показан репозитории, которому он делегирует работу с базами данных.

Переключаясь между агрегатами представление запрашивает данные с серверной части на одной из точек входа предоставляемых контроллером.

Репозиторий использует базы данных серверов 2 уровня агрегатов. В конфигурационном файле приложения строки подключения к ним называются **oracleLFVOD** и **oracle**.

## Screenshots-контроллер

Контроллер предназначен для работы с мгновенными данными состояний АРМ и компрессорной.

### Прием изображений на сервер

Метод **Publish** служит точкой приема потоковой информации от служб снимков экранов АРМ (замена существующей схемы доставки изображений на сервер). Получая потоковые данные на эту точку входа, контроллер сохраняет изображение в локальную папку, откуда клиентская часть отображает меняющиеся скриншоты.

В настоящий момент на сервере запущен скрипт, копирующий изображения с сервера второго уровня, являющегося общей точкой сбора скриншотов.

### Оперативные данные по компрессорной

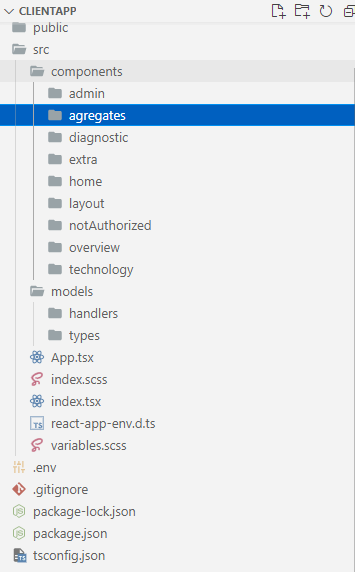
Метод **ReadCompressorAsync** предоставляет точку получения оперативных данных по компрессорной. Делегирует работу с базой данных репозиторию **CompressorRepository**, использующему строку подключения **sqlTIS** и локальный сервер MSSQL (10.2.19.193).

Мгновенные данные для компрессорной аккумулируются в таблице **tis\_runtime.ccm**

# КЛИЕНТСКАЯ ЧАСТЬ РЕШЕНИЯ

Клиентская часть приложения представляет собой набор скомпилированных скриптов, загружаемых пользователем при первом обращении к серверу. При дальнейшей работе именно этот набор скриптов осуществляет взаимодействие с серверной частью приложения (на заднем фоне, не видимом для пользователя) и вносит изменения в браузере клиента без перезагрузки страницы. Таким образом, реализуется концепция **Single page application**.

Исходные файлы клиентской части представляют из себя типовой проект фреймворка **create-react-app** с модификацией **Typescript**.

Общее назначение папок описано в главе Структура клиентской части.

Папка-контейнер для всех компонентов **src** содержит две основные подпапки, разделяющие код компонентов и объектную модель клиентской части, построение которой возможно благодаря статической типизации Typescript и современному стандарту ES2015.

Папка **types** содержит базовые типы клиентского приложения по аналогии с серверной частью, вокруг которых строится вся объектная информационная модель.

Подпапка **handlers** содержит различные общие обработчики, не работающие с графикой и UI. Эти обработчики только обрабатывают информацию, не отображая ничего.

Подпапки в папке components отражают структуру компонентов react-приложения. Структура идентична представлениям, отображаемым в браузере.

## Модель взаимодействия с серверной частью

Все взаимодействие с серверной частью осуществляется централизованно с помощью обработчиков, находящихся в папке **DbHandlers**.

Компоненты для получения данных не запрашивают необходимую информацию самостоятельно, а делегируют это соответствующему обработчику. Один обработчик работы с серверной частью может быть использован несколькими компонентами. Все обработчики построены с применением шаблона Singleton, что гарантирует достоверность полученной и переданной информации.

## Особенности css

Для того, чтобы разграничить и инкапсулировать css классы компонентов, а также для удобства восприятия подключена обработка современного стандарта работы с веб-разметкой страниц SASS. Применение этой технологии очень сильно упрощает читаемость кода и позволяет добиться максимальной защищенности классов внутри отдельных модулей одного компонента.

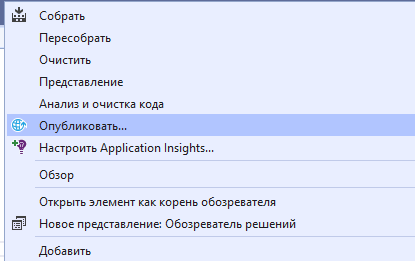
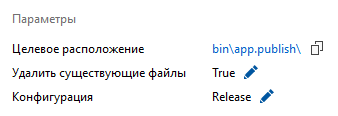
Большая часть веб-разметки выполнена с использованием современной технологии **Grid**. Поэтому основным требованием для использования приложения является наличие современного браузера, способного правильно интерпретировать современный подход к верстке.

# ПУБЛИКАЦИЯ ПРИЛОЖЕНИЯ

## Компиляция проектов

### Серверная часть

Механизм удаленной публикации через FTP удобен, если приложением никто не пользуется и скорость сети позволяет делать это комфортно. В условиях ЭСПЦ, к сожалению, от него следует отказаться ввиду очень низкой скорости сети и слабого оборудования (все сервера развернуты на виртуальных гипервизорах).

Поэтому необходимо щелкнуть правой кнопкой по названию проекта, нажать на «**Опубликовать**» и выбрать профиль «Папка».

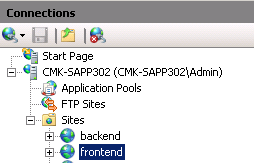
После публикации релиз серверной части будет доступен в указанной папке. Эти файлы необходимо перенести в соответствующий раздел сервера IIS на целевой машине.

### Клиентская часть

Для получения релиза клиентской части необходимо выполнить команду **npm run build**. После окончания сборки релиз доступен в папке build проекта.

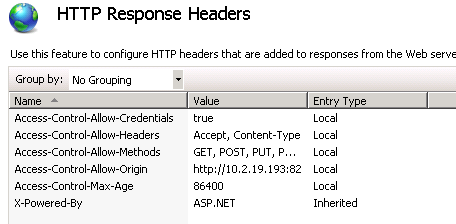
Эти файлы также необходимо скопировать на целевую машину в соответствующий раздел сервера IIS.

## Сайты на сервере IIS

После загрузки скомпилированных файлов на сервер, необходимо создать два сайта для клиентской и серверной частей приложения. Релизы необходимо разложить соответственно.

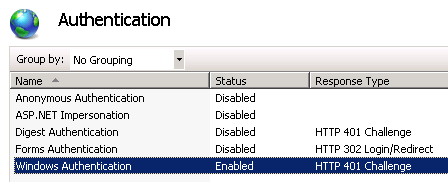
Привязки сайтов необходимо выполнить на статический адрес 10.2.19.193 и на отдельный порт каждый. Эти порты задаются в обработчиках **DbHandlers** на клиентской части и в заголовках ответа сервера на серверной части приложения.

## Заголовки ответа сервера

Важный момент в настройке серверной части приложения заключается в правильной настройке заголовков. Ввиду того, что запросы на серверную часть поступают с другого порта, включается механизм CORS. Без правильно выставленных заголовков ответа клиент просто не сможет получить ответ от сервера.

Заголовки ответа можно посмотреть на рисунке справа.

## Аутентификация

Так как приложение использует доменную авторизацию для идентификации пользователей, необходимо включить аутентификацию Windows в настройках серверной части приложения.

Клиентская часть не требует настройки аутентификации и можно оставить способ Anonymous по умолчанию.