УТВЕРЖДЕН

RU.46939656.58.29.29.000-02 13 01-ЛУ

Специализированное программное обеспечение

«Платформа управления документами и бизнес-процессами «ГосСЭД»

Описание программы

RU.46939656.58.29.29.000-02 13 01

Листов 77

2024

Подл. и дата	
Инв. № дубл.	
Взам. инв. №	
Подл. и дата	
Инв. № подл.	

АННОТАЦИЯ

Настоящий программный документ содержит описание специализированного программного обеспечения «Платформа управления документами и бизнес-процессами «ГосСЭД»» (далее – платформа).

В документе приведены общие сведения о платформе, определено функциональное назначение, описана логическая структура, указаны используемые технические средства и программное обеспечение, вызов и загрузка, входные и выходные данные.

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ

1.1. Обозначение и наименование платформы

Наименование и обозначение программного изделия – Специализированное программное обеспечение «Платформа управления документами и бизнес-процессами «ГосСЭД»».

Наименование организации-изготовителя: ООО «РТК-Цифровой документооборот».

1.2. Программное обеспечение, необходимое для функционирования платформы

1.2.1. Системное программное обеспечение

Платформа функционирует под управлением следующих операционных систем, приведенных в таблице 1.

Таблица 1 - ОС, на которых функционирует платформа

Тип объекта вычислительной техники	Требования к операционным системам платформы
Сервер приложений	Astra Linux Special Edition (версия 1.7) Astra Linux Special Edition (версия 1.8)
Сервер СУБД	Astra Linux Special Edition (версия 1.8)

1.2.2. Прикладное программное обеспечение

Платформа функционирует на базе прикладного программного обеспечения, приведенного в таблице 2.

Таблица 2 - Прикладное ПО, с помощью которого функционирует платформа

Тип объекта вычислительной техники	Требования к программному обеспечению платформы
Сервер приложений	Apache

Сервер СУБД	PostgreSQL 15 и выше

Все прикладное программное обеспечение, на базе которого эксплуатируется платформа, должно быть лицензионно чистым.

1.3. Языки программирования, на которых написана платформа

Исходные тексты платформы написаны на языках программирования C# (версия 10), JavaScript/TypeScript, SQL (PL/pgSQL). При разработке платформы использовалась среда разработки Microsoft Visual Studio 2022 (версии 17.10.1 или старше).

2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

2.1. Функциональные возможности платформы

Платформа классифицируется как «система электронного документооборота и управления бизнес-процессами» и предназначена для автоматизации управления документами и бизнес-процессами в области государственного управления и управленческой деятельности организаций, в операционных процессах деятельности ведомств(организаций), с возможностью использования для автоматизации внутриведомственного и межведомственного документооборота и контроля исполнения поручений в федеральных органах исполнительной власти и других организаций.

Платформа является прикладным программным обеспечением со встроенными средствами защиты от несанкционированного доступа к информации с ограниченным доступом, не составляющей государственную тайну.

Платформа обеспечивает следующие функциональные возможности для хранения и обработки различных типов документов:

- настраиваемый состав атрибутов и представления карточек документов;
- удобное заполнение атрибутов карточек документов с функцией автозавершения;
- обеспечение безопасности информации, включая механизмы разграничения прав доступа к объектам системы;
- поддержание связей между документами;
- организация связанных представлений;
- организация процессов обработки документов;
- атрибутивный (возможностью указания типа отношений И\ИЛИ между значениями атрибута при множественном поиске) и полнотекстовый поиск;
- сканирование документов и доведение их до исполнителей в электронном виде.

Платформа позволяет создавать решения по автоматизации следующих типовых задач документооборота:

- регистрация входящей и исходящей корреспонденции;
- автоматизации процессов электронного документооборота ориентированных на государственное управление, а также контроль исполнительской дисциплины:
- Актов Президента;
- Актов Правительства, законопроектов и планов-графиков их разработки и выпуска; Нормативных правовых актов Министерств и Ведомств;
- Документов Совещаний (повестки, протоколы, материалы);
- Поручений Президента, Председателя Правительства, Заместителей Председателя Правительства, Министров;
- Обращений Граждан;
- Архивов.

Платформа обеспечивает возможности для дальнейшего развития:

- цифровых документов;
- маршрутизации документов и комплектов документов;
- разработки календаря событий (контрольных сроков поручений\мероприятий);
- разработки интеграций информационными система ми единого информационного пространства государственного документооборота.

2.2. Возможности для интеграции

Платформа предоставляет следующие возможности для интеграции с внешними информационными системами (далее – ИС):

- использование справочников из внешних ИС без необходимости синхронизации данных;
- использование и поиск информации из других ИС;

- ссылки на объекты других ИС и виртуальные карточки (карточки с информацией, собранной из других ИС, без дублирования данных);
- отображение смешанной информации на форме карточки (карточки с информацией как из платформы, так и из других ИС);
- представление информации как виртуального файла, приложенного к карточке;
- обмен данными с другими ИС по ходу бизнес-процесса;
- веб-ссылки на объекты платформы из других ИС;
- обращения к объектам платформы через веб-сервис.

2.3. Доступ к ПО платформы

Автоматизированные рабочие места для доступа к платформе развёртываются на базе следующих программных компонент из состава ПО платформы:

- Полнофункциональный веб-клиент с использованием веб-браузера (далее – Веб-клиент).

2.4. Функции безопасности платформы

Комплекс средств защиты платформы обеспечивает следующие функции безопасности:

1) Идентификацию и аутентификацию субъектов доступа и объектов доступа (далее - ИАФ), в части:
   а) идентификации и аутентификации пользователей, являющихся работниками оператора (ИАФ.1), которые позволяют:
      - идентифицировать и аутентифицировать пользователей, являющихся работниками оператора, и сессий, запускаемых от имени этих пользователей;
      - аутентифицировать пользователя с использованием пароля;
      - однозначно сопоставлять идентификатор пользователя с запускаемой от его имени сессии;

б) управления идентификаторами, в том числе создания, присвоения, уничтожения идентификаторов (ИАФ.3), которое обеспечивает:

- определение должностного лица (роль – «администратор»), которому доступны функциональные возможности по созданию, присвоению и уничтожению идентификаторов пользователей;
- формирование идентификатора, однозначно идентифицирующего пользователя;
- присвоение идентификатора пользователю;
- блокирование идентификатора пользователя, либо на постоянной основе, либо до заданной даты и времени;
- уничтожение идентификатора с использованием статуса «Вход запрещен»;
- изменение идентификатора пользователя;

в) управления аутентификационной информацией, в том числе хранения, выдачи, инициализации, блокирования аутентификационной информации (ИАФ.4), позволяющего:

- определять отдельную роль (поле «уровень доступа» – администратор), которой доступны функциональные возможности по инициализации, блокированию аутентификационной информации в случае утраты и (или) ее компрометации;
- генерировать и выдавать начальную аутентификационную информацию;
- изменять текущую аутентификационную информацию для пользователя;
- самостоятельно изменять свой пароль пользователем с использованием управления своими настройками;
- устанавливать характеристики пароля, используемого в платформе в качестве аутентификационной информации:
  - задание минимальной сложности пароля с определяемыми

требованиями к регистру символов, количеству символов, сочетанию букв верхнего и нижнего регистра, цифр и специальных символов;

- задание минимальной длины пароля;
- задание минимального количества неповторяющихся паролей;
- задание максимального времени действия пароля (в днях, пустое поле – бессрочное действие текущего пароля);
- задание срока начала отправки уведомления о необходимости обновления аутентификационной информации (задания нового пароля) с периодичностью, установленной оператором (в днях);

- защищать аутентификационную информацию от неправомерного доступа к ней и модифицирования при хранении (хранение пароля в виде хэш-кода);

г) защиты обратной связи при вводе аутентификационной информации (ИАФ.5);

2) Управление доступом субъектов доступа к объектам доступа (далее - УПД), в части:

а) управления (заведения, активации, блокирования и уничтожения) учетными записями пользователей (УПД.1), которое позволяет:

- определять тип учетной записи (уровень доступа: «Администратор», «Обычный»);
- объединять учетные записи в группы (при необходимости);
- осуществлять заведение, производить активацию, блокировать и уничтожать учетные записи пользователей;
- корректировать параметры учетных записей пользователей;
- присваивать учётным записям пользователей подмножество ролей, определяющих возможности пользователей по доступу к объектам платформы, доступ к которым контролируется механизмами ролевого доступа;

б) реализации необходимых методов (ролевой), типов (изменение) и правил разграничения доступа (УПД.2);

в) разделения полномочий пользователей и администраторов (УПД.4);

г) ограничения неуспешных попыток входа в платформу (доступа к платформе) (УПД.6);

д) блокирования сеанса доступа в платформу после установленного времени бездействия (неактивности) пользователя (УПД.10);

3) Регистрация событий безопасности (далее - РСБ), в части:

а) сбора, записи и хранения информации о событиях безопасности (РСБ.3);

б) предоставления средств мониторинга (просмотра, анализа) результатов регистрации событий безопасности (РСБ.5);

в) защиты информации о событиях безопасности (РСБ.7);

4) Для аутентификации пользователей, а также внешних сервисов разработан сервис с поддержкой протокола OAuth 2.0.

2.5. Ограничения, накладываемые на область применения

Платформа предназначена для обработки информации, в том числе, имеющей ограниченный доступ (исключая персональные данные), и не предназначена для защиты информации ограниченного доступа, составляющей сведения государственной тайны.

3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ ПЛАТФОРМЫ

3.1. Общее описание архитектуры платформы

Платформа предназначена для построения сложных процессов обработки документов, работы в сложных условиях и специально оптимизирована для работы на слабых каналах связи.

Платформа построена по классическому принципу трехзвенной архитектуры: клиентское приложение (Веб-клиент), сервер приложений, сервер баз данных (под управлением СУБД). Пользователи запускают клиентское приложение, которое после запуска подключается к серверу приложений платформы. Сервер приложений взаимодействует с сервером базы данных, обеспечивает аутентификацию пользователей и взаимодействие с клиентскими приложениями.

3.1.1. Клиентское приложение

Клиентское приложение (Веб-клиент) предоставляют пользователю визуальный интерфейс для работы с системой, функционирующей на базе платформы.

3.1.2. Сервер приложений

Сервер приложений является центральным местом обработки информации и осуществляет всю работу, связанную с организацией и хранением информации, формированием рабочих процессов.

3.1.3. Сервер аутентификации

Сервер аутентификации позволяющий аутентифицировать пользователя по протоколу OAuth 2.0 для дальнейшей авторизации в Системе.

3.1.4. Сервер баз данных и файловое хранилище

Сервер баз данных предоставляет услуги как основное хранилище данных платформы. Файловое хранилище обеспечивает хранение файлов вне базы данных

СУБД. Сервер баз данных (под управлением СУБД) и файловое хранилище не входят в состав платформы.

Архитектура платформы основывается на централизованной модели управления. Эта модель подразумевает применение общих правил обработки информации.

3.1.5. Сервер индексации и поиска

Сервер индексации и поиска представляет инструменты индексации хранимой информации для последующего поиска по включению или полнотекстового поиска в документах.

3.1.6. Сервер кеширования

Сервер кеширования оптимизирует работу с данными, требующими оперативного доступа.

3.1.7. Сервер файлового хранилища

Сервер файлового хранилища, построен на основе систем поддерживающих работу с большими массивами данных по протоколу S3. Позволяет оптимизировать доступ к файлам, разграничить права доступа к файлам по необходимости и обеспечить отказоустойчивость хранилища в случае отказа части серверов или дисков.

3.1.8. Сервер очередей сообщений

Сервер очередей сообщений, предоставляет интерфейс для упорядочивания работы между несвязанными подсистемами и микросервисами.

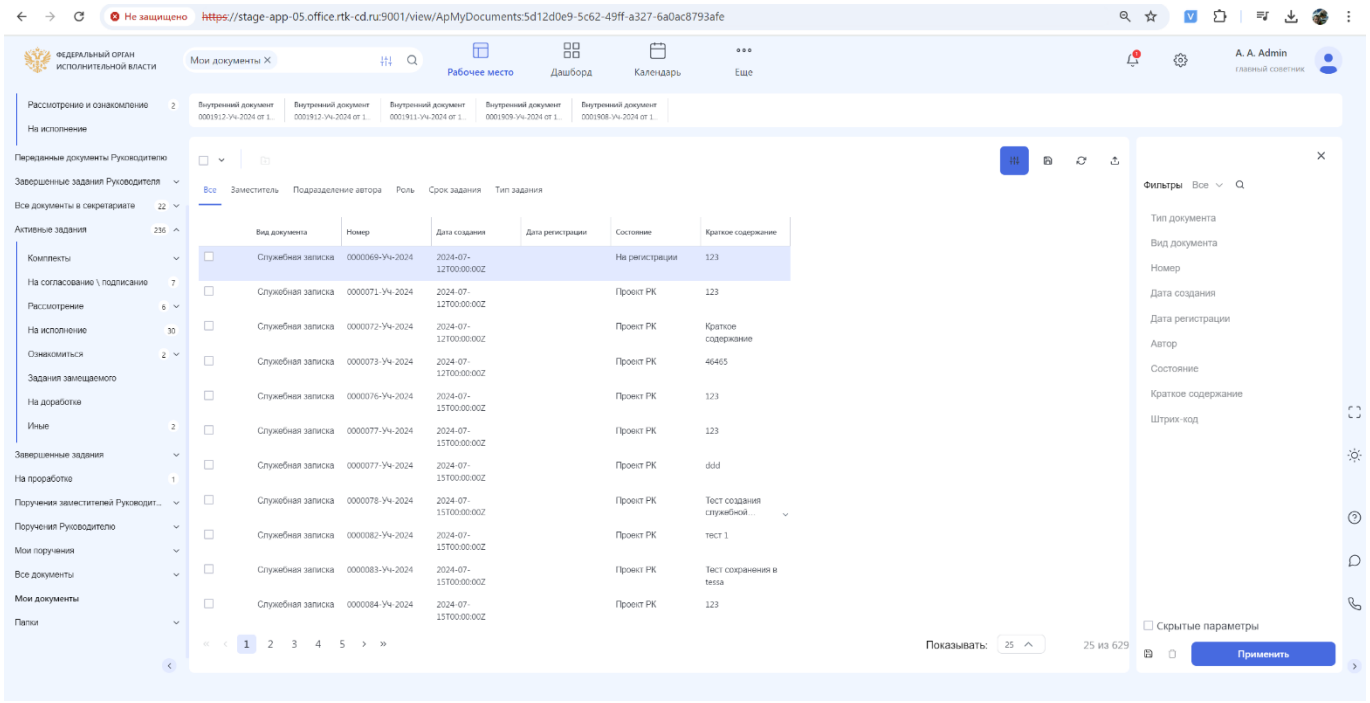
3.1.9. Подсистема обеспечения безопасности информации

Подсистема обеспечения безопасности платформы предназначена для обработки информации, в том числе, имеющей ограниченный доступ (исключая персональные данные) и не предназначена для защиты информации ограниченного доступа, составляющей сведения государственной тайны. Реализует подмножество

мер защиты информации в рамках функциональных блоков, описание которых приведено в разделе 2.4.

3.1.10. Визуальный интерфейс

Визуальный интерфейс клиентского приложения (веб-клиент) предоставляет гибкие возможности для отображения и обработки информации. Основными элементами интерфейса являются: левое меню (дерево узлов-представлений), меню обрабатываемой карточки, меню представлений. Для каждого представления есть возможность настройки\разработки поиска по атрибутам\контексту документов.



3.1.11. Свойства типовой карточки

Основные свойства карточки:

– Имя – уникальное имя типа карточки. Оно же определяет имя узла в редакторе, который соответствует типу. Имя может быть использовано в расширениях, а также при настройке решения, но обычный пользователь его не будет видеть;

– Заголовок – заголовок типа карточки, т.е. его отображаемое для пользователей название. Значение может быть неуникальным, оно

фигурирует при создании карточки, в заголовке вкладки для открытой карточки и в различных представлениях (например, в представлении со списком шаблонов, доступных пользователю);

- Группа – название группы типов карточек. Оно требуется не только для группировки в редакторе, но и для группировки плиток при создании карточки, когда пользователю доступно для создания очень много карточек различных типов, и эти типы группируются в соответствии с этим полем. Причём пользователь по умолчанию видит именно название группы, но при разработке можно написать простое расширение, которое для заданного имени группы подставит другую локализованную строку. Именно таким образом локализуются все группы для стандартных типов карточек;
- Формат дайджеста – определяет строку форматирования для дайджеста карточки, т.е. для названия, используемого в заголовке вкладки, истории действий и т.д. Для использования полей карточки доступна стандартная система плейсхолдеров (см. Руководство администратора платформы). Наведя курсор на поле можно посмотреть всплывающую подсказку с примером. Если поле не заполнено - вычисление дайджеста выполняется стандартным образом (например, это номер карточки, если она включена в типовое решение и в ней присутствует секция с номером);
- Идентификатор типа – идентификатор типа карточки. С помощью кнопок, расположенных справа от поля можно скопировать идентификатор типа или скопировать идентификатор типа и имя для использования в расширениях;
- Флажок «Административный» указывает, что тип карточки предназначен только для использования администратором. Например, флажок ставится для карточек ролей, правил доступа и настроек. Система гарантирует, что пользователь, не являющийся администратором, не сможет создать карточку, изменить её или удалить, но сможет открыть в режиме только для чтения;
- Флажок «Скрытый» позволяет скрыть тип карточки из списка на создание, т.е. ни один пользователь или администратор не сможет явно создать

карточку стандартным образом. Флажок обычно ставится для вспомогательных карточек (карточек-сателлитов), которые создаются расширениями (например, это карточка-сателлит типового процесса согласования с информацией по этапам согласования);

- Флажок «Разрешить задания» разрешает использование заданий для карточек этого типа. Включение этой настройки не добавляет автоматически команды для создания заданий в интерфейсе, но оно разблокирует системную вкладку История заданий и позволяет добавить такие команды, как начало процесса согласования или постановка задачи, в настройках типового решения (что будет рассмотрено ниже). Для таких типов карточек, как карточки ролей, карточки настроек и карточки-сателлиты, задания не требуются, поэтому эту настройку отключают;
- Флажок «Фиксировать действия» активирует ведение логов аудита при любом действии с карточкой (например, при создании, открытии, изменении, удалении, импорте, экспорте и др.);
- Флажок «Удалять в корзину» активирует возможность по удалению карточки в корзину с возможностью восстановления. Если пользователь случайно удалит карточку, то в течение нескольких дней администратор сможет восстановить карточку (по умолчанию 30 дней, срок настраивается). Удалённая карточка физически удаляется из всех таблиц, поэтому она также исчезает из всех отчётов;
- Флажок «Единственный экземпляр» определяет, что только один экземпляр карточки может быть создан. Обычно эта опция активируется для всех карточек настроек. Такую карточку можно открыть, не зная её идентификатора, но зная имя типа карточки или идентификатор типа. Также для таких карточек активируется кэширование на сервере и на клиентском приложении (веб-клиент), это значительно ускоряет доступ к данным карточки, что также актуально для карточек настроек;

– Флажок «Загружать при инициализации» – признак того, что карточка, существующая в единственном экземпляре, будет загружена и добавлена в кэш на клиентском приложении (веб-клиент) в процессе загрузки клиента.

3.2. Алгоритмы программы

Алгоритмы программы представляют собой механизмы ввода, хранения, сравнения и визуализации данных.

3.2.1. Алгоритмы идентификации и регистрации

Алгоритмы идентификации и регистрации описывают механизмы идентификации и аутентификации пользователя в платформе.

3.2.1.1. Создание идентификатора (ввод пользователя в систему)

Создание и присвоение идентификатора выполняется посредством создания карточки работника, которое выполняется администратором системы с помощью алгоритма, блок-схема которого приведена на рисунке 1. Администратор вводит логин и пароль работника, и сохраняет карточку.

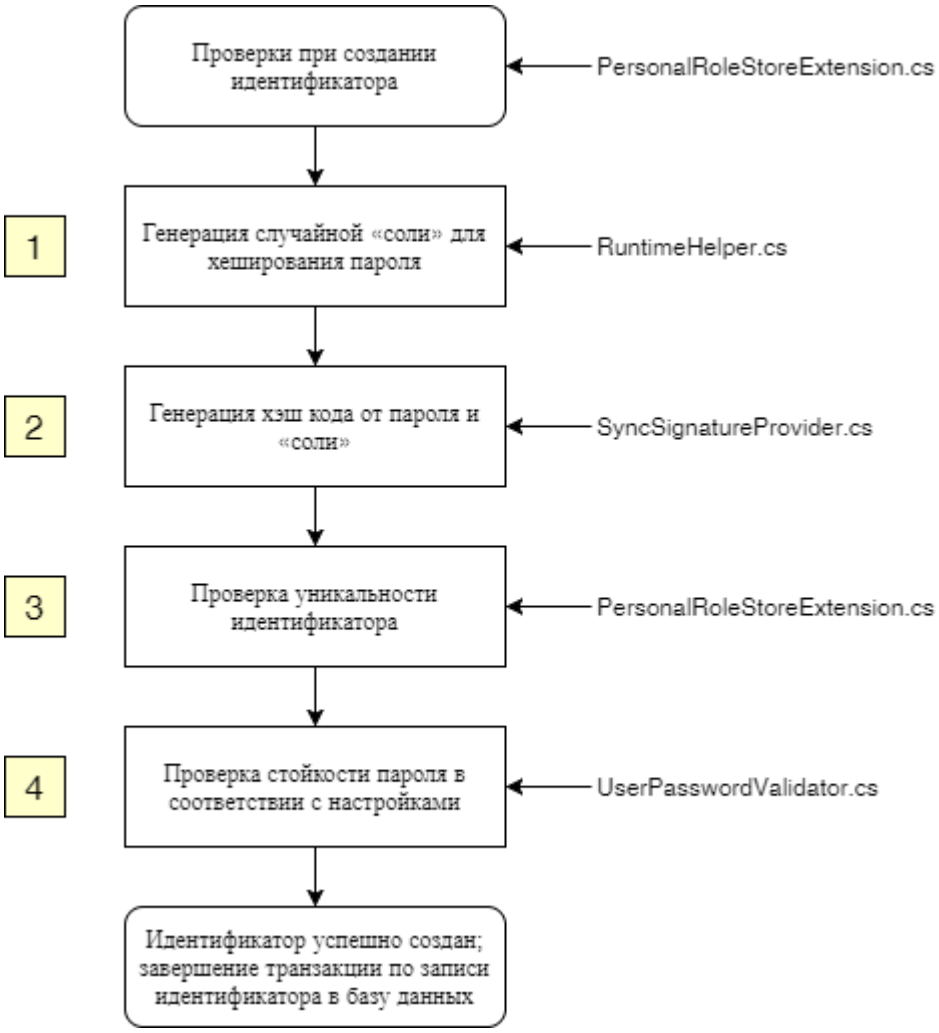


Рисунок 1 – Алгоритм создания идентификатора при создании работника.

Блок 1. Генерация случайной «соли» для хеширования пароля

Генерируется набор из случайно сгенерированных 64 байт, при этом задействуется стандартный генератор псевдослучайных чисел Random в .NET. Такой набор байт – «соль» - будет использоваться при вычислении криптостойкого хэш-кода от пароля, который будет сохранён в базе данных и будет в дальнейшем проверяться.

В базе данных не хранится сам пароль (его нельзя узнать), и за счёт использования «соли» для одного и того же пароля, используемого в разные моменты разными пользователями, сгенерированный хэш код будет разный, и по нему нельзя будет определить, были пароли одинаковыми или нет.

Генерация выполняется в классе RuntimeHelper, метод GenerateSignature (файл …\Platform\Runtime\RuntimeHelper.cs).

Блок 2. Генерация хэш кода от пароля и «соли»

При сохранении карточки «Работник» выполняется серверное расширение в классе PersonalRoleStoreExtension (файл …\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs). Перед сохранением в методе BeforeRequest этого класса выполняется генерация хэш кода от пароля с использованием соли, полученной ранее.

Метод GetPasswordBytesToSign в классе RuntimeHelper возвращает побайтовое представление строки с паролем в кодировке UTF-8.

Используя «соль», полученную на предыдущем шаге в методе GenerateSignature, создаётся объект криптопровайдера (интерфейс ISignatureProvider, экземпляр класса SyncSignatureProvider, файл …\Platform\SyncSignatureProvider.cs).

Объект криптопровайдера генерирует значение хэша по криптостойкому алгоритму SHA256, для которого в качестве ключа передаются байты сгенерированной «соли» (passwordKey). Подписание выполняется в методе Sign класса SyncSignatureProvider. Реализация алгоритма SHA256 сделана самостоятельно и не использует реализацию из классов .NET Core.

Значения хэш-кода пароля passwordKey и «соли» passwordHash, сохраняется в секции карточки fields, поэтому они будут сохранены в базу данных вместе с карточкой после успешного выполнения других проверок.

Блок 3. Проверка уникальности идентификатора.

Перед началом сохранения в базу данных выполняется проверка того, что введённый администратором логин является уникальным, т.е. в системе не существует других работников с тем же логином. Проверка выполняется в классе PersonalRoleStoreExtension, в методе AfterBeginTransaction.

При этом выполняется SQL-запрос, возвращающий имя работника с тем же логином, что и у создаваемого работника, или NULL, если такого пользователя нет. Результат запроса записывается в переменную userNameWithSameLogin. Тот же код проверки выполняется в дальнейшем при изменении логина, поэтому в запрос также передаётся идентификатор текущей карточки card.ID в параметре запроса «ID». При создании работника это новый сгенерированный идентификатор, которого нет в базе

данных, а при изменении карточки работника, это будет её идентификатор, т.е. система не будет сообщать о неуникальности логина, если он был изменён на такое же значение, какое уже есть в базе данных у этого же работника (например, дописали символ, тут же стёрли его и сохранили карточку работника).

Блок 4. Проверка стойкости пароля в соответствии с настройками.

Администратор системы в соответствии с принятыми в организации требованиями безопасности, может указать различные условия, которым должен удовлетворять пароль.

При создании работника выполняется проверка соответствия введённого пароля таким требованиям. Также этот код выполняется при изменении пароля у работника, в т.ч. при таком изменении, которое выполняет сам работник.

В классе PersonalRoleStoreExtension в методе BeforeCommitTransaction выполняется проверка пароля.

Метод userSecurityProvider.GetSecurityObject создаёт новый объект (поскольку карточка работника новая и ещё не существует), который содержит информацию по последним использованным паролям по парам «хэш-код + соль» для того, чтобы при последующих изменениях пароля можно было проверить настройку «Количество неповторяющихся паролей» и сравнить новый введённый пароль с предыдущими. Для созданного работника это его первый пароль, поэтому его хэш-код passwordHash и «соль» passwordKey добавляются в списки PreviousPasswordHashList и PreviousPasswordKeyList соответственно.

Далее вызывается метод проверки сложности пароля userPasswordValidator.Validate, в который передаются строка с паролем password, объект SecurityObject obj и настройки безопасности options (которые соответствуют настройкам на рисунке **Ошибка! Источник ссылки не найден.**). Вызывается класс UserPasswordValidator, метод Validate (файл …\Platform\Runtime\UserPasswordValidator.cs).

В методе в первую очередь проверяется длина пароля password.Length < options.MinPasswordLength. Если пароль короче, чем указано в настройке, то

возвращается объект с сообщением об ошибке ValidationResult, который будет отображён на клиентском приложении (веб-клиент) в виде диалогового окна.

Далее при активированной настройке options.EnforceStrongPasswords («Специальные символы, цифры и разные регистры символов» на рисунке **Ошибка! Источник ссылки не найден.**) проверяется сложность пароля в методе PasswordIsStrong этого класса. Если метод вернул false, то пароль является слабым, о чём сообщается в сообщении об ошибке.

Для того чтобы пароль считался «сильным», требуется наличие хотя бы одного символа из каждой категории символов Unicode: цифры (IsDigit), буквы нижнего регистра (IsLower), буквы верхнего регистра (IsUpper), а также хотя бы один символ в одной из категорий: пунктуация (IsPunctuation) или специальный символ (IsSymbol, такой как знак валюты, диез и др.).

Затем проверяется уникальность пароля относительно предыдущих введённых паролей. Совпадение пароля означает совпадение хэш кода для сохранённой ранее «соли» в списках PreviousPasswordHashList и PreviousPasswordKeyList для объекта UserSecurityObject.

Поскольку выполняется создание работника, то в объекте UserSecurityObject списки предыдущих паролей будут пустыми, поэтому эта проверка выполнится успешно и вернёт null. Если бы пароль изменялся для уже существующего работника, то такая проверка определила бы уникальность паролей.

Если пароль удовлетворил всем проверкам выше, то метод Validate возвращает успешный результат ValidationResult.Empty, и алгоритм сохранения продолжается.

В завершающей части алгоритма информация по предыдущим паролям сохраняется в базу данных вызовом userSecurityProvider.StoreSecurityObject. Сохранение выполняется в поле карточки работника SQL-запросом UPDATE. Это метод StoreSecurityObject класса UserSecurityProvider (файл …\Platform\Runtime\UserSecurityProvider.cs).

При сохранении выполняется сериализация свойств объекта в формате бинарного JSON, массив байт сохраняется в бинарное поле Security в таблице PersonalRoles по ID работника.

На этом завершаются проверки, связанные с созданием идентификатора. Если карточка работника удовлетворяет всем прочим проверкам, связанным с бизнес-логикой (например, поля с полным и кратким именем работника непустые), то система завершает транзакцию и работник считается успешно созданным. После этого может быть выполнен вход в систему по указанному идентификатору.

3.2.1.2. Аутентификация пользователя в системе

Аутентификация пользователя в системе начинается со страницы аутентификации.

Визуальный интерфейс окна аутентификации (Рисунок 2) содержит следующие элементы:

- Поле для заполнения «Логин».
- Поле для заполнения «Пароль».
- Кнопка «Войти».

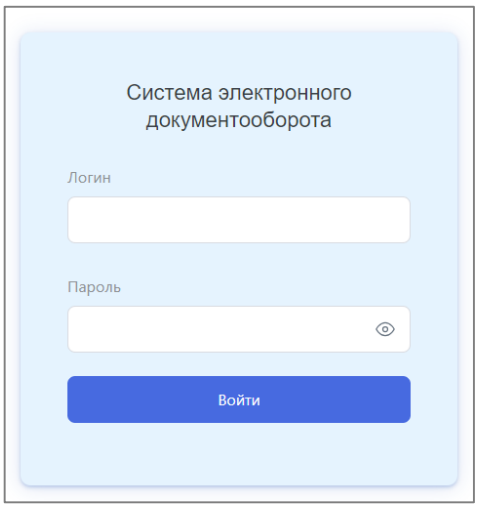


Рисунок 2 Окно аутентификации

Аутентификация пользователя и получение новых токенов «Доступа» и «Обновления» представлено на схеме ниже (Рисунок 3**Ошибка! Источник ссылки не найден.**).

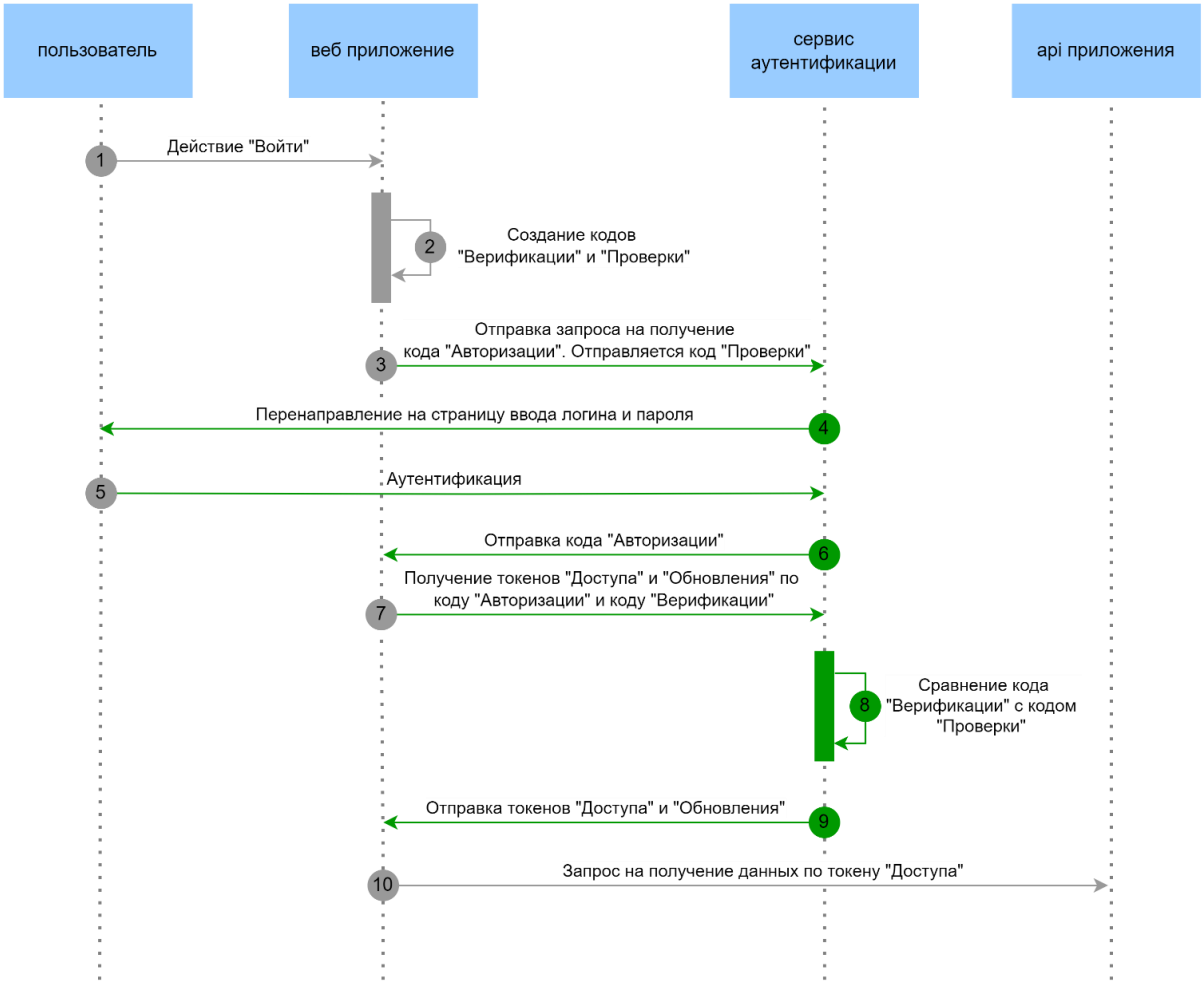


Рисунок 3 Аутентификация пользователя. Получение новых токенов «Доступа» и «Обновления»

Описание схемы (см. **Ошибка! Источник ссылки не найден.**):

5) Отправка запроса на получение кода «Авторизации»:

Выполнение GET запроса /connect/authorize, с параметрами, передаваемыми в строке:

- response_type – авторизационный flow;
- client_id – идентификатор клиента, в рамках которого происходит авторизация;
- code_challenge – хеш от code_verifier (сгенерированная строка в рамках процесса получения токена) с использоанием алгоритма, указанного в параметре code_challenge_method приведенного к виду base64UrlEndode;
- code_challenge_method – алгоритм используемый для подсчета параметра code_challenge;

– redirect_uri – адрес, на который будет перенаправлен клиент после успешной авторизации;
– scope – список скопов, через пробел, на которые пользователь хочет получить доступ.

Пример заполнения параметров:

```
response_type = code
client_id = admin
code_challenge = WZRHGrsBESr8wYFZ9sx0tPURuZgG2lmzyvWpwXPKz8U
code_challenge_method = S256
redirect_uri = https://oidcdebugger.com/debug
scope = offline_access docs_api
```

6) Перенаправление на страницу ввода логина и пароля.

Сервер отвечает http кодом 302. В заголовке ответа в поле Location находится адрес страницы аутентификации.

Пример: «https://localhost:9001/auth?ReturnUrl=https://localhost:7276/connect/authorize».

Далее пользователь перенаправляется на страницу ввода логина и пароля. (файл: apps/app_auth/src/app/api/axios.ts, метод instance.interceptors.response.use(участок кода - const isRedirectUrl = !!config.request.responseURL.match(ReturnUrlExp)).

7) Аутентификация.

– [Веб-клиент].

Пользователь на форме аутентификации (файл - app_auth/src/features/widgets/LoginForm/container/index.tsx) вводит логин и пароль. После нажатия на кнопку «Войти» срабатывает функция onClickLoginHandler, которая вызывает POST метод /connect/authenticate.

– [API сервиса аутентификации].

Получение запроса POST /connect/authenticate с Content-Type: application/x-www-form-urlencoded и параметрами в теле запроса:

– username – логин пользователя;
– password – пароль пользователя.

Пример заполнения параметров:

– username – admin;
– password – admin.

8) Отправка кода «Авторизации».

Запрос, аналогичный пунту 3, но пользователь аутентифицирован и результатом выполнения является ответ с http кодом 302. В заголовке ответа в поле Location находится адрес страницы, на которую необходимо вернуться с параметрами:

– code – код авторизации;
– iss – эмитент кода авторизации.

Пример значения в параметре Location заголовка «https://oidcdebugger.com/debug?code=nMHgTYqRiN1Kad6YnhupjuLKCPYdpgRKxjDoQ9eqLvs&iss=https%3A%2F%2Flocalhost%3A5001%2F».

9) Получение токенов «Доступа» и «Обновления»,

Выполнение POST запроса /connect/token authenticate (файл – AuthorizationController.cs, контроллер AuthorizationController, вызываемый метод Exchange) с Content-Type: application/x-www-form-urlencoded и параметрами:

– client_id – идентификатор клиента, в рамках которого происходит авторизация;
– client_secret – секрет клиента, в рамках которого происходит авторизация;
– grant_type – флоу авторизации;
– code – код авторизации;
– code_verifier – оригинальная строка использованная для подсчета code_challenge;
– redirect_uri – адрес, куда будет перенаправлен клиент после успешной авторизации.

Пример:

– client_id – admin;
– client_secret – admin;
– grant_type – authorization_code;

– code – 2r_tXMq4j4dvHezpbYptrTo9-sYET-bQtXpqLY1-QPw;
– code_verifier – 12345;
– redirect_uri – https://oidcdebugger.com/debug.

10) Верификация.

Производиться сравнение параметров «code_verifier» и «code_challenge». Проверка релизована внутри OpenIdDict.

11) Отправка токенов «Доступа» и «Обновления».

Получаем ответ с параметрами:

```
access_token – токен доступа
token_type – тип токена
expires_in – время жизни токена
scope – скопы доступные с помощью токена
refresh_token – токен обновления
```

3.2.2. Алгоритмы работы механизма регистрации событий безопасности

Данные алгоритмы описывают механизм регистраций событий безопасности в платформе.

3.2.2.1. Регистрация события входа в систему

При успешной регистрации пользователя в системе в методе OpenSession (описанной в алгоритме «Регистрация пользователя в системе») выполняется запись информации по событию входа в журнал аудита. Регистрация выполняется из метода AddToActionHistorySafe класса SessionServer (файл …\Platform\Runtime\SessionServer.cs).

Журнал аудита доступен администратору системы на вкладке «Администратор», узел дерева «История действий». В табличной форме доступны колонки «Карточка», «Тип», «Действие», «Дата и время», «Работник». Пример журнала аудита представлен на рисунке 5.

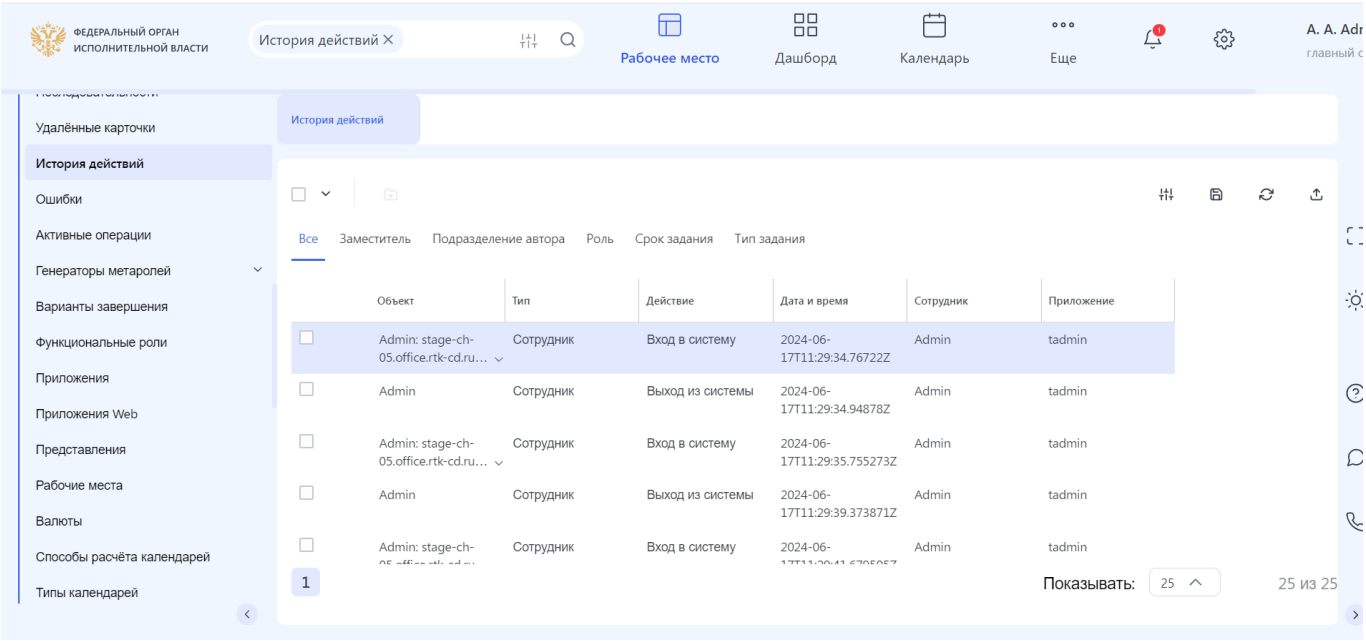


Рисунок 4 Представление журнала аудита в интерфейсе администратора

Опишем, как эта информация регистрируется в системе, на основании алгоритма, блок-схема которого приведена на рисунке 5.

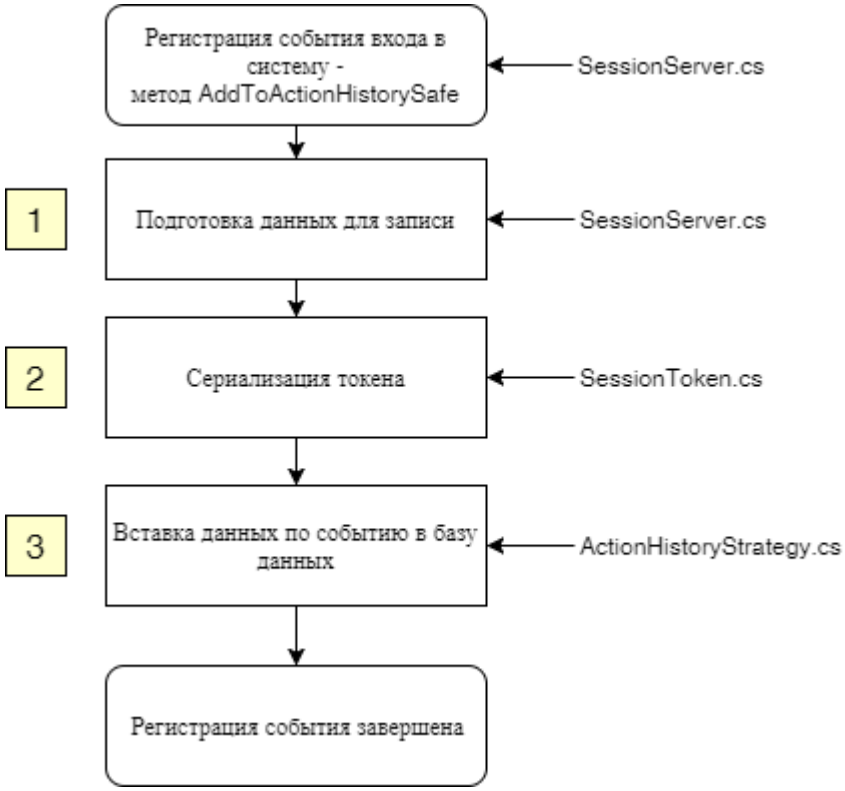


Рисунок 5 - Алгоритм регистрации события входа в систему

Блок 1. Подготовка данных для записи.

При регистрации события входа метод вызывается нужным образом (из метода OpenSession).

На входе передаётся следующая информация:

- В качестве типа действия указывается «Login» («Вход в систему»);
- userID – уникальный идентификатор карточки работника, которая найдена для идентификатора, введённого пользователем при входе;
- userName – сокращённое имя работника на момент входа;
- hostIP – IP-адрес компьютера, с которого выполняет вход пользователь;
- hostName – DNS-имя компьютера, с котором выполняет вход пользователь;
- utcNow – текущее время в формате UTC (часовой пояс UTC+00), которое определяет дату/время входа пользователя в систему;
- sessionID – уникальный идентификатор сессии, сгенерированный при входе в методе OpenSession;
- token – объект токена сессии с полной информацией по сеансу работника, который был создан в методе OpenSession (алгоритм «Регистрация пользователя в системе»).

Для определения строки, которая записывается в колонку «Карточка» журнала аудита (см. рисунок **Ошибка! Источник ссылки не найден.**), метод GetAddressName того же класса SessionServer объединяет hostIP и hostName, после чего слева дописывается «userName: …».

Если DNS-имя hostName не удалось определить, то указывается только IP-адрес. Если IP-адрес не удалось определить, то выводится константа «адрес неизвестен».

Блок 2. Сериализация токена

Перечисленная информация будет записана в таблицу «ActionHistory» в следующем блоке алгоритма. В таблице есть колонка Request, содержащая массив байт, в который была сериализована информация по записи в журнале в формате бинарного JSON. При регистрации события входа в эту колонку будет сериализован токен сессии SessionToken, который был сформирован в методе OpenSession.

Собственно, сериализация выполняется в методе SerializeToStorageCore класса SessionToken (файл …\Platform\Runtime\SessionToken.cs), этот метод вызывается из метода SerializeToStorage цепочкой тривиальных вызовов.

Следующая информация будет записана в поле Request, откуда десериализована и выведена в интерфейсе системы (см. рисунок **Ошибка! Источник ссылки не найден.**):

- ApplicationID – идентификатор приложения; возможные приложения и их идентификаторы задаются в таблице «ApplicationNames»;
- LicenseType – тип лицензии (конкурентная, персональная, лицензия не занята – в случае интеграционных связей);
- AccessLevel – уровень доступа (роль работника «Пользователь» или «Администратор»);
- DeviceType – идентификатор типа устройства (например, ПК, планшет или смартфон, возможные типы устройств перечислены в таблице «DeviceTypes»);
- UserID – уникальный идентификатор карточки работника, которая соответствует текущему пользователю;
- UserName – сокращённое имя пользователя на момент входа (например, «Иванов И.И.»);
- UserLogin – идентификатор, введённый пользователем при входе;
- HostIP – IP-адрес компьютера пользователя (может быть строкой «адрес неизвестен»);
- HostName – DNS-имя компьютера пользователя (может быть пустым);
- OSName – имя операционной системы, установленной на компьютер пользователя, передаётся в метод OpenSession со стороны клиентского приложения (веб-клиент), (может быть пустым);
- UserAgent – строка «User Agent», через которую браузер идентифицирует себя, свою версию и список поддерживаемых механизмов. Поле заполняется при входе через Веб-клиент;

- Created – дата/время входа в систему в формате UTC;
- Expires – дата/время истечения токена, после которого сессию необходимо будет открыть повторно. В текущей версии системы – это время равно 7 астрономическим дням. По истечении срока клиентскому приложению (веб-клиент) возвращается специальная ошибка в методе OpenSession. Веб-клиент выполняет переход на страницу входа, где пользователь вводит свой идентификатор и пароль;
- Culture – идентификатор языка для вывода чисел и денежных знаков (LCID-идентификатор, уникальный для каждого языка);
- UICulture – идентификатор языка для локализации текста в приложении, который видит пользователь, т.е. язык интерфейса (LCID-идентификатор, уникальный для каждого языка);
- UtcOffsetMinutes – смещение часового пояса, которое сообщило клиентское приложение (веб-клиент) в параметрах метода OpenSession (от часового пояса пользователя до часового пояса UTC+0).

Блок 3. Вставка данных по событию в базу данных

Подготовленные данные вставляются в базу данных как одна строка таблицы «ActionHistory». Для этого сначала вызывается метод AddToActionHistorySafe класса SessionServer.

Этот метод вызывает метод Insert класса ActionHistoryStrategy (файл …\Platform\Runtime\ActionHistoryStrategy.cs). Методу передаются следующие параметры:

- actionType – тип действия в журнал аудита «Вход в систему»;
- cardID –идентификатор карточки, с которым связано действий – это уникальный идентификатор карточки работника UserID;
- cardTypeID – это идентификатор типа карточки, с которым связано действие, здесь передаётся константа RoleHelper.PersonalRoleTypeID, которая соответствует идентификатору типа карточки «Работник»;

- cardTypeCaption – это отображаемое название типа карточки, выводимое в представлении на рисунке **Ошибка! Источник ссылки не найден.**, здесь передаётся константа RoleHelper.PersonalRoleTypeID, которая соответствует идентификатору типа карточки «Работник»;
- digest – название карточки, здесь это комбинация полей UserName, HostIP и HostName;
- request – сериализуемый объект, описывающий действие произвольным образом, которое затем выводится в поле «Описание изменений» в записи журнала аудита (см. рисунок **Ошибка! Источник ссылки не найден.**). Здесь это сериализованный токен сессии SessionToken;
- user – объект, определяющий уникальный идентификатор и сокращённое имя работника, от имени которого было выполнено действие; здесь это работник, для которого выполнялся вход, т.е. UserID и UserName;
- modified – дата/время выполнения действия в журнале, здесь это текущие дата/время в формате UTC;
- sessionID – уникальный идентификатор сессии (сеанса работы пользователя с приложением), с которым связывается запись в журнале аудита, здесь это идентификатор созданной сессии (см. рисунок **Ошибка! Источник ссылки не найден.**);
- rowID – уникальный идентификатор записи в журнале аудита; здесь передаётся null, что соответствует генерации нового уникального идентификатора.

Метод сериализует переданный объект в массив байт requestBytes в формате бинарного JSON, после чего выполняет запрос INSERT в базу данных, передавая все те же параметры, которые были описаны выше. Метод возвращает идентификатор добавленной строки, который в случае регистрации события входа в систему в дальнейшем не используется.

3.2.2.2. Регистрация события выхода из системы

Событие выхода из системы определяет вызов серверного метода CloseSession из класса SessionServer (файл …\Platform\Runtime\SessionServer.cs).

Алгоритм регистрации события выхода из системы описан на рисунке 6.

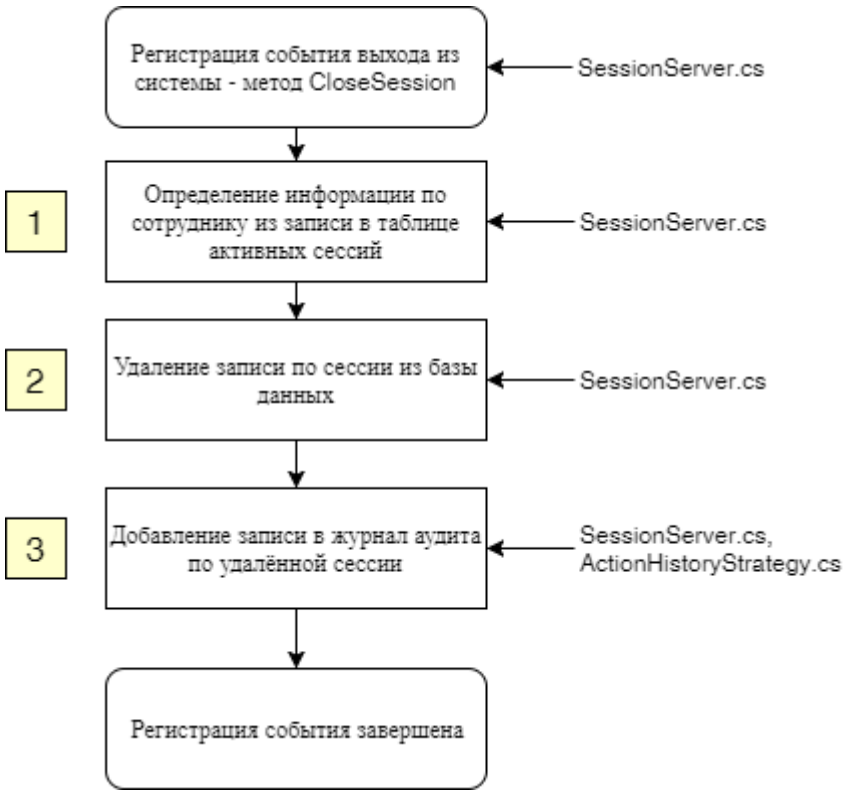


Рисунок 6 – Алгоритм регистрации события выхода из системы

Метод получает уникальный идентификатор сессии, который клиентское приложение (веб-клиент) передаёт на сервер, получив его из токена сессии (объект, созданный методом OpenSession при регистрации пользователя в системе).

Также метод получает параметр IUser closedByAdmin – уникальный идентификатор и краткое имя работника-администратора, который закрывает сессию, отличную от своей, или null, если сессию закрывает тот же работник, который её открыл. Указать значение, отличное от null, может только пользователь в роли «Администратор».

Блок 1. Определение информации по работнику из записи в таблице активных сессий.

В первую очередь метод CloseSession вызывает SQL-запрос SELECT к таблице с активными сессиями «Sessions» по переданному в метод уникальному

идентификатору сессии в колонке «ID». Если в таблице нет строки с таким идентификатором, то метод CloseSession прекращает свою работу и возвращает false.

Если строка есть, то в поле userID записывается уникальный идентификатор работника, который открыл сессию, а в поле userName – сокращённое имя работника на момент открытия сессии.

Блок 2. Удаление записи по сессии из базы данных.

Далее выполняется вызов метода RemoveSession в этом же классе SessionServer.

Метод удаляет строку из таблицы «Sessions» по тому же идентификатору закрываемой сессии sessionID.

Блок 3. Добавление записи в журнал аудита по удалённой сессии.

Если указан параметр closedByAdmin, то в качестве типа действия передаётся «Закрытие сессии администратором», т.е. принудительное завершение работы работника или удалении записи по активной сессии, которая не была удалена штатным образом из-за ошибок.

Если параметр closedByAdmin передан как null, то в качестве типа действия передаётся «Выход из системы», т.е. штатное закрытие сеанса пользователя. Тип действия доступен в колонке «Действие» в представлении журнала аудита (см. рисунок **Ошибка! Источник ссылки не найден.**).

Также передаются параметры:

- userID – уникальный идентификатор карточки работника, который был записан в сессии, т.е. который выполнял вход в систему в момент создания сессии;
- userName – сокращённое имя работника на момент входа;
- DateTime.UtcNow – дата/время закрытия сессии, это текущее время в формате UTC на момент вызова метода CloseSession;
- closedByAdmin – это работник, от имени которого будет записано действие (т.е. это администратор при закрытии сессии средствами администратора), или null, здесь записывается пользователь в сессии, что аналогично передаче объекта new User(userID, userName);
- sessionID – уникальный идентификатор сессии, которая закрывается;

– поле request не передаётся, что аналогично передаче «null», поэтому дополнительная информация не будет выведена в поле «Описание изменений»;
– поле digest не передаётся, поэтому в качестве имени карточки в записи журнала аудита указывается имя работника (без IP-адреса или DNS-имени).

Журнал для события «Выход из системы» выглядит, как показано на рисунке 7.

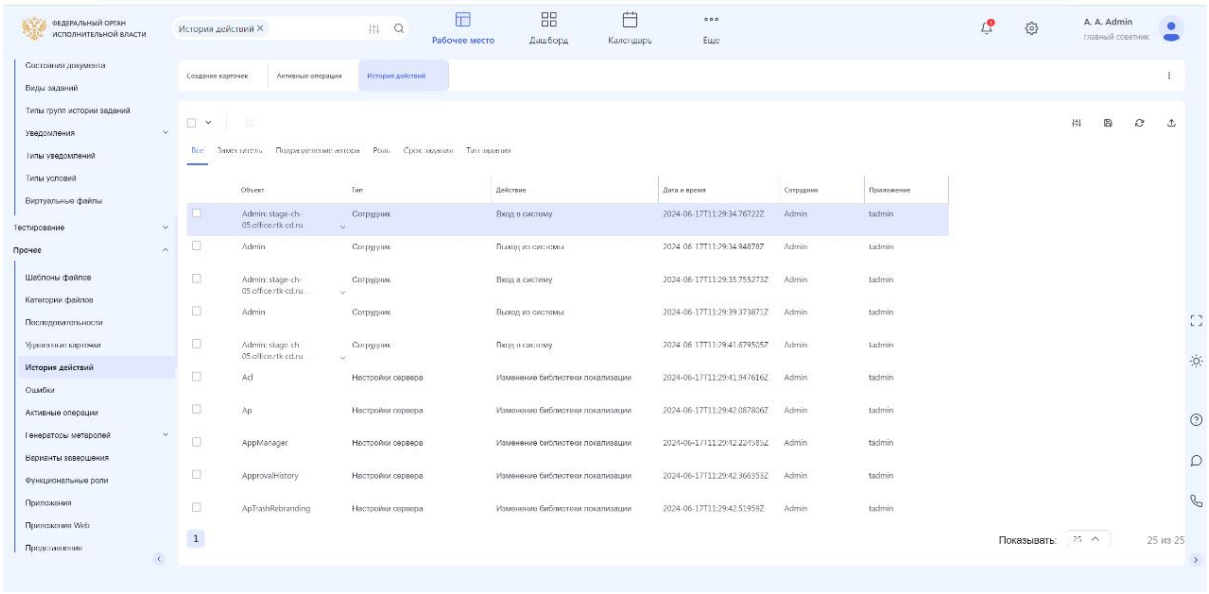


Рисунок 7 - Журнал событий для записи с регистрацией события выхода из системы

По двойному клику открывается запись журнала события, где не отображается «Описание изменений», как было сказано выше, но дополнительно указывается идентификатор сессии (см. рисунок 7).

3.2.3. Алгоритмы работы механизма ролевого разграничения доступа

Данные алгоритмы описывают механизмы ролевого разграничения доступа к функциям управления платформы.

3.2.3.1. Разграничение ролевого доступа к представлениям

Если работник входит в роль «Администратор», то он может получить доступ к данным любых представлений, которые настроены в системе. Если работник не входит эту роль, то ему доступны данные только тех представлений, в которых указан список групп (карточек ролей), к хотя бы одной из которых принадлежит работник.

Список групп для представлений настраивается в справочнике «Представления» на вкладке «Администратор» как показано на рисунке 8).

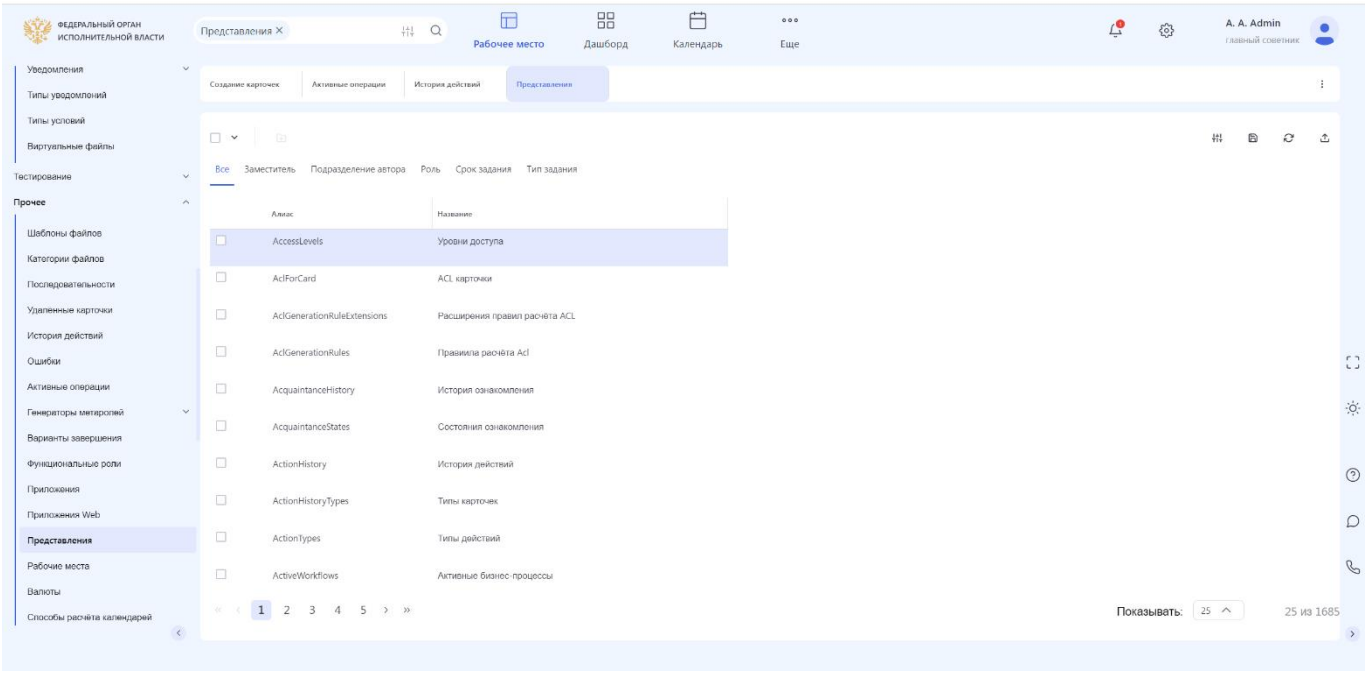


Рисунок 8 – Справочник представлений, предоставляющий настройку доступа

Описание алгоритма разграничения доступа представлено на рисунке 9.

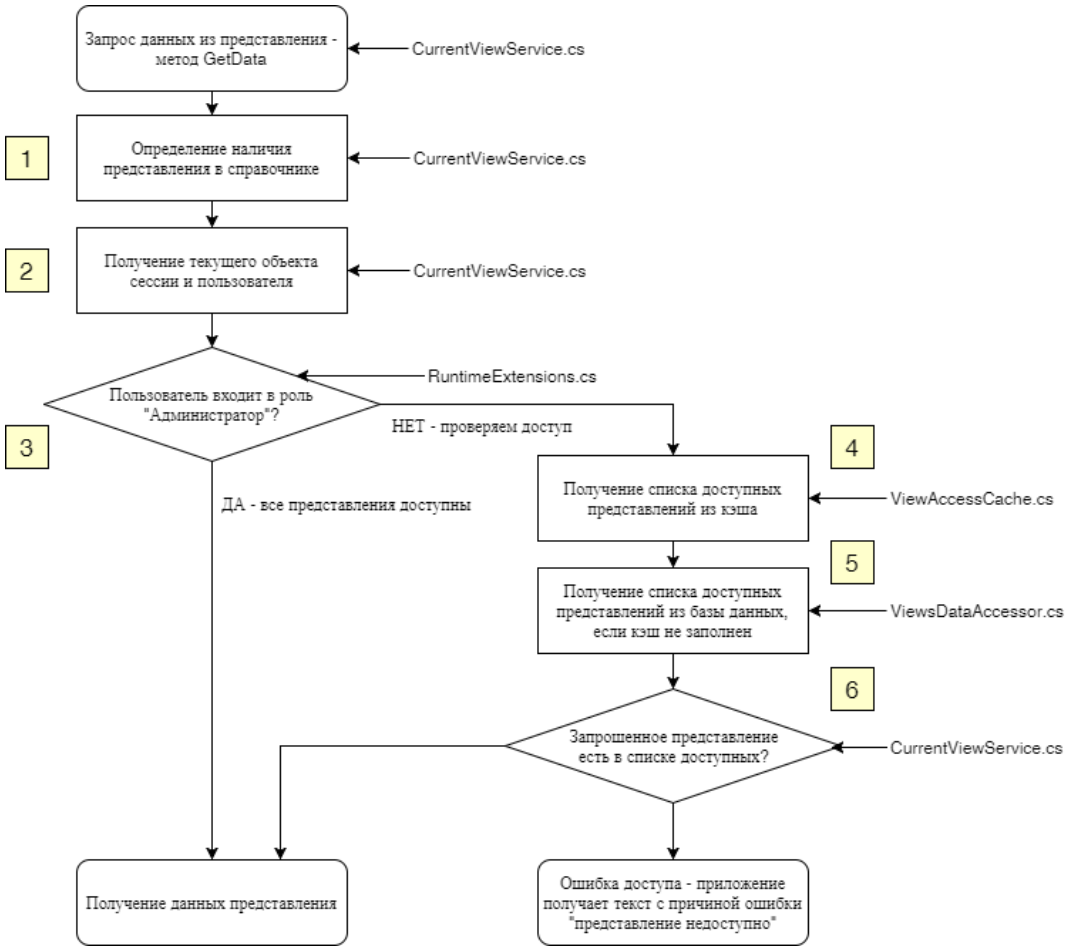


Рисунок 9 – Алгоритм разграничения доступа к представлениям

Любой запрос к веб-сервису на получение данных представления приводит к вызову метода GetData из класса CurrentUserViewService (файл …\Views\CurrentUserViewService.cs).

Блок 1. Определение наличия представления в справочнике.

Метод запрашивает объект представления по имени viewService.GetByName, и если оно отсутствует – возвращает ошибку в виде исключения о том, что представление не существует.

Затем вызывается метод ThrowIfAccessDenied этого же класса CurrentUserViewService, задача которого – выбросить исключение с ошибкой, если доступ к представлению отсутствует у пользователя в текущей сессии. Если метод не выбросил исключение, то вызывается метод view.GetData, который непосредственно запрашивает данные представления (выполняет SQL-запрос) и возвращает результат.

Блок 2. Получение текущего объекта сессии и пользователя.

Метод ThrowIfAccessDenied вызывает sessionFactory для получения объекта текущей сессии (библиотека Unity обеспечивает связь компонента сессий из класса Session и компонента представлений из класса CurrentUserViewService). По объекту сессии извлекается информация о пользователе session.User, данные которого заполняются из объекта токена сессии (SessionToken).

Блок 3. Пользователь входит в роль "Администратор"?

Для объекта пользователя вызывается метод-расширение user.IsAdministrator, он же метод IsAdministrator в классе RuntimeExtensions (файл …\Platform\Runtime\RuntimeExtensions.cs).

Метод проверяет свойство AccessLevel объекта user, которое соответствует полю AccessLevelID в таблице активных сеансов Sessions, которое в свою очередь заполняется из карточки работника, что было показано в алгоритме «Регистрация пользователя в системе». Это свойство содержит одно из значений перечисления UserAccessLevel – роль, в которую входит работник в соответствии с настройкой в карточке работника (файл …\Platform\Runtime\UserAccessLevel.cs).

Если работник входит в роль администратора, то метод IsAdministrator возвращает true. В этом случае метод ThrowIfAccessDenied пропускает две последующие проверки и возвращает управление (а метод GetData после этого выполнит запрос и вернёт данные клиентскому приложению).

Блок 4. Получение списка доступных представлений из кэша.

Если работник не входит в роль администратора, то выполняется метод viewAccessCache.GetViews класса ViewAccessCache (файл …\Views\ViewAccessCache.cs). Метод возвращает список уникальных имён представлений (из поля «Алиас»), которые доступны работнику.

Класс либо вернёт значение из кэша в памяти, либо заполнит его на основании списка ролей, указанных в представлении (см. рисунок 9). Кэш ViewAccessCache автоматически сбрасывается при любом изменении представления в рамках сервера приложений.

Блок 5. Получение списка доступных представлений из базы данных, если кэш не заполнен.

В качестве функции, заполняющей кэш для пользователя с уникальным идентификатором userId именами представлений getFunc – передаётся метод LoadAccess класса CurrentUserViewService.

Метод LoadAccess вызывает accessor.GetAccessibleUserViews для класса ViewsDataAccessor (файл …\Views\ViewsDataAccessor.cs), который выполняет SQL-запрос и возвращает список имён представлений, доступных пользователю с заданным уникальным идентификатором, на основании его вхождения в группы (карточки ролей – таблицы ViewRoles и RoleUsers) для соответствующих представлений (таблица Views).

Далее метод LoadAccess получает аналогичный список для «виртуальных» представлений, которые настраиваются не в справочнике представлений, а серверными расширениями (реализующими интерфейс IViewAccess и зарегистрированными в Unity), и дополняет список доступных имён представлений этими расширениями, если они в методе GetRoles возвращают одну из ролей, в которую входит пользователь. В сертифицированной версии отсутствуют «виртуальные» представления и их нельзя добавить, не добавив компилируемый модуль .dll с кодом представлений, поэтому здесь не рассматривается дополнение прав виртуальными представлениями.

Метод LoadAccess возвращает вычисленный список имён представлений, который метод GetViews кэширует для последующего ускорения проверок, после чего список возвращается в метод ThrowIfAccessDenied.

Блок 6. Запрошенное представление есть в списке доступных

Метод ThrowIfAccessDenied выполняет проверку на вхождение запрошенного имени представления в список доступных имён:

Если проверка не пройдена, то выбрасывается исключение с текстом о недостатке доступа, который возвращается на клиентское (веб-клиент) и отображается пользователю. Если же проверка пройдена, то метод ThrowIfAccessDenied возвращает управление в метод GetData, который уже

выполняет SQL-запрос представления и возвращает результат клиентскому приложению (веб-клиент), которое отображает его в форме таблицы.

3.2.3.2. Разграничение ролевого доступа к справочнику работников

Ролевой доступ к справочнику работников определяется возможностью сохранить (создать, изменить или удалить) карточку работника. Если пользователь входит в роль администратора, то ему доступны любые действия с любыми карточками работников; в противном случае выполняется ряд проверок, который гарантирует, что пользователь изменяет только те данные, которые ему дозволено изменять: меняет свой пароль, свои настройки, свои замещения или замещения работников, которые разрешили изменять их замещения (поле «Кто может редактировать» в диалоге «Мои настройки»).

Схема алгоритма приведена на рисунке 10.

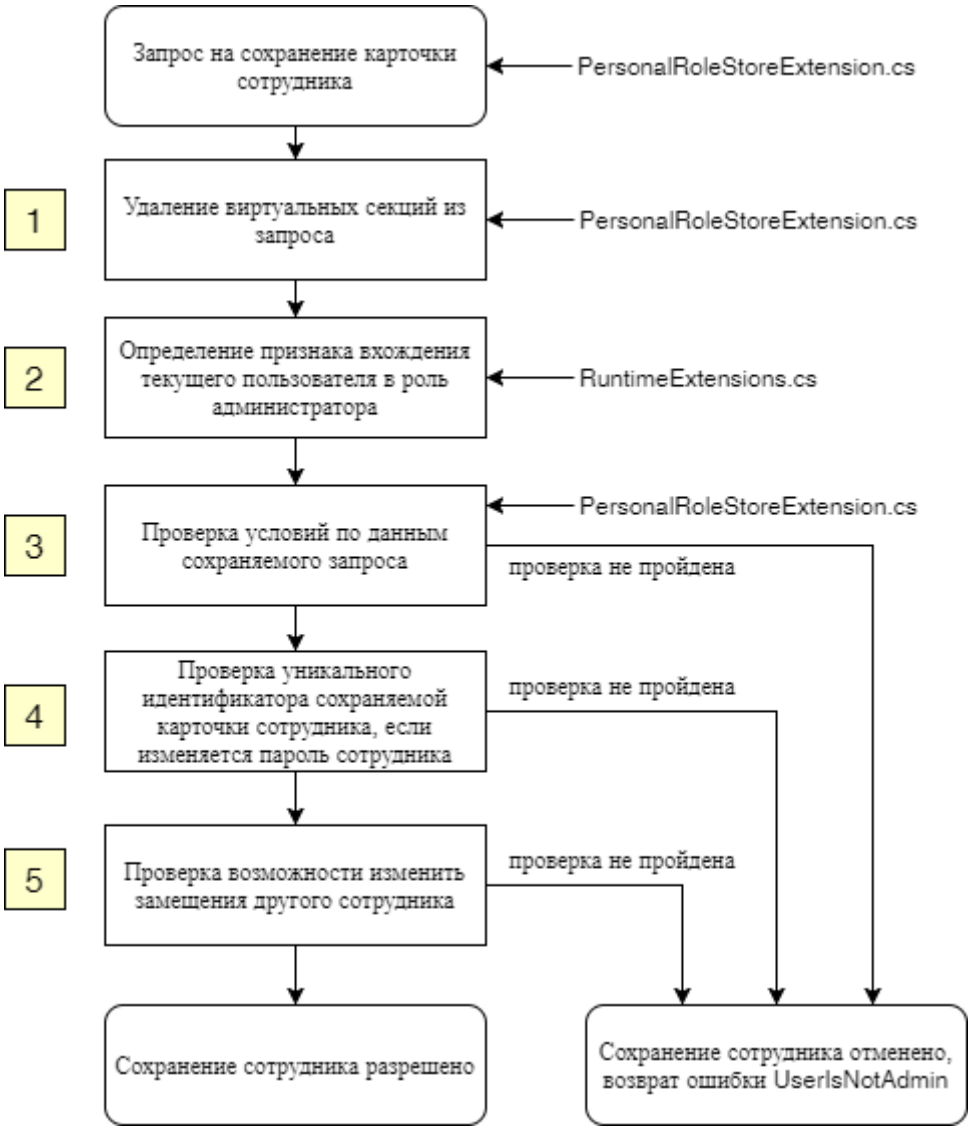


Рисунок 10 – Алгоритм разграничения ролевого доступа к справочнику работников

Проверки выполняются в расширении на сохранение карточки работника в классе PersonalRoleStoreExtension (файл …\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs).

Блок 1. Удаление виртуальных секций из запроса.

Перед сохранением выполняется метод BeforeRequest класса PersonalRoleStoreExtension. В первую очередь, в нём удаляются виртуальные секции из запроса на сохранение, информация по которым доступна на вкладках «Мои настройки», «Боковые панели» и «Web клиент» в карточке работника. Если этого не сделать, то секции всё равно не будут сохранены (они виртуальные, поэтому не хранятся в базе данных и используются в расширениях), а дальнейшие проверки

выдали бы ошибку при сохранении для пользователя, не входящего в роль администратора.

Блок 2. Определение признака вхождения текущего пользователя в роль администратора.

Сессия работника при сохранении карточки уже доступна в свойстве context.Session. Для объекта пользователя вызывается метод-расширение context.Session.User.IsAdministrator, он же метод IsAdministrator в классе RuntimeExtensions (файл …\Platform\Runtime\RuntimeExtensions.cs).

На основании данных в токене текущей сессии SessionToken определяется, что пользователь входит в роль администратора, т.е. является администратором системы, в этом случае метод возвращает true.

Блок 3. Проверка условий по данным сохраняемого запроса.

Далее проверяются условия и параметры сохраняемой карточки (поле mainCard) для пользователя, не входящего в роль администратора:

Если любое из нижеописанных условий выполняется, то в объект context.ValidationResult добавляется сообщение об ошибке UserIsNotAdmin, которое выводится пользователю.

Если любое условие вернуло истину, то возвращается ошибка и дальнейшие проверки не выполняются.

Проверяются следующие условия:

- Карточка работника создаётся: mainCard.StoreMode == CardStoreMode.Insert. Пользователи, не входящие в роль администратора, не могут создавать карточки работников.
- В карточке добавляется, удаляется или изменяется хотя бы один файл: mainCard.Files.Count > 0. Пользователи, не входящие в роль администратора, не могут менять файлы, приложенные к карточке своего или другого работника. В качестве таких файлов, например, может выступать фотография работника, которая затем отображается в отчётах.

- Есть хотя бы одна сохраняемая (т.е. изменённая) секция карточки, которая не входит в список personalRoleSectionsToIgnore (это таблицы на вкладке «Мои замещения», а также секция PersonalRoleVirtualSection).
- Сохраняются данные секции PersonalRoleVirtualSection и изменяется поле с языком интерфейса. PersonalRoleVirtualSection (константа «PersonalRolesVirtual») – это секция для сохранения и общих настроек работника (задаваемых через диалог «Мои настройки»), и языка интерфейса (поле «Язык» в карточке работника), и ввода пароля при его изменении (см. далее). Данная проверка запрещает изменять язык интерфейса через карточку работника (для этого у пользователя есть интерфейс на правой выпадающей панели, который изменяет язык отдельным запросом).

Блок 4. Проверка уникального идентификатора сохраняемой карточки работника, если изменяется пароль работника.

Если все предыдущие условия не выполнились или пользователь входит в роль администратора, то выполняется проверка о том, что при изменении пароля (поля Password и PasswordRepeat в секции PersonalRoleVirtualSection) сохраняется либо карточка текущего работника из сессии, или пользователь входит в роль администратора.

Если пользователь не входит в роль администратора, и каким-то образом выполняется запрос на изменение пароля другого работника, то на клиентское приложение (веб-клиент) возвращается ошибка UserIsNotAdmin и дальнейшие проверки прекращаются.

Далее система открывает транзакцию в базе данных, и вызывается метод AfterBeginTransaction в классе PersonalRoleStoreExtension. Если выполнение дошло до этого момента, то текущий пользователь либо входит в роль администратора, либо сохраняется его же пароль или настройки, либо изменяются замещения для этого работника (всегда разрешено) или для других работников (должно быть разрешение в поле «Кто может редактировать»).

Блок 5. Проверка возможности изменить замещения другого работника.

Поэтому в случае, если пользователь не входит в роль администратора и уникальный идентификатор сохраняемой карточки отличается от идентификатора текущего пользователя, то выполняется проверка на то, что в таблице «RoleDeputiesManagementAccess» (поле «Кто может редактировать») для сохраняемого работника (card.ID) записана строка с текущим пользователем (user.ID).

Если доступ на изменение замещений отсутствует, то на клиентское приложение (веб-клиент) возвращается ошибка UserIsNotAdmin и дальнейшее сохранение прерывается, система выполняет откат открытой транзакции в базе данных.

Если же проверки выше пройдены, то сохранение разрешается и выполняется штатными механизмами системы по сохранению карточек.

3.2.4. Алгоритмы работы механизма ролевого разграничения доступа к информационным ресурсам

Данные алгоритмы описывают механизм ролевого разграничения доступа к информационным ресурсам в платформы.

3.2.4.1. Разграничение ролевого доступа при открытии карточки

Разрешение доступа при открытии карточек рассчитывается в серверном классе расширения KrCheckPermissionsGetExtension (файл …\Workflow\KrPermissions\KrCheckPermissionsGetExtension.cs). Обработка производится в методе AfterRequest, который выполняется после фактической загрузки карточки из базы данных, но до её передачи клиентскому приложению (веб-клиент).

Если расширение добавит сообщение об ошибке в объект context.ValidationResult, то платформа удалит загруженную карточку из ответа на запрос context.Response (удаление в методе EraseOnError, класс CardGetResponse, файл …\Cards\CardGetResponse.cs). Т.о. при ошибке доступа вместо карточки будет возвращён ответ на запрос, содержащий только сообщения об ошибках.

Рассмотрим схему алгоритма ролевого разграничения доступа к при открытии карточки. Схема алгоритма приведена на рисунке 11.

Блок 1. Проверка включения типа карточки в настройки типового решения.

Система проверяет, включён ли тип загруженной карточки в типовое решение (на правой панели Настройки -> Типовое решение). Ролевая система разграничения прав, основанная на справочнике правил доступа (все карточки прав доступа), функционирует только для типов карточки, которые включены в типовое решение. Для других карточек (таких, как карточек «Работник») выполняются другие расширения, проверяющие доступ.

Наличие типа карточки в типовом решении определяется в методе GetKrComponents класса KrComponentsHelper (файл …\Workflow\KrProcess\KrComponentsHelper.cs), задача которого – определить набор функций типового решения, которые включены для типа карточки. Если включена функция Base, то тип включён в типовое решение, и поэтому дальнейшие проверки по правилам доступа имеют смысл.

Блок 2. Определение списка прав для расчёта при открытии карточки.

Далее в методе AfterRequest определяется, какие права требуется рассчитать, т.е. наличие каких прав требуется проверить.

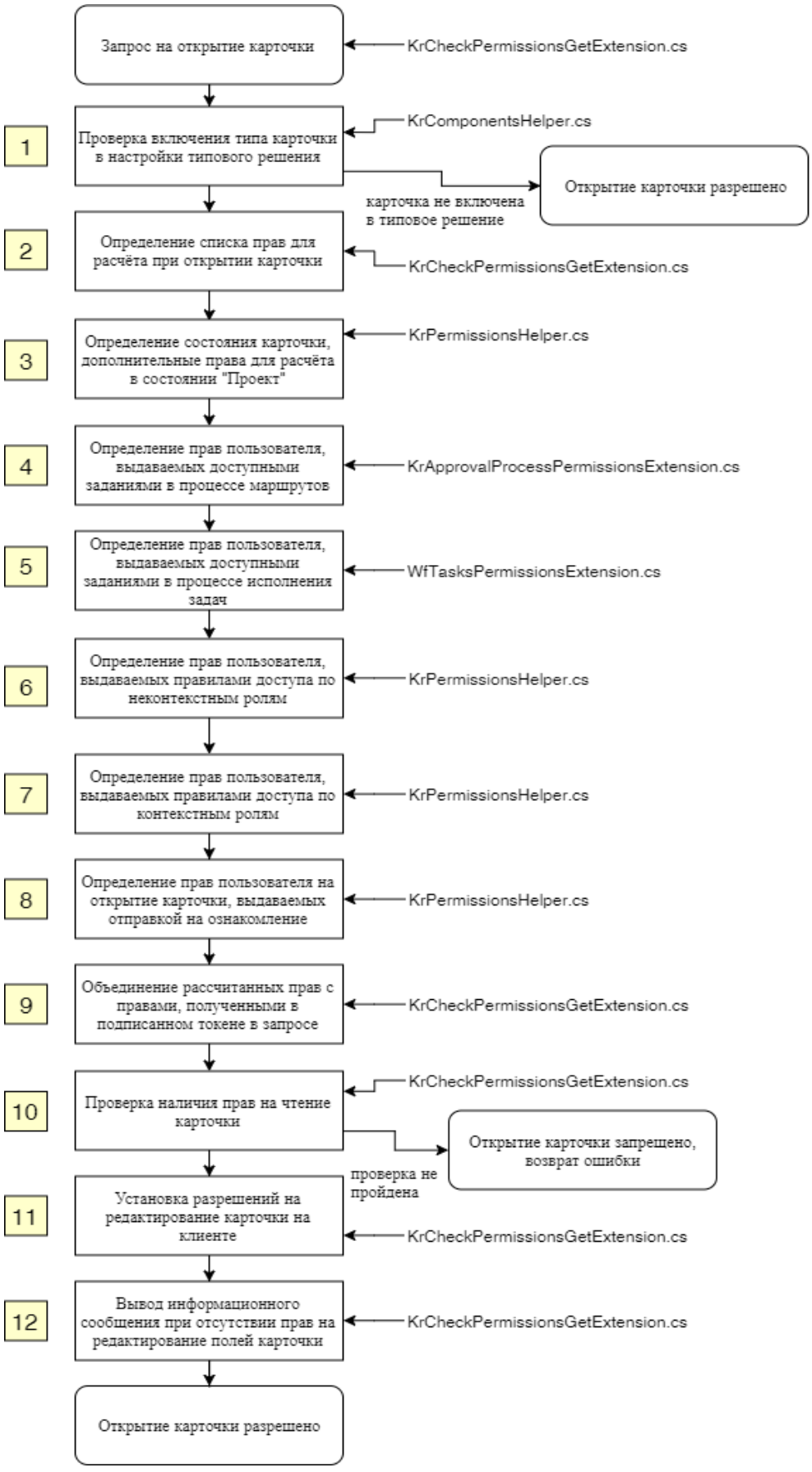


Рисунок 11 – Алгоритм ролевого разграничения при открытии карточки

Право на чтение карточки KrPermissionFlags.ReadCard проверяется всегда, но при загрузке карточки, выполняемый в результате нажатия кнопки «Редактировать» (параметр запроса KrPermissionsHelper.CalculatePermissionsMark), рассчитываются права

KrPermissionFlags.AllCardEditingAndAllFiles, которые включают в себя следующие компоненты (перечисление KrPermissionFlags, файл …\Workflow\KrProcess\KrPermissionFlags.cs.

Т.о. при загрузке после нажатия кнопки «Редактировать» будут рассчитываться права на чтение карточки, редактирование полей карточки, добавление, редактирование, удаление и подписание файлов, редактирование маршрута и ручное редактирование номера документа. Также при нажатии «Редактировать» устанавливается «needMessage = true», чтобы в дальнейшем при отсутствии прав на редактирование полей карточки отобразить пользователю сообщение с перечислением доступных прав.

Флаг KrPermissionsHelper.CalculateResolutionPermissionsMark может быть установлен в запросе, если выполняется отправка на массовое ознакомление. Поскольку права на массовое ознакомление зависят от наличия разрешения «Инициация типового процесса отправки задач» в карточке правила доступа, то в этом случае вместо права на чтение карточки проверяет прав на отправку задач KrPermissionFlags.CreateResolutions.

Блок 3. Определение состояния карточки, дополнительные права для расчёта в состоянии «Проект».

Далее выполняется расчёт имеющихся у пользователя прав доступа в методе CheckHasPermissions класса KrPermissionsHelper (файл …\Workflow\KrPermissions\KrPermissionsHelper.cs).

Метод получает флаговое перечисление с разрешениями required, для которых требуется определить, есть ли у пользователя соответствующие разрешения. Затем метод определяет состояние карточки, причём для состояния «Проект» (KrState.Draft) запрашивается расчёт полных прав KrPermissionFlags.AllCardEditingAndAllFiles, как

и при нажатии кнопки «Редактировать», чтобы в состоянии «Проект» пользователям не требовалась нажимать кнопку «Редактировать».

Блок 4. Определение прав пользователя, выдаваемых доступными заданиями в процессе маршрутов.

Если хотя бы одно разрешение запрошено, то в первую очередь метод вычисляет разрешения, связанные с задачами. Задачи будут присутствовать в коллекции карточки card.Tasks только в том случае, если они видны текущему пользователю как исполнителю (с учётом взятия задачи в работу), заместителю исполнителя, автору или заместителю автора.

В зависимости от типа задачи task.TypeID выполняется расширение на права задач ITaskPermissionsExtension, которое выдаёт соответствующие права. Для задач типовых процессов маршрутов выдаваемые разрешения описаны в расширении KrApprovalProcessPermissionsExtension (файл …\Workflow\KrPermissions\KrApprovalProcessPermissionsExtension.cs).

Для любых задач в маршрутах выдаются права на чтение карточки и подписание файлов. Т.е. если даже у пользователя нет прав на документ по правилам доступа, но ему доступна задача из маршрута (например, он согласующий), то система выдаёт ему права на чтение карточки и подписание любых файлов.

Если это задание запроса комментария, то даже не взятое в работу задание сразу предоставляет права на добавление файлов и редактирование своих файлов. Так сделано, потому что задание запроса комментария автоматически берётся в работу, т.е. оно отображается сразу с активным полем ввода комментария, после чего пользователь вводит текст, нажимает кнопку и одновременно берёт задание в работу и завершает его.

Любые другие виды заданий предоставляют права только исполнителям и их заместителям (но не автору) после того, как задание взято в работу.

Для заданий «Доработка» автоматически предоставляются полные права на редактирование карточки KrPermissionFlags.AllCardEditingAndAllFiles. Т.о. инициатору при доработке документа не потребуется нажимать кнопку «Редактировать».

Для заданий «Согласование» и «Подписание» выдаются права на добавление и редактирование своих файлов. Также, если в настройках этапа маршрута указаны флажки «Редактировать карточку» и «Редактировать любые файлы», то выдаются права, соответственно, на редактирование полей карточки и редактирование любых приложенных файлов.

Для заданий «Дополнительное согласование» выдаются права на добавление файлов и редактирование своих файлов.

Блок 5. Определение прав пользователя, выдаваемых доступными заданиями в процессе исполнения задач.

Также выполняется расширение WfTasksPermissionsExtension (файл …\Workflow\Wf\WfTasksPermissionsExtension.cs), которое предоставляет права для задач процесса исполнения, которые могли быть добавлены либо в этапе маршрута «Задача», либо кнопкой «Поставить задачу».

Для задач исполнения (типы заданий WfResolution***) выдаются права на чтение карточки и подписание любых файлов. Если задание взято в работу исполнителем, то ему предоставляются права на добавление файлов и редактирование своих файлов.

Далее метод CheckHasPermissions класса KrPermissionsHelper после всех прав, которые выданы за счёт заданий (если они были доступны), определяет список прав stillRequired, для которых всё ещё требуется определить, есть ли они у пользователя.

Если таких прав нет, т.е. задания предоставили все запрашиваемые права, и, возможно, права сверх того, то возвращается набор прав, предоставленных заданием. «Пересечение» двух наборов прав выполняется методом CheckHasRequired того же класса KrPermissionsHelper, который учитывает, что право на редактирование любых файлов EditFiles автоматически добавляет право на редактирование собственных файлов EditOwnFiles. И аналогично, право на удаление любых файлов DeleteFiles включает в себя и право на удаление собственных файлов DeleteOwnFiles.

Блок 6. Определение прав пользователя, выдаваемых правилами доступа по неконтекстным ролям.

Теперь метод CheckHasPermissions выполняет получение списка правил доступа с запрошенными флагами KrPermissionFlags в виде хеш-таблицы rulesForStaticRoles, где каждому отдельному флагу сопоставляется список подходящих для текущего пользователя правил доступа.

Такой список вычисляется запросом к базе данных в методе GetRulesForStaticRoles этого же класса.

По полученным правилам также вызывается расширение permissionsExtension, которое отдельно настраивается в рамках проектного решения. Разработка такого расширение связана с написанием программного кода, его компиляцией и конфигурированием в папке сервера. Поэтому мы считаем, что в сертифицированной версии нельзя написать такое расширение, а в приведённых фрагментах кода переменная permissionsExtension равна null.

Метод CheckRulesForStaticRolesWithExtension того же класса KrPermissionsHelper агрегирует флаги разрешений по полученным правилам доступа для тех ролей в системе, которые имеют состав, т.е. в таблице RoleUsers которой перечислены пользователи в роли. Фактически такими ролями являются любые роли, кроме контекстных.

Поскольку расширение permissionExtension равно null, то наличие флага в правиле доступа гарантирует, что этот флаг будет доступен в результате метода, который агрегируется с имеющимися разрешениями в переменной result.

Блок 7. Определение прав пользователя, выдаваемых правилами доступа по контекстным ролям.

Управление возвращается в метод CheckHasPermissions, который определяет список оставшихся разрешений среди тех, которые будут рассчитаны. Если такие разрешения есть, то выполняется расчёт правил доступа по контекстным ролям, т.е. ролям, состав которых зависит от текущей карточки. При таком расчёте для каждой роли выполняется запрос, указанный в самой контекстной роли, например, запрос на вычисление инициатора документа для контекстной роли «Инициатор».

Метод CheckRulesForContextRolesWithExtension того же класса KrPermissionsHelper выполняет операции по запросу данных из контекстных ролей. Результирующие разрешения агрегируются с флагами в переменной «result».

Блок 8. Определение прав пользователя на открытие карточки, выдаваемых отправкой на ознакомление.

Наконец, метод CheckHasPermissions проверяет, что если среди всех рассчитанных до настоящего момента разрешений отсутствует разрешение на чтение карточки, но такое разрешение было запрошено, то метод определяет, есть ли у пользователя запрос на ознакомление.

Запрос выполняется к базе данных в методе CheckHasInformingRow того же класса. Если у пользователя есть запись в таблице «Мне на ознакомление» (MassInformingRows) для текущей карточки, то пользователю предоставляются права на чтение карточки.

Результирующие права метод CheckHasPermissions возвращает как права, которые доступны пользователю. Это те права из рассчитанных, которые требовались (пересечение «result & required») плюс права от заданий «taskResult».

Блок 9. Объединение рассчитанных прав с правами, полученными в подписанном токене в запросе.

Управление возвращается в класс расширения KrCheckPermissionsGetExtension. Он проверяет что, если в рассчитанных методом CheckHasPermissions разрешениях effectivePermissions присутствует право на чтение карточки KrPermissionFlags.ReadCard, то создаётся объект токена KrToken, который ниже записывается в card.Info загруженной карточки.

Объект токена содержит идентификатор карточки, номер её версии, срок истечения действий (равен двум дням от момента выписывания без учёта календаря, см. файл …\Workflow\KrProcess\KrTokenProvider.cs) и набор флагов с расширениями KrPermissionFlags.

Это позволяет в момент сохранения получить токен из card.Info (поскольку в запросе на сохранение будет отправлен тот же объект card с клиентского приложения (веб-клиент)), проверить его подпись (доступную только на сервере в поле

SignatureKey конфигурационного файла app.json в папке веб-сервисов) и, если она валидна – использовать полученные права как уже доступные пользователю и не выполнять их расчёт заново.

Если в запросе уже есть токен и он валидный, то вместо рассчитанных прав (в которых в этой ветке выполнения нет прав на чтение карточки) будут использоваться права из запроса. Такие права обычно пробрасываются с клиентского приложения (веб-клиент) при обновлении карточки, т.к. если права уже были доступны для этой версии карточки (карточка не менялась), вкладка с карточкой ещё не закрыта на клиентском приложении (веб-клиент) и дата истечения токена не просрочена (не прошло двух дней), то будут использованы ранее вычисленные права. В противном случае – используются права, которые рассчитаны в процессе текущей загрузки карточки (даже если там нет прав на чтение).

Блок 10. Проверка наличия прав на чтение карточки.

Если в карточке отсутствуют права на чтение, то добавляется сообщение об ошибке в объект context.ValidationResult (строка локализации), ошибка будет выведена пользователю, а из запроса будет удалена карточка перед её возвратом на клиентское приложение (веб-клиент).

Блок 11. Установка разрешений на редактирование карточки на клиентском приложении (веб-клиент).

Система устанавливает разрешения card.Permissions на редактирование карточки в зависимости от рассчитанных прав. Такие разрешения являются только рекомендацией для клиентских приложений (веб-клиент) не позволять редактировать данные пользователю, т.е. чтобы пользователь видел, что редактирование тех или иных элементов карточки запрещено. Даже если клиентское приложение (веб-клиент) игнорирует эти разрешения, то при сохранении карточки будет возвращён рассчитанный здесь токен KrToken, по которому и по данным в базе данных (если в токене не было каких-то прав – они могли появиться) будут выполнены фактические проверки при сохранении различных данных.

Разрешения устанавливаются в методе SetCardPermissions этого же класса KrPermissionsHelper, в котором в зависимости от рассчитанных (или полученных из

предыдущего токена) разрешений token.Permissions будет указано, что те или иные поля карточки, строки в таблицах, файлы или другие элементы интерфейса (в т.ч. возможность удаления карточки) будут доступны только для чтения (для удаления – не будут доступны), пользователь не сможет их редактировать или использовать в клиентских приложениях (Веб-клиент).

Блок 12. Вывод информационного сообщения при отсутствии прав на редактирование полей карточки.

После того, как разрешения установлены в свойстве card.Permissions, в методе AfterRequest расширения KrCheckPermissionsGetExtension выполняется проверка для ситуации, когда пользователь явно нажал кнопку «Редактировать» (переменная needMessage = true), но у пользователя нет прав на редактирование полей карточки EditCard:

В этом случае пользователю добавляется «информационное» сообщение с перечислением доступных прав. Наличие такого сообщения не влияет на успешное открытие карточки.

После выполнения всех других расширений, в случае их успешного выполнения карточка вместе с рассчитанным токеном с разрешениями KrToken и разрешениями card.Permissions отправляются на клиентское приложение (веб-клиент), где пользователь может взаимодействовать с карточкой.

3.3. Используемые методы

Методы языков программирования C#, SQL, Javascript/TypeScript.

3.3.1. Структура программы с описанием функций составных частей и связи между ними

Платформа состоит из следующих составных частей:

1) Модули и компоненты сервера платформы:
   - Ядро системы (Gossed, Gossed.Compilation, Gossed.Linux, Gossed.Extensions.PostgreSql.Server);

- Фоновые сервисы Chronos (Gossed.Chronos.Platform, Gossed.Chronos.Contract, Gossed.Chronos.Core, Chronos.Platform.Linux, Gossed.Chronos, Gossed.Extensions.Default.Chronos);
- Веб-сервис (Gossed.Web.Server.Core, Gossed.Web.Client, Gossed.Web, Gossed.Extensions.Default.Server, Gossed.Extensions.Default.Shared);
- Конфигуратор tadmin (Gossed.Console, Gossed.Admin.Console.Core);

2) Модули и компоненты «Веб-Клиент».
   - Рабочее место пользователя «Веб-Клиент». Клиентское приложение на базе веб-браузера. Веб-клиент, используя веб-сервис сервера приложений, предоставляет доступ к рабочему месту пользователя.

3.3.2. Описание составных частей платформы

Описание функционального назначения файлов с исходными текстами составных частей платформы структурированное по каталогам, приведено в документе «Описание порядка создания дистрибутива из исходных текстов программ» (RU.46939656.58.29.29.000-02 90 01).

Описание составных частей платформы на уровне файлов и функций безопасности информации, реализуемых этими файлами, приведено в Таблице 3 и Таблице 4.

Таблица 3 - Список файлов из описания алгоритмов функций безопасности информации

Имя файла	Назначение	Каталог
PersonalRoleStoreExtension.cs	Проверки безопасности бизнес-логика при создании или изменении карточки работника	…\Extensions.Platform.Server\Roles
RuntimeHelper.cs	Константы, настройки и вспомогательные методы для генерации и проверки подписи, для чтения и замены ключа безопасности TokenSignature в конфигурационном файле, и различные методы, не связанные с безопасностью	…\Platform\Runtime

Имя файла	Назначение	Каталог
	(получение информации о приложении при публикации, парсинг доменного имени и др.)	
SyncSignatureProvider.cs	Объект, предоставляющий криптографические средства для подписания и проверки подписи синхронным криптоалгоритмом HMACSHA256	…\Platform
UserPasswordValidator.cs	Объект, выполняющий проверку пароля работника на соответствие настройкам безопасности (длина пароля, использование символов разных регистров, чисел и спец. символов, и неповторяемость последних N паролей)	…\Platform\Runtime
SessionServer.cs	Объект, обеспечивающий взаимодействие с сессиями на сервере, в т.ч. открытие сессии, валидация при очередном запросе, закрытие сессии	…\Platform\Runtime
WebSessionHostInfoProvider.cs	Объект, предоставляющий информацию по компьютеру пользователя – по IP-адресу и имени компьютера	…\Services\
DefaultSessionLoginProvider.cs	Объект, предоставляющий информацию по входу работника в систему с использованием стандартного справочника работникаов. Формирует SELECT к таблице с работниками и возвращает данные из БД	…\Platform\Runtime
UserSecurityProvider.cs	Объект, управляющий хранением объекта с настройками безопасности работника UserSecurityObject. Это объект, хранимый в БД в каждой карточке работника в виде поля с массивом байт, в котором содержится информация по количеству последних	…\Platform\Runtime

Имя файла	Назначение	Каталог
	неудачных попыток и их дата/время, и список последних использованных хешей от паролей и солей для хеширования	
UserBlockingManager.cs	Объект, выполняющий SQL-запросы для установки и снятия блокировки работника	…\Platform\Runtime
RuntimeExtensions.cs	Вспомогательные методы для пространства имён Platform.Runtime, из них к безопасности относятся методы генерации подписываемого массива байт по информации из сессии, методы проверки этого массива с использованием интерфейса криптопровайдера (связывается с классом SyncSignatureProvider.cs). К подписываемым данным относятся уникальный идентификатор сессии sessionID, код сервера serverCode (строка из конфига), имя экземпляра сервера instanceName (строка из конфига)	…\Platform\Runtime
SessionToken.cs	Объект, содержащий все свойства сессии, включая подписываемые данные (sessionID, serverCode, instanceName), собственно подпись (в форме base64-строки) и информацию по работнику (имя, идентификатор-логин, уникальный идентификатор, роль Администратор=да/нет)	…\Platform\Runtime
ActionHistoryStrategy.cs	Объект, выполняющий SQL-запросы к таблице ActionHistory, содержащей историю действий (логи аудита по всем событиям в системе). Метод Insert вставляет строку с описанием действия (параметры указываются вызывающим кодом). Метод TryGet	…\Platform\Runtime

Имя файла	Назначение	Каталог
	загружает строку для её последующего просмотра как карточки в представлении «История действий». Метод Delete удаляет строку, но в сертифицированной версии его вызов отключён (расширение DeleteHistoryRequestExtension не вызывается)	
CurrentUserViewService.cs	Класс, обеспечивающий проверку доступа к представлениям для текущего пользователя в зависимости от вхождения этого пользователя в группы, указанные в списке «Роли» в представлении	…\Views
UserAccessLevel.cs	Перечисление (список констант) с указанием роли, в которую входит работника: обычный пользователь Regular или пользователь в роли администратора Administrator	…\Platform\Runtime
ViewAccessCache.cs	По уникальному идентификатору работника (Guid) кэширует список доступных ему представлений по вхождению пользователя в группы (в поле «Роли»). Кэш автоматически сбрасывается при изменении ролей (или при перезапуске сервиса). Метод, наполняющий кэш, передаётся из CurrentUserViewService.cs	…\Views
ViewsDataAccessor.cs	Объект, выполняющий SQL-запросы для проверки вхождения пользователя в группы, заданные для представления (метод GetUserRoles), и для изменения списка групп для представления, а также для загрузки списка всех представлений, и для импорта	…\Views

Имя файла	Назначение	Каталог
	представлений (в момент первичной установки)	
CardGetResponse.cs	Ответ на запрос по загрузке карточки, содержит загруженную карточку Card и/или сообщения, возникшие при загрузке ValidationResult. Если среди сообщений есть ошибки, то Card очищается в методе EraseOnError (базовая реализация в классе CardValueResponseBase)	…\Cards
KrCheckPermissionsGetExtension.cs	Расширение на проверку прав по правилам доступа при загрузке карточки	…\Workflow\KrPermissions
KrComponentsHelper.cs	Вспомогательные методы для проверки наличия включённых компонентов (опций) для типов карточек и типов документов в типовом решении. Такими компонентами являются (см. перечисление KrComponents): Base (признак вхождения типа в настройки типового решения, если отсутствует, то правила доступа не используются); DocTypes (признак того, что для типа карточки доступны отдельные настройки в различных типах документов, например, для типа карточки «Договорной документ» есть типы документов «Договор» и «Доп. соглашение», для которых отдельно задаются правила доступа); Routes (для документа включена подсистема маршрутов, т.е. можно запускать бизнес-процессы); Registrations (включена регистрация, можно выделить регистрационный номер);	…\Workflow\KrProcess

Имя файла	Назначение	Каталог
	Resolutions (можно поставить задачу кнопкой в левом меню)	
KrPermissionFlags.cs	Перечисление, значения которого соответствуют флажкам в карточке правил доступа, т.е. отдельным правам: на создание карточки, на редактирование полей, на добавление файлов и др. Все значения этого перечисления, у которых нет атрибута [ExcludeFromSql], используются при формировании SELECT-а к карточкам правил доступа, причём имя значения соответствует имени колонки в таблице правил доступа KrPermissions с префиксом “Can”, например: флаг с правом на редактирование карточки EditCard соответствует колонке KrPermissions.CanEditCard. Формирование запроса на права по правилам доступа происходит в методе GetRulesForStaticRoles в файле KrPermissionsHelper.cs	…\Workflow\KrProcess
KrPermissionsHelper.cs	Методы, формирующие SQL-запросы и выполняющие проверки на вхождение в группы (роли) по правилам доступа, а также по заданиям (наличие задания, взятого в работу, предоставляет права на чтение карточки, добавление файлов, подписание файлов и редактирование своих файлов)	…\Workflow\KrPermissions
KrApprovalProcessPermissionsExtension.cs	Расширение на права для заданий типового процесса маршрутов: задания согласования, доп. согласования, запроса комментариев, подписания, регистрации, доработки. Вызывается по типу задачи в	…\Workflow\KrPermissions

Имя файла	Назначение	Каталог
	KrPermissionsHelper для получения дополнительных прав от наличия в карточке заданий	
WfTasksPermissionsExtension.cs	Расширение на права для типовых задач: постановка задачи по кнопке в левом меню, делегированная (отправленная) задача, возвращённая на контроль задача, подзадача. Вызывается по типу задачи в KrPermissionsHelper для получения дополнительных прав от наличия в карточке заданий	…\Workflow\Wf
KrTokenProvider.cs	Объект, создающий токен с правилами доступа для карточки с заданными идентификатором и версией, и генерирующий подпись и проверяющий подпись на валидность (используя класс SyncSignatureProvider.cs)	…\Workflow\KrProcess
CheckTaskAccessStoreExtension.cs	Расширение, которое выполняется при сохранении любой карточки, и вызывает метод CheckTaskAccessHelper.CheckAccess, который в свою очередь проверяет права текущего пользователя на завершение задания, если в запросе на сохранение есть хотя бы одно задание. Метод-расширение IgnoreTaskAccessCheck может получить из текущего контекста расширения context.Info идентификаторы заданий, для которых игнорируется проверка, но хеш-таблица context.Info формируется на сервере, поэтому значение может быть заполнено только другими серверными расширениями	…\Cards

Имя файла	Назначение	Каталог
CardRequestExtensions.cs	Вспомогательные методы, связанные с объектами запросов и контекстов расширений, относящиеся к объектам карточек (в пространстве имён Cards). Содержит метод IgnoreTaskAccessCheck, позволяющий проверить признак того, что проверку доступа для идентификатора заданий можно пропустить. Список игнорируемых заданий может быть заполнено только на сервере (когда из одного расширения данные переносятся в другие расширения), поэтому на безопасность запросов это не влияет	…\Cards
CheckTaskAccessHelper.cs	Содержит метод, выполняющий SQL-запрос с проверкой прав текущего работника на сохранение или завершении задачи. Права проверяются по признаку того, является ли текущий пользователь (в сессии) исполнителем задания или его заместителем, или же автором задания. Если пользователь ими не является, то операция сохранения завершается с ошибкой	…\Extensions.Platform.Server\Cards

Таблица 4 - Реализация функциональных возможностей подсистемы безопасности информации платформы на уровне файлов

Наименование функциональной возможности	Реализация на уровне исходных файлов
Идентификация и аутентификация пользователей и сессий (ИАФ.1)	…\Platform\Runtime\SessionServer.cs

Наименование функциональной возможности	**Реализация на уровне исходных файлов**
Определение отдельной роли пользователя	…\Platform\Runtime\DefaultSessionLoginProvider.cs
Создание идентификатора	…\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs
Присвоение идентификатора	…\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs
Блокирование идентификатора	…\Platform\Runtime\UserBlockingManager.cs
Изменение идентификатора	…\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs
Уничтожение (удаление) идентификатора (ИАФ.3)	…\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs
Определение отдельной роли пользователя (ИАФ.4)	…\Platform\Runtime\DefaultSessionLoginProvider.cs
Генерация и выдача начальной аутентификационной информации	…\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs
Изменение текущей аутентификационной информации	…\Extensions.Platform.Server\Roles\PersonalRoleStoreExtension.cs
Установление характеристик пароля (ИАФ.4)	…\Platform\Runtime\UserPasswordValidator.cs

Наименование функциональной возможности	**Реализация на уровне исходных файлов**
Защита обратной связи при вводе аутентификационной информации (ИАФ.5)	…\Platform\SyncSignatureProvider.cs
Управление (заведение, активация, блокирование и уничтожение) учетными записями пользователей (УПД.1)	…\Extensions.Platform.Server\Roles\ PersonalRoleStoreExtension.cs
Разделение полномочий (ролей) (УПД.4)	…\Platform\Runtime\RuntimeExtensions.cs
Создание карточки	…\Workflow\KrPermissions\KrCheckCanCreateNewExtension.cs
Чтение карточки	…\Workflow\KrPermissions\KrCheckPermissionsGetExtension.cs
Редактирование карточки	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs
Редактированию собственных файлов	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs
Редактирование всех файлов	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs

Наименование функциональной возможности	**Реализация на уровне исходных файлов**
Удаление собственных файлов	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs
Удаление всех файлов	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs
Добавление файлов	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs
Редактирование маршрута согласования	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs
Удаление карточки	…\Workflow\KrPermissions\KrCheckCanDeleteCard.cs
Запуск процесса согласования	…\Workflow\KrProcess\Requests\KrButtonActionStoreExtension.cs
Отзыв процесса согласования или отмене регистрации	…\Workflow\KrProcess\Requests\KrButtonActionStoreExtension.cs

Наименование функциональной возможности	**Реализация на уровне исходных файлов**
Возврат карточки на доработку	…\Workflow\KrProcess\Requests\KrButtonActionStoreExtension.cs
Отмена процесса согласования	…\Workflow\KrProcess\Requests\KrButtonActionStoreExtension.cs
Регистрация карточки	…\Workflow\KrProcess\Requests\KrButtonActionStoreExtension.cs …\Workflow\KrPermissions\KrCheckPermissionsGetExtension.cs
Ручное редактирование номера	…\Workflow\Wf\WfWorkflowStoreExtension.cs
Инициирование типового процесса отправки задач Подписание файлов (УПД.2)	…\Workflow\KrPermissions\KrCheckPermissionsStoreExtension.cs
Ограничение неуспешных попыток входа в информационную систему (УПД.6)	…\Platform\Runtime\SessionServer.cs
Блокирование сеанса доступа в информационную систему после установленного времени бездействия (неактивности)	…\Platform\Runtime\UserBlockingManager.cs

Наименование функциональной возможности	**Реализация на уровне исходных файлов**
пользователя или по его запросу (УПД.10)	
Определение состава и содержания информации о событиях безопасности, подлежащих регистрации (РСБ.2**)**	…\Cards\ComponentModel\CardActionTextDescriptionBuilder.cs
Сбор, запись и хранение информации о событиях безопасности (РСБ.3**)**	…\Platform\Runtime\ActionHistoryStrategy.cs
Предоставление средств мониторинга (просмотр, анализ) результатов регистрации событий безопасности (РСБ.5)	…\Extensions.Platform.Server\Cards\ActionHistoryGetExtension.cs
Защита информации о событиях безопасности (РСБ.7)	…\Platform\Runtime\ActionHistoryStrategy.cs

3.4. Связи платформы с другими программами

Платформа осуществляет экспорт/импорт данных в/из приложений офисного пакета Р7-Офис. Экспорт данных из платформы осуществляется в виде файлов форматов (.doc, .docx, .rtf и другие). Импорт данных в платформу осуществляется в виде файлов форматов Adobe (.pdf).

4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Минимальные и рекомендуемые технические требования к аппаратному обеспечению функционирования платформы для сервера приведены в таблице 5.

Таблица 5 - Требования к аппаратному обеспечению сервера

Тип сервера	Требования на 100 пользователей	Требования на 500 пользователей
Сервер приложений:	Процессор: 8 ядра и более, 2 ГГц и выше; Оперативная память (RAM): 24 ГБ и более; Система хранения данных (HDD/SSD): от 500 ГБ	Процессор: 32 ядра и более, 2 ГГц и выше; Оперативная память (RAM): 128 ГБ и более; Система хранения данных (HDD/SSD): от 500 ГБ
Сервер БД (баз данных)	Процессор: 8 ядра, 2,5 ГГц и выше; Оперативная память (RAM): 32 ГБ и более; Система хранения данных (HDD/SSD): от 2 ТБ, рекомендуется RAID, не менее 200 IOPS	Процессор: 128 ядер и более, 3 ГГц и выше; Оперативная память (RAM): 128 ГБ и более; Система хранения данных (HDD/SSD): от 2 ТБ, рекомендуется RAID, не менее 500 IOPS
Сервер аутентификации	Процессор: 2 ядра и более, 2 ГГц и выше; Оперативная память (RAM): 4 ГБ и более; Система хранения данных (HDD/SSD): от 200 ГБ	Процессор: 4 ядра и более, 2 ГГц и выше; Оперативная память (RAM): 8 ГБ и более; Система хранения данных (HDD/SSD): от 200 ГБ
Сервер индексации и поиска	В минимальной конфигурации не используется	Согласно рекомендациям производителя Manticore Search
Сервер кеширования	В минимальной конфигурации не используется	Согласно рекомендациям производителя Redis Cache
Сервер файлового хранилища S3	В минимальной конфигурации не используется, взамен используется хранение на	Согласно рекомендациям производителя файлового хранилища S3

	файловой подсистеме, например сетевом диске	
Сервер очередей сообщений	В минимальной конфигурации не используется	Согласно рекомендациям производителя Rabbit MQ

В таблице 5 приведены усредненные требования к аппаратному обеспечению СЭД. Расчет аппаратного обеспечения зависит от ряда параметров таких как количество одновременно работающих пользователей, пиковой нагрузке, количества документов и производится индивидуально для каждого случая.

Минимальные и рекомендуемые технические требования к аппаратному обеспечению функционирования рабочего места платформы (рабочей станции) приведены в таблице 6.

Таблица 6 - Требования к аппаратному обеспечению рабочей станции

Тип объекта вычислительной техники	**Требования к аппаратному обеспечению платформы**
Клиентское приложение (веб-клиент)	Процессор: 2 ГГц (Pentium 4/AMD Athlon); Оперативная память (RAM): 8 ГБ; Система хранения данных: 1 Гб свободного места на диске; Видеоподсистема: любая видеокарта, включая встроенные; Разрешение экрана: 1024x768; Сетевое соединение: 100 Мбит/с

5. ВЫЗОВ И ЗАГРУЗКА

Вызов и загрузка платформы осуществляются из файловой системы, поддерживаемой системным программным обеспечением (операционной системой) на базе которого оно функционирует путем запуска исполняемого (бинарного/загрузочного) модуля, в результате чего происходит загрузка и выполнение исполняемых модулей платформы.

6. ВХОДНЫЕ ДАННЫЕ

6.1. Характер, организация и предварительная подготовка входных данных

В общем случае, входными данными для платформы являются запросы пользователей к платформе с использованием комплекса средств защиты платформы от несанкционированного доступа (механизмам: ролевого контроля доступа, регистрации, идентификации и аутентификации) через среду функционирования (системное и прикладное программное обеспечение приведенные в п. 1.2 настоящего документа), путем выбора функциональных элементов в указанных средствах манипулятором типа мышь и ввода данных через клавиатуру.

6.2. Формат, описание и способ кодирования входных данных

Введенные и передаваемые пользователем в платформу данные (текстовые данные – идентификационная и аутентификационная информация (логин, пароль), запрос на доступ к функциональным элементам платформы: механизмам ролевого контроля доступа, механизму регистрации событий; и данным, которые можно получить через указанные механизмы) с помощью аппаратного и программного обеспечения через среду взаимодействия (каналам передачи данных) подвергаются специальному преобразованию (кодированию) транспортными механизмами протокола TLS, которые в последствие подвергаются повторному специальному преобразованию (декодированию) и оцениваются составными частями платформы.

7. ВЫХОДНЫЕ ДАННЫЕ

7.1. Характер и организация выходных данных

Выходной информацией является результат выполнения запроса пользователей на доступ в платформу, доступ к объектам доступа (защищаемым ресурсам); на изменение правил разграничения доступа, изменение списка субъектов доступа (пользователей) и списка объектов доступа; на выборочное ознакомление с регистрационной информацией; на ввод данных и попытку разграничения привилегий.

Результатом выполнения запроса пользователя (администратора) могут быть:

- пустое множество данных (нулевые данные);
- сообщение об ошибке или некорректно введенных данных, если пользователь не имеет требуемых для выполнения запроса прав или ошибся при вводе идентификационной и аутентификационной информации;
- данные, доступ пользователя к которым является санкционированным в соответствии с определенной ролью пользователя и заданными правилами разграничения доступа к объектам.

7.2. Формат, описание и способ кодирования выходных данных

Передаваемые субъектам доступа запрашиваемые данные (ответ в результате запроса на доступ к механизму управления доступа, разграничения привилегий, механизму регистрации событий и их функциональным элементам: объектам доступа, изменение списка субъектов и объектов доступа, изменение правил разграничения доступа, в результате ввода идентификационной и аутентификационной информации, запроса регистрационной информации) подвергаются специальному преобразованию (кодированию) транспортными механизмами протокола TLS, через которые происходит взаимодействие субъектов доступа с платформой.

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

Сокращение	Расшифровка
DNS	(англ. Domain Name System «система доменных имён») — компьютерная распределённая система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты и/или обслуживающих узлах для протоколов в домене
IP	(от англ. Internet Protocol) — уникальный числовой идентификатор устройства в компьютерной сети, работающий по протоколу TCP/IP
JSON	(англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript
SQL	(англ. Structured Query Language, язык структурированных запросов) — язык программирования, предназначенный для управления данными в СУБД
TLS	(англ. transport layer security — протокол защиты транспортного уровня) — криптографические протоколы, обеспечивающие защищённую передачу данных между узлами в сети Интернет
UTF-8	(от англ. Unicode Transformation Format, 8-bit — «формат преобразования Юникода, 8-бит») — распространённый стандарт кодирования символов, позволяющий более компактно хранить и передавать символы Юникода, используя переменное количество байт (от 1 до 4), и обеспечивающий полную обратную совместимость с 7-битной кодировкой ASCII
БД	база данных
ИС	информационная система
ОС	операционная система
ПО	программное обеспечение
СУБД	система управления базами данных

ПЕРЕЧЕНЬ ПРИНЯТЫХ ТЕРМИНОВ

Термин	Определение
Hash	См. «Хэш»
Аутентификация	Процедура проверки подлинности
Бизнес-процесс	Совокупность взаимосвязанных мероприятий или работ, направленных на создание определённого продукта или услуги для потребителей
Веб-сервис	Идентифицируемая уникальным веб-адресом программная система со стандартизированными интерфейсами, а также HTML-документ сайта, отображаемый браузером пользователя
Интерфейс	Совокупность средств и правил, обеспечивающих взаимодействие отдельных систем (например, человека, программного обеспечения, аппаратного обеспечения и т. п.)
Кэш	(англ. cache, от фр. cacher — «прятать») — промежуточный буфер с быстрым доступом к нему, содержащий информацию, которая может быть запрошена с наибольшей вероятностью. Доступ к данным в кэше осуществляется быстрее, чем выборка исходных данных из более медленной памяти или удалённого источника, однако её объём существенно ограничен по сравнению с хранилищем исходных данных
Логин	Уникальное имя учётной записи пользователя в компьютерной системе
Объект	Некоторая сущность, обладающая определённым состоянием и поведением, имеющая определённые свойства (атрибуты) и операции над ними (методы)
Плейсхолдер	Вспомогательный текст внутри полей ввода (подсказка по формату заполнения поля)
Пользователь	Лицо или организация, которое использует действующую систему для выполнения конкретной функции
Тип документа	Подвид типа карточки. Все документы разных типов, но одного типа карточки – имеют одинаковый набор полей, форму, но могут иметь разные права, нумерацию, возможность регистрации, согласования, использования задач. Типы карточек и базовые типы документов предоставляются для удобства и могут меняться или не использоваться в каждом конкретном решении. Можно

Термин	Определение
	настраивать собственные типы карточек и включать их в типовое решение
Тип карточки	Тип карточки, описанный как отдельный тип в платформе, со своей отдельной структурой (набором полей) и внешним видом. Все базовые типы карточек имеют общие поля, которые не используются в типовом решении и добавлены для удобства. Если они не находят применения в конкретном внедрении – их можно просто скрыть в настройках формы типа карточки. Типы карточек и базовые типы документов предоставляются для удобства и могут меняться или не использоваться в каждом конкретном решении. Можно настраивать собственные типы карточек и включать их в типовое решение
Транзакция	Группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными
Учетная запись	Хранимая в системе совокупность данных о пользователе, необходимая для его опознавания и предоставления доступа к его личным данным и настройкам
Хэш, хеширование	(англ. hash function от hash — «превращать в фарш», «мешанина»), или функция свёртки — функция, осуществляющая преобразование массива входных данных произвольной длины в выходную битовую строку установленной длины, выполняемое определённым алгоритмом. Преобразование, производимое хеш-функцией, называется хешированием. Исходные данные называются входным массивом, «ключом» или «сообщением». Результат преобразования называется «хешем», «хеш-кодом», «хеш-суммой», «сводкой сообщения»

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ документа	Входящий № сопроводи-тельного документа	Подпись	Дата
	измененных	замененных	новых	аннулированных					