МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО

ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения   
и администрирования информационных систем

**Система обработки документов: хранение и поиск**

Бакалаврская работа

Направление 09.03.03 Прикладная информатика   
Профиль Прикладная информатика в юриспруденции



|  |  |  |
| --- | --- | --- |
| Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_ | д. ф.-м. н., проф. | Артёмов М. А. |
| Обучающийся\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Дегтярёва А. В. |
| Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_ | к.т.н., преп. | Старикова А. А. |

Воронеж 2019

**Аннотация**

Данная работа посвящена проектированию и практической реализации системы обработки документов. В ходе работы рассмотрены основные этапы разработки информационной системы: проведен анализ функциональности существующих систем электронного документооборота и сформированы требования к проектируемой системе, описаны модели данных, с применением основных средств языка UML описана структура и основные функции системы. На основе проведенного анализа созданы базы данных, выполнена практическая реализация спроектированных модулей и разработан прототип клиентского приложения, взаимодействующего с системой.

Содержание

[Введение 4](#_Toc9182727)

[1. Постановка задачи 6](#_Toc9182728)

[2. Анализ 7](#_Toc9182729)

[2.1. Анализ существующих приложений 7](#_Toc9182730)

[2.2. Анализ функциональности приложения 9](#_Toc9182731)

[2.3. Анализ структуры приложения 12](#_Toc9182732)

[2.4. Анализ моделей данных 15](#_Toc9182733)

[2.4.1. Анализ реляционной модели данных 15](#_Toc9182734)

[2.4.2. Анализ нереляционной модели данных 16](#_Toc9182735)

[2.5. Анализ пользовательского интерфейса 17](#_Toc9182736)

[3. Средства реализации 18](#_Toc9182737)

[4. Требования к программному и аппаратному обеспечению 19](#_Toc9182738)

[5. Реализация 20](#_Toc9182739)

[5.1. Структура системы 20](#_Toc9182740)

[5.2. Модели данных 22](#_Toc9182741)

[5.2.1. Реляционная модель данных 22](#_Toc9182742)

[5.2.2. Документоориентированная модель данных 22](#_Toc9182743)

[5.3. Диаграмма классов 24](#_Toc9182744)

[6. Интерфейс пользователя 25](#_Toc9182745)

[7. Web API документация 26](#_Toc9182746)

[8. Тестирование 27](#_Toc9182747)

[Заключение 28](#_Toc9182748)

[Список литературы 29](#_Toc9182749)

[Приложение 1. Диаграммы компонентов микросервиса администрирования системы 30](#_Toc9182750)

[Приложение 2. SQL-скрипт для создания базы данных 32](#_Toc9182751)

[Приложение 3. Описание сущностей базы данных 34](#_Toc9182752)

[Приложение 4. Диаграмма классов микросервиса администрирования системы 37](#_Toc9182753)

[Приложение 5. Листинг классов данных микросервиса администрирования системы 38](#_Toc9182754)

Введение

Сотрудники любой организации тратят большое количество времени на поиск и обработку необходимых документов и информации, объем которой с каждым днем увеличивается. Внедрение системы электронного документооборота (СЭД) дает ряд преимуществ по сравнению с ручной обработкой: сокращение временных затрат на все рутинные операции с документами, обеспечение конфиденциальности информации, сокращение дублирования и оптимизация хранения документов, повышение качества данных, а следовательно, качества работы сотрудников и обслуживания клиентов.

В настоящее время существует множество СЭД, обладающих богатыми функциональными возможностями и широкой областью применения. Однако для большинства предприятий малого и среднего бизнеса функциональность этих систем избыточна, кроме того, их внедрение влечет высокие начальные затраты и скрупулезную работу по обучению пользователей.

Одной из важнейших функций таких систем является хранение и поиск документов. Любой документ, хранящийся в системе электронного документооборота, состоит из двух частей: его содержимого и набора реквизитов, так называемой карточки электронного документа, на основе которых осуществляется поиск, классификация и группировка. Если документ подлежит регистрации, то на него заводится еще и регистрационная карточка (РК). Возможен полнотекстовый поиск, поиск по реквизитам и регистрационным данным. Полнотекстовый поиск осуществляется по данным в самом документе, в том числе по словоформам через встроенные средства СУБД. Атрибутивный поиск осуществляется через специальную форму по нескольким значениям из полей карточек. Эта функциональность может использоваться отдельно от систем документооборота, а также интегрироваться с любой другой системой.

Таким образом, целесообразно создать программу, которая обеспечит легкий поиск и сохранность данных, быстрый доступ к документам и будет обладать понятным интерфейсом, легкостью внедрения и возможностью интеграции с другими системами с целью повышения эффективности работы организации.

1. Постановка задачи

Спроектировать систему обработки документов, предоставляющую следующую функциональность:

* загрузка и регистрация документов;
* представление документов в виде набора значений атрибутов;
* хранение документов;
* хранение атрибутов документов;
* атрибутивный поиск документов;
* полнотекстовый поиск документов;
* просмотр текста документов;
* просмотр карточек документов;
* удаление документов;
* редактирование карточек документов;
* создание шаблонов для каждого вида документов;
* создание каталогов;
* разграничение прав доступа.

Реализовать спроектированные модули и разработать прототип клиентского приложения, взаимодействующего с системой.

Провести модульное и интеграционное тестирование.

1. Анализ
   1. Анализ существующих приложений

В настоящее время существует большое число систем электронного документооборота. Наиболее известными являются: Documentum, Е1 Евфрат, Directum, Дело, 1С: Документооборот 8, Docsvision, ELMA. Сравнительный анализ их функциональности приведен в табл. 2.1.

Таблица 2.1. Сравнительный анализ СЭД

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Характеристики** | **Documentum** | **Е1 Евфрат** | **Directum** | **Дело** | **1С: Документооборот 8** | **Docsvision** | **ELMA** |
| **Загрузка и регистрация документов** | | | | | | | |
| Сканирование | + | + | + | + | + | + | + |
| Распознавание | + | + | + | + | + | + | + |
| Возможность добавления в РК дополнительных атрибутов при регистрации | + | – | – | + | + | – | – |
| Конструктор карточек | – | + | + | – | – | + | + |
| Зависимость набора атрибутов РК от выбранного типа документа | + | + | + | + | + | + | + |
| Наличие обязательных атрибутов регистрационной карточки | + | + | + | + | + | + | + |
| Заполнение атрибутов значениями из системных справочников | + | + | + | + | + | + | + |
| Автоматическое формирование регистрационного номера | + | + | + | + | + | + | + |
| Поточное сканирование документов | + | + | + | + | + | + | + |
| Автоматическое извлечение атрибутов | + | + | + | + | – | + | + |
| **Хранение документов** | | | | | | | |
| Хранение атрибутов | + | + | + | + | + | + | + |
| СУБД | + | + | + | + | + | + | + |
| **Поиск** | | | | | | | |
| Создание каталогов | + | + | + | + | + | + | + |
| Полнотекстовый поиск | + | + | + | + | + | + | + |
| Атрибутивный поиск | + | + | + | + | + | + | + |
| Поиск по шаблонам | + | – | – | – | + | + | + |
| Создание шаблонов поиска | + | – | – | – | + | + | + |
| Сохранение поисковых запросов | + | + | + | + | + | + | – |
| **Разграничение прав доступа** | | | | | | | |
| Разграничение прав доступа на уровне каталога | – | + | + | + | + | + | + |
| Разграничение прав доступа на уровне вида документов | + | – | – | + | + | – | + |
| Разграничение прав доступа на уровне конкретного документа | + | + | + | + | + | + | + |
| **Интеграция со сторонними системами** | | | | | | | |
| Встроенная интеграция с системами организации | + | + | + | + | + | + | + |
| Наличие открытого API | + | + | + | + | + | + | + |

* 1. Анализ функциональности приложения

На основе проведенного анализа существующих решений выделена необходимая функциональность приложения:

* загрузка и регистрация документов;
* хранение документов;
* поиск документов;
* просмотр документов;
* удаление документов;
* редактирование карточек документов;
* создание шаблонов для каждого вида документов;
* управление доступом к данным.

Загрузка и регистрация должна включать следующую функциональность:

* загрузка документа;
* автоматическое присвоение документу уникального регистрационного номера;
* представление документа в виде набора значений атрибутов (заполнение регистрационной карточки документа).

Хранение должно включать следующую функциональность:

* настройка иерархической структуры хранения (каталогов);
* хранение файлов документов;
* хранение значений атрибутов документов в базе данных.

Поиск документов должен включать следующую функциональность:

* просмотр содержимого каталогов;
* атрибутивный поиск по основным атрибутам (вид документа, регистрационный номер, дата загрузки, ФИО регистратора);
* расширенный атрибутивный поиск по шаблонам поиска;
* полнотекстовый поиск документов.

Просмотр документов должен включать следующую функциональность:

* просмотр текста документа;
* просмотр содержимого карточки документа.

Создание шаблонов подразумевает под собой формирование наборов атрибутов с указанием их обязательности. Такая функциональность необходима, чтобы предоставить организации возможности настройки системы в зависимости от используемых видов документов и их атрибутов. Созданные шаблоны могут использоваться при регистрации документов или для расширенного атрибутивного поиска.

Управление доступом должно осуществляться с помощью механизма создания ролей, которые назначаются пользователю и определяют набор его прав. Разграничение прав доступа должно включать следующую функциональность:

* разграничение прав доступа к модулям системы;
* разграничение прав доступа к документам на уровне иерархии хранения, которое подразумевает под собой наличие или отсутствие прав на просмотр содержимого каталога;
* разграничение прав доступа на уровне видов документов.

Для каждого вида документа должна быть предусмотрена возможность предоставления пользователям следующих прав на документы данного вида:

* право на просмотр текста документа;
* право на просмотр карточки документа;
* право на редактирование карточки;
* право на удаление документа.

В ходе анализа и формирования требований к проектируемой системе выделены три функциональных блока и, соответственно, три основные роли пользователя системы. Функциональность, связанная с администрированием системы доступна пользователю с ролью «Администратор». Возможность загрузки и регистрации документов доступна пользователям, которые обладают ролью «Регистратор». Остальные возможности системы не требуют наличия особой роли и доступны всем пользователям в соответствии с политикой разграничения доступа конкретной организации. Диаграммы вариантов использования системы обработки документов приведены на рис. 2.1 – 2.3.

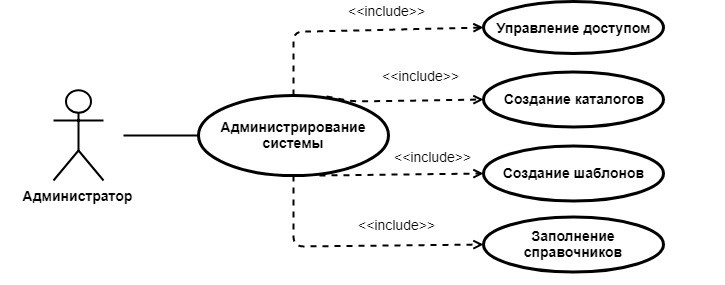


Рис. 2.1. Диаграмма вариантов использования системы пользователем с ролью «Администратор»

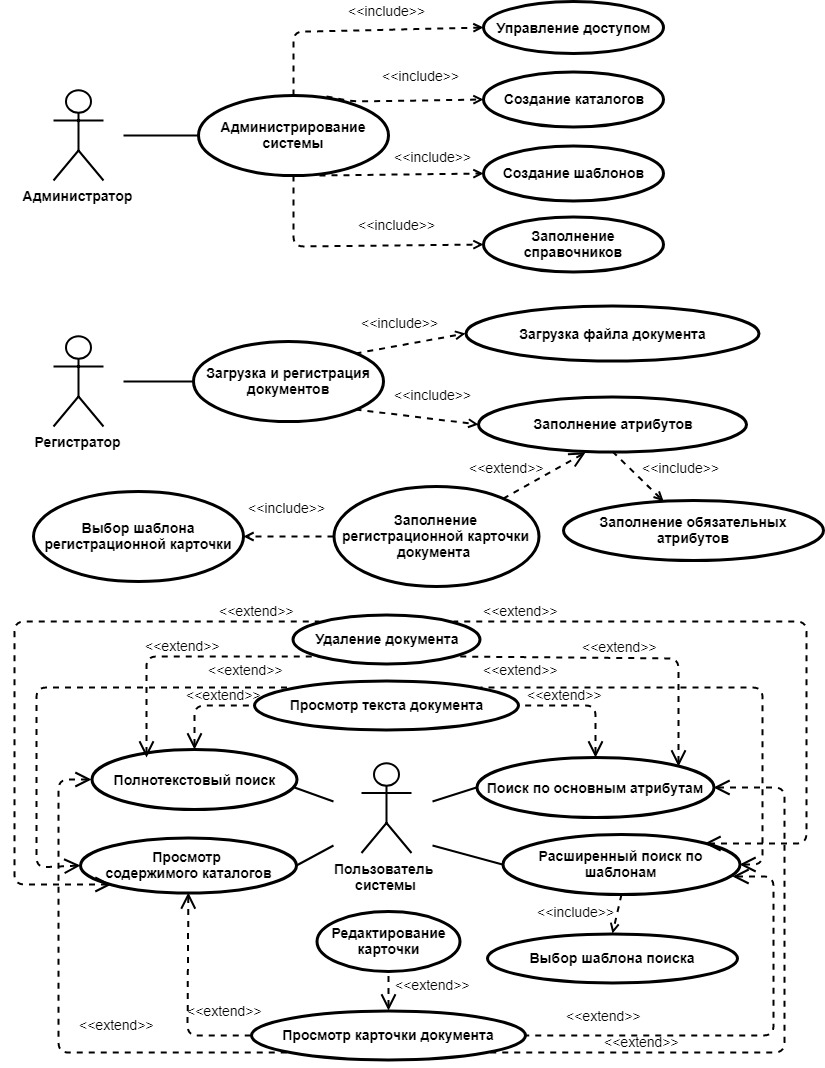


Рис. 2.2. Диаграмма вариантов использования системы пользователем с ролью «Регистратор»

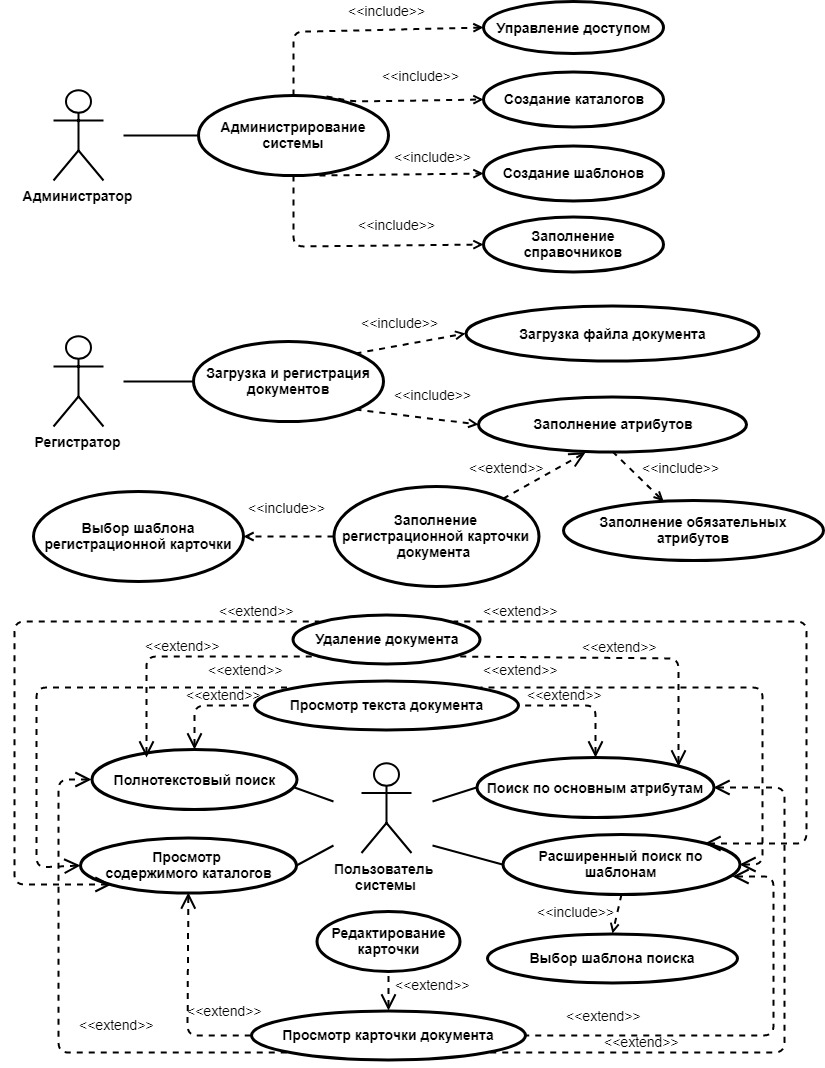


Рис. 2.3. Диаграмма вариантов использования системы пользователем, не обладающим специальной ролью

* 1. Анализ структуры приложения

Исходя из наличия нескольких обособленных функциональных блоков, для разрабатываемой системы выбрана сервис-ориентированная архитектура, при которой на стороне сервера имеется несколько независимо обновляемых и заменяемых компонентов – микросервисов.

Данная архитектура имеет следующие преимущества:

* возможность независимого развертывания микросервисов;
* возможность независимого масштабирования микросервисов;
* возможность независимой разработки каждого компонента.

**Независимое развертывание** позволяет изменять функциональность одного сервиса, не затрагивая другие компоненты и даже не приостанавливая их работу, а также заменить один микросервис другим или использовать каждый в отдельности. К тому же, у микросервисов выше доступность: даже если один из них вышел из строя, это не обязательно приведет к сбою всего приложения.

**Независимая масштабируемость** позволяет отмасштабировать один из сервисов, во время повышенной нагрузки на него, без необходимости в масштабировании каждого компонента, что делает масштабирование гибким и снижает расходы.

**Независимая разработка** дает возможность использования наиболее подходящих технологий при разработке каждого микросервиса, в том числе использование различных типов баз данных.

Перечисленные преимущества позволят в дальнейшем менять конфигурацию системы обработки документов в зависимости от нужд конкретной организации.

Структурная схема приложения приведена на рис. 2.4.

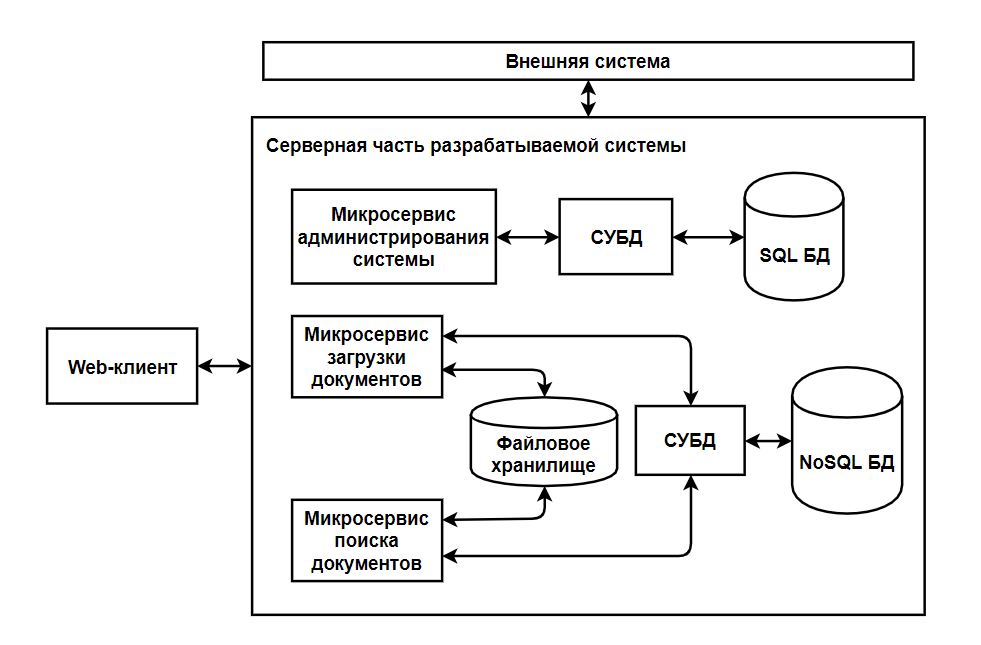


Рис. 2.4. Структурная схема приложения

Микросервис администрирования необходим для настройки системы администратором и управления доступом. Данный компонент предоставит возможности заполнения справочников, создания каталогов и шаблонов, а также предоставления пользователям прав доступа. Для хранения информации, связанной с работой рассматриваемого сервиса, выбрана реляционная база данных.

Микросервис загрузки документов предоставит возможность загрузки и регистрации документов. Загружаемые документы будут храниться в виде файлов на сервере, образуя файловое хранилище. Значения атрибутов загружаемых документов будут храниться в нереляционной (NoSQL) базе данных.

Ниже перечислены причины выбора NoSQL базы данных для хранения метаданных документов:

* быстродействие, обеспечивающее эффективный поиск;
* возможность хранения больших объемов информации;
* легкость масштабирования;
* отсутствие ограничений на типы хранимых данных;
* отсутствие схемы и определенной структуры, что позволит хранить данные, состоящие из произвольного набора атрибутов и обеспечит независимость системы от предметной области.

Микросервис поиска документов предоставит пользователям системы функциональность, связанную с поиском и обработкой документов. Для этого он взаимодействует с нереляционной базой данных, описанной выше.

Клиентское приложение реализует пользовательский интерфейс, формирует запросы к сервисам и обрабатывает ответы от них. Отделение пользовательского интерфейса (UI) от микросервисов позволит поддерживать UI целиком, сохраняя его связность. В случае изменений границ в UI не придётся одновременно обновлять несколько компонентов, нарушая изолированность и независимость, обеспечиваемые архитектурой.

В качестве клиента системы может выступать любое приложение, однако использование web-клиента на рабочих местах дает определенные преимущества:

* отсутствие необходимости устанавливать приложение по клиентским местам и платить за дополнительные лицензии;
* техническая простота, так как от пользователя требуется просто запустить браузер и правильно набрать адрес;
* поддержка любых браузеров и, соответственно, любых операционных систем и всех гаджетов;
* доступ к системе из любой точки мира, где есть доступ во всемирную сеть.

Клиентское приложение и внешняя система будут взаимодействовать с микросервисами по сети. Все микросервисы должны предоставлять API для доступа к их данным.

* 1. Анализ моделей данных
     1. Анализ реляционной модели данных

Реляционная модель данных включает:

* атрибут;
* шаблон;
* вид документа;
* каталог;
* роль;
* права на вид документа;
* права на каталог.

Для атрибута необходимо хранить:

* уникальный идентификатор;
* наименование;

Для шаблона необходимо хранить:

* уникальный идентификатор;
* наименование;
* набор атрибутов шаблона с указанием их обязательности.

Для вида документа необходимо хранить:

* уникальный идентификатор;
* наименование;
* набор шаблонов для данного вида.

Для каталога необходимо хранить:

* уникальный идентификатор;
* наименование;
* идентификатор родительского каталога;

Для роли необходимо хранить:

* уникальный идентификатор;
* наименование.

Для прав на вид документа необходимо хранить:

* уникальный идентификатор вида документа;
* уникальный идентификатор роли, которой принадлежат права;
* наличие права доступа к тексту документов данного вида;
* наличие права доступа к карточке документов данного вида;
* наличие права удаления документов данного вида;
* наличие права изменения карточки документов данного вида.

Для прав на каталог необходимо хранить:

* уникальный идентификатор каталога;
* уникальный идентификатор роли, которой принадлежит право доступа к каталогу.
  + 1. Анализ нереляционной модели данных

Нереляционная модель данных включает метаданные документа. Для документа необходимо хранить:

* регистрационный номер;
* вид документа;
* каталог;
* дату регистрации;
* путь к файлу документа;
* сведения о пользователе, загрузившем документ;
* текст;
* набор значений атрибутов регистрационной карточки;
* набор прав на документ.
  1. Анализ пользовательского интерфейса

Интерфейс приложения должен включать:

* главную страницу;
* страницу администрирования системы;
* страницу загрузки и регистрации документов;
* страницу поиска документов;
* страницу просмотра документа;
* страницу редактирования карточки документа.

1. Средства реализации

Для разработки приложения выбраны следующие программные средства:

* среда разработки IntelliJ IDEA Ultimate 2019.1;
* язык программирования Java;
* фреймворк для автоматизации сборки проектов Apache Maven;
* фреймворк для Java-платформы Spring Framework;
* язык программирования SQL;
* язык моделирования UML;
* СУБД MySQL Community Server 8.0;
* СУБД MongoDB Community Server 4.0.9;
* программный инструмент моделирования StarUML 2.8.1;
* инструментарий для проектирования баз данных и программных систем Sybase PowerDesigner 16.5.

1. Требования к программному и аппаратному обеспечению
2. Реализация
   1. Структура системы

Структура системы представлена на диаграмме компонентов на рис. 5.1.

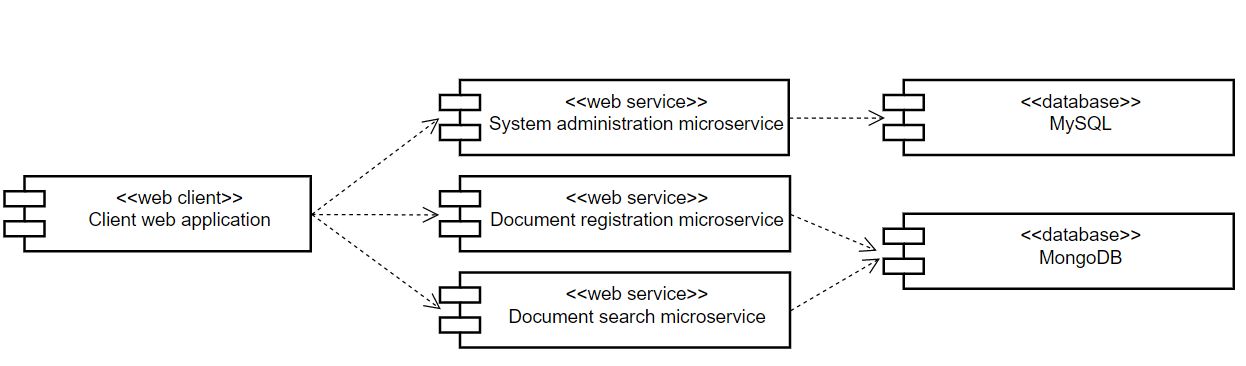


Рис. 5.1. Диаграмма компонентов системы обработки документов

Компонент **System administration microservice** содержит функциональность, связанную с администрированием системы. Его структура представлена на диаграмме компонентов на рис. 5.2.

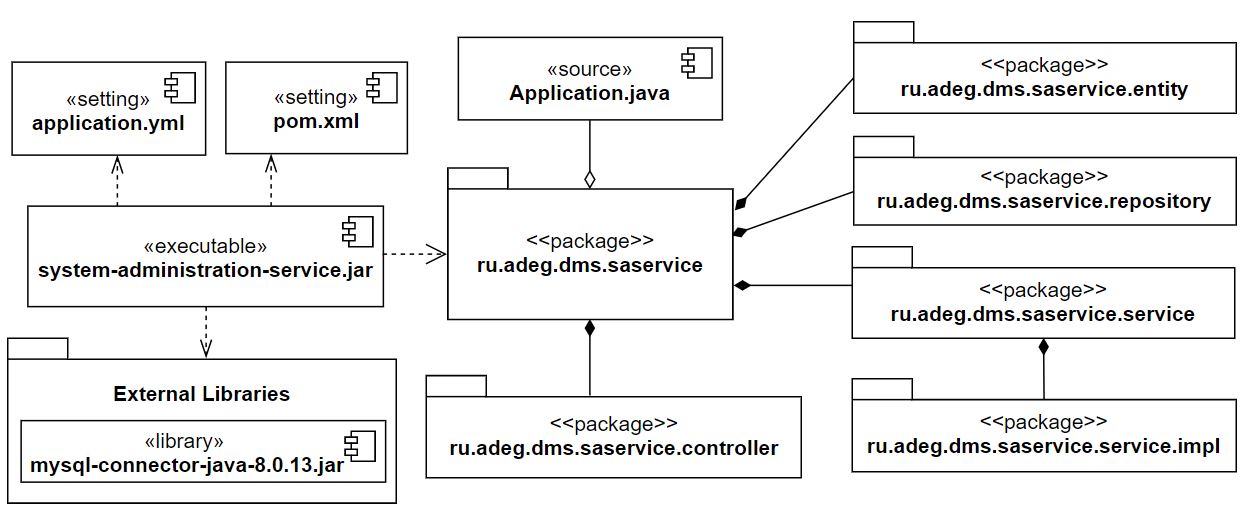


Рис. 5.2. Диаграмма компонентов микросервиса администрирования системы

Пакет **ru.adeg.dms.saservice.entity** содержит классы данных, которые соответствуют сущностям базы данных.

Пакет **ru.adeg.dms.saservice.repository** содержит интерфейсы, определяющие методы доступа к данным.

Пакет **ru.adeg.dms.saservice.service** содержит интерфейсы, определяющие методы бизнес логики. Классы, реализующие эти интерфейсы, находятся в пакете **ru.adeg.dms.saservice.service.impl.**

Пакет **ru.adeg.dms.saservice.controller** содержит классы с логикой обработки HTTP-запросов к сервису.

Диаграммы компонентов пакетов рассматриваемого микросервиса представлены в приложении 1. Зависимости между компонентами пакетов на примере сущности Attribute представлены на диаграмме компонентов на рис. 5.3.

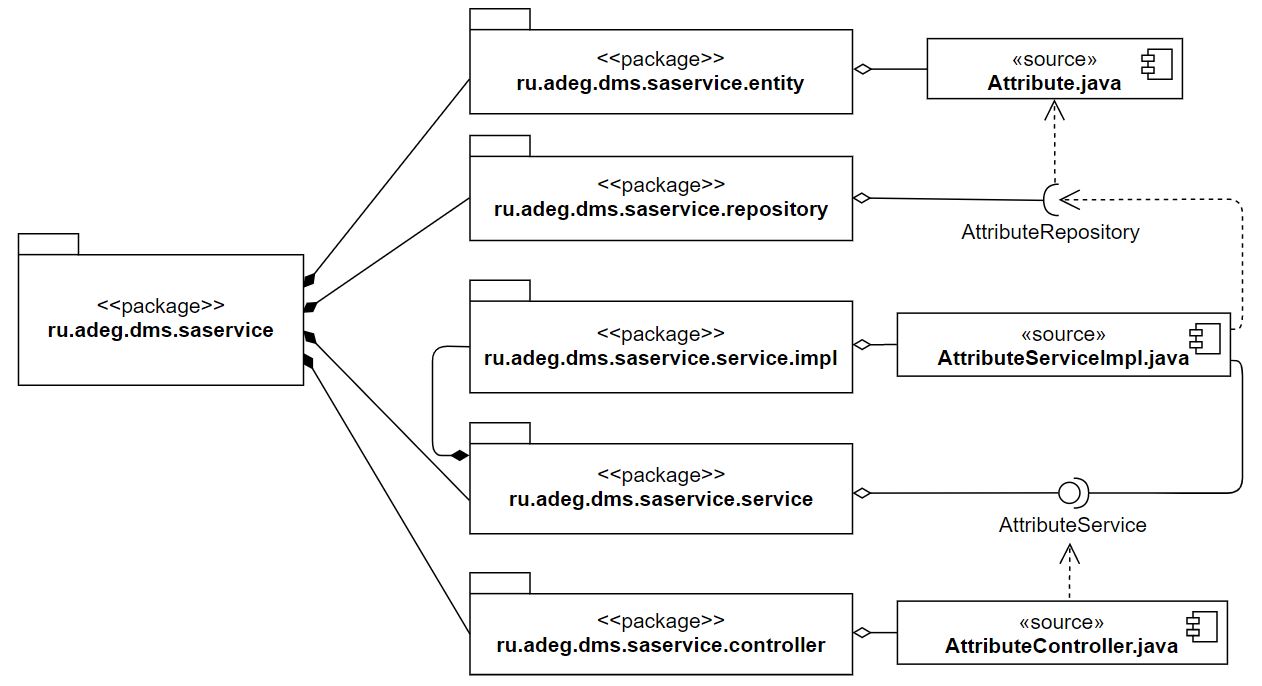


Рис. 5.3. Диаграмма компонентов, отражающая зависимости между компонентами различных пакетов

Компонент **Document registration microservice** содержит функциональность, связанную с загрузкой и регистрацией документов.

Компонент **Document search microservice** содержит функциональность, связанную с поиском документов.

Микросервисы загрузки и поиска документов имеют структуру, схожую со структурой микросервиса администрирования системы.

**Client web application** – веб-приложение, содержащее пользовательский интерфейс, логику отправки HTTP-запросов к сервисам системы и обработки ответов от них.

Для хранения данных микросервиса администрирования системы используется база данных MySQL, для хранения метаданных загружаемых документов используется документоориентированная база данных MongoDB.

* 1. Модели данных
     1. Реляционная модель данных

На основе анализа реляционной модели данных спроектирована база данных. Логическая модель базы данных в нотации Баркера представлена на рис. 5.4.

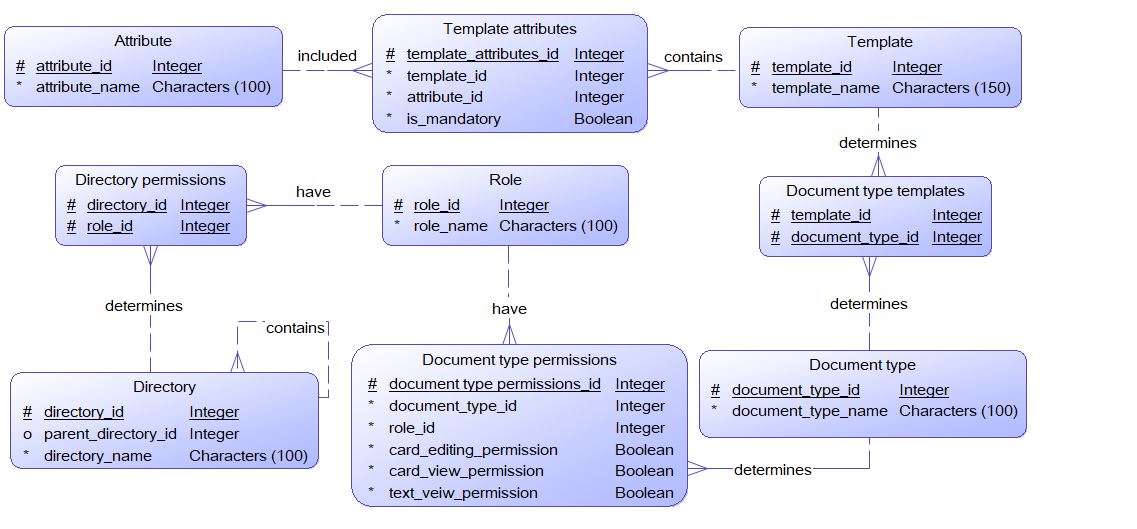


Рис. 5.4. Логическая модель базы данных в нотации Баркера

Листинг SQL-скрипта для создания базы данных приведен в приложении 2. Описание сущностей базы данных содержится в приложении 3.

* + 1. Документоориентированная модель данных

Документоориентированная база данных (ДБД) хранит данные в коллекциях, состоящих из объектов (документов) в формате JSON, которые не имеют заранее определенного формата и схемы. На основе анализа модели данных спроектирована база данных, состоящая из одной коллекции, в которой хранятся объекты следующего вида:

{

\_id: ‘…’,

upload\_date: ‘…’,

directory\_name: ‘…’,

document\_type: ‘…’,

user: ‘…’,

file\_path: ‘…’,

text: ‘…’,

attributes:

{

attribute\_name: ‘attribute\_value’,

attribute\_name: ‘attribute\_value’,

…

}

permissions:

{

card\_editing: [‘role\_name’, ‘role\_name’, …],

card\_view: [‘role\_name’, ‘role\_name’, …]

text\_view: [‘role\_name’, ‘role\_name’, …]

delete: [‘role\_name’, ‘role\_name’, …]

}

}

Описание полей документов содержится в табл. 5.1.

Таблица 5.1. Описание полей объектов, хранящихся в ДБД

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Описание** |
| \_id | Object ID | Уникальный идентификатор документа,  который генерируется автоматически. Используется в качестве регистрационного номера документа. |
| upload\_date | Date | Дата и время загрузки документа |
| directory\_name | String | Каталог, в котором хранится документ |
| document\_type | String | Вид документа |
| user | String | ФИО пользователя, загрузившего документ |
| file\_path | String | Путь к файлу документа в файловом хранилище |
| text | String | Содержимое документа для полнотекстового поиска |
| attributes | Object | Набор атрибутов и их значений |
| permissions | Object | Набор возможных прав на документ |
| card\_editing | Array | Набор имен ролей, которые имеют право редактирования карточки данного документа |
| card\_view | Array | Набор имен ролей, которые имеют право просмотра карточки данного документа |
| text\_view | Array | Набор имен ролей, которые имеют право просмотра текста данного документа |
| delete | Array | Набор имен ролей, которые имеют право удаления данного документа |

* 1. Диаграмма классов

В соответствии с моделью данных микросервиса администрирования системы реализованы классы данных. Структура и взаимосвязи между ними представлены на диаграмме классов в приложении 4. Листинг классов данных приведен в приложении 5.

1. Интерфейс пользователя
2. Web API документация
3. Тестирование

Заключение

В ходе данной работы спроектирована система обработки документов, предоставляющая следующую функциональность:

* загрузка и регистрация документов;
* представление документов в виде набора значений атрибутов;
* хранение документов;
* хранение атрибутов документов;
* атрибутивный поиск документов;
* полнотекстовый поиск документов;
* просмотр текста документов;
* просмотр карточек документов;
* удаление документов;
* редактирование карточек документов;
* создание шаблонов для каждого вида документов;
* создание каталогов;
* разграничение прав доступа.

Выполнена практическая реализация спроектированных модулей и разработан прототип клиентского приложения, взаимодействующего с системой.

Проведено модульное и интеграционное тестирование.

Список литературы

Приложение 1. Диаграммы компонентов микросервиса администрирования системы

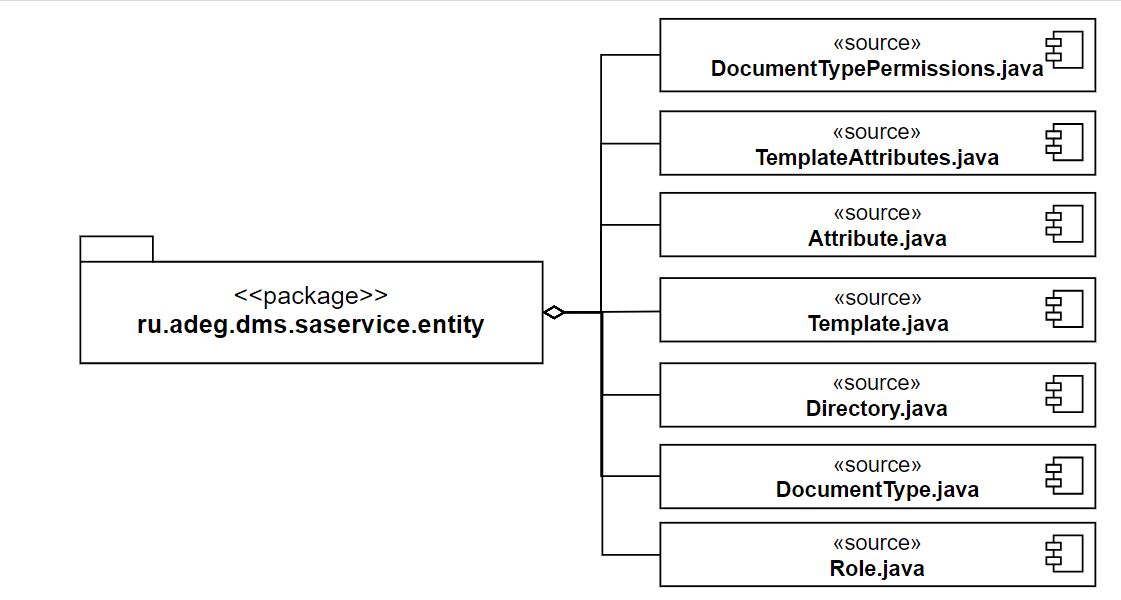


Рис. 1. Диаграмма компонентов пакета ru.adeg.dms.saservice.entity

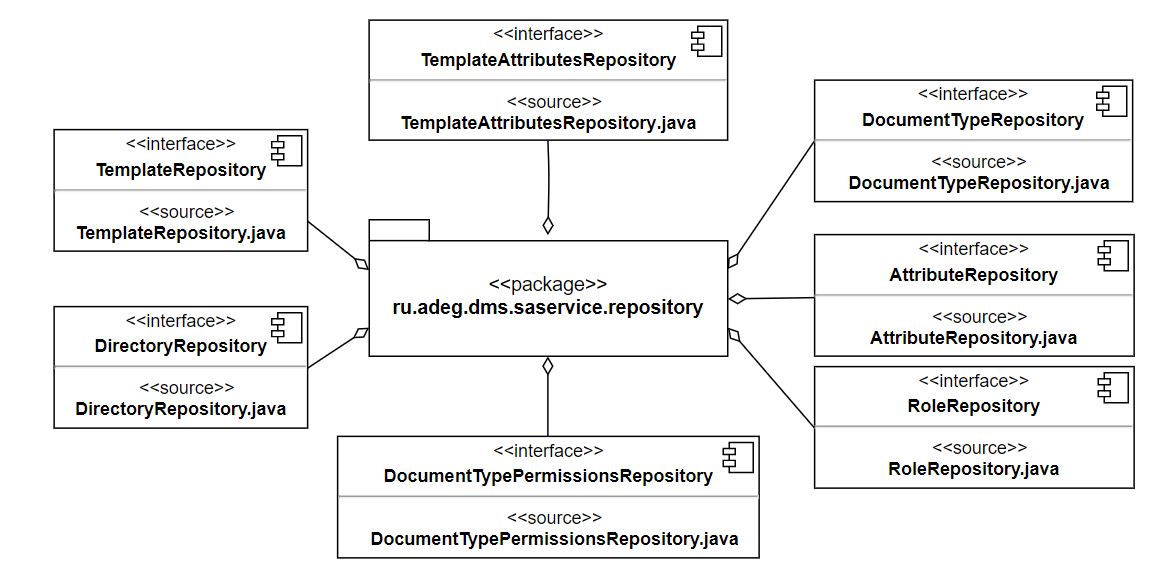


Рис. 2. Диаграмма компонентов пакета ru.adeg.dms.saservice.repository

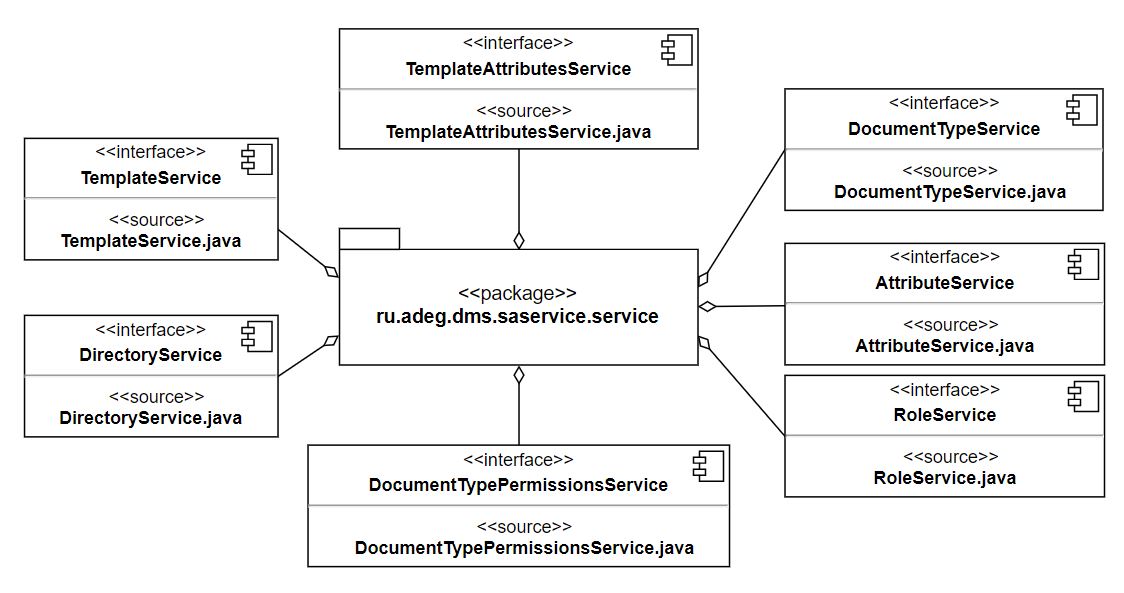


Рис. 3. Диаграмма компонентов пакета ru.adeg.dms.saservice.service

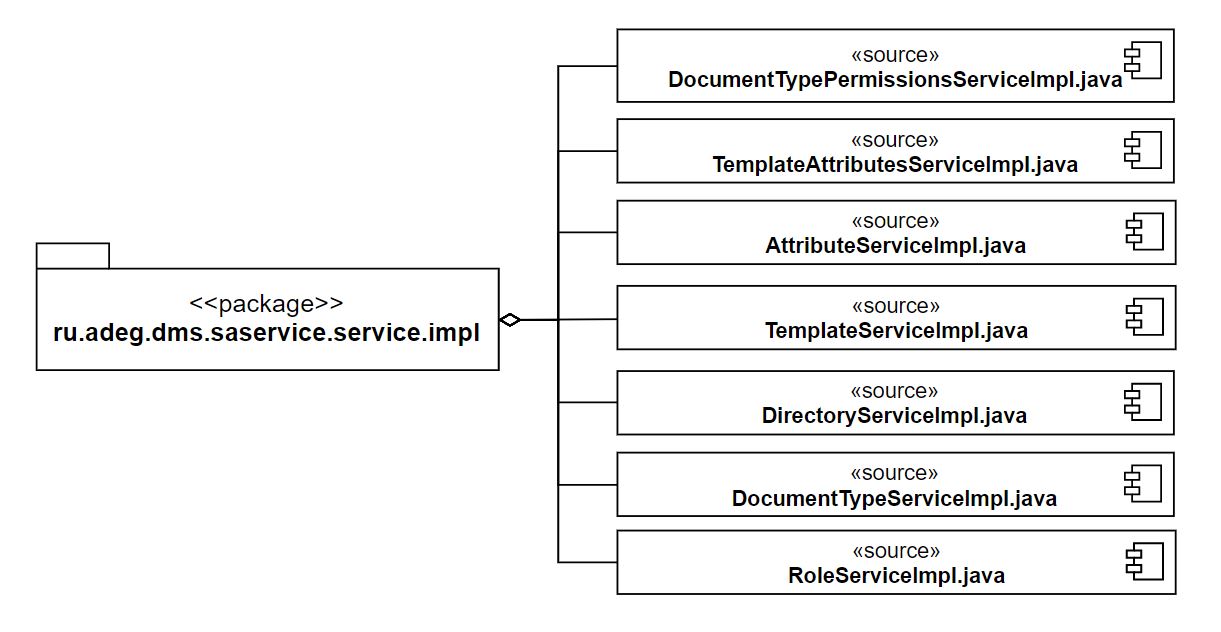


Рис. 4. Диаграмма компонентов пакета ru.adeg.dms.saservice.service.impl

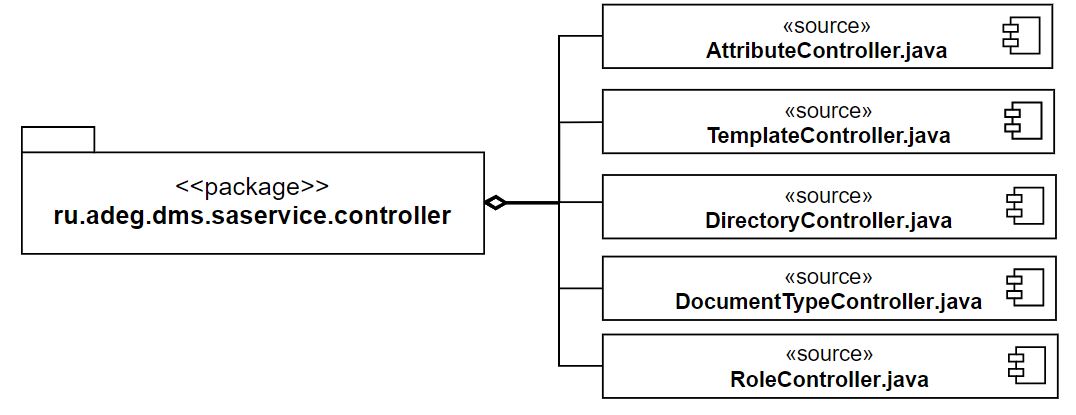


Рис. 5. Диаграмма компонентов пакета ru.adeg.dms.saservice.service.controller

Приложение 2. SQL-скрипт для создания базы данных

CREATE SCHEMA `documents`;

CREATE TABLE `documents`.`attribute` (

`attribute\_id` int(11) NOT NULL AUTO\_INCREMENT,

`attribute\_name` varchar(100) NOT NULL,

PRIMARY KEY (`attribute\_id`),

UNIQUE KEY `attribute\_name\_UNIQUE` (`attribute\_name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`template` (

`template\_id` int(11) NOT NULL AUTO\_INCREMENT,

`template\_name` varchar(150) NOT NULL,

PRIMARY KEY (`template\_id`),

UNIQUE KEY `template\_name\_UNIQUE` (`template\_name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`template\_attributes` (

`template\_attributes\_id` int(11) NOT NULL,

`template\_id` int(11) NOT NULL,

`attribyte\_id` int(11) NOT NULL,

`is\_mandatory` tinyint(4) NOT NULL,

PRIMARY KEY (`template\_attributes\_id`),

UNIQUE KEY `templ\_attr\_UNIQUE` (`template\_id`,`attribyte\_id`),

KEY `template\_fk\_idx` (`template\_id`),

KEY `attribute\_fk\_idx` (`attribyte\_id`),

CONSTRAINT `attrib\_fk` FOREIGN KEY (`attribyte\_id`) REFERENCES `attribute` (`attribute\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT `templ\_fk` FOREIGN KEY (`template\_id`) REFERENCES `template` (`template\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`document\_type` (

`document\_type\_id` int(11) NOT NULL AUTO\_INCREMENT,

`document\_type\_name` varchar(100) NOT NULL,

PRIMARY KEY (`document\_type\_id`),

UNIQUE KEY `document\_type\_name\_UNIQUE` (`document\_type\_name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`document\_type\_templates` (

`template\_id` int(11) NOT NULL,

`document\_type\_id` int(11) NOT NULL,

PRIMARY KEY (`template\_id`,`document\_type\_id`),

KEY `doc\_type\_fk\_idx` (`document\_type\_id`),

CONSTRAINT `doc\_type\_fk` FOREIGN KEY (`document\_type\_id`) REFERENCES `document\_type` (`document\_type\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT `template\_fk` FOREIGN KEY (`template\_id`) REFERENCES `template` (`template\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`role` (

`role\_id` int(11) NOT NULL AUTO\_INCREMENT,

`role\_name` varchar(100) NOT NULL,

PRIMARY KEY (`role\_id`),

UNIQUE KEY `role\_name\_UNIQUE` (`role\_name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`document\_type\_permissions` (

`document\_type\_permissions\_id` int(11) NOT NULL AUTO\_INCREMENT,

`document\_type\_id` int(11) NOT NULL,

`role\_id` int(11) NOT NULL,

`card\_editing\_permission` tinyint(4) NOT NULL,

`card\_view\_permission` tinyint(4) NOT NULL,

`text\_view\_permission` tinyint(4) NOT NULL,

`delete\_document\_permission` tinyint(4) NOT NULL,

PRIMARY KEY (`document\_type\_permissions\_id`),

UNIQUE KEY `document\_type\_perm\_UNIQUE` (`document\_type\_id`,`role\_id`),

KEY `doc\_type\_perm\_fk\_idx` (`document\_type\_id`),

KEY `role\_perm\_fk\_idx` (`role\_id`),

CONSTRAINT `doc\_type\_perm\_fk` FOREIGN KEY (`document\_type\_id`) REFERENCES `document\_type` (`document\_type\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT `role\_perm\_fk` FOREIGN KEY (`role\_id`) REFERENCES `role` (`role\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`directory` (

`directory\_id` int(11) NOT NULL AUTO\_INCREMENT,

`parent\_directory\_id` int(11) DEFAULT NULL,

`directory\_name` varchar(100) NOT NULL,

PRIMARY KEY (`directory\_id`),

KEY `parent\_dir\_fk\_idx` (`parent\_directory\_id`),

CONSTRAINT `parent\_dir\_fk` FOREIGN KEY (`parent\_directory\_id`) REFERENCES `directory` (`directory\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

CREATE TABLE `documents`.`directory\_permissions` (

`directory\_id` int(11) NOT NULL,

`role\_id` int(11) NOT NULL,

PRIMARY KEY (`directory\_id`,`role\_id`),

KEY `role\_dir\_perm\_fk\_idx` (`role\_id`),

CONSTRAINT `dir\_perm\_fk` FOREIGN KEY (`directory\_id`) REFERENCES `directory` (`directory\_id`),

CONSTRAINT `role\_dir\_perm\_fk` FOREIGN KEY (`role\_id`) REFERENCES `role` (`role\_id`) ON DELETE CASCADE ON UPDATE CASCADE

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

Приложение 3. Описание сущностей базы данных

Таблица 1. Attribute

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| attribute\_id | Уникальный идентификатор атрибута |
| attribute\_name | Наименование атрибута |

Таблица 2. Template

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| template\_id | Уникальный идентификатор шаблона |
| template\_name | Наименование шаблона |

Таблица 3. Template attributes

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| template\_attributes\_id | Уникальный идентификатор сущности |
| template\_id | Уникальный идентификатор шаблона |
| attribute\_id | Уникальный идентификатор атрибута |
| is\_mandatory | Обязательность атрибута для данного шаблона |

Таблица 4. Document type

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| document\_type\_id | Уникальный идентификатор вида документа |
| document\_type\_name | Наименование вида документа |

Таблица 5. Document type templates

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| template\_id | Уникальный идентификатор шаблона |
| document\_type\_id | Уникальный идентификатор вида документа |

Таблица 6. Role

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| role\_id | Уникальный идентификатор роли |
| role\_name | Наименование роли |

Таблица 7. Document type permissions

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| document\_type\_permissions\_id | Уникальный идентификатор сущности |
| document\_type\_id | Уникальный идентификатор вида документа |
| role\_id | Уникальный идентификатор роли |
| card\_editing\_permission | Наличие у роли права редактирования карточки документов данного вида |
| card\_view\_permission | Наличие у роли права просмотра карточки документов данного вида |
| text\_view\_permission | Наличие у роли права просмотра текста документов данного вида |
| delete\_document\_permission | Наличие у роли права удаления документов данного вида |

Таблица 8. Directory

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| directory\_id | Уникальный идентификатор каталога |
| parent\_directory\_id | Уникальный идентификатор родительского каталога |
| directory\_name | Наименование каталога |

Таблица 9. Directory permissions

|  |  |
| --- | --- |
| **Атрибут** | **Описание** |
| directory\_id | Уникальный идентификатор каталога |
| role\_id | Уникальный идентификатор роли |

Приложение 4. Диаграмма классов микросервиса администрирования системы

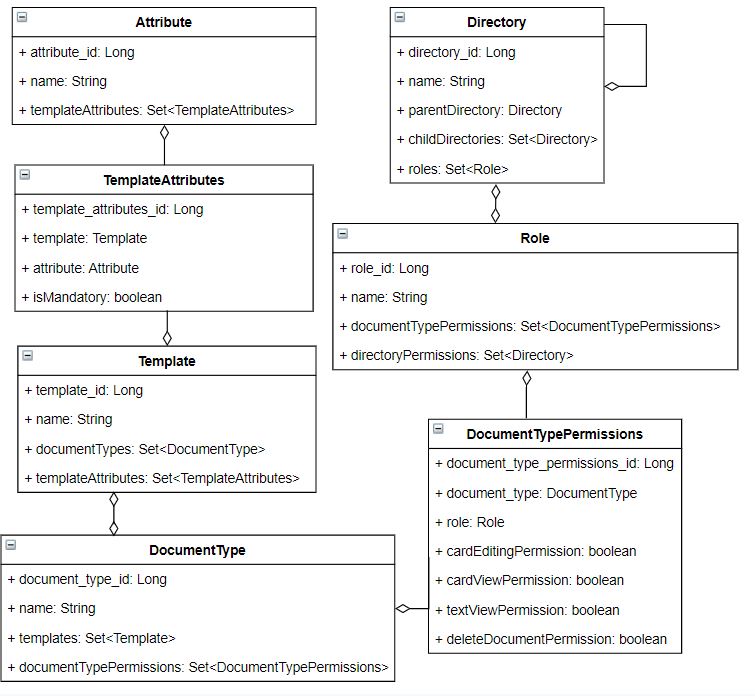


Рис. 1. Диаграмма классов микросервиса администрирования системы

Приложение 5. Листинг классов данных микросервиса администрирования системы

package ru.adeg.dms.saservice.entity;

import javax.persistence.\*;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import java.util.HashSet;

import java.util.Objects;

import java.util.Set;

@Entity

@Table(name = "attribute")

public class Attribute

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long attribute\_id;

@Column(name = "attribute\_name", nullable = false, unique = true)

private String name;

@OneToMany(mappedBy = "attribute", orphanRemoval = true, cascade = CascadeType.ALL)

private Set<TemplateAttributes> templateAttributes = new HashSet<TemplateAttributes>(0);

public Attribute() {}

public Attribute(String name) {

this.name = name;

}

public void addTemplate(Template template, Boolean isMandotory) {

TemplateAttributes temp = new TemplateAttributes(template, this, isMandotory);

templateAttributes.add(temp);

template.getTemplateAttributes().add(temp);

}

public Long getId() {

return attribute\_id;

}

public void setId(Long attribute\_id) {

this.attribute\_id = attribute\_id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Set<TemplateAttributes> getTemplateAttributes() {

return templateAttributes;

}

public void setTemplateAttributes(Set<TemplateAttributes> templateAttributes) {

this.templateAttributes = templateAttributes;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Attribute)) return false;

Attribute attribute = (Attribute) o;

return Objects.equals(getName(), attribute.getName());

}

@Override

public int hashCode() {

return Objects.hash(getName());

}

}

package ru.adeg.dms.saservice.entity;

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Objects;

import java.util.Set;

@Entity

@Table(name = "directory")

public class Directory {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long directory\_id;

@Column(name = "directory\_name", nullable = false)

private String name;

@ManyToOne(cascade=CascadeType.ALL)

@JoinColumn(name = "parent\_directory\_id", referencedColumnName = "directory\_id")

private Directory parentDirectory;

@OneToMany(mappedBy = "parentDirectory", orphanRemoval = true, cascade=CascadeType.ALL)

private Set<Directory> childDirectories;

@ManyToMany(fetch = FetchType.LAZY, cascade=CascadeType.ALL)

@JoinTable(name = "directory\_permissions",

joinColumns = { @JoinColumn(name = "directory\_id") },

inverseJoinColumns = { @JoinColumn(name = "role\_id") })

private Set<Role> roles = new HashSet<Role>();

public Directory() {}

public Directory(String name) {

this.name = name;

}

public Directory(String name, Directory parentDirectory) {

this.name = name;

this.parentDirectory = parentDirectory;

}

public Directory(String name, Directory parentDirectory, Set<Directory> childDirectories) {

this.name = name;

this.parentDirectory = parentDirectory;

this.childDirectories = childDirectories;

}

public Long getId() {

return directory\_id;

}

public void setId(Long id) {

this.directory\_id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Directory getParentDirectory() {

return parentDirectory;

}

public void setParentDirectory(Directory parentDirectory) {

this.parentDirectory = parentDirectory;

}

public Set<Directory> getChildDirectories() {

return childDirectories;

}

public void setChildDirectories(Set<Directory> childDirectories) {

this.childDirectories = childDirectories;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Directory)) return false;

Directory directory = (Directory) o;

return getId().equals(directory.getId()) &&

getName().equals(directory.getName()) &&

Objects.equals(getParentDirectory(), directory.getParentDirectory()) &&

Objects.equals(getChildDirectories(), directory.getChildDirectories());

}

@Override

public int hashCode() {

return Objects.hash(getId(), getName(), getParentDirectory(), getChildDirectories());

}

}

package ru.adeg.dms.saservice.entity;

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Objects;

import java.util.Set;

@Entity

@Table(name = "document\_type")

public class DocumentType {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long document\_type\_id;

@Column(name = "document\_type\_name", nullable = false)

private String name;

@ManyToMany(fetch = FetchType.LAZY, cascade=CascadeType.ALL)

@JoinTable(name = "document\_type\_templates",

joinColumns = { @JoinColumn(name = "document\_type\_id") },

inverseJoinColumns = { @JoinColumn(name = "template\_id") })

private Set<Template> templates = new HashSet<Template>();

@OneToMany(mappedBy = "document\_type", orphanRemoval = true, cascade={CascadeType.REMOVE, CascadeType.MERGE})

private Set<DocumentTypePermissions> documentTypePermissions = new HashSet<DocumentTypePermissions>();

public DocumentType() {}

public DocumentType(String name) {

this.name = name;

}

public DocumentType(String name, Set<Template> templates, Set<DocumentTypePermissions> documentTypePermissions) {

this.name = name;

this.templates = templates;

this.documentTypePermissions = documentTypePermissions;

}

public Long getId() {

return document\_type\_id;

}

public void setId(Long id) {

this.document\_type\_id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Set<Template> getTemplates() {

return templates;

}

public void setTemplates(Set<Template> templates) {

this.templates = templates;

}

public Set<DocumentTypePermissions> getDocumentTypePermissions() {

return documentTypePermissions;

}

public void setDocumentTypePermissions(Set<DocumentTypePermissions> documentTypePermissions) {

this.documentTypePermissions = documentTypePermissions;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof DocumentType)) return false;

DocumentType that = (DocumentType) o;

return getName().equals(that.getName());

}

@Override

public int hashCode() {

return Objects.hash(getName());

}

}

package ru.adeg.dms.saservice.entity;

import javax.persistence.\*;

import java.util.Objects;

@Entity

@Table(name = "document\_type\_permissions", uniqueConstraints = { @UniqueConstraint( columnNames = { "document\_type\_id", "role\_id" } ) })

public class DocumentTypePermissions {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long document\_type\_permissions\_id;

@ManyToOne(cascade = {CascadeType.REMOVE, CascadeType.MERGE})

@JoinColumn(name = "document\_type\_id", referencedColumnName = "document\_type\_id")

private DocumentType document\_type;

@ManyToOne(cascade = {CascadeType.REMOVE, CascadeType.MERGE})

@JoinColumn(name = "role\_id", referencedColumnName = "role\_id")

private Role role;

@Column(name = "card\_editing\_permission", nullable = false, columnDefinition = "TINYINT(1)")

private boolean cardEditingPermission;

@Column(name = "card\_view\_permission", nullable = false, columnDefinition = "TINYINT(1)")

private boolean cardViewPermission;

@Column(name = "text\_view\_permission", nullable = false, columnDefinition = "TINYINT(1)")

private boolean textViewPermission;

@Column(name = "delete\_document\_permission", nullable = false, columnDefinition = "TINYINT(1)")

private boolean deleteDocumentPermission;

public DocumentTypePermissions() {

}

public DocumentTypePermissions(DocumentType document\_type, Role role, boolean cardEditingPermission, boolean cardViewPermission, boolean textViewPermission, boolean deleteDocumentPermission) {

this.document\_type = document\_type;

this.role = role;

this.cardEditingPermission = cardEditingPermission;

this.cardViewPermission = cardViewPermission;

this.textViewPermission = textViewPermission;

this.deleteDocumentPermission = deleteDocumentPermission;

}

public DocumentType getDocument\_type() {

return document\_type;

}

public void setDocument\_type(DocumentType document\_type) {

this.document\_type = document\_type;

}

public Role getRole() {

return role;

}

public void setRole(Role role) {

this.role = role;

}

public boolean isCardEditingPermission() {

return cardEditingPermission;

}

public void setCardEditingPermission(boolean cardEditingPermission) {

this.cardEditingPermission = cardEditingPermission;

}

public boolean isCardViewPermission() {

return cardViewPermission;

}

public void setCardViewPermission(boolean cardViewPermission) {

this.cardViewPermission = cardViewPermission;

}

public boolean isTextViewPermission() {

return textViewPermission;

}

public void setTextViewPermission(boolean textViewPermission) {

this.textViewPermission = textViewPermission;

}

public boolean isDeleteDocumentPermission() {

return deleteDocumentPermission;

}

public void setDeleteDocumentPermission(boolean deleteDocumentPermission) {

this.deleteDocumentPermission = deleteDocumentPermission;

}

public Long getDocument\_type\_permissions\_id() {

return document\_type\_permissions\_id;

}

public void setDocument\_type\_permissions\_id(Long document\_type\_permissions\_id) {

this.document\_type\_permissions\_id = document\_type\_permissions\_id;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof DocumentTypePermissions)) return false;

DocumentTypePermissions that = (DocumentTypePermissions) o;

return getDocument\_type().equals(that.getDocument\_type()) &&

getRole().equals(that.getRole());

}

@Override

public int hashCode() {

return Objects.hash(getDocument\_type(), getRole());

}

}

package ru.adeg.dms.saservice.entity;

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Objects;

import java.util.Set;

@Entity

@Table(name = "role")

public class Role {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long role\_id;

@Column(name = "role\_name", nullable = false, unique = true)

private String name;

@OneToMany(mappedBy = "role", orphanRemoval = true, cascade={CascadeType.REMOVE, CascadeType.MERGE})

private Set<DocumentTypePermissions> documentTypePermissions = new HashSet<DocumentTypePermissions>();

@ManyToMany (mappedBy = "roles", cascade=CascadeType.ALL,

fetch = FetchType.EAGER)

private Set<Directory> directoryPermissions = new HashSet<Directory>();

public Role() {}

public Role(String name) {

this.name = name;

}

public Role(String name, Set<DocumentTypePermissions> documentTypePermissions) {

this.name = name;

this.documentTypePermissions = documentTypePermissions;

}

public Long getId() {

return role\_id;

}

public void setId(Long id) {

this.role\_id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Set<DocumentTypePermissions> getDocumentTypePermissions() {

return documentTypePermissions;

}

public void setDocumentTypePermissions(Set<DocumentTypePermissions> documentTypePermissions) {

this.documentTypePermissions = documentTypePermissions;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Role)) return false;

Role role = (Role) o;

return getName().equals(role.getName());

}

@Override

public int hashCode() {

return Objects.hash(getName());

}

}

package ru.adeg.dms.saservice.entity;

import javax.persistence.\*;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.Table;

import java.util.HashSet;

import java.util.Iterator;

import java.util.Objects;

import java.util.Set;

@Entity(name = "Template")

@Table(name = "template", schema = "documents")

public class Template{

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long template\_id;

@Column(name = "template\_name", nullable = false, unique = true)

private String name;

@ManyToMany (mappedBy = "templates", cascade=CascadeType.ALL,

fetch = FetchType.EAGER)

private Set<DocumentType> documentTypes = new HashSet<>();

@OneToMany(mappedBy = "template", orphanRemoval = true, cascade = CascadeType.ALL)

private Set<TemplateAttributes> templateAttributes = new HashSet<TemplateAttributes>(0);

public Template() {}

public Template(String name) {

this.name = name;

}

public void addAttribute(Attribute attribute, Boolean isMandotory) {

TemplateAttributes temp = new TemplateAttributes(this, attribute, isMandotory);

templateAttributes.add(temp);

attribute.getTemplateAttributes().add(temp);

}

public void removeAttribute(Attribute attribute) {

for (Iterator<TemplateAttributes> iterator = templateAttributes.iterator();

iterator.hasNext(); ) {

TemplateAttributes templatesAttributes = iterator.next();

if (templatesAttributes.getTemplate().equals(this) &&

templatesAttributes.getAttribute().equals(attribute)) {

iterator.remove();

templatesAttributes.getAttribute().getTemplateAttributes().remove(templatesAttributes);

templatesAttributes.setTemplate(null);

templatesAttributes.setAttribute(null);

}

}

}

public Long getId() {

return template\_id;

}

public void setId(Long id) {

this.template\_id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Set<DocumentType> getDocumentTypes() {

return documentTypes;

}

public void setDocumentTypes(Set<DocumentType> documentTypes) {

this.documentTypes = documentTypes;

}

public Set<TemplateAttributes> getTemplateAttributes() {

return templateAttributes;

}

public void setTemplateAttributes(Set<TemplateAttributes> templateAttributes) {

this.templateAttributes = templateAttributes;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Template)) return false;

Template template = (Template) o;

return Objects.equals(getName(), template.getName());

}

@Override

public int hashCode() {

return Objects.hash(getName());

}

}

package ru.adeg.dms.saservice.entity;

import javax.persistence.\*;

import java.io.Serializable;

import java.util.Objects;

@Entity

@Table(name = "template\_attributes", uniqueConstraints = { @UniqueConstraint( columnNames = { "template\_id", "attribute\_id" } ) } )

public class TemplateAttributes implements Serializable {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long template\_attributes\_id;

@ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)

@JoinColumn(name = "template\_id", referencedColumnName = "template\_id")

private Template template;

@ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL)

@JoinColumn(name = "attribute\_id", referencedColumnName = "attribute\_id")

private Attribute attribute;

@Column(name = "is\_mandatory", nullable = false, columnDefinition = "TINYINT(1)")

private boolean isMandatory;

public TemplateAttributes() {}

public TemplateAttributes(Template template, Attribute attribute, boolean isMandatory) {

this.template = template;

this.attribute = attribute;

this.isMandatory = isMandatory;

}

public Template getTemplate() {

return template;

}

public void setTemplate(Template template) {

this.template = template;

}

public Attribute getAttribute() {

return attribute;

}

public void setAttribute(Attribute attribute) {

this.attribute = attribute;

}

public boolean isMandatory() {

return isMandatory;

}

public void setMandatory(boolean mandatory) {

isMandatory = mandatory;

}

public Long getTemplate\_attributes\_id() {

return template\_attributes\_id;

}

public void setTemplate\_attributes\_id(Long template\_attributes\_id) {

this.template\_attributes\_id = template\_attributes\_id;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof TemplateAttributes)) return false;

TemplateAttributes that = (TemplateAttributes) o;

return Objects.equals(getTemplate(), that.getTemplate()) &&

Objects.equals(getAttribute(), that.getAttribute());

}

@Override

public int hashCode() {

return Objects.hash(getTemplate(),getAttribute());

}

}