# **Содержание**

[Введение 5](#_Toc513157240)

[1 Описание предметной области 6](#_Toc513157241)

[2 Обзор методов и алгоритмов решения поставленной задачи 8](#_Toc513157242)

[3 Функциональное моделирование системы на основе стандарта IDEF0 11](#_Toc513157243)

[4 Информационная модель системы и ее описание 14](#_Toc513157244)

[5 Разработка моделей проектируемой системы на основе стандарта UML 2.0 17](#_Toc513157245)

[5.1 Диаграмма вариантов использования (прецедентов) (use case diagram) 18](#_Toc513157251)

[5.2 Диаграмма классов (class diagram) 18](#_Toc513157252)

[5.3 Диаграмма последовательности (sequence diagram) 19](#_Toc513157253)

[5.4 Диаграмма состояний (statechart diagram) 19](#_Toc513157254)

[5.5 Диаграмма компонентов (component diagram) 19](#_Toc513157255)

[5.6 Диаграмма развертывания (deployment diagram) 20](#_Toc513157256)

[6 Описание алгоритмов, реализующих бизнес-логику проектируемой системы 22](#_Toc513157257)

[7 Руководство пользователя 24](#_Toc513157258)

[7.1 Назначение программы 24](#_Toc513157259)

[7.2 Условия выполнения 24](#_Toc513157260)

[7.3 Работа с веб-приложением 25](#_Toc513157261)

[8 Результаты тестирования системы и оценка выполнения поставленных задач 32](#_Toc513157262)

[Заключение 33](#_Toc513157263)

[Список использованных источников 34](#_Toc513157264)

[Приложение А (обязательное) Диаграммы модели разрабатываемого приложения в формате IDEF0 35](#_Toc513157265)

[Приложение Б (обязательное) UML-диаграммы разрабатываемой системы 40](#_Toc513157266)

[Приложение В (обязательное) Листинг кода скрипта генерации базы данных 44](#_Toc513157267)

[Приложение Г (обязательное) Листинг кода разрабатываемого приложения 48](#_Toc513157268)

# Введение

Веб-приложения – это специальный вид приложений, которые работают в глобальной сети Интернет или внутренней локальной сети по протоколу HTTP(S). Как правило, веб-приложения не требуют установки дополнительного программного обеспечения на стороне клиента, а вся логика, в основном, выполняется на стороне сервера.

Для отображения пользовательского интерфейса используется браузер – программа, способная распознавать язык разметки HTML (и сопутствующие технологии – таблицы стилей CSS, клиентский скриптовой язык программирования JavaScript и т.д.). 

Для взаимодействия клиента и сервера используется протокол HTTP, который работает по схеме «запрос-ответ». В момент, когда клиент хочет обратиться к серверу, он генерирует запрос, который отправляется серверу. Сервер обрабатывает этот запрос и подготавливает ресурсы, которые будут отправлены клиенту. После этого сервер генерирует ответ, в котором содержаться все необходимые данные и отправляет клиенту. Работа веб-приложений заключается в формировании необходимых данных как раз в момент подготовки ресурсов на сервере. Обычно в этот момент запускается некоторый программный код, который содержит определенную бизнес логику.

По сравнению с настольными приложениями, веб-приложения обладают более ограниченными возможностями по формированию пользовательского интерфейса и клиентской функциональности. Развитие веб-технологий доказало, что веб-приложения также могут реализовывать богатые сценарии и успешно конкурировать с настольными приложениями. Кроме того, за последние несколько лет очень активно развиваются технологии, позволяющие сделать веб-приложения еще более интерактивными.

Основная цель данного курсового проекта – создание веб-приложения с организацией взаимодействия с базой данных на объектно-ориентированном языке Java.

Для достижения цели была поставлена задача создать систему учета документооборота производственного подразделения. Уровень сложности этой задачи позволяет в достаточной мере ознакомиться с основными этапами разработки веб-приложений и приобрести определенные навыки программирования.

# Описание предметной области

Производственное подразделение – самостоятельное структурное подразделение, выполняющее определенные функции в системе управления предприятием. Обычно таким подразделением является финансовый отдел, планово-производственный отдел и др. Его структура и численность зависит от организационно-правовой формы предприятия, характера финансовой деятельности, объема производства, количества работающих на предприятии.

Производственное подразделение является частью единого механизма управления хозяйственной деятельностью, и поэтому оно тесно связано с другими службами предприятия. Так, в результате тесных контактов с бухгалтерией, структурному подразделению предприятиия представляются планы производства, списки кредиторов и дебиторов, документы по выплате зарплаты работникам, суммах денежных средств, находящихся на его счетах, и суммах предстоящих расходов. В свою очередь планово-финансовый отдел, обрабатывая эту информацию, анализируя ее, дает квалифицированную оценку платежеспособности предприятия, ликвидности его активов, кредитоспособности, составляет платежный календарь, готовит аналитические отчеты по другим параметрам финансового состояния предприятия и знакомит бухгалтерию с финансовыми планами и аналитическими отчетами об их выполнении, которая в своей ежедневной деятельности руководствуется этими сведениями.

От отдела маркетинга производственное подразделение получает планы по сбыту продукции и использует ее при планировании доходов и составлении оперативных финансовых планов. Для проведения успешной маркетинговой компании производственное подразделение обосновывает реализационные цены, утверждает систему уступок в цене контракта, анализирует сбытовые и маркетинговые издержки, осуществляет сравнительную оценку конкурентоспособности продукции предприятия, оптимизирует ее рентабельность, создавая, таким образом, условия для заключения крупных сделок.

Основными задачами структурных производственных подразделений в организации являются:

* финансирование затрат на производство продукции, капитальных вложений и других расходов;
* выполнение финансовых обязательств перед госбюджетом, банками, поставщиками, вышестоящими организациями, рабочими и служащими;
* изыскание путей увеличения денежных доходов, прибыли и повышения рентабельности;
* обеспечение сохранности оборотных средств и ускорение их оборачиваемости;
* осуществление контроля за эффективным использованием основных производственных фондов и инвестиций;
* организация совершенствование форм денежных расчетов с поставщиками, покупателями, рабочими и служащими, вышестоящей организацией, госбюджетом, банками.

Финансовый отдел получает от бухгалтерии, планового отдела, отделов снабжения и сбыта, технического отдела, отдела капитального строительства, отдела главного механика и других структурных подразделений и служб необходимые плановые и отчетные материалы; бухгалтерские отчеты и балансы; проекты и утвержденные планы выпуска товарной продукции и ее себестоимости, реализации продукции и прибыли; планы движения ТМЦ (производственных запасов, незавершенного производства и готовой продукции); расчеты экономической эффективности капитальных вложений и новой техники, технико-экономические показатели; сметы затрат на производство и расходование спецсредств.

Организация управление документооборотом – одна из отраслей управленческой деятельности, поэтому среди руководителей предприятий и их структурных подразделений становятся актуальными и приобретают широкое распространение решения для автоматизации документооборота. В настоящее время появилось немало информационных программных систем для управления документооборотом, значительно упрощающих все операции документооборота. Данные системы обеспечивают автоматизацию документооборота, электронную обработку документов и предоставляют эффективные механизмы организации документооборота на предприятии и управления документооборотом.

# Обзор методов и алгоритмов решения поставленной задачи

Решение поставленной задачи основывается на анализе процессов регистрации и учета документов, поступающих в планово-производственную службу предприятия, а также выделении из них тех задач, которые можно автоматизировать путем создания веб-приложения с доступом к базе данных.

В ходе реализации серверной части приложения принято решение организовать взаимодействие с базой данных MySQL-сервера посредством драйвера, предоставленного производителем СУБД.

В процессе разработки сервера поставлена задача реализовать функции отправки/получения данных из СУБД, их консолидации и формирования веб-страниц, отображаемых в браузере пользователя.

Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода.

Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос (портирование) приложения на любые базы данных SQL.

Hibernate обеспечивает прозрачную поддержку сохранности данных для POJO-объектов (стандартных Java-объектов, не унаследованных от какого-то специфического объекта и не реализующих никаких служебных интерфейсов сверх тех, которые нужны для бизнес-модели).

Сопоставление Java-классов с таблицами базы данных осуществляется с помощью конфигурационных XML-файлов или Java-аннотаций. При использовании файла XML Hibernate может генерировать скелет исходного кода для классов длительного хранения.

Hibernate поддерживает отображение пользовательских типов значений. Это делает возможными такие сценарии:

* переопределение типа по умолчанию SQL, Hibernate выбирает при отображении столбца свойства.
* проецирование перечисляемого типа Java на поле БД, будто они являются обычными свойствами.
* проецирование одного свойства в несколько колонок.

Hibernate обеспечивает использование SQL-подобного языка Hibernate Query Language (HQL), который позволяет выполнять SQL-подобные запросы, записанные рядом с объектами данных Hibernate.

Ниже приведен листинги XML-файлов конфигурации Hibernate и одного из XML-файлов сопоставления, используемых при разработке ПО в рамках данного курсового проекта:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ fin\_dept\_docs?zeroDateTimeBehavior=convertToNull</property>

<property name="hibernate.connection.username">dbadmin</property>

<property name="hibernate.connection.password">dbadmin</property>

<property name="hibernate.current\_session\_context\_class">thread</property>

<property name="hibernate.enable\_lazy\_load\_no\_trans">true</property>

<property name="hibernate.query.factory\_class">org.hibernate.hql.internal.classic. ClassicQueryTranslatorFactory</property>

<property name="hibernate.show\_sql">true</property>

<mapping resource="models/Initiator.hbm.xml"/>

<mapping resource="models/Status.hbm.xml"/>

<mapping resource="models/Document.hbm.xml"/>

<mapping resource="models/Type.hbm.xml"/>

<mapping resource="models/Performer.hbm.xml"/>

</session-factory>

</hibernate-configuration>

*Листинг XML-файла hibernate.cfg*

<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<!-- Generated 24.12.2019 15:13:28 by Hibernate Tools 4.3.1 -->

<hibernate-mapping>

<class name="models.Document" table="document" catalog="fin\_dept\_docs" optimistic-lock="version">

<id name="id" type="java.lang.Integer">

<column name="id" />

<generator class="identity" />

</id>

<many-to-one name="initiator" class="models.Initiator" fetch="select">

<column name="initiator\_id" not-null="true" />

</many-to-one>

<many-to-one name="performer" class="models.Performer" fetch="select">

<column name="performer\_id" not-null="true" />

</many-to-one>

<many-to-one name="status" class="models.Status" fetch="select">

<column name="status\_id" not-null="true" />

</many-to-one>

<many-to-one name="type" class="models.Type" fetch="select">

<column name="type\_id" not-null="true" />

</many-to-one>

<property name="inputNumber" type="string">

<column name="input\_number" length="10" not-null="true" />

</property>

<property name="regDate" type="date">

<column name="reg\_date" length="10" not-null="true" />

</property>

<property name="name" type="string">

<column name="name" length="200" not-null="true" />

</property>

<property name="deadline" type="date">

<column name="deadline" length="10" not-null="true" />

</property>

</class>

</hibernate-mapping>

# Функциональное моделирование системы на основе стандарта IDEF0

IDEF0 – методология функционального моделирования. С помощью наглядного графического языка IDEF0 изучаемая система предстает перед разработчиками и аналитиками в виде набора взаимосвязанных функций (функциональных блоков – в терминах IDEF0). Как правило, моделирование средствами IDEF0 является первым этапом изучения любой системы.

В основе методологии лежат четыре основных понятия:

Первым из них является понятие функционального блока (*Activity Box*). Функциональный блок графически изображается в виде прямоугольника (см. рис. 3.1) и олицетворяет собой некоторую конкретную функцию в рамках рассматриваемой системы. По требованиям стандарта название каждого функционального блока должно быть сформулировано в глагольном наклонении.

Каждая из четырех сторон функционального блока имеет своё определенное значение (роль), при этом:

* Верхняя сторона имеет значение «Управление» (Control);
* Левая сторона имеет значение «Вход» (Input);
* Правая сторона имеет значение «Выход» (Output);
* Нижняя сторона имеет значение «Механизм» (Mechanism).

Каждый функциональный блок в рамках единой рассматриваемой системы должен иметь свой уникальный идентификационный номер.

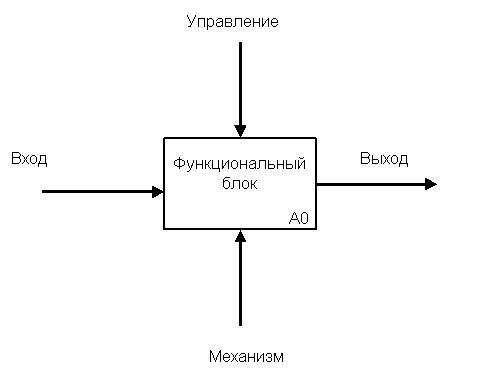
****

Рисунок 3.1 – Функциональный блок IDEF0

Вторым «китом» методологии IDEF0 является понятие интерфейсной дуги (*Arrow*). Также интерфейсные дуги часто называют потоками или стрелками. Интерфейсная дуга отображает элемент системы, который обрабатывается функциональным блоком или оказывает иное влияние на функцию, отображенную данным функциональным блоком.

Графическим отображением интерфейсной дуги является однонаправленная стрелка. Каждая интерфейсная дуга должна иметь свое уникальное наименование (*Arrow Label*). По требованию стандарта, наименование должно быть оборотом существительного.

С помощью интерфейсных дуг отображают различные объекты, в той или иной степени определяющие процессы, происходящие в системе. Такими объектами могут быть элементы реального мира (детали, вагоны, сотрудники и т.д.) или потоки данных и информации (документы, данные, инструкции и т.д.).

В зависимости от того, к какой из сторон подходит данная интерфейсная дуга, она носит название «входящей», «исходящей» или «управляющей». Кроме того, «источником» (началом) и «приемником» (концом) каждой функциональной дуги могут быть только функциональные блоки, при этом «источником» может быть только выходная сторона блока, а «приемником» любая из трех оставшихся.

Третьим основным понятием стандарта IDEF0 является декомпозиция (*Decomposition*). Принцип декомпозиции применяется при разбиении сложного процесса на составляющие его функции. При этом уровень детализации процесса определяется непосредственно разработчиком модели.

Декомпозиция позволяет постепенно и структурированно представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

Модель IDEF0 всегда начинается с представления системы как единого целого – одного функционального блока с интерфейсными дугами, простирающимися за пределы рассматриваемой области. Такая диаграмма с одним функциональным блоком называется контекстной диаграммой, и обозначается идентификатором «А-0».

В процессе декомпозиции, функциональный блок, который в контекстной диаграмме отображает систему как единое целое, подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня содержит функциональные блоки, отображающие главные подфункции функционального блока контекстной диаграммы и называется дочерней (*Child diagram*) по отношению к нему (каждый из функциональных блоков, принадлежащих дочерней диаграмме соответственно называется дочерним блоком – *Child Box*). В свою очередь, функциональный блок – предок называется родительским блоком по отношению к дочерней диаграмме (*Parent Box*), а диаграмма, к которой он принадлежит – родительской диаграммой (*Parent Diagram*). Каждая из подфункций дочерней диаграммы может быть далее детализирована путем аналогичной декомпозиции соответствующего ей функционального блока. Важно отметить, что в каждом случае декомпозиции функционального блока все интерфейсные дуги, входящие в данный блок, или исходящие из него фиксируются на дочерней диаграмме. Этим достигается структурная целостность IDEF0 – модели.

Последним из понятий IDEF0 является глоссарий (*Glossary*). Для каждого из элементов IDEF0: диаграмм, функциональных блоков, интерфейсных дуг существующий стандарт подразумевает создание и поддержание набора соответствующих определений, ключевых слов, повествовательных изложений и т.д., которые характеризуют объект, отображенный данным элементом. Этот набор называется глоссарием и является описанием сущности данного элемента.

IDEF0 модели разрабатываемой системы изображены на рис.А.1-А.5 [приложения А](#_Приложение_А_(обязательное)).

# Информационная модель системы и ее описание

Информационная модель – [модель](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C) объекта, представленная в виде [информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F), описывающей существенные для данного рассмотрения [параметры](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80) и переменные величины объекта, связи между ними, входы и выходы объекта и позволяющая путём подачи на модель информации об изменениях входных величин моделировать возможные состояния объекта.

Информационная модель (в широком, общенаучном смысле) – совокупность информации, характеризующая существенные свойства и состояния объекта, процесса, явления, а также взаимосвязь с внешним миром.

IDEF1X – это методология семантического моделирования данных. Она разработана с учетом следующих требований:

1. *Поддерживает разработку концептуальных схем*

Синтаксис IDEF1X поддерживает семантические конструкции, необходимые для разработки концептуальной схемы. Окончательная версия IDEF1X-модели обладает желаемыми характеристиками – непротиворечивостью, расширяемостью и адаптируемостью.

1. *Обеспечивает ясный язык*

IDEF1X имеет простую, ясную, непротиворечивую структуру и четкие семантические понятия. Синтаксис и семантика IDEF1X сравнительно легки для понимания, хотя и являются достаточно мощным средством.

1. *Проста для изучения*

Семантическое моделирование данных - новое понятие для многих пользователей IDEF1X. Проблема обучаемости этому языку является важным факторомом. Язык рассчитан на понимание и использование как профессиональными бизнесменами и системными аналитиками, так и администраторами данных и разработчиками баз данных. Он может служить эффективным средством коммуникации в коллективах, состоящих из различных специалистов.

1. *Надежно проверена на практике*

IDEF1X базируется на многолетнем опыте предшествующих методологий и тщательно проверена как в проектах ВВС, так и в промышленности.

1. *Возможность автоматизации*

IDEFlX-диаграммы могут создаваться большим числом графических программных пакетов. ВВС США на основе концептуальной схемы разработали активный трехсхемный словарь для построения прикладных программ и обработки запросов в распределенной неоднородной среде. Существует также коммерческое программное обеспечение, поддерживающее детализацию, анализ и управление конфигурацией IDEFlX-моделей.

Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после того, как все информационные ресурсы исследованы и решение о внедрении реляционной базы данных, как части корпоративной информационной системы, было принято.

Для разработки оптимальной структуры реляционной БД необходимо проанализировать составляющие ее элементы и отношения между ними, а затем создать нормализованные таблицы. Нормализация таблиц базы данных – первый шаг на пути проектирования структуры реляционной базы данных. База данных считается нормализованной, если ее таблицы представлены как минимум в третьей нормальной форме.

Главная цель нормализации базы данных – устранение избыточности и дублирования информации. При нормализации необходимо, чтобы любое значение хранилось в базе в одном экземпляре. Как следствие, значительно сокращается вероятность появления противоречивых данных, облегчается администрирование базы и обновление информации в ней, сокращается объем дискового пространства.

В ходе разработки структуры базы данных проектируемой системы все таблицы были приведены к 3 нормальной форме, т.е. все они отвечают следующим требованиям:

* запрет на повторяющиеся столбцы (содержащие одинаковую по смыслу информацию);
* запрет на множественные столбцы (содержащие значения типа список и т.п.);
* обязательное определение первичного ключа для таблицы;
* зависимость неключевых столбцов таблиц от первичного ключа в целом, но не от его части;
* независимость неключевых столбцов от других неключевых столбцов, а зависимость только от первичного ключа.

а) Сущность document– содержит в себе информацию о документе. Таблица важна для того, чтобы узнать характеристики документа: кто инициатор, название документа, описание, дата регистрации, исполнитель, дата готовности, статус документа, входящий номер.

б) Сущность performer – содержит информацию об исполнителях документов: должность, имя, телефон, ник.

в) Сущность type – тип документа.

г) Сущность initiator – содержит информацию об инициаторе документа (отдел предприятия).

д) Сущность status – это таблица, содержащая информацию о статусе документа.

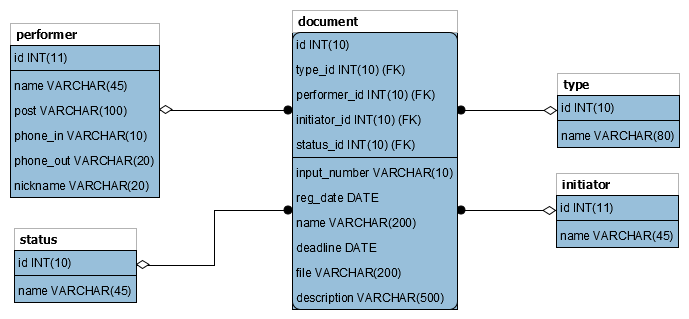


Рисунок 4.1 – Информационная модель проектируемой системы

# Разработка моделей проектируемой системы на основе стандарта UML 2.0

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. Язык UML одновременно является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения.

С помощью UML можно разработать детальный план создаваемой системы, содержащей не только ее концептуальные элементы, такие как системные функции и бизнес-процессы, но и конкретные особенности, например классы, написанные на специальных языках программирования, схемы баз данных и программные компоненты многократного использования.

Язык UML предназначен для опи­сания моделей. На UML можно содержательно описывать классы, объекты и компо­ненты в различных предметных областях, часто сильно отличающихся друг от друга.

Как любой язык, UML состоит из словаря и правил, позволяющие комбинировать входящие слова и получать осмысленные конструкции.

Моделирование необходимо для понимания системы. Обычно при этом единственной модели никогда не бывает достаточно. Поэтому приходиться разрабатывать большое количество взаимосвязанных моделей.

Словарь в UML включает 3 вида основных конструкций:

* Сущности – абстракции, являющиеся основными элементами модели.
* Отношения – связи между сущностями.
* Диаграммы – группируют множество сущностей и отношений.
* В языке UML определены четыре типа отношений:
* зависимость;
* ассоциация;
* обобщение;
* реализация.

Эти отношения являются основными связующими строительными блоками в UML и применяются для создания корректных моделей.

Зависимость *(Dependency*) – это семантическое отношение между двумя сущностями, при котором изменение одной из них, независимой, может повлиять на семантику другой, зависимой.

Ассоциация (*Association*) – структурное отношение, описывающее совокупность связей; связь - это соединение между объектами.

Обобщение (*Generalization*) – отношение «специализация/обобщение», при котором объект специализированного элемента (потомок) может быть подставлен вместо объекта обобщенного элемента.

Реализация (*Realization*) – это семантическое отношение между классификаторами, при котором один классификатор определяет «контракт», а другой гарантирует его выполнение

Диаграммы UML – графическое представление набора элементов, изображенное чаще всего в виде связанного графа с вершинам (сущностями) и ребрами (отношениями).

В ходе выполнения курсового проектирования осуществлена разработка следующих моделей системы на основе стандарта UML 2.0.



## Диаграмма вариантов использования (прецедентов) (use case diagram)

Суть диаграммы вариантов использования ([Приложение Б](#_Приложение_Б_(обязательное)), рис. Б.1) состоит в следующем: проектируемая система представляется в форме так называемых вариантов использования, с которыми взаимодействуют некоторые внешние сущности или актеры. При этом актером или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. В свою очередь вариант использования служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой и собственно выполнение вариантов использования.

## Диаграмма классов (class diagram)

Диаграмма классов ([Приложение Б](#_Приложение_Б_(обязательное)), рис. Б.2-Б.3) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов может служить дальнейшим развитием концептуальной модели проектируемой системы.

## Диаграмма последовательности (sequence diagram)

С помощью диаграммы последовательности ([Приложение Б](#_Приложение_Б_(обязательное)), рис. Б.4) можно описать полный контекст взаимодействий как своеобразный временной график «жизни» всей совокупности объектов, взаимодействующих между собой для реализации варианта использования программной системы, достижения бизнес-цели или выполнения какой-либо задачи. На диаграмме последовательности также изображаются объекты, которые непосредственно участвуют во взаимодействии, при этом никакие статические связи с другими объектами не визуализируются. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени.

## Диаграмма состояний (statechart diagram)

Главное предназначение диаграммы состояний (рисунок 5.1) – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы в течение всего ее жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

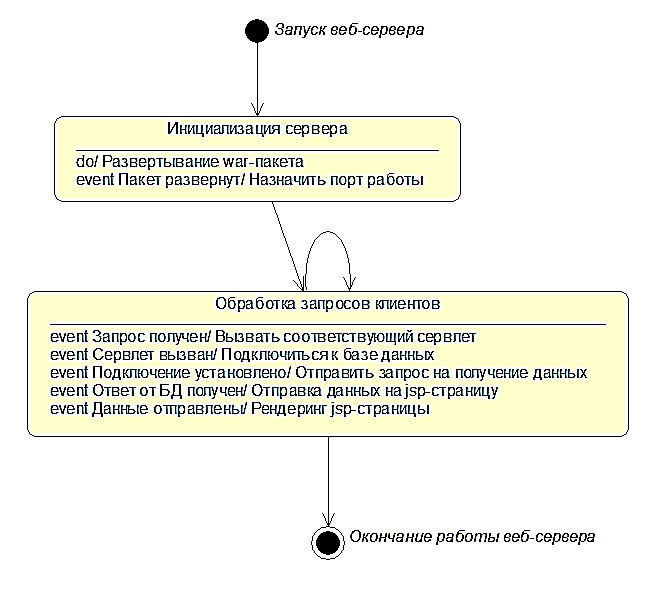


Рисунок 5.1 – Диаграмма состояний

## Диаграмма компонентов (component diagram)

Диаграмма компонентов (рисунок 5.2) описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

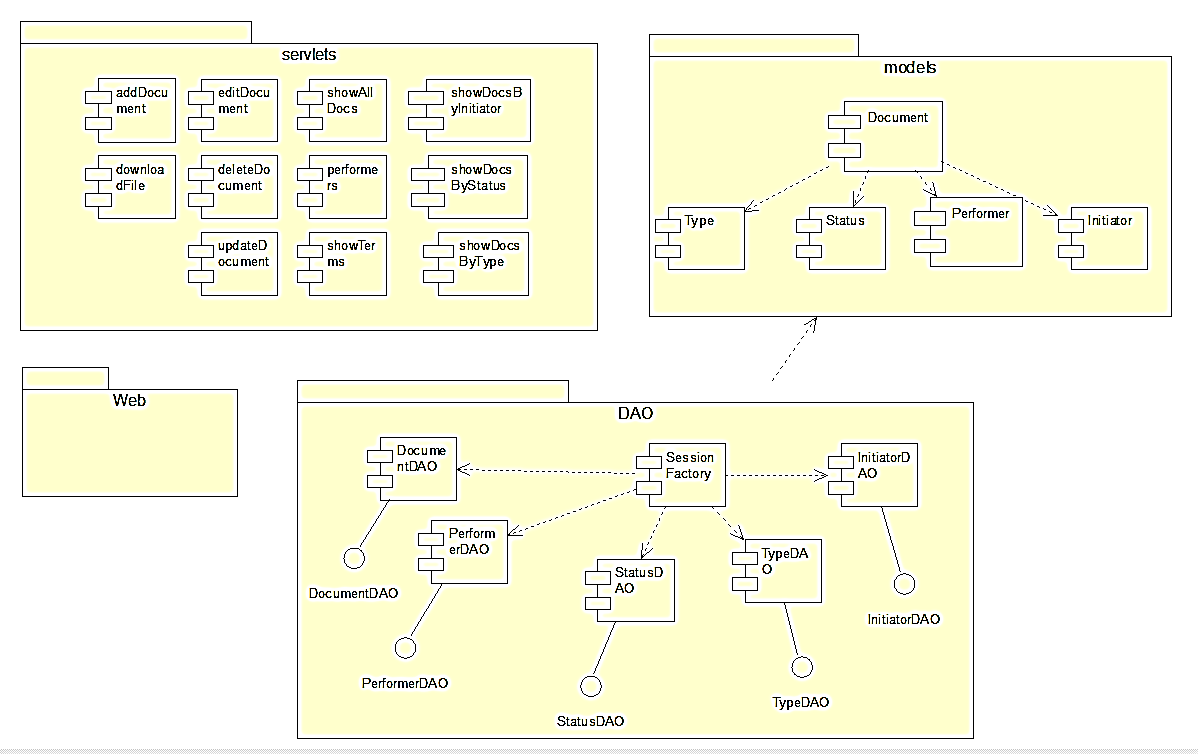


Рисунок 5.2 – Диаграмма компонентов

Диаграмма компонентов разрабатывается для следующих целей:

* визуализация общей организации структуры исходного кода программной системы;
* спецификация исполнимого варианта программной системы;
* обеспечение многократного использования отдельных фрагментов программного кода;
* представление концептуальной и физической схем баз данных.

## Диаграмма развертывания (deployment diagram)

Диаграмма развертывания (рисунок 5.3) применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений – маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

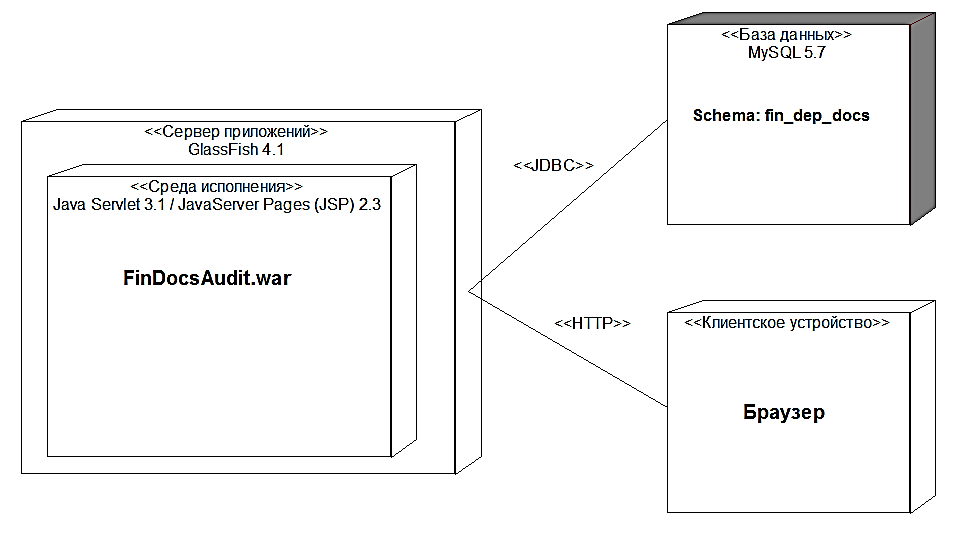


Рисунок 5.3 – Диаграмма развертывания

# Описание алгоритмов, реализующих бизнес-логику проектируемой системы

На рисунке 6.1 изображена блок-схема алгоритма разрабатываемой функции получения списка документов по идентификатору инициатора. Алгоритмы функций получения списка документов по идентификаторам типов документов, исполнителей и статусов аналогичны приведенному ниже.

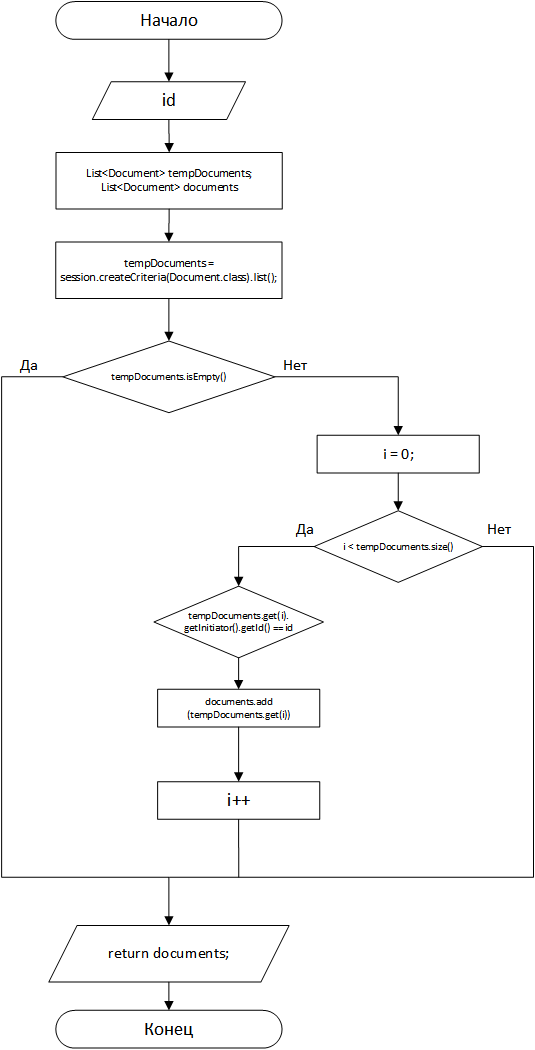


Рисунок 6.1 – Блок-схема функции получения списка документов по идентификатору

Запросы к базе данных осуществляются при помощи инструментария фреймворка Hibernate 4.3, содержащего механизмы отображения в реляционной базе данных объектов Java. Данная библиотека предназначенная для решения задач объектно-реляционного отображения (ORM). Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования при работе в объектно-ориентированных средствах в реляционной базе данных. Разработчик может использовать Hibernate как в процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой данных.

# Руководство пользователя

## Назначение программы

Приложение «Документация структурного производственного подразделения» (далее – Приложение), разработано для организации работы сотрудников отдела по регистрации, учету и контролю исполнения входящих документов.

Приложение предназначено для следующих целей:

* ведение базы данных входящих документов (добавление, редактирование, удаление информации о документах, поступивших на исполнение в планово-производственный отдел);
* загрузка на сервер электронных версий документов;
* загрузка с сервера ранее загруженных файлов документов;
* отображение статистики по документам, находящихся на исполнении у конкретного сотрудника планово-производственного отдела;
* отображение актуальной информации о количестве документов с нарушением контрольных сроков исполнения, находящихся на исполнении у конкретного сотрудника планового-производственного отдела;
* отображение актуальной информации о количестве документов с приближающимся контрольным сроком исполнения.

## Условия выполнения

Для работы с клиентской частью Приложения необходимо наличие браузера MS Edge или MS Internet Explorer, Opera, Mozilla Firefox, Google Chrome, Safari с включенной поддержкой JavaScript.

Серверная часть Приложения предназначена для работы на серверах предприятия и должно отвечать следующим минимальным техническим и программным характеристикам:

* процессор ­­– Pentium 2 266 МГц и выше;
* свободное место на жестком диске – не менее 1 Гб (250 Мб – для JDK8, около 250 Мб – для сервера приложений GlassFish 4.1, около 500 Мб – для СУБД MySQL);
* свободная оперативная память – не менее 512 Мб;
* ОС MS Windows Vista SP2+ (Windows Server 2008 R2+), Red Hat Enterprise Linux 5.5+, Oracle Linux 5.5+, Suse Linux Enterprise Server 10 SP2+, Ubuntu Linux 12.04 LTS+, Mac OS X 10.8.3+;
* программная платформа JDK версии не ниже 8.0;
* СУБД MySQL 5.5+;

## Работа с веб-приложением

Запуск веб-приложения осуществляется путем ввода пользователем в адресную строку браузера адреса, порта и корневого контекстного адреса приложения (context-root).

*Примечание.*

*В случае применения настроек по умолчанию и инсталляции сервера приложений на локальный хост веб-адресом приложения будет являться:* [*http://localhost:8080/web/*](http://localhost:8080/web/)*.*

В случае корректной настройки веб-приложения и СУБД в браузере отобразится веб-страница, представленная на рисунке 7.1:

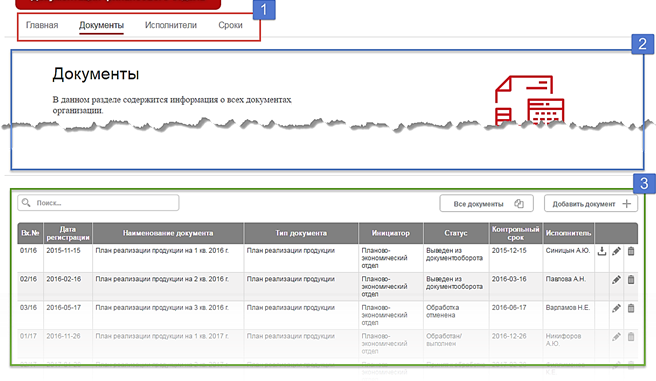


Рисунок 7.1 – Общий вид информационной веб-страницы

Все информационные веб-страницы приложения стилизованы по общим параметрам и состоят из трех областей: области навигационного меню (п.1, выделена красным цветом, содержит гиперссылки на другие веб-страницы приложения), области описания страницы (п.2, выделена синим цветом, содержит заголовок страницы и краткое описание предполагаемых действий пользователя), области отображения рабочего контента (п.3, выделена зеленым цветом, содержит веб-элементы для интерактивного взаимодействия с пользователем).

Все веб-страницы работы с формами (добавление, удаление, редактирование информации) стилизованы по общим параметрам и состоят из четырех областей (см. рисунок 7.2): (п.1, область выделена красным цветом, содержит гиперссылки на другие веб-страницы приложения), области описания страницы (п.2, выделена синим цветом, содержит заголовок страницы и краткое описание предполагаемых действий пользователя), области форм (п.3, выделена зеленым цветом, содержит веб-формы для внесения данных пользователем), дополнительной области навигации (п.4, область выделена оранжевым цветом, содержит гиперссылки на другие веб-страницы приложения или кнопки отправки данным форм).

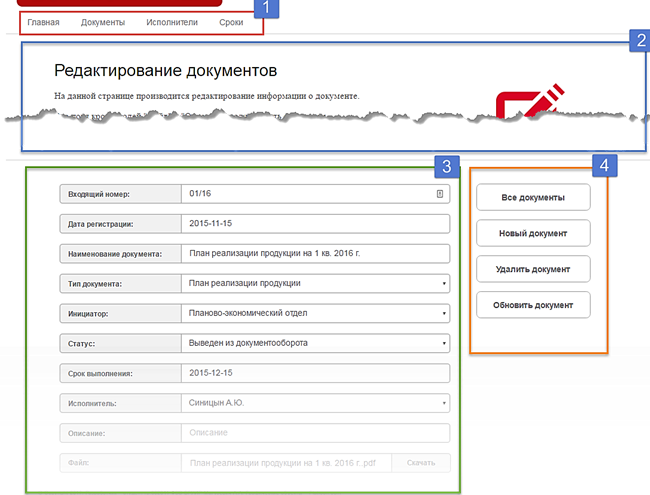


Рисунок 7.2 – Общий вид веб-страницы работы с файлами

Начальной страницей работы веб-приложения является веб-страница «Документы» (см. рисунок 7.3). Далее по тексту будут приводиться снимки только интересующих нас областей.

На данной странице отображена информация о всех зарегистрированных документах, независимо от присвоенного им статуса и контрольного срока. Изначально строки таблицы отсортированы по идентификаторам документов, хранящихся в базе данных, однако предусмотрена сортировка по выбранному критерию – для этого пользователю необходимо нажать на любой из заголовков таблицы (п.1, рисунок 7.3, область выделена синим цветом).

С целью быстрого доступа к странице редактирования конкретного документа пользователю необходимо нажать на интересующую его ячейку столбца «Наименование документа» (п.2, рисунок 7.3, область выделена красным цветом).

С целью осуществления выборки документов по конкретному критерию пользователю необходимо нажать на соответствующее поле таблицы с интересующим его значением. Для выборки доступны любое из полей столбцов «Тип документа», «Инициатор» и «Статус» (п.3, рисунок 7.3, область выделена зеленым цветом).

С целью быстрого доступа к странице конкретного исполнителя пользователю необходимо нажать на интересующую его ячейку столбца «Исполнитель» (п.4, рисунок 7.3, область выделена оранжевым цветом).

Для быстрого доступа к определенным действиям с документами пользователю необходимо нажать на соответствующую пиктограмму (п.5, рисунок 7.3, выделена желтым цветом). Доступны следующие быстрые действия (перечислены в порядке их отображения): «Загрузить с сервера электронный файл документа», «Редактировать документ» и «Удалить документ». Отсутствие пиктограммы «Загрузить с сервера электронный файл документа» означает отсутствие на сервере загруженного файла документа.

Для быстрого доступа к странице добавления документа необходимо нажать на кнопку «Добавить документ» (п.6, рисунок 7.3).

Для поиска содержимого таблиц по любому из критериев пользователю необходимо воспользоваться соответствующей поисковой строкой (п.7, рисунок 7.3).

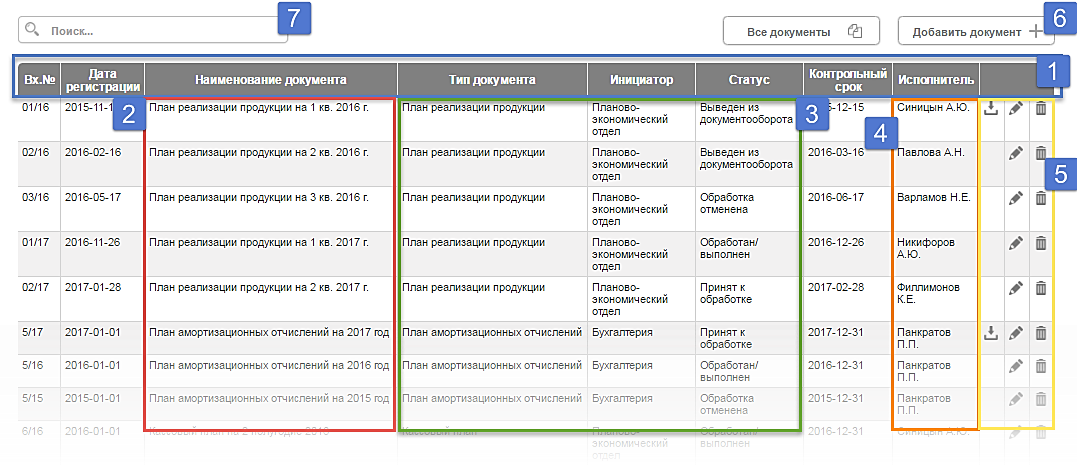


Рисунок 7.3 – Общий вид веб-страницы «Документы»

На странице «Исполнители» (см. рисунок 7.4) отображена информация о конкретном исполнителе и документах, находящихся у него на исполнении. Данная страница содержит четыре области: область информации об исполнителе (п.1, рисунок 7.4, выделена красным цветом), область информации о документах, находящихся на исполнении у конкретного исполнителя (п.2, рисунок 7.4, выделена синим цветом), область статистики по исполнителям (п.3, рисунок 7.4, выделена зеленым цветом), область поиска конкретного исполнителя (п.4, выделена оранжевым цветом).

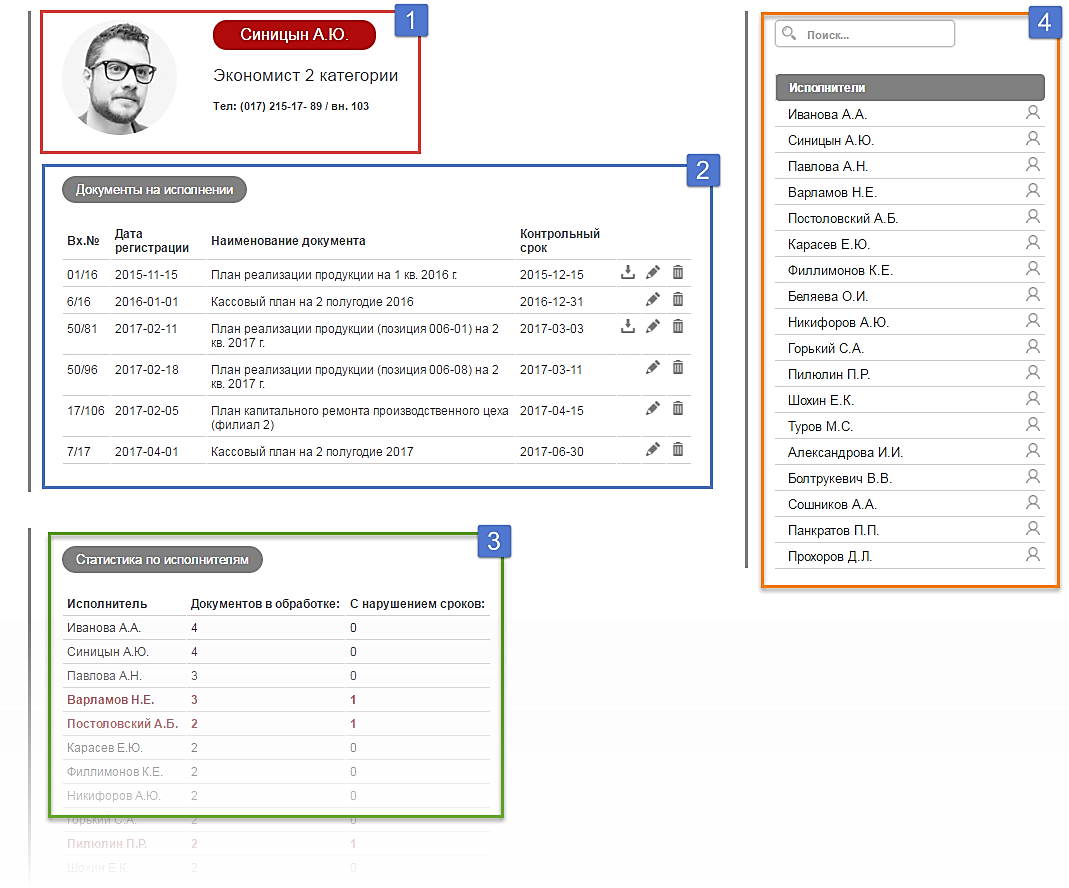


Рисунок 7.4 – Общий вид веб-страницы «Исполнители»

На странице «Сроки» (см. рисунок 7.5) отображены две информационные таблицы, информирующие пользователя о документах с превышением контрольного срока (п.1, рисунок 7.5, область таблицы выделена красным цветом) и документах с истекающим сроком исполнения в ближайшие 10 дней (п.2, рисунок 7.5, область таблицы выделена синим цветом).

Изначально строки таблиц отсортированы по полям «Дней просрочено» и «Осталось дней» соответственно. Однако предусмотрена сортировка по выбранному критерию – для этого пользователю необходимо нажать на любой из заголовков таблицы.

С целью быстрого доступа к странице конкретного исполнителя пользователю необходимо нажать на интересующую его ячейку столбца «Исполнитель».

Для быстрого доступа к определенным действиям с документами пользователю необходимо нажать на соответствующую пиктограмму. Доступны следующие быстрые действия (перечислены в порядке их отображения): «Загрузить с сервера электронный файл документа», «Редактировать документ» и «Удалить документ». Отсутствие пиктограммы «Загрузить с сервера электронный файл документа» означает отсутствие на сервере загруженного файла документа.

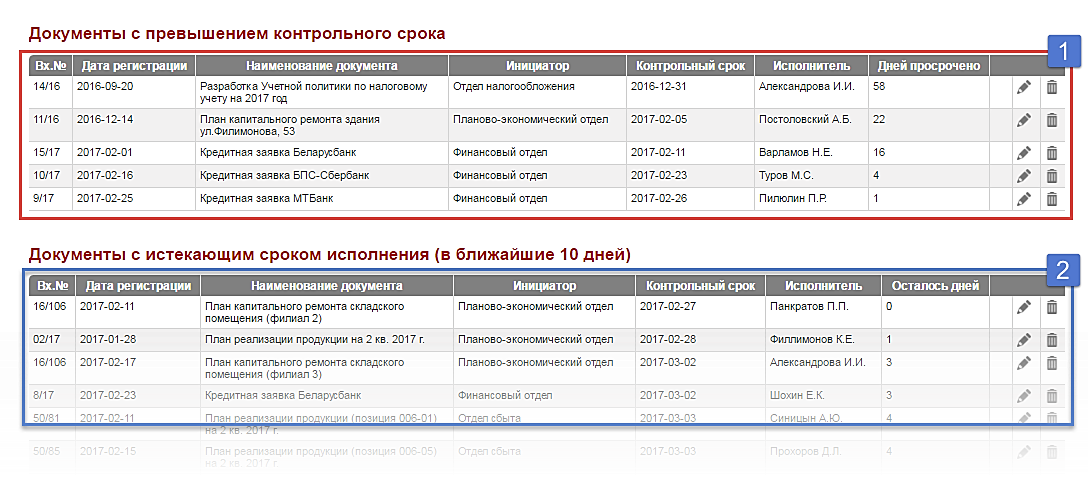


Рисунок 7.5 – Общий вид веб-страницы «Сроки»

Для добавления, изменения и удаления записей о документах в Приложении доступны соответствующие веб-страницы, которые вызываются способами, описанными выше. На рисунке 7.6 представлен общий вид страницы добавления документа.

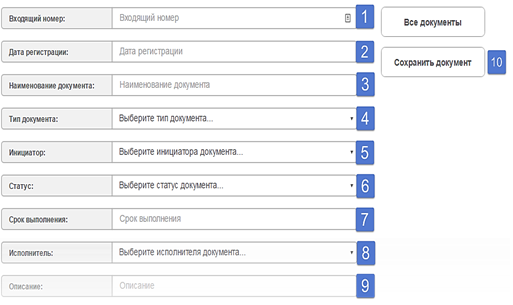


Рисунок 7.7 – Общий вид веб-страницы «Добавление документы»

Данная страница состоит из 9 текстовых полей или полей одиночного выбора, обязательными к заполнению являются поля 1-8 (рисунок 7.6). После заполнения указанных полей с целью сохранения введенной информации о документе в базу данных пользователю необходимо нажать кнопку «Сохранить документ» (п.10, рисунок 7.6), после чего произойдет перенаправление на страницу «Документы», в таблице которой уже будет отображена введенная информация. В случае некорректного ввода произойдет перенаправление на страницу с ошибкой (см. рисунок 7.7), а внесения информации в базу данных не произойдет.

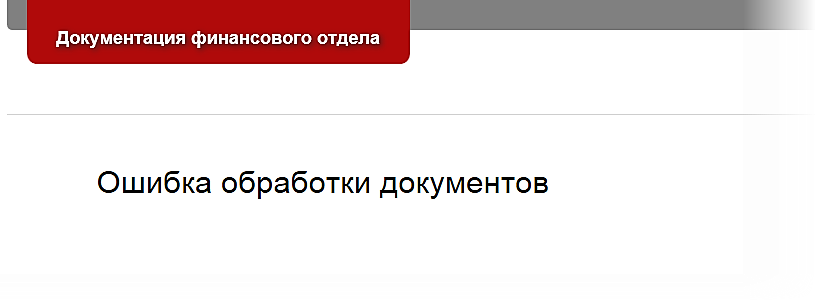


Рисунок 7.7 – Общий вид веб-страницы «Ошибка обработки документов»

При доступе к странице редактирования вновь или ранее созданного документа дополнительно станет доступным поле загрузки файла, если ранее файл загружен не был (см. рисунок 7.8).



Рисунок 7.8 – Вид области добавления файла веб-страницы «Редактирование документа»

К загрузке на сервер доступны файлы размером до 10 Мб. После нажатия на кнопку «Добавить» (п.2, рисунок 7.8) и выбора файла в области справа появится кнопка «Загрузить файл» (п.3, рисунок 7.8), по нажатию которой произойдет загрузка файла на сервер.

# Результаты тестирования системы и оценка выполнения поставленных задач

Для всех методов бизнес-логики были использованы юнит-тесты. В результате их выполнения была получено 100% прохождения тестирования. Это означает, что классы бизнес-логики предусматривают работу во всех возможных ситуациях использования кода.

В программе учтены следующие элементы:

- валидация вводимой информации;

- контроль целостности транзакций.

При тестировании функционирования системы не выявлено никаких сбоев в системе.

Проведена ревизия кода на предмет ошибок синхронизации доступа к общим ресурсам. Были выявлены и устранены ошибки синхронного доступа к некоторым коллекциям и объектам.

Нагрузочное тестирование показало, что в процессе использования приложения до ста пользователями одновременно не происходило никаких сбоев и перегрузок сервера.

При значительном росте данных в базе скорость запросов снижается, но остается на приемлемом уровне.

Таким образом, можно заключить, что поставленные требования к системе выполнены в достаточной мере.

# Заключение

Подготовка к разработке приложения «Документация структурного производственного отдела» была начата с ознакомления задачами и функциями подразделения предприятия, а также с процессами и этапами учета документации, поступающей на исполнение в данное подразделение. Полученная информация стала основой для создания веб-приложения с серверной частью с организацией взаимодействия с соответствующей базой данных.

В ходе разработки произведен обзор методов и алгоритмов решения поставленной задачи, осуществлено функциональное моделирование проектируемой системы на основе стандарта IDEF0, составлена информационная модель системы, а также разработаны модели системы на основе стандарта UML 2.0.

В ходе реализации серверной части приложения организовано взаимодействие с базой данных MySQL посредством драйвера, предоставленного производителем СУБД. В процессе разработки сервера реализованы функции отправки/получения данных из/в СУБД, их консолидации и формирования веб-страниц, отображаемых в браузере пользователя.

С целью освобождения от значительного объёма сравнительно низкоуровневого программирования при работе с реляционной базе данных изучены и применены технологии объектно-реляционного отображения (ORM), такие как Hibernate, и сопутствующий язык запросов HQL.

В ходе реализации веб-интерфейса приложения разработана и внедрена единая таблица стилей (CSS) jsp-страниц, применен инструментарий языка JSTL. При помощи языка JavaScript и сопутствующих библиотек (jQuery, sortable и др.) добавлена интерактивность для веб-содержимого, что облегчило взаимодействие пользователя с веб-приложением.

Разработанное приложение обеспечивает ведение базы данных входящих документов (добавление, редактирование, удаление информации о документах, поступивших на исполнение в отдел), загрузку на/с сервер(а) электронных версий документов, отображение статистики по документам, находящихся на исполнении у конкретного сотрудника, отображение актуальной информации о количестве документов с нарушением контрольных сроков исполнения, отображение актуальной информации о количестве документов с нарушением контрольных сроков исполнения, находящихся на исполнении у конкретного сотрудника производственного отдела.

Исходя из изложенного выше, можно сделать вывод о том, что разработанное приложение может использоваться по назначению в представленной предметной области.

# Список использованных источников

1. Буч Г. / Язык UML. Руководство пользователя. / Грэйди Буч, Джеймс Рамбо, Айвар Джекобсон: Пер. с англ. Слинкин А. А. – 2-ое изд., стер. – М.: ДМК «Пресс»; СПБ.: Питер, 2006 – 432 с.: ил.
2. Глухова Л.А. / Технологии разработки программного обеспечения. Учеб. пособие.
3. Методология функционального моделирования IDEF0 / РД IDEF 0 – 2000.
4. Фаулер М. / UML. Основы, 3-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 192 с., ил.
5. Леоненков А. / Самоучитель UML, 2-е издание. – СПб: БХВ-Петербург, 2005. – 432 с.: ил.
6. Монахов В.В. / Язык программирования и среда NetBeans – 3-е издание перераб. И доп. –СПб.: БХВ-Петербург, 2011. –704 с.
7. Шилдт Г. / Java 8. Руководство для начинающих, 6-е изд., ООО «И.Д. Вильямс», 2015, –720 с.
8. Т. М. Унучек [и др.]. / Языки программирования для разработки сетевых приложений: язык программирования JAVA: лаб. практикум для студ. спец. I-27 01 01 «Экономика и организация производства», I-26 02 03 «Маркетинг» днев. формы обуч. В 2 ч. /– Минск : БГУИР, 2008. –64 с.
9. UML. Классика CS. 2-у изд./Пер. с англ.; Под общей редакцией проф. С.Орлова - СПб.: Питер, 2007. - 736 с.: ил.
10. Блинов И.Н., Романчик В.С. Java**.** Промышленное программирование – УниверсалПресс, 2007. - 704с.
11. Басс Л., Клементс П., Кацман Р. Архитектура программного обеспечения на практике. 2-е издание. - СПб.: Питер, 2007. - 575 с.: ил.

# **Приложение А** (обязательное)

**Диаграммы модели разрабатываемого приложения в формате IDEF0**

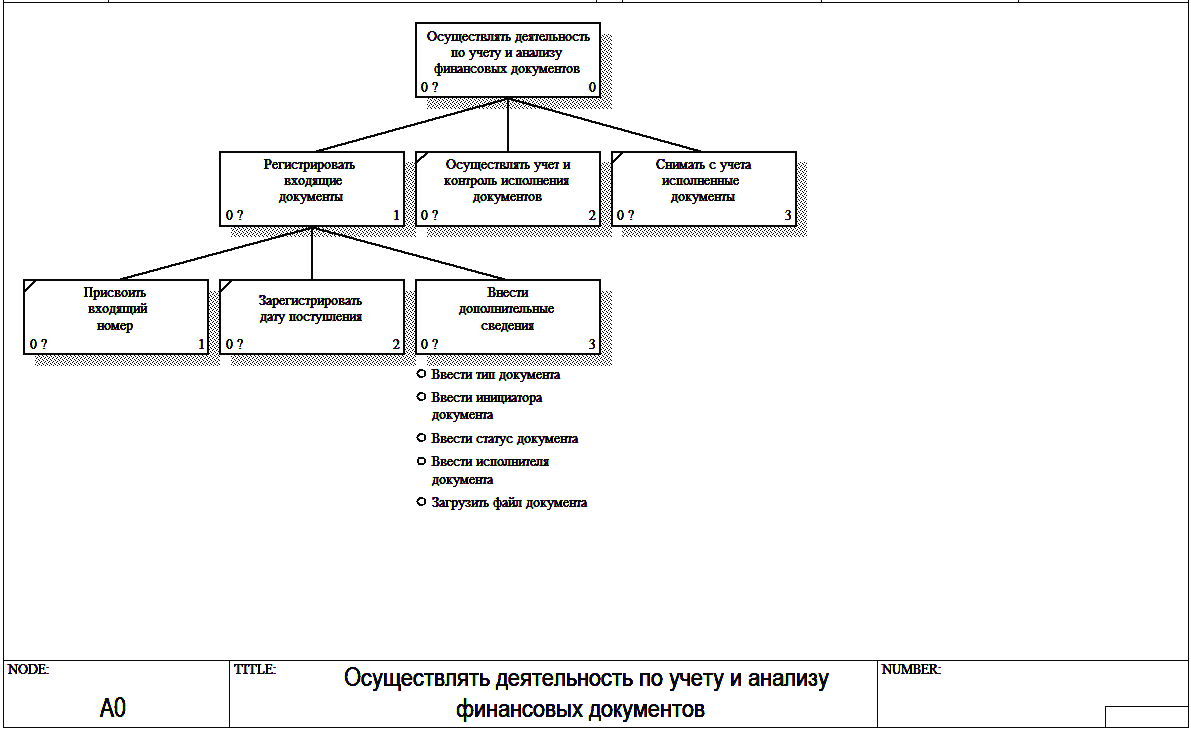


Рисунок А.1 – Контекстная диаграмма

Приложение А (продолжение)

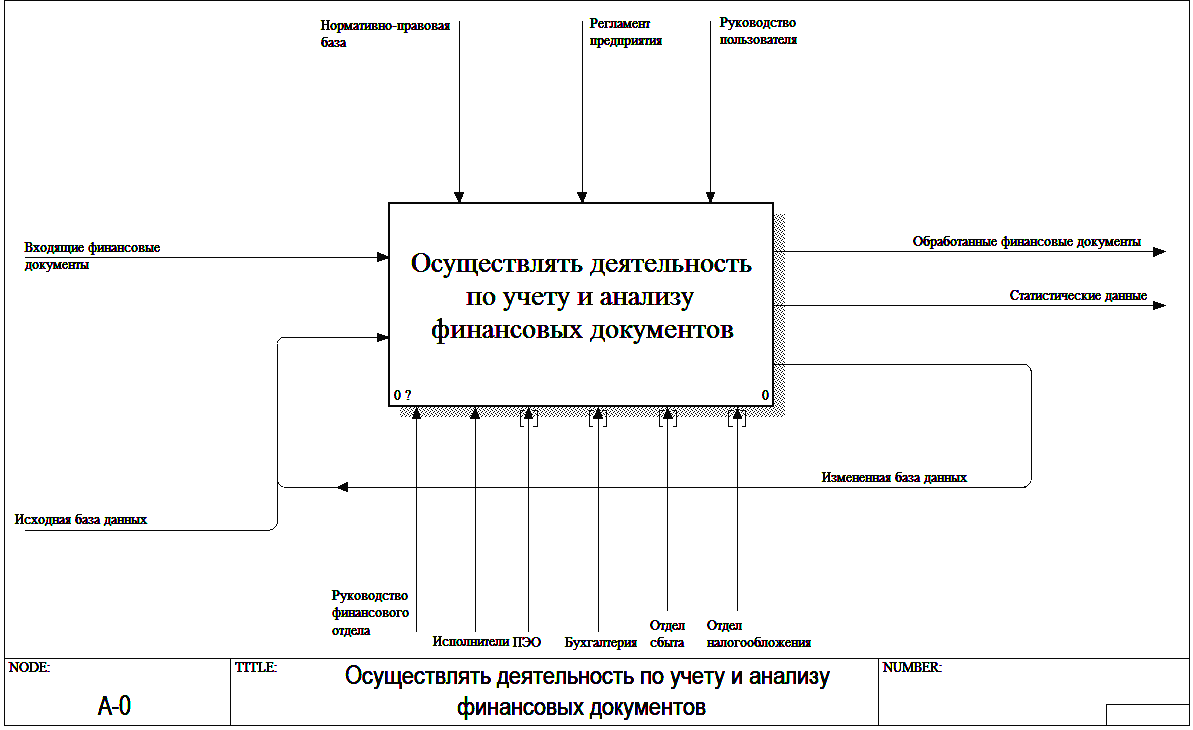


Рисунок А.2 – Диаграмма декомпозиции, уровень А-0

Приложение А (продолжение)

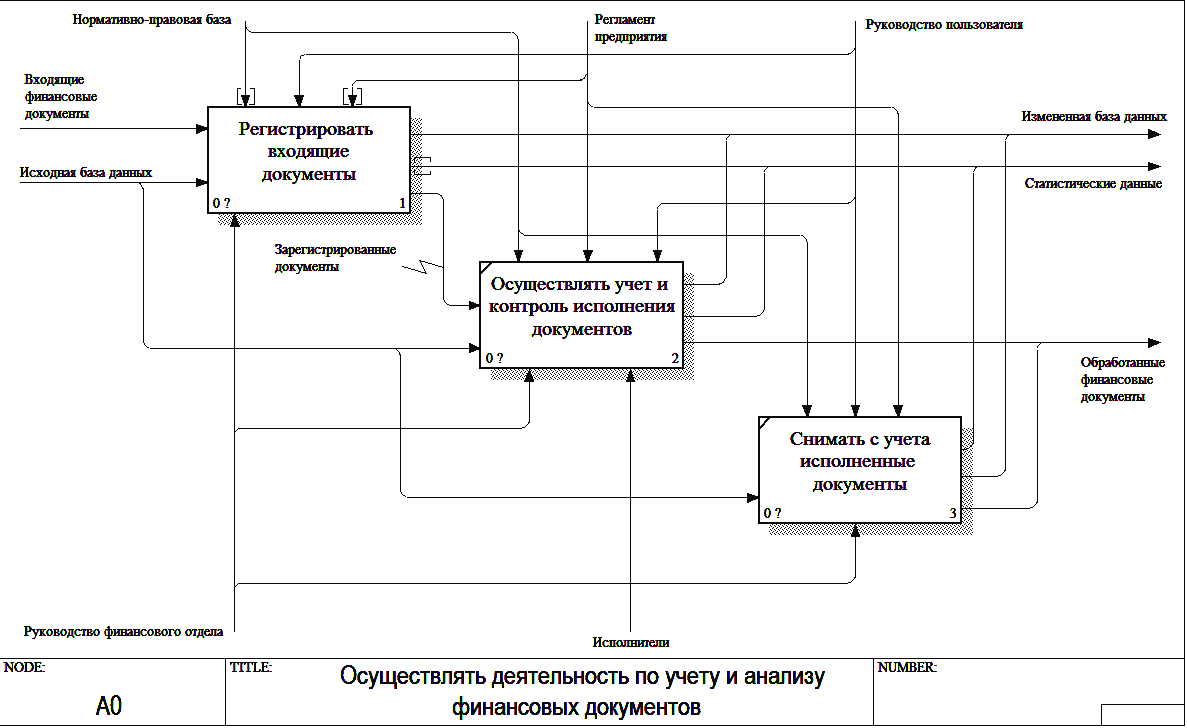


Рисунок А.3 – Диаграмма декомпозиции, уровень А0

Приложение А (продолжение)

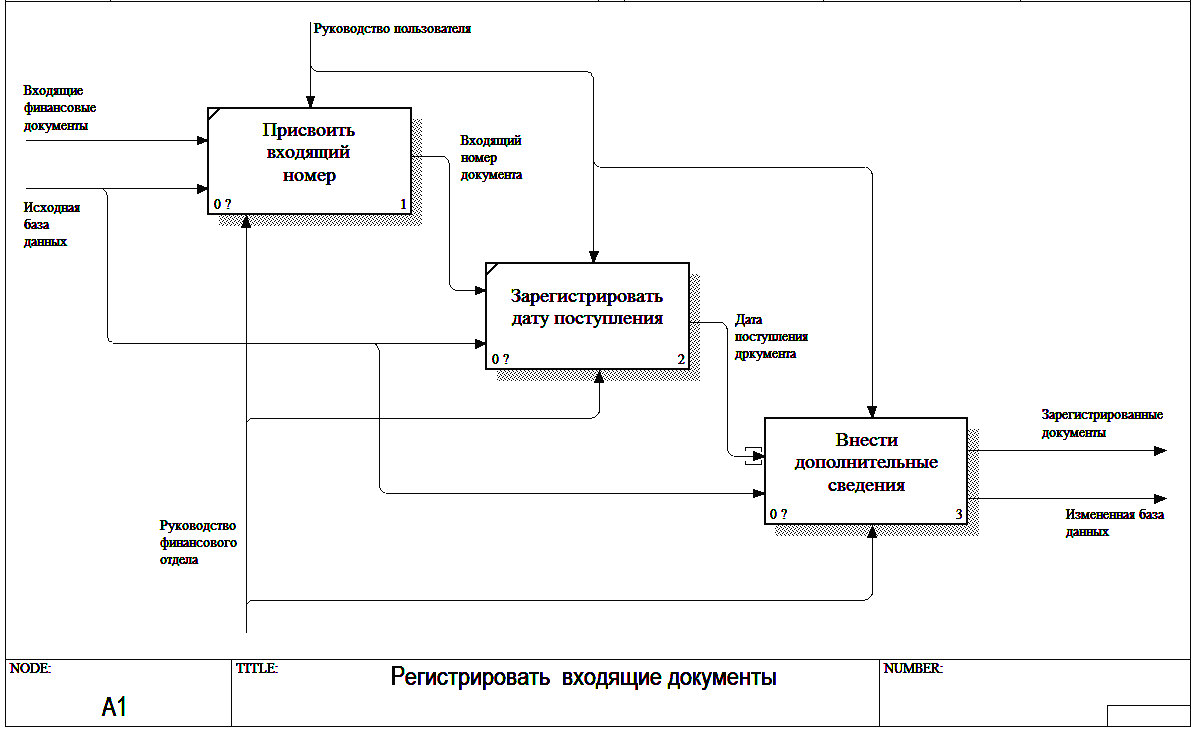


Рисунок А.4 – Диаграмма декомпозиции, уровень А1

Приложение А (продолжение)

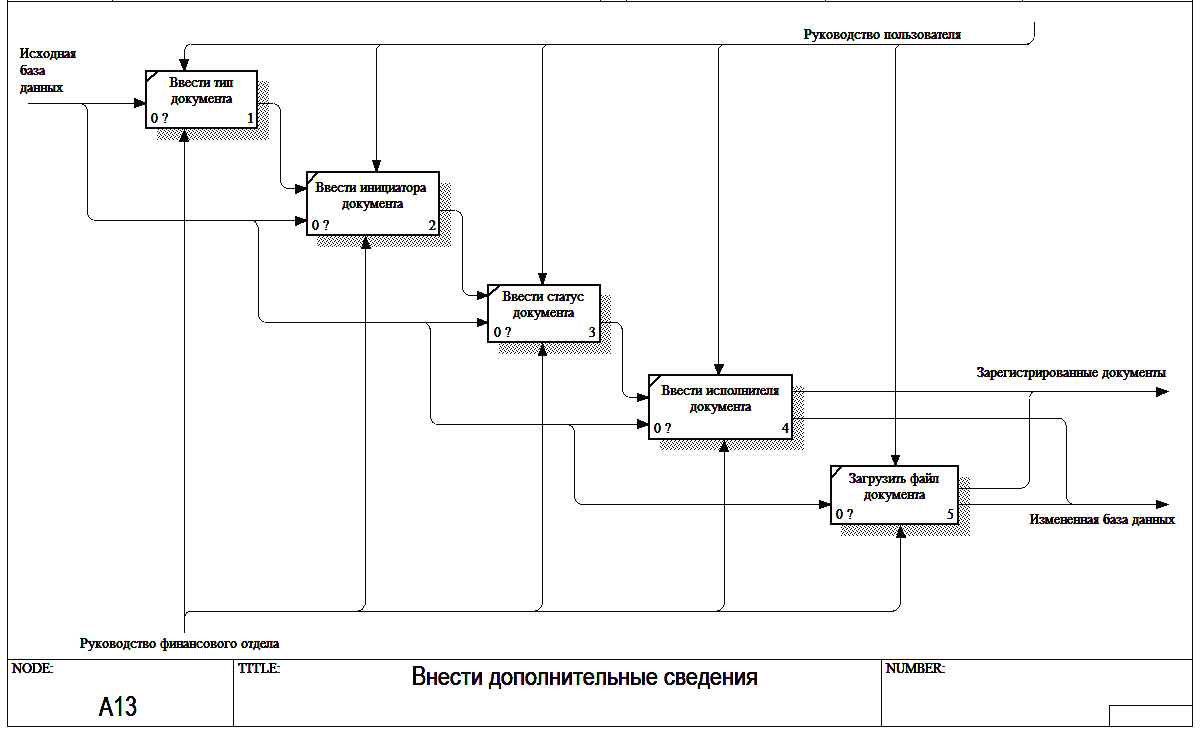


Рисунок А.5 – Диаграмма декомпозиции, уровень А13

# **Приложение Б** (обязательное)

**UML-диаграммы разрабатываемой системы**

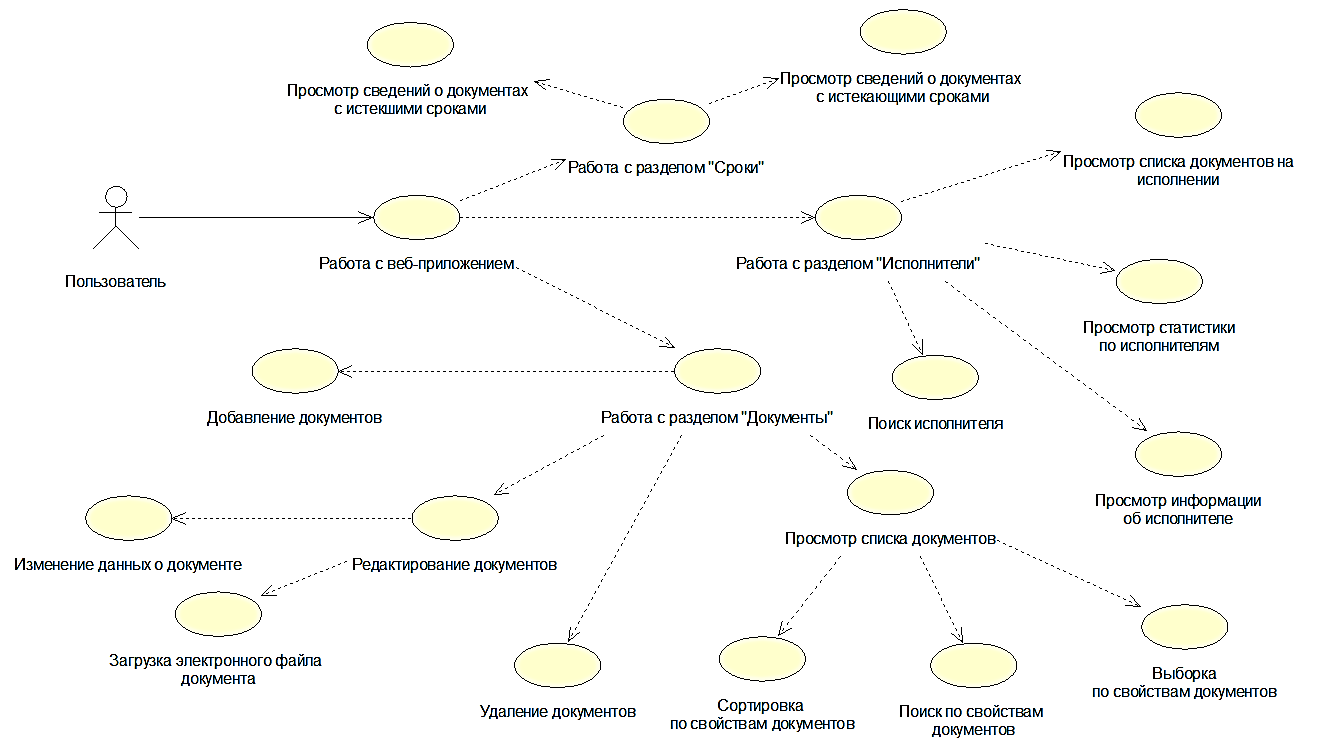


Рисунок Б.1 – Диаграмма вариантов использования

Приложение Б (продолжение)

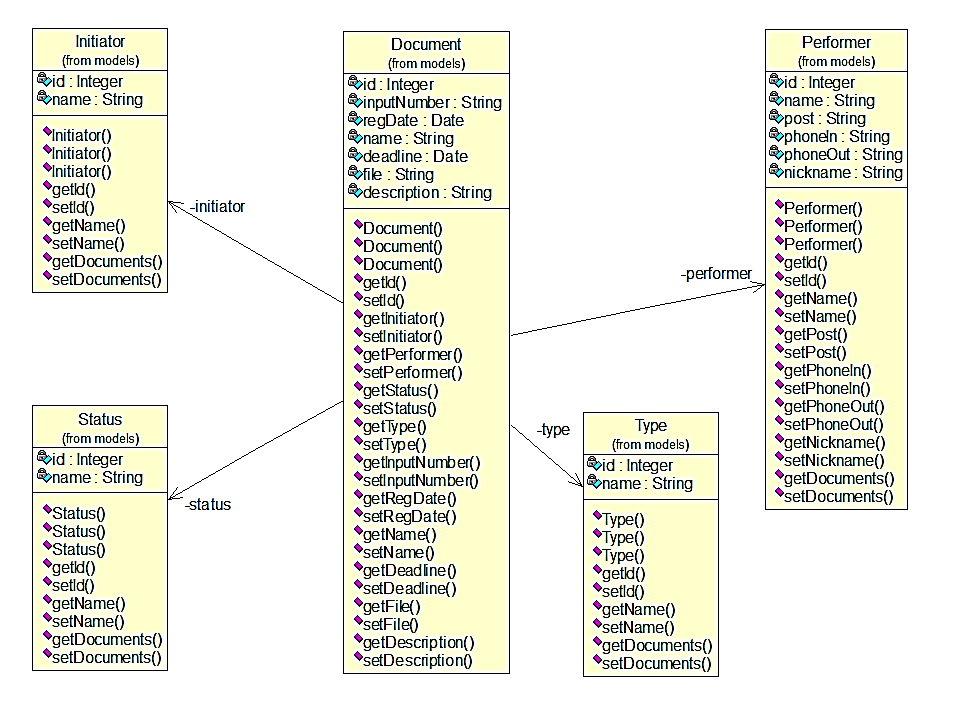


Рисунок Б.2 – Диаграмма классов (модели)

Приложение Б (продолжение)

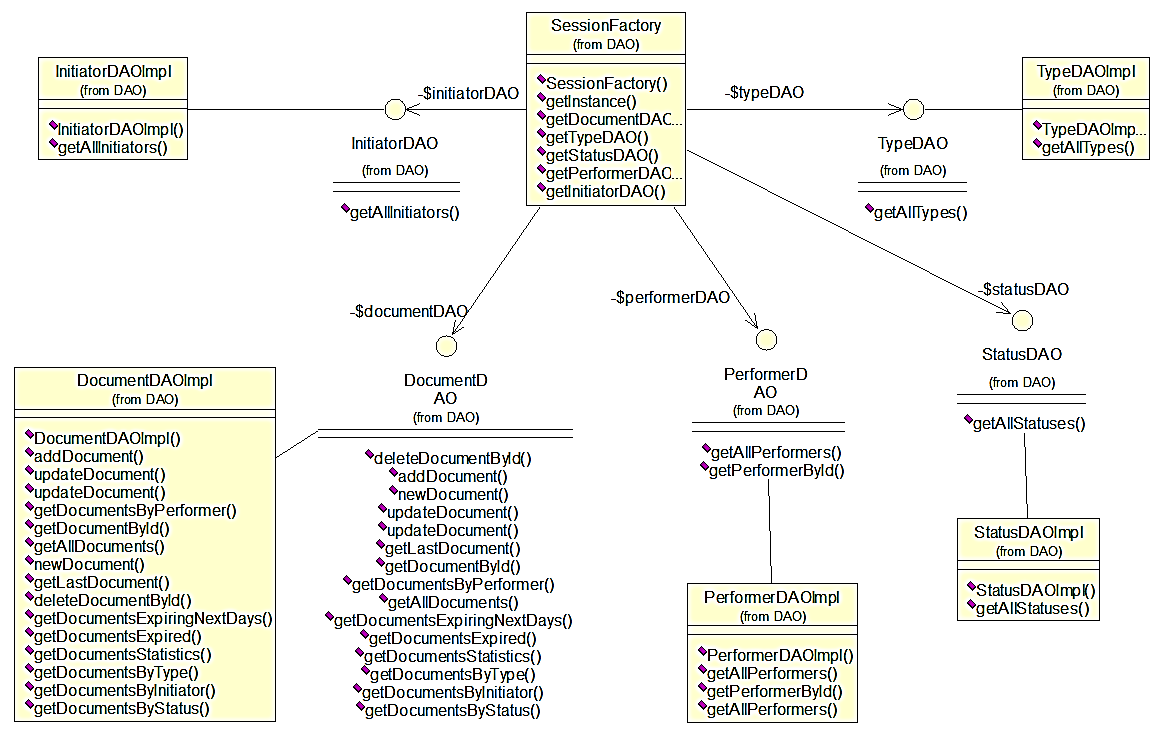


Рисунок Б.3 – Диаграмма классов (DAO)

Приложение Б (продолжение)

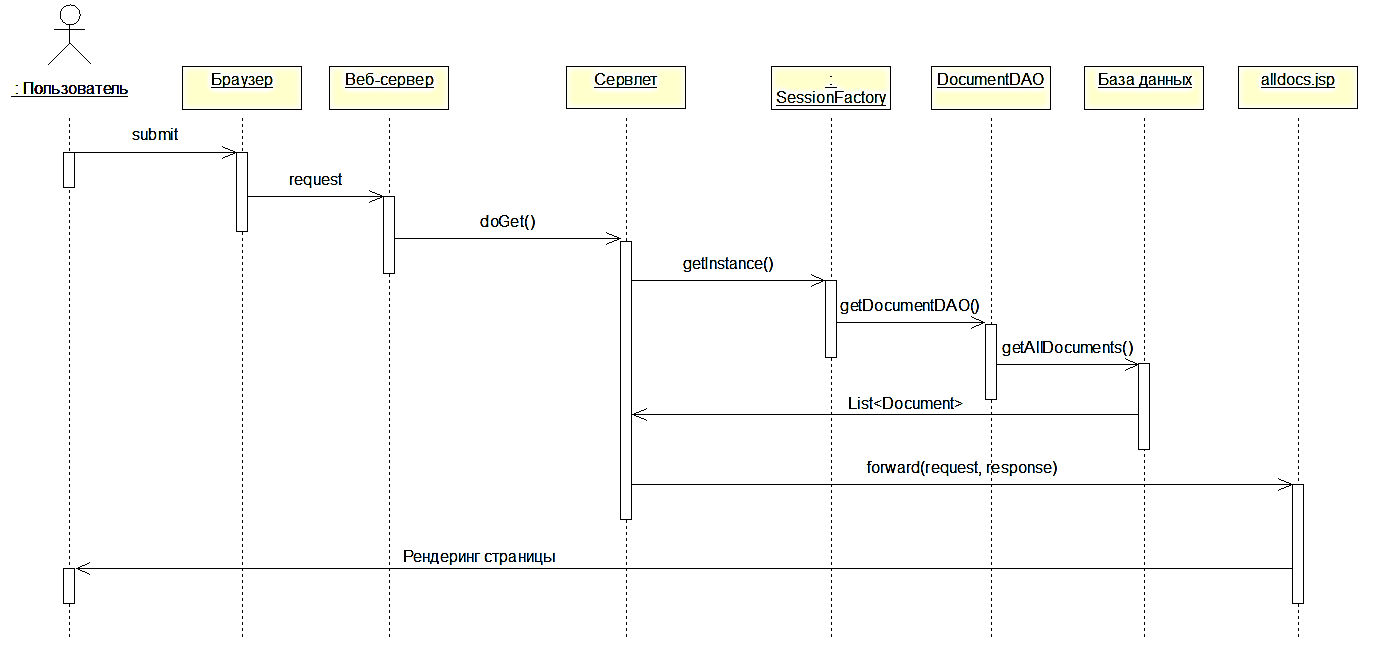


Рисунок Б.4 – Диаграмма последовательности

# **Приложение В** (обязательное)

**Листинг кода скрипта генерации базы данных**

CREATE DATABASE IF NOT EXISTS `fin\_dept\_docs`;

USE `fin\_dept\_docs`;

CREATE USER 'dbadmin'@'localhost' IDENTIFIED BY 'dbadmin';

GRANT ALL PRIVILEGES ON \* . \* TO 'dbadmin'@'localhost';

--

-- Table structure for table `document`

--

DROP TABLE IF EXISTS `document`;

CREATE TABLE `document` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`input\_number` varchar(10) NOT NULL,

`reg\_date` date NOT NULL,

`name` varchar(200) NOT NULL,

`type\_id` int(10) unsigned NOT NULL,

`performer\_id` int(10) unsigned NOT NULL,

`initiator\_id` int(10) unsigned NOT NULL,

`deadline` date NOT NULL,

`status\_id` int(10) unsigned NOT NULL,

`file` varchar(200) DEFAULT NULL,

`description` varchar(500) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `doc\_id\_UNIQUE` (`id`),

UNIQUE KEY `file\_UNIQUE` (`file`),

KEY `FK\_initiator` (`initiator\_id`),

KEY `FK\_status` (`status\_id`),

KEY `FK\_type` (`type\_id`),

KEY `FK\_performer` (`performer\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=utf8;

--

-- data for table `document`

--

LOCK TABLES `document` WRITE;

INSERT INTO `document` VALUES (1,'01/16','2015-11-15','План реализации продукции на 1 кв. 2016 г.',5,2,3,'2015-12-15',5,'План реализации продукции на 1 кв. 2016 г..pdf',''),(2,'02/16','2016-02-16','План реализации \n\nпродукции на 2 кв. 2016 г.',5,3,3,'2016-03-16',5,NULL,NULL),(3,'03/16','2016-05-17','План реализации продукции на 3 кв. 2016 г.',5,4,3,'2016-06-17',3,NULL,NULL),(5,'01/17','2016-11-26','План реализации продукции на 1 кв. 2017 г.',5,9,3,'2016-12-26',1,NULL,NULL),(6,'02/17','2017-01-28','План реализации продукции на 2 кв. 2017 \n\nг.',5,7,3,'2017-02-28',2,NULL,NULL),(22,'5/17','2017-01-01','План амортизационных отчислений на 2017 год',6,17,2,'2017-12-31',2,'План амортизационных отчислений на 2017 год.pdf',''),(23,'5/16','2016-01-01','План амортизационных отчислений на 2016 год',6,17,2,'2016-12-31',1,NULL,''),(24,'5/15','2015-01-01','План амортизационных отчислений на 2015 год',6,17,2,'2015-12-31',3,NULL,''),(25,'6/16','2016-01-01','Кассовый план на 2 полугодие 2016',4,2,3,'2016-12-31',1,NULL,''),(26,'6/17','2017-01-01','Кассовый план на 1 полугодие 2017',4,10,3,'2017-03-31',2,'Кассовый план на 1 полугодие 2017.pdf',''),(27,'7/17','2017-04-01','Кассовый план на 2 полугодие 2017',4,2,3,'2017-06-30',2,NULL,''),(28,'8/17','2017-02-23','Кредитная заявка Беларусбанк',11,12,1,'2017-03-02',2,NULL,NULL),(29,'9/17','2017-02-25','Кредитная заявка МТБанк',11,11,1,'2017-02-26',2,'Кредитная заявка МТБанк.pdf',''),(30,'10/17','2017-02-16','Кредитная заявка БПС-Сбербанк',11,13,1,'2017-02-23',2,NULL,''),(31,'11/16','2016-12-14','План капитального ремонта здания ул.Филимонова, 53',8,5,3,'2017-02-05',2,NULL,''),(32,'6/16','2016-11-01','План

Приложение В (продолжение)

модернизации VoIP-телефонии офиса',8,10,3,'2017-03-16',2,'План модернизации VoIP-телефонии офиса.pdf',''),(33,'7/16','2016-09-13','Строительство торгового павильона',7,11,3,'2017-06-24',2,NULL,''),(34,'12/17','2017-02-22','Выплаты из прибыли 2016 года',9,7,2,'2017-04-07',2,NULL,''),(36,'7/16','2016-11-01','Разработка Положения о закупках',2,6,1,'2017-04-01',2,NULL,''),(37,'13/16','2016-09-06','Разработка Положения о премировании',2,3,5,'2017-04-13',2,NULL,''),(38,'14/16','2016-09-20','Разработка Учетной политики по налоговому учету на 2017 год',1,14,6,'2016-12-31',2,NULL,''),(39,'7/15','2015-08-11','Разработка Учетной политики по налоговому учету на 2016 год',1,6,6,'2016-01-01',1,NULL,''),(40,'15/17','2017-02-01','Кредитная заявка Беларусбанк',11,4,1,'2017-02-11',2,NULL,''),(41,'15/46','2017-02-20','План распределения балансовой прибыли на 2 кв. 2017 года',9,1,2,'2017-03-15',2,NULL,''),(42,'15/47','2017-02-01','Кредитная заявка Беларусбанк (в евро)',11,4,1,'2017-03-11',2,NULL,NULL),(43,'15/48','2017-02-26','Кредитная заявка Беларусбанк (в долларах США)',11,9,1,'2017-03-17',2,NULL,NULL),(44,'15/79','2017-02-24','Кредитная заявка МТБанк (в евро)',11,4,1,'2017-03-22',2,NULL,NULL),(45,'15/88','2017-02-25','Кредитная заявка БПС-Сбербанк (в евро)',11,9,1,'2017-03-22',2,NULL,NULL),(46,'16/106','2017-02-01','План капитального ремонта складского помещения (филиал 1)',8,1,3,'2017-03-15',2,'План капитального ремонта складского помещения (филиал 1).pdf',NULL),(47,'16/106','2017-02-11','План капитального ремонта складского помещения (филиал 2)',8,17,3,'2017-02-27',2,NULL,NULL),(48,'16/106','2017-02-17','План капитального ремонта складского помещения (филиал 3)',8,14,3,'2017-03-02',2,NULL,NULL),(49,'16/106','2017-02-05','План капитального ремонта складского помещения (филиал 4)',8,16,3,'2017-03-10',2,NULL,NULL),(50,'50/81','2017-02-11','План реализации продукции (позиция 006-01) на 2 кв. 2017 г.',5,2,4,'2017-03-03',2,'План реализации продукции (позиция 006-01) на 2 кв. 2017 г..pdf',NULL),(51,'50/82','2017-02-12','План реализации продукции (позиция 006-02) на 2 кв. 2017 г.',5,3,4,'2017-03-06',2,NULL,NULL),(52,'50/83','2017-02-13','План реализации продукции (позиция 006-03) на 2 кв. 2017 г.',5,5,4,'2017-03-14',2,NULL,NULL),(53,'50/84','2017-02-14','План реализации продукции (позиция 006-04) на 2 кв. 2017 г.',5,6,4,'2017-03-06',2,NULL,NULL),(54,'50/85','2017-02-15','План реализации продукции (позиция 006-05) на 2 кв. 2017 г.',5,18,4,'2017-03-03',2,NULL,NULL),(55,'50/93','2017-02-16','План реализации продукции (позиция 006-06) на 2 кв. 2017 г.',5,16,4,'2017-03-08',2,NULL,NULL),(56,'50/94','2017-02-17','План реализации продукции (позиция 006-07) на 2 кв. 2017 г.',5,12,4,'2017-03-04',2,NULL,NULL),(57,'50/96','2017-02-18','План реализации продукции (позиция 006-08) на 2 кв. 2017 г.',5,2,4,'2017-03-11',2,NULL,NULL),(58,'50/99','2017-02-19','План реализации продукции (позиция 006-09) на 2 кв. 2017 г.',5,3,4,'2017-03-22',2,NULL,NULL),(59,'17/105','2017-02-03','План капитального ремонта заготовительного цеха (филиал 7)',8,1,3,'2017-04-17',2,NULL,''),(60,'17/106','2017-02-05','План капитального ремонта производственного цеха (филиал 2)',8,2,3,'2017-04-15',2,NULL,NULL),(61,'17/13','2017-02-18','План капитального ремонта производственного цеха (филиал 3)',8,13,3,'2017-04-18',2,NULL,NULL),(62,'12/98','2017-02-10','План капитального ремонта производственного цеха (филиал 4)',8,1,3,'2017-04-11',2,NULL,NULL),(63,'15/87','2017-02-10','План капитального ремонта производственного цеха (филиал 5)',8,18,3,'2017-04-10',2,NULL,NULL);

UNLOCK TABLES;

--

-- Table structure for table `initiator`

--

DROP TABLE IF EXISTS `initiator`;

CREATE TABLE `initiator` (

`id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `id\_UNIQUE` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8;

--

-- data for table `initiator`

--

LOCK TABLES `initiator` WRITE;

INSERT INTO `initiator` VALUES (1,'Финансовый отдел'),(2,'Бухгалтерия'),(3,'Планово-экономический отдел'),(4,'Отдел сбыта'),(5,'Отдел труда и зарплаты'),(6,'Отдел налогообложения');

Приложение В (продолжение)

UNLOCK TABLES;

--

-- Table structure for table `performer`

--

DROP TABLE IF EXISTS `performer`;

CREATE TABLE `performer` (

`id` int(11) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`post` varchar(100) DEFAULT NULL,

`phone\_in` varchar(10) DEFAULT NULL,

`phone\_out` varchar(20) DEFAULT NULL,

`nickname` varchar(20) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `id\_UNIQUE` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=19 DEFAULT CHARSET=utf8;

--

-- data for table `performer`

--

LOCK TABLES `performer` WRITE;

INSERT INTO `performer` VALUES (1,'Иванова А.А.','Заместитель начальника отдела','102','(017) 215-17-89',NULL),(2,'Синицын А.Ю.','Экономист 2 категории','103','(017) 215-17-

89',NULL),(3,'Павлова А.Н.','Экономист 1 категории','104','(017) 215-17-96',NULL),(4,'Варламов Н.Е.','Специалист','105','(017) 215-17-98',NULL),(5,'Постоловский

А.Б.','Экономист 2 категории','106','(017) 215-18-15',NULL),(6,'Карасев Е.Ю.','Начальник отдела','101','(017) 215-18-20',NULL),(7,'Филлимонов К.Е.','Финансовый

аналитик','107','(017) 215-18-25',NULL),(8,'Беляева О.И.','Ведущий специалист','108','(017) 215-18-27',NULL),(9,'Никифоров А.Ю.','Экономист 2 категории','109','(017) 215-18-

26',NULL),(10,'Горький С.А.','Экономист 1 категории','110','(017) 215-18-27',NULL),(11,'Пилюлин П.Р.','Финансовый аналитик','111','(017) 215-18-88',NULL),(12,'Шохин

Е.К.','Специалист','112','(017) 215-18-36',NULL),(13,'Туров М.С.','Экономист 1 категории','113','(017) 215-18-64',NULL),(14,'Александрова И.И.','Финансовый

аналитик','114','(017) 215-18-17',NULL),(15,'Болтрукевич В.В.','Ведущий специалист','115','(017) 215-18-74',NULL),(16,'Сошников А.А.','Экономист 2 категории','116','(017)

215-18-66',NULL),(17,'Панкратов П.П.','Ведущий специалист','117','(017) 215-18-78',NULL),(18,'Прохоров Д.Л.','Экономист 1 категории','118','(017) 215-18-33',NULL);

UNLOCK TABLES;

--

-- Table structure for table `status`

--

DROP TABLE IF EXISTS `status`;

CREATE TABLE `status` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `status\_id\_UNIQUE` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=6 DEFAULT CHARSET=utf8;

--

-- data for table `status`

--

LOCK TABLES `status` WRITE;

INSERT INTO `status` VALUES (1,'Обработан/выполнен'),(2,'Принят к обработке'),(3,'Обработка отменена'),(4,'Не требует обработки'),(5,'Выведен из документооборота');

Приложение В (продолжение)

UNLOCK TABLES;

--

-- Table structure for table `type`

--

DROP TABLE IF EXISTS `type`;

CREATE TABLE `type` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`name` varchar(80) NOT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `type\_id\_UNIQUE` (`id`),

UNIQUE KEY `type\_name\_UNIQUE` (`name`)

) ENGINE=InnoDB AUTO\_INCREMENT=12 DEFAULT CHARSET=utf8;

--

-- Dumping data for table `type`

--

LOCK TABLES `type` WRITE;

INSERT INTO `type` VALUES (3,'График'),(4,'Кассовый план'),(11,'Кредитная заявка'),(6,'План амортизационных отчислений'),(10,'План по налогу \n\nс оборота'),(9,'План

распределения балансовой прибыли'),(5,'План реализации продукции'),(8,'План финансирования капитального ремонта основных фондов'),(7,'План финансирования капитальных

вложений'),(2,'Положение'),(1,'Проект документа');

UNLOCK TABLES;

-- FK

ALTER TABLE `fin\_dept\_docs`.`document`

ADD CONSTRAINT `FK\_initiator`

FOREIGN KEY (`initiator\_id`)

REFERENCES `fin\_dept\_docs`.`initiator` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

ADD CONSTRAINT `FK\_perfomer`

FOREIGN KEY (`performer\_id`)

REFERENCES `fin\_dept\_docs`.`performer` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

ADD CONSTRAINT `FK\_status`

FOREIGN KEY (`status\_id`)

REFERENCES `fin\_dept\_docs`.`status` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

ADD CONSTRAINT `FK\_type`

FOREIGN KEY (`type\_id`)

REFERENCES `fin\_dept\_docs`.`type` (`id`)

ON DELETE NO ACTION

ON UPDATE NO ACTION;

# **Приложение Г** (обязательное)

**Листинг кода разрабатываемого приложения**

***DocumentDAO.java***

package DAO;

import java.sql.SQLException;

import java.util.Collection;

import models.Document;

public interface DocumentDAO {

public void addDocument(Document doc) throws SQLException ;

public void newDocument(String input\_number, String reg\_date, String name,

String type\_id, String initiator\_id, String status\_id, String deadline,

String performer\_id, String description) throws SQLException ;

public void updateDocument(int id, Document doc) throws SQLException;

public void updateDocument(int id, String input\_number, String reg\_date, String name,

String type\_id, String initiator\_id, String status\_id, String deadline, String performer\_id,

String file, String description);

public Document getLastDocument() throws SQLException;

public Document getDocumentById(int id) throws SQLException;

public void deleteDocumentById(int id) throws SQLException;

public Collection getDocumentsByPerformer(int performer\_id) throws SQLException;

public Collection getAllDocuments() throws SQLException;

public Collection getDocumentsExpiringNextDays(int days) throws SQLException;

public Collection getDocumentsExpired() throws SQLException;

public Collection getDocumentsStatistics() throws SQLException;

public Collection getDocumentsByType(int id) throws SQLException;

public Collection getDocumentsByInitiator(int id) throws SQLException;

public Collection getDocumentsByStatus(int id) throws SQLException;

}

***DocumentDAOImpl.java***

package DAO;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.text.DateFormat;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Date;

import java.util.List;

import java.util.Locale;

import java.util.logging.Level;

import java.util.logging.Logger;

import models.Document;

import models.Initiator;

import models.Type;

import models.Performer;

import models.Status;

import org.hibernate.Criteria;

import org.hibernate.HibernateException;

Приложение Г (продолжение)

import org.hibernate.Query;

import org.hibernate.SQLQuery;

import org.hibernate.Session;

import util.HibernateUtil;

public class DocumentDAOImpl implements DocumentDAO {

@Override

public void addDocument(Document doc) throws SQLException {

Session session = null;

try {

session = HibernateUtil.getSessionFactory().getCurrentSession();

session.beginTransaction();

session.save(doc);

session.getTransaction().commit();

} catch (HibernateException e) {

DriverManager.println("Ошибка при вставке");

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

}

@Override

public void updateDocument(int id, Document doc) throws SQLException {

Session session = null;

try {

session = HibernateUtil.getSessionFactory().getCurrentSession();

session.beginTransaction();

String hql = "update fin\_dept\_docs.document set"

+ " document.input\_number = :input\_number, "

+ " document.reg\_date = :reg\_date, "

+ " document.name = :name, "

+ " document.type\_id = :type\_id, "

+ " document.initiator\_id = :initiator\_id, "

+ " document.status\_id = :status\_id, "

+ " document.deadline = :deadline, "

+ " document.performer\_id = :performer\_id, "

+ " document.file = :file, "

+ " document.description = :description "

+ "WHERE id = :id";

Query query = (Query) session.createSQLQuery(hql);

query.setParameter("input\_number", doc.getInputNumber());

query.setParameter("reg\_date", doc.getRegDate());

query.setParameter("name", doc.getName());

query.setParameter("type\_id", doc.getType().getId());

query.setParameter("initiator\_id", doc.getInitiator().getId());

query.setParameter("status\_id", doc.getStatus().getId());

query.setParameter("deadline", doc.getDeadline());

query.setParameter("performer\_id", doc.getPerformer().getId());

query.setParameter("file", doc.getFile());

query.setParameter("description", doc.getDescription());

query.setParameter("id", id);

Приложение Г (продолжение)

query.executeUpdate();

session.getTransaction().commit();

} catch (HibernateException e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

}

@Override

public void updateDocument(int id, String input\_number, String reg\_date, String name,

String type\_id, String initiator\_id, String status\_id, String deadline, String performer\_id,

String file, String description) {

Session session = null;

try {

session = HibernateUtil.getSessionFactory().openSession();

session.beginTransaction();

String hql = "update document set"

+ " input\_number = :input\_number, "

+ " reg\_date = :reg\_date, "

+ " name = :name, "

+ " type\_id = :type\_id, "

+ " initiator\_id = :initiator\_id, "

+ " status\_id = :status\_id, "

+ " deadline = :deadline, "

+ " performer\_id = :performer\_id, ";

if (file != "") {

hql += " file = :file, ";

}

hql += " description = :description "

+ "WHERE id = :id";

Query query = (Query) session.createSQLQuery(hql);

query.setParameter("input\_number", input\_number);

query.setParameter("reg\_date", reg\_date);

query.setParameter("name", name);

query.setParameter("type\_id", type\_id);

query.setParameter("initiator\_id", initiator\_id);

query.setParameter("status\_id", status\_id);

query.setParameter("deadline", deadline);

query.setParameter("performer\_id", performer\_id);

if (file != "") {

query.setParameter("file", file);

}

query.setParameter("description", description);

query.setParameter("id", id);

query.executeUpdate();

session.getTransaction().commit();

} catch (HibernateException e) {

throw e;

Приложение Г (продолжение)

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

}

@Override

public Collection getDocumentsByPerformer(int performer\_id) throws SQLException {

Session session = null;

List documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

String sql = "SELECT \* FROM document WHERE performer\_id = " + performer\_id + " ORDER BY deadline ASC";

SQLQuery query = session.createSQLQuery(sql);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);

documents = query.list();

} catch (Exception e) {

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

@Override

public Document getDocumentById(int id) throws SQLException {

Session session = null;

Document doc = new Document();

try {

session = HibernateUtil.getSessionFactory().openSession();

doc = (Document) session.load(Document.class, id);

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return doc;

}

@Override

public Collection getAllDocuments() throws SQLException {

Session session = null;

List<Document> documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

documents = session.createCriteria(Document.class).list();

} catch (Exception e) {

throw e;

Приложение Г (продолжение)

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

@Override

public void newDocument(String input\_number, String reg\_date, String name,

String type\_id, String initiator\_id, String status\_id, String deadline,

String performer\_id, String description) throws SQLException {

Session session = null;

try {

session = HibernateUtil.getSessionFactory().getCurrentSession();

session.beginTransaction();

Document doc = new Document();

DateFormat format = new SimpleDateFormat("yyyy-MM-dd", Locale.ENGLISH);

doc.setInputNumber(input\_number);

Date regDate = format.parse(reg\_date);

doc.setRegDate(regDate);

doc.setName(name);

Type type = new Type();

type.setId(Integer.parseInt(type\_id));

doc.setType(type);

Initiator initiator = new Initiator();

initiator.setId(Integer.parseInt(initiator\_id));

doc.setInitiator(initiator);

Status status = new Status();

status.setId(Integer.parseInt(status\_id));

doc.setStatus(status);

Date deadlineDate = format.parse(deadline);

doc.setDeadline(deadlineDate);

Performer performer = new Performer();

performer.setId(Integer.parseInt(performer\_id));

doc.setPerformer(performer);

doc.setDescription(description);

doc.setFile(null);

session.save(doc);

session.getTransaction().commit();

} catch (HibernateException e) {

throw e;

} catch (ParseException ex) {

Logger.getLogger(DocumentDAOImpl.class.getName()).log(Level.SEVERE, null, ex);

Приложение Г (продолжение)

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

}

@Override

public Document getLastDocument() throws SQLException {

Session session = null;

Document lastDoc = new Document();

List<Document> documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

documents = session.createCriteria(Document.class).list();

lastDoc = documents.get(documents.size() - 1);

} catch (HibernateException e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return lastDoc;

}

@Override

public void deleteDocumentById(int id) throws SQLException {

Session session = null;

try {

session = HibernateUtil.getSessionFactory().openSession();

session.beginTransaction();

session.delete(getDocumentById(id));

session.getTransaction().commit();

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

}

@Override

public Collection getDocumentsExpiringNextDays(int days) throws SQLException {

Session session = null;

List documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

String sql = "SELECT \n"

+ "document.id, \n"

+ "document.input\_number, \n"

+ "document.reg\_date,\n"

+ "document.`name`,\n"

+ "performer.name AS performerName,\n"

+ "initiator.name AS initiatorName,\n"

+ "deadline,\n"

Приложение Г (продолжение)

+ "status.name AS statusName,\n"

+ "DATEDIFF(deadline,CURDATE()) AS expDays"

+ "\n"

+ "FROM fin\_dept\_docs.document \n"

+ "\n"

+ "JOIN fin\_dept\_docs.initiator ON document.initiator\_id = initiator.id\n"

+ "JOIN fin\_dept\_docs.performer ON document.performer\_id = performer.id\n"

+ "JOIN fin\_dept\_docs.status ON document.status\_id = status.id\n"

+ "\n"

+ "WHERE deadline >= curdate() AND DATEDIFF(deadline,CURDATE()) <=" + days + "\n"

+ "\n"

+ "ORDER BY deadline ASC";

SQLQuery query = session.createSQLQuery(sql);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);

documents = query.list();

} catch (Exception e) {

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

@Override

public Collection getDocumentsExpired() throws SQLException {

Session session = null;

List documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

String sql = "SELECT \n"

+ "document.id, \n"

+ "document.input\_number, \n"

+ "document.reg\_date,\n"

+ "document.`name`,\n"

+ "performer.name AS performerName,\n"

+ "initiator.name AS initiatorName,\n"

+ "deadline,\n"

+ "status.name AS statusName,\n"

+ "DATEDIFF(CURDATE(), deadline) AS expDays"

+ "\n"

+ "FROM fin\_dept\_docs.document \n"

+ "\n"

+ "JOIN fin\_dept\_docs.initiator ON document.initiator\_id = initiator.id\n"

+ "JOIN fin\_dept\_docs.performer ON document.performer\_id = performer.id\n"

+ "JOIN fin\_dept\_docs.status ON document.status\_id = status.id\n"

+ "\n"

+ "WHERE deadline < curdate() AND document.status\_id = 2\n"

+ "\n"

+ "ORDER BY deadline ASC";

SQLQuery query = session.createSQLQuery(sql);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);

Приложение Г (продолжение)

documents = query.list();

} catch (Exception e) {

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

@Override

public Collection getDocumentsStatistics() throws SQLException {

Session session = null;

List documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

String sql = "SELECT \n"

+ "`document`.performer\_id,\n"

+ "`performer`.name AS performer,\n"

+ "(\n"

+ " SELECT count(`document`.id)\n"

+ " FROM `fin\_dept\_docs`.document\n"

+ " WHERE `document`.status\_id = 2 AND `performer`.id = `document`.performer\_id\n"

+ " ) AS documents,\n"

+ "(\n"

+ " SELECT count(`document`.id)\n"

+ " FROM `fin\_dept\_docs`.document\n"

+ " WHERE `document`.deadline < curdate() AND `document`.status\_id = 2 AND `performer`.id = `document`.performer\_id\n"

+ " ) AS exp\n"

+ " \n"

+ "FROM `fin\_dept\_docs`.document\n"

+ "JOIN `fin\_dept\_docs`.performer ON `performer`.id = `document`.performer\_id\n"

+ "\n"

+ "group by performer\n"

+ "order by `performer`.id";

SQLQuery query = session.createSQLQuery(sql);

query.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);

documents = query.list();

} catch (Exception e) {

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

@Override

Приложение Г (продолжение)

public Collection getDocumentsByType(int id) throws SQLException {

Session session = null;

List<Document> tempDocuments = new ArrayList<>();

List<Document> documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

tempDocuments = session.createCriteria(Document.class).list();

if (!tempDocuments.isEmpty()) {

for (int i = 0; i < tempDocuments.size(); i++) {

if (tempDocuments.get(i).getType().getId() == id) {

documents.add(tempDocuments.get(i));

}

}

}

} catch (Exception e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

@Override

public Collection getDocumentsByInitiator(int id) throws SQLException {

Session session = null;

List<Document> tempDocuments = new ArrayList<>();

List<Document> documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

tempDocuments = session.createCriteria(Document.class).list();

if (!tempDocuments.isEmpty()) {

for (int i = 0; i < tempDocuments.size(); i++) {

if (tempDocuments.get(i).getInitiator().getId() == id) {

documents.add(tempDocuments.get(i));

}

}

}

} catch (Exception e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

@Override

public Collection getDocumentsByStatus(int id) throws SQLException {

Session session = null;

List<Document> tempDocuments = new ArrayList<>();

Приложение Г (продолжение)

List<Document> documents = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

tempDocuments = session.createCriteria(Document.class).list();

if (!tempDocuments.isEmpty()) {

for (int i = 0; i < tempDocuments.size(); i++) {

if (tempDocuments.get(i).getStatus().getId() == id) {

documents.add(tempDocuments.get(i));

}

}

}

} catch (Exception e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return documents;

}

}

***InitiatorDAO.java***

package DAO;

import java.sql.SQLException;

import java.util.Collection;

public interface InitiatorDAO {

public Collection getAllInitiators() throws SQLException;

}

***InitiatorDAOImpl.java***

package DAO;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import models.Initiator;

import org.hibernate.Session;

import util.HibernateUtil;

public class InitiatorDAOImpl implements InitiatorDAO {

@Override

public Collection getAllInitiators() throws SQLException {

Session session = null;

List<String> initiators = new ArrayList<>();

try {

Приложение Г (продолжение)

session = HibernateUtil.getSessionFactory().openSession();

initiators = session.createCriteria(Initiator.class).list();

} catch (Exception e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return initiators;

}

}

***PerformerDAO.java***

package DAO;

import java.sql.SQLException;

import java.util.Collection;

import models.Performer;

public interface PerformerDAO {

public Collection getAllPerformers() throws SQLException;

public Performer getPerformerById(int id) throws SQLException;

}

***PerformerDAOImpl.java***

package DAO;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import models.Performer;

import org.hibernate.Session;

import util.HibernateUtil;

public class PerformerDAOImpl implements PerformerDAO {

@Override

public List<Performer> getAllPerformers() throws SQLException {

Session session = null;

List<Performer> performers = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

performers = session.createCriteria(Performer.class).list();

} catch (Exception e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

Приложение Г (продолжение)

session.close();

}

}

return performers;

}

@Override

public Performer getPerformerById(int id) throws SQLException {

Session session = null;

Performer performer = new Performer();

try {

session = HibernateUtil.getSessionFactory().openSession();

performer = (Performer) session.get(Performer.class, id);

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return performer;

}

}

***StatusDAO.java***

package DAO;

import java.sql.SQLException;

import java.util.Collection;

public interface StatusDAO {

public Collection getAllStatuses() throws SQLException;

}

***StatusDAOImpl.java***

package DAO;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import models.Status;

import org.hibernate.Session;

import util.HibernateUtil;

public class StatusDAOImpl implements StatusDAO {

@Override

public Collection getAllStatuses() throws SQLException {

Session session = null;

List<String> statuses = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

statuses = session.createCriteria(Status.class).list();

} catch (Exception e) {

Приложение Г (продолжение)

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return statuses;

}

}

***TypeDAO.java***

package DAO;

import java.sql.SQLException;

import java.util.Collection;

public interface TypeDAO {

public Collection getAllTypes() throws SQLException;

}

***TypeDAOImpl.java***

package DAO;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import models.Type;

import org.hibernate.Session;

import util.HibernateUtil;

public class TypeDAOImpl implements TypeDAO {

@Override

public Collection getAllTypes() throws SQLException {

Session session = null;

List<String> types = new ArrayList<>();

try {

session = HibernateUtil.getSessionFactory().openSession();

types = session.createCriteria(Type.class).list();

} catch (Exception e) {

throw e;

} finally {

if (session != null && session.isOpen()) {

session.close();

}

}

return types;

}

}

Приложение Г (продолжение)

***SessionFactory.java***

package DAO;

public class SessionFactory {

private static DocumentDAO documentDAO = null;

private static TypeDAO typeDAO = null;

private static StatusDAO statusDAO = null;

private static PerformerDAO performerDAO = null;

private static InitiatorDAO initiatorDAO = null;

private static SessionFactory instance = null;

public static synchronized SessionFactory getInstance() {

if (instance == null) {

instance = new SessionFactory();

}

return instance;

}

public DocumentDAO getDocumentDAO() {

if (documentDAO == null) {

documentDAO = new DocumentDAOImpl();

}

return documentDAO;

}

public TypeDAO getTypeDAO() {

if (typeDAO == null) {

typeDAO = new TypeDAOImpl();

}

return typeDAO;

}

public StatusDAO getStatusDAO() {

if (statusDAO == null) {

statusDAO = new StatusDAOImpl();

}

return statusDAO;

}

public PerformerDAO getPerformerDAO() {

if (performerDAO == null) {

performerDAO = new PerformerDAOImpl();

}

return performerDAO;

}

public InitiatorDAO getInitiatorDAO() {

if (initiatorDAO == null) {

initiatorDAO = new InitiatorDAOImpl();

}

return initiatorDAO;

}

}

Приложение Г (продолжение)

***addDocument.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

public class addDocument extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

try (PrintWriter out = response.getWriter()) {

/\* TODO output your page here. You may use following sample code. \*/

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet addDocument</title>");

out.println("</head>");

out.println("<body>");

out.println("<h1>Servlet addDocument at " + request.getContextPath() + "</h1>");

out.println("</body>");

out.println("</html>");

}

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.setCharacterEncoding("UTF-8");

try {

List<String> allTypes = (List<String>) SessionFactory.getInstance().getTypeDAO().getAllTypes();

request.setAttribute("allTypes", allTypes);

List<String> allStatuses = (List<String>) SessionFactory.getInstance().getStatusDAO().getAllStatuses();

request.setAttribute("allStatuses", allStatuses);

List<String> allPerformers = (List<String>) SessionFactory.getInstance().getPerformerDAO().getAllPerformers();

request.setAttribute("allPerformers", allPerformers);

List<String> allInitiators = (List<String>) SessionFactory.getInstance().getInitiatorDAO().getAllInitiators();

request.setAttribute("allInitiators", allInitiators);

Document newDoc = new Document();

request.setAttribute("newDoc", newDoc);

request.getRequestDispatcher("/newdoc.jsp").forward(request, response);

Приложение Г (продолжение)

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.setCharacterEncoding("UTF-8");

try {

String input\_number = request.getParameter("input\_number");

String reg\_date = request.getParameter("reg\_date");

String name = request.getParameter("name");

String type\_id = request.getParameter("type");

String initiator\_id = request.getParameter("initiator");

String status\_id = request.getParameter("status");

String deadline = request.getParameter("deadline");

String performer\_id = request.getParameter("performer");

String desc = request.getParameter("desc");

SessionFactory.getInstance().getDocumentDAO().newDocument(input\_number, reg\_date, name, type\_id, initiator\_id, status\_id, deadline, performer\_id, desc);

List<Document> allDocuments = null;

allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getAllDocuments();

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} catch (SQLException ex) {

Logger.getLogger(addDocument.class.getName()).log(Level.SEVERE, null, ex);

request.getRequestDispatcher("/errordocs.jsp").forward(request, response);

}

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***deleteDocument.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

Приложение Г (продолжение)

import models.Document;

public class deleteDocument extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

try (PrintWriter out = response.getWriter()) {

/\* TODO output your page here. You may use following sample code. \*/

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet delDocument</title>");

out.println("</head>");

out.println("<body>");

out.println("<h1>Servlet delDocument at " + request.getContextPath() + "</h1>");

out.println("</body>");

out.println("</html>");

}

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

int id = Integer.parseInt(request.getParameter("docid"));

SessionFactory.getInstance().getDocumentDAO().deleteDocumentById(id);

List<String> allTypes = (List<String>) SessionFactory.getInstance().getTypeDAO().getAllTypes();

request.setAttribute("allTypes", allTypes);

List<String> allStatuses = (List<String>) SessionFactory.getInstance().getStatusDAO().getAllStatuses();

request.setAttribute("allStatuses", allStatuses);

List<String> allPerformers = (List<String>) SessionFactory.getInstance().getPerformerDAO().getAllPerformers();

request.setAttribute("allPerformers", allPerformers);

List<String> allInitiators = (List<String>) SessionFactory.getInstance().getInitiatorDAO().getAllInitiators();

request.setAttribute("allInitiators", allInitiators);

List<Document> allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getAllDocuments();

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

request.getRequestDispatcher("/errordocs.jsp").forward(request, response);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

Приложение Г (продолжение)

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***downloadFile.java***

package servlets;

import DAO.SessionFactory;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.net.URLEncoder;

import java.sql.SQLException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class downloadFile extends HttpServlet {

private static final String SAVE\_DIR = "data\\files";

@Override

protected void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

response.setCharacterEncoding("UTF-8");

request.setCharacterEncoding("UTF-8");

int docid = Integer.parseInt(request.getParameter("docid"));

String appPath = getServletContext().getRealPath("");

String savePath = appPath + File.separator + SAVE\_DIR;

String docFileName = null;

try {

docFileName = SessionFactory.getInstance().getDocumentDAO().getDocumentById(docid).getFile();

String filePath = savePath + File.separator + docFileName;

if (docFileName == null) {

request.getRequestDispatcher("/errordownload.jsp").forward(request, response);

} else {

File downloadFile = new File(filePath);

String fileName = URLEncoder.encode(downloadFile.getName(), "UTF-8");

OutputStream outStream;

try (FileInputStream inStream = new FileInputStream(downloadFile)) {

Приложение Г (продолжение)

String relativePath = getServletContext().getRealPath("");

System.out.println("relativePath = " + relativePath);

ServletContext context = getServletContext();

String mimeType = context.getMimeType(filePath);

if (mimeType == null) {

mimeType = "application/octet-stream";

}

response.setContentType(mimeType);

response.setContentLength((int) downloadFile.length());

response.setHeader("Content-Disposition", "attachment;filename=" + fileName);

outStream = response.getOutputStream();

byte[] buffer = new byte[4096];

int bytesRead = -1;

while ((bytesRead = inStream.read(buffer)) != -1) {

outStream.write(buffer, 0, bytesRead);

}

}

outStream.close();

}

} catch (SQLException ex) {

Logger.getLogger(downloadFile.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***editDocument.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

public class editDocument extends HttpServlet {

@Override

Приложение Г (продолжение)

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

int id = Integer.parseInt(request.getParameter("docid"));

Document currentDoc = SessionFactory.getInstance().getDocumentDAO().getDocumentById(id);

List<String> allTypes = (List<String>) SessionFactory.getInstance().getTypeDAO().getAllTypes();

request.setAttribute("allTypes", allTypes);

List<String> allStatuses = (List<String>) SessionFactory.getInstance().getStatusDAO().getAllStatuses();

request.setAttribute("allStatuses", allStatuses);

List<String> allPerformers = (List<String>) SessionFactory.getInstance().getPerformerDAO().getAllPerformers();

request.setAttribute("allPerformers", allPerformers);

List<String> allInitiators = (List<String>) SessionFactory.getInstance().getInitiatorDAO().getAllInitiators();

request.setAttribute("allInitiators", allInitiators);

if (currentDoc != null) {

request.setAttribute("currentDoc", currentDoc);

request.getRequestDispatcher("/editdocs.jsp").forward(request, response);

} else {

request.getRequestDispatcher("/errordocs.jsp").forward(request, response);

}

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

request.getRequestDispatcher("/errordocs.jsp").forward(request, response);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

private void processRequest(HttpServletRequest request, HttpServletResponse response) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

}

***performers.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.SQLException;

Приложение Г (продолжение)

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

import models.Performer;

@WebServlet(name = "performers", urlPatterns = {"/performers"})

public class performers extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

try (PrintWriter out = response.getWriter()) {

/\* TODO output your page here. You may use following sample code. \*/

out.println("<!DOCTYPE html>");

out.println("<html>");

out.println("<head>");

out.println("<title>Servlet performers</title>");

out.println("</head>");

out.println("<body>");

out.println("<h1>Servlet performers at " + request.getContextPath() + "</h1>");

out.println("</body>");

out.println("</html>");

}

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

int id = Integer.parseInt(request.getParameter("performerid"));

if (id == 0) { // страница по умолчанию

List<Performer> allPerformers = (List<Performer>) SessionFactory.getInstance().getPerformerDAO().getAllPerformers();

request.setAttribute("allPerformers", allPerformers);

request.getRequestDispatcher("/performers.jsp").forward(request, response);

}

if (id > 0) { // страница с выбранным исполнителем

List<Performer> allPerformers = (List<Performer>) SessionFactory.getInstance().getPerformerDAO().getAllPerformers();

request.setAttribute("allPerformers", allPerformers);

Performer currentPerformer = (Performer) SessionFactory.getInstance().getPerformerDAO().getPerformerById(id);

request.setAttribute("currentPerformer", currentPerformer);

List<Document> performerDocs = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getDocumentsByPerformer(id);

request.setAttribute("performerDocs", performerDocs);

Приложение Г (продолжение)

List<Document> performersStat = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getDocumentsStatistics();

request.setAttribute("performersStat", performersStat);

request.getRequestDispatcher("/performers.jsp").forward(request, response);

} else {

request.getRequestDispatcher("/errordocs.jsp").forward(request, response);

}

} catch (SQLException ex) {

Logger.getLogger(performers.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***showAllDocs.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

public class showAllDocs extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

List<Document> allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getAllDocuments();

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

Приложение Г (продолжение)

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//processRequest(request, response);

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***showDocsByInitiator.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

@WebServlet(name = "showDocsByInitiator", urlPatterns = {"/showdocsbyinitiator"})

public class showDocsByInitiator extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

int id = Integer.parseInt(request.getParameter("initiatorid"));

List<Document> allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getDocumentsByInitiator(id);

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//processRequest(request, response);

}

Приложение Г (продолжение)

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***showDocsByStatus.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

@WebServlet(name = "showDocsByStatus", urlPatterns = {"/showdocsbystatus"})

public class showDocsByStatus extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

int id = Integer.parseInt(request.getParameter("statusid"));

List<Document> allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getDocumentsByStatus(id);

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//processRequest(request, response);

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***showDocsByType.java***

package servlets;

Приложение Г (продолжение)

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

@WebServlet(name = "showDocsByType", urlPatterns = {"/showdocsbytype"})

public class showDocsByType extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

int id = Integer.parseInt(request.getParameter("typeid"));

List<Document> allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getDocumentsByType(id);

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//processRequest(request, response);

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***showTerms.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

Приложение Г (продолжение)

import javax.servlet.http.HttpServletResponse;

import models.Document;

@WebServlet(name = "showTerms", urlPatterns = {"/showterms"})

public class showTerms extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

List<Document> expiredDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getDocumentsExpired();

request.setAttribute("expiredDocuments", expiredDocuments);

List<Document> expiringNextDaysDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getDocumentsExpiringNextDays(10);

request.setAttribute("expiringNextDaysDocuments", expiringNextDaysDocuments);

request.getRequestDispatcher("/terms.jsp").forward(request, response);

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

//processRequest(request, response);

}

@Override

public String getServletInfo() {

return "Short description";

}

}

***updateDocument.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

public class updateDocument extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

Приложение Г (продолжение)

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.setCharacterEncoding("UTF-8");

try {

int id = Integer.parseInt(request.getParameter("docid"));

Document currentDoc = SessionFactory.getInstance().getDocumentDAO().getDocumentById(id);

List<String> allTypes = (List<String>) SessionFactory.getInstance().getTypeDAO().getAllTypes();

request.setAttribute("allTypes", allTypes);

List<String> allStatuses = (List<String>) SessionFactory.getInstance().getStatusDAO().getAllStatuses();

request.setAttribute("allStatuses", allStatuses);

List<String> allPerformers = (List<String>) SessionFactory.getInstance().getPerformerDAO().getAllPerformers();

request.setAttribute("allPerformers", allPerformers);

List<String> allInitiators = (List<String>) SessionFactory.getInstance().getInitiatorDAO().getAllInitiators();

request.setAttribute("allInitiators", allInitiators);

String input\_number = request.getParameter("input\_number");

String reg\_date = request.getParameter("reg\_date");

String name = request.getParameter("name");

String type\_id = request.getParameter("type");

String initiator\_id = request.getParameter("initiator");

String status\_id = request.getParameter("status");

String deadline = request.getParameter("deadline");

String performer\_id = request.getParameter("performer");

String file = request.getParameter("file");

String desc = request.getParameter("description");

SessionFactory.getInstance().getDocumentDAO().updateDocument(id, input\_number, reg\_date, name,

type\_id, initiator\_id, status\_id, deadline, performer\_id, file, desc);

if (currentDoc != null) {

List<Document> allDocuments = null;

allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getAllDocuments();

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} else {

request.getRequestDispatcher("/errordocs.jsp").forward(request, response);

}

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

@Override

Приложение Г (продолжение)

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}

***uploadFile.java***

package servlets;

import DAO.SessionFactory;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import models.Document;

public class updateDocument extends HttpServlet {

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.setCharacterEncoding("UTF-8");

try {

int id = Integer.parseInt(request.getParameter("docid"));

Document currentDoc = SessionFactory.getInstance().getDocumentDAO().getDocumentById(id);

List<String> allTypes = (List<String>) SessionFactory.getInstance().getTypeDAO().getAllTypes();

request.setAttribute("allTypes", allTypes);

List<String> allStatuses = (List<String>) SessionFactory.getInstance().getStatusDAO().getAllStatuses();

request.setAttribute("allStatuses", allStatuses);

List<String> allPerformers = (List<String>) SessionFactory.getInstance().getPerformerDAO().getAllPerformers();

request.setAttribute("allPerformers", allPerformers);

List<String> allInitiators = (List<String>) SessionFactory.getInstance().getInitiatorDAO().getAllInitiators();

request.setAttribute("allInitiators", allInitiators);

String input\_number = request.getParameter("input\_number");

String reg\_date = request.getParameter("reg\_date");

String name = request.getParameter("name");

String type\_id = request.getParameter("type");

String initiator\_id = request.getParameter("initiator");

String status\_id = request.getParameter("status");

String deadline = request.getParameter("deadline");

String performer\_id = request.getParameter("performer");

String file = request.getParameter("file");

Приложение Г (продолжение)

String desc = request.getParameter("description");

SessionFactory.getInstance().getDocumentDAO().updateDocument(id, input\_number, reg\_date, name,

type\_id, initiator\_id, status\_id, deadline, performer\_id, file, desc);

if (currentDoc != null) {

List<Document> allDocuments = null;

allDocuments = (List<Document>) SessionFactory.getInstance().getDocumentDAO().getAllDocuments();

request.setAttribute("allDocuments", allDocuments);

request.getRequestDispatcher("/alldocs.jsp").forward(request, response);

} else {

request.getRequestDispatcher("/errordocs.jsp").forward(request, response);

}

} catch (SQLException ex) {

Logger.getLogger(showAllDocs.class.getName()).log(Level.SEVERE, null, ex);

}

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

processRequest(request, response);

}

@Override

public String getServletInfo() {

return "Short description";

}// </editor-fold>

}