Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Владимирский государственный университет

им. А.Г. и Н.Г. Столетовых

Кафедра информационных систем и программной инженерии

***КУРСОВОЙ ПРОЕКТ***

*Прототип информационной системы «Документооборот малого предприятия»*

*Выполнил: студент гр. ЗИСТд-119*

*Зыков И.С.*

*Принял:*

*Владимир 2024*

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc484641976)

[1 ПОСТАНОВКА ЗАДАЧИ 3](#_Toc484641977)

[2 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ 5](#_Toc484641978)

[2.1 Описание предметной области 5](#_Toc484641979)

[2.2 Требования к системе 6](#_Toc484641980)

[2.3 Сценарий взаимодействия пользователя с системой 6](#_Toc484641981)

[2.4 Аутентификация 7](#_Toc484641982)

[2.5 Главная страница 7](#_Toc484641983)

[2.6 Просмотр архива 8](#_Toc484641984)

[2.7 Просмотр документов 8](#_Toc484641985)

[2.8 Просмотр персонала, гостов, компаний и заказчиков 9](#_Toc484641986)

[2.9 Добавление в базу 10](#_Toc484641987)

[3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ 11](#_Toc484641989)

[3.1 Общая организация системы 11](#_Toc484641990)

[3.2 Организация бизнес-логики 12](#_Toc484641991)

[3.3 Организация веб-интерфейса 14](#_Toc484641992)

[4 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ 18](#_Toc484641993)

[4.1 Реализация бизнес-логики системы 19](#_Toc484641994)

[4.2 Реализация web-интерфейса системы 22](#_Toc484641995)

[4.3 Структура базы данных 25](#_Toc484641996)

[4.4 Реализация доступа к данным 26](#_Toc484641999)

[4.5 Обеспечение целостности данных 28](#_Toc484642000)

[5 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ 30](#_Toc484642002)

[ЗАКЛЮЧЕНИЕ 33](#_Toc484642003)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc484642004)

[ПРИЛОЖЕНИЕ А Листинг компонентов системы 35](#_Toc484642005)

ВВЕДЕНИЕ

Документ является основным способом предоставления информации на любом современном предприятии. Неоспорима важность сохранение и умелое использования информационных ресурсов предприятия для успешного ведения бизнеса.

Тема делопроизводства в системе управления имеет сейчас большую актуальность. Эффективность управления предприятием зависит от того, на сколько разумно организованно управление документооборотом. Фактически, малоэффективное использование накопленной информации (или, еще, хуже ее утрата) может привести к потере всего бизнеса.

Ведь вовремя использованная информация или документ – это, прежде всего, потерянные деньги, время и упущенные возможности. В следствие этого, на любом предприятии, где ведется активная работа с различными документами, рано или поздно встает проблема систематизации, обработки и безопасного хранения значительных объемов информации. Важную роль в оптимизации деятельности предприятия любого размера и профиля деятельности играют современные системы электронного документооборота.

В данном курсовом проекте был реализован прототип программной системы «Документооборт малого предприятия».

Данную программную систему можно использовать на любом предприятии, на котором осущевстляется документооборт.

Программная система облегчает процесс поиска и просморта документов и ведения учета. Человеку просто нужно зайти в программную систему и найти необходимую информацию.

1 ПОСТАНОВКА ЗАДАЧИ

**Разработать следующие вопросы:**

1. Анализ предметной области, функциональная декомпозиция системы;
2. Структура базы данных, содержащая не менее 6 взаимосвязанных таблиц;
3. Структура программной системы с разделением на бизнес-логику и логику веб-интерфейса;
4. Реализация бизнес-логики и веб-интерфейса с использованием заданных технологий;
5. Обеспечение целостности данных;
6. Нагрузочное тестирование прототипа.

**Исходные данные:**

1. Sun JDK1.8.0 Java EE 6 – совместимый сервер приложений Glassfish 4.1;
2. СУБД: MySQL 5.7;
3. Технологии: JSF 2.2, JPA 2.1, EJB 3.1;
4. Базы данных: Документы, Котрагенты;
5. Функции прототипа: просмотр докуметации, учет документов (добавление в архив).

**Функционал системы:**

1. Работа с просмотром данных;
2. Работа с редактирование данных;
3. Работа с пользователями.

Данная программная система облегчает процесс документооборота. Обычный пользователь системы – посетитель сайта может создать документ и провести его, а аткже просмотреть дополнительную информацию о госте и персонале.

В данном проекте будет использоваться второе хранилище данных, которое будет содержать информацию о контрагентах, а также данные пользователей системы. Данная специфика необходима для обеспечения безопасности данных, связанных с частью системы, отвечающей за финансовую составляющую предприятия и за безопасность персональных данных пользователей.

Результат обработки этих данных будет реплицироваться на первую базу данных.

2 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ

2.1 Описание предметной области

В данном курсовом проекте был реализован прототип программной системы «Документооборот малого предприятия».

Предметная область – документооборот малого предприятия. У предприятия есть всходящая и исходящая документация, после того как документ проведен, он передается в архив со всеми внесенными его данными. Все документы опираются на ГОСТы. Документы заключаются с компаниями-контрагентами. У каждой такой компании есть закрытый список участников договора, которые могут учувствовать в проведении документа. Для оформления документа у каждой компании есть так называемое «кодовое слово», которое сотрудник должен указать, при оформлении заявки на документ. 

Рисунок 1. IDEF0-диаграмма AS IS процесса проведения документа

Для полноценного пользования системой человеку необходимо авторизоваться. Затем он может просматривать элементы системы

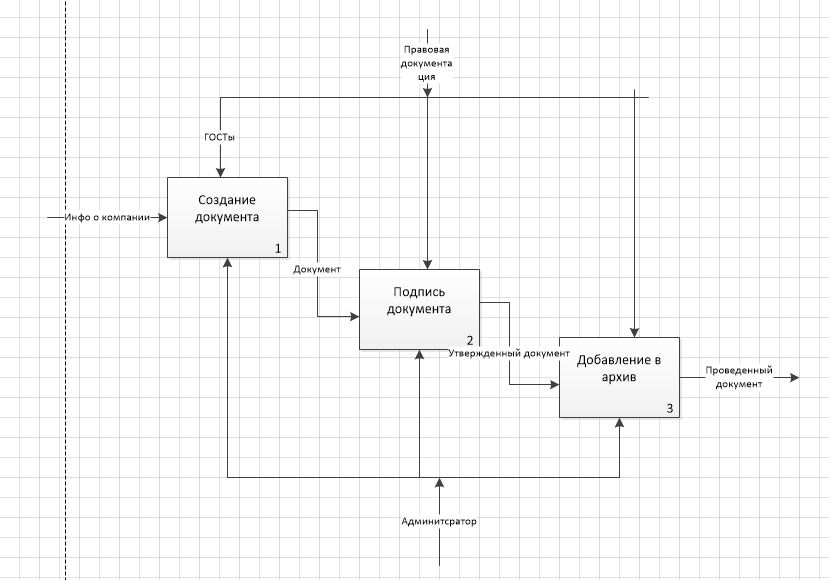


Рисунок 2. Декомпозиция процесса

2.2 Требования к системе

После анализа предметной области «Документооборот предприятия» были выделены следующие сущности.

Требования:

1. Система работает в беспрерывном режиме
2. Восстановление после отказа системы не менее 15 минут 30 секунд
3. Обслуживание 120 пользователей одновременноего права на создание отчетов о доходе.
   1. Сценарий взаимодействия пользователя с системой

Все, что нужно пользователю для работы – доступ к интернету. Взаимодействие с системой осуществяется посредством браузера. Для использования системы пользователь должен быть заведен в системе и должен быть авторизован.

* 1. Аутентификация

Пользователь заходит на сайт для документообората малого предприятия. Там он вводит свой логин и пароль, а затем нажимает кнопку «Войти». Если вход был выполнен, то его перекидывает на главную страницу, в случае введения неправильных данных пользователь перемещается на страницу с ошибкой.

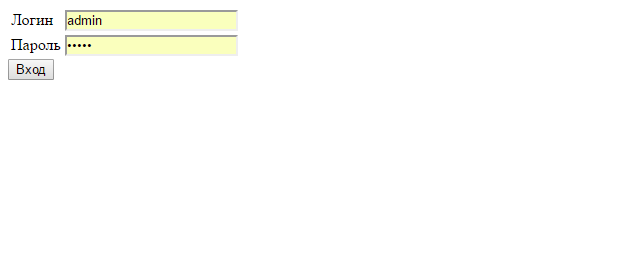


Рисунок 3. Аутентификация

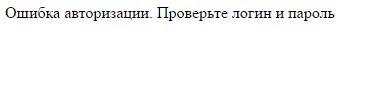


Рисунок 4. Неверные данные

* 1. Главная страница

На главной странице располагаются ссылки на просмотр отображения сущностей системы, также отображается имя пользователя, который авторизоался в системе (рис.6).



Рисунок 6. Главная страница

* 1. Просмотр архива

С главной страницы пользователь может перейти на страницу маршрутов, нажав кнопку «Архив». На данной странице (рис.7) пользователь может просмотреть список всех документов в архиве.

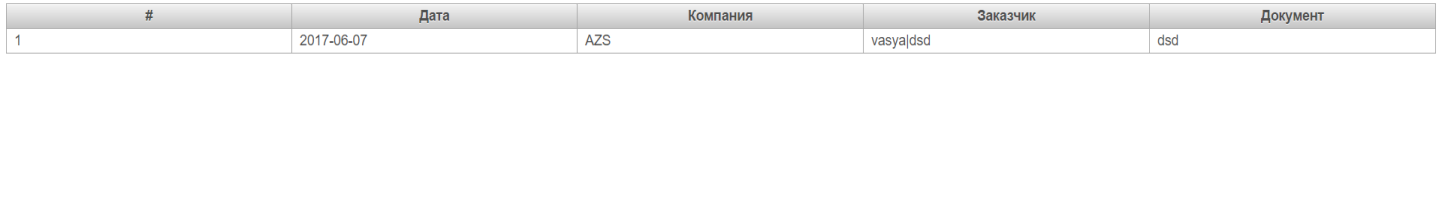


Рисунок 7. Просмотр списка архива

* 1. Просмотр документов

С главной страницы пользователь может перейти на страницу маршрутов, нажав кнопку «Документы». На данной странице (рис.8) пользователь может просмотреть список документов.

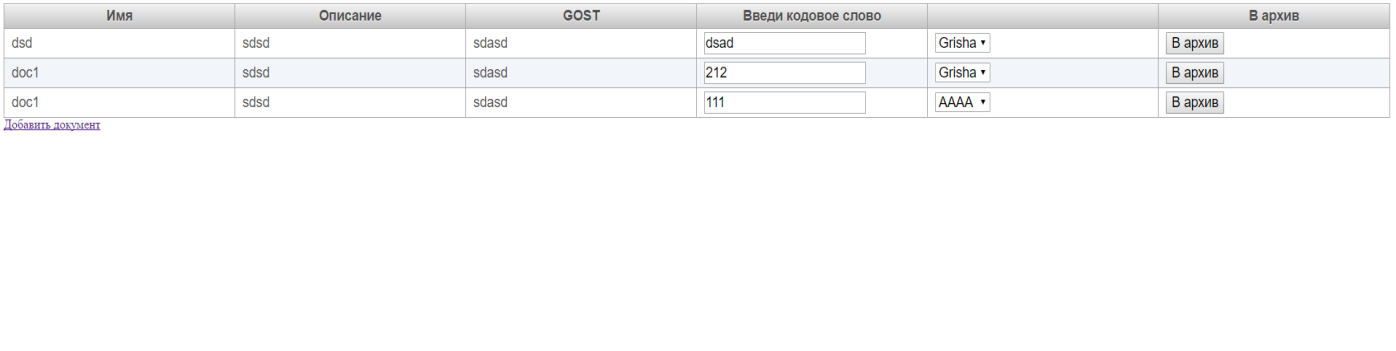


Рисунок 8. Документы

* 1. Просмотр персонала, гостов, компаний и заказчиков

Аналогично преддущим пунктам можно посмотреть данные о других сущностях (рис.9).

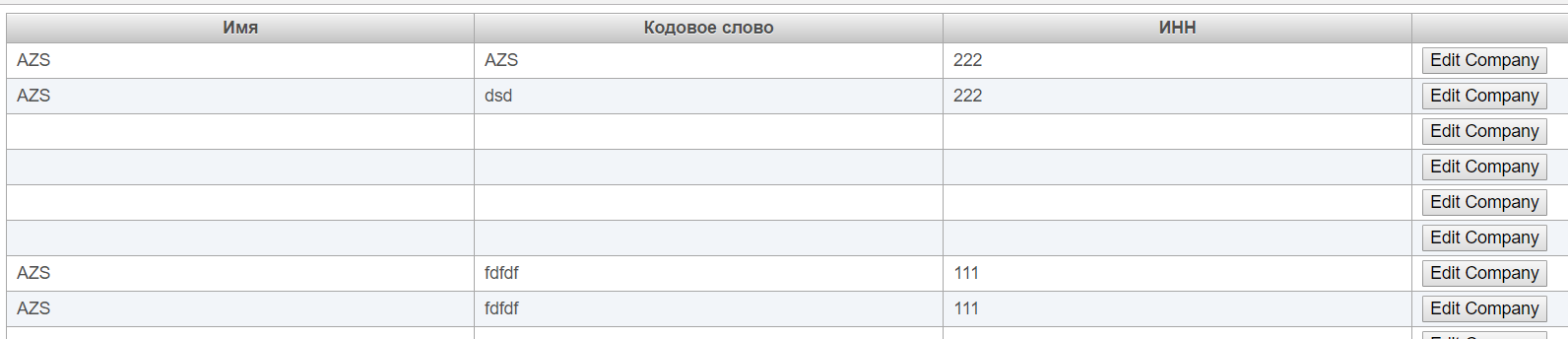


Рисунок 9. Просмотр компаний

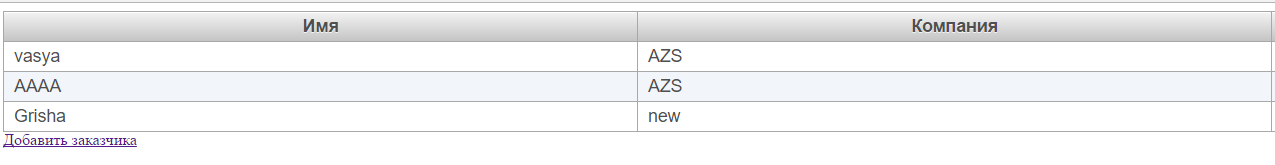


Рисунок 10. Просмотр заказчиков

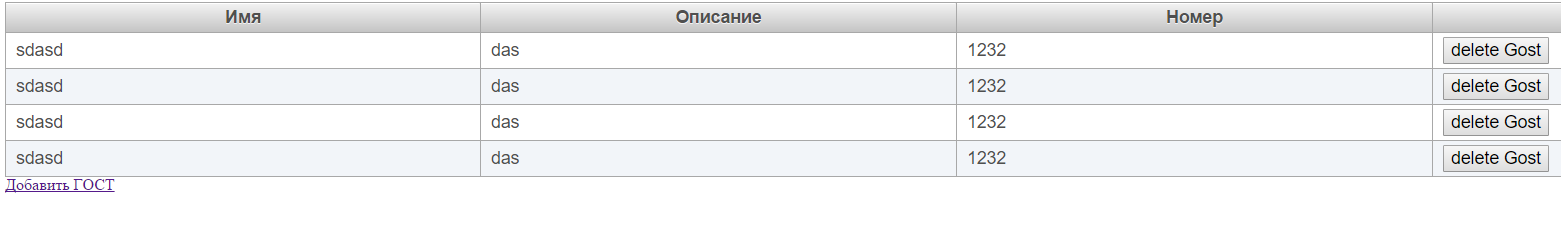


Рисунок 11. Просмотр ГОСТов.

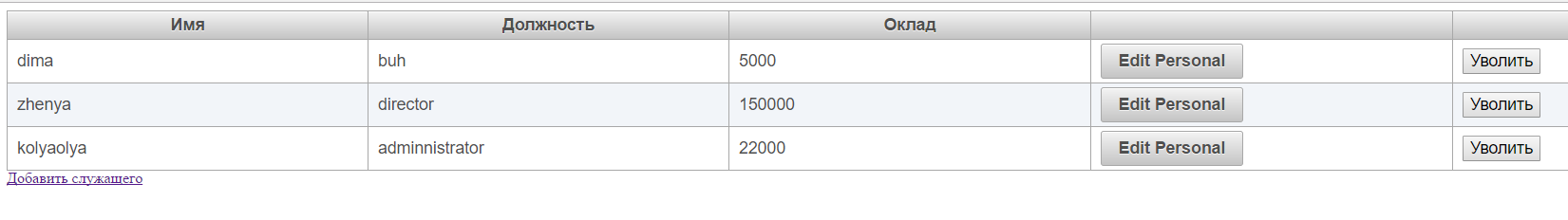


Рисунок 12. Просмотр персонала

* 1. Добавление в базу

Добавление к сущостям происходит одинаково для всех сущностей, рассмотрим на примере добавления документа.

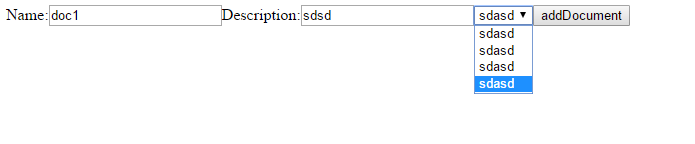


Рисунок 12. Добавление документа

2.10 Просмотр персонала, гостов, компаний и заказчиков

Редактирование сщности происходит при помощи кнопки «Edit». Пользователь перенаправляются на страницу с редактирование сщности.

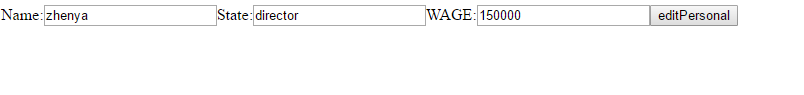


Рисунок 13. Редактиование прсонала

1. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

3.1 Общая организация системы

Model-view-controller (MVC, «Модель-вид-поведение», «Модель-представление-контроллер») — архитектура программного обеспечения, в которой модель данных приложения, пользовательский интерфейс и управляющая логика разделены на три отдельных компонента, так, что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты. Шаблон MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента.

Модель. Предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

Представление. Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает [форма (окно)](https://ru.wikipedia.org/wiki/%D0%9E%D0%BA%D0%BD%D0%BE_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) с графическими элементами.

Контроллер. Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

Основная цель применения этой концепции состоит в разделении бизнес-логики от её визуализации. За счет такого разделения повышается возможность [повторного использования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D1%82%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения.

Слой представления - это единственный слой, непосредственно связанный с пользователем. Слой представления используется для получения данных от пользователя и передачи их слою бизнес-логики для дальнейшей обработки, и когда данные получены в объекте-значении, он отвечает за представление объекта-значения в надлежащей форме, которая понятна пользователю.

Слой бизнес-логики (BUS) работает как мост между уровнем представления и DAO. Все значения пользователя, полученные от уровня представления, передаются в BUS. Слой бизнес-логики содержит всю бизнес-логику программы.

Слой доступа к базе данных (DAO) строит запрос на основе параметров, полученных от слоя бизнес-логики и используя соединение с базой данных, выполняет требуемые операции. Затем происходит возврат результатов из DAO классов в слой бизнес-логики.

* 1. Организация бизнес-логики

Диаграммой прецедентов называется диаграмма, на которой показана совокупность прецедентов и актеров, а также отношения между ними. Диаграмма прецедентов представлена на рисунке 14.

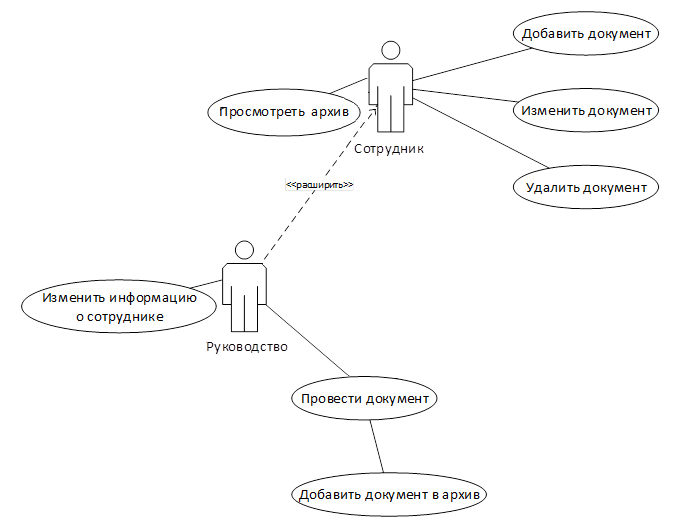


Рисунок 14. Диаграмма прецедентов

Сотрудник – участник нашей компании

Руководство – участник нашей компании с расширенными полномочиями, часть руководства

Описание прецедентов:

**Название: Добавить документ в архив**

Актер: Сотрудник

Предусловие: Создан документ

Основной поток: Вся информация верна, заявитель есть в закрытом списке, кодовое слово указано верно, документ проводится.

Альтернативный поток: Заявителя нет в закрытых списках или неверно кодовое слово, заявитель получает отказ.

Постусловие: -

**Название: Добавить документ**.

Актер: Сотрудник

Предусловие: Сотрудник входит в раздел создания документа

Основной поток: Документ добавлен.

Альтернативный поток: Не выполнено предусловие, функция недоступна

Постусловие: -

* 1. Организация веб-интерфейса

Современные концепции проектирования распределенных информационных систем корпоративного уровня предполагают общение пользователя системы с базой данных посредством браузера и технологий Internet. Для обеспечения возможности взаимодействия пользователя с браузером применяется Веб-интерфейс — это совокупность средств, при помощи которых пользователь взаимодействует с веб-сайтом или любым другим приложением через браузер.

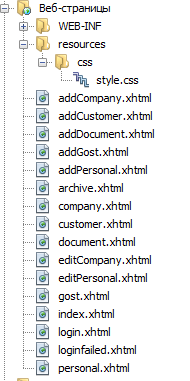


Рисунок 15. Веб-страницы

В ходе анализа системы был разработан слой представления. Компоненты представления находятся в war модуле. Каждый класс пердставления организует проежуточный слой между данными и их отображением. Данные классы идентичны, приведем в пример ArchiveView:

@Named(value = "archiveView")

@SessionScoped

public class ArchiveView implements Serializable{

@EJB

ArchiveFacadeLocal archiveFacade;

@EJB

DocumentFacadeLocal documentFacade;

public ArchiveView(){

this.archive = new Archive();

this.codeName = new String();

}

private Archive archive;

private String codeName;

public String getCodeName() {

return codeName;

}

public void setCodeName(String codeName) {

this.codeName = codeName;

}

public Archive getArchive() {

return archive;

}

public void setArchive(Archive archive) {

this.archive = archive;

}

public List<Archive> getAllArhive(){

List<Archive> list = archiveFacade.findAll();

return list;

}

public void addToArchive(int idDoc){

Document document = documentFacade.find(idDoc);

archiveFacade.addDoc(document, codeName, archive.getCustomer());

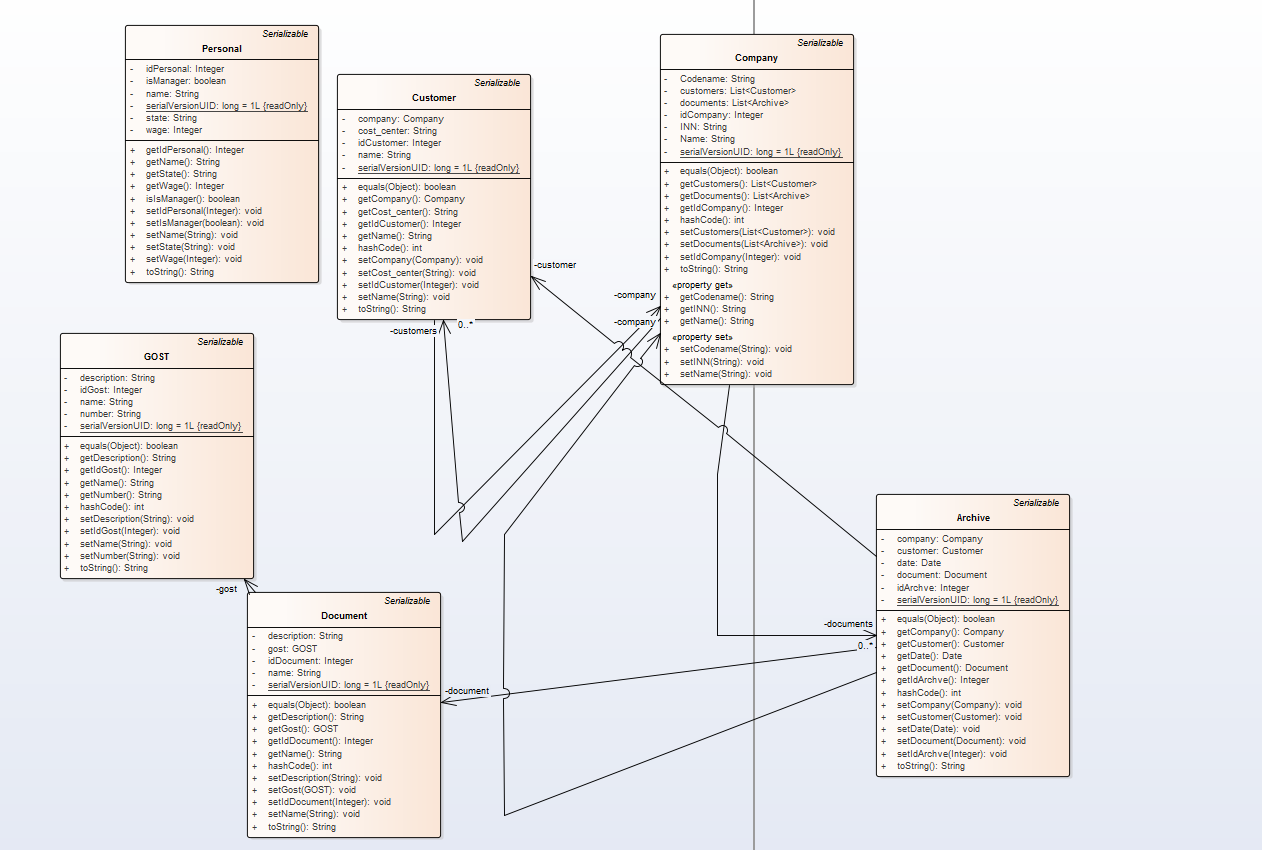
}

}

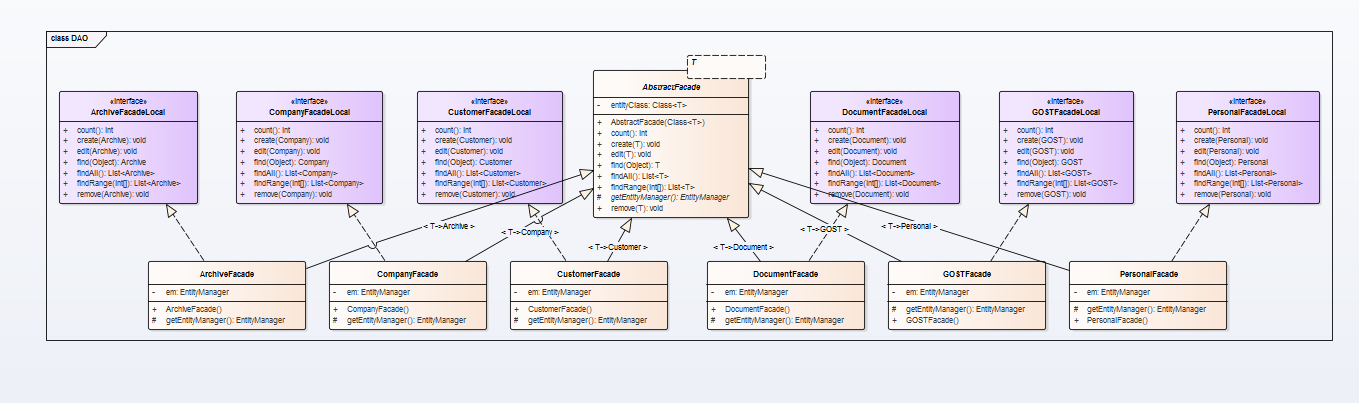
4 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ

В рамках программной системы была реализована структура классов, указанная на рисунке 16, включающая в себя сущности, классы бизнес-логики и классы веб-интерфейса.

Уровень сущности



Доступ к данным



Бизнесс-логика/представление

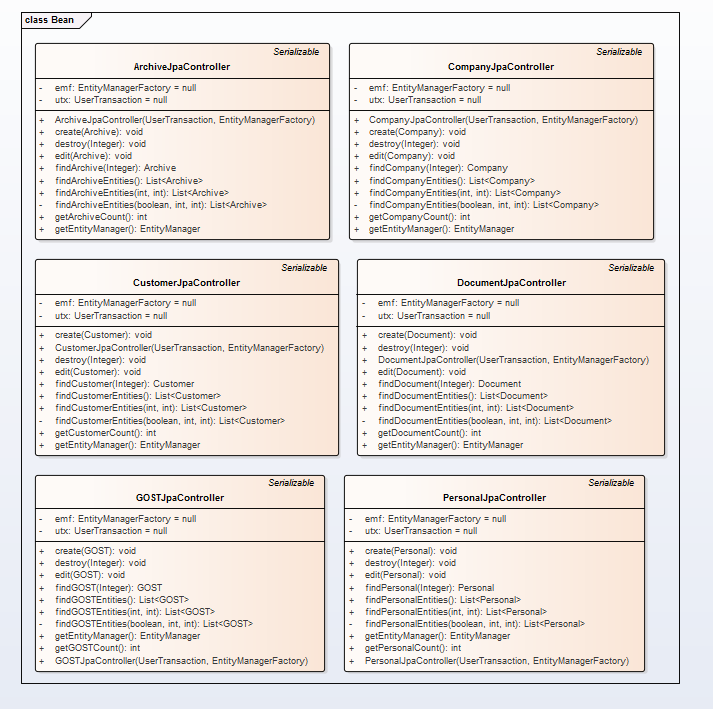


Рисунок 16. Диаграмма классов уровня реализации

* 1. Реализация бизнес-логики системы

Бизнес-логика прототипа реализована при помощи технологии EJB. EJB – это компоненты, которые предназначены для реализации так называемой бизнес-логики приложения. В отличие от веб-компонентов, которые обеспечивают веб-интерфейс приложения, EJB-компоненты не формируют какого-либо пользовательского интерфейса. Основная задача у них – обеспечить реализацию процедур обработки данных и бизнес-правил, специфичных для приложения. EJB-компоненты формируют промежуточный слой, который обеспечивает изоляцию клиентских приложений (веб-приложений, веб-сервисов, GUI-приложений) от деталей внутренней организации данных (типа и местоположения БД, используемого для связи с ними API и их внутренней структуры), предоставляя им некоторый прикладной программный интерфейс, ориентированный на решение определенного круга задач для конкретного клиента. Были использованы EJB-компоненты без хранения состояния и компоненты-сущности.

Настроенный EJB модуль в составе Enterprise JavaEE приложения:

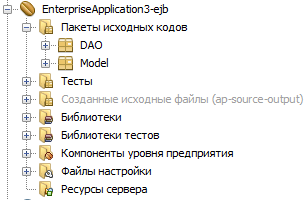


Рисунок 17. EJB модуль

Набор классов доступа к данным:

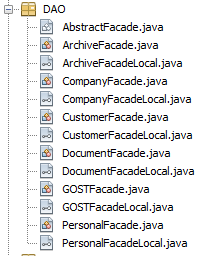


Рисунок 18. DAO

Для обращения к бд были созданы фасады. Каждый фасад наследует основные CRUD методы от Абстрактного фасада, в котором прописаны обращение к классу-сущности. Также созданы локальные интерфейсы.

public abstract class AbstractFacade<T> {

private Class<T> entityClass;

public AbstractFacade(Class<T> entityClass) {

this.entityClass = entityClass;

}

protected abstract EntityManager getEntityManager();

public void create(T entity) {

getEntityManager().persist(entity);

}

public void edit(T entity) {

getEntityManager().merge(entity);

}

public void remove(T entity) {

getEntityManager().remove(getEntityManager().merge(entity));

}

public T find(Object id) {

return getEntityManager().find(entityClass, id);

}

public List<T> findAll() {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

cq.select(cq.from(entityClass));

return getEntityManager().createQuery(cq).getResultList();

}

public List<T> findRange(int[] range) {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

cq.select(cq.from(entityClass));

javax.persistence.Query q = getEntityManager().createQuery(cq);

q.setMaxResults(range[1] - range[0] + 1);

q.setFirstResult(range[0]);

return q.getResultList();

}

public int count() {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

javax.persistence.criteria.Root<T> rt = cq.from(entityClass);

cq.select(getEntityManager().getCriteriaBuilder().count(rt));

javax.persistence.Query q = getEntityManager().createQuery(cq);

return ((Long) q.getSingleResult()).intValue();

}

}

* 1. Реализация web-интерфейса системы

Для реализации веб-интерфейса прототипа информационной системы был использован фреймворк JavaServer Faces. JavaServer Faces (JSF) — это фреймворк для веб-приложений, написанный на Java. Он служит для того, чтобы облегчать разработку пользовательских интерфейсов для Java EE-приложений. В отличие от прочих MVC-фреймворков, которые управляются запросами, подход JSF основывается на использовании компонентов. Состояние компонентов пользовательского интерфейса сохраняется когда пользователь запрашивает новую страницу и затем восстанавливается, если запрос повторяется. Кроме того, была использована библиотека PrimeFaces.

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:f="http://xmlns.jcp.org/jsf/core"

xmlns:c="http://xmlns.jcp.org/jsp/jstl/core"

xmlns:p="http://primefaces.org/ui">

<h:head>

<title>Facelet Title</title>

</h:head>

<h:body>

<f:view>

<h:form>

<p:dataTable value="#{documentView.allDocument}" var="document" >

<p:column>

<f:facet name="header">Имя</f:facet>

<h:outputText value = "#{document.name}"/>

</p:column>

<p:column>

<f:facet name="header">Описание</f:facet>

<h:outputText value = "#{document.description}"/>

</p:column>

<p:column>

<f:facet name="header">GOST</f:facet>

#{document.gost.name}

</p:column>

<p:column>

<f:facet name="header">Введи кодовое слово</f:facet>

<h:inputText value="#{archiveView.codeName}"/>

</p:column>

<p:column>

<h:selectOneMenu value="#{archiveView.archive.customer}" converter="#{customerConverter}">

<f:selectItems value="#{customerView.allCustomers}" var="customer"

itemValue="#{customer}" itemLabel="#{customer.name}" />

</h:selectOneMenu>

</p:column>

<p:column>

<f:facet name="header">В архив</f:facet>

<h:commandButton action="#{archiveView.addToArchive(document.idDocument)}" value="В архив"/>

</p:column>

</p:dataTable>

<h:outputLink value="faces/addDocument.xhtml">Добавить документ</h:outputLink><br></br>

</h:form>

</f:view>

</h:body>

</html>

* 1. Структура базы данных

Физические модели определяет способ размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне.

База данных 1

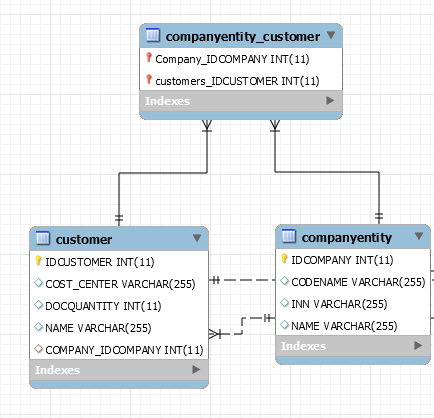


Рисунок 19. Физическая модель БД контрагентов

База данных 2

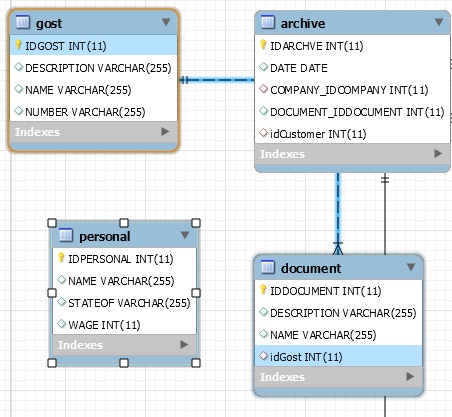


Рисунок 20. Физическая модель БД документов

* 1. Реализация доступа к данным

Доступ к данным организован на основе технологии Java Persistence API. Спецификация Java Persistence API определяет набор прикладных интерфейсов языка Java для управления персистентностью и объектно-реляционного преобразования на платформах Java SE и Java EE.

Сущность – это легкий устойчивый объект предметной области, связанный с БД с помощью аннотации @Entity. Сущности представляют работу с данными как с системой объектов, при этом состояние объектов, моделируемое их полями, отражает состояние данных в соответствующем источнике. Вызывая методы объектов, клиент изменяет их состояние. Изменение состояния объектов синхронизируется с источником данных автоматически под управлением контейнера или по требованию приложения. Клиент работает с экземпляром сущности напрямую, логика реализуется классом сущности.

@Entity

@Table(name="companyEntity")

public class Company implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer idCompany;

private String Name;

private String INN;

private String Codename;

@OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.REMOVE)

private List<Customer> customers;

@OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.REMOVE)

private List<Archive> documents;

Модуль персистентности представляет собой средство упаковки и установки классов сущностей. Модуль персистентности позволяет отобразить управляемые с его помощью классы сущностей на единственную реляционную базу данных, определяя, таким образом, контекст для запросов и связей между сущностями. Это позволяет представить реляционную БД сложной структуры в виде композиции проблемно-ориентированных модулей персистентности.

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

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd">

<persistence-unit name="EAEA-warPU" transaction-type="JTA">

<jta-data-source>jdbc/myzoo</jta-data-source>

<class>entities.Author</class>

<class>entities.Message</class>

<class>entities.Client</class>

<class>entities.Company</class>

<exclude-unlisted-classes>true</exclude-unlisted-classes>

<properties>

</properties>

</persistence-unit>

<persistence-unit name="EAEA-warPU2" transaction-type="JTA">

<jta-data-source>jdbc/flower</jta-data-source>

<class>entities.Orders</class>

<exclude-unlisted-classes>true</exclude-unlisted-classes>

<properties>

<property name="javax.persistence.schema-generation.database.action" value="create"/>

</properties>

</persistence-unit>

</persistence>

* 1. Обеспечение целостности данных

В базе данных организована ссылочная целостность благодаря тому, что все отношения связаны при помощи механизма внешних ключей. Поэтому невозможно удалить запись, связанную с другой записью посредством внешнего ключа.

Организация распределенных транзакций.

Транзакция – это последовательность операторов манипулирования данными, выполняющаяся как единое целое (все или ничего) и переводящая базу данных из одного целостного состояния в другое целостное состояние.

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД.

Для любой реализации системы управления распределенными транзакциями используется модель обработки распределенных транзакций X/Open (X/Open DTP).

В данноv проекте распределенная транзакция опредеена для добавлеия документа в архив. Если было введено правильное кодовое слово компании – запись происходит успешно и изменяется информация о заказчике, на которого провели документ. Если кодовое слово неверное – транзакцивая завершается откатом.



Рисунок 21. Диаграмма последовательности

5 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ

Нагрузочное тестирование — подвид тестирования производительности, сбор показателей и определение производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе.

Были сформулированиы следующие требования:

1. Система должна поддерживать одновременную работу 125 пользователей (так как система предназначена для работы в некрупных городах);
2. Отклик системы при работе каждого пользователя не должен превышать 2 с. не зависимо от количества одновременно работающих пользователей (так как система должна обеспечивать возможность достаточно быстрой параллельной работы с БД);

Для проведения тестов был разработан тестовый сценарий.

Для проведения тестов был разработан следующий сценарий:

1) Пользователь заходит в систему

2) Пользователь просматривает список ГОСТов

3) Пользователь создает ГОСТ

4) Пользователь просматривает список документов

5) Пользователь создает документ

6) Пользователь помещает документ в архив

Результаты выполнения тестов:

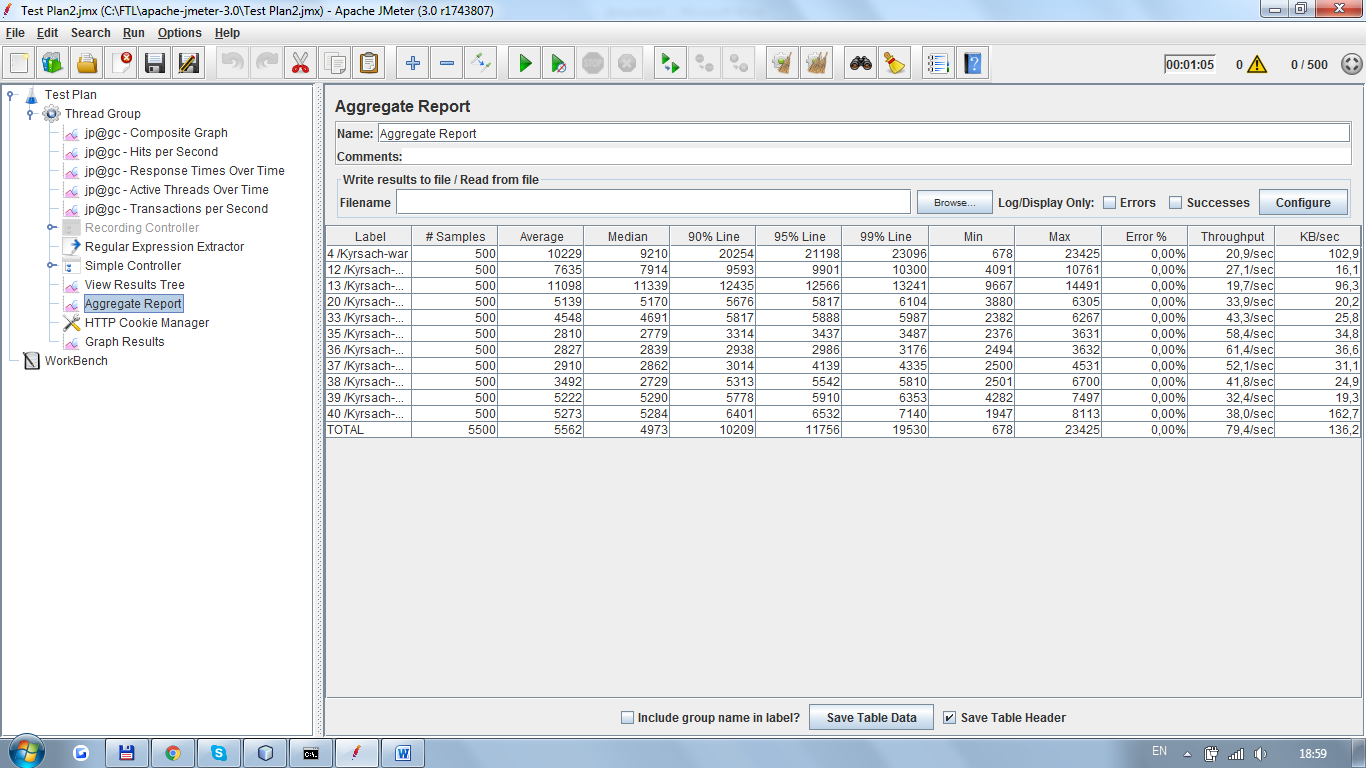


Рисунок 22. Агрегационный отчет

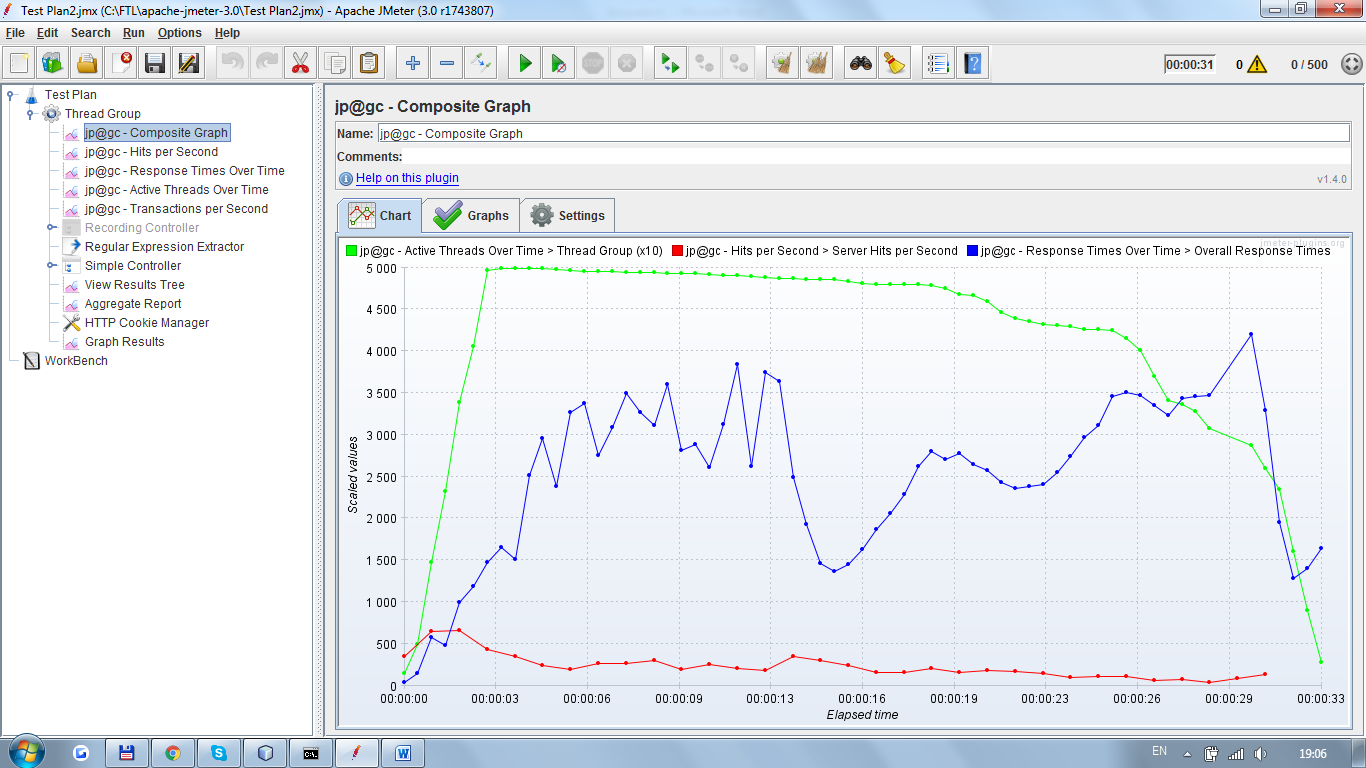


Рисунок 23. Пример результата выполнения теста

После проведения многочисленных тестов был проведен анализ полученных результатов, диаграмма которых представлена на рисунке 24.

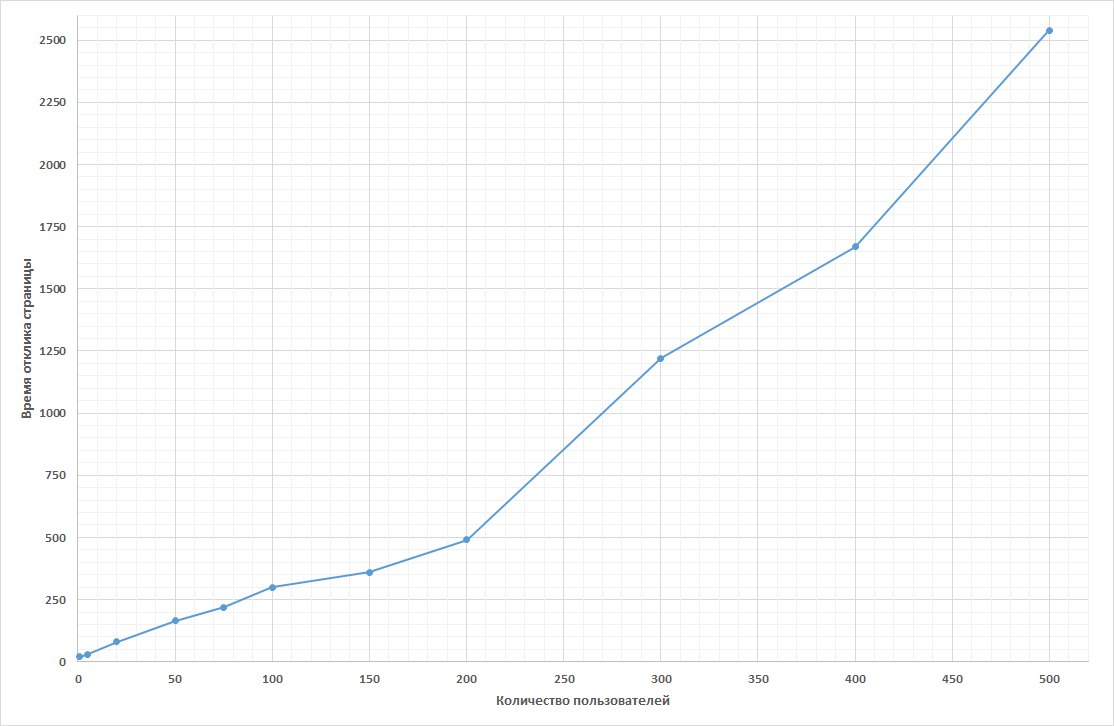


Рисунок 24. Диаграмма производительности системы

В результате тестирования было выявлено, что система почти полностью удовлетворяет заявленным требованиям, но для полноценной и безотказной работы данная система должна быть размещена на более мощном сервере, либо разнесена по нескольким серверам.

ЗАКЛЮЧЕНИЕ

Несомненно, тема делопроизводства в системе управления имеет сейчас большую актуальность. Эффективность управления предприятием зависит от того, на сколько разумно организованно управление документооборотом. Фактически, малоэффективное использование накопленной информации (или, еще, хуже ее утрата) может привести к потере всего бизнеса.

Осознавая значимость эффективного управления документооборотом на предприятии в ходе выполнения данной курсовой работы был реализован прототип программной системы «Документооборт малого предприятия».

В процессе работы были решены следующие задачи:

* + проведен анализ предметной области, функциональная декомпозиция системы;
  + создана структура базы данных, содержащая не менее 6 взаимосвязанных таблиц;
  + создана структура программной системы с разделением на бизнес-логику и логику веб-интерфейса;
  + реализованы бизнес-логика и веб-интерфейсс с использованием заданных технологий;
  + осуществеленно нагрузочное тестирование прототипа.

Данную программную систему можно использовать на любом предприятии, на котором осущевстляется документооборт.

Программная система облегчает процесс поиска и просмотра документов и ведения учета, обеспечивает целостность хранимой информации. Эффективность управления бизнес-процессами заметно повысится, так как все документы предприятия будут находится в единой базе и структурированы, что естественным образом приведет к минимизации риска утери документов и несанкционированного доступа к конфиденциальной информации. Имея необходимый доступ человеку просто нужно зайти в программную систему и найти необходимую информацию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Будилов, В.А. Интернет-программирование на Java / В.А.Будилов – СПб.: БХВ-Петербург, 2003
  2. Дейтел, Х.М. Технологии программирования на Java 2 / Х.М.Дейтел, П.Дж.Дейтел, С.И.Сантри – М.: ООО "Бином-Пресс", 2003
  3. Вершинин, М.М. Java 2 Enterprise Edition. Технологии проектирования и разработкич / М.М. Вершинин, Е.Б. Иванова – СПб.: БХВ-Петербург, 2003
  4. Генри, Д.М. JavaServer Faces: Библиотека профессионала / Дэвид М. Генри, Кей С. Хорстманн – М. : Вильямс, 2008. – 576 с. – ISBN 978-5-8459-1396-8, 978-0-13-173886-7

ПРИЛОЖЕНИЕ А  
Листинг компонентов системы

AbstractFacade.java

package DAO;

import java.util.List;

import javax.persistence.EntityManager;

public abstract class AbstractFacade<T> {

private Class<T> entityClass;

public AbstractFacade(Class<T> entityClass) {

this.entityClass = entityClass;

}

protected abstract EntityManager getEntityManager();

public void create(T entity) {

getEntityManager().persist(entity);

}

public void edit(T entity) {

getEntityManager().merge(entity);

}

public void remove(T entity) {

getEntityManager().remove(getEntityManager().merge(entity));

}

public T find(Object id) {

return getEntityManager().find(entityClass, id);

}

public List<T> findAll() {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

cq.select(cq.from(entityClass));

return getEntityManager().createQuery(cq).getResultList();

}

public List<T> findRange(int[] range) {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

cq.select(cq.from(entityClass));

javax.persistence.Query q = getEntityManager().createQuery(cq);

q.setMaxResults(range[1] - range[0] + 1);

q.setFirstResult(range[0]);

return q.getResultList();

}

public int count() {

javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();

javax.persistence.criteria.Root<T> rt = cq.from(entityClass);

cq.select(getEntityManager().getCriteriaBuilder().count(rt));

javax.persistence.Query q = getEntityManager().createQuery(cq);

return ((Long) q.getSingleResult()).intValue();

}

}

**ArchiveFacade.java**

package DAO;

import Model.Archive;

import Model.Company;

import Model.Customer;

import Model.Document;

import java.util.Calendar;

import java.util.List;

import javax.annotation.Resource;

import javax.ejb.EJB;

import javax.ejb.SessionContext;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

@Stateless

public class ArchiveFacade extends AbstractFacade<Archive> implements ArchiveFacadeLocal {

@PersistenceContext(unitName = "EnterpriseApplication3-ejbPU")

private EntityManager em;

@Resource

private SessionContext sc;

@EJB

DocumentFacadeLocal documentFacade;

@EJB

CompanyFacadeLocal companyFacade;

@EJB

CustomerFacadeLocal customerFacade;

@Override

protected EntityManager getEntityManager() {

return em;

}

@Override

public void addDoc(Document document, String code, Customer customer){

List<Document> list = documentFacade.findAll();

int lastId = list.size();

Company company = companyFacade.find((document.getIdDocument()));

if (code.equals(company.getCodename())){

Archive archive = new Archive();

archive.setDocument(document);

archive.setCustomer(customer);

archive.setCompany(company);

java.sql.Date date = new java.sql.Date(Calendar.getInstance().getTime().getTime());

archive.setDate(date);

customer.setName(customer.getName()+"|"+document.getName());

em.flush();

em.persist(customer);

em.persist(archive);

}

else {

sc.setRollbackOnly();

}

}

public ArchiveFacade() {

super(Archive.class);

}

}

**CompanyFacade**

package DAO;

import Model.Company;

import javax.ejb.Stateless;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

@Stateless

public class CompanyFacade extends AbstractFacade<Company> implements CompanyFacadeLocal {

@PersistenceContext(unitName = "EnterpriseApplication3-ejbPU2")

private EntityManager em;

@Override

protected EntityManager getEntityManager() {

return em;

}

public CompanyFacade() {

super(Company.class);

}

}

**ArchiveView.java**

package presentation;

import DAO.ArchiveFacadeLocal;

import DAO.DocumentFacadeLocal;

import java.io.Serializable;

import javax.ejb.EJB;

import javax.enterprise.context.SessionScoped;

import javax.inject.Named;

import Model.Archive;

import static Model.Archive\_.document;

import Model.Document;

import java.util.List;

@Named(value = "archiveView")

@SessionScoped

public class ArchiveView implements Serializable{

@EJB

ArchiveFacadeLocal archiveFacade;

@EJB

DocumentFacadeLocal documentFacade;

public ArchiveView(){

this.archive = new Archive();

this.codeName = new String();

}

private Archive archive;

private String codeName;

public String getCodeName() {

return codeName;

}

public void setCodeName(String codeName) {

this.codeName = codeName;

}

public Archive getArchive() {

return archive;

}

public void setArchive(Archive archive) {

this.archive = archive;

}

public List<Archive> getAllArhive(){

List<Archive> list = archiveFacade.findAll();

return list;

}

public void addToArchive(int idDoc){

Document document = documentFacade.find(idDoc);

archiveFacade.addDoc(document, codeName, archive.getCustomer());

}

}

**PersonalView.java**

package presentation;

import DAO.PersonalFacadeLocal;

import Model.Personal;

import java.io.Serializable;

import java.util.List;

import javax.ejb.EJB;

import javax.enterprise.context.SessionScoped;

import javax.inject.Named;

@Named(value = "personalView")

@SessionScoped

public class PersonalView implements Serializable{

@EJB

PersonalFacadeLocal personalFacade;

public PersonalView(){

this.personal = new Personal();

}

private Personal personal;

public Personal getPersonal() {

return personal;

}

public void setPersonal(Personal personal) {

this.personal = personal;

}

public List<Personal> getAllPersonal(){

List<Personal> list = personalFacade.findAll();

return list;

}

public String addPersonal(){

personalFacade.create(personal);

return "/faces/personal.xhtml";

}

public void deletePersonal(int id){

personal = personalFacade.find(id);

personalFacade.remove(this.personal);

}

public String editPersonal(int id){

this.personal = personalFacade.find(id);

return "/faces/editPersonal.xhtml";

}

public String edit(){

personalFacade.edit(this.personal);

return "faces/personal.xhtml";

}

}

**Company.java**

package Model;

import java.io.Serializable;

import java.util.List;

import java.util.Objects;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.Table;

@Entity

@Table(name="companyEntity")

public class Company implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer idCompany;

private String Name;

private String INN;

private String Codename;

@OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.REMOVE)

private List<Customer> customers;

@OneToMany(fetch=FetchType.LAZY, cascade = CascadeType.REMOVE)

private List<Archive> documents;

public Integer getIdCompany() {

return idCompany;

}

public void setIdCompany(Integer idCompany) {

this.idCompany = idCompany;

}

public String getName() {

return Name;

}

public void setName(String Name) {

this.Name = Name;

}

public String getINN() {

return INN;

}

public void setINN(String INN) {

this.INN = INN;

}

public String getCodename() {

return Codename;

}

public void setCodename(String Codename) {

this.Codename = Codename;

}

public List<Customer> getCustomers() {

return customers;

}

public void setCustomers(List<Customer> customers) {

this.customers = customers;

}

public List<Archive> getDocuments() {

return documents;

}

public void setDocuments(List<Archive> documents) {

this.documents = documents;

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null) {

return false;

}

if (getClass() != obj.getClass()) {

return false;

}

final Company other = (Company) obj;

if (!Objects.equals(this.idCompany, other.idCompany)) {

return false;

}

return true;

}

@Override

public String toString() {

return "Company{" + "idCompany=" + idCompany + '}';

}

}

**GOST.java**

package Model;

import java.io.Serializable;

import java.util.Objects;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="GOST")

public class GOST implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer idGost;

private String number;

private String name;

private String description;

public Integer getIdGost() {

return idGost;

}

public void setIdGost(Integer idGost) {

this.idGost = idGost;

}

public String getNumber() {

return number;

}

public void setNumber(String number) {

this.number = number;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null) {

return false;

}

if (getClass() != obj.getClass()) {

return false;

}

final GOST other = (GOST) obj;

if (!Objects.equals(this.idGost, other.idGost)) {

return false;

}

return true;

}

@Override

public String toString() {

return "GOST{" + "idGost=" + idGost + '}';

}

}

**CompanyConverter.java**

package Converter;

import DAO.CompanyFacadeLocal;

import Model.Company;

import Model.GOST;

import javax.annotation.ManagedBean;

import javax.ejb.EJB;

import javax.enterprise.context.RequestScoped;

import javax.faces.application.FacesMessage;

import javax.faces.component.UIComponent;

import javax.faces.context.FacesContext;

import javax.faces.convert.Converter;

import javax.faces.convert.ConverterException;

import javax.inject.Named;

@ManagedBean

@RequestScoped

@Named("companyConverter")

public class CompanyConverter implements Converter{

@EJB

CompanyFacadeLocal companyFacade;

@Override

public Object getAsObject(FacesContext context, UIComponent component, String submittedValue) {

if (submittedValue == null || submittedValue.isEmpty()) {

return null;

}

try {

return companyFacade.find(Integer.valueOf(submittedValue));

} catch (NumberFormatException e) {

throw new ConverterException(new FacesMessage(String.format("%s is not a valid bonus ID", submittedValue)), e);

}

}

@Override

public String getAsString(FacesContext context, UIComponent component, Object modelValue) {

if (modelValue == null) {

return "";

}

if (modelValue instanceof Company) {

return String.valueOf(((Company) modelValue).getIdCompany());

} else {

throw new ConverterException(new FacesMessage(String.format("%s is not a valid bonus", modelValue)));

}

}

}

**CustomerConverter.java**

package Converter;

import DAO.CustomerFacadeLocal;

import Model.Customer;

import javax.annotation.ManagedBean;

import javax.ejb.EJB;

import javax.enterprise.context.RequestScoped;

import javax.faces.application.FacesMessage;

import javax.faces.component.UIComponent;

import javax.faces.context.FacesContext;

import javax.faces.convert.Converter;

import javax.faces.convert.ConverterException;

import javax.inject.Named;

@ManagedBean

@RequestScoped

@Named("customerConverter")

public class CustomerConverter implements Converter{

@EJB

CustomerFacadeLocal customerFacade;

@Override

public Object getAsObject(FacesContext context, UIComponent component, String submittedValue) {

if (submittedValue == null || submittedValue.isEmpty()) {

return null;

}

try {

return customerFacade.find(Integer.valueOf(submittedValue));

} catch (NumberFormatException e) {

throw new ConverterException(new FacesMessage(String.format("%s is not a valid bonus ID", submittedValue)), e);

}

}

@Override

public String getAsString(FacesContext context, UIComponent component, Object modelValue) {

if (modelValue == null) {

return "";

}

if (modelValue instanceof Customer) {

return String.valueOf(((Customer) modelValue).getIdCustomer());

} else {

throw new ConverterException(new FacesMessage(String.format("%s is not a valid bonus", modelValue)));

}

}

}

**DocumentFacadeLocal.java**

package DAO;

import Model.Document;

import java.util.List;

import javax.ejb.Local;

@Local

public interface DocumentFacadeLocal {

void create(Document document);

void edit(Document document);

void remove(Document document);

Document find(Object id);

List<Document> findAll();

List<Document> findRange(int[] range);

int count();

}

Исходный код программы целиком Вы можете увидеть, перейдя по ссылке

https://github.com/Haragan/RPS