СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время Интернет становиться все более развитой средой для осуществления коммуникаций с потребителями. В тоже время, существенным является и тот факт, что Интернет становиться удобной и достаточно дешевой «торговой площадкой». Все большее количество фирм старается представить свою продукцию в on-line среде. При этом такое представление не ограничивается только лишь созданием промо-сайтов и размещением рекламных баннеров и статей в электронных журналах и на информационных порталах. С развитием интернет-среды развивается и само предложение. Теперь люди могут не только получать интересующую их информацию, но и совершать покупки. При этом с помощью интернет-магазинов можно приобретать товары совершенно разных категорий, как элементарные потребительские, так и высокотехнологичные.

Такое положение вещей обусловлено, во-первых, постоянным, стабильным ростом аудитории пользователей глобальной сети.

Основную часть аудитории сети Интернет составляют люди, работающие в офисах и проводящие большую часть времени за компьютером. Как правило, их образ жизни не позволяет им тратить большое количество времени на походы по off-line магазинам в поисках именно того, что им нужно. Возможность заказать интересующий продукт в Интернете является для них действительно выходом из ситуации. Интернет существенно ограничивает возможности представления товара, поэтому подходит в большинстве случаев только для повторной покупки, например при приобретении смартфонов .

Целью данной курсовой работы является:

1. Создание удобной WEB - системы, которая предоставит пользователю полную информацию об интересующей продукции.
2. Создание удобной системы, которая дает возможность пользователям заказать любую продукцию, не выходя из дома.

Для достижения цели необходимо решить следующие задачи:

− создать систему регистрации пользователей с жесткими требованиями к внесению контактной информации, по которой к нему можно будет обращаться.

− создать административный модуль управления сайтом, который позволит администратору получить неограниченный доступ к реляционной базе данных и предоставит возможность удалять, редактировать и добавлять записи

− изучить особенности организации и управления интернет-магазином

1 ОПИСАНИЕ ПРОЕКТА

1.1 Основные понятия и определения

Электронная коммерция − такая форма поставки продукции, при которой выбор и заказ товаров осуществляется через компьютерные сети, а расчеты между покупателем и поставщиком осуществляются с использованием электронных документов и/или средств платежа. При этом в качестве покупателей товаров (или услуг) могут выступать как частные лица, так и организации.

Глобальная сеть Internet сделала электронную коммерцию доступной для фирм любого масштаба. Электронная витрина в World Wide Web дает любой компании возможность привлекать клиентов со всего мира. Подобный on-line бизнес формирует новый канал для сбыта - "виртуальный", почти не требующий материальных вложений. Если информация, услуги или продукция (например, программное обеспечение) могут быть поставлены через Web, то весь процесс продажи (включая оплату) может происходить в on-line режиме.

Под определение электронной коммерции подпадают системы, ориентированные на Internet - "электронные магазины", речь о которых пойдет далее в курсовой работе. Несмотря на то, что WWW является технологической базой электронной коммерции, в ряде систем используются и другие коммуникационные возможности. Так, запросы к продавцу для уточнения параметров товара или для оформления заказа могут быть посланы и через электронную почту.[1]

Виртуальный магазин — это реализованное в сети Интернет представительство путем создания Web-сервера для продажи товаров и услуг другим пользователям сети Интернет. Виртуальный магазин называют также Интернет-магазином. К нему полностью подходит определение виртуального предприятия. Иначе говоря, виртуальный магазин — это сообщество территориально разобщенных сотрудников магазина (продавцов, кассиров) и покупателей, которые могут общаться и обмениваться информацией через электронные средства связи при полном (или минимальном) отсутствии личного прямого контакта.

1.2 Проблемы и преимущества интернет-магазина

Какие преимущества дает потребителю покупка товаров в интернет-магазине?

Во-первых, экономия времени. Интернет-магазин позволяет сделать покупку не выходя из офиса в любое время, а выбор и заказ товара займет у него несколько минут, если он точно знает, что хочет купить. Служба доставки интернет-магазина доставит выбранный товар в удобно время и место. Кроме этого выбор и оценка свойств товара происходит в интернет-магазине намного быстрее и удобнее чем в обычном магазине.

Во-вторых, неограниченный ассортимент и информативность. Ассортимент интернет-магазина ничем не ограничен (как, например, ассортимент обычного магазина ограничен площадью торгового павильона). А если предусмотрен поиск по параметрам, то Вы просто можете указать характеристики, которым должен соответствовать товар, и затем уже выбирать из списка моделей, удовлетворяющих вашему запросу. И еще одно очень важное замечание — ни один менеджер по продажам не в состоянии помнить столько информации о товаре, сколько может предоставить Вам интернет-магазин. Кроме того, интернет-магазин в состоянии выдать Вам такую информацию, как покупательский рейтинг, советы и отзывы о товаре, статьи о товаре, которые могут предопределить Ваш выбор.

В-третьих, экономия денег. Затраты на работу интернет-магазина, включая доставку, существенно ниже чем у обычного. В отличие от обычного магазина интернет-магазин способен обслужить несколько сотен клиентов одновременно. Хотя на практике такого не встретишь.

В свою очередь конечно потребитель сталкивается и с проблемами при покупке товаров online. Недостатки интернет-магазинов для покупателя:

− нельзя "пощупать", нельзя узнать больше, чем написано (пример: мебель, одежда)

− проблемы гарантии, сопровождения

− зачастую долгая доставка

1.3 Постановка задачи

Для успешной реализации курсового проекта необходимо, в первую очередь, выявить задачи, которым должна отвечать разрабатываемая система автоматизации рабочего места фармацевта.

Основными задачами являются следующие:

1) Разработка базы данных для MySQL Server 5.6, используя пакет HeidiSQL 8.1;

2) Система должна работать в рамках архитектуры клиент-сервер, используя протокол HTTP и браузер в качестве клиента;

3) Система должна предоставлять удобный и минималистичный интерфейс для конечного пользователя;

4) В клиентской части программы со стороны администратора должны быть предусмотрены следующие возможности:

– CRUD (добавление, просмотр, изменение, удаление) всех доступных данных;

– Просмотр заказов.

5) В клиентской части программы со стороны пользователя должны быть предусмотрены следующий возможности:

– Просмотр информации о товарах;

– Добавление товара в корзину

– Поиск информации

– Оформление заказа

Для графического описания модели автоматизации рабочего места фармацевта был использован стандарт IDEF0 (Integration Definition for Function Modeling), который подставляет собой методологию функционального моделирования, предназначенную для описания бизнес-процессов.

Функциональная модель на основе стандарта IDEF0 позволяет учесть все условия поставленной задачи и автоматизировать её. Исходя из принципов стандарта IDEF0, необходимо определить главную функцию проекта, затем осуществить её декомпозицию.

В данном проекте главная функция – «Процесс работы интернет-магазина». Входная информация: неавторизованные клиенты, авторизованные клиенты, поставляемая поставщиками продукция. Информация, полученная на входе, обрабатывается и преобразуется в выходные данные: доставленный товар, денежные средства, финансовые документы. Механизмами являются: персонал, заказчик, интерфейс по работе с интернет-магазином. Управляющими механизмами являются онлайн каталог товаров, налоговый кодекс, закон о защите прав потребителей.

Цель моделирования – показать процесс покупки товаров через интернет-магазин.

Описание процесса покупки товаров через интернет-магазин приводится с точки зрения потребителя и представляет собой полный процесс, начиная с регистрации и заканчивая получением и проверкой товаров.

Описание бизнес-процесса покупки товаров через интернет-магазин изображено на рисунке 2.1.

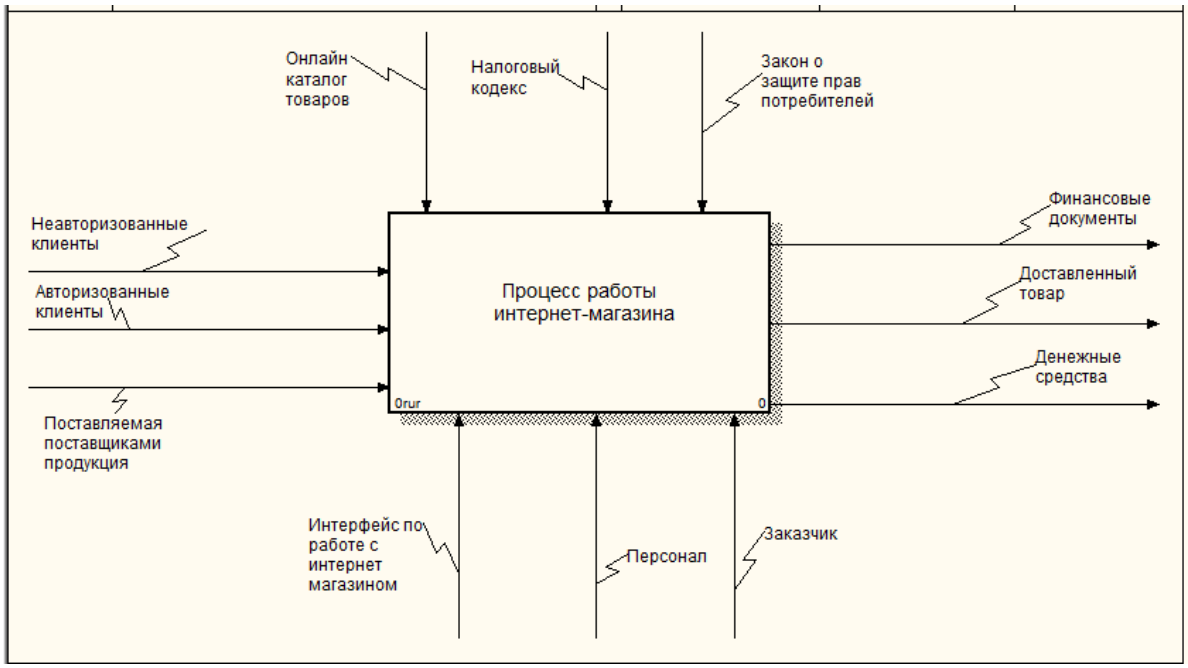

Рисунок 1.1 – Бизнес-процесс покупки товаров через интернет-магазин

2 ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

2.1 Определение требований к системе

В связи с поставленными задачами, необходимо придерживаться следующих требований к системе:

1) Приложение должно осуществлять свою работу в рамках архитектуры клиент-сервер. В качестве языка для реализации выбран объектно-ориентированный язык Java[2]. В данной работе будут реализованы: собственная иерархия классов, расширение базовых классов предоставляемых SDK, реализация паттерна проектирования Proxy[4], сокрытие данных (инкапсуляция), перегрузка методов, переопределение методов, параметризированные классы, сериализация и десериализация, интерфейсы и абстрактные классы, передача параметров по ссылке и по значению, статические методы, обработка исключительных ситуаций[2].

2) Бизнес-логика системы реализована на сервере данного приложения. Сервер должен предусматривать параллельную обработку запросов.

3) СУБД – MySQL 5.6. Для разработки информационной модели системы должен быть использован пакет моделирования ERWin. Доступ к данным в СУБД осуществляется через драйвер, предоставляемый производителем СУБД.

3) База данных приведена к 3-ей нормальной форме.

4) База данных должна генерироваться sql-скриптом.

5) Функционал серверной части не менее 6 Use Case’ов (вариантов использования). [3]

6) Паттерн реализации: «MVC»[4]

7) Транспортный уровень. Взаимодействие между серверной и клиентскими частями должно осуществляется с использованием протокола HTTP.

2.2 Обоснование решений по использованию технических и программных средств реализации

В данном разделе рассматриваются используемые в курсовой работе программные и технические средства с точки зрения эффективности и удобства их использования.

В качестве средства описания и моделирования бизнес-процесса автоматизации работы фармацевта используется AllFusion Process Modeler 7

(ранее BPwin), который является эффективным инструментом для анализа, документирования и оптимизации бизнес-процессов. AllFusion Process Modeler 7 можно использовать для графического представления бизнес-процессов. Графически представленная схема выполнения работ, обмена информацией, документооборота визуализирует модель бизнес-процесса. Графическое изложение этой информации позволяет перевести задачи управления организацией из области сложного ремесла в сферу инженерных технологий.

AllFusion Process Modeler 7 (BPwin) помогает четко документировать важные аспекты любых бизнес-процессов: действия, которые необходимо предпринять, способы их осуществления и контроля, требующиеся для этого ресурсы, а также визуализировать получаемые от этих действий результаты. AllFusion Process Modeler 7 повышает бизнес-эффективность ИТ-решений, позволяя аналитикам и проектировщикам моделей соотносить корпоративные инициативы и задачи с бизнес-требованиями и процессами информационной архитектуры и проектирования приложений. Таким образом, формируется целостная картина деятельности: от потоков работ в небольших подразделениях до сложных организационных функций.

AllFusion Process Modeler 7 (BPwin) эффективен в проектах, связанных с описанием действующих баз предприятий, реорганизацией бизнес-процессов, внедрением корпоративной информационной системы. Продукт позволяет оптимизировать деятельность предприятия и проверить ее на соответствие стандартам ISO 9000 , снизить издержки, исключить ненужные операции и повысить эффективность. В основу продукта заложены общепризнанные методологии моделирования, например, методология IDEF0 рекомендована к использованию Госстандартом РБ и является федеральным стандартом США. Простота и наглядность моделей Process Modeler упрощает взаимопонимание между всеми участниками процессов.

Для описания информационной модели системы автоматизации риэлтерской компании используется ERwin Data Modeler , который представляет CASE-средство для проектирования и документирования баз данных, которое позволяет создавать, документировать и сопровождать базы данных, хранилища и витрины данных.

AllFusion ERwin Data Modeler (ERwin) предназначен для разработки и использования базы данных, для администраторов баз данных, системных аналитиков, проектировщиков баз данных, разработчиков, руководителей проектов.

AllFusion ERwin Data Modeler (ERwin) позволяет наглядно отображать сложные структуры данных. Удобная в использовании графическая среда AllFusion ERwin Data Modeler упрощает разработку базы данных и автоматизирует множество трудоемких задач.

В качестве целевой СУБД в данном курсовом проекте взят MySQL версии 5.6. MySQL — свободная реляционная система управления базами данных. MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты.

2.3 Спецификация вариантов использования системы «Интернет-магазин»

Диаграмма вариантов использования показывает, какая функциональность должна быть реализована в системе, основные функции, которые должны быть включены в систему (use case), их окружение (actors) и взаимодействие функций с окружением.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования. При этом актером или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) 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[3].

В данной работе будут реализованы следующие варианты использования:

1) Войти в систему – позволяет не вошедшим в систему пользователями авторизоваться.
2) Регистрация – позволяет пользователю зарегистрироваться в системе.
3) Поиск товара – позволяет пользователям искать товар.
4) Просмотр корзины – позволяет пользователю просматривать и управлять своей корзиной.
5) Оформить заказ – позволяет пользователю оформить заказ.
6) Просмотр каталогов – позволяет пользователю просмотреть товар по каталогам.

7) Управление системой – позволяет администратору управлять системой.

Диаграмма вариантов использования данного курсового проекта представлена на рисунке 2.1.

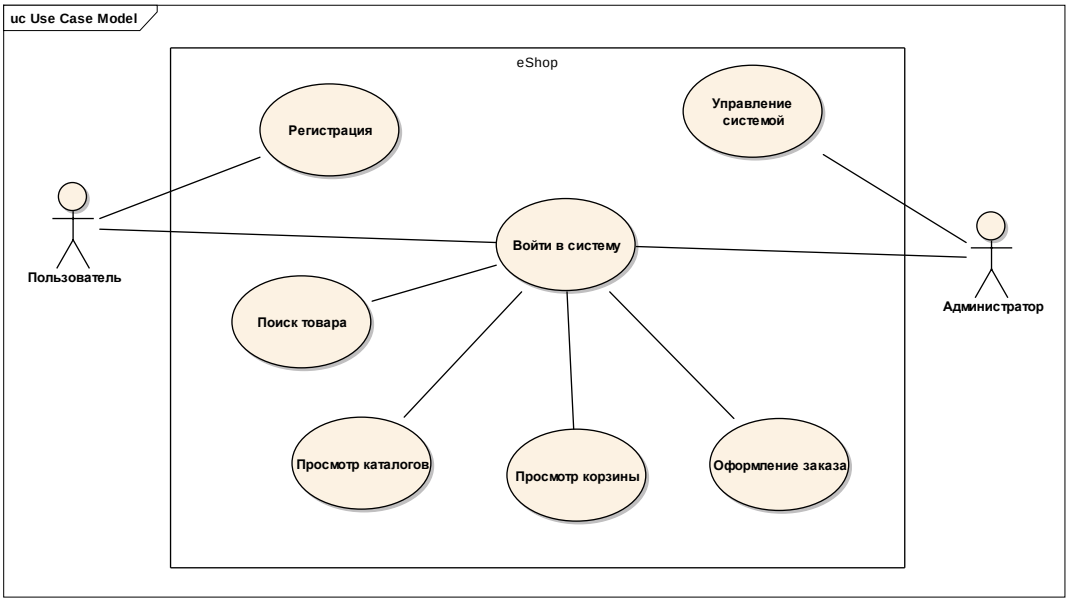

Рисунок 2.1 – Диаграмма вариантов использования.

3 АРХИТЕКТУРНЫЙ ШАБЛОНПРОЕКТИРОВАНИЯ MVC

Паттерн проектирования, или шаблон проектирования представляет собой модель взаимодействия классов для решения той или иной задачи. Существует огромное количество разнообразных паттернов, с которыми так или иначе сталкивался каждый. Примером простейшего паттерна может быть итератор, используемый для перемещения по контейнеру или коллекции.

Во всех современных проектах можно встретить тот или иной паттерн проектирования, причем не обязательно только один на программу. Одной из проблем, вытекающих из использования паттернов является то, что не все из использованных шаблонов обозначены разработчиками ПО. Проект, в котором явно указаны и разобраны все использованные паттерны проще для понимания, более удобен и легко управляем.[4]

Информация о паттерне, используемом в проекте сразу дает четкое представление о том, как работает система. Также паттерны позволяют упростить управление проектом благодаря тому, что обобщение удачных решений каких-либо задач в паттерны позволяет их использовать в последующих проектах, что существенно ускоряет ход разработки.

При написании данной работы были использованы паттерны – «MVC», «Command», «Factory», «Singleton», «DAO». Диаграммы классов данных паттернов приведены на рисунках 3.1 – 3.5.

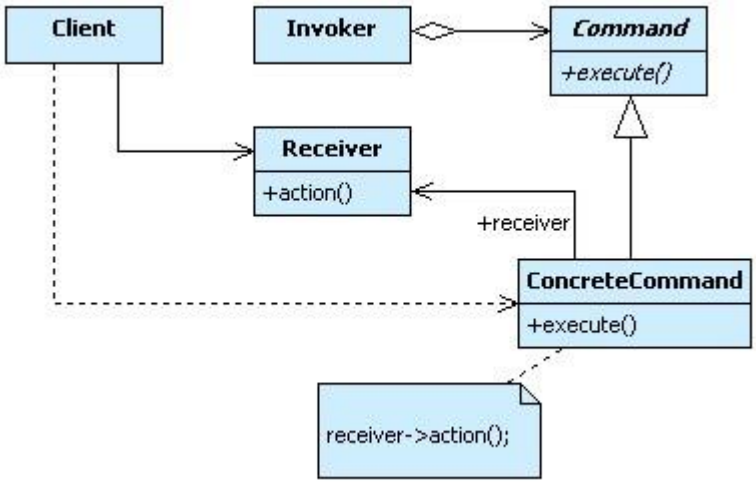

Рисунок 3.1 – Диаграмма классов паттерна «Command»

Рисунок 3.2 – Диаграмма классов паттерна «Singletone»

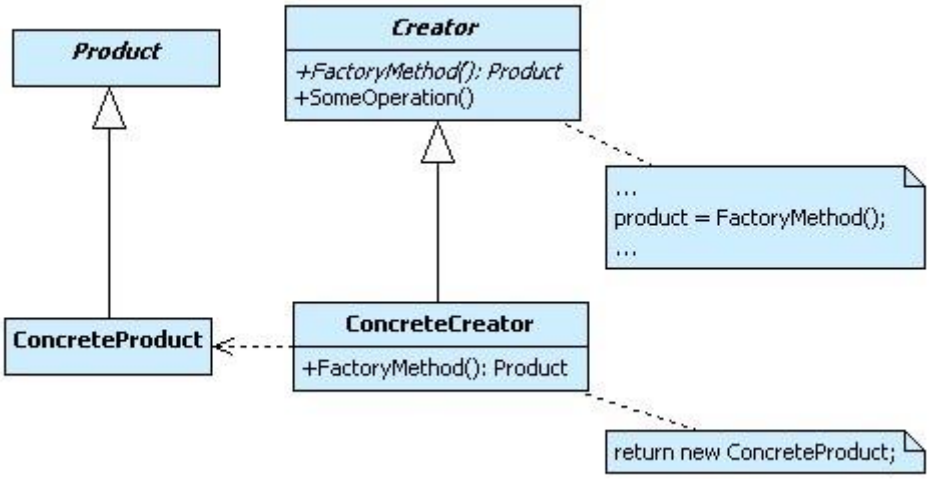

Рисунок 3.3 – Диаграмма классов паттерна «Factory»

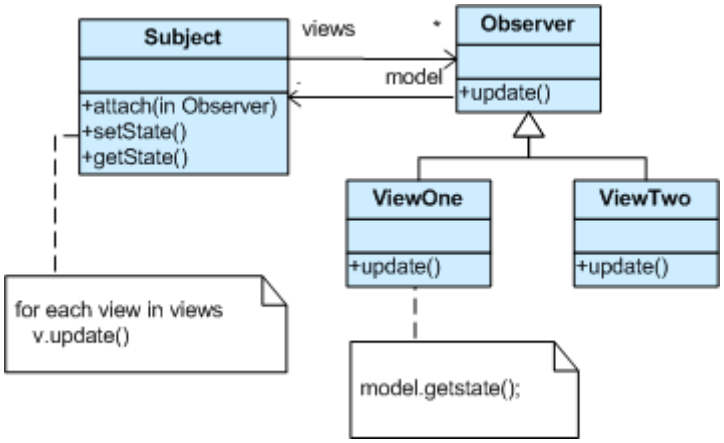

Рисунок 3.4 – Диаграмма классов паттерна «MVC»

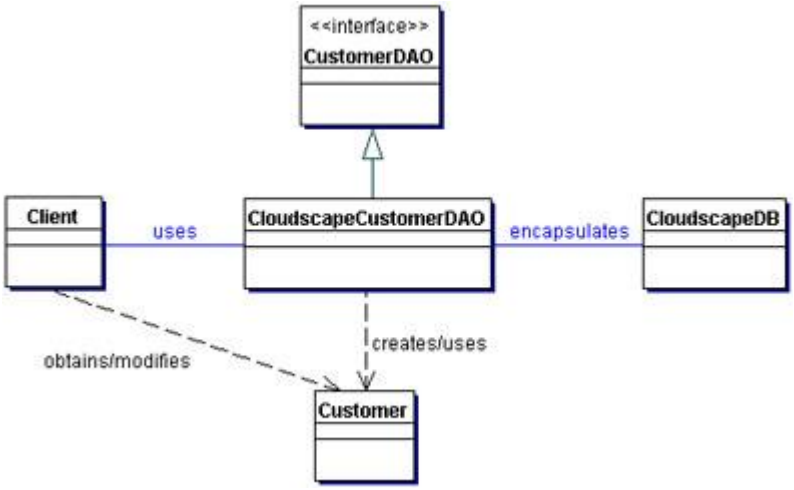

Рисунок 3.5 – Диаграмма классов паттерна «DAO»

3.1 Модели представления системы

Диаграмма в UML – это графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями). Основная цель диаграмм – визуализация разрабатываемой системы с разных точек зрения.

3.1.2 Диаграмма последовательности

Диаграмма последовательности — это упорядоченная по времени диаграмма взаимодействия, читать ее следует сверху вниз. У каждого варианта использования имеется большое количество альтернативных потоков. Каждая диаграмма последовательности описывает один из потоков варианта использования. Участвующие в потоке объекты нарисованы в прямоугольниках в верхней части диаграммы. У каждого объекта имеется линия жизни (lifeline), изображаемая в виде вертикальной штриховой линии под объектом. Сообщения, соответствующие коммуникациям между объектами, рисуют между линиями жизни объектов. Сообщение показывает, что один объект вызывает функцию другого.

Диаграмма последовательности авторизации представлена в приложении В.

3.1.3 Диаграммы состояния заказа

Диаграммы состояний являются хорошо известным средством описания поведения систем. Они определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате влияния некоторых событий.

Состояние (statement) − это ситуация, когда моделируемый объект выполняет какое либо условие, проводит операцию или ожидает события.

Диаграмма состояния данного проекта описывает состояние отчета и этапы, через которые он проходит.

Диаграмма состояния для данного проекта представлена в приложении Г.

4 ШАБЛОН ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ

4.1 Информационная модель системы и ее описание

В данной курсовой работе для моделирования информационной системы было использовано техническое средство ERWin DataModeler.

Основные составляющие части визуального представления ERwin – это сущности, атрибуты и связи. Каждая сущность является множеством сходных объектов, называемых экземплярами. Каждый экземпляр индивидуален и должен отличаться от всех остальных экземпляров. Атрибут выражает некоторое свойство объекта. В базе данных сущности соответствует таблица, экземпляру сущности − строка (запись) в таблице, а атрибуту – колонка (поле) таблицы. Построение информационной модели данных предполагает определение сущностей и атрибутов.

В ERwin существуют два уровня представления — логический и физический.

Логический уровень означает непосредственное отображение фактов из реальной жизни. На нем данные представляются так, как они выглядят в реальном мире. Объекты модели, представляемые на логическом уровне, называются сущностями и атрибутами. Логическая модель данных не связана с конкретной СУБД. Логический уровень данной информационной модели представлен на рисунке 4.1.

Физическая же модель данных зависит от конкретной СУБД. Значит, одной и той же логической модели могут соответствовать несколько различных физических моделей. Физический уровень модели представлен на рисунке 4.2.

Рассмотрим подробнее каждую сущность данной модели, а также связи между ними.

Сущность «Пользователь» используется для хранения информации о пользователях системы. Ключевым полем является Код пользователя. Неключевые поля: Логин, Пароль, Права, Статус, Номер телефона. Связана с сущностью «Заказ» связью один ко многим.

Сущность «Категория» используется для хранения информации о категориях товаров. Ключевым полем является Код категории. Неключевые поля: Название категории, Путь к изображению. Связана с сущность «Товар» связью один ко многим.

Сущность «Товар» используется для хранения информации о товарах в наличии. Ключевым полем является Код товара. Неключевые поля: Цена, Наименование, Описание, Путь к изображению. Связана с сущностью

«Категория» связью многие к одному и с сущностью «Заказ» связью один ко многим.

Сущность «Заказ» используется для хранения информации о заказах. Ключевым полем является Код заказа. Неключевые поля: Дата, Статус подтверждения, Статус доставки, Количество. Связана с сущность «Пользователь» и «Товар» связью многие к одному.

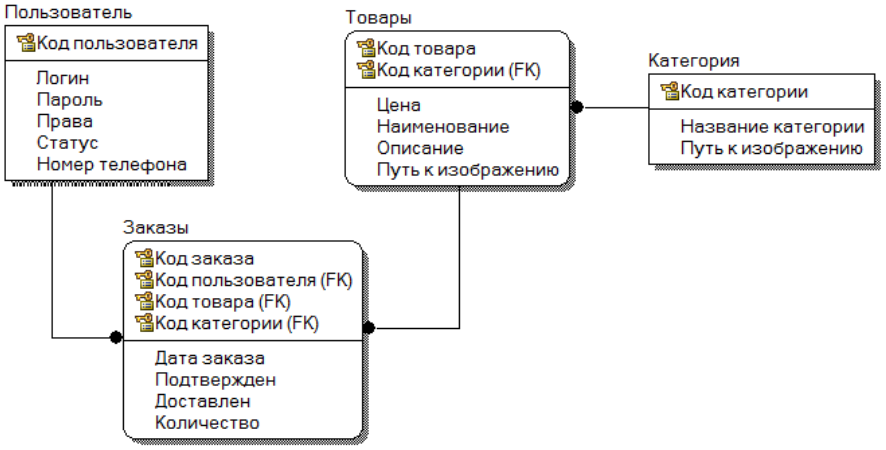

Рисунок 4.1 – Логический уровень информационной модели

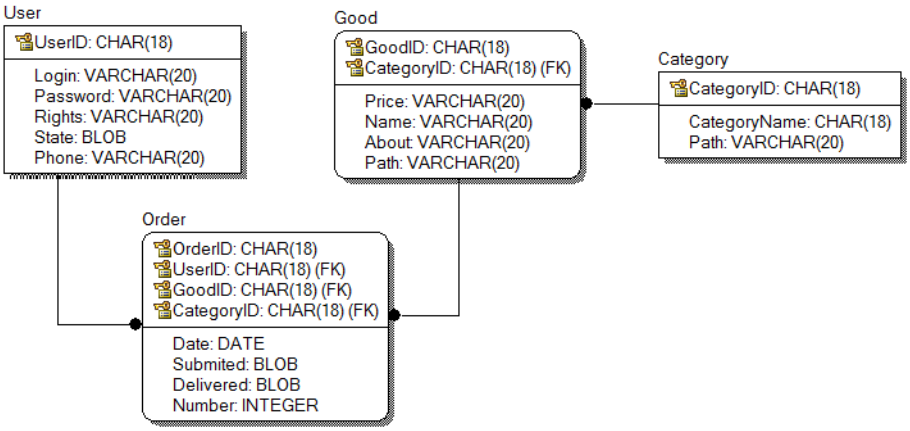

Рисунок 4.2 – Физический уровень информационной модели

4.3 Алгоритм авторизации

Данный алгоритм описывает процесс авторизации пользователя в системе.

```
public String execute(HttpServletRequest request) throws ServiceException {
        String page = null;

        if(Validator.validateLoginForm(request)) {
```

```
            try {
                User user = userDAO.readUser(request.getParameter("login"));
                request.getSession(true).setAttribute("user", user);
                page = JspConfig.getProperty(JspConfig.INDEX);
            } catch (DAOException e) {
                LOGGER.error(e);
                throw new ServiceException(e);            }
        } else {
            page = JspConfig.getProperty(JspConfig.LOGIN);
        }
       return page;
    }
```

1) Валидатор проверяет запрос на соответствие требованиям и наличию пользователя в БД. Если запрос проходит проверку, то выполняется шаг 2, иначе шаг 5.
2) Информация о пользователе читается из БД.
3) Информация о пользователе записывается в текущую сессию.
4) Формируется новое представление, затем шаг 6.
5) Формируется представление с сообщением об ошибке
6) Возвращается представление

Данный алгоритм представлен на рисунке Б.1 в приложении Б.

4.4 Алгоритм фильтрации доступа

Данный алгоритм описывает процесс фильтрации доступа пользователей к системе.

```
     public void doFilter(ServletRequest req, ServletResponse resp,
FilterChain chain) throws ServletException, IOException {

  HttpSession session = ((HttpServletRequest) req).getSession();
  User user = (User) session.getAttribute("user");

  if(user==null || user.getRights()!= "ROLE_ADMIN") {
        ((HttpServletResponse) resp).sendRedirect(((HttpServletRequest)
  req).getContextPath()+"/jsp/error/access-denied.jsp");
  }

  chain.doFilter(req, resp);
}
```

1) Получение текущей сессии из запроса
2) Получение атрибута «пользователь» из сессии
3) Если атрибут пуст или не обладает доступом администратора, то шаг 4, иначе шаг 5
4) Перенаправление запроса на страницу с сообщение об ограниченном доступе
5) Переход к следующему фильтру

Данный алгоритм представлен на рисунке Б.2 приложения Б.

5 ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

При запуске пользователь увидит следующую страницу:

Рисунок 5.1 – Главная страница приложения

При входе в систему пользователь видит следующею страницу:

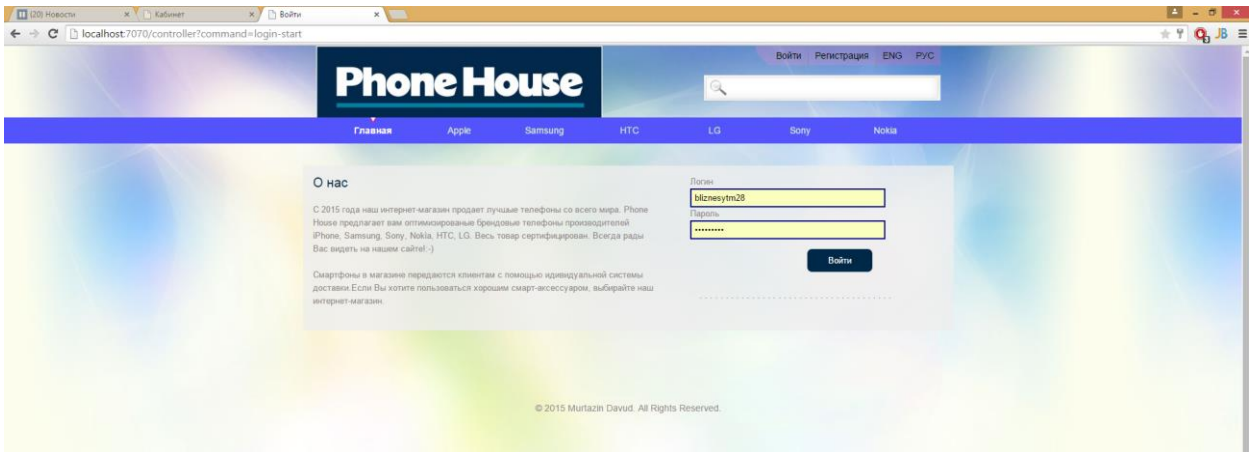

Рисунок 5.2. – Страница входа в систему

При регистрации пользователь попадает на страницу, изображенную на рисунке 5.3. Все поля для ввода валидируются.

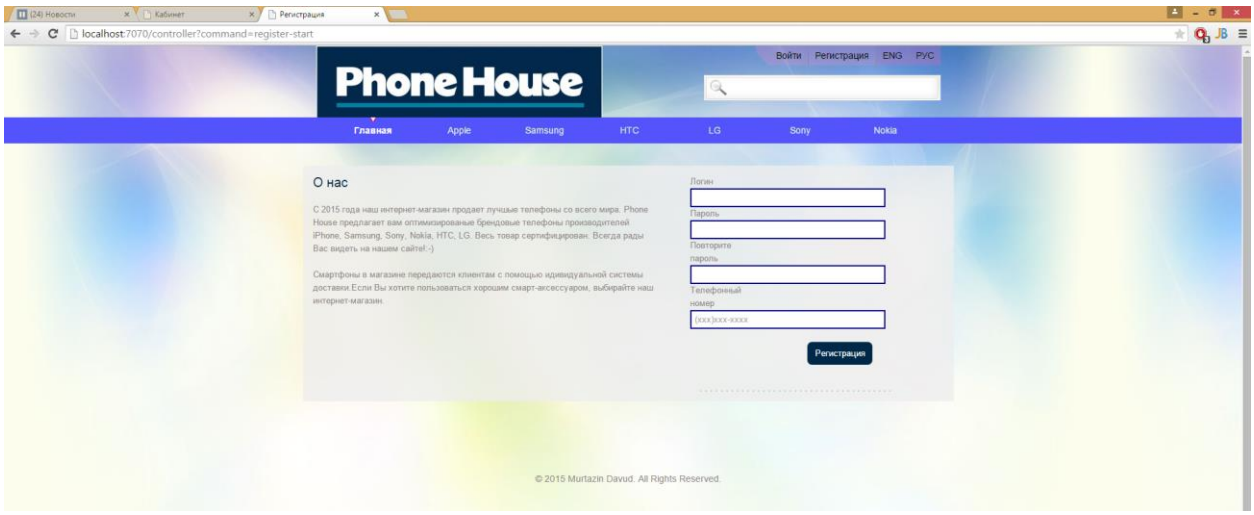

Рисунок 5.3 – Страница регистрации

При редактировании информации администратор видит страницу, изображенную на рисунке 5.4 (для каждой сущности своя страница)

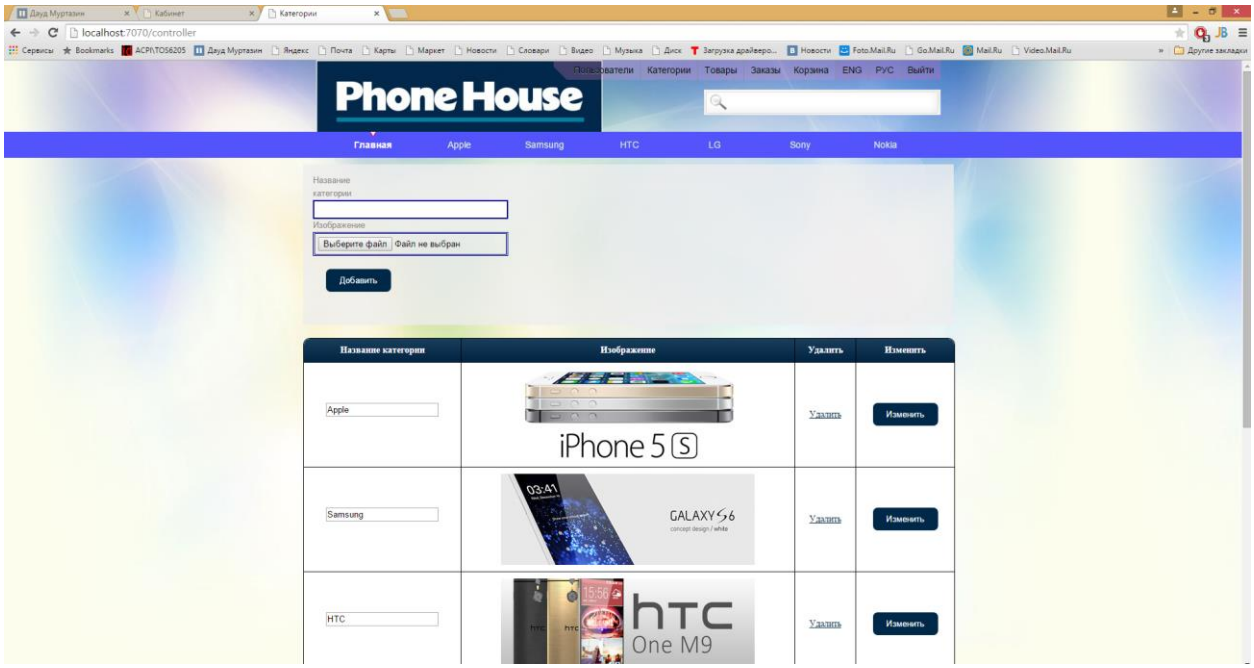

Рисунок 5.4 – Страница редактирования информации

При просмотре корзины пользователь увидит следующую страницу:

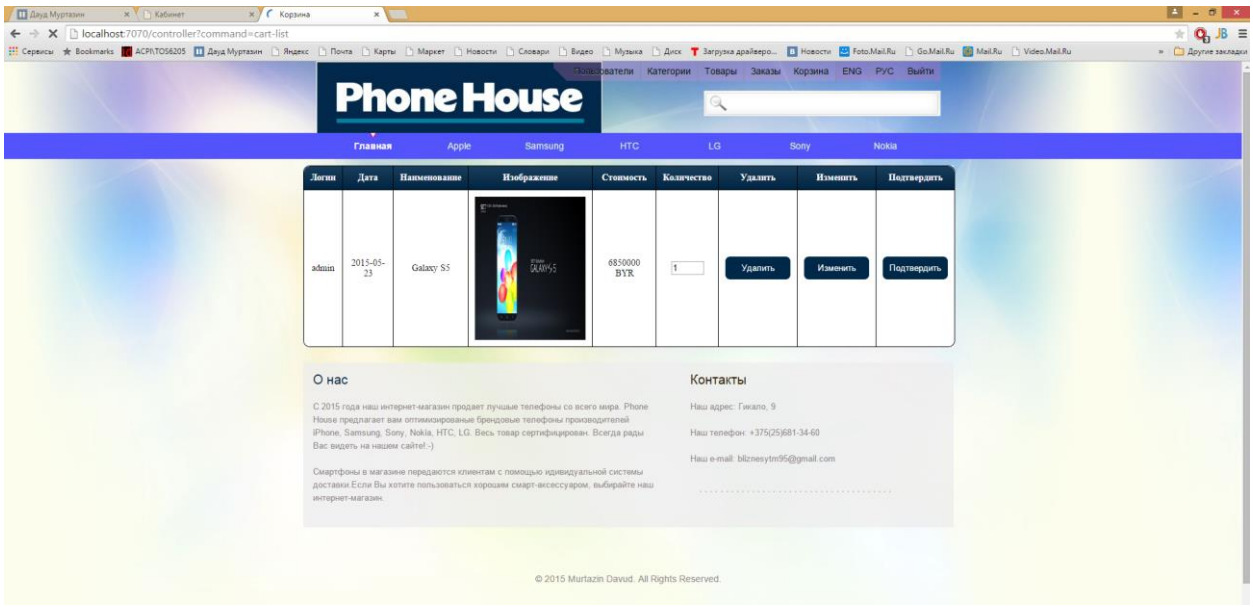

Рисунок 5.5 – Корзина пользователя

В качестве результатов поиска и просмотра товара по категориям пользователю открывается следующая страница:

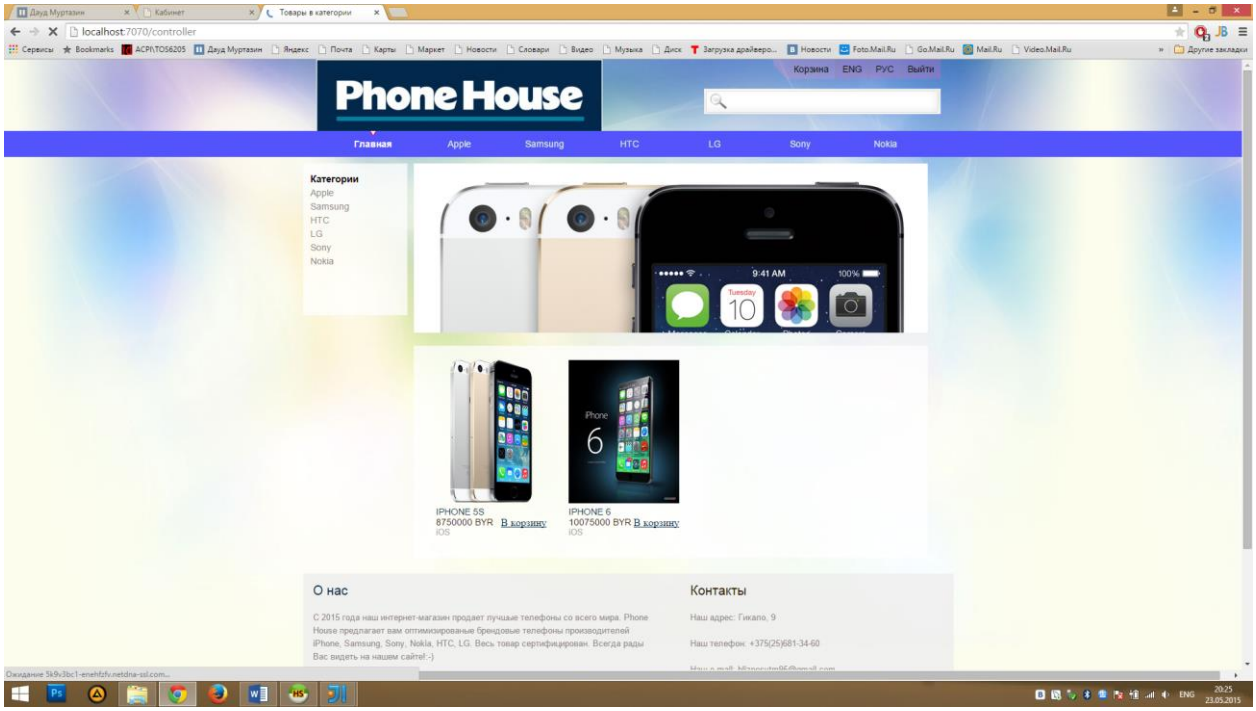

Рисунок 5.6 – Страница с результатами поиска

ЗАКЛЮЧЕНИЕ

Итогом данной работы является программный продукт, который представляет собой упрощенную систему по покупке товаров через интернет-магазин.

Оценив работу программы, можно сделать выводы:

– в программе предусмотрена защита от несанкционированного доступа к данным, за счет введения авторизации в приложение;

– программа выполнена в архитектуре клиент-сервер;

– предусмотрена работа сервера одновременно с несколькими пользователями;

– в программе предусмотрен удобный и понятный интерфейс;

− решены все поставленные задачи.

Разработанное приложение может быть улучшено путем усовершенствования программного продукта за счет улучшения интерфейса, а также введением дополнительных функций, которые обеспечат более легкое пользование данным приложением.

Для этого нужно уделить большое внимание разработке информационной системы. Построить и выбрать наиболее эффективную модель, спроектировать сайт магазина, понятный всем без исключения пользователям, сделать его привлекательным для покупателей, разработав дизайн.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Разработка интернет-магазина [Электронный ресурс]. – Электронные данные. – Режим доступа: http://bibliofond.ru/view.aspx?id=551931

[2] Герберт Ш. Java. Полное руководство / Ш. Герберт. – М.: Вильямс, 2012.

[3] Буч, Г. Язык UML. Руководство пользователя. / Гради Буч, Джеймс Рамбо, Ивар Якобсон

[4] Фримен, Эр. Паттерны проектирования / Эр. Фримен, Эл. Фримен. – СПб.: Питер, 2011. – 200 с.

[5] Бек, К. Экстремальное программирование: разработка через тестирование. Библиотека программиста / К. Бек. – СПб. : Питер, 2003.

[6] Wikipedia [Электронный ресурс] – электронная энциклопедия. – Электронные данные. – Режим доступа: http://www.wikipedia.org

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

```
/**
 * Servlet class
 */
@WebServlet(name="ShopServlet", urlPatterns = "/controller")
public class ShopServlet extends HttpServlet {
    public static Logger LOGGER = Logger.getLogger(ShopServlet.class);

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Process requests from users
     *
     * @param request           request to process
     * @param response          response
     * @throws ServletException in case of exception
     * @throws IOException      in case of exception
     */
    private void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        String page;

        page = JspConfig.getProperty(JspConfig.INDEX);

        try {
            String commandName = request.getParameter("command");
            ICommand command = null;

            if(!ServletFileUpload.isMultipartContent(request)) {
                String name = request.getParameter("command");
                command = CommandUtil.getCommand(name);
            } else {
                command = CommandMultipartUtil.getCommand(request);
            }

            page = command.execute(request);
            ServletUtil.setCategory(request);
        } catch (ServiceException e) {
            e.printStackTrace();
            page = JspConfig.getProperty(JspConfig.ERROR);
        }

        request.getRequestDispatcher(page).forward(request, response);
    }

    @Override
    public void init(ServletConfig sce) throws ServletException {
        super.init();
        Properties logProperties = new Properties();
```

```
        try {

System.out.println(sce.getServletContext().getRealPath("log4j.properties"));
            logProperties.load(new
FileInputStream(sce.getServletContext().getRealPath("log4j.properties")));
            PropertyConfigurator.configure(logProperties);
            LOGGER.info("Logging initialized");
        } catch (IOException e) {
            LOGGER.error(e);
        }

        ConnectionPool.getInstance();
    }

    @Override
    public void destroy() {
        ConnectionPool.getInstance().releasePool();
    }
}

/**
 * Connection Pool (CP)
 */
public class ConnectionPool {
    public static final Logger LOGGER =
Logger.getLogger(ConnectionPool.class);

    /**
     * Part of singletone
     */
    private static ConnectionPool instance;
    private static ReentrantLock lock = new ReentrantLock();

    private BlockingQueue<Connection> connections;
    private boolean flag;

    /**
     * Part of singletone
     */
    private ConnectionPool() {
        String url = DBConfig.getProperty(DBConfig.URL);
        String user = DBConfig.getProperty(DBConfig.USER);
        String password = DBConfig.getProperty(DBConfig.PASSWORD);
        Integer poolSize =
Integer.parseInt(DBConfig.getProperty(DBConfig.POOL_SIZE));

        try {
            DriverManager.registerDriver(new Driver());
            connections = new ArrayBlockingQueue<>(poolSize);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }

        for(int i=0; i<poolSize; i++) {
            try {
                Connection connection = DriverManager.getConnection(url,
user, password);
                if(!connections.offer(connection)) {
                    throw new RuntimeException("Can't initialize pool
properly");
                }
            } catch (SQLException e) {
                LOGGER.error(e);
```

```
                throw new RuntimeException(e);
            }
        }

        flag = true;
    }

    /**
     * Part of singletone
     * @return          CP instance
     */
    public static ConnectionPool getInstance() {
        if(instance==null) {
            try {
                lock.lock();

                instance = new ConnectionPool();
            } finally {
                lock.unlock();
            }
        }

        return instance;
    }

    /**
     * Get connection from pool
     * @return              sql connection
     * @throws DBException
     */
    public Connection getConnection() throws DBException {
        if(flag) {
            try {
                Connection connection = connections.take();

                return connection;
            } catch (InterruptedException e) {
                LOGGER.error(e);
                throw new RuntimeException(e);
            }
        } else {
            LOGGER.error("Pool is not working");
            throw new DBException("Pool is not working");
        }
    }

     /**
     * Return connection to pool
     * @param connection    sql connection
     */
    public void releaseConnection(Connection connection) {
        try {
            if(connection != null && !connection.isClosed()) {
                if(!connections.offer(connection)) {
                    LOGGER.error("Error returning connection to pool");
                    throw new RuntimeException("Error returning connection to
pool");
                }
            } else {
                LOGGER.error("Error returning connection to pool");
                throw new RuntimeException("Error returning connection to
pool");
            }
```

```
        } catch (SQLException e) {
            LOGGER.error(e);
            throw new RuntimeException(e);
        }
    }

    /**
     * Releases pool
     */
    public void releasePool() {
        Integer poolSize =
Integer.parseInt(DBConfig.getProperty(DBConfig.POOL_SIZE));

        while(!connections.isEmpty()) {
            try {
                Connection connection = connections.take();

                connection.close();
            } catch (InterruptedException | SQLException e) {
                LOGGER.error(e);
            }
        }
    }
}

/**
 * Command interface
 */
public interface ICommand {
    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    String execute(HttpServletRequest request) throws ServiceException;
}
```

```
/**

 * Enum of commands to execute
 */
public enum CommandEnum {
    NO_COMMAND("void"),
    LANGUAGE("language"),
    REGISTER("register"),
    REGISTER_START("register-start"),
    LOGIN("login"),
    LOGIN_START("login-start"),
    LOGOUT("logout"),
    USER_START("user-start"),
    USER_BANN("user-bann"),
    USER_RIGHTS("user-rights"),
    USER_DELETE("user-delete"),
    CATEGORY_START("category-start"),
    CATEGORY_ADD("category-add"),
    CATEGORY_EDIT("category-edit"),
    CATEGORY_DELETE("category-delete"),
    GOODS_START("good-start"),
    GOODS_ADD("goods-add"),
    GOODS_DELETE("goods-delete"),
    GOODS_EDIT_START("goods-edit-start"),
    GOODS_EDIT("goods-edit"),
    ORDER_START("order-start"),
    ORDER_DECLINE("order-decline"),
    ORDER_ACCEPT("order-accept"),
    ORDER_FILTER("order-filter"),
    GOODS_CATEGORY_LIST("goods-list-command"),
    CART_LIST("cart-list"),
    CART_SUBMIT("cart-submit"),
    CART_EDIT("cart-edit"),
    CART_DELETE("cart-delete"),
    CART_ADD("cart-add"),
    CART_FIND("cart-find");

    private String type;

    private CommandEnum(String type) {
        this.type = type;
    }

    /**
     * Get enum type according to string pType
     * @param pType     string to get enum type
     * @return          enum type
     */
    static public CommandEnum getType(String pType) {
        for (CommandEnum type: CommandEnum.values()) {
            if (type.getType().equals(pType)) {
                return type;
            }
        }

        return NO_COMMAND;
    }

    public String getType() {
        return type;
    }
}
/**
 * Change Language command
```

```
 */
public class LanguageCommand implements ICommand {
    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        request.getSession(true).setAttribute("lang",
request.getParameter("loc"));

        return JspConfig.getProperty(JspConfig.INDEX);
    }
}

/**
 * No Command class
 */
public class NoCommand implements ICommand {
    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        return JspConfig.getProperty(JspConfig.INDEX);
    }
}

/**
 * Process login command
 */
public class LoginCommand implements ICommand {
    public static final Logger LOGGER = Logger.getLogger(LoginCommand.class);

    private IUserDAO userDAO = new UserDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        if(Validator.validateLoginForm(request)) {
            try {
                User user = userDAO.readUser(request.getParameter("login"));
                request.getSession(true).setAttribute("user", user);
                page = JspConfig.getProperty(JspConfig.INDEX);
} catch (DAOException e) {
                LOGGER.error(e);
                throw new ServiceException(e);
            }
```

```
        } else {
            page = JspConfig.getProperty(JspConfig.LOGIN);
        }

        return page;
    }
}

/**
 * Open login page command
 */
public class LoginStartCommand implements ICommand {
    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        return JspConfig.getProperty(JspConfig.LOGIN);
    }
}

/**
 * Logout command
 */
public class LogoutCommand implements ICommand  {
    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        request.getSession().setAttribute("user", null);

        return new NoCommand().execute(request);
    }
}

/**
 * Register new user command
 */
public class RegisterCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(RegisterCommand.class);
    public static final String USER_RIGHTS = "ROLE_USER";

    private IUserDAO userDAO = new UserDAO();
```

```
/**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        if(Validator.validateRegisterForm(request)) {
            String login = request.getParameter("login");
            String password = request.getParameter("password");
            String phoneNo = request.getParameter("phone-no");

            try {
                User user = new User();
                user.setLogin(login);
                user.setPassword(PasswordEncrypter.encrypt(password));
                user.setPhoneNo(phoneNo);
                user.setRights(USER_RIGHTS);
                user.setState(true);

                userDAO.createUser(user);

                page = JspConfig.getProperty(JspConfig.REGISTER_OK);
            } catch (DAOException | NoSuchAlgorithmException e) {
                LOGGER.error(e);
                throw new ServiceException(e);
            }
        } else {
            request.setAttribute("errorSet", Validator.getErrorSet());
            page = JspConfig.getProperty(JspConfig.REGISTER);
        }

        return page;
    }
}

/**
 * Open register page command
 */
public class RegisterStartCommand implements ICommand {
    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        return JspConfig.getProperty(JspConfig.REGISTER);
    }
}
/**
 * Add new goods to cart command
 */
public class CartAddCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CartAddCommand.class);
    private IOrderDAO orderDAO = new OrderDAO();
```

```
    private IGoodDAO goodDAO = new GoodDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Order order = new Order();
            order.setDate(new Date());

            User user = (User) request.getSession().getAttribute("user");
            order.setUser(user);

            Integer goodId =
Integer.parseInt(request.getParameter("goodId"));
            Good good = goodDAO.readGood(goodId);
            order.setGood(good);

            orderDAO.createOrder(order);

            page = new CartListCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Delete good form cart command
 */
public class CartDeleteCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CartDeleteCommand.class);
    private IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
       String page = null;

        try {
            Integer orderId =
Integer.parseInt(request.getParameter("orderId"));
            Order order = new Order();
            order.setOrderId(orderId);
            orderDAO.deleteOrder(order);
```

```
            page = new CartListCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Edit number of goods in cart command
 */
public class CartEditCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CartEditCommand.class);
    private IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer orderId =
Integer.parseInt(request.getParameter("orderId"));
            Integer number =
Integer.parseInt(request.getParameter("number"));
            Order order = orderDAO.readOrder(orderId);
            order.setNumber(number);
            orderDAO.updateOrder(order);

            page = new CartListCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}
/**
 * Find goods by name command
 */
public class CartFindCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CartFindCommand.class);
    private IGoodDAO goodDAO = new GoodDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
```

```
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            String value = request.getParameter("value");
            List<Good> goodList = goodDAO.readGoodByName(value);
            request.setAttribute("goodList", goodList);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return JspConfig.getProperty(JspConfig.GOODS_BY_CATEGORY);
    }
}

/**
 * List goods in cart by user command
 */
public class CartListCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CartListCommand.class);
    private IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            User user = (User) request.getSession(true).getAttribute("user");
            List<Order> orderList = orderDAO.readOrderByUser(user);
            request.setAttribute("orderList", orderList);

            page = JspConfig.getProperty(JspConfig.CART);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }
        return page;
    }
}

/**
 * Submit cart command
 */
public class CartSubmitCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CartSubmitCommand.class);
    IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
```

```
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer orderId =
Integer.parseInt(request.getParameter("orderId"));
            Order order = orderDAO.readOrder(orderId);
            order.setSubmited(true);
            orderDAO.updateOrder(order);

            page = new CartListCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * List goods by category command
 */
public class GoodsListCategoryCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(GoodsListCategoryCommand.class);
    private IGoodDAO goodDAO = new GoodDAO();
    private ICategoryDAO categoryDAO = new CategoryDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        Integer categoryId =
Integer.parseInt(request.getParameter("categoryId"));
        Integer startPage = 0;
        if(request.getParameter("startPage")!=null) {

            startPage = Integer.parseInt(request.getParameter("startPage"));
        }

        try {
            int num = goodDAO.readNumGoodsByCategory(categoryId);
            request.setAttribute("numGoods", num);
            List<Good> goodList = goodDAO.readGoodByCategory(categoryId,
startPage);
            request.setAttribute("goodList", goodList);
            Category category = categoryDAO.readCategory(categoryId);
            request.setAttribute("category", category);
            request.setAttribute("startPage", startPage);
            request.setAttribute("pageNo",
request.getParameter("pageNo")!=null?request.getParameter("pageNo"):"1");
            request.setAttribute("lastPage", num/8);
        } catch (DAOException e) {
            LOGGER.error(e);
```

```
            throw new ServiceException(e);
        }

        return JspConfig.getProperty(JspConfig.GOODS_BY_CATEGORY);
    }
}

/**
 * Bann user command
 */
public class UserBannCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(UserBannCommand.class);
    private IUserDAO userDAO = new UserDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer userId =
Integer.parseInt(request.getParameter("userId"));
            User user = userDAO.readUser(userId);
            user.setState(!user.getState());
            userDAO.updateUser(user);

            page = new UserStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}/**
 * Delet user command
 */
public class UserDeleteCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(UserDeleteCommand.class);
    private IUserDAO userDAO = new UserDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer userId =
Integer.parseInt(request.getParameter("userId"));
```

```
            User user = new User();
            user.setUserId(userId);

            userDAO.deleteUser(user);

            page = new UserStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Change user access command
 */
public class UserRightsCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(UserRightsCommand.class);
    public static final String USER_RIGHTS = "ROLE_USER";
    public static final String ADMIN_RIGHTS = "ROLE_ADMIN";

    private IUserDAO userDAO = new UserDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer userId = Integer.parseInt(request.getParameter("userId"));
            User user = userDAO.readUser(userId);

            if(USER_RIGHTS.equals(user.getRights())) {
                user.setRights(ADMIN_RIGHTS);
            } else {
                user.setRights(USER_RIGHTS);
            }

            userDAO.updateUser(user);

            page = new UserStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Open user list view command
 */
public class UserStartCommand implements ICommand {
```

```
    public static final Logger LOGGER =
Logger.getLogger(UserStartCommand.class);
    private IUserDAO userDAO = new UserDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            List<User> userList = userDAO.readUser();
            request.setAttribute("userList", userList);

            page = JspConfig.getProperty(JspConfig.ADMIN_USER);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}
/**
 * Accept order command
 */
public class OrderAcceptCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(OrderAcceptCommand.class);
    private IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer orderId =
Integer.parseInt(request.getParameter("orderId"));
            Order order = orderDAO.readOrder(orderId);
            order.setDelivered(true);

            orderDAO.updateOrder(order);

            page = new OrderStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
```

```
}

/**
 * Decline order command
 */
public class OrderDeclineCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(OrderDeclineCommand.class);
    private IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer orderId =
Integer.parseInt(request.getParameter("orderId"));
            Order order = new Order();
            order.setOrderId(orderId);
            orderDAO.deleteOrder(order);

            page = new OrderStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Find orders between dates command
 */
public class OrderFilterCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(OrderFilterCommand.class);
    private IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

        try {
            Date low = dateFormat.parse(request.getParameter("low"));
            Date high = dateFormat.parse(request.getParameter("high"));

            List<Order> orderList = orderDAO.readOrder(low, high);
```

```
            request.setAttribute("orderList", orderList);

            page = JspConfig.getProperty(JspConfig.ADMIN_ORDERS);
        } catch (DAOException | ParseException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * View order list page command
 */
public class OrderStartCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(OrderStartCommand.class);
    IOrderDAO orderDAO = new OrderDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            List<Order> orderlist = orderDAO.readOrder();
            request.setAttribute("orderList", orderlist);

            page = JspConfig.getProperty(JspConfig.ADMIN_ORDERS);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * View good list page command
 */
public class GoodsStartCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(GoodsStartCommand.class);
    private IGoodDAO goodDAO = new GoodDAO();
    private ICategoryDAO categoryDAO = new CategoryDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
```

```
        String page = null;

        try {
            List<Good> goodList = goodDAO.readGood();
            request.setAttribute("goodList", goodList);

            List<Category> categoryList = categoryDAO.readCategory();
            request.setAttribute("categoryList", categoryList);

            page = JspConfig.getProperty(JspConfig.ADMIN_GOODS);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}
```

```
/**
 * Edit good start command
 */
public class GoodsEditStartCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(GoodsEditStartCommand.class);
    IGoodDAO goodDAO = new GoodDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer goodsId =
Integer.parseInt(request.getParameter("goodId"));
            Good good = goodDAO.readGood(goodsId);
            request.setAttribute("editGoods", good);

            page = new GoodsStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Edit good commit command
 */
public class GoodsEditCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(GoodsEditCommand.class);
    private IGoodDAO goodDAO = new GoodDAO();
    private ICategoryDAO categoryDAO = new CategoryDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Good good = new Good();
            good.setGoodId(Integer.parseInt(request.getParameter("goodId")));
            good.setName(request.getParameter("name"));
            good.setAbout(request.getParameter("about"));
good.setCategory(categoryDAO.readCategory(Integer.parseInt(request.getParamet
er("categoryId"))));
            good.setPrice(Integer.parseInt(request.getParameter("price")));
```

```
            goodDAO.updateGood(good);

            page = new GoodsStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Delete good command
 */
public class GoodsDeleteCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(GoodsDeleteCommand.class);
    IGoodDAO goodDAO = new GoodDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer goodsId =
Integer.parseInt(request.getParameter("goodId"));
            Good good = new Good();
            good.setGoodId(goodsId);

            goodDAO.deleteGood(good);

            page = new GoodsStartCommand().execute(request);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Add new good command
 */
public class GoodsAddCommand  extends AbstractFileUploadCommand implements
ICommand {
    public static final Logger LOGGER =
Logger.getLogger(GoodsAddCommand.class);
    private ICategoryDAO categoryDAO = new CategoryDAO();
    private IGoodDAO goodDAO = new GoodDAO();

    /**
     * Process request
     * @param request           request to process
```

```
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        try {
            Good good = new Good();

            for(FileItem item: multiparts) {
                String name = item.getFieldName();

                switch (name) {
                    case "name":
                        good.setName(item.getString());
                        break;
                    case "price":
                        good.setPrice(Integer.parseInt(item.getString()));
                        break;
                    case "about":
                        good.setAbout(item.getString());
                        break;
                    case "categoryId":
                        Category category =
categoryDAO.readCategory(Integer.parseInt(item.getString()));
                        good.setCategory(category);
                        break;
                }
            }

            good.setPath("images\\goods\\"+String.valueOf(good.hashCode()));

            goodDAO.createGood(good);

            for(FileItem item: multiparts) {
                if(!item.isFormField()) {
                    File uploadedFile = null;
                    //выбираем файлу имя пока не найдём свободное
                    String path = good.getPath();
                    uploadedFile = new
File(request.getSession().getServletContext().getRealPath(path));
                    //создаём файл
                    uploadedFile.createNewFile();

System.out.println(request.getSession().getServletContext().getRealPath(path)
);
                    //записываем в него данные
                    item.write(uploadedFile);
                 }
            }
        } catch (Exception | DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return new GoodsStartCommand().execute(request);
    }
}

public class CategoryStartCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CategoryStartCommand.class);
    private ICategoryDAO categoryDAO = new CategoryDAO();
```

```
    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            List<Category> categoryList = categoryDAO.readCategory();
            request.setAttribute("categoryList", categoryList);

            page = JspConfig.getProperty(JspConfig.ADMIN_CATEGORY);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return page;
    }
}

/**
 * Edit category command
 */
public class CategoryEditCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CategoryEditCommand.class);
    private ICategoryDAO categoryDAO = new CategoryDAO();

    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        if(Validator.validateCategoryForm(request)) {
            Integer categoryId =
Integer.parseInt(request.getParameter("categoryId"))
            Category category = new Category();
            category.setName(request.getParameter("name"));
            category.setPath(request.getParameter("path"));
            category.setCategoryId(categoryId);

            try {
                categoryDAO.updateCategory(category);
            } catch (DAOException e) {
                LOGGER.error(e);
                throw new ServiceException(e);
            }
        } else {
            request.setAttribute("errorSetEdit", Validator.getErrorSet());
        }

        page = new CategoryStartCommand().execute(request);

        return page;
    }
}
```

```
/**
 * Delete category command
 */
public class CategoryDeleteCommand implements ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CategoryDeleteCommand.class);
    private ICategoryDAO categoryDAO = new CategoryDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{
        String page = null;

        try {
            Integer categoryId =
Integer.parseInt(request.getParameter("categoryId"));
            Category category = new Category();
            category.setCategoryId(categoryId);
            categoryDAO.deleteCategory(category);
        } catch (DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        page = new CategoryStartCommand().execute(request);

        return page;
    }
}

/**
 * Add new category command
 */

public class CategoryAddCommand extends AbstractFileUploadCommand implements
ICommand {
    public static final Logger LOGGER =
Logger.getLogger(CategoryAddCommand.class);
    private ICategoryDAO categoryDAO = new CategoryDAO();

    /**
     * Process request
     * @param request           request to process
     * @return                  name of new view
     * @throws ServiceException
     */
    @Override
    public String execute(HttpServletRequest request) throws ServiceException
{

        try {
            Category category = new Category();

            for(FileItem item: multiparts) {
                String name = item.getFieldName();

                switch (name) {
```

```
                    case "name":
                        category.setName(item.getString());
                        break;
                }
            }


category.setPath("images\\categories\\"+String.valueOf(category.hashCode()));

            categoryDAO.createCategory(category);

            for(FileItem item: multiparts) {
                if(!item.isFormField()) {
                    File uploadedFile = null;
                    //выбираем файлу имя пока не найдём свободное
                    String path = category.getPath();
                    uploadedFile = new
File(request.getSession().getServletContext().getRealPath(path));
                    //создаём файл
                    uploadedFile.createNewFile();

                    //записываем в него данные
                    item.write(uploadedFile);
                }
            }
        } catch (Exception | DAOException e) {
            LOGGER.error(e);
            throw new ServiceException(e);
        }

        return new CategoryStartCommand().execute(request);
    }
}
```

Листинг скрипта генерации базы данных

```
CREATE DATABASE IF NOT EXISTS `shopdb`
USE `shopdb`;

CREATE TABLE IF NOT EXISTS `category` (
  `CategoryID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` varchar(20) NOT NULL,
  `Path` varchar(100) NOT NULL DEFAULT 'images\\categories\\no-image.jpg',
  PRIMARY KEY (`CategoryID`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `good` (
  `GoodID` int(11) NOT NULL AUTO_INCREMENT,
  `Price` int(11) NOT NULL,
  `Name` varchar(50) NOT NULL,
  `About` varchar(1000) NOT NULL,
  `Path` varchar(100) NOT NULL,
  `CategoryID` int(11) NOT NULL,
  PRIMARY KEY (`GoodID`),
  KEY `FK_book_category` (`CategoryID`),
  CONSTRAINT `FK_good_category` FOREIGN KEY (`CategoryID`) REFERENCES
`category` (`CategoryID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=18 DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `order` (
  `OrderID` int(11) NOT NULL AUTO_INCREMENT,
  `Date` date NOT NULL,
  `Submited` bit(1) NOT NULL DEFAULT b'0',
  `Delivered` bit(1) NOT NULL DEFAULT b'0',
  `Number` int(11) NOT NULL DEFAULT '1',
  `UserID` int(11) NOT NULL,
  `GoodID` int(11) NOT NULL,
  PRIMARY KEY (`OrderID`),
  KEY `FK_order_user` (`UserID`),
  KEY `FK_order_good` (`GoodID`),
  CONSTRAINT `FK_order_good` FOREIGN KEY (`GoodID`) REFERENCES `good`
(`GoodID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `FK_order_user` FOREIGN KEY (`UserID`) REFERENCES `user`
(`UserID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `user` (
  `UserID` int(11) NOT NULL AUTO_INCREMENT,
  `Login` varchar(25) NOT NULL,
  `Password` varchar(64) NOT NULL,
  `Rights` varchar(15) NOT NULL,
  `State` bit(1) NOT NULL,
  `PhoneNo` varchar(20) NOT NULL,
  PRIMARY KEY (`UserID`),
  UNIQUE KEY `Login` (`Login`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8;
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Блок-схемы

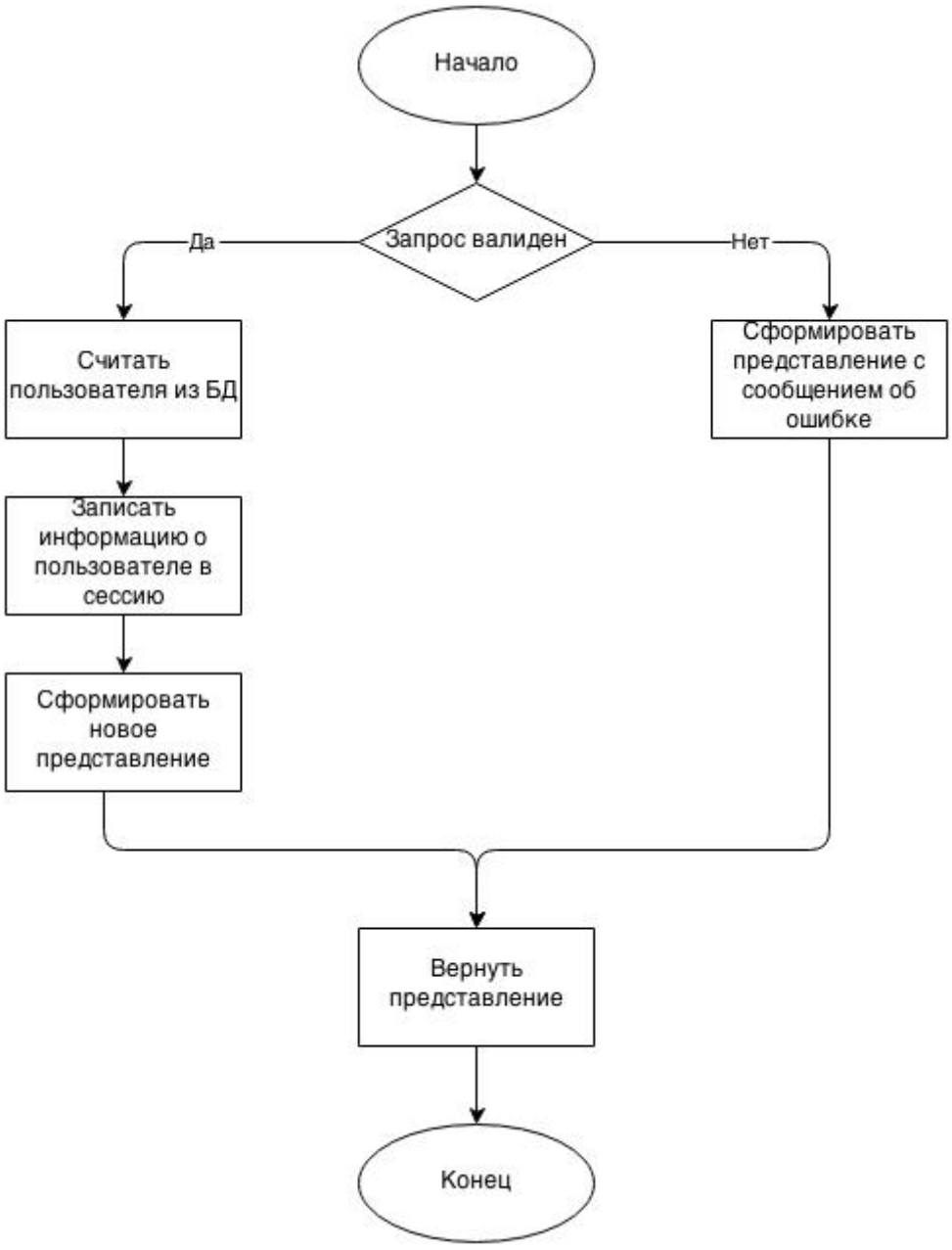

Рисунок Б.1 – Блок-схема алгоритма авторизации

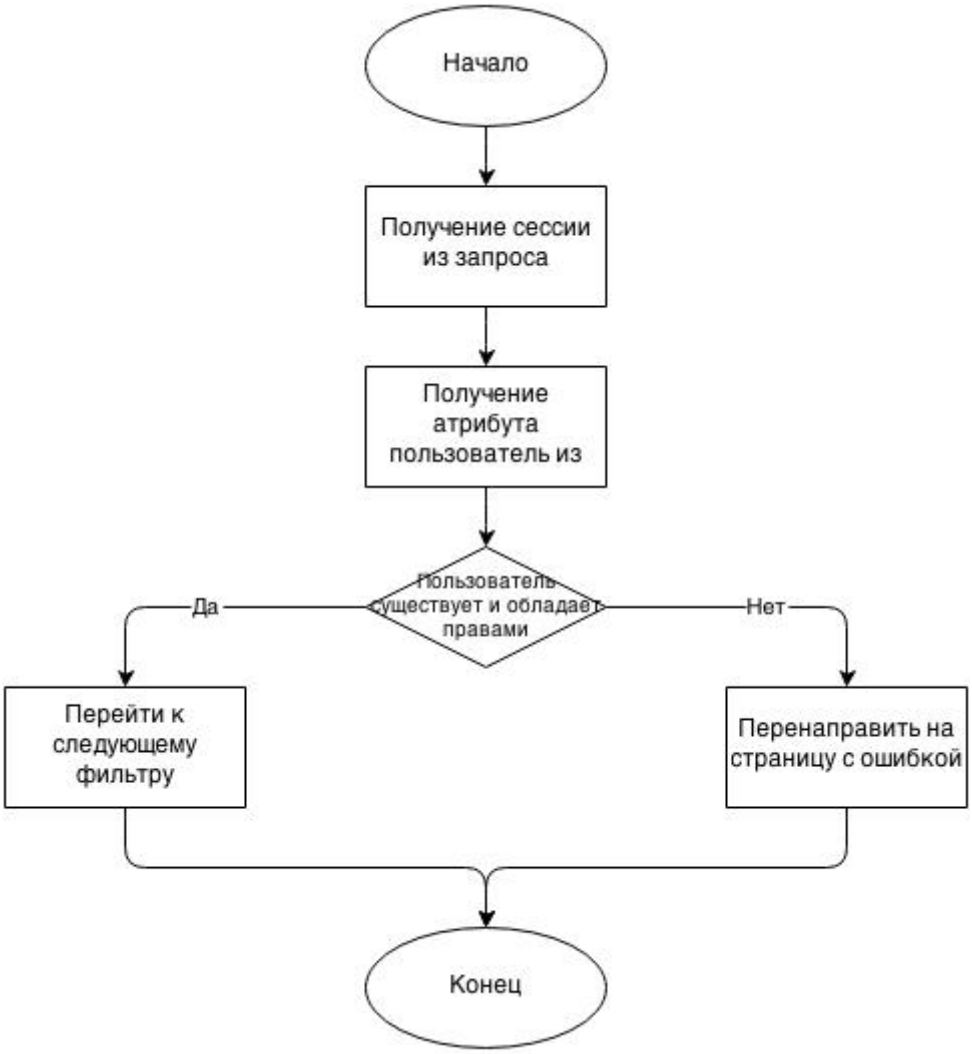

Рисунок Б.2 – Блок-схема алгоритма фильтрации доступа

ПРИЛОЖЕНИЕ В